

Memoria de Proyecto Integrado

Francisco Rafael Luna Pereira



Soleus

Curso 2021/2022

2º DAM CESUR

Índice

| | |
|--|----|
| 1. Resumen..... | 4 |
| 2. Introducción | 5 |
| 3. Objetivos y características del proyecto | 6 |
| 4. Finalidad | 8 |
| 4.1 Finalidad: Clientes..... | 8 |
| 4.2 Finalidad: Empleados | 9 |
| 4.3 Finalidad: Aplicación de escritorio | 9 |
| 4.4 Finalidad: Base de datos..... | 9 |
| 5. Medio materiales usados..... | 10 |
| 5.1 Android Studio | 10 |
| 5.2 Eclipse | 11 |
| 5.3 XAMPP | 12 |
| 5.4 MySQL..... | 13 |
| 5.5 PhpMyAdmin | 13 |
| 5.6 Adobe Creative Cloud Express | 14 |
| 5.7 Material.io | 15 |
| 5.8 Draw.io | 16 |
| 6. Planificación del proyecto..... | 17 |
| 6.1 Diseño..... | 17 |
| 6.1.1 Selección de colores | 19 |
| 6.1.2 Creación de botones y logos | 19 |
| 6.1.3 Otros recursos de diseño..... | 20 |
| 6.2 Modelos y planificación | 21 |
| 6.2.1 Modelo de datos..... | 21 |
| 6.2.2 Entidades | 24 |
| 6.2.3 Vista general de la aplicación: Diagrama de casos de uso | 25 |
| 6.3 Actividades, vistas y clases | 26 |
| 6.3.1 ClientNet: Clase cliente | 27 |
| 6.3.2 Vistas de clientes | 27 |
| 6.3.3 Vistas de empleado | 33 |
| 6.3.4 Vistas de administrador | 37 |

| | |
|---|----|
| 6.3.5 Clase Utils..... | 47 |
| 6.3.6 Servidor | 48 |
| 7. Fase de pruebas | 50 |
| 8. Conclusiones y trabajos futuros o posibles mejoras | 52 |
| 8.1 Conclusiones..... | 52 |
| 8.2 Posibles mejoras | 52 |
| 9. Referencias bibliográficas..... | 54 |

1. Resumen

En este documento se presenta la memoria del proyecto integrado realizado para los estudios del Grado Superior de Desarrollo de Aplicaciones Multiplataforma cursados por Francisco Rafael Luna Pereira. El proyecto nos presenta la creación de una aplicación móvil Android llamada Soleus que, pensada para su uso en pequeños establecimientos hoteleros o de alojamientos turísticos, nos permitirá la creación de peticiones por parte de los clientes (asociadas al mantenimiento o la limpieza) y su posterior guardado en una base de datos MySQL. Estas peticiones quedarán almacenadas asociadas a la habitación del usuario, permitiendo que el personal las marque posteriormente como finalizadas; Soleus permitirá además gestionar tanto los usuarios como las peticiones mediante el uso de un usuario administrador. Se presentará también la creación de una aplicación de escritorio que sin necesidad de interfaz gráfica se ejecutará en el servidor deseado para permitir y gestionar estas operaciones.

2. Introducción

Soleus nace con la idea de permitir la gestión de peticiones de clientes alojados en aquellos establecimientos en los que no se disponga de un sistema similar por cuestiones logísticas o económicas, siendo funcional y pudiendo ampliar estas funcionalidades en el futuro. El proyecto puede dividirse en tres componentes: la aplicación móvil, la aplicación de escritorio y la base de datos, siendo los dos primeros los componentes más importantes y aquellos en los que se centrará en mayor parte este documento.

3. Objetivos y características del proyecto

Los objetivos del desarrollo de la aplicación es la puesta a prueba y la ampliación de los conocimientos adquiridos a lo largo de los dos años de estudio, específicamente de mis habilidades con el lenguaje de programación Java, el desarrollo de aplicaciones móviles y mi manejo del entorno de desarrollo Android Studio (más información en el punto 5, Medio materiales usados). Partiendo de estos objetivos principales, añadiendo por supuesto el deber de realizarlos para completar los estudios satisfactoriamente, surgen nuevos objetivos relacionados con la aplicación en sí, que listaremos a continuación:

- La creación de una aplicación intuitiva, “agradable” para el usuario que pueda ser utilizada sin necesidad de comprender su funcionamiento interno, que sea directa y concisa. Esto conlleva además la elección adecuada de colores para el diseño.
- Que brinde la oportunidad de poder facilitar las tareas de los departamentos de pisos, mantenimiento y recepción de un hotel. Lo común es que el departamento de recepción, como departamento central, tenga que encargarse de intermediar entre los departamentos de back-office y el cliente, con el uso de la aplicación el cliente interactúa directamente con estos departamentos eliminando esa intermediación, ganando tiempo y efectividad. Esto no sólo mejoraría el trabajo de los dos departamentos a los que va dirigidos, sino que, además, el ya mencionado departamento de recepción, se vería librado de su necesidad de comunicar las necesidades del cliente continuamente.

- La posibilidad de que los empleados puedan acceder de manera rápida a las peticiones que hayan recibido sus departamentos, evitando las pantallas intermedias y permitiendo que desde la propia lista se marquen como finalizadas.
- Por razones de gestión y ante (de momento) la inexistencia de una aplicación web, la posibilidad mediante un usuario administración de acceder a la información recogida que ya no es necesaria para el empleado (peticiones finalizadas y filtrado de las mismas según el asunto). Este usuario administrador debe además poder gestionar los usuarios que tendrán acceso a la aplicación y su contenido, mediante la creación, modificación y, si requerido, eliminación de los mismos.

4. Finalidad

La finalidad principal de Soleus, como hemos visto en puntos anteriores, es permitir una gestión sencilla de las peticiones relacionadas con los departamentos de pisos o mantenimiento que puedan surgir a los clientes durante su estancia en algún establecimiento de tamaño moderado. De esta manera, como se explica en el punto anterior (3. Objetivos y características del proyecto) la aplicación móvil debe dar servicio directo a los usuarios (empleados y clientes), la aplicación de escritorio debe dar soporte a la aplicación móvil conectándola con la base de datos y funcionando como back-end, y la base de datos nos permitirá almacenar estas transacciones realizadas.

4.1 Finalidad: Clientes

Tras autenticarse utilizando un nombre de usuario (número de habitación a lo largo del desarrollo) y una contraseña de seis caracteres, la aplicación móvil permite a los clientes la creación de peticiones relacionadas con los departamentos de pisos y de mantenimiento de un hotel, pudiendo solicitarles por lo tanto una variada selección de asuntos: limpieza de la habitación, nuevos amenities, reparación de problemas surgidos con los componentes eléctricos, etc. De esta manera sus solicitudes llegarán a los empleados de estos departamentos sin la necesidad de la intermediación de otro departamento (comúnmente recepción).

4.2 Finalidad: Empleados

De la misma manera que el cliente, el empleado debe primero autenticarse utilizando un nombre de usuario y una contraseña. Tras ello se le presentan directamente las peticiones que se encuentren activas relacionadas con su departamento, permitiéndoles desde esa pantalla marcarlas como finalizadas y actualizar la lista, eliminándolas de la pantalla (sin ser eliminadas de la base de datos por motivos de persistencia). Esto hace más sencillo el control de las peticiones y aumenta la velocidad de respuesta de los departamentos involucrados, así como pretende también aumentar la satisfacción del cliente.

4.3 Finalidad: Aplicación de escritorio

La aplicación de escritorio, actualmente sin interfaz gráfica, se encarga de realizar las labores del servidor: recibe y contesta las peticiones de la aplicación móvil y se encarga de almacenar, modificar, consultar o eliminar datos en las tablas de la base de datos que da soporte a las aplicaciones.

4.4 Finalidad: Base de datos

Para garantizar un correcto funcionamiento de las dos aplicaciones precisamos de una base de datos que dé soporte, almacenando y poniendo a disposición los datos relacionados con los clientes y con las peticiones de los mismos.

5. Medio materiales usados

Para garantizar un correcto funcionamiento de las dos aplicaciones precisamos de una base de datos que dé soporte, almacenando y poniendo a disposición los datos relacionados con los clientes y con las peticiones de los mismos.

5.1 Android Studio

Este programa ha sido la herramienta principal utilizada a lo largo de todo el desarrollo. Android Studio es el IDE (Entorno de desarrollo) oficial de Android, fue creado exclusivamente para poder acelerar el desarrollo y ayudar en la compilación de aplicaciones de alta calidad, dirigidas a dispositivos Android (Sitio Web de Android Studio, 2022). Al haber sido una herramienta utilizada a lo largo del segundo año del Grado Superior, es una herramienta con la que ya estaba familiarizado y tenía experiencia (a nivel de estudios). Nos permite generar vistas/actividades mediante XML o utilizando una interfaz en la que pueden añadirse directamente los componentes que queramos que sean utilizados en nuestra aplicación.

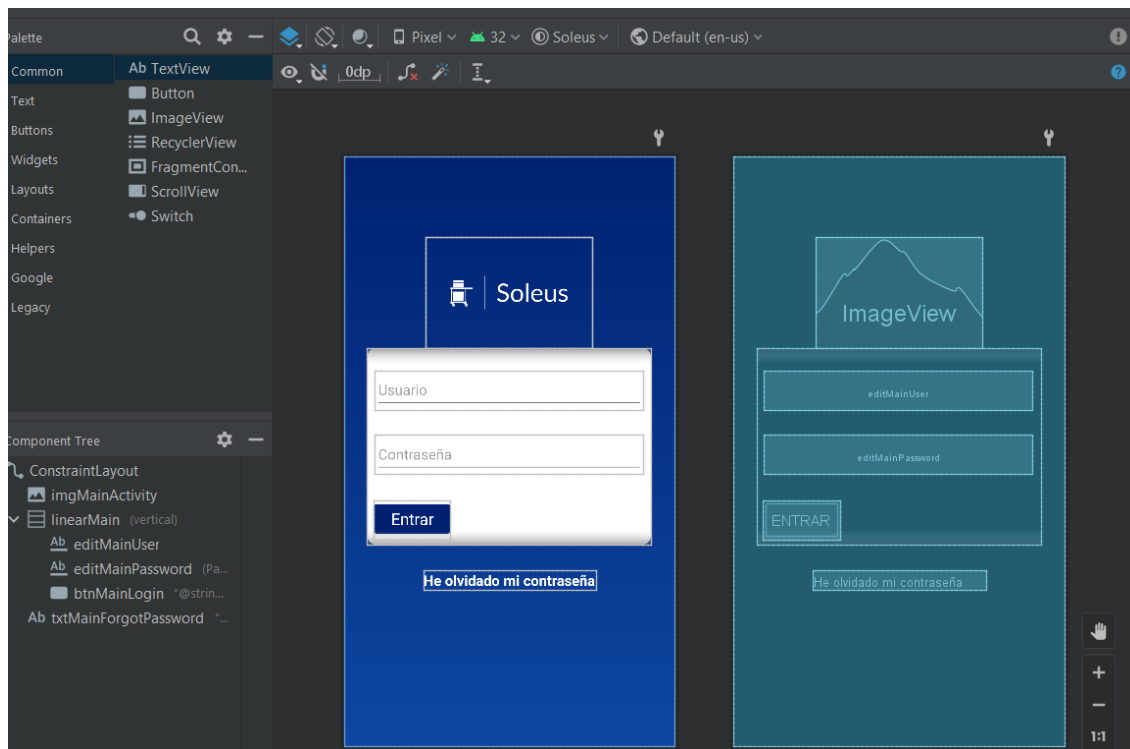


Figura 1. Interfaz de diseño de Android Studio, imagen propia

5.2 Eclipse

Si bien la parte principal del proyecto a considerar, y la que deseo que sea presentada, es la aplicación, esta precisa de un servidor que interactúe con la Base de Datos, como veremos en los puntos siguientes de la memoria, dónde se explicará con más detalle el funcionamiento de la aplicación. Con Eclipse elaboramos estas clases que nos sirven de apoyo, utilizando el lenguaje de programación Java.

Eclipse es un entorno de desarrollo que nos permite extender sus funcionalidades a través de plug-ins. En un principio se ideó para convertirse en una plataforma de integración de herramientas de desarrollo. Es un IDE genérico, lo que significa que no nos sirve para programar únicamente en un lenguaje específico. Es sin embargo uno de los IDE más populares entre los

desarrolladores de Java. Eclipse nos proporciona herramientas para desplegar, escribir, ejecutar y depurar aplicaciones (calendamaia, 2014).

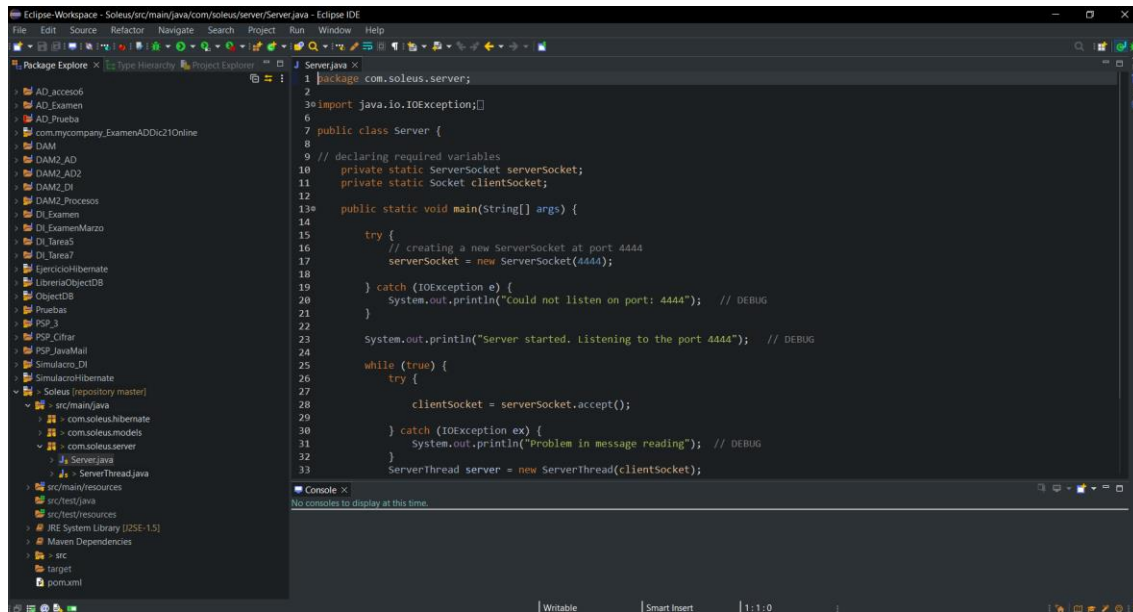


Figura 2. Eclipse, imagen propia

5.3 XAMPP

XAMPP es una distribución gratuita, un paquete de programas que incluye MariaDB (MySQL), PHP y Pearl. Nos permite generar un servidor web de Apache en nuestro ordenador de hogar, siendo un paquete diseñado específicamente para ser muy sencillo de instalar (VMware, s.f.). Habiéndolo utilizado a lo largo de los estudios realizados, el estar familiarizado con él ha permitido un trabajo más sencillo sobre las bases de datos.

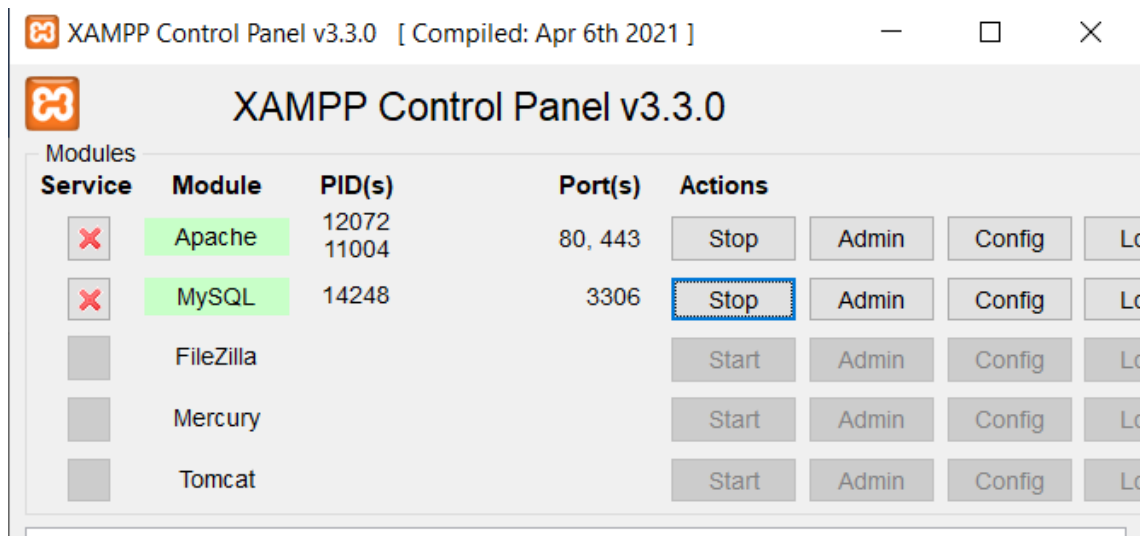


Figura 3. Imagen de XAMPP. Imgen propia

5.4 MySQL

Como se explica en el punto anterior, una base de datos es necesaria para el desarrollo normal de la ejecución de la aplicación, muchas de los procesos requieren de una consulta en esta base de datos y muchos otros requieren del almacenamiento de datos. MySQL es un sistema de gestión de bases de datos relacional, estando considerada como una la base de datos de código abierto más popular del mundo.

5.5 PhpMyAdmin

Teniendo que utilizar una base de datos MySQL, es necesario interactuar con ella también para la creación de tablas, estructuración, modificación de registros, consultas, etc.; especialmente desde el punto de vista del desarrollador a lo largo del proceso de creación de la aplicación. Para ello, necesitamos de un sistema que nos permita gestionar estas bases de datos.

En un principio podemos realizarlo vía terminal de comandos, sin embargo, esta herramienta, si bien totalmente funcional, no es tan cómoda de utilizar debido a la falta de interfaz. Para evitar esto y hacer más amenos estos procesos, phpMyAdmin fue la herramienta escogida, la interfaz gráfica que nos ha permitido acceder al servidor MySQL y consultar, crear y modificar las tablas y sus registros. Es destacable que phpMyAdmin es una aplicación web, no una aplicación de escritorio (Zúñiga, 2021).

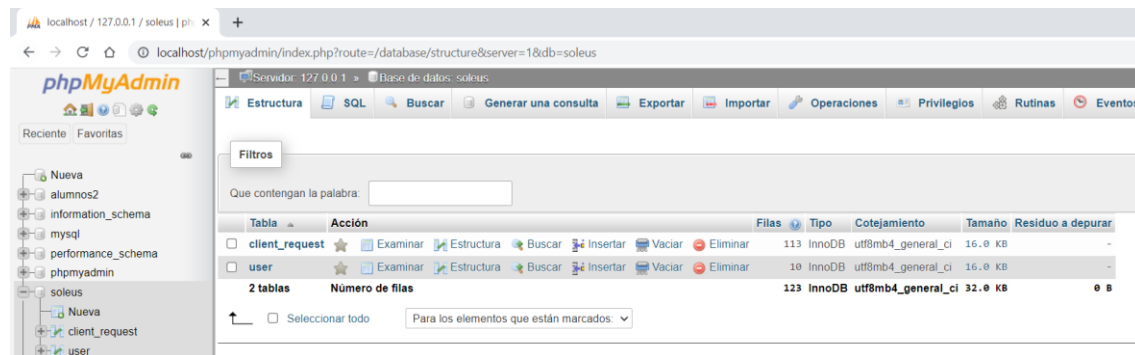


Figura 4. Portal de phpMyAdmin, imagen propia

5.6 Adobe Creative Cloud Express

Portal web de acceso gratuito que nos ofrece herramientas de diseño de logos, rótulos, etc. Sin necesidad de descargar ninguna aplicación adicional al navegador. A lo largo del proyecto, este portal ha sido utilizado para numerosas tareas: diseño del logo de la aplicación, diseño de logos presentes en las vistas/actividades y diferentes diseños de botones (como veremos más adelante, ImageButtons). El portal es una herramienta muy útil para la creación y descarga de logos sin necesidad de tener conocimiento de herramientas de diseño, fácil de utilizar para un público desconocedor de la industria.

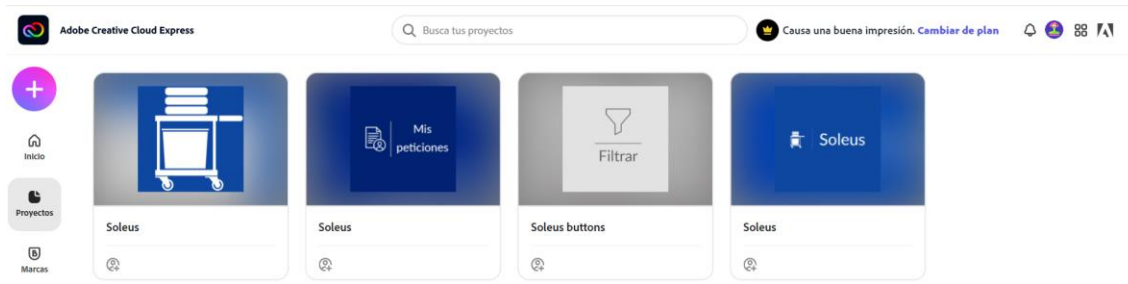


Figura 5. Adobe Creative Cloud Express. Imagen propia

5.7 Material.io

Portal web de Google que nos permite indagar sobre el concepto de los diseños Material y nos proporciona herramientas para diseñar una aplicación más atractiva para el usuario, contando con un asistente de colores, que se ha utilizado a lo largo del proyecto para la elección de los colores “corporativos”.

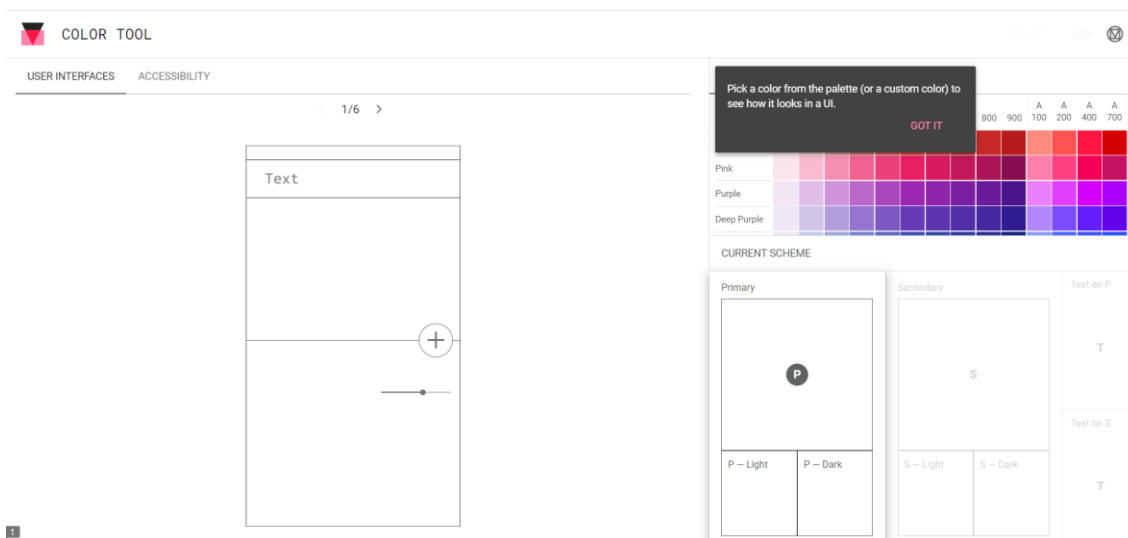


Figura 6. Material.io, paleta de colores. Imagen propia

5.8 Draw.io

Herramienta de diagramación gratuita con la que se puede dibujar cualquier tipo de mapas mentales, mapas conceptuales, esquemas o diferentes representaciones gráficas, como diagrama de jerarquía o conjuntos. Utilizada para realizar entre otros, el diagrama de casos de uso que veremos en puntos posteriores (Vázquez, s.f.).

6. Planificación del proyecto

En este apartado desarrollaremos puntos relacionados con la planificación de la aplicación, entrando más en detalle en el código y en las diferentes ventanas que nos ofrece la aplicación, así como en otras variantes relacionadas.

6.1 Diseño

Si bien el diseño final fue lo último que se desarrolló de la aplicación, siendo inicialmente programada con un diseño plano, sin ningún tipo de personalización, para probar las funcionalidades, por cuestiones de redacción y contexto lo desarrollaremos en primer lugar en esta memoria.

El objetivo era crear un diseño sencillo y funcional, que no contase con numerosos componentes en la misma pantalla para evitar el mal entendimiento de las funciones que ofrece la aplicación.

Para ello, la mayor parte de las ventanas de la aplicación muestran una estructura similar:

- Cabecera con un título descriptivo de la ventana abierta
- Logo de la aplicación en la esquina derecha arriba
- Resto de componentes de la aplicación (botones, imágenes, campos de texto, etc.) sobre un fondo blanco para distinguirlo del color de fondo genérico de la aplicación.

Las únicas dos ventanas de la aplicación que nos comparten este diseño son las creadas para mostrar un *login* inicial y aquella ideada para que los usuarios puedan recuperar su contraseña.

Ambas comparten un diseño similar: logo centrado horizontalmente, fondo blanco y cuadros de texto y botones sobre el mismo.



Figura 7. Ejemplo de ventana de la aplicación. Imagen propia

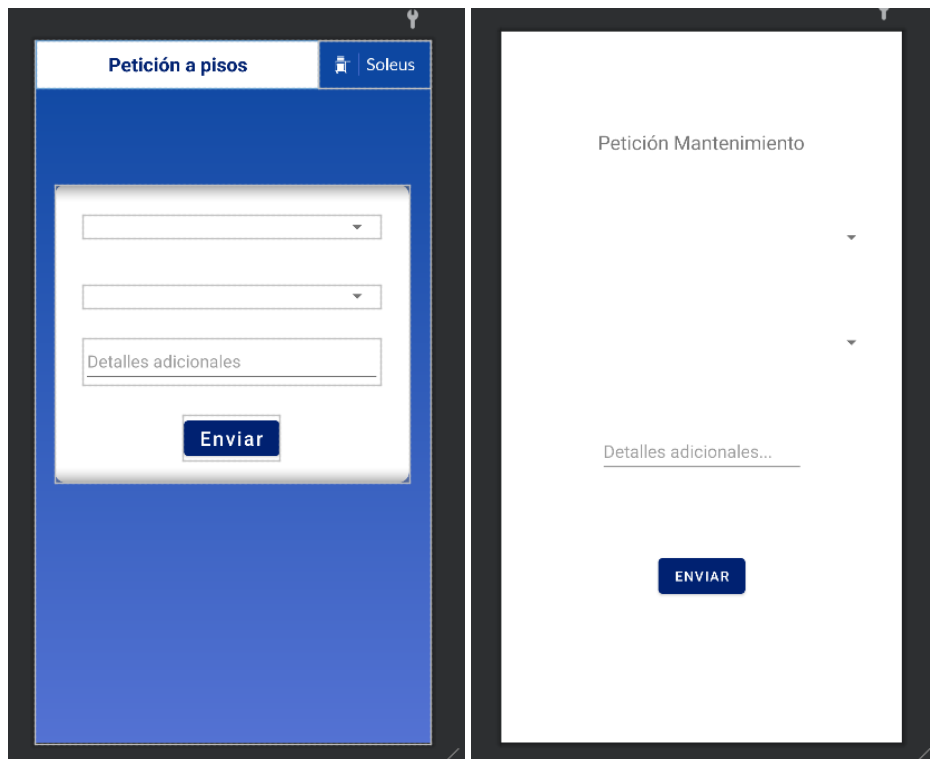










Figura 8. Comparativa entre el diseño final y el inicial. Imagen propia

6.1.1 Selección de colores

Los colores de la aplicación juegan un papel importante en el apartado visual: son los protagonistas en los fondos de todos los componentes, pantallas, bordes, botones, etc. Los colores escogidos para el desarrollo de Soleus son los siguientes:

- Color principal: #0D47A1 
- Color principal oscuro: #002171 
- Color principal claro: #5472D3 
- Color del texto: #002171 , #E1E2E1 (Gris) , #686263 
- Otros: #FFFFFF (Blanco), #FF1744 , #00E676 

Los fondos se realizan utilizando o bien el color blanco o degradados utilizando los tres colores principales.

6.1.2 Creación de botones y logos

Gracias a la ayuda de la herramienta Adobe Creative Cloud Express se pudieron diseñar todos los botones de la aplicación que incluyan logotipos. En algunas de las ventanas, estos botones cambian de color tras interactuar con ellos. Por esta razón, algunos tienen dos versiones, una utilizando el color principal oscuro de la aplicación y otra utilizando el color gris del texto.

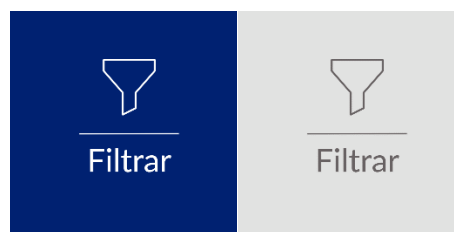


Figura 9. Ejemplo de botones. Imagen propia

Tal y como los botones, los logos de la aplicación se han creado utilizando la misma plataforma de Adobe. Estos, al igual que el resto del diseño de la aplicación, han pasado por varias versiones de las que se pueden destacar dos: la primera y la final.

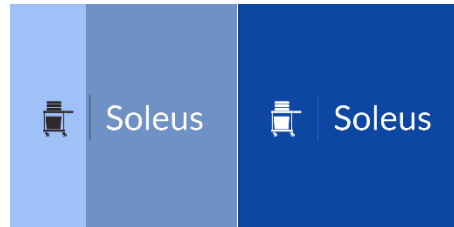


Figura 10. Diseño inicial y diseño final del logo. Imagen propia

6.1.3 Otros recursos de diseño

Para facilitar la estructura de la ventana o para destacar alguno de los componentes, se han utilizado archivos de recursos de Android Studio. Estos archivos se crearon para poder “dar forma” a algunos de los componentes de la aplicación, como son los característicos fondos blancos centrales con bordes redondeados de las ventanas o cuadros de texto con bordes para destacar su contenido.

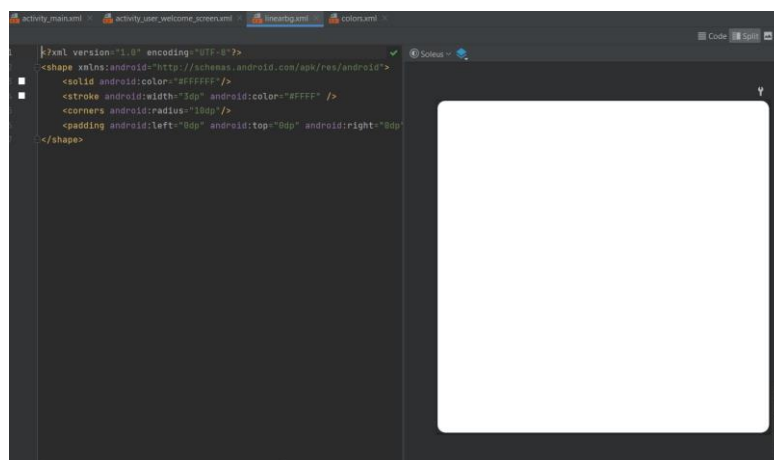


Figura 11. Ejemplo de recursos. Imagen propia

6.2 Modelos y planificación

El paso previo al comienzo de la escritura del código fue la planificación de los agentes que interactuarían en la aplicación, principalmente comenzamos con el modelo de datos y el modelo de las entidades.

6.2.1 Modelo de datos

Al ser el objetivo principal de la aplicación el envío de peticiones por parte de clientes a los departamentos de un hotel, esas peticiones tendrán que ser almacenadas en una base de datos, sobre la que luego se realizarán las consultas pertinentes. Dentro de la base de datos podremos encontrar también a los propios usuarios, con los datos necesarios para su utilización en la aplicación. De esta manera, dentro de nuestra base de datos Soleus encontramos las dos tablas siguientes:

| Nombre del campo | Tipo |
|------------------|---------|
| room | Varchar |
| password | Varchar |
| name | Varchar |
| department | Varchar |

Figura 12. Tabla user. Elaboración propia

- room: como el nombre indica, almacena el valor del número de habitación (identificador para los empleados). Es la clave primaria de la tabla.
- password: contiene el valor encriptado de la contraseña que se asigna a cada usuario. Esta contraseña puede ser modificada por el usuario administrador y por el propio cliente si se trata de un usuario cliente.

- Se introdujeron algunos datos de prueba tras la creación de la base de datos, creando 4 usuarios cliente (101, 102, 103, 104, posteriormente en la imagen adicionamos el 106 como dato de prueba), cuatro empleados (HK1, HK2, MT1, MT2) y un administrador.

Figura 13. Tabla user, datos. Imagen propia

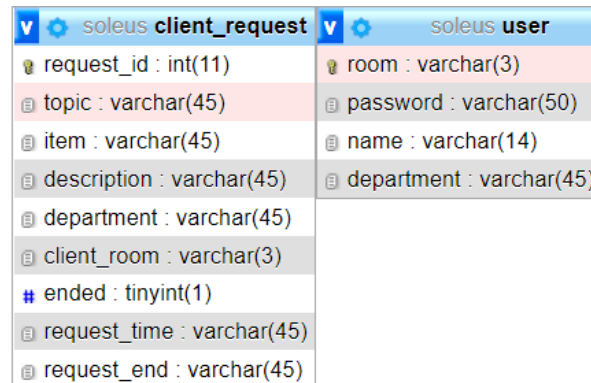
Página 22 de 54

| | |
|--------------|---------|
| client_room | Varchar |
| ended | Tinyint |
| request_time | Varchar |
| request_end | Varchar |

Figura 14. Tabla client_request. Elaboración propia

- request_id: Clave autogenerada con la que se identifica una petición.
- topic: Asunto de la petición. Es una generalización del asunto sobre el que puede tratar la petición del cliente, opciones como Baño, Cama, Limpieza, etc.
- item: Asociado al *topic*, es una especificación del motivo de la petición. Siguiendo el ejemplo anterior: toalla extra, cambio de sábanas, etc.
- description: Breve descripción con caracteres limitados en la aplicación para que el cliente pueda dar detalles adicionales. Ejemplo: introducir el número de artículos necesarios.
- client_room: Usuario origen de la petición, aquel que la ha realizado.
- ended: Booleano que nos indica si una petición ha sido cancelada o finalizada por el empleado. False inicialmente (0).
- request_time: Fecha en formato cadena de texto que incluye la hora, el día, el mes y el año en el que se realiza la petición.
- request_end: Fecha en formato cadena texto que incluye la hora, el día, el mes y el año en el que se termina o cancela la petición. Nula hasta que ended pase a ser true (1).

Al ser la ventana de creación de peticiones una de las primeras vistas que se programaron para la aplicación, esta tabla no contiene datos de prueba, se recuperarían posteriormente los datos generados en las pruebas realizadas durante el desarrollo.



| soleus client_request | soleus user |
|----------------------------|--------------------------|
| request_id : int(11) | room : varchar(3) |
| topic : varchar(45) | password : varchar(50) |
| item : varchar(45) | name : varchar(14) |
| description : varchar(45) | department : varchar(45) |
| department : varchar(45) | |
| client_room : varchar(3) | |
| ended : tinyint(1) | |
| request_time : varchar(45) | |
| request_end : varchar(45) | |

Figura 15. Vista de las tablas en phpMyAdmin. Imagen propia

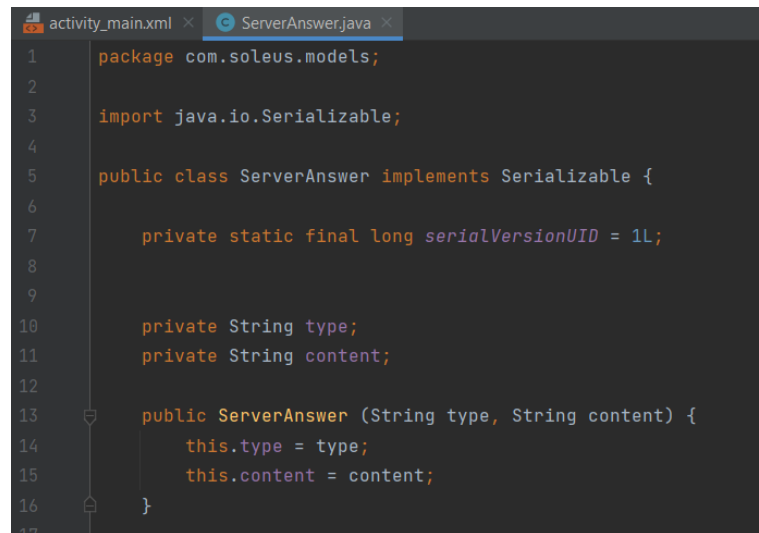
6.2.2 Entidades

Soleus no sólo necesitó de la generación de clases modelo para las entidades de la base de datos, sino que durante el desarrollo se hicieron necesarias dos adicionales. La aplicación hace uso de cuatro entidades en total: RoomRequest, UserModel, ClientRequest y ServerAnswer.

Detallaremos brevemente sus usos para mejor comprensión de los puntos posteriores:

- RoomRequest: Modelo para las peticiones, persisten en la tabla client_request. Cuentan con parámetros similares a los de la tabla de la base de datos y un id de serialización para poder ser enviada y recibida en interacciones con el servidor.
- UserModel: Modelo para usuarios, persiste en la tabla user. Al igual que RoomRequest, tiene un id de serialización y sus parámetros son similares a los de la base de datos. Ambos modelos cuentan con diferentes constructores que se utilizan a lo largo del código.

- ClientRequest: Modelo para las peticiones que hace el cliente (aplicación móvil) al servidor (código sin interfaz gráfica). Especifica al servidor el tipo de acción a realizar para que este se prepare para recibir los datos correctos.
- ServerAnswer: Modelo de respuesta del servidor, recibido por la aplicación para conocer el estado de las *ClientRequest* que envíe. Estas dos últimas entidades no persisten en la base de datos ya que son utilizadas únicamente para el desarrollo de actividades de la aplicación.



```
1 package com.soleus.models;
2
3 import java.io.Serializable;
4
5 public class ServerAnswer implements Serializable {
6
7     private static final long serialVersionUID = 1L;
8
9
10    private String type;
11    private String content;
12
13    public ServerAnswer (String type, String content) {
14        this.type = type;
15        this.content = content;
16    }
17 }
```

Figura 16. Captura del modelo ServerAnswer. Imagen propia

6.2.3 Vista general de la aplicación: Diagrama de casos de uso

Durante los próximos puntos (a partir de la sección 6.3) comenzaremos a revisar las diferentes vistas de la aplicación añadiendo detalle sobre su funcionamiento y si necesaria explicación de algunas líneas de código. Para poder comprender el funcionamiento general de la aplicación y como hoja de ruta, podemos ver en la imagen siguiente el diagrama de casos de uso inicial que se diseñó con el propósito del desarrollo de la aplicación.

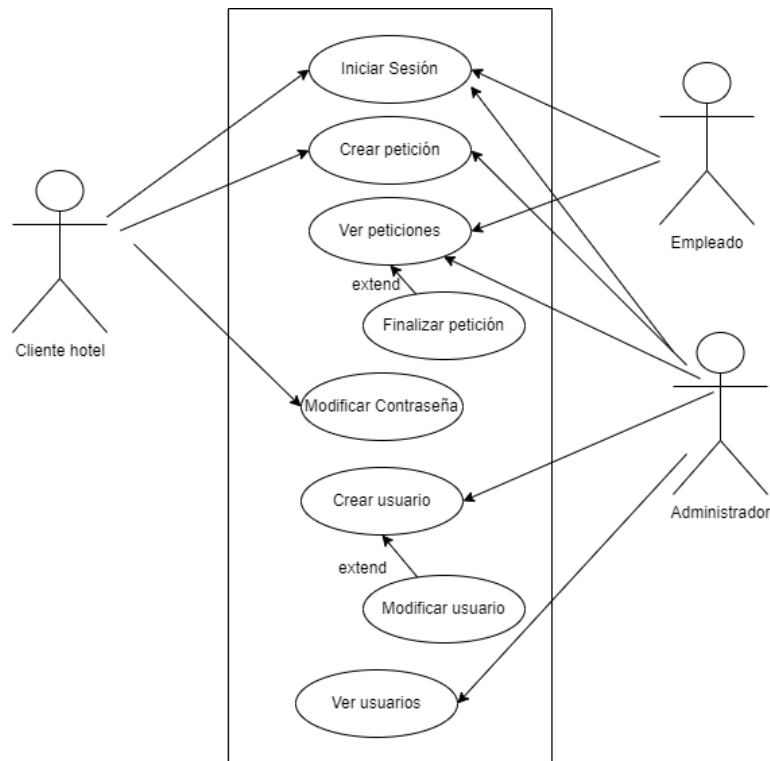


Figura 17. Diagrama inicial de casos de uso. Elaboración propia mediante Draw.io

6.3 Actividades, vistas y clases

A lo largo de este apartado recorreremos las diferentes actividades/vistas que la aplicación pone a disposición de los usuarios, explicando su funcionamiento y en algunos casos, si necesario, añadiendo líneas tomadas directamente del código fuente de la aplicación. Este recorrido se hará de manera secuencial, en orden de aparición, distinguiendo entre los tres diferentes tipos de usuario que pueden interactuar con la aplicación: clientes, empleados y administradores. Es además el mismo recorrido que se ha seguido durante el desarrollo, por lo que es al mismo tiempo un seguimiento de éste.

6.3.1 ClientNet: Clase cliente

Antes de iniciar el recorrido, es necesario explicar el propósito de la clase ClientNet de la aplicación, una clase que se comenzó a desarrollar desde el inicio y que se ha ido actualizando hasta la actualización de las últimas funciones de Soleus. ClientNet es una clase que implementa la interfaz Runnable y cuyos constructores son llamados desde las otras clases de la aplicación cada vez que es necesario el envío o recepción de contenido de la base de datos. Esta clase se encarga de comunicarse con el servidor, incluyendo todos los métodos necesarios para esto, lo que supone además la inclusión de métodos que abren otras actividades. Durante el desarrollo de los próximos apartados, se hablará de esta clase cada vez que se deba enviar o recibir información involucrando al servidor.

6.3.2 Vistas de clientes

La primera interacción del cliente con la aplicación es la actividad principal, `activity_main`, una vista sencilla en la que se incluyen el logo de Soleus, dos campos de texto editables y dos botones, uno de ellos siendo una etiqueta de texto (`TextView`). En esta actividad el cliente tiene dos opciones: introducir su usuario y contraseña para acceder a Soleus o modificar su contraseña en caso de haberla olvidado. Al introducir el nombre de usuario y contraseña correctos (comprobación realizada con la ayuda de ClientNet), se abrirá la siguiente actividad, `activity_user_welcome_screen`. Es importante destacar que mediante ClientNet y la clase Utils (se desarrollará en puntos posteriores) se encripta la contraseña del usuario, que si bien está en principio pensada como una contraseña establecida por el administrador y por lo tanto no debería ser compartida con otros de los servicios del que haga uso el cliente, podría darse el caso al modificarla. Por ello, para garantizar la seguridad de la información, la base de datos no guarda este tipo de información confidencial sin ser primero encriptada.

```

public void onClick(View view) {

    switch (view.getId()) {

        case R.id.btnMainLogin:
            String user = editUser.getText().toString().trim();
            String password = editPassword.getText().toString().trim();
            if (TextUtils.isEmpty(user) || TextUtils.isEmpty(password)) {
                Utils.showEmptyFieldsToast(context: this);
            } else {
                UserModel userModel = new UserModel(user, password);
                Thread login = new Thread( new ClientNet(userModel, "LOGIN", view, activity: this)) ;
                login.start();
            }
            break;

        case R.id.txtMainForgotPassword:
            Intent openForgotPassword = new Intent( packageContext: this, ForgotPasswordActivity.class);
            startActivity(openForgotPassword);
            break;

    }

}

```

Figura 18. Eventos lanzados al pulsar los botones. Imagen propia



Figura 19. activity_main. Imagen propia

He olvidado mi contraseña: activity_forgot_password

Si hacemos click por otra parte en el botón de olvido de contraseña, se nos abrirá la vista que podemos ver en la próxima imagen. En esta vista el cliente tiene a su disposición tres cuadros de texto (EditText) que debe rellenar de manera satisfactoria para poder modificar su contraseña. Los datos se contrastan con la base de datos gracias a ClientNet y son actualizados en la misma. Si la operación tiene un resultado satisfactorio, se autentica automáticamente al usuario y nos dirigimos a la misma actividad a la que nos dirigíamos tras realizar un login satisfactorio.



Figura 20. activity_forgot_password. Imagen propia

Bienvenida de usuario *logueado*: activity_user_welcome_screen

Una vez autenticados, la aplicación nos dispone tres diferentes opciones: la creación de una nueva petición para el departamento de mantenimiento, la creación de una petición para el departamento de pisos y la revisión de las peticiones pendientes que tenemos. Sólo esta última opción requiere de un llamado a la clase ClientNet, ya que se debe realizar para poder mostrarnos la siguiente actividad. La creación de peticiones no interactúa con el servidor hasta que estas son creadas dentro de su propia actividad. Esto se puede ver en la figura 22, donde al hacer click sobre el botón “Mis peticiones” creamos un nuevo hilo (*Thread*) de ClientNet.



Figura 21. activity_user_welcome_screen. Imagen propia

```

public void onClick(View view) {

    switch (view.getId()) {
        case R.id.btnWelcomeHkRequest:
            Intent openHousekeepingRequest = new Intent( packageContext: this, HousekeepingRequestActivity.class);
            openHousekeepingRequest.putExtra( name: "userLogged", userLogged);
            startActivity(openHousekeepingRequest);
            break;
        case R.id.btnWelcomeMtRequest:
            Intent openMaintenanceRequest = new Intent( packageContext: this, MaintenanceRequestActivity.class);
            openMaintenanceRequest.putExtra( name: "userLogged", userLogged);
            startActivity(openMaintenanceRequest);
            break;
        case R.id.btnWelcomemyRequests:
            Thread getRoomRequests = new Thread( new ClientNet(userLogged,
                "GET_RR_LIST", view, activity: this)) ;
            getRoomRequests.start();
            break;
    }
}

```

Figura 22. Eventos relacionados con los botones de la actividad. Imagen propia

Nuevas peticiones: activity_maintenance_request, activity_housekeeping_request

Si hacemos click en cualquiera de los dos botones “Pisos” o “Mantenimiento”, se nos abrirá una ventana permitiéndonos la creación de una nueva petición, que irá dirigida al departamento que escojamos. Estas ventanas son similares en diseño, cambiando únicamente el contenido de las listas desplegables (Spinners) para adaptarse al departamento correcto. Ambas incluyen además un botón “Enviar” con el que se enviará el contenido de nuestra petición al servidor, usando ClientNet, creando con nuestra petición una instancia del modelo RoomRequest.

```

/* Populating the topic Spinner */
topicSpinnerAdapter = new ArrayAdapter<>( context: this,
    R.layout.spinners, topics);
topicSpinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerTopic.setAdapter(topicSpinnerAdapter);

/* Preparing adapters to be used by item spinner on onItemSelected() */
cleaningSpinnerAdapter = new ArrayAdapter<>( context: this,
    R.layout.spinners, cleaningSpinnerContent);
cleaningSpinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

bedSpinnerAdapter = new ArrayAdapter<>( context: this,
    R.layout.spinners, bedSpinnerContent);
bedSpinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

bathroomSpinnerAdapter = new ArrayAdapter<>( context: this,
    R.layout.spinners, bathroomSpinnerContent);
bathroomSpinnerAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

// Populates the items spinner based on the topic selected on the first spinner */
spinnerTopic.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        String spinnerTopicValue = spinnerTopic.getSelectedItem().toString();
        if (spinnerTopicValue.equals(topics[0])) {
            spinnerItem.setAdapter(cleaningSpinnerAdapter);
        } else if (spinnerTopicValue.equals(topics[1])) {
            spinnerItem.setAdapter(bathroomSpinnerAdapter);
        } else {
            spinnerItem.setAdapter(bedSpinnerAdapter);
        }
    }
}

```

Figura 23. Los spinners secundarios se pueblan en base al asunto escogido. Imagen propia

Ver mis peticiones: activity_worker

Si bien su nombre se debe a que inicialmente fue pensada como una actividad exclusiva de los usuarios de empleado, se ha adaptado también como vista para que los clientes puedan ver sus propias peticiones, habiéndose añadido esta funcionalidad en la última actualización, tras haber ya completado el diseño final de la aplicación. Mediante una línea adicional en el servidor, este nos envía una lista con las peticiones asociadas al usuario. Debido a que es la actividad principal

de los empleados, se desarrollará más en el apartado correspondiente, siendo únicamente el contenido aquello que se mostrará de manera diferente.

6.3.3 Vistas de empleado

Al igual que los usuarios de cliente, los empleados deben acceder usando la actividad principal, introduciendo su usuario y contraseña. En su caso, no podrán hacer uso de la opción de modificar contraseña, teniendo que acudir al administrador. En el transcurso de su inicio de sesión, ClientNet envía el usuario al servidor que verifica el departamento al que pertenece y lanza una consulta a la base de datos para obtener una lista de las peticiones (RoomRequest) activas del departamento. Tras ello, ClientNet recibe la lista e invoca un método para pasar la lista a la siguiente actividad, que se nos abrirá, presentándonos las peticiones.

```
private void checkUserLogin(ObjectOutputStream output, ObjectInputStream inputStreamReader)
    throws IOException, ClassNotFoundException {

    System.out.println("cliente conectado"); // DEBUG

    userReceived = (UserModel) reader.readObject();

    hibernateUsers = new UserModelDAO();
    hibernateRequests = new RoomRequestDAO();

    if (hibernateUsers.checkUserCredentials(userReceived.getUser(), userReceived.getPassword())) {

        writer.writeObject(successAnswer);
        userLogged = hibernateUsers.getUserModel(userReceived.getUser());
        writer.writeObject(userLogged);

        if (userLogged.getDepartment().equals(housekeepingDepartment)
            || userLogged.getDepartment().equals(maintenanceDepartment)) {
            requestList = hibernateRequests.getRequestList(userLogged);
            writer.writeObject(requestList);
        }

    } else {
        writer.writeObject(failAnswer);
    }
}
```

Figura 24. Método que realiza el login en el servidor. Imagen propia

Actividad de empleado: activity_worker

Esta vista está formada por la cabecera habitual, con un título y el logo de la aplicación, y un RecyclerView que nos servirá para presentar las peticiones activas del departamento. Cuenta además con un botón en la parte inferior de la pantalla para actualizar la lista, y es que al dejar pulsada (onLongClick) cualquier petición, el empleado tendrá la opción de marcar la petición como finalizada, desapareciendo esta de la lista de peticiones al actualizarla (sólo se muestran las peticiones de interés para el empleado, las activas). Esta opción sirve además para que el cliente pueda cancelar sus propias peticiones si ha cometido algún error al enviarlas.

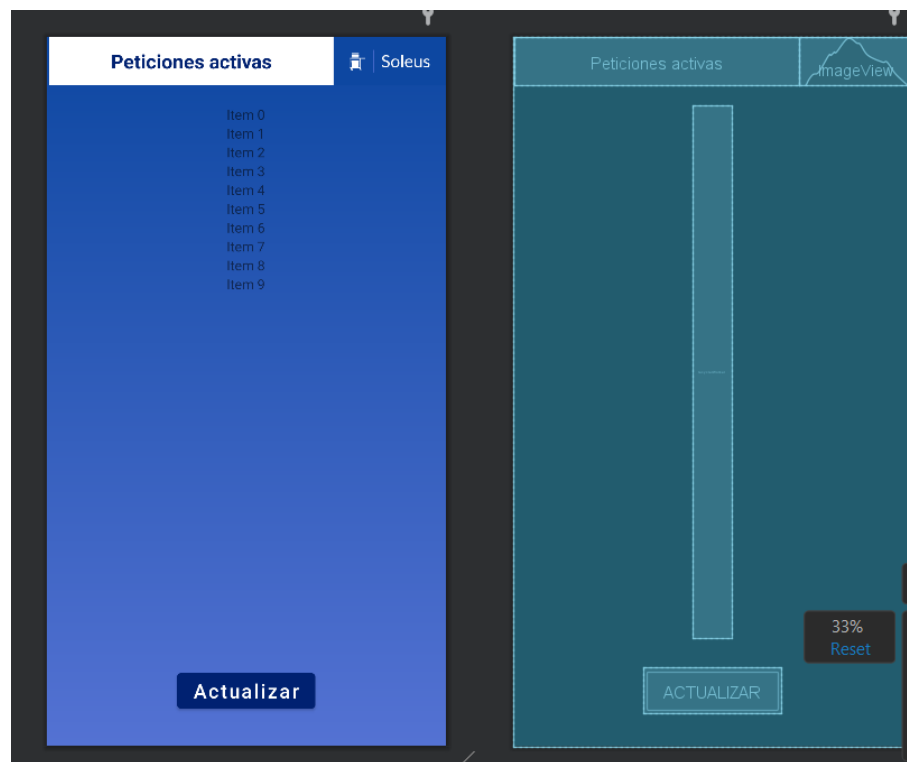


Figura 25. activity_worker. Imagen propia

En la imagen anterior podemos ver el RecyclerView en la parte central de la vista. Este se puebla con el diseño predeterminado de las vistas creadas en el archivo room_request_card,

mostrándonos los siguientes datos: ID de la petición, número de la habitación que haya realizado la petición, motivo de la misma (item), descripción, hora de realización y hora de finalización (si no ha sido finalizada saldrá como “Pendiente”. Además de ello, el elemento donde se nos muestra la hora de finalización tendrá un fondo diferente dependiendo de si la actividad sigue activa o no, siendo rojo o verde dependiendo del caso: rojo si no ha sido marcada como finalizada y verde en caso contrario.



Figura 26. Peticiones activas del departamento de pisos. Imagen propia

En la imagen anterior podemos observar las peticiones activas del departamento de pisos, si algún empleado del mismo las marcara como finalizadas, estas desaparecerían de la lista al ser

actualizadas, siendo posible ver la fecha y hora de finalización si se utiliza un usuario administrador.



Figura 27. AlertDialog permitiendo marcar las peticiones como finalizadas. Imagen propia

Esta clase se basta en todo momento de la clase RoomRequestAdapter que nos sirve para poblar el RecyclerView de manera adecuada, indicando en qué componente deben aparecer los datos. Al marcar como finalizada una petición, la clase ClientNet se encarga de informar al servidor del id de la petición a finalizar.

```

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    holder.txtIdCard.setText(String.valueOf(roomRequestList.get(position).getRequestId()));
    holder.txtRoomCard.setText(roomRequestList.get(position).getClientRoom());
    holder.txtItemCard.setText(roomRequestList.get(position).getRequestItem());
    holder.txtDescriptionCard.setText(roomRequestList.get(position).getRequestDescription());
    holder.txtTimeCard.setText(roomRequestList.get(position).getRequestTime());
    if (roomRequestList.get(position).isRequestEnded()) {
        holder.txtEndedCard.setText(roomRequestList.get(position).getRequestEndTime());
        holder.txtEndedCard.setBackgroundResource(R.drawable.textviewbordergreen);
    } else {
        holder.txtEndedCard.setText("Pendiente");
        holder.txtEndedCard.setBackgroundResource(R.drawable.textviewborderred);
    }
}
} // onBindViewHolder

```

Figura 28. Extracto de RoomRequestAdapter

6.3.4 Vistas de administrador

Actualmente sólo existe un usuario de tipo administrador en la base de datos de la aplicación, sólo uno por lo tanto será aceptado como tal y podrá tener acceso a las vistas que se expondrán en los próximos puntos. Es posible que en un caso real, en el que se cuente con jefes de departamento, fuese interesante la creación de más administradores o de usuarios semi-administradores, capaces de modificar únicamente cuestiones de su propio departamento.

Tal y como el resto de usuarios, el administrador debe primero autenticarse en la actividad principal, siendo enviados sus datos al servidor mediante ClientNet. Una vez comprobado que

los datos son correctos, la aplicación verifica que el departamento del usuario sea el de administración y abre la siguiente actividad inicial: activity_admin.

```
private void checkLogin(ObjectOutputStream writer, ObjectInputStream reader, Socket client,
    UserModel login) throws ClassNotFoundException {

    try {
        login = Utils.encrypt(login);
        writer.writeObject(login);
        serverAnswer = (ServerAnswer) reader.readObject();
        if (serverAnswer.getType().equals(successAnswer)) {
            userReceived = (UserModel) reader.readObject();
            if (userReceived.getDepartment().equals(clientLogged)) {
                openUserWelcome();
                client.close();
            } else if (userReceived.getDepartment().equals(housekeepingLogged) ||
                userReceived.getDepartment().equals(maintenanceLogged)) {
                roomRequestList = (List<RoomRequest>) reader.readObject();
                openWorkerActivity(userReceived);
                client.close();
            } else if (userReceived.getDepartment().equals(adminLogged)) {
                openAdminActivity();
                client.close();
            }
        }
    }
}
```

Figura 29. El método checkLogin de la clase ClientNet se encarga de interactuar con el servidor y verificar el departamento del usuario. Imagen propia.

Vista inicial del administrador: activity_admin

Esta actividad es la bienvenida del usuario administrador, funcionando como cuadro de mandos desde el que pueden modificarse la mayor parte de los datos de Soleus. Partiendo de dos

botones principales, Usuarios y Peticiones, el administrador podrá elegir opciones más específicas de ambos al pulsar alguno de los dos.



Figura 30. activity_admin. Imagen propia

Haciendo click en cualquiera de los dos botones de la imagen, se nos desplegarán nuevos botones para poder escoger acciones relacionadas con el botón pulsado: si pulsamos en usuario nos aparecerán dos nuevos botones para ver la lista de usuarios o crear uno nuevo, si pulsamos sobre peticiones nos ocurrirá algo similar, permitiéndonos ver la lista de peticiones o crear

nuevas, si pulsamos esta última opción nos dará a elegir el departamento al que se realizará la petición, tal y como en la pantalla de bienvenida de clientes.

Iniciaremos la explicación por las vistas relacionadas con las opciones de usuarios, comenzando con la opción “Ver” usuarios.



Figura 31. Opciones relacionadas con usuarios. Imagen propia.

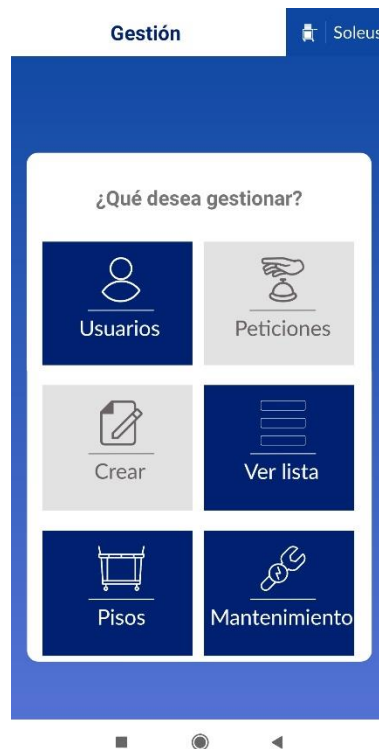


Figura 32. Opciones relacionadas con las peticiones. Imagen propia.

Gestión de usuarios: `activity_user_manager`

Al hacer click sobre el botón, se nos abre una vista en la que se dispone una pantalla similar a aquella en la que se nos muestran las peticiones activas de un departamento, cambiando el contenido del RecyclerView y el diseño de las vistas en las que se nos mostrará el mismo. Cuenta además con un menú diferente que response al evento `onLongClick`, dándonos la opción de borrar usuario o de modificarlo, abriéndonos la vista de creación de usuario con los datos precargados del usuario escogido (a excepción de la contraseña) para ser modificados. Una vez

más, si decidimos borrar un usuario, ClientNet enviará al servidor el usuario escogido y este realizará la modificación en la base de datos.

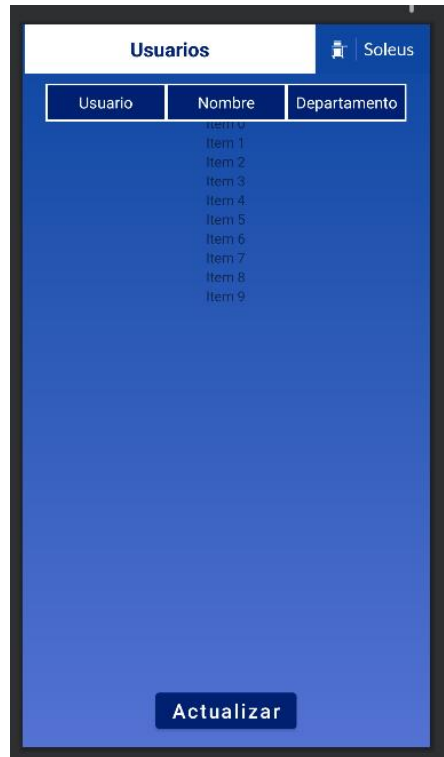


Figura 33. activity_user_manager. Imagen propia

Al igual que en la vista de lista de peticiones, los datos de los usuarios existentes se muestran en el RecyclerView siguiendo el diseño predefinido de otra vista: user_model_card.



| Usuario | Nombre | Departamento |
|---------|---------|--------------|
| 101 | LUNA | CLIENT |
| 102 | PEREIRA | CLIENT |
| 103 | LUNA | CLIENT |
| 104 | PEREIRA | CLIENT |
| 106 | LUNA | CLIENT |

Actualizar

Figura 34. Usuarios existentes. Imagen propia



Figura 35. Menú ofrecido al dejar pulsado un usuario. Imagen propia

Crear usuario: activity_create_or_modify_user

Tanto si en la actividad de bienvenida del administrador pulsamos “Crear” usuario como si decidimos darle a modificar un usuario en la lista del apartado anterior, se nos abrirá una nueva vista con la cabecera habitual, cuatro campos de texto (EditText) y un botón. Esta vista está creada para poder introducir los datos de un nuevo usuario que deseemos crear (usuario, contraseña, nombre y departamento). Los datos se enviarán al servidor que verificará si existe en la base de datos, actualizándolo en ese caso o creándolo en caso contrario, informándonos Soleus con un Toast si la operación ha sido completada con éxito. La clase asociada a esta vista, de nombre similar a la propia activity, se encarga de verificar que los datos introducidos en los

campos de texto sean válidos (que el departamento sea uno de los preestablecidos, que la contraseña tenga un mínimo de caracteres, etc.).



The image shows a web interface for user management. At the top, there is a header with the text 'Gestión de usuarios' and a logo for 'Soleus'. Below the header, there is a form with four input fields: 'Nuevo usuario', 'Contraseña', 'Nuevo nombre', and 'Departamento'. Below the fields is a blue button labeled 'Enviar'.

Figura 36. activity_create_or_modify_user. Imagen propia

Lista de peticiones: activity_room_request_manager

Para evitar redundancias, no se cubrirán en este documento las actividades relacionadas con la creación de peticiones por parte del administrador, ya que estas son similares y utilizan las mismas vista que aquellas utilizadas por los usuarios de cliente. Hay sin embargo una diferencia al elegir la opción de ver la lista de peticiones, disponiendo el administrador de dos opciones: o bien ver una lista completa, sin filtrar, en la que se muestran todas las peticiones activas o no con su fecha de finalización; actividad similar a la lista de peticiones que pueden ver los empleados; o bien de filtrar las peticiones para ver aquellas activas según el item que se desee. Para ello se

tendrá que pulsar sobre el botón “Filtrar”, apareciendo tras ello un Spinner con los diferentes asuntos disponibles a la hora de crear peticiones. Esto nos enviará de nuevo a la habitual vista de la lista de peticiones (activity_worker), mostrando el resultado del filtro.



Figura 37. activity_room_request_manager. Imagen propia



Figura 38. Listado de peticiones sin filtrar. Imagen propia

6.3.5 Clase Utils

La clase Utils es utilizada a lo largo de la ejecución de Soleus para principalmente dos actuaciones: encriptado de la contraseña de los usuarios y generación de Toasts para informar del resultado de las operaciones (creación correcta de nuevo usuario, creación correcta de peticiones, etc.). Por ello la clase utiliza métodos estáticos requiriendo los parámetros necesarios.

| |
|--|
| Utils |
| <pre> - secretKey : SecretKey {readOnly} - decodedKey : byte[] {readOnly} - key : String {readOnly} ~ cipher : Cipher - forbidden : String {readOnly} - wrongUser : String {readOnly} - modifiedUser : String {readOnly} - wrongDepartment : String {readOnly} - wrongLength : String {readOnly} - emptyFields : String {readOnly} - userCreatedConfirmation : String {readOnly} - requestSentConfirmation : String {readOnly} - severErrorToast : String {readOnly} - toastFailedLogin : String {readOnly} - Utils() + showToastFailedLogin(context : Context) : void + showServerErrorToast(context : Context) : void + showRequestSentToast(context : Context) : void + showCreatedUserToast(context : Context) : void + showEmptyFieldsToast(context : Context) : void + showWrongLengthToast(context : Context) : void + showWrongDepartmentToast(context : Context) : void + showModifiedUserToast(context : Context) : void + showWrongUser(context : Context) : void + showForbiddenUser(context : Context) : void + encrypt(userModel : UserModel) : UserModel + decrypt(userModel : UserModel) : UserModel </pre> |

Figura 39. Diagrama de clase de Utils. Generación propia

6.3.6 Servidor

La finalidad final del desarrollo anteriormente descrito es la generación de una aplicación funcional, atractiva e intuitiva, siendo el servidor algo necesario pero no protagonista al mismo nivel que la aplicación móvil. Es por ello que el servidor se limita a dar apoyo a la aplicación respondiendo a sus llamadas, registrando en la base de datos las acciones necesarias. El servidor está desarrollado en Java, utilizando un ordenador en la red local como tal, gracias al entorno de desarrollo Eclipse y no cuenta con interfaz gráfica en la actualidad más allá de una simple ventana con un botón para iniciar el servidor. Consta de una clase Server en la que se inicia nuestro servidor en el puerto establecido, de una clase ServerThread en la que podemos encontrar todos los métodos que se inician cada vez que el servidor es contactado por el cliente y cuenta además con los modelos de las entidades que participan en la ejecución y con el CRUD

de las tablas de ambas entidad, creado gracias a la ayuda de la herramienta Hibernate. El código se encuentra a disposición en un repositorio público propio del siguiente enlace:

<https://github.com/franciscoluna94/Soleus-Desktop>.

7. Fase de pruebas

Al ser un proyecto realizado Android nativo, sin la ayuda de herramientas como Flutter, escrito en Java, se han ido realizando pruebas continuas a lo largo del desarrollo. Tras la escritura de cada uno de los métodos o tras la mínima modificación del diseño, la aplicación se reinstaló en el móvil de pruebas y se volvió a lanzar. De la misma manera, cada método escrito en el servidor se probó utilizando la aplicación móvil e incluso reiniciando en ocasiones el servidor de la base de datos. Por ello, no hay una documentación detallada y continua de las pruebas, y en cualquier caso ocuparía una extensión prácticamente similar al documento actual. Sí podemos sin embargo destacar grandes cambios o errores que han surgido durante el desarrollo.

Inicialmente el servidor y el cliente interactuaban enviándose tipos primitivos, por lo general cadenas de texto. Estas cadenas de texto se extraían de los objetos en el cliente y luego volvían a instanciar uno en el servidor. Esto fue modificado posteriormente (ver árbol de commits en los enlaces al repositorio) para poder enviarse objetos y así reducir el número de líneas de código y hacer mucho más fácil el desarrollo de nuevos métodos. Tras esta modificación, se enviaron erróneamente múltiples peticiones al servidor en un formato incorrecto, significando errores relacionados con Hibernate. Esto terminó afectando a la estructura de la base de datos, impidiendo el grabado de nuevos objetos en ella e impidiendo la lectura de los existentes. Más allá de eso, impidió además finalmente el inicio de MySQL para poder acceder a la base de datos. Para solucionarlo, tras consultar varias veces fuentes de internet, hubo que modificar la configuración en los ficheros instalados en el ordenador y una vez pudimos volver a levantar el servidor, hubo que borrar y rehacer la base de datos al completo.

Un error tal vez menor comparado con el anterior pero que resultó ser igualmente problemático fue el cambio de icono de la aplicación móvil, que al importar los archivos a Android Studio generó un conflicto e impidió cualquier ejecución de la aplicación por errores del Gradle. En ese momento había prácticamente finalizado la codificación ese día pero aún no había sido guardada con un commit, lo que significaba una pérdida grande de información al hacer un Rollback. Hubo finalmente que borrar múltiples archivos de diseño para poder volver a ejecutar Soleus.

8. Conclusiones y trabajos futuros o posibles mejoras

8.1 Conclusiones

Al iniciar el desarrollo, nuestros objetivos eran la creación de una aplicación intuitiva y útil tanto para clientes como empleados de un hotel/establecimiento hotelero, que permitiese tener un control de las peticiones realizadas. Habiendo finalizado el desarrollo, podemos ver que el proceso ha sido satisfactorio, teniendo como resultado una aplicación atractiva y sencilla, funcional, que puede adaptarse a un lugar de trabajo como el mencionado. Otro de los objetivos, el ampliar conocimientos sobre las tecnologías usadas, se ha cumplido también, teniendo que profundizar en conceptos como las vistas de Android Studio, aprendiendo más sobre nuevos componentes o la estructura cliente-servidor, utilizando de una manera más realista Hibernate en Eclipse y especialmente mejorando conceptos de diseño respecto a las aplicaciones Android, habiendo tenido que realizar todo el diseño sin ayuda de frameworks externos más allá de Android Studio y su herramienta de diseño. Soleus es el resultado del trabajo que se ha descrito en este documento, permitiéndonos además realizar nuevas mejoras en el futuro.

8.2 Posibles mejoras

A pesar de que el resultado sea satisfactorio partiendo de los objetivos iniciales, no podemos ser ajenos a las tendencias de la industria ni ser conformistas con el mismo. Es innegable que las aplicaciones web están muy presentes actualmente, siendo muy útil una interfaz gráfica disponible de esta manera. De esta forma, las siguientes mejoras podrían marcar la hoja de ruta de Soleus en el futuro:

- Creación de un web api: pensada principalmente para empleados y administradores, un web api permitiría desde los ordenadores de los empleados acceder a las mismas funciones de la aplicación, facilitando la gestión de las peticiones activas. Actualmente, en mis prácticas formativas, estoy trabajando con .NET (C#) y las functions de Azure, lo que podría permitirme si sigo desarrollando mis conocimientos crear esta aplicación web en el futuro utilizando C#.
- Modificación del servidor: Actualmente, debido a la arquitectura de la aplicación esta sólo funciona si ambos dispositivos (servidor y aplicación) se encuentran conectados a la misma red. Esto puede ser útil para el hotel e incluso intencionado ya que así se evitaría cualquier acceso desde puntos diferentes al establecimiento, pero podría ser interesante ofrecer la oportunidad a los clientes de realizar sus peticiones desde fuera del hotel. Para ello habría que modificar el sistema de autenticación y el propio servidor, que podría así además encontrarse en un lugar diferente al del propio establecimiento o usar una red privada de los dispositivos de trabajo.

9. Referencias bibliográficas

calendamaia. (10 de Enero de 2014). *Genbeta*. Obtenido de <https://www.genbeta.com/desarrollo/eclipse-ide>

Sitio Web de Android Studio. (2022). Obtenido de <https://developer.android.com/studio/features>

Vázquez, J. (s.f.). *Observatorio de Tecnología Educativa*. Obtenido de https://intef.es/observatorio_tecno/draw-io-mucho-mas-que-mapas-mentales

VMware. (s.f.). *ApacheFriends*. Obtenido de <https://www.apachefriends.org/es/index.html>

Zúñiga, F. G. (25 de Noviembre de 2021). *Arsys*. Obtenido de <https://www.arsys.es/blog/phpmyadmin>