Development of an app for mobile devices with `Android`

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es

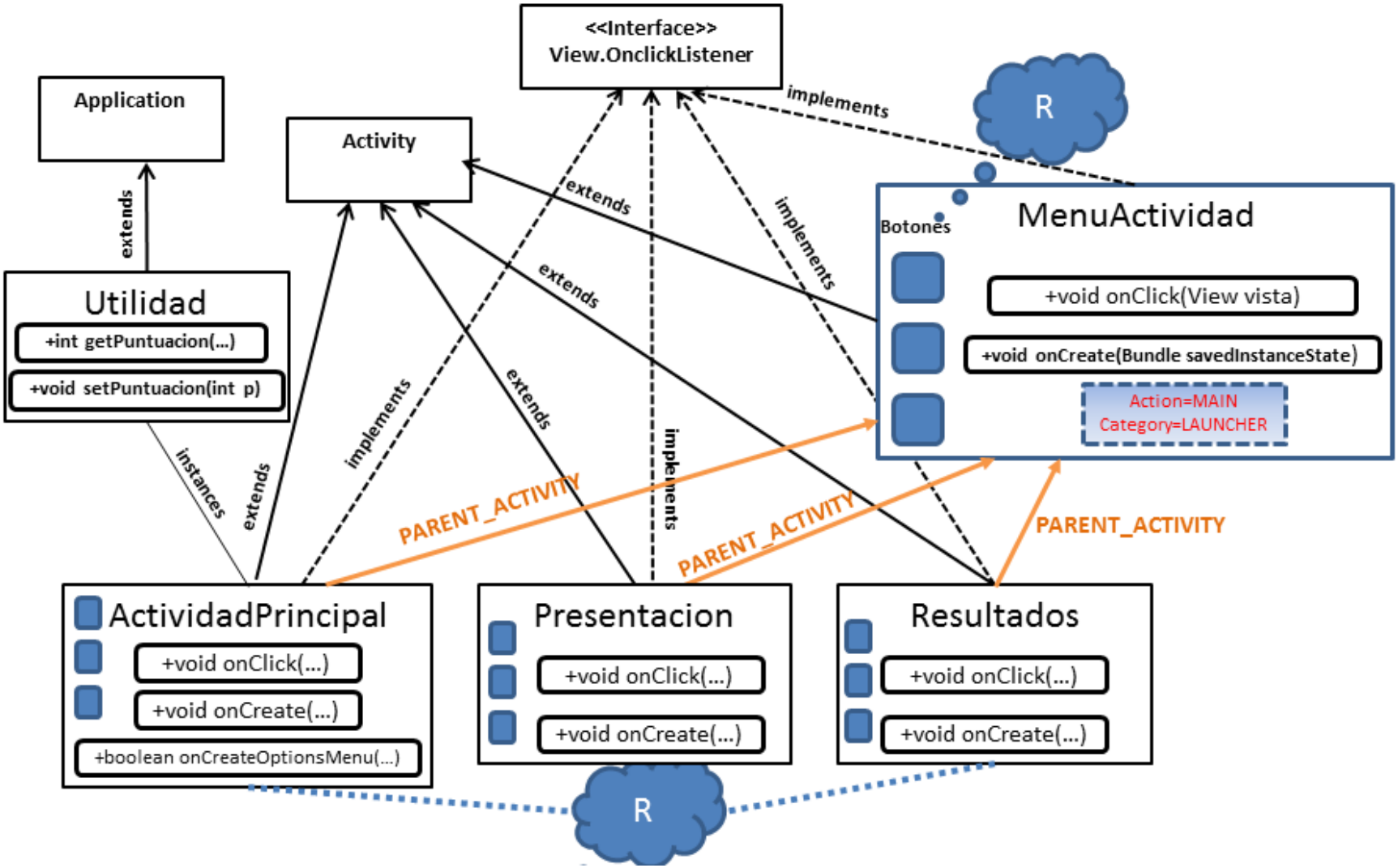
DSBCS

Máster en Ingeniería Informática

14 de noviembre de 2024

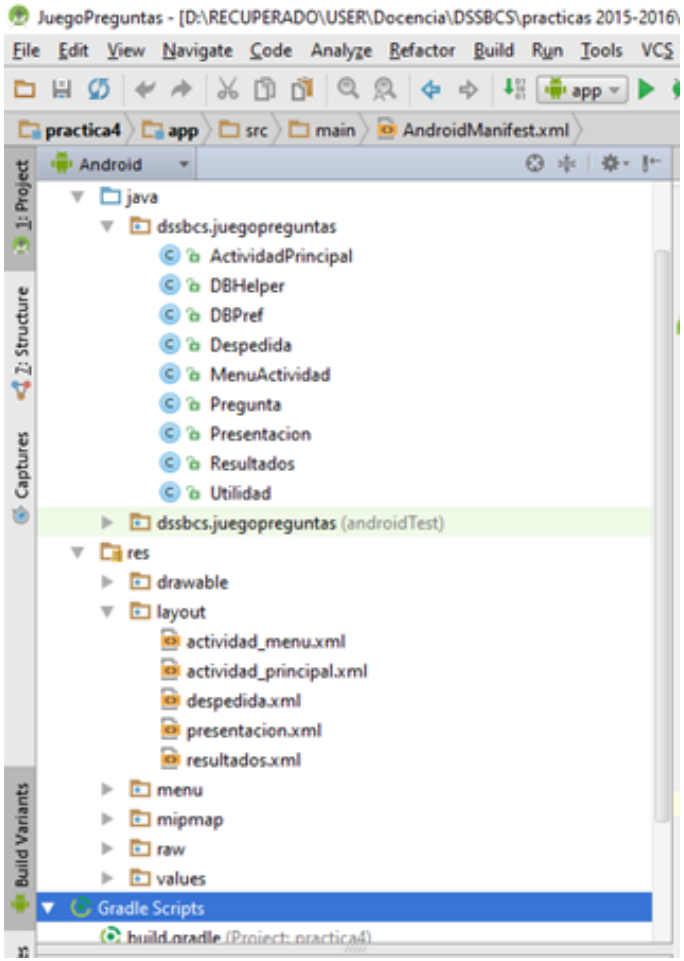
Agenda

General architecture



Possible software architecture of the app

Android project structure



Folder structure of an *Android Studio* project

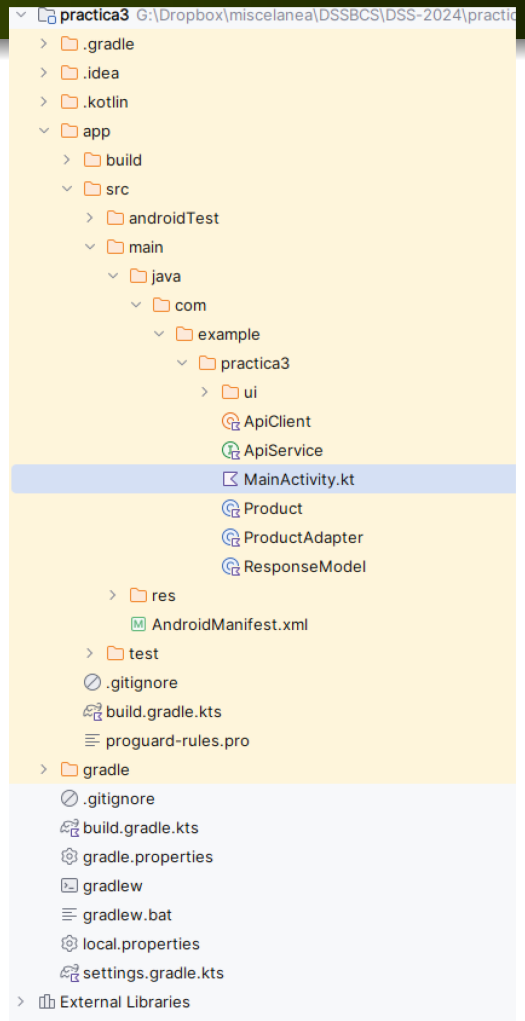
Introduction to the Practica3 Android App

- Overview of the app's purpose
- Features: Displaying products, API integration, RecyclerView, and Jetpack Compose

Technologies Used

- Android Jetpack Components
- Retrofit for API calls
- RecyclerView for UI
- Kotlin and Jetpack Compose

Project Structure



- Key files: MainActivity, ApiClient, ApiService, ProductAdapter, Product, and UI XML files

MainActivity Overview

- Primary activity, setting up the UI and API data integration

```
  class MainActivity : ComponentActivity() {
    private lateinit var productAdapter:
        ProductAdapter
    private lateinit var recyclerView: RecyclerView
    private val apiService = ApiClient.retrofit.
        create(ApiService::class.java)
    override fun onCreate(savedInstanceState: Bundle
        ?) {
        super.onCreate(savedInstanceState)
.....
    }
}
      @Composable
        fun Greeting(name: String, modifier:
            Modifier = Modifier) {
    Text(
        text = "Hello␣$name!",
        modifier = modifier
        )
```

onCreate Overview

- Lifecycle management in onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    // Set up RecyclerView
    recyclerView = findViewById(R.id.
        recyclerViewProducts)
    recyclerView.layoutManager = LinearLayoutManager
        (this)
    productAdapter = ProductAdapter(sampleProducts)
    recyclerView.adapter = productAdapter
    // Example API call
    .....
```

onCreate Overview-II

```
    .....
 // Example API call
  apiService.getAllProducts().enqueue(object : Callback<
      List<Product>> {
  override fun onResponse(call: Call<List<Product>>,
      response: Response<List<Product>>) {
  if (response.isSuccessful) {
     val data = response.body()
// Handle the response data, a List<Product>
     data?.let { productList ->
     // Process productList here
     productAdapter = ProductAdapter(productList)
     recyclerView.adapter = productAdapter
     } }else {Log.e("API_ERROR", "Error code: ${response
         .code()}")} }
   override fun onFailure(call: Call<List<Product>>, t:
       Throwable) {
      // Handle error
Log.e("API_ERROR", "Failure:${t.message}")
     }
```

Manifest and Permissions

```
    <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/
    res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:ignore="ExtraText"
    package="com.example.practica3">
    <uses-permission android:name="android.permission.
        INTERNET"
        tools:ignore="WrongManifestParent" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/
            data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Practica3"
        tools:targetApi="31">
```

Manifest and Permissions-2

```
    ....
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.Practica3">
            <intent-filter>
                <action android:name="android.intent.
                    action.MAIN" />

                <category android:name="android.intent.
                    category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Internet permission to allow network requests
- MainActivity setup with launcher intent and theme

Product Data Class

- Product model with fields: id, name, and price
- Purpose: Holds product data for display and API use

```
data class Product (
        val id: Long,
    val name: String,
    val price: Double
)
```

Retrofit API Integration

- Retrofit for making HTTP requests
- Purpose: Fetch data from backend server

ApiClient Singleton

- Configures Retrofit with base URL and Gson for JSON parsing
- Creates a retrofit instance for API calls

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
  object ApiClient {
    private const val BASE_URL = "http://10.0.2.2:8080/"
    val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create
            ())
        .build()
  }
```

Setting Base URL

- BASE_URL = "http://10.0.2.2:8080/"to access server for Android emulator

ApiService Interface

- Defines endpoints for CRUD operations
- GET and POST annotations for each endpoint

```
    import retrofit2.Call
import retrofit2.http.GET
import retrofit2.http.POST
import retrofit2.http.Path
import retrofit2.http.Query

interface ApiService {
    // Get all products
    @GET("/products")
    fun getAllProducts(): Call<List<Product>>

    // Add a product (POST request example)
    @POST("/products/add")
    fun addProduct(
        @Query("name") name: String,
        @Query("price") price: Double
    ): Call<Void>
```

Fetching Products with getAllProducts

- GET(/products") retrieves all products
- Returns Call<List<Product» for Retrofit to handle

```
import retrofit2.Call
    import retrofit2.http.GET
    import retrofit2.http.POST
    import retrofit2.http.Path
    import retrofit2.http.Query
    interface ApiService {
     // Get all products
@GET("/products")
fun getAllProducts(): Call<List<Product>>
 ..........
    }
```

Adding a Product with addProduct

- POST(/products/add") endpoint to add a new product
- Accepts name and price parameters

```
.....
// Add a product (POST request example)
@POST("/products/add")
fun addProduct(
    @Query("name") name: String,
    @Query("price") price: Double
): Call<Void>

.......
```

Editing a Product with editProduct

- POST(/products/edit/id") to modify a product by ID
- Accepts id, name, and price as parameters

```
.......
// Edit a product by ID
@POST("/products/edit/{id}")
fun editProduct(
    @Path("id") id: Long,
    @Query("name") name: String,
    @Query("price") price: Double
): Call<Void>

.........
```

Deleting a Product with deleteProduct

- POST(/products/delete/id") to remove a product by ID

```
    .........
    // Delete a product by ID
    @POST("/products/delete/{id}")
    fun deleteProduct(
        @Path("id") id: Long
    ): Call<Void>
}
```

RecyclerView Overview

- Displays a list of products using RecyclerView
- Benefits: Efficient display of large datasets

Setting Up RecyclerView in MainActivity

- Configuring recyclerView in onCreate with LinearLayoutManager
- Adapter initialization with sample data

```
// Set up RecyclerView
    recyclerView = findViewById(R.id.
        recyclerViewProducts)
    recyclerView.layoutManager = LinearLayoutManager
        (this)
    productAdapter = ProductAdapter(sampleProducts)
    recyclerView.adapter = productAdapter
```

ProductAdapter Class

- Custom adapter for RecyclerView
- Manages data binding between Product list and UI

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

class ProductAdapter(private val productList: List<
    Product>) :

RecyclerView.Adapter<ProductAdapter.ProductViewHolder>()
     {
   ..........
.........

    override fun onCreateViewHolder(parent: ViewGroup,
        viewType: Int): ProductViewHolder {
        val view = LayoutInflater.from(parent.context).
```

ProductViewHolder in ProductAdapter

- Inner class for ProductAdapter
- Holds references to item views (textViewName, textViewPrice)

```
// ViewHolder class to hold references to each items
     views
class ProductViewHolder(itemView: View) :
    RecyclerView.ViewHolder(itemView) {
    val textViewName: TextView = itemView.
        findViewById(R.id.textViewName)
    val textViewPrice: TextView = itemView.
        findViewById(R.id.textViewPrice)
}
```

Binding Data in onBindViewHolder

- Sets product name and price to each item in RecyclerView
- "{product.price}"format for price display

```
override fun onBindViewHolder(holder:
    ProductViewHolder, position: Int) {
    val product = productList[position]
    holder.textViewName.text = product.name
    holder.textViewPrice.text = "$${product.price}"
}
```

Making API Calls in MainActivity

- getAllProducts() call with Retrofit's enqueue for async processing
- onResponse: Checks for success and populates RecyclerView with product data

```
// Example API call
    apiService.getAllProducts().enqueue(object :
        Callback<List<Product>> {
        override fun onResponse(call: Call<List<
            Product>>, response: Response<List<
            Product>>) {
            if (response.isSuccessful) {
                val data = response.body()
                // Handle the response data, which
                    is a List<Product>
                data?.let { productList ->
                    // Process productList here
                    productAdapter = ProductAdapter(
                        productList)
                    recyclerView.adapter =
                        productAdapter
```

Handling API Response

- onFailure: Logs errors for debugging

```
    override fun onFailure(call: Call<List<Product>>, t:
        Throwable) {
                // Handle error
                Log.e("API_ERROR", "Failure: ${t.message
                    }")
            }
```

Error Logging with Log.e

- Logs API errors and responses for troubleshooting
- Console output for monitoring network issues

Composable Functions in the App

- Greeting composable: simple example of Jetpack Compose
- Overview of declarative UI

Jetpack Compose Preview

- @Preview annotation to preview Greeting
- Benefits of previews for UI development

Conclusion and Q&A

- Summary of app architecture and functionality
- Open floor for questions