

# SHARE MY BIKE

Alfonso Rincón Cuerva

2º DAM

Programación Multimedia y de Dispositivos Móviles

## ÍNDICE

HERRAMIENTAS Y TECNOLOGÍAS .....	1
PROBLEMA PLANTEADO .....	2
SOLUCIÓN AL PROBLEMA .....	2
FUNCIONES UTILIZADAS .....	3
PRUEBAS REALIZADAS .....	7
ESTRUCTURA DE LA APLICACIÓN.....	8
EJECUCIÓN DE LA APLICACIÓN .....	8
COSAS QUE HE APRENDIDO .....	9
BIBLIOGRAFÍA.....	10

## HERRAMIENTAS Y TECNOLOGÍAS VERSIONES

En el archivo **libs.versions.toml** encontramos versiones de diferentes bibliotecas y plugins del proyecto.

- **agp**: 8.5.2.
- **activity**: 1.9.3.
- **navigationFragment**: 2.6.0.

```
2 agp = "8.5.2"
3 junit = "4.13.2"
4 junitVersion = "1.2.1"
5 espressoCore = "3.6.1"
6 appcompat = "1.7.0"
7 material = "1.12.0"
8 activity = "1.9.3"
9 constraintlayout = "2.2.0"
10 playServicesLocation = "21.3.0"
11 navigationFragment = "2.6.0"
12 navigationUi = "2.6.0"
```

La versión de Gradle es de 8.7.

```
gradle-wrapper.properties
1 #Sat Nov 23 13:14:22 CET 2024
2 distributionBase=GRADLE_USER_HOME
3 distributionPath=wrapper/dists
4 distributionUrl=https\://services.gradle.org/distributions/gradle-8.7-bin.zip
5 zipStoreBase=GRADLE_USER_HOME
6 zipStorePath=wrapper/dists
7
```

En el archivo **build.gradle.kts**, podremos ver lo siguiente:

- **minSdk**: en esta aplicación es de 21. Esto significa que la versión de API mínima que soporta la app es la 21.
- **targetSdk**: es la versión de SDK para la cual está optimizada la app. En esta app, es de 34.

```
defaultConfig {
    applicationId = "edu.pruebas.sharemybike"
    minSdk = 21
    targetSdk = 34
    versionCode = 1
    versionName = "1.0"
```

## EMULADORES

- **Pixel API 30**: 5,0" 1080x1920 420dpi. **System image**: R (API 30)
- **Pixel 5 API 30**: 6,0" 1080x2340 440dpi. **System image**: R (API 30)
- **Pixel 6 API 30**: 6,4" 1080x2400 420dpi **System image**: R (API 30)

## PLATAFORMA

El programa fue creado en la plataforma **Android Studio**, que es un entorno de desarrollo integrado oficial para crear aplicaciones Android. Incluye características como editor de código, diseñador de interfaces, emulador de Android y sistema de compilación Gradle.

## LENGUAJES

Los lenguajes utilizados para la creación de esta aplicación son los siguientes:

- **Java**: para la lógica de la aplicación e interactuar con los usuarios.
- **XML**: para agregar características y estilos a la aplicación.
- **Gradle**: es el sistema de compilación que utiliza Android.

## PROBLEMA PLANTEADO

Se nos pide crear una aplicación para que los viajeros que vayan a una ciudad puedan utilizar bicicletas prestadas de forma gratuita. Uno de los objetivos es ayudar a la sostenibilidad incentivando que los viajeros utilicen para sus desplazamientos cortos la bicicleta, en lugar de utilizar otros medios de locomoción como el taxi, autobuses o vehículos de alquiler.

La aplicación estará formada por una pantalla principal, en la cual el usuario tendrá que mostrar su ubicación y aportar un email, para así poder recibir información de las bicis de su zona. Una vez agregado, elegirá la fecha en la cual quiere utilizar la bici, y recibirá una lista con todas las bicis disponibles.

## SOLUCIÓN AL PROBLEMA



Para poder crear esta aplicación, primero he creado una Actividad que funcionará como el menú principal, donde el usuario podrá indicar su ubicación y email. En esta tendremos también el logo de la aplicación. Una vez insertados los datos, podremos pulsar en botón Login para poder ir a la segunda Actividad.

La segunda Actividad es un **Basic Views Activity**, la cual creará 2 fragment. El primero servirá para mostrar el calendario con la fecha en la cual el usuario querrá utilizar la bici. El segundo Fragment nos servirá para mostrar todas las bicis disponibles en un RecyclerView.





Cada bici tendrá un botón de email, el cual el usuario podrá pulsar para poder contactar con el proveedor de la bici.

## FUNCIONES UTILIZADAS

**ICONO:** en el archivo AndroidManifest.xml, asignaré un nuevo icono para la aplicación. Este consiste en un diseño de una bici.

```
12  android:icon="@mipmap/ic_launcher_round"
13 android:label="@string/ShareMyBike"
14  android:roundIcon="@mipmap/ic_launcher_round"
```

**VARIABLES PARA COLORES:** en la carpeta **values** encontramos el archivo **colors.xml**. En este creo variables para todos los colores que usaré para cambiar el color de los textos y botones.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3  <color name="black">#FF000000</color>
4  <color name="white">#FFFFFFFF</color>
5  <color name="rojoAnaranjado">#C83100</color>
6  <color name="naranjaClaro">#FF5900</color>
7 </resources>
```

**VARIABLES PARA STRINGS:** tendrá diferentes textos en inglés, principalmente los de la MainActivity y los de los Fragments.

```
1 <resources>
2 <string name="app_name">ShareMyBike</string>
3 <!-- Strings used for fragments for navigation -->
4 <string name="first_fragment_label">Select Date</string>
5 <string name="second_fragment_label">List of available bikes</string>
6 <string name="next">SELECT DATE</string>
7 <string name="previous">Previous</string>
8
9 <!--Strings de la MainActivity-->
10 <string name="ubi_denegada">Location permission denied</string>
11 <string name="obtener_ubicacion">You must obtain the location first</string>
12 <string name="ubi_error">Unable to get the location</string>
13
14 <string name="direccion_postal_error">Error retrieving the address</string>
15 <string name="direccion_postal_noDisp">Postal address not available</string>
16
17 <string name="agregar_email">You must add an email</string>
18 <string name="email_no_valido">Invalid email</string>
19 </resources>
```

**ACTIVIDAD PRINCIPAL:** esta actividad, a parte de contener el logo de la aplicación, contiene 2 funcionalidades para poder entrar correctamente en la aplicación.

- **EMAIL:** contiene un método para poder verificar que el email sea un email realmente. Para ello, se utiliza la clase que nos ofrece Android Studio **Patterns**.

```
// Método para verificar si el correo obtenido es un email o no
private boolean esEmail(String email) { 1 usage 1 alfonsoaco
    return Patterns.EMAIL_ADDRESS.matcher(email).matches();
}
```

- **UBICACIÓN:** esta es mucho más compleja que la anterior. Contiene diferentes métodos para verificar que el usuario haya especificado su ubicación:
  - **pedirPermisosDeUbicacion:** tras haber agregado los permisos necesarios al *manifest.xml*, se lanza esta función, la cual lanzará un Launcher para poder verificar si el usuario ha aceptado o ha rechazado los permisos para acceder a la ubicación del dispositivo.

```
// Es un Launcher, se lanzará en el método pedirPermisosDeUbicacion()
pedirPermisos=registerForActivityResult(
    new ActivityResultContracts.RequestMultiplePermissions(), result -> {
        // Para acceder a la ubicación mediante GPS
        Boolean fineLocation=result.get(Manifest.permission.ACCESS_FINE_LOCATION);
        // Para acceder a la ubicación mediante Internet
        Boolean coarseLocation=result.get(Manifest.permission.ACCESS_COARSE_LOCATION);

        // Si se ha aceptado, recibiremos la ubicación
        if (fineLocation != null && fineLocation) {
            obtenerUbicacion();
        } else if (coarseLocation != null && coarseLocation) {
            obtenerUbicacion();
        } else {
            // Cuando se rechace el permiso
            Toast.makeText( context: this, R.string.ubi_denegada, Toast.LENGTH_SHORT).show();
        }
    }
);
```

- **obtenerUbicacion:** tras haber aceptado los permisos, se lanza este método. Obtiene la ubicación actual, utilizando el siguiente método, al cual la pasa la latitud y longitud obtenida.
- **abrirMaps:** abre Google Maps, y se sitúa en la ubicación actual mediante la latitud y longitud. Esta actividad es la que lanza el Intent para llegar a la ubicación.
- **obtenerDireccionPostal:** ya se ha obtenido la dirección, ahora queda insertarla en el TextView del MainActivity. Mediante el **Geocoder** convierte estas coordenadas en una Dirección Postal y la inserta en el TextView en cuestión. Esta se ejecutará cuando se llegue al estado de onResume, es decir, al volver a la aplicación tras haber entrado en Google Maps, y siempre y cuando, haya obtenido una dirección.

**BIKE ACTIVITY:** es un Basic Views Activity que abre el primer fragment, que veremos a continuación.

```
@Override  * Alfonso *
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    binding = ActivityBikeBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    setSupportActionBar(binding.toolbar);

    NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment_content_bike);
    appBarConfiguration = new AppBarConfiguration.Builder(navController.getGraph()).build();
    NavigationUI.setupActionBarWithNavController( activity: this, navController, appBarConfiguration);
}

@Override  5 usages * Alfonso *
public boolean onSupportNavigateUp() {
    NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment_content_bike);
    return NavigationUI.navigateUp(navController, appBarConfiguration) || super.onSupportNavigateUp();
}
```

**PRIMER FRAGMENT:** este primer Fragment va a contener un calendario, en el cual el usuario deberá seleccionar la fecha en la cual quiere utilizar la bici. Para ver que fecha se ha seleccionado, se utiliza el Listener **setOnDateChangeListener**.

```
public void onViewCreated(@NonNull View view, Bundle savedInstanceState) { * Alfonso *
    super.onViewCreated(view, savedInstanceState);

    calendarView = view.findViewById(R.id.calendarView);
    txtFecha = view.findViewById(R.id.txtFecha);

    // SharedPreferences para guardar la fecha
    sharedPreferences = getActivity().getSharedPreferences( name: "Preferencias", Context.MODE_PRIVATE);
    String fecha = sharedPreferences.getString( s: "fecha", s: "Select the date to list the available bikes");
    txtFecha.setText(fecha);

    // Botón para pasar al siguiente Fragment
    binding.buttonFirst.setOnClickListener(new View.OnClickListener() { * Alfonso *
        @Override  * Alfonso *
        public void onClick(View view) {
            // Evitar cambiar de fragment si no se ha seleccionado fecha
            if (cambiarFragment == false) {
                Toast.makeText(getContext(), text: "Select a date", Toast.LENGTH_SHORT).show();
            } else {
                // Transición entre Fragments
                NavOptions animacionNav = new NavOptions.Builder().setExitAnim(R.anim.left_out).setEnterAnim(R.anim.right_in).build();
                NavHostFragment.findNavController( fragment: FirstFragment.this).navigate(R.id.action_FirstFragment_to_SecondFragment, args: null, animacionNav);
            }
        }
    });
}
```

**SHARED PREFERENCES:** en el First Fragment se hace un SharedPreferences, para guardar la fecha en la que el usuario ha seleccionado la bici al salir de la app.

```
// Guardar la nueva fecha en SharedPreferences
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString( s: "fecha", fechaSeleccionada);
editor.apply();
```

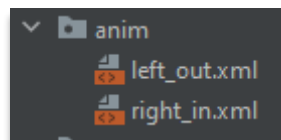
**TRANSICIONES:** se agregan transiciones entre 2 cambios de pantalla distintos:

- **MainActivity – BikeActivity:** esta transición se inserta entre estas 2 pantallas. Al ser en un intent, se utiliza el método **overridePendingTransition**. Estas transiciones ya vienen definidas por defecto en Android Studio.

```
// Transición para la pantalla de login
overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out);
```

- **FirstFragment – SecondFragment:** esta otra transición se va a ejecutar al ir desde el primer Fragment al segundo. Al ser Fragments y no Intent, defino las animaciones en un XML aparte.

```
NavOptions animacionNav=new NavOptions.Builder().setExitAnim(R.anim.left_out).setEnterAnim(R.anim.right_in).build();
NavHostFragment.findNavController( fragment: FirstFragment.this).navigate(R.id.action_FirstFragment_to_SecondFragment, args: null, animacionNav);
```



**SECOND FRAGMENT:** este Fragment contendrá un RecyclerView con todas las bicis disponibles, especificando su Ciudad, Autor, Dirección y Descripción. También se agrega un **Handler** para el **Placeholder**, de forma que aparecerá la barra de carga durante 1 segundo, y luego ya aparecerán los resultados, dando aspecto de que está buscando información.

```
public class SecondFragment extends Fragment { Alfonso

    private FragmentSecondBinding binding; 1 usage
    RecyclerView recycler; 5 usages

    @Override Alfonso
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view=inflater.inflate(R.layout.fragment_second, container, attachToRoot: false);

        recycler=view.findViewById(R.id.recycler);
        recycler.setLayoutManager(new LinearLayoutManager(getContext()));

        // Placeholder para cuando cargan los elementos. Primero estará visible el progressBar, y el resto invisible
        ProgressBar placeholder=view.findViewById(R.id.progressBar);
        placeholder.setVisibility(View.VISIBLE);
        recycler.setVisibility(View.GONE);

        // Cargar los datos del JSON
        BikesContent.loadBikesFromJSON(requireContext());

        // Tras 1 segundo, aparecen los resultados. Primero desaparece el progressBar, y el resto aparece
        new Handler().postDelayed(new Runnable() { Alfonso
            @Override new
            public void run() {
                placeholder.setVisibility(View.GONE);
                recycler.setVisibility(View.VISIBLE);
                recycler.setAdapter(new MyItemRecyclerViewAdapter(requireContext(), BikesContent.ITEMS));
            }
        }, delayMillis: 1000);

        return view;
    }
}
```

**BIKESCONTENT:** es la clase que se utilizará para crear el RecyclerView, la cual contiene los atributos de las bicis.



**CLASE ADAPTER:** la clase **MyItemRecyclerViewAdapter** servirá como Adapter para almacenar las bicis en el RecyclerView. En esta clase se añaden los elementos a cada fila del Recycler, con sus respectivos datos, y con la funcionalidad de que al pulsar en el botón, se abre el GMAIL para poder enviar un email ya predefinido.

```

63 // Método que vincula los datos de una bicicleta con su vista correspondiente
64 @Override // alfonso
65 public void onBindViewHolder(@NonNull final ViewHolder holder, int position) {
66     BikesContent.Bike bike=bikelist.get(position);
67
68     // Se asignan los valores a cada elemento del RecyclerView
69     holder.txtAutor.setText(bike.getOwner());
70     holder.txtDescripcion.setText(bike.getDescription());
71     holder.imagenBici.setImageBitmap(bike.getPhoto());
72     holder.txtCiudad.setText(bike.getCity());
73     holder.txtDireccion.setText(bike.getLocation());
74
75     // Configura el botón para enviar un Email
76     holder.btnEmailAutor.setOnClickListener(new View.OnClickListener() { // alfonso
77         @Override // alfonso
78         public void onClick(View view) {
79             // Contenido que inserta el email, el asunto y el mensaje
80             String contenidoEnviar="mailto:" + Uri.encode(bike.getEmail()) +
81                 "?subject=" + Uri.encode("Bike Request") +
82                 "&body=" + Uri.encode("Dear " + bike.getOwner() + ",\n\n" +
83                 "I'd like to use your bike at " + bike.getLocation() + " (" + bike.getCity() + ").\n\n" +
84                 "Can you confirm its availability?\n\nKindest regards!");
85             Uri uri=Uri.parse(contenidoEnviar);
86
87             Intent emailIntent = new Intent(Intent.ACTION_SENDTO, uri);
88             context.startActivity(emailIntent);
89         }
90     });
91 }

```

## PRUEBAS REALIZADAS

Se realizan diversas pruebas de caja negra para verificar la funcionalidad del programa. Tras ello, se programan sus respectivas soluciones:

### 1. PROBAR DISTINTOS FORMATOS

Se han introducido diferentes cadenas de texto en la MainActivity, para verificar que no permite acceder a la aplicación introduciendo Emails inválidos.

### 2. DISTINTAS RESOLUCIONES:

Se ha probado la aplicación en pantallas con diferentes resoluciones, además de en dispositivos móviles reales. En todos funciona correctamente.

## ESTRUCTURA DE LA APLICACIÓN

**anim:** XML con las animaciones para la transición.

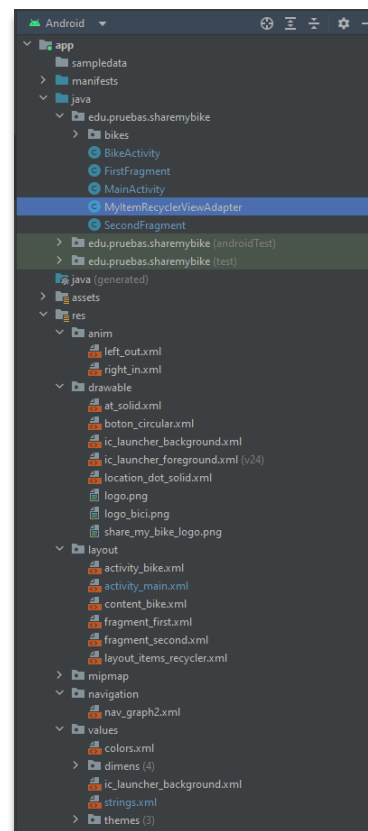
**drawable:** aquí están todos los XML, imágenes y vectores.

**mipmap:** dentro de esta carpeta están los archivos del icono de la aplicación.

**values:** en sus archivos *colors.xml* y *string.xml* se encontrarán todas las variables de colores y strings que hemos creado.

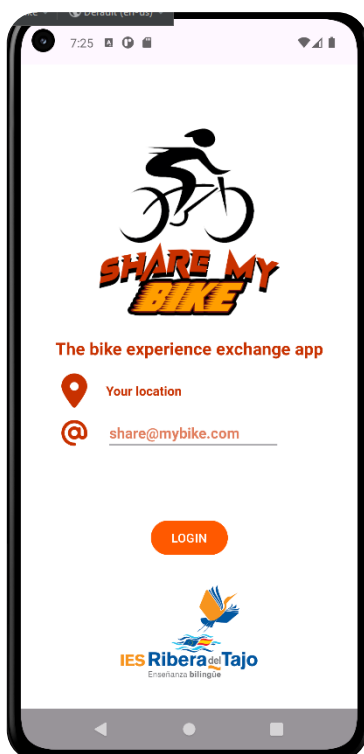
**navigation:** contiene el ToolBar de los Fragments.

**assets:** esta carpeta contiene las imágenes de las bicis, además de el JSON con sus datos.



## EJECUCIÓN DE LA APLICACIÓN

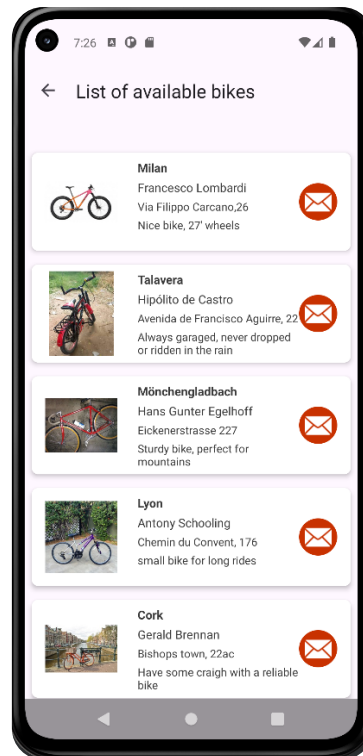
MAIN ACTIVITY



FIRST FRAGMENT



## SECOND FRAGMENT



## ICONO



## COSAS QUE HE APRENDIDO

1. Usar el CalendarView y su Listener.
2. Usar Intents.
3. Obtener la ubicación.
4. Crear la carpeta **assets**.

## BIBLIOGRAFÍA

Códigos de Programación - MR. (2022, 3 noviembre). *Cargar mapas y seleccionar ubicación* - *Android Studio* | *Java* [Vídeo]. YouTube.  
[https://www.youtube.com/watch?v=Df\\_CruiKxFc](https://www.youtube.com/watch?v=Df_CruiKxFc)