Proyecto de Desarrollo de Aplicaciones Multiplataforma

Proyecto Fin de Grado

ALUMNO: JOSÉ ALFONSO RODRÍGUEZ CABRERO

ÍNDICE

Agradecimientos

En primer lugar, quiero dar las gracias a mi familia sin cuyo apoyo no habría conseguido las metas que me propuse.

Por otro lado, agradecer a todos los profesores que me han impartido clases durante estos años y que son responsables de los conocimientos que actualmente dispongo.

A mi tutora del TFG, Raquel, por haberme ayudado a lo largo del desarrollo y por orientar mis esfuerzos en la mejor dirección posible.

1. Introducción

La principal motivación para la realización de este TFG ha sido profundizar en el desarrollo de aplicaciones móviles a la vez que creaba una aplicación susceptible de ser utilizada en las empresas de las que soy socio, reduciendo la dependencia de aplicaciones de terceros.

La aplicación que se presenta en el proyecto busca facilitar el registro horario de los trabajadores de una empresa mediante dispositivos android, así como permitir la solicitud de días de permiso y de vacaciones.

Con la realización de la aplicación se han puesto en uso los conocimientos adquiridos durante los dos años de curso DAM realizados dando como resultado un prototipo funcional que servirá de partida para las posteriores versiones del mismo.

2. Objetivos del proyecto

En el mercado existen actualmente numerosas soluciones enfocadas al registro horario que permiten realizar las operaciones de registro horario y control de permisos y vacaciones.

Una característica de todas ellas es que se trata de software privativo que utiliza servidores de terceros por lo que en el caso de querer cambiar de proveedor, la migración puede resultar complicada.

El objetivo del proyecto es disponer de una aplicación que permita realizar las funciones anteriormente comentadas (registro horario y solicitud de permisos y vacaciones) y que el código de dicha plataforma sea propiedad de mis empresas.

De esta manera, además de la considerable reducción de costes, nos aseguramos el control total de la herramienta y sus datos eliminando en gran medida la dependencia de terceros.

3. Módulos formativos utilizados en el TFG

La realización del presente proyecto ha estado basada en los siguientes módulos formativos:

Programación multimedia y dispositivos móviles.
La realización del proyecto ha implicado la creación de una aplicación funcional en dispositivos android que suponía uno de los puntos fundamentales de la asignatura. También se ha utilizado la herramienta android studio, cuyo uso se explicó también en la asignatura.

Bases de datos y acceso a bases de datos
Los registros generados por la utilización de la aplicación se almacenan en una base de datos noSQL por lo que la persistencia y acceso a la base de datos se realiza de manera frecuente en la aplicación. Pese al hecho de utilizar una base de datos noSQL cuyo funcionamiento no se estudió en el curso, muchos de los conceptos y estructura que comparten en común las bases de datos SQL y noSQL fueron adquiridos a lo largo del mismo.

Entornos de desarrollo
La aplicación se ha desarrollado con el IDE Android Studio a la vez que se hacía un uso intensivo del sistema de control de versiones git y de los repositorios de Github.

Lenguaje de marcas
El desarrollo de las vistas en android studio se realiza a través de documentos XML cuyos usos y características se vieron en la asignatura.

4. Herramientas y Lenguajes utilizados

Las herramientas utilizadas para la elaboración del presente TFG han sido las siguientes:

4.1 Android Studio

(https://developer.android.com/studio) Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas GNU/Linux, macOS, Microsoft Windows y Chrome OS. Ha sido diseñado específicamente para el desarrollo de Android.

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de 2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014.1

Desde el 7 de mayo de 2019, Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones de Android.2 Aun así, Android Studio admite otros lenguajes de programación, como Java y C ++. (*Fuente: Wikipedia* https://es.wikipedia.org/wiki/Android_Studio)

La realización del proyecto se ha realizado sobre el Android SDK 26.

4.2 Firebase Realtime

La base de datos Firebase Realtime almacena y sincroniza datos de la aplicación con la base de datos NoSQL alojada en la nube. Los datos se sincronizan con todos los clientes en tiempo real y se mantienen disponibles cuando la app no tiene conexión. (https://firebase.google.com/docs/database?hl=es-419)

Firebase Realtime Database es una base de datos alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. Cuando compilas apps multiplataforma con los SDK de plataformas de Apple, Android y JavaScript, todos los clientes comparten una instancia de Realtime Database y reciben actualizaciones automáticamente con los datos más recientes. (*Fuente:* Google)

4.3 StarUML

(https://staruml.io/) StarUML es una herramienta UML de MKLab. El software tenía licencia bajo una versión modificada de GNU GPL hasta 2014, cuando se lanzó una versión reescrita 2.0.0 bajo una licencia propietaria.

Después de estar abandonado por un tiempo, el proyecto tuvo un renacimiento para pasar de Delphi a Java, antes de detenerse nuevamente. Mientras tanto, se creó una bifurcación comunitaria no afiliada con el nombre de WhiteStarUML. En 2014, los desarrolladores originales reescribieron StarUML y lo lanzaron como Shareware bajo una licencia propietaria.

El objetivo declarado del proyecto era reemplazar aplicaciones comerciales más grandes, como Rational Rose y Borland Together.

StarUML admite la mayoría de los tipos de diagramas especificados en UML 2.0. Desde la versión 4.0.0 (29 de octubre de 2020) incluye diagramas generales de tiempo e interacción.

4.4 Mockflow

MockFlow es una herramienta en línea de las numerosas alternativas existentes en el mercado para diseñar y prototipar interfaces de usuario para diferentes plataformas, como por ejemplo Windows, Mac, aplicaciones móviles (iOS y Android) y Web. (https://www.mockflow.com/)

4.5 Git

(https://git-scm.com/) Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. 2Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. 3Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux. (*Fuente: Wikipedia* https://es.wikipedia.org/wiki/Git)

4.6 GitHub

(https://github.com/) GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena generalmente de forma pública.

El 4 de junio de 2018, Microsoft compró GitHub por la cantidad de 7500 millones de dólares.12 Al inicio, el cambio de propietario generó preocupaciones y la salida de algunos proyectos de este sitio;3 sin embargo, no fueron representativos. GitHub continúa siendo la plataforma más importante de colaboración para proyectos de código abierto. (*Fuente: Wikipedia* https://es.wikipedia.org/wiki/GitHub)

4.7 Java

(https://www.java.com/es/) Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

4.8 XML

XML, siglas en inglés de eXtensible Markup Language, traducido como 'Lenguaje de Marcado Extensible' o 'Lenguaje de Marcas Extensible', es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

XML no ha nacido únicamente para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande, con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. (*Fuente: Wikipedia* https://es.wikipedia.org/wiki/Extensible_Markup_Language)

5. Fases del proyecto

El desarrollo de la aplicación tiene como elementos fundamentales su funcionalidad y la estructura de la base de datos que sirve de almacenamiento a las diferentes interacciones del usuario con la misma.

A continuación se explican las distintas fases por las que ha pasado el proyecto.

5.1 Modelo de datos utilizado

Tal y como se ha indicado anteriormente, la base de datos utilizada como soporte de la aplicación es Firebase Realtime. Esta base de datos NoSQL en la nube nos va a permitir trabajar de manera segura y delegar todo su mantenimiento en Google.

El hecho de que se trate de una base de datos NoSQL nos permite una mayor flexibilidad en su uso frente a las bases de datos SQL.

Las principales características de una base de datos no relacional son las siguientes:

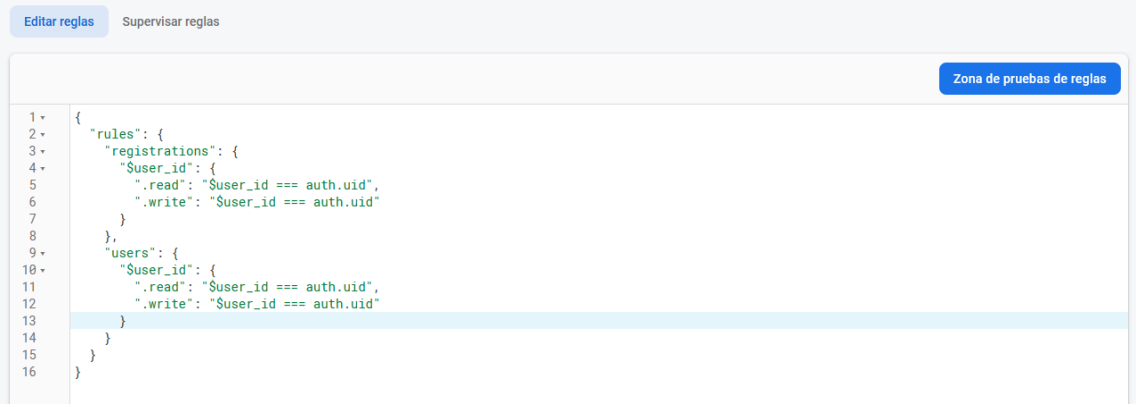
- Flexibilidad: las bases de datos NoSQL generalmente ofrecen esquemas flexibles que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
- Escalabilidad: las bases de datos NoSQL generalmente están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores caros y sólidos. Algunos proveedores de la nube manejan estas operaciones en segundo plano, como un servicio completamente administrado.
- Alto rendimiento: la base de datos NoSQL está optimizada para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que el intento de lograr una funcionalidad similar con bases de datos relacionales.

En resumen, las bases de datos NoSQL están diseñadas para que se puedan insertar datos sin un esquema predefinido, por lo que resulta fácil realizar modificaciones importantes en las aplicaciones en tiempo real sin interrupciones del servicio, de modo

que el desarrollo es más rápido, la integración del código es más fiable y los administradores de las bases de datos tienen menos trabajo.

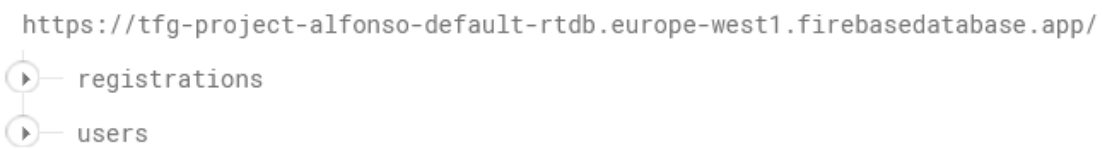
Haciendo uso de estas características de las bases de datos NoSQL el modelo de datos utilizado se muestra a continuación a través de los registros de prueba.

El primer paso antes de comenzar a usar Firebase Realtime es configurar sus reglas de lectura y escritura. En el caso de este proyecto se ha establecido que los usuarios únicamente puedan leer y escribir sus propios datos:



La base de datos cuenta con dos colecciones de documentos diferentes:

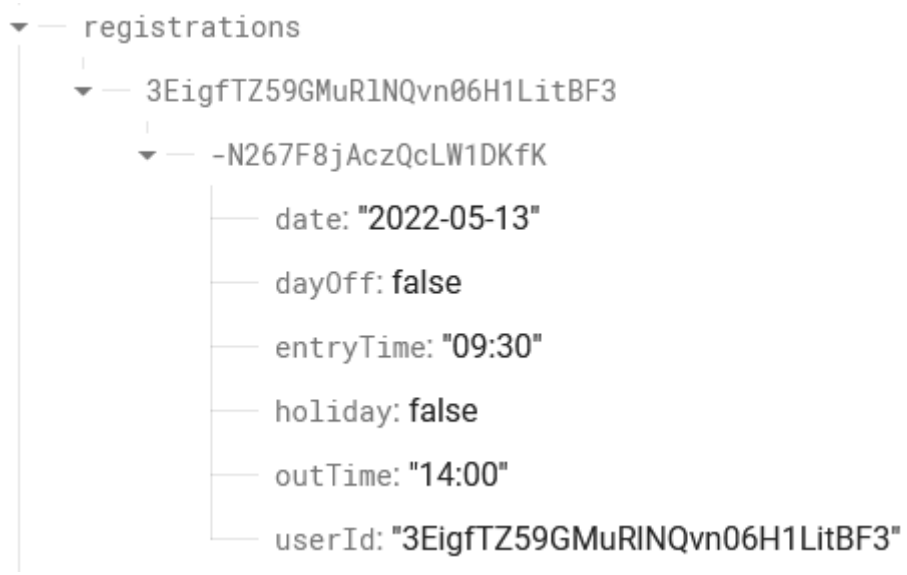
- registrations.- Dentro de esta colección se van a recoger todos los registros de los empleados ordenados por el identificador del empleado asignado por Firebase Realtime.
- users.- Dentro de esta colección se van a recoger todos los usuarios de la aplicación ordenados por el identificador del empleado asignado por Firebase Realtime.



La colección registrations va a almacenar todos los datos con los registros realizados por el usuario así como sus solicitudes de permisos y vacaciones.

En función de si se trata de un registro de un día de trabajo con sus horarios de entrada y de salida, un día de permiso o un día de vacaciones, los campos (pares clave-valor) utilizados serán diferentes.

En el caso de un registro horario los campos a persistir son los siguientes:

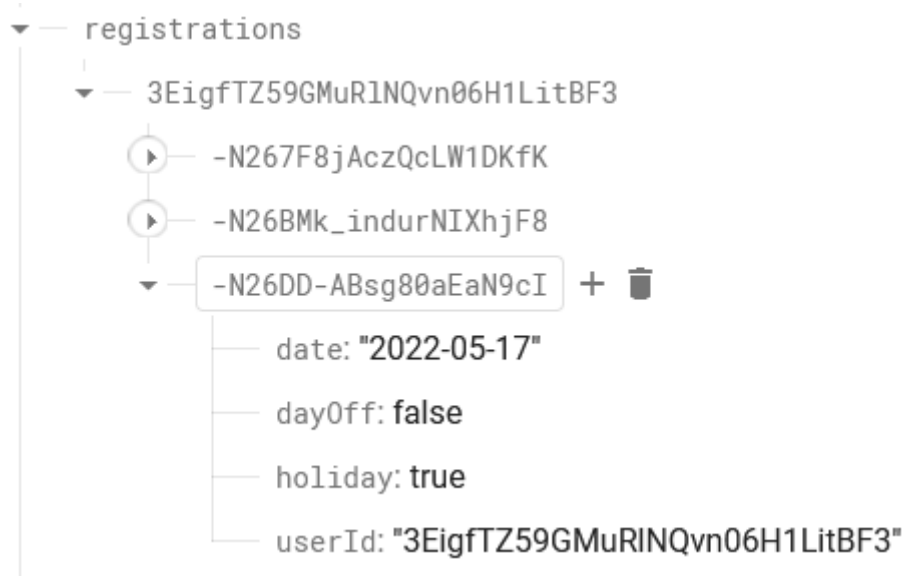


- Identificador del empleado.- Se utiliza para agrupar todos los registros realizados por cada empleado.
    - Identificador del registro.- Cada registro va a contar con un identificador único para asegurar su trazabilidad. Cada registro de horarios va a contar con los mismos campos.
        - date: String.- Recoge la fecha del registro.
        - dayOff: boolean.- Indica si el día se ha solicitado como permiso.
        - entryTime; String.- Recoge la hora de comienzo de trabajo.
        - holiday: boolean.- Indica si el día se ha solicitado como vacaciones.
        - outTime; String.- Recoge la hora de finalización de trabajo.
        - userId: String.- Recoge el identificador del usuario.

En el caso de un registro de un día de permiso los campos a persistir son los siguientes:

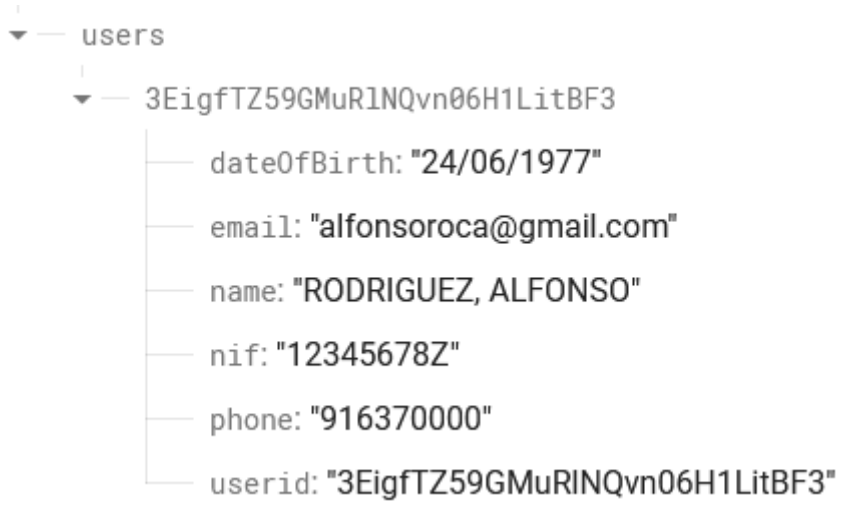
- Identificador del empleado.- Se utiliza para agrupar todos los registros realizados por cada empleado
    - Identificador del registro.- Cada registro va a contar con un identificador único para asegurar su trazabilidad. Cada registro de permiso va a contar con los mismos campos.
        - date: String.- Recoge la fecha del registro.
        - dayOff: boolean.- Indica si el día se ha solicitado como permiso.
        - holiday: boolean.- Indica si el día se ha solicitado como vacaciones.
        - reason: String.- Motivo del día de permiso.
        - userId: String.- Recoge el identificador del usuario.

Por último, en el caso de un registro de un día de vacaciones los campos a persistir son los siguientes:



- Identificador del empleado.- Se utiliza para agrupar todos los registros realizados por cada empleado
    - Identificador del registro.- Cada registro va a contar con un identificador único para asegurar su trazabilidad. Cada registro de vacaciones va a contar con los mismos campos.
        - date: String.- Recoge la fecha del registro.
        - dayOff: boolean.- Indica si el día se ha solicitado como permiso.
        - holiday: boolean.- Indica si el día se ha solicitado como vacaciones.
        - userId: String.- Recoge el identificador del usuario.

La colección users va a almacenar todos los datos relacionados con los datos del usuario.



- Identificador del empleado.- Se utiliza para agrupar los campos de cada empleado.
    - dateOfBirth: String.- Recoge la fecha de nacimiento del empleado.
    - email; String.- Recoge el email del empleado.
    - name: String.- Recoge el nombre del empleado.
    - nif: String.- Recoge el nif del empleado.
    - phone: String.- Recoge el teléfono del empleado.
    - userId: String.- Recoge el identificador del usuario.

5.2 Diagrama de casos de uso

Un caso de uso es la descripción de una acción o actividad. Un diagrama de caso de uso es una descripción de las actividades que deberá realizar alguien o algo para llevar a cabo algún proceso. Los personajes o entidades que participarán en un diagrama de caso de uso se denominan actores. En el contexto de ingeniería del software, un diagrama de caso de uso representa a un sistema o subsistema como un conjunto de interacciones que se desarrollarán entre casos de uso y entre estos y sus actores en respuesta a un evento que inicia un actor principal. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones. Los diagramas de casos de uso se utilizan para ilustrar los requisitos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo. (*Fuente wikipedia* https://es.wikipedia.org/wiki/Caso_de_uso)

Los actores que intervienen en la aplicación son 2:

- Empleados.- Usuario final de la aplicación.
- Administrador.- Usuario encargado de la gestión de los empleados.

Hay que tener en cuenta que el administrador interactúa con la base de datos de la aplicación a través de los servicios web para realizar las tareas de administración de la base de datos.

Diagrama de casos de uso del empleado.

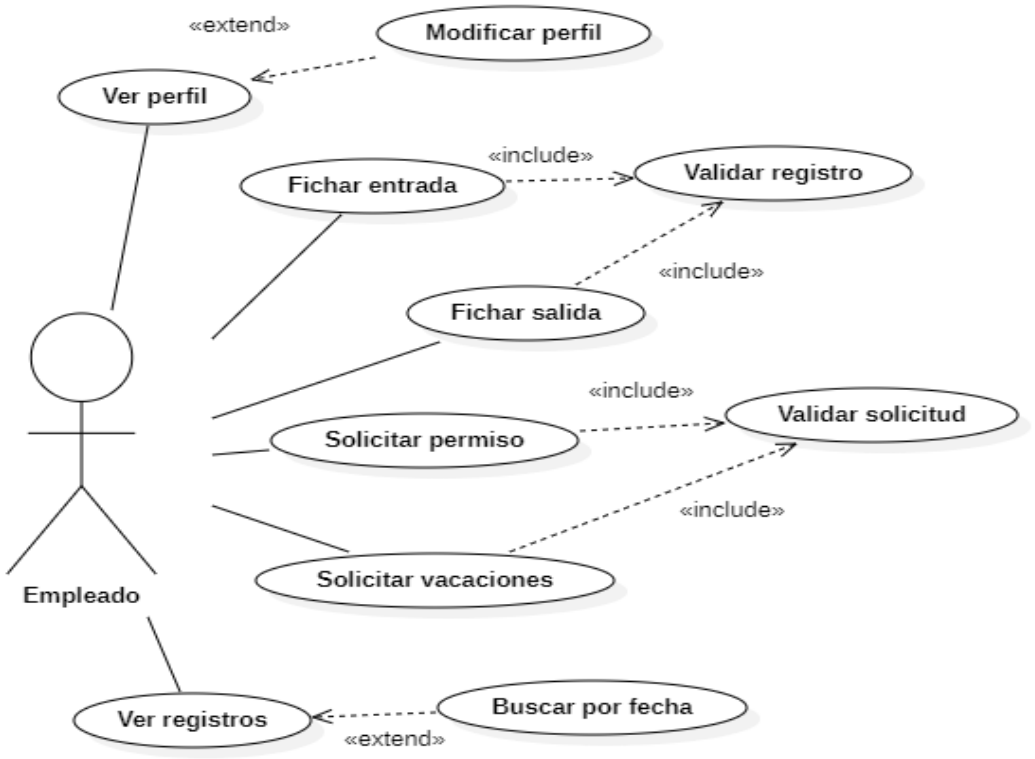
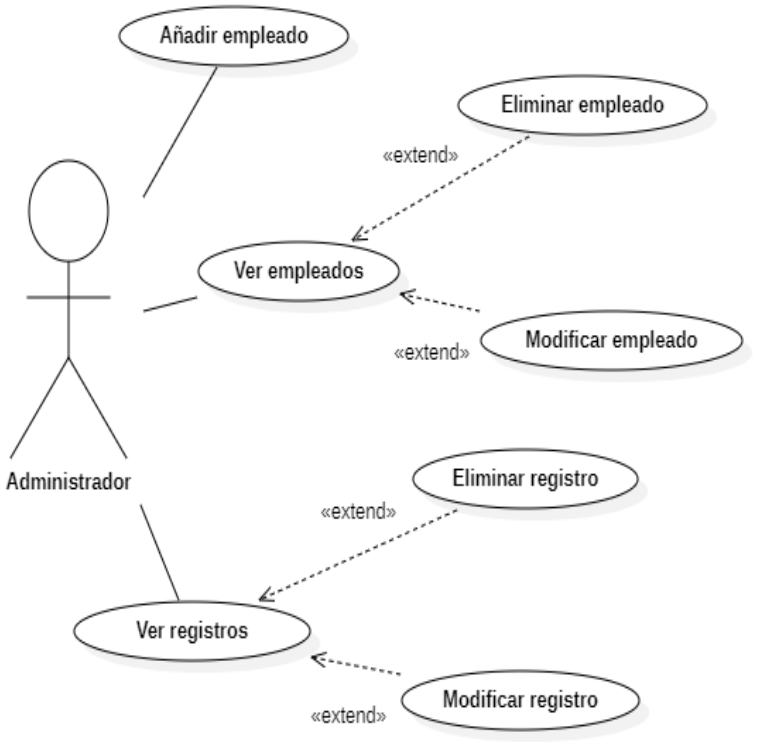


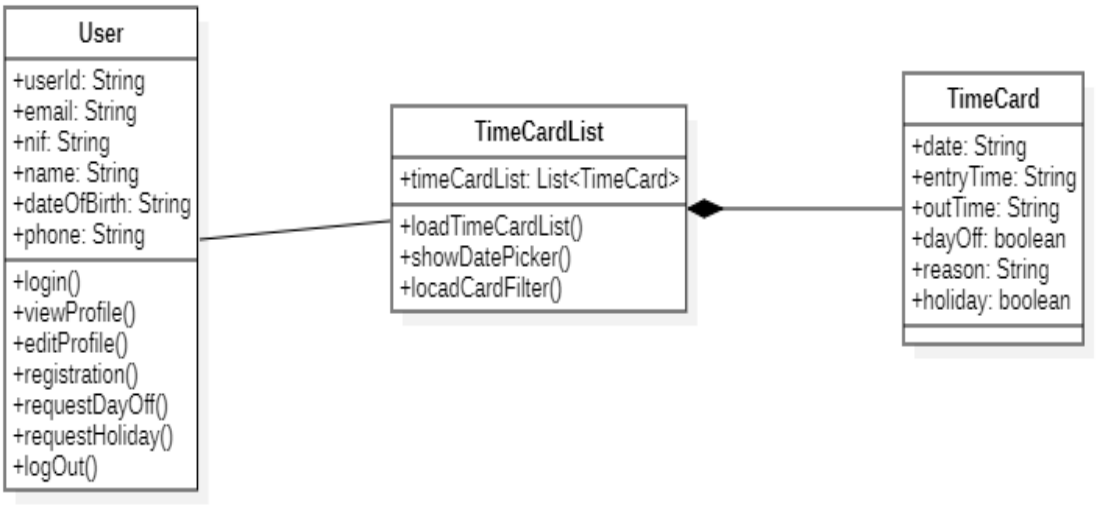
Diagrama de casos de uso del administrador



5.3 Diagrama de clases

En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. (*Fuente:* wikipedia https://es.wikipedia.org/wiki/Diagrama_de_clases)

El diagrama de clases de la aplicación presenta la siguiente estructura:



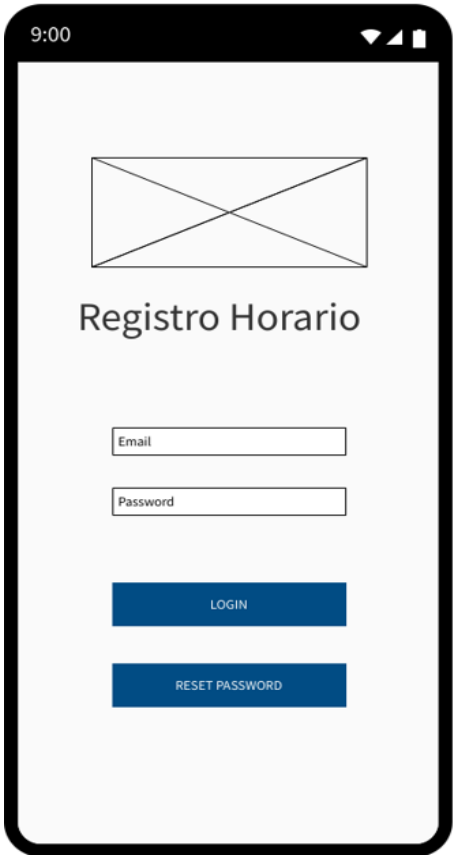
5.4 Wireframe / mockup

Los wireframes son una representación visual de una interfaz (UI) en la que se utilizan formas simples como cajas, círculos, líneas y flechas para así mostrar una estructuración básica del contenido y jerarquía de la información.

Se trata de dibujos sencillos para que se pueda apreciar una etapa inicial de la aplicación sin que se haya desarrollado, con el fin de poder tomar decisiones básicas en cuanto a la organización del contenido.

Un mockup es un prototipo hecho antes del desarrollo del trabajo para transformar ideas en funcionalidades.

Los mockup utilizados para el desarrollo de la aplicación se muestran a continuación:



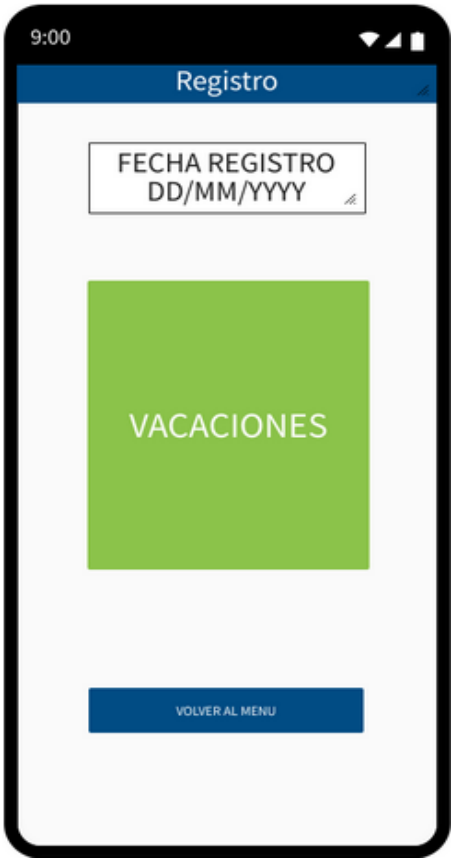
Pantalla de login.- Pantalla desde la que se accede a la aplicación o se solicita un cambio de contraseña en el caso de que se haya olvidado o sea el primer inicio en la aplicación.

Pantalla solicitud de contraseña.- Pantalla desde la que se solicita un cambio de contraseña introduciendo la dirección de email que previamente se ha cargado en la base de datos.

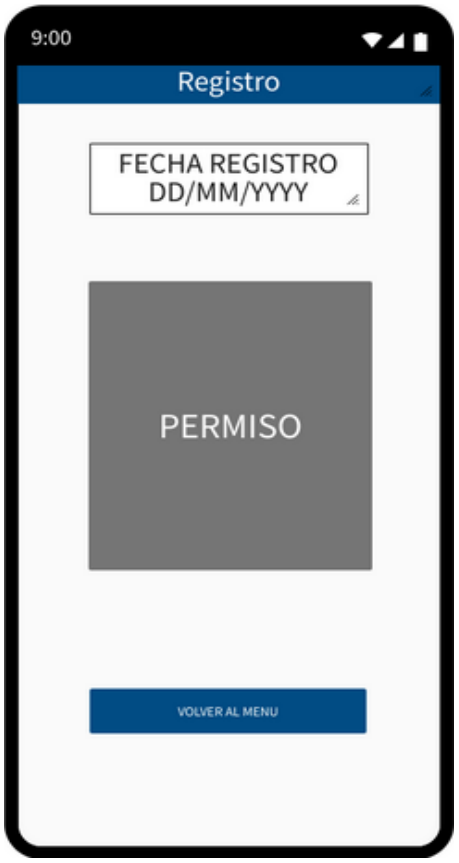
Pantalla de menú.- Pantalla desde la que el usuario puede acceder a todas las funcionalidades de la aplicación.

Pantalla de registro de entrada.- Pantalla desde la que el usuario registra el comienzo de su horario de trabajo.

Pantalla de registro de salida.- Pantalla desde la que el usuario registra la finalización de su horario de trabajo. Siempre que exista un registro de salida pendiente de validar será el primero que visualice el usuario, de esta manera nos aseguramos de que no se queden registros incompletos



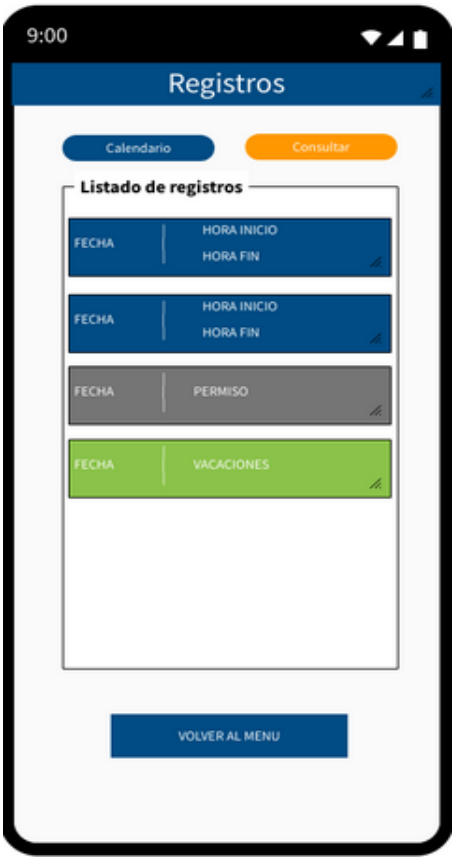
Pantalla de registro cuando el día figura como vacaciones.- Pantalla que ve el usuario al intentar registrar su jornada cuando previamente ha solicitado el día de vacaciones para la fecha actual. La aplicación no permite ningún registro horario si el día fue solicitado como vacaciones.



Pantalla de registro cuando el día figura como permiso.- Pantalla que ve el usuario al intentar registrar su jornada cuando previamente ha solicitado el día de permiso para la fecha actual. La aplicación no permite ningún registro horario si el día fue solicitado como permiso.

Pantalla de solicitud de permisos.- Pantalla desde la que el usuario registra la solicitud de un día de permiso.

Pantalla de solicitud de vacaciones.- Pantalla desde la que el usuario registra la solicitud de vacaciones.



Pantalla de consulta de registros.- Pantalla desde la que el usuario puede visualizar los registros existentes por orden cronológico y realizar búsquedas de registros en un día concreto.

Pantalla de visualización y edición de datos del usuario.- Pantalla desde la que el usuario puede visualizar sus datos así como proceder a la edición de los mismos, a excepción del email que sólo puede modificarlo el administrador.

5.5 Modelo Vista Controlador

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. *(Fuente wikipedia https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador)*

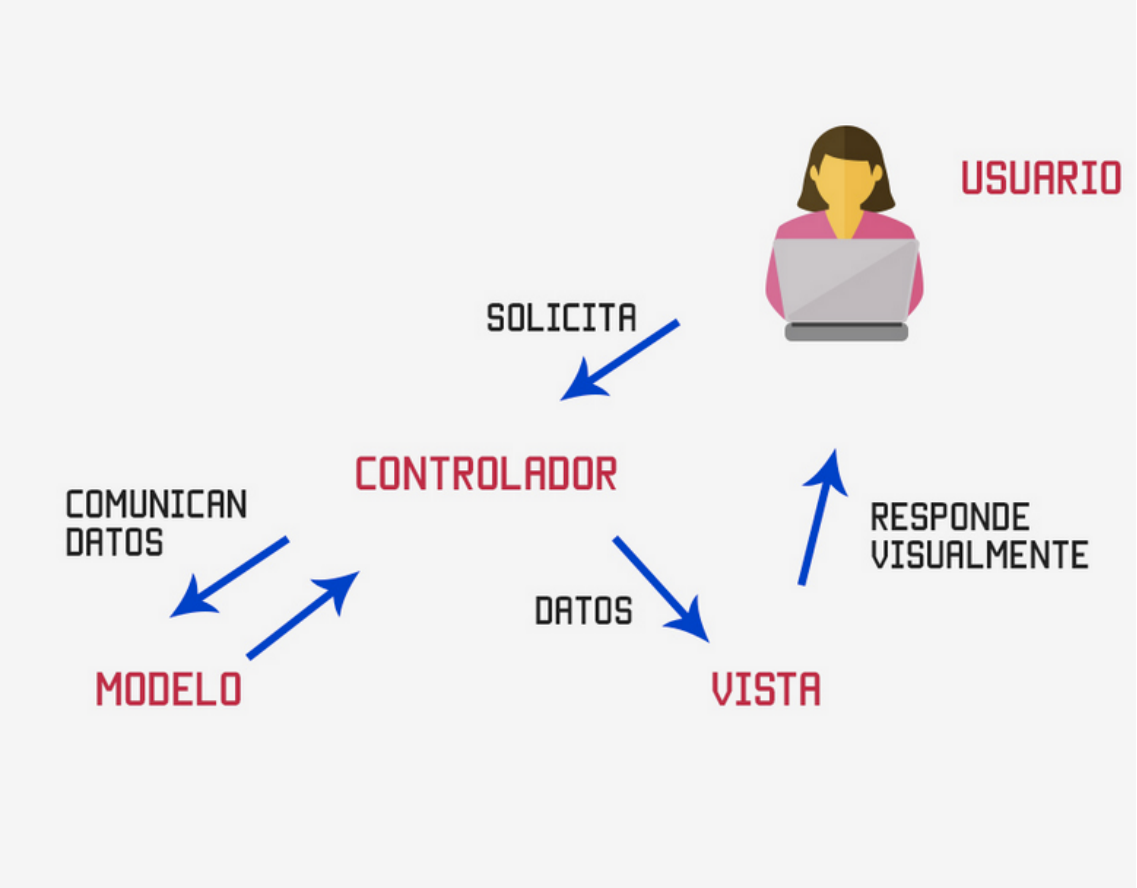
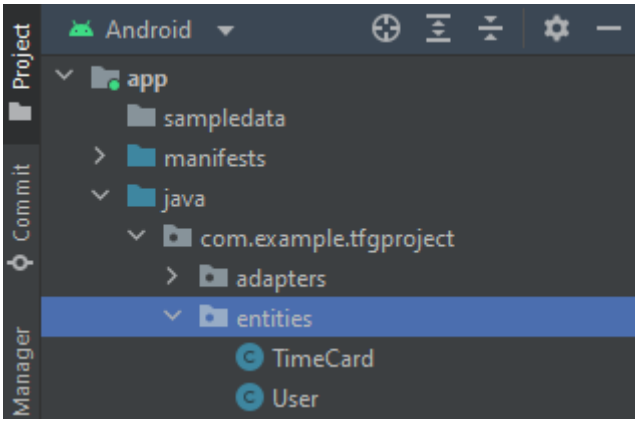


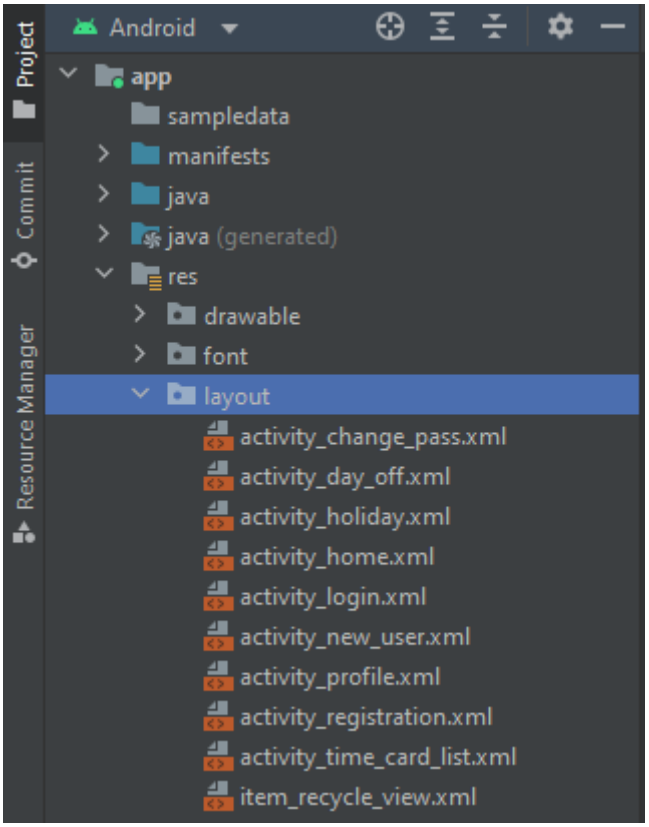
Imagen: codigofacilito.com

En el proyecto el patrón MVC se encuentra compuesto de la siguiente manera:

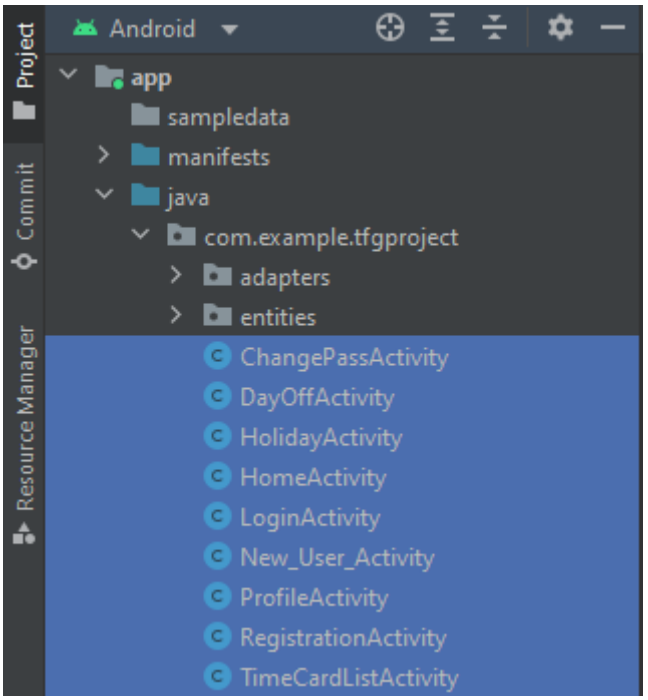
- Modelo.- Se encuentra formado por los ficheros .java que contienen el código de las entidades.



- Vista.- Las vistas en el proyecto se encuentran recogidas por los ficheros XML que conforman los distintos layouts. Los layouts son contenedores que especifican los diferentes elementos de diseño.



- Controlador.- El controlador se encuentra contenido en los ficheros Activity.java que dotan de lógica a la aplicación y que sirven de conexión entre las diferentes vistas.



6. Conclusiones y mejoras del proyecto

Cuando me propuse la ejecución de este proyecto mi meta era elaborar una aplicación que además de cubrir los objetivos funcionales propuestos, su utilización fuese tan fácil que no requiriese de ninguna guía de uso.

La búsqueda de esa facilidad y simplicidad de uso ha tenido como resultado una aplicación que se puede utilizar sin manual de instrucciones y que siempre que el usuario realice una acción recibe una respuesta de la misma indicando el éxito o fracaso de sus interacciones con la aplicación.

La mayor carga de trabajo del proyecto la ha supuesto el aprendizaje y comprensión del funcionamiento de la base de datos Firebase Realtime, utilizada para la persistencia de los datos. Al tratarse de una base de datos noSQL junto con el hecho de que se trata de una base de datos que tiene sus propios métodos de autenticación y funcionamiento han requerido de varias horas de estudio y búsqueda de documentación.

Las funcionalidades iniciales sólamente incluían el registro horario y la visualización de los registros efectuados pero gracias a las sugerencias de mi tutora, amplié dichas funcionalidades a la solicitud de días de permiso y vacaciones.

Viendo el funcionamiento de la aplicación creo que los objetivos iniciales, y aquellos que se incorporaron por el camino, se han conseguido y la aplicación es un prototipo plenamente funcional que puede servir de base para los siguientes desarrollos.

Si bien es cierto que las funcionalidades establecidas al inicio del proyecto se han cumplido, durante el desarrollo del mismo he podido detectar las siguientes mejoras futuras:

- Diseño y colores.- He de reconocer que este siempre ha sido mi punto débil y la interfaz de usuario puede ser mejorada, sobre todo en lo relacionado con el uso de los colores.
- Desarrollo de aplicación web.- Si bien no es el objeto del presente proyecto, me gustaría desarrollar una aplicación web que permita la conexión con la base de datos para que puedan interactuar con ella tantos los empleados, realizando las mismas funciones que desde el terminal android, como los administradores para que puedan gestionar mejor la base de datos.

- Subida y almacenamiento de archivos.- En desarrollos futuros sería muy interesante implementar la subida de archivos (imágenes y pdf) que ampliarían la funcionalidad de la aplicación ya que permitiría registrar justificantes de permisos, gastos,.....

7. Bibliografía

Android Studio https://developer.android.com/studio

Android Studio Fuente: Wikipedia https://es.wikipedia.org/wiki/Android_Studio

Firebase Realtime https://firebase.google.com/docs/database?hl=es-419

StarUML https://staruml.io/

StarUML Fuente: Wikipedia https://en.wikipedia.org/wiki/StarUML

Mockflow (https://www.mockflow.com/)

Git (https://git-scm.com/)

Git Fuente: Wikipedia https://es.wikipedia.org/wiki/Git

GitHub https://github.com/

GitHub Fuente: Wikipedia https://es.wikipedia.org/wiki/GitHub

Java https://www.java.com/es/

XML Fuente: Wikipedia https://es.wikipedia.org/wiki/Extensible_Markup_Language

Diagrama de casos de uso Fuente: Wikipedia https://es.wikipedia.org/wiki/Caso_de_uso

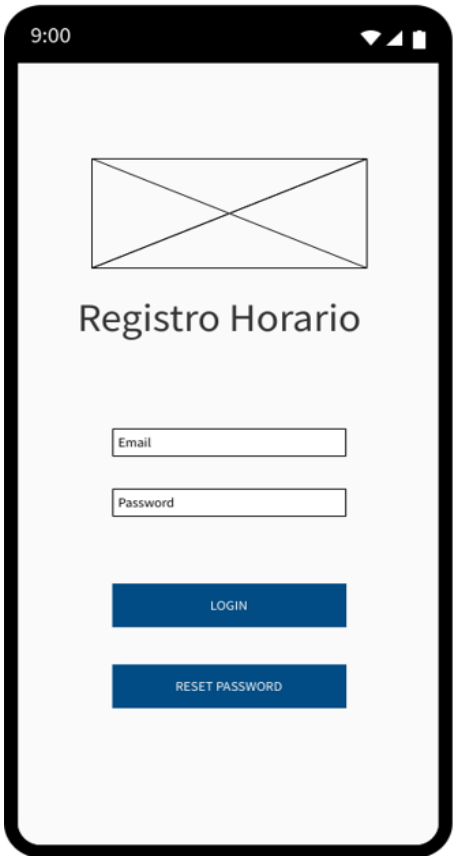
Diagrama de clases Fuente: wikipedia https://es.wikipedia.org/wiki/Diagrama_de_clases

Modelo vista controlador Fuente; wikipedia https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

8. Anexos - Activitys

En esta parte de anexos se va a mostrar y explicar el código más relevante de las diferentes Activity que componen la aplicación.

8.1 LoginActivity



Las funcionalidades que debe cubrir la pantalla de login son 3:

- Login del usuario con los datos cumplimentados.- En este caso debe dar paso a la HomeActivity.
- Login del usuario sin los datos cumplimentados.- Dará paso a NewUserActivity para que se introduzcan todos los datos antes de poder acceder a las funcionalidades de la aplicación.
- Solicitud de restablecimiento de la contraseña.- Muestra ChangePassActivity para solicitar el restablecimiento de la contraseña.

La lógica que dota de funcionalidad se encuentra dentro del método *login()* en donde después de validar el contenido de los inputs, se accede a Firebase Realtime y en el caso de que no existan los datos del usuario se da paso a NewUserActivity para que se proceda a la cumplimentación de dichos datos. En el caso de que los datos del usuario ya se encuentren cargados se inicializa la aplicación a través de HomeActivity.

El código que cumple este cometido es el siguiente:

```java
} else {
    fbAuth.signInWithEmailAndPassword(email, pass)
            .addOnCompleteListener( activity: LoginActivity.this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {

                    // Login correcto
                    if(task.isSuccessful()){
                        userId = fbUser.getUid();
                        // Carga de datos de usuario
                        dbUsersReference.child(userId).addValueEventListener(new ValueEventListener() {
                            @Override
                            public void onDataChange(@NonNull DataSnapshot snapshot) {
                                // Si los datos de usuario no existen, hay que rellenarlos
                                if(!snapshot.exists()){
                                    startActivity(new Intent( packageContext: LoginActivity.this, New_User_Activity.class));
                                } else {
                                    Toast.makeText( context: LoginActivity.this, text: "Sesión iniciada", Toast.LENGTH_LONG).show();
                                    startActivity(new Intent( packageContext: LoginActivity.this, HomeActivity.class));
                                }
                                // Eliminamos el listener
                                dbUsersReference.child(userId).removeEventListener(this);
                            }
                            @Override
                            public void onCancelled(@NonNull DatabaseError error) {
                            }
                        });
                    }
                    // Login erróneo
                    else{
                        Toast.makeText( context: LoginActivity.this, text: "Error: Compruebe las credenciales", Toast.LENGTH_LONG).show();
                    }
                }
            });
}
```

La funcionalidad de recuperación de contraseña se gestiona a través de un listener que invoca a ChangePassActivity.

8.2 ChangePassActivity

La funcionalidad que debe ser cubierta por esta Activity consiste en poder solicitar el restablecimiento o modificación de la contraseña.

El código que otorga esta funcionalidad se muestra a continuación.

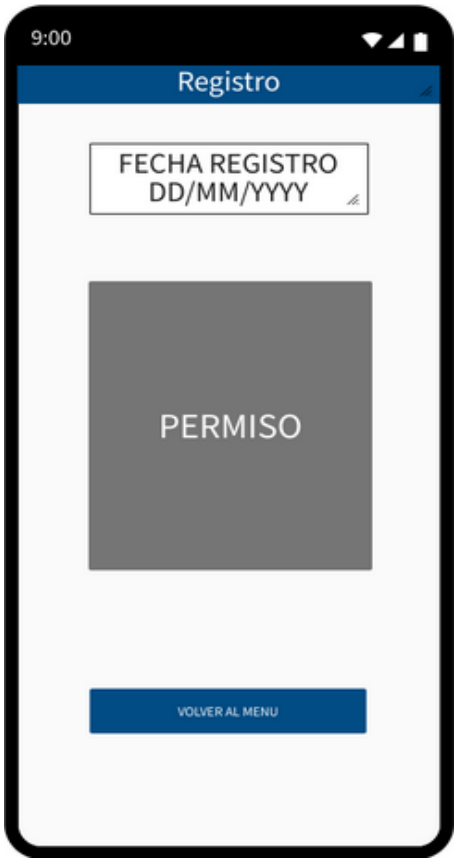
```
64      // Lógica de la solicitud del cambio de contraseña
65      private void resetPassword() {
66
67          fbAuth.setLanguageCode("es");
68          fbAuth.sendPasswordResetEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
69              @Override
70              public void onComplete(@NonNull Task<Void> task) {
71                  if (task.isSuccessful()){
72                      startActivity(new Intent( packageContext: ChangePassActivity.this, LoginActivity.class));
73                  }else{
74                      Toast.makeText( context: ChangePassActivity.this,  text: "Error al reestablecer contraseña", Toast.LENGTH_SHORT).show();
75                  }
76              }
77          });
78      }
```

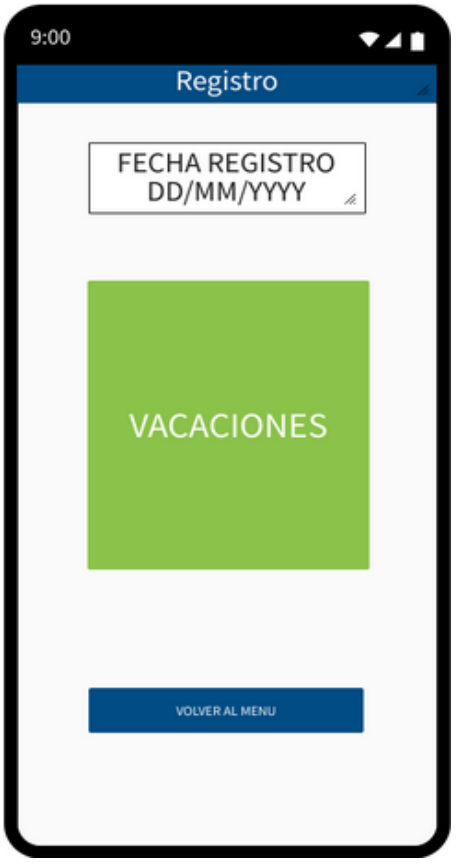
8.3 HomeActivity

Desde esta Activity el usuario podrá acceder a toda la funcionalidad de la aplicación pulsando los diferentes botones.

El código de esta Activity está compuesto por los listeners de los botones que lanzarán el resto de las Activity.

8.4 RegistrationActivity





La ActivityRegistration se gestiona el registro horario de trabajo por parte del trabajador e implementa la siguiente lógica:

- Los fichajes toman como fecha el día actual y no está permitido seleccionar fechas anteriores o posteriores al día actual.
- Antes de poder realizar el registro de entrada, se comprueba si existe algún registro de salida pendiente de realizar con el fin de que no se produzcan incoherencias en los registros.
- Los días de permiso o de vacaciones no está permitido el registro.

La lógica de esta funcionalidad se consigue a través de accesos a la base de datos y condicionales que cambian la vista que se muestra al usuario.

RegistrationActivity.java

```
70      // Lógica de carga de registros abiertos, días de permiso o de vacaciones
71      private void loadRegistration() {
72
73          dbRegistrationsReference.child(userId).orderByChild("date")
74                  .equalTo(today.toString()).addValueEventListener(new ValueEventListener() {
75              @Override
76              public void onDataChange(@NonNull DataSnapshot snapshot) {
77                  if (snapshot.exists()){
78                      for (DataSnapshot s : snapshot.getChildren()) {
79                          // Cargamos la TimeCard con los datos existentes
80                          timeCard = s.getValue(TimeCard.class);
81                          if(timeCard.isDayOff()) {
82                              // Actuamos sobre las vistas
83                              tv_InputTime.setVisibility(View.INVISIBLE);
84                              tv_OutputTime.setVisibility(View.INVISIBLE);
85                              tv_Time.setVisibility(View.INVISIBLE);
86                              btn_Time.setVisibility(View.INVISIBLE);
87                              ibtn_InputTime.setVisibility(View.INVISIBLE);
88                              ibtn_OutputTime.setVisibility(View.INVISIBLE);
89                              tv_DofHol.setText("Permiso");
90                              tv_DofHol.setBackgroundColor(Color.parseColor( colorString: "#546E7A"));
91                              tv_DofHol.setVisibility(View.VISIBLE);
92                          } else if (timeCard.isHoliday()){
93                              // Actuamos sobre las vistas
94                              tv_InputTime.setVisibility(View.INVISIBLE);
95                              tv_OutputTime.setVisibility(View.INVISIBLE);
96                              tv_Time.setVisibility(View.INVISIBLE);
97                              btn_Time.setVisibility(View.INVISIBLE);
98                              ibtn_InputTime.setVisibility(View.INVISIBLE);
99                              ibtn_OutputTime.setVisibility(View.INVISIBLE);
100                             tv_DofHol.setText("Vacaciones");
101                             tv_DofHol.setBackgroundColor(Color.parseColor( colorString: "#FF018786"));
102                             tv_DofHol.setVisibility(View.VISIBLE);
103                         }
104                     }
105                 }
106             }
107             @Override
108             public void onCancelled(@NonNull DatabaseError error) {
109             }
110     });
```

RegistrationActivity.java

```
        dbRegistrationsReference.child(userId).orderByChild("outTime")
                .equalTo("").addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if(snapshot.exists()){
                            for (DataSnapshot s : snapshot.getChildren()) {
                                // Cargamos la TimeCard con los datos existentes
                                timeCard = s.getValue(TimeCard.class);
                                // Vistas de salida
                                tv_InputTime.setVisibility(View.VISIBLE);
                                tv_OutputTime.setVisibility(View.VISIBLE);
                                tv_Time.setVisibility(View.VISIBLE);
                                btn_Time.setVisibility(View.VISIBLE);
                                tv_OutputTime.setVisibility(View.VISIBLE);
                                tv_OutputTime.setText("SALIDA: ");
                                ibtn_OutputTime.setVisibility(View.VISIBLE);
                                // Cargamos las vistas
                                tv_Date.setText(timeCard.getDate());
                                tv_InputTime.setText("ENTRADA: " + timeCard.getEntryTime());
                                // Ocultamos el fichaje de entrada y permisos / vacaciones
                                ibtn_InputTime.setVisibility(View.INVISIBLE);
                                tv_DofHol.setVisibility(View.INVISIBLE);
                                // Obtenemos el id del nodo
                                idRegistration = s.getKey();
                            }
                        } else  {
                            tv_OutputTime.setVisibility(View.INVISIBLE);
                            ibtn_OutputTime.setVisibility(View.INVISIBLE);
                        }
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                    }
                });
    }

```

Una vez validados los registros existentes, el usuario puede acceder al registro de la jornada seleccionando la hora de entrada a través de un selector.

```
    private void changeTime() {

        TimePickerDialog timePickerDialog = new TimePickerDialog( context: this, R.style.PickerStyle,new TimePickerDialog.OnTimeSetListener() {

            @Override
            public void onTimeSet(TimePicker timePicker, int h, int m) {
                hour =h;
                minute = m;
                String selectedTime = String.format("%02d", h) + ":" + String.format("%02d", m);
                tv_Time.setText("HORA SELECCIONADA: " + selectedTime);
            }

        }, hour, minute,  is24HourView: true);

        timePickerDialog.show();
    }
}
```

RegistrationActivity.java

```java
// Lógica que gestiona el registro de la hora de entrada
private void entryTime() {

    String entryTime = String.format("%02d", hour) + ":" + String.format("%02d", (minute));
    TimeCard timeCard = new TimeCard();
    timeCard.setDate(today.toString());
    timeCard.setEntryTime(entryTime);
    timeCard.setOutTime("");
    timeCard.setReason(null);
    timeCard.setUserId(userId);

    dbRegistrationsReference.child(userId).push()
            .setValue(timeCard).addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {
            if (task.isSuccessful()){
                Toast.makeText( context: RegistrationActivity.this, text: "Entrada registrada", Toast.LENGTH_SHORT).show();
                startActivity(new Intent(getApplication(), HomeActivity.class));
            } else {
                Toast.makeText( context: RegistrationActivity.this, text: "Error al guardar el registro", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

En el caso de que el usuario esté realizando el registro de salida, se comprobará que la hora de salida es mayor que la de entrada.

RegistrationActivity.java

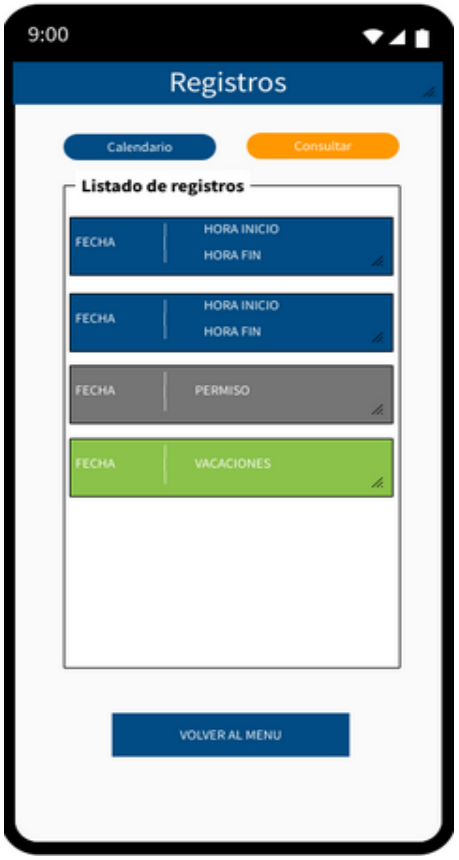
```java
// Lógica que gestiona el registro de la hora de salida
private void exitTime() {

    String outTime = String.format("%02d", hour) + ":" + String.format("%02d", (minute));

    // Validación hora salida >= hora entrada
    if(outTime.compareTo(timeCard.getEntryTime()) >= 0){
        timeCard.setOutTime(outTime);
        timeCard.setReason(null);
        TimeCard tc = timeCard;
        dbRegistrationsReference.child(userId).child(idRegistration)
                .setValue(tc).addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()){
                    Toast.makeText( context: RegistrationActivity.this, text: "Salida registrada", Toast.LENGTH_SHORT).show();
                    startActivity(new Intent(getApplication(), HomeActivity.class));
                } else {
                    Toast.makeText( context: RegistrationActivity.this, text: "Error al guardar el registro", Toast.LENGTH_SHORT).show();
                }
            }
        });

    } else {
        Toast.makeText( context: RegistrationActivity.this, text: "La salida debe ser posterior a la entrada", Toast.LENGTH_SHORT).show();
    }
}
```

8.5 TimeCardListActivity

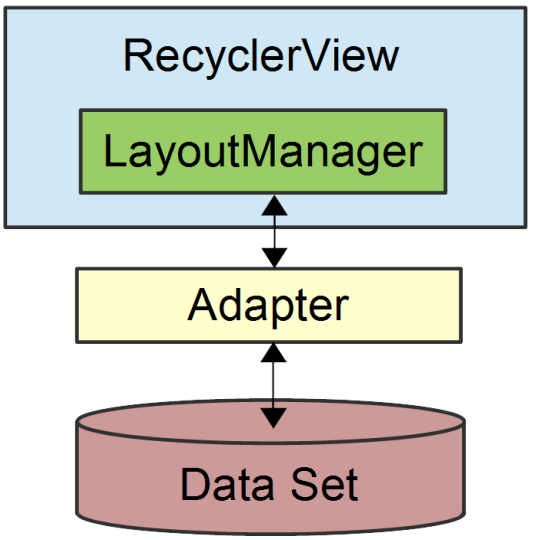


Esta Activity tiene por objeto mostrar los registros horarios, permisos y vacaciones que ha realizado el usuario en orden descendente cronológicamente.

También permite que el usuario seleccione una determinada fecha para visualizar sus registros.

Cada registro existente tiene una codificación mediante colores: registros horarios - azul, permisos - gris y vacaciones - verde.

El elemento fundamental de esta Activity es el RecyclerView que nos va a permitir gestionar de manera más eficiente que el ListView la visualización de los registros del usuario.



El RecyclerView precisa de un Adapter que va servir de conexión con Firebase Realtime y que va a cargar la información en los elementos que el RecyclerView va a gestionar.

El código que carga el RecyclerView es el siguiente:

TimeCardListActivity.java | Adapter.java

```
87
88      private void loadRecycleview() {
89          rv.setLayoutManager(new LinearLayoutManager( context: this));
90          list = new ArrayList<>();
91          adapter = new Adapter (list);
92          rv.setAdapter(adapter);
93      }
```

Y los datos de la lista se obtienen con el código:

TimeCardListActivity.java

```
108     private void loadTimeCardList() {
109
110         FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
111         String id = user.getUid();
112
113         fbDatabase.getReference( path: "registrations").child(id)
114                 .orderByChild("date").addValueEventListener(new ValueEventListener() {
115             @Override
116             public void onDataChange(@NonNull DataSnapshot snapshot) {
117                 if(snapshot.exists()){
118                     list.clear();
119                     for (DataSnapshot s: snapshot.getChildren()){
120                         TimeCard timeCard = s.getValue(TimeCard.class);
121                         list.add(timeCard);
122                         adapter.notifyDataSetChanged();
123                     }
124                     Collections.reverse(list);
125                     tv_infoDate.setText("");
126                 }else{
127                     tv_infoDate.setText("No existen registros");
128                 }
129             }
130
131             @Override
132             public void onCancelled(@NonNull DatabaseError error) {
133             }
134         });
135     }
```

Por el lado del Adapter tenemos:

Adapter.java

```java

public class Adapter extends RecyclerView.Adapter<Adapter.TimeCardViewHolder> {

    List<TimeCard> list;

    public Adapter(List<TimeCard> list) { this.list = list; }

    public static class TimeCardViewHolder extends RecyclerView.ViewHolder {

        private TextView date;
        private TextView entry;
        private TextView out;
        private TextView dayOff;
        private CardView cardView;

        public TimeCardViewHolder(@NonNull View itemView) {
            super(itemView);
            date = (TextView) itemView.findViewById(R.id.textview_date);
            entry = (TextView) itemView.findViewById(R.id.textview_entry);
            out = (TextView) itemView.findViewById(R.id.textview_out);
            dayOff = (TextView) itemView.findViewById(R.id.textview_dayOff);
            cardView = (CardView) itemView.findViewById(R.id.card_viewOut);
        }
    }
```

Adapter.java

```java

    @NonNull
    @Override
    // Generamos el holder
    public TimeCardViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_recycle_view, parent, attachToRoot: false);
        TimeCardViewHolder holder = new TimeCardViewHolder(v);
        return holder;
    }

    // Cargamos los elementos especificando su holder
    @Override
    public void onBindViewHolder(@NonNull TimeCardViewHolder holder, int position) {
        TimeCard timecard = list.get(position);
        if (timecard.isDayOff()) {
            holder.date.setText(timecard.getDate());
            holder.entry.setVisibility(View.INVISIBLE);
            holder.out.setVisibility(View.INVISIBLE);
            holder.dayOff.setVisibility(View.VISIBLE);
            holder.dayOff.setText("Permiso");
            holder.cardView.setCardBackgroundColor(Color.parseColor( colorString: "#546E7A"));

        } else if (timecard.isHoliday()) {
            holder.date.setText(timecard.getDate());
            holder.entry.setVisibility(View.INVISIBLE);
            holder.out.setVisibility(View.INVISIBLE);
            holder.dayOff.setVisibility(View.VISIBLE);
            holder.dayOff.setText("Vacaciones");
            holder.cardView.setCardBackgroundColor(Color.parseColor( colorString: "#FF018786"));

        } else {
            holder.date.setText(timecard.getDate());
            holder.entry.setVisibility(View.VISIBLE);
            holder.out.setVisibility(View.VISIBLE);
            holder.entry.setText("Inicio: " + timecard.getEntryTime());
            holder.out.setText("Fin: " + timecard.getOutTime());
            holder.dayOff.setVisibility(View.INVISIBLE);
            holder.cardView.setCardBackgroundColor(Color.parseColor( colorString: "#3797CF"));
        }
    }
```

Como puede apreciarse, el holder modificará su aspecto en función de los datos que contenga la *TimeCard* para diferenciar visualmente el tipo de registro.

```
84      // Número de elementos del RecyclerView
85      @Override
86      public int getItemCount() { return list.size(); }
89
90  }
```

8.6 DayOffActivity

La solicitud de días de permiso se realiza a través de esta Activity.

El usuario deberá introducir el motivo del día de permiso y la fecha, que siempre deberá ser igual o mayor al día actual.

Además, no se podrán solicitar permisos en fechas que ya contienen registros.

Una vez que se compruebe que se cumple lo anterior, se pasará a registrar el día solicitado como permiso.

La lógica que controla los permisos se encuentra en:

DayOffActivity.java

```
97          // No se pueden solicitar permisos con retroactividad
98          else if (today.compareTo(date) <= 0){
99
100             dbRegistrationsReference.child(userId).orderByChild("date")
101                     .equalTo(date).addValueEventListener(new ValueEventListener() {
102                 @Override
103                 public void onDataChange(@NonNull DataSnapshot snapshot) {
104                     // No se pueden solicitar permisos en fechas que ya tienen registros
105                     if (!snapshot.exists()){
106                         TimeCard tc = new TimeCard();
107                         tc.setDate(date);
108                         tc.setEntryTime(null);
109                         tc.setOutTime(null);
110                         tc.setDayOff(true);
111                         tc.setReason(reason);
112                         tc.setUserId(userId);
113
114                         dbRegistrationsReference.child(userId).push()
115                                 .setValue(tc).addOnCompleteListener(new OnCompleteListener<Void>() {
116                             @Override
117                             public void onComplete(@NonNull Task<Void> task) {
118                                 if (task.isSuccessful()){
119                                     Toast.makeText( context: DayOffActivity.this,  text: "Permiso registrado", Toast.LENGTH_SHORT).show();
120                                     startActivity(new Intent(getApplicationContext(), HomeActivity.class));
121                                 }
122                             }
123                         });
124                     } else {
125                         Toast.makeText( context: DayOffActivity.this,  text: "Ya existe un registro para ese día", Toast.LENGTH_SHORT).show();
126                     }
127                     // Eliminamos el listener
128                     dbRegistrationsReference.child(userId).removeEventListener(this);
129                 }
130
131                 @Override
132                 public void onCancelled(@NonNull DatabaseError error) {
133                 }
134             });
135
136         } else { Toast.makeText( context: DayOffActivity.this,  text: "La fecha debe ser igual o mayor a hoy", Toast.LENGTH_SHORT).show(); }
137     }
```

8.7 HolidayActivity

La solicitud de vacaciones se organiza a través de esta Activity.

El usuario deberá seleccionar la fecha de inicio y de final de las vacaciones mediante los selectores y se tendrán que cumplir las siguientes condiciones:

- La fecha de inicio debe ser superior a la fecha actual.
- La fecha de fin debe ser igual o superior a la fecha de inicio.

Una vez que se comprueba lo anterior, en un ArrayList se cargan todas las fechas comprendidas entre la fecha de inicio y la fecha de fin y a través de un bucle for each se persisten los registros.

El código que implementa la lógica de esta funcionalidad es el siguiente:

HolidayActivity.java

```
132
133         // Las vacaciones no se pueden pedir con retroactividad, hay que solicitarlas con antelación de al menos un dia
134         // y hay que validar las fechas de inicio y de fin
135         else if (startHoliday.compareTo(endHoliday) <= 0 && (startHoliday.compareTo(today.toString())) > 0){
136
137             // Cargamos un array con las fechas solicitadas de vacaciones
138             LocalDate ld_startHoliday = LocalDate.parse(startHoliday);
139             LocalDate ld_endHoliday = LocalDate.parse(endHoliday);
140
141             while (!ld_startHoliday.isAfter(ld_endHoliday)) {
142                 holidays.add(ld_startHoliday.toString());
143                 ld_startHoliday = ld_startHoliday.plusDays(1);
144             }
```

```java
for (String d : holidays) {

    dbRegistrationsReference.child(userId).orderByChild("date")
            .equalTo(d).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if (!snapshot.exists()){
                TimeCard tc = new TimeCard();
                tc.setDate(d);
                tc.setEntryTime(null);
                tc.setOutTime(null);
                tc.setReason(null);
                tc.setDayOff(false);
                tc.setHoliday(true);
                tc.setUserId(userId);

                dbRegistrationsReference.child(userId).push()
                        .setValue(tc).addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        if(!task.isComplete()){
                            Toast.makeText( context: HolidayActivity.this, text: "Error en registro", Toast.LENGTH_SHORT).show();
                        }
                    }
                });
            }
            // Eliminamos el listener
            dbRegistrationsReference.child(userId).removeEventListener(this);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
        }
    });
}

Toast.makeText( context: this, text: "Vacaciones registradas", Toast.LENGTH_SHORT).show();
startActivity(new Intent( packageContext: HolidayActivity.this, HomeActivity.class));

} else { Toast.makeText( context: HolidayActivity.this, text: "Revisa las fechas solicitadas", Toast.LENGTH_SHORT).show();}
}
```

8.8 NewUserActivity / ProfileActivity

Estas dos Activity son muy similares ya que en ambos caso lo que se hace es comprobar que los campos de los datos del usuario no se encuentran vacíos para a continuación grabar los datos en Firebase Realtime.

```
        } else {

            User user = new User();
            user.setUserid(userId);
            user.setEmail(userEmail);
            user.setNif(nif);
            user.setName(name);
            user.setDateOfBirth(dob);
            user.setPhone(phone);

            dbUsersReference.child(userId).setValue(user).addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        Toast.makeText(getApplication(), text: "Usuario modificado", Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(getApplicationContext(), HomeActivity.class));
                    } else {
                        Toast.makeText(getApplication(), text: "Error al guardar el registro", Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    }
```