IES Francisco de Quevedo
Instituto bilingüe de la Comunidad de Madrid

UNIÓN EUROPEA
FONDO SOCIAL EUROPEO
INVIERTE EN TU FUTURO

IHOUSE

JUAN JOSÉ GARCÍA SÁNCHEZ

2º DESARROLLO DE APLICACIONES EN MULTIPLATAFORMA

18 / 06 / 2021

Índice

1. Introducción

IHOUSE es un proyecto de domótica básica, desarrollado como parte de la formación del CFGS Desarrollo de Aplicaciones en Multiplataforma.

La finalidad de esta aplicación es la de, con un bajo presupuesto, lograr domotizar una vivienda entera. IHOUSE se encarga de controlar dispositivos físicos dispuestos en una vivienda, a través de diferentes peticiones a un servidor web ya creado.
Las principales funcionalidades son las de mover motores o persianas y apagar y encender aparatos y luces.

La aplicación tiene la posibilidad de conectarse a cualquier vivienda que conste de una placa Raspberry PI que contenga el servidor web, conectada por cable o por WIFI a la misma. La conexión se realiza mediante la obtención e inserción de la dirección URL de la placa en el servidor web.

Al ejecutar por primera vez el servidor web, se crea por defecto un usuario administrador, con nombre y contraseña “root”, que será quien añada y administre a los demás usuarios o “miembros” de la casa, y todos los aparatos que la conforman; también tiene la posibilidad de crear nuevamente otros administradores.

Asimismo, el “administrador” es el único tipo de usuario que tiene la posibilidad de realizar cambios en cuanto a habitaciones, componentes y miembros de la casa; es decir, el único que podrá simular la distribución de la vivienda añadiendo, eliminando y actualizando.

En este proyecto se ponen en práctica competencias y conocimientos obtenidos en varios módulos cursados durante el ciclo, como se detalla a continuación.

La aplicación cliente está desarrollada en Android con lenguaje Java 1.8, haciendo uso de los conocimientos adquiridos en Programación Multimedia y Dispositivos Móviles.
El servicio web al que se conecta la aplicación cliente se ha desarrollado en base a lo aprendido en Programación de Servicios y Procesos.
Por último, la BBDD que almacena toda la información de los usuarios, habitaciones y componentes está diseñada y manipulada gracias a las competencias adquiridas en el módulo de Acceso a Datos.

2. Justificación del proyecto y objetivos

La ambición por llevar a cabo esta idea surge de un conjunto de carencias del mercado. La domótica es un campo de la tecnología caracterizado por su elevado coste a la hora de adquirir los dispositivos necesarios para hacerse con una buena instalación. Tiene un carácter complejo y la inversión necesaria no está al alcance de cualquiera, a pesar de los buenos resultados que esta pueda ofrecer; por ello poca gente se plantea llevar a cabo una reforma de este tipo en su hogar.

IHOUSE es un proyecto ideado con la finalidad de proporcionar comodidad y eficiencia en el día a día. Su punto de interés es el bajo presupuesto que requiere, pudiéndose instalar casi en cualquier vivienda, y pudiendo ser usada, por la sencillez de su interfaz, por casi cualquier persona, de cualquier edad. Como ejemplo: una persona con discapacidad o de edad avanzada, que no tenga a su alcance la movilidad o la fuerza que normalmente posee una persona, puede salir altamente beneficiada de una aplicación como esta.

Estos son factores con mucho peso debido a que el mercado actual de la domótica no está excesivamente desarrollado y en la mayoría de hogares no existe este tipo de tecnología.
Si la aplicación lograra llegar a una cantidad amplia de personas, la domótica podría tomar un camino muy prometedor y globalizado, proporcionando así mayor calidad de vida en todos los hogares.

3. Parte experimental

3.1 Análisis

Lo que la aplicación es actualmente no es todo lo lejos que querría llegar con ella. La idea que tengo en mente es que con la misma aplicación y la misma base de datos se puedan manejar diferentes servidores simultáneamente; es decir, que una misma persona pueda contar en la aplicación con diferentes hogares y pueda ser miembro de todos.
Por este motivo continuaré analizando y comparando ambos diagramas: con el que actualmente cuenta la aplicación y el que debería ser el resultado final.

3.1.1 Diagrama entidad relación

La estructura principal del proyecto, es decir, el resultado final que debería adquirir, tiene el siguiente modelo entidad relación.

Hay cuatro entidades principales: Usuario, Habitación, Componente y Casa.
Los atributos de cada entidad han sido suprimidos de la representación visual para simplificar su comprensión.

Los usuarios pueden habitar en diferentes casas y en la casa pueden habitar diferentes usuarios, creando esto una tercera tabla de relación N:N.

En la casa puede haber diferentes habitaciones pero cada habitación va a pertenecer a una única casa. Del mismo modo, las habitaciones pueden contener diferentes componentes pero cada componente solo pertenece a una única habitación.

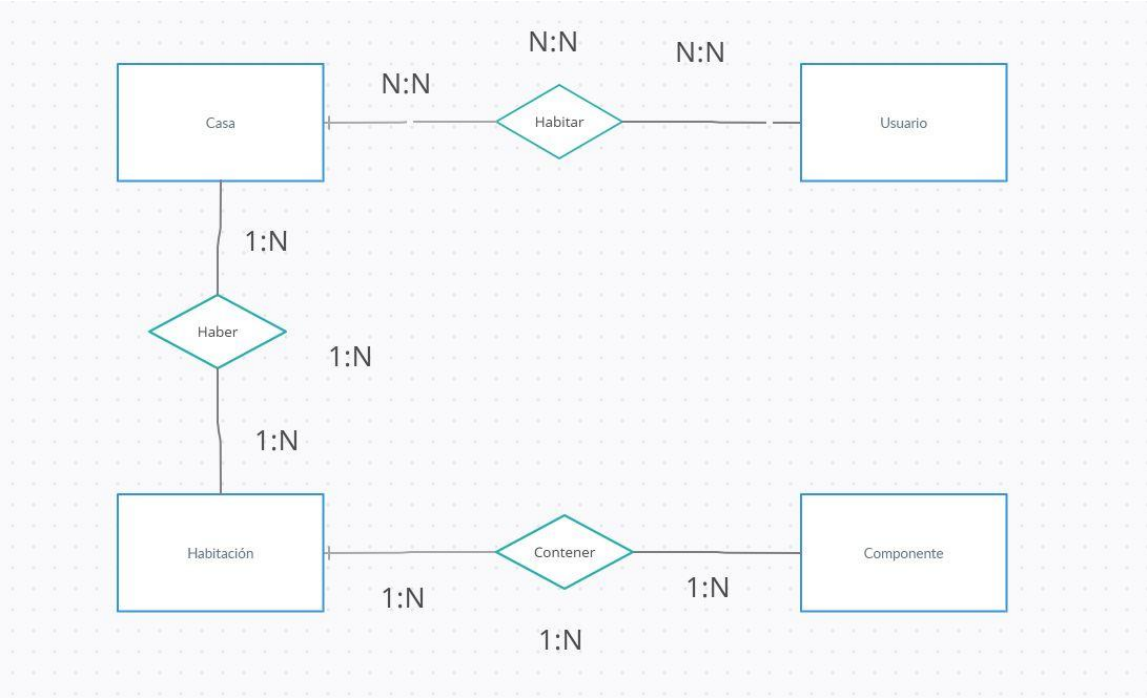


Figura 1. Diagrama Entidad-Relación (Final).

Como se puede ver en el diagrama aquí expuesto, actualmente, en la aplicación, existen tres entidades: Usuario, Habitación y Componente

Usuario es una entidad que, de momento, no tiene relación con ninguna otra, en cambio podemos ver que la entidad Habitación se relaciona con la entidad Componente, de tal manera que la Habitación puede contener varios componentes pero un componente solo puede estar en una habitación.

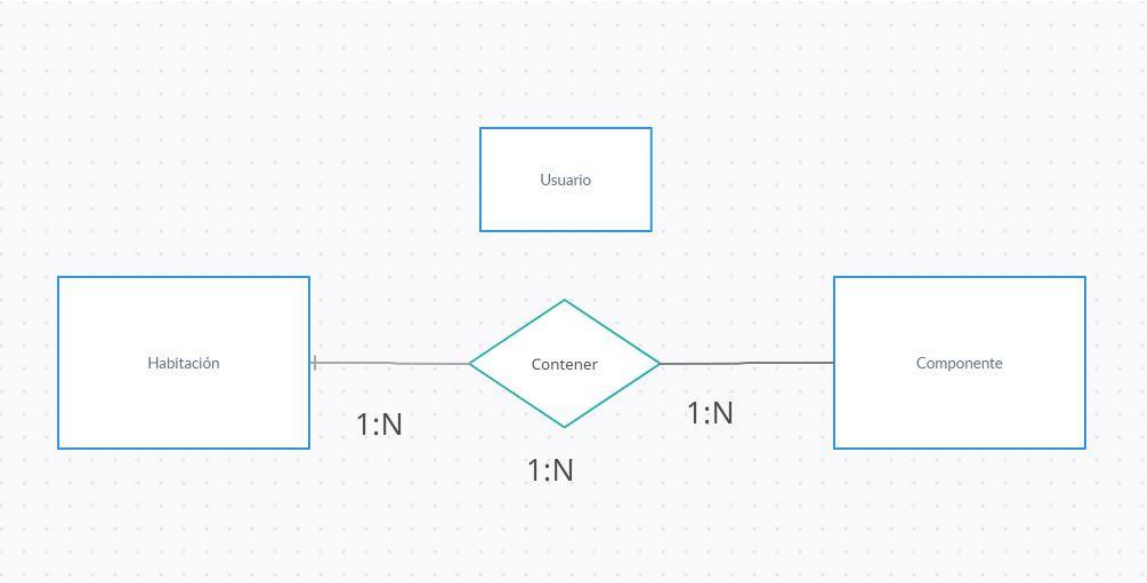


Figura 2. Diagrama Entidad-Relación (Actual).

3.1.2 Análisis de requisitos funcionalidad final

A continuación detallaremos la funcionalidad que debería tener la aplicación final, explicada desde el punto de vista del usuario. Dividiremos la explicación en dos ramas para facilitar la comprensión. Una de las ramas comprenderá un

Usuario con el rol de “administrador” de la aplicación, y la otra rama un Usuario con el rol de “miembro” en la aplicación.

Usuario Administrador:

- Añadir, eliminar o actualizar usuarios
- Añadir, eliminar o actualizar componentes
- Añadir, eliminar o actualizar habitaciones
- Programar rutinas en la casa

Usuario Miembro:

- Visualizar las habitaciones de la casa y sus componentes
- Mover los componentes de la casa
- Visualizar los miembros que hay la casa
- Visualizar las rutinas programadas

3.1.3 Diagrama de clases

El Diagrama de Clase es el diagrama principal de análisis y diseño para un sistema. En nuestro proyecto disponemos de un servidor con una aplicación web, y de un cliente que, en este caso, va a ser un dispositivo móvil Android con una aplicación instalada.

Estos diagramas nos permitirán apreciar las clases utilizadas en ambos casos, y podremos visualizar la separación en tres capas, que serían la capa de

interfaz (dirige los comportamientos de los componentes de la pantalla), la capa de servicios, la cual procesa los datos de entrada o salida, y por último la capa de DAO, que realiza peticiones para obtener o guardar los datos necesarios.

En nuestra aplicación móvil se realizan peticiones a través de API REST al servidor, y este las realiza a la base de datos obteniendo lo necesario y contestando a la llamada de API REST.

3.1.3.1 Diagrama de clases del Servidor

En el prototipo de servidor ideado contaremos con cuatro clases principales, que serán las encargadas de recoger todas las peticiones que se realicen al servidor; sin embargo en el actual proyecto disponemos de tres, ya que en esta versión de la aplicación manejamos solo una casa a la vez, más un filtro en el que se validará al usuario que intenta realizar la petición.
Las tres clases de las cuales disponemos actualmente, para recoger llamadas, son las clases Rest de Habitaciones, Componentes y Usuarios.

La capa Rest se comunica directamente con la capa de servicios, donde se recibirán los datos para validarlos o manipularlos. Por ejemplo, cuando hacemos login, esta capa comprueba que el objeto recibido contiene todos los requisitos necesarios para enviarlo a la capa del DAO.

La capa del DAO es la que se comunica con la base de datos (esta puede ser de cualquier tipo, ya sea en disco duro o remota) y realiza las peticiones necesarias. Por ejemplo, al hacer el login se comprueba en la base de datos si el usuario existe, y si es así, lo recoge.

Más adelante, en caso de que dicho usuario sea recogido, los servicios lo comparan con el obtenido anteriormente, para comprobar si la contraseña es correcta.

Nuevamente se pasaría el objeto correspondiente a la primera capa (REST) respondiendo a la llamada y dándola por finalizada.

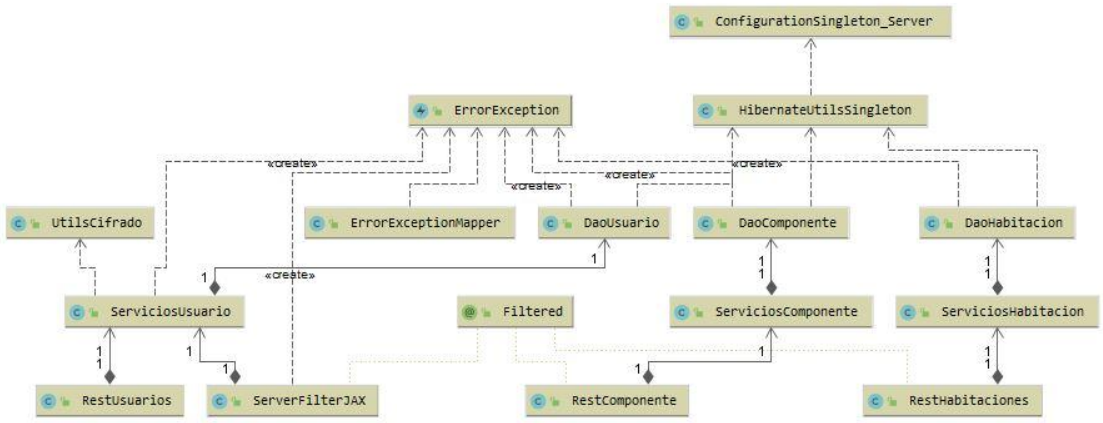


Figura 3. Diagrama de clases del servidor.

3.1.3.2 Diagrama de clases del cliente Android

En el cliente tenemos la misma distribución, es decir, todo está organizado en capas, que serían: la capa de la interfaz de usuario, llamada Fragment
y Activity, las capas de servicios y DAO, que tienen la misma funcionalidad que en el servidor, con la salvedad de que la capa del DAO es la que realiza las peticiones API REST al servidor, no interactúa con la base de datos como hemos visto anteriormente.

La capa de interfaz de usuario, Fragments y Activity, serían las que muestran al usuario la información recogida, a través de las peticiones al servidor.
Esta capa también es la que contiene los componentes, como pueden ser botones y entradas de texto. Estas últimas permiten al usuario editar, borrar o añadir datos.

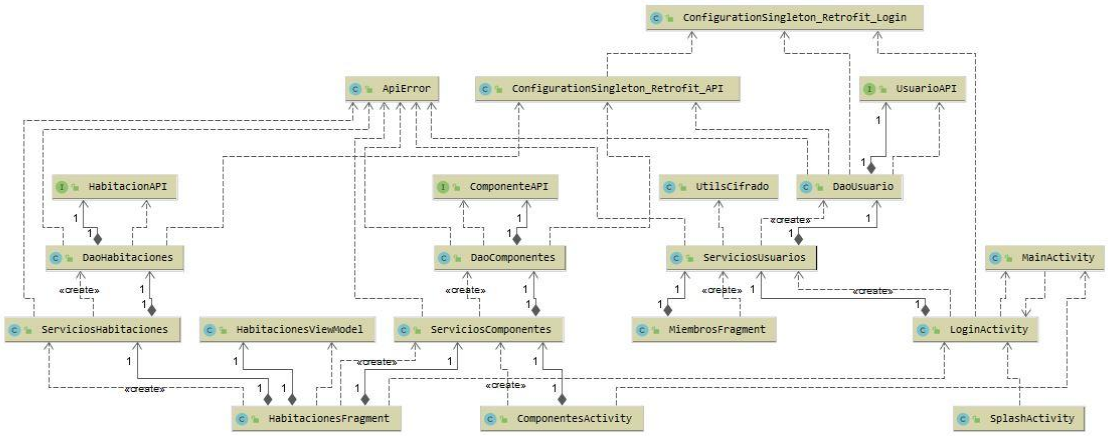


Figura 4. Diagrama de clases del cliente Android.

3.1.4 Diagrama de casos de uso

Pasamos a comprender la funcionalidad de la aplicación de una manera más sencilla a nivel visual, a través de un diagrama de casos de uso. En él se representan las acciones que pueden realizar los diferentes tipos de usuario, y donde actores externos interactúan con nuestra aplicación.

En este diagrama visualizamos dos actores, uno que tiene el rol de MIEMBRO y otro que tiene el rol de ADMIN (administrador) en la aplicación.

En esta aplicación siempre va a existir un usuario con el rol de ADMIN (administrador), y este será el que cree los demás usuarios con los roles que él elija.

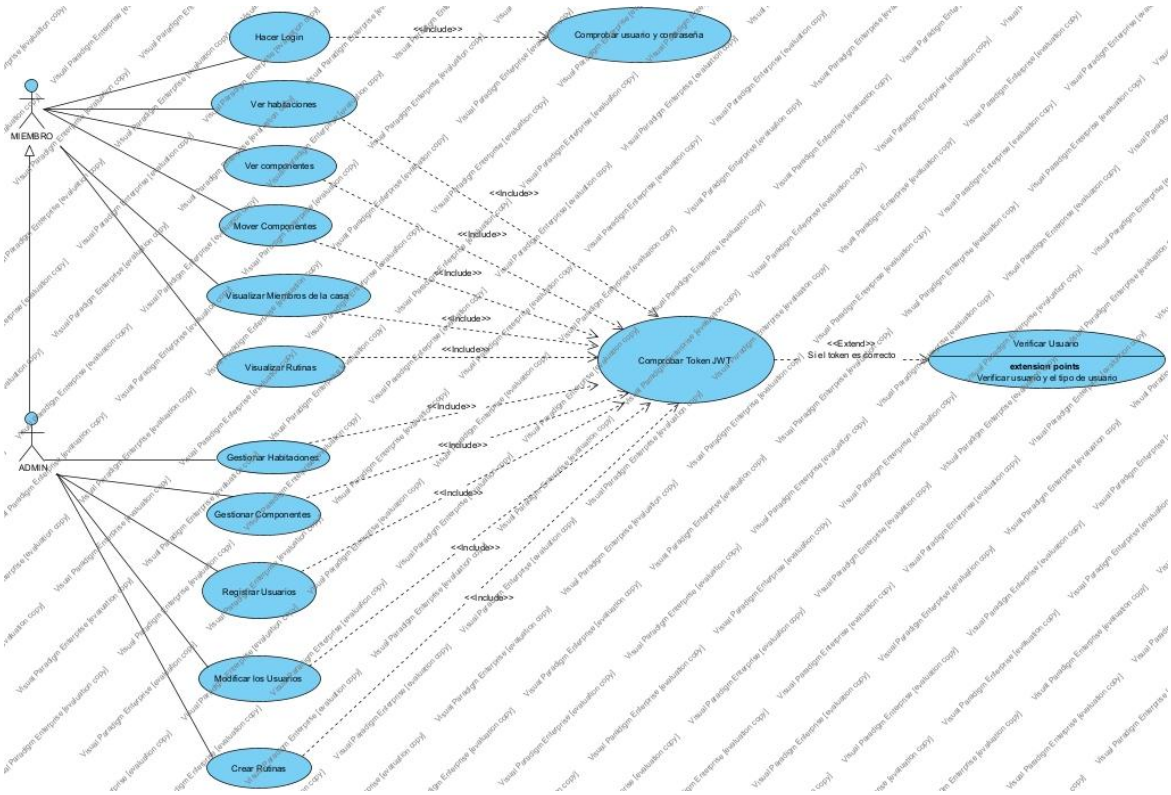


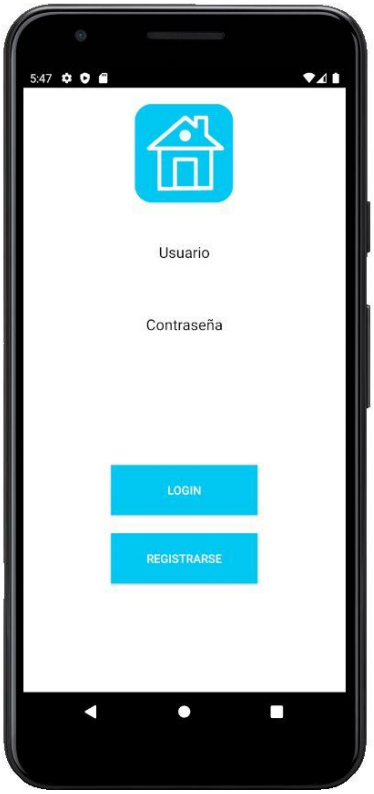
Figura 5. Diagrama de casos de uso.

3.2 Diseño

En este apartado enseñaremos el prototipado de las pantallas de interfaz de usuario, explicaremos la arquitectura del proyecto, centrándonos en las tecnologías que hemos usado en él, y por último, hablaremos también de la estructura de la base de datos, en cuanto a las tablas utilizadas, soporte logico y fisico.

3.2.1 Prototipado de las pantallas

La idea de la interfaz de las pantallas, en un principio, era tener dos Activity y tres fragmentos principales, donde de las dos Activity, una fuera la de login, y otra la de registro. El resto de fragmentos serían uno para las habitaciones y componentes correspondientes a cada habitación, otra para los miembros de la casa y otra para las rutinas.



En la pantalla de inicio, al entrar, nos aparece un login en el cual debemos insertar nuestro usuario y contraseña para poder iniciar sesión, además de dos botones: uno el de login y otro el de registro. El botón de registro nos redirige al Activity de registro.

Figura 6. Pantalla prototipo de Login.

En la parte superior de esta pantalla encontramos el logotipo de la aplicación; debajo, cuatro campos de texto para rellenar con la información necesaria y, por último, el botón del registro que hace la petición de registro de la aplicación.

Esta pantalla al final no se llegó a diseñar, ya que decidí que los usuarios administradores serían los que iban a tener el poder de crear otros usuarios miembros.

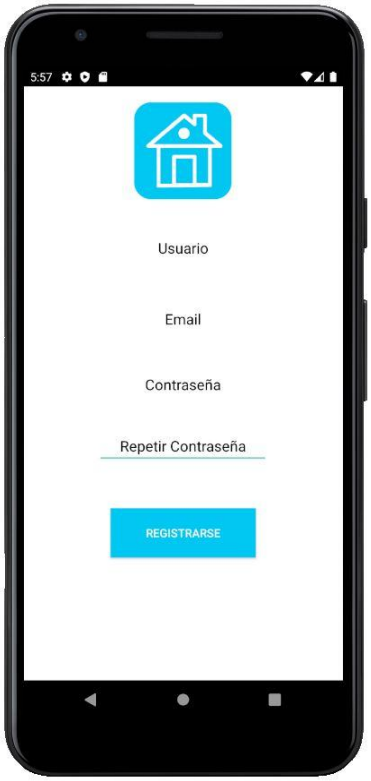
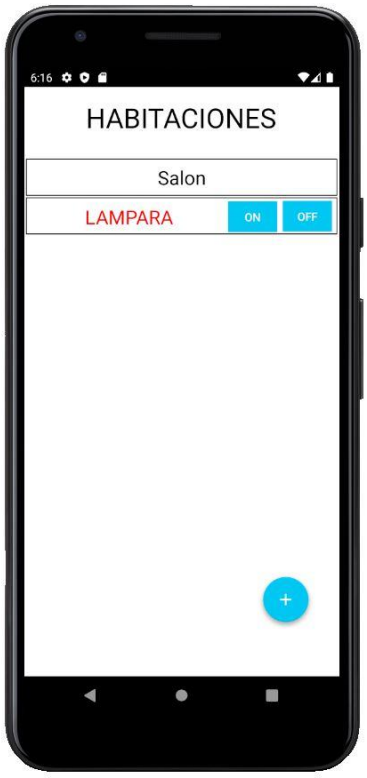


Figura 7. Pantalla prototipo de registro.



Esta pantalla es el Fragment de habitaciones, en el cual podemos encontrar el título de la pantalla, y debajo de este un ExpandableList, que contiene una lista de todas las habitaciones; al pulsar una de ellas se despliega un panel que muestra los componentes de dicha habitación. En cada componente existen dos botones: uno para encender (ON) y otro para apagar (OFF).

Por último, podemos observar en la parte inferior izquierda un botón con un signo de suma; se trata de un Floating Button Menu, que contiene tres botones necesarios para editar, eliminar y crear habitaciones y componentes.

Figura 8. Pantalla prototipo de Fragment habitaciones.

Esta pantalla es el Fragment de Miembros, compuesto por el título del Fragment en el encabezado y, debajo del mismo, un RecyclerView, que enumera todos los miembros que existen en la casa.

Igual que en el Fragment anterior, podemos observar un Floating Button Menu en la parte inferior izquierda, dentro del cual contamos con dos botones: uno para editar y otro para eliminar miembros. Este menú únicamente lo pueden visualizar los usuarios administradores.

Figura 9. Pantalla prototipo de Fragment de miembros.

Este es el Fragment de rutinas. Nuevamente tenemos el título del Fragment en el encabezado y, debajo de este, otro RecyclerView en el cual se nos muestran las rutinas ya creadas.

En la parte inferior izquierda tenemos un Floating Button Menu que contiene tres botones, cuyas utilidades son editar, crear y eliminar las rutinas. Este botón únicamente lo puede visualizar el usuario administrador.

Figura 10. Pantalla prototipo de Fragment de rutinas.

3.2.2 Estructura de base de datos

En la aplicación estamos utilizando una base de datos MYSQL remota.
La que se muestra es la estructura de base de datos actual; en el diagrama de ENTIDAD-RELACIÓN explicado con anterioridad hemos visto el diagrama final y el que tenemos actualmente.
Basándose en el diagrama actual, se han creado estas tres tablas. Cada una de ellas cuenta con los datos necesarios para almacenar.

En la tabla de usuario contamos con una id_usuario generada por el servidor y será un UUID, esta será la Primary key. También vemos que el nombre de usuario y email deberán ser únicos. Para que no haya confusiones en la base de datos, se almacena también el tipo de usuario, y por último el resto de campos son los necesarios para el cifrado de la contraseña del usuario.

En la Tabla de Habitaciones podemos encontrar la id_habitación, también generada por el servidor, ya que será un UUID siendo la Primary Key. En ella encontramos el nombre de la habitación y el lugar en el que se halla esta misma.

Por último, en la tabla de componentes, tenemos una id_componente que también se rellenará en el servidor, siendo esta un UUID. En esta hallamos información básica, como puede ser el nombre del componente, la fecha de instalación y el tipo de componente. Asimismo tenemos una Foreign Key a la tabla de habitaciones, ya que en cada habitación puede haber varios componentes. Por último, encontramos los pines, para los cuales se utilizan diferentes campos, dependiendo de si el componente es un motor o una luz. Estos son un caso aparte, ya que un pin no puede contener dos componentes.

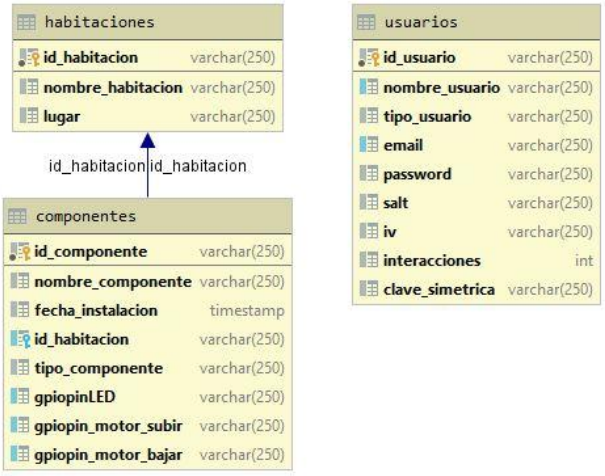


Figura 11. Estructura de la base de datos.

3.2.3 Arquitectura

En este apartado hablaremos detalladamente de todas las tecnologías que hemos tenido que poner en práctica para el desarrollo del proyecto. Dividiremos esta sección entre cliente Android y servidor, ya que hemos utilizado tecnologías diferentes.

3.2.3.1 Servidor

En ambos casos hemos hecho uso de un lenguaje Java. La diferencia entre un caso y otro es que en el servidor hemos utilizado un lenguaje Java 11, y en la aplicación Android un lenguaje Java 1.8.

Figura 12. Logo de Java.

Java nace en 1991 con el nombre "OAK", posteriormente cambiado por Green por problemas legales, y finalmente con la denominación actual JAVA.
El objetivo de java era crear un lenguaje de programación parecido a C++ en estructura y sintaxis, fuertemente orientado a objetos, pero con una máquina virtual propia.

Payara Micro

Para realizar un deploy de la aplicación hemos usado como servidor Payara.
En concreto Payara Micro, ya que este es el que se utiliza en las Raspberry Pi, de las cuales hablaremos a continuación.

Payara Micro es el primer lanzamiento de una nueva forma de ejecutar aplicaciones Java EE. Basado en el soporte de Java EE 7 de su núcleo GlassFish 4.1, Payara Micro le permite ejecutar archivos war desde la línea de comandos sin ninguna instalación de servidor de aplicaciones.

Figura 13. Logo de Payara Micro.

MySQL Connector JAVA

MySQL Connector/J es un driver nativo de Java que convierte las llamadas generadas por JDBC en el protocolo de red que utiliza la base de datos de MySQL. Permite al desarrollador trabajar con el lenguaje de programación Java y de esta forma construir programas que interactúan con Mysql.

Hibernate ORM

Hibernate es un marco de trabajo Java que proporciona mecanismos de mapeo objeto/relacional para definir cómo se almacenan, eliminan, actualizan y recuperan los objetos Java. Además, Hibernate ofrece servicios de consulta y recuperación que pueden optimizar los esfuerzos de desarrollo dentro de entornos SQL y JDBC.

LOG4J

Es una librería de código abierto desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o logs en tiempo de ejecución y no en tiempo de compilación como se realizaba comúnmente.

Figura 14. Logo de LOG4J.

YAML

YAML es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado en RFC 2822 (publicaciones RFC). YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki.

GPIO/PI4J

PI4J es un proyecto que provee un puente entre librerías nativas y Java para ganar acceso a los pines de comunicación GPIO del Raspberry PI.

ModelMapper

ModelMapper es una librería Java para copiar o mapear propiedades de un tipo de objeto a otro tipo de objeto, permitiendo copiar también los datos de las referencias a los objetos que contengan.

BouncyCastle

Es una librería de cifrado de la información utilizando criptografía. Es de libre distribución y tiene distribuciones para todos los jre de java existentes, incluso para clientes j2me. Java, nos proporciona un soporte de cifrado pero no es completo, de ahí la necesidad de usar librerías adicionales que lo cumplimenten. Esta librería ofrece una completa y efectiva solución a esta problemática, debido a esto aparte de esta librería usamos también nimbus-jose-jwt.

Raspberry Pi

Raspberry Pi es una placa de microordenador, que como su propio nombre indica, es de pequeñas dimensiones a la cual se le pueden dar multitud de usos como veremos más adelante. La Raspberry Pi apareció en febrero de 2012, 6 años después de que comenzara el proyecto principal de esta placa, y para agosto del 2012 ya habrían vendido 500.000 unidades, y un mes después ya se había realizado la primera revisión “B” de la placa original. La primera unidad tenía 256 MB de ram y un procesador a 700 MHz, tenía el característico conector de 26 pines GPIO y salida de video por HDMI o RCA además de un conector de 3.5mm para el audio, el primer modelo carecía de puerto ethernet.

En este proyecto estamos usando un Raspberry Pi 3

3.2.3.2 Cliente Android

Figura 15. Logo de Android

IO.VAVR

Esta dependencia nos permite que nuestro código pueda quedar más claro a la hora de programar, además, en este proyecto hemos utilizado también el Objeto Either, que nos otorga la oportunidad de devolver dos tipos de objetos en una función dependiendo de si esta ha funcionado correctamente o no.
Este objeto se distribuye únicamente en izquierda, donde nosotros devolveremos un error, y en la derecha donde devolvemos una respuesta correcta.

RXJAVA

RxJava es una implementación open-source de la librería ReactiveX que le ayuda a crear aplicaciones en el estilo de programación reactivo. Aunque RxJava está diseñado para procesar flujos sincrónicos y asincrónicos de datos, no se restringe a tipos de datos "tradicionales".

OKHTTP3

Es una librería de código abierto, que permite realizar operaciones tanto en HTTP como en SPDY de manera sencilla y eficiente en ambientes Java (versión 1.7 como mínimo) y Android (2.3 como mínimo), sin necesidad de cambiar el código de la aplicación entre ambas plataformas, con una interfaz fluida.

Retrofit

Retrofit es un cliente de servidores REST para Android y Java desarrollado por Square, muy simple y muy fácil de aprender. Permite hacer peticiones al servidor tipo: GET, POST, PUT, PATCH, DELETE y HEAD, y gestionar diferentes tipos de parámetros, paseando automáticamente la respuesta a un tipo de datos.

3.2.3.3 Arquitectura común de servidor y cliente Android

Gson

Gson (también conocido como Google Gson) es una biblioteca de código abierto para el lenguaje de programación Java que permite la serialización y deserialización entre objetos Java y su representación en notación JSON.

JWT

JWT (JSON Web Token) es un estándar abierto (publicado en el RFC 7519) que define un método compacto y autocontenido para encapsular y compartir aserciones (claims) sobre una entidad (subject) de manera segura entre distintas partes mediante el uso de objetos JSON.

Lombok

Lombok es un proyecto que nació en el año 2009 que, mediante contribuciones, ha ido ganando en riqueza y variedad de recursos. Es una librería para Java que a través de anotaciones nos reduce el código que codificamos, es decir, nos ahorra tiempo y mejora la legibilidad del mismo.

Figura 16. Logo de Lombok.

3.3 Implementación y pruebas

En este apartado hablaremos de uno de los códigos principales de la aplicación y su proceso, a parte de eso veremos las pruebas que he llevado a cabo en el código.

3.3.1 Desarrollo de la aplicación

La programación de la aplicación ha sido algo llevadero y fluido; el hecho de conocer el lenguaje y la mayoría de tecnologías usadas bastante bien lo ha hecho todo mucho más sencillo.

Un código principal y del que vamos hablar ahora mismo es el de la seguridad del servidor. En este código hemos usado JWT y un filtro.

Funciona de la siguiente manera: al iniciar el servidor se auto genera un certificado con una clave pública y una clave privada. Para guardar la clave privada dentro del certificado también se genera una clave aleatoria, que es cifrada y guardada en disco, al igual que el certificado. Esta función se ejecuta desde el Listener del servidor.

```
@WebListener()
public class ListenerConfig implements ServletContextListener
{
   // Public constructor is required by servlet spec
   public ListenerConfig() {
   }
   // -----------------------------------------------------
   // ServletContextListener implementation
   // -----------------------------------------------------
   public void contextInitialized(ServletContextEvent sce) {
     /* This method is called when the servlet context is
        initialized(when the Web application is deployed).
        You can initialize servlet context related data here.
     */

ConfigurationSingleton_Server.cargarInstance(sce.getServletContext().getResourceAsStream("/WEB-INF/
config/config.yaml"));
```

```
            UtilsCifrado.crearClaves(sce.getServletContext().getRealPath("WEB-INF"));
        }

        public void contextDestroyed(ServletContextEvent sce) {
          /* This method is invoked when the Servlet Context
             (the Web application) is undeployed or
             Application Server shuts down.
          */
            // Cerrar pool
        }
    }
```

Una vez tenemos las claves creadas el servidor está listo para recibir peticiones. Cuando un usuario hace login y los servicios verifican que este es correcto, los servicios crean un JWT en el que se introduce parte de la información del usuario con una validez de 30 minutos.

```
private String generarJwt(UsuarioMapper um, String path) {
    try {
        Gson gson = new Gson();
        Password psswd = gson.fromJson(Files.newBufferedReader(new File(Paths.get(path +
"/psswd.json").toAbsolutePath().toString()).toPath()), Password.class);
        psswd = UtilsCifrado.descifrarContraseña(psswd);

        KeyStore ksLoad = KeyStore.getInstance("PKCS12");
        ksLoad.load(new FileInputStream(path + "/server_cert.pfx"), "".toCharArray());
        KeyStore.PasswordProtection pp = new
KeyStore.PasswordProtection(psswd.getPassword().toCharArray());
        KeyStore.PrivateKeyEntry privateKeyEntry = (KeyStore.PrivateKeyEntry)
ksLoad.getEntry("privada", pp);
        PrivateKey clavePrivada = privateKeyEntry.getPrivateKey();

        String jwt = Jwts.builder()
                .setIssuer("THEFACTORY:IHOUSE")
                .setSubject("SERVIDOR_IHOUSE")
                .claim("nombreUsuario", um.getNombreUsuario())
                .claim("tipoUsuario", um.getTipoUsuario().toString())
                .claim("idUsuario", um.getIdUsuario())
                .setIssuedAt(Date.from(LocalDate.now()
                        .atStartOfDay(ZoneId.systemDefault())
                        .toInstant()))

.setExpiration(Date.from(LocalDateTime.now().plusMinutes(30).toInstant(ZoneOffset.UTC)))
                .signWith(clavePrivada, SignatureAlgorithm.RS512)
                .compact();

        return jwt;

    } catch (Exception e) {
        log.error(e.getMessage(), e);
        throw new ErrorException("Error al generar JWT", Response.Status.BAD_REQUEST);
    }
}
```

A partir de ahí, el Cliente introduce el JWT en la cabecera de una configuración Singleton que será la que se use para realizar peticiones al servidor. A continuación mostraremos el código del DAO del cliente, viendo cómo recibe el JWT y luego el Singleton creando una instancia estática para las peticiones.

Esta es la función de la capa DAO del cliente Android donde se realiza la petición de login y se ejecuta la carga del JWT:

```
public Either<ApiError, String> loginUsuario(Usuario u) {
    try {
        Call<String> call = usuarioAPILogin.loginUsuario(u);
        Response<String> response = call.execute();
        if (response.isSuccessful()) {
            //Se recoge el JWT y se inserta en la configuracion Singleton
            ConfigurationSingleton_Retrofit_API.cargarInstance(response.body());
            usuarioAPI = ConfigurationSingleton_Retrofit_API.getInstance().create(UsuarioAPI.class);
            return Either.right(response.body());
        } else {
            if (response.errorBody().contentType().equals(MediaType.get("application/json"))) {
                return Either.left(toApiError(response.errorBody().string()));
            } else {
                return Either.left(ApiError.builder().message("Error de comunicacion").fecha(LocalDateTime.now()).build());
            }
        }
    } catch (Exception e) {
        Log.e(DAOUSUARIO_DEBUG, e.getMessage(), e);
        return Either.left(ApiError.builder().message("Error al hacer login Usuario").fecha(LocalDateTime.now()).build());

    }
}
```

Esta es la función de la configuración del Singleton donde se carga la instancia para las peticiones:

```
public static void cargarInstance(String jwt) {
    CookieManager cookieManager = new CookieManager();
    cookieManager.setCookiePolicy(CookiePolicy.ACCEPT_ALL);

    clientOK = new OkHttpClient.Builder()
            .readTimeout(Duration.of(10, ChronoUnit.MINUTES))
            .callTimeout(Duration.of(10, ChronoUnit.MINUTES))
            .connectTimeout(Duration.of(10, ChronoUnit.MINUTES))
            .addInterceptor(chain -> {
                        Request original = chain.request();
                        Request.Builder builder1 = original.newBuilder()
                                .header("Authorization", "Bearer: " + jwt);
                        Request request = builder1.build();
                        return chain.proceed(request);
```

```
                }
            )
            .cookieJar(new JavaNetCookieJar(cookieManager))
            .build();

    Gson gson = new GsonBuilder()
            .registerTypeAdapter(LocalDate.class, new JsonDeserializer<LocalDate>() {
                @Override
                public LocalDate deserialize(JsonElement jsonElement, Type type, JsonDeserializationContext
jsonDeserializationContext) throws JsonParseException {
                    return LocalDate.parse(jsonElement.getAsJsonPrimitive().getAsString());
                }
            }).registerTypeAdapter(LocalDate.class, new JsonSerializer() {
                @Override
                public JsonElement serialize(Object o, Type type, JsonSerializationContext jsonSerializationContext) {
                    return new JsonPrimitive(LocalDate.now().toString());
                }
            })
            .create();
    retrofit = new Retrofit.Builder()
            .baseUrl(ConfigurationSingleton_Retrofit_Login.getUrl())
            .addConverterFactory(ScalarsConverterFactory.create())
            .addConverterFactory(GsonConverterFactory.create(gson))
            .client(clientOK)
            .build();
}
```

A partir de aquí, cualquier petición que se haga pasará por un filtro, ya que el login es la única petición no filtrada. En el filtro se recoge el JWT conseguido anteriormente y se comprueba que no está caducado; una vez validado, se recoge la información que contienen del usuario y se ejecuta una función a través de los servicios de usuario redirigida a la base de datos, donde se comprueba que el usuario existe, y que es el tipo de usuario que dice ser.

Este es el filtro del servidor:

```
@Override
public void filter(ContainerRequestContext containerRequestContext) {
    String jwt = request.getHeader("Authorization");
    PublicKey clavePublica = getClavePublica();

    if (jwt != null) {
        try {
            Jws<Claims> jwts = Jwts.parserBuilder()
                    .setSigningKey(clavePublica)
                    .build()
                    .parseClaimsJws(jwt.substring(8));

            UsuarioMapper um = UsuarioMapper.builder()
                    .idUsuario((String) jwts.getBody().get("idUsuario"))
                    .tipoUsuario(TipoUsuario.valueOf((String) jwts.getBody().get("tipoUsuario")))
                    .nombreUsuario((String) jwts.getBody().get("nombreUsuario"))
                    .build();

            su.verificarUsuario(um);
        } catch (ExpiredJwtException ex) {
```

```
            throw new ErrorException("El token a expirado", Response.Status.UNAUTHORIZED);
        } catch (JwtException ex) {
            throw new ErrorException("Clave publica del token erronea", Response.Status.UNAUTHORIZED);
        }
    }
}
```

Y esta función en el DAO, llamada desde servicios, es la que comprueba el usuario:

```
public void verificarUsuario(UsuarioMapper um) {
    Session session = HibernateUtilsSingleton.getInstance().getSession();
    Usuario u = null;
    try {
        session.beginTransaction();
        u = session.createQuery("FROM Usuario WHERE nombreUsuario =: nombreUsuario", Usuario.class)
                .setParameter("nombreUsuario", um.getNombreUsuario())
                .list()
                .stream()
                .findFirst()
                .orElse(null);
        session.getTransaction().commit();

        if (u != null) {
            if (!TipoUsuario.valueOf(u.getTipoUsuario()).equals(um.getTipoUsuario())) {
                throw new ErrorException("Error al verificar usuario", Response.Status.UNAUTHORIZED);
            }
        }else {
            throw new ErrorException("Error al verificar usuario", Response.Status.UNAUTHORIZED);
        }
    } catch (Exception e) {
        session.getTransaction().rollback();
        log.error(e.getMessage(), e);
        if (!TipoUsuario.valueOf(u.getTipoUsuario()).equals(um.getTipoUsuario())) {
            throw new ErrorException("Error al verificar usuario", Response.Status.UNAUTHORIZED);
        }
        throw new ErrorException("Error al verificar usuario", Response.Status.BAD_REQUEST);
    } finally {
        session.close();
    }
}
```

En caso de que esto devuelva un error al cliente, la aplicación se cierra y obliga a hacer de nuevo el login.

Figura 17. Captura de pantalla Android

3.3.2 Pruebas realizadas

Código de prueba	Descripción de la prueba	Resultado esperado	Estado
1. Como Usuario quiero hacer login			
1.1	Hacer login con el usuario y contraseña	El login es satisfactorio y el fragment habitaciones aparece	APTO
1.2	Intentar hacer login con campos vacíos	Sale un mensaje de error diciendo que faltan campos por rellenar	APTO

Código de prueba	Descripción	Resultado esperado	Estado
2. Como Administrador quiero gestionar las habitaciones			
2.1	Realizar petición para recoger las habitaciones	Salen todas las habitaciones en el fragment de habitaciones	APTO
2.2	Crear una nueva habitación	La habitación se añade a BBDD y a la Expandable List del fragment	APTO
2.3	Editar una habitación ya existente	La habitación se actualiza en la BBDD y en la Expandable List del Fragment	APTO
2.4	Eliminar una habitación existente	La habitación se elimina de la BBDD y de la ExpandableList	APTO

		del fragment	
2.5	Crear una habitación con campos vacíos	Sale un mensaje de error diciendo que faltan campos por rellenar	APTO
2.6	Intentar actualizar dos habitaciones a la vez	Sale un mensaje de error diciendo que solo se puede actualizar una	APTO

Código de prueba	Descripción	Resultado esperado	Estado
3. Como administrador quiero gestionar los componentes			
3.1	Recoger los componentes de cada habitación	Los componentes aparecen dentro de la ExpandableList del Fragment dentro de su habitación correspondiente.	APTO
3.2	Mover un componente	El componente se mueve tal y como se ha indicado	APTO
3.3	Añadir un nuevo componente	El componente se añade en la BBDD	APTO
3.4	Editar un componente existente	El componente se actualiza en la BBDD y en la ExpandableList	APTO
3.5	Eliminar un componente	El componente se eliminar de la BBDD y de la ExpandableList del Fragment	APTO
3.6	Crear un componente con campos vacíos	Sale un mensaje de error diciendo que	APTO

		faltan campos por rellenar	

Código de prueba	Descripción	Resultado esperado	Estado
4. Como administrador quiero gestionar los miembros			
4.1	Registrar un miembro	Aparece el miembro añadido en la BBDD y en RecyclerView del Fragment aparte se puede hacer login con este	APTO
4.2	Editar un miembro	El miembro se actualiza en la BBDD y aparece en el RecyclerView del Fragment	APTO
4.3	Recoger los miembros de la casa	Los miembros de la casa aparecen en el RecyclerView del Fragment	APTO
4.4	Eliminar un miembro	El miembro desaparece de la BBDD y de la RecyclerView del Fragment	APTO
4.5	Intentar realizar una petición después de haberme cambiado el tipo de usuario	Sale un popup diciendo que hay un error y obligandome a iniciar sesion de nuevo	APTO
4.6	Intentar crear un usuario con campos vacíos	Sale un mensaje de error diciendo que faltan campos por rellenar	APTO

3.3.3 Implantación y documentación

Para la implantación de nuestra aplicación debemos hacer diferentes procesos debido a que tenemos dos aplicaciones: una en un dispositivo móvil Android y otra en un servidor, que sería una aplicación web.

3.3.3.1 mplantación de la aplicación web en el servidor

Estamos utilizando un servidor Payara Micro en un Raspberry y será necesario que este también contenga Java 11, así que lo que debemos hacer es compilar el servidor en un formato .war para poder transferirlo a la Raspberry. Para compilar nos vamos a la pestaña de Build y seleccionamos Build Artifacts.

Figura 18. Muestra del paquete .war.

Una vez tenemos la aplicación web debemos transferirla; en mi caso, voy a usar la aplicación MobaXterm ya que permite una conexión ssh y ftp a la vez con mi Raspberry PI y es tan sencillo como arrastrar el archivo de un lado a otro.

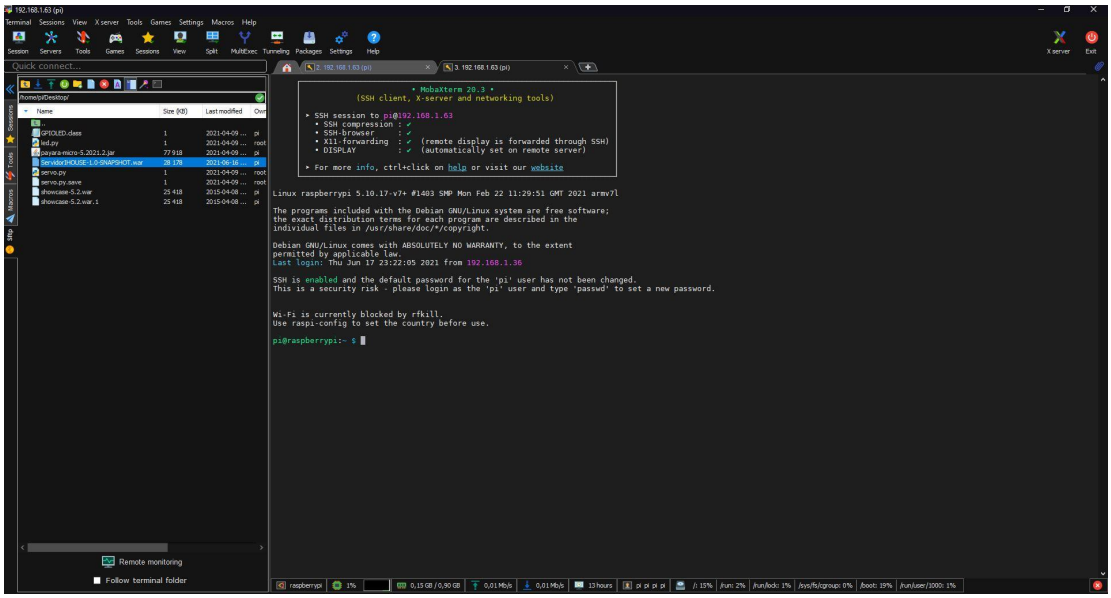


Figura 19. Captura de Mobaxterm conectado a la Raspberry Pi.

Por último nos movemos en la consola al directorio donde tengamos el Payara Micro y la aplicación Web, y ejecutamos el siguiente comando, realizando un Deploy de la aplicación

java -jar payara-micro-5.2021.2.jar --deploy ServidorIHOUSE-1.0-SNAPSHOT.war

3.3.3.2 Implantación de la aplicación en el cliente Android

En Android deberemos compilar nuestra aplicación en un paquete APK
APK significa: Android Aplication Package. Es decir un archivo ejecutable de aplicaciones para Android. Un archivo con extensión .APK es un paquete para el sistema operativo Android. Este formato es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados para la plataforma Android, tanto smartphones como tablets. APK hace referencia a un tipo de formato para archivos Android, en la mayoría de los casos se trata de aplicaciones o juegos, que nos permite instalarlos en nuestro dispositivo sin necesidad de utilizar Play Store.

Por lo que en Android studio podremos realizar nuestra APK de dos maneras distintas una manera es una apk común en la que podemos instalarla en cualquier dispositivo sin ningún tipo de problema y otra manera es compilarla firmandola con con una clave pública de Google Play Store para poder publicarla que esta es la que yo creo, que es la manera más útil.
Para Realizar esto deberemos dirigirnos a la pestaña de Build -> Generate Signed APK y rellenar los campos correspondientes una vez así ya tendremos compilada nuestra aplicación y podremos publicarla y descargarla en nuestro teléfono o directamente transferirla a través de un cable USB.

3.3.4 Scrum

El proyecto se ha gestionado a través del método ágil Scrum como desarrollo de la aplicación. Este se ha usado en la aplicación de Taiga.io

Scrum se ejecuta en bloques temporales que son cortos y periódicos, denominados Sprints, que por lo general de entre 2 hasta 4 semanas, que es el plazo para feedback y reflexión.
Cada Sprint es una entidad en sí misma, esto es, proporciona un resultado completo, una variación del producto final que ha de poder ser entregado al cliente con el menor esfuerzo posible cuando éste lo solicite.
El proceso tiene como punto de partida una lista de objetivos/requisitos que conforman el plan de proyecto. Es el cliente del proyecto el que prioriza estos objetivos teniendo en cuenta un balance del valor y el coste de los mismos, es así como se determinan las iteraciones y consecuentes entregas.

En IHOUSE he conseguido completar cuatro sprints, tres de dos semanas y uno de una.

4. Resultados y discusión

El resultado del proyecto, en mi opinión, es bastante bueno, ya que ha superado las expectativas iniciales del proyecto. El único imprevisto real ha sido que el trabajo se ha salido un poco de los plazos establecidos, y no me ha sido posible realizar toda la funcionalidad pensada. Esta, como ya he mencionado, se dejará para futuras mejoras.

Al contrario de lo esperado, no han surgido demasiados problemas. Aún así ha sido necesaria una investigación a fondo, desde cero, para poder usar la Raspberry Pi del modo que yo quería.

El estado actual del proyecto es bastante bueno, de tal manera que la aplicación se podría publicar en Google Play Store, y la aplicación web se podría meter dentro de un Docker, para que cualquier persona pudiera usarla.

En el transcurso del trabajo se me han ocurrido algunas mejoras que podrían lograr hacer que la aplicación fuera más útil y fácil de usar. Serían las siguientes:

- Como he mencionado anteriormente, la idea sería modificar las tablas y crear un nuevo modelo llamado “casa”, pudiendo así guardar diferentes casas en la base de datos y poder gestionarlas simultáneamente desde la aplicación móvil.
- Poder tener más componentes, como medidores de temperatura o cámaras dentro del hogar en directo.
- Tener una cámara en directo grabando en un disco duro 24H y poder acceder a la grabación a través de la aplicación móvil.

- Poder tener sensores de movimiento y poder activarlos y desactivarlos como si fueran una alarma, recibiendo así una notificación en el móvil ante cualquier movimiento extraño.
- Que cada usuario pueda restablecer su contraseña y no lo tenga que hacer el administrador.
- Las rutinas deberían haberse hecho al principio, pero como no ha dado tiempo, es necesario realizar las rutinas de la casa para poder mover los componentes como se indiquen a una determinada hora del día.

5. Conclusiones

Siento que la elaboración de este proyecto ha causado en mí una evolución y una mejora en la absorción de los conocimientos y aptitudes adquiridas durante los dos años que ha durado este Ciclo Formativo de Grado Superior.
La necesidad de utilizar todos mis conocimientos y mostrar mi máximo potencial para dar por finalizada esta etapa de mi formación educativa, ha hecho que mi ambición por adquirir nuevas competencias y comenzar nuevos proyectos se disparara.

Además de esto, gracias a estos años de formación en el campo de la programación me he dado cuenta de lo importante que es el seguir estudiando día a día, debido a que en un mundo tan globalizado como el nuestro, nacen nuevas tecnologías útiles para la programación constantemente.Si no eres capaz, acabas quedándote anticuado.

En conclusión, ha sido una buena experiencia para el futuro y una buena manera de tener un ejemplo de lo que puede llegar a ser un gran proyecto el día de mañana.

6. Bibliografía y referencias

Java
http://www.tuprogramacion.com/programacion/historia-de-java/#:~:text=Java%20nace%20en%201991%20con,con%20una%20m%C3%A1quina%20virtual%20propia

MySQL Connector
https://desarrolloweb.com/articulos/901.php

Hibernate ORM
https://programacion.net/articulo/persistencia_de_objetos_java_utilizando_hibernate_306#:~:text=Hibernate%20es%20un%20marco%20de,de%20entornos%20SQL%20y%20JDBC.

Lombok
https://www.paradigmadigital.com/dev/proyecto-lombok-facilitame-la-vida/#:~:text=Lombok%20es%20un%20proyecto%20que,mejora%20la%20legibilidad%20del%20mismo.

Log4j
http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/historico/1.4.0/contenido-recurso-226.html

Yaml
https://es.wikipedia.org/wiki/YAML#:~:text=YAML%20es%20un%20formato%20de,Net%20y%20Oren%20Ben%2DKiki.

Pi4j
https://unpocodejava.com/2013/08/08/pi4j-control-del-gpio-de-raspberry-pi-con-java/

ModelMapper
https://picodotdev.github.io/blog-bitix/2020/05/copiar-datos-de-un-tipo-de-objeto-a-otro-con-modelmapper/#:~:text=ModelMapper%20es%20una%20librer%C3%ADa%20Java,a%20los%20objetos%20que%20contengan.

GSON
https://chemamegino.wordpress.com/tag/gson/

JWT
https://www.bbva.com/es/json-web-tokens-jwt-claves-para-usarlos-de-manera-segura/

BouncyCastle
http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/235

RXJAVA
https://code.tutsplus.com/es/tutorials/kotlin-reactive-programming-with-rxjava-and-rxkotlin--cms-31577

OKHTTP3
http://nombre-temp.blogspot.com/2015/04/peticiones-http-asincronas-con-okhttp.html#:~:text=Es%20una%20librer%C3%ADa%20de%20c%C3%B3digo,plataformas%2C%20con%20una%20interfaz%20fluida

APK
https://www.samsung.com/latin/support/mobile-devices/what-is-an-apk/

Sitios de documentacion tenidos en cuenta para la realización del proyecto
https://linuxize.com/post/install-java-on-raspberry-pi/
https://www.jormc.es/2013/11/18/raspi-java-y-tomcat-en-raspberrypi/
https://unpocodejava.com/2013/08/08/pi4j-control-del-gpio-de-raspberry-pi-con-java/#:~:text=Que%20es%20GPIO,con%20tan%20solo%20dos%20pines

https://tutoriales.online/bitacora/es/usando-raspberry-pi-gpio-con-java
https://tutoriales.online/bitacora/es/usando-raspberry-pi-gpio-con-java
https://blog.payara.fish/running-java-ee-applications-with-payara-micro-on-the-raspberry-pi

5. Anexos

7.1 ANEXO I - Sprints del scrum

Nº Sprint	Titulo	Fecha Inicio	Fecha Fin	Duracion (Días)
1	Crear las clases para gestionar las habitaciones	12/04/21	26/04/21	14
2	Crear las clases para gestionar los componentes	4/05/21	18/05/21	19 (Excedido de tiempo)
3	Crear las clases para gestionar los miembros	25/05/21	8/06/21	14
4	Crear la seguridad del servidor	09/06/21	16/06/21	7

Nº historia	Nº Sprint	Titulo historia	Fecha Inicio	Fecha Fin	Duración (Días)

1	1	Como usuario quiero poder ver las habitaciones	12/04/21	18/04/21	7
2	1	Como administrador quiero editar las habitaciones	19/04/21	26/04/21	7
3	2	Como usuario quiero poder ver los componentes de cada habitación	03/05/21	10/05/21	7
4	2	Como administrador quiero gestionar componentes	10/05/21	17/05/21	7
5	2	Como usuario quiero mover componentes	17/05/21	22/05/21	7
6	3	Como usuario quiero ver los miembros de la casa	25/05/21	2/06/21	7
7	3	Como administrador quiero poder	3/06/21	08/06/21	7

		gestionar los miembros de la casa			
8	4	Como Usuario quiero que mis datos esten protegidos	09/06/21	16/06/21	7

Tarea	Nº Historia	Fecha Inicio	Fecha Fin	Duración (Días)
Crear Capa servicios y dao en servidor de las habitaciones	1	12/04/21	13/04/21	1
Crear capa rest en servidor de las habitaciones	1	13/04/21	13/04/21	0.5
Crear capa servicios y dao en Android de las habitaciones	1	13/04/21	14/04/21	1
Crear retrofit y RecyclerView en Android de las habitaciones	1	17/04/21	18/04/21	1
Crear querys en capa dao del servidor de las habitaciones	2	19/04/21	20/04/21	1

Solucionar errores de hibernate	2	20/04/21	22/04/21	2
Crear peticiones api rest en Android	2	22/04/21	22/04/21	0.5
Crear componentes en android para la edición y asignarle funciones	2	22/04/21	26/04/21	4
Crear capas de servicios y dao en servidor de los componentes	3	03/05/21	04/05/21	1
Crear capas de rest de componentes en servidor	3	04/05/21	05/05/21	1
Crear capas de servicios y dao en Android de componentes	3	06/05/21	07/05/21	1
Crear ExpandableList de habitaciones y componentes en Android	3	08/05/21	09/05/21	1

Crear query de la capa del dao de los componentes	4	11/05/21	13/05/21	2
Crear peticiones Api rest en Android de los componentes	4	13/05/21	14/05/21	1
Crear componentes de Android para gestionar el modelo componente	4	15/05/21	18/05/21	3
Crear circuito de un led para conectarlo a las raspberry pi 3	5	19/05/21	20/05/21	1
Crear en servicios función para mover componentes	5	20/05/21	23/05/21	3
Crear las capas de servicios y dao en servidor de los usuarios	6	25/05/21	26/05/21	1
Crear las capas de servicios y Dao de Android de los usuarios	6	26/05/21	27/05/21	1
Crear Retrofit y RecyclerView de miembros en Android	6	27/05/21	29/05/21	2

Crear querys en el dao de los usuarios	7	30/05/21	31/05/21	1
Crear peticiones retrofit de Android de los usuarios	7	1/06/21	2/06/21	1
Crear componentes en Android para gestionar los miembros	7	2/06/21	8/06/21	6
Implementar JWT en la función login del servidor	8	09/06/21	12/06/21	3
Modificar configuración de Android para implementar JWT	8	12/06/21	15/06/21	3
Crear filtro en servidor para las peticiones	8	15/06/21	16/06/21	1

7.2 ANEXO I - Diagrama de Gantt

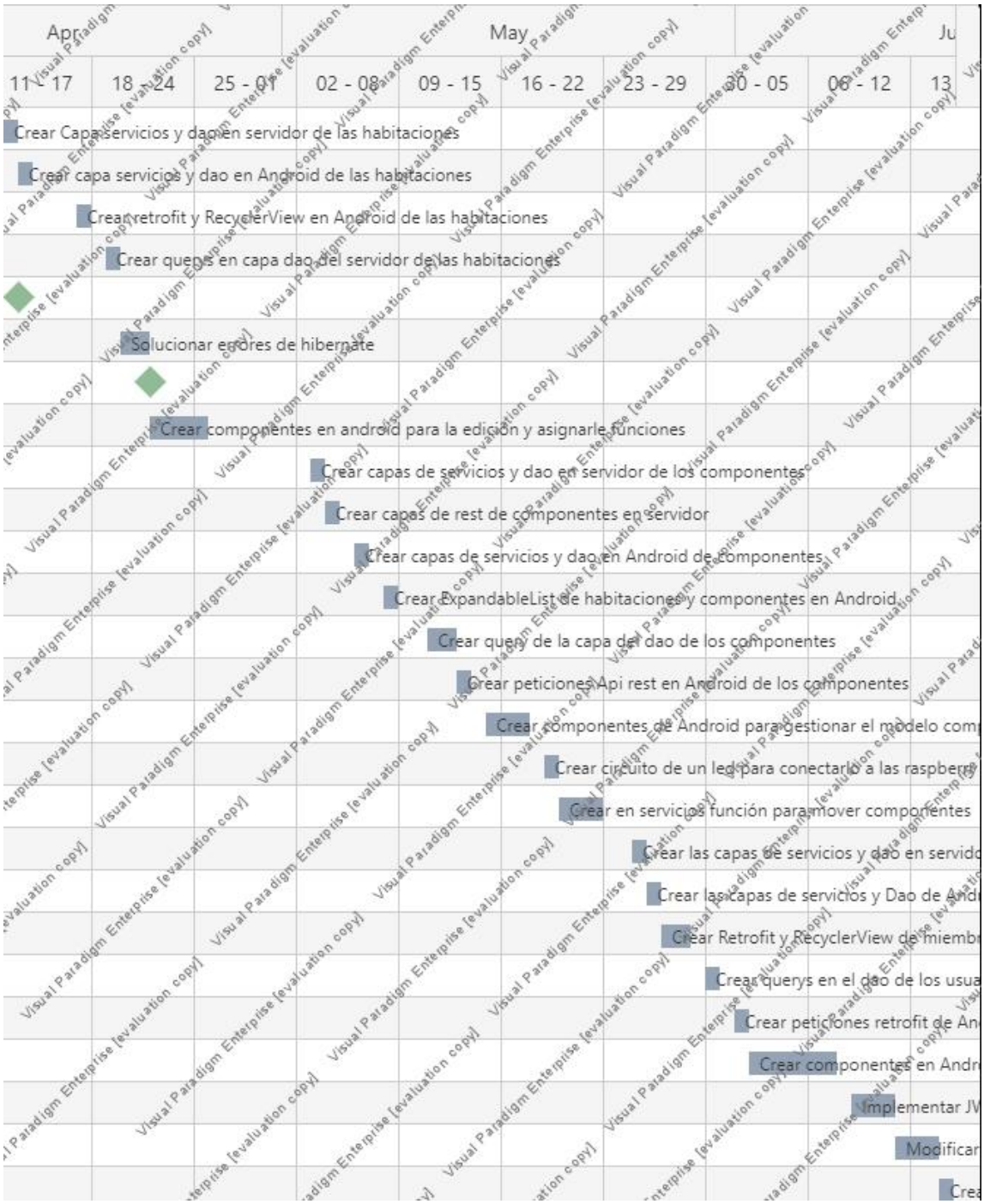


Figura 20. Diagrama de Gantt