I.E.S LAS SALINAS

PROYECTO FINAL FIN DE GRADO

CURSO 22-23

Taskodoro

CICLO FORMATIVO GRADO SUPERIOR

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Tutor: Oscar Lillo Díaz

Autor: Jorge Borja Valdivia

Resumen

Emplear efectivamente nuestro tiempo y completar las tareas que nos proponemos es el objetivo de cualquier persona y lo que la mayoría de empresas quieren de sus empleados y el uso de aplicaciones para ello cada vez es más extendido.

Este trabajo abordó el desafío de encontrar necesidades no cubiertas por las aplicaciones ya presentes en el mercado y cubrirlas, además, de las decisiones de diseño para garantizar una usabilidad clara y una interfaz simple y minimalista.

El funcionamiento de la aplicación se resumiría en la creación de sesiones con sus tareas, la instancia de la sesión en sí, donde tendremos el temporizador personalizable para adaptarse a la necesidades del usuario y la lista de tareas las cuales podremos modificar su estado de forma rápida mediante un checkbox para mejorar la visibilidad de la aplicación y hacerla más intuitiva. Una vez acabada la sesión, podremos ver el tiempo empleado en ella en la pantalla de historial de sesiones.

Todos los datos de la sesión y del usuario son almacenados en Firebase, ya que nos ofrece un sistema de bases de datos no relacionales que se adecua a las necesidades de la aplicación y nos garantiza una seguridad ante posibles ataques a la base de datos.

Taskodoro es una aplicación minimalista para mejorar nuestras sesiones de pomodoros y poder poner más el foco en las tareas y que ofrece una interfaz amigable y relajada.

Abstract

Effectively using our time and completing the tasks we set ourselves is the goal of any person and what most companies want from their employees and the use of applications for this purpose is becoming more and more widespread.

This work addressed the challenge of finding needs not covered by the applications already present in the market and covering them, in addition to design decisions to ensure a clear usability and a simple and minimalist interface.

The operation of the application would be summarized in the creation of sessions with their tasks, the instance of the session itself, where we will have a customizable timer to suit the user's needs and the list of tasks which we can modify their status quickly through a checkbox to improve the visibility of the application and make it more intuitive. Once the session is finished, we can see the time spent on it in the session history screen.

All session and user data are stored in Firebase, since it offers a non-relational database system that suits the needs of the application and guarantees security against possible attacks to the database.

Taskodoro is a minimalist application to improve our pomodoros sessions and to be able to put more focus on the tasks and that offers a friendly and relaxed interface.

ÍNDICE

Justificación

Taskodoro nace para solventar el problema del manejo del tiempo de productividad y el saber establecer los objetivos del mismo, para así conseguir ser más productivos y que cada jornada de trabajo sea aprovechada al máximo. Pese la popularización de este método la mayoría de aplicaciones existentes se centran únicamente en el temporizador simple y las adicciones que añaden son más visuales y no se centran en el buen uso del método en sí mismo, aquí es donde Taskodoro se diferencia del resto y proporciona al usuario herramientas para poder mejorar su forma de administrar el tiempo activo y sacar el máximo partido de cada jornada.

Mediante la investigación de estudios que tratan el tema del uso de temporizadores pomodoros y el funcionamiento del cerebro y su capacidad de mantener la concentración se establece la idea de tener una lista de tareas para poder mantener un flujo de trabajo que permita al usuario seguir el ritmo de la sesión sin desperdiciar su tiempo y de esa forma completar su trabajo sin perder la concentración.

Por último, la aplicación tiene como target al sector de la informática, donde muchas empresas requieren a sus empleados llevar un registro del tiempo que tardan en realizar sus tareas, en Taskodoro estas empresas encontrarán una solución completa y fácil de usar.

Introducción

El proyecto procede de la posibilidad de cubrir un hueco de mercado y de la necesidad de diversas empresas de tener un temporizador pomodoro, lista de tareas y registros de tiempo en una sola aplicación, estos criterios fueron definidos después de analizar el flujo de trabajo de numerosos equipos y comprobar que al usar temporizadores pomodoros sin analizar tareas se perdía el foco del trabajo y generaba conflictos en el flujo de trabajo del grupo.

El proyecto fue llevado a cabo por el equipo de desarrollo de Kaizen Ware, una empresa dedicada al software, con el fin de lanzar una app que reuniera las características definidas anteriormente y siguiendo la metodología de la empresa. El resultado fue una aplicación fácil de usar y que permite a los empleados manejar sus sesiones de pomodoro de una forma cómoda y óptima, ayudando al usuario final a trabajar de forma más ordenada y mejorando la forma en la que se realizan las tareas diarias.

Organización

La empresa KaizenWare, genera soluciones tecnológicas, por lo tanto ya tienen un recorrido en la creación de software, Taskodoro es su primer desarrollo móvil, por ello los recursos destinados no han sido abundantes y el proyecto en sí es más una primera entrada al mercado para coger experiencia y testear las oportunidades futuras.

Al no tratarse del negocio central de la empresa y ser un desarrollo nuevo la aplicación estaría en su primera versión, esto se debe a que se diseñó mediante la metodología waterfall y en el futuro se podría entregar una nueva versión añadiendo características si se necesitaran, esto ayuda a recoger feedback de los usuarios de forma inmediata y permite a la empresa mejorar la aplicación poco a poco.
El equipo encargado del desarrollo estaría formado únicamente por dos integrantes del equipo de desarrollo de la empresa.

El equipo estaría dividido en:

- Un encargado del análisis de requisitos y el diseño de la arquitectura de la aplicación
- Un encargado de la programación y diseño gráfico de la aplicación

Ambos empleados disponen del mismo sueldo, el cual es de 20.000€ brutos al año, suponiendo un coste mensual a la empresa de 2.166€ y un total de 12.996€ durante los 3 meses del desarrollo de la aplicación.

En cuanto a los equipos usados por los empleados son unos portátiles Vant MOOVE3-14, ambos usan el sistema operativo Ubuntu 22.04.2 LTS, con las siguientes especificaciones:

En cuanto a las ganancias de la empresa por el desarrollo de la app se optó por un sistema de anuncios no intrusivos dentro de la aplicación, esto se hizo después de analizar a los competidores y ver que los usuarios preferían este tipo de modelo de monetización.

Objetivos

El objetivo principal del proyecto se podría resumir en facilitar el manejo del tiempo a los usuarios de la aplicación y ofrecer una manera cómoda de analizar tus sesiones pomodoro y ver cómo poder mejorarlas.

Los objetivos de la aplicación serían:

- **Mejorar la gestión del tiempo** mediante la posibilidad de ver cuánto tiempo usa en sus sesiones
- **Facilitar** la forma de definir tareas a la hora de usar el método pomodoro
- **Ayudar** con la capacidad de mantener la concentración durante los periodos de productividad

- **Aportar utilidad** al usuario a la hora de estudiar, trabajar, etc.

Desarrollo

El desarrollo del proyecto ha estado formado por una aplicación para smartphone realizada en el IDE Android Studio usando Java como lenguaje para programar la parte lógica de la aplicación.

Metodología

Para el desarrollo del proyecto se eligió la metodología waterfall, ya que es la que más se adecuaba al tipo de proyecto y permite tener un ciclo de desarrollo completo para generar una primera versión de la aplicación completamente funcional que puede ir mejorando y actualizando a medida que se requiera, al usar la gestión en cascada estamos planificando el proyecto a largo plazo y creando una estructura desde el principio que será llevada a cabo durante todo el proyecto.

Esquema metodología waterfall

- En la fase de requisitos se analizan todas las necesidades del proyecto, su viabilidad y la rentabilidad, a partir de este primer análisis se va a crear un plan que se va a seguir durante el resto de desarrollo, ya que la metodología en cascada no permite cambios grandes durante el desarrollo.
- Una vez detallados todos los requisitos y la viabilidad del proyecto se pasa a la fase de diseño donde se formula una solución en base a las exigencias, tareas y estrategias definidas en la fase anterior.
- Terminada la fase de diseño, se generó el plan a seguir en la fase de desarrollo, en esta fase se ejecutó la programación de la aplicación, la búsqueda de errores, diseño de la interfaz, etc.
- Al acabar la fase de desarrollo teníamos ya el producto funcional, el cual es probado en la siguiente fase, las pruebas consistieron en integrar el proyecto en el entorno

seleccionado para el mismo, estas pruebas fueron establecidas en la primera fase como las exigencias del proyecto, una vez pasadas las pruebas el producto es considerado como válido y está listo para ser lanzado.

- La última fase constó de la entrega de la aplicación y su mantenimiento así como la mejora del producto, la cual consistirá en recorrer el ciclo otra vez con sus nuevas exigencias y objetivos.

La selección de esta metodología sirve para garantizar que el producto final es el deseado desde el principio ya que al obligarnos a en cada paso haber completado correctamente el anterior siempre se está comprobando el desarrollo del proyecto.

Además del uso de desarrollo en cascada se usó el sistema de tableros Kanban para organizar el análisis de las tareas técnicas, nos decantamos por usar la herramienta Trello por su popularidad y la facilidad de uso, además de contar con una licencia gratuita que nos aportaba todo lo necesario.

Fase de requisitos

Como se mencionó antes esta fase fue la más importante del proyecto, ya que fue donde se declaró toda la lista de objetivos, requisitos, análisis del producto, etc. En este punto se hicieron estudios de mercado para ver qué factores podrían hacer al producto destacar ante la competencia, se establece la identidad del proyecto, siendo esta la de generar un producto capaz de mejorar al usuario en su dia a dia y dándole un valor a largo plazo, esta idea es extraída de la filosofía Kaizen. También, se realizaron investigaciones referentes a los problemas que pretende solventar la aplicación y obteniendo como resultados las tácticas a emplear para conseguirlo.

También se realizó un estudio para elegir el sistema operativo sobre el que desarrollar la aplicación, el elegido fue Android y en concreto se establece como versión mínima la conocida como Android Oreo o Android 8, esto nos garantiza cubrir la mayoría de dispositivos del mercado.

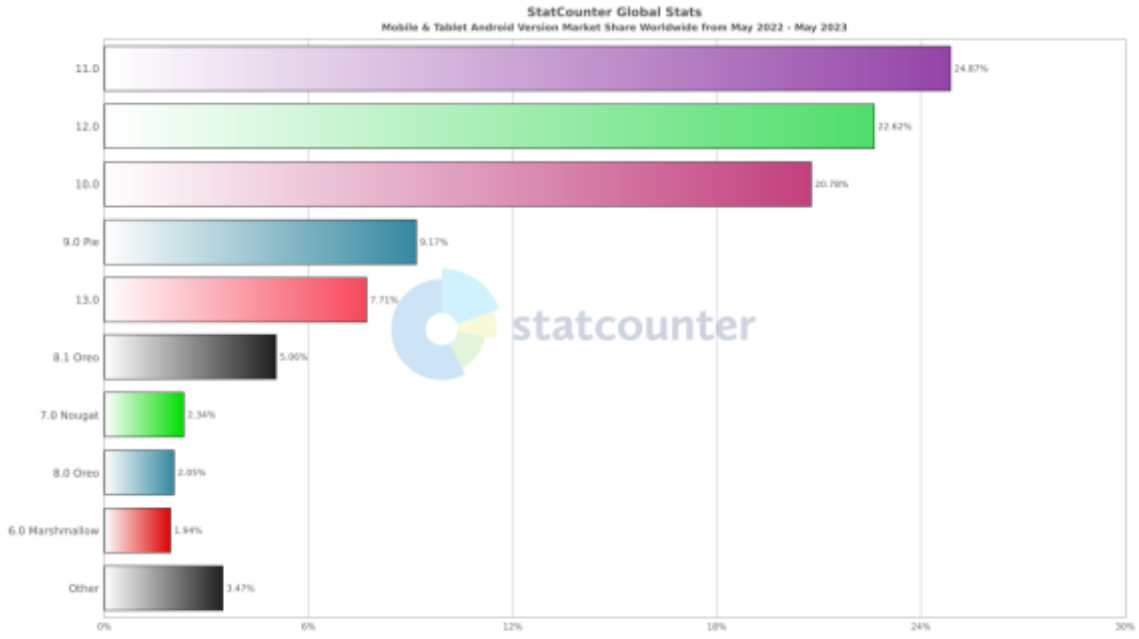


Tabla de porcentaje de uso de versiones de Android

A partir de esta fase fue donde el proyecto se establece como válido, se decidió continuar con su ejecución, pasando así al apartado de diseño.

Fase de diseño

Partiendo de las directrices marcadas en la etapa de análisis de requisitos se plantea el diseño del proyecto. Dentro del diseño hay varios factores a determinar:

Tecnologías

Las tecnologías usadas en el proyecto son el IDE de Android Studio, siendo el más usado para crear aplicaciones de Android y ofreciendo muchas comodidades a la hora de desarrollar la aplicación. Como lenguaje de programación se escogió Java al ser un lenguaje con mucho recorrido y en concreto se usó la versión 8.

Para la gestión de la base de datos se optó por usar Firebase en su versión spark, la cual es gratuita y se adapta a los requisitos del proyecto. Se decidió optar por Firebase ya que al ser un sistema de bases de datos no relacionales se adapta al proyecto y disminuye la complejidad de las consultas haciéndolo más ágil, además, Firebase nos aporta una seguridad ante posibles ataques y una fácil gestión de autenticación de usuarios.

También se usó la librería Volley para realizar llamadas API a Inspiration, una API gratuita que proporciona frases motivadoras.

Diseño gráfico de la aplicación

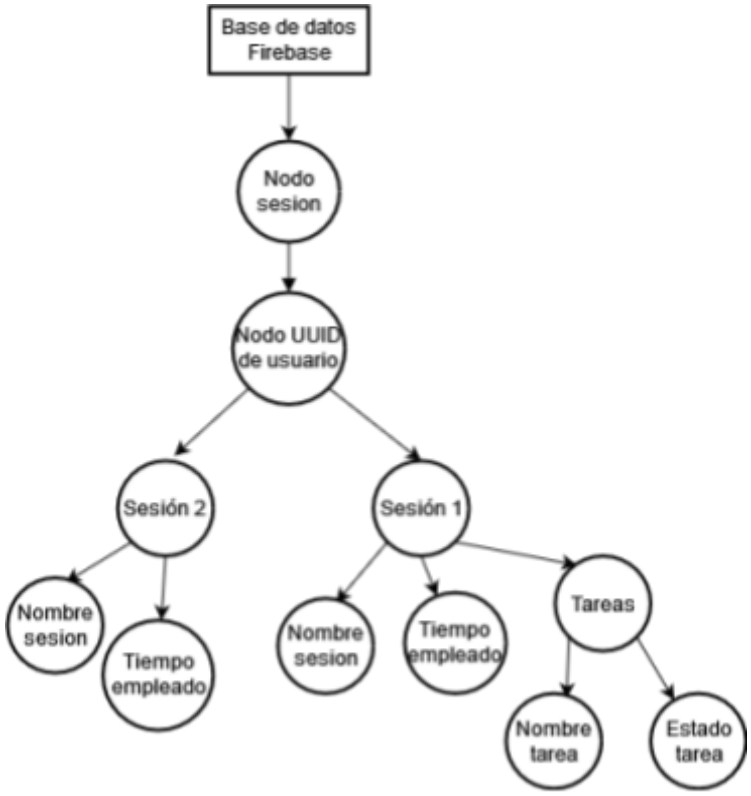
Se creó un prototipo de la interfaz gráfica de las principales funciones de la aplicación, este prototipo servirá a tipo de sketch a la hora de más tarde generar la interfaz gráfica final.

Esquema prototipo de la interfaz gráfica

Estructura de datos y flujo de la aplicación

En este caso la arquitectura de la aplicación es Cliente-Servidor, siendo el cliente la aplicación y el servidor Firebase, al cual se le harán peticiones de escritura y lectura.

Firebase se estructura por nodos de manera que no creamos una tabla si no una estructura de datos, la estructura es la siguiente, siendo el nodo de tareas opcional al usuario poder o no crear tareas para la sesión:



Esquema de datos de la aplicación

Para estructurar el uso de la aplicación se crea un diagrama de flujo estándar el cual será el seguido a la hora de realizar el desarrollo de la parte lógica.

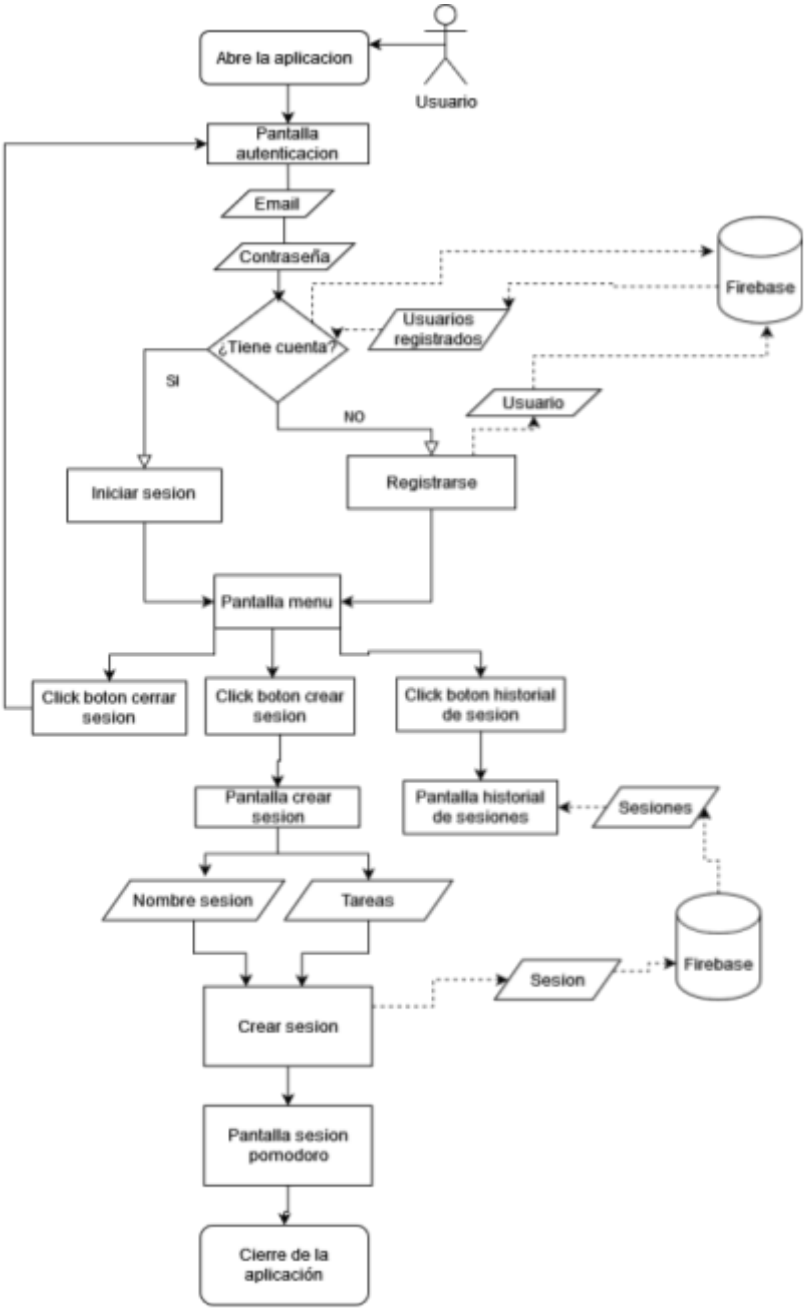


Diagrama de flujo de la aplicación

Estructura de clases

En esta fase se declaran las dependencias que tendrá cada clase para así tener la arquitectura de la aplicación ya generada y agilizar el proceso de desarrollo. La estructura del programa se puede ver en el diagrama de clases generado una vez terminada la aplicación, el cual muestra los atributos de clase, variables, métodos, etc.

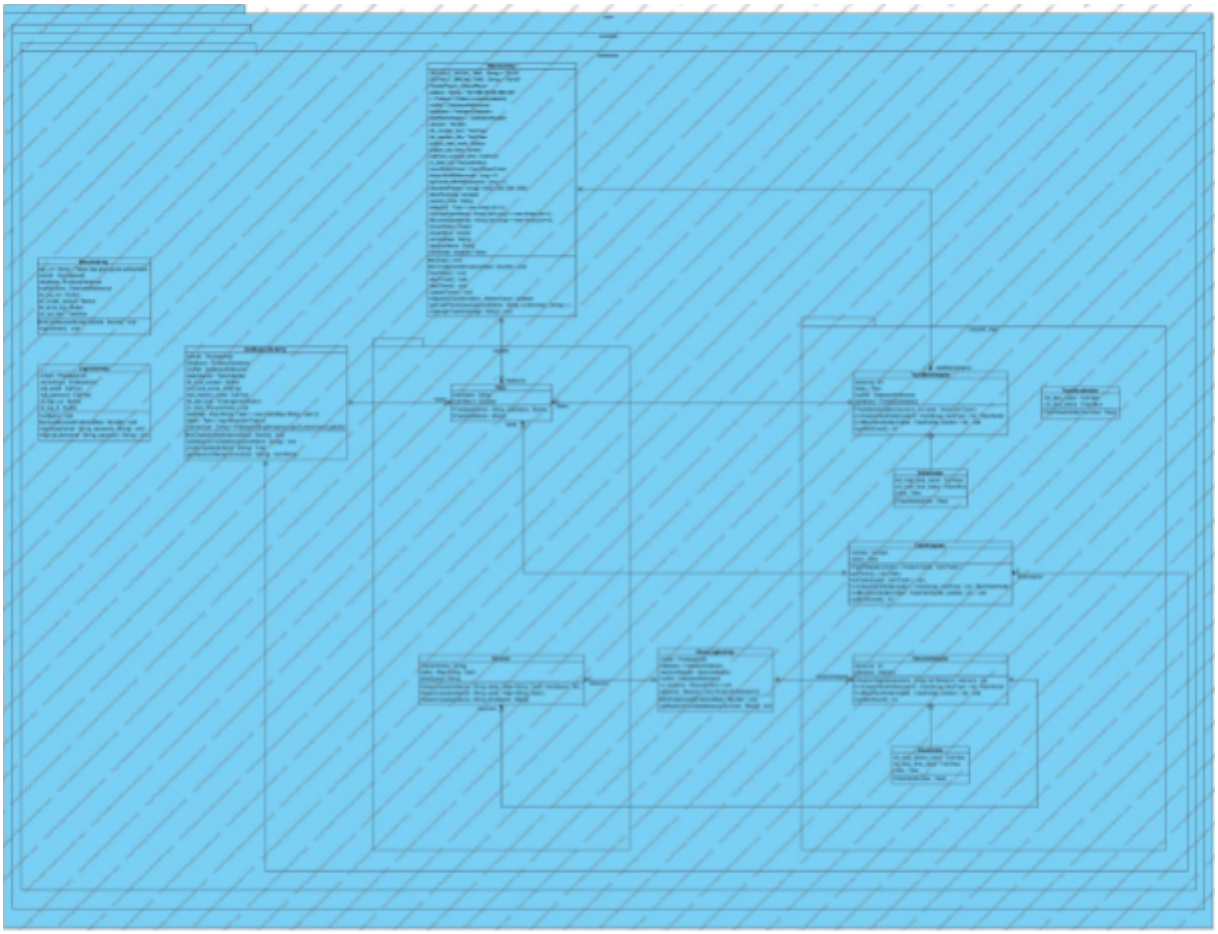


Diagrama de clases de la aplicación

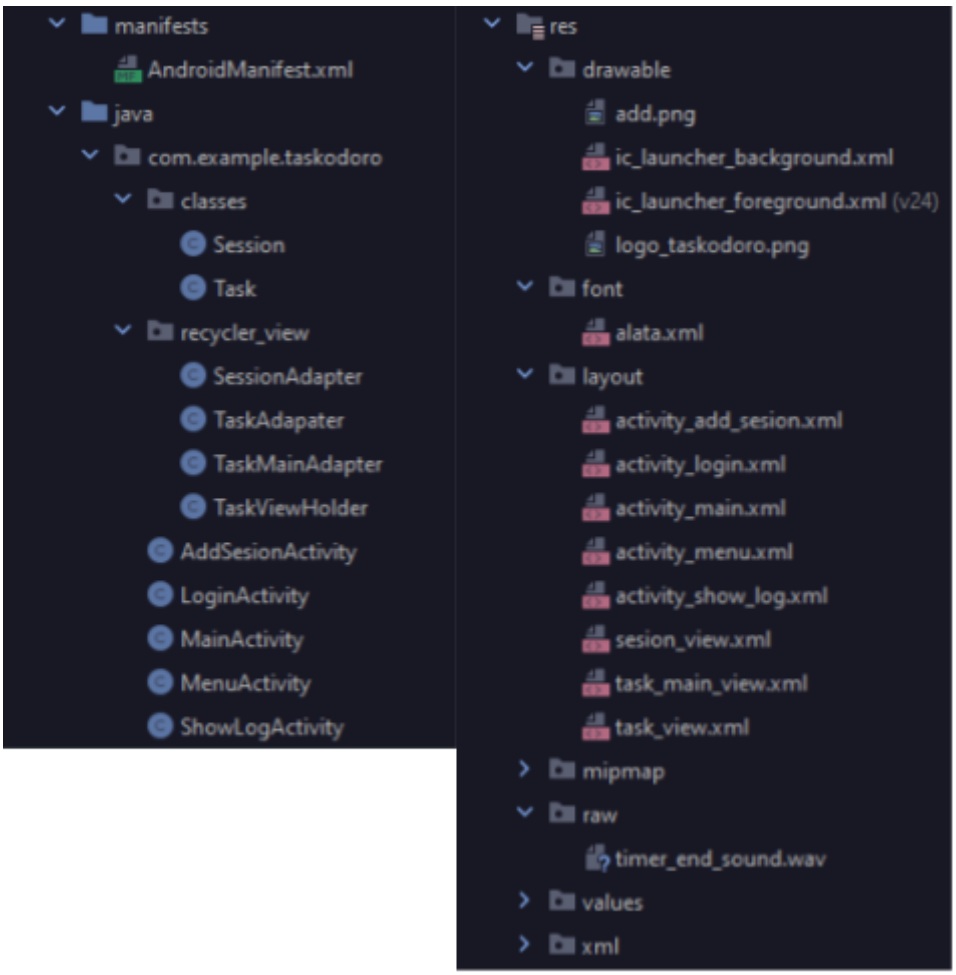
Fase de desarrollo

La fase de desarrollo es la etapa más larga debido a que es donde se aplica todo lo antes analizado y diseñado.

La aplicación se empezó con un proyecto de Android desde cero y se fueron implementando todas las funcionalidades de la app. Se trabajó la aplicación añadiendo funcionalidad junto a su interfaz gráfica, para ir añadiendo cada funcionalidad después de su testeo y correcto funcionamiento, esta forma de trabajo fue posible gracias al diagrama de flujo generado en la fase anterior.

Estructura del código de la aplicación

En esta parte diferenciaremos entre parte lógica y parte gráfica, en Android Studio tendremos la clase con toda la lógica, denominada Activity y su layout correspondiente, que será la parte gráfica de esa misma Activity.



Estructura del proyecto en Android Studio

- **Carpeta classes**: En esta carpeta están las clases que usaremos como objetos para crearlos luego en el flujo de la aplicación.
- **Carpeta recycler_view**: Esta carpeta contiene toda la lógica de los recycler views que son elementos de la interfaz gráfica que permiten mostrar una lista de items.
- **Activities:** Las clases que tienen toda la lógica funcional de la aplicación y que van linkeadas a sus interfaces gráficas correspondientes
- **Drawables:** Recursos gráficos de la aplicación, logo, iconos, etc.
- **Font**: En esta carpeta se almacenarán las fuentes importadas que usemos en el proyecto, en este caso al usar la fuente atlanta se crea un archivo con sus datos para poder usarla.

- **Layout**: En esta carpeta están todas las interfaces gráficas de las activities y de los recycler views.
- **Raw:** Carpeta que contiene el archivo de audio del final del temporizador.

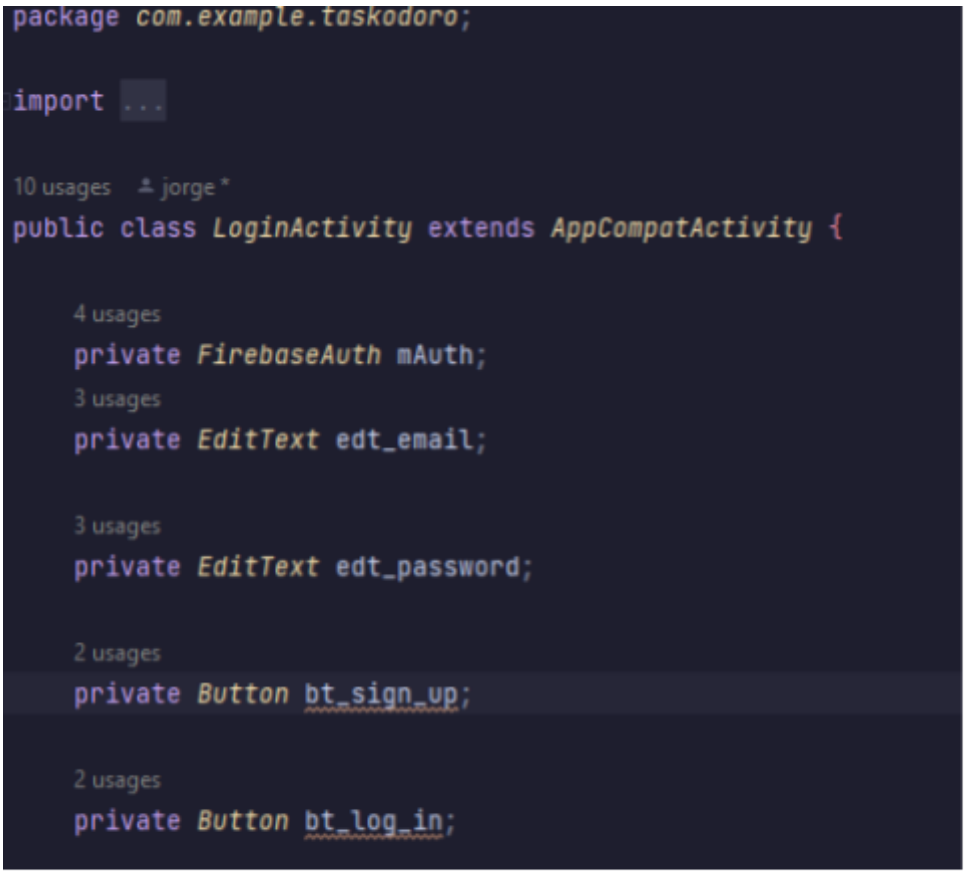
Funcionamiento del código

Después de haber dejado claro cómo funciona la app a nivel usuario vamos a ver como funciona a nivel de código.

Autenticacion de usuarios

Este apartado realizará la conexión con Firebase para la autenticación de usuarios con nuestra aplicación.

En la clase se definen unas variables privadas necesarias para realizar la conexión con el sistema de autenticación de Firebase



Variables privadas de la actividad de Autenticación

La variable mAuth es de tipo FirebaseAuth la cual nos permite conseguir una instancia de nuestro servicio de autenticación de Firebase con la cual podremos realizar varias operaciones relacionadas con este servicio.

También se crean las variables de los elementos de la interfaz gráfica.

Todas las actividades tiene un método onCreate(), el cual voy a explicar a continuación para no repetirlo por cada actividad, este método se ejecuta cuando la activity pasa a estar en pantalla del usuario, es decir cuando abrimos la aplicación se ejecuta este método el primero de todos, suele contener la conexión de los elementos de la view de la activity con la propia activity y también es donde obtendremos la instancia de Firebase, tanto de la base de datos como de la autenticación.

```
jorge
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    edt_email = (EditText) findViewById(R.id.edt_email);
    edt_password = (EditText) findViewById(R.id.edt_password);
    bt_log_in = (Button) findViewById(R.id.bt_log_in);
    bt_sign_up = (Button) findViewById(R.id.bt_sign_up);

    mAuth = FirebaseAuth.getInstance();
```

Método onCreate de la actividad de Autenticación

Aquí vemos cómo damos el valor de la instancia a la variable mAuth.

También generamos la conexión de los campos de texto de la interfaz gráfica y de los botones, para poder acceder a sus valores y métodos.

Para los botones crearemos los callbacks a ser invocados sobre los botones, es decir esperaremos un evento de clic sobre el botón y una vez realizado se ejecutará el bloque de código.

```
jorge
bt_log_in.setOnClickListener(new View.OnClickListener() {
    jorge
    @Override
    public void onClick(View view) {
        String email = String.valueOf(edt_email.getText()).trim();
        String password = String.valueOf(edt_password.getText()).trim();

        if (email.isEmpty() || password.isEmpty()) {
            Toast.makeText( context: LoginActivity.this,  text: "text fiels can't be empty", Toast.LENGTH_SHORT).show();
        } else {
            loginUser(email, password);
        }
    }
});

jorge
bt_sign_up.setOnClickListener(new View.OnClickListener() {
    jorge
    @Override
    public void onClick(View view) {
        String email = String.valueOf(edt_email.getText()).trim();
        String password = String.valueOf(edt_password.getText()).trim();

        if (email.isEmpty() || password.isEmpty()) {
            Toast.makeText( context: LoginActivity.this,  text: "text fiels can't be empty", Toast.LENGTH_SHORT).show();
        } else {
            signUpUser(email, password);
        }
    }
});
```

Métodos onClick de los botones de log in y sign up

Ambos bloques tienen las mismas validaciones para comprobar que no esten vacios los campos de texto y en caso de estar completos llamaran a sus respectivos métodos, uno para el registro y otro para el inicio de sesión.

```
public void loginUser(String email, String password) {
    mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                Toast.makeText( context: LoginActivity.this,  text: "login completed", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent( packageContext: LoginActivity.this, MenuActivity.class);
                startActivity(intent);
            } else {
                Toast.makeText( context: LoginActivity.this,  text: "error on login", Toast.LENGTH_SHORT).show();
            }
        }
    });
}

public void signUpUser(String email, String password) {
    mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                Toast.makeText( context: LoginActivity.this,  text: "user signed in", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent( packageContext: LoginActivity.this, MenuActivity.class);
                startActivity(intent);
            } else {
                Toast.makeText( context: LoginActivity.this,  text: "user couldn't be signed in", Toast.LENGTH_SHORT).show();
            }
        }

    });
```

Métodos para el inicio de sesión y registro de usuario

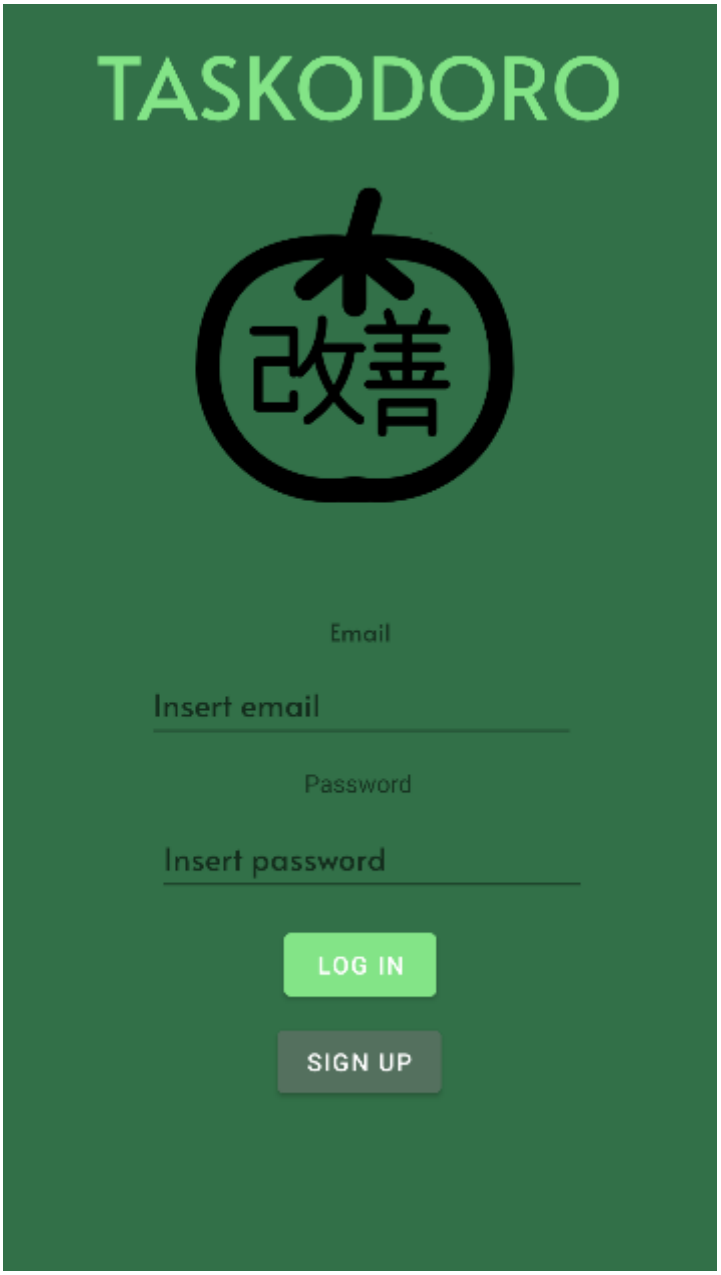
El método de loginUser() tiene como parámetros el email y la contraseña del usuario las cuales usará para llamar al método signInWithEmailAndPassword() de nuestra instancia de FirebaseAuth, se añade también el método onCompleteListener() que espera a que se complete la petición a Firebase y nos devolverá el resultado de la misma, en caso de completarse se recogerá el resultado en la variable task y se hará una comprobación para ver si la petición ha sido exitosa o ha habido un error, en caso de ser correcto se envía un mensaje de confirmación al usuario y se inicia la próxima activity, que será la MenuActivity. Para el método de signUpUser() el código es el mismo solo que en este caso el método usado de la instancia de FirebaseAuth es el de createUserWithEmailAndPassword(), para crear el usuario. Ambos métodos mostraron un error en caso de fallar en la petición a FirebaseAuth.

La interfaz gráfica de esta actividad cuenta con dos campos de texto, uno para el email y otro para la contraseña, ambos tienen su tipo especificado para adecuarse al contenido.

```
<EditText
    android:id="@+id/edt_email"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="7dp"
    android:layout_marginEnd="16dp"
    android:autofillHints="Insert email"
    android:ems="10"
    android:fontFamily="@font/alata"
    android:hint="Insert email"
    android:inputType="textEmailAddress"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.507"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/txt_email"
    tools:ignore="TextContrastCheck" />

<EditText
    android:id="@+id/edt_password"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="40dp"
    android:layout_marginEnd="16dp"
    android:autofillHints=""
    android:ems="10"
    android:fontFamily="@font/alata"
    android:hint="Insert password"
    android:inputType="textPassword"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.552"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/edt_email"
    tools:ignore="TextContrastCheck" />
```

Parámetros XML de los campos de texto de la actividad de autenticación

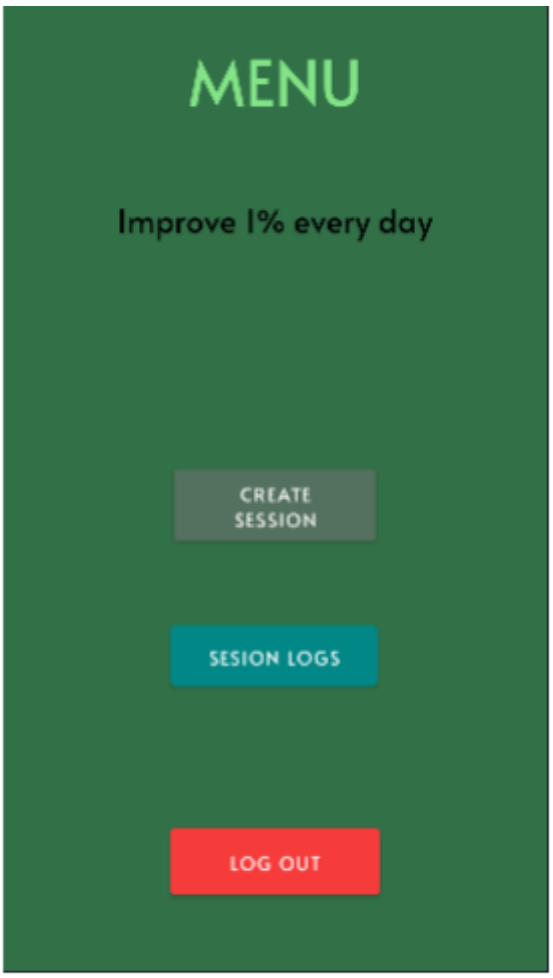
En esta parte se declaran todos los parámetros de los elementos de la interfaz, tamaño, color del texto, fuente, identificador, etc.



La interfaz también muestra el nombre de la aplicación y el logo de la misma. El logo fue creado juntando el concepto del pomodoro, nombre del método de temporizadores usado por la aplicación y también es el nombre del tomate en italiano y los kanjis o caracteres japonés de la palabra Kaizen. La gama de colores escogida para el proyecto va acorde a las necesidades definidas en la fase de requisitos, donde se definió el diseño de la aplicación como minimalista y amigable.

Menú de la aplicación

Para facilitar la navegación en la aplicación creamos un simple menú que nos permitirá movernos de forma clara por la interfaz gráfica.

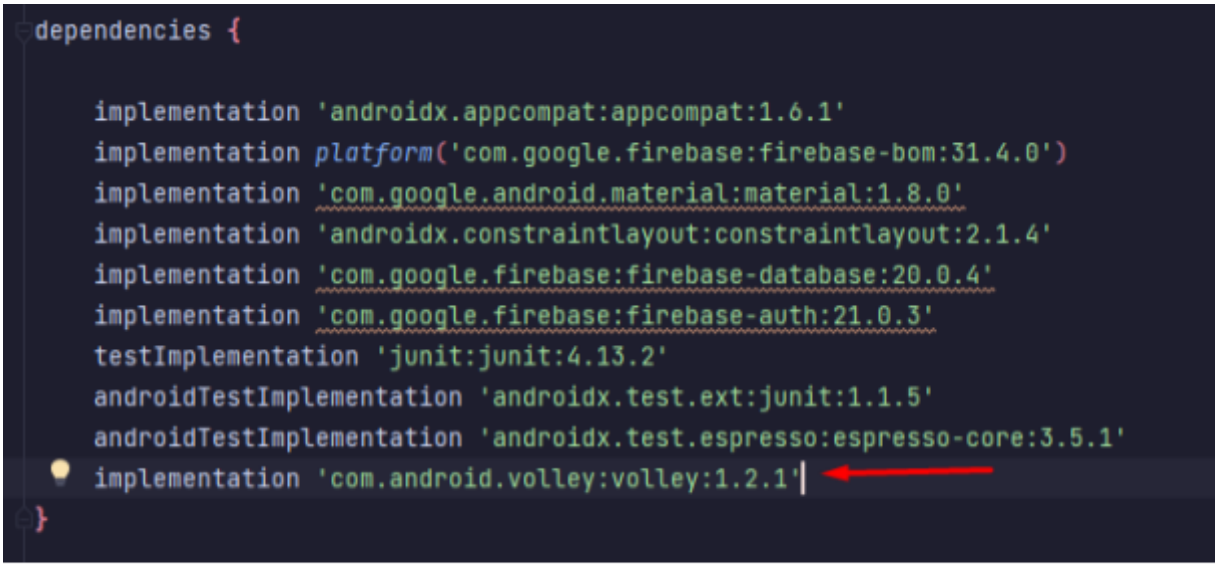


Interfaz gráfica del menú de la aplicación

En esta pantalla contaremos con dos botones para acceder a las funciones principales de la aplicación y un botón para cerrar la sesión, este botón nos devolverá a la pantalla anterior.

El texto de “Improve 1% every day” es un texto alternativo en caso de que haya un fallo al llamar a la API de frases motivadoras, en caso de éxito nos mostrará una frase aleatoria proporcionada por la respuesta de la petición a la API.

Para realizar llamadas HTTP en Android Studio usamos la librería Volley, esta librería tiene que ser importada antes de empezar a usarla, para ello la debemos requerir en nuestro archivo de dependencias de la aplicación:



Dependencias de la aplicación

Una vez requerido podemos usarlo en la clase que queramos creando un objeto de request. El constructor de este objeto necesita la url a la que vamos a realizar la petición, comúnmente conocida como endpoint; el cuerpo de la petición en caso de necesitarlo y un callback para la respuesta además de un callback para el caso de haber un error en la petición.

Variable privada con la url de la API

Bloque de código de la llamada a la API

Una vez creada la petición debemos establecer que vamos a hacer una se complete, en nuestro caso queremos recoger la frase que devuelve la llamada y cambiar el texto de la frase del menú a la que devuelva la API, en caso de error se dejará la frase alternativa como comentamos anteriormente.

Más abajo creamos una cola de peticiones, la cual será la que ejecute todas las peticiones que tengamos declaradas en la clase.

El resto de métodos de la clase son los necesarios para navegar al resto de pantallas de la aplicación y para cerrar sesión.

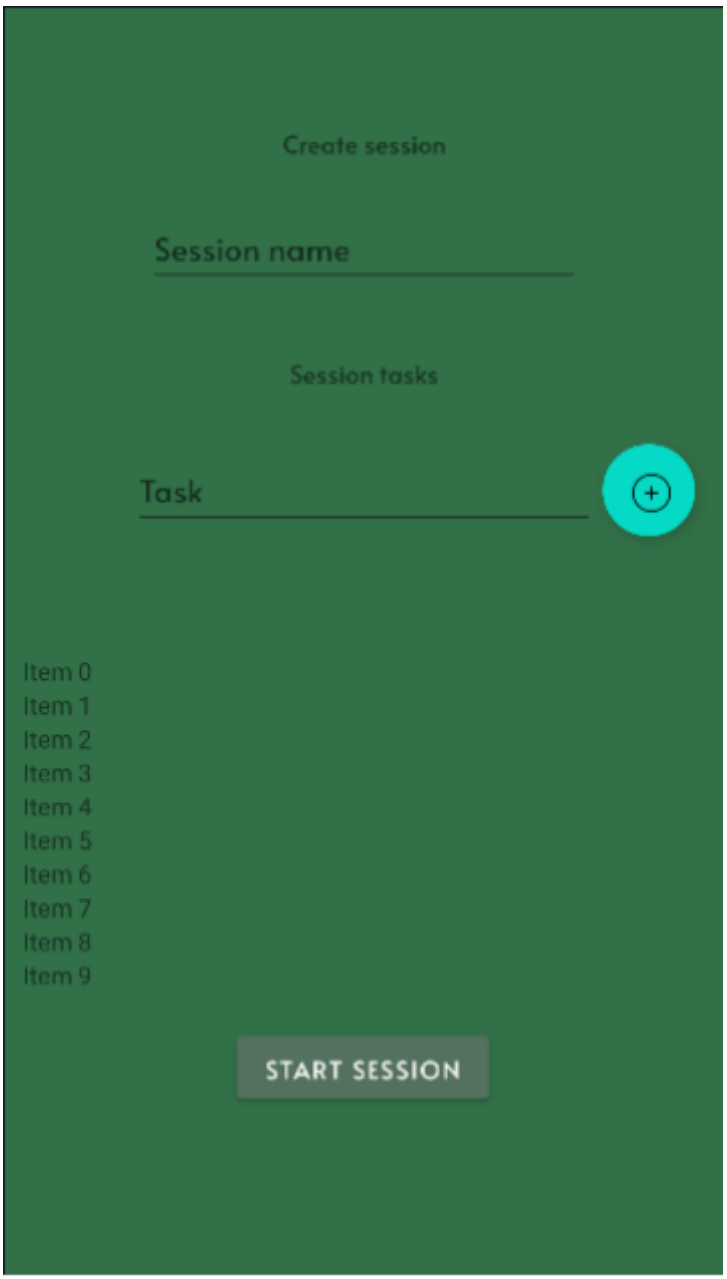
```
    bt_show_log.setOnClickListener(new View.OnClickListener() {
        new *
        @Override
        public void onClick(View view) {
            Intent intent = new Intent( packageContext: MenuActivity.this, ShowLogActivity.class);
            startActivity(intent);
        }
    });

    jorge *
    bt_create_session.setOnClickListener(new View.OnClickListener() {
        jorge *
        @Override
        public void onClick(View view) {
            Intent intent = new Intent( packageContext: MenuActivity.this, AddSessionActivity.class);
            startActivity(intent);
        }
    });

    jorge
    bt_log_out.setOnClickListener(new View.OnClickListener() {
        jorge
        @Override
        public void onClick(View view) { logOutuser(); }
    });

}

1 usage   jorge *
public void logOutuser() {
    FirebaseAuth.getInstance().signOut();
    Toast.makeText( context: MenuActivity.this,  text: "user signed out", Toast.LENGTH_SHORT).show();
    finish();
}
```

Métodos onClick del menú

En el botón de ver el historial indicamos que inicie la actividad ShowLogActivity, en el botón de cerrar sesión iniciara la actividad AddSessionActivity y en el botón de cerrar sesión ejecutará el método logOutUser(), el cual accede a la instancia de FirebaseAuth y ejecutará el método de cerrar sesión, mostrará un mensaje informando de ello y por último cerrará la pantalla y nos devolverá a la pantalla de autenticación de nuevo.

Crear una sesión pomodoro

Una vez elegido en el menú el crear sesión accederemos a la pantalla para establecer el nombre de la sesión y las tareas que deseemos, las cuales se mostrarán en forma de lista una vez añadidas.



Interfaz gráfica crear sesión

Esta parte de la aplicación es una de las más importantes pues es la que genera toda la estructura de datos que más tarde vamos a usar en Firebase, para ello partirá del nodo principal de la base de datos y creará un nodo padre de sesiones, este nodo luego creará una lista de sesiones para cada usuario. En esta parte es donde vamos a requerir los datos necesarios para crear un objeto sesión el cual tendrá todos los datos necesarios para generar los nodos en Firebase.

Lo primero es comprobar que sesiones tiene el usuario ya creadas para no permitir utilizarlas, de esta forma evitamos que se sobreescriban datos. Para ello asignamos a una lista de nombres el nombre de las sesiones ya existentes para ese usuario.

```
List<String> usedNames =  getSessionsName(currentUser);
```

Lista de nombres de sesiones

```
private List<String> getSessionsName(String currentUser){
    List<String> sessionsNames = new ArrayList<>();
    jorge
    myRef.child( pathString: "sessions").child(currentUser).addValueEventListener(new ValueEventListener() {
        2 usages  jorge
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.exists()){
                for(DataSnapshot ds: snapshot.getChildren()){
                    String sessionName = ds.child( path: "sessionName").getValue().toString();
                    sessionsNames.add(sessionName);
                }
            }
        }

        jorge
        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
    return sessionsNames;
}
```

Método getSessionsName()

En este método haremos una consulta a Firebase desde el nodo del identificador único del usuario actual y este nos devolverá una lista de datos de la cual vamos a extraer el nodo del nombre de la sesión para luego añadirlo a la lista que vamos a devolver al finalizar el método.

Una vez tenemos todos los nombres de sesiones podemos validar los datos que introduzca el usuario al hacer clic en el botón de crear la sesión.

```
bt_start_session.setOnClickListener(new View.OnClickListener() {
    jorge
    @Override
    public void onClick(View view) {
        String sessionName = String.valueOf(edt_session_name.getText());
        if(TextUtils.isEmpty(sessionName)){
            edt_session_name.setError("Session name can't be empty");
        } else if (usedNames.contains(sessionName)) {
            edt_session_name.setError("Session name already exists");
        } else{
            addSessionToDatabse(sessionName);
        }
    }
});
```

Metodo onClick() boton crear sesion

Primero cogeremos el valor del campo de texto del nombre de sesión introducido por el usuario y realizaremos las comprobaciones de que no esté vacío y de que la lista de nombre de sesiones ya utilizados no contenga el nombre introducido, una vez pasadas las validaciones se ejecutará el método para crear la sesión en la base de datos.

```
private void addSessionToDatabse(String sessionName) {
    Session newSession = new Session(sessionName,taskMap);
    myRef.child( pathString: "sessions").child(currentUser).child( pathString: "Session " + newSession.getSessionName()).setValue(newSession);

    Intent intent = new Intent( packageContext: AddSessionActivity.this, MainActivity.class);
    intent.putExtra( name: "sessionName",  value: "Session " + newSession.getSessionName());
    startActivity(intent);
}
```

Método addSessionToDatabase()

Este método crea un objeto de tipo sesión a partir del nombre de la sesión y un mapa clave valor de las tareas, este mapa se rellena mediante el método onClick() del botón para añadir tareas que se explica más adelante.

Una vez tengamos el objeto sesión crearemos un nodo con el nombre de la sesión y el valor será el objeto sesión al completo. Luego iniciaremos la siguiente actividad, la actividad MainActivity que será la sesión pomodoro que acabamos de crear, además pasaremos el valor del nombre de la sesión para que sirva de título en la pantalla de la actividad principal.

```
bt_add_task.setOnClickListener(new View.OnClickListener() {
    jorge
    @Override
    public void onClick(View view) {
        String taskName = String.valueOf(edt_task_name.getText());
        if(TextUtils.isEmpty(taskName)){
            edt_task_name.setError("Task name can't be empty");
        }
        else{
            Task newTask = createTask(taskName);
            taskMap.put(newTask.getTaskName(), newTask);
            tasks.add(newTask);
            taskAdapter.notifyDataSetChanged();
            edt_task_name.setText("");
        }
    }
});
```

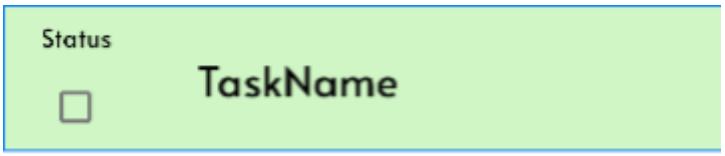
Método onClick() del botón añadir tarea

Al hacer clic en el botón de añadir se validará el campo de texto para comprobar que no está vacío, una vez se ingrese texto se crea un objeto de tipo tarea mediante el método createTask().

```
private Task createTask(String taskName) { return new Task(taskName, taskStatus: false); }
```

Método createTask()

Una vez creado el objeto tarea lo añadiremos al mapa de clave valor, siendo el nombre de la tarea la clave y el valor el objeto tarea en sí mismo, esto nos permite identificar un nombre de tarea con sus atributos. También se añadirá a una lista la tarea para alimentar el recycler view y mostrar las tareas que vamos añadiendo mediante un observador de eventos del tipo cambio de datos.



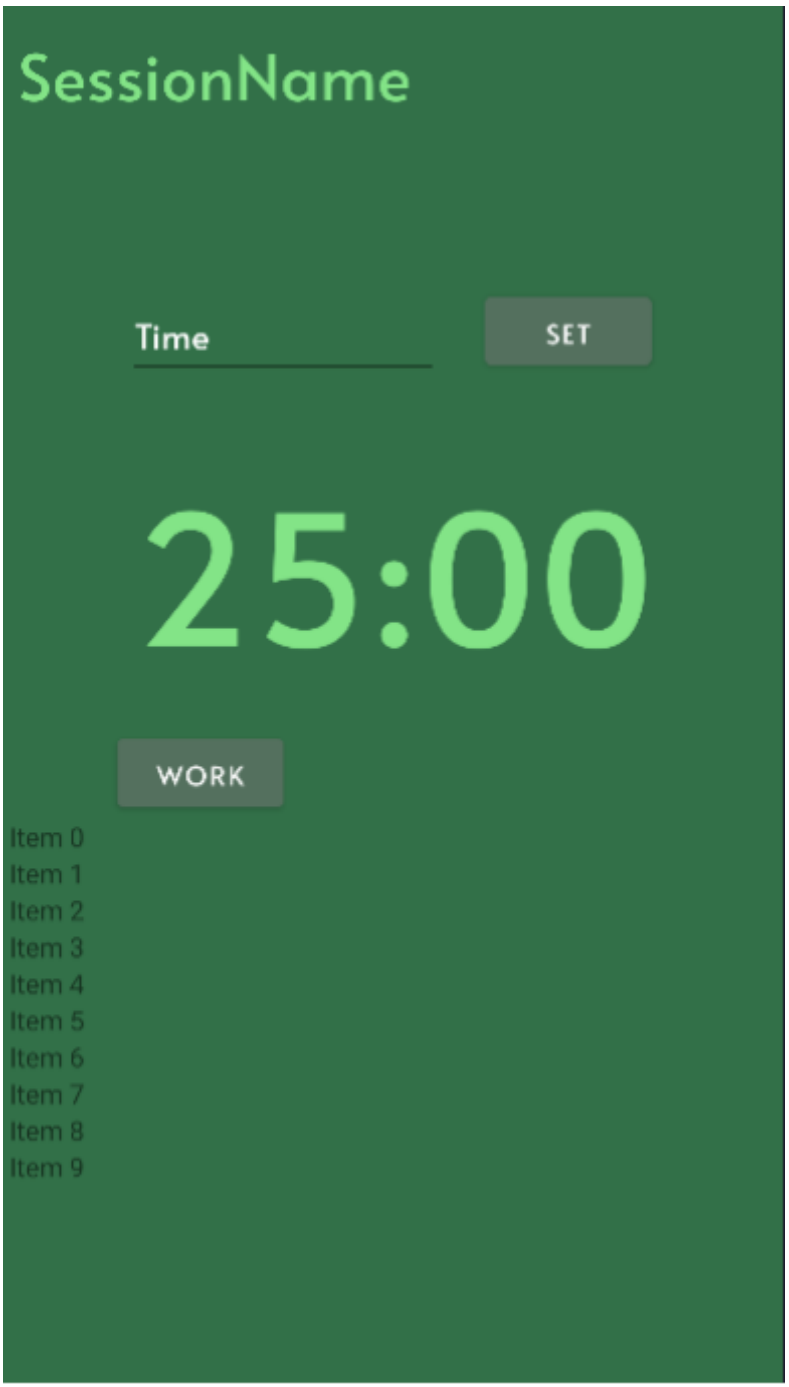
Interfaz gráfica de la lista de tareas

La parte gráfica del recycler view se denomina viewHolder, es decir donde se va almacenar la parte visual de la lista de items y luego la conectaremos a un adaptador que se encargará de cargar esta interfaz con los datos especificados.

Una vez completada la ejecución de esta actividad pasaremos a la siguiente, la cual es la sesión pomodoro.

Sesión pomodoro

Esta clase es la más importante ya que carga con la principal función del proyecto, el temporizador pomodoro y la lista de tareas.



Interfaz gráfica temporizador pomodoro

En esta actividad contaremos con el nombre de la sesión, el campo de texto para personalizar el temporizador, el tiempo del temporizador, el botón para iniciar y pausar el temporizador y la lista de tareas.

En cuanto a la lista de tareas el uso y creación es igual al del apartado anterior con la diferencia de que ahora la lista de tareas se recogerá desde Firebase.

Método getTasksFromSession()

Como ya vimos anteriormente realizaremos una consulta a la base de datos sobre el nodo de las tareas asignadas a esa sesión creada por el usuario y recogeremos los atributos del nodo tarea y añadimos un nuevo objeto tarea a nuestra lista de tareas creada anteriormente, por último le pasaremos al adaptador del recycler view la lista de tareas a mostrar y el viewHolder de la misma para especificar la interfaz gráfica a usar.

El siguiente bloque de código del onCreate() de la aplicación es el cual contiene los métodos onClick() del botón para cambiar el tiempo y el botón para el temporizador.

Para ello tenemos varias variables, usadas para trabajar con el tiempo introducido por el usuario, dos de ellas constantes usadas para almacenar las cadenas de texto de tiempo por defecto.

Variables usadas para asignar el tiempo del temporizador

La variable pattern almacena la expresión regular que vamos a usar para sacar los minutos y segundos introducidos por el usuario y la variable r es de tipo Pattern que será la que compile la expresión regular y nos permita pasarla al matcher que creamos en el método onClick().

```
Matcher m = r.matcher(DEFAULT_WORK_TIME);

if (m.find()) {
    txt_counter_text.setText(DEFAULT_WORK_TIME);
    timeLeftInMilliseconds = Long.parseLong(m.group(1)) * 60000;
}
 jorge *
button_set_time.setOnClickListener(new View.OnClickListener() {
     jorge *
    @Override
    public void onClick(View view) {
        custom_time = String.valueOf(editText_custom_time.getText());
        Matcher m = r.matcher(custom_time);

        if (m.find() && !custom_time.equalsIgnoreCase( anotherString: "00:00")) {
            txt_counter_text.setText(custom_time);
            timeLeftInMilliseconds = Long.parseLong(m.group(1)) * 60000; // with the rege
            secondsLeftInMilliseconds = Long.parseLong(m.group(2)) * 1000; // with the re
            timeLeftInMilliseconds = timeLeftInMilliseconds + secondsLeftInMilliseconds;
            messageToast("Timer value changed");
        } else {
            editText_custom_time.setError("Format has to be: (xx:xx) ");
        }
    }
});
```

Método onClick() para personalizar el tiempo

En esta parte vamos a especificar de cuánto será el temporizador y para ello usaremos expresiones regulares que hemos creado junto a un matcher, la clase matcher necesita de un patrón que analizar y una cadena de texto sobre la que hacer la comparación.

El primer matcher es para establecer el tiempo estándar por eso se usa la constante DEFAULT_WORK_TIME que es de 25 minutos, luego se realiza una comprobación para que se establezca el tiempo si el matcher valida la cadena de texto, se cambia el texto del temporizador y luego pasamos el tiempo a milisegundos para poder usarlo en la cuenta atrás.

En caso de introducir un tiempo personalizado debemos hacer clic sobre el botón para cambiar el tiempo y para ello haremos un matcher que actuará sobre ese campo de texto en vez de usar el tiempo por defecto, esta vez al poder introducir tanto como minutos y

segundos deberemos hacer la suma de ambas conversiones para obtener el total en milisegundos, mostraremos un mensaje al usuario para informar de la acción y cambiaremos el texto del temporizador, en caso de error de sintaxis se mostrará un mensaje indicando la sintaxis correcta.

El método onClick() del botón del contador ejecutará el método principal del contador el cual manejara el hilo de ejecución del mismo.

```
button_start_work.setOnClickListener(new View.OnClickListener() {
    jorge
    @Override
    public void onClick(View view) { startStop(); }
});
```

Método onClick() botón contador

```
public void startStop() {
    if (timerRunning) {
        stopTimer();
        instantEnd = Instant.now();
        long timeElapsed = Duration.between(instantStart, instantEnd).toMillis();
        long minutes = TimeUnit.MILLISECONDS.toMinutes(timeElapsed);
        timeElapsed -= TimeUnit.MINUTES.toMillis(minutes);
        long seconds = TimeUnit.MILLISECONDS.toSeconds(timeElapsed);
        minutesSpentArray.add(minutes);
        secondsSpentArray.add(seconds);

        button_set_time.setVisibility(View.VISIBLE);
        editText_custom_time.setVisibility(View.VISIBLE);
    } else {
        startTimer();
        instantStart = Instant.now();
        button_set_time.setVisibility(View.INVISIBLE);
        editText_custom_time.setVisibility(View.INVISIBLE);
    }
}
```

Método startStop()

El método ejecuta un bloque o otro depende de si el temporizador está en ejecución o no.

En caso de no estar en ejecución, se ejecutará el método startTimer() el cual iniciará el temporizador, a la vez también se recogerá en una variable de la clase instant el momento en el cual se a iniciado el temporizador, esto nos servirá para poder calcular el tiempo usado en la sesión una vez la acabemos.

En caso de estar en ejecución, se ejecutará el método stopTimer() que parara la ejecución del temporizador, también recogeremos el momento en el que se para el temporizador y calcularemos la diferencia de tiempo frente a cuando empezó el temporizador y así poder almacenar ese tiempo usado en la sesión, que al final de la misma se sumará para tener el tiempo total empleado en la sesión.

```
private void stopTimer() {
    countDownTimer.cancel();
    if(isABreak) {
        button_start_work.setText("BREAK");
    }
    else {
        button_start_work.setText("WORK");
    }
    timerRunning = false;
}
```

Método stopTimer()

El método stopTimer() usara el método cancel del objeto countDownTimer lo cual hará que el contador que estuviera en ejecución termine, también se maneja el flujo de fases de trabajo y descanso, en caso de ser un descanso se cambiara el texto a “BREAK” y en caso de ser fase de trabajo el texto será “WORK”, pòr ultimo se cambia el estado de timerRunning a false pues acabamos de parar el temporizador.

```
private void startTimer() {
    jorge
    countDownTimer = new CountDownTimer(timeLeftInMilliseconds,  countDownInterval: 1000) {
        jorge
        @Override
        public void onTick(long l) {
            timeLeftInMilliseconds = l;
            updateTimer();
        }

        jorge
        @Override
        public void onFinish() {...}
    }.start();

    button_start_work.setText("STOP");
    timerRunning = true;
}
```

Método startTimer()

El metodo startTimer() crea un objeto de la clase CounDownTimer el cual como su nombre indica es una cuenta atrás, este objeto se construye con el tiempo total de la cuenta atrás en milisegundos y el intervalo de la cuenta atrás en milisegundos, en este caso es 1000 porque es un segundo en milisegundos. El método startTimer() siempre ejecuta el método start() para iniciar el contador y cambia el texto del botón a “STOP” para aclarar que el contador esta activo, ademas de cambiar el estado de timerRunning a true pues acabamos de iniciar el contador.

Esta clase nos obliga a implementar los métodos onTick() que establece que se hará con cada intervalo, es decir cada vez que el contador baje un segundo y el método onFinish() que establece qué hacer cuando la cuenta atrás llegue a su fin.

Primero veremos el método onTick(), el cual recoge el tiempo después de cada intervalo y se lo asignamos al tiempo restante, es decir va restando 1 segundo al contador cada vez y actualizar ese valor, tanto a nivel de la variable timeLeftInmMilliseconds como a nivel de la interfaz gráfica con el método updateTimer().

```java
private void updateTimer() {
    int minutes = (int) timeLeftInMilliseconds / 60000;
    int seconds = (int) timeLeftInMilliseconds % 60000 / 1000;

    String timeLeftText;

    timeLeftText = "" + minutes;
    timeLeftText += ":";
    if (seconds < 10) timeLeftText += "0";
    timeLeftText += seconds;

    txt_counter_text.setText(timeLeftText);
}
```

Método updateTimer()

Primero sacamos el valor de los minutos y segundos y luego vamos a crear una string a la cual añadiremos los minutos, los dos puntos divisores y luego los segundos especificando que en caso de ser menor a 10 añadiremos un 0 antes del número para evitar que se quede el numero solo, por último asignamos esta cadena de texto al texto del temporizador.

Dentro del método onFinish() tenemos varios bloques de código, el primero se encarga de notificar al usuario la finalización del contador.

```java
@Override
public void onFinish() {
    if(mediaPlayer.isPlaying()){
        mediaPlayer.stop();
        mediaPlayer.release();
        mediaPlayer = MediaPlayer.create(getApplicationContext(), R.raw.timer_end_sound);
    }
    mediaPlayer.start();

    vibrator.vibrate(VibrationEffect.createWaveform(vibrationPattern,  repeat: 0));
```

Primer bloque de código del método onFinish()

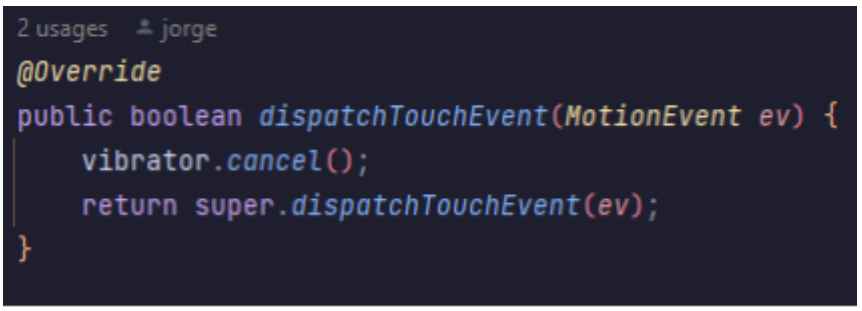
```java
private long[] vibrationPattern = {500, 550, 500, 550};
vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
mediaPlayer = MediaPlayer.create( context: this, R.raw.timer_end_sound);
```

Recursos para notificar al usuario la finalización del contador

A la hora de finalizar el contador haremos vibrar el teléfono con un patrón y también emitirá un sonido, para ello debemos usar la clase MediaPlayer y la clase Vibrator.

La clase MediaPlayer nos permite reproducir audio o video dentro de la aplicación, para ello se requiere del archivo a reproducir, en este caso el timer_end_sound almacenado en la carpeta raw. En el método onFinish() comprobamos que no esté reproduciendo el archivo para no solapar sonidos, si está sonando paramos la reproducción y con el método release() libera la memoria usada por el mediaPlayer y así poder crear una nueva instancia la cual será la que se use la siguiente vez que acabe el temporizador, de esta manera manejamos el flujo de vida de mediaPlayer. En caso de no estar reproduciendo nada iniciaremos la reproducción.

La clase Vibrator nos permite manejar el módulo de vibración del dispositivo y junto la clase VibrationEffect podremos crear patrones de vibración para generar alertas. En este caso creamos un patrón de vibraciones cortas con la lista de vibrationPattern la cual almacena los valores de el tiempo de cada vibración en milisegundos, al usar el método createWaveform() y pasarle nuestra lista de intervalos junto a él numero de veces a repetir, en caso de ser 0 se repite continuamente hasta que se cancele, para cancelarlo sobreescribimos el evento de tocar la pantalla, de forma que cuando el usuario toque la pantalla se accione esta acción.



Sobreescribir el evento de tocar la pantalla

Dentro decimos que una vez se toque la pantalla se cancele la vibración.

Una vez se ha tonificado al usuario que el temporizador ha finalizado hay que almacenar el instante en el que acaba la sesión para poder realizar el conteo del tiempo empleado una vez acabe la sesión.

```
instantEnd = Instant.now();
long timeElapsed = Duration.between(instantStart, instantEnd).toMillis();
long minutes = TimeUnit.MILLISECONDS.toMinutes(timeElapsed);
timeElapsed -= TimeUnit.MINUTES.toMillis(minutes);
long seconds = TimeUnit.MILLISECONDS.toSeconds(timeElapsed);
minutesSpentArray.add(minutes);
secondsSpentArray.add(seconds);
```

Segundo bloque de código del método onFinish()

Como ya vimos antes aquí recogemos el instante en el que acaba el temporizador y calculamos la diferencia desde que empezó, el resultado será el tiempo empleado en esta vuelta del contador y lo almacenaremos separado en minutos y segundos en una lista.

Ahora queda manejar el flujo de la fase de trabajo y descanso, para poder reiniciar una vez acabe y que el usuario no tenga que hacerlo de forma manual.

```
if(!isABreak){
    Matcher m = r.matcher(DEFAULT_BREAK_TIME);

    if (m.find()) {
        txt_counter_text.setText(DEFAULT_BREAK_TIME);
        timeLeftInMilliseconds = Long.parseLong(m.group(1)) * 60000;
        button_start_work.setText("BREAK");
        isABreak = true;
        countDownTimer.cancel();
        button_start_work.performClick();
    }

}
else
{
    Matcher m = r.matcher(DEFAULT_WORK_TIME);

    if (m.find()) {
        txt_counter_text.setText(DEFAULT_WORK_TIME);
        timeLeftInMilliseconds = Long.parseLong(m.group(1)) * 60000;
        button_start_work.setText("WORK");
        isABreak = false;
        countDownTimer.cancel();
        button_start_work.performClick();
    }
}
```

Tercer bloque de código del método onFinish()

Para saber si es fase de trabajo o descanso usamos la variable isABreak y la negamos, es decir en caso de que sea un descanso nos dará falso y en caso de no serlo nos dará verdadero. Ambas partes de la condición hacen lo mismo solo que cambiarán el tiempo del temporizador, el texto del botón y también cambiarán el estado de la variable isABreak, cancelará el temporizador actual e hará clic en el botón del contador para empezar una nueva ejecución del contador.

La parte final de esta actividad es la de calcular el tiempo empleado en la sesión que hemos ido viendo a lo largo de su ejecución, el total de este tiempo se calcula una vez la actividad se ha parado de ejecutar, para ello usamos el método onStop() para una vez la actividad llegue al apartado de onStop en el ciclo de vida ejecute el código.

```
@Override
protected void onStop() { // to save the spent time when user stops the activity execution
    super.onStop();
    long totalMinutesTimeSpent = 0;
    long totalSecondsTimeSpent = 0;
    for (long minutesTimeValue : minutesSpentArray) {
        totalMinutesTimeSpent += minutesTimeValue;
    }
    for (long secondsTimeValue : secondsSpentArray) {
        totalSecondsTimeSpent += secondsTimeValue;
    }
    long totalMinutesSpent = totalMinutesTimeSpent - totalMinutesTimeSpent/2; //this is because when resetting the timer we perform
    long totalSecondsSpent = totalSecondsTimeSpent - totalSecondsTimeSpent/2;
    String timeSpentToString = String.format("%02d:%02d", totalMinutesSpent, totalSecondsSpent);
    myRef.child( pathString: "sessions").child(currentUser).child(sessionName).child( pathString: "timeSpend").setValue(timeSpentToString);
}
```

Método onStop() de la activity MainActivity

Ya que vamos a calcular el total del tiempo creamos dos variables del tiempo total, una para minutos y otra para segundos, luego recorremos la lista de los minutos y los segundos usados a lo largo de la ejecución de la actividad y los almacenamos en el total en forma de suma de todos los valores, debido a que para reiniciar el contador pulsamos el botón el tiempo se duplica al coger el instante dos veces, por eso dividimos entre 2 para conseguir el tiempo real y quitar la duplicidad, por último vamos a pasar el resultado total a una cadena de texto con el formato de tiempo deseado y ese tiempo lo actualizaremos en el nodo timeSpend de la sesión en Firebase.

Historial de sesiones

Una vez hemos realizado al menos una sesión podremos verla desde el historial de sesiones, tendremos el nombre de la sesión y el tiempo que hemos empleado en la sesión.

Interfaz gráfica historial de sesiones

Esta interfaz incluye únicamente un recycler view que contendrá las sesiones disponibles en la base de datos de ese usuario.

```
jorge *
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_show_log);

    rv_sessions = (RecyclerView) findViewById(R.id.rv_sessions);

    database = FirebaseDatabase.getInstance();
    myRef = database.getReference();
    String currentUser = FirebaseAuth.getInstance().getCurrentUser().getUid();

    RecyclerView.LayoutManager layoutManager = new LinearLayoutManager( context: this);
    rv_sessions.setLayoutManager(layoutManager);
    getSessionsFromDatabase(currentUser);

    DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(rv_sessions.getContext(),
            ((LinearLayoutManager) layoutManager).getOrientation());
    rv_sessions.addItemDecoration(dividerItemDecoration);
}
```

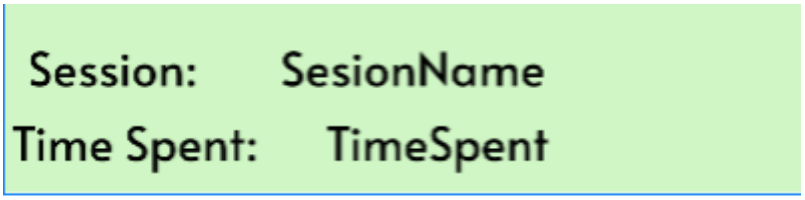
Método onCreate() actividad historial de sesiones

Para recuperar las sesiones del usuario necesitamos su identificador único y la instancia de Firebase, las asignamos a variables y vamos a ejecutar el método getSessionsFromDatabase(), que alimentará el recycler view con la lista de sesiones.

```
private void getSessionsFromDatabase(String currentUser){
    jorge
    myRef.child( pathString: "sessions").child(currentUser).addValueEventListener(new ValueEventListener() {
        2 usages  jorge
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.exists()) {
                for (DataSnapshot ds : snapshot.getChildren()) {
                    String sessionName = ds.child( path: "sessionName").getValue().toString();
                    String timeSpend = ds.child( path: "timeSpend").getValue().toString();
                    sessions.add(new Session(sessionName, timeSpend));
                }
                sessionAdapter = new SessionAdapter(sessions, R.layout.sesion_view);
                rv_sessions.setAdapter(sessionAdapter);
            }
        }

        jorge
        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    });
}
```

Método getSessionsFromDatabase() actividad historial de sesiones

Realizaremos una consulta al nodo del identificador único del usuario del cual son hijos los nodos de las sesiones, crearemos un objeto sesion el cual añadiremos a la lista de sesiones y se lo pasaremos al adaptador del recycler view para que conecte los datos con el viewHolder mostrando en pantalla el nombre de la sesión y el tiempo usado.



Interfaz gráfica lista de sesiones

Con todo esto termina la fase de desarrollo de la aplicación la cuál fue complementada con su fase de pruebas posterior en la cual se probaron todas las funciones en un dispositivo de pruebas y con un grupo de usuarios de testeo para comprobar que la aplicación cumplía satisfactoriamente los requisitos.

Manual de uso de la aplicación

El diseño de la aplicación garantiza poder navegar por ella de forma simple y cómoda para todo tipo de usuarios. A continuación se adjunta el enlace al manual de uso de la aplicación para poder ayudar a los usuarios a familiarizarse con la interfaz.

Manual Taskodoro

Conclusiones

En el desarrollo de este proyecto de aplicación móvil, se logró diseñar, implementar y entregar una solución funcional y eficiente que permite a los usuarios mejorar su productividad y gestionar su tiempo de manera efectiva. A lo largo de este trabajo, se han alcanzado los siguientes objetivos:

- Diseño intuitivo y atractivo: Se ha logrado crear una interfaz de usuario atractiva y fácil de usar, que brinda una experiencia agradable al usuario. El diseño intuitivo permite a los usuarios navegar y utilizar las diferentes funcionalidades de la aplicación de manera sencilla, sin necesidad de conocimientos técnicos previos.
- Funcionalidades completas: La aplicación desarrollada ofrece todas las funcionalidades necesarias para implementar la técnica Pomodoro de manera efectiva. Los temporizadores de trabajo y descanso están disponibles, con opciones para personalizar la duración de cada uno. Además, se incluyen la lista de tareas y el historial de sesiones para analizar el uso de nuestro tiempo.
- Rendimiento y eficiencia: Durante las pruebas realizadas, la aplicación ha demostrado el rendimiento esperado y un uso de recursos reducido. Esto asegura una experiencia fluida y correcta para los usuarios de la aplicación.
- Documentación completa: Se ha elaborado una documentación exhaustiva que describe todos los aspectos del desarrollo de la aplicación, incluyendo el análisis de requisitos, diseño de la arquitectura, implementación, etc. Esta documentación servirá como referencia para futuros desarrollos o mejoras en la aplicación, así como para facilitar el mantenimiento.

En conclusión, este proyecto ha logrado desarrollar una aplicación móvil de temporizadores Pomodoro que cumple con todos los objetivos establecidos. La solución implementada brinda a los usuarios una herramienta efectiva para mejorar su productividad y gestión del tiempo. Además, se ha realizado un trabajo exhaustivo en términos de diseño, funcionalidad, rendimiento y documentación. Se espera que esta aplicación sea útil para una amplia gama de usuarios, tanto en entornos académicos como profesionales, y que pueda adaptarse y expandirse en el futuro para ofrecer aún más funcionalidades y características innovadoras.

Bibliografía

Pagina documentacion de Android Studio usada durante el proyecto: https://developer.android.com/studio

Wikipedia de tecnica pomodoro ejecutada en el proyecto: https://es.wikipedia.org/wiki/T%C3%A9cnica_Pomodoro

Wikipedia filosofía Kaizen en la cual se basa la idea del proyecto: https://es.wikipedia.org/wiki/Kaizen

Excalidraw, programa para crear pizarras, fue usado para desarrollar el prototipo de la interfaz grafica de la aplicacion: https://excalidraw.com/

Pagina web de estadísticas globales : https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/#monthly-202205-202305-bar

API gratuita con frases motivadoras usada en la pantalla del menu de la aplicación: https://api.goprogram.ai/inspiration/docs/

Fuente usada en la aplicación: https://fonts.google.com/specimen/Alata

Libreria para realizar peticiones HTTP en Android: https://google.github.io/volley/