Desarrollo de app Android con CityBike y OpenStreetMap

Configuración, Implementación y Mantenimiento de Sistemas Informáticos. Tópico 5.

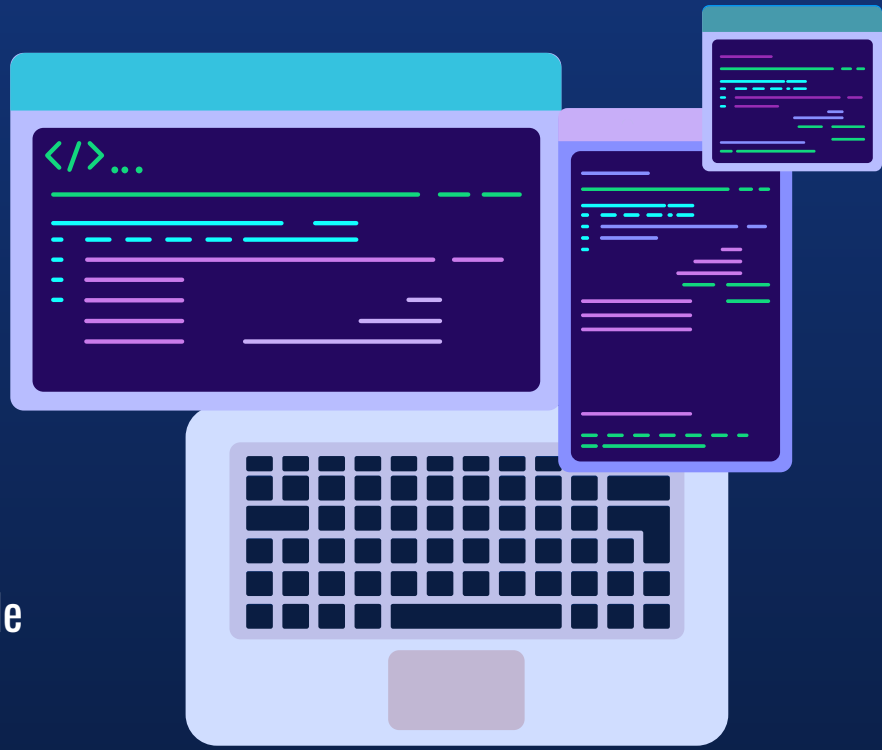


Tabla de Contenidos

1. Resumen


Google AdMob


Google
Analytics


mockito


Firebase Firestore


circleci


OpenStreetMap
The Free Wiki World Map


SEVICI


Google Authenticator


SENTRY

Tabla de Contenidos

2. Introducción

La digitalización ha añadido nuevas oportunidades para aprovechar en nuestro día a día, una de ellas es la movilidad compartida con la que se pueden conseguir ventajas con servicios ágiles y baratos de micromovilidad en las ciudades y ayuda al medio ambiente.

Nuestro proyecto presenta una forma de agilizar el uso del servicio de Sevici ofreciendo información sobre la disponibilidad tanto de bicicletas como de estaciones.

Tabla de Contenidos

3. Estado del Arte

En la actualidad existen varias soluciones software capaces de proveer servicios de *bikesharing* usando diferentes APIs públicas. Tenemos que hacer una distinción entre dos tipos de apps móviles:

- Apps de que trazan rutas, como CityBike
- Apps que ofrecen un servicio privado cuyo objetivo es aportar la infraestructura relativa al medio de transporte.

Algunas de estas apps son: Sevici, Meep, Moovit...

3. Estado del Arte

Principales características de los distintos servicios de *bikesharing* ofrecidos en la actualidad:

	¿Ofrece infraestructura?	Público/Privado	Disponibilidad en ciudades
Sevici	✓	Público	Sevilla
Meep	✗	Privado	17 ciudades
Moovit	✗	Privado	150 ciudades
Mobike	✓	Privado	8 ciudades

Tabla de Contenidos

Arquitectura de la app

API CityBike: podremos obtener información en tiempo real sobre el estado de las estaciones de Sevici y sus datos de disponibilidad junto con la ubicación.

Para usar la API usamos el SDK nativo de Android junto al lenguaje Kotlin para implementar distintos servicios, librerías o arquitecturas.

CircleCI

Una de las plataformas CI/CD más grandes del mundo. Puede llegar a procesar más de 1 millón de desarrollos por día.

Funcionalidades...

- Automatiza despliegue de software para móviles y web
- Gran integración con otras herramientas (Jira)
- Creación de informes/análisis
- Gestión de requisitos

Firebase Firestore

Se trata de una plataforma móvil creada por Google, cuya principal función es desarrollar y facilitar la creación de apps de elevada calidad de una forma rápida, sin perder escalabilidad.

Firestore es la base de datos más reciente de Firebase para el desarrollo de apps para dispositivos móviles. Es una base de datos no relacional (NoSQL) la cual es capaz de almacenar una gran cantidad de datos en tiempo real.

Sentry

Proporciona un rastreador de errores de código abierto para supervisar y responder a errores y fallas en cualquier lugar de tu aplicación en tiempo real.

Registra todos los posibles crashes y bugs que se ejecutan en la app y los almacena en una base de datos, para su posterior tratamiento.

Posee compatibilidad con diversas plataformas, como pueden ser:

- Front-ends desarrollados usando frameworks de JavaScript como ReactJs, VueJS o AngularJS
- Apps móviles usando los SDKs nativos de iOS y Android

Google Analytics

Google Analytics es la herramienta de monitoreo y análisis de sitios web y aplicaciones más utilizada en el mundo.

Nos ofrece diversos KPIs los cuales nos pueden ayudar a obtener información acerca del tiempo que dedica cada usuario.

Ventajas

- Facilidad para configurar
- Integración con otros servicios de Google
- Versión gratuita muy completa

Picasso

Es una potente biblioteca de descarga y almacenamiento en caché de imágenes en tiempo real para Android que nos permite mostrar imágenes en nuestra app dada una URL.

Picasso permite la carga de imágenes sin problemas en tu aplicación.

Maneja muchos de los errores comunes de la carga de imágenes:

- Reciclaje y cancelación de descargas en un adaptador
- Transformación de imágenes complejas
- Almacenamiento en caché de disco y memoria automático

OpenStreetMap

Esta herramienta de código abierto es la alternativa por excelencia a Google Maps. Nos permite mostrar un mapa en nuestra app.

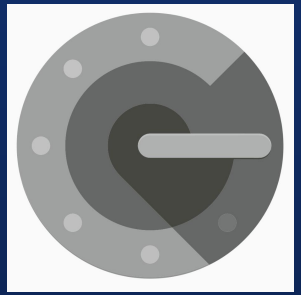
Google AdMob

Plataforma publicitaria para dispositivos móviles enfocada en generar ingresos con aplicaciones móviles.

Monetizar las aplicaciones le permite a los desarrolladores crear una fuente de ingresos suficiente como para seguir creando y desarrollando aplicaciones de alta calidad.

Google Authentication Service

Es un software basado en autenticación con contraseña de un solo uso desarrollado por Google.



Este servicio de rápida implementación nos ofrece la capa de autenticación que necesitamos.

GitHub

Es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador.

La web utiliza el sistema de control de versiones Git.

Características

- Simplicidad
- Multiplataforma
- Multitud de interfaces de usuario

Git

Es un sistema de control de versiones distribuido de código abierto, con el que los desarrolladores pueden administrar su proyecto.

El control de versiones permite descargar un software, realizar cambios y subir la versión que han modificado.

Todas las modificaciones se guardan en versiones independientes, no sobreescribiendo el archivo original.

Mockito

Es un framework para testing de código abierto que permite simular el comportamiento de una clase de forma dinámica.

Con Mockito nos aislamos de las dependencias de otras clases y sólo testeamos la funcionalidad concreta que queremos (test unitario).

AndroidX

Es una biblioteca rediseñada para que los nombres de los paquetes sean más claros.

La jerarquía de Android es solo para las clases predeterminadas de Android, que vienen con el propio sistema operativo. El resto de las bibliotecas o dependencias formarán parte de AndroidX.

Los paquetes de AndroidX reemplazan por completo la biblioteca de compatibilidad, ya que proporcionan paridad de funciones y bibliotecas nuevas.

Los paquetes de AndroidX se mantienen y actualizan por separado mediante un control semántico de versiones.

Android Studio

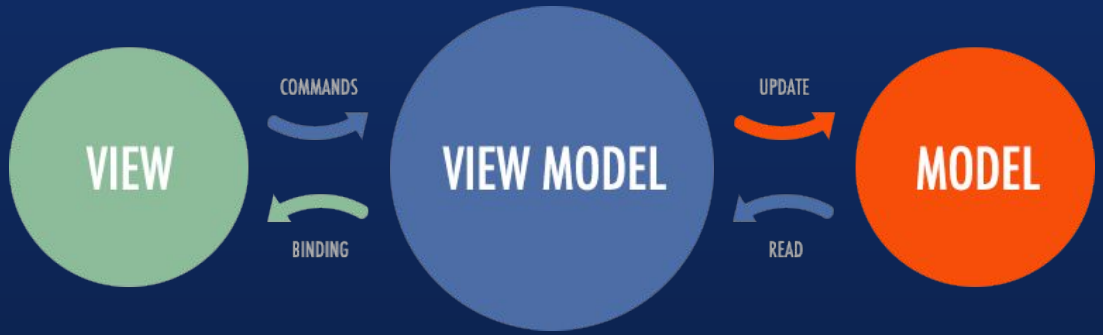
Entorno de desarrollo integrado (IDE) para el desarrollo de apps para Android, basado en Intellij Idea.

Este software cuenta con herramientas y servicios necesarios para crear nuevas aplicaciones. Esto incluye desde el código al diseño de la interfaz de usuario de la aplicación.

Es posible utilizarlo para macOs, Windows, Linux y ChromeOS.

MVVM (Model-View-ViewModel)

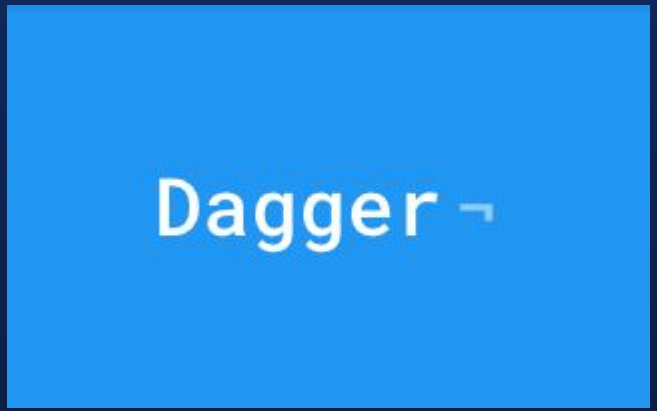
Es el patrón de arquitectura de software más avanzado en la actualidad. Se usa en el desarrollo de software para dispositivos Android e iOS, así como proyectos que hacen uso de microservicios. Su principal característica es aislar al máximo posible la interfaz de usuario de la lógica de negocio de la aplicación.



Dagger

Dagger es una librería, actualmente de Google, para inyección de dependencia (DI). La DI es una solución que tienen los desarrolladores para modularizar la creación de objetos y encapsular sus instancias.

A pesar de que la DI tiene una integración compleja, su implementación es prácticamente obligatoria en cualquier proyecto actual, ya que su potencial y posibilidad de personalización para la escalabilidad de la app son muy potentes.



Retrofit

Es una librería para Android y Java compatible con Kotlin para hacer llamadas de red, obtener el resultado y “parsearlo” de forma auténtica a su objeto.

Facilita mucho realizar peticiones a un API y procesar la respuesta.

Código fuente

- Dominios
- Dependencias
- API
- Adaptadores
- Activities
- Presentadores
- ...

Dominios

```
@IgnoreExtraProperties
class FavStation(
    var name: String? = null,
    var address: String? = null,
    var latitude: String? = null,
    var longitude: String? = null,
)
```

```
data class Extra(
    var address: String,
    var status: String
)
```

```
data class Network(
    var name: String,
    var location: NetworkLocation?,
    var stations: List<Station>
)
```

```
data class NetworkLocation(
    var city: String,
    var country: String,
    var latitude: Double,
    var longitude: Double
)
```

```
data class NetworkResponse(var network: Network?)
```

```
data class Station(
    var name: String,
    var latitude: Double,
    var longitude: Double,
    var empty_slots: Int,
    var free_bikes: Int,
    var extra: Extra
)
```

Dependencias

```
@Module
class BikesApiModule {
    @Provides
    fun provideBikesApi(): BikesServiceAPI = BikesServiceAPI.create()
}
```

```
@Component(modules = [BikesApiModule::class, ActivityModule::class])
interface ActivityComponent {
    fun inject(activity: MainActivity)
    fun inject(favoriteStationsActivity: FavoriteStationsActivity)
    fun inject(signInActivity: SignInActivity)
}
```

```
@Module
abstract class ActivityModule {
    @Binds
    abstract fun bindPresenter(presenter: MainActivityPresenter): MainActivityContract.Presenter
}
```

API

```
const val CityBikeBaseUrl = "https://api.citybik.es"

enum class NetworkId(private val id: String) {
    Sevici( id: "sevici");

    override fun toString(): String = id
}

interface BikesServiceAPI {
    @GET( value: "/v2/networks/{id}")
    suspend fun loadNetwork(@Path( value: "id") networkId: String): NetworkResponse

    @GET( value: "/v2/networks/{id}?fields=stations")
    suspend fun loadStations(@Path( value: "id") networkId: String): NetworkResponse

    companion object Factory {
        fun create(): BikesServiceAPI = Retrofit.Builder()
            .addConverterFactory(GsonConverterFactory.create())
            .baseUrl(CityBikeBaseUrl)
            .build()
            .create(BikesServiceAPI::class.java)
    }
}
```

Adaptadores

```
internal class StationsAdapter(private var itemsList: List<FavStation>) :
    RecyclerView.Adapter<StationsAdapter.MyViewHolder>() {

    internal inner class MyViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        var name: TextView = view.findViewById(R.id.item_station_name_text_view)
        var address: TextView = view.findViewById(R.id.item_station_address_text_view)
    }

    @NonNull
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val itemView = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_fav_station, parent, attachToRoot: false)
        return MyViewHolder(itemView)
    }

    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
        val item = itemsList[position]
        holder.name.text = item.name
        holder.address.text = item.address
    }

    override fun getItemCount(): Int {
        return itemsList.size
    }
}
```

Activities

```
class MainActivity : AppCompatActivity(), MainActivityContract.View {

    @Inject
    lateinit var presenter: MainActivityContract.Presenter
    lateinit var mAdView: AdView
    lateinit var mGoogleSignInClient: GoogleSignInClient
    val db = Firebase.firestore
    private val auth by lazy {...}

    override fun onCreate(savedInstanceState: Bundle?) {...}

    private fun getFavoriteStationsData() {...}

    private fun initializeUIObjects() {...}

    private fun initializeGoogleSignIn() {
        val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken("522814181367-0jp1og4om1epljq7r5bard90c8m5lfq7.apps.googleusercontent.com")
            .build()
        mGoogleSignInClient = GoogleSignIn.getClient(this, gso)
        // AdMob banner id: ca-app-pub-5467669858609367/5422714505
        // For testing purposes: ca-app-pub-3940256099942544/6300978111
    }

    private fun initializeAdMob() {
        MobileAds.initialize( context: this)
        mAdView = findViewById(R.id.adView)
        val adRequest = AdRequest.Builder().build()
        mAdView.loadAd(adRequest)
```

Project update recommended
Android Gradle Plugin can be upgraded.

Device File Explorer
Emulator

Presentadores

```
class MainActivityPresenter @Inject constructor(private val bikesApi: BikesServiceAPI) :
    MainActivityContract.Presenter {

    private lateinit var view: MainActivityContract.View
    private var stationsUpdater: Job? = null

    override fun attach(view: MainActivityContract.View) {
        this.view = view
    }

    override fun loadNetwork(networkId: NetworkId) {
        println("Loading city bikes network of $networkId...")
        CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
            val response = try {
                bikesApi.loadNetwork(networkId.toString())
            } catch (e: Exception) {
                withContext(Dispatchers.Main) { callFailed(e.message) }
                null
            }

            withContext(Dispatchers.Main) {  this: CoroutineScope
                processMapUpdate(response?.network)
            }
        }
    }

    private fun processMapUpdate(network: Network?) = if (network != null) {
        try {
            view.updateMap(
```

Device File Explorer

Emulator

Project update recommended

Android Gradle Plugin can be upgraded.

Tabla de Contenidos

5. Características de alta disponibilidad

Para aumentar la disponibilidad de nuestro servicio hemos implementado Firestore para conservar los datos de las estaciones que tengamos como favoritas cuando la API no esté disponible.

Tabla de Contenidos

6. Como escalar el servicio

Usamos la arquitectura MVVM haciendo el servicio 100% escalable debido a que la inyección de dependencias nos permite implementar más llamadas a la API con el código ya implementado.

En nuestro caso, *MainActivity* tomaría de una variable global las distintas estaciones obtenidas por la API sin adentrarse en "cómo" se obtienen esas estaciones.

Por lo cual, para *MainActivity* no existe forma para obtener esos datos, solo usa los que consigue una clase "Utilidades" que está modularizada y puede obtener dichos datos.

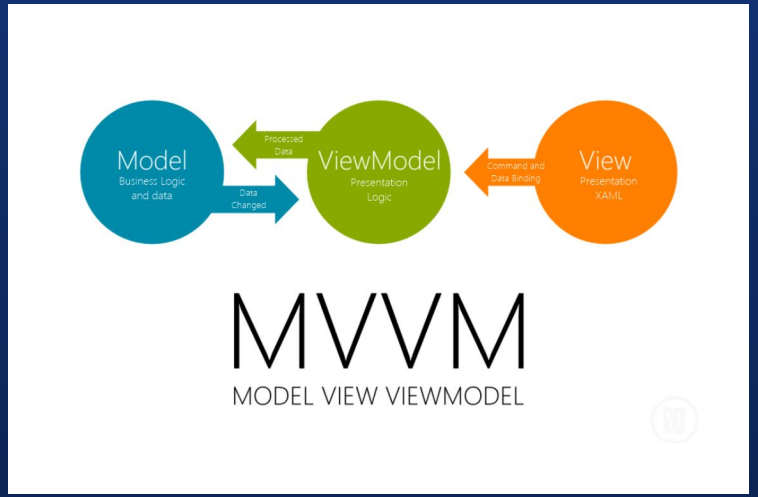


Tabla de Contenidos

Mantenimiento de una App Móvil

Las apps son sometidas a numerosas actualizaciones. Pueden ser:

- Actualización de la versión del lenguaje de programación usado (Kotlin, Java, ...)
- Actualización de Gradle o Maven (Gestores de paquetes Android)
- Actualización de librerías de terceros que incluyen nuevas funcionalidades en su software
- Actualizaciones propias del SDK de Android

Tabla de Contenidos

Trabajo futuro

- Mejorar el patrón de diseño haciendo uso de programación reactiva(RxKotlin) para poder ofrecer así un proyecto de ejemplo acerca de cómo usar programación reactiva en Android usando GitHub.

- Modularizar el uso de la API, añadiendo información sobre patinetes eléctricos, motos eléctricos...

Tabla de Contenidos

Bibliografía y referencias

- https://en.wikipedia.org/wiki/Android_Studio
- https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vista#Historia
- https://en.wikipedia.org/wiki/Mockito
- https://dagger.dev/
- https://es.wikipedia.org/wiki/Firebase
- https://circleci.com/
- https://es.wikipedia.org/wiki/Google_Analytics
- https://www.atlassian.com/es/git/tutorials/what-is-git

Bibliografía y referencias

Diversos cursos que nos sirvieron para desarrollar la aplicación:

- **Curso Android (2h 24m):** https://www.youtube.com/watch?v=ebQphhLpJG0
- **Firebase Js (53m):** https://www.youtube.com/watch?v=ebQphhLpJG0
- **MongoDB (1h21m):**https://www.youtube.com/watch?v=IWMemPN9t6Q
- **AWS Amplify (1h) :** https://aws.amazon.com/es/getting-started/hands-on/build-android-app-amplify/module-one/