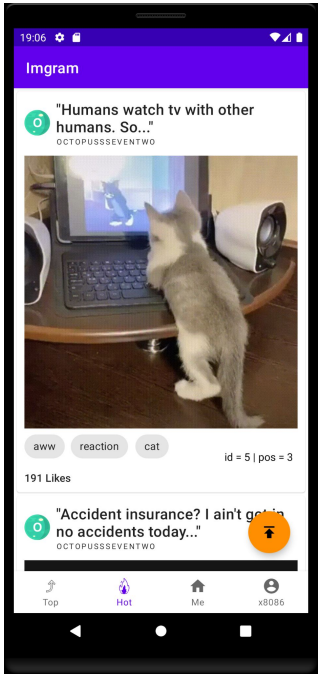
Android Avanzado

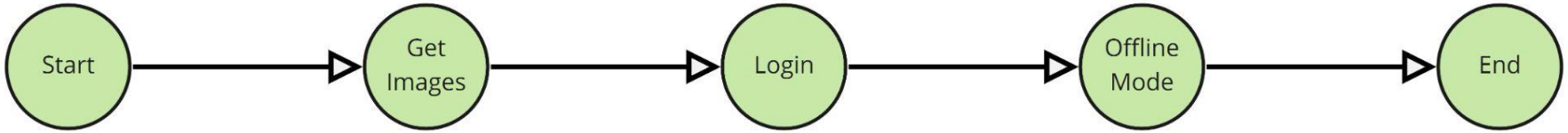
KEEPCODING
Tech School

Objetivo



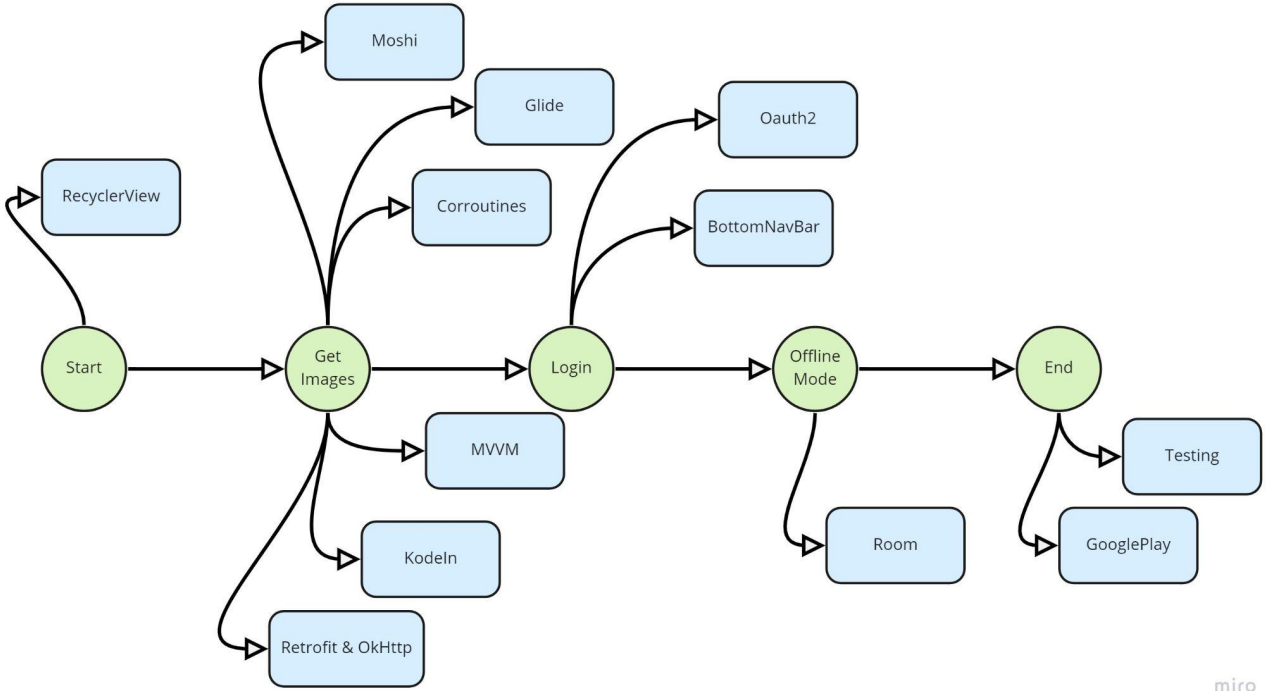
imgur

Roadmap

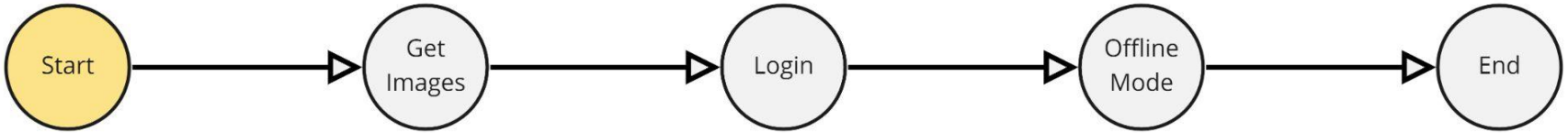


miro

Real Roadmap

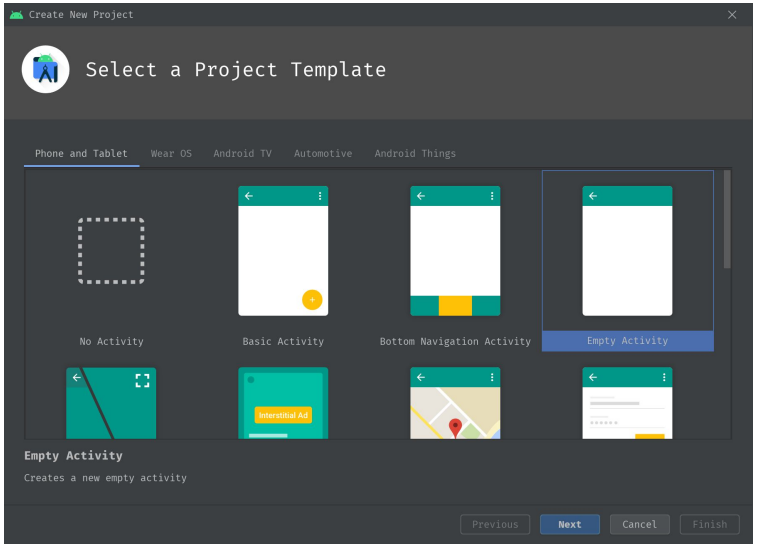


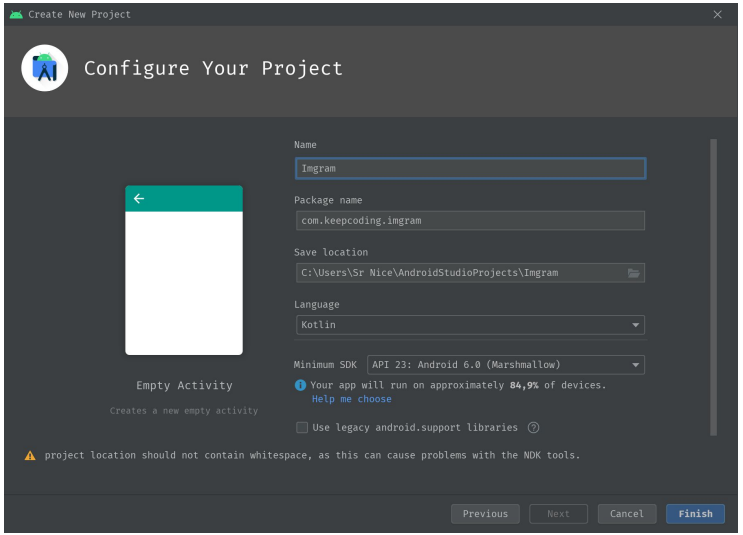
Start



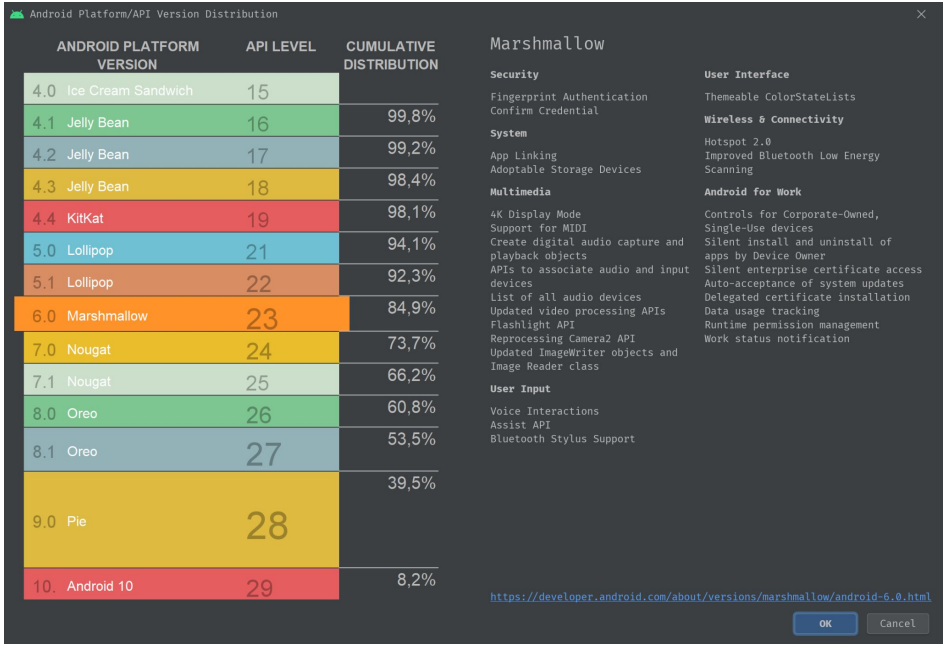
miro

Setup project





Android fragmentacion



ViewBinding

```
//build.gradle

android {
    ...
    buildFeatures {
        viewBinding  = true
    }
}
```

- Generación de código
- Cada fichero dentro de layout:
  - home_activity.xml -> HomeActivityBinding

Crear RecyclerView

```
<androidx.recyclerview.widget.RecyclerView
        android:id="@+id/gallery_recycler_view"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
```

Crear Adapter

```
class GalleryRecyclerAdapter : RecyclerView.Adapter<GalleryItemViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): GalleryItemViewHolder {
        TODO("Not yet implemented")
    }

    override fun onBindViewHolder(holder: GalleryItemViewHolder, position: Int) {
        TODO("Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO("Not yet implemented")
    }
}
```

Definir modelo de vista

```
data class Image(
    val id: String,
    val title: String?,
    val url: String
)
```

Crear Gallery row

```
<com.google.android.material.card.MaterialCardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp">

        <TextView
            android:id="@+id/gallery_item_title_text"
            style="@style/TextAppearance.MaterialComponents.Headline6"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:text="Title" />

        <androidx.appcompat.widget.AppCompatImageView
            android:id="@+id/gallery_item_image_view"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_marginTop="12dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/gallery_item_title_text" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```

Crear recycler ViewHolder

```
class GalleryItemViewHolder(
    private val binding: GalleryItemLayoutBinding
) : RecyclerView.ViewHolder(binding.root) {
```


Implementar metodos Adapter

```
var images: List<Image> = emptyList()
    set(value) {
        field = value
        notifyDataSetChanged()
    }

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) =
    GalleryItemLayoutBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        .run { GalleryItemViewHolder(this, id) }

override fun onBindViewHolder(holder: GalleryItemViewHolder, position: Int) {
    holder.bind(image = images[position])
}

override fun getItemCount(): Int = images.count()
```

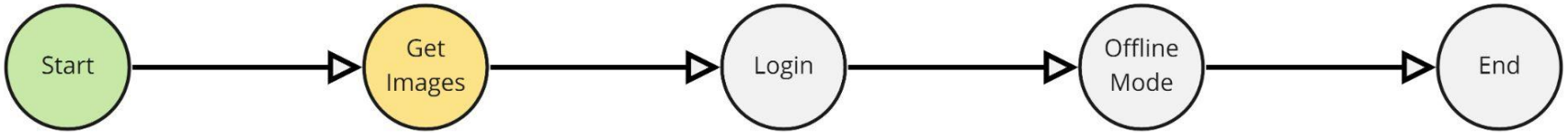
Configurar RecyclerView

```
binding = HomeActivityBinding.inflate(layoutInflater).also { setContentView(it.root) }
val adapter = GalleryRecyclerAdapter().also { binding.galleryRecyclerView.adapter = it }
adapter.images = listOf(
    Image(
        id = "1",
        title = "Titulo 1",
        url =  "https://picsum.photos/200/300"
    )
)
```

Bindear datos a la celda

```
fun bind(image: Image) {
    with(binding) {
        galleryItemTitleText.text = image.title ?: "No title"
        galleryItemImageView.load(image.url)
    }
}
```

Get Images

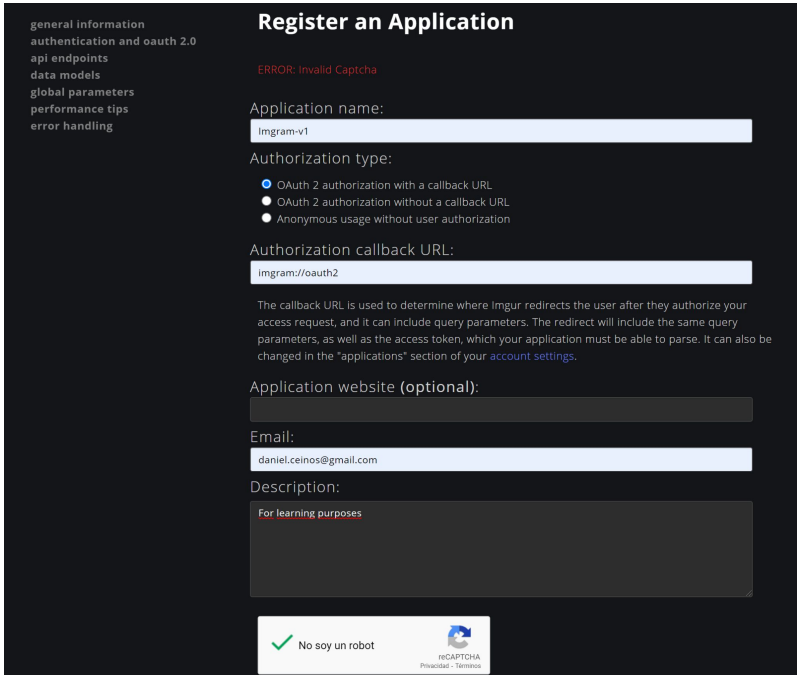


Imgur API lookup

https://apidocs.imgur.com/

Imgur crear aplicacion

https://api.imgur.com/oauth2/addclient



OkHttp3

https://square.github.io/okhttp/
https://github.com/square/okhttp

```
// build.gradle

implementation("com.squareup.okhttp3:okhttp:4.9.1")
```


Crear cliente OkHttp3

```
val okHttp3Client = OkHttpClient().newBuilder().build()
```


Permiso internet

```
<manifest>

    <uses-permission android:name="android.permission.INTERNET" />
    ...

</manifest>
```


Our first network request

```
val okHttp3Client = OkHttpClient().newBuilder().build()
val request: Request = Request.Builder().url(image.url).build()
val response = okHttp3Client.newCall(request).execute()
```

Asincronia en Android

- Threads
- Callbacks
- Handler
- Rx(Similar a combine)
- Coroutines

Threads

```
fun main() {
    Thread().run {
        myFunction("Hey!")
    }
}

fun myFunction(msg: String) {
    println(msg)
}
```

Pros:

- Nuevo hilo distinto del hilo de UI.

Cons:

- Manejar threads directamente no es nunca una buena idea.

Callbacks

```
fun main() {
   myFunction(4) {
       result -> println(result)
   }
}

fun myFunction(number: Int,
               callback: (result: Int) -> Unit) {
    callback(number * number)
}
```

Pros:

- Facil de entender

Cons:

- Callback Hell

Rx

```
fun main() {
    myFunction(5).subscribe {
        println(it)
    }
}

fun myFunction(number: Int): Observable<Int> {
    return Observable.just(number)
}
```

Pros:

- Muy usada en todo tipo de aplicaciones
- Tiene potentes operadores

Cons:

- Curva de aprendizaje

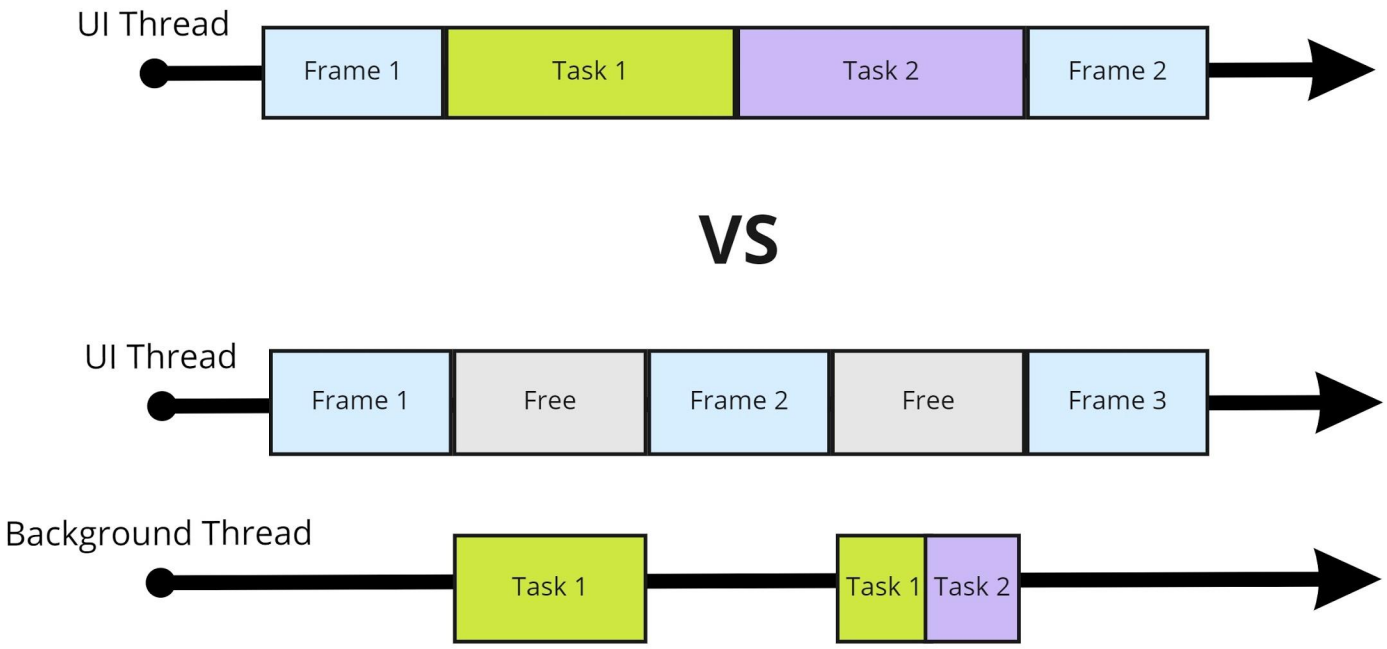
Corroutinas

```
fun main() {
    GlobalScope.launch {
        val result = myFunction(4)
        println(result)
    }
}

suspend fun myFunction(number: Int): Int {
    return number * number
}
```

Pros:

- Full Kotlin
- Abstraccion de threads
- Operadores
- Semantica

UI Thread

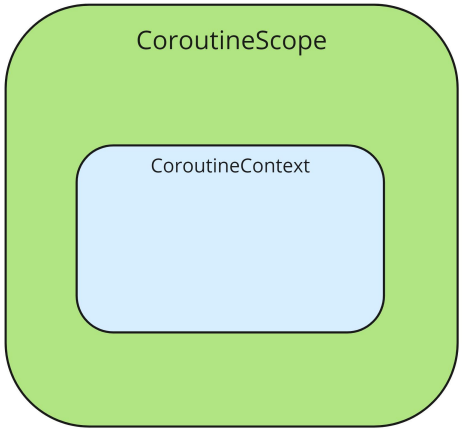


Coroutines

Coroutines != Threads

- En un corrutina los hilos no se bloquean se *suspenden*.
- Al lanzar una coroutinas especificamos el tipo de hilo en el que esta será ejecutada, principalmente tendremos:
  - Main : es el hilo principal de la aplicación, en el caso de Android el hilo de UI
  - IO : Diseñado para las tareas bloqueantes de entrada y salida como escritura en disco o llamadas de red

Scopes y Context



Los CorroutineScope se usarán para agregar corrutinas que vivirán en un scope con un ciclo de vida determinado, como una pantalla o toda la Aplicación.

Ejemplo podemos declarar un Scope por cada viewmodel y vincular ambos ciclos de vida.

Scopes y Context

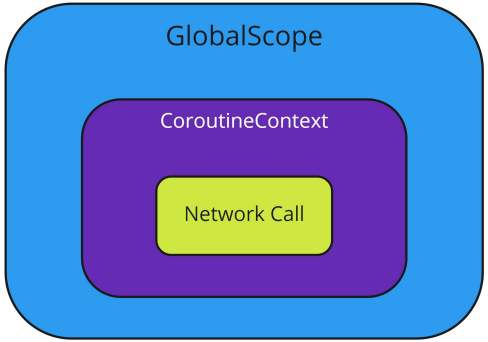
Cada CorroutineScope contendrá un contexto, ese contexto extenderá de *CorroutineContext*, y contendrá toda la info para que las corrutinas sean ejecutadas.

⚠️ Una suspend function solo podrá ser invocada dentro de un contexto de corutina.

Scopes y Context

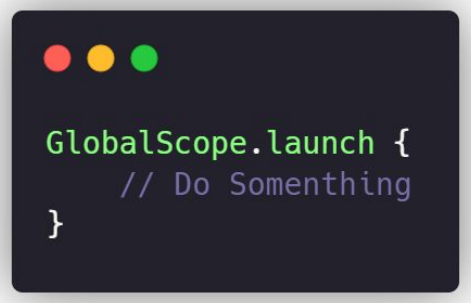
```
fun main() {
    GlobalScope.launch {
        networkCall()
    }

    networkCall() <- ⚠ Error ⚠
}

suspend fun networkCall() {
    val result = api.getImages()
}
```

Se **lanzará** una corrutina dentro **GlobalScope** y se ejecutará la llamada de red en el **contexto** de **GlobalScope**



Launch

Launch iniciará una nueva corrutina.



Suspend function

```
fun main() {
    GlobalScope.launch {
      val result = myFunction(2)
    }
}

suspend fun myFunction(number: Int): Int {
    return otherFunction(number)
}

suspend fun otherFunction(number: Int): Int {
    return number * number
}
```

Una función anotada como *suspend* será capaz de pausar la corrutina hasta que la ejecución de esta función acabe.

Dentro de una *suspend* function podremos llamar a otras *suspend* functions.

GlobalScope

GlobalScope es *global* , si es cancelado no se podrán lanzar nuevas corrutinas en él.

Job

```
val job = GlobalScope.launch {
    networkCall()
}

suspend fun networkCall() {
    val result = api.getImages()
}
```

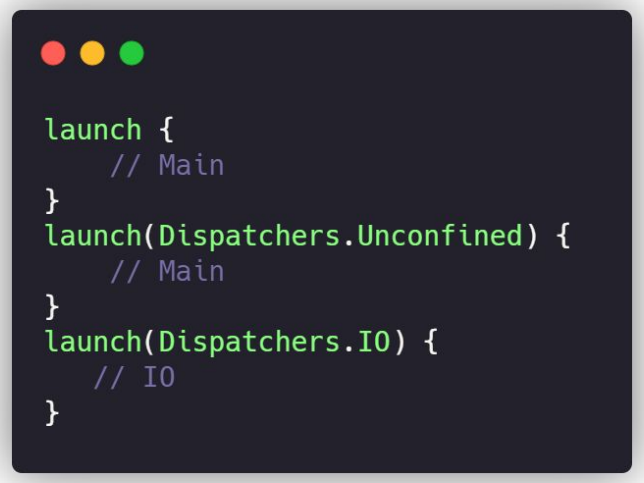
Un *Job* es un objeto que contiene una referencia a la corrutina.
Mediante la interacción con este podremos obtener datos como el estado en el que se encuentra (en curso, terminada, cancelada, etc), permitir esperar a que este concluya.

Esperar a un Job

```
val job = GlobalScope.launch {
    networkCall()
}

job.join()
```

Mediante la suspend function *join()* podremos pausar la corrutina hasta que este Job se encuentre en un estado completado.

Dispatchers



```
launch {
    // Main
}
launch(Dispatchers.Unconfined) {
    // Main
}
launch(Dispatchers.IO) {
   // IO
}
```


Añadir corroutinas

```
private fun loadImage(url: String, imageView: ImageView) {
    CoroutineScope(Dispatchers.IO).launch {
        val okHttp3Client = OkHttpClient().newBuilder().build()
        val request: Request = Request.Builder().url(url) .build()
        val response = okHttp3Client.newCall(request).execute()
        val bitmap = BitmapFactory.decodeStream(response.body!!.byteStream())
        imageView.setImageBitmap(bitmap)
    }
}
```

Cambio de hilo

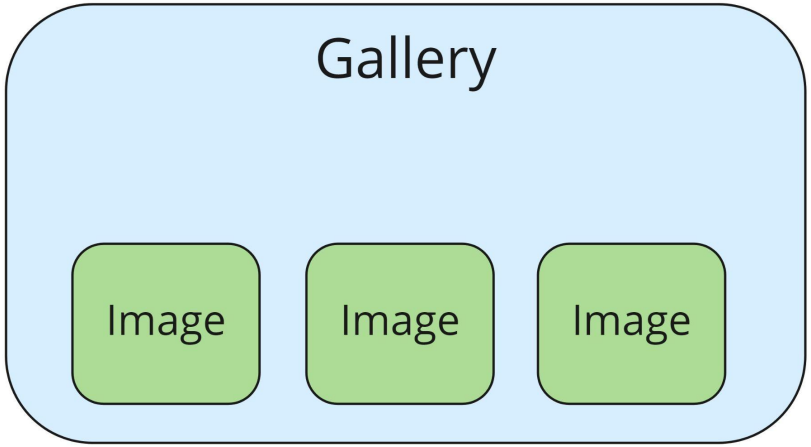
```
withContext(Dispatchers.Main) {
    imageView.setImageBitmap(bitmap)
}
```

Para ciertas operaciones como actualizar elementos de la UI **deben hacerse desde el hilo principal**

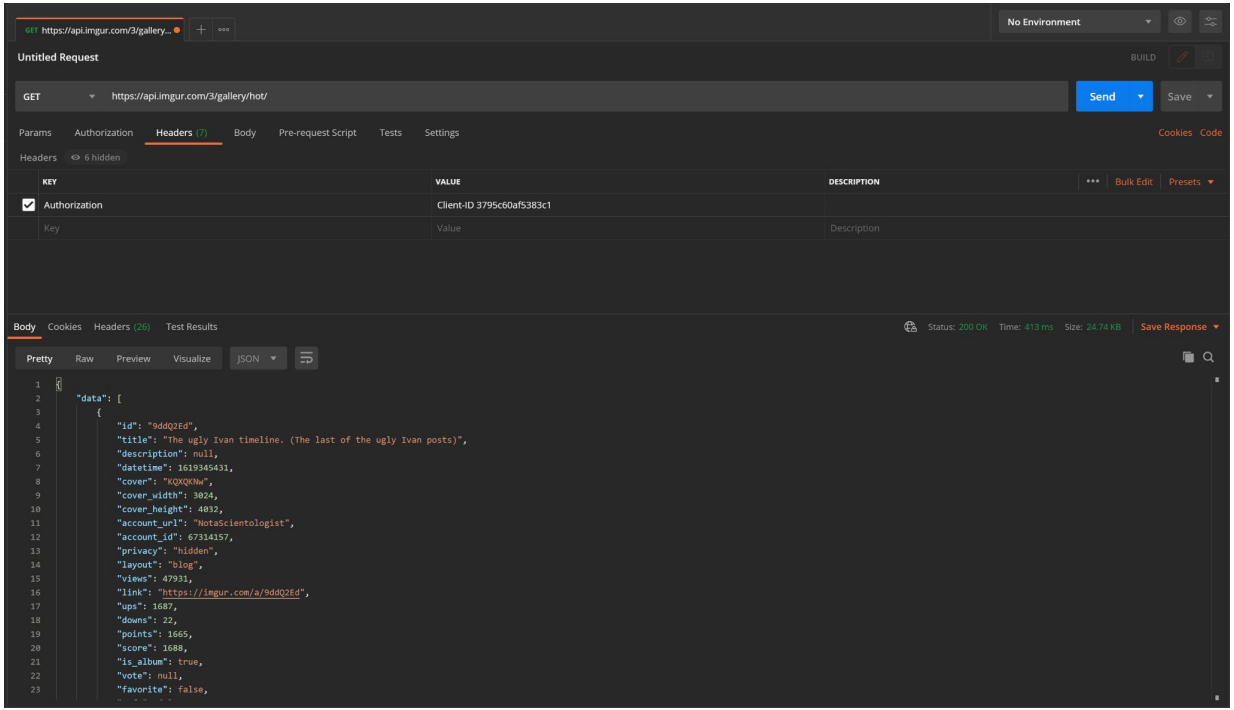
API Imgur Obtener imagenes

https://api.imgur.com/3/gallery/hot/

Creación de los modelos de Gallery e Image



Creación de los modelos de Gallery e Image



Creación de los modelos de Gallery e Image

```
data class ImgurImage(
    val account_id: String?,
    val account_url: String?,
    val datetime: Long,
    val deletehash: String?,
    val description: String?,
    val favorite: Boolean,
    val id: String,
    val in_gallery: Boolean,
    val in_most_viral: Boolean,
    val link: String,
    val name: String?,
    val tags: List<ImgurTag>,
    val title: String?,
    val type: String,
    val views: Int
)
```

```
data class ImgurGallery(
    val account_id: String?,
    val account_url: String?,
    val comment_count: Int,
    val datetime: Long,
    val description: String?,
    val downs: Int,
    val favorite: Boolean,
    val favorite_count: Int,
    val id: String,
    val in_gallery: Boolean,
    val in_most_viral: Boolean,
    val images: List<ImgurImage>?,
    val is_album: Boolean,
    val link: String,
    val points: Int,
    val score: Int,
    val section: String,
    val tags: List<ImgurTag>,
    val title: String,
    val topic: String?,
    val topic_id: String?,
    val type: String?,
    val ups: Int,
    val views: Int,
)
```

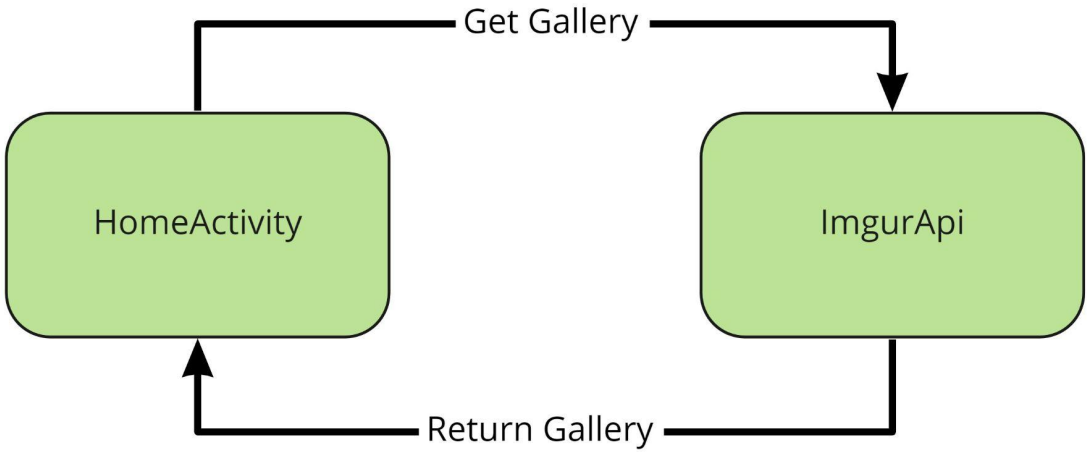
```
data class ImgurTag(
    val accent: String?,
    val description: String?,
    val display_name: String,
    val name: String
)
```

Retrofit y moshi

```
val moshi = Moshi.Builder().add(KotlinJsonAdapterFactory()).build()

val retrofit = Retrofit.Builder()
    .baseUrl("https://api.imgur.com/3/")
    .client(instance())
    .addConverterFactory(MoshiConverterFactory.create(moshi))
    .build()

val imgurApi = retrofit.create(ImgurApi::class.java)
```

```
interface ImgurApi {

    @Headers("Authorization: Client-ID 3795c60af5383c1")
    @GET("gallery/hot/")
    suspend fun getHotGallery(): ImgurResponse<List<ImgurGallery>>
}
```

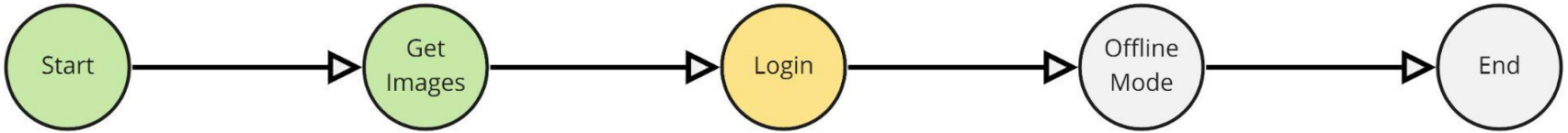
Llamar a la API



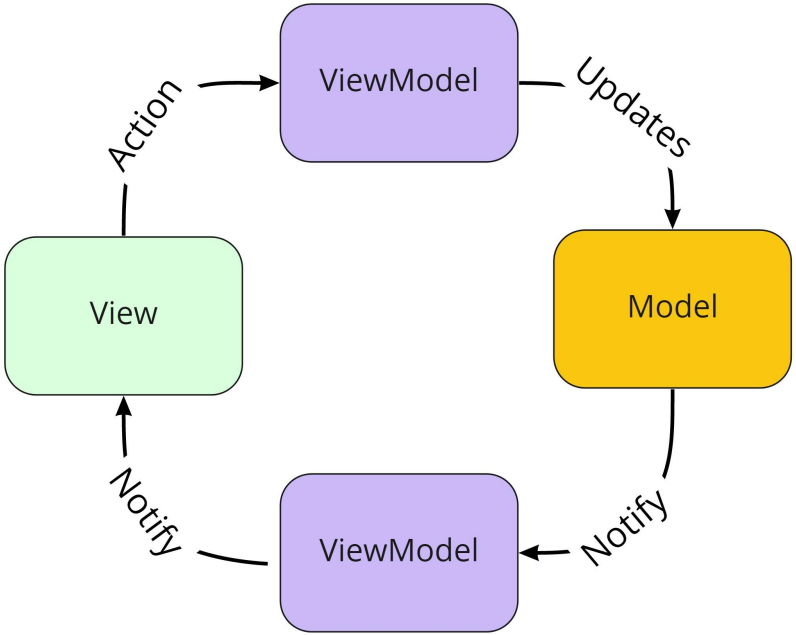
Glide for image loading

https://github.com/bumptech/glide

Login



MVVM



Ayuda a desacoplar la lógica de presentación de la vista.

Nos permite tener un flujo de la información de una sola direccion

Home MVVM

```
class HomeViewModel(private val imgurApi: ImgurApi) {

    private val homeMutableStateFlow = MutableStateFlow(HomeViewData.empty)
    val homeVD: StateFlow<HomeViewData>
        get() = homeMutableStateFlow


    fun loadHotGallery() {
        fetchGallery { imageRepository.fetchHot() }
    }

    private fun fetchGallery() {
        viewModelScope.launch(Dispatchers.IO) {
            homeMutableStateFlow.value = HomeViewData(emptyList(), true)
            val images = imgurApi.getHotGallery()
            homeMutableStateFlow.value = HomeViewData.from(images)
        }
    }
}

data class HomeViewData(val images: List<Image>, val loading: Boolean)
```

Android ViewModel

- Vincula el ViewModel con el ciclo de vida de la actividad
- Añade un viewModelScope para lanzar corrutinas
- La inicialización requiere hacerla mediante un Provider

Inyección de dependencias con Kodein

La inyección de dependencias nos permite delegar la construcción de nuestras clases a un punto común.

Existen multiples DI:

- Dagger/Hilt
- KodeIn
- Koin

Inyección de dependencias con Kodein

```
implementation 'org.kodein.di:kodein-di:7.1.0'
implementation 'org.kodein.di:kodein-di-framework-android-x:7.1.0'
```

Injeccion de las clases de red

```
object NetworkDIModule {

    fun create() = DI.Module(name = this::class.simpleName!!, allowSilentOverride = false, init =
builder)

    private val builder: DI.Builder.() -> Unit = {
        bind<OkHttpClient>() with singleton {
            return@singleton OkHttpClient().newBuilder()
                .addInterceptor(logging)
                .build()
        }
        bind<Retrofit>() with singleton {
            val moshi = Moshi.Builder().add(KotlinJsonAdapterFactory()).build()

            return@singleton Retrofit.Builder()
                .baseUrl("https://api.imgur.com/3/")
                .client(instance())
                .addConverterFactory(MoshiConverterFactory.create(moshi))
                .build()
        }
    }
}
```

Injeccion de los ViewModel

```
class AppDIModule(private val application: Application) {

    fun create() = DI.Module(name = this::class.simpleName!!, allowSilentOverride = false, init =
builder)

    private val builder: DI.Builder.() -> Unit = {
        bind<Application>() with singleton { application }

        bind<SharedPreferences>() with singleton { instance<Application>
().getSharedPreferences("shared", Context.MODE_PRIVATE) }
        bind<ViewModelProvider.Factory>() with singleton { DIViewModelFactory(di) }
    }
}

class DIViewModelFactory(private val di: DI) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T =
di.direct.Instance(erased(modelClass))
}
```

```
object ViewModelDIModule {

    fun create() = DI.Module(name = this::class.simpleName!!, allowSilentOverride = false, init =
builder)

    private val builder: DI.Builder.() -> Unit = {
        bind<HomeViewModel>() with provider { HomeViewModel(instance()) }
    }
}
```

Inicialización KodeIn

```
class App : Application(), DIAware {

    override val di by DI.lazy {
        import(AppDIModule(this@App).create())
        import(ViewModelDIModule.create())
        import(NetworkDIModule.create())
    }
}
```

Observando los cambios

```
private fun observeViewModel() {
    lifecycleScope.launchWhenStarted {
        viewModel.homeVD.collect {
            adapter.images = it.images
            binding.homeProgressBar.visibility = when (it.loading) {
                true -> View.VISIBLE
                false -> View.GONE
            }
        }
    }
}
```


LiveData or StateFlow?

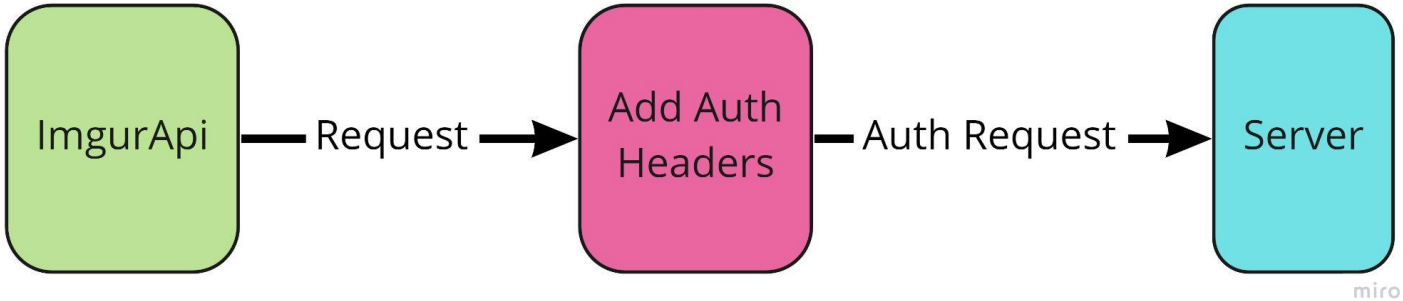
StateFlow actualmente se encuentra en estado experimental, pero su uso es bastante similar a LiveData y tiene todos los beneficios de los operadores de corutinas

BottomNavigationBar

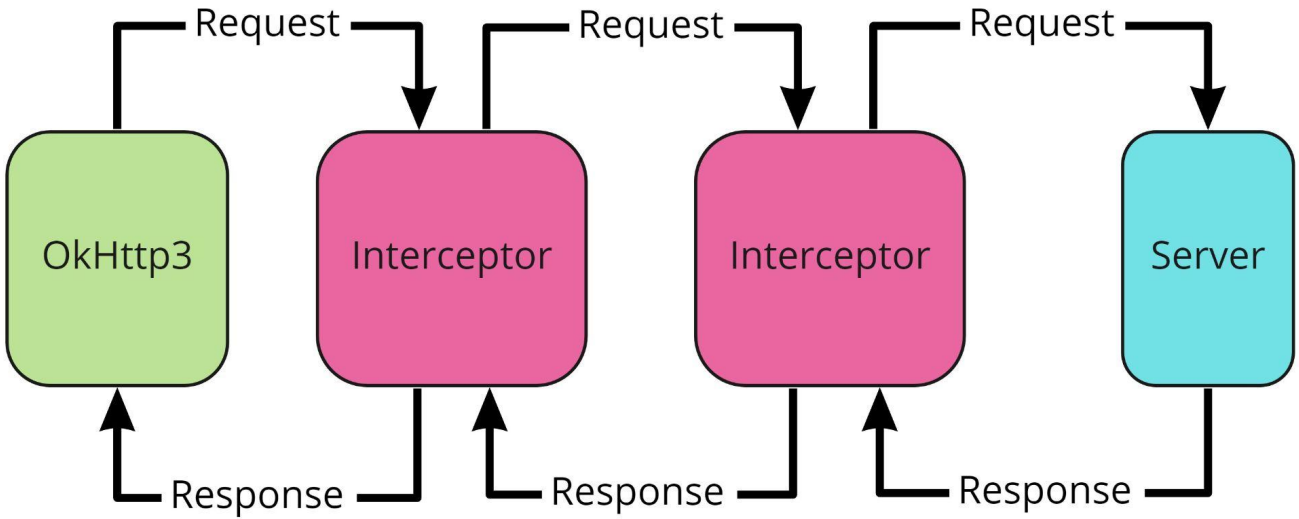
```
<com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/home_bottom_nav_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:labelVisibilityMode="labeled"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:menu="@menu/home_bottom_nav_menu" />
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_top"
        android:enabled="true"
        android:icon="@drawable/ic_top"
        android:iconTint="#00FFFFFF"
        android:title="Top" />

    <item
        android:id="@+id/menu_me"
        android:enabled="true"
        android:icon="@drawable/ic_home"
        android:iconTint="#00FFFFFF"
        android:title="Me"
        android:visible="false" />

    <item
        android:id="@+id/menu_account"
        android:enabled="true"
        android:icon="@drawable/ic_account"
        android:iconTint="#00FFFFFF"
        android:title="Login" />
</menu>
```

How session works



OkHttp3 Interceptors



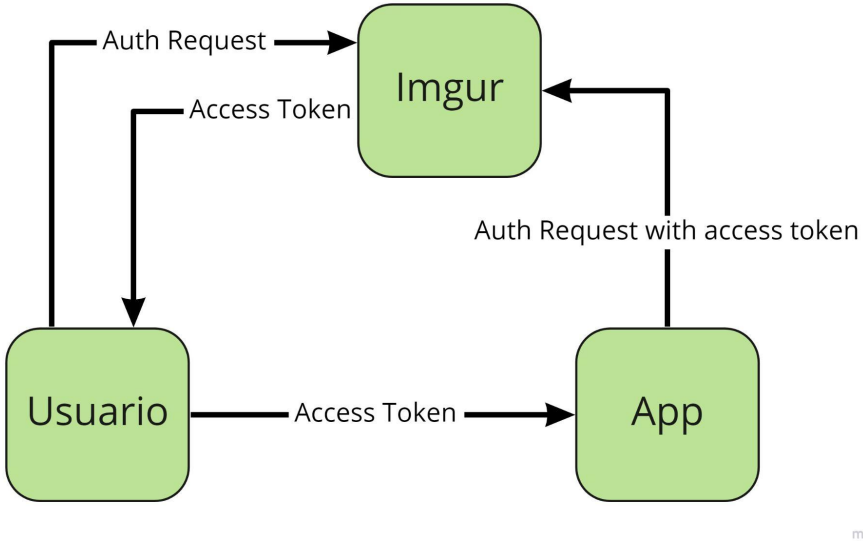
Creando un interceptor

```
class LogInterceptor: Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request()
        Timber.d(request.toString())
        val response = chain.proceed(request)
        Timber.d(response.toString())
        return response
    }
}
```

```
OkHttpClient().newBuilder()
    .addInterceptor(LogInterceptor())
    .build()
```

Oauth2 flow

https://tools.ietf.org/html/rfc6749



Oauth2 Imgur

- Invocamos la url de auth de imgur con nuestro client id https://api.imgur.com/oauth2/authorize?client_id=clientid&response_type=token
- Esta web devolverá una redirección a imgram://oauth2
- Capturamos esta url en nuestra app
- Leemos el token de secion

Invocando Oauth2

```
private fun configureBottomNav() {
    binding.homeBottomNavBar.setOnNavigationItemSelectedListener {
        when (it.itemId) {
            R.id.menu_hot -> viewModel.loadHotGallery()
            R.id.menu_account -> goToOauth2()
        }
        true
    }
    binding.homeBottomNavBar.selectedItemId = R.id.menu_top
}
```

```
private fun goToOauth2() {
    val url = "https://api.imgur.com/oauth2/authorize?client_id=clientId&response_type=token"
    Intent(Intent.ACTION_VIEW).apply { data = Uri.parse(url) }.also { startActivity(it) }
}
```

Capturar redirección Oauth2

```
<manifest>

    <application>
        <activity android:name=".ui.home.HomeActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter android:label="oauth2">
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <!-- Accepts URIs that begin with "imgram://oauth2" -->
                <data
                    android:host="oauth2"
                    android:scheme="imgram" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Recuperar token de sesión

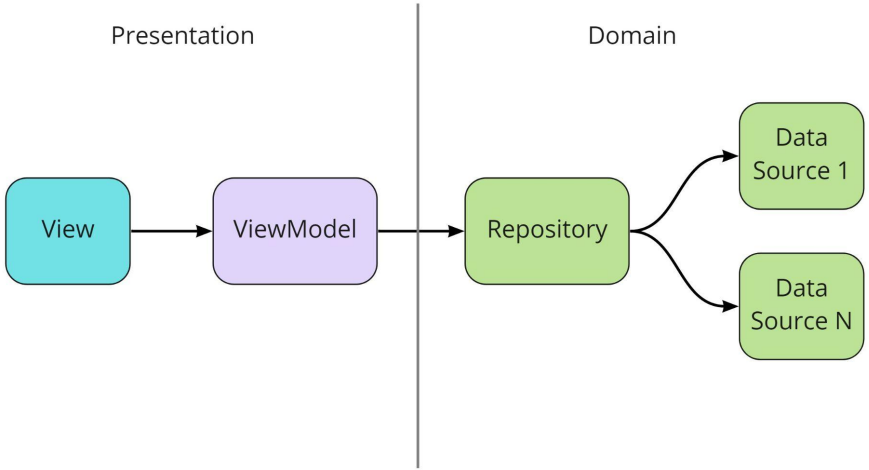
```
fun saveSessionFromOauth2(url: String): Session {
        return "imgram://oauth2.+".toRegex().matches(url).alsoIfTrue {
            val session = Session(
                accessToken = "access_token=(\\w+)".toRegex().find(url)!!.groupValues[1],
                expiresAt = "expires_in=(\\w+)".toRegex().find(url)!!.groupValues[1].toLong() +
System.currentTimeMillis(),
                tokenType = "token_type=(\\w+)".toRegex().find(url)!!.groupValues[1],
                refreshToken = "refresh_token=(\\w+)".toRegex().find(url)!!.groupValues[1],
                accountUsername = "account_username=(\\w+)".toRegex().find(url)!!.groupValues[1],
                accountId = "account_id=(\\w+)".toRegex().find(url)!!.groupValues[1]
            )
        }
    }
```

SharedPreferences

```
val sharedPrefs = getSharedPreferences("shared", Context.MODE_PRIVATE)

sharedPrefs.putString("VALOR_CLAVE", "mi valor")
val value = sharedPrefs.getString("VALOR_CLAVE")
```

Útil para guardar valores simples clave - valor

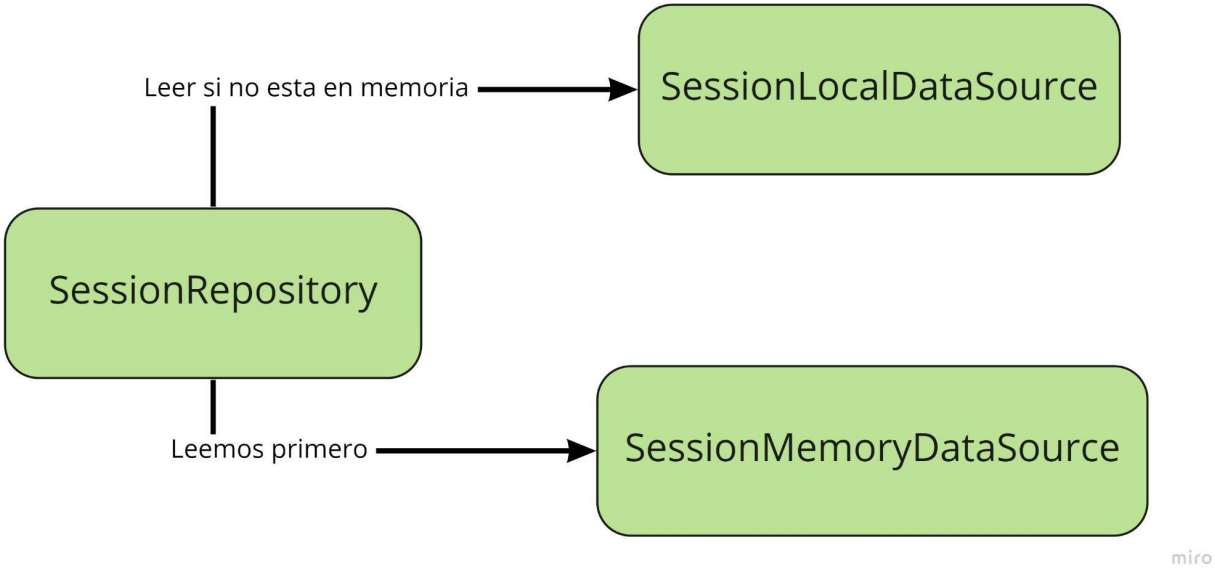
Repository pattern



Objetivo:

- Desacoplar las fuentes de datos de nuestras vistas

Store and retrieve session



Store and retrieve session

```
class HomeViewModel(
    private val imgurApi: ImgurApi,
    private val sessionRepository: SessionRepository
) {

    private val sessionMutableStateFlow = MutableStateFlow(SessionViewData.empty)
    val sessionVD: StateFlow<SessionViewData>
        get() = sessionMutableStateFlow


    fun processIntentData(uri: Uri?) {
        sessionRepository.saveSessionFromOauth2(uri.toString())
        val session = sessionRepository.getSession()
        sessionMutableStateFlow.value = SessionViewData(session != null, session?.accountUsername)
    }
}
```

Creando AuthInterceptor

```
class AuthenticatorInterceptor(private val sessionRepository: SessionRepository): Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request()
        val session = sessionRepository.getSession()

        return if (session == null) {
            chain.proceed(request)
        } else {
            val authorizedRequest = request.newBuilder()
                .header("Authorization", "${session.tokenType.capitalize()} ${session.accessToken}")
            return chain.proceed(authorizedRequest.build())
        }
    }
}
```

Retrieve logged users images

```
@GET("account/{username}/images/")
suspend fun getUsernameImages(@Path("username") username: String): ImgurResponse<List<ImgurImage>>
```

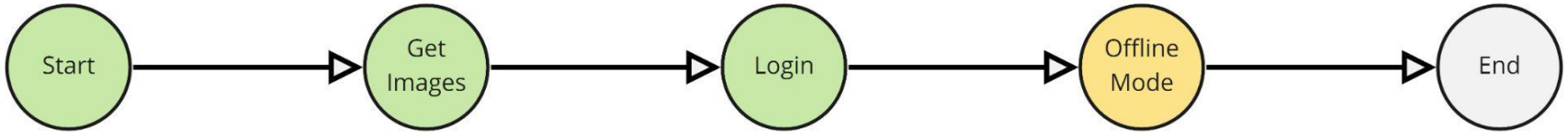
```
fun loadMyGallery() {
    viewModelScope.launch(Dispatchers.IO) {
        homeMutableStateFlow.value = HomeViewData(emptyList(), true)
        val images = imgurApi.getUsernameImages(loggedUsername)
         homeMutableStateFlow.value = HomeViewData.from(images)
    }
}
```

Cancel ongoing requests

Para cancelar una petición podemos cancelar el Job de la corrutina que invocó la petición

```
private fun fetchGallery() {
    galleryJob?.cancel()
    galleryJob = viewModelScope.launch(Dispatchers.IO) {
        homeMutableStateFlow.value = HomeViewData(emptyList(), true)
        val images = imgurApi.getTopImages()
        homeMutableStateFlow.value = HomeViewData.from(images)
    }
}
```

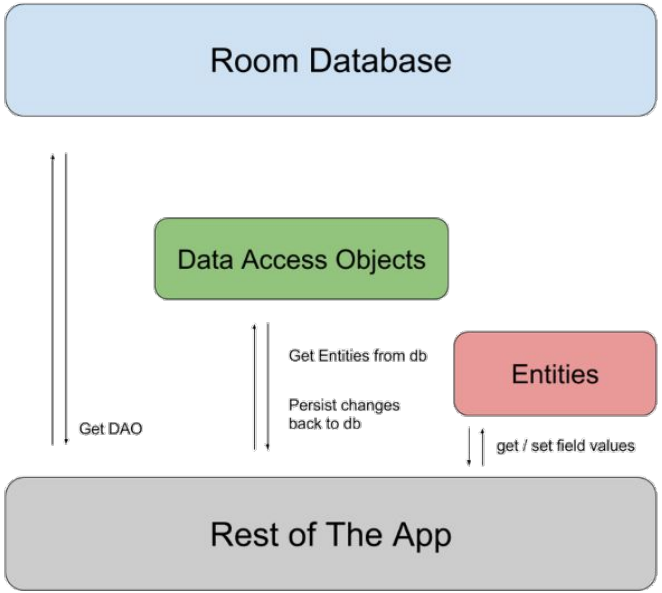
Offline Mode



Store ways

	Pros	Contras
Memoria	Rápido acceso	No es persistente
Shared preferences	Rápido acceso	Solo permite almacenar clave-valor
Room	SQL Like	“Pesado” de configurar
Data Store	Simple y rápido acceso	Util solo para datos simples no relacionales

Room



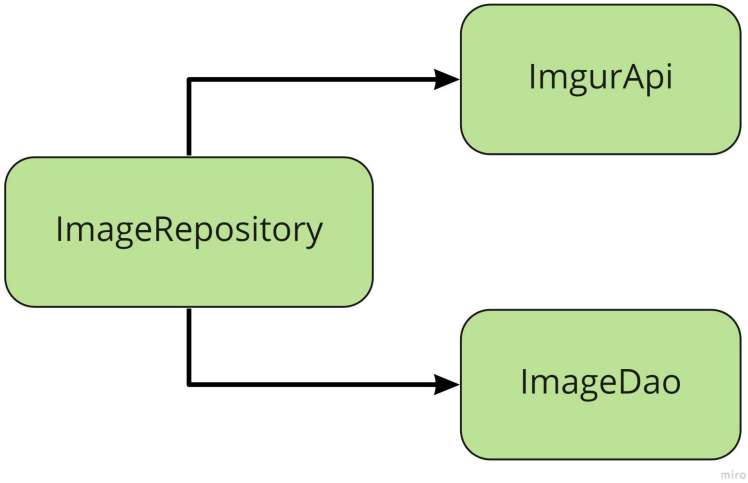
Añadir Room

```
dependencies {
  def room_version = "2.2.6"

  implementation "androidx.room:room-runtime:$room_version"
  kapt "androidx.room:room-compiler:$room_version"
  implementation "androidx.room:room-ktx:$room_version"
}
```

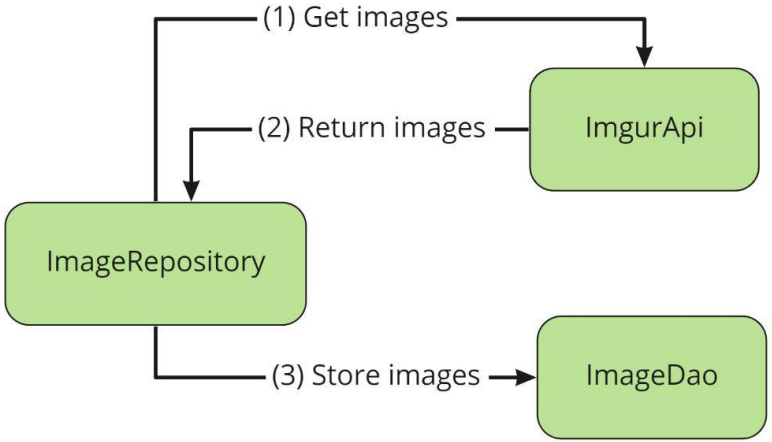
Añadir Room

```
@Database(entities = [RoomImage::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): ImageDao
}

@Dao
interface ImageDao {
    @Query("SELECT * FROM images WHERE author = :author")
   suspend fun getAll(author: String): List<RoomImage>

    @Insert(onConflict = REPLACE)
    suspend fun insertAll(images: List<RoomImage>)

    @Delete
    suspend fun delete(image: RoomImage)
}

@Entity(tableName = "images")
data class RoomImage(
    @PrimaryKey val hash: String,
    @ColumnInfo(name = "title") val title: String?,
    @ColumnInfo(name = "url") val url: String,
    @ColumnInfo(name = "likes") val likes: Int,
    @ColumnInfo(name = "datetime") val datetime: Long,
    @ColumnInfo(name = "author") val author: String?
)
```

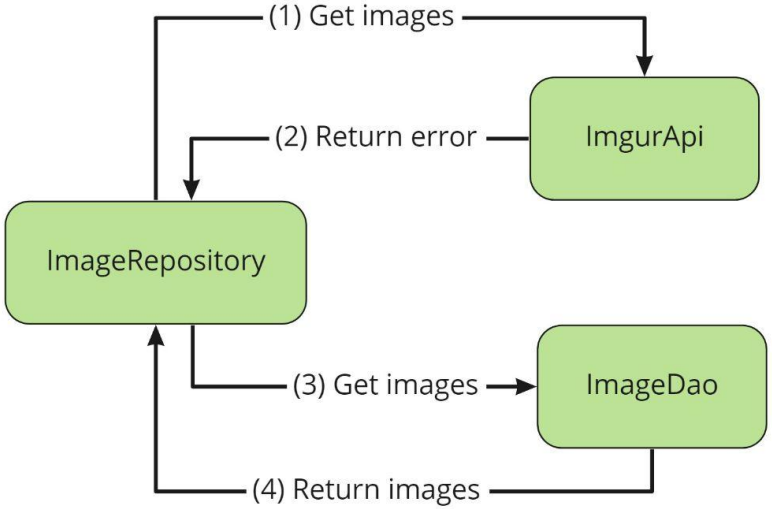
Modo Offline



Modo Offline



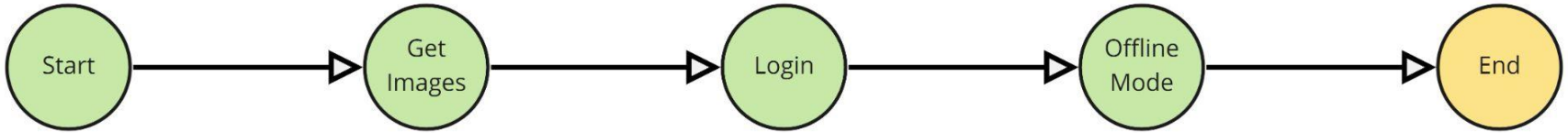
Modo online



Modo offline

miro

End



Unit testing with JUnit

```
testImplementation "org.mockito.kotlin:mockito-kotlin:3.1.0"
testImplementation 'org.mockito:mockito-inline:2.13.0'
```

Unit testing with JUnit

```
@Test
  fun `retrieve no session ok`() {
      val local = mock<SessionLocalDataSource> {
          on { getSession() }.thenReturn(null)
      }
      val memory = mock<SessionMemoryDataSource> {
          on { getSession() }.thenReturn(null)
      }

      val repository = SessionRepository(local, memory)

      assertNull(repository.getSession())
  }
```

Just the beginning