UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ARMLiU (Augmented reality. Machine Learning In Use).

Asistente virtual para una Biblioteca.

Trabajo de Fin de Máster

Máster Universitario en Ingeniería Informática

Curso académico:

2022-2023

Autor:

Enrique Garcerá Rayo

Director:

Dr. Pablo Ruipérez García.

Defensa:

Septiembre de 2023

1 Resumen del trabajo.

La Realidad Aumentada es un recurso muy potente que nos permite conectar un mundo real en el que estamos inmersos, con uno virtual cada vez más realista. El proceso de modernización y digitalización de las bibliotecas tradicionales comenzó hace décadas y en el presente trabajo se ha realizado una pequeña aproximación a las iniciativas más relevantes que emplean la realidad aumentada para la explotación de los diferentes catálogos bibliográficos. Una vez analizadas las diferentes posibilidades existentes, se ha diseñado y desarrollado una aplicación para dispositivos móviles con Sistema Operativo Android, que, apoyándose en el uso de la Realidad Aumentada y el del Aprendizaje Automático, pretende mejorar la experiencia de usuario en tareas cotidianas como la consulta de un catálogo bibliográfico.

Siendo conscientes de la envergadura del proyecto y del tiempo disponible para la elaboración del mismo, se plantean una serie de propuestas de mejora/vías futuras, en pro de dotarlo de continuidad en el tiempo y entendiendo el material entregado para su defensa como una primera versión de una herramienta que, una vez alcanzada la madurez mínima necesaria, aspira a convertirse en una realidad dentro de la plataforma de distribución de aplicaciones “Play Store”.

2 Palabras clave.

Realidad aumentada, aprendizaje automático, catálogo bibliográfico, app, Android, PMB, OPAC, OAI-PMH, SQLite, MySql.

3 Abstract.

Augmented Reality is a very powerful resource that allows us to connect a real world in which we are immersed, with an increasingly realistic virtual one. The process of modernization and digitization of traditional libraries began decades ago and in this paper a small approximation of the most relevant initiatives that use augmented reality for the exploitation of the different bibliographic catalogs has been made. Once the different existing possibilities have been analyzed, an application for mobile devices with the Android Operating System has been designed and developed, which, based on the use of Augmented Reality and Machine Learning, aims to improve the user experience in daily tasks such as consultation of a bibliographic catalogue.

Being aware of the magnitude of the project and the time available for its preparation, a series of proposals for improvement/future paths are proposed, in order to provide it with continuity over time and understanding the material delivered for its defense as a first version. of a tool that, once it has reached the minimum necessary maturity, aspires to become a reality within the "Play Store" application distribution platform.

4 Keywords.

Augmented reality, machine learning, bibliographic catalogue, app, Android, PMB, OPAC, OAI-PMH, SQLite, MySql.

5 Agradecimientos.

Supongo que las personas somos seres sociables y que en múltiples ocasiones sentimos la necesidad de expresar nuestros sentimientos a alguien o en su defecto al universo y no seré yo el que desaproveche esta ocasión.

Siendo consciente de la inmensa capacidad que tiene la muerte de eclipsar incluso a la vida y de que nos podría consolar el pensar que aquellos seres queridos que nos abandonaron, quizás nos estén escuchando u observando desde algún rincón del universo por muy remoto que este sea, quiero dar las gracias muy especialmente a mi padre y a mi suegra que me abandonaron en el mes de marzo y abril del presente año. El agradecimiento es simple, no existen palabras para agradecer toda una vida de dedicación sin esperar recibir nada a cambio. Dejáis un vació inmenso en mi interior que intuyo, solo el tiempo podrá minimizar. Os quiero y os quise con toda la fuerza de mi ser. Y siendo consciente de aquella frase que me decía mi padre muy a menudo “Enrique, para morirse no hace falta tanto” y que con tantas cosas que hacer en un día nos olvidamos de disfrutar de los seres que nos rodean antes de que sea demasiado tarde, quiero agradecer con toda mi alma, a mi compañera de viaje, a mi amor inmortal, Mara, que siempre esté ahí, en los buenos y en los malos momentos y el haberme hecho encontrar la felicidad plena. A mis hijos, Enrique y Vicent, que le dan sentido a mi vida y que me hacen comprender porque mis padres fueron capaces de dar tanto sin esperar nada a cambio. Gracias por ser dos seres tan maravillosos y por tanta comprensión en cuanto a las horas de dedicación al presente trabajo y a tantos otros.

Por supuesto, eternamente agradecido a mi madre y a mis hermanas, son la mejor madre y hermanas que una persona puede tener. Gracias por ser como sois y por el privilegio de haber crecido a vuestro lado. Mamá, me comprometo públicamente a ir a verte más a menudo.

Y por último, pero no por ello menos importante, al tutor del presente trabajo, Dr., Pablo Ruipérez García. Muchísimas gracias por su apoyo durante todo este tiempo y por centrarme para que los objetivos perseguidos fueran más realistas.

Índice.

6 Listado de figuras.

7 Listado de tablas.

8 Introducción.

El presente documento describe los objetivos marcados en el TFM (Trabajo fin de Máster), aquellos aspectos más relevantes en cuanto al proceso seguido, las metodologías empleadas y los problemas encontrados durante su elaboración, las conclusiones que hemos extraído, así como propuestas de mejora que propiciarán que el esfuerzo y los recursos obtenidos durante este tiempo sirvan como punto de partida para otros proyectos.

8.1 Motivación.

Cuando accedemos a una biblioteca con la intención de consultar un ejemplar bibliográfico, la localización de este no siempre es un proceso sencillo. Si bien es cierto que las bibliotecas suelen contar con algún tipo de recurso tecnológico dónde en función del sistema de catalogación empleado en dicha biblioteca, al realizar una breve búsqueda nos indica la ubicación exacta dentro de la misma, no lo es menos que los ejemplares no siempre están en su sitio. En ocasiones, dentro de una misma librería, los ejemplares se encuentran desordenados y aún en el caso de que no lo estén, según la cantidad, la localización puede requerir de cierto tiempo. También es cierto que el uso de dispositivos móviles está ampliamente extendido y que un porcentaje muy alto de la población dispone de uno. Este es el motivo por el que hemos decidido implementar una aplicación que se ejecutará sobre un dispositivo móvil y que, empleando técnicas de RA (realidad aumentada), nos asistirá en la localización de los diferentes ejemplares.

8.2 Objetivos Propuestos.

Se pretende diseñar y desarrollar una aplicación que sirva como Asistente Virtual en una Biblioteca. La aplicación se ejecutará sobre un dispositivo móvil con S.O (Sistema Operativo) Android y se apoyará en la cámara del dispositivo móvil para hacer uso de técnicas de RA. A groso modo, las funcionalidades de las que dispondrá la aplicación son:

- **[1] Reconocimiento de los libros disponibles en la biblioteca:** Al "enfocar" sobre alguno de los ejemplares presentes en una estantería, la aplicación ofrecerá información con respecto a dicho ejemplar. La información proporcionada dependerá de la configuración realizada en la

aplicación. Por defecto, mostrará elementos como si el ejemplar es prestable o no, si existen otros ejemplares en otras estanterías/bibliotecas, breve descripción del libro, etc..

- **[2] Interacción por voz:** Se podrá interactuar con la aplicación por voz. Dotándola de cierta comodidad para el público en general y de cierta utilidad para personas con visión reducida o nula.
- **[3] Búsqueda de ejemplares:** Se podrá realizar una búsqueda dentro de la aplicación, indicando ciertos patrones de búsqueda con vistas a localizar un determinado ejemplar dentro de la biblioteca.

Como posibles propuestas de mejora que serán abordadas o no en función del tiempo disponible y la evolución del TFM:

- [4] En la búsqueda de un ejemplar, la aplicación detectará la ubicación de la persona y le guiará en caso de que lo desee para que encuentre la ubicación física del mismo.
- [5] Se plantea la construcción de unas gafas de RA que se conectarán a la aplicación móvil y que servirán de interfaz de usuario para esta. Físicamente las gafas de realidad aumentada serán un complemento acoplable a cualquier gafa común. Dispondrán de una serie de sensores táctiles con el objetivo de interactuar con ellas, una cámara que captará imágenes del mundo real y un sistema de proyección que mostrará una serie de textos y representaciones gráficas muy sencillas al usuario final. Soportarán la interacción por voz, disponiendo las mismas de un micrófono incorporado. Las gafas contarán con un microcontrolador tipo Arduino Nano o similar y se conectarán al dispositivo móvil de forma inalámbrica. La aplicación Android dispondrá de la mayor parte de la lógica de negocio, implementando sobre las gafas unas funcionalidades mínimas para que sean mínimamente operativas en situación de desconexión del dispositivo móvil.
- [6] Se aplicarán técnicas de AA (aprendizaje automático) dónde, en función de los hábitos de búsqueda y en función de los ejemplares prestados a un usuario en concreto que esté utilizando la aplicación, se le propondrán obras bibliográficas que podrían ser de su interés.

9 Estado del arte.

A continuación, realizaremos un breve análisis sobre:

- Algunas de las herramientas que dan soporte a la catalogación y gestión de los diferentes ejemplares bibliográficos, y que por tanto despiertan nuestro interés. Estás reciben el nombre común "**Sistema Integrado de Gestión de Bibliotecas**" (SIGB).
- Herramientas que permiten la **consulta/interacción** sobre un determinado catálogo mediante alguna aplicación específica o mediante el navegador web.
- Iniciativas que emplean la **RA** en el contexto Bibliotecario.

9.1 Sistemas Integrados de Gestión de Bibliotecas (SIGB).

Las necesidades específicas de una biblioteca concreta suelen ser idénticas o muy similares a las del resto y este es un aspecto destacable, dado que, hace años los formatos y sistemas empleados para la gestión de las mismas era muy dispar. A partir de un cierto momento fueron apareciendo iniciativas en pro de homogeneizar tanto las herramientas como los formatos empleados para la persistencia de los datos y el acceso a los mismos, dando origen a lo que se conoce como SIGB. Algunos de los SIGB más comunes, son:

- Vufind: nos proporciona la posibilidad de implementar un portal de contenidos accesible desde el navegador que centralice los diferentes recursos de nuestra biblioteca. En definitiva, es una aplicación de descubrimiento en la que tenemos la posibilidad de definirle diferentes orígenes de datos/SIGB sobre los que de forma transparente al usuario y homogeneizando la interfaz de búsqueda y explotación de los resultados, nos permitirá abstraernos del SIGB subyacente y acceder al catálogo de documentos, a la biblioteca digital de artículos, al repositorio institucional, etc. Pese a que inicialmente experimentamos con dicho sistema para emplearlo como portal sobre el que interactuar con la aplicación móvil que hemos desarrollado llegando a instalarlo y a configurarlo, las posibilidades de interacción con el mismo estaban muy limitadas, dado que está pensado principalmente para que la explotación del sistema se realice mediante el

navegador web o un dispositivo móvil sobre el que proyectar dicho portal de forma atómica. En cualquier caso y fruto de esta experimentación, conseguimos habilitar en Vufind el servidor OAI-PMH (Open Archive Initiative-Protocol for Metadata Harvesting), sin embargo, este tipo de interacción está muy restringida en cuanto a los filtros de búsqueda, dado que está pensada para “cosechar” información para su posterior explotación en los sistemas que la obtienen y pese a que vufind es de código abierto, esta implementado mediante lenguaje PHP (Hypertext Preprocessor) en su mayoría y existe la posibilidad de implementarle una serie de servicios que cumplan con nuestras necesidades, esto habría consumido un tiempo excesivo y se escapaba de nuestras pretensiones iniciales, más enfocadas a interactuar con un sistema realizando en este ninguna o pocas modificaciones. No se descarta realizar estas modificaciones en un futuro. Consideramos importante destacar que Vufind se apoya en el Servidor Web Apache (Apache Web Server, 2023), el motor de búsqueda solr (Apache Solr, 2023) , el SGBD (Sistema de Gestión de Bases de datos) Mysql (Oracle MySQL. , 2023) y el lenguaje de programación PHP (PHP Group, 2023).

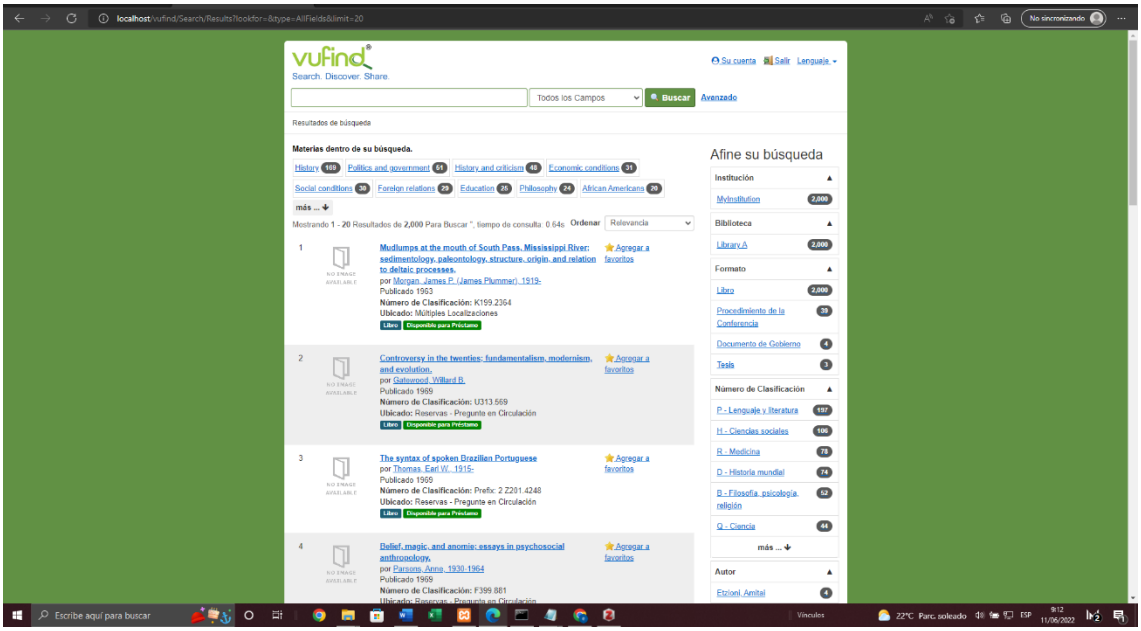


figura 1. Ejemplo de ejecución de Vufind en servidor local.

En la figura 1, podemos observar la ejecución de Vufind una vez importados una serie de datos de la Biblioteca Nacional de España y tras ponerlo en funcionamiento en el servidor local.

- Zotero: probamos también la herramienta Zotero, pero pese a su potencial, vimos que estaba muy enfocada a una ejecución local y muy especializada en la obtención de citas bibliográfica. Realizamos su instalación y al intentar interactuar con ella pudimos comprobar como la interacción con sistemas externos estaba muy centrada en la importación y exportación de archivos con determinado formato y por tanto fue descartada. Las pretensiones iniciales fueron enlazar Vufind con Zotero con vistas a que nuestra aplicación interactuara con Vufind y la información estuviera almacenada en Zotero (Zotero. Importación de Citas., 2023).

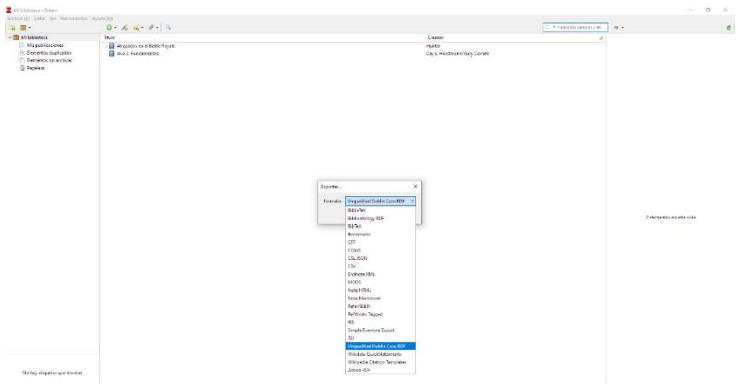


figura 2. Ejemplo de ejecución de Zotero en PC local.

- Koha: pese a que no llegamos a instalar Koha y por tanto no pudimos experimentar con este SIGB, nos pareció una alternativa bastante interesante, dado que nos encontramos ante un sistema de código abierto con una amplia comunidad que respalda su desarrollo, cuenta con numerosa documentación traducida a diferentes idiomas y ofrece una serie de servicios web (Koha. Servicios Web., 2023) de serie que podrían ser ampliados con nuestros propios desarrollos y que nos permiten interactuar con el servidor. De entre estos servicios, destacar la posibilidad de habilitar un servidor OAI-PMH y la interacción empleando el protocolo Search/Retrieve vía URL (Uniform Resource Locator) (SRU). Sin embargo, pese al interés despertado, nos hemos decanta por el empleo de PMB por los motivos descritos en el apartado correspondiente.

- Exlibris Primo, Alma: Primo es un servicio de descubrimiento similar a Vufind pero desarrollado y gestionado por la corporación Exlibris Group (Exlibris Group, 2023) y centrado en la interacción con el SIGB Alma, que es una plataforma de servicios de biblioteca basada en la computación en la nube. Pese a que proporciona una serie de API (Application Programming Interface) Rest (Representational State Transfer) que nos permitirían interactuar con una serie de servicios Web que emplean estándares abiertos (Exlibris Group. Desarrolladores., 2023), para este proyecto buscábamos una plataforma más abierta. No obstante, podría ser interesante en un futuro dar soporte al enlace entre nuestra aplicación y Exlibris Primo, dado que, entre otras virtudes, se encuentra la de que es la plataforma utilizada por la biblioteca de la UNED (Universidad Nacional de Educación a Distancia), el CSIC (Consejo Superior de Investigaciones Científicas) y la UPV (Universidad Politécnica de Valencia) entre otras.
- AbsysNet: este es un sistema que emplean entre otras entidades, la INAP (Biblioteca Nacional del Instituto Nacional de Administración Pública) y todas aquellas bibliotecas de la Comunidad Valenciana unidas a la “Xarxa Electrònica de Lectura Pública”. Este software es propietario de Baratz (Baratz. Desarrollo e Implentación de Software para Bibliotecas., 2023.). Que lleva desarrollándolo desde hace unos 30 años. La versión más extendida es la que se instala en cada una de las bibliotecas de forma independiente, permitiendo la interconexión con el resto de las bibliotecas. No hemos podido tener acceso a este software.
- PMB (PhpMyBibli) (PMB Services, 2023): este es el sistema por el que finalmente nos hemos decantado. Es un SIGB de código abierto desarrollado por la compañía francesa PMB Services y cuyo uso es completamente libre. Su implementación tiene el formato de aplicación web que para funcionar requiere de: un Servidor Web (normalmente el de Apache), soporte para la ejecución del código escrito en PHP y el SGBD MySQL (My Structured Query Language) que nos proporcionara la persistencia de los datos (la estructura de Base de Datos es muy estable entre versiones y ampliamente documentada). Es un sistema muy completo, con una documentación muy extensa tanto a nivel de usuario como a nivel técnico que nos permite no tan solo la catalogación de ejemplares

bibliotecarios, sino una completa gestión de usuarios de la biblioteca, del proceso de préstamos y una serie de servicios muy interesantes como podría ser la sindicación de contenidos mediante RSS (Really Simple Syndication), servidor OPAC (Online Public Access Catalog) integrado, etc. Ofrece una serie de mecanismos que se pueden habilitar/deshabilitar y que posibilitan a PMB la interacción con sistemas externos, tanto para obtener información (por ejemplo, a la hora de catalogar un recurso bibliográfico, en lugar de tener que introducir todos los datos de forma manual podemos obtenerlos de un sistema externo), como para proporcionarla (este es el caso de la aplicación desarrollada, que se nutre de la información proporcionada por PMB). Además, respeta estándares internacionales en el contexto de la biblioteconomía y que facilitan la interrelación entre sistemas. Por tanto, un sistema que nos permite un control absoluto sobre el mismo, nos permite adaptarlo a nuestras necesidades (dado que es de código abierto), con un lenguaje de programación con el que medianamente nos podemos desenvolver y que es el SIGB empleado mayoritariamente en las bibliotecas presentes en los centros educativos de la Comunidad Valenciana y que por tanto nos hace prever un número de usuarios potenciales muy elevado con cierta tendencia al uso de los dispositivos móviles, nos ha hecho valorarlo como una muy buena opción de uso.

9.2 Herramientas de consulta/interacción.

En cuanto a herramientas específicas con posibilidad de ejecutarse en un dispositivo móvil, hemos consultado sobre la “Play Store” si ya existía alguna iniciativa similar a la planteada y pese a que hemos encontrado herramientas de consulta asociadas a diferentes instituciones, estas estaban basadas en el empleo de mecanismos tradicionales a la hora de realizar la búsqueda. No haciendo uso de la realidad aumentada. De las aplicaciones instaladas, destacar “Ex Libris Library Mobile” siendo esta una de las que hemos probado con mayor profundidad, dado que nos permite el acceso al catálogo de la UNED y dado que somos estudiantes en activo, no hemos tenido ninguna limitación en las consultas.

9.3 Empleo de la realidad aumentada.

Hemos pensado también en probar Google Lens, desarrollada por Google LLC, pero, pese a que por lo que destaca esta aplicación es por el uso de la realidad aumentada dado que, dándole una imagen detecta los textos, objetos, etc., presentes en ella y nos proporciona información ampliada, esta información es muy genérica e interesada y nos posibilita traducir los textos detectados, encontrar información específica para "hacer los deberes", nos ofrece posibilidades de compras, sitios web, comidas, pero nada tan específico como la consulta del catálogo bibliográfico de una entidad concreta.

Hemos encontrado un proyecto que podría tener cierta similitud con el planteado, y es el de la biblioteca de la Universidad de Illinois, (Illinois Library, 2017). Este es un tanto más ambicioso, dado que persigue el acceso/recorrido de la biblioteca de manera virtual empleando la realidad virtual, aumentada y

figura 3. Uso de la RA. Universidad de Illinois.

mixta. Sin embargo, este proyecto requiere de recursos menos livianos, dado que, cuando describen los recursos necesarios del proyecto, entre otras, plantea el uso de las HoloLens de Microsoft, las HTC Vive Pro W, etc.

10 Metodología.

Para alcanzar los objetivos marcados, nos hemos servido de una serie de métodos que se describirán en el presente apartado y dónde las técnicas de experimentación y observación han cobrado un papel muy relevante.

Como punto de partida estuvimos pensando de qué manera se podría poner la tecnología al servicio del ser humano haciendo que la interacción dentro de un contexto bibliotecario fuera más natural. Y este punto de partida derivó en una serie de objetivos generales que han ido dando paso a otros mucho más detallados y específicos entre los que surgió la idea de desarrollar una aplicación que pudiera ser ejecutada en un dispositivo móvil, y fueron conformados los diferentes elementos que debería contener. Al valorar las diferentes plataformas sobre las que podría ser ejecutada la aplicación, pudimos observar como la cuota de mercado a nivel mundial estaba claramente polarizada hacia dos plataformas: Sistema operativo iOS y Sistema Operativo Android. Y finalmente encontramos claras ventajas en el empleo de la plataforma Android.

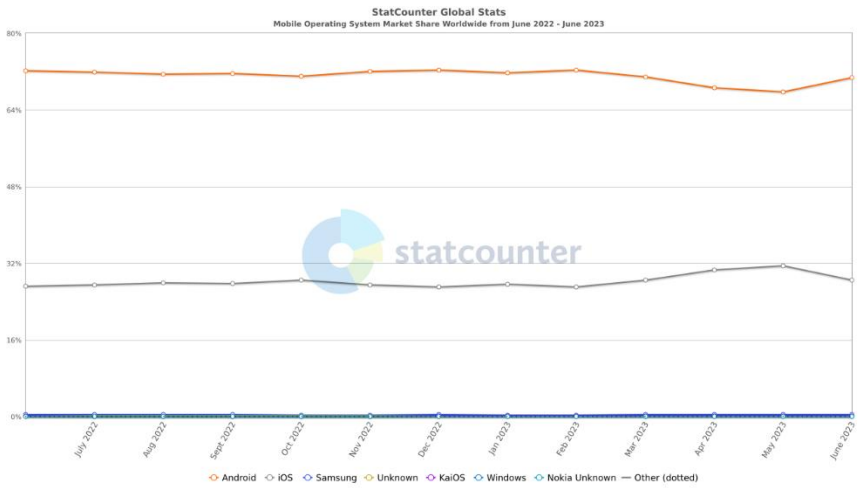


figura 4. Cuota de mercado de diferentes Sistemas Operativos para dispositivos móviles. Datos obtenidos de StatCounter.

Algunas de las ventajas:

- Como se puede observar en la figura 4, la cuota de mercado de Android a nivel mundial es mucho más elevada que la de iOS. Y

por tanto esto nos permite contar como punto de partida con un número de usuarios potenciales mucho mayor.

- Sin entrar en la posibilidad de comprar un dispositivo móvil reacondicionado que abarata el precio de compra, ni en el debate en cuanto a la relación calidad-precio que podría derivar en lo justo o injusto que es realizar una comparativa entre dos dispositivos móviles con precios muy dispares, a priori el terminal más básico que ejecuta iOS tiene un precio mucho más elevado que el más básico que ejecuta Android y por tanto eligiendo la plataforma Android se intenta democratizar el uso de la aplicación desarrollada.
- Estamos más familiarizados con las herramientas y tecnologías de desarrollo empleadas por la plataforma Android y por tanto a priori se espera una curva de aprendizaje mucho menor que nos permitirá adentrarnos en lo desconocido como son la aplicación de la RA y el AA.
- A pesar de que la tendencia va cambiando hacia Kotlin, para desarrollar la aplicación es posible emplear el lenguaje de programación Java con el que estamos más familiarizados.
- A título particular consideramos que Google es capaz de salir de la zona de confort, innovando, rectificando en caso de ser necesario y descartando proyectos que quizás no fueron acertados o quizás fueron demasiado prematuros para la época en la que fueron lanzados y esto nos atrae.

Algunas de las desventajas:

- El ecosistema hardware sobre el que se ejecuta Android es altamente heterogéneo y la mayoría de fabricantes de dispositivos además le añaden al sistema software de base sus propias capas de personalización, por lo que la labor de realizar pruebas y certificación de una app para que pueda ser ejecutada sobre unas versiones del sistema operativo, se complica enormemente.

- El ritmo al que evoluciona el API de desarrollo de Android es vertiginoso y pese a que existen una serie de librerías de compatibilidad y en esencia una forma de hacer que más o menos se mantiene, conforme evolucionaba el proyecto, nos ha quedado la sensación de que desde el primer día hemos estado adaptando el código de la aplicación a los nuevos métodos propuestos por Google que no siempre funcionan plenamente y por tanto requieren del mantenimiento de los antiguos con respecto a los que suele haber ciertas incompatibilidades y limitaciones de uso. Sin duda, este es el aspecto más relevante que ha puesto a prueba nuestra paciencia.

Para lograr el objetivo general de obtener una app (aplication) capaz de ejecutarse sobre la plataforma Android, nos hemos servido de la siguiente infraestructura hardware/software:

- Un ordenador portátil (plataforma de desarrollo) con sistema operativo Windows 10, 8GB de memoria RAM y un procesador AMD Ryzen 5. Conectado a un monitor de 24 pulgadas.
- Un servidor web Apache instalado sobre la plataforma de desarrollo y sobre el que se ha instalado PMB.
- Un SGBD MySQL instalado sobre la plataforma de desarrollo.
- El IDE (Entorno de desarrollo integrado) de desarrollo “Android Studio” instalado sobre la plataforma de desarrollo.
- La herramienta StarUML para documentar empleando UML(Lenguaje de Modelado Universal).
- Un repositorio Git local para el control del código fuente de la aplicación.
- Una librería en la que se han conformado una serie de ejemplares bibliográficos empleando para su catalogación la CDU (Clasificación Decimal Universal) dado que es el sistema más empleado en las Bibliotecas Públicas españolas.

- Una Tablet Lenovo Tab p11 con sistema operativo Android 12 (plataforma de pruebas 1).
- Diferentes emuladores con sistema operativo Android 8.0 o superior. (plataforma de pruebas 2).

10.1 Plan de acción.

Como plan de acción, a grosso modo hemos seguido la siguiente secuencia:

- Desarrollar la idea apoyándonos en un Wireframing de la aplicación.
- Investigar el estado actual y la existencia de herramientas similares a la ideada.
- Probar diferentes herramientas.
- Realizar un trabajo de campo, visitando diferentes bibliotecas para comprobar los sistemas de clasificación y catalogación empleados.
- Definir, instalar y configurar el entorno de desarrollo.
- Definir, instalar y configurar los entornos de prueba.
- Completar el análisis, diseñar, desarrollar, probar y documentar la aplicación.
- Elaborar la memoria del TFM partiendo de los diferentes materiales que se han ido generando a lo largo del proyecto.

Se ha comentado previamente que las técnicas de experimentación y observación han sido relevantes en el proyecto y a continuación se enumeran algunos contextos dónde han sido empleadas:

- Se han visitado diferentes bibliotecas interrogando al personal técnico de las mismas para investigar las normas adoptadas y las particularidades de cada uno de los sistemas empleados.
- Se ha realizado ingeniería inversa sobre las URLs que nos permiten consultar el catálogo bibliotecario a través del servidor OPAC de varias instituciones como son la UNED y la UPV.

- Se ha experimentado con diferentes entornos como pueden ser Vufind, PMB, XAMPP (X, Apache, MaríaDB/MySQL, PHP, Perl), Wampserver, Android Studio, Git, MLKit (machine learning kit), etc.
- Se han experimentado y observado los diferentes mecanismos de interacción que nos ofrecen las bases de datos documentales de la BNE (Biblioteca Nacional de España).
- Se han consultado diferentes fuentes bibliográficas.
- Hemos investigado sobre los principales mecanismos de interacción y formatos ofrecidos por los SIGB, como puede ser OAI-PMH empleando el formato DC (Dublin Core) o Marc en sus diferentes variantes.

11 Diferentes sistemas de clasificación y catalogación.

En el contexto de la biblioteconomía existen ciertas diferencias entre clasificar y catalogar. En líneas generales, se entiende por clasificar, a la agrupación por la temática contenida en la obra que queremos clasificar y por catalogar a la asignación a dicha obra de diferentes campos que típicamente contiene el ejemplar físico o lógico. Estos campos suelen ser el título, la descripción, el número de páginas, el autor, el ISBN, etc. Con vistas a poder integrar este conocimiento a la aplicación a desarrollar y para conformar la librería de prueba de una forma más realista, se ha realizado un pequeño análisis de las diferentes posibilidades intentando encontrar las más comunes y visitando in situ varias bibliotecas. Sin embargo, una explicación detallada de los diferentes sistemas de clasificación y catalogación de ejemplares se escapa de las pretensiones del presente trabajo, dado que hemos descubierto que dicho contexto es muy extenso, en continua evolución y con independencia de las normas que se han ido adoptando en el ámbito nacional e internacional, existen infinidad de casuísticas particulares que han derivado en sistemas desarrollados para unos fines particulares.

Simplificando al máximo, cuando en una biblioteca se recepciona un nuevo material, existen tres tareas básicas a realizar. La primera es el **registro** de dicho material, la segunda el **sellado** del material y la tercera, la asignación

de una **signatura** especifica que defina la ubicación de dicho material dentro de la biblioteca.

El **registro** del material consiste en añadir al inventario de materiales contenidos en la biblioteca (en el libro de registro) la información asociada al material que hemos recepcionado. En el libro de registro se identificará dicho ejemplar con un código alfanumérico que podría ser por ejemplo R-825. La R se corresponde con registro y el 825 al orden de entrada con respecto al resto de materiales.

En el **sellado**, se añade el sello identificativo de la biblioteca sobre el material que hemos recepcionado.

En la asignación de la **signatura**, se le añade al material un código alfanumérico (signatura) que se colocará en una etiqueta (tejuelo) que pegaremos en el “lomo” del libro. Este código alfanumérico es el que nos permitirá encontrar posteriormente el ejemplar dentro de las estanterías de la biblioteca y por lo tanto el que definirá su ubicación dentro de estas.

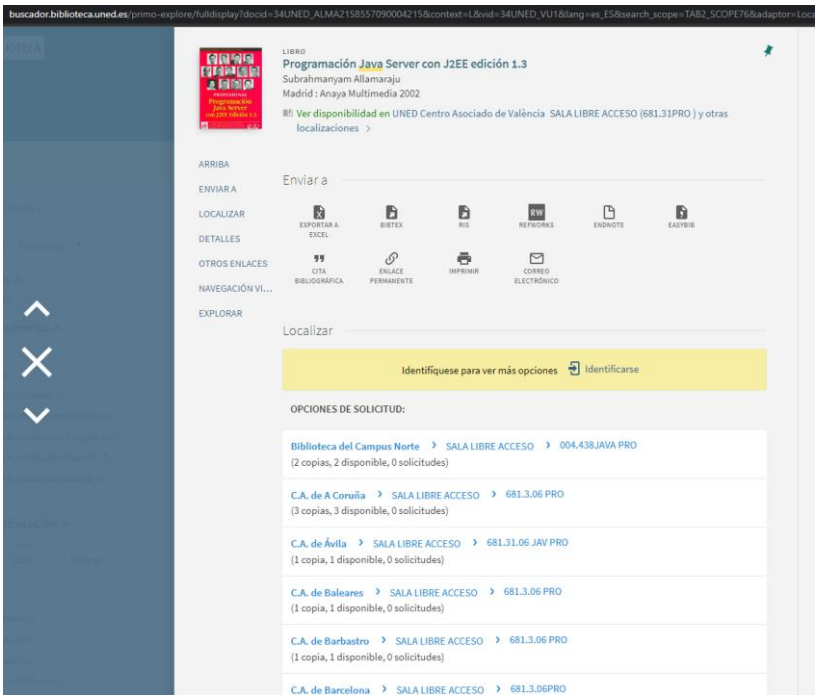


figura 5. Signatura de diferentes ejemplares. Bibliotecas UNED.

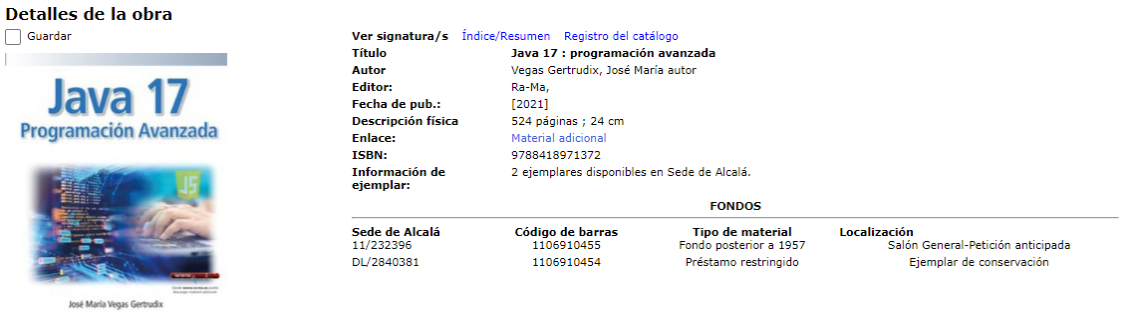


figura 6. Signatura de diferentes ejemplares presentes en la BNE.

11.1.1 Clasificación Decimal Universal.

Pese a que existen multitud de sistemas de clasificación, el Sistema de Clasificación Decimal Universal es una de los más empleados a nivel mundial y es empleado por ejemplo por la BNE (Biblioteca Nacional de España) (Bibloteca Nacinal de España, 1999). El organismo responsable de actualizar y editar esta normativa es AENOR (Asociación Española de Normalización y Certificación). Este sistema de clasificación pretende abarcar la clasificación de cualquier tipo de material y se estructura separando mediante un punto, una secuencia de cifras arábicas cada tres dígitos. El empleo de cifras arábicas persigue que este tipo de codificación no dependa de alfabetos específicos y por tanto se pueda emplear universalmente. Cada uno de los dígitos puede variar con un valor del 0 al 9. Significando cada uno de estos valores:

- 0. Generalidades.
- 1. Filosofía. Psicología.
- 2. Religión. Teología.
- 3. Ciencias Sociales.
- 4. Vacante (se puede emplear el número 4 para cualquier fin).
- 5. Matemáticas y Ciencias Naturales.
- 6. Ciencias Aplicadas.
- 7. Arte.
- 8. Lingüística y Literatura.

- 9. Geografía e Historia.

Existe una selección de clases mediante las que clasificar los diferentes materiales, que constituyen el sumario de las mismas y que, se distribuye a través de la página del consorcio de la CDU (Universal Decimal Classification Consortium., 2023) . Posee una distribución jerárquica y por tanto sobre la

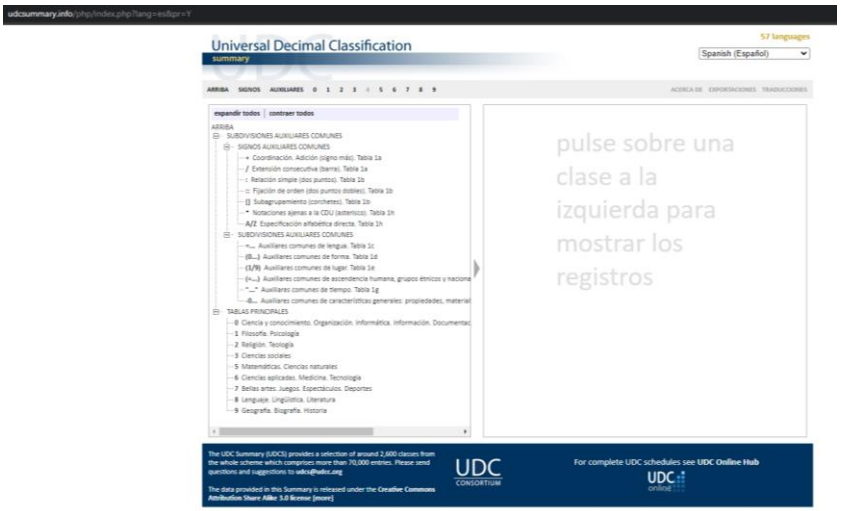


figura 7. Sumario de la Clasificación Decimal Universal.

primera cifra que clasifica el contenido de la obra ya en una clase, podemos ir añadiendo números a la derecha que reflejarán las subclases a las que pertenece dicha obra.

figura 8. Ejemplo de consulta sobre el catálogo de la BNE. Patrón de Búsqueda "java".

A continuación, se muestra un ejemplo de uso. Como se puede observar en la figura 8, la información obtenida de la BNE incorpora un campo denominado CDU. Podemos observar que contiene el valor 004.438. En el listado anterior hemos clasificado el código 0 como "Generalidades" y de esa clase genérica podemos ir recorriendo las diferentes subclases para realizar una clasificación más precisa. Si nos fijamos en la parte del sumario presente en la figura 9, el código 004 se corresponde con Ciencia y Tecnología de los Ordenadores, 004.4 se corresponde a soporte lógico del ordenador y el 004.43 a lenguajes de ordenador, habiéndole añadido el auxiliar presente en la figura 10, referente a lingüística para completar el CDU 004.438.

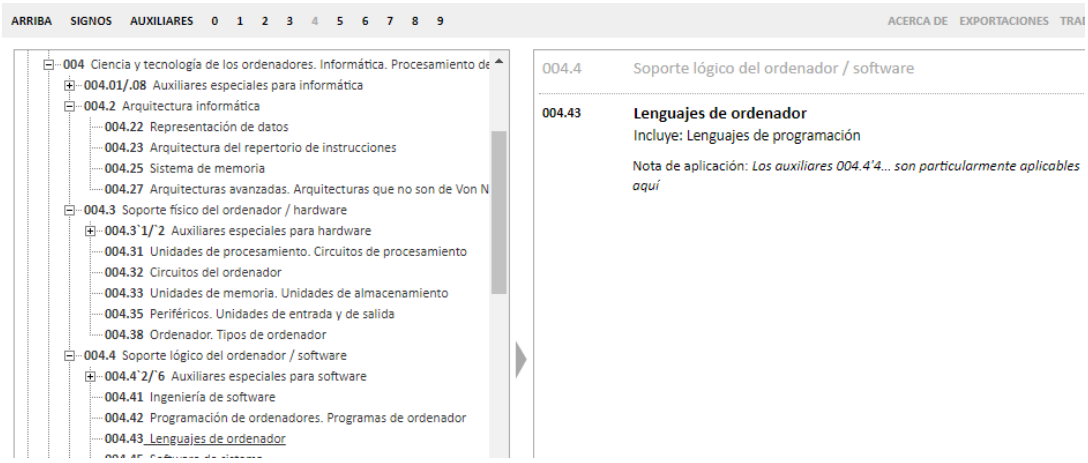


figura 9. Detalle del Sumario de la CDU.

figura 10. Auxiliar de código CDU.

11.1.2 Signatura y ubicación de ejemplares en diferentes bibliotecas.

11.1.2.1 Biblioteca pública de Valencia Pilar Faus.

figura 11. Acceso a la biblioteca Pilar Faus de Valencia.

La signatura incorporada a cada uno de los ejemplares de la presente biblioteca, incluye el CDU como primera línea, las tres primeras letras del apellido del autor en mayúsculas en la segunda línea y las tres primeras letras del título del autor en minúsculas en la tercera. Sin embargo, en aquellas obras para las que por su naturaleza no consideran una clasificación muy en detalle, como por ejemplo es la poesía o la novela, en la primera línea indican únicamente una P o una N respectivamente.

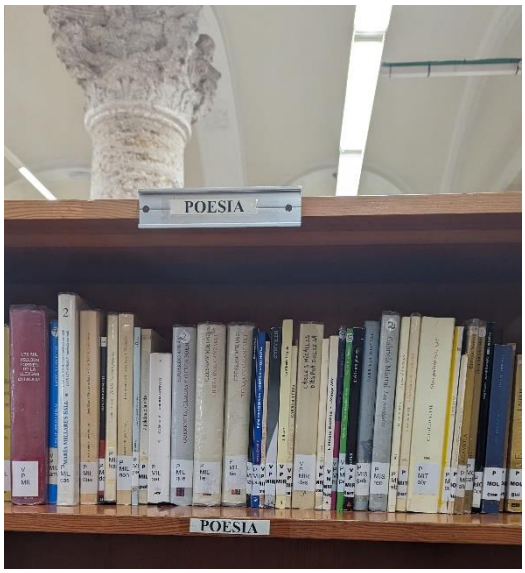


figura 12. Signatura obras cuya temática es la Poesía.

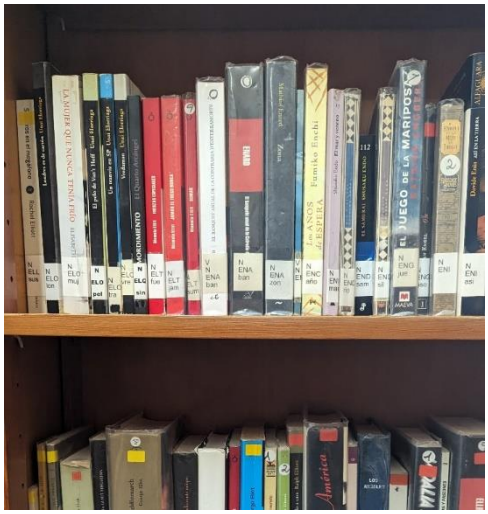


figura 13. Signatura de obras cuya temática es la Novela.

También en determinados soportes sustituyen el lugar dónde iría la CDU por el tipo de soporte como puede ser CD-ROM, DVD, etc. Omitiendo por simplicidad la clasificación del contenido de dicho soporte.

figura 14. Signatura para DVDs y CDs.

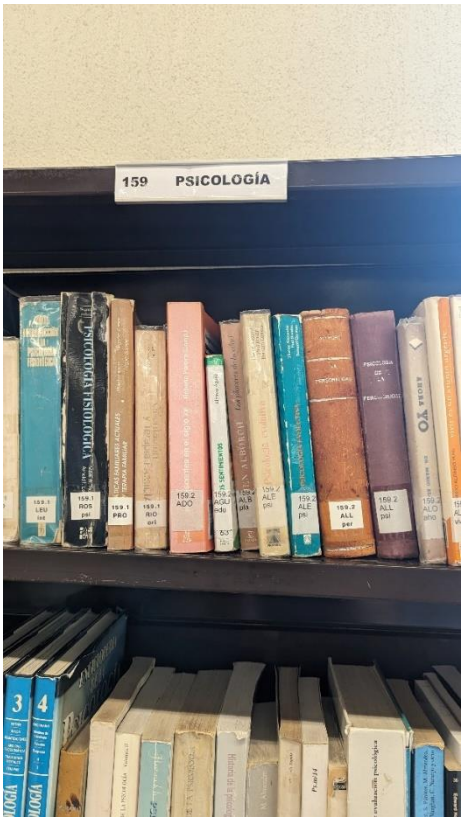


figura 16. Empleo de subclases de la CDU.

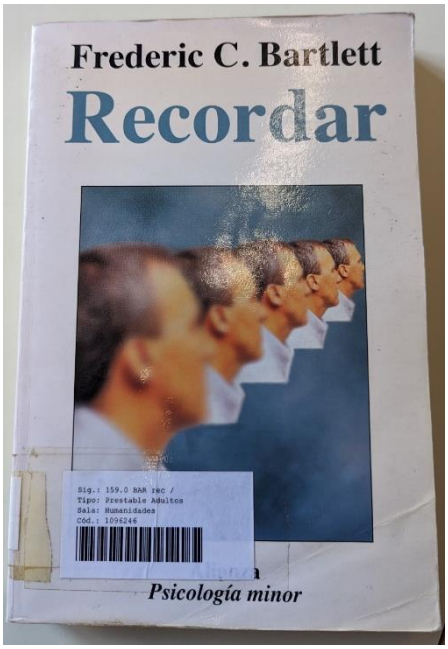


figura 15. Ejemplo de ejemplar bibliográfico. Psicología.

En el caso particular de la Psicología, pudimos observar como emplean subclases de la CDU para dar un mayor nivel de detalle a la clasificación. En la

figura 15, se muestra un ejemplar de la librería de psicología, donde podemos observar el código de la CDU (159.0), las tres primeras letras del apellido del autor (BAR) y la tres primeras letras del título (rec). Sobre dicho ejemplar podemos observar un código de barras que según información proporcionada por el personal técnico de la biblioteca asigna el programa Absys automáticamente.

Del mismo modo, nos comentaron que el número de registro tradicional ya no se realizaba y que había sido substituido por el código que asignaba Absys en su inclusión al catálogo y que podemos observar en el ejemplo de la figura 17, bajo el identificativo T. También podemos observar el sellado del ejemplar y el código de barras asignado bajo el identificador CB.

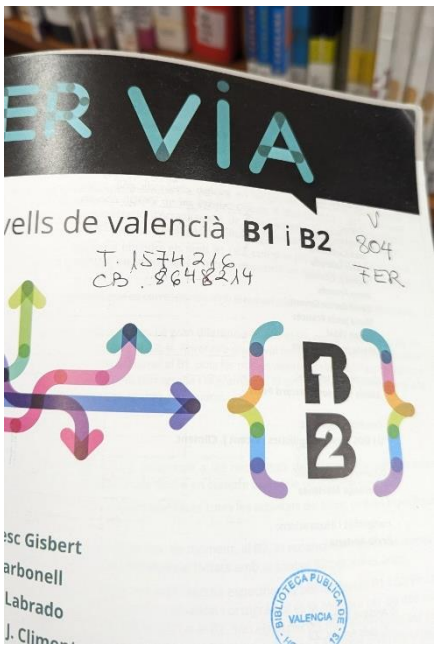


figura 17. Nuevo código de registro de la Biblioteca de Valencia.

11.1.2.2 Biblioteca de la Universidad Politécnica de Valencia.

En el caso de la UPV, el personal técnico nos informó de que el CDU no había sido extrapolado a la signatura y que empleaban un sistema específico de la universidad. Es por este motivo por el que hemos considerado muy interesante incluir el sistema empleado por esta biblioteca en la memoria. Destacar la problemática surgida en la importación de datos, por parte del resto

de bibliotecas, dado que, el personal técnico de la biblioteca pública de Valencia, nos comentó que estaban teniendo muchos problemas al importar información de los diferentes ejemplares de la UPV, justamente por las particularidades de dicho sistema. Sobre un ejemplo explicaremos el sistema:

Si consideramos la signatura B-0-11/02587, la B se refiere al edificio donde se encuentra el ejemplar (en este caso Biblioteca Central), el 0-11 se refiere a la librería donde se encuentra ubicado y el código 2587 se refiere al lugar dónde está ubicado dentro de la librería. El lugar dentro de la librería es un número consecutivo que se asigna por orden de llegada. Tradicionalmente las etiquetas poseían un color que indicaba el lugar dentro de dicha biblioteca dónde estaba ubicado el ejemplar (algo parecido al lugar por temática), pero nos comentaron que esto había cambiado y que estaba en desuso).

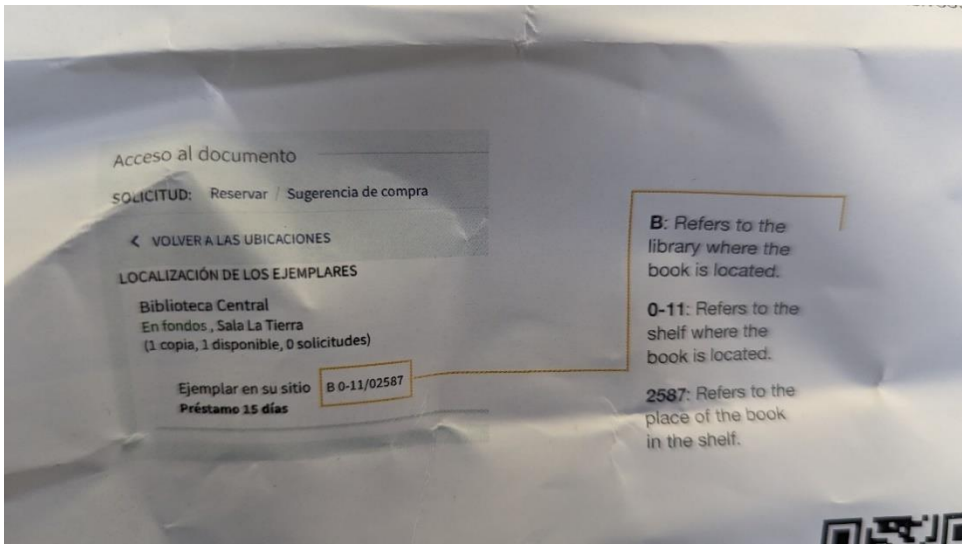


figura 18. Detalle del sistema de codificación empleado para la Signatura. UPV.

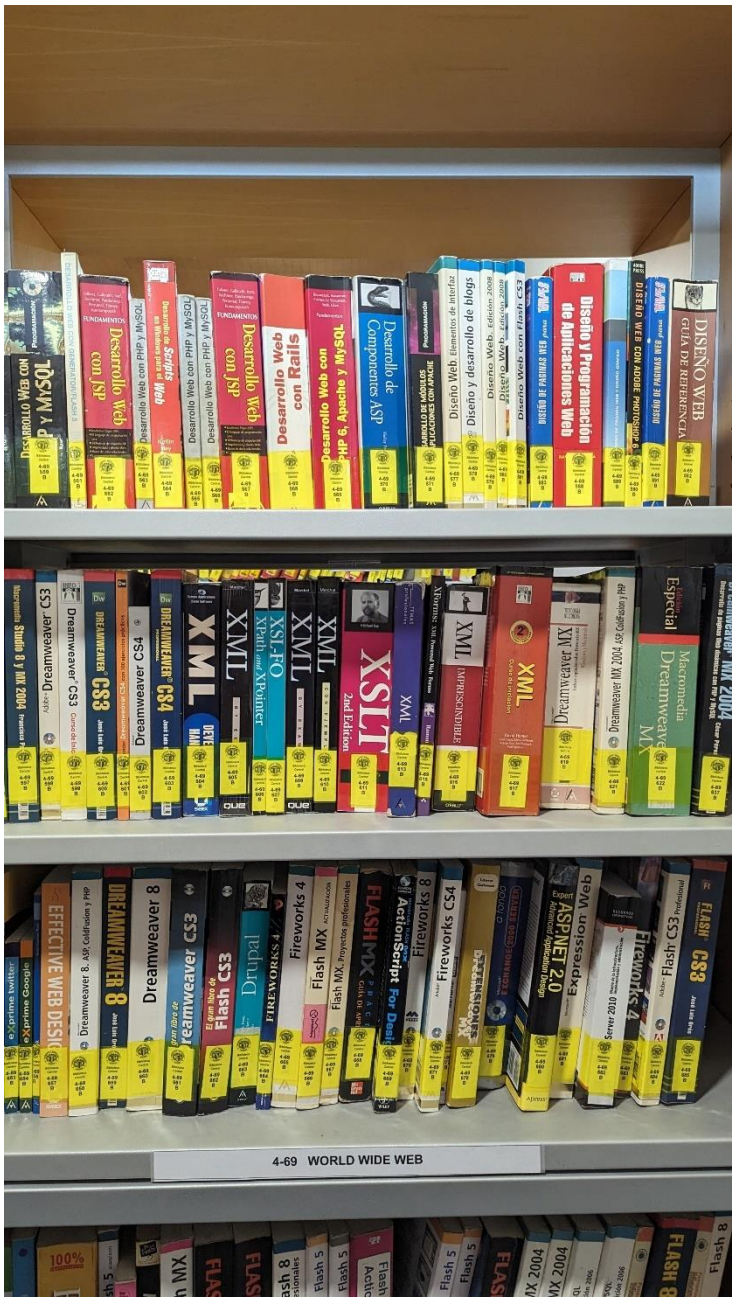


figura 19. Ejemplo de librería presente en la Biblioteca Central de la UPV.

11.1.2.3 Biblioteca de Ciencias Sociales de la Universidad de Valencia.

figura 20. Hall de la Biblioteca de Ciencias Sociales de la Universidad de Valencia.

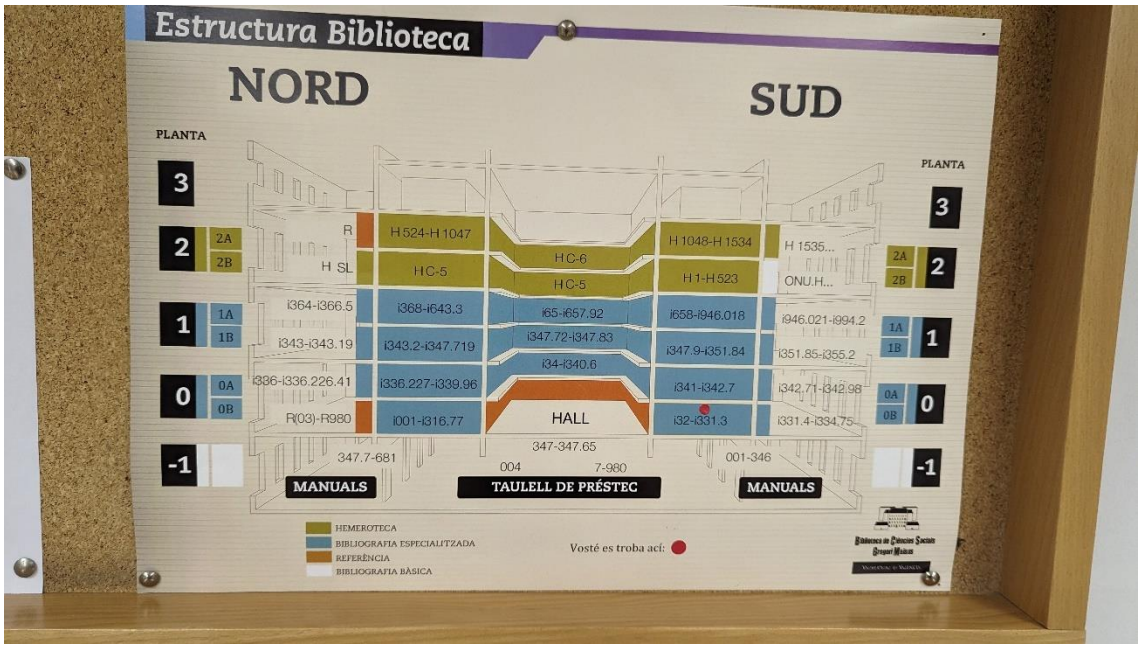


figura 21. Estructura de la biblioteca de ciencias sociales con las siglas añadidas a la signatura en función del tipo de material.

Como se puede observar en la figura 22, en la signatura de los diferentes ejemplares se incluye el DCU, pero en lugar de como en el caso de la biblioteca de Valencia, solo incluyen tres letras en mayúsculas, que en ocasiones se corresponden con las tres primeras letras del apellido del autor y en otras todo apunta que se refiere a las tres primeras letras del título.

figura 22. Librería de ejemplo. Muestra de codificación de signatura.

11.1.2.4 Biblioteca pública de Tavernes Blanques.

En esta ocasión el sistema es muy similar al empleado por la Biblioteca de Valencia, dado que esta biblioteca está adherida a la red pública de bibliotecas de Valencia. Se incluye el CDU, las tres letras del apellido del autor y las tres letras del título. También emplean la letra N para novela y algunas letras que identifican de manera particular en la Biblioteca como puede ser JV para Juvenil, I para infantil, IV para infantil valenciano, etc. Del mismo modo en la zona infantil, los ejemplares poseen etiquetas de colores que identifican el rango de edad para el que los ejemplares están recomendados. Se incluyen algunas imágenes tomadas en la biblioteca para que se puedan observar los diferentes sistemas de codificación.

figura 23. Biblioteca municipal de Tavernes Blanques.

figura 24. Biblioteca Municipal de Tavernes Blanques. Clasificación por temática.

figura 25. Biblioteca Tavernes Blanques. Clasificación DVDs. I, infantil. A, Adultos.

figura 26. Biblioteca Tavernes Blanques. Clasificación por código de colores.

11.1.2.5 Biblioteca Instituto de Educación Secundaria Patacona.

La consulta sobre el catálogo de dicha biblioteca la hemos realizado accediendo al OPAC del servidor PMB que han puesto en línea para su libre consulta. Se muestran a continuación una serie de imágenes que reflejan algunas de las consultas realizadas. Podemos observar en estas, en primer lugar un uso real de PMB (sistema que nosotros hemos empleado para nuestro proyecto) y un sistema de catalogación muy similar al de la Biblioteca de Valencia.

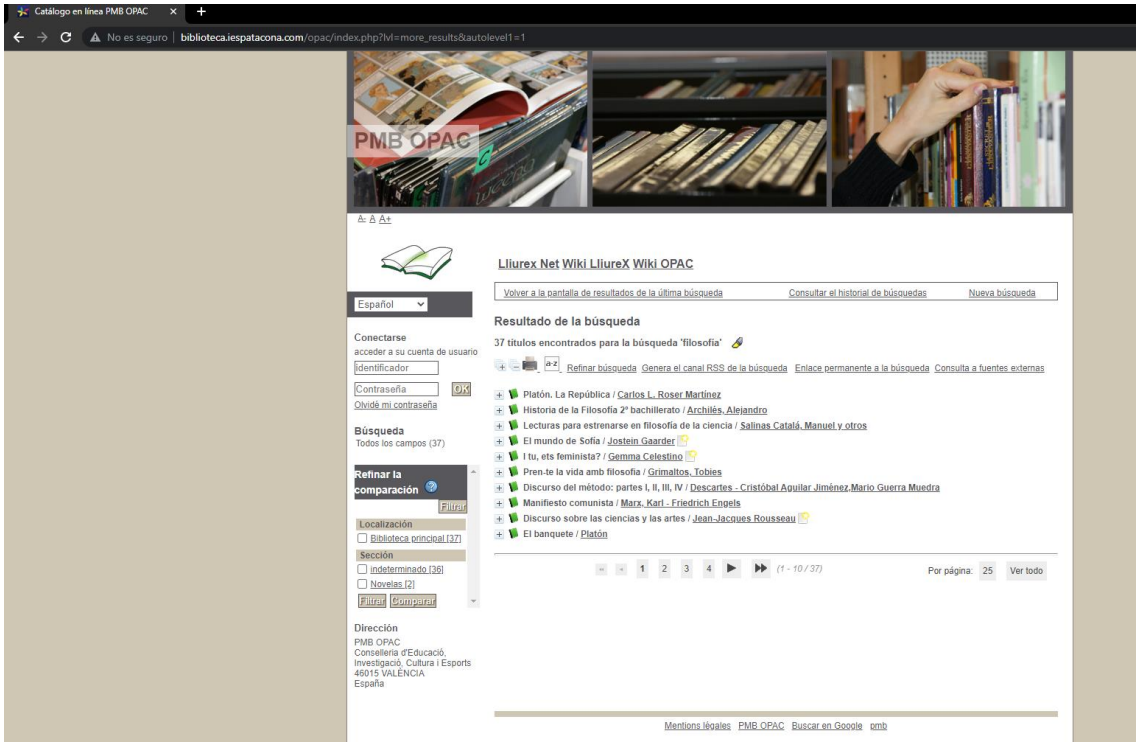


figura 27. Consulta sobre el servidor PMB del Instituto "La Patacona".

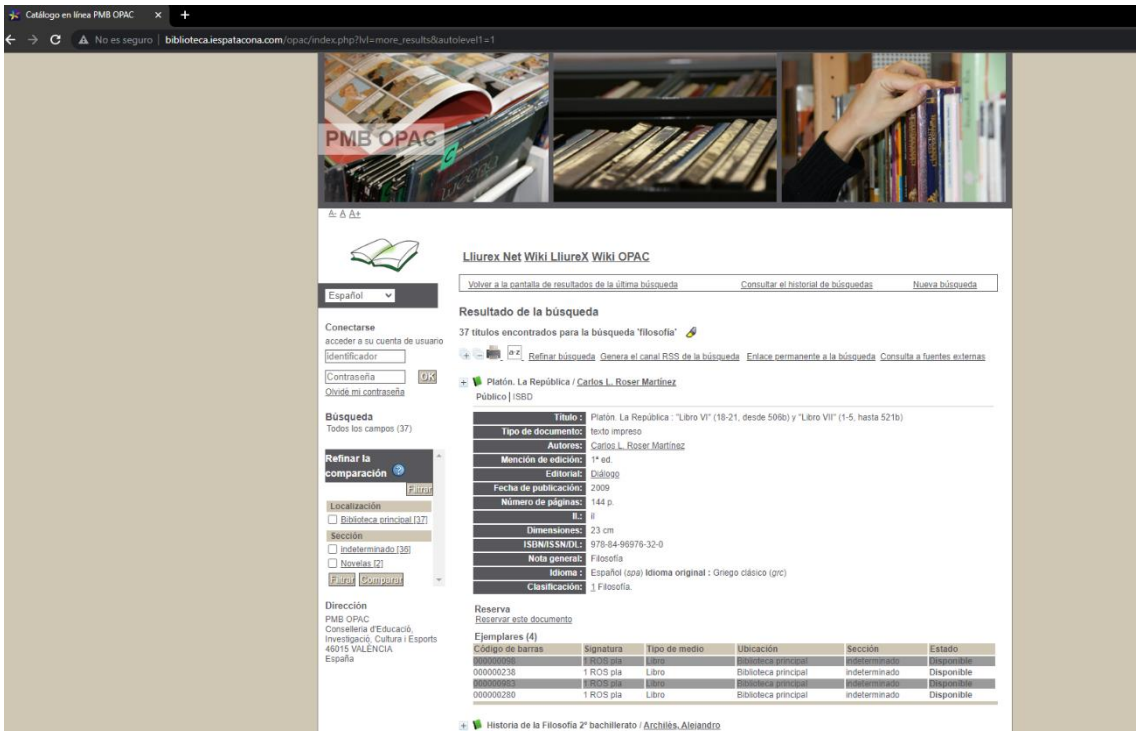


figura 28. Búsqueda sobre el servidor PMB del instituto de la Patacona. Patrón de búsqueda, "filosofía".

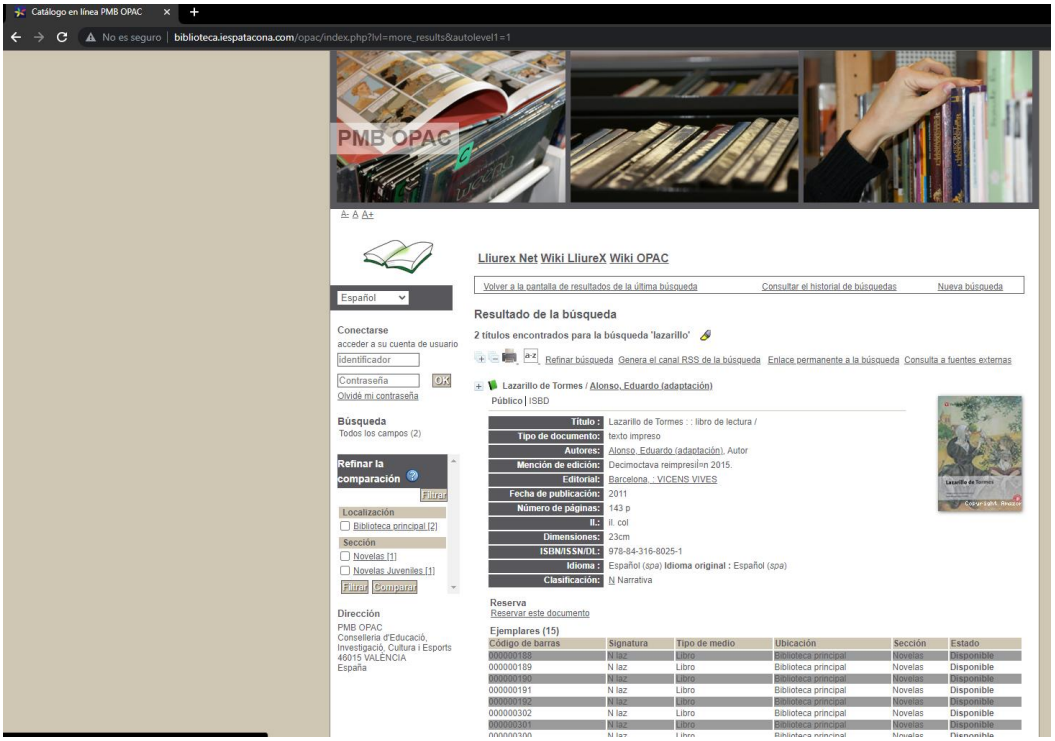


figura 29. Búsqueda sobre el servidor PMB de la Patacona. Patrón de búsqueda, "lazarillo".

12 Infraestructura Empleada.

12.1 Desarrollo y pruebas de la aplicación.

12.1.1 MySQL + Apache Web Server con PHP (XAMPP).

En pro de simplificar la instalación y configuración de los diferentes elementos para el entorno de desarrollo, nos hemos apoyado en la herramienta

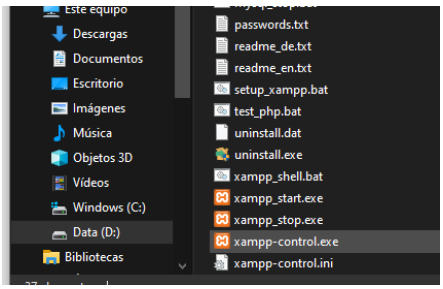


figura 31. Launcher panel de control XAMPP.

XAMPP. Esta integra entre otras herramientas, el servidor Web Apache junto al SGBD MySQL, facilitándonos la tarea de inicio y parada de servicios de forma gráfica. Para ello hemos de ejecutar el panel de control de XAMPP (figura 31) y posteriormente iniciar los servicios que nos interesen.

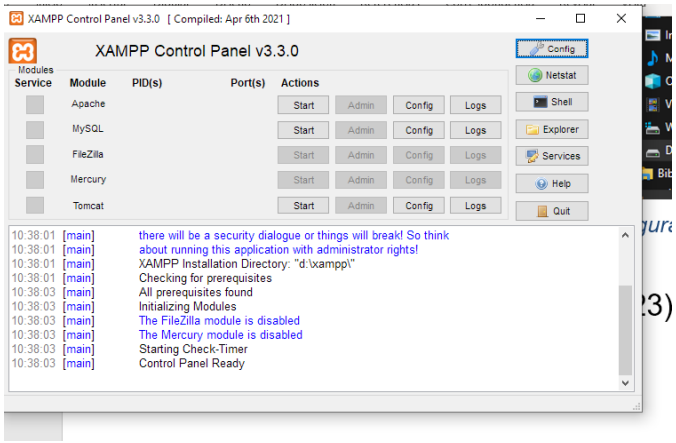


figura 30. Panel de control de XAMPP.

Del mismo modo, el panel de control incluye acceso a la consola, ficheros de configuración, ficheros de log, etc. Una vez instalados e iniciados los servicios, podremos acceder a la administración y uso de los mismos mediante el navegador web.

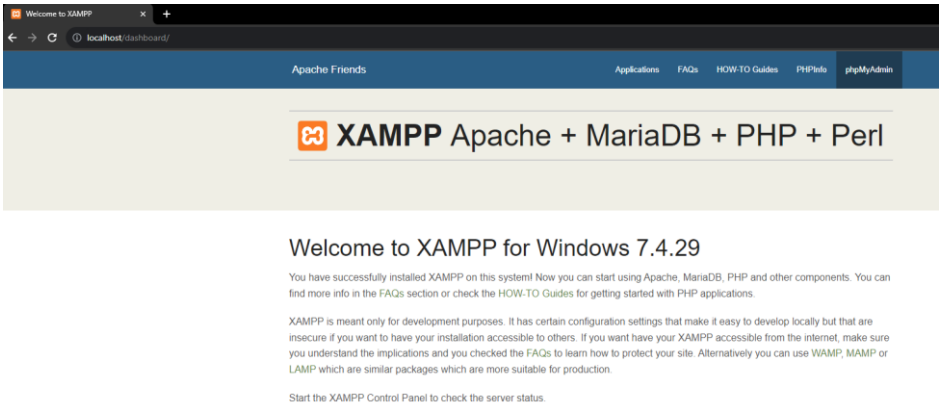


figura 32. Página principal del servidor local instalado.

Al acceder a la página principal del servidor local, se nos ofrecen algunas herramientas interesantes, cómo “PHPInfo” que nos proporciona información sobre el soporte y versión de PHP disponible y “phpMyAdmin” que nos permite administrar las bases de datos MySQL.

figura 33. Acceso a PHPInfo en el servidor local.

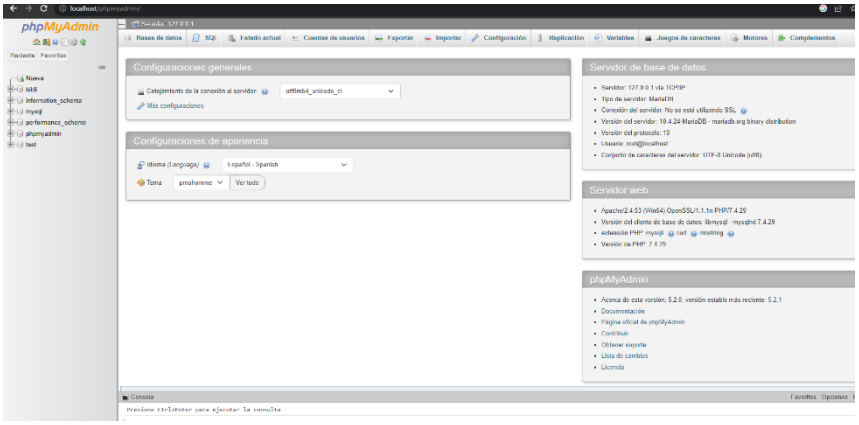


figura 34. Acceso al administrador de MySQL en servidor local.

Denotar también, la utilidad de las “How-to guides”, con información tan útil como la de configuración de una versión de PHP distinta.

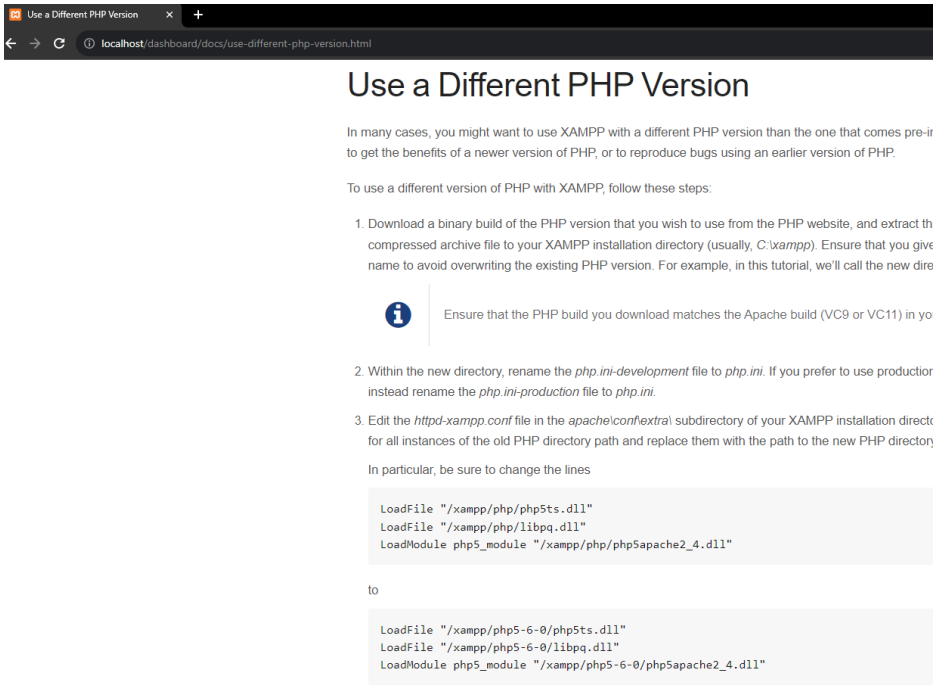


figura 35. Guía de configuación de la versión de php empleada por el servidor local.

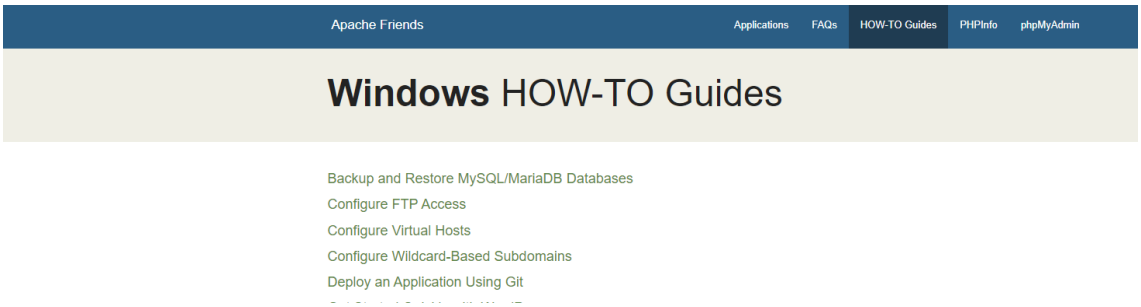


figura 36. HOW-TO Guides en servidor local.

Es importante tener en cuenta que para poder conectar nuestra aplicación con la base de datos MySQL, hemos tenido que descargar la librería asociada al conector correspondiente (fichero .jar) y añadirla a nuestro proyecto (en la carpeta lib del mismo).

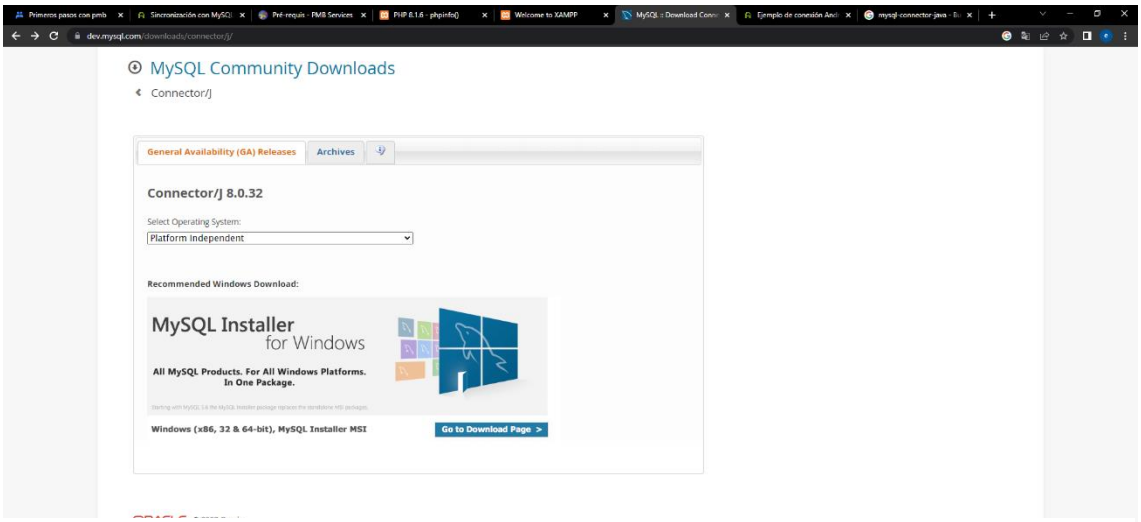


figura 37. Descarga del conector necesario para conectarse a la base de datos MySql desde nuestra aplicación.

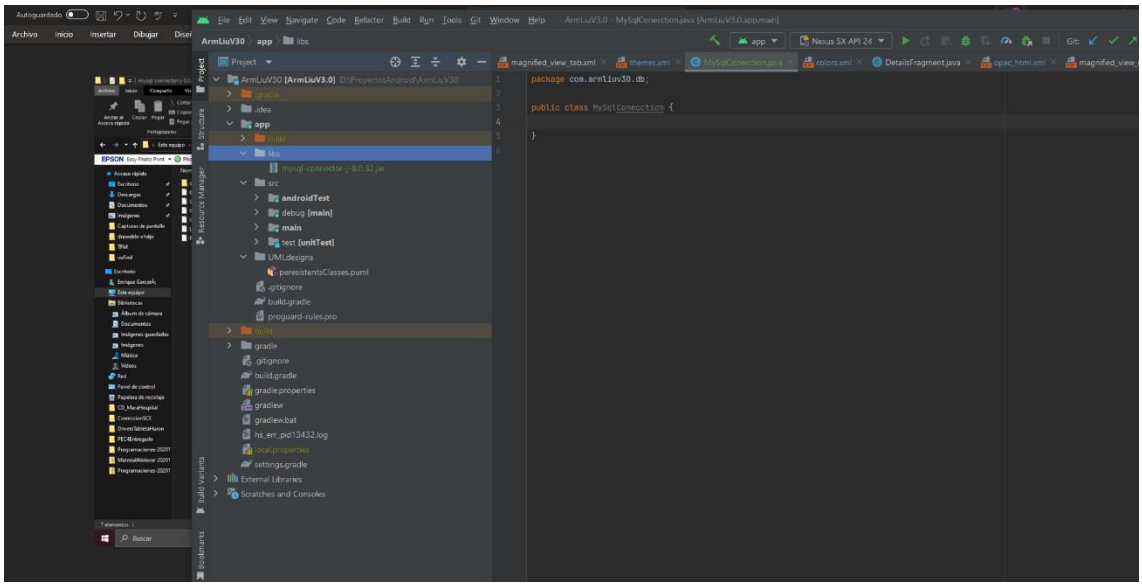


figura 38. Inserción de la librería de conexión a bases de datos MySql en Android Studio.

12.1.2 PMB (PhpMyBibli).

PMB hace uso del servidor Web Apache para ofrecernos la interfaz de administración con diferentes perfiles de usuario y poner a disposición el OPAC de la biblioteca, y de MySQL para almacenar, administrar e interactuar con la base de datos sobre la que se sustenta la persistencia de datos del mismo. En el “

Anexo I. Instalación y configuración de PMB.”, se ofrece una pequeña explicación del proceso de instalación seguido y los problemas más importantes con los que nos hemos encontrado. En el presente apartado se describen los aspectos más relevantes en cuanto a la administración del contenido presente en PMB.

Al acceder al servidor PMB mediante el navegador, tenemos dos opciones:

- Acceso al módulo de gestión para administrar el sistema o realizar préstamos entre otros (figura 39).
- Acceso al catálogo publico suministrado mediante el servidor OPAC integrado en PMB (figura 40).

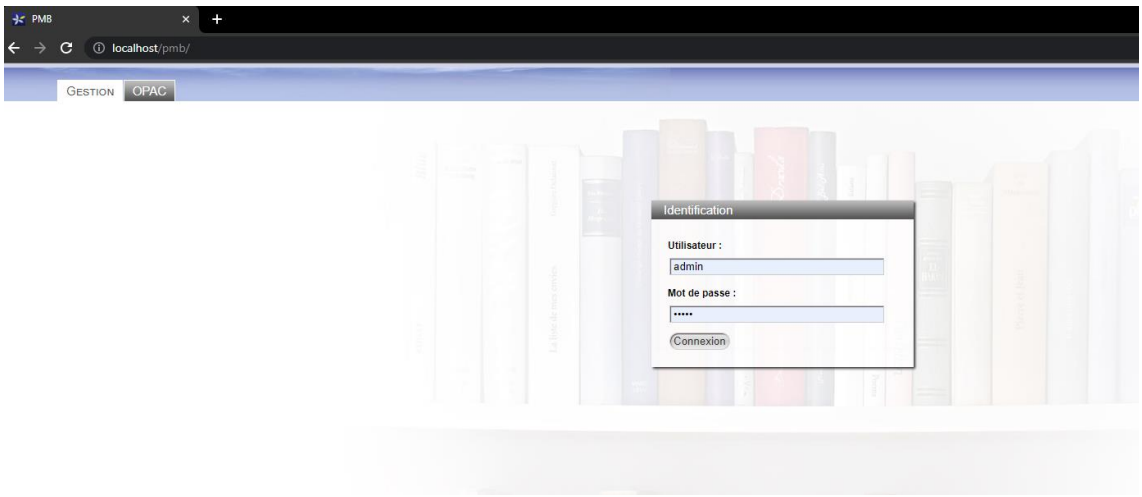


figura 39. Acceso al módulo de gestión de PMB empleando la url: host/pmb.

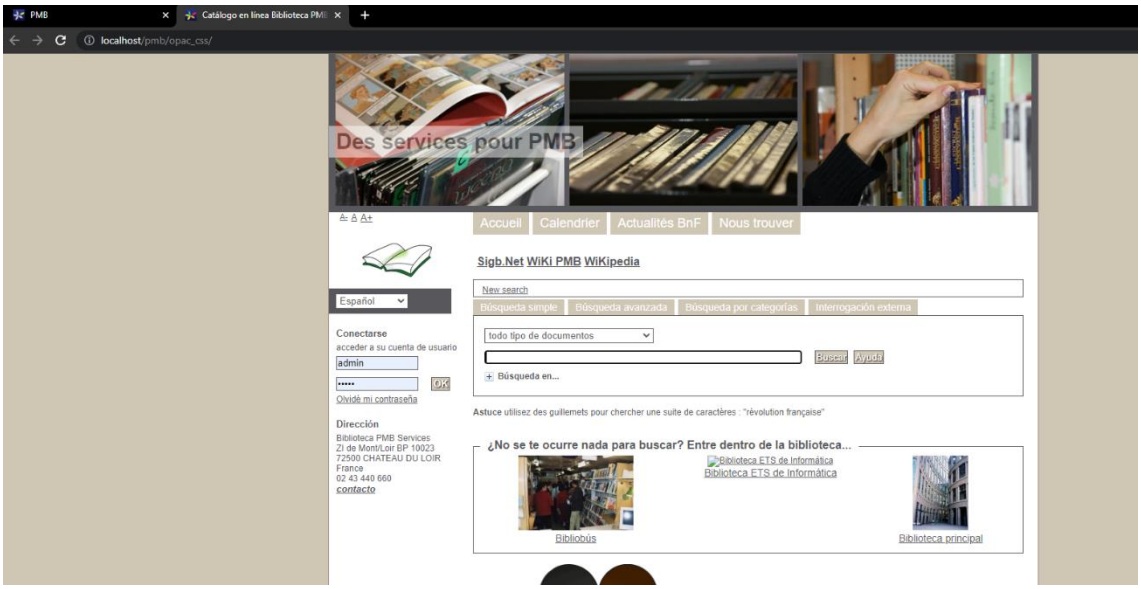


figura 40. Acceso al catálogo OPAC de PMB, empleando la URL: host/pmb/opac_css

12.1.2.1 Módulo de gestión de PMB.

El módulo de gestión de PMB nos permite realizar tareas como: el alta y baja de usuarios, la creación de registros y ejemplares, la gestión de los prestamos bibliotecarios, etc. En los siguientes subapartados se expone brevemente el modo de proceder para realizar las tareas más comunes.

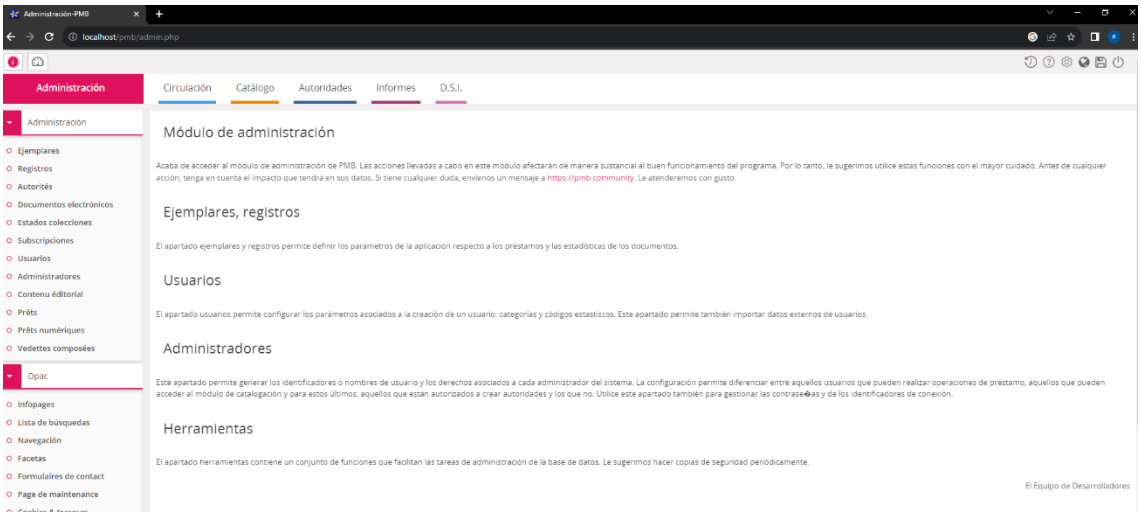


figura 41. Página principal del módulo de gestión. Perfil usuario administrador.

Como se puede observar en la figura 41, existen seis pestañas sobre las que se agrupan múltiples posibilidades de administración de usuarios, registros y ejemplares. En los siguientes subapartados, se describirán brevemente cada una de ellas a través de las funcionalidades principales de PMB.

12.1.2.1.1 Administración de usuarios.

La pestaña Administración nos permite crear, modificar y borrar, registros, ejemplares sobre registros y usuarios. Es importante distinguir los tipos de usuarios que podemos crear en PMB. Por un lado, estarán los **usuarios**

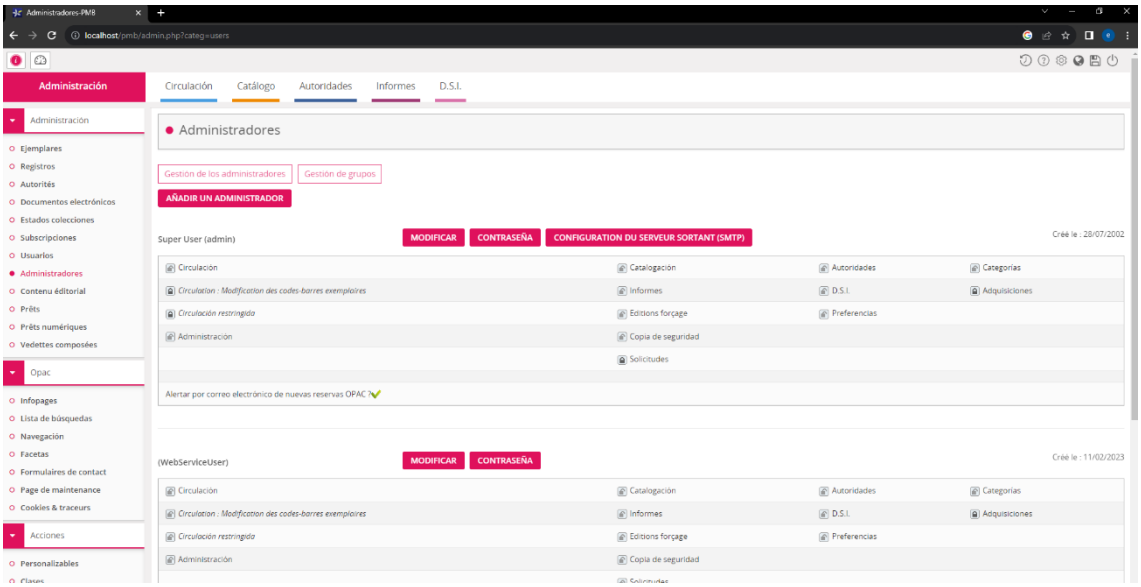


figura 42. Usuarios administradores de PMB.

convencionales (cuenta lector) de la biblioteca sobre los que se podrán realizar préstamos y devoluciones de ejemplares bibliográficos y por otro, los usuarios **administradores** de la misma.

Si nos situamos en la pestaña administración sobre la opción de menú administradores, como podemos observar en la figura 42, nos encontramos con el usuario “admin” que nos proporciona PMB por defecto y el usuario “WebServiceUser” creado por nosotros mismos. También disponemos de la posibilidad de crear y administrar usuarios lectores, seleccionando la opción de menú usuarios.

Para crear un nuevo usuario lector, debemos seleccionar la pestaña “Circulación” y la opción de menú “Nuevo usuario”. Indicamos un número para el código de barras (nos propone el siguiente al último creado), dado que, cada

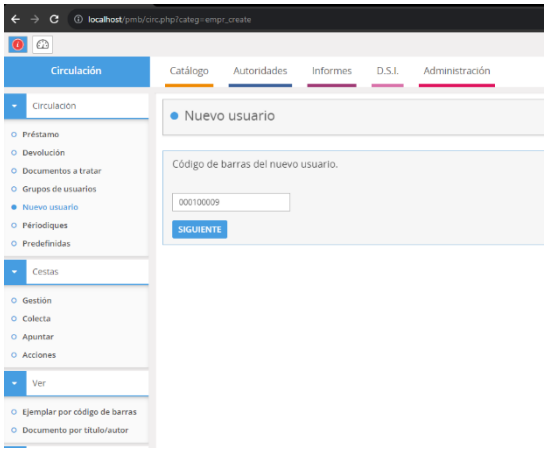


figura 43. Creación de usuario lector.

usuario se asocia a uno y seguidamente introducimos los datos del nuevo usuario respetando aquellos definidos como obligatorios. Denotar que los campos obligatorios han sido definidos en la opción de menú: Administración → usuarios → cuenta lector, como se puede observar en la figura 45. Algunos aspectos interesantes del formulario de creación de un nuevo usuario lector, es que podemos añadir un usuario a un grupo de usuarios que facilitará posteriores consultas y que nos permite definir si el usuario tendrá acceso al OPAC definiendo el identificador, la contraseña y el lenguaje asociado a dicho acceso.

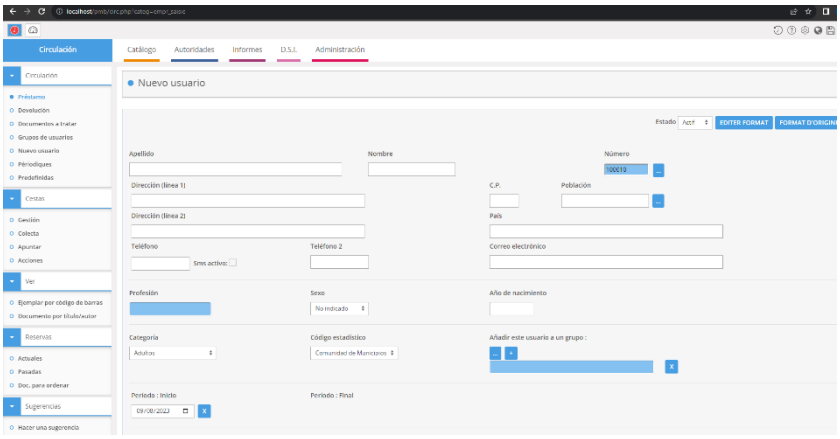


figura 45. Creación de un usuario lector. Formulario de alta de usuario.

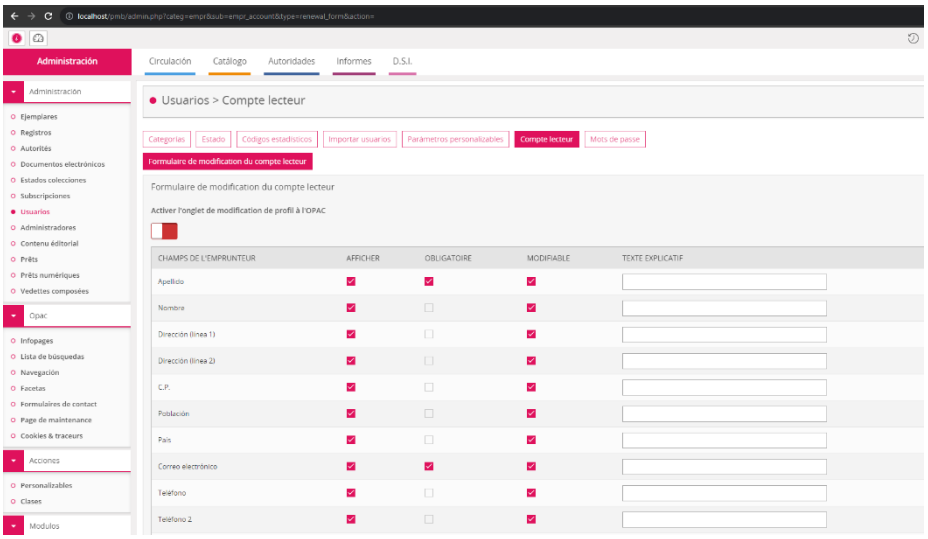


figura 44. Campos presentes en la creación de usuarios lectores.

Denotar que la categoría asociada a un usuario lector deberá haber sido creada previamente (figura 46).

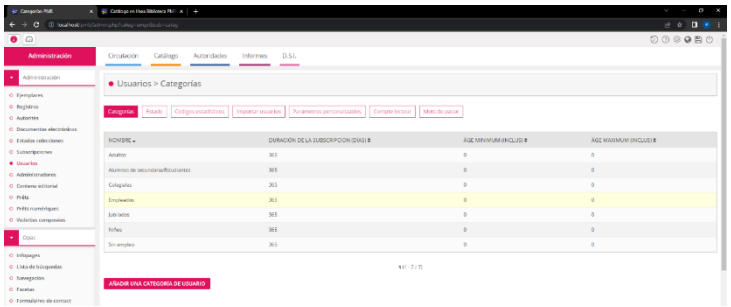


figura 46. Administración de categorías de usuarios lectores en PMB.

Para obtener información de los usuarios existentes, podemos acceder a la pestaña “Informes” y seleccionar de las opciones de menú de la izquierda la que pone “Usuarios”. Sobre este grupo de opciones de menú podremos obtener información detallada sobre los usuarios actuales, abonos vigentes o caducados, etc.

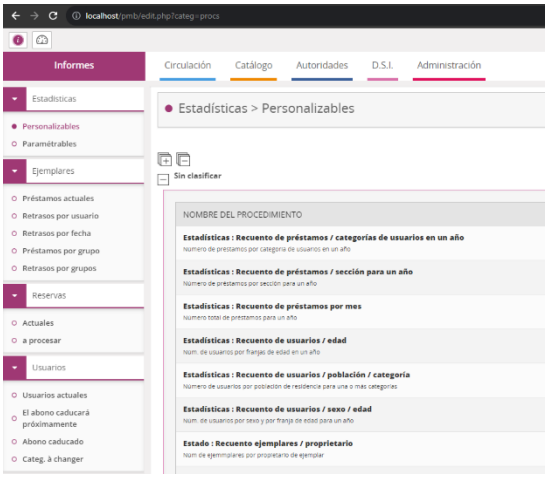


figura 49. Pestaña informes de PMB.

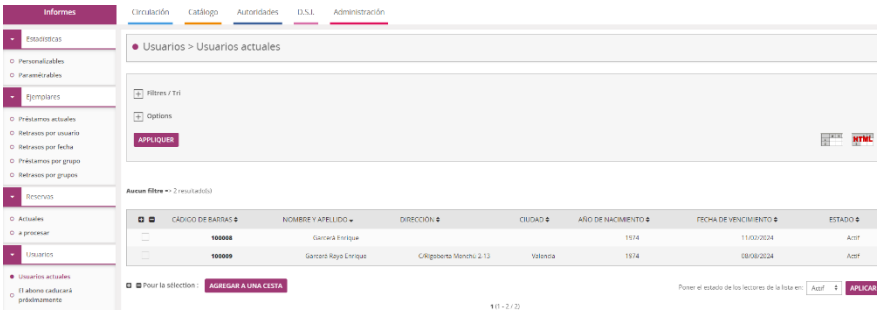


figura 48. Informe de usuarios lectores en PMB

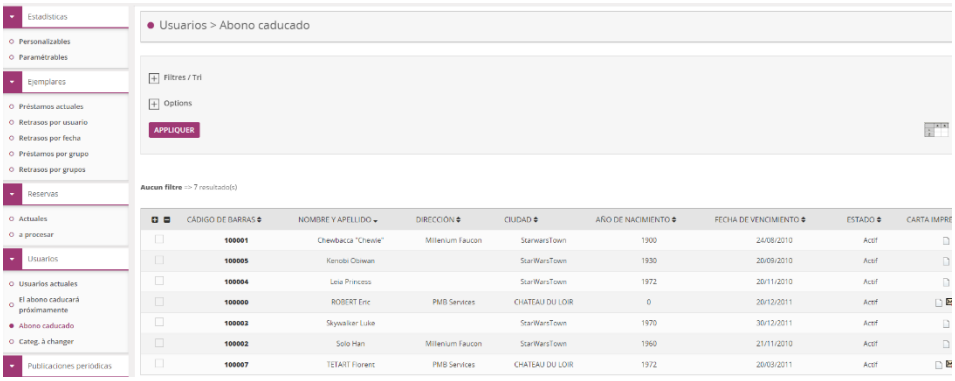


figura 47. Listado de usuarios lectores con abono caducado en PMB.

12.1.2.1.2 Administración del material bibliográfico.

La pestaña “Catálogo” dispone de múltiples opciones que nos permiten administrar el contenido bibliográfico de nuestra biblioteca. Es importante destacar que PMB distingue entre **registro** y **ejemplar**. Un registro es la definición de la obra en sí, a la que se le asocia un ISBN, un depósito legal, unos autores/as, etc., mientras que un ejemplar está asociado a un registro y se corresponde con el soporte físico o lógico que podemos consultar, prestar, etc. A un registro se le pueden asociar múltiples ejemplares y para que exista un ejemplar, debe existir previamente un registro al que ser asociado.

12.1.2.1.2.1 Administración de registros.

Podemos crear un registro accediendo a la pestaña catálogo y seleccionando la opción de menú “Nuevo registro”. Esta acción en primer lugar nos pedirá el ISBN o EAN asociado a la obra que deseamos registrar.

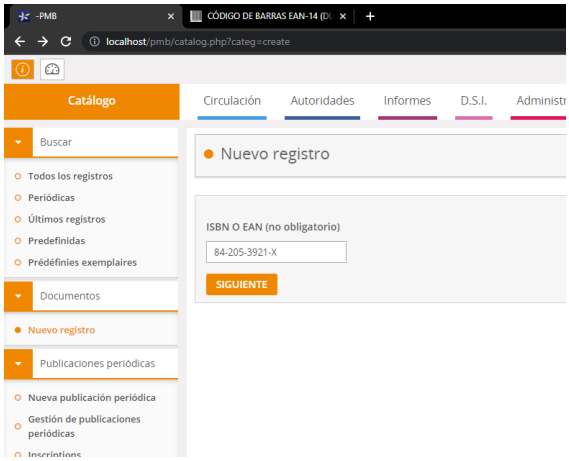


figura 50.Creación de un nuevo registro en PMB. Introducción del ISBN.

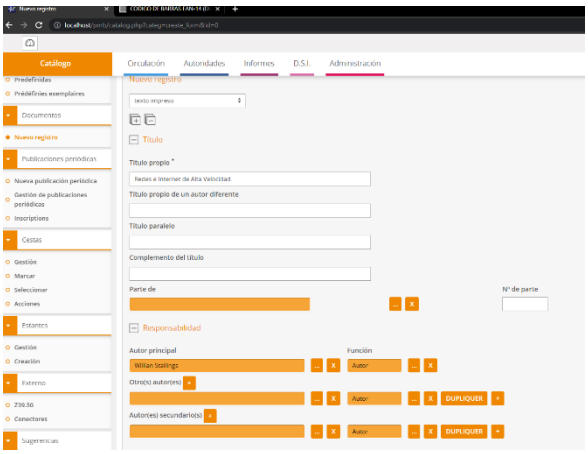


figura 51. Formulario de creación de un nuevo registro en PMB.

En la figura 52, se puede observar el resultado después de la creación de un registro.

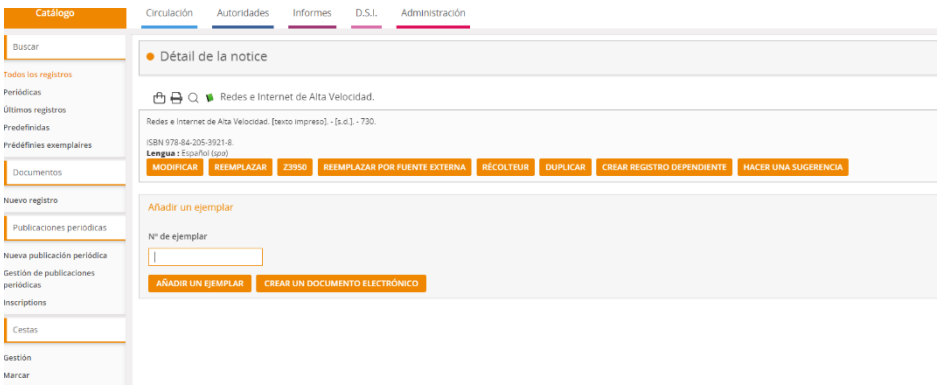


figura 52. Consulta de un registro bibliográfico en PMB.

Denotar que hemos creado el registro bibliográfico, pero de momento no hemos añadido ninguno de los ejemplares que acabarán en la estantería. En el apartado 13.3.3, se encuentra una pequeña explicación de cómo administrar los ejemplares.

Uno de los datos asociados a un registro es el de autor/es. En la pestaña “Autoridades” podemos administrar (figura 53) los mismos para su posterior uso (figura 54).

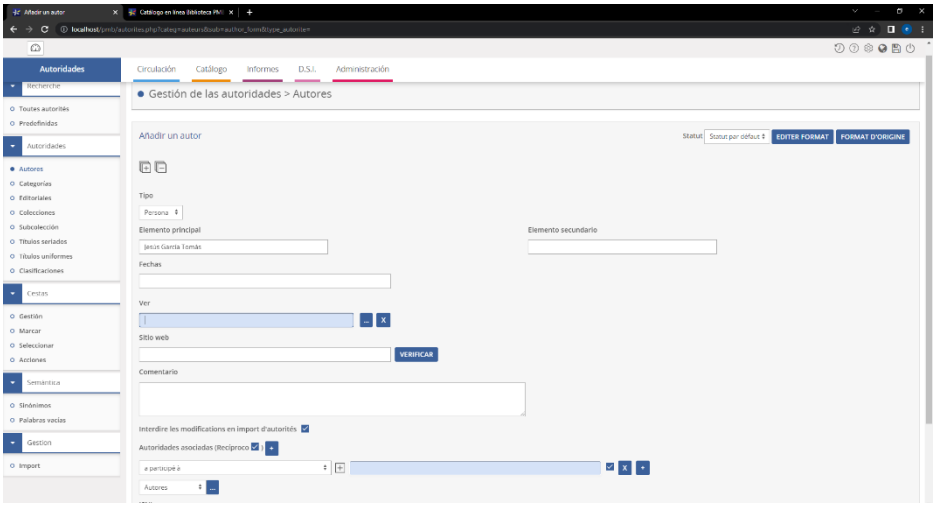


figura 53. Creación de un autor bibliográfico en PMB.

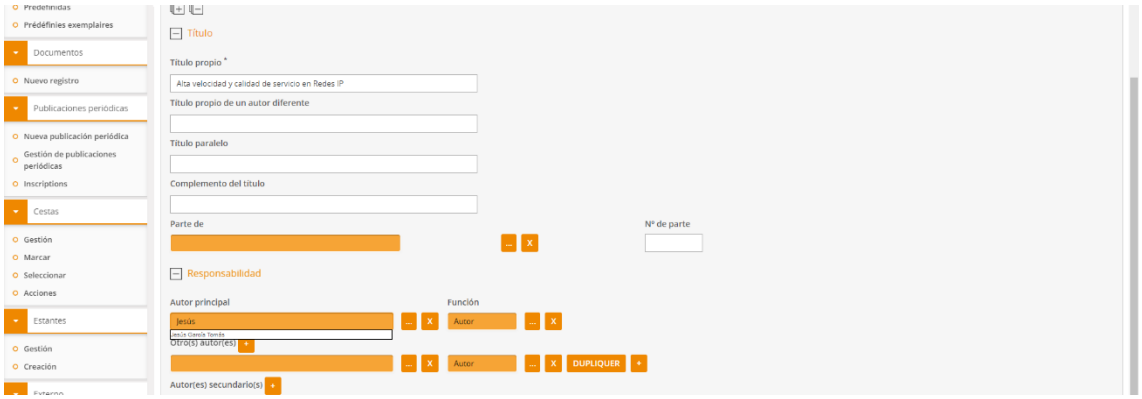


figura 54. Asignación de un autor a una obra en PMB.

En la figura 55, podemos observar como un registro puede tener un estado, en esta mostramos el ejemplo del préstamo rápido que se pondrá a disposición de los ejemplares de dicho registro. Podemos identificar dicho registro mediante la selección de un color, añadir una etiqueta identificativa en el OPAC, hacer que sea visible o no, etc.

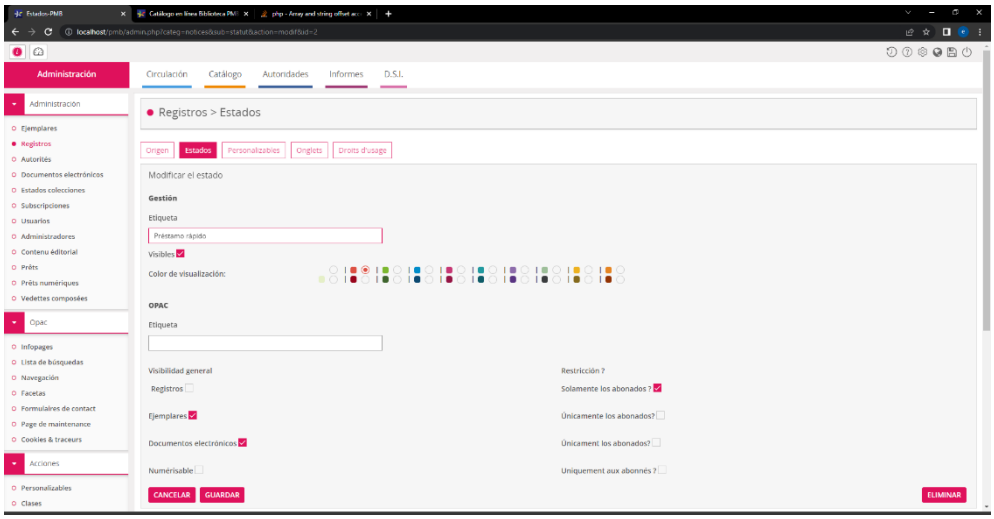


figura 55. Estados de un registro bibliográfico en PMB.

12.1.2.1.2.1.1 Importación de datos de registro de fuentes externas.

PMB nos permite obtener información sobre la catalogación de registros de servidores z3950 externos para completar los datos, o reemplazar por una fuente externa, o modificar o crear un nuevo ejemplar de dicho registro. Esto simplificará enormemente la tarea de introducción de la información de cada registro en caso de que se encuentre disponible en una fuente externa. Al intentar utilizar este servicio en PMB nos ha dado error (figura 56) y al perseguir el mismo, hemos descubierto que había que instalar y configurar la librería de PHP “yaz”.

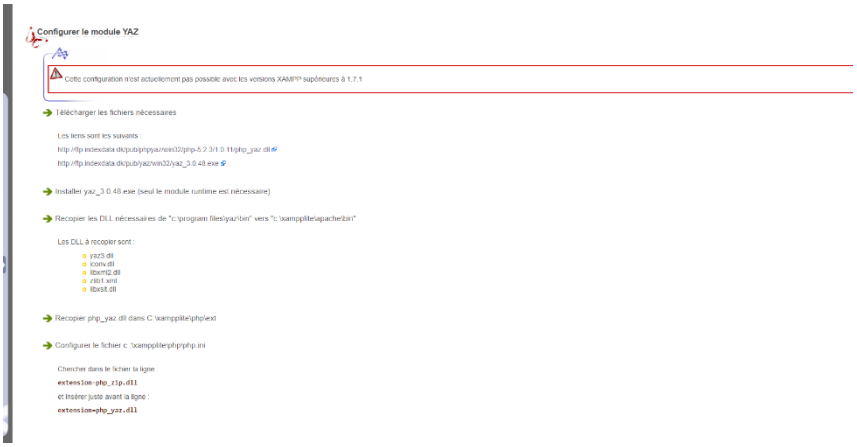


figura 56. Error vinculado al requisito de la librería PHP YAZ.

Descargamos los ficheros de yaz asociados a nuestra versión de PHP para su instalación.

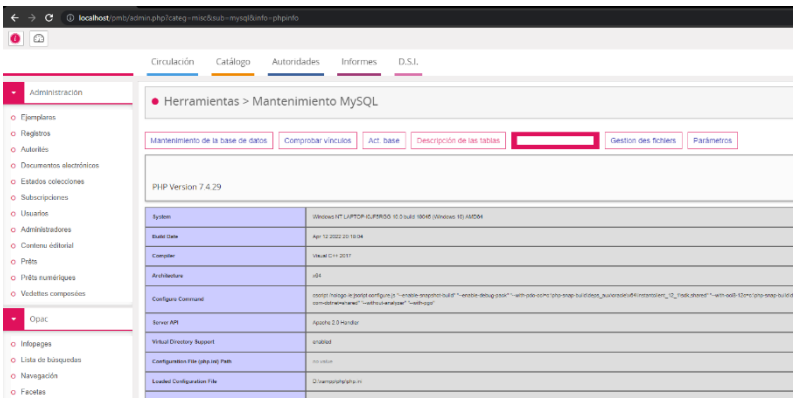


figura 57.Versión de PHP del servidor local.

figura 59. Descarga de ficheros para instalar YAZ.

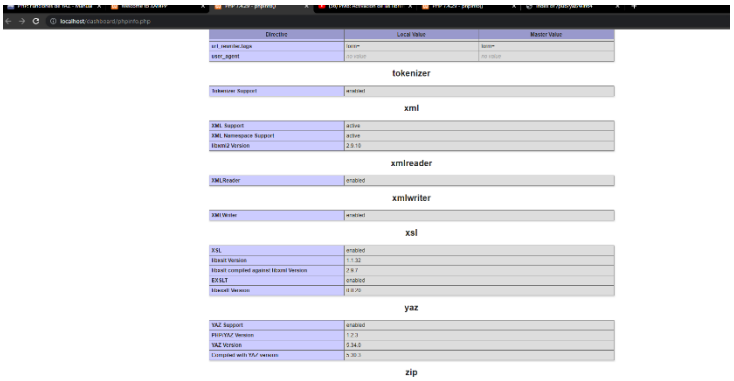


figura 58. Acceso a php info para comprobar que yaz ha sido activado.

De los ficheros descargados, copiamos “libxml2.dll”, “libxslt.dll y “yaz5.dll en la carpeta “\xampp\php” de nuestro servidor local. Y el fichero “php_yaz.dll” en la carpeta \xampp\php\ext. Después, es necesario editar el fichero “php.ini” y añadir la extensión yaz. Es muy importante omitir en el texto “extension=yaz” la extensión .dll que indicaba el manual de instalación oficial (PHP YAZ., 2023), dado que de esa forma no funciona.

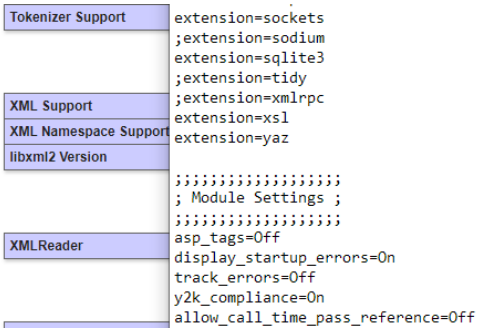


figura 60. Edición del fichero php.ini

En la figura 61, podemos observar la búsqueda realizada sobre los servidores externos z3950 que tenemos registrados en nuestro servidor PMB sobre el registro “Redes e internet de alta velocidad” que previamente habíamos creado pero con escasa información del mismo.

figura 61. Respuesta servidores z3950 de nuestro sistema PMB.

En la última línea de la figura podemos observar como el servidor que hemos registrado de la Universidad de Valencia ha devuelto un registro, y si pulsamos sobre esa línea accedemos a la información del registro disponible para ser importado.

figura 62. Resultado búsqueda servidor z3950 de la Universidad de Valencia.

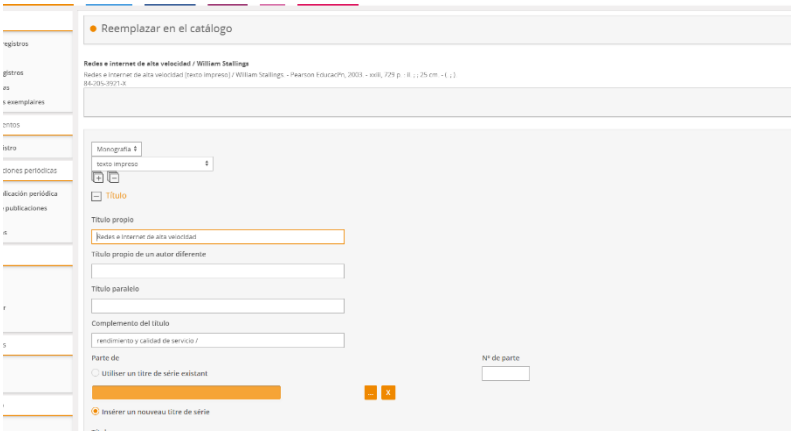


figura 65. Reemplazo de la información de un registro mediante servidor externo en PMB.

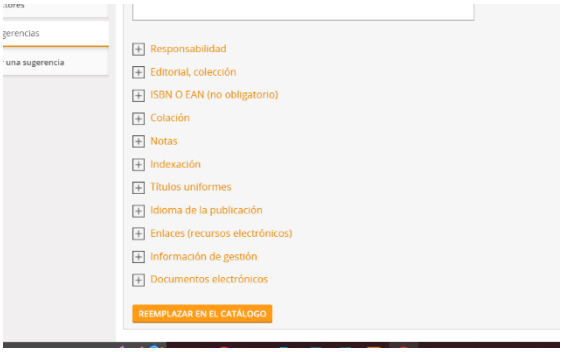


figura 64. Reemplazo de un registro en PMB.

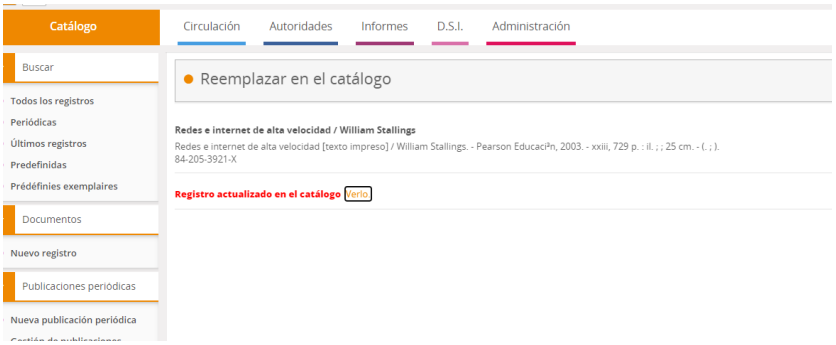


figura 63. Resultado del reemplazo de un registro en PMB.

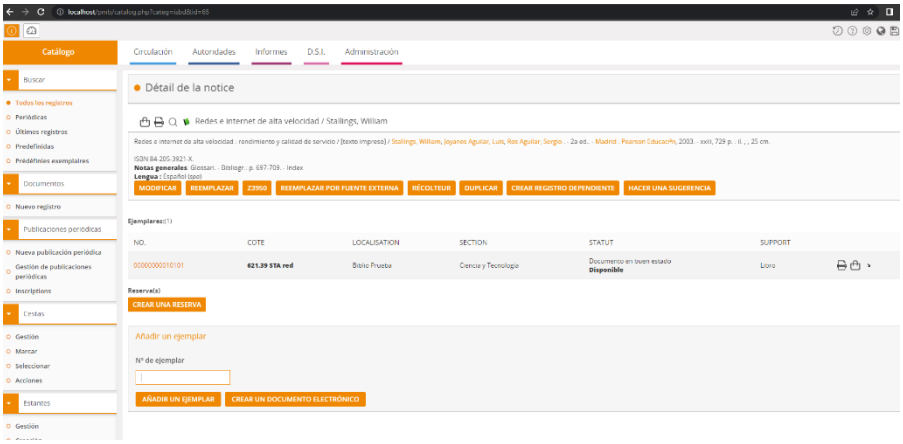


figura 66. Consulta del resultado de la importación de datos de un servidor z3950.

12.1.2.1.2.1.2 Inclusión servidor z3950.

Entre los servidores z3950 que hemos incorporado al servidor local se encuentra el de la Universidad de Valencia. Uno de los aspectos más importantes es obtener la información necesaria que permita la comunicación con el mismo. En este caso estos los podemos observar en la figura 67.

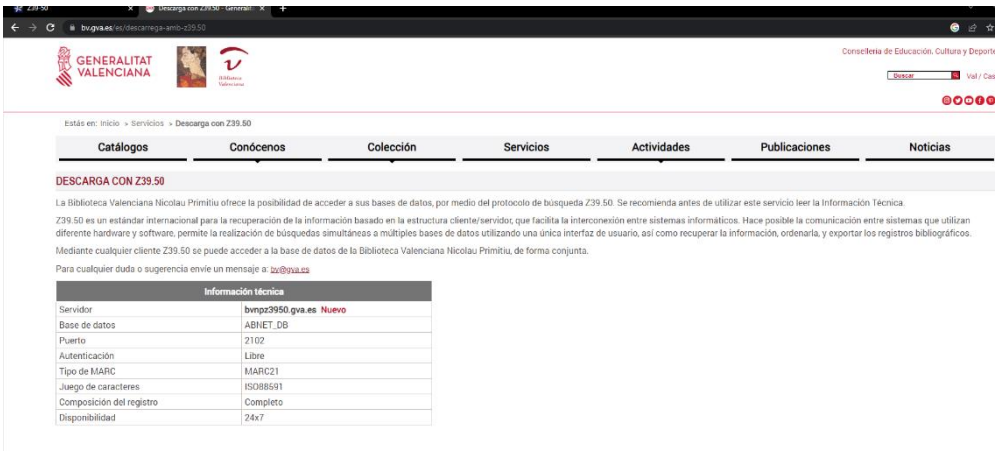


figura 67. Datos de configuración del servidor z3950 de la Universidad de Valencia.

Al acceder a la pestaña de administración, nos encontramos con la opción z3950 y podremos crear uno nuevo o editar alguno existente.

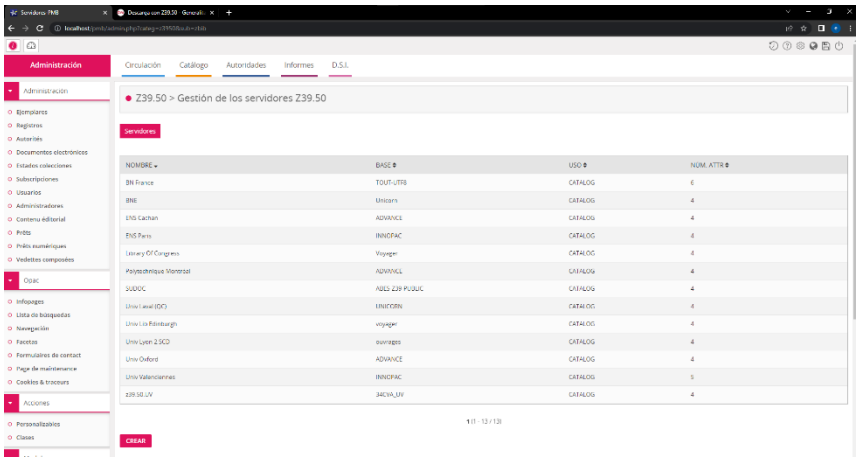


figura 68. Administración de servidores z3950 en PMB.

12.1.2.1.2.2 Administración de ejemplares.

Como se puede observar en la figura 52, sobre un registro se nos permite crear un ejemplar mediante el botón “Añadir ejemplar”. Si pulsamos sobre este botón, podemos observar cómo se nos ofrece la introducción de un número asociado al código de barras como campo obligatorio. Cada ejemplar es identificado por este código de barras en PMB. Al introducir este, se nos sitúa en el formulario de creación de un registro para que introduzcamos una serie de campos obligatorios y otros opcionales. Con respecto al código de barras hemos empleado un número de 14 dígitos que se corresponde con la normalización EAN-14(DUN-14) dado que nos ofrecerá múltiples veneficios apoyarnos en dicha normalización (por ejemplo, el empleo de lectores de códigos de barras estandarizados).

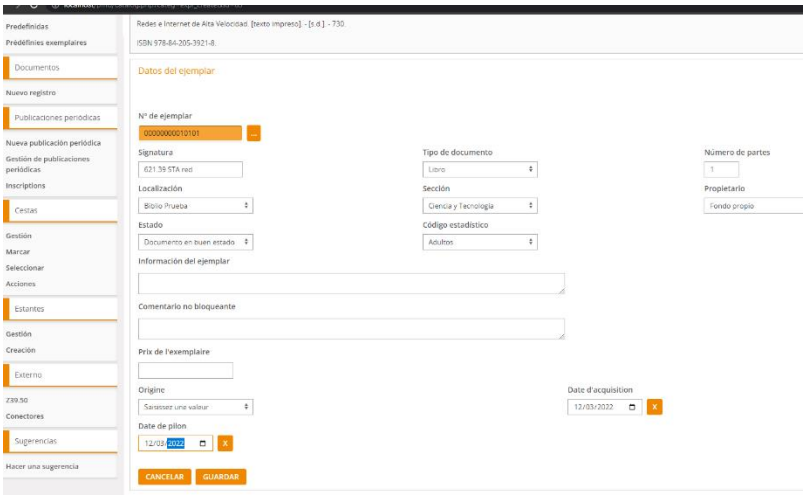


figura 69. Ejemplo creación ejemplar en PMB. Librería de pruebas.

Parte de la información importante de un ejemplar, es la de su localización (por ejemplo, la sala general de una biblioteca). En la figura 69, podemos observar como existe un campo localización con un desplegable que nos ofrecerá las disponibles y otro sección. Para que tanto localizaciones como secciones aparezcan en dichos desplegables, antes debemos haberlas creado mediante la pestaña de administración. En la figura 70, se puede observar algunas de las localizaciones disponibles al importar los datos de prueba de PMB y algunas creadas por nosotros mismo. En la figura 71, observamos la creación de la localización que hemos empleado para albergar los ejemplares de la librería de prueba. Un dato importante es que podemos decidir externalizar o no esa localización a través del servidor OPAC.

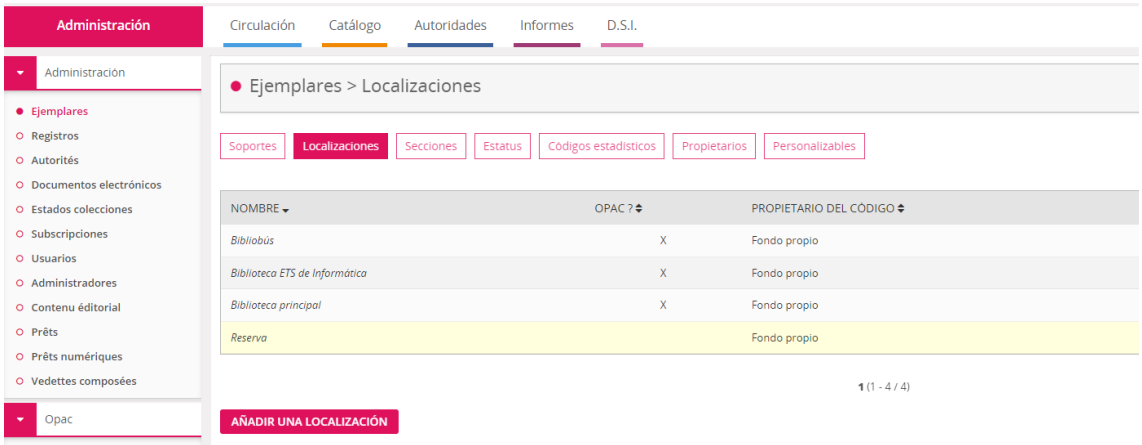


figura 70. Localizaciones disponibles en PMB.

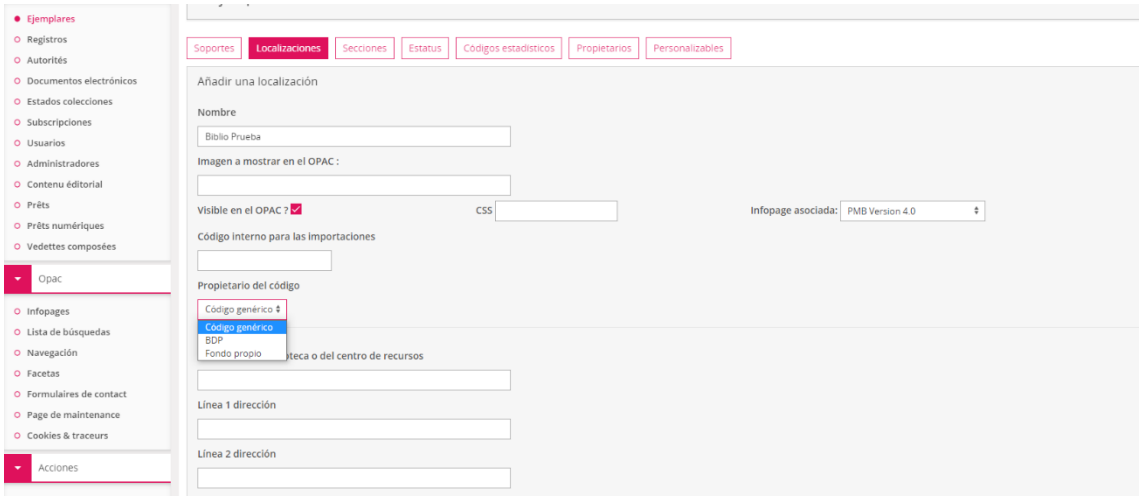


figura 71. Creación de la localización asociada a la biblioteca de prueba en PMB.

figura 72. Listado de localizaciones una vez creada la biblioteca de prueba en PMB.

Para poder usar una localización, es decir asignar un ejemplar a ella, por lo menos la localización debe contener alguna sección, dado que el ejemplar se asocia al binomio localización – sección dentro de la localización. En la figura 73, podemos observar la creación de una nueva sección “Ciencia y Tecnología”. En la parte inferior derecha de dicha figura, podemos observar como está activo “Biblioteca Prueba”. Esta es la manera de asociar una sección a una localización. Como podemos observar podríamos marcar más de una localización para dicha sección y por tanto estaría disponible para más de una.

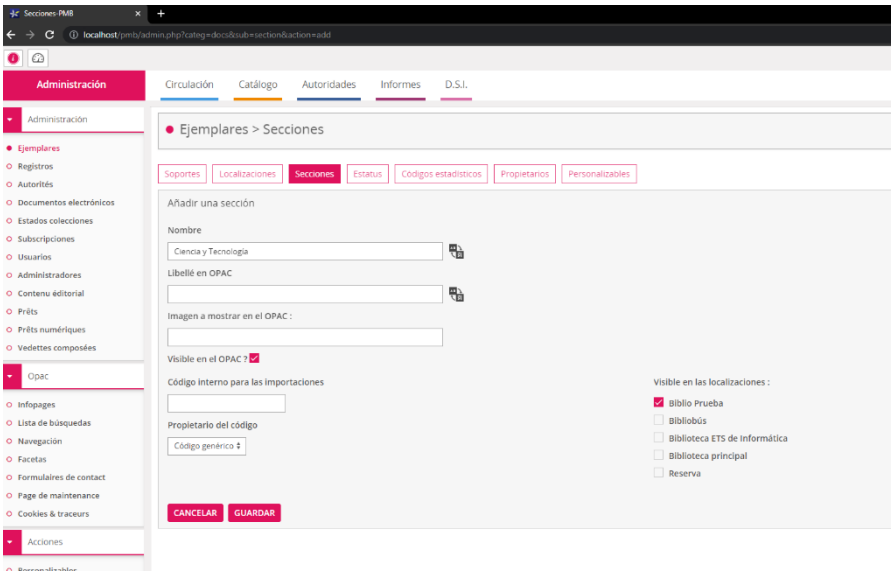


figura 73. Creación de una sección asociada a la biblioteca de prueba en PMB.

En la figura 74, podemos observar como la nueva localización y sección se encuentra disponible en el momento de modificar un ejemplar bibliográfico.

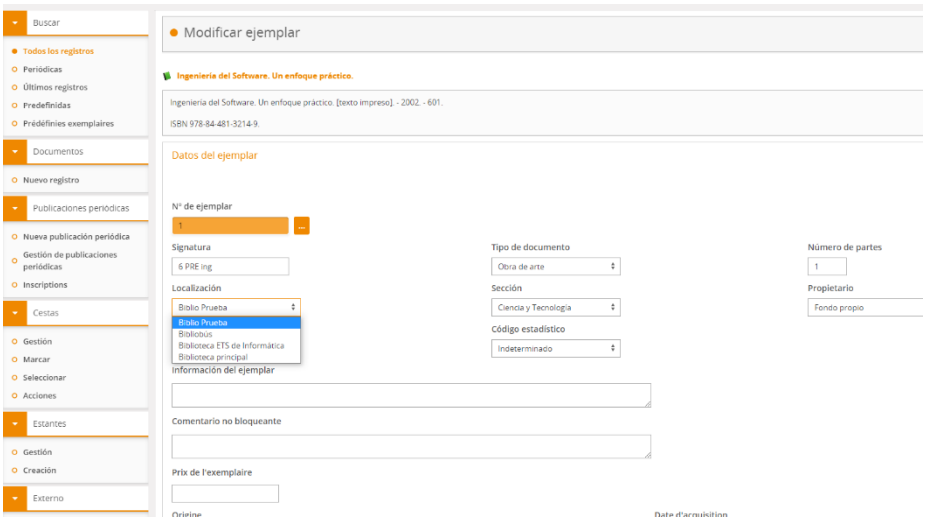


figura 74. Modificación de un ejemplar para variar su localización y sección dentro de PMB.

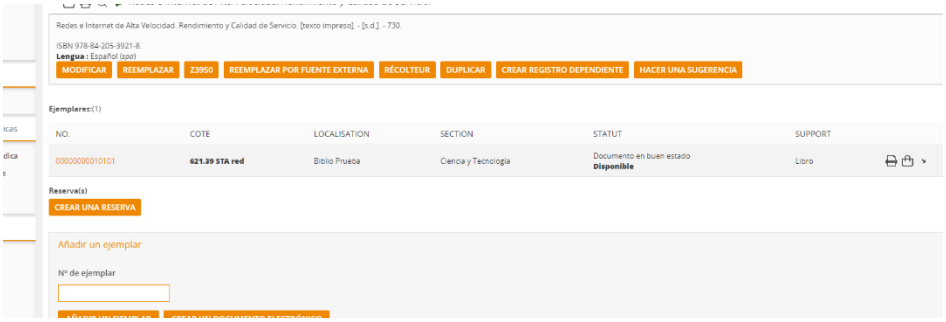


figura 75. Ejemplar disponible en librería de prueba. Consulta en PMB.

12.1.2.1.2.2.1 Organización de los ejemplares.

En función de la biblioteca, nos podemos encontrar con diferentes organizaciones de materiales. Por poner un ejemplo, según conversación telefónica mantenida con el personal técnico de la BNE, en dicha biblioteca los ejemplares los organizan por el tamaño físico de cada ejemplar (los tallan). Según nos comentaron, este tipo de organización está motivada para no tener problemas de espacio maximizando el uso del mismo. En el apartado correspondiente a nuestra librería de prueba realizamos una pequeña explicación de como hemos catalogado y organizado nuestros ejemplares de prueba. En PMB añadido a la posibilidad de localizaciones y secciones, podemos crear “estantes” donde ubicar nuestros ejemplares.

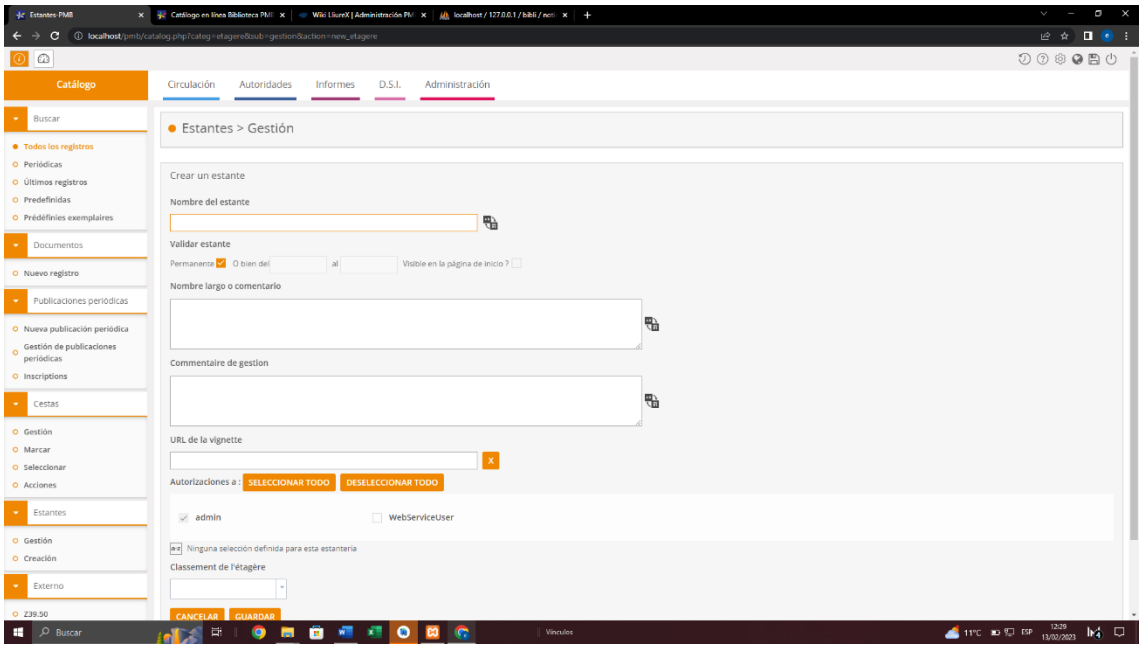


figura 76. Creación de un estante en PMB.

12.1.2.1.2.2.2 Préstamo de ejemplares.

El préstamo de ejemplares está asociado a los usuarios que desean solicitar su préstamo. Este está centralizado en la pestaña “Circulación” (figura 77).

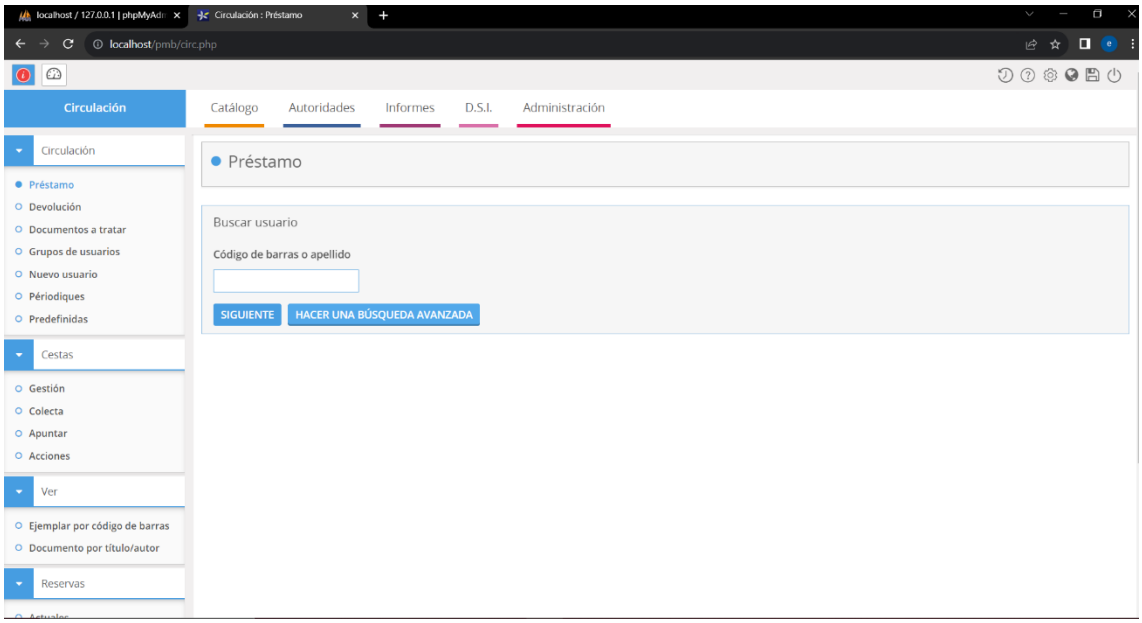


figura 77. Pestaña Circulación asociada al préstamo de ejemplares dentro de PMB.

Si buscamos el usuario “100008” creado previamente, accederemos a su ficha dónde se nos ofrece cierta información del mismo, así como ciertas acciones a realizar sobre dicho usuario (figura 78). Podemos observar que pertenece al grupo “LectoresValencia”, categoría adultos, etc. También que se nos ofrece realizar un préstamo.

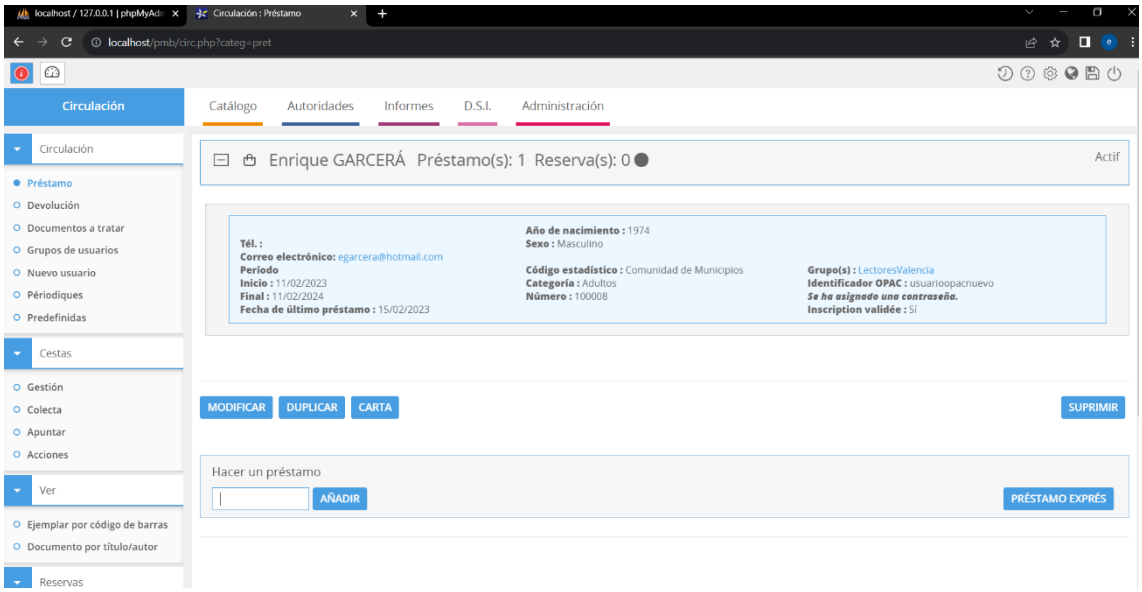


figura 78. Ficha de usuario dentro de PMB.

En el cuadro “Hacer un préstamo”, podemos indicar el código de barras del ejemplar que desea el usuario. Si pulsamos sobre el botón “Préstamo Exprés” se nos muestra un formulario para buscar ejemplares por una serie de campos como el del ISBN por ejemplo.

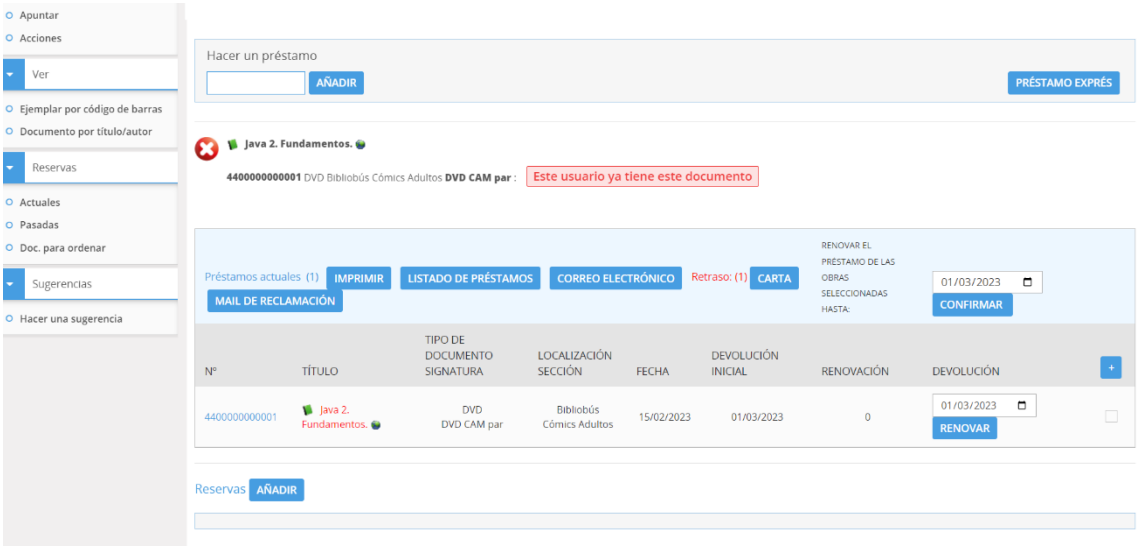


figura 79. Acción de préstamo de un ejemplar.

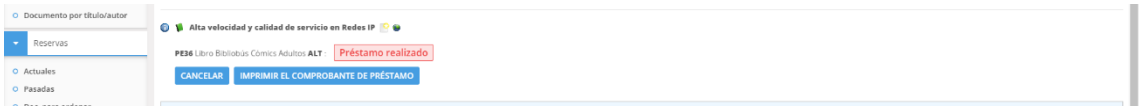


figura 80. Préstamo de ejemplar realizada satisfactoriamente.

En la figura 79, podemos observar como el sistema da error, dado que el ejemplar solicitado ya había sido prestado al usuario y nos ofrece su renovación. En la figura 80, el préstamo de un ejemplar de manera satisfactoria.

Las acciones modificar, duplicar, carta y suprimir se refieren a la administración del usuario con independencia de los ejemplares. Por ejemplo, al pulsar sobre suprimir, eliminaremos al usuario. Al pulsar sobre el botón

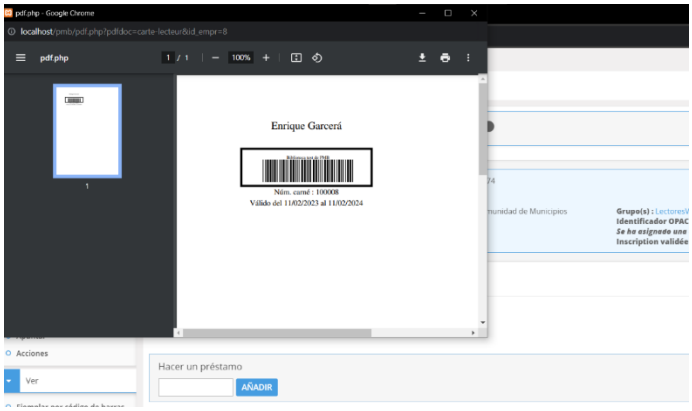


figura 81. Código de barras asociado a un usuario dentro de PMB.

“carta”, se nos imprimirá un fichero .pdf con el código de barras asociado al usuario. De este modo, con un lector de código de barras sería más sencilla la identificación del usuario a la hora de realizar préstamos o devoluciones.

El préstamo de un ejemplar estará condicionado a la definición realizada en su creación. Por ejemplo, si hemos creado ese ejemplar como de “solo

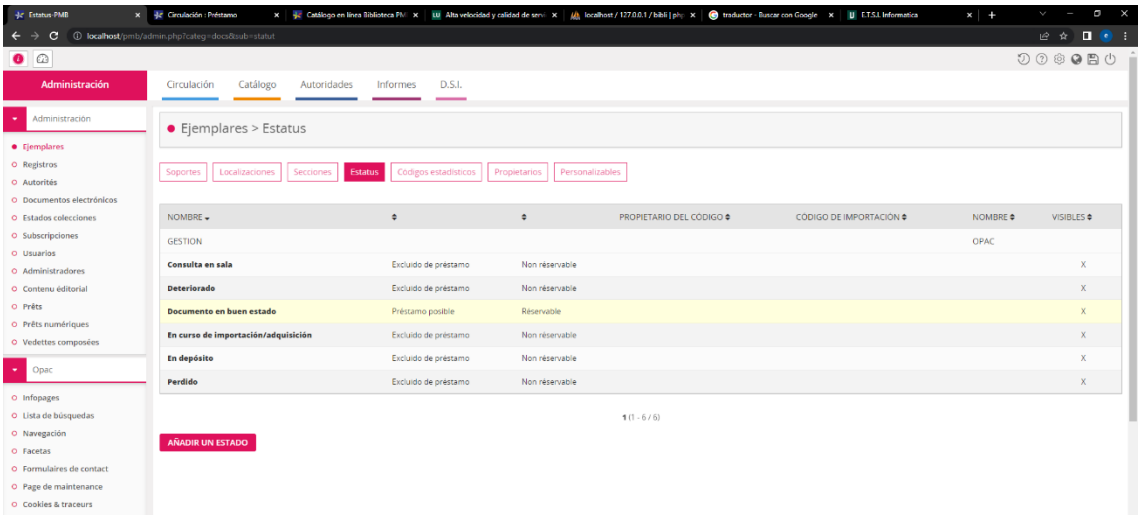


figura 82. Posibles estados en los que se puede encontrar un ejemplar dentro de PMB.

consulta en sala”, este no podrá ser prestado.

En la figura 82, podemos observar como cada posible estado de un ejemplar es asociado a que este sea prestable o no. También que podemos crear nuevos estados.

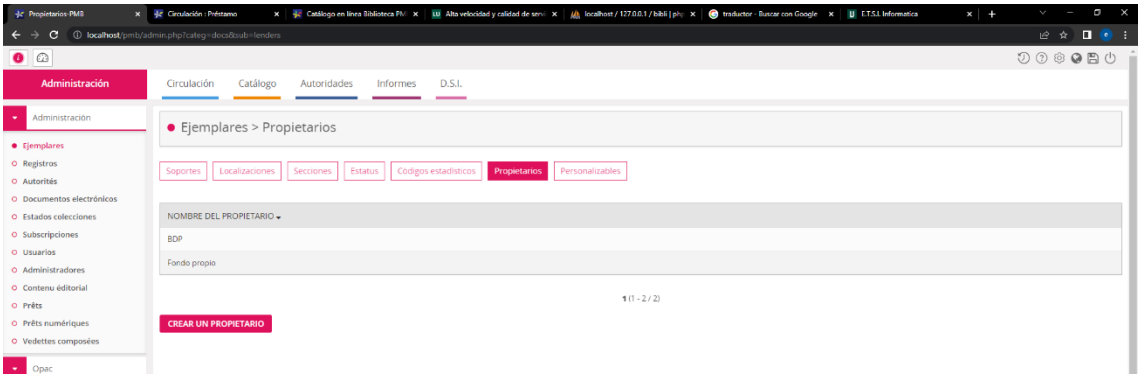


figura 83. Posibles propietarios de un ejemplar. PMB.

Del mismo modo, un ejemplar debe ser asociado a un propietario y en la figura 83 podemos observar los existentes y la posibilidad de crear nuevos.

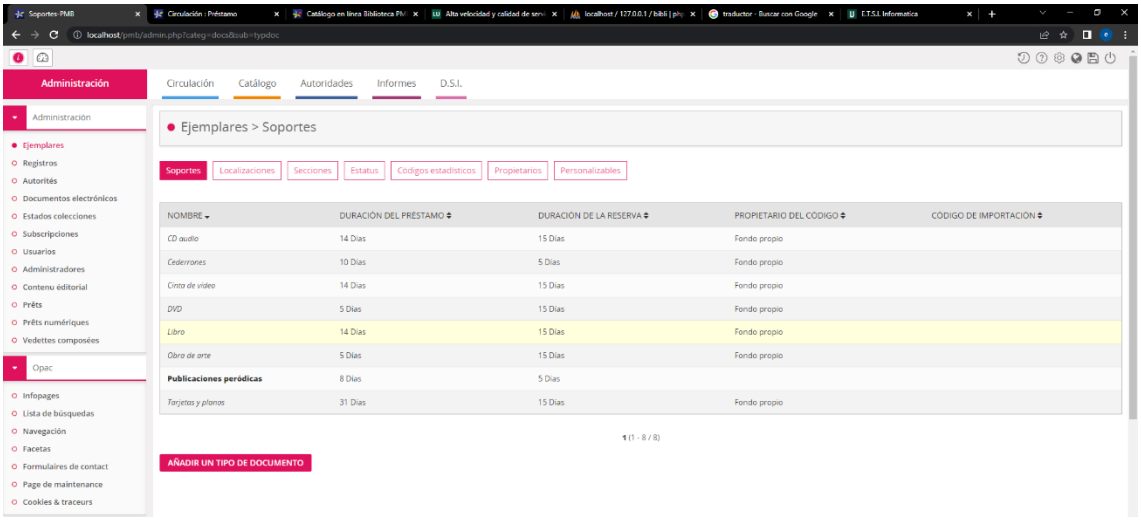


figura 84. Tipos de documentos a los que se puede asociar un ejemplar. PMB.

Un ejemplar debe ser asociado a un tipo de soporte y como se puede observar en la figura 84, podemos crear nuevos y un dato importante es que al tipo de soporte se le asocia una duración de préstamo y de reserva.

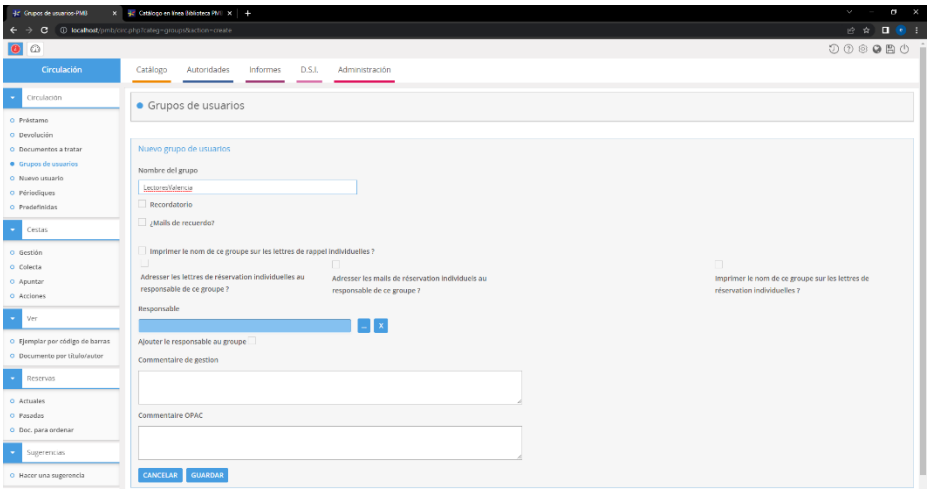


figura 85. Creación de un grupo de usuarios en PMB.

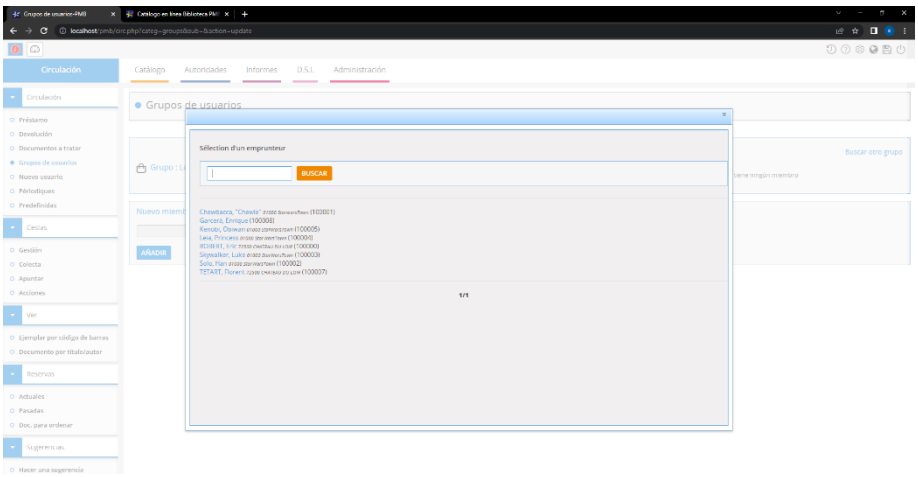


figura 87. Listado de usuarios lectores disponibles. PMB.

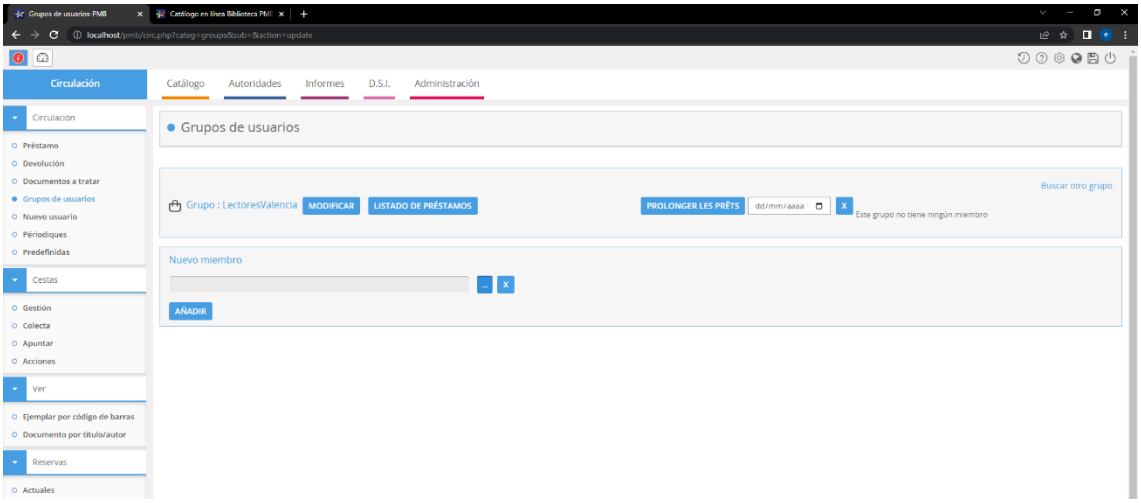


figura 86. Opciones de administración sobre un grupo de usuarios lectores. PMB.

12.1.2.1.2.2.3 Impresión de códigos de barra.

Se ha comentado previamente que cada ejemplar estará asociado a un código numérico correspondiente al código de barras. Dado que esto códigos están pensados y diseñados para poder usar un lector de códigos de barras y de esta manera simplificar la tarea de préstamo y devolución de ejemplares, PMB nos ofrece la posibilidad de imprimir etiquetas contenedoras de dichos códigos de barras. Para imprimirlos, hemos de acceder a la pestaña “informes”, opción de menú, “códigos de barra” e “impresión libre”. La primera vez que accedimos a dicha opción, obtuvimos el error de la figura 88.

figura 88. Error al intentar imprimir los códigos de barras.

Y después de investigar un poco, descubrimos que a partir de PHP 7.4, la referencia a los elementos de un array, en lugar de hacerse {i} se hace como [i]. Reemplazando las llaves por los corchetes en el fichero “generate.php” presente en el servidor, la impresión de los códigos de barra funcionó correctamente.

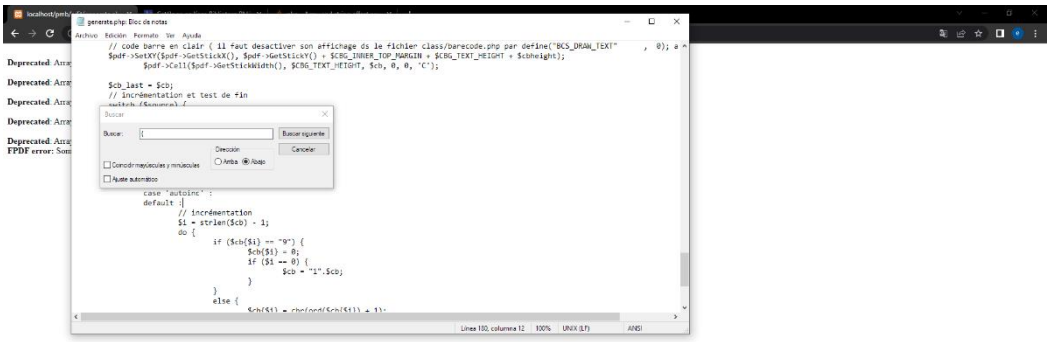


figura 90. Solución del error de impresión de códigos de barras.

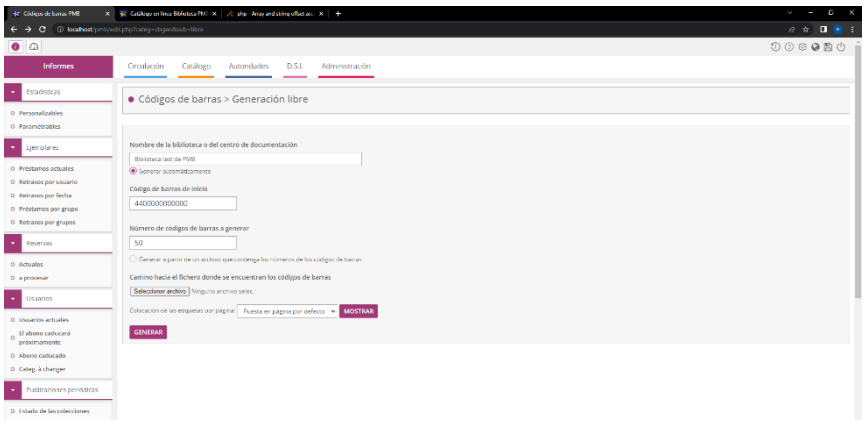


figura 89. Ejemplo de impresión de códigos de barras para identificar ejemplares.

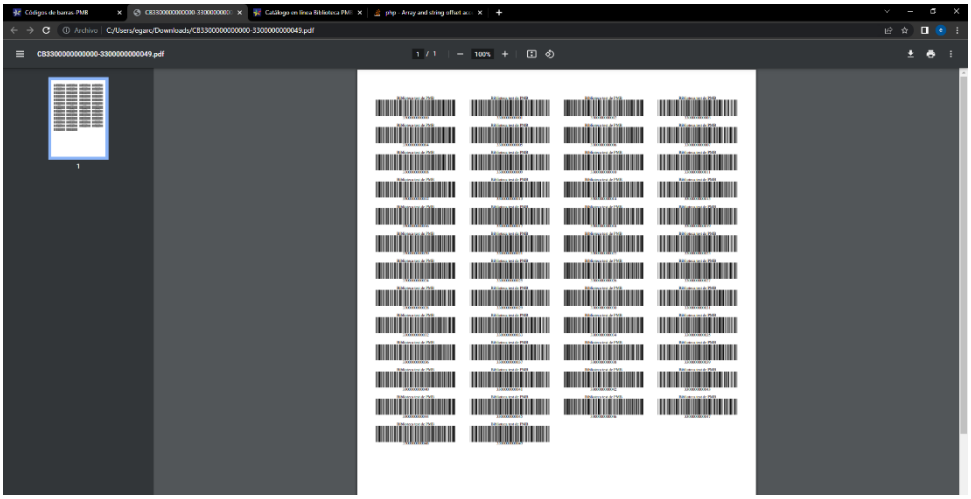


figura 91. Pantalla de generación de códigos de barras.

12.1.3 Android Studio.

Dada la evolución del IDE de desarrollo “Android Studio”, hemos optado por esta opción frente a otras como Eclipse, NetBeans o Visual Studio Code añadiéndoles el SDK de desarrollo. Para instalar el entorno podemos descargarlo de la página oficial (Google. Android Studio., s.f.). Su uso es completamente gratuito e incluye infinidad de utilidades que nos darán soporte en la tarea de diseño, codificación y pruebas de la aplicación.

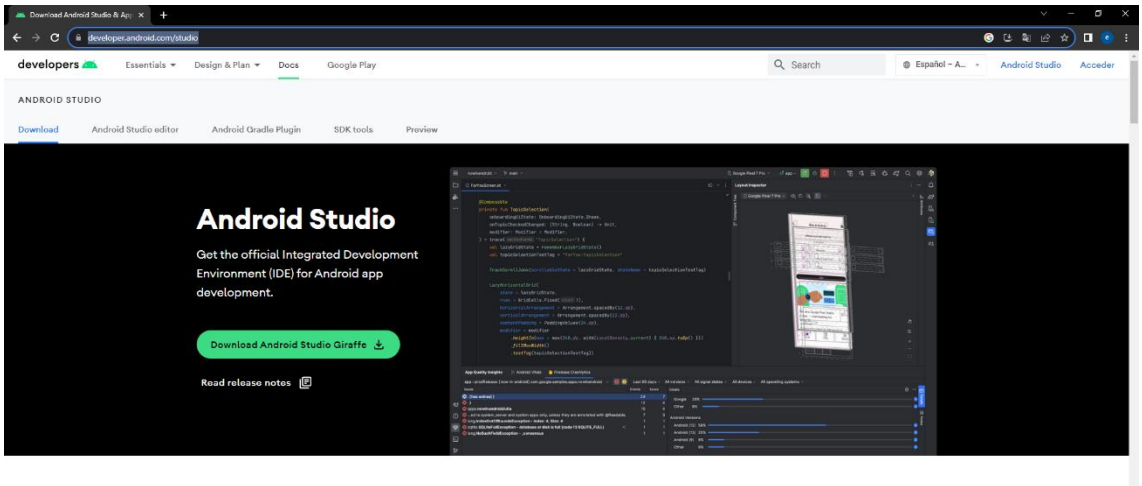


figura 92. Página principal de Android Studio.

Incluye opciones como la generación automática de los métodos get y set de una clase, múltiples emuladores de dispositivos, conexión a dispositivos reales para depuración mediante conexión USB o Wifi, autocompletado de código, generación y firma de los instaladores de la aplicación, integración de diferentes herramientas para interactuar con sistemas externos y un sinfín de elementos de los que una mera enumeración se escapa de las pretensiones

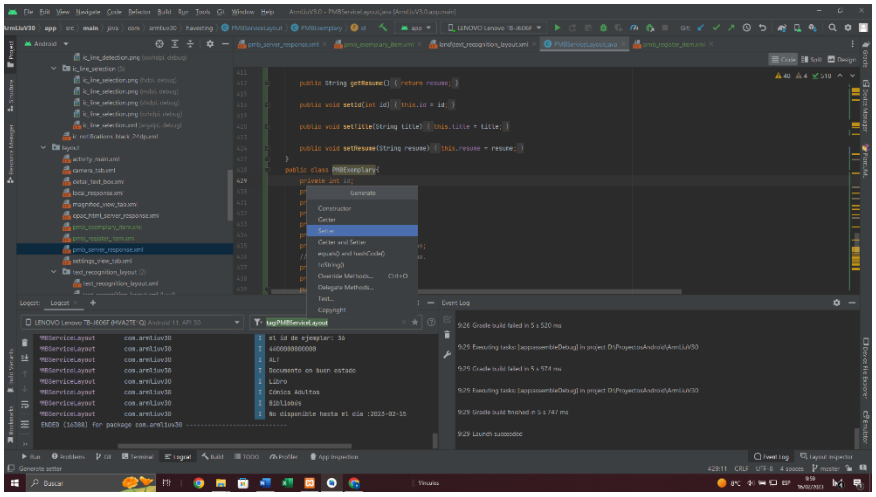


figura 93. Generación automática de los métodos get y set de una clase. Android Studio.

del presente trabajo. No obstante, en el apartado de implementación de la aplicación, de manera transversal se irán realizando algunas anotaciones sobre Android Studio.

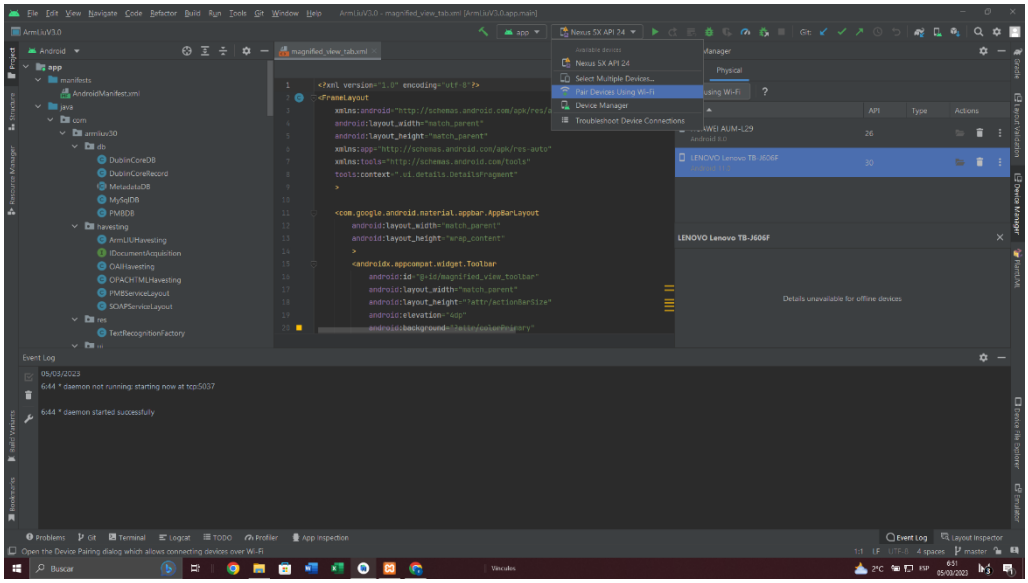


figura 94. Depuración mediante conexión inalámbrica de un dispositivo. Android Studio.

12.1.4 Git. Control de versiones y salvaguarda del código fuente.

Nos ha parecido de vital importancia el uso de una herramienta que nos permita realizar un control de versiones del código fuente de la aplicación. La generación de diferentes versiones del código fuente nos permitirá recuperar codificaciones anteriores para múltiples necesidades como podrían ser las de depuración del mismo, la generación de una versión concreta de la aplicación, etc. Hemos optado por emplear uno de los VCS (Sistemas de Control de Versiones) integrado en Android Studio. Frente a otras alternativas, hemos optado por emplear Git como control de versiones de código abierto (creado por Linus Tolvards en 2005), por sus múltiples ventajas y su completa integración con el entorno de desarrollo y con el servicio de hosting de repositorios GitHub en caso de que quisiéramos hacer público nuestro código fuente.

Para utilizar Git dentro de Android Studio se requieren una serie de pasos:

- Acceder al menú VCS de Android Studio y activar la integración del control de versiones y el tipo a emplear (figura 95 y figura 96).

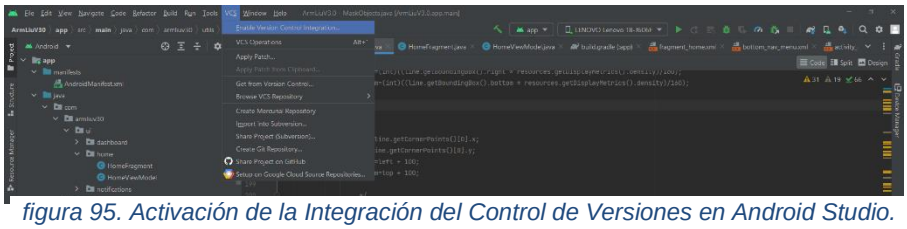


figura 95. Activación de la Integración del Control de Versiones en Android Studio.

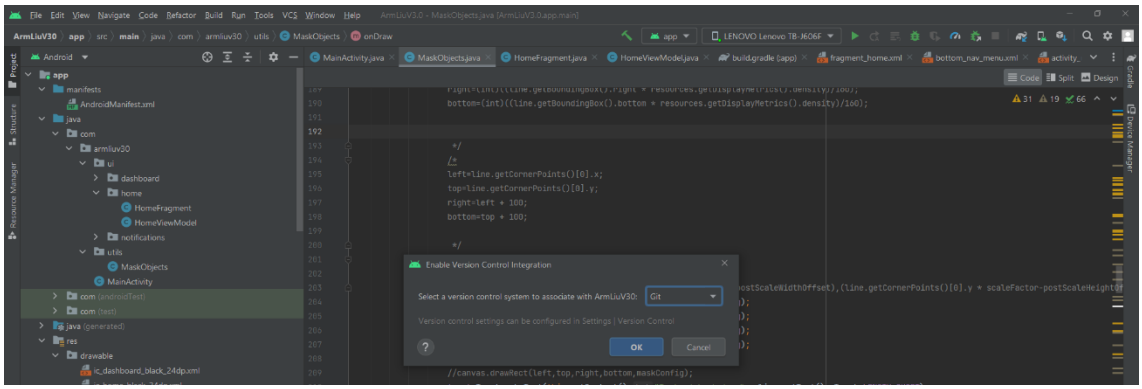


figura 96. Selección del tipo de control de versiones en Android Studio.

- Instalar Git dentro de Android Studio.

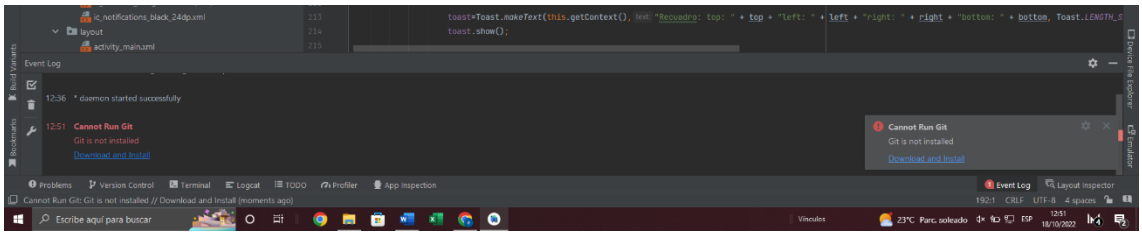


figura 97. Enlace a la instalación de Git dentro de Android Studio.

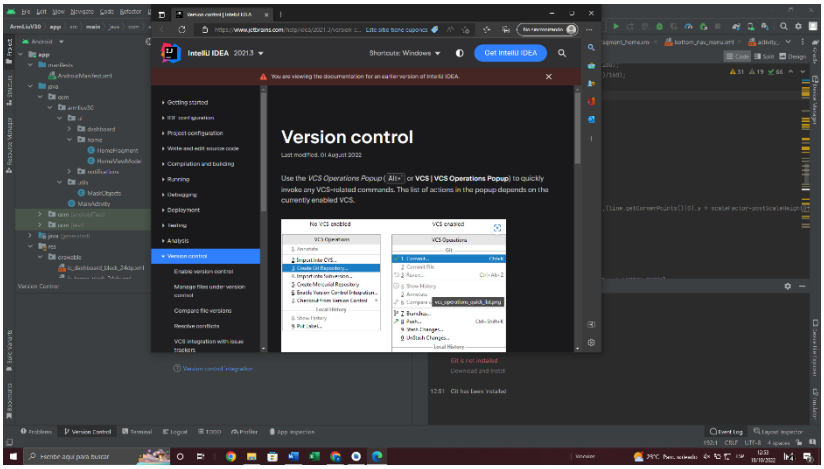


figura 98. Instalación de Git dentro de Android Studio.

- Inicializar el repositorio Git para el proyecto. En este paso es necesario prestar especial interés en situarse sobre la raíz del proyecto para que inicialice el repositorio con todos los ficheros presentes en el mismo. De no tener seleccionada esta al pulsar sobre la opción de inicialización del repositorio, nos creará uno vacío.

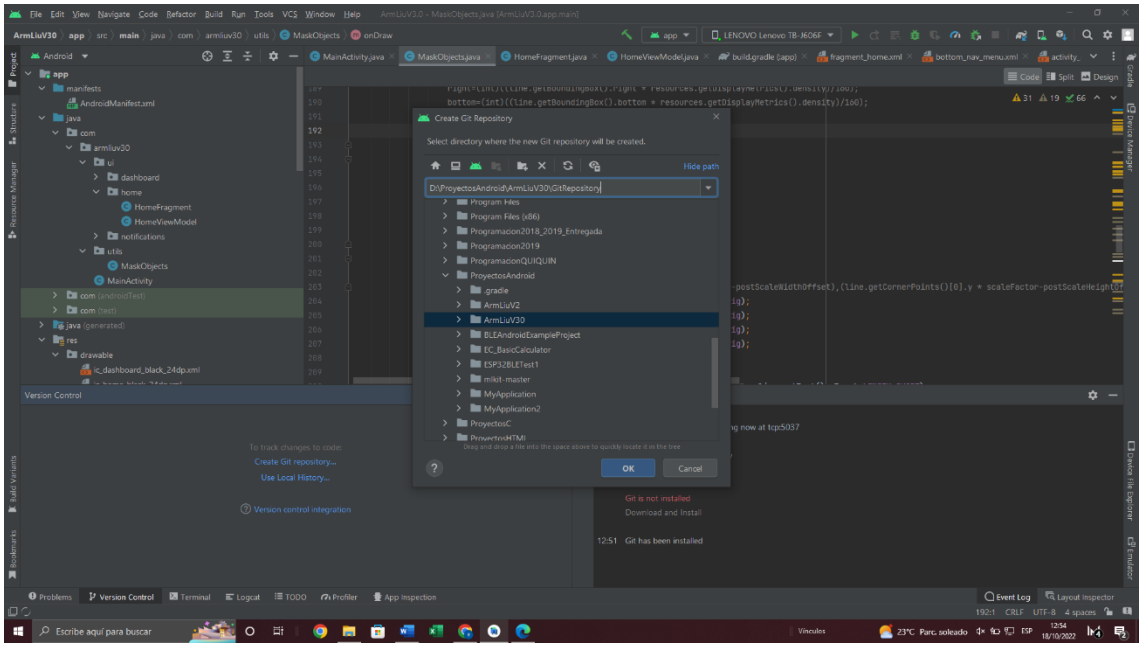


figura 99. Creación de un repositorio Git para nuestro proyecto.

- Etiquetar las diferentes versiones de la aplicación. Este paso es realmente importante, dado que, con independencia de que siempre podremos recuperar versiones anteriores de un fichero concreto de todas las veces que sobre el mismo hayamos ejecutado la acción “commit”, etiquetar una versión nos permitirá recuperar versiones completas de la aplicación. Por ejemplo, podríamos recuperar la versión 1.0 de la aplicación para generar el instalador de la aplicación para que se pueda ejecutar en dispositivos móviles cuya versión de Android es anterior. Debemos tener en cuenta que trabajamos sobre la versión etiquetada que tengamos cargada en ese momento, y por tanto los “commits” serán realizados sobre la misma.

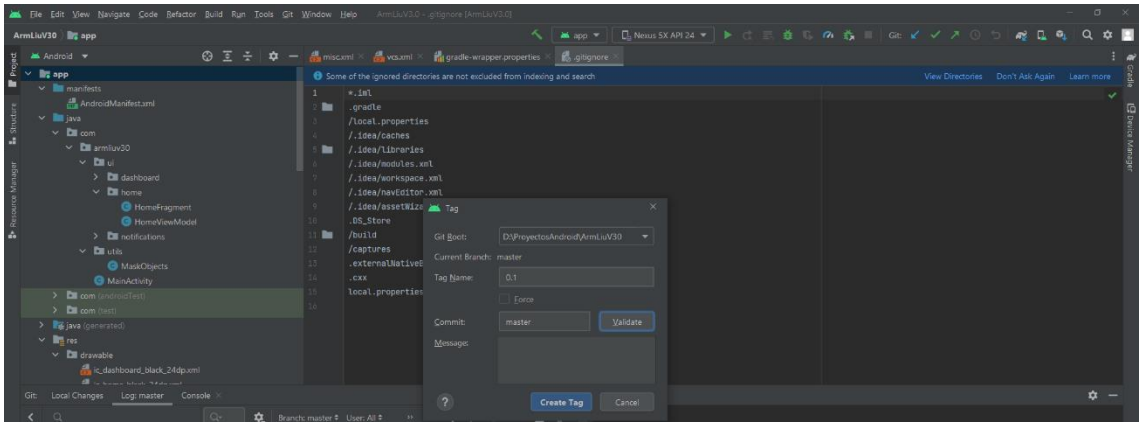


figura 100. Etiquetado de versiones en Git.

En la figura 101, podemos observar las diferentes opciones que nos ofrece Git. Para nuestro caso particular las opciones más útiles han sido: el

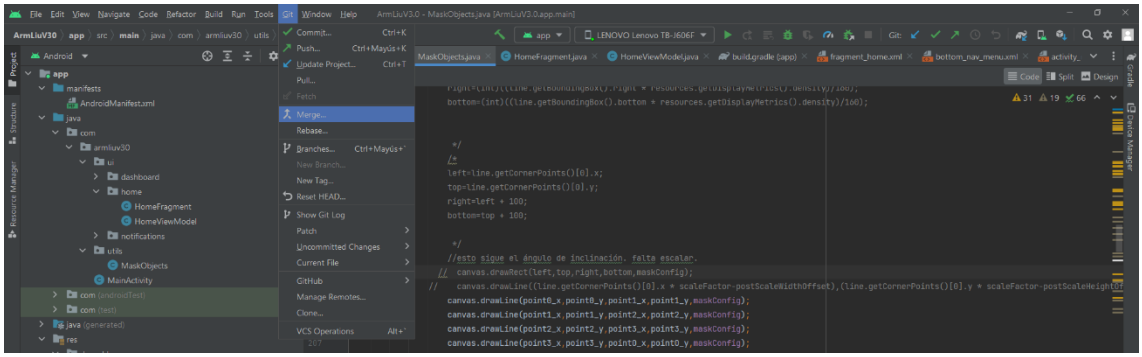


figura 101. Opciones disponibles en Git.

etiquetado de las diferentes versiones (new Tag), el registro de los cambios en determinados momentos sobre cada uno de los ficheros (commit), la opción que muestra las diferencias entre diferentes versiones de un fichero (show diferences) y la opción que muestra todos los "commits" realizados sobre un fichero (show history). Para acceder a estas opciones tan solo debemos mostrar el menú contextual sobre el fichero concreto y seleccionar el apartado Git dónde se mostrarán todas las opciones disponibles.

También nos ha parecido muy interesante la forma de representar mediante un color diferente, el nombre de aquellos ficheros que pertenecen al repositorio y los que no, así como aquellos ficheros que han sufrido cambios desde el último "commit" para advertirnos de esa situación.

En la figura 102, podemos observar cómo al realizar "commit" sobre un fichero, nos muestra los cambios que se han realizado sobre el mismo.

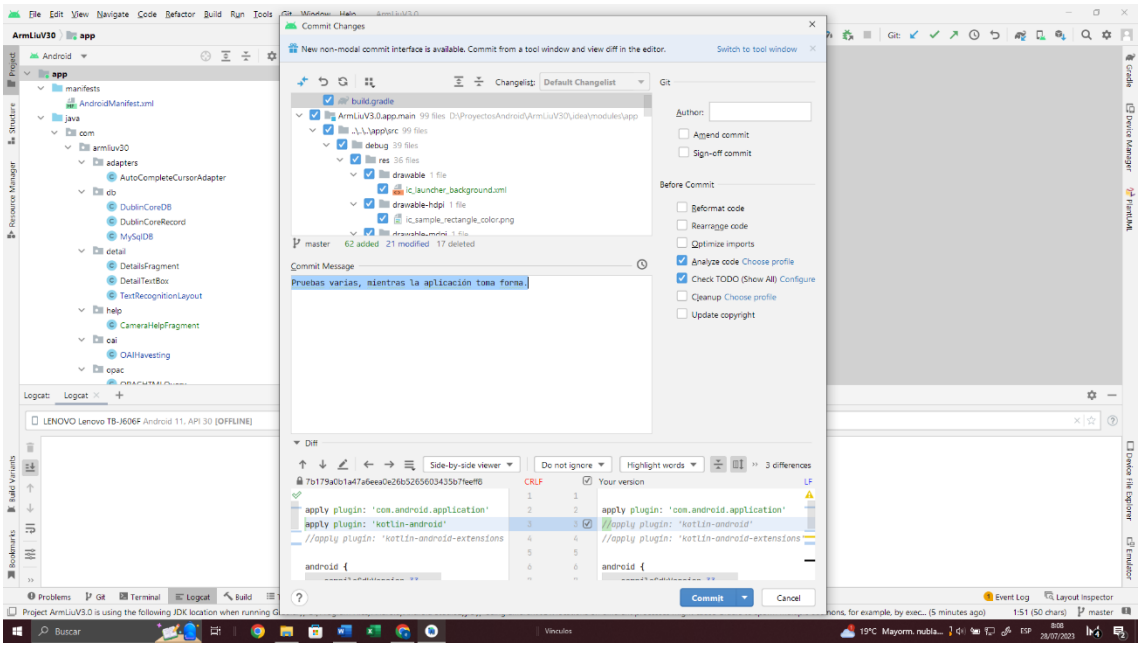


figura 102. Acción commit sobre un fichero.Tema claro.

12.1.5 Librería de prueba con diferentes ejemplares bibliográficos.

De forma genérica, para la catalogación de los diferentes ejemplares de la librería empleada para las pruebas, como **signatura** hemos empleado la codificación más común: CDU + Tres primeras del autor en mayúsculas + Tres primeras letras del título en minúsculas.

De forma particular, para las **revistas** hemos colocado la signatura en cada ejemplar de la colección en un tejuelo aunque el ejemplar sea fino y quede poco espacio para su colocación. Dado que la RAE define hemeroteca como la “biblioteca en la que principalmente se guardan y sirven al público diarios y otras publicaciones periódicas”, para las publicaciones periódicas hemos empleado la H de hemeroteca, seguido del número de fascículo y el año de publicación (H n.13 (2006), por ejemplo). Y para **CDs, DVD**, etc., hemos identificado el tipo de material + tres primeras letras del autor en mayúsculas + tres primeras letras del título en minúsculas (DVD CAM par, por ejemplo).

figura 103. Librería empleada para realizar las pruebas. Selección del sistema de clasificación y catalogación sobre un conjunto de ejemplares bibliográficos.

Hemos impreso unas etiquetas tanto para el tejuelo como para el código de barras de cada ejemplar, y después de su recorte se han pegado sobre los mismos.

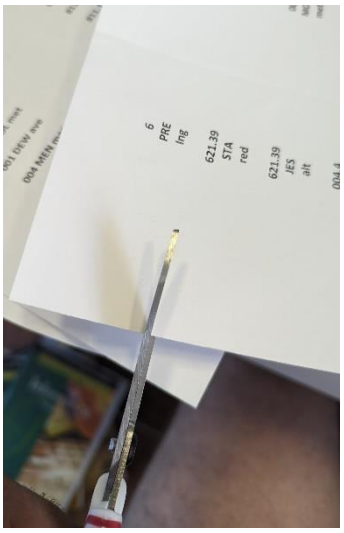


figura 104. Tejuelo de los ejemplares.

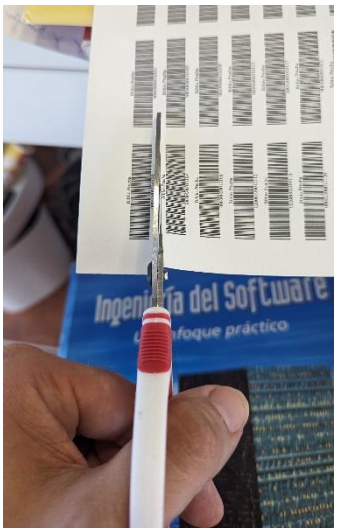


figura 105. Códigos de barra de los ejemplares.

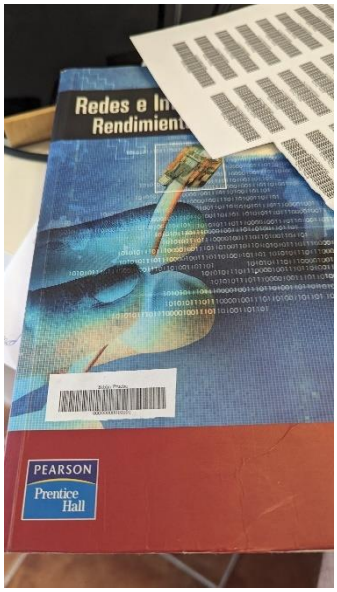


figura 106. Ejemplo ubicación del código de barras en un ejemplar.

12.1.6 Introducción de los datos asociados a la librería de prueba en PMB.

El listado de ejemplares catalogados en la librería de prueba y que se pueden observar en la figura 103, es el que sigue:

- Ingeniería del software. 6 PRE ing (C.B: 00000000010100)
- Redes e Internet de Alta Velocidad Rendimiento y Calidad de Servicio**.** 621.39 STA red (C.B: 00000000010101)
- Alta Velocidad y Calidad de Servicio en Redes IP. 621.39 JES alt (C.B: 00000000010103)
- Java 2. Fundamentos. 004.4 HOR jav (C.B: 00000000010104)
- Java 2. Características Avanzadas. HOR jav (C.B: 00000000010105)
- Estructura de Computadores. 004.3 PON est (C.B: 00000000010106)
- Bases de datos Relacionales. 004.6 CEL bas (C.B: 00000000010107)
- Diseño Lógico. 004.3 GIL dis (C.B: 00000000010108)
- Estadística. 311 ROM est (C.B: 00000000010109)
- Instalaciones Eléctricas en Media y Baja Tensión. 621.3 GAR ins (C.B: 00000000010110)
- Reglamento Electrotécnico para Baja Tensión. 621.3 MIN reb (C.B: 00000000010112)
- Libro Blanco de las Energías Renovables. 537 ESC lib (C.B 00000000010113)
- Metodología y Tecnología de la programación. 005 MOL met (C.B: 00000000010114).

- Aventuras Informáticas. Los mundos del ordenador. 001 DEW ave (C.B:00000000010102)
- Manual de Hack 2. 004 MEN man (C.B: 00000000010111)
- Sistemas y aplicaciones Informáticas. Temario. 004 LEY sis (C.B: 00000000010115).
- Informática. Volumen I. 004 GAR inf (C.B: 00000000010116).
- Informática. Volumen II. 004 GAR inf (C.B: 00000000010117).
- Informática. Volumen III. 004 SAM inf (C.B: 00000000010118).
- Informática. Volumen IV. 004 GAR inf (C.B: 00000000010119).
- Paco de Lucia. Retrato de Familia Con Guitarra. 78 TEL pac (C.B: 00000000010120).
- Mozart. 78 BRI moz (C.B: 00000000010121).
- La gran colección de música. Música del Romanticismo. Para Piano. 78 FLO rom (C.B: 00000000010122)
- La gran colección de música. Música del Barroco. Para Piano. 78 FLO bar (C.B: 00000000010123)
- La gran colección de música. Piezas breves. Para Piano. Volumen 1. 78 FLO pie (C.B: 00000000010124)
- La gran colección de música. Piezas breves. Para Piano. Volumen 2. 78 FLO pie (C.B: 00000000010125)
- La gran colección de música. Wolfgang Amadeus Mozart. Para Piano. . 78 FLO moz (00000000010126).
- La gran colección de música. Ludwing van Beethoven. Para Piano. 78 FLO bee (00000000010127)
- WolfGang Amadeus Mozart. Complete Works. CD BRI moz (C.B: 00000000010128)
- Bach Edition. 160 CD Box. CD BRI bac (C.B: 00000000010129).
- Ludwig Van Beethoven. Complete Works. CD BRI bee (C.B: 00000000010130)
- Collins. Diccionario Inglés. 811.111 GRI col (C.B: 00000000010131)
- English Grammar. 811.111 HEI eng (C.B: 00000000010132)
- PTE. Pearson Test Of English General. 811.111 BAX pte
- Voyage. Course Book. 811.111 ROB voy
- Voyage. Work Book. 811.111 ALD voy

Denotar que, pese a que en el listado anterior solo se indica una fracción del título, el texto que aparecerá en el tejuelo y el código de barras que aparecerá en PMB, en el servidor de PMB se han introducido aquellos datos más relevantes para realizar las pruebas, como son, el ISBN, los autores/as, etc., siguiendo las directrices dadas en apartados anteriores y completando en ocasiones estos datos mediante los servidores z3950 expuestos. En las siguientes figuras podemos observar algunos ejemplos de la consulta de estos ejemplares sobre el servidor OPAC local.

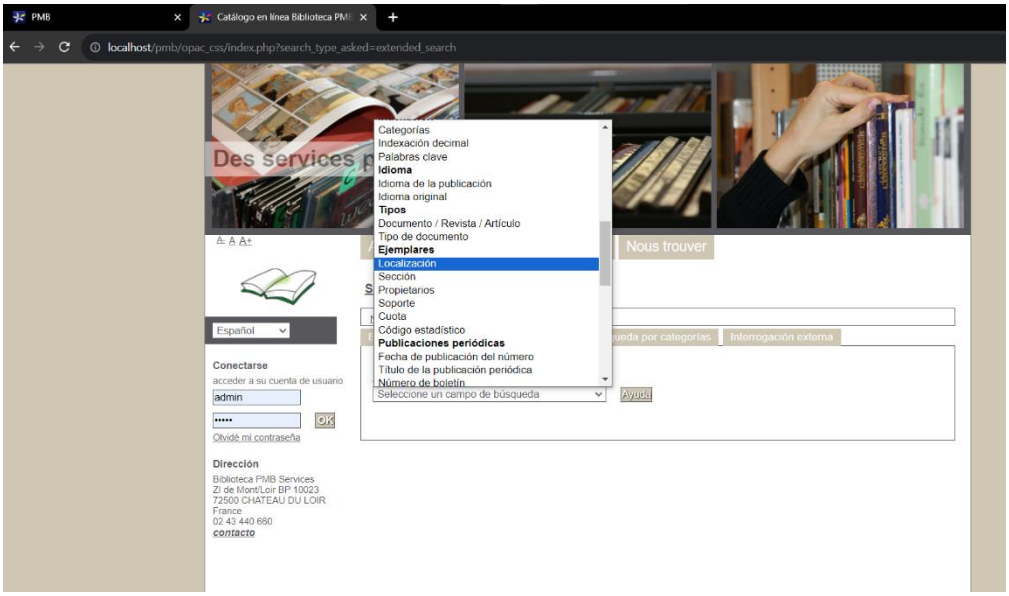


figura 109. Búsqueda por localización de ejemplares. OPAC local.

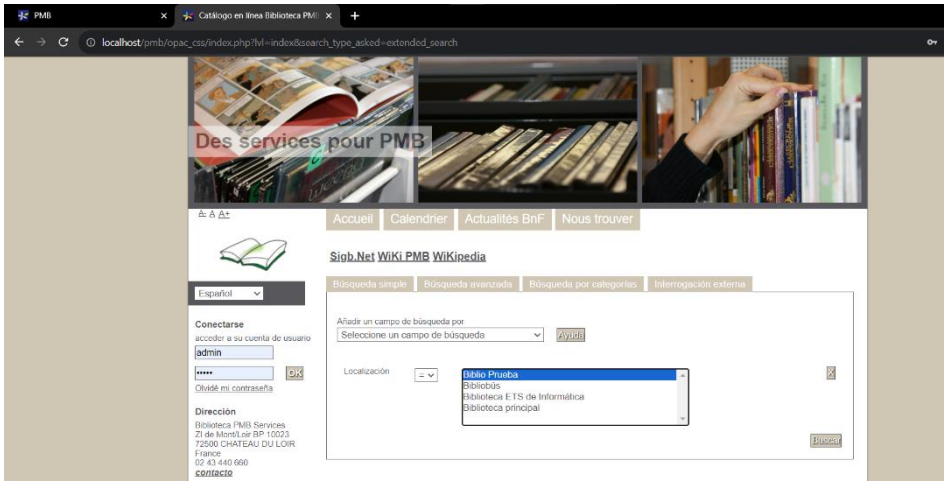


figura 108. Selección de la localización en la búsqueda de ejemplares. OPAC Local.

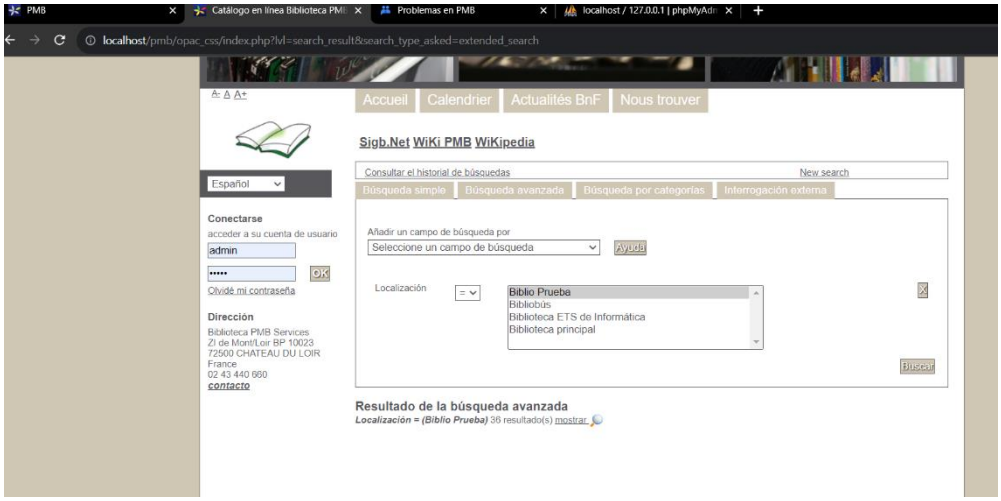


figura 107. Resultado de los ejemplares disponibles en la biblioteca de prueba. OPAC Local.

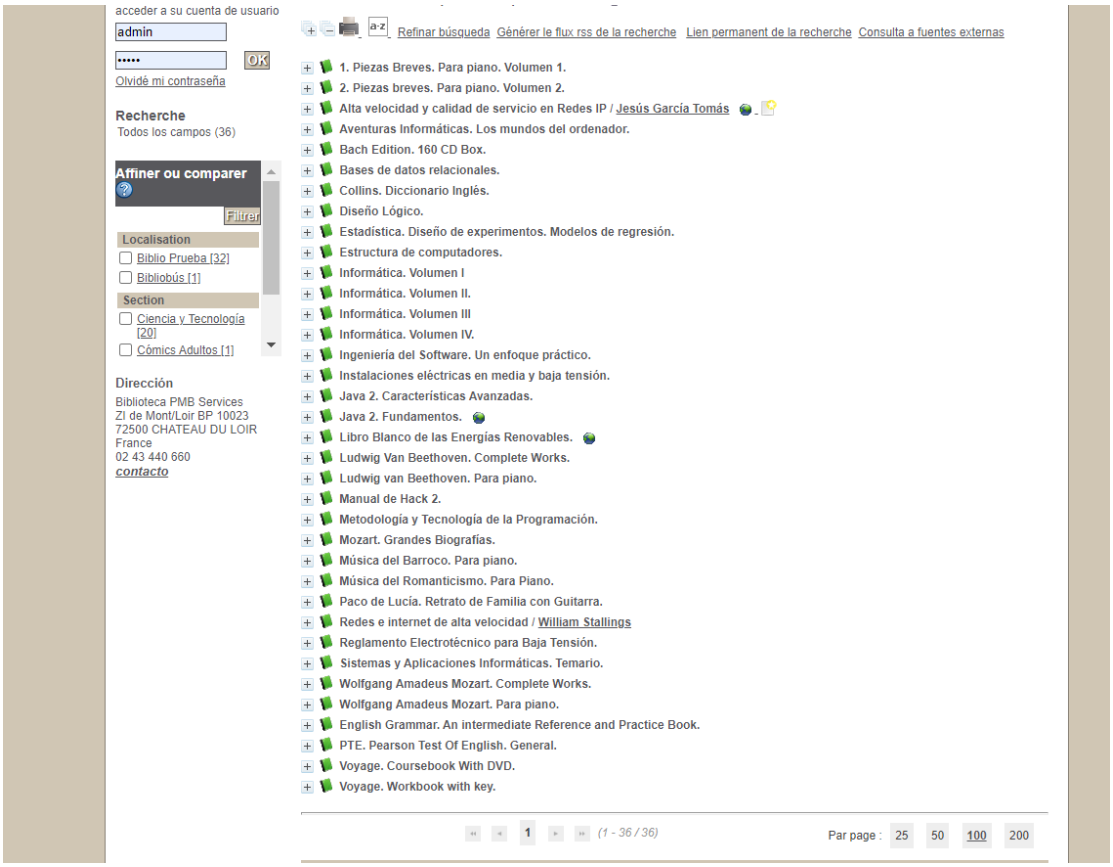


figura 111. Listado de ejemplares presentes en la librería de prueba. OPAC Local.

figura 110. Detalle de consulta de un ejemplar en el servidor OPAC Local.

figura 114. Detalle de consulta de un ejemplar en el servidor OPAC Local.

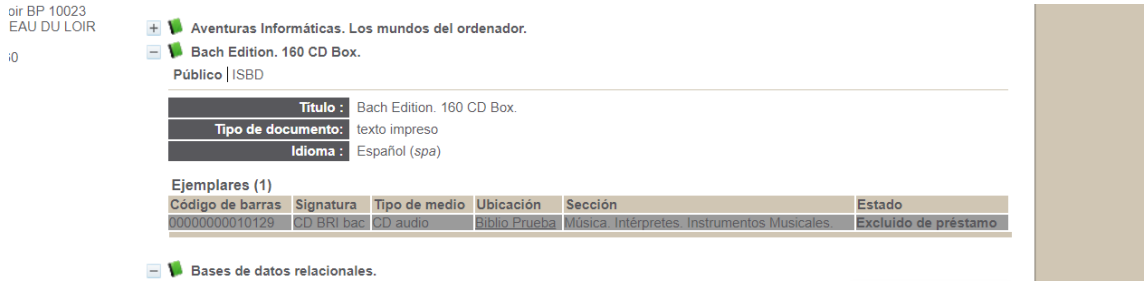


figura 113. Detalle de consulta de un ejemplar en el servidor OPAC Local.

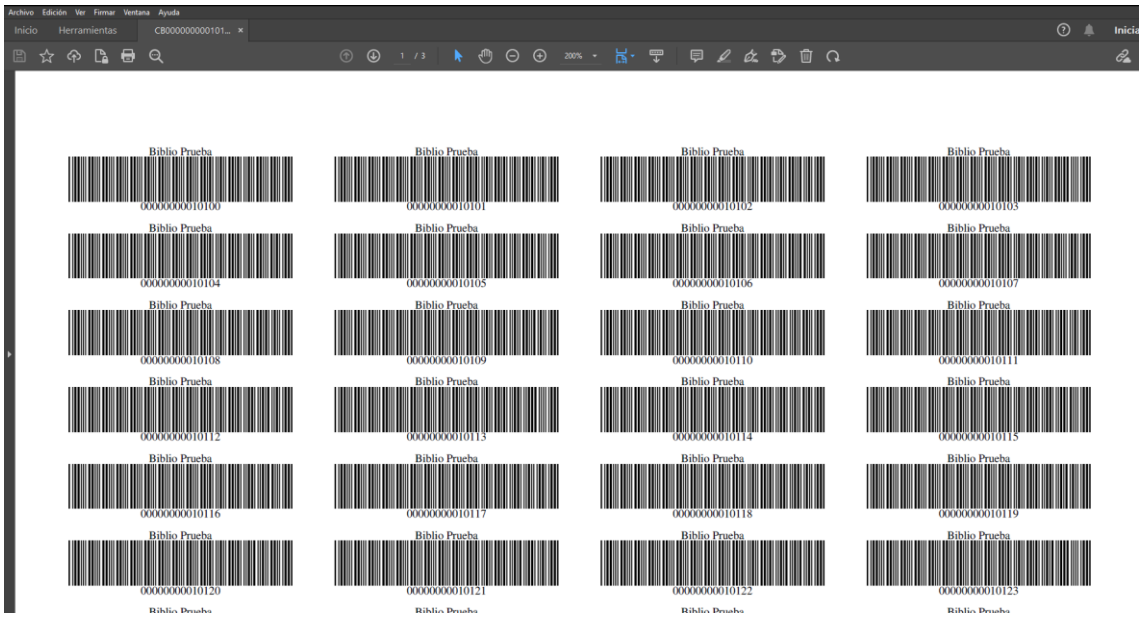


figura 112. Muestra de la impresión de los códigos de barra empleados en la biblioteca de prueba.

12.1.7 Emuladores del S.O Android.

Android Studio nos ofrece la posibilidad de utilizar una serie de dispositivos virtuales o reales. Si bien la Tablet Android que describiremos en el siguiente apartado es un dispositivo real sobre la que hemos realizado diferentes pruebas, en el presente aparatado se explica brevemente la forma de emular un dispositivo para realizar diferentes pruebas aportando algunas anotaciones sobre su creación y configuración.

En primer lugar, debemos crear un dispositivo virtual con la opción “create device” de Android Studio.

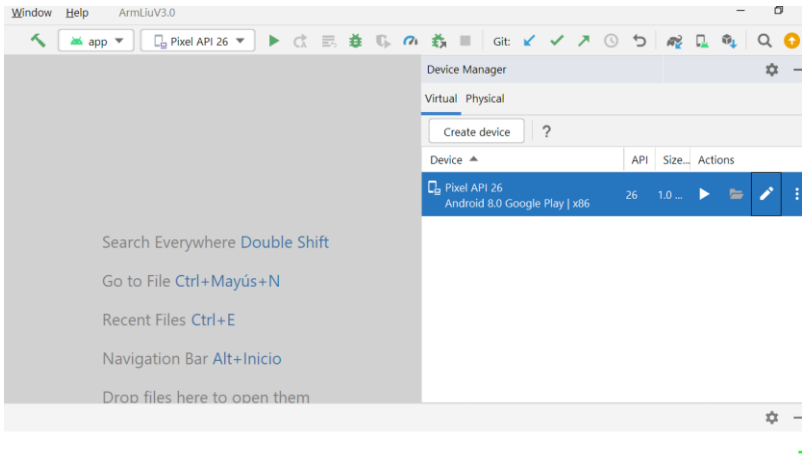


figura 115. Creación de un dispositivo virtual de Android.

Se nos ofrece la creación de un dispositivo de un tipo concreto (por ejemplo, de la familia “Pixel”) o uno genérico de una resolución y tamaño

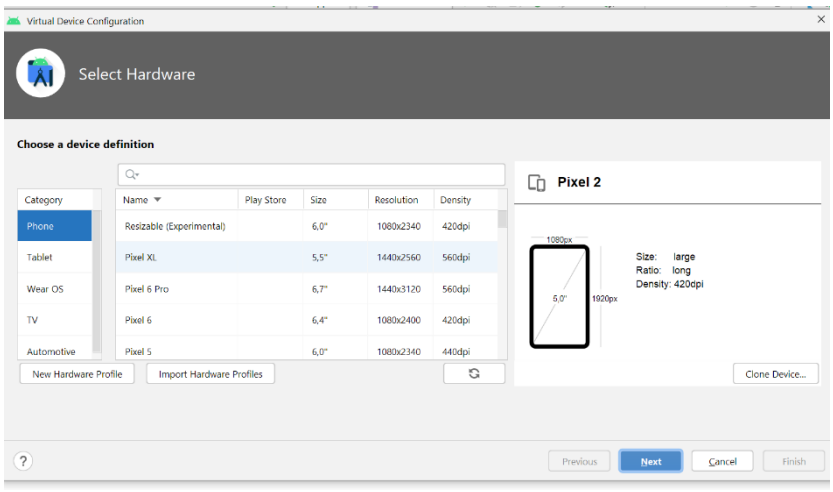


figura 116. Elección del tipo de dispositivo virtual Android.

determinado. Escogemos nuestra elección y al pasar a la pantalla siguiente se nos ofrece la posibilidad de elegir un nivel de API que es equivalente a la versión de S.O. que deseamos emplear. En nuestro caso hemos elegido la versión Oreo que se corresponde con Android 8.0, dado que, si observamos los datos de la figura 118, el porcentaje actual de esta versión es del 3,14% y añadido a que conforme hemos ido desarrollando la aplicación, elementos del SDK como “MLKit” nos han ido forzando a incrementar la versión mínima de Android que podíamos utilizar por motivos de compatibilidad con dicho kit de aprendizaje automático, no tiene sentido intentar mantener una retrocompatibilidad con versiones de Android Anteriores.

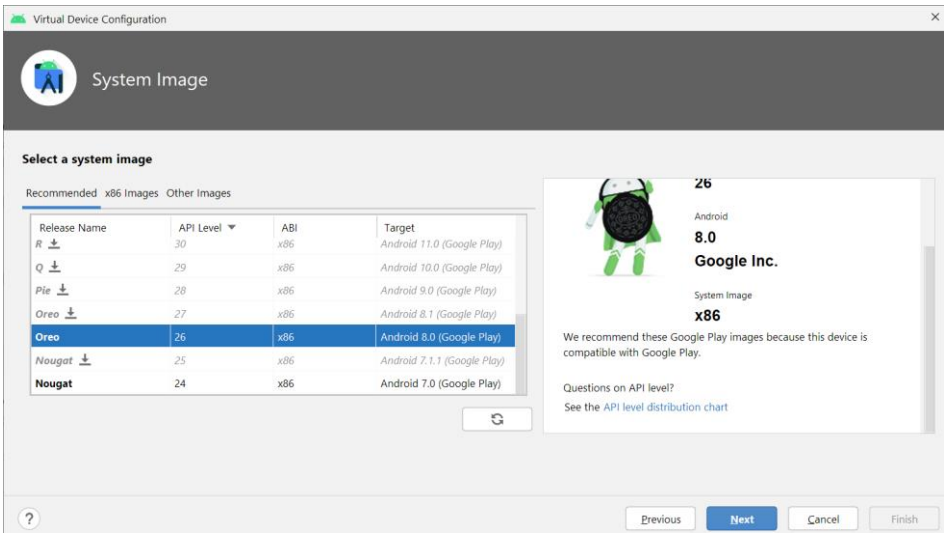


figura 117. Elección de la versión de Android para su instalación en dispositivo virtual.

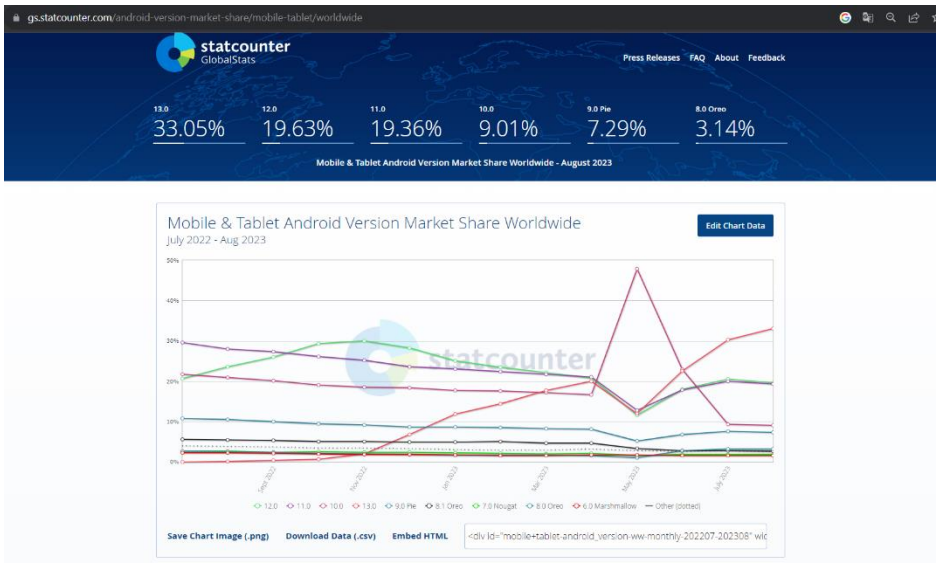


figura 118. Porcentaje de uso de determinadas versión de Android. Obtenido de StatCounter.

Además, al pulsar sobre la opción “See the API level distribution Chart”, se nos proporciona los cambios más relevantes de cada versión, pudiendo observarlos en la figura 119, donde aspectos como la nueva forma de asignación de permisos de forma dinámica en tiempo de ejecución los iconos adaptativos y el “Picture-in-Picture” nos han parecido relevantes para partir de esta versión.

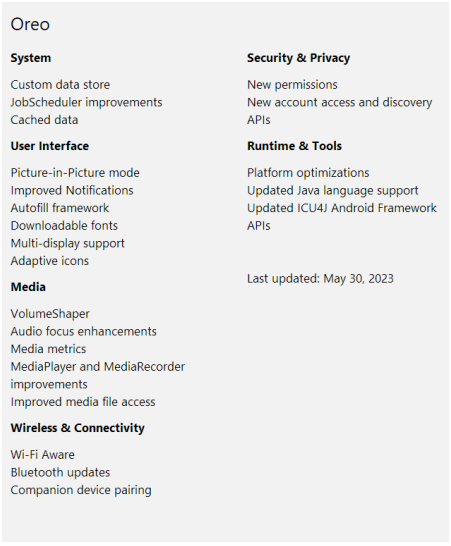


figura 119. Cambios de Oreo 8.0 con respecto a versiones anteriores.

Otro aspecto importante es el del espacio que ocupa en disco el dispositivo virtual que vamos a emplear. Podemos observar en la figura 117, que sobre la versión Oreo 8.0 no aparece una flecha de descarga al lado del texto. Esto es debido a que descargamos dicha versión de API previamente. En caso de que deseásemos emplear cualquier otra versión al mismo tiempo deberíamos de descargarla previamente, y contando que cuando la descargamos esta ocupará alrededor de 1GB y al ejecutarla comienza a incrementarse su tamaño por ejemplo a 10GB, este aspecto es importante si no disponemos de mucho espacio en disco duro como fue nuestro caso.

Una vez creado nuestro dispositivo virtual nos aparecerá en Android Studio una línea similar a la de la figura 115, “Pixel API 26” dependiendo del dispositivo y versión de Android elegido (este texto lo podríamos haber cambiado en la creación por uno de nuestro interés). El uso de este dispositivo virtual es simple, tenemos el típico botón de “Play” para ejecutarlo y el de “Stop” para pararlo, junto a una serie de opciones que aparecerán al pulsar sobre los típicos tres puntos verticales que poco a poco ha ido implantando Google.

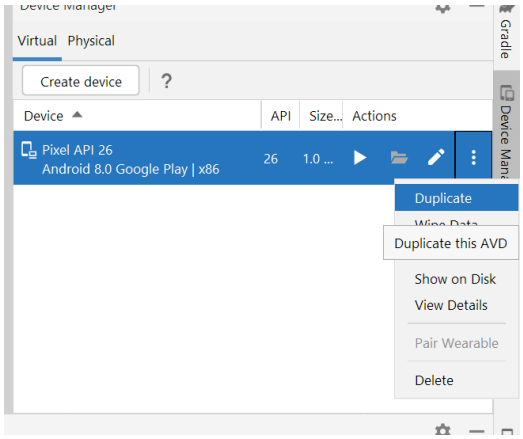


figura 120.Uso de dispositivo virtual.

En nuestro caso particular, antes de ejecutar el dispositivo virtual es necesario realizar una pequeña configuración de la cámara, dado que la nuestra es una aplicación de realidad aumentada y de lo contrario no podríamos probarla. Si pulsamos sobre el lápiz presente en la figura 120, accederemos a la configuración del dispositivo. Cada vez que cambiemos algo de dicha configuración deberemos pararlo y volverlo a arrancar para que surtan efecto. Si al editar pulsamos sobre opciones avanzadas, entre las múltiples

opciones de configuración encontraremos las de la cámara. Por defecto Android Studio configura una imagen estática y si deseamos usar la cámara real de la computadora dónde estamos ejecutando Android Studio, deberemos seleccionar la opción WebCam.

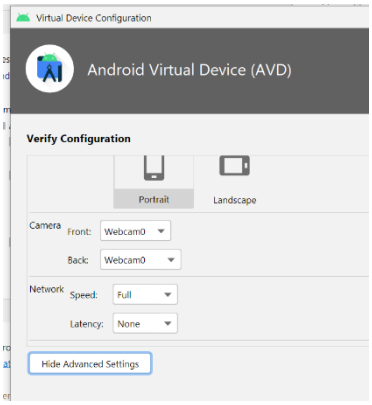


figura 121. Configuración de la cámara del dispositivo virtual.

Ahora sí, ejecutamos el dispositivo y podemos instalar la aplicación de la forma habitual pulsando sobre el botón “play” de la barra superior de Android Studio.

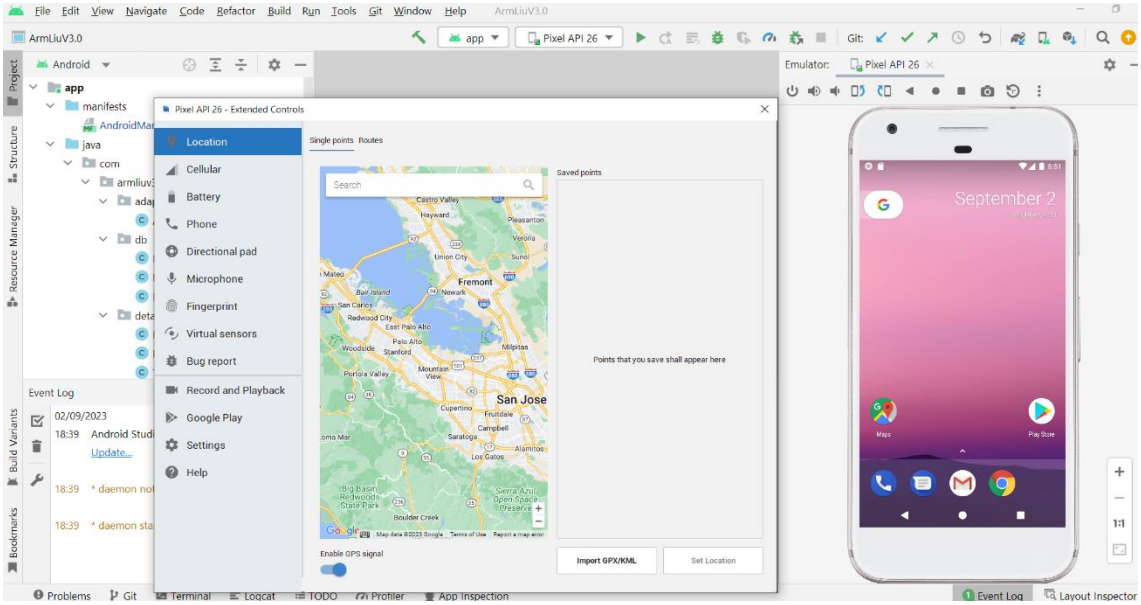


figura 122. Ejemplo de ejecución de dispositivo virtual.

Denotar que en la figura 122, hemos mostrado las opciones extendidas que se muestran pulsando sobre los tres puntos verticales que aparecen

encima del dispositivo virtual. También que para que nuestra aplicación funcione deberemos ir a la configuración de esta en el dispositivo virtual para otorgarle permisos de uso de la cámara, micrófono y archivos.

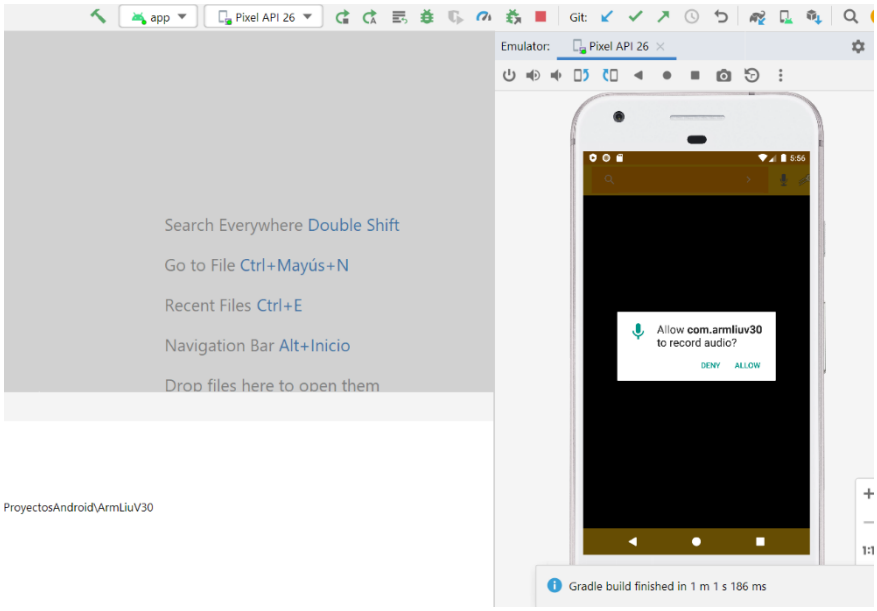


figura 123. Asignación de permisos en tiempo de ejecución.

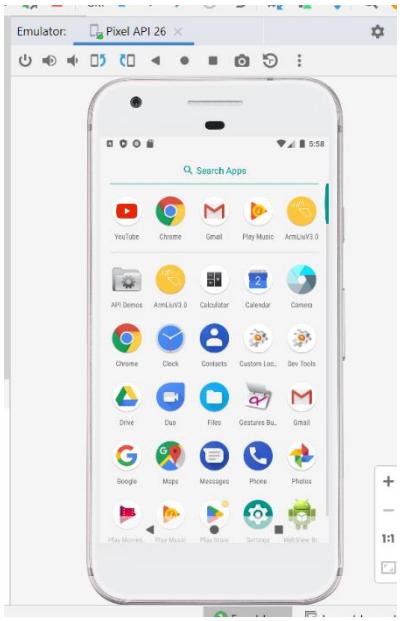


figura 124. Launcher de la aplicación con icono personalizado.

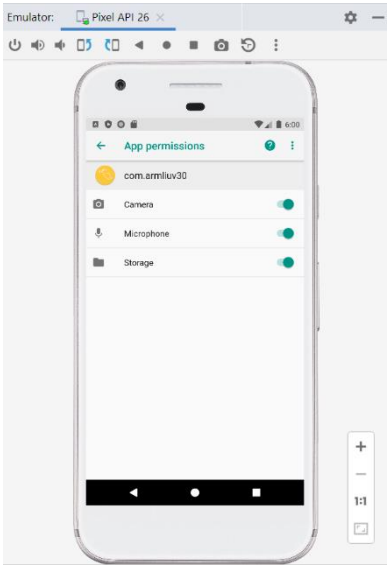


figura 125. Asignación de permisos en la configuración de la aplicación.

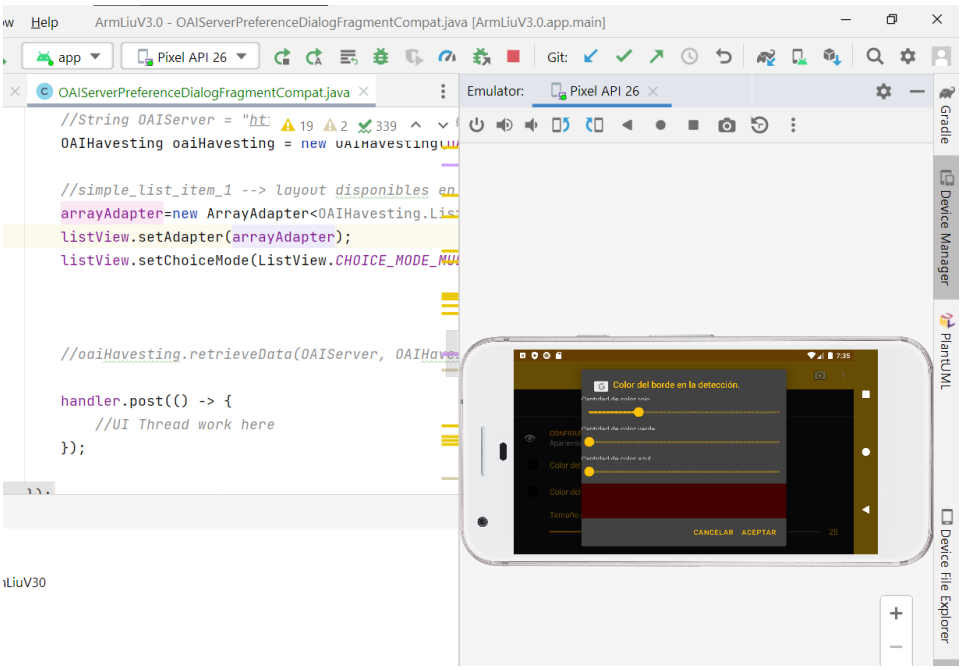


figura 126. Ejemplo de ejecución de la aplicación sobre dispositivo virtual. Configuración del color de la máscara de detección en la vista previa.

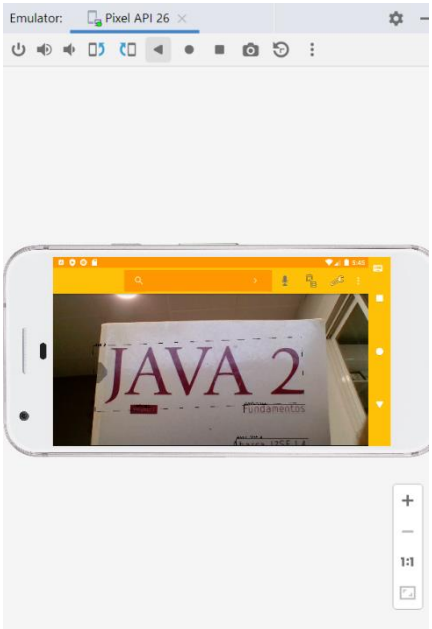


figura 127. Ejemplo de ejecución en dispositivo virtual. Pantalla de detección de ejemplares.

12.1.8 Tablet Lenovo.

Para poder emplear un dispositivo real para probar y depurar la aplicación a desarrollar, en primer lugar, es imprescindible realizar la conexión del mismo mediante un cable que conecte dicho dispositivo con el puesto sobre el que estemos desarrollando la aplicación. También existe la posibilidad de conectarlo de forma inalámbrica, sin embargo, esta opción nos dio algún problema y la descartamos. En segundo lugar, hemos de activar las opciones para desarrolladores accediendo a la configuración de la tableta, sección “sistema”, entrando a la opción “información de la Tablet” y pulsando repetidamente sobre el texto dónde nos indica el “número de compilación” hasta que aparezca el mensaje “Ahora están activadas las opciones para desarrolladores”. Una vez realizadas las acciones anteriores su uso es muy similar al de un dispositivo virtual. Para ello abrimos el manejador de dispositivos, bien mediante el menú herramientas, bien mediante el icono

destinado a tal fin (móvil con una porción del Androide en color verde). En esta ocasión tenemos que seleccionar la pestaña “Physical” (figura 128)

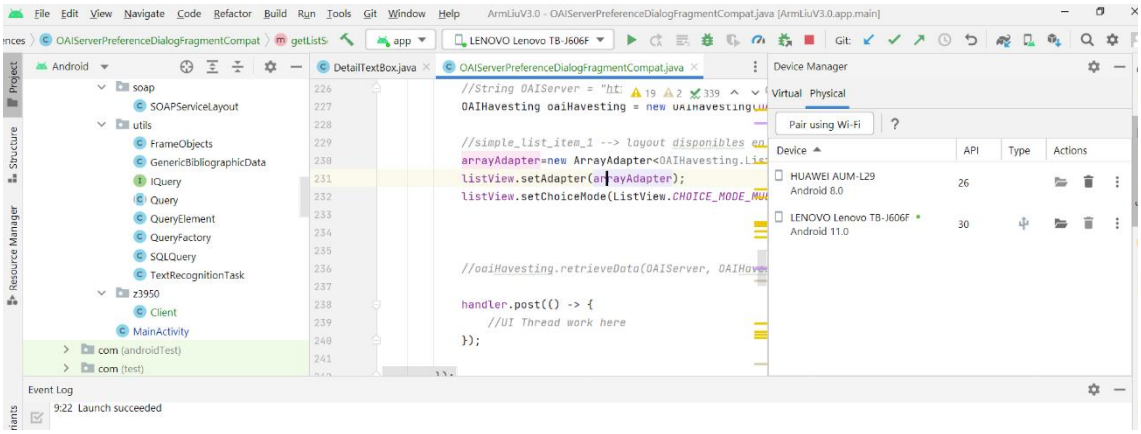


figura 128. Manejador de dispositivos de Android Studio. “Physical”.

Para instalar y ejecutar la aplicación lo realizaremos de la forma habitual, seleccionando el dispositivo sobre el que deseamos realizar dicha acción y pulsando “play” (figura 129).

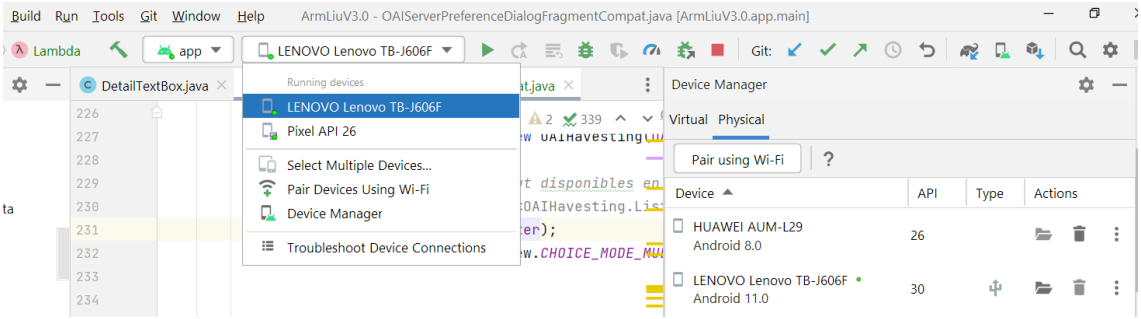


figura 129. Ejecución de aplicación sobre un dispositivo real.

En la figura 130, se muestra un ejemplo de ejecución de la aplicación sobre la tableta. En ella podemos observar como se está enfocando un ejemplar bibliográfico y dado que se está realizando una detección de textos en tiempo real, algunos de los textos detectados permanecen estables y bien enmarcados (Java, 2, fundamentos), mientras otros están en proceso de detección (Volumen I no ha sido bien detectado y se muestra Valum en su lugar) por el movimiento de la tableta que es sustentada por la mano. Al fondo de la imagen vemos la pantalla del portátil que está ejecutando Android Studio.

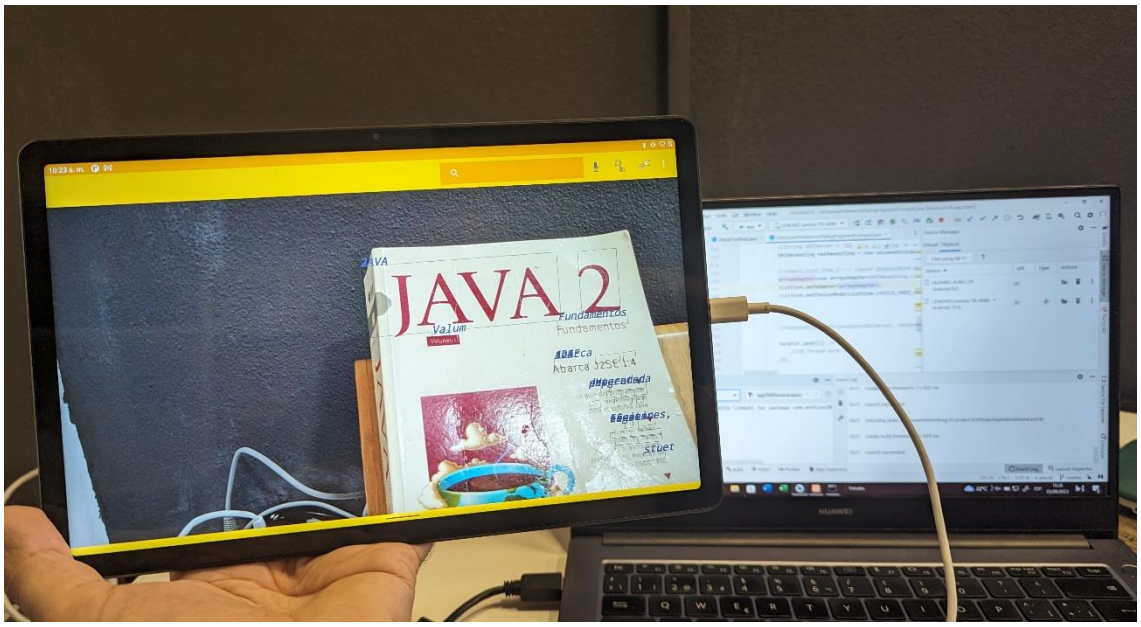


figura 130. Ejemplo de ejecución de la aplicación sobre dispositivo real.

12.2 Plataforma de puesta en producción.

Con vistas a poder realizar la puesta en producción de la aplicación, se ha creado una cuenta de desarrollador de Google y se ha empleado “Play Console” para poder poner esta a disposición de una serie de usuarios de nuestra elección para que puedan realizar las pruebas correspondientes como paso previo a su puesta en producción definitiva en el “Play Store” de Google, que está prevista para un futuro como opción de mejora.

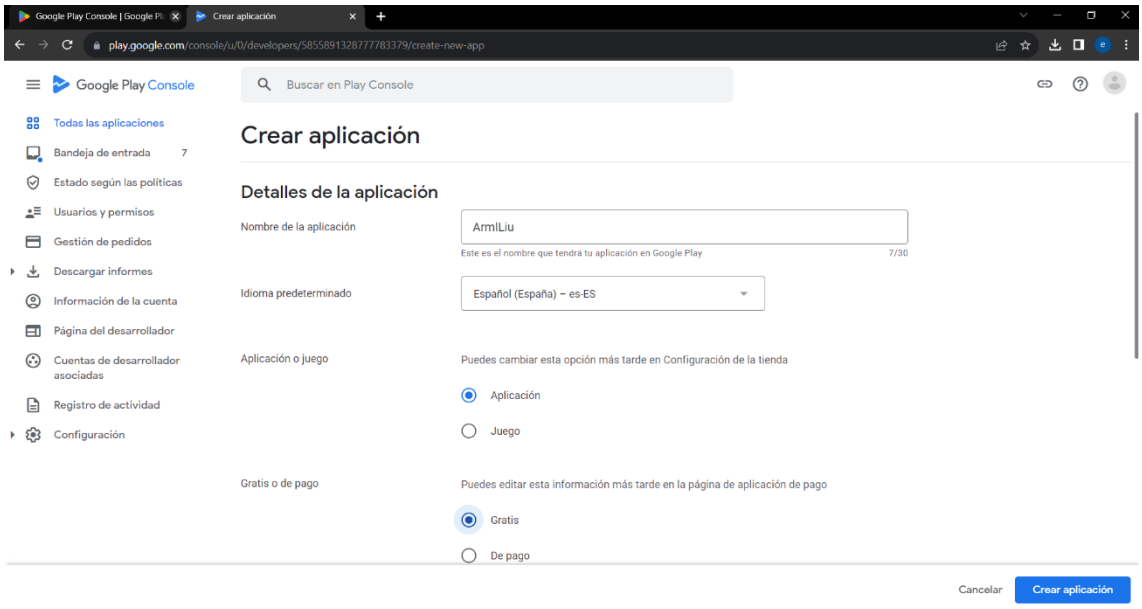


figura 131. Creación de la aplicación dentro del "Play Console" de Google.

13 Descripción de la aplicación desarrollada.

13.1 Especificación de requisitos. Fase de análisis.

En cuanto a la especificación de requisitos, hemos optado por complementar el lenguaje textual con el lenguaje UML, dado que, es un lenguaje de modelado ampliamente empleado y consideramos facilitará la comprensión de la documentación generada con él.

13.1.1 Requisitos funcionales.

13.1.1.1 Reglas de negocio.

- La aplicación deberá ejecutarse sobre un dispositivo móvil con sistema operativo Android 8.0 o superior.
- La aplicación empleará la cámara del dispositivo móvil para detectar el texto presente en la imagen que se esté enfocando en ese momento y a través de la consulta sobre un SIGB externo, obtendrá información que será mostrada al usuario sobre el texto real que se esté detectando (RA).
- La aplicación deberá permitir el filtraje de los textos a detectar, mediante la introducción de un patrón de filtraje.

- La aplicación deberá tener un apartado de configuración de aquellos elementos más relevantes presentes en la misma.
- La aplicación deberá soportar varios idiomas.
- Se podrá interactuar con determinadas funcionalidades de la aplicación empleando la voz.

13.1.1.2 WireFraming de la aplicación

Entendiendo el "WireFraming" de la aplicación, como un boceto que representa la interfaz de esta y por tanto la estructura de cada una de las pantallas presentes en la misma de una forma más o menos detallada, hemos creído conveniente incluirlo en la presente memoria. Pese a que existen una serie de herramientas que podemos destinar a tal fin, como podrían ser: Wireframe.cc, MockFlow, Axure, o la propia vista de diseño dentro del IDE Android Studio, inicialmente se realizaron una serie de dibujos a mano alzada que sirvieron para definir las diferentes opciones a desarrollar. Sin embargo, dada su baja calidad y que la memoria ha sido completada una vez implementada la aplicación, hemos considerado que una serie de "pantallazos" de la aplicación clarificarían en mayor medida los diferentes aspectos tenidos en cuenta en los bocetos iniciales y por tanto se han empleado estos. También hemos decidido realizar una descripción más detallada de estas pantallas con vistas a suplir el manual de usuario y no incrementar más si cabe la extensión de la presente memoria.

13.1.1.2.1 Apertura de la aplicación.

En la figura 132, podemos observar el icono de apertura de la aplicación. Se ha diseñado con el objetivo de que siga la estética de la misma.

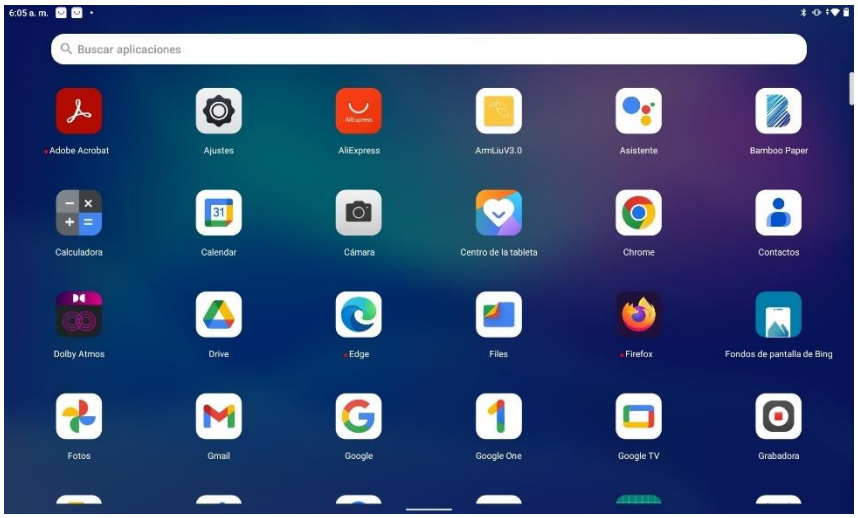


figura 132. WireFraming de la aplicación. Icono de apertura.

13.1.1.2.2 Pantalla de vista previa. Detección de textos en tiempo real.

En la figura 133, podemos observar la pantalla de vista previa sobre la que en la parte superior nos podemos encontrar la barra de herramientas. En la barra de herramientas tenemos disponibles principalmente las siguientes opciones por orden de aparición:

- Cuadro de filtraje: al introducir un texto en este cuadro, tan solo se enmarcarán y se mostrará la información flotante de aquellos detectados que contengan el patrón introducido mediante dicho texto.
- Interacción por voz: podemos observar como en la parte derecha del cuadro aparece un icono en forma de micrófono. Al pulsar sobre él, iniciaremos la interacción por voz y se irá escribiendo dentro del cuadro el texto que vayamos pronunciando.
- Acceso a la vista de detalle: pulsando sobre este icono, accederemos a la vista de detalle que será descrita en el apartado siguiente, con vistas a obtener información de determinados servidores.
- Acceso a la pantalla de preferencias de la aplicación que será descrita en otro apartado.
- Otras opciones como ayuda, términos de uso, acerca de, etc.

figura 133. WireFraming de la aplicación. Vista previa.

Sobre la figura 133, podemos observar como se han enmarcado una serie de ejemplares indicando el texto detectado como información flotante en la parte superior de dicho marco. Dado que se trata de una vista previa y para hacer captura de pantalla requeríamos de pulsar de dos botones del dispositivo, hemos captado la detección un tanto inestable debido al movimiento de la cámara y esto se refleja en que algunos textos no han sido detectados correctamente y en que algunos marcos están un tanto desplazados.

figura 134. WireFraming de la aplicación. Reconocimiento por Voz.

Denotar que, en la presente pantalla existen varias opciones configurables mediante la pantalla de preferencias. Por un lado, el color del marco y del texto es configurable (el texto se puede poner además en negrita y en cursiva, así como configurar su tamaño). También podemos configurar que aparezca solo el marco de cada elemento, pero no el texto y que la detección sea de un párrafo, de una línea o palabra a palabra. En la figura 136, se puede observar un ejemplo de la opción de autocompletado.

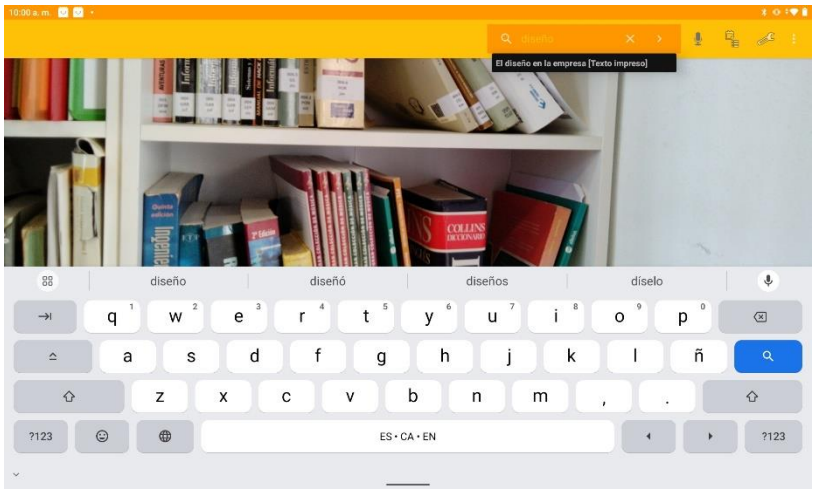


figura 135. WireFraming de la aplicación. Vista previa posición horizontal. Opción de autocompletado.

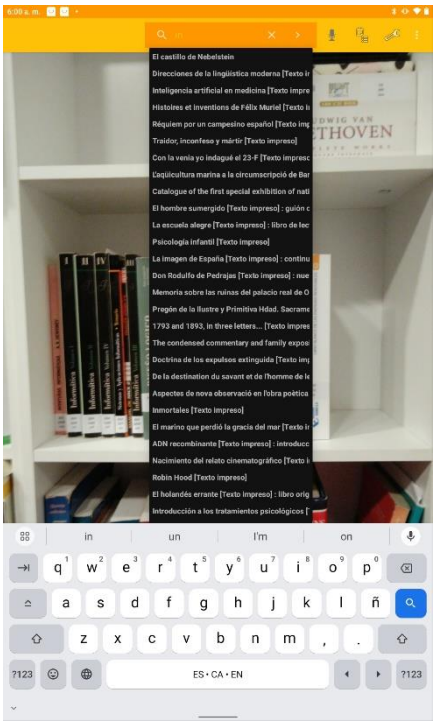


figura 136. WireFraming de la aplicación. Vista previa vertical. Opción de autocompletado.

13.1.1.2.3 Pantalla de detalle. Detección de textos sobre la imagen capturada.

La pantalla de detalle nos permite consultar datos sobre los ejemplares detectados en función de los filtros de búsqueda establecidos. La búsqueda de información se basa en los campos que contiene un registro bibliográfico, como son: título, ISBN, editorial y autores. Dado que consideramos que sería de utilidad, se plantea como vía futura poder realizar la búsqueda basándose también en los campos que contiene un ejemplar, como son signatura y código de barras. Mediante la presente pantalla pretendemos obtener información de los ejemplares asociados a uno o varios registros bibliográficos, y si especificamos como campo de búsqueda el título activando la palabra detectada “Java”, el sistema buscará sobre los registros cuyo título contienen la palabra Java y mostrará todos los ejemplares asociados a estos.

En la figura 137, podemos observar un ejemplo referente a la detección dentro de la pantalla de detalle. En ella podemos observar varios aspectos relevantes:

- Se muestra como fondo la imagen capturada en la vista previa, objeto de un análisis más profundo dentro de esta.
- En la parte izquierda de la pantalla disponemos de dos botones grises sobre los que podemos pulsar para desplegar los resultados. El superior se corresponde con los resultados que obtengamos del servidor OPAC HTML y el inferior del servidor PMB.
- La barra de herramientas tiene características comunes con la explicada en la pantalla de vista previa.
- Los textos se han acumulado, dado el alto contenido de estos en la imagen y por tanto se hace vital la posibilidad de filtraje que se explicó en la vista previa y que en esta está disponible.
- El patrón de filtraje establecido en la vista previa se mantendrá al acceder a esta pantalla y por tanto no será necesario volver a introducirlo.
- En esta ocasión el enmarcado de cada texto detectado es más elaborado y posee un “conmutador” en la parte izquierda para su activación o desactivación.

- En la parte inferior derecha nos encontramos con dos grupos de acciones. Por un lado, podemos establecer un tipo de consulta sobre los servidores de tipo “OR” o de tipo “AND”. Hay otro grupo de acciones que nos permite detectar los textos como bloques de textos más grandes (párrafo), como líneas de texto o como palabras individuales.
- Existe una barra deslizante en la parte derecha de la pantalla, que, sobre un fondo blanco, nos permitirá hacer más o menos transparente la imagen capturada para poder distinguir con mayor facilidad los textos detectados.
- Podemos pulsar sobre la parte izquierda de un cuadro de texto para efectuar la acción de “arrastrar y soltar” con vistas a reposicionarlo y ver los textos con mayor claridad.

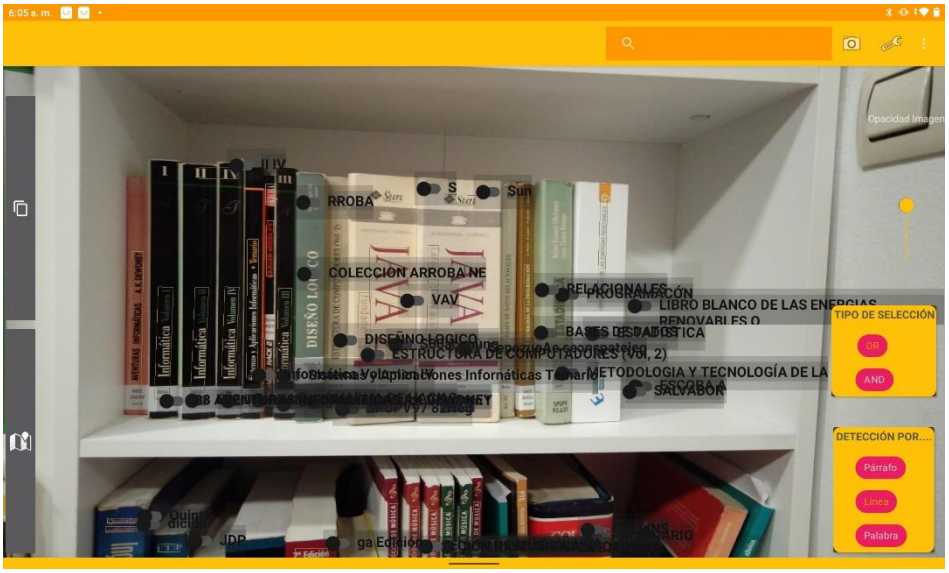


figura 137. WireFraming de la aplicación. Ejemplo de pantalla de detalle.

En la figura 138, podemos observar como hemos establecido el patrón de filtraje “di” y por tanto solo enmarca aquellos ejemplares que contienen dicho patrón. También como hemos desplegado en la barra de herramientas las opciones ocultas bajo los tres puntos verticales (ayuda, términos de servicio,

etc.). En la figura 139, podemos observar como hemos activado los cuadros correspondientes a “diseño” y a “lógico”, hemos desplegado el selector de campo de búsqueda (parte derecha del cuadro) y hemos seleccionado título. También podemos observar como la detección ha sido por palabra y en la parte izquierda tenemos seleccionada el tipo de búsqueda “OR”, por lo que se mostrarán todos aquellos ejemplares que posean la palabra “diseño” y los que posean la palabra “lógico”. En caso de haber seleccionado el tipo de búsqueda “AND”, solo mostraría los ejemplares que poseyeran las dos palabras al mismo tiempo “diseño lógico”.

figura 138. WireFraming de la aplicación. Ejemplo de pantalla de detalle estableciendo un filtro.

En el cuadro de resultados de la figura 139, podemos observar la información obtenida del registro bibliográfico (coloreada en franjas azuladas y naranjas) y la del ejemplar asociado (franjas azuladas y blancas). También que, en la parte superior de este cuadro de resultados, disponemos de unos botones para mostrar u ocultar la información referente a título, ISBN, autor, etc.

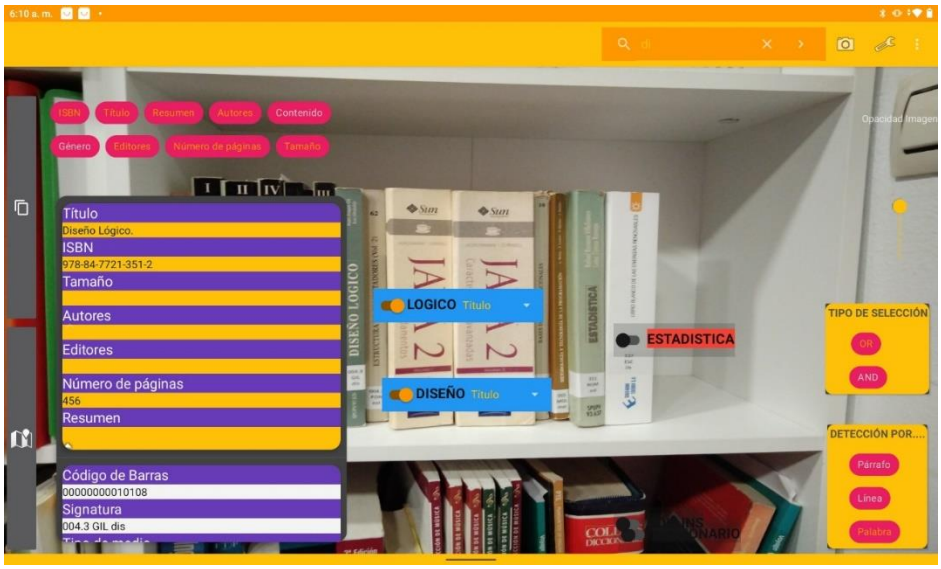


figura 139. WireFraming de la aplicación. Ejemplo de consulta sobre el servidor PMB.

En la figura 140, podemos observar el resultado de la consulta realizada al catálogo en línea de la biblioteca de la UNED. El modo de proceder es el mismo que en la consulta al servidor de PMB, solo que en esta ocasión solo hemos seleccionado la palabra diseño. Dado que los resultados se actualizan en tiempo real cada vez que activamos un campo de texto u otro, o variamos el campo de búsqueda (título, ISBN, etc.), realmente se interrogan al mismo tiempo ambos servidores, por lo que tan solo deberemos desplegar un cuadro de resultados u otro para visualizar los mismos. En la figura 141, podemos observar la consulta mediante la detección de un ejemplar fuera de la estantería de pruebas. Con este ejemplo podemos ver el potencial de la herramienta, dado que puede detectar cualquier texto presente en cualquier hoja del ejemplar, las funcionalidades de la aplicación podrían irse ampliando con el tiempo y en función de las necesidades.

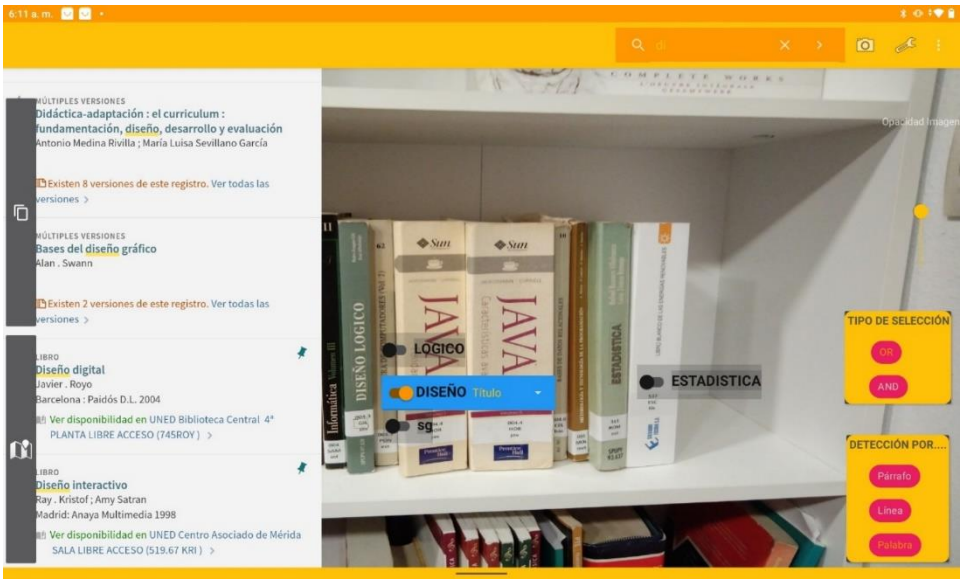


figura 140. WireFraming de la aplicación. Ejemplo de consulta sobre el catálogo bibliotecario de la UNED.

figura 141. WireFraming de la aplicación. Pantalla de detalle. Ejemplo de consulta sobre un ejemplar extraído de la librería de pruebas.

13.1.1.2.4 Pantalla de preferencias.

En la pantalla de preferencias podemos configurar algunos aspectos relevantes de la aplicación en cuanto a forma de proceder, estilo y contenidos a mostrar. En la figura 142, podemos observar la sección correspondiente a las preferencias de la pantalla de vista previa de nuestra aplicación. Se han agrupado en “contenido a mostrar” y en “apariencia”. En cuanto al contenido a mostrar, podemos decidir que aparezca la información flotante (texto detectado) o no y si deseamos realizar una detección por párrafo, línea o palabra. Para cambiar la preferencia actual, en el caso primero debemos conmutar el “Switch” presente a la derecha del texto, y en el caso de la selección del tipo de detección, pulsando sobre el texto se nos abrirá un cuadro de dialogo emergente para que seleccionemos la opción deseada.

En esta misma figura podemos observar el apartado de configuración de la apariencia de la vista previa. Por un lado, podemos modificar el color actual de la línea con la que enmarcaremos los textos detectados, y el color de la información flotante (el texto detectado). Para ello pulsaremos sobre la opción deseada y se nos abrirá un cuadro de dialogo como el de la figura 145. Del mismo modo, podemos variar el tamaño del texto a mostrar deslizando de forma horizontal la opción correspondiente que en la figura podemos observar que se encuentra en un tamaño de texto de 45. Poner el texto en negrita o cursiva es otra opción, para la que simplemente habrá que marcar los cuadros de la derecha del texto.

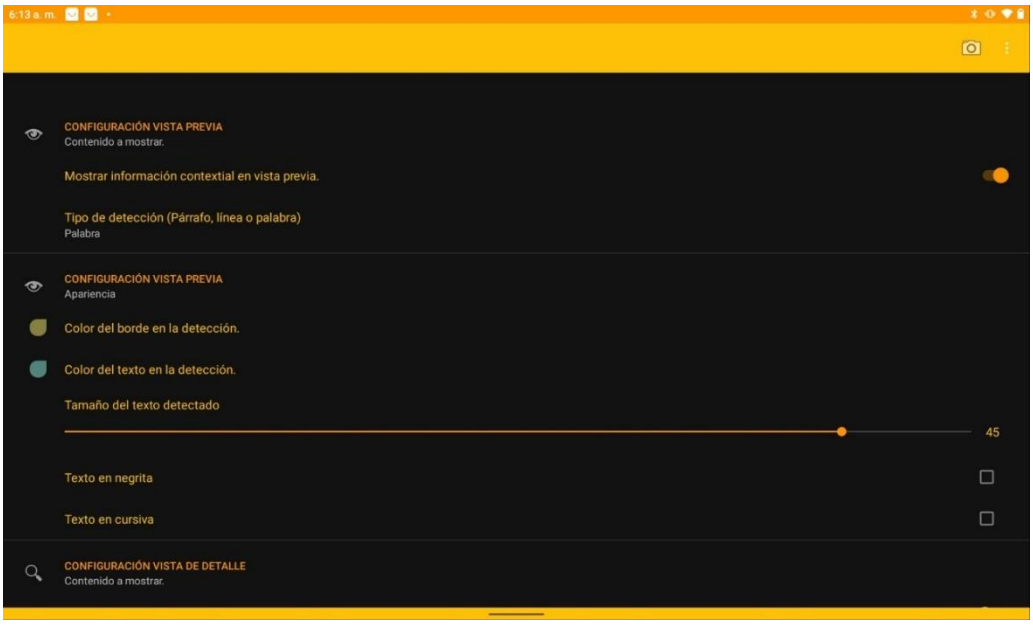


figura 142. WireFraming de la aplicación. Pantalla de preferencias de la vista previa.

En cuanto a la configuración de las preferencias asociadas a la pantalla de detalle de nuestra aplicación, las podemos observar en la figura 143. En primer lugar, nos encontramos con un texto que nos indica si deseamos activar la lectura en voz alta. Al cambiar de posición el conmutador que aparece en la parte derecha del texto, habilitaremos dicha opción y al detectar los textos en la vista de detalle, estos se nos leerán en voz alta.

A continuación, podemos observar un listado de elementos que nos indica Mostrar Título, Mostrar ISBN, etc. Estos se corresponden con la información que deseamos mostrar en la vista de detalle sobre los resultados obtenidos de un registro bibliotecario al interrogar a PMB. El marcaje de alguno de estos campos hará que, al acceder a la pantalla de detalle, este se muestre de forma predeterminada. Sin embargo, en tiempo de ejecución como ya se ha explicado en el apartado anterior, se podrá decidir si mostrar dicha información o no seleccionando las opciones correspondientes. En la figura 144, podemos observar el resto de opciones configurables para la pantalla de detalle. Por un lado, podemos decidir si mostrar información del servidor OPAC HTML o del servidor PMB de forma independiente o simultánea. Esto se traducirá en que nos aparecerá un solo botón para desplegar los resultados, o ambos.

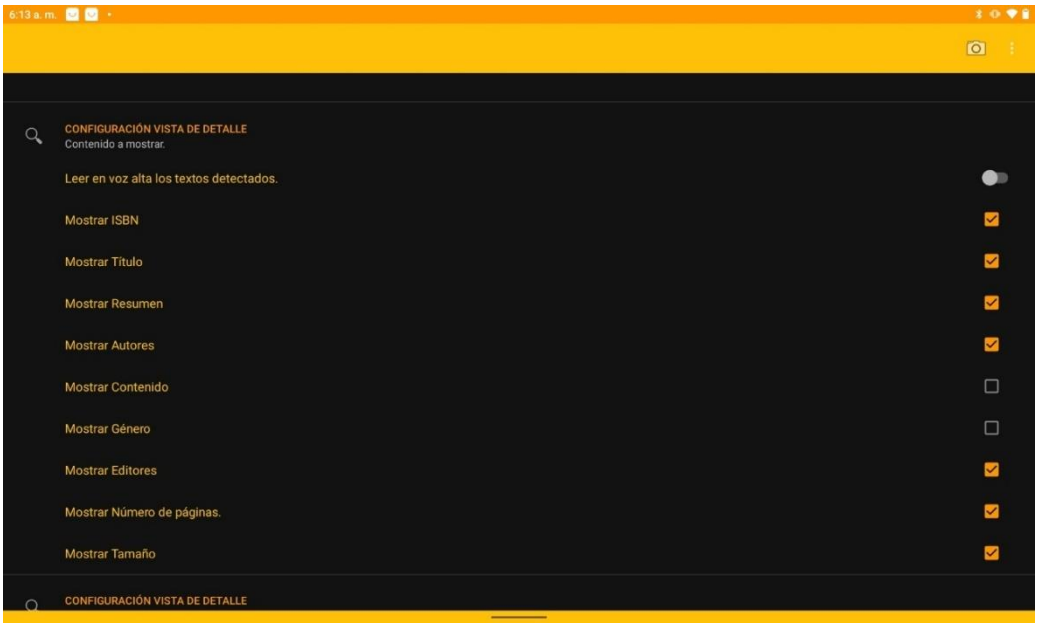


figura 143. WireFraming de la aplicación. Pantalla de preferencias. Configuración vista detalle.

También podemos observar cómo existen tres opciones para configurar el servidor PMB. Por un lado, la que nos permite indicar la dirección del mismo y que al pulsarla hará que nos aparezca un cuadro de dialogo como el de la figura 146 y por otro el usuario y contraseña que será empleado para comunicarse con el servidor PMB.

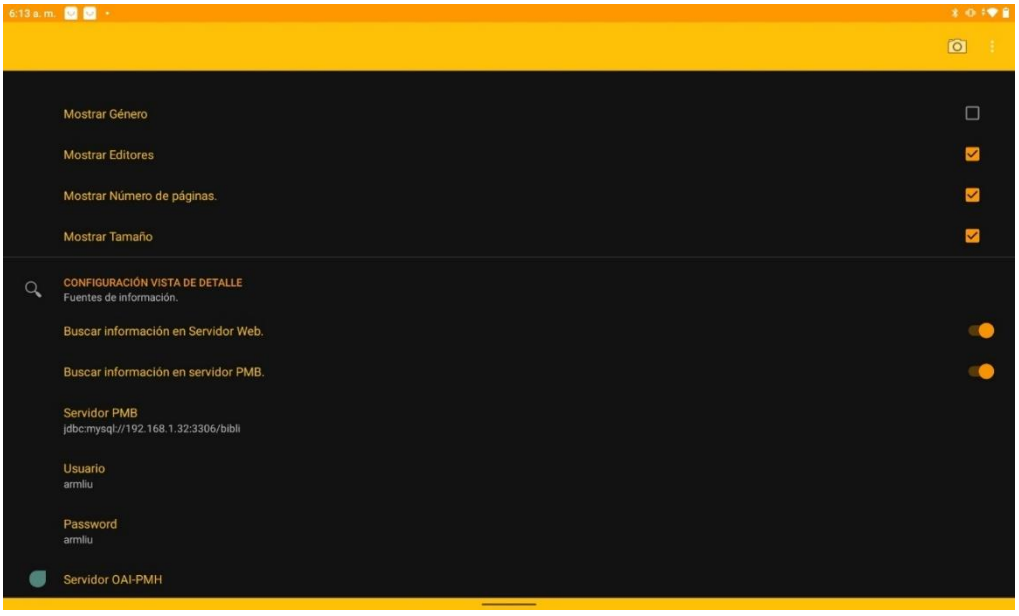


figura 144. WireFraming de la aplicación. Pantalla de preferencias. Configuración vista de detalle.

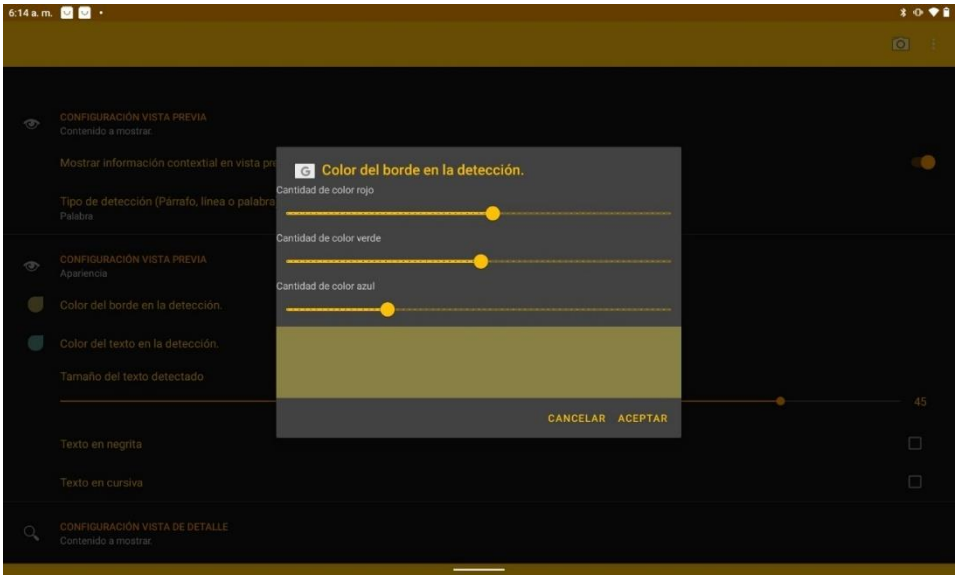


figura 145. WireFraming de la aplicación. Pantalla de preferencias. Configuración del color.

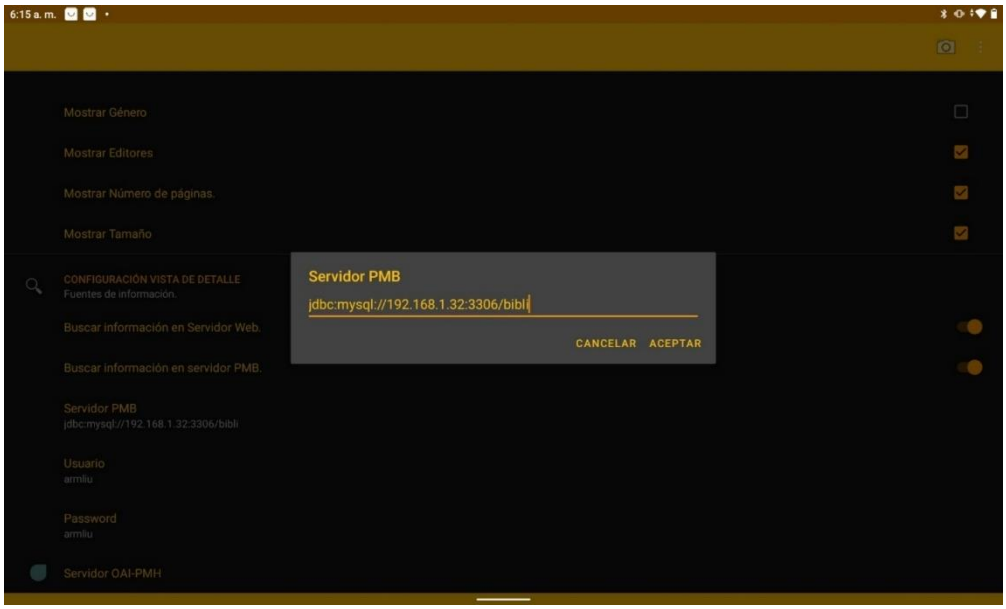


figura 146. WireFraming de la aplicación. Pantalla de preferencias. Configuración del servidor PMB a utilizar.

Como última opción de la figura 144, nos encontramos con el texto Servidor OAI-PMH. Al pulsar sobre esta opción se nos mostrará un cuadro de dialogo como el de la figura 147. Sobre este podemos realizar varias acciones. Por un lado, podemos indicar un nuevo servidor OAI de información (por defecto aparece configurado el de la Biblioteca Nacional de España) cambiando el texto correspondiente y pulsando sobre el botón aceptar para validar los cambios. Por otro y sobre el servidor configurado, podemos cargar nuevos datos sobre la base de datos local del dispositivo. Estos datos se corresponderán con los ofrecidos en la opción de autocompletado del cuadro de filtraje (tanto de la pantalla de vista previa como de la de detalle).

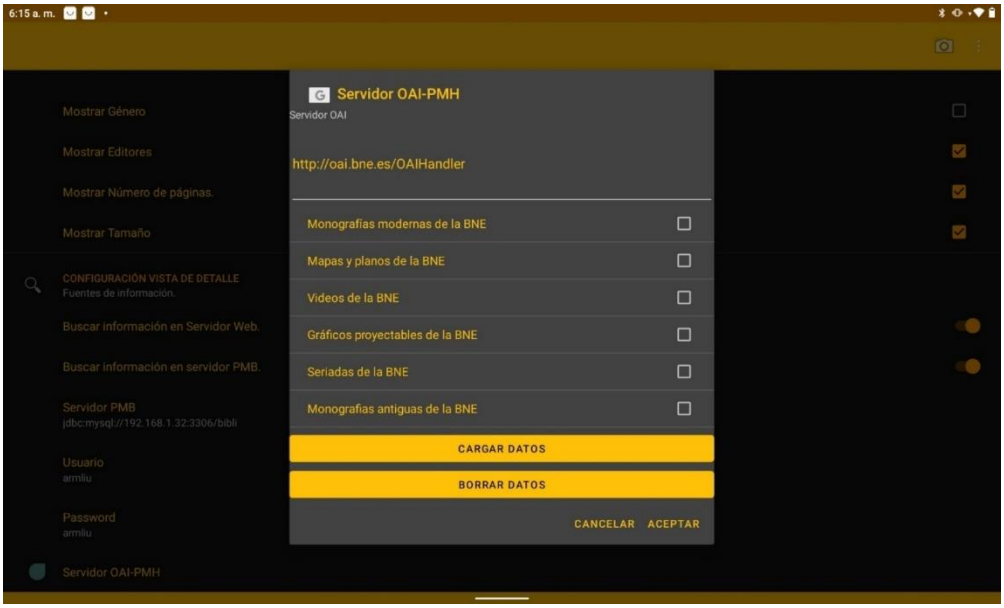


figura 147. Borrado y carga de datos del servidor OAI.

En la figura 147, podemos observar como se nos ofrece una serie de elementos para que los seleccionemos y procedamos a su importación. Denotar que este listado se obtiene de forma dinámica realizando una consulta sobre el servidor OAI en cuanto a los grupos de los que dispone. Por lo que, en función de los datos presentes en el servidor en el momento de realizar la carga, aparecerán unos u otros. Si seleccionamos algún grupo y pulsamos sobre el botón Cargar Datos, estos serán almacenados de forma local.

Podemos observar como también disponemos de un botón de borrado. Al ejecutar esta acción, los datos de la base de datos local del dispositivo serán eliminados.

13.1.1.3 Casos de uso.

De entre las múltiples posibilidades, finalmente hemos empleado StarUML para elaborar los presentes casos de uso. Los diagramas de caso de uso los hemos usado para complementar la definición de los requisitos funcionales. Pertenecen al grupo de "diagramas de comportamiento" definidos en el lenguaje UML y nos permiten definir de forma inequívoca los actores externos que interactuarán con nuestro sistema y a través de que funcionalidades.

13.1.1.3.1 Especificación de actores

Podemos considerar un actor como algo o alguien externo al sistema que interactúa de forma directa con él. En nuestro caso disponemos de los actores que se representan en la Tabla 1.

Actor	Descripción del actor.
Usuario	Cualquier usuario que utilice la aplicación. Dado que no se han definido perfiles de usuario, todos los usuarios podrán realizar el conjunto de acciones que permite la aplicación.
PMB	Servidor PMB.
OPAC_HTML	Catálogo en línea de la biblioteca de la UNED.
OAI.	Servidor OAI empleado para cosechar datos.
SQLite	Base de datos local al dispositivo.

Tabla 1. Actores empleados en los casos de uso.

13.1.1.3.2 Caso de uso de nivel superior.

En la figura 148, podemos observar los tres casos de uso principales de nuestro sistema junto a los actores con los que interactúa cada uno de ellos.

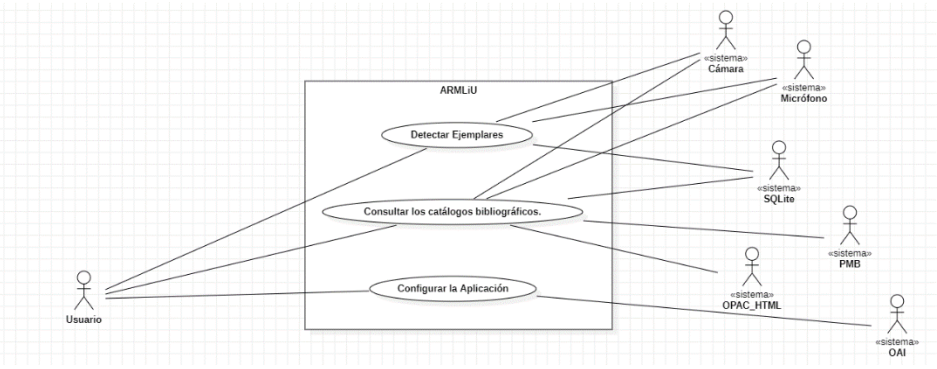


figura 148. ARMLiu. Diagrama de Casos de Uso de nivel superior.

El caso de uso “Detectar Ejemplares”, se corresponde con la vista previa de nuestra aplicación. Mediante el proceso de detección en tiempo real, vamos recorriendo las diferentes estanterías e identificando el/los ejemplares que nos interesan. El caso de uso “Consultar los catálogos bibliográficos” se corresponde con nuestra pantalla de detalle que, una vez identificados los ejemplares, nos permite interrogar diferentes sistemas externos y mostrar los resultados. Y el caso de uso “Configurar la Aplicación”, se corresponde con la pantalla de preferencias.

13.1.1.3.3 Caso de uso Detección de ejemplares.

En la figura 149, podemos observar los casos de uso que integran el caso

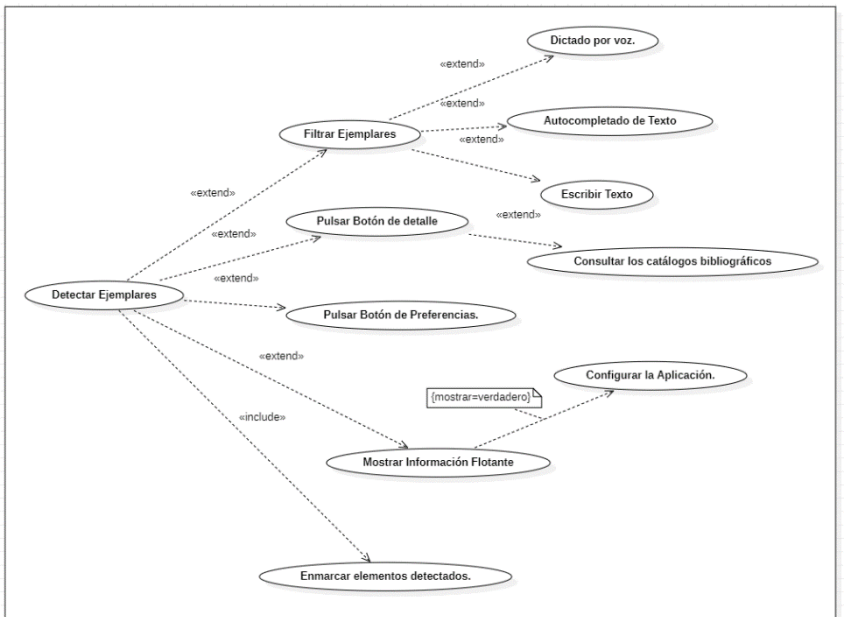


figura 149. Detalle caso de uso Detección de Ejemplares.

de uso “detección de ejemplares”. El estereotipo <<extends>> lo hemos empleado como una inclusión de otro caso de uso, pero de manera opcional (a diferencia del estereotipo <<include>>). Por ejemplo, el filtraje de elementos puede ser realizado o no.

13.1.1.3.4 Caso de uso Consultar los Catálogos Bibliográficos.

En la figura 150, podemos observar los casos de uso relacionados con el caso de uso Consultar Catálogos Bibliográficos.

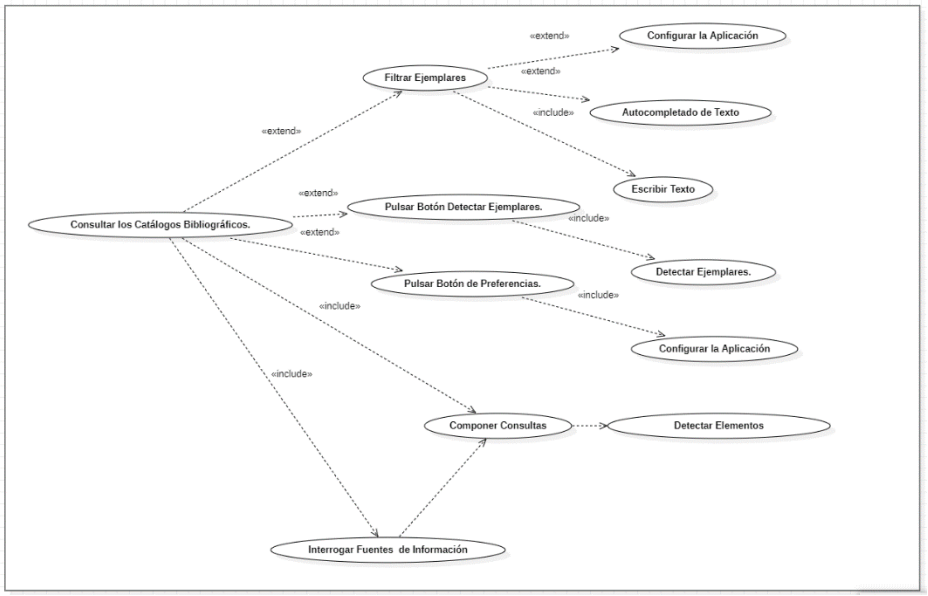


figura 150. Detalle caso de uso Consultar los Catálogos Bibliográficos.

13.1.1.3.5 Caso de uso Configurar la aplicación.

En la figura 151, podemos observar el detalle de los casos de uso que componen el caso de uso Configurar la Aplicación.

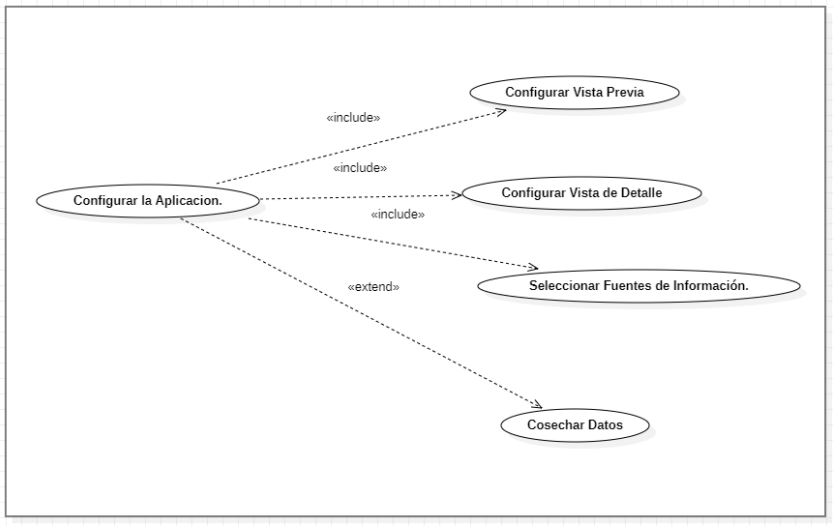


figura 151. Detalle caso de uso Configurar la Aplicación.

13.1.2 Requisitos no funcionales.

La aplicación deberá respetar las diferentes recomendaciones/directrices marcadas por Google para los desarrolladores Android y ofrecer unos tiempos de respuesta razonables, tanto al cargar las diferentes pantallas, como al

realizar el proceso de detección de textos, dónde la latencia en cuanto a la detección de textos deberá ser lo suficientemente pequeña como para ofrecer una buena experiencia de usuario.

La calidad en la detección de los textos deberá ser lo suficientemente alta como para que los textos detectados se correspondan con los textos presentes en la realidad. La interacción con los sistemas externos deberá ser lo suficientemente segura como para no poner en entredicho ni la seguridad del sistema al que se esté accediendo, ni la del dispositivo del usuario.

En definitiva, el usuario deberá percibir que está ejecutando una aplicación estable, de calidad y segura, dónde su experiencia es altamente gratificante y se cumplan sus expectativas de uso.

13.2 Proyecto en Android Studio.

13.2.1 Descripción del código fuente.

Dada la extensión del código fuente de la aplicación, no es viable su inclusión completa en la presente memoria. En su defecto se realizará una descripción de aquellos aspectos más relevantes del mismo y dado que se pone a la disposición, instamos a su consulta.

El proyecto se ha estructurado en una serie de **paquetes de clases Java** que complementa la estructura por defecto impuesta por Android Studio. Los paquetes más relevantes se corresponden con:

adapters: incluye las clases que hemos creado para trabajar con adaptadores.

db: incluye las clases creadas para trabajar con diferentes tipos de bases de datos.

detail: incluye las clases asociadas a la vista de detalle de la aplicación.

help: incluye las clases asociadas a la ayuda de la aplicación.

oai: incluye las clases para trabajar con servidores OAI-PMH.

opac: incluye las clases asociadas para trabajar con diferentes catálogos online de diferentes bibliotecas basadas en HTML.

pmb: incluye las clases asociadas al trabajo con el servidor PMB.

preferences: incluye las clases asociadas con la gestión de preferencias de la aplicación por parte del usuario.

preview: incluye las clases asociadas con la vista previa de la aplicación.

utils: incluye clases de soporte empleadas por el resto.

Dentro de la estructura por defecto de Android Studio tenemos la carpeta res (resources) y las configuraciones Gradle tenidas en cuenta para la compilación y generación de diferentes “releases” de la aplicación. Dentro de la carpeta “res”:

color: la hemos creado para albergar nuestros selectores de color en función de los estados en los que se encuentran diferentes elementos de nuestra aplicación. Por ejemplo, los “Chips” de selección del tipo de búsqueda a realizar, o el color del componente de detalle en función de si se encuentra seleccionado o no.

drawable: en esta carpeta nos encontramos la definición de los diferentes iconos de la aplicación.

layout: ficheros XML (eXtensible Markup Language) asociados a las diferentes interfaces de usuario (contenedores de cuadros de texto, botones, etc.) y que serán asociados con la lógica de negocio correspondiente.

menu: menús empleados en la aplicación, en lugar de emplear una barra de herramientas genérica y pese a la similitud de esta en las diferentes pantallas, se ha optado por crear un menú para cada pantalla, que pese a que hace más laboriosa las modificaciones futuras nos permitirá mayor flexibilidad y variabilidad en estas.

mipmap: es un caso específico de drawable, dado que son los iconos empleados para mostrar al usuario en su grupo de aplicaciones para que pulse sobre él y ejecute la aplicación, el icono que aparece mientras se está abriendo, etc. En definitiva, los iconos empleados para el lanzamiento de la aplicación.

navigation: carpeta contenedora de los gráficos de navegación.

values: carpeta contenedora de diferentes valores que se expresan dentro de ficheros XML. Por ejemplo, esta carpeta contiene ficheros XML asociados con la definición de temas y estilos de la aplicación, textos que serán referenciados desde las diferentes etiquetas para poder modificar los textos de la aplicación de una forma centralizada (strings.xml) y para poder indicar diferentes versiones de ficheros de strings.xml (como es nuestro caso) y que la aplicación esté disponible en varios idiomas en función del "locale" configurado por el usuario en su dispositivo. También la definición de "arrays" (clave-valor) para cargar de forma dinámica valores en los cuadros de selección de valores y por tanto hacer más dinámica su posterior modificación.

xml: por defecto la carpeta contenedora de los ficheros asociados a las "SharedPreferences" de nuestra aplicación.

13.2.1.1 AndroidManifest.xml.

El fichero de manifiesto contiene información relevante de nuestro proyecto como la actividad que será iniciada en primera instancia y con qué clase estará asociada esta. En la figura 154, podemos observar el uso de la clase MainActivity para lanzar la actividad y como le estamos indicando mediante un Intent (objeto que nos sirve para intercambiar mensajes entre los diferentes componentes de un aplicación) la acción que deseamos realizar "android.intent.action.MAIN" y a que categoría pertenece "android.intent.category.LAUNCHER". En definitiva, que deseamos utilizar la clase MainActivity como punto de entrada a nuestra aplicación. En esta ocasión la acción se corresponde con la principal.

También le podemos indicar aquellos complementos/dependencias que deseamos que se descarguen automáticamente al instalar nuestra aplicación, como es el caso de la figura 153, en la que le indicamos que descargue los ficheros asociados al kit de ML (Machine Learning) que hemos empleado para el reconocimiento de texto.

```
<meta-data
    android:name="com.google.mlkit.vision.DEPENDENCIES"  android:value="barcode,face,ocr,ica,custom_ica"/>
```

figura 153. Fichero de Manifiesto. Descarga automática de dependencias de la aplicación.

```
<!-- permiso para que funcione el micrófono y query para que funcione el servicio de reconocimi
<uses-permission android:name="android.permission.RECORD_AUDIO" />
**<queries>
<intent>
    <action android:name="android.speech.RecognitionService" />
</intent>
</queries>**

<!-- con required=true le indicamos que si no tiene cámara o micrófono la aplicación no debe ser instalada -->
<uses-feature android:name="android.hardware.camera" android:required="true"/>

<uses-feature android:name="android.hardware.microphone" android:required="true"/>
<uses-feature android:name="android.hardware.vr.headtracking" android:required="true"/>

<uses-feature android:name="android.permission.CLEAR_APP_CACHE" />
<uses-feature android:name="android.permission.DELETE_CACHE_FILES"/>
```

figura 152. Inclusión de permisos necesarios dentro del fichero de manifiesto.

Otro contenido importante del fichero de manifiesto es la solicitud de permisos para que nuestra aplicación funcione correctamente. En nuestro caso, en caso de no poder emplear la cámara presente en el dispositivo para aplicar técnicas de realidad aumentada, la aplicación pierde una de las características más importantes y por tanto debemos incluir en el “manifest” un listado de aquellos permiso que el usuario deberá otorgar (figura 152).

De forma genérica la forma de indicar la necesidad de determinados permisos es bajo la etiqueta “uses-permission” tal cual se puede observar en la figura en la que solicitamos el permiso de grabación de audio para poder emplear el reconocimiento de voz (voz a texto) presente en los filtros de la pantalla de previsualización y detalle de nuestra aplicación. Denotar que además de solicitar los permisos necesarios, deberíamos de indicar aquellas características (hardware por ejemplo) que necesitará nuestra aplicación. Para tal fin empleamos la etiqueta “uses-feature” donde le indicamos además si una característica será imprescindible o no. Indicando “android:required=’true’” (por

defecto es true), estamos expresando nuestro deseo de que si no está disponible dicha característica la aplicación no debería ser instalada, dado que carecería de sentido. Indicar las características y si son requeridas o no es de vital importancia, dado que “Google Play Store” se basa en esta información para ofrecer la aplicación a determinados dispositivos o no.

13.2.1.2 MainActivity

Como ya se ha comentado, esta clase es la que empleamos como punto de entrada a nuestra aplicación y en nuestro caso extiende de “AppCompatActivity”. En nuestro caso la hemos convertido en una clase muy simple donde básicamente posee del método “onCreate”, donde realizamos el “inflado” de la actividad principal.

figura 154. Clase MainActivity.

Frente a métodos tradicionales (basados principalmente en el uso del método findViewById) dónde la vinculación de una vista y por tanto del fichero XML con los objetos que deberán ser creados para poderlos instanciar posteriormente (por ejemplo la instancia de un TextView donde presentar un texto al usuario), a partir de Android Studio 3.6, se nos brinda la posibilidad de emplear el “inflate” de objetos para realizar el “binding” (enlazado) de forma automática. Esta forma de proceder es mucho más cómoda y simplifica enormemente el proceso. En la figura 154, podemos observar como inflamos el contenido del fichero **activity_main.xml**, dado que estamos empleando la clase ActivityMainBinding y tomará este fichero por defecto.

figura 155. Fichero activity_main.xml

Podemos observar en la figura 155, que este fichero contiene la vista de un contenedor de fragmentos. Nuevamente y frente otras prácticas tradicionales, Google recomienda el uso de una única actividad que contenga uno o múltiples fragmentos. En nuestro caso los fragmentos principales son: el de vista previa, el de pantalla de detalle, el de pantalla de configuración, así como los de la pantalla de ayuda y términos de uso. Se ha creado un gráfico de navegación entre dichos fragmentos que nos permitirá definir la secuencia de acciones para navegar entre estos de forma gráfica y textual mediante XML (figura 156).

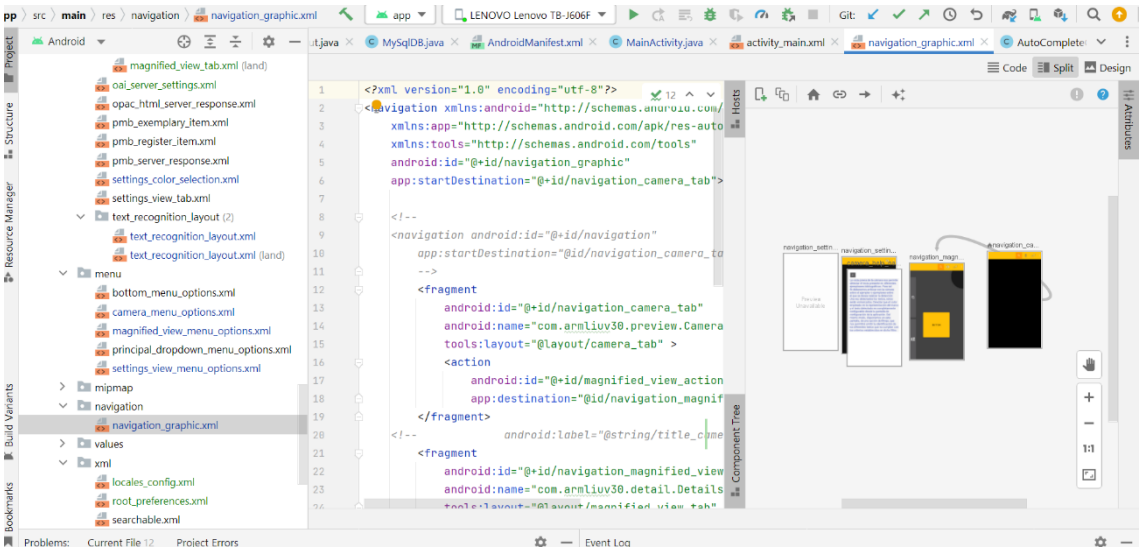


figura 156. Gráfico de navegación entre fragmentos.

Para cada fragmento le indicamos, el id dentro del gráfico de navegación para que determinadas acciones puedan referenciar a este, la clase y el "layout" sobre el que se apoya, así como la siguiente acción a realizar (dónde debe ir cuando se solicite dicha acción). Si nos fijamos por ejemplo, en el "fragment" dentro del gráfico de navegación identificado como "navigation_camera_tab", este en "name" contiene la clase "CameraFrament", en "layout" "CameraTab" (fichero XML asociado a dicho fragmento) y en acción, por un lado un id que será el empleado para desde cualquier clase válida, referenciando dicho id (magnified_view_action), se solicitará ejecutar esta acción que en este caso concreto consiste en desplazarse al elemento presente en el árbol de navegación, "navigation_magnified_view_tab" (desplazamiento a la vista de detalle). El gráfico de navegación está estrechamente relacionado con las barras de herramientas que se han establecido para cada una de las pantallas de la aplicación. Es decir, pulsar por ejemplo un botón de dicha barra de herramientas (que nos muestra las diferentes opciones de menú), invocará una acción del presente gráfico de navegación. En la figura 158, podemos observar las diferentes opciones de menú presentes en la vista de detalle. En la figura 157, la lógica asociada a la pulsación de los diferentes elementos de la barra de herramientas, denotando que por ejemplo, "R.id.navigation_camera_tab" invoca a una de las acciones presentes en el gráfico de navegación descrito anteriormente.

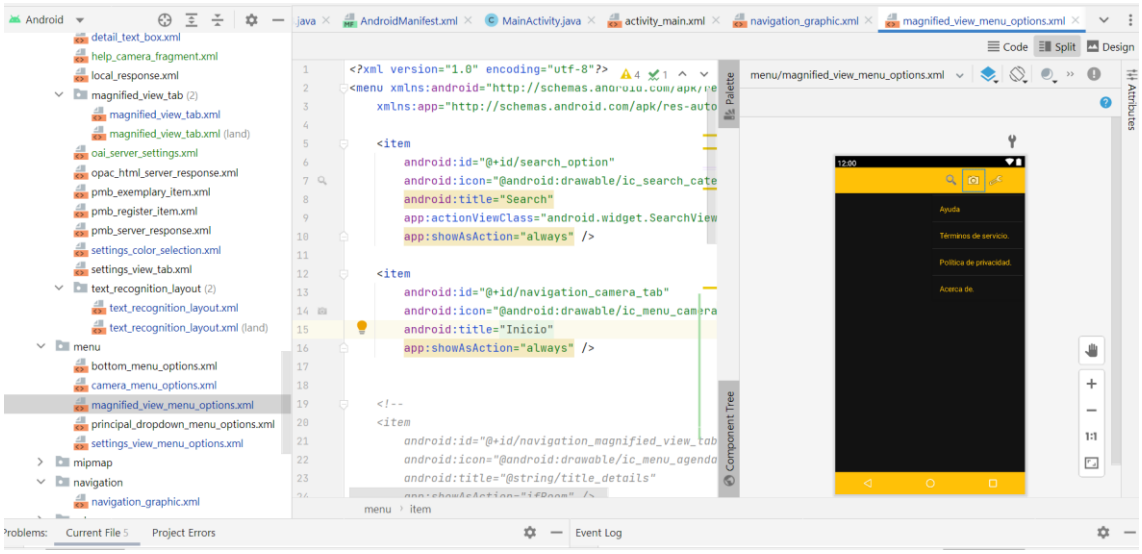


figura 158. Opciones de menú asociadas a la vista de detalle.

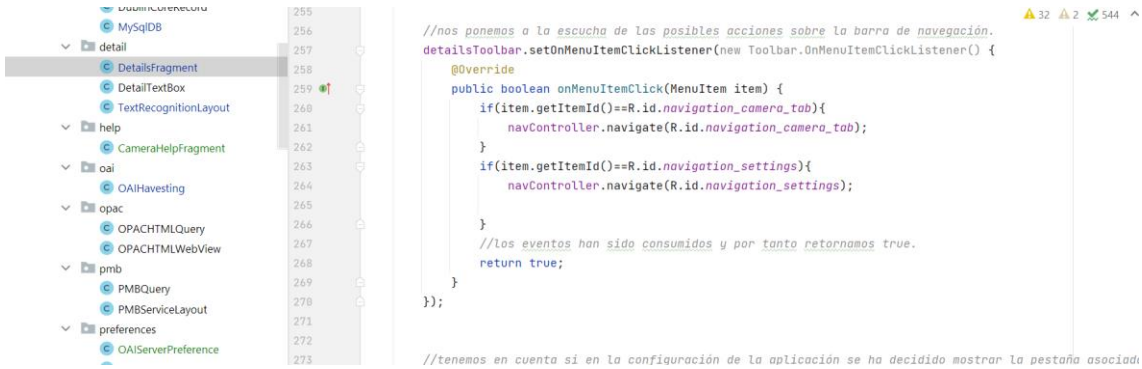


figura 157. Lógica asociada a las diferentes opciones de menú de la vista de detalle.

13.2.1.3 DublinCoreDB.

Esta clase extiende la clase “SQLLiteOpenHelper” que nos permite interactuar con la base de datos SQLite local. Esta clase la hemos creado para abstraernos del proceso de almacenamiento y obtención de datos en formato DC (Dublin Core) a nivel local. Finalmente, los datos obtenidos de un Servidor OAI-PMH y que hemos almacenado serán empleados para mostrar sugerencias de autocompletar en los filtros presentes en la pantalla de vista previa y en la de detalle. En la figura 160, podemos observar la composición de

la consulta de inserción de datos, viéndose claramente como están presentes en ella todos aquellos atributos que contiene un registro en formato DC.

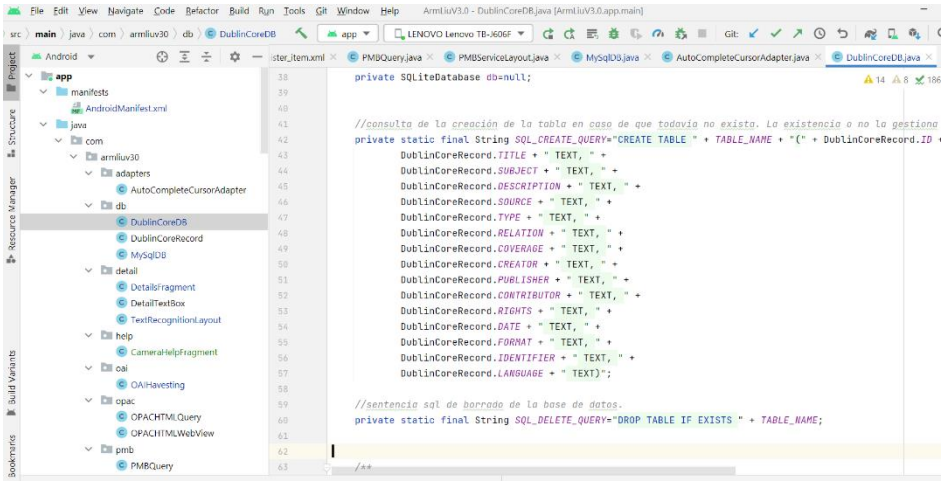


figura 160. Clase DublinCoreDB.

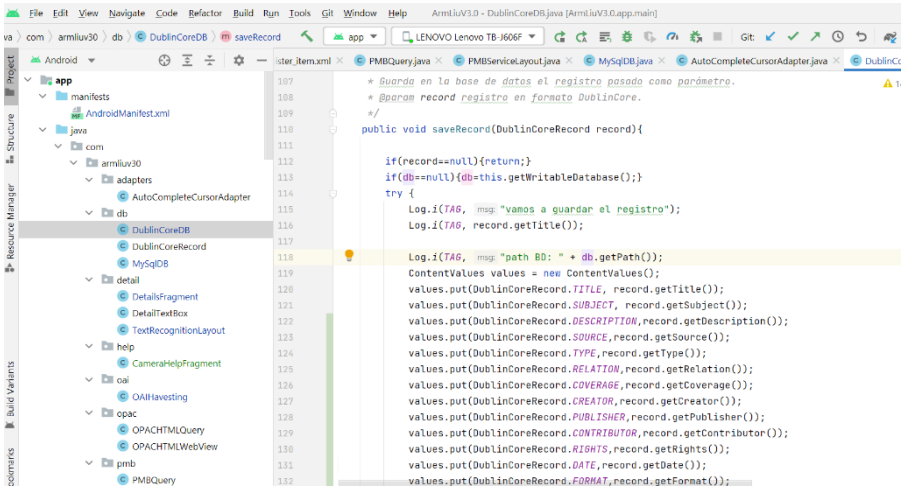


figura 159. Empleo de DublinCoreRecord.

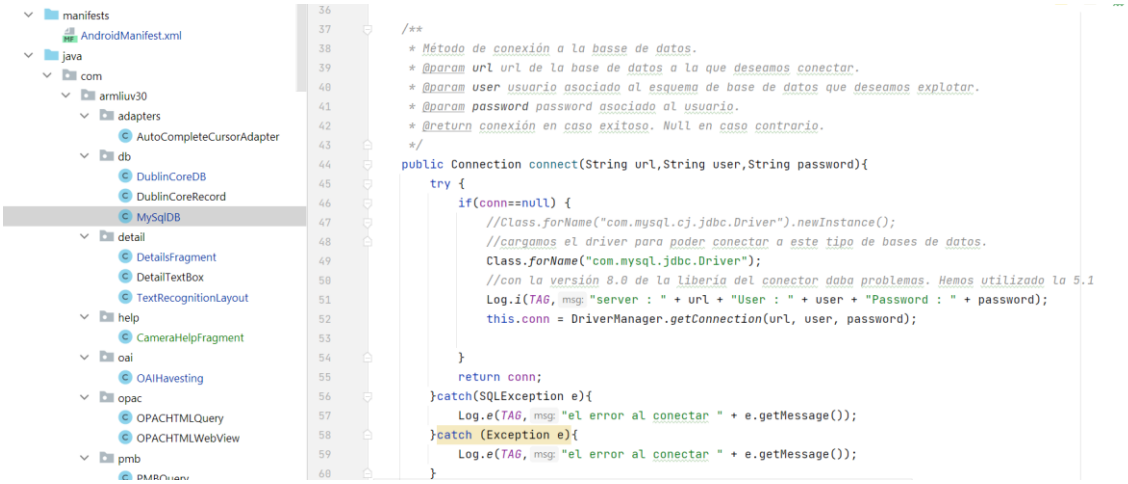
Nos apoyamos en la clase “DublinCoreRecord” que hemos creado a modo de estructura de datos para simplificar el proceso de explotación de los datos almacenados.

```
String groupby = null;
String having = null;
String orderby = null;
String limitRecords = null;

Cursor cursor = db.query(TABLE_NAME, columns, where, whereArgs, groupby, having, orderby, limitRecor
cursor.moveToFirst();
do {
    record = new DublinCoreRecord();
    record.setId(cursor.getLong(cursor.getColumnIndexOrThrow(DublinCoreRecord.ID)));
    record.setTitle(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.TITLE)));
    record.setSubject(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.SUBJECT)));
    record.setDescription(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.DESCRIPTION
    record.setSource(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.SOURCE)));
    record.setType(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.TYPE)));
    record.setRelation(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.RELATION)));
    record.setCoverage(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.COVERAGE)));
    record.setCreator(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.CREATOR)));
    record.setPublisher(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.PUBLISHER)));
    record.setContributor(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.CONTRIBUTOR
    record.setRights(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.RIGHTS)));
    record.setDate(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.DATE)));
    record.setFormat(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.FORMAT)));
    record.setIdentiier(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.IDENTIFIER)))
    record.setLanguage(cursor.getString(cursor.getColumnIndexOrThrow(DublinCoreRecord.LANGUAGE)));
```

figura 161. Recorrido del curso presente en la clase DublinCoreDB.

13.2.1.4 MySqlDB.

Clase empleada para trabajar con bases de datos MySql y sobre la que se apoyarán las consultas a la base de datos bibli de PMB. Se emplea como primer nivel de abstracción para interactuar con este tipo de bases de datos.




```
/**
 * Método de conexión a la basse de datos.
 * @param url url de la base de datos a la que deseamos conectar.
 * @param user usuario asociado al esquema de base de datos que deseamos explotar.
 * @param password password asociado al usuario.
 * @return conexión en caso exitoso. Null en caso contrario.
 */
public Connection connect(String url,String user,String password){
    try {
        if(conn==null) {
            //Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
            //cargamos el driver para poder conectar a este tipo de bases de datos.
            Class.forName("com.mysql.jdbc.Driver");
            //con la versión 8.0 de la libería del conector daba problemas. Hemos utilizado la 5.1
            Log.i(TAG, msg: "server : " + url + "User : " + user + "Password : " + password);
            this.conn = DriverManager.getConnection(url, user, password);

        }
        return conn;
    }catch(SQLException e){
        Log.e(TAG, msg: "el error al conectar " + e.getMessage());
    }catch (Exception e){
        Log.e(TAG, msg: "el error al conectar " + e.getMessage());
    }
```

figura 162. Clase MySqlDB.

13.2.1.5 PMBQuery.

Extiende “Query” y nos abstrae de la consulta a la base de datos bibli, permitiendo el filtraje de valores por los campos presentes en la vista de detalle: Título, ISBN, editorial y autores.

En esta clase nos encontramos con la composición de la “query sql” y con la adaptación del nombre de los campos presentes en nuestra aplicación y que se ha generalizado para poder interrogar diferentes sistemas como son el OPAC HTML, o las tablas de la base de datos bibli.

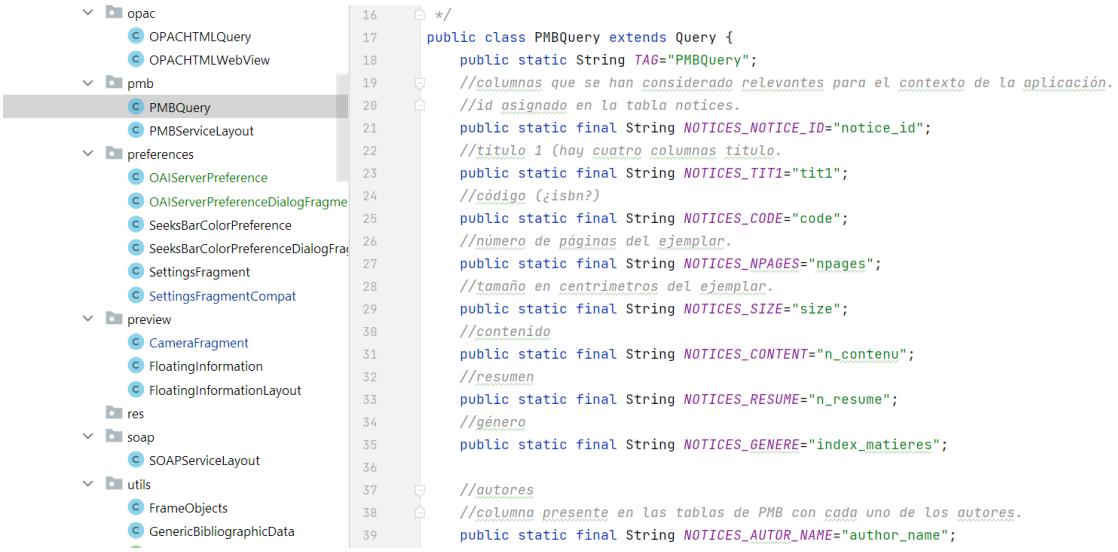


figura 163. Fragmento PMBQuery

figura 164. Composición de la Query en PMBQuery

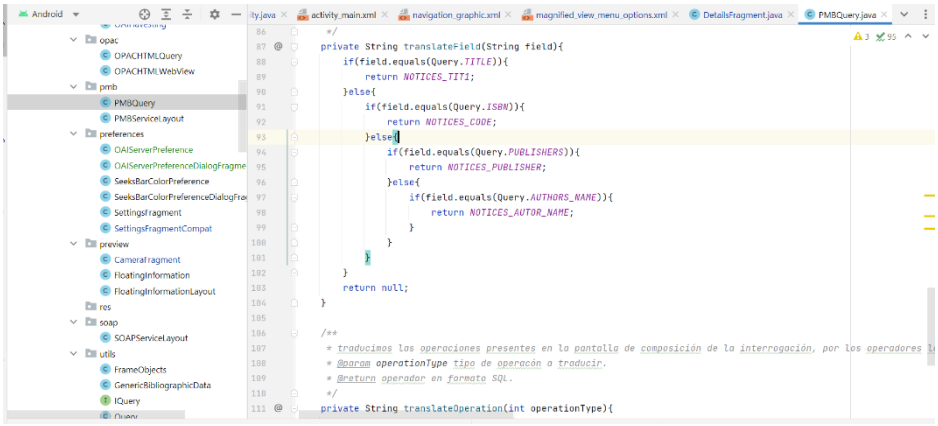


figura 165. Adaptación de campos de búsqueda y tipo de operación a la sintaxis presente en las tablas de la base de datos bibli de PMB.

13.2.1.6 PMBServiceLayout.

Obtiene los datos de la base de datos bibli sobre la que se apoya PMB y los presenta en un “Layout” para mostrárselos al usuario, teniendo en cuenta la configuración de las preferencias de la aplicación y los filtros establecidos en tiempo de ejecución sobre los datos que se desean obtener. Las consultas las realiza principalmente sobre la tabla “notices” que se corresponde con cada uno de los registros bibliográficos y sobre la tabla “exemplaires” que se corresponde con cada uno de los ejemplares asociados a cada registro, obteniendo información adicional de otras tablas.

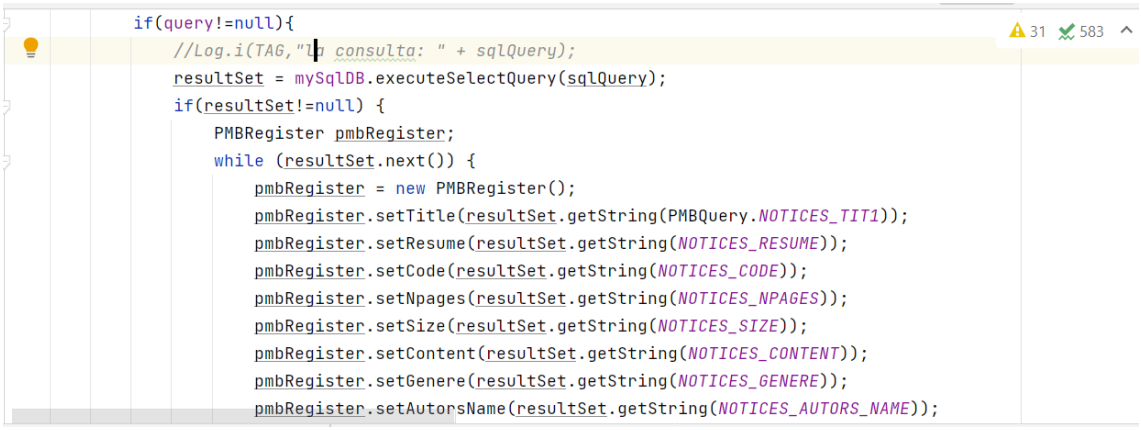
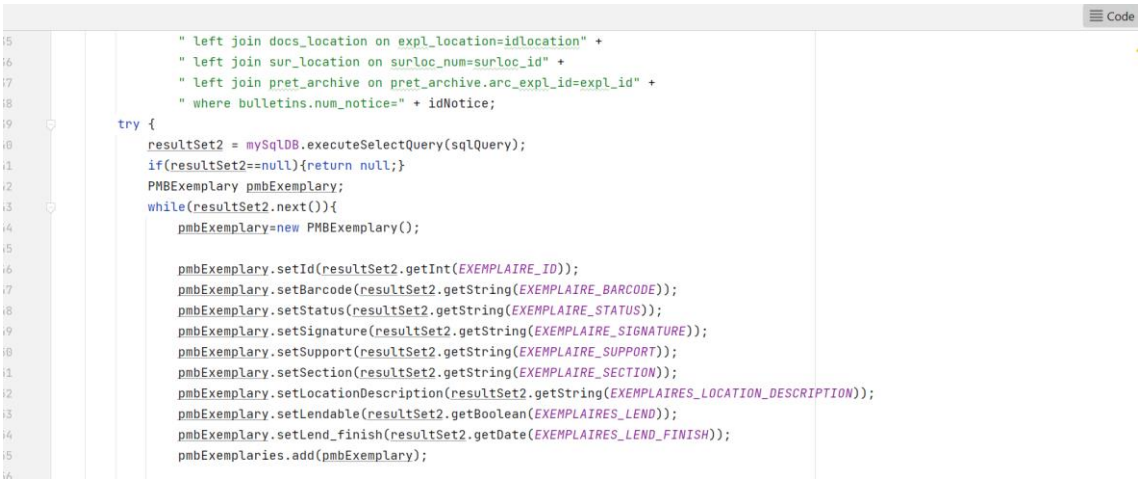


figura 166. Ejecución de la Query sobre notices.

```
String sqlQuery="select expl_id, expl_cb, expl_cote, expl_statut,statut_libelle, expl_typdoc, tdoc_libelle, expl_note, expl_comment," +
        "expl_section, section_libelle, expl_owner, lender_libelle, expl_codestat, codestat_libelle," +
        "expl_date_retour, expl_date_depot, expl_note, pret_flag, expl_location, location_libelle, expl_prix,ifnull(surloc_id,0) as surloc_id, ifnull(surloc
        " left join docs_statut on expl_statut=idstatut" +
        " left join docs_type on expl_typdoc=idtyp_doc" +
        " left join docs_section on expl_section=idsection" +
        " left join docs_codestat on expl_codestat=idcode" +
        " left join lenders on expl_owner=idlender" +
        " left join docs_location on expl_location=idlocation" +
        " left join sur_location on surloc_num=surloc_id" +
        " left join pret_archive on pret_archive.arc_expl_id=expl_id" +
        " where expl_notice=" + idNotice +
        " union  select expl_id, expl_cb, expl_cote, expl_statut,statut_libelle, expl_typdoc, tdoc_libelle, expl_note, expl_comment," +
        "expl_section, section_libelle, expl_owner, lender_libelle, expl_codestat, codestat_libelle," +
        "expl_date_retour, expl_date_depot, expl_note, pret_flag, expl_location, location_libelle, expl_prix, ifnull(surloc_id,0) as surloc_id, ifnull(surloc
        " left join bulletins on expl_bulletin=bulletin_id" +
        " left join docs_statut on expl_statut=idstatut" +
        " left join docs_type on expl_typdoc=idtyp_doc" +
        " left join docs_section on expl_section=idsection" +
        " left join docs_codestat on expl_codestat=idcode" +
        " left join lenders on expl_owner=idlender" +
        " left join docs_location on expl_location=idlocation" +
        " left join sur_location on surloc_num=surloc_id" +
        " left join pret_archive on pret_archive.arc_expl_id=expl_id" +
```

figura 168. Query SQL sobre registros y ejemplares.



```
            " left join docs_location on expl_location=idlocation" +
            " left join sur_location on surloc_num=surloc_id" +
            " left join pret_archive on pret_archive.arc_expl_id=expl_id" +
            " where bulletins.num_notice=" + idNotice;
    try {
        resultSet2 = mySqlDB.executeSelectQuery(sqlQuery);
        if(resultSet2==null){return null;}
        PMBExemplary pmbExemplary;
        while(resultSet2.next()){
            pmbExemplary=new PMBExemplary();

            pmbExemplary.setId(resultSet2.getInt(EXEMPLAIRE_ID));
            pmbExemplary.setBarcode(resultSet2.getString(EXEMPLAIRE_BARCODE));
            pmbExemplary.setStatus(resultSet2.getString(EXEMPLAIRE_STATUS));
            pmbExemplary.setSignature(resultSet2.getString(EXEMPLAIRE_SIGNATURE));
            pmbExemplary.setSupport(resultSet2.getString(EXEMPLAIRE_SUPPORT));
            pmbExemplary.setSection(resultSet2.getString(EXEMPLAIRE_SECTION));
            pmbExemplary.setLocationDescription(resultSet2.getString(EXEMPLAIRES_LOCATION_DESCRIPTION));
            pmbExemplary.setLendable(resultSet2.getBoolean(EXEMPLAIRES_LEND));
            pmbExemplary.setLend_finish(resultSet2.getDate(EXEMPLAIRES_LEND_FINISH));
            pmbExemplaries.add(pmbExemplary);
```

figura 167. Recorrido de los datos obtenidos de la tabla de ejemplares.

La presente clase extiende a “LinearLayout” porque será la que relacionaremos con el “layout” de detalle para mapear los resultados obtenidos del servidor PMB. En la figura 169, podemos observar su inclusión dentro del fichero “pmb_server_response.xml” que compone diferentes elementos a mostrar al usuario en la vista de detalle de la aplicación.

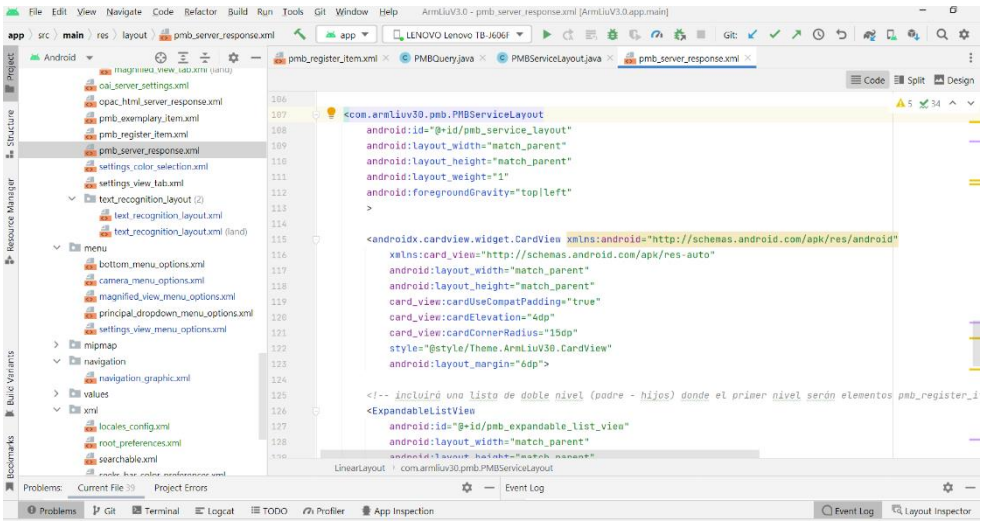


figura 169. Parte del layout correspondiente a la vista de detalle.

En la figura 170, podemos observar el resultado en la parte inferior izquierda. Este está relacionado con el “ExpandableListView” presente en la figura 169, que representa los datos obtenidos a través de la clase “PMBServiceLayout” basándonos en la detección de un ejemplar bibliográfico y consulta por título del mismo.

figura 170. Ejemplo de obtención de resultados del servidor PMB.

Android Studio nos ofrece una herramienta muy potente para separar la vista de usuario del modelo de negocio del que nos abstrae dicha vista. En la medida de lo posible se han implementado las interfaces de usuario empleando ficheros .XML conectados a la lógica de la aplicación. En los casos necesarios se han realizado dos versiones del mismo fichero. Por un lado, la asociada a la orientación horizontal y por otro a la vertical.

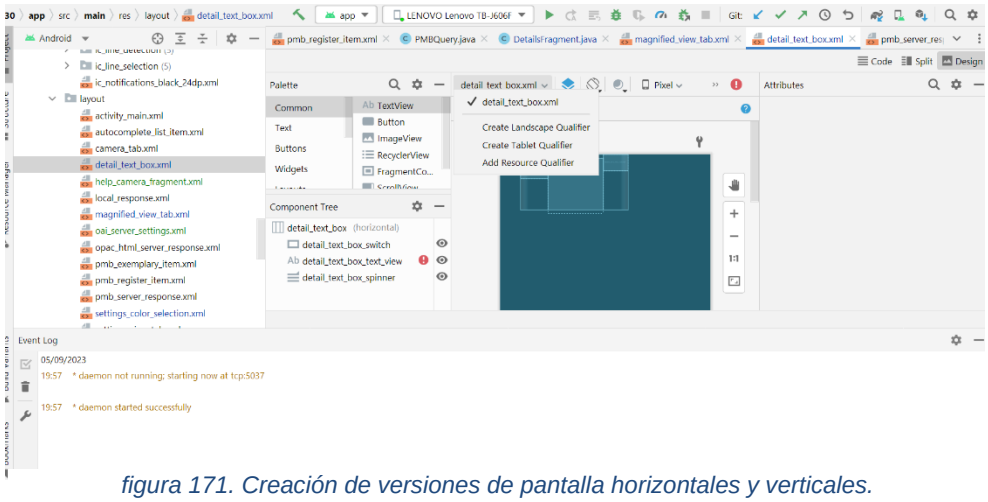


figura 171. Creación de versiones de pantalla horizontales y verticales.

Un fichero de “layout” define la disposición de los diferentes elementos (cuadros de texto, botones, etiquetas, barras de desplazamiento, barras de herramientas, etc.). Para trabajar con Layouts, en Android disponemos de la vista de diseño que nos permite trabajar con los diferentes elementos de forma gráfica (arrastrando/soltando), la vista de código (formato texto basado en secciones XML) o un híbrido (vemos el código y la vista de diseño al mismo tiempo). Desde nuestro punto de vista la vista de código basada en código XML es muy potente. En cualquier caso, si seleccionamos un fichero de “layout” en el árbol del proyecto y seleccionamos la vista de diseño, al desplegar mediante la flecha situado al lado derecho del título del fichero, tenemos varias posibles acciones a realizar (figura 171). “Create Landscape Qualifier” para crear la versión horizontal, “Create Tablet Qualifier” para crear por ejemplo una versión de la aplicación para tablets, o “Add Resource Quelifier” donde podríamos elegir entre diferentes cualificadores (idioma, tamaño del dispositivo, densidad de pixel, etc.) y crear layouts específicos para cada uno de ellos. En caso de

crear un solo fichero de Layout sin Cualificar, este será empleado para todos los casos (versión horizontal, vertical, todos los tamaños de pantalla, etc.).

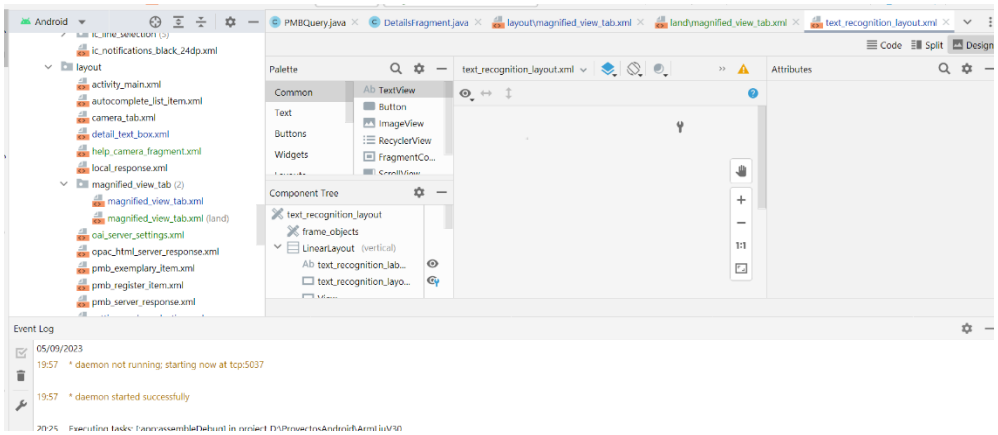
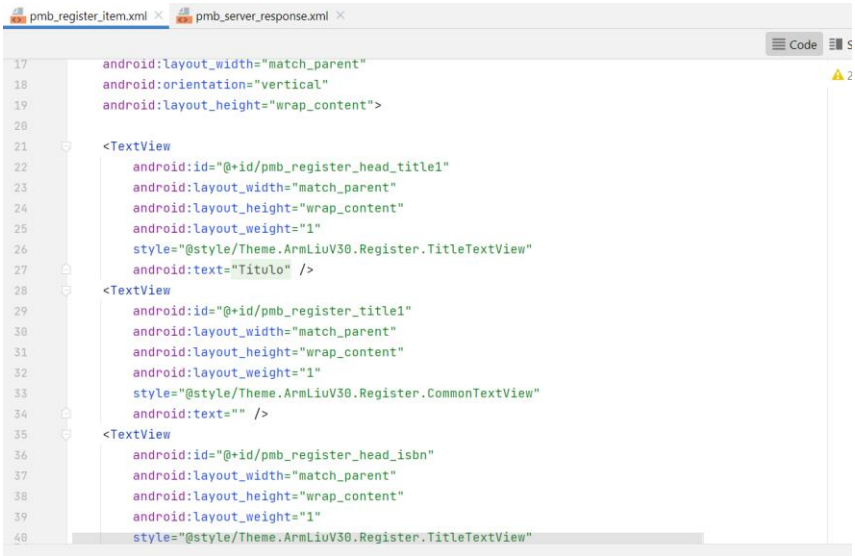


figura 172. Detalle de dos versiones de layout.

En la figura 172, podemos observar como el fichero “magnificated_view_tab.xml” dispone de dos versiones, la asociada a la posición horizontal del dispositivo y la asociada a la posición vertical.

Hemos creado las clases PMBRegister y PMBExemplary con vistas a utilizar ArrayList y adaptadores que cargarán automáticamente los datos obtenidos en la interfaz de usuario empleando unos Layouts en formato XML. Esta forma de mostrar los datos al usuario nos permite separar el modelo de la vista asociada a la interfaz de usuario.



```
17      android:layout_width="match_parent"
18      android:orientation="vertical"
19      android:layout_height="wrap_content">
20
21      <TextView
22          android:id="@+id/pmb_register_head_title1"
23          android:layout_width="match_parent"
24          android:layout_height="wrap_content"
25          android:layout_weight="1"
26          style="@style/Theme.ArmLiuV30.Register.TitleTextView"
27          android:text="Titulo" />
28      <TextView
29          android:id="@+id/pmb_register_title1"
30          android:layout_width="match_parent"
31          android:layout_height="wrap_content"
32          android:layout_weight="1"
33          style="@style/Theme.ArmLiuV30.Register.CommonTextView"
34          android:text="" />
35      <TextView
36          android:id="@+id/pmb_register_head_isbn"
37          android:layout_width="match_parent"
38          android:layout_height="wrap_content"
39          android:layout_weight="1"
40          style="@style/Theme.ArmLiuV30.Register.TitleTextView"
```

figura 173. pmb_register_item.xml

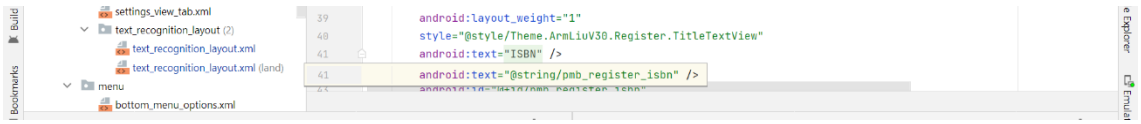


figura 174. Detalle del referenciado de textos al fichero string dentro de un Layout.

13.2.1.7 CameraFragment.

Esta clase extiende a Fragment e incluye una gran cantidad de la lógica de negocio necesaria para que funcione la aplicación. Es la base de la vista previa de la misma, en la que se va analizando en tiempo real la imagen captada por la cámara del dispositivo y detectando los diferentes textos presentes en ella. Como ya se ha comentado, la vista previa se apoya en determinadas clases que nos simplifican tanto el uso de la cámara, como la detección de texto y por tanto como en cualquier otro contexto hay que importarlos.

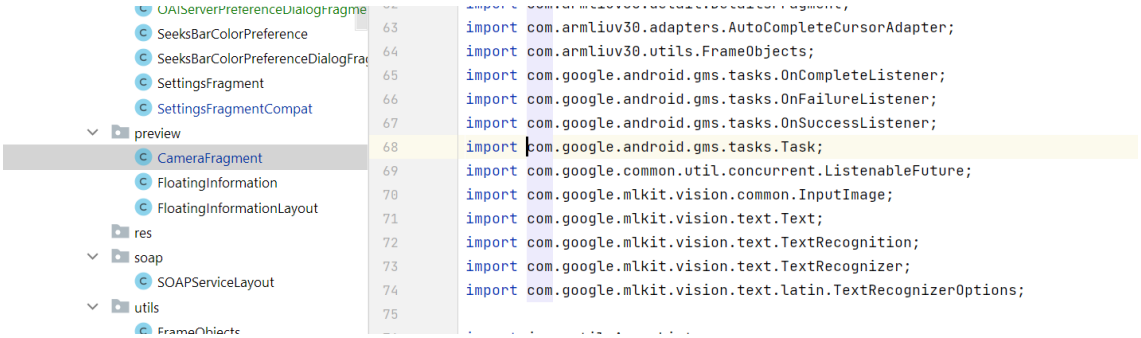


figura 175. Detalle de importación de algunas clases pertenecientes a MLKit.

Algunas de las clases más relevantes empleadas se corresponden con las de la figura 176, en la que podemos observar la clase “PreviewView” que emplearemos para crear un caso de uso de “MLKit” de vista previa, la clase “FrameObjects” que se corresponderá con una capa transparente que se posicionará encima de la imagen obtenida por la cámara para ir enmarcando (grosor del trazo y color configurables) en tiempo real los textos detectados, la clase “FloatingInformationLayout” que se corresponderá con otra capa transparente que representará los textos detectados siendo configurable su tamaño y color y TextRecognizer que es la clase para reconocimiento de texto proporcionada por MLKit. Se ha comentado que la detección de textos se

```
//caso de uso vista previa de la cámara.
private PreviewView previewView;

//layout que muestra la máscara correspondiente a cada texto detectado.
private FrameObjects maskObjects;

//layout que representa los textos detectados en caso de que se haya configurado que se desean mostrar en
private FloatingInformationLayout floatingInformationLayout;


private FrameLayout cameraTabFrameLayout;

//reconocedor de texto.
TextRecognizer recognizer;


//representa el resultado del cómputo asíncrono. Permite registrar las devoluciones del proveedor de la cá
private ListenableFuture<ProcessCameraProvider> cameraProviderFuture;

//imagen capturada de la vista previa y que pasaremos a la vista de detalle (DetailsFragment).
private ImageCapture imageCapture;
```

figura 176. Instancia de algunas clases dentro de la vista previa.

realiza en tiempo real y este aspecto nos dio bastantes problemas, dado que en los desarrollos iniciales la latencia era demasiado alta y el enmarcado y muestras de textos se realizaba a “saltos”. Sin embargo, adaptando el código y optimizándolo tal cual está en estos momentos estos tiempos de latencia son medianamente aceptables.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //vamos a anular acciones de la barra de herramientas de la actividad para añadir nuevas respuestas...
    //informamos a la aplicación de que deseamos participar en la propagación del menú opciones.
    setHasOptionsMenu(false);
    //instanciamos la base de datos de la que ofreceremos las sugerencias.
    suggestionsDB=new DublinCoreDB(getContext());


}
```

figura 177. Método onCreate de la clase CameraFragment.

En el método “onCreate” de la presente clase instanciamos la clase “DublinCoreDB” porque posteriormente obtendremos información para cargar en el cuadro de texto de filtraje para ofrecer sugerencias de filtraje al usuario de las cargadas desde un servidor OAI en la base de datos local. En el método “onCreateView” inflamos la vista que está asociada con el fichero “camera_tab.xml” (figura 179).

figura 179. Camera_tab.xml

Dentro de este fichero hemos empleado un “FrameLayout”, dado que este a diferencia de un “LinearLayout” nos permite superponer unos objetos con otros, y justo este es el efecto que deseamos (la superposición de las capas correspondientes al enmascarado y muestra de los textos detectados). En la figura 179, podemos observar como uno de los primeros elementos que nos encontramos es la barra de herramientas de la aplicación para ofrecer las diferentes opciones al usuario.

figura 178. Contenido camera_tab.xml

En la figura 178, podemos observar la inclusión de los elementos que serán superpuestos, el elemento de vista previa de la cámara que nos proporciona el paquete oficial “androidx” y las clases de elaboración propia “FrameObjects” y “FloatingInformationLayout”.

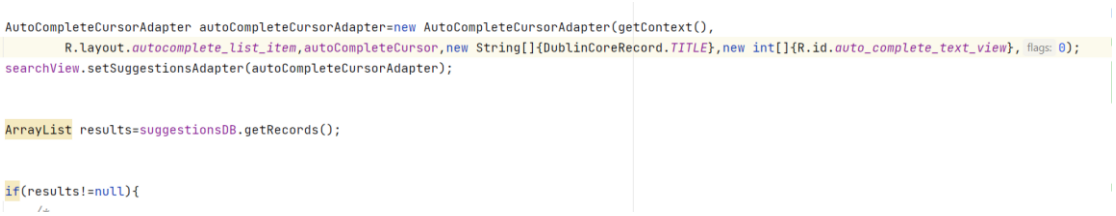
En el método “onViewCreated” de la clase (es importante no confundir con el método “onCreateView”, dado que en este algunos objetos todavía no

han sido inicializados y nos daría en error en tiempo de ejecución al intentar acceder a ellos), nos encontramos con aspectos importantes como la creación, instanciación y configuración del campo de filtraje por texto de los elementos de la imagen a detectar. Para ello empleamos entre otras, la clase “SearchManager” y “Pattern” (figura 180).

```
SearchManager searchManager = (SearchManager) this.getActivity().getSystemService(Context.SEARCH_SERVICE);
searchView = (SearchView) camera_tab_toolbar.findViewById(R.id.search_option);

searchView.setSearchableInfo(searchManager.getSearchableInfo(this.getActivity().getComponentName()));
searchView.setIconifiedByDefault(false); // Do not iconify the widget; expand it by default
searchView.setSubmitButtonEnabled(true); //para que muestre un botón de enviar. Por defecto no lo muestra
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    //este método se ejecuta cuando se envía el patrón introducido en el campo de búsqueda (pulsando ente
    @Override
    public boolean onQueryTextSubmit(String query) {

        return false;
    }

    //se ha optado por realizar el filtraje de elementos actualizando este  cada vez que el usuario intr
    //usamos este porque deseamos que se filtre la búsqueda sin necesidad de darle al botón de validar e
    @Override
    public boolean onQueryTextChange(String newText) {
        if(newText.isEmpty()){
            patternFilter=null;
        }else {
            patternFilter = Pattern.compile(newText, Pattern.CASE_INSENSITIVE);
        }
        //Mostramos sugerencias con la metainformación cargada del servidor OAI empleando el formato Dubli
        //dar la opción en las propiedades de configurar campo o campos por los que queremos autocompletar
```

figura 180. Método onViewCreate. Campo de filtraje.

Hemos optado por tener en cuenta cualquier variación del texto dentro del cuadro de filtraje, en lugar de esperar a que introduzcan el patrón completo y le den a enviar. Por eso hemos utilizado el método “onQueryTextChang” del “listener” asociado, retornando falso en el “onQueryTextSubmit” para indicar que este evento no lo estamos gestionando.



```
AutoCompleteCursorAdapter autoCompleteCursorAdapter=new AutoCompleteCursorAdapter(getContext(),
        R.layout.autocomplete_list_item,autoCompleteCursor,new String[]{DublinCoreRecord.TITLE},new int[]{R.id.auto_complete_text_view}, flags: 0);
searchView.setSuggestionsAdapter(autoCompleteCursorAdapter);


ArrayList results=suggestionsDB.getRecords();


if(results!=null){
```

figura 181. Uso del adaptador para ofrecer sugerencias de filtraje.

Para ofrecer las sugerencias nos hemos basado en nuestro propio adaptador. En la figura 181, podemos observar como este hace uso del fichero “autocomplete_list_item.xml” que será empleado para crear el listado de

elementos mapeando el diseño con los datos obtenidos en tiempo de ejecución y que hemos creado nosotros. Este fichero incluye la definición de un campo de texto (TextView) para mostrar las sugerencias. También podemos observar en la figura como estamos asociando el campo título de cada uno de los registros del tipo “DublinCoreRecord” que vayamos obteniendo al interrogar la base de datos local con dicho campo de texto. Con la instrucción

```
searchView.setOnSuggestionListener(this);
```

nos ponemos a la escucha de los cambios producidos en el campo de filtraje.

```
//gestionamos las diferentes acciones presentes en la barra de herramientas.
camera_tab_toolbar.setOnMenuItemClickListener(new Toolbar.OnMenuItemClickListener() {
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        if(item.getItemId()==R.id.navigation_magnified_view_tab){
            CaptureImage(imageCapture);
            return true;
        }
        if(item.getItemId()==R.id.navigation_settings){
            navController.navigate(R.id.navigation_settings);

        }
        if(item.getItemId()==R.id.speech_to_text_option){
            //abrimos el reconocedor de texto que copiará el texto dictado en el cuadro de filtraje d
            TextRecognize();
        }
        if(item.getItemId()==R.id.help_camera_option){
            navController.navigate(R.id.camera_help_navigation);

        }
        //los eventos han sido consumidos y por tanto retornamos true.
        return true;
    }
```

figura 182. Gestión de las opciones de menú dentro de la clase.

En la figura 182, podemos observar como se realiza la gestión de las diferentes opciones de menú de esta pantalla. Destacando la que invoca a “TexRecognize()” cuya función está destinada al reconocimiento de voz y su traducción en formato texto para rellenar el cuadro de filtraje, y la que invoca el método “CaptureImage”, cuya opción de menú se corresponde con la acción de desplazarse a la vista de detalle pasando a esta mediante un objeto “Bundle”, tanto el patrón de filtraje que ha establecido el usuario, como la última imagen que está visualizando el usuario en ese momento y de la que desea realizar un análisis más en detalle obteniendo la información correspondiente de los diferentes servidores de información.

```
private void TextRecognize(){

    Log.i(TAG, msg: "vamos a iniciar la escucha....");
    speechRecognizer.startListening(speechIntent);


}
```

figura 183. Inicio del servicio de transcripción de voz a texto.

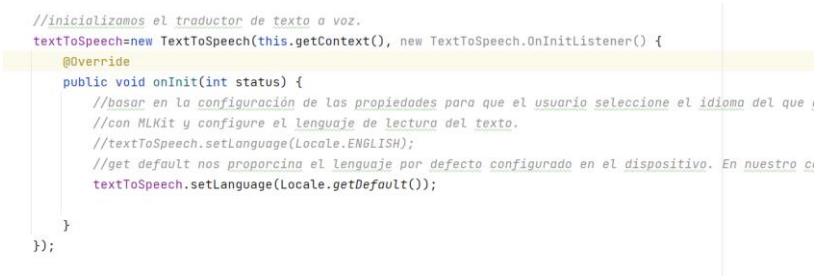
Es importante haber inicializado previamente el reconocedor de texto comprobando si se disponen de los permisos necesarios, así como realizar una serie de configuraciones como la del idioma a emplear.

```
//comprobamos si tenemos el permiso de acceso al micrófono y en caso contrario lo solicitamos.


if(ContextCompat.checkSelfPermission(getContext(),Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED){
    if(ActivityCompat.shouldShowRequestPermissionRationale(getActivity(),Manifest.permission.RECORD_AUDIO)){

    }else{
        //el usuario no necesita explicación y continuamos con la solicitud del permiso.
        ActivityCompat.requestPermissions(getActivity(),
                new String[]{Manifest.permission.RECORD_AUDIO},
                requestCode: 1);
    }
```

figura 184. Comprobación y solicitud de permisos de acceso al micrófono y grabación de audio.

```
//creamos el reconocedor de voz.
speechRecognizer=SpeechRecognizer.createSpeechRecognizer(getActivity());

//creamos el intent.
speechIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);


speechIntent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,getClass().getPackage().getName());

//indicamos el modelo de reconocedor de lenguaje a emplear.
speechIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

speechIntent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, value: 100);

//indicamos el lenguaje a emplear. En este caso el de por defecto que en nuestro caso probablemente será el castellano.
speechIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());

speechIntent.putExtra( RecognizerIntent.EXTRA_PROMPT, value: "Indica el patrón de búsqueda....." );
```

figura 185. Creación y configuración del reconocedor de texto.



```
//inicializamos el traductor de texto a voz.
textToSpeech=new TextToSpeech(this.getContext(), new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        //basar en la configuración de las propiedades para que el usuario seleccione el idioma del que
        //con MLKit y configure el lenguaje de lectura del texto.
        //textToSpeech.setLanguage(Locale.ENGLISH);
        //get default nos proporcina el lenguaje por defecto configurado en el dispositivo. En nuestro c
        textToSpeech.setLanguage(Locale.getDefault());

    }
});
```

figura 186. Elección de idioma para el reconocedor de voz en el método onViewCreated

```
void CaptureImage(ImageCapture imageCapture){

    imageCapture.takePicture(ContextCompat.getMainExecutor(this.getActivity().getApplicationContext()), new Im

        @Override
        public void onCaptureSuccess(@NonNull ImageProxy img) {
            bimage=imageToBitmap(img);
            img.close();

            //pasamos una serie de parámetros al fragment la vista de detalle (DetailsFrament).
            capturedImageBundle.putParcelable(DetailsFragment.IMAGE_PARAMETER,bimage);
            //encapsulamos en Bundle que de forma nativa implementa parcelable y soporta una serie de tipos de
            Bundle params=new Bundle();
            if(patternFilter!=null) {
                params.putString(DetailsFragment.FILTER_PARAMETER, patternFilter.pattern());

            }
            capturedImageBundle.putParcelable(DetailsFragment.FILTER_PARAMETER,params);

            Navigation.findNavController(getView()).navigate(R.id.magnified_view_action,capturedImageBundle);
```

figura 187. Método ImageCapture.

El método “ImageCapture” (figura 187), incluye tanto la captura de la imagen invocando a “takePicture”, como su empaquetado mediante la clase “Bundle” que implementa “parcelable” de forma nativa y nos permite su serialización para enviárselo a la pantalla de detalle que espera su recepción una vez invocada. Denotar que para los casos de uso de reconocimiento de texto no funcionaba cualquier tipo de formato de imagen, y es por este motivo por el que esta está en formato Bitmap realizando su conversión mediante el método “imageToBitmap”.

```
/**
 * Convierte la imagen pasada como parámetro en un Bitmap.
 * @param image @see ImageProxy
 * @return
 */
Bitmap imageToBitmap(ImageProxy image){
    try{
        @SuppressLint("RestrictedApi") byte[] data= ImageUtil.jpegImageToJpegByteArray(image);
        return BitmapFactory.decodeByteArray(data, offset: 0,data.length);
    }catch (Exception e){
        return null;
    }
}
```

figura 188. Método de conversión a bitmap de una imagen.

Dentro de esta clase nos encontramos con el método “bindPreview” (figura 189) al que le pasamos como parámetro el proveedor de cámara que emplearemos y que entre otras acciones crea una vista previa de esta mediante la clase “Preview”.

```
/**
 * Enlazamos la vista previa de la cámara registrando los diferentes casos de uso empleados para realizar la deti
 *
 * @param cameraProvider proveedor de la cámara.
 */
void bindPreview(@NonNull ProcessCameraProvider cameraProvider) {

    //construimos la vista previa.
    Preview preview = new Preview.Builder().build();


    //seleccionamos la cámara trasera (para nuestro caso la principal del dispositivo, dado que el usuario desea
    CameraSelector cameraSelector = new CameraSelector.Builder()
            .requireLensFacing(CameraSelector.LENS_FACING_BACK)
            .build();

    //enlazamos la vista previa con el layout sobre el que será representada.
    preview.setSurfaceProvider(previewView.getSurfaceProvider());

    //creamos el caso de uso asociado a la captura de imagen. Priorizamos menor tiempo de latencia en la captura
    imageCapture=(new ImageCapture.Builder()).setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY).build()
```

figura 189. Método de enlace de la previsualización.

Podemos observar en la figura 189, como seleccionamos la cámara principal del dispositivo mediante el uso del atributo LENS_FACING_BACK. Denotar que se están empleando expresiones Lambda para simplificar el código. Este es el caso de la instrucción “Preview.Builder().build;” que condensa una serie de líneas de código instanciando a clases y a métodos en una sola. A partir de este punto, lo que hacemos dentro de la función es crear una serie de casos de uso del kit de aprendizaje automático. En concreto ya hemos creado el caso de uso de vista previa que deberemos enlazar posteriormente con el proveedor de cámara. En la figura 190, podemos

consultar las instrucción de creación de los casos de uso de captura de imagen y el de análisis de imagen. Con la propiedad “CAPTURE_MODE_MINIMIZE_LATENCY” configuramos una mínima latencia, y este aspecto es importante. Es importante observar también el formato de imagen elegido para realizar el análisis dado que este aspecto nos dio bastantes problemas cuando empleábamos otros formatos (teóricamente soportados), así como la estrategia de mantener solo la última imagen (STRATEGY_KEEP_ONLY_LATEST) que llega y da tiempo de atender, descartando el resto, en lugar de ir creando una cola de imágenes que se van atendiendo de forma secuencial pero que genera retrasos considerables en el enmarcado de los elementos detectados.

```
//creamos el caso de uso asociado a la captura de imagen. Priorizamos menor tiempo de latencia en la captura frente a mayor calidad de l
imageCapture=(new ImageCapture.Builder()).setCaptureMode(ImageCapture.CAPTURE_MODE_MINIMIZE_LATENCY).build();

//configuramos el analizador.
ImageAnalysis imageAnalysis =
        new ImageAnalysis.Builder()
                .setOutputImageFormat(ImageAnalysis.OUTPUT_IMAGE_FORMAT_YUV_420_888)
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .setOutputImageRotationEnabled(true)
                //.setImageQueueDepth(10)
                .build();
CaptureConfig captureConfig;

/**
 * ImageAnalysis adquiere imágenes de la cámara a través del ImageReader. Cada imagen se proporciona a una función ImageAnalysis.Analyze
 * implementarse mediante el código de la aplicación, dónde puede acceder a los datos de la imagen para el análisis a través de un Image
 */

imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor(this.getContext()), new ImageAnalysis.Analyzer() {
    @Override
```

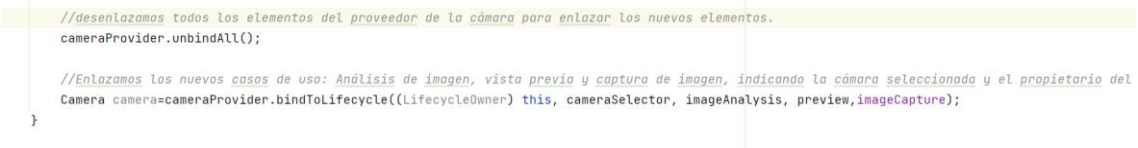
figura 190. Definición de casos de uso: captura de imagen y análisis de imagen.

Uno de los métodos que se irá disparando de forma automática es el método “analyze” que nos proporcionará una imagen como parámetro (figura 191). Podemos observar como está definido como “synchronized” para evitar que existan múltiples hilos ejecutando este mismo método. Este método es de vital importancia, dado que nos permitirá realizar el reconocimiento de texto sobre la imagen y mostrar los datos al usuario. Este reconocimiento lo realizamos mediante el método que hemos creado “runTextRecognition” el cual devuelve una tarea (Task) que nos permite comprobar cuando ha finalizado el reconocimiento de texto sobre una imagen antes de comenzar otro y con vistas a ir eliminando tanto los objetos Imagen que se van creando en tiempo de ejecución para liberar memoria (de no cerrar las imágenes para liberar dicha

imagen, la memoria del dispositivo se nos desbordaba), así como para eliminar (“.clearAnimation()”) el enmascaramiento de textos correspondiente con reconocimientos previos, para enmascarar los nuevos reconocimientos.

```
imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor(this.getContext()), new ImageAnalysis.Analyzer() {
    @Override
    public synchronized void analyze(@NonNull ImageProxy imageProxy) {
        try {

            @SuppressLint("UnsafeOptInUsageError") int width=imageProxy.getImage().getWidth();
            @SuppressLint("UnsafeOptInUsageError") int heigth=imageProxy.getImage().getHeight();
            @SuppressLint("RestrictedApi") byte[] img=ImageUtil.yuv_420_888toNv21(imageProxy);

            int rotationDegrees=imageProxy.getImageInfo().getRotationDegrees();
            //vital recoger la finalización de la tarea a este nivel para poder destruir las imagenes y objetos creados. De lo contrario desborda
            // al realizar el reconocimiento y enmascaramiento
            runTextRecognition(img,width,heigth,rotationDegrees, ImageFormat.NV21).addOnCompleteListener(new OnCompleteListener<Text>() {
                @Override
                public void onComplete(@NonNull Task<Text> task) {
                    imageProxy.close();
                    maskObjects.clearAnimation();
                }
            });


        }catch(Exception e){
            if(e!=null&& e.getMessage()!=null) {
```

figura 191. Método analyze asociado al caso de uso.



```
    //desenlazamos todos los elementos del proveedor de la cámara para enlazar los nuevos elementos.
    cameraProvider.unbindAll();

    //Enlazamos los nuevos casos de uso: Análisis de imagen, vista previa y captura de imagen, indicando la cámara seleccionada y el propietario del
    Camera camera=cameraProvider.bindToLifecycle((LifecycleOwner) this, cameraSelector, imageAnalysis, preview,imageCapture);
}
```

figura 192. Enlace de los diferentes casos de uso con el proveedor de cámara.

En la figura 192, podemos observar el enlace de los diferentes casos de uso con el proveedor de cámara.

Consideramos relevante describir un mínimo el método “runTextRecognition” (figura 193). A este método le proporcionamos la imagen sobre la que deseamos realizar el reconocimiento, sobre la que invocamos al método “process” de la instancia de “recognitzer”. Este método procesará el reconocimiento sobre dicha imagen y en caso de que este tenga éxito llamará al método “onSucces” proporcionándonos los “Text” detectados y permitiéndonos recuperar los bloques de texto detectados. Si el usuario ha introducido un “patternFilter”, filtraremos el conjunto de bloques de texto obtenidos incluyendo solo aquellos que cumplan dicho patrón y por tanto solo sobre estos se realizará el enmascaramiento del elemento.

```
vate synchronized Task<Text> runTextRecognition(InputImage image, int rotationDegrees){

 //procesamos el reconocimiento de textos de la imagen.
 return recognizer.process(image)
         .addOnSuccessListener(
                 new OnSuccessListener<Text>() {
                     @Override
                     public void onSuccess(Text texts) {
                         if(maskObjects!=null){
                             List<Text.TextBlock> txtBlocks;
                             //recuperamos los bloques de texto detectados.
                             txtBlocks=texts.getTextBlocks();
                             if(patternFilter!=null) {
                                 //Limitamos la lista de text blocks a enmascarar según el filtro establecido por el usuario en el cámpo de búsqueda.
                                 txtBlocks = txtBlocks.stream().filter(line -> patternFilter.matcher(line.getText()).find()).collect(Collectors.toList());

                             }
```

figura 193. Contenido del método runTextRecognition.

Una vez filtrados los bloques que cumplen el patrón, forzaremos el redibujado de aquellos contornos sobre textos y muestra de textos que correspondan a la detección actual, eliminando los dibujados previamente. Para ello se invoca al método que hemos creado “redrawDetection” al que le pasamos como parámetros los bloques de texto detectados y el ancho y alto de la imagen sobre la que se detectó con vistas a poder escalar de forma apropiada el tamaño de la previsualización, el de la imagen obtenida y el de las capas sobre las que se mostrará la información de enmascaramiento y muestra de textos.

```
            //redibujamos máscaras de texto e información flotante.
        redrawDetection(txtBlocks,image.getWidth(),image.getHeight());
        }
```

figura 194. Creación de máscaras e información flotante.

El método “redrawDetection” (figura 199) en primer lugar realiza los ajustes de escala para que los marcos y la información flotante se posicionen en el lugar correcto. Para ello, utiliza el ancho y alto de los diferentes elementos, como la relación de aspecto.

```
String detectionType=sharedPreferences.getString( s: "previewview_detection_type_setting", s1: "element");
//  Log.i(TAG,"tipo de detección para la vista previa de la cámara " + detectionType);

//los dos false indican detección por palabra.
boolean blockDetection=false;
boolean lineDetection=false;

if(detectionType.equals("block")){
    blockDetection=true;
}else{
    if(detectionType.equals("line")){
        lineDetection=true;
    }
}
```

figura 195. Tipo de representación deseada.

En segundo lugar, recupera de las preferencias de la aplicación, el tipo de detección que se desea aplicar a la vista previa (párrafo, línea o palabra), para tenerlo en cuenta en el momento de enmascarar y mostrar la información flotante (figura 195).

```
Text.TextBlock block;

while(blocks.hasNext()){
    block=(Text.TextBlock) blocks.next();
    if(block!=null) {
        //detección por bloque.
        if(blockDetection){
            //texto.setText(block.getText());
            maskObjects.addPath(createPath(block.getCornerPoints(),scaleFactor,postScaleWidthOffset,postScaleHeightOffset));
            if(sharedPreferences.getBoolean( s: "show_contextual_information", b: true)){
                floatingInformationLayout.addFloatingInformation(createFloatingInformation(block.getCornerPoints(), scaleFactor, postScaleWidthO
               // textToSpeech.speak(block.getText(),TextToSpeech.QUEUE_FLUSH,null,null);
            }
        }else{
            for(Text.Line line: block.getLines()){
                //detección por linea.
                if(lineDetection){
                    maskObjects.addPath(createPath(line.getCornerPoints(),scaleFactor,postScaleWidthOffset,postScaleHeightOffset));
                    if(sharedPreferences.getBoolean( s: "show_contextual_information", b: true)){
                        floatingInformationLayout.addFloatingInformation(createFloatingInformation(block.getCornerPoints(), scaleFactor, postSca
                //        textToSpeech.speak(block.getText(),TextToSpeech.QUEUE_FLUSH,null,null);
                    }

                }else {
```

figura 196. Recorrido de bloques para enmascarar y mostrar información flotante.

Y en tercer lugar, en función del tipo de detección elegida, recorre los textos detectados estableciendo subniveles de iteración en función de si son bloques (párrafo), líneas (línea) o elementos (palabra).

Podemos observar en la figura 196, que se emplea la propiedad compartida “show_contextual_information” para decidir si se muestra información flotante o no. También que se emplea la clase “Path”, empleando

nuestra función de creación de los mismos “createPath”. Esta fue una de las estrategias que derivó en unos mejores tiempos de respuestas de nuestra aplicación, dado que inicialmente se añadía el enmascaramiento de elementos conforme se recorría y esto ralentizaba demasiado la pantalla, por lo que se decidió crear un conjunto de rutas (paths), para posteriormente dibujarlas todas de golpe para cada imagen analizada.

```
                    }
                    //enmascaramos todos los textos detectados.
                    maskObjects.drawPaths();
                    //mostramos los textos con información flotante.
                    floatingInformationLayout.drawFloatingInformations();
                }
            }
        }
    }
}
```

figura 197. Dibujo de máscaras e información flotante en lote.

```
/**
 * Crea la información textual que se mostrará al usuario de forma flotante.
 * @param points Puntos necesarios para calcular la posición del texto a mostrar.
 * @param scaleFactor Factor de escala en función del tamaño de la imagen sobre la que se ha detectado el texto y el layout donde se muestra dicha informa
 * @param postScaleWidthOffset Offset horizontal.
 * @param postScaleHeightOffset Offset vertical.
 * @param information Texto a mostrar.
 * @return Objeto con la información a mostrar y la posición dónde se debe mostrar.
 */
private FloatingInformation createFloatingInformation(Point[] points, float scaleFactor,float postScaleWidthOffset,float postScaleHeightOffset, String
    if(points==null){return null;}


    float point0_x=(points[0].x * scaleFactor)-postScaleWidthOffset;
    float point0_y =(points[0].y * scaleFactor)-postScaleHeightOffset;

    Point point=new Point();
    FloatingInformation floatingInformation=new FloatingInformation(point0_x,point0_y, information);
    return floatingInformation;

}
```

figura 198. Creación de información flotante asociada al texto detectado.

```
private Path createPath(Point[] points, float scaleFactor,float postScaleWidthOffset,float postScaleHeightOffset){

    if(points==null){return null;}
    float point0_x=(points[0].x * scaleFactor)-postScaleWidthOffset;
    float point0_y =(points[0].y * scaleFactor)-postScaleHeightOffset;
    float point1_x=(points[1].x * scaleFactor)-postScaleWidthOffset;
    float point1_y =(points[1].y * scaleFactor)-postScaleHeightOffset;
    float point2_x=(points[2].x * scaleFactor)-postScaleWidthOffset;
    float point2_y =(points[2].y * scaleFactor)-postScaleHeightOffset;
    float point3_x=(points[3].x * scaleFactor)-postScaleWidthOffset;
    float point3_y =(points[3].y * scaleFactor)-postScaleHeightOffset;

    Path path=new Path();
    path.moveTo(point0_x,point0_y);
    path.lineTo(point1_x,point1_y);
    path.lineTo(point2_x,point2_y);
    path.lineTo(point3_x,point3_y);
    path.lineTo(point0_x,point0_y);
    return path;
}
```

figura 200. Creación de rutas para dibujar la máscara de cada texto detectado.

```
public void redrawDetection(List<Text.TextBlock> textBlocks,int detectionImageWidth,int detectionImageHeight){
    if(textBlocks==null){return;}
    maskObjects.clearPaths();
    floatingInformationLayout.clearFloatingInformations();

    /* calculamos los factores de escala para ajustar la máscara a la imagen capturada en la vista previa del análisis. */

    float viewAspectRatio = (float) cameraTabFrameLayout.getWidth() / cameraTabFrameLayout.getHeight();
    float imageAspectRatio = (float) detectionImageWidth / detectionImageHeight;
    float postScaleWidthOffset = 0;
    float postScaleHeightOffset = 0;
    float scaleFactor=1;
    if (viewAspectRatio > imageAspectRatio) {
        //Debemos ajustar la imagen verticalmente para que se muestre en la vista.
        scaleFactor = (float) cameraTabFrameLayout.getWidth() / detectionImageWidth;
        postScaleHeightOffset = ((float) cameraTabFrameLayout.getWidth() / imageAspectRatio - cameraTabFrameLayout.getHeight()) / 2;
    } else {
        // Debemos ajustar la imagen horizontalmente.
        scaleFactor = (float) cameraTabFrameLayout.getHeight() / detectionImageHeight;
        postScaleWidthOffset = ((float) cameraTabFrameLayout.getHeight() * imageAspectRatio - cameraTabFrameLayout.getWidth()) / 2;
    }
    String detectionType=sharedPreferences.getString( s: "previewview_detection_type_setting", s1: "element");
```

figura 199. Contenido del método redrawDetection.

13.2.1.8 FloatingInformation

Esta clase encapsula la posición x e y de cada texto flotante a mostrar sobre la pantalla. Pese a que se ha añadido un campo de texto como información, este texto podría evolucionar hacia otro tipo de información más compleja y este es el motivo por el que hemos creado una clase específica.

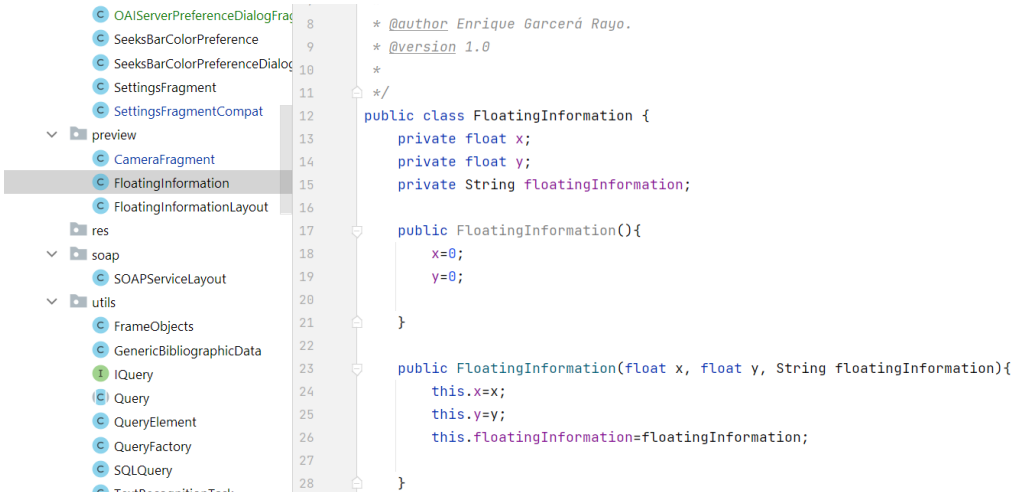


figura 201. Clase FloatingInformation.

13.2.1.9 FloatingInformationLayout

Esta clase extiende de View y representará cada uno de los objetos “FloatingInformation”, mediante el método onDraw(Canvas canvas), recuperando de las preferencias compartidas los valores RGB asociados al color con el que queremos que aparezca el texto y el tamaño del texto a mostrar.

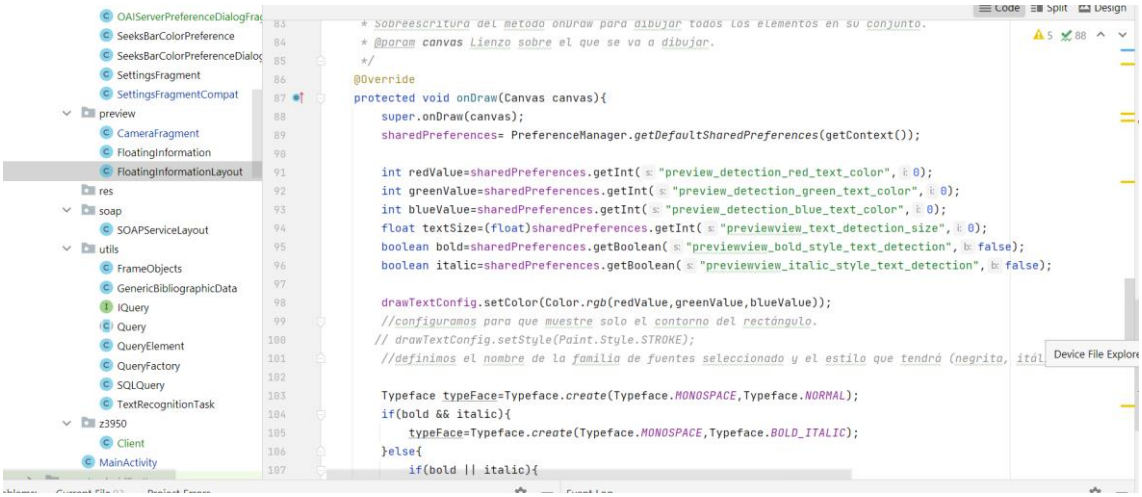


figura 202. Clase FloatingInformationLayout.

```
    Iterator<FloatingInformation> iter=this.floatingInformations.iterator();
    while(iter.hasNext()){
        FloatingInformation floatingInformation=iter.next();
        if(floatingInformation!=null) {
            canvas.drawText(floatingInformation.getFloatingInformation(), floatingInformation.getX(), floatingInfo

        }
    }

}
```

figura 203. Método onDraw de FloatingInformationLayout. Dibujo de textos sobre el lienzo.

13.2.1.10 DetailsFragment

La clase “DetailsFrament” está asociada a la vista de detalle de nuestra aplicación y al igual que la clase “CameraFragment” contiene bastante lógica de negocio pese a que esta ha sido balanceada en otras clases que incluye. En cuanto a la detección de textos del detalle, se omitirán aquellos aspectos ya descritos con anterioridad en otras clases. Esta clase se asocia con el fichero “magnificated_view_tab” empleado para explicar con anterioridad el uso de diferentes cualificadores para adaptar la aplicación a determinadas características de un dispositivo determinado (formato horizontal y vertical por ejemplo). Y el modo de inflar dicha vista en el método “onCreate” es tan simple como referenciar la instancia binding que Android crea por defecto empleando el nombre del archivo XML, pero quitando los guiones y comenzando cada palabra por mayúsculas. Es decir “MagnificatedViewTabBinding” al llamar al método “inflate”, automáticamente se instanciarán todos aquellos objetos presentes en la vista.

```
@CallSuper
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    //almacenamos en variables de la clase para su posterior uso.
    //recuperamos el layout para mostrar la imagen tomada de la vista en tiempo real para analizarl
    imageView=binding.showResults;
    showHideButton=binding.showHideButton;
    showHidePmb=binding.showHidePmb;
    showHideLocal=binding.showHideLocal;
    buttonsLayout=binding.buttonsLayout;
    opachtmlWebView =binding.htmlResponse.opacHtmlWebView;


    try {

        //recuperamos la imagen de la que deseamos reconocer su texto.
        Bitmap bimage = getArguments().getParcelable(IMAGE_PARAMETER);
        if (bimage != null) {
            loadImage=bimage;
            imageView.setImageBitmap(bimage);

        }
```

figura 204. Asignación a variables locales de los objetos contenidos por el binding de la vista de detalle.

```
/**
 * Método que se ejecuta al crear la vista.
 *
 * @param inflater
 * @param container
 * @param savedInstanceState
 * @return
 */
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container, Bundle savedInstanceState) {


    //inflamos la vista y almacenamos el objeto inflado un un binding. Esto nos evitará emplear tanto el méto
    binding = MagnifiedViewTabBinding.inflate(inflater, container, attachToParent: false);
    View root = binding.getRoot();
```

figura 205. Inflado del binding de la vista de detalle.

Después de inflar la vista, podemos hacer uso de los diferentes objetos que la componen recuperando del “binding” estos a partir de que la vista ya ha sido cargada (figura 204). Es por este motivo por lo que la asignación la realizamos en el método “onViewCreated” en lugar de en el “onCreateView” por ejemplo. En esta figura, podemos observar también la forma de recuperar la imagen que nos han pasado como parámetro (en un principio desde la vista previa de la aplicación).

Para entender mejor el funcionamiento de la presente clase, comentaremos brevemente el contenido del fichero “magnified_view_tab.xml”.

figura 206. Fichero xml, magnified_view_tab.xml.

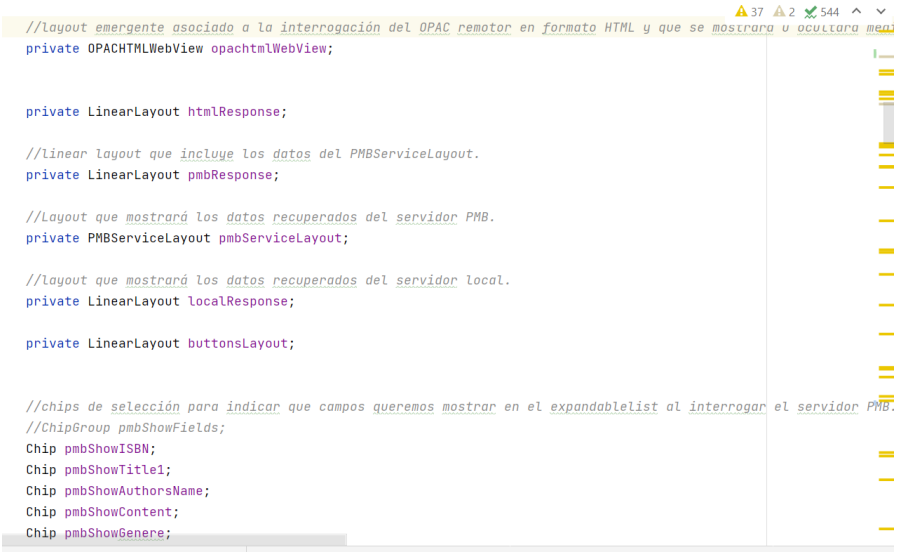
En primer lugar nos encontramos con la barra de herramientas, para luego insertar una <ImageView> que contendrá la imagen pasada como parámetro y que será cargada por “DetailsFragment”. Luego insertamos el layout “text_recognition_layout” que contendrá todos los textos detectados al modo que lo hacíamos en la vista previa, pero en esta caso de forma mucho más elaborada, dado que este layout generará los cuadros de textos con los posibles elementos de selección y desplegable asociado al campo de búsqueda. Este layout será descrito en el correspondiente apartado. El siguiente elemento que nos encontramos es el identificado como “htmlResponse” cuyo layout se corresponde con “opac_html_server_response” y que se asocia con los resultados a mostrar como elemento emergente a la izquierda de la pantalla de detalle. Estos resultados se corresponden con los obtenidos de un servidor OPAC HTML (HyperText Markup Lanuage). El siguiente elemento es “pmb_response” que se corresponde con el cuadro emergente de la pantalla de detalle que nos muestra los resultados obtenidos del servidor PMB configurado en la aplicación. Local response, está en desuso y se conservó por si en algún momento deseábamos mostrar tres orígenes de información.

Los siguientes elementos se corresponden con el layout (figura 207) que en un principio hemos alineado a la izquierda de la vista de detalla y que contendrá los Botones para ocultar o mostrar los cuadros de resultados. Se optó por dar la opción de mostrar u ocultar dichos cuadros para ganar visibilidad en la imagen de fondo.

```
<LinearLayout
    android:id="@+id/buttons_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="100dp"
    android:orientation="vertical"
    >
<ImageButton
    android:id="@+id/show_hide_button"
    android:layout_width="50dp"
    android:layout_marginBottom="0dp"
    android:layout_marginLeft="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:contentDescription=" "
    android:src="?attr/actionModeCopyDrawable"
/>
<ImageButton
    android:id="@+id/show_hide_pmb"
    android:layout_width="50dp"
    android:layout_marginBottom="0dp"
    android:layout_marginLeft="0dp"
```

figura 207. Magnificated_view_tab.xml. Continuación.

Una vez enumerados los elementos que contiene la vista de detalle, continuamos con la descripción de la clase DetailsFragment.



```
//layout emergente asociado a la interrogación del OPAC remotor en formato HTML y que se mostrara u ocultara mediante
private OPACHTMLWebView opachtmlWebView;


private LinearLayout htmlResponse;

//linear layout que incluye los datos del PMBServiceLayout.
private LinearLayout pmbResponse;

//Layout que mostrará los datos recuperados del servidor PMB.
private PMBServiceLayout pmbServiceLayout;

//layout que mostrará los datos recuperados del servidor local.
private LinearLayout localResponse;

private LinearLayout buttonsLayout;


//chips de selección para indicar que campos queremos mostrar en el expandablelist al interrogar el servidor PMB.
//ChipGroup pmbShowFields;
Chip pmbShowISBN;
Chip pmbShowTitle1;
Chip pmbShowAuthorsName;
Chip pmbShowContent;
Chip pmbShowGenere;
```

figura 208. Elementos presentes en la clase DetailsFrament.

En la figura 208, podemos observar como están presentes los “Chips” de selección de los diferentes campos que queremos que aparezcan en los resultados del servidor PMB. Estos elementos son los botones con forma ovalada. También los diferentes layouts descritos con anterioridad

(“OPACHTMLWebView”, “PMBServiceLayout”,etc.). Denotar que emplearemos “WebView” que, salvando las distancias, se corresponde con un navegador embebido en nuestra aplicación.

```
SharedPreferences sharedPreferences=PreferenceManager.getDefaultSharedPreferences(this.

//mostrarmos u ocultamos información relativa a los diferentes campos. Se habilita o desabiltan dichos campo
//los botones, por si el usuario cambia de opinión y quier mostrar dicha información mientras ejecuta la ap
pmbShowISBN.setChecked(sharedPreferences.getBoolean( s: "detailview_show_isbn_setting", b: true));
pmbShowTitle1.setChecked(sharedPreferences.getBoolean( s: "detailview_show_title_setting", b: true));
pmbShowResume.setChecked(sharedPreferences.getBoolean( s: "detailview_show_summary_setting", b: true));
pmbShowAuthorsName.setChecked(sharedPreferences.getBoolean( s: "detailview_show_authors_setting", b: true));
pmbShowContent.setChecked(sharedPreferences.getBoolean( s: "detailview_show_content_field_setting", b: true));
pmbShowGenere.setChecked(sharedPreferences.getBoolean( s: "detailview_show_gender_setting", b: true));
pmbShowPublishers.setChecked(sharedPreferences.getBoolean( s: "detailview_show_editors_setting", b: true));
pmbShowNPages.setChecked(sharedPreferences.getBoolean( s: "detailview_show_npages_setting", b: true));
pmbShowSize.setChecked(sharedPreferences.getBoolean( s: "detailview_show_size_setting", b: true));

//una vez tenido en cuenta los campos de texto a mostrar según las propiedades configuradas por el usuario,
pmbShowISBN.setOnCheckedChangeListener(this);
pmbShowTitle1.setOnCheckedChangeListener(this);
pmbShowAuthorsName.setOnCheckedChangeListener(this);
pmbShowContent.setOnCheckedChangeListener(this);
pmbShowGenere.setOnCheckedChangeListener(this);
pmbShowNPages.setOnCheckedChangeListener(this);
pmbShowPublishers.setOnCheckedChangeListener(this);
pmbShowResume.setOnCheckedChangeListener(this);
pmbShowSize.setOnCheckedChangeListener(this);
pmbServiceLayout.setVisibilityFields(pmbShowTitle1.isChecked(),pmbShowResume.isChecked(),pmbShowISBN.isCheck
```

figura 209. Campos a mostrar según propiedades configuradas en la aplicación.

En la figura 209, podemos observar como se recuperan las propiedades para activar/desactivar cada uno de los “Chip” asociados a los campos de los datos de cada registro bibliográfico recuperado de PMB en función del estado definido por el usuario, así como el establecimiento de los “listeners” que nos permitirán detectar cualquier pulsación sobre estos con vistas a mostrar u ocultar un determinado campo con independencia de la configuración inicial.

```
    showHideButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (htmlResponse.getVisibility() == View.GONE) {


                pmbResponse.setVisibility(View.GONE);
                localResponse.setVisibility(View.GONE);
                htmlResponse.setVisibility(View.VISIBLE);
                htmlResponse.bringToFront();


            } else {
                htmlResponse.setVisibility(View.GONE);
                textRecognitionLayout.bringToFront();

            }
            buttonsLayout.bringToFront();

        }
    });
}else{
    showHideButton.setVisibility(View.GONE);
```

figura 210. Forzado del orden en la profundidad de los diferentes elementos que componen la pantalla.

Dado que estamos jugando con el posicionamiento de unos elementos delante o detrás de otro para superponer información. Es importante que los que deben situarse detrás, no se sitúen delante (por ejemplo, si la imagen se sitúa delante de la capa de enmascaramiento de elementos, esta última no será visible. En la figura 210, podemos observar el uso de “bringToFront”, para forzar que un elemento se sitúe delante de otro. El comportamiento por defecto de Android consiste en situar delante aquellos elementos que aparecen en último lugar en el fichero de layout XML.

```
// mostramos u ocultamos el tab de información ampliada obtenida desde el servidor local.
//en este mosmento este no es accesible, dado que en los XML de la aplicación ha sido deshabilitado mient
showHideLocal.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(localResponse.getVisibility()==View.GONE){

            htmlResponse.setVisibility(View.GONE);
            pmbResponse.setVisibility(View.GONE);


            localResponse.bringToFront();

            localResponse.setVisibility(View.VISIBLE);
            for(int i=0;i<localResponse.getChildCount();i++) {
                localResponse.getChildAt(i).setVisibility(View.VISIBLE);
                localResponse.bringChildToFront(localResponse.getChildAt(i));

            }
            localResponse.invalidate();

        }else{
            localResponse.setVisibility(View.GONE);
            textRecognitionLayout.bringToFront();
        }
```

figura 211. Forzado del orden de elementos de la pantalla.

La configuración del “WebView” empleado nos dio bastantes problemas. Incluso queda pendiente, eliminar una zona blanca que aparece en la cabecera de la página que cargamos al interrogar al servidor OPAC de la biblioteca de la UNED. Se plantea como vía futura. No obstante, en la figura 212 se muestra la configuración que mejores resultados nos ha dado.

```
htmlResponse.getLayoutParams().width=(int)this.getParentFragment().getView().getWidth()/3;
localResponse.getLayoutParams().width=(int)this.getParentFragment().getView().getWidth()/3;
pmbResponse.getLayoutParams().width=(int)this.getParentFragment().getView().getWidth()/3;

htmlResponse.setVisibility(View.GONE);
localResponse.setVisibility(View.GONE);
pmbResponse.setVisibility(View.GONE);

//configuramos el webview.
opachtmlWebView.getSettings().setJavaScriptEnabled(true);
//true carga la página completamente alejada (para que coja en el marco).
opachtmlWebView.getSettings().setLoadWithOverviewMode(true);
opachtmlWebView.getSettings().setBuiltInZoomControls(true);
opachtmlWebView.getSettings().setGeolocationEnabled(false);
//true hace que la vista web tenga una ventana normal de navegador. False restringe a las medidas
opachtmlWebView.getSettings().setUseWideViewPort(false);
opachtmlWebView.getSettings().setDomStorageEnabled(true);
opachtmlWebView.getSettings().setLoadsImagesAutomatically(true);
opachtmlWebView.getSettings().setDatabaseEnabled(true);
opachtmlWebView.setWebChromeClient(new WebChromeClient());
```

figura 212. Configuración del WebView.

13.2.1.11 DetailTextBox.

Esta clase extiende a “LinearLayout” y es la empleada para embeber el texto detectado (TextView) dentro de un cuadro que incluye dicho texto, un “Switch” para seleccionarlo/deseleccionarlo y un cuadro desplegable (Spinner) que nos permite indicar si la activación del presente elemento buscará resultados haciendo coincidir el texto detectado con el campo de título, el ISBN, los autores o la Editorial, sobre el servidor que estamos interrogando.

figura 213. Composición de la clase DetailTextBox.

Esta clase la hemos asociado con el fichero “detail_text_box.xml”. Uno de los aspectos destacables de esta clase, es que hemos creado nuestro propio “Listener” para que otros objetos se pongan a la escucha de los eventos como el cambio de estado del “Switch”, o el cambio de selección en cuanto al campo de búsqueda elegido. En la figura 215, podemos observar la definición de la interfaz que deberán implementar aquellas clases que se pongan a la escucha de los eventos propagados por esta. El método “OnCheckedChange” notificará el cambio de estado del Switch de selección y el método “OnFieldChange” el cambio de campo de selección. En ambos casos se podrá recuperar como parámetro el objeto sobre el que se ha producido el cambio.

En la presente clase tenemos un método “inicializa” que carga los datos del “Spinner” mediante un adaptador y se pone a la escucha de los eventos generados por el Checkbox, Spinner, etc. para lanzar los eventos que hemos personalizado (figura 214).

```
private void inicializa(){
    listener=null;
    this.setActivated(true);
    //posibles campos a asociar (ISBN, Titulo, etc.) y registro del adaptador que manejará los datos.
    spinnerIDs=getResources().getStringArray(R.array.detail_text_box_entries);
    spinnerValues=getResources().getStringArray(R.array.detail_text_box_values);

    ArrayAdapter<CharSequence> adapter=ArrayAdapter.createFromResource(this.getContext(),R.array.detail_text_box_v
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    detailSpinner.setAdapter(adapter);

    //registramos el listener para escuchar los cambios de selección del campo de búsqueda.
    detailSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

        //se ha seleccionado un nuevo elemento del desplegable que contien los posibles campos de ISBN, Titulo,etc
        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view,
                                   int position, long id) {
            Object item = adapterView.getItemAtPosition(position);
            if (item != null) {
                //disparamos el evento para informar al resto de clases que usan DetailTextBox.
                triggerOnFieldChange();
                //Log.i(TAG,"id seleccionado" + spinnerIDs[position]);
                //Log.i(TAG,"texto seleccionado " + item.toString());
```

figura 214. Uso de los adaptadores necesarios, escucha de los diferentes eventos y disparo de los eventos personalizados.

En la figura 216, nos encontramos con los métodos creados para realizar el disparo de nuestros propios eventos. Como se puede observar, se añade como parámetro la instancia del objeto (this) sobre el que se ha producido el evento y el estado en que se encuentra.

```
 */
public interface DetailTextBoxListener extends EventListener{
    public void OnCheckedChange(DetailTextBox detailTextBox, boolean isChecked);
    public void OnFieldChange(DetailTextBox detailTextBox);
}
private DetailTextBoxListener listener;

/**
 * Constructor de la clase.
 * @param context contexto sobre el que se desea operar.
 */
public DetailTextBox(Context context) {
    super(context);

    View.inflate(context, R.layout.detail_text_box,  root: this);
    this.detailTextView=(TextView) findViewById(R.id.detail_text_box_text_view);
    this.detailSpinner=(Spinner) findViewById(R.id.detail_text_box_spinner);
    detailSwitch=(SwitchMaterial) findViewById(R.id.detail_text_box_switch);
    inicializa();


}
```

figura 215. Listener propio de la clase DetailTextbox.

```
/**
 * Disparo del evento Checked Change.
 */
private void triggerOnCheckedChange(){
    if(listener!=null){
        listener.OnCheckedChange( detailTextBox: this, this.detailSwitch.isChecked());
    }

}


/**
 * Disparo del evento fieldChange.
 */
public void triggerOnFieldChange(){
    if(listener!=null){
        listener.OnFieldChange( detailTextBox: this);
    }
}
```

figura 216. Métodos empleados para disparar nuestros propios eventos.

Las clases que deseen ponerse a la escucha de los eventos deberán invocar los métodos mostrados en la figura 217.

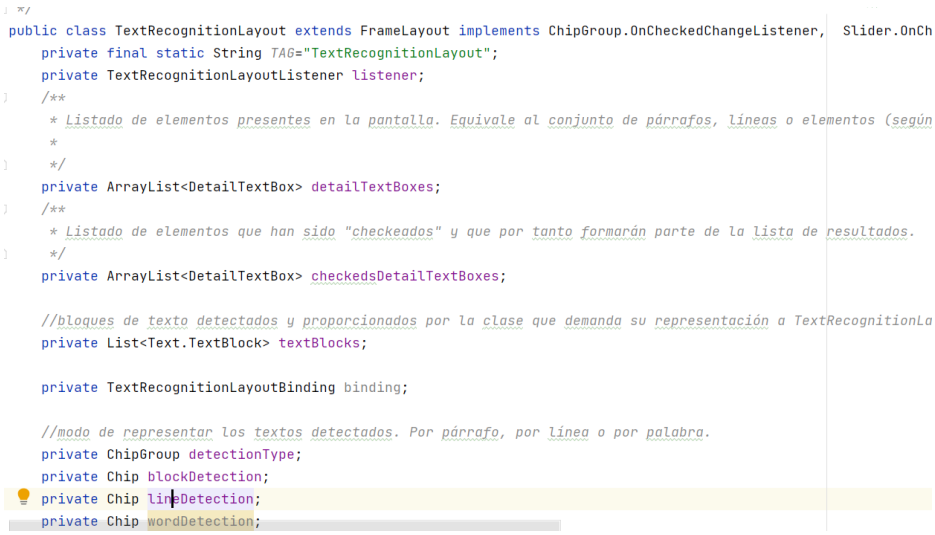
```
/**
 * Permite al resto de clases registrarse como listener.
 * @param detailTextBoxListener
 */
public void addDetailTextBoxEventListener(DetailTextBoxListener detailTextBoxListener){

    this.listener=detailTextBoxListener;

}

/**
 * Elimina el listener que se puso a la escucha previamente.
 * @param detailTextBoxListener Listener que deseamos eliminar.
 */
public void removeDetailTextBoxEvenListener(DetailTextBoxListener detailTextBoxListener){
    listeners.remove(detailTextBoxListener);
}
```

figura 217. Métodos a invocar para ponerse a la escucha de los eventos.

13.2.1.12 TextRecognitionLayout

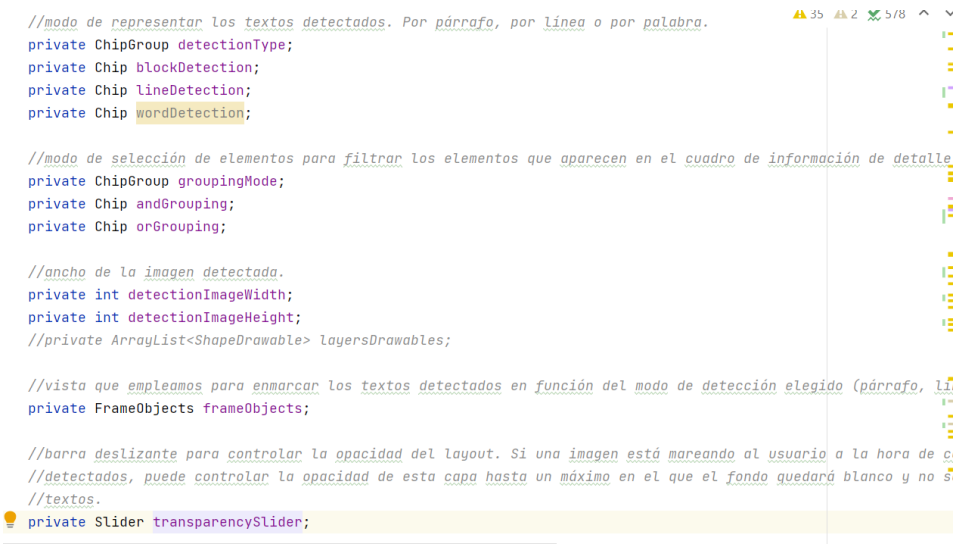
Esta clase extiende de “FrameLayout” e implementa diferentes interfaces referentes a la gestión de eventos. Entre ellas DetailTextoBoxListener. En la presente clase se encuentra gran parte de la lógica de negocio que se ha eliminado de la clase “DetailsFragment”. Como podemos observar en las siguientes figuras, nos encontramos con una lista de “detailTextBoxes” que son todas aquellos presentes en la pantalla y otra lista de los que han sido seleccionados. El contenido de esta lista irá variando de forma dinámica en tiempo de ejecución. En la variable “textBlocks” nos encontramos con los bloques de texto detectados, en la variable “binding”, el enlace a los elementos del presente layout. Tenemos dos grupo de chips de selección, uno para seleccionar el tipo de consulta a realizar sobre los datos (or u and), y otro para seleccionar el tipo de detección que queremos emplear (párrafo, línea o palabra), las dimensiones de la imagen a detectar, un slider que nos permitirá configurar la transparencia de la imagen, para poder ver más claramente los “DetailTexBoxes” creados, las preferencias compartida, para adaptar la funcionalidad a estas, y la variable “textToSpeech” que emplearemos para la conversión de texto a voz. Dado que esta clase realiza acciones ya descrita en

otras, como el reconocimiento de textos, el consumo de los eventos propagados por otras, el inflado del fichero XML, etc., se recomienda encarecidamente la consulta del código fuente para obtener un mayor nivel de detalle de la misma.



```
*/
public class TextRecognitionLayout extends FrameLayout implements ChipGroup.OnCheckedChangeListener,  Slider.OnCh
    private final static String TAG="TextRecognitionLayout";
    private TextRecognitionLayoutListener listener;
    /**
     * Listado de elementos presentes en la pantalla. Equivale al conjunto de párrafos, líneas o elementos (según
     *
     */
    private ArrayList<DetailTextBox> detailTextBoxes;
    /**
     * Listado de elementos que han sido "checkeados" y que por tanto formarán parte de la lista de resultados.
     */
    private ArrayList<DetailTextBox> checkedsDetailTextBoxes;

    //bloques de texto detectados y proporcionados por la clase que demanda su representación a TextRecognitionLa
    private List<Text.TextBlock> textBlocks;

    private TextRecognitionLayoutBinding binding;

    //modo de representar los textos detectados. Por párrafo, por línea o por palabra.
    private ChipGroup detectionType;
    private Chip blockDetection;
    private Chip lineDetection;
    private Chip wordDetection;
```

figura 218. Elementos que contiene la clase TextRecognitionLayout.



```
//modo de representar los textos detectados. Por párrafo, por línea o por palabra.
private ChipGroup detectionType;
private Chip blockDetection;
private Chip lineDetection;
private Chip wordDetection;

//modo de selección de elementos para filtrar los elementos que aparecen en el cuadro de información de detalle
private ChipGroup groupingMode;
private Chip andGrouping;
private Chip orGrouping;

//ancho de la imagen detectada.
private int detectionImageWidth;
private int detectionImageHeight;
//private ArrayList<ShapeDrawable> layersDrawables;

//vista que empleamos para enmarcar los textos detectados en función del modo de detección elegido (párrafo, li
private FrameObjects frameObjects;

//barra deslizante para controlar la opacidad del layout. Si una imagen está mareando al usuario a la hora de c
//detectados, puede controlar la opacidad de esta capa hasta un máximo en el que el fondo quedará blanco y no s
//textos.
private Slider transparencySlider;
```

figura 219. Elementos contenidos en la clase TextRecognitionLayout (cont).

```
private SharedPreferences sharedPreferences;

//lectura de texto a voz.
TextToSpeech textToSpeech;
```

figura 220. Contenido TextRecognitionLayout (cont 2).

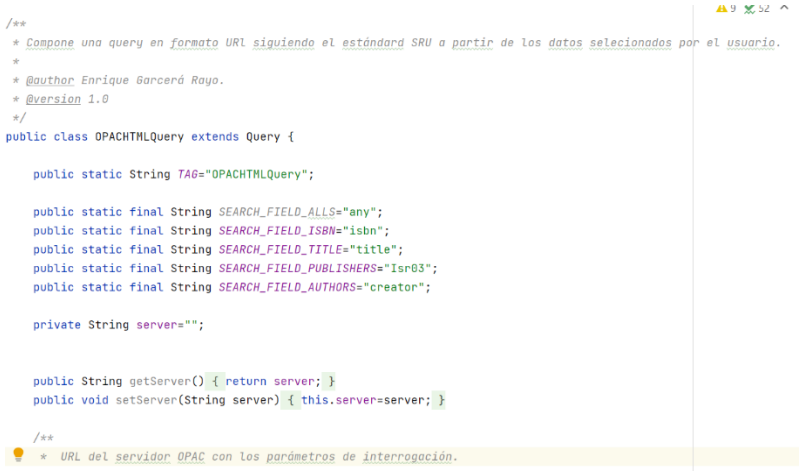
13.2.1.13 OPACHTMLWebView.

Esta clase extiende la clase “WebView”, y sirve para cargar los datos obtenidos del servidor OPAC Html. En la figura 221, se puede observar la parte más relevante de la presente clase, y consiste en la carga de los datos obtenidos de la consulta. Para realizar la consulta, nos apoyamos en la clase que hemos creado “OPACHTMLQuery”.

figura 221. Clase OPACHTMLWebView.

13.2.1.14 OPACHTMLQuery

La presente clase está estrechamente relacionada con el apartado de ingeniería inversa realizada sobre los servidores OPAC Html de la UPV y de la UNED, debido a que de esta se ha deducido la forma de interrogar a los mismos. Esta clase hereda de la clase “Query”. En la figura 222, podemos observar la declaración de una serie de atributos con el nombre de los diferentes campos asociados en el servidor al título, ISBN, etc. En la figura 223, la definición del servidor sobre el que vamos a realizar la consulta y la consulta de la misma. En la figura 224, la adaptación del nombre de los campos y de los operadores relacionales empleados en nuestra aplicación, por los que requiere la consulta al servidor.

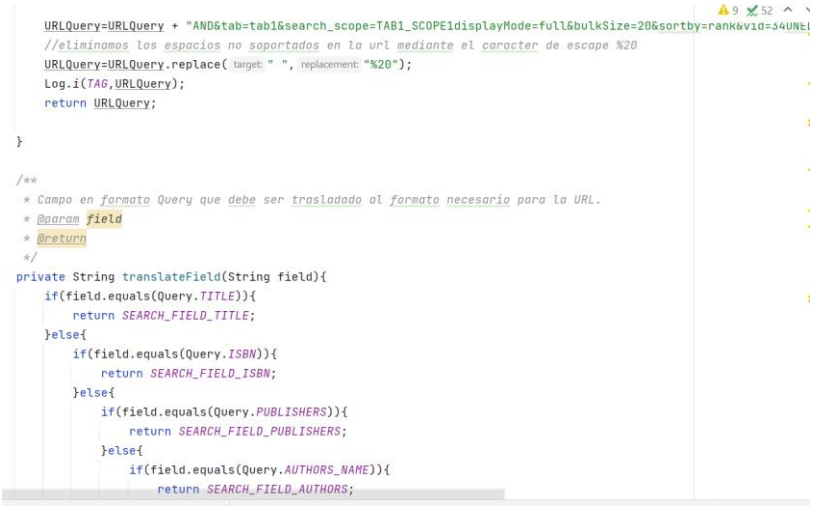


```java
/**
 * Compone una query en formato URl siguiendo el estándard SRU a partir de los datos selecionados por el usuario.
 *
 * @author Enrique Garcerá Rayo.
 * @version 1.0
 */
public class OPACHTMLQuery extends Query {

    public static String TAG="OPACHTMLQuery";

    public static final String SEARCH_FIELD_ALLS="any";
    public static final String SEARCH_FIELD_ISBN="isbn";
    public static final String SEARCH_FIELD_TITLE="title";
    public static final String SEARCH_FIELD_PUBLISHERS="lsr03";
    public static final String SEARCH_FIELD_AUTHORS="creator";

    private String server="";


    public String getServer() { return server; }
    public void setServer(String server) { this.server=server; }

    /**
     *  URL del servidor OPAC con los parámetros de interrogación.
```

figura 222. Contenido de la clase OPACHTMLQuery.

```java
 */
@Override
public String getQueryString(){
    if(this.queryElements==null){return "";}
    Iterator<QueryElement> iter=this.queryElements.iterator();
    if(iter==null){return "";}
    QueryElement queryElement;
    //componemos la consulta.
    //a futuro recuperaremos el servidor de las propiedades, pero habrá que definir la casuística de interrogación
    server="https://buscador.biblioteca.uned.es/primo-explore/search?";
    String URLQuery=server;
    //URLQuery=URLQuery;
    while(iter.hasNext()){
        queryElement=iter.next();
        if(queryElement!=null && iter.hasNext()){
            URLQuery=URLQuery  + "query=" + translateField(queryElement.getField()) + ",contains," + queryElement.

        }else{
            if(queryElement!=null && !iter.hasNext()) {
                URLQuery=URLQuery +  "query=" + translateField(queryElement.getField()) + ",contains," + queryElem
            }

        }

    }
```

figura 223. Contenido de la clase OPACHTMLQuery (cont).

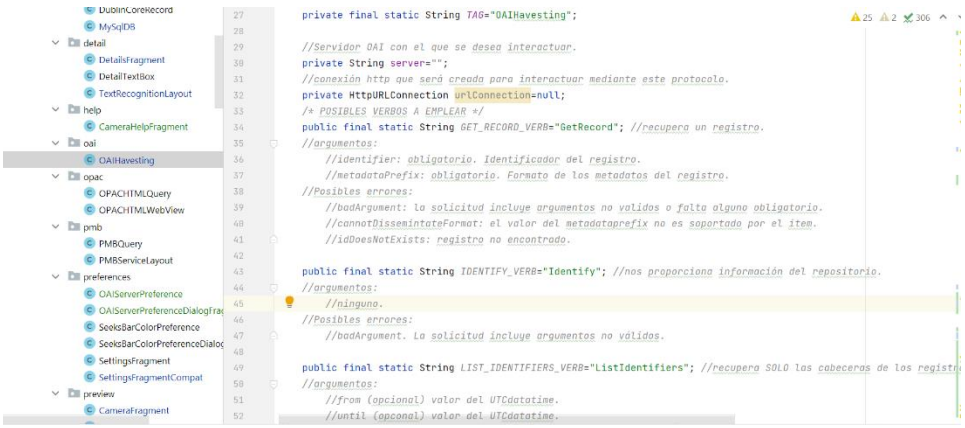


```
    URLQuery=URLQuery + "AND&tab=tab1&search_scope=TAB1_SCOPE1displayMode=full&bulkSize=20&sortby=rankkvid=34UNE
    //eliminamos los espacios no soportados en la url mediante el caracter de escape %20
    URLQuery=URLQuery.replace( target: " ", replacement: "%20");
    Log.i(TAG,URLQuery);
    return URLQuery;

}

/**
 * Campo en formato Query que debe ser trasladado al formato necesario para la URL.
 * @param field
 * @return
 */
private String translateField(String field){
    if(field.equals(Query.TITLE)){
        return SEARCH_FIELD_TITLE;
    }else{
        if(field.equals(Query.ISBN)){
            return SEARCH_FIELD_ISBN;
        }else{
            if(field.equals(Query.PUBLISHERS)){
                return SEARCH_FIELD_PUBLISHERS;
            }else{
                if(field.equals(Query.AUTHORS_NAME)){
                    return SEARCH_FIELD_AUTHORS;
```

figura 224. Clase OPACHTMLQuery.

13.2.1.15 OAIHavesting.

Esta clase es empleada para interrogar al servidor OAI configurado mediante las preferencias de la aplicación. Estas preferencias disponen de un apartado dónde indicándole el servidor OAI, nos permiten cargar datos en la base de datos SQLite local para ofrecer sugerencias al usuario cuando utilice la aplicación. Esta clase es bastante extensa, dado que desarrolla múltiples posibilidades de interrogación sobre un servidor OAI empleando el formato DC, por ejemplo, al emplear el verbo “Identify” el servidor nos devolverá información sobre el repositorio, si ya disponemos del identificador de un registro, podremos recuperar este mediante el uso del verbo “GetRecord”, etc.



```
DublinCoreRecord
MySqlDB
detail
DetailsFragment
DetailTextBox
TextRecognitionLayout
help
CameraHelpFragment
oai
OAIHavesting
opac
OPACHTMLQuery
OPACHTMLWebView
pmb
PMBQuery
PMBServiceLayout
preferences
OAIServerPreference
OAIServerPreferenceDialogFra
SeeksBarColorPreference
SeeksBarColorPreferenceDialog
SettingsFragment
SettingsFragmentCompat
preview
CameraFragment
```

```
private final static String TAG="OAIHavesting";

//Servidor OAI con el que se desea interactuar.
private String server="";
//conexión http que será creada para interactuar mediante este protocolo.
private HttpURLConnection urlConnection=null;
/* POSIBLES VERBOS A EMPLEAR */
public final static String GET_RECORD_VERB="GetRecord"; //recupera un registro.
//argumentos:
    //identifier: obligatorio. Identificador del registro.
    //metadataPrefix: obligatorio. Formato de los metadatos del registro.
//Posibles errores:
    //badArgument: la solicitud incluye argumentos no validos o falta alguno obligatorio.
    //cannotDisseminateFormat: el valor del metadataprefix no es soportado por el item.
    //idDoesNotExists: registro no encontrado.

public final static String IDENTIFY_VERB="Identify"; //nos proporciona información del repositorio.
//argumentos:
    //ninguno.
//Posibles errores:
    //badArgument. La solicitud incluye argumentos no válidas.

public final static String LIST_IDENTIFIERS_VERB="ListIdentifiers"; //recupera SOLO las cabeceras de los registr
//argumentos:
    //from (opcional) valor del UTCdatatime.
    //until (opconal) valor del UTCdatatime.
```

figura 225. Verbos disponibles para interrogar el servidor OAI.

Algunos de los posibles parámetros a emplear para interrogar al servidor se pueden observar en la figura 226.

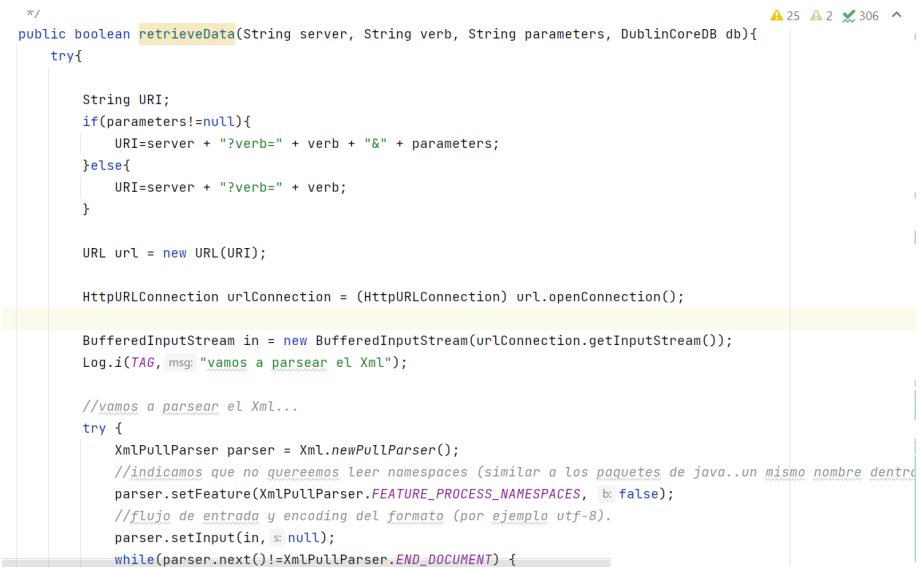
```
    //noSetHierarchy

/* POSIBLES PARÁMETROS A EMPLEAR */
public final static String RESUMPTION_TOKEN_PARAMETER="resumptionToken";
public final static String FROM_PARAMETER="from";
public final static String UNTIL_PARAMETER="until";
public final static String SET_PARAMETER="set";
public final static String METADATA_PREFIX_PARAMETER="metadataPrefix";
public final static String IDENTIFIER_PARAMETER="identifier";

/* FORMATOS SOPORTADOS POR LO SERVIDORES OAI-PMH */
public final static String OAI_DC_FORMAT="oai_dc";
public final static String MARCXML_FORMAT="marcxml";


/**
 * Constructor de la clase.
 * @param server servidor OAI con el que deseamos conectar.
 */
public OAIHavesting(String server){
    this.server=server;

}
```

figura 226. Parámetros disponibles para interrogar al servidor OAI.

En la figura 227, podemos observar la composición de la consulta que será enviada mediante un flujo de datos empleando el protocolo HTTP (HyperText Transfer Protocol). Una vez recibida la respuesta y dado que el servidor nos responderá con datos en formato XML, nos hemos apoyado de un “parser xml” para interpretar la estructura de dichos datos. El “parser” empleado es “XmlPullParser”. Un fragmento del código desarrollado lo podemos encontrar en la figura 228.

```
/**
 * Crea un flujo de entrada de datos a partir del verbo y los parámetros. Un ejemplo de interrogacion a un servido
 * http://oai.bne.es/OAIHandler?verb=GetRecord&identifier=oai.bne.esmonografias_antiguas:bima0000029021&metadataPr
 * @param verb la parte del ejemplo correspondiente a: GetRecord
 * @param parameters la parte del ejempo correspondiente a: identifier=oai.bne.esmonografias_antiguas:bima00000290
 * @return flujo de dentrada de datos.
 */
public InputStream getDataInputStream(String verb,String parameters){
    InputStream inputStream=null;
    try{

        String URI;
        if(parameters!=null){
            URI=server + "?verb=" + verb + "&" + parameters;
        }else{
            URI=server + "?verb=" + verb;
        }

        URL url = new URL(URI);

        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        inputStream= urlConnection.getInputStream();
```

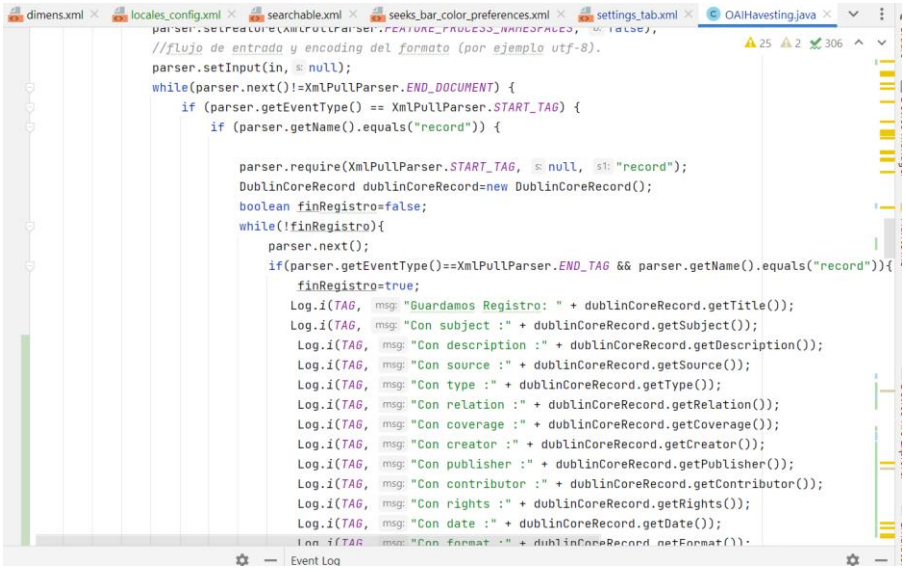
figura 227. Composición de la consulta para interrogar al servidor OAI.



```
*/
public boolean retrieveData(String server, String verb, String parameters, DublinCoreDB db){
    try{

        String URI;
        if(parameters!=null){
            URI=server + "?verb=" + verb + "&" + parameters;
        }else{
            URI=server + "?verb=" + verb;
        }

        URL url = new URL(URI);

        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

        BufferedInputStream in = new BufferedInputStream(urlConnection.getInputStream());
        Log.i(TAG, msg: "vamos a parsear el Xml");

        //vamos a parsear el Xml...
        try {
            XmlPullParser parser = Xml.newPullParser();
            //indicamos que no quereemos leer namespaces (similar a los paquetes de java..un mismo nombre dentro
            parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, b: false);
            //flujo de entrada y encoding del formato (por ejemplo utf-8).
            parser.setInput(in, s: null);
            while(parser.next()!=XmlPullParser.END_DOCUMENT) {
```

figura 228. Respuesta del servidor y uso del parser XML (XMLPullParser).

Como somos conocedores de la estructura de los datos que responde el servidor, dado que investigamos sobre esta y analizamos diferentes respuestas obtenidas del servidor OAI de la BNE, somos conocedores de las diferentes etiquetas y la relación padre-hijo presente entre las mismas.



```
dimens.xml | locales_config.xml | searchable.xml | seeks_bar_color_preferences.xml | settings_tab.xml | OAIHavesting.java
            //flujo de entrada y encoding del formato (por ejemplo utf-8).
            parser.setInput(in, s: null);
            while(parser.next()!=XmlPullParser.END_DOCUMENT) {
                if (parser.getEventType() == XmlPullParser.START_TAG) {
                    if (parser.getName().equals("record")) {

                        parser.require(XmlPullParser.START_TAG, s: null, s1: "record");
                        DublinCoreRecord dublinCoreRecord=new DublinCoreRecord();
                        boolean finRegistro=false;
                        while(!finRegistro){
                            parser.next();
                            if(parser.getEventType()==XmlPullParser.END_TAG && parser.getName().equals("record")){
                                finRegistro=true;
                                Log.i(TAG, msg: "Guardamos Registro: " + dublinCoreRecord.getTitle());
                                Log.i(TAG, msg: "Con subject :" + dublinCoreRecord.getSubject());
                                Log.i(TAG, msg: "Con description :" + dublinCoreRecord.getDescription());
                                Log.i(TAG, msg: "Con source :" + dublinCoreRecord.getSource());
                                Log.i(TAG, msg: "Con type :" + dublinCoreRecord.getType());
                                Log.i(TAG, msg: "Con relation :" + dublinCoreRecord.getRelation());
                                Log.i(TAG, msg: "Con coverage :" + dublinCoreRecord.getCoverage());
                                Log.i(TAG, msg: "Con creator :" + dublinCoreRecord.getCreator());
                                Log.i(TAG, msg: "Con publisher :" + dublinCoreRecord.getPublisher());
                                Log.i(TAG, msg: "Con contributor :" + dublinCoreRecord.getContributor());
                                Log.i(TAG, msg: "Con rights :" + dublinCoreRecord.getRights());
                                Log.i(TAG, msg: "Con date :" + dublinCoreRecord.getDate());
                                Log.i(TAG, msg: "Con format :" + dublinCoreRecord.getFormat());
Event Log
```

figura 229. Ejemplo de recorrido de la respuesta del servidor mediante el parser XML.

En la figura 229, podemos observar como buscamos el TAG “record” con vistas a obtener los datos presentes entre <record></record>. Esto es equivalente al uso de XmlPullParser.START_TAG y XmlPullParser.END_TAG. Los datos que vamos recuperando los vamos cargando en objetos de tipo “DublinCoreRecord” para su posterior inserción en la base de datos SQLite local. En general se ha realizado una documentación del código fuente bastante detallada, pero en esta clase en concreto los comentarios son si cabe más abundantes, y simplemente leyéndolos es posible hacerse una idea bastante clara del mecanismo de consulta, recuperación e interpretación de resultados sobre un servidor OAI, por lo que se recomienda encarecidamente la consulta de dicho código fuente si se desea ampliar información.

13.2.1.16 QueryFactory.

Para favorecer el empleo del polimorfismo, hemos hecho uso del patrón “factory”. Dado que es posible que tengamos que realizar una “query” a diferentes sistemas, cada uno de ellos con una casuística particular, hemos creado una factoría de “Querys”. En la figura 230, podemos observar el código fuente correspondiente a dicha factoría. El empleo del método estático “getNewQuery” y que por tanto puede ser utilizado por cualquier clase sin necesidad de realizar la operación new sobre la clase QueryFactory, nos devolverá una query del tipo pasado como parámetro. Esto que aparentemente pueda parecer que no presenta mucha utilidad, es realmente muy útil, dado que, si observamos la figura 231, podemos ver como en lugar de crear una variable del modo:

“SQLQuery sqlquery=new SQLQuery()”, o “OPACHTMLQuery=new OPACHTMLQuery()”, que nos condiciona en tiempo de diseño el tipo de query que vamos a realizar, instanciamos de la manera Query query=QueryFactor(tipoQuery). De esta manera se puede decidir en tiempo de ejecución el tipo de servidor sobre el que vamos a realizar la consulta, adaptándose a este y nos evitamos a lo largo del resto del código la necesaria discriminación entre si es un tipo de servidor u otro.

Hay que tener en cuenta que uno de los requisitos que debe cumplirse al emplear este patrón para poder emplear la palabra “Query” en lugar de la específica al tipo de servidor, es que todas las clases que podemos instanciar en la factoría implementen una interfaz común (IQuery). Es decir, por ejemplo, en la figura 231, empleamos la instrucción query.addQueryElement(elemento), por lo que cualquier clase que pueda ser instanciada en la factoría, deberá implementar dicho método adaptado a sus necesidades (polimorfismo).

figura 230. Código fuente de la clase QueryFactory.

figura 231. Uso del polimorfismo dentro de la clase TextRecognitionLayout.

13.2.1.17 QueryElement.

La presente clase se empleará como mecanismo de encapsulamiento de las selecciones realizadas en la pantalla de detalle y que por tanto servirán para interrogar al servidor correspondiente (por ejemplo el servidor PMB). Esta es una clase muy simple que contiene un atributo “field” que almacenará el

campo seleccionado (Título, ISBN, Editorial o Autor) y otro “text” que contendrá el valor sobre el que realizar la búsqueda (texto detectado y mostrado en cada uno de los DetailTextBoxes). Empleando buenas prácticas, hemos implementado los métodos “get” y “set” para no acceder directamente al valor del atributo. En la figura 231, se puede observar el empleo de la misma en el momento que estamos recolectando los campos seleccionados por el usuario en la pantalla de detalle para componer la Query de interrogación.

13.2.1.18 OAIServerPreferenceDialogFragmentCompat.

Pese a que en el kit de desarrollo de Android Studio existen unas preferencias genéricas con una interfaz bastante simple y limitada que se deben poner en la carpeta XML del proyecto, se pueden personalizar estas para un resultado de cara al usuario mucho más elaborado. Este es el caso de la presente clase que, además de emplear las preferencias compartidas y mostrarse en la pantalla de preferencias, nos permite cargar los metadatos del servidor OAI a la base de datos SQLite local. La presente clase hereda de la clase “PreferenceDialogFragmentCompat”. Para poder emplear la presente clase en la pantalla de preferencias debemos crear un método estático que devuelva una instancia de la misma pasando la clave de la preferencia como parámetro (figura 232).

```
public static OAIServerPreferenceDialogFragmentCompat newInstance(String key) {
    final OAIServerPreferenceDialogFragmentCompat fragment = new OAIServerPreferenceDialogFragmentCompat();

    final Bundle b = new Bundle( capacity: 1);
    //pasamos la clave de la preferencia con el argumento por defecto "key". Empleamos la constante ARG_KEY para
    b.putString(ARG_KEY, key);
    fragment.setArguments(b);

    return fragment;

}
```

figura 232. Método estático del dialogo de carga de datos del servidor OAI.

Del mismo modo y dado que se trata de un dialogo, hemos implementado el método “onDialogClosed” que recibe como parámetro un valor lógico de verdadero o falso. Dado que el dialogo muestra el botón de aceptar y el de cancelar, en caso de pulsar aceptar, el valor recibido como parámetro será verdadero y en nuestro caso lo hemos empleado para dar por válido el nuevo valor de la propiedad asociada a la URL del servidor OAI.

```
/**
 * En caso de que el usuario haya pulsado sobre aceptar, almacenamos el nuevo valor de la preferencia
 * @param positiveResult true si han pulsado sobre el botón Aceptar, false en caso contrario.
 */
@Override
public void onDialogClosed(boolean positiveResult) {
    if(positiveResult){

        SharedPreferences.Editor editor=sharedPreferences.edit();

        editor.putString( s: "oai_server_url",OAIServerTextInput.getText().toString());
        editor.apply();

    }

}
```

figura 233. Implementación del método onDialgoClosed.

Del mismo modo, hemos implementado el método “onClick” para que, en caso de pulsar sobre el botón de confirmación de la “carga de datos”, se proceda a esta y en caso de pulsar sobre el botón cancelar, no.

```
@Override
public void onClick(View view) {
    if(view.getId()==loadDataButton.getId()) {
        AlertDialog.Builder confirmationDialog = new AlertDialog.Builder(this.getContext());
        confirmationDialog.setMessage("seguro?????");
        //confirmationDialog.setPositiveButton()
        confirmationDialog.setCancelable(true);
        //al mostrar lista desplegable ocupamos la zona de mensajes, por lo que mostramos en el titulo el mensa
        confirmationDialog.setTitle("¿Seguro que deseas recargar los datos?. Este proceso ra...");
        confirmationDialog.setPositiveButton( text: "Aceptar", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                loadDataButton.setEnabled(false);
                //cerramos el dialogo sin esperar la carga de datos.
                dialog.dismiss();
                //cosechamos lo datos cargandolos en la base de datos local.
                harvest();
            }
        });

        confirmationDialog.setNegativeButton( text: "Cancelar", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                //finalmente no quieren realizar la carga de datos en la base de datos local.
            }
        });
```

figura 234. método onClick para confirmar o denegar la carga de datos del servidor OAI.

Para no monopolizar el hilo de ejecución principal de la aplicación en el proceso de carga de datos que puede tardar bastante tiempo, hemos creado un nuevo hilo de ejecución (figura 235).

```
public void harvest() {

    this.loadDataButton.setEnabled(false);
    //esta clase nos permite ejecutar una tarea asíncrona en un nuevo hilo de ejecución y recuperar el resulta
    //permite Runnable y Callable.
    ExecutorService executor = Executors.newSingleThreadExecutor();

    //creamos un manejador sobre el hilo principal para intercambiar mensajes y poder mostrar por pantalla la
    Handler handler = new Handler(Looper.getMainLooper());
    if(listView.getCheckedItemCount()==0){
        //no hay items que cargar. Carga finalizada.
        return;
    }

    //necesario declararlo final para emplear expresiones lambda.
    final SparseBooleanArray checkedItemsPositions=listView.getCheckedItemPositions();


    Future<String> future= (Future<String>) executor.submit(() -> havestingTask(checkedItemsPositions));
    try {
```

figura 235. Cosecha de datos del servidor OAI. Creación de un nuevo hilo de ejecución.

13.2.1.19 OAIServerPreferences.

La presente clase extiende de “DialogPreferences” y consiste en un cuadro de dialogo que será mostrado al usuario para que pueda modificar la URL del servidor OAI configurado en la aplicación.

```
public class OAIServerPreference extends DialogPreference {
    private static final String TAG="OAIServerPreference";

    //preferencias compartidas.
    private SharedPreferences sharedPreferences;

    //url del servidor OAI.
    private String OAIServer;


    /**
     * Constructor de la clase.
     * @param context
     * @param attrs
     * @param defStyleAttr
     * @param defStyleRes
     */
    public OAIServerPreference(@NonNull Context context, @Nullable AttributeS
        super(context, attrs, defStyleAttr, defStyleRes);
         initialize();
    }
```

figura 236. Clase OAIServerPreference.

```
private void initialize(){

    setDialogLayoutResource(R.layout.oai_server_settings);

        sharedPreferences= PreferenceManager.getDefaultSharedPreferences(getContext());

        OAIServer=sharedPreferences.getString( s: "oai_server_url", s1: "");


}
```

figura 237. Clase OAIServerPreferences. Asignación del servidor OAI.

13.2.1.20 SeekBarColorPreferenceDialog.

La presente clase extiende de “PreferenceDialogFragmentCompat” y mostrará tres barras deslizantes para poder realizar la selección de un color empleando la codificación RGB. Con vistas a poder utilizar el presente código en varios contextos (seleccionar el color del texto de la pantalla de detalle o vista previa, seleccionar el color del marco que rodea el texto detectado, etc.), dentro de la clase “SeeksBarColorPreference” se han implementado los métodos “getKeyRedPreference”, “getKeyGreenPreference” y “getKeyBluePreference”. De este modo al crear las preferencias asociadas al color, asignaremos una clave distinta a las tres preferencias RGB (Red, Green, Blue) empleadas para el color de texto de detalle, otras claves para las del marco, etc.

```
/**
 * Retorna la clave de la propiedad donde se almacena la cantidad de color rojo.
 * @return clave de la preferencia donde se alamacena el color.
 */
```

figura 238. getKeyRedPreference de la clase SeeksBarColorPrefence.

```
*/
public SeeksBarColorPreference(@NonNull Context context, @Nullable AttributeSet attrs, int defStyleAttr, int defS
    super(context, attrs, defStyleAttr, defStyleRes);

    final TypedArray typedArray=this.getContext().obtainStyledAttributes(attrs,R.styleable.SeeksBarColorPreferenc
    //recuperamos el nombre de la clave asocida a cada una de las preferencias donde deseamos almacenar el valor
    keyRedPreference=TypedArrayUtils.getString(typedArray,R.styleable.SeeksBarColorPreference_keyRedPreference,R.
    keyGreenPreference=TypedArrayUtils.getString(typedArray,R.styleable.SeeksBarColorPreference_keyGreenPreferenc
    keyBluePreference=TypedArrayUtils.getString(typedArray,R.styleable.SeeksBarColorPreference_keyBluePreference,

    initialize();
}
```

figura 239. Clase SeeksBarColorPreference.

```
 * @param view Vista correspondiente al cuadro de dialogo.
 */
@Override
public void onBindDialogView(View view){

    //recuperamos los elementos del panel de dialogo.
    panelShowColor=view.findViewById(R.id.settings_color_selection_show_color_panel);
    redSlider=view.findViewById(R.id.settings_color_selection_red_slider);
    greenSlider=view.findViewById(R.id.settings_color_selection_green_slider);
    blueSlider=view.findViewById(R.id.settings_color_selection_blue_slider);

    //recuperamos los valores cargados previamente en la preferencia para cargarlos en el panel.
    sharedPreferences= PreferenceManager.getDefaultSharedPreferences(getContext());

    redValue=sharedPreferences.getInt(((SeeksBarColorPreference)this.getPreference()).getKeyRedPreference(), 0
    greenValue=sharedPreferences.getInt(((SeeksBarColorPreference)this.getPreference()).getKeyGreenPreference()
    blueValue=sharedPreferences.getInt(((SeeksBarColorPreference)this.getPreference()).getKeyBluePre
                                                                                    Device File Exp

    redSlider.setValue((float)redValue);
    greenSlider.setValue((float)greenValue);
    blueSlider.setValue((float)blueValue);

    panelShowColor.setBackgroundColor(Color.rgb(redValue,greenValue,blueValue));
```

figura 241. Método de enlace de dialogo de preferencias para la selección de un color.

```
private void initialize(){

    setDialogLayoutResource(R.layout.settings_color_selection);


    if(this.getIcon()!=null) {
        int redValue;
        int greenValue;
        int blueValue;
        sharedPreferences= PreferenceManager.getDefaultSharedPreferences(getContext());

        redValue=sharedPreferences.getInt(keyRedPreference, 0);
        greenValue=sharedPreferences.getInt(keyGreenPreference, 0);
        blueValue=sharedPreferences.getInt(keyBluePreference, 0);


        this.getIcon().setTint(Color.rgb(redValue,greenValue,blueValue));
    }

}
```

figura 240. Clase SeeksBarKeyPreference.

En la figura 240, destacar la instrucción “getIcon().setTint” que nos permite cambiar el código de color de un icono que hemos puesto en la pantalla de selección de color, para que, en tiempo real, el usuario sepa que color está seleccionando al deslizar las barras de selección RGB. También hemos puesto otro en la pantalla de propiedades, para, sin necesidad de abrir el cuadro de selección de color, saber cuál es el color actual en las preferencias de la aplicación.

13.2.1.21 SettingsFragment.

Esta clase hereda de “Fragment” y es el fragmento que se carga al invocar la pantalla de preferencias de la aplicación. Como se puede observar en la figura 242, al inflar el fragmento, el fichero XML que se está empleando es “settings_view_tab.”.

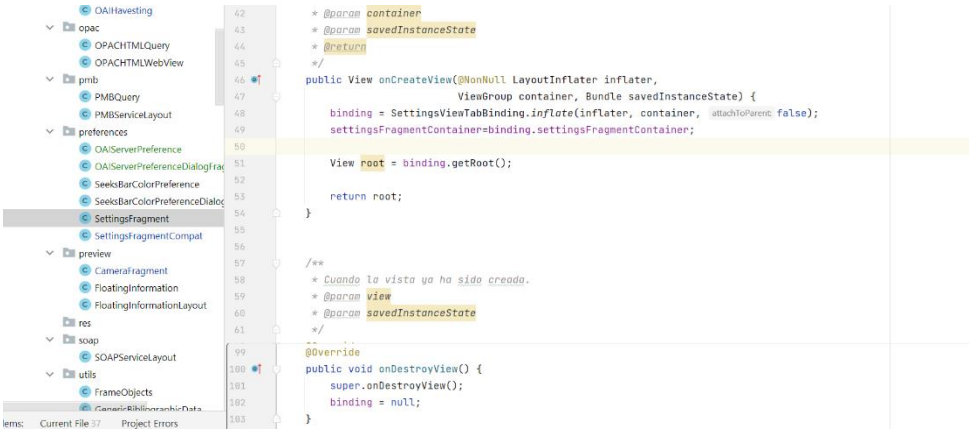


figura 242. Clase SettingsFragment.

figura 243. Detalle de la instanciación de la clase SettingsFragmentCompat dentro de la clase SettingsFragment.

13.2.1.22 Settings_view_tab.xml

Como se puede observar, el contexto del presente fichero XML es la clase “SettingsFragment” y como ya sea ha comentado, es el punto de entrada de la pantalla de preferencias de nuestra aplicación. En este fragmento se incluye la barra de herramientas presente en la pantalla de configuración de las propiedades de la aplicación y el contenedor de fragmentos que nos permitirá usar el gráfico de navegación.

figura 244. fichero settings_view_tab.xml

13.2.1.23 Settings_tab.xml

El presente fichero XML contiene el conjunto de preferencias que se pueden configurar en nuestra aplicación. Dada su extensión, solo se describirá alguna de ellas con vistas a entender cómo funciona. En la figura 245, podemos observar como el fichero comienza con la etiqueta “PreferenceScreen”. Este “tag” es único y se corresponde con la pantalla de preferencias que hemos mostrado previamente en el Wireframing de la aplicación. Podemos observar que su contexto se corresponde con la clase “SettingsFragmentCompat”. Como subetiquetas de esta etiqueta, nos encontramos con <PrefenceCategory> que es simplemente una agrupación de determinadas preferencias y que nos permite asociar un título para dicha agrupación. Por ejemplo, “Configuración Vista Previa”. Finalmente, y dentro de cada categoría nos encontramos con las preferencias propiamente dichas. Estas pueden mostrarse al usuario de diversas maneras. Por ejemplo, como un

“Switch” para que el usuario lo active o desactive, como un listado de preferencias o incluso, como un tipo de preferencia personalizada como la de nuestra “SeeksBarColorPreference”.

```xml
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".preferences.SettingsFragmentCompat">

    <PreferenceCategory
        android:icon="@android:drawable/ic_menu_view"
        android:iconSpaceReserved="true"
        android:key="previewview_settings_content"
        android:title="CONFIGURACIÓN VISTA PREVIA"
        android:summary="Contenido a mostrar.">
        <!-- configuración relativa a los contenidos a mostrar en la vista previa. -->

        <SwitchPreference
            android:key="show_contextual_information"
            android:defaultValue="true"
            android:title="Mostrar información contextial en vista previa." />
        <!--
        <CheckBoxPreference
            android:defaultValue="true"
```

figura 245. Fichero XML de preferencias de la aplicación.

```xml
    <ListPreference
        android:key="previewview_detection_type_setting"
        android:defaultValue="block"
        android:entries="@array/preview_view_detection_type_entries"
        android:entryValues="@array/preview_view_detection_type_values"
        android:title="Tipo de detección (Párrafo, línea o palabra)"
        app:useSimpleSummaryProvider="true" />

</PreferenceCategory>

<!-- configuraciones relativas a la apariencia deseada para la vista previa -->
<PreferenceCategory
    android:key="previewview_settings_appearance"
    android:icon="@android:drawable/ic_menu_view"
    android:iconSpaceReserved="true"
    android:title="CONFIGURACIÓN VISTA PREVIA"
    android:summary="Apariencia"
>
    <!-- selección del color para el borde de la detección le pasamos como parámetro las propiedades que almac
    <com.armliuv30.preferences.SeeksBarColorPreference
            android:dialogIcon="@drawable/common_google_signin_btn_text_disabled"
            android:icon="?android:attr/textSelectHandleLeft"
            android:iconSpaceReserved="true"
```

figura 246. Uso de la preferencia personalizada SeeksBarColorPreference.

```
<com.armliuv30.preferences.OAIServerPreference
    android:dialogIcon="@drawable/common_google_signin_btn_text_disabled"
    android:icon="?android:attr/textSelectHandleLeft"
    android:iconSpaceReserved="true"
    android:key="oai_server_settings"
    android:title="Servidor OAI-PMH"
    app:negativeButtonText="Cancelar"
    app:positiveButtonText="Aceptar"
    />

<!-- preferencias ocultas para el servidor OAI -->
<Preference
    android:defaultValue="http://oai.bne.es/OAIHandler"
    android:key="oai_server_url"
    app:isPreferenceVisible="false" />
```

figura 247. Definición de la preferencia OAIServerPreference.

```
    />
</com.google.android.material.appbar.AppBarLayout>

<FrameLayout

    android:id="@+id/settings_fragment_container"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_marginTop="100dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="false"
    app:navGraph="@navigation/navigation_graphic" >
</FrameLayout>
```

figura 248. Contenido settings_view_tab.xml

13.2.1.24 SettingsFragmentCompat.

Esta clase la empleamos para mostrar tanto los cuadros de dialogo de selección de color, como el de carga de datos del servidor OAI. Esta es la forma a través de la que hemos conseguido enlazar, un “fragment” convencional, con el sistemas de preferencias definido por el kit de desarrollo. En la figura 249, podemos observar como se especifica el fichero XML de preferencias a emplear y en la figura 250, como en tiempo de ejecución y en función del tipo de instancia de la que se trate, el método “onPreferenceDisplayDialog” tiene un comportamiento polimórfico generando

una nueva instancia de nuestro dialogo de selección de color o de nuestro dialogo de carga de datos de un servidor OAI.

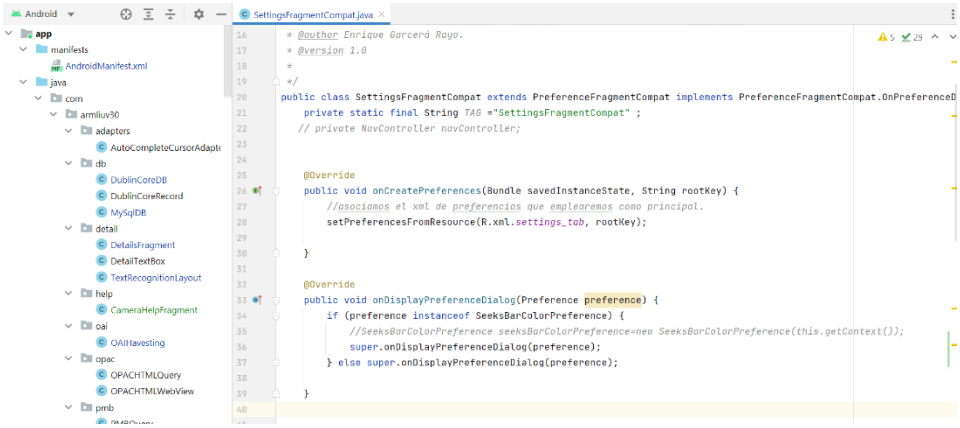


figura 249. Método onCreatePreferences de SettingsFragmentCompat.

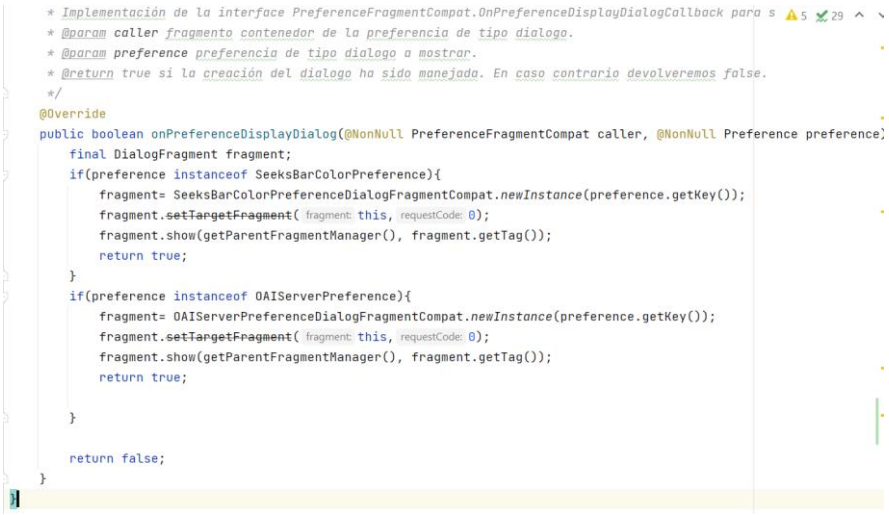


figura 250. Método onPreferenceDisplayDialog de la clase SettingsFragmentCompat.

13.2.1.25 Detail_text_box_state_selector.xml.

El presente fichero XML, es uno de los múltiples selectores que hemos creado en nuestro proyecto. Consideramos que los selectores son realmente útiles, dado que nos permiten decidir en tiempo de ejecución, por ejemplo, como en el caso que nos ocupa, el color de un determinado elemento en función del estado en el que se encuentra.

figura 251. Selector de color en función del estado de un elemento.

En la figura 251, podemos observar como especificamos la etiqueta xml <selector> para crear uno y como dentro de esta etiqueta indicamos los <item> que contiene dicho selector. La última línea de ítems, indica el color que se tendrá en cuenta frente a cualquier estado no definido en los ítems anteriores. En cada ítem podemos observar como se define el estado y el color asociado. También podemos observar como hemos definido un elemento “drawable” para cada ítem. Esta definición es muy importante, dado que para la versión de Android 13 por ejemplo, la aplicación funcionaba correctamente sin especificarla, pero para versiones de Android 8.0, en caso de no definir el “drawable” para cada uno de los ítems, daba error en tiempo de ejecución y la aplicación se paraba.

13.2.1.26 Themes.xml.

Para que nuestra aplicación disponga de un estilo homogéneo a lo largo de todas sus funcionalidades, hemos creado nuestro propio tema heredando de “Theme.MaterialComponents.NoActionBar”, para, partiendo de las recomendaciones de Google sobre “Material Designs”, personalizar este en función de nuestras necesidades. El tema del que hemos heredado nos permite no incluir el “actionbar” de serie, dado que, como hemos comentado, hemos creado una específica para cada pantalla de nuestra aplicación.

Hemos denominado nuestro tema como “Theme.ArmliuV3.0”

```
<style name="Theme.ArmLiuV30" parent="Theme.MaterialComponents.NoActionBar">
```

figura 252. Tema de nuestra aplicación. Herencia de Material Components.

Para poder emplear un tema, hemos de especificarlo en el fichero de manifiesto de la aplicación.

```
<application
    android:largeHeap="true"
    android:allowBackup="true"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:icon="@mipmap/ic_launcher"
    android:supportsRtl="true"
    android:theme="@style/Theme.ArmLiuV30"

    android:usesCleartextTraffic="true"
    android:localeConfig="@xml/locales_config"
    >
```

figura 253. Configuración de la aplicación en el fichero de manifiesto.

Dentro del presente fichero (figura 254), nos encontramos con una serie de etiqueta <item> con diferentes nombres como, “textColorPrimary”, “colorPrimary”, etc. El valor asignado a dichos ítems definirá el color de cada uno de los elementos de la aplicación. Algunas de estas son:

- colorPrimary, es el color principal de nuestra aplicación.
- colorPrimaryVariant, es una variación del color principal de nuestra aplicación. Normalmente más oscura.
- colorSuface: color de determinadas superficies de nuestra aplicación como los botones inferiores o las tarjetas.
- colorOnSurface: color de los elementos que se encuentran encima de una superficie y por tanto está relacionado con colorSurface.
- colorBackground: color que se emplea detrás del contenido. Por ejemplo, un texto.
- colorOnBackground: color de los elementos que se encuentran encima del background.

```
<!-- color de la barra de herramientas y elementos primarios de la aplicación. -->
<item name="colorPrimary">@color/color_primary
</item>
<!-- color de la barra del sistema donde se suele motrar la señal wifi, batería, etc... (barra de estado) -->
<item name="colorPrimaryDark">#FF9800
</item>
<!-- color de los textos principales de la aplicación. Por ejemplo del del toolbar. -->
<!--
<item name="android:textColorPrimary">#FFC107
</item>
-->
<!-- los configuramos marrones para que en la pantalla de propiedades quede más elegante el texto sobre fond
<item name="android:textColorPrimary">#FFC107
</item>


<item name="android:navigationBarColor">@color/color_primary</item>
<!--        <item name="colorPrimaryTitleText">#000000</item> -->
<!-- color de los controles de la interfaz de usuario. Por ejemplo los check boxes. -->
<item name="colorAccent">@color/color_accent
</item> <!-- color de los preferenceCategory por ejemplo -->
<item name="android:windowNoTitle">true</item>
<item name="android:windowActionBar">false</item>
```

figura 254. Contenido del fichero Themes.xml.

También podemos observar como en algunos ítems se ha especificado el color empleando un valor hexadecimal y como en otras hacemos una referencia al fichero donde están definidos determinados colores (“@color/color_primary” por ejemplo).

Los ítems anteriores deberán ubicarse dentro de una etiqueta <style> como la de la figura 255 por ejemplo. En esta figura, podemos observar como los estilos heredan de estilos proporcionados por el API de desarrollo en algunos casos y como en otros heredan de nuestros propios estilos.

```
<style name="Theme.ArmLiuV30.SearchViewStyle" parent="android:Widget.Material.SearchView">
    <!-- Background for the search query section (e.g. EditText) -->
    <item name="android:background">#FF9800</item>


</style>

<!-- creamos subestilos para referenciar a estos dentro de los componenetes de la alicación. De este modo si
distionto a ....MaterialComponents, solo habrá que cambiar una referencia en el valor de parent del del esti
<!-- sintaxis equivalente a <style name="CardView" parent="Theme.ArmLiuV30"> -->
<style name="Theme.ArmLiuV30.CardView">
    <item name="cardUseCompatPadding">true</item>

</style>

<!-- estilo asignado a la tarjeta en la que se muestran cada uno de los registros bibliográficos. -->
<style name="Theme.ArmLiuV30.Register.CardView" parent="Theme.ArmLiuV30.CardView">

    <item name="cardBackgroundColor">@color/backgroundYellow
    </item> <!-- color distinto al del registro para diferenciar ejemplares de forma visual. Por espacio ele
</style>
```

figura 255. Ejemplo de estilos empleados por el tema de nuestra aplicación.

El uso de estilos no se limita a la definición de un color y nos permite, por ejemplo, definir el contorno de un elemento empleando una figura. En la figura 256 podemos ver por ejemplo el fichero “border_chips_groups.xml” que define el color y el contorno con puntas redondeadas de una figura (shape).

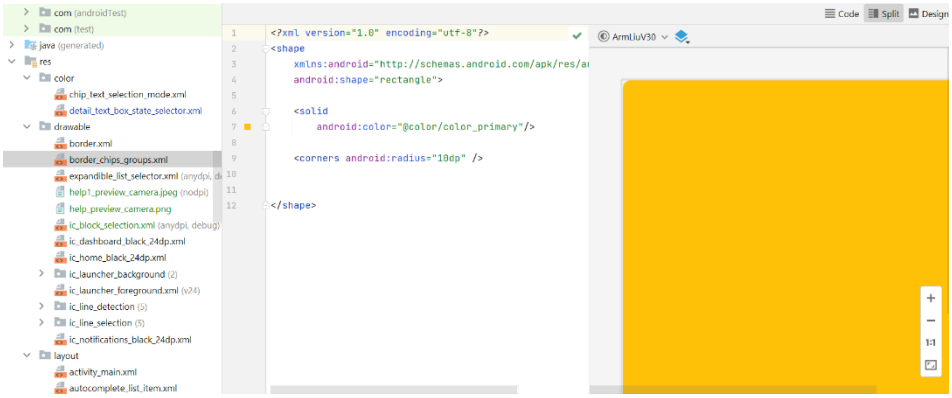


figura 256. Figura empleada para definir un estilo personalizado.

figura 257. Uso del estilo personalizado en la definición de un "Chip" presente en la aplicación.

13.2.1.27 Strings.xml.

El uso del presente fichero es de vital importancia, dado que centraliza todos los textos presentes en nuestra aplicación. Pese a que en las figuras que se han ido mostrando para explicar el contenido de nuestro proyecto se puede observar un texto concreto, realmente este no ha sido especificado directamente en dichos ficheros. Simplemente es que Android Studio, en lugar de mostrar la referencia de donde se encuentra cada uno de esos textos, nos muestra en pantalla el texto en concreto. Sin embargo, todos los textos de etiquetas sobre un elemento, títulos, etc., han sido definidos en el presente fichero y desde el resto de la aplicación, se ha referenciado a este de la manera "@string/identificador_del_texto_a_mostrar".

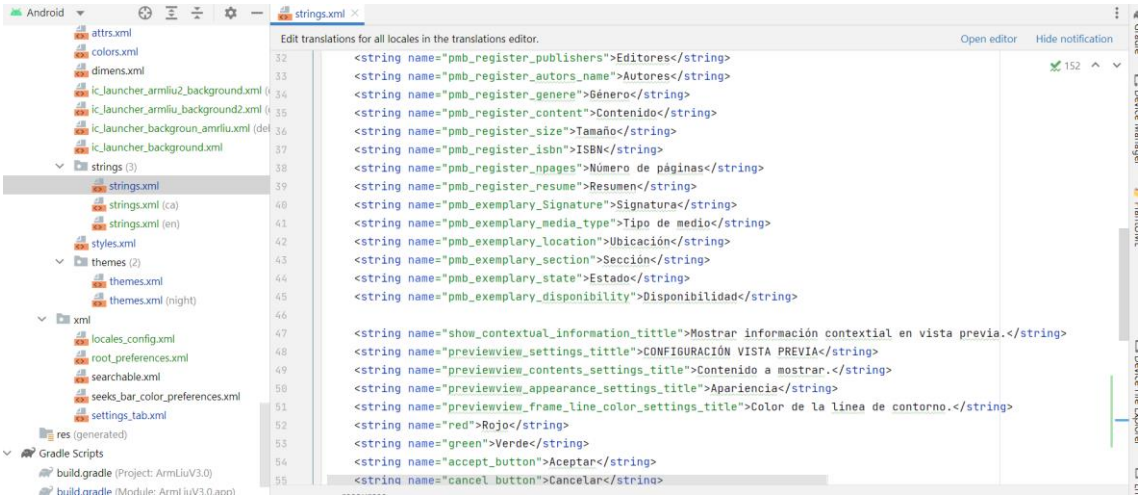


figura 258. Contenido Stirng.xml. Versión Castellano.

En la figura 258, podemos observar como existe una etiqueta <string> por cada texto presente en nuestra aplicación. Dentro de esta etiqueta nos encontramos con el atributo "name" que es el identificador que deberá ser

figura 259. Configuración de los idiomas admitidos por nuestra aplicación.

empleado al usar el texto. Si, por ejemplo, quisiéramos que apareciera en una etiqueta de la aplicación el texto ISBN, deberíamos de indicar la referencia "@string/pmb_register_isbn" y de este modo, en tiempo de ejecución aparecería dicho texto en la etiqueta. En la figura 258, también podemos observar como en el árbol de proyectos de la aplicación nos encontramos con tres ficheros con el nombre Strings.xml, strings.xml(ca) y strings.xml(en). Esto es debido a que hemos creado tres versiones del fichero de textos para que nuestra aplicación se encuentre en tres idiomas distintos (castellano, catalán e inglés). En función del "locale" seleccionado en su dispositivo por parte del usuario, los textos de la aplicación aparecerán en un idioma u otro. Pese a que es necesario mantener tres ficheros con los mismos nombres de elemento, esta manera de proceder para ofrecer la "internacionalización" de nuestra aplicación es bastante efectiva frente a prácticas anteriores.

13.2.1.28 Gradle Scripts.

Hace bastantes años aparecieron herramientas como "Ant", que servían para automatizar el proceso de compilación y generación de versiones de nuestra aplicación. Gradle, que está completamente integrado en Android Studio, sigue la misma filosofía y nos permite definir ciertos aspectos a tener en cuenta a la hora de construir nuestra aplicación, bien sea para su versión de desarrollo, pruebas o puesta en producción. Para tal fin, se emplean una serie de ficheros como el "build.gradle" que nos permite definir dependencias de paquetes java, versión mínima de Android para la que ha sido testeada nuestra aplicación (y por tanto la versión a partir de la cual funcionará), versión del SDK empleada para el proyecto, dependencias, etc.

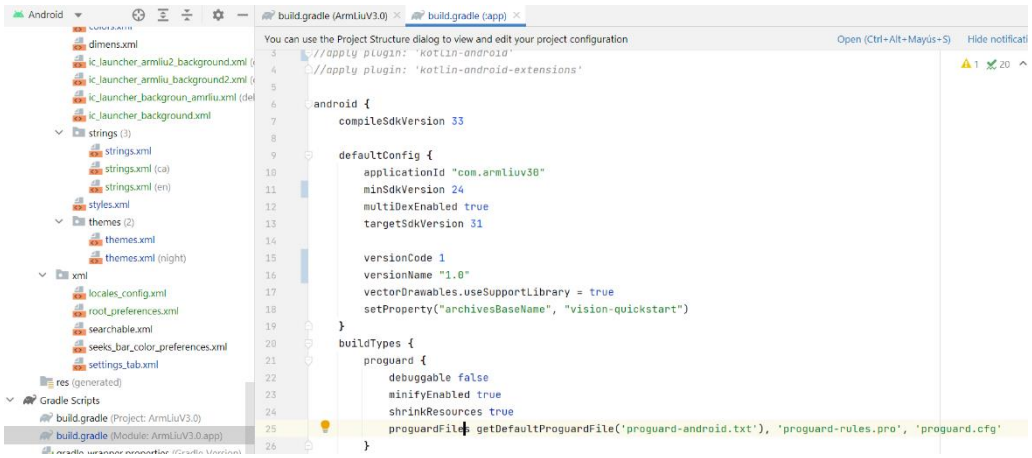


figura 260. Contenido del fichero build.gradle.

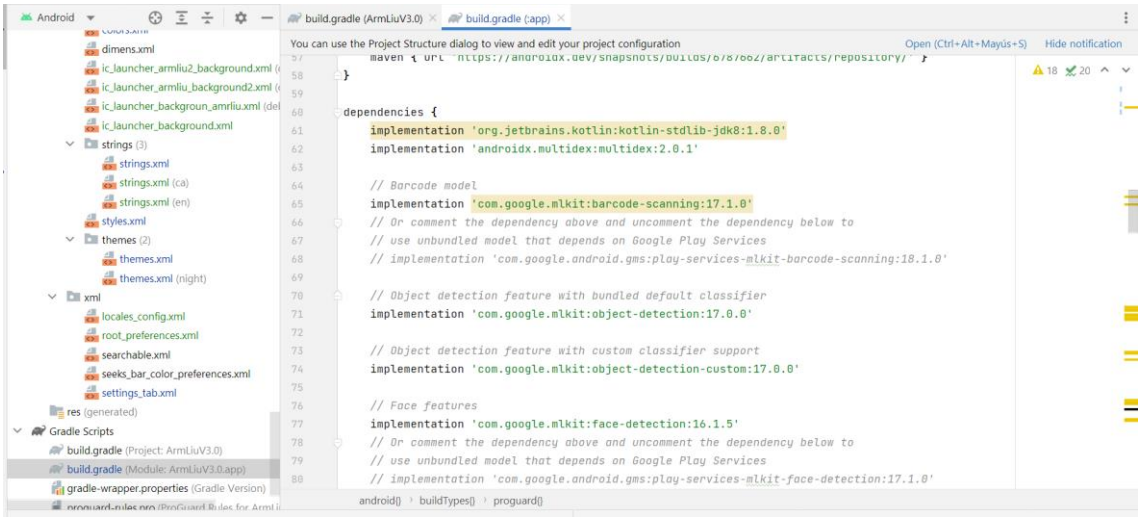


figura 261. Fichero build.gradle. Algunas dependencias de nuestro proyecto.

13.2.2 Documentación mediante JavaDoc

El kit de desarrollo de java dispone de una herramienta muy potente como es la generación automática de la documentación del de nuestra aplicación en formato HTML. Esta documentación se generará mediante JavaDoc y requiere de que, en el momento de comentar el código fuente de nuestra aplicación, se sigan unas directrices mínimas que serán descritas brevemente a continuación.

/** …. */ se interpretará como un comentario Javadoc. Dado que se acabará generando documentos en formato HTML, podemos incluir en estos

comentarios las etiquetas típicas como por ejemplo <p> para indicar el comienzo de un párrafo.

Hemos optado por los siguientes tipos de comentario para que la documentación sea homogénea a lo largo de todas las clases, métodos, etc.

Las etiquetas javadoc empleadas serán:

@author.- autor que creó la clase.

@version.- versión del elemento.

@since.- desde que versión fue añadido el elemento. Por ejemplo @since 1.3 indica que el elemento fue añadido desde la versión de api java 1.3.

@param.- parámetros asociados a un método (una etiqueta @param por cada parámetro).

@return.- valor de retorno de un método.

@exception o @throw.- excepciones que puede generar un método.

{@link}.- Enlace a recursos en línea que suele emplearse para documentar a nivel de paquete.

@see.- referencia a otro elemento (por ejemplo las clases empleadas en los parámetros o como retorno). Lo hemos empleado principalmente para clases y métodos.

Para cada clase se ha especificado:

/**

* Descripción de la clase.

* @author xxxxxx

* @version xxxxxx

* @see xxxxxxxx

*/

Para los métodos se ha especificado:

/**

* Descripción del método.

*@param xxxxx

*@return xxxxxx

*@exception xxxx

*@see xxxxx

Para algunas variables muy concretas para las que se ha considerado útil incluirlas en la documentación del API de la aplicación, también se ha añadido algún comentario Javadoc

/**

* Descripción de la variable.

* datos relevantes sobre los valores que puede tomar dicha variable.

*/

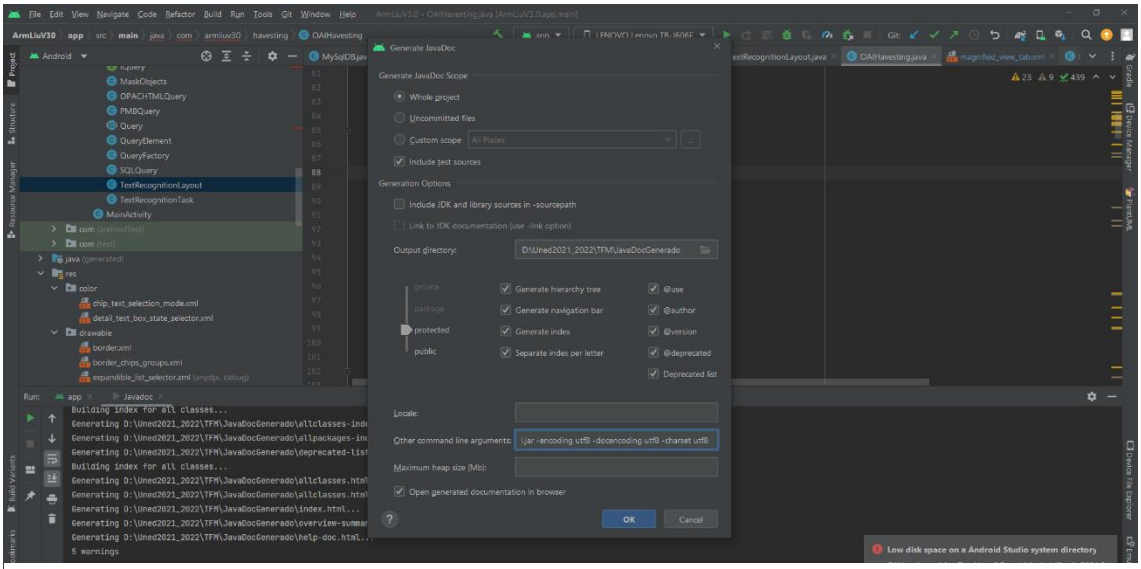


figura 262. Configuración del juego de caracteres empleado por javadoc.

Para conseguir que nos funcionara javadoc dentro de Android Studio, dado que inicialmente nos daba error, hemos configurado el juego de caracteres que estamos empleando (por ejemplo las tildes) y se lo hemos indicado mediante el parámetro “charset” añadiéndolo a “Other command line arguments”.

También hemos tenido que indicarle dónde se encuentra la versión de sdk que estamos empleando. Para ello hemos usado el argumento “-sourcepath” que también se indica en “Other command line arguments”.

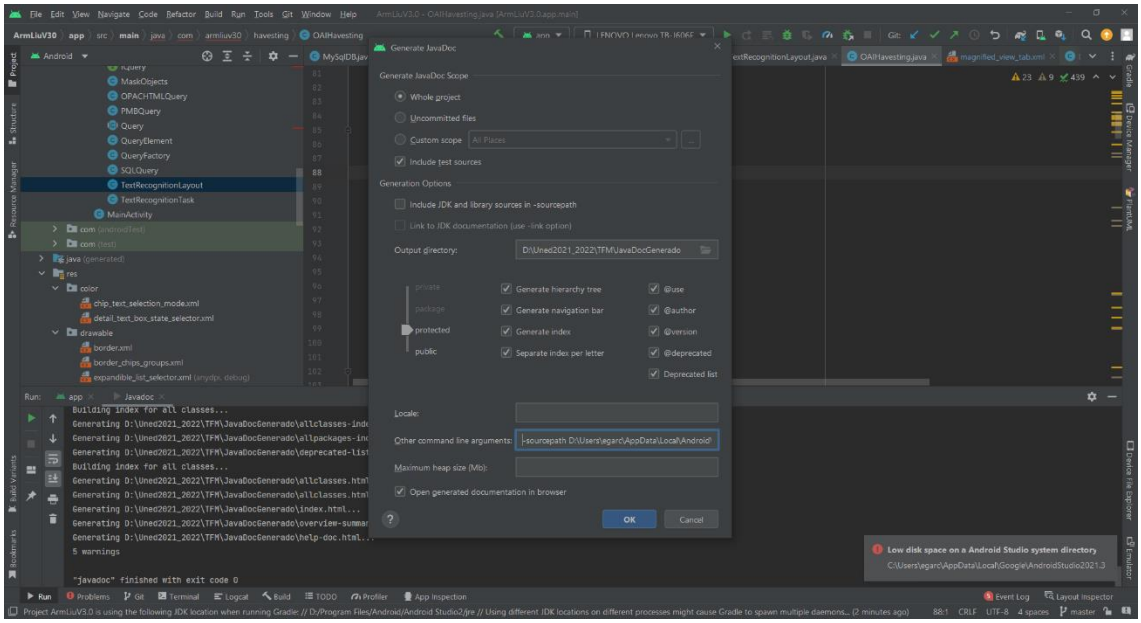


figura 263. Especificación de la ruta del SDK para javadoc.

Por ejemplo, para la versión de sdk-31 hemos indicado la ruta donde se encuentra en nuestro proyecto

(D:\Users\egarc\AppData\Local\Android\Sdk\platforms\android-32\android.jar).

Del mismo modo es vital añadir la dependencia a la estructura del proyecto con dicha ruta. De lo contrario javadoc no funcionará (menú “File”, “Project Structure”). En modules vemos la versión de SDK (Kit de Desarrollo de Software) que estamos empleando (figura 264) y en “sdkLocation” la ruta donde se encuentra el SDK.

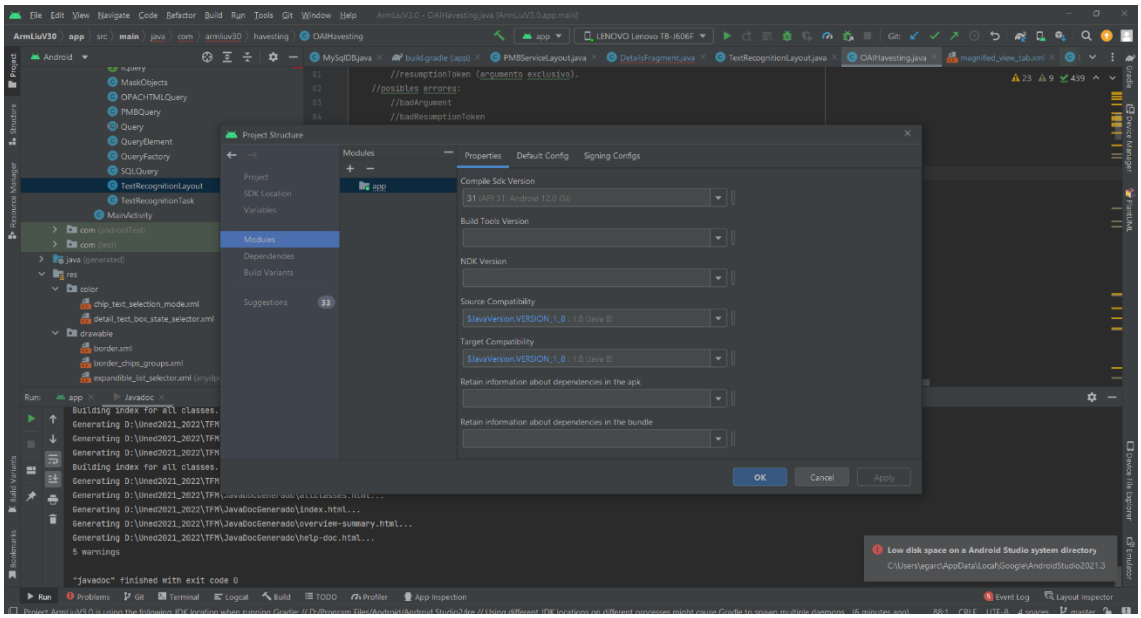


figura 264. Versión del SDK que está empleando nuestro proyecto.

Una vez obtenida la información anterior, en el explorador del sistema operativo confirmamos que disponemos del fichero “android.jar”

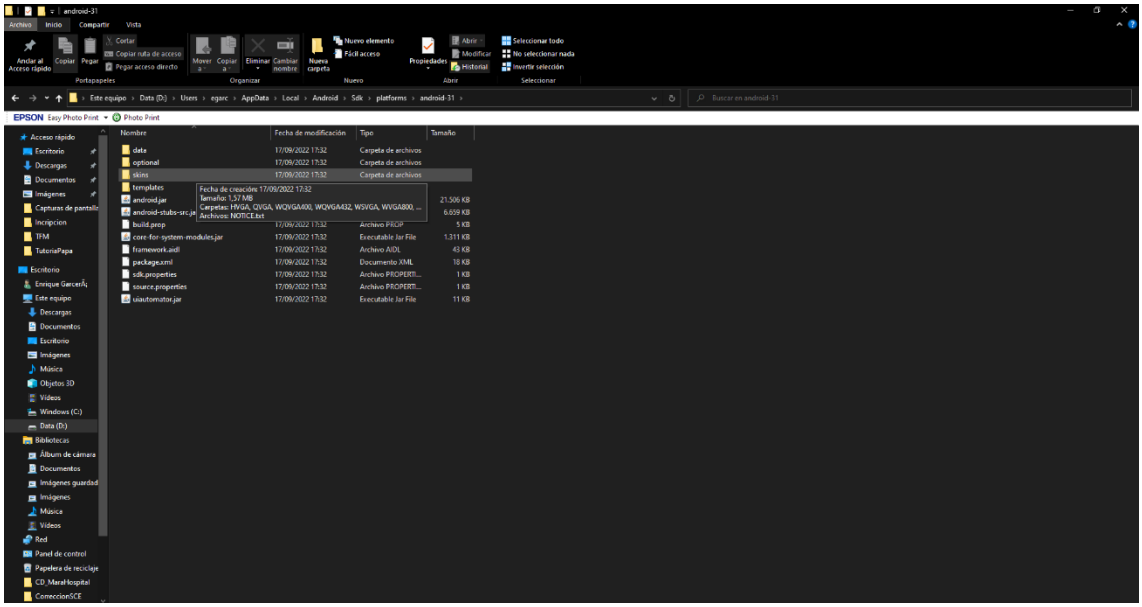


figura 265. Ruta del SDK empleado por el proyecto.

Y finalmente añadimos la dependencia de tipo “jar” Indicando la ruta completa dónde se encuentra “android.jar” dentro de la versión SDK elegida.

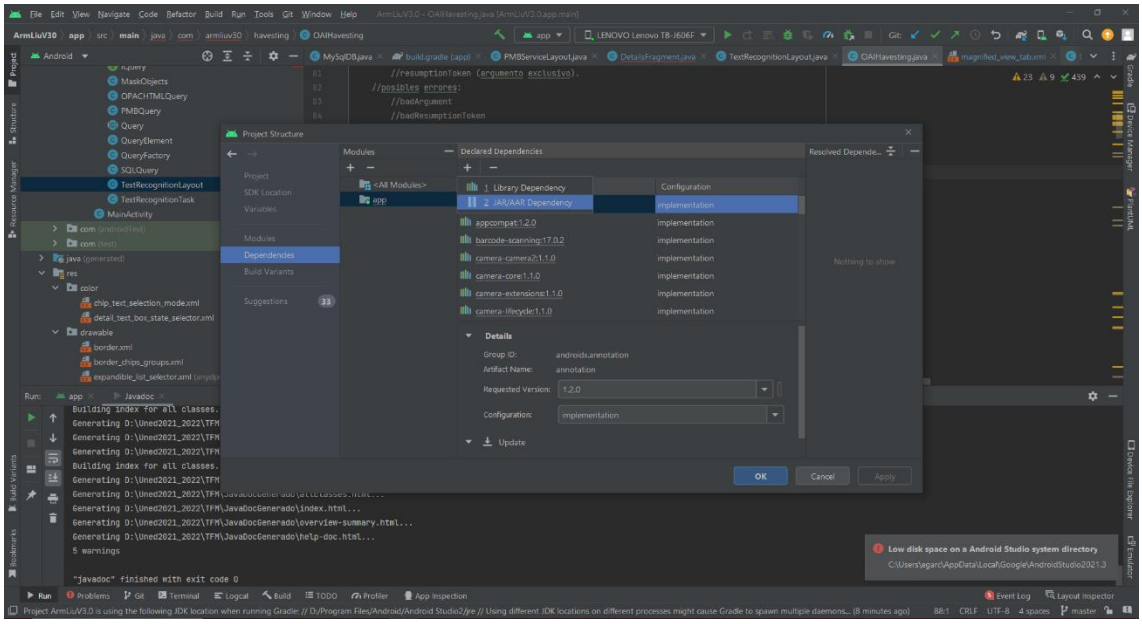


figura 267. Inclusión de la dependencia para que funcione javadoc.

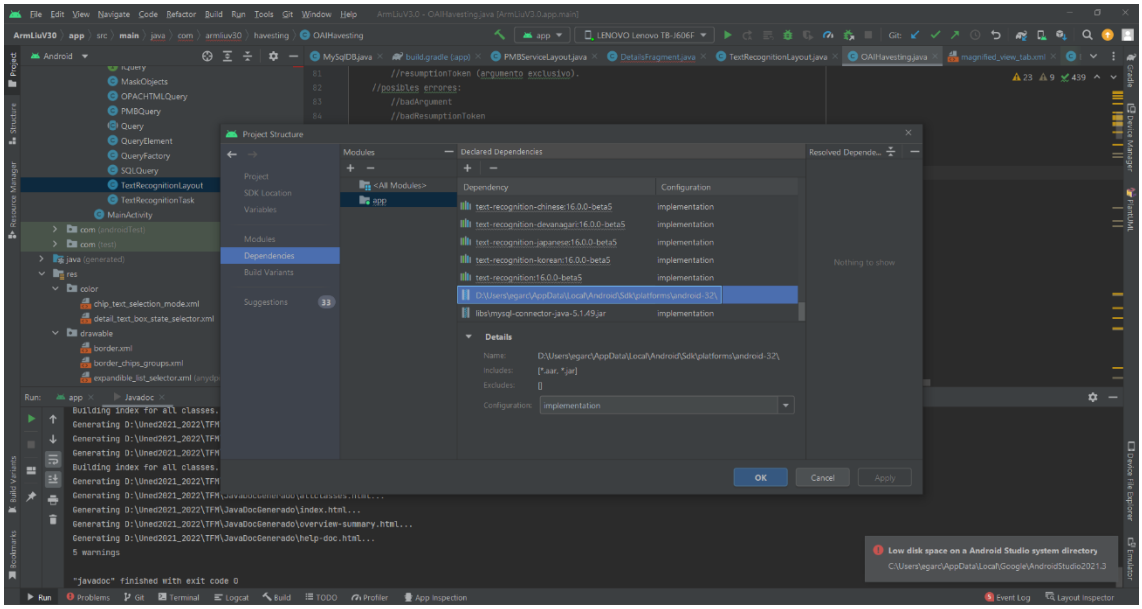


figura 266. Inclusión de la dependencia para que funcione javadoc. Paso 2.

13.2.3 Directrices de Google.

En la medida de lo posible hemos seguido la directriz de Google “Material Design for Android” como guía para el diseño visual, interactivo y de movimiento (Google Inc. Guías., s.f.), (Google Inc. Especificación Material Design., s.f.).

- Se ha empleado una barra de apps o de acciones con vistas a seguir las directrices de Google y con el objetivo principal de facilitar al usuario las acciones y forma de configurar la aplicación de una forma homogénea a como está acostumbrado para otras aplicaciones Android. Por ejemplo, en Android es común mostrar tres puntos en la barra de acciones para desplegar el menú de configuración y el uso del widget “Toolbar” (ActionBar ha sido reemplazado gradualmente por Toolbar que homogeniza el comportamiento para todos los dispositivos Android).
- Se han empleado un conjunto de los componentes presentes en la especificación de Material Design como por ejemplo las “Cards” y los “Chips” que ofrecen una interfaz más actualizada frente a sus antecesores.

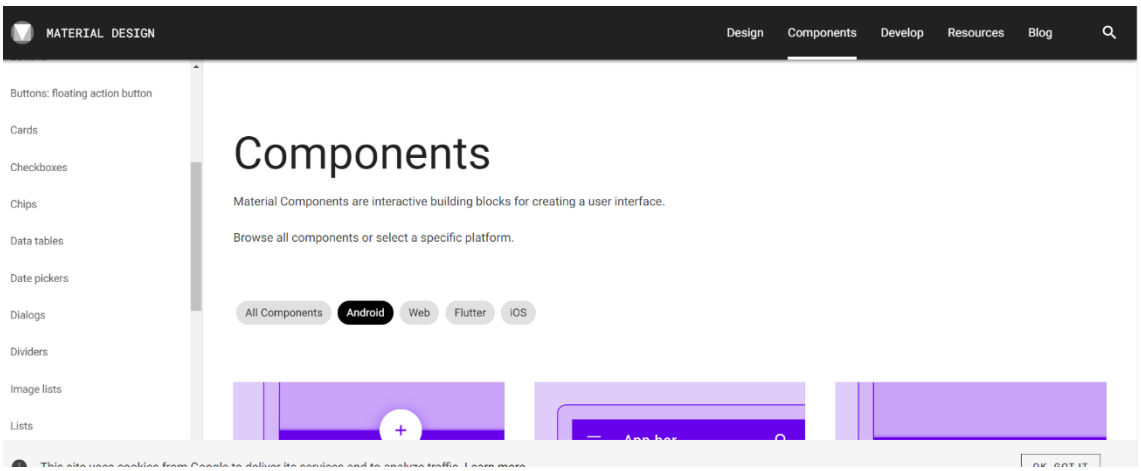


figura 268. Especificación de Material Design.

13.2.4 Base de datos SQLite local al dispositivo.

En la figura 269, podemos observar la consulta de creación de la tabla (presente en la clase DublinCoreDB) en la base de datos SQlite local.

```
//consulta de la creación de la tabla en caso de que todavía no exista. La existencia o no la gestiona el haber heredado de SQLiteOpenHelper
private static final String SQL_CREATE_QUERY="CREATE TABLE " + TABLE_NAME + "(" + DublinCoreRecord.ID +" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +
        DublinCoreRecord.TITLE + " TEXT, " +
        DublinCoreRecord.SUBJECT + " TEXT, " +
        DublinCoreRecord.DESCRIPTION + " TEXT, " +
        DublinCoreRecord.SOURCE + " TEXT, " +
        DublinCoreRecord.TYPE + " TEXT, " +
        DublinCoreRecord.RELATION + " TEXT, " +
        DublinCoreRecord.COVERAGE + " TEXT, " +
        DublinCoreRecord.CREATOR + " TEXT, " +
        DublinCoreRecord.PUBLISHER + " TEXT, " +
        DublinCoreRecord.CONTRIBUTOR + " TEXT, " +
        DublinCoreRecord.RIGHTS + " TEXT, " +
        DublinCoreRecord.DATE + " TEXT, " +
        DublinCoreRecord.FORMAT + " TEXT, " +
        DublinCoreRecord.IDENTIFIER + " TEXT, " +
        DublinCoreRecord.LANGUAGE + " TEXT)";
```

figura 269. Consulta de creación de la tabla local para almacenar los datos cosechados del servidor OAI.

Con esta consulta nos podemos hacer una idea de los campos presentes en dicha tabla y que se corresponde con los presentes en cada uno de los registros en formato DC recuperados del servidor.

En la siguiente tabla se relacionan estos campos con una pequeña descripción de cada uno de ellos.

Nombre columna.	Descripción.
Id	Identificador asignado al insertar una fila en la tabla.
Title	Título del recurso obtenido del servidor.
Subject	Claves o frases que describen el recurso.
Description	Descripción textual del recurso.
Source	Fuente de la que proviene el recurso.
Type	Tipo de recurso (poema, diccionario, etc.)
Relation	Contiene un identificador de otro recurso con el que se relaciona.
Coverage	Cobertura espacial o temporal del recurso.
Creator	Autor/creador del recurso.
Publisher	Entidad responsable de hacer que el recurso se encuentre disponible.
Contributor	Colaboradores secundarios.
Rights	Derechos de autor. Suele ser una URL.
Date	Fecha en la que el recurso se puso a disposición del usuario.
Format	Formato de datos de un recurso.
Identifier	Identificador del recurso.
Language	Idioma o idiomas del contenido intelectual del recurso.

Tabla 2. Tabla DublinCore de la base de datos local SQLite.

13.3 Base de datos Bibli de PMB.

En el presente apartado, se describen brevemente las tablas junto a los atributos presentes en las mismas más relevantes de la base de datos “Bibli” sobre la que se apoya PMB dado que es de esta base de datos de la que estamos obteniendo la información que mostramos en nuestra aplicación.

Las dos tablas más importantes para nuestro contexto son la tabla “notices” y la tabla “exemplaires”. La primera se corresponde con los registros bibliográficos, y la segunda con los ejemplares asociados. Sin embargo, existe multitud de información complementaria que se encuentra referenciada desde dichas tablas. Debido a que tras analizar las diferentes tablas presentes en la base de datos junto a sus registros de ejemplo no conseguíamos obtener la información necesaria sobre la relación entre estas, el apartado de préstamos, los tipos de usuario, etc. Decidimos investigar el código fuente de PMB, dado que está escrito en lenguaje PHP y tras la instalación de PMB, este es accesible en el servidor en formato de ficheros que contienen texto plano. Esta labor nos permitió crear las diferentes “selects” necesarias para que nuestra aplicación funcione.

En nuestra clase PMBQuery nos encontramos con la composición de la “select” sobre la tabla “notices” complementada con información necesaria de otras tablas como los autores, editorial, etc.

```
String sqlQuery=
"select
notices.notice_id,notices.code,notices.n_resume,notices.npages,notices.size,no
tices.n_contenu,notices.index_matieres,notices.tit1,GROUP_CONCAT(author_name,
',') as authors_name,  GROUP_CONCAT(author_rejete, ',') as authors_rejete,
GROUP_CONCAT(ed_name,',') as ed_names from notices LEFT JOIN " +
        "(SELECT * FROM publishers,authors,responsability) as extradata ON
extradata.responsability_notice = notices.notice_id and (notices.ed1_id =
extradata.ed_id OR notices.ed2_id = extradata.ed_id) AND
extradata.responsability_author = extradata.author_id " +
        sqlWhere + " " +
        "GROUP BY
notices.notice_id,notices.code,notices.n_resume,notices.npages,notices.size,no
tices.n_contenu,notices.index_matieres,notices.tit1 " +
        "ORDER BY notices.notice_id";
```

Tabla 3. Select sobre la tabla notices de la base de datos Bibli.

Atributo	Descripción.
notice_id	Id asignado en la tabla notices.
tit1	Título uno. Disponemos de cuatro títulos (tit1, tit2, tit3, tit4) en la tabla notices pero solo hemos empleado el primero.
code	ISBN
npages	Número de páginas.
size	Tamaño.
n_contetu	Contenido.
n_resume	Resumen.
index_materies	Género.
author_name	En la tabla existe el atributo author_name, con un registro para cada autor. En la select hemos utilizado la instrucción GROUP_CONCAT(author_name, ',') as authors_name, para incluir todos en una celda. Esta instrucción equivale a STRING_AGG de Transact SQL
ed_name	Al igual que con el nombre de autor, en la tabla existe una columna ed_name, pero un registro bibliográfico puede poseer más de una editorial, por lo que realizamos la operación GROUP_CONCAT sobre esta columna.
lien	URL a un recurso externo.

Tabla 4. Atributos más relevantes de la tabla notices.

En nuestra clase PMBServiceLayout nos encontramos con la "select" sobre la tabla "exempleaires", nuevamente complementada con información del resto de tablas de la base de datos bibli como su localización, si es prestable o no, etc.

```
String sqlQuery=
"select expl_id, expl_cb, expl_cote, expl_statut,statut_libelle, expl_typdoc,
tdoc_libelle, expl_note, expl_comment," +
        "expl_section, section_libelle, expl_owner, lender_libelle,
expl_codestat, codestat_libelle," +
        "expl_date_retour, expl_date_depot, expl_note, pret_flag,
expl_location, location_libelle, expl_prix,ifnull(surloc_id,0) as surloc_id,
ifnull(surloc_libelle,'') as surloc_libelle,arc_fin from exemplaires" +
        " left join docs_statut on expl_statut=idstatut" +
        " left join docs_type on expl_typdoc=idtyp_doc" +
        " left join docs_section on expl_section=idsection" +
        " left join docs_codestat on expl_codestat=idcode" +
        " left join lenders on expl_owner=idlender" +
        " left join docs_location on expl_location=idlocation" +
        " left join sur_location on surloc_num=surloc_id" +
        " left join pret_archive on pret_archive.arc_expl_id=expl_id" +
        " where expl_notice=" + idNotice +
        " union  select expl_id, expl_cb, expl_cote,
expl_statut,statut_libelle, expl_typdoc, tdoc_libelle, expl_note,
expl_comment," +
        "expl_section, section_libelle, expl_owner, lender_libelle,
expl_codestat, codestat_libelle," +
        "expl_date_retour, expl_date_depot, expl_note, pret_flag,
expl_location, location_libelle, expl_prix, ifnull(surloc_id,0) as surloc_id,
ifnull(surloc_libelle,'') as surloc_libelle,arc_fin from exemplaires" +
        " left join bulletins on expl_bulletin=bulletin_id" +
        " left join docs_statut on expl_statut=idstatut" +
        " left join docs_type on expl_typdoc=idtyp_doc" +
        " left join docs_section on expl_section=idsection" +
        " left join docs_codestat on expl_codestat=idcode" +
        " left join lenders on expl_owner=idlender" +
        " left join docs_location on expl_location=idlocation" +
        " left join sur_location on surloc_num=surloc_id" +
        " left join pret_archive on pret_archive.arc_expl_id=expl_id" +
        " where bulletins.num_notice=" + idNotice;
```

Tabla 5. Select sobre la tabla exemplaires de la base de datos Bibli

Columnas más relevantes sobre un ejemplar para nuestro contexto.

Atributo	Descripción.
Expl_id	Identificador del registro.
Expl_cb	Código de barras.
Expl_cote	Signatura
Status_libelle	Descripción del estado del documento (consulta solo sala, documento en buen estado, etc.).
Expl_notice	Id de notice con la que se asocia el ejemplar.
Tdoc_libelle	Tipo de medio en el que se encuentra el ejemplar (texto impreso, cd audio, etc.).
Section_libelle	Descripción de la sección dónde se encuentra el ejemplar.
Location_libelle	Descripción del lugar dónde se encuentra el ejemplar.
Pret_flag	Flag que indica si el ejemplar es prestable o no. 1 indica prestable. 0 indica no prestable.
Arc_fin	Fecha fin de préstamo del ejemplar. Nulo en caso de que el ejemplar no esté prestado.

Tabla 6. Atributos más relevantes de la tabla exemplaires.

figura 270. Forma de comprobar si un ejemplar está disponible en función de los datos devueltos por la consulta SQL.

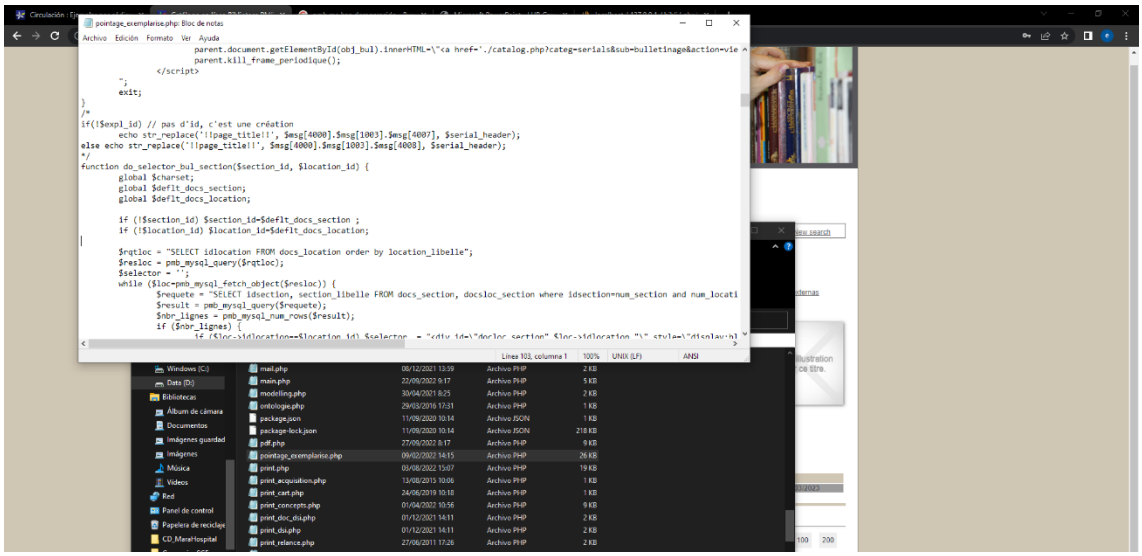


figura 271. Ejemplo uno del proceso de análisis del código fuente de PMB.

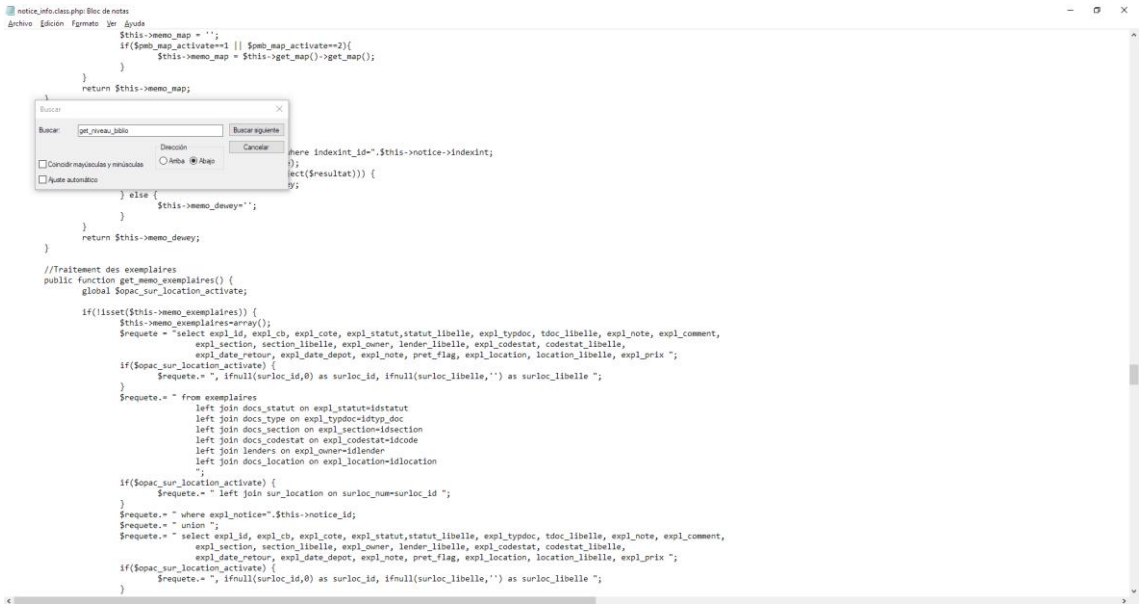


figura 272. Ejemplo dos del proceso de análisis del código fuente de PMB.

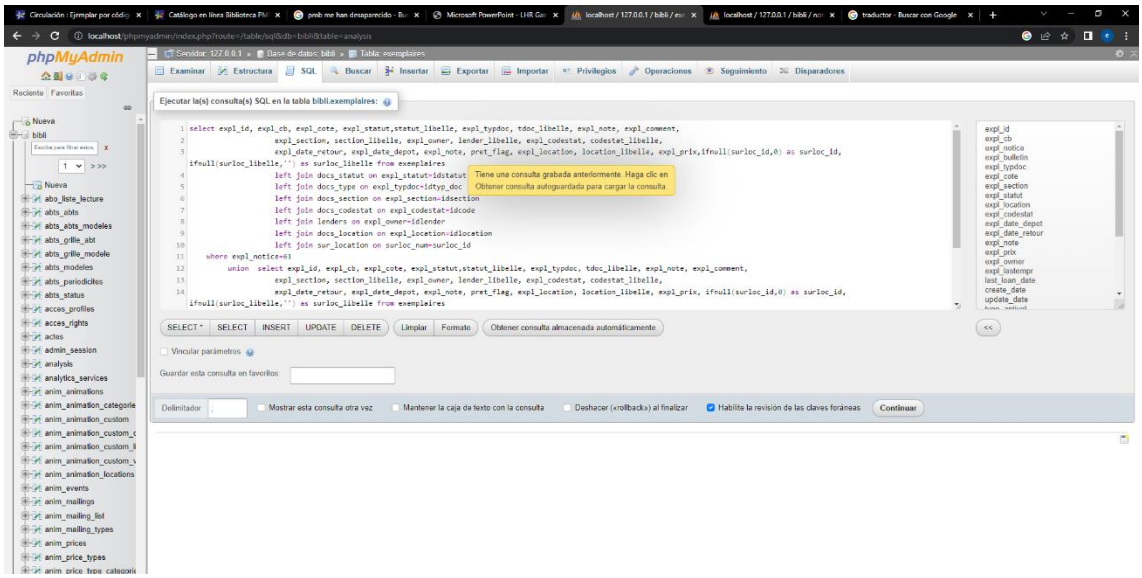


figura 273. Ejemplo de consulta ejecutada sobre la base de datos bibli para comprobar la relación entre tablas y columnas presentes en estas.

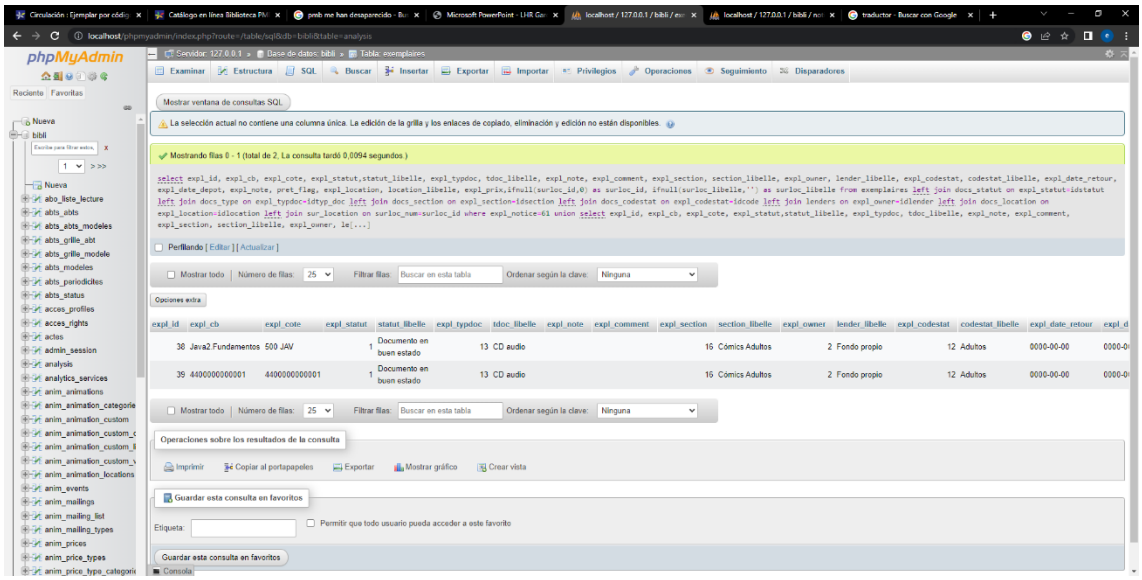


figura 274. Ejemplo de ejecución de consulta sobre la base de datos bibli para comprobar la relación entre tablas y columnas presentes en estas.

13.4 Ingeniería inversa sobre el OPAC de la UNED y el de la UPV.

Pese a que basar la interrogación del catálogo en línea mediante consultas realizadas mediante la URL y la representación del resultado obtenido integrándolo en el componente WebKit de Android embebido en nuestra aplicación puede ser un enfoque muy arriesgado por la probabilidad media de que los patrones de consulta varíen, nos hemos encontrado con la imposibilidad de emplear cualquier otro mecanismos de interacción para consultar el catálogo de la UNED y dado el interés que despierta la explotación del mismo por su extensión y similitud con otros sistemas OPAC HTML, hemos optado por analizar el comportamiento de la URL del catálogo OPAC puesto a disposición mediante el navegador Web.

Confiamos en que esta forma de interrogar perdure como consecuencia de la interrelación de este sistema con otros internos y externos pertenecientes a la corporación UNED.

Los elementos que nos permiten filtrar/obtener la información son los siguientes:

- Posibles campos de búsqueda:
    - Todos, Título, Autor/creador, Materia, Editor, ISBN, ISSN, Asignatura(código)
- Posibles patrones de composición de la Query:
    - Contiene, es (exacto), empieza con.
- Operadores relacionales permitidos:
    - OR, AND y NOT.

Query de ejemplo:

https://buscador.biblioteca.uned.es/primo-explore/search?query=any,contains,casa,AND&tab=tab1&search_scope=TAB1_SCOPE1&sortby=rank&vid=34UNED_VU1&lang=es_ES&mode=advanced&offset=0

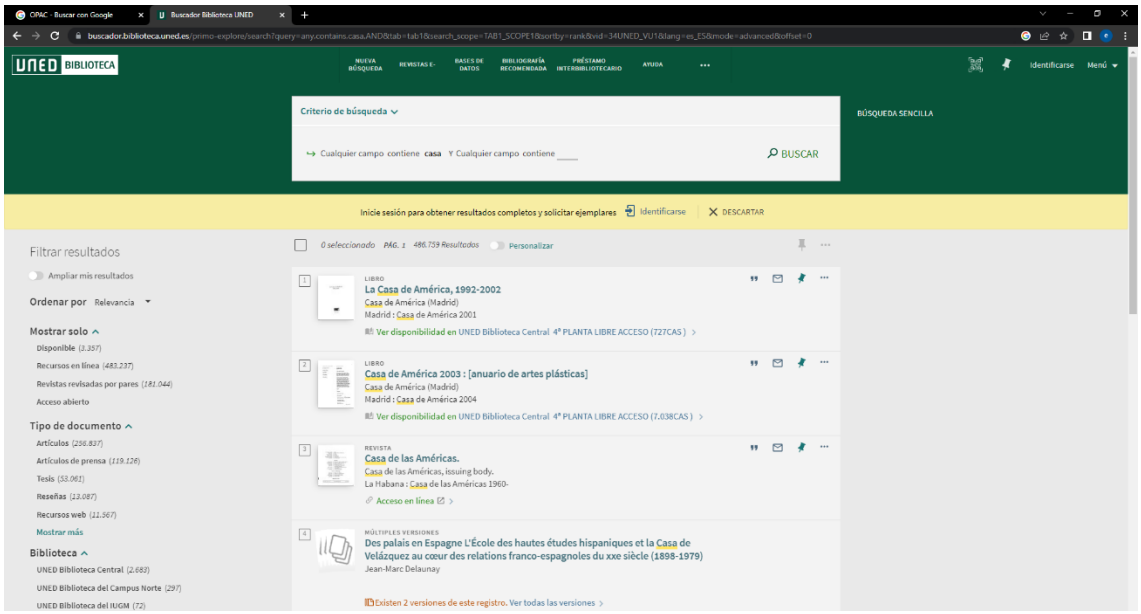


figura 275. Resulta de la consulta realizada sobre el catálogo en línea de la biblioteca de la UNED.

Hemos ido lanzando diferentes consultas al catálogo y observando el comportamiento y el formato de la URL asociada a cada consulta. El resumen de la información obtenida se muestra a continuación:

Las URL's siguen el estándar SRU, donde se usa el patrón clave=valor para proporcionar la información. De la query de ejemplo anterior, podemos extraer la siguiente información:

Servidor de consulta:

https://buscador.biblioteca.uned.es/primo-explore/search?

Consulta:

Utiliza formato: campo de consulta,relación,patrón de búsqueda (separando por comas).

Comienzo de la query → **query=**

Los campos de búsqueda del formulario son reemplazados por las palabras:

Todos→ **any**

Título → **tittle**

Autor/creador → **creator**

Materia → **sub**

Editor → **lsr03**

ISBN → isbn

ISSN → **issn**

Código de asignatura → **lsr01**

Las relaciones son reemplazadas por las siguientes palabras:

Contiene → **contains**

Es(exacto) → **exact**

Empieza con → **with**

Cuando concatenamos varias filas como criterio de búsqueda, las concatena empleando:

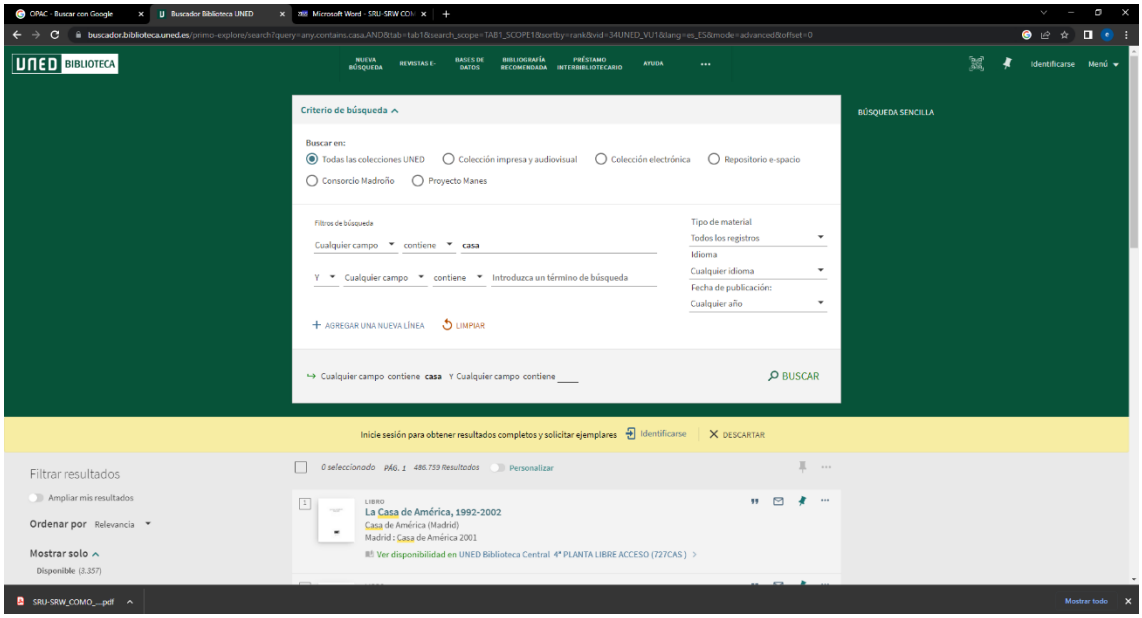


figura 276. Búsqueda avanzada catálogo en línea de la biblioteca de la UNED.

Operador relacional & query=….. y después de query lo descrito anteriormente.

El remplazo de las palabras correspondiente a los diferentes operadores relacionales para su uso en el formulario es el siguiente:

OR → OR

AND → AND

NOT → NOT

Ejemplo:

https://buscador.biblioteca.uned.es/primo-explore/search?query=title,begins_with,casa,NOT&query=title,contains,casa%202,AND&tab=tab1&search_scope=TAB1_SCOPE1&sortby=title&vid=34UNED_VU1&lang=es_ES&mode=advanced&offset=0

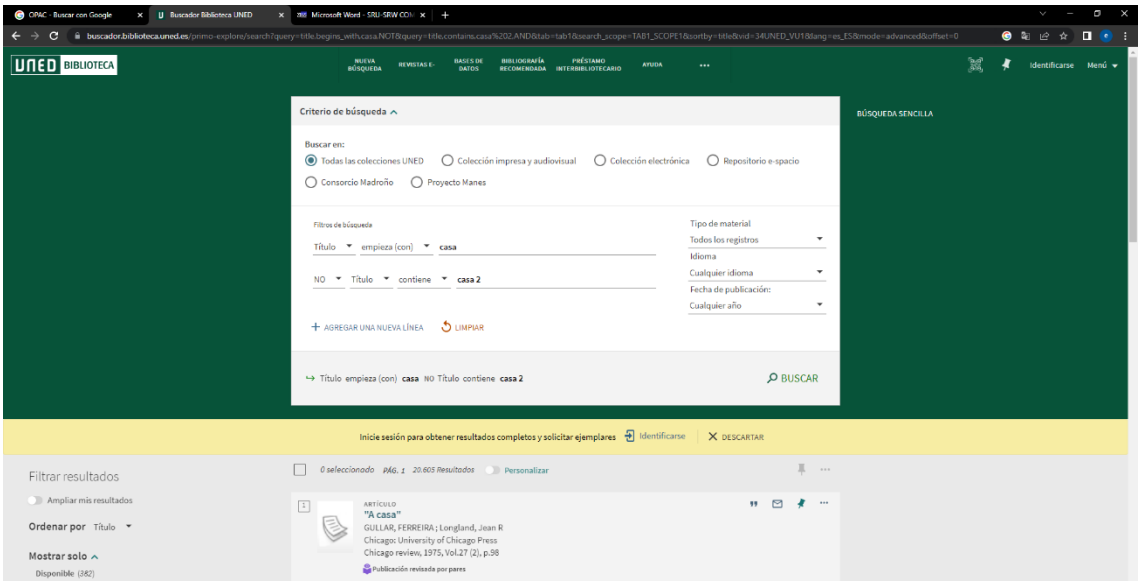


figura 277. Consulta sobre el catálogo en línea de la biblioteca de la UNED.

La URL de consulta finaliza siempre con:

,AND&tab=tab1&search_scope=TAB1_SCOPE1&sortby=title&vid=34UNED_VU1&lang=es_ES&mode=advanced&offset=0

- El parámetro offset=x nos indica la página en la que estamos. Va de 10 en 10 y se corresponde con n-1. Página 1 → offset=0, Página 2 → offset=10, Página 3 → offset=20, etc.
- El par clave valor tab=tabX, se corresponde con el filtro del catálogo sobre el que buscar:
    - Posibles valores de tab:
        - tab1 → Todas las colecciones de la UNED.
        - tab2 → Colección impresa y audiovisual.
        - tab3 → Colección electrónica.
        - tab4 → Repositorio e-spacio.
        - tab5 → Consorcio Madroño.
        - tab6 → Proyecto Manes.
- Search_scope va relacionado con el parámetro tab. Cuando variamos tab el valor de search_scope varía de la siguiente manera
    - Posibles valores de search_scope: Tab=tab1 → &search_scope=TAB1_SCOPE1, Tab=tab2 → &search_scope=TAB2_SCOPE1, etc.
- La parte final de search_scope (SCOPEX) sirve para representar los resultados de un modo u otro. Por ejemplo, al poner TAB1_SCOPE2 obtenemos la misma pantalla pero con un “toggle buttón” “Ampliar mis resultados” además nos indica que nos identifiquemos, por lo que entendemos que se realizará parte de una búsqueda interna que no sea de acceso libre.
- “sortby” se corresponde con el campo de ordenación de los resultados.
- Vid se corresponde con:
    - Lang se corresponde con el idioma del formulario de consulta y los resultados. Castellano → lang=es_ES. Para buscar ejemplares en otros idiomas la consulta añade el parámetro “pfilter” (si no especificamos este parámetro busca en cualquier idioma). Español → AND&pfilter=lang,exact,spa, Inglés → AND&pfilter=lang,exact,eng, Francés → AND&pfilter=lang,exact,fre, Alemán → AND&pfilter=lang,exact,ger, Catalán →

AND&pfilter=lang,exact,cat, Gallego → AND&pfilter=lang,exact,glg, Vasco → AND&pfilter=lang,exact,baq

- Para filtrar por tipo de material, añade nuevamente otro parámetro pfilter indicando el tipo de material. Libros → ,AND&pfilter= pfilter,exact,books, Artículos → AND&pfilter=pfilter,exact,articles, Revistas → AND&pfilter=pfilter,exact,journals, Imágenes → &pfilter=pfilter,exact,images, Audiovisual → AND&pfilter=pfilter,exact,audio_video, Actas de congreso → AND&pfilter=pfilter,exact,conference_proceedings, Tesis → AND&pfilter=pfilter,exact,dissertations, Proyectos final de carrera → AND&pfilter=pfilter,exact,34mad_finaldegreeprojects, Memorias de investigación → &pfilter=pfilter,exact,34mad_finaldegrees
- "mode" se corresponde con el tipo de consulta. Consulta avanzada → mode=advanced, Consulta simple → no pone esta clave valor en la URL de consulta.

Pese a que existen más parámetros de filtraje asociados a cómo será conformado el árbol de elementos presente en la parte derecha de la página web de resultados, hemos optado por plantear un análisis más exhaustivo como propuesta de mejora asociada a la evolución del proyecto. De momento que hemos optado por centrarnos en los siguientes campos de búsqueda:

ISBN, Título, Editor y Autor, que en un principio son los que soportará nuestra aplicación como elemento homogeneizador de búsqueda en diferentes sistemas (OPAC HTML, Servidor PMB sobre base de datos, etc.).

Hemos observado algún otro OPAC HTML cómo el presente en la Universidad Politécnica de Valencia (UPV). Y hemos podido comprobar, como este servidor sigue casi el mismo patrón de comportamiento que el de la UNED cambiando el nombre de los catálogos y de algunos campos de búsqueda.

Si lanzamos la siguiente consulta sobre el Poli Buscador:

https://polibuscador.upv.es/discovery/search?query=title,contains,inform%C3%A1tica,AND&tab=BUS_GENERAL&search_scope=MyInst_and_CI&vid=34UPV_INST:bibupv&mode=advanced&offset=0

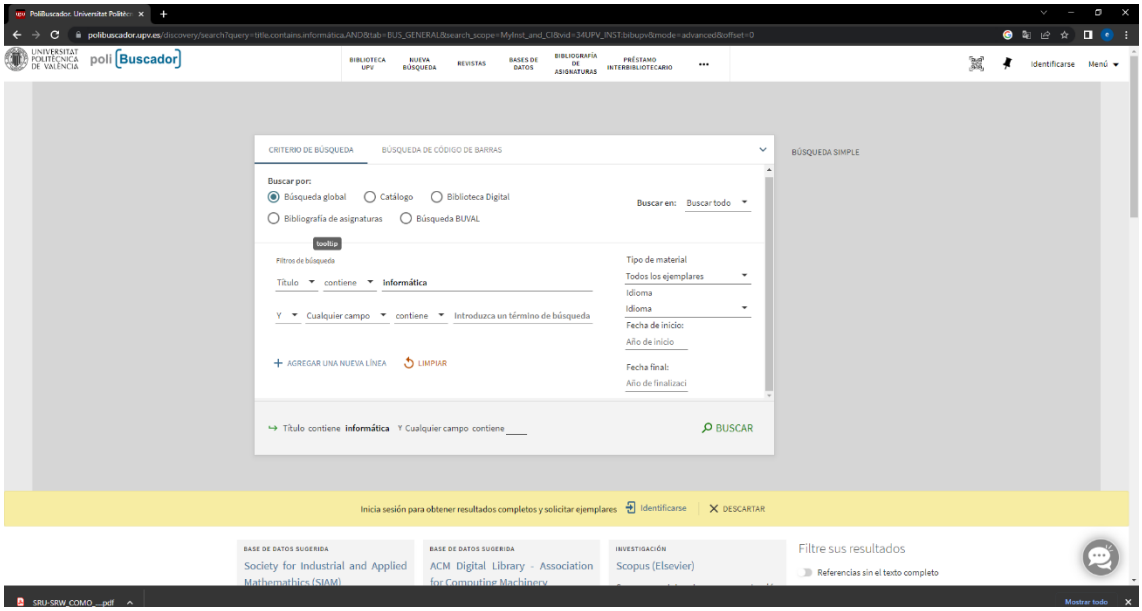


figura 278. Ejemplo de consulta sobre el catálogo en línea de la biblioteca de la UPV.

Obtenemos los resultados de la figura 278. Podemos observar que para una búsqueda simple podríamos extrapolar el análisis realizado para el servidor OPAC de la UNED, siendo necesario adaptar:

Servidor:

https://polibuscador.upv.es/discovery/search?

Lugar de búsqueda:

Búsqueda global → tab=BUS_GENERAL

Catálogo → tab=CATALOGO

Bib Digital → tab=BIB_DIGI

Bibliografía de Asignaturas → tab=BIB_RECOMENDADA

Asociado a la búsqueda general que será la adoptada:

search_scope=MyInst_and_CI

vid=34_UPV_INST:bibupv

En el presente proyecto nos centraremos en la explotación del catálogo de la UNED, parametrizando al máximo posible mediante propiedades de la aplicación con vistas a que sea posible la adaptación a otros repositorios con los mínimos cambios posibles.

13.5 Servidores OAI

Con vistas a familiarizarnos con este tipo de servidores y los diferentes formatos que soportan, hemos estado lanzando una serie de consultas de ejemplo sobre el servidor OAI de la BNE a la par que avanzábamos en la casuística particular de cada uno de los formato soportados y los diferentes mecanismos de interacción. Como formatos soportados por los diferentes servidores, los más comunes son MarcXML y Dublin Core. En nuestro caso nos hemos centrado en el último de ellos y los campos presentes en cada uno de los registros de información que recuperemos del servidor en dicho formato, los podemos consultar en la Tabla 2, dónde, para explicar los datos que se almacenan de forma local al obtener interrogar el servidor OAI, se detallaba cada uno de ellos. Una vez aclarada la estructura de los datos a manejar, procederemos a explicar brevemente como interactuar con un servidor OAI.

Denotar que los servidores OAI se basan en el protocolo OAI-PMH y que podemos encontrar gran cantidad de información en (Open Archives Initiative., 2023). También que la BNE nos ofrece los datos de acceso a su servidor OAI en (Biblioteca Nacional de España, 2023) (figura 279).

Para interactuar sobre un servidor OAI nos apoyamos en lo que se conoce como verbo. Al indicar la palabra “verb” en la URL seguido del verbo a emplear, le estamos indicando al servidor que acción queremos realizar sobre el mismo. Cada uno de los posibles verbos se detallan en la Tabla 7.

verb	Descripción.
Identify	Nos proporciona información del archivo (servidor OAI).
ListMetadataFormats	Nos proporciona los formatos de metadatos soportados por el servidor.
ListSets	Nos proporciona el listado de grupos disponibles en el servidor.

ListIdentifiers	Nos proporciona solo las cabeceras de los registros. Estas incluyen el identificador de cada uno de ellos.
ListRecords	Nos proporciona los registros completos dentro de un repositorio.

Tabla 7. Verbos disponibles en un servidor OAI.

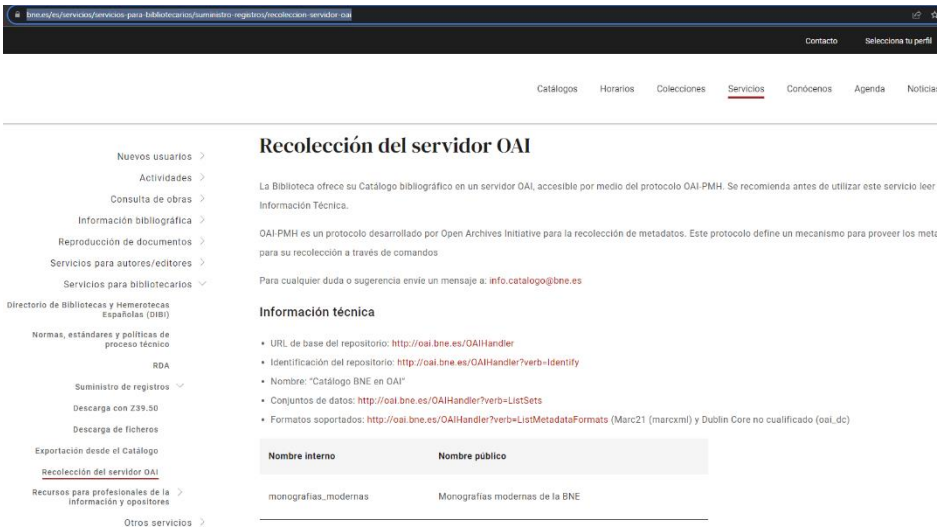


figura 279. Datos de acceso al servidor OAI de la BNE.

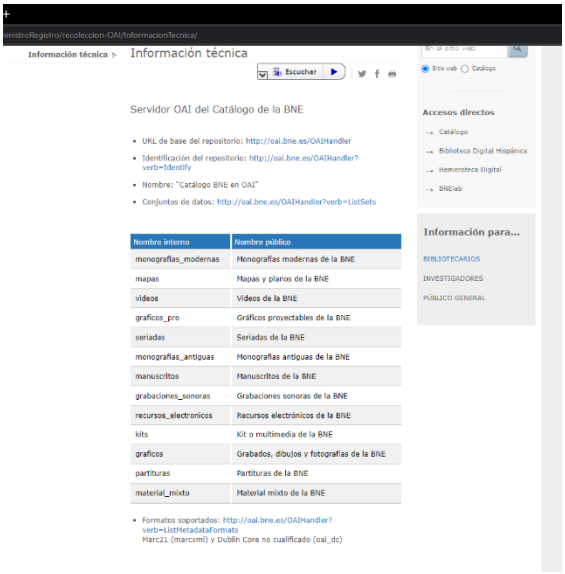


figura 280. ListSpec sobre la BNE. Grupos de archivos disponibles.

Si lanzamos la siguiente consulta de ejemplo sobre el servidor de la BNE:

http://oai.bne.es/OAIHandler?verb=ListRecords&set=monografias_antiguas&metadataPrefix=marcxml

Obtendremos un documento XML con estructura MarcXML (figura 281) y si analizamos un poco los datos obtenidos, podremos comprobar como dentro de las etiquetas <ListRecords> </ListRecords> que definen dicha sección, aparece el conjunto de registros delimitados por las etiquetas <record></record> para cada uno de ellos.

También es importante observar el formato de la consulta, dónde, por un lado, le indicamos el verbo “ListRecords” por lo que pretendemos que nos devuelva los registros completos en lugar de solo las cabeceras con los identificadores (para tal fin habríamos utilizado ListIdentifiers), además le hemos indicado el conjunto de datos por el que queremos filtrar mediante el par clave-valor “set=monografías_antiguas” y por último le hemos indicado el formato deseado (marcxml).

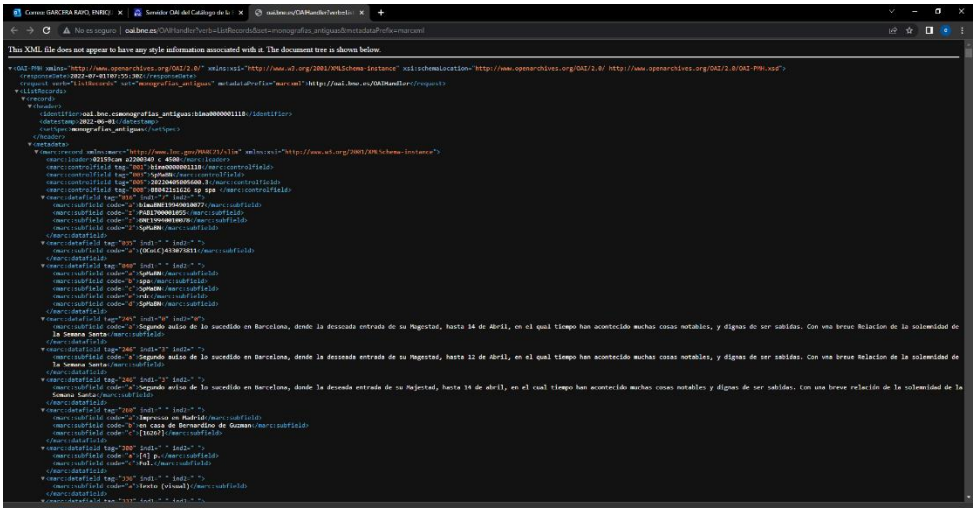


figura 281. Resultado de consulta sobre el servidor OAI de la BNE en formato marcxml.

Para facilitar nuestra labor de aprendizaje del modo de proceder de un servidor OAI, nos apoyamos en la herramienta “Podman”, que nos permitió lanzar solicitudes al servidor y analizar los resultados de una forma más cómoda.

En la figura 282, podemos observar un ejemplo de consulta que lanzamos al servidor en la que le indicamos que queremos que nos devuelva los formatos que soporta. En el parámetro “verb”, le indicamos “ListMetadataFormats” y del resultado obtenido podemos observar que solo soporta marcxml (Formato Marc) y oai_dc (Formato DC).

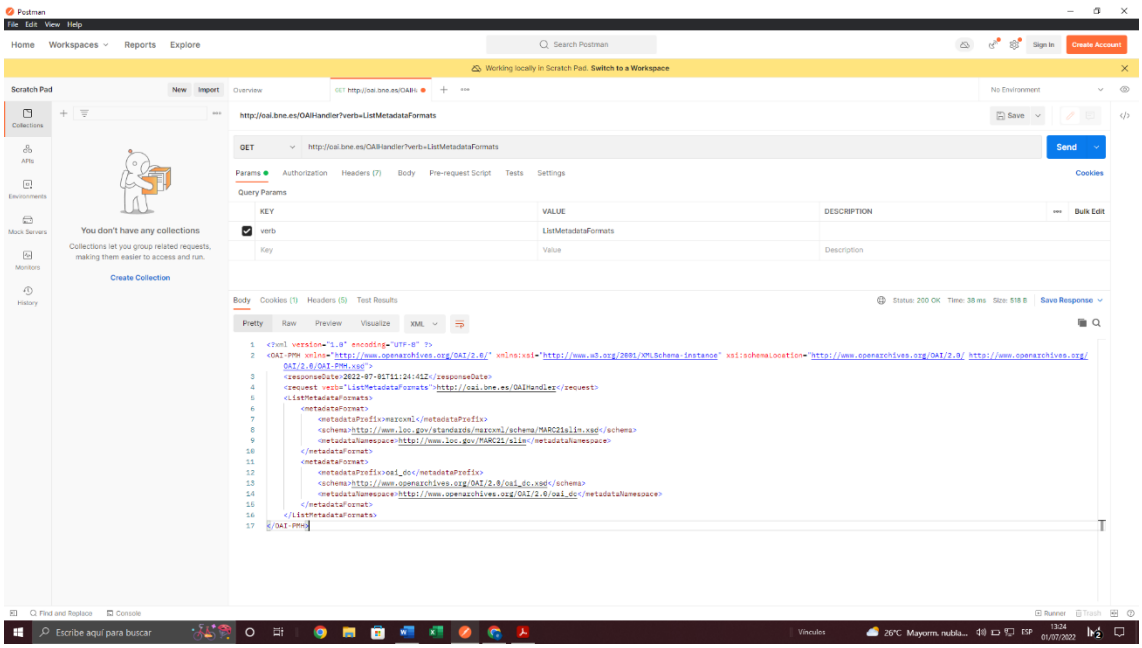


figura 282. Consulta de formatos soportados por servidor OAI, lanzada con la herramienta Podman.

Si lanzamos otra consulta empleando como parámetro “verb” el valor “Identify”, nos proporcionará información sobre el servidor OAI.

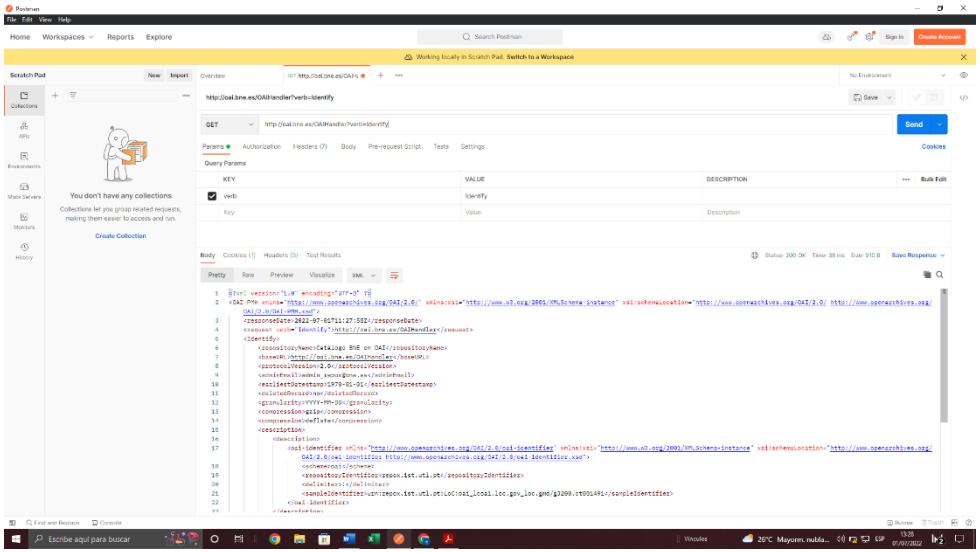


figura 283. Resultado de consulta sobre servidor OAI. Información sobre el servidor.

Dado que un manual detallado en cuanto al protocolo OAI-PMH se escapa de las pretensiones del presente documento, se insta a la consulta de la bibliografía recomendada.

14 Conclusiones.

Podríamos comenzar este apartado con una reflexión demoledora sobre como el tiempo poco a poco nos ha ido dando un gélido baño de realidad. Aquellas expectativas iniciales poco a poco se han ido tornando poco realistas, en primer lugar, por subestimar el tiempo necesario para ponerlas en práctica y en segundo, porque no todo camino que parece que llega a un sitio finalmente llega a él. Pese a que no nos hemos ido tanto de los objetivos marcados en la propuesta inicial del TFM, en cierto modo llegamos a la conclusión de que los usuarios que son un poco reacios al uso de determinadas herramientas informáticas, no están del todo faltos de razón. Desde una edad muy temprana sentimos pasión por la tecnología en general y por la informática y la electrónica en particular y nos invade la emoción al ver cómo han evolucionado estas en unas pocas décadas, pero, ¿Tiene sentido tanto sufrimiento y horas de dedicación para poner en funcionamiento una herramienta que debería de funcionar sin más y de la que dependemos?. En la memoria se han intentado reflejar algunos de los problemas encontrados para contribuir a su rápida solución por parte de otras personas.

Sin embargo, y a pesar de lo expuesto en el párrafo anterior, estamos realmente orgullosos de los logros conseguidos. Podrían ser distintos, pero, ha habido muchísimo esfuerzo por el camino y la satisfacción de los logros conseguidos y la ilusión de que existen vías futuras que abordar, nos genera una agradable sensación.

También hemos llegado a la conclusión de que los equipos multidisciplinares son vitales en determinados contextos. En este en concreto, un especialista en Biblioteconomía habría sido de agradecer. Pese a que hemos intentado suplir nuestras carencias con la búsqueda en la red y las consultas realizadas a terceras personas, por ejemplo, al visitar las diferentes bibliotecas, el escaso conocimiento que poseemos en este momento, nos habría ahorrado mucho tiempo y esfuerzo.

14.1 Logros alcanzados.

- Hemos elaborado la presente memoria que documenta aquellos aspectos más relevantes en cuanto al trabajo realizado durante todo este tiempo y dónde tras su lectura se dejan entrever los logros y las vías futuras.
- Se ha realizado un trabajo de campo visitando diferentes bibliotecas y buscando extensa información en la red para descubrir los principales sistemas de clasificación y catalogación empleados y entendiendo que esto no es algo trivial.
- Hemos investigado sobre los diferentes sistemas que permiten la consulta en línea de los catálogos bibliotecarios. Entrando en detalle sobre la infraestructura software empleada y los principales protocolos de comunicación y tomando la decisión sobre cual/es de ellos adoptaríamos para nuestro desarrollo y pruebas.
- Hemos intentado encontrar herramientas que empleen la realidad aumentada en el contexto bibliotecario.
- Hemos diseñado, desarrollado y probado una aplicación que se puede ejecutar en dispositivos cuyo S.O. sea Android, intentando

seguir las mejores prácticas y la herramienta oficial proporcionada por Google (Android Studio) y el kit de AA MLKit entre otras.

- Hemos montado una infraestructura de desarrollo y pruebas, tanto a nivel software como a nivel físico. Por un lado, instalando un servidor local de PMB con el que interactuar como si de uno remoto se tratase, y por otro creando una librería de pruebas en la que hemos ubicado diferentes ejemplares bibliográficos reales después de su clasificación y catalogación. Hemos introducido los datos de estos ejemplares en el servidor de PMB local.
- Hemos conseguido conectar nuestra aplicación con servidores OPAC accediendo al catálogo solo accesible desde el navegador web.
- Hemos conseguido cosechar datos del servidor OAI de la BNE almacenando un subconjunto de estos en la base de datos local del dispositivo sobre el que se ejecuta nuestra aplicación y aprendiendo la forma de proceder.

14.2 Líneas de trabajo futuro.

Las líneas de trabajo futuro están estrechamente relacionadas con los logros conseguidos. Se expone un listado de aquellas que consideramos más relevantes.

- La aplicación desarrollada no debería de “quedarse en un cajón” y en algún momento debería de ponerse en producción en el “Play Store”. Sin embargo, y a hilo de lo comentado en la introducción de las conclusiones, no son los usuarios los que deberían de probar y depura la aplicación, por lo menos si no son usuarios que de manera intencionada han decidido hacerlo. Es por ello, que, por limitaciones de tiempo, la aplicación no se encuentra lo suficiente madura para su puesta en producción y debería ser depurada de una forma más profunda.

- La aplicación debería de ser probada en un contexto real y dado que los centros educativos de la Comunidad Valenciana utilizan PMB, esta podría ser una buena opción.
- Para la consulta sobre PMB, deberían ampliarse las posibilidades de búsqueda, añadiendo determinados campos asociados a los ejemplares bibliográficos como la signatura o el código de barras creado en PMB.
- Se plantea añadirle a la aplicación un lector de códigos de barras dado que es algo que el Kit de desarrollo soporta.
- Las funcionalidades de la aplicación podrían ampliarse añadiendo la opción de reserva, ampliación del préstamo de un ejemplar, etc.
- Se plantea explorar otras vías de interacción con el servidor PMB como la interacción mediante SOAP (Simple Object Access Protocol), que pese a estar disponible en PMB, no conseguimos hacerla funcionar.
- Se plantea perseguir el problema por el que al consultar sobre el catálogo de la UNED empleando WebKit, se muestra una franja blanca en la cabecera de los resultados.

15 Lista de referencias y bibliografía,

Apache Solr. (2023). *Solr. Motor de búsqueda.* Obtenido de https://solr.apache.org/

Apache Web Server. (2023). *Servidor Web de Apache*. Obtenido de https://httpd.apache.org/

Baratz. Desarrollo e Implentación de Software para Bibliotecas. (2023.). *Soluciones Software para Bibliotecas.* Obtenido de https://www.baratz.es/software-para-bibliotecas/

Biblioteca Nacional de España. (2023). *Datos de acceso al servidor OAI.* Obtenido de https://www.bne.es/es/servicios/servicios-para-bibliotecarios/suministro-registros/recoleccion-servidor-oai

Biblioteca Nacional de Madrid. (2004). *Formato IBERMARC para registros de fondos y localizaciones.* Recuperado el 2023, de https://www.bne.es/sites/default/files/repositorio-archivos/Registros_Fondos_IBERMARC%5B1%5D.pdf

Bibloteca Nacinal de España. (1999). *Reglas de catalogación de la Biblioteca Nacional de España. Catalogación vigente.* Recuperado el 2023, de http://m.bne.es/gl/Inicio/Perfiles/Bibliotecarios/NormasNacionales/ReglasDeCatalogacion/

Bibloteca Nacional de España. Marc 21. (2022). *Marc 21 para información de fondos.*

Ex-ilibs. (s.f.). Obtenido de https://exlibrisgroup.com/es/productos/alma/

Exlibris Group. (2023). *Exlibris Primo.* Obtenido de https://exlibrisgroup.com/es/productos/primo/

Exlibris Group. Desarrolladores. (2023). *API de desarrollo. Exlibris.* Obtenido de https://developers.exlibrisgroup.com/alma/apis/

Google Inc. Especificación Material Design. (s.f.). *Especificación de cada uno de los elementos presentes en Material Design.* Obtenido de https://m2.material.io/design

Google Inc. Guías. (s.f.). *Guías de diseño. Material Desing.* Obtenido de https://developer.android.com/guide/topics/ui/look-and-feel?hl=es-419

Google. Android Studio. (s.f.). *Página principal del IDE Android Studio.* Obtenido de https://developer.android.com/studio

Illinois Library. (2017). *Realidad Virtual. Biblioteca de Illinois.* Retrieved from https://www.library.illinois.edu/leitc/virtual-reality/

Illinoys University Library. (2023). *Equipamiento necesario para interactuar con sus sistemas de Realidad Aumentada.* Obtenido de https://uiuc.libcal.com/equipment?lid=5488

Koha. Servicios Web. (5 de 5 de 2023). *Koha. Servicios Web Disponibles.* Obtenido de https://koha-community.org/manual/22.11/es/html/webservices.html

Library Of Congress. (2023). *Librería del Congreso. Página principal.* Obtenido de https://www.loc.gov/about/informacion-general/

Open Archives Initiative. (2023). *Página oficial de la organización.* Obtenido de https://www.openarchives.org/

Oracle MySQL. . (2023). *Sistema de Gestión de Bases de Datos MySQL.* Obtenido de https://www.mysql.com/

PHP Group. (2023). *Lenguaje de programación PHP*. Obtenido de https://www.php.net/

PHP YAZ. (2023). *Instalación de Yaz php.* Obtenido de https://www.php.net/manual/en/ref.yaz.php

PMB Services. (2023). *Página principal de PMB Services.* Obtenido de https://www.sigb.net/

Pressman., R. S. (2001). *Ingeniería del Software. Un enfoque práctico.* Mc Graw Hill.

SIGB. . (s.f.). *Instrucciones de instalación de PMB.* Obtenido de http://pmb.ac-noumea.nc/doc_user/sigb/es_ES/html-install/ch01s02.html

SIGB. Descarga PMB. (s.f.). *Página principal de pmb.* Obtenido de www.sigb.net

SIGB. Descarga PMB. (s.f.). *Página principal de pmb.* Obtenido de www.sigb.net

Universal Decimal Classification Consortium. (2023). *Universal Decimal Classfication Summary.* Recuperado el 2023, de https://udcsummary.info/php/index.php?lang=es&pr=Y

Universidad de Alicante. (s.f.). *Identificadores bibliográficos.* Obtenido de https://publicaciones.ua.es/p/3827_identificadores-bibliograficos/

Universidad de León. (2023). *Ayuda para la localización de ejemplares en las estanterías. Clasificación mediante CDU.* Obtenido de https://bibliotecas.unileon.es/ciencias-economicas-empresariales/2012/12/12/una-ayuda-para-localizar-ejemplares-en-las-estanterias/

Vázquez., N. A. (junio de 2016). *Experiencias de realidad aumentada en blibliotecas. Estado de la cuestión.* Obtenido de https://bid.ub.edu/es/36/arroyo.htm

Zotero. Importación de Citas. (2023). *Importación de Citas Bibliográficas en Zotero.* Obtenido de https://www.zotero.org/support/plugins

16 Listado de siglas, abreviaturas y acrónimos.

AA	Aprendizaje automático.
AENOR	Asociación Española de Normalización y Certificación.
API	Application Programming Interface.
app	Aplication.
BNE	Biblioteca Nacional de España.
CB	Código de Barras.
CDU	Clasificación Decimal Universal.
CSIC	Consejo Superior de Investigaciones Científicas.
DC	Dublin Core.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IDE	Entorno de desarrollo integrado.
INAP	Biblioteca Nacional del Instituto Nacional de Administración Pública.
ML	Machine Learning.
MLKit	Machine Learning Kit.
MySQL	My Structured Query Language.
OAI-PMH	Open Archive Initiative-Protocol for Metadata Harvesting.
OPAC	Online Public Access Catalog.
PHP	Hypertext Preprocessor.
PMB	PhpMyBibli.
RA	Realidad Aumentada.

Rest	Representational State Transfer.
RGB	Red, Green, Blue.
RSS	Really Simple Syndication.
SGBD	Sistema de Gestión de Bases de datos.
SIGB	Sistema Integrado de Gestión de Bibliotecas.
S.O.	Sistema Operativo.
SDK	Kit de Desarrollo de Software.
TFM	Trabajo fin de máster.
UML	Lenguaje de Modelado Universal.
UNED	Universidad Nacional de Educación a Distancia.
UPV	Universidad Politécnica de Valencia.
URL	Uniform Resource Locator.
VCS	Sistemas de Control de Versiones.
XAMPP	X, Apache, MaríaDB/MySQL, PHP, Perl.
XML	eXtensible Markup Language.

17 Anexo I. Instalación y configuración de PMB.

Para instalar PMB, en primer lugar, lo hemos descargado de la página oficial (SIGB. Descarga PMB., s.f.) (SIGB. Descarga PMB., s.f.) y hemos seguido las instrucciones de instalación (SIGB. , s.f.).

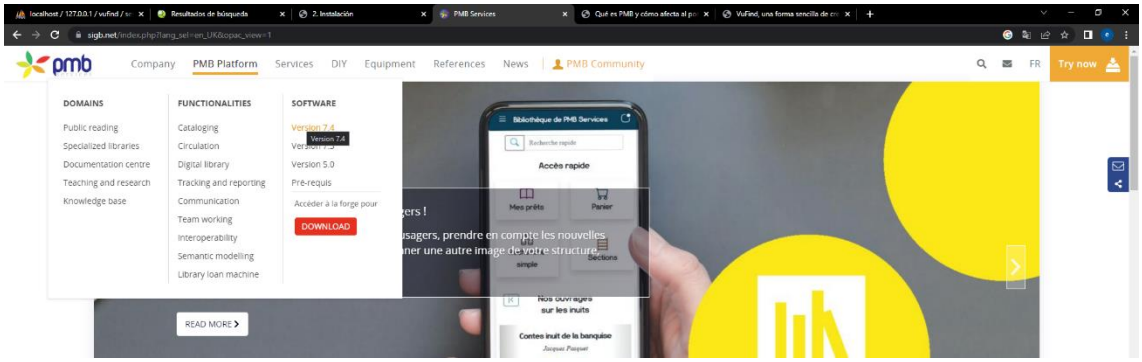


figura 284. Página oficial de PMB. "sigb.net".

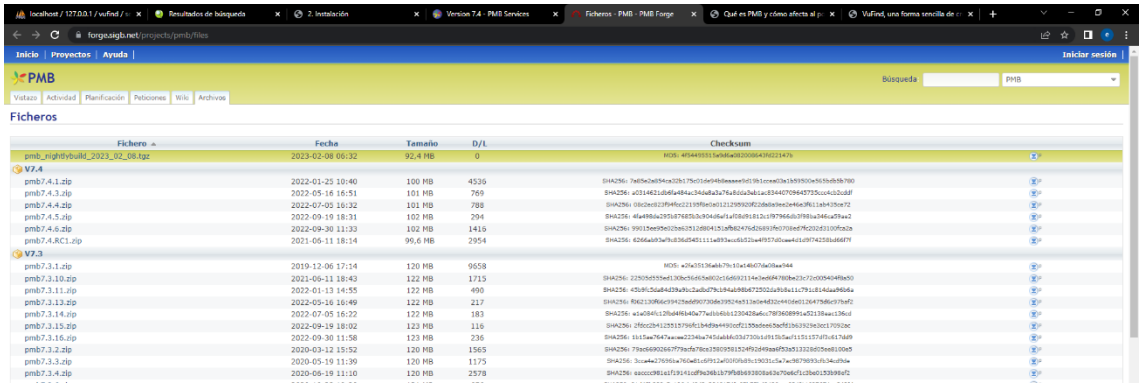


figura 285. Versiones de PMB disponibles.

Una vez descargada la versión de PMB (en nuestro caso hemos elegido la versión 7.4.6), hemos copiado el contenido descomprimido en la carpeta “htdocs” de nuestro servidor apache (c:\xampp\htdocs). En el caso particular de PMB, es realmente importante comprobar los requisitos de cada versión, dado que en primera instancia nos dio bastantes problemas y los mensajes de error no reflejaban el error de una manera demasiado evidente. Por poner un ejemplo, al principio en el servidor teníamos una versión de PHP más actual de la que podía soportar esta versión de PMB y tuvimos que configurar la versión de PHP apropiada (versión 7.4.29).

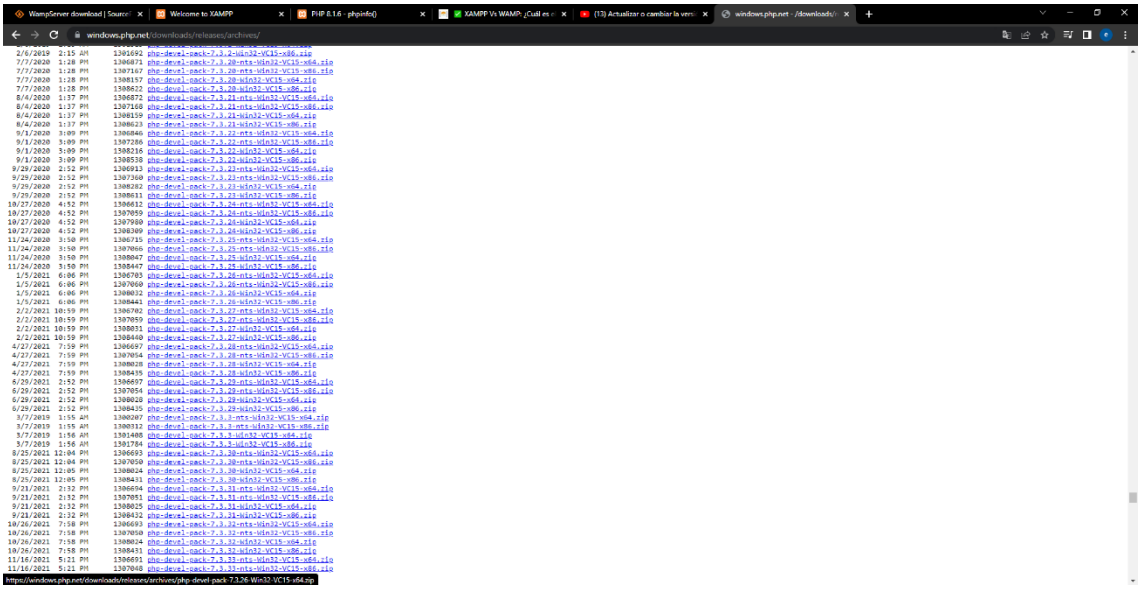


figura 286. URL de descarga de la versión apropiada de PHP.

Una vez copiados los ficheros en la carpeta “htdocs”, debemos abrir el navegador y acceder a la siguiente URL:

http://localhost/pmb/tables/install.php

Mostrándose una pantalla como la de la figura 287.

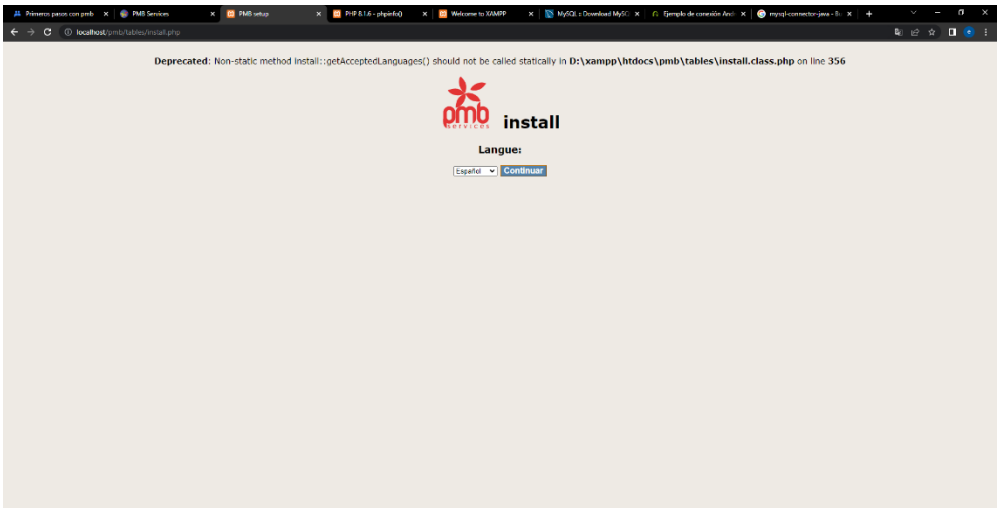


figura 287. Pantalla inicial de la instalación de PMB.

Al finalizar el proceso se nos mostrará una pantalla como la de la figura 288, dónde se nos indican aquellos elementos que han podido ser instalados correctamente y aquellos para los que ha fallado su instalación debido a que ciertos requisitos del sistema no se cumplen y por tanto la instalación de PMB no puede completarse. En concreto nos indica que ciertos módulos los tenemos deshabilitados en el servidor. Para habilitarlos hemos de editar el fichero “php.ini” de nuestro XAMPP y descomentar las líneas correspondientes a determinadas extensiones, para que al iniciar el servidor se activen los módulos necesarios. En concreto debemos descomentar las líneas presentes en la figura 289 y que se corresponden con el listado de errores de la pantalla (intl, soap, sockets, sqlite3, xsl). Una vez descomentadas estas líneas (eliminando el ; de la izquierda) y reiniciado el servidor, hemos podido completar la instalación de PMB.

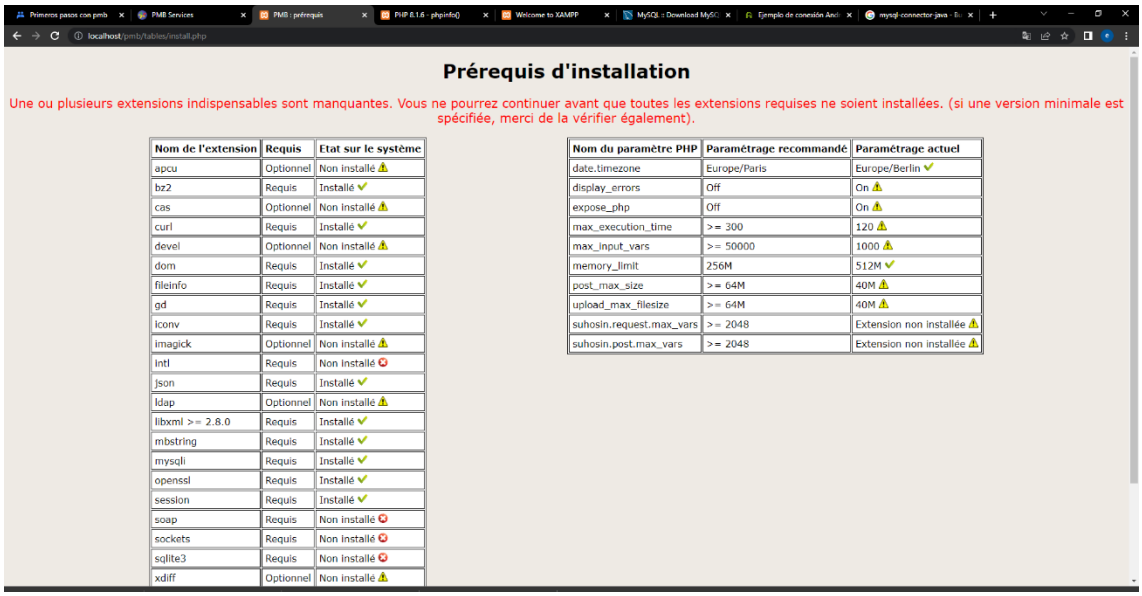


figura 288. Resultado de la instalación de PMB.

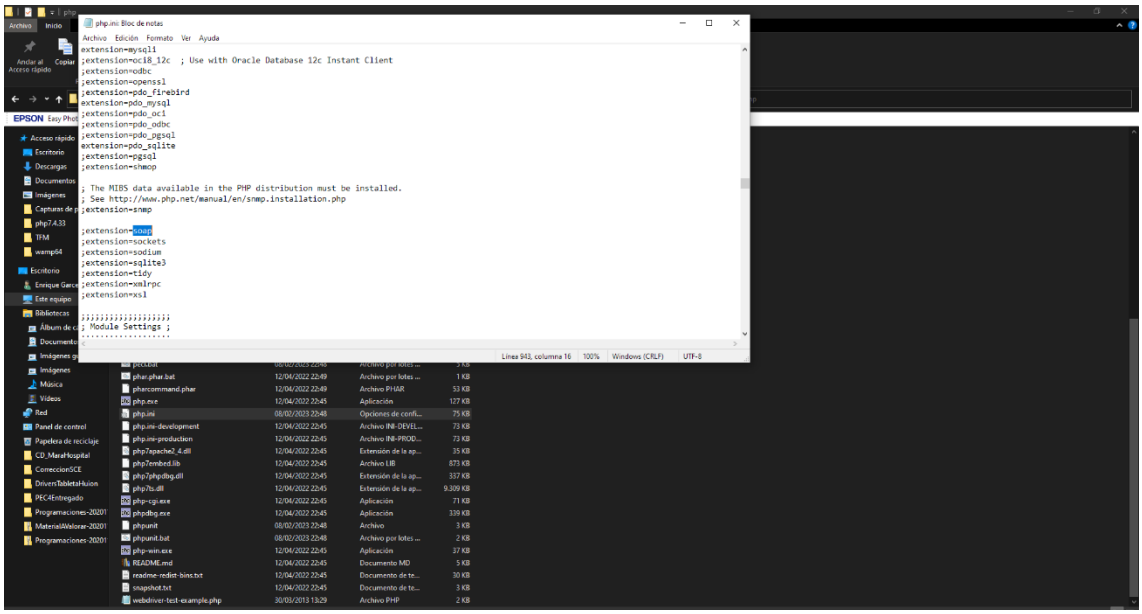


figura 289. Edición del fichero php.ini para habilitar las extensiones necesarias.

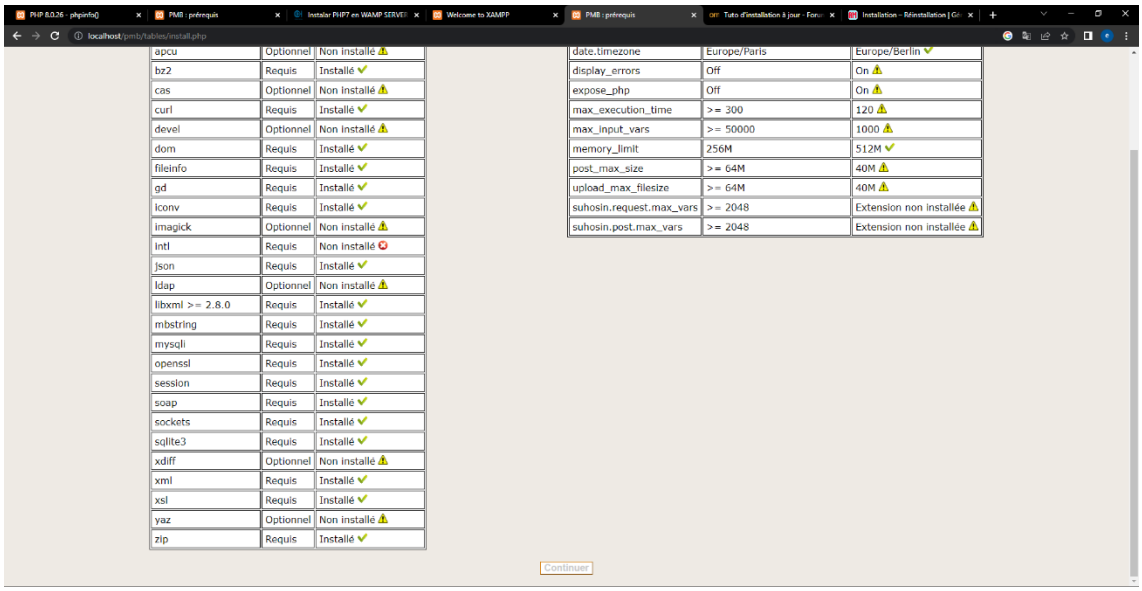


figura 291. Instalación de PMB. Primer paso realizado con éxito.

El siguiente paso de instalación consiste en crear la base de datos sobre la que se almacenarán los datos. Para ello necesitamos conectarnos con un usuario que posea permisos de administración como el usuario root. La instalación de PMB nos pide esta información mediante la pantalla de la figura 290. En este apartado nuevamente debemos tener un aspecto en cuenta. Cuando instalamos XAMPP, la contraseña del usuario “root” es un valor vacío y sin embargo, para PMB dicha contraseña es un valor obligatorio y no nos deja avanzar con la instalación. Para solucionar este problema, hemos accedido a PHPMyAdmin y hemos asignado una contraseña para el usuario “root” (123456).

figura 290. Instalación de PMB. Datos de conexión a la base de datos.

Uno de los últimos fallos que debimos solucionar en el proceso de instalación fue el del juego de caracteres y la colación en el servidor (figura 292). Para cambiar el cotejamiento, accedimos al servidor de base de datos y realizamos la modificación oportuna (figura 293).

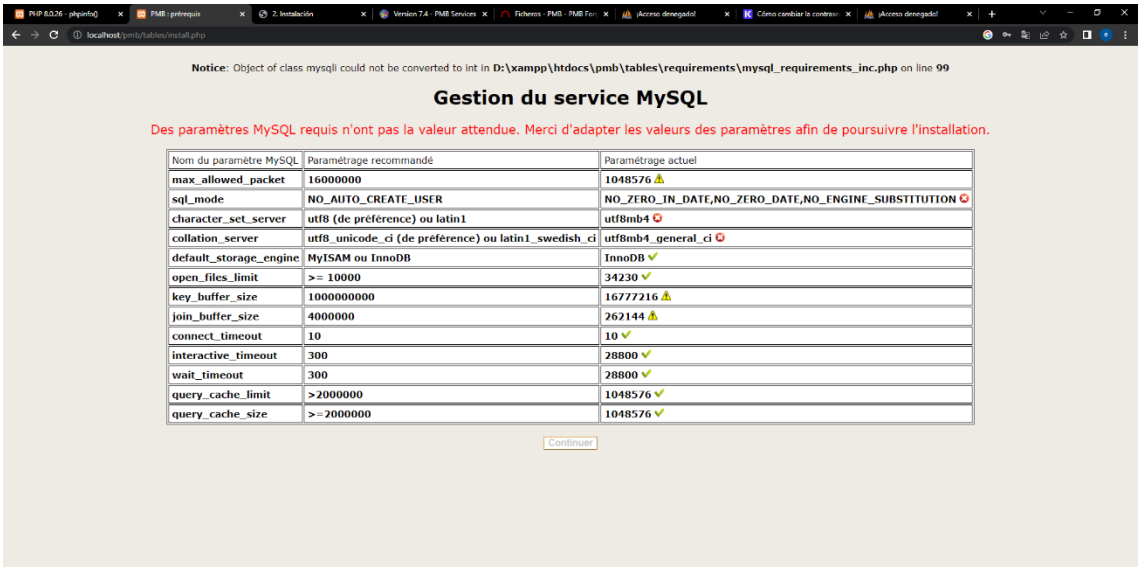


figura 292. Error en el juego de caracteres empleado por el servidor.

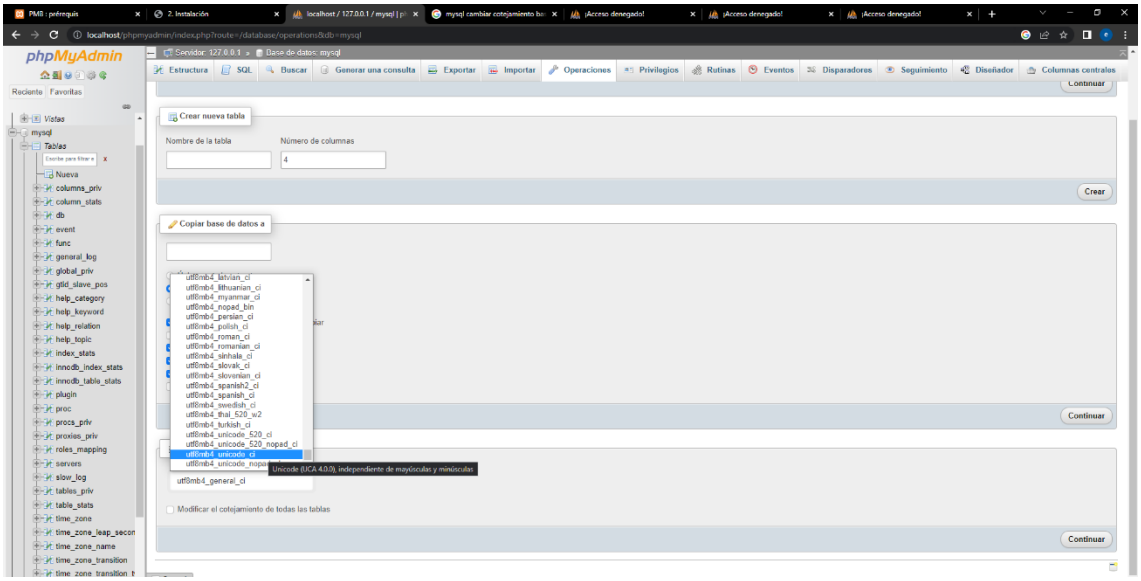


figura 293. Cambio del cotejamiento del servidor de bases de datos.

El juego de caracteres y la colación también la podemos cambiar del fichero my.ini (figura 294).

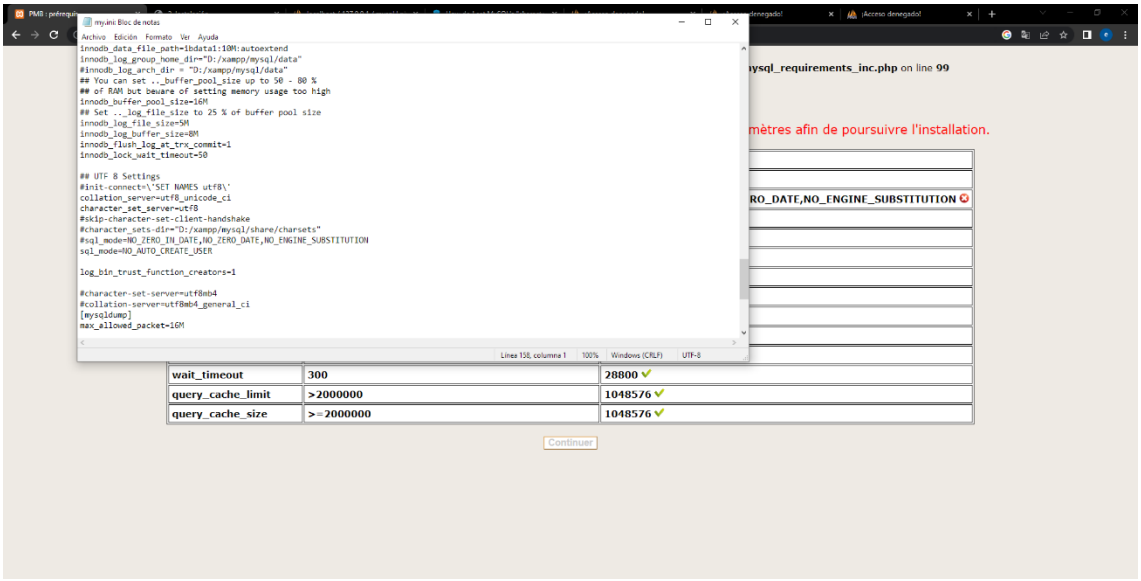


figura 294. Cambio del juego de caracteres y la colación desde el fichero my.ini.

Una vez resuelto el problema del juego de caracteres, por fin podemos continuar con la instalación asignando una contraseña para el usuario “bibli”. Es importante destacar que PMB crea unos usuarios por defecto como el usuario “admin” al que le asigna la contraseña “admin”, por lo que es importante cambiar esta contraseña por defecto para proteger el sistema.

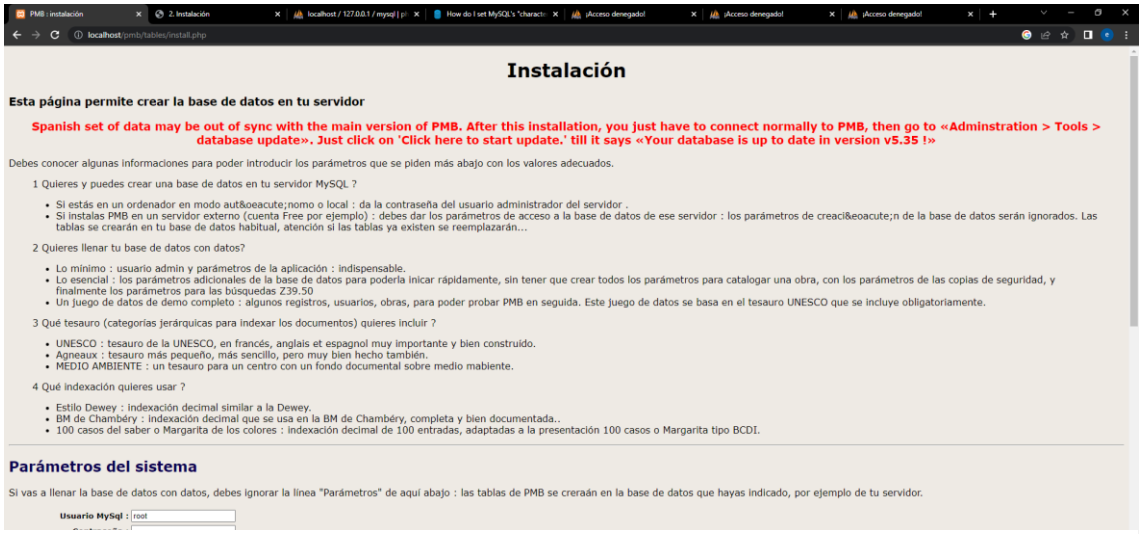


figura 295. Proceso de instalación de PMB. Configuración usuario MySQL.

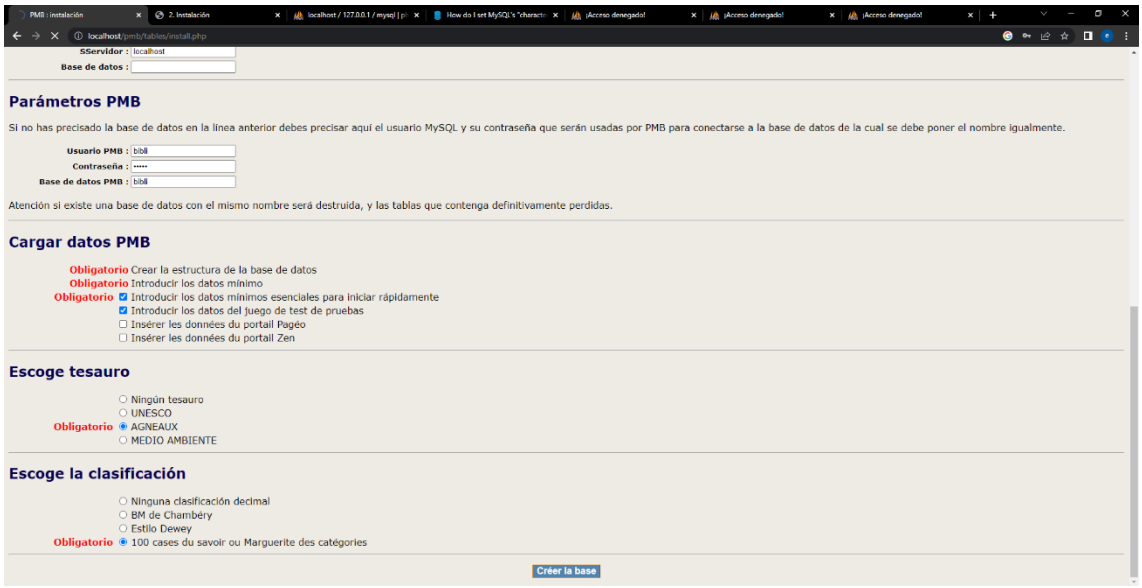


figura 296. Proceso de instalación de PMB. Configuración usuario bibli.

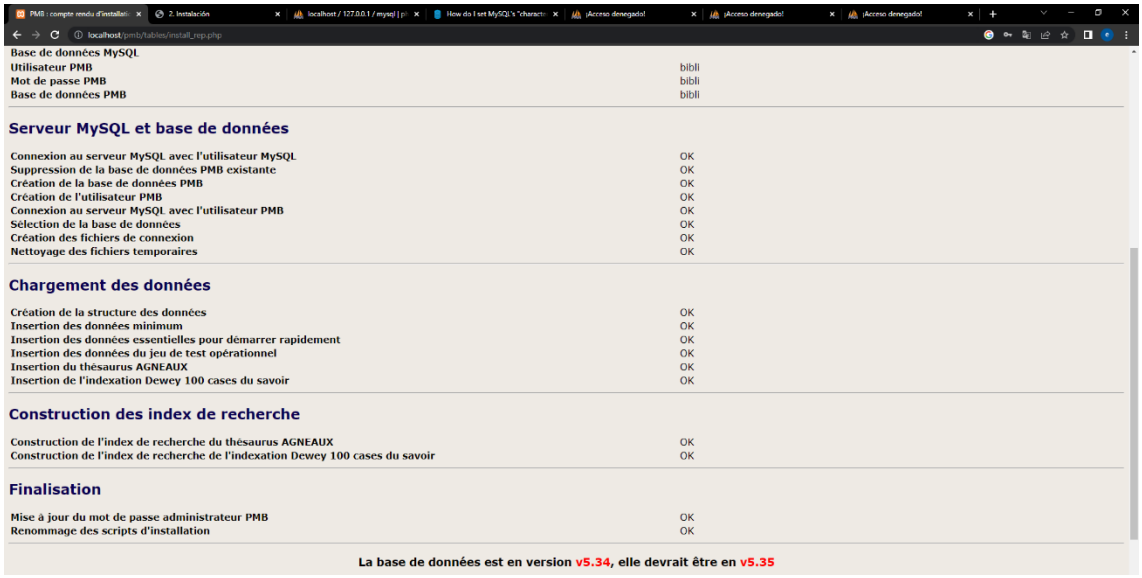


figura 297. Instalación de PMB realizada con éxito.

18 Anexo II. Pruebas internas de la aplicación.

Una vez probada la aplicación mediante los mecanismos disponibles en Android Studio (depuración mediante un dispositivo virtual, depuración sobre un dispositivo real con el modo desarrollador habilitado, etc.), podemos subir la aplicación desarrollada al “Play Store” mediante la herramienta “Play Console”.

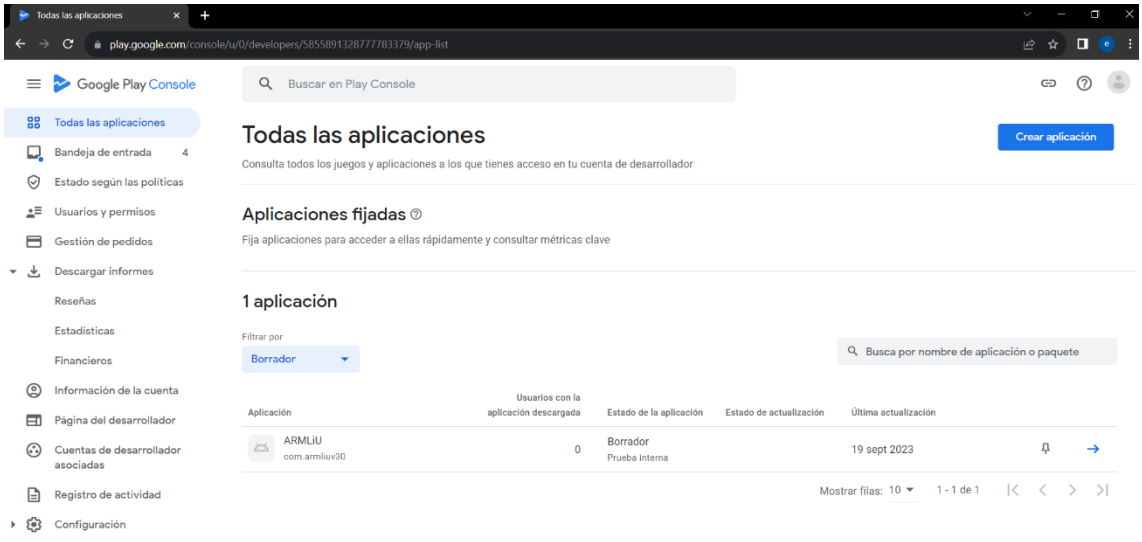


figura 298. Muestra de las aplicaciones disponibles dentro de “Play Console”.

Sobre esta herramienta podemos configurar básicamente tres tipos de pruebas: Pruebas internas, pruebas cerradas y pruebas abiertas. Las pruebas internas nos permiten subir la aplicación para que la valoren un grupo de usuarios de nuestra elección que se denominan “TestersInternos”.

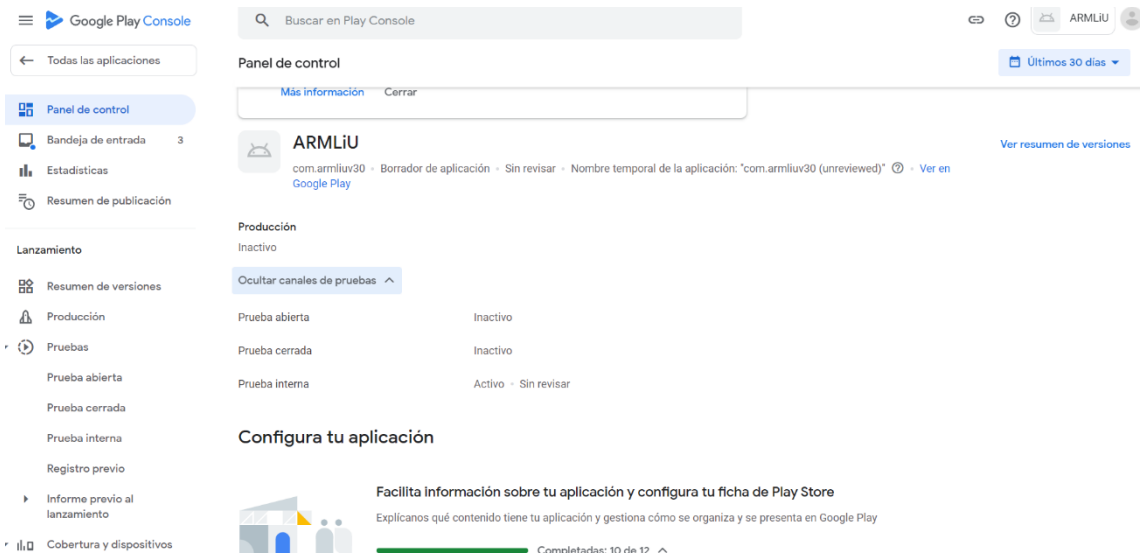


figura 299. Vista de “Play Console”. Pruebas activas para la aplicación ARMLiU.

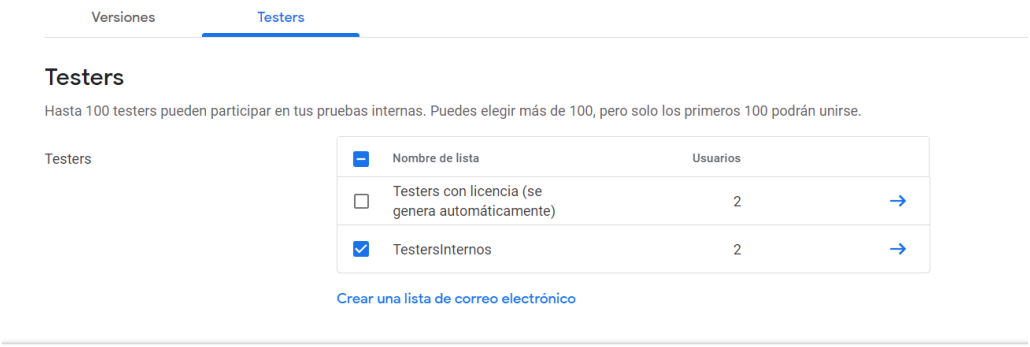


figura 300. Número de "testers" habilitados para las pruebas internas de ARMLiU.

Existen dos tipos de fichero que nos permiten poner la aplicación a disposición de los usuarios. Los de tipo APK (Android Application Package) y los de tipo "Bundle". El tipo APK es el que se ha utilizado tradicionalmente para distribuir aplicaciones, pero está siendo reemplazado por el tipo "Bundle" que encapsula un número de elementos adicionales frente a los de tipo APK. Por poner un ejemplo, si estuviéramos empleando como es nuestro caso algún tipo de código nativo, deberíamos acompañar el fichero APK de otro fichero que contuviera el conjunto de símbolos empleados. Mientras que el fichero de tipo "Bundle" lo incluiría en su interior. Para generar el fichero de tipo APK

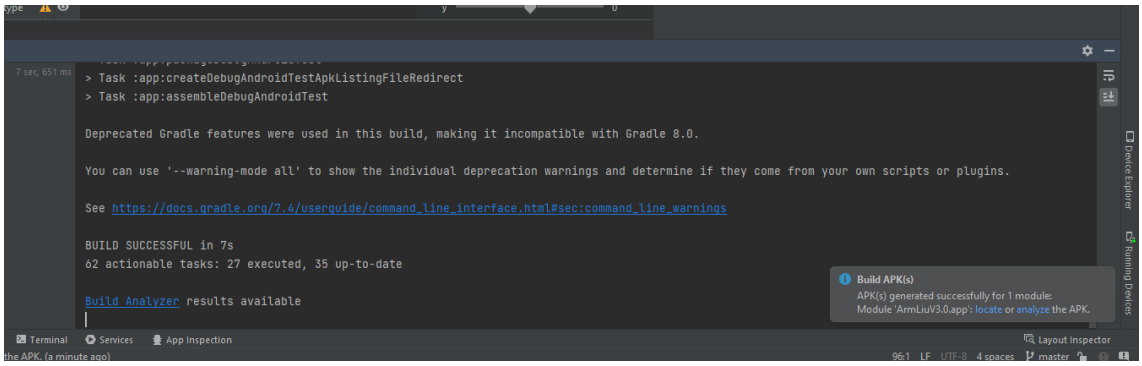


figura 301. Generación del fichero de tipo APK de la aplicación.

seleccionamos la opción de menú "Build" junto al submenú "Build APK" (figura 301).

Para generar el fichero de tipo "Bundle" de la versión de la aplicación deseada (debug, proguard o release), seleccionamos nuevamente la opción de menú "Build" y "Generate Signed Bundle/apk". Esta acción, la primera vez que la realicemos nos guiará sobre la creación de una clave de firma. Y finalmente

nos creará unos archivos con extensión “.aab” correspondientes a los ficheros de tipo “Bundle”.

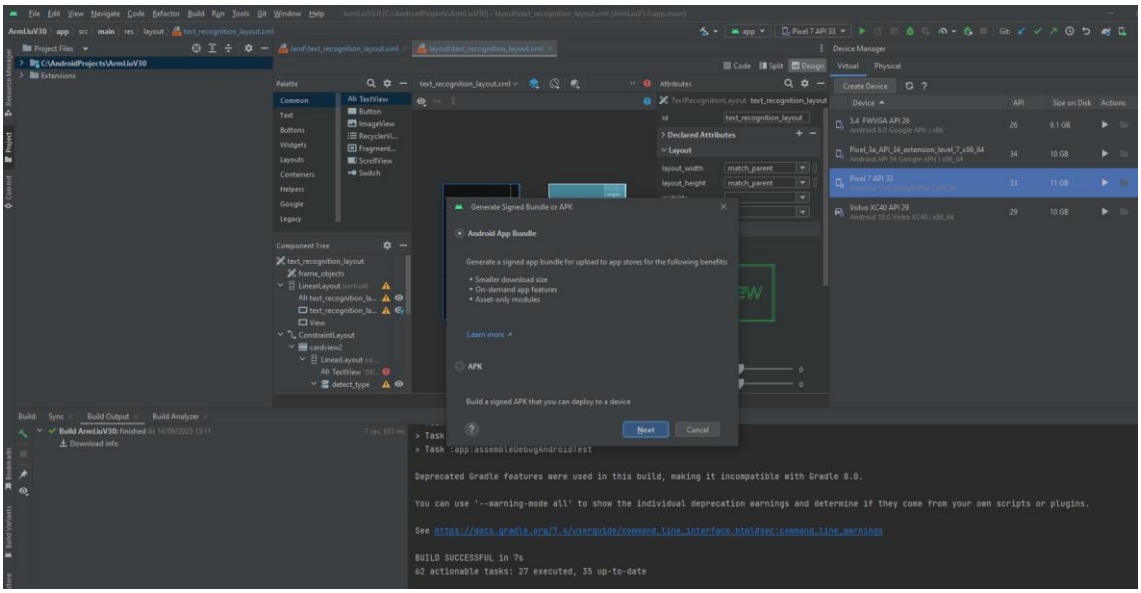


figura 302. Creación de los “Bundles” correspondientes a la aplicación.

Finalizado el proceso, deberemos crear la aplicación mediante la herramienta “Play Console” y subir el fichero generado cogiéndolo de la carpeta “release” del proyecto. Denotar que es de vital importancia revisar de manera pormenorizada todos aquellos errores y advertencias que nos proporciona Android Studio para cada uno de los ficheros presentes en el proyecto, dado que, la ejecución mediante las herramientas de depuración, es bastante más permisiva que el sistema de puesta en producción de la aplicación. Debemos prestar especial atención a los ficheros de manifiesto y a los de configuración de “Gradle”.

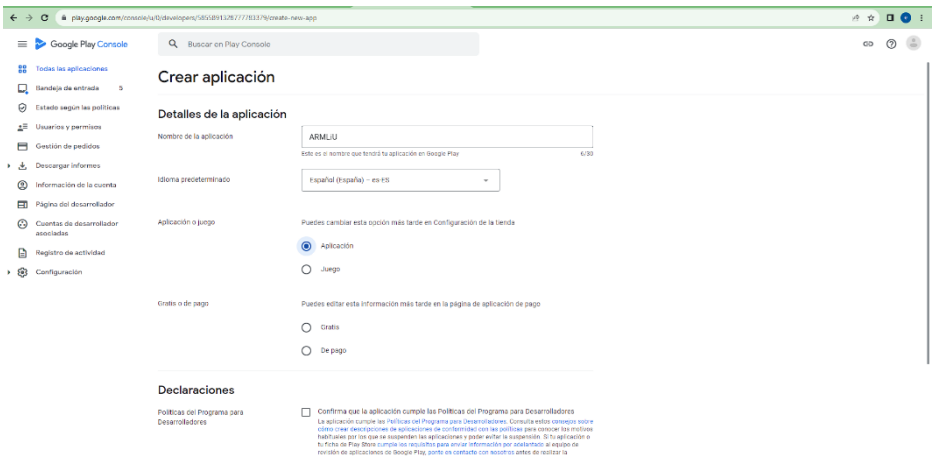


figura 303. Creación de la aplicación mediante “Play Console”.

El proceso de creación de la aplicación nos guiará mediante una serie de pasos para que esta esté disponible para los “testers” internos. Destacar la creación de una firma para la que hemos elegido la opción “Usar una clave generada por Google”, así como la cumplimentación de múltiples formularios donde se proporciona información sobre las características y términos de uso de nuestra aplicación (declaraciones).

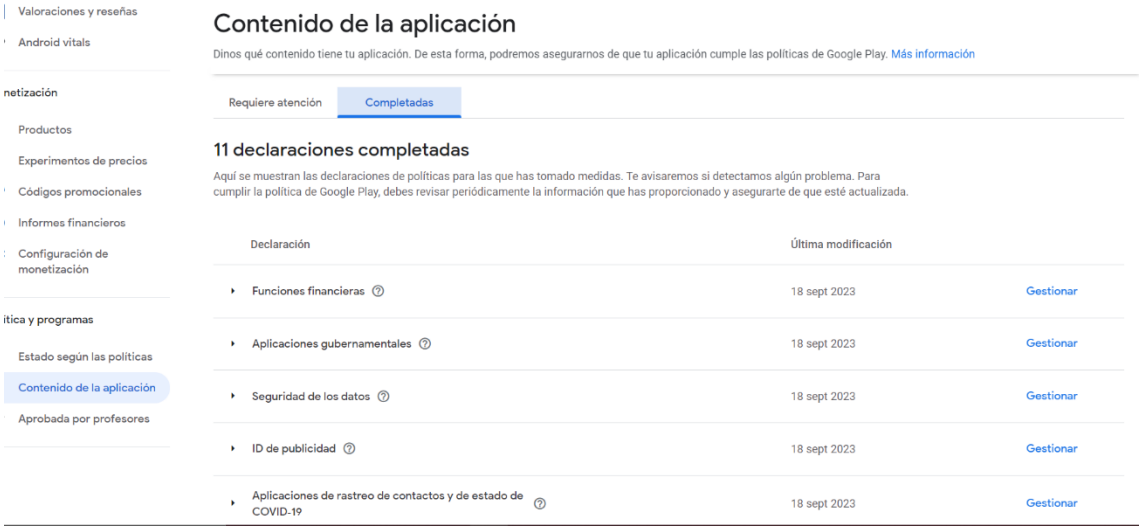


figura 304. Declaraciones aceptadas para poder poner a disposición de los “testers” la aplicación.

Una de las declaraciones, consiste en la Política de Privacidad de la aplicación. En este formulario debemos indicar una URL que contendrá las políticas asociadas a nuestra aplicación. En nuestro caso hemos creado un “Google Sites” en el espacio asociado a nuestra cuenta de desarrollador.

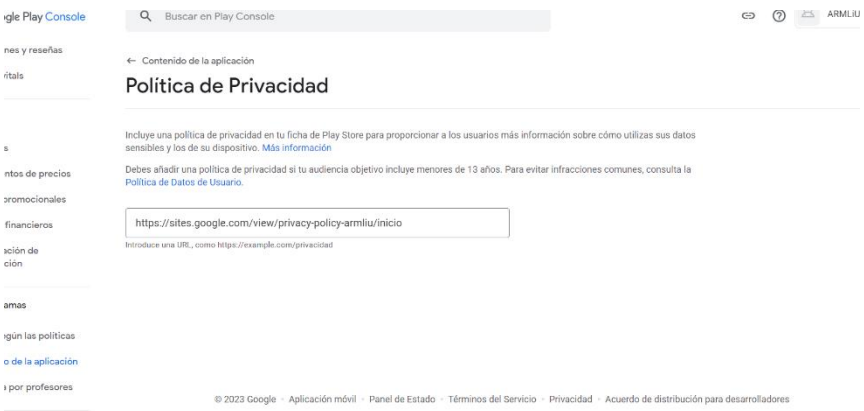


figura 305. Enlace al "Google Sites" creado para contener las Políticas de Privacidad.

figura 306. Políticas de Privacidad de la aplicación.

Accediendo al enlace proporcionado a los “testers” de la aplicación y después de aceptar la invitación, accederán a la instalación de la aplicación dentro del “Play Store”.

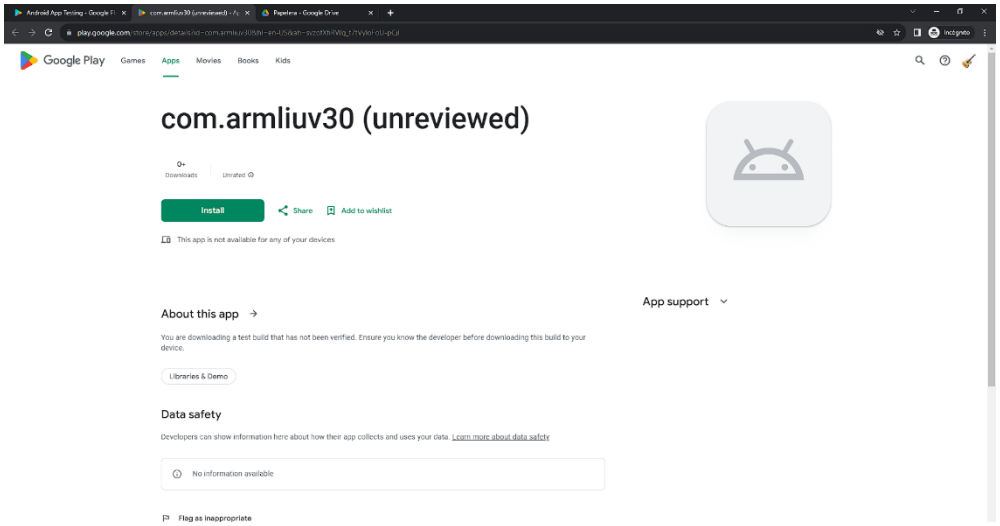


figura 307. Instalación de la aplicacación dentro del "Play Store". Acceso restringido a los "testers".

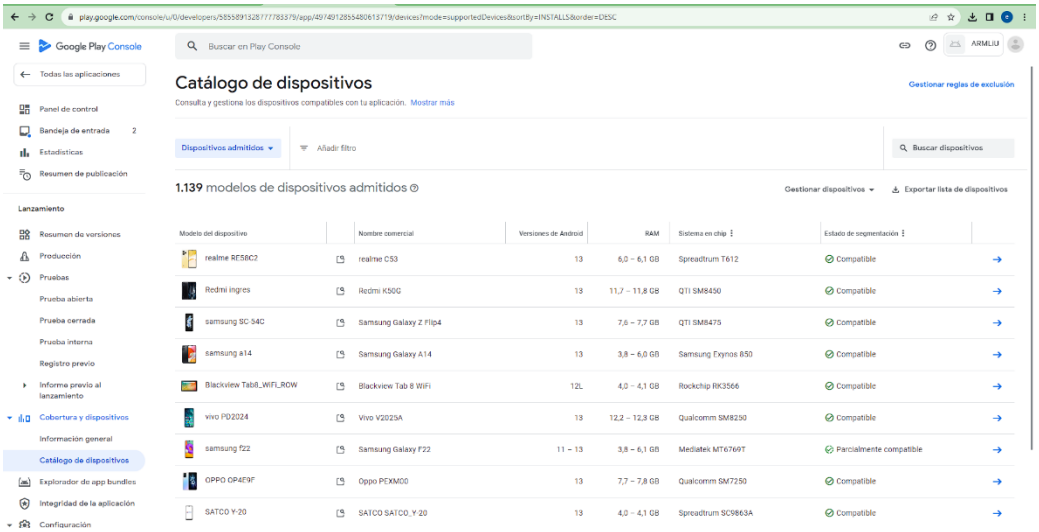


figura 308. Dispositivos compatibles con la versión de la aplicación.