

UNIVERSIDAD PONTIFICIA DE SALAMANCA

Máster Universitario Informática Móvil

Trabajo de la asignatura Desarrollo de aplicaciones Android

**Aplicación Android de los** **Simpsons**

Carlos Caño Gómez

Salamanca, mayo 2025

## **Índice**

Tabla de contenido

[Índice 2](#_Toc195470763)

[Desarrollo de la Aplicación 4](#_Toc195470764)

[Características Generales 4](#_Toc195470765)

[Descripción de la Aplicación 4](#_Toc195470766)

[Esquema de la Aplicación 5](#_Toc195470767)

[:app (Módulo principal) 5](#_Toc195470768)

[remote (Flavor remoto / producción) 11](#_Toc195470769)

[mock (Flavor local / test) 11](#_Toc195470770)

[test (Tests unitarios) 11](#_Toc195470771)

[androidTest (Tests instrumentados / UI / integración) 11](#_Toc195470772)

[screenshotTest  (Tests visuales por Captura) 12](#_Toc195470773)

[Otros ficheros 12](#_Toc195470774)

[Ficheros 14](#_Toc195470775)

[*MainActivity.kt* 14](#_Toc195470776)

[*TheSimpsonPlaceApp.kt* 14](#_Toc195470777)

[**Carpeta utils** 14](#_Toc195470778)

[*DateExtensions.kt* 14](#_Toc195470779)

[Logger.kt 14](#_Toc195470780)

[LocaleHelper.kt 14](#_Toc195470781)

[**Carpeta assets** 15](#_Toc195470782)

[*personajes\_data.json / personajes\_test.json* 15](#_Toc195470783)

[*imagenes\_data.json / imagenes\_test.json* 15](#_Toc195470784)

[*episodios\_data.json / episodios\_test.json* 15](#_Toc195470785)

[*citas\_test.json* 15](#_Toc195470786)

[**Carpeta res** 15](#_Toc195470787)

[Carpeta *drawable* 15](#_Toc195470788)

[Carpeta *values/strings* 15](#_Toc195470789)

[Arquitectura MVVM 16](#_Toc195470790)

[Carpeta *data* 16](#_Toc195470791)

[**1. Carpeta *daos*** 16](#_Toc195470792)

[**1.1.** **Carpeta *local*** 16](#_Toc195470793)

[**1.1.1.** **Carpeta *datastore*** 16](#_Toc195470794)

[*GameDao.kt* y *UserDao.kt* 17](#_Toc195470795)

[GameDaoImpl*.kt y* UserDaoImpl.kt 17](#_Toc195470796)

[**1.1.2.** **Carpeta *room*** 17](#_Toc195470797)

[CharacterDatabaseDao.kt, EpisodeDatabaseDao.kt, QuoteDatabaseDao.kt 17](#_Toc195470798)

[*TheSimpsonsConverters.kt* 18](#_Toc195470799)

[TheSimpsonsDatabase.kt 18](#_Toc195470800)

[***1.2.*** **Carpeta *remote*** 18](#_Toc195470801)

[CharacterDao.kt, EpisodeDao.kt, QuoteDao.kt 18](#_Toc195470802)

[**2. Carpeta *di*** 19](#_Toc195470803)

[DatabaseModule.kt 19](#_Toc195470804)

[DataModule.kt 19](#_Toc195470805)

[*DomainModule.kt* 20](#_Toc195470806)

[**3.** **Carpeta *entities*** 20](#_Toc195470807)

[Carpeta *character* 20](#_Toc195470808)

[Carpeta *episode* 20](#_Toc195470809)

[Carpeta *quote* 20](#_Toc195470810)

[Carpeta *user* 20](#_Toc195470811)

[*CharacterDTO.kt* 20](#_Toc195470812)

[*CharacterEntity.kt* 21](#_Toc195470813)

[*Gender.kt* 21](#_Toc195470814)

[*ImageDTO.kt* 21](#_Toc195470815)

[***4.*** **Carpeta *mappers*** 21](#_Toc195470816)

[Carpeta *domain* 21](#_Toc195470817)

[**1. Carpeta *mappers*** 21](#_Toc195470818)

[**2. Carpeta *models*** 22](#_Toc195470819)

[**3. Carpeta *repository*** 22](#_Toc195470820)

[*CharacterRepository.kt, etc* 22](#_Toc195470821)

[*CharacterRepositoryImpl.kt, etc* 22](#_Toc195470822)

[***2.*** **Carpeta *usecases*** 22](#_Toc195470823)

[4.1. Carpeta *character* 22](#_Toc195470824)

[4.2. Carpeta *episode* 22](#_Toc195470825)

[4.3. Carpeta *quote* 23](#_Toc195470826)

[4.4. Carpeta *game* 23](#_Toc195470827)

[4.5. Carpeta *user* 23](#_Toc195470828)

[4.6. Carpeta *impl* 23](#_Toc195470829)

[*GetAllCharactersDbUseCase.kt y GetAllCharactersDbUseCaseImpl.kt* 23](#_Toc195470830)

[Carpeta presentation 23](#_Toc195470831)

[**1. Carpeta *ui*** 24](#_Toc195470832)

[1.1.1. Carpeta *character* 24](#_Toc195470833)

[1.1.2. Carpeta *episode* 24](#_Toc195470834)

[1.1.3 Carpeta *quote* 24](#_Toc195470835)

[1.1.4. Carpeta *game* 25](#_Toc195470836)

[1.4.1. Carpeta characterSection 25](#_Toc195470837)

[1.4.2. Carpeta episodeSection 26](#_Toc195470838)

[1.4.3. Carpeta quoteSection 28](#_Toc195470839)

[1.4.4. Carpeta gameQuotes 28](#_Toc195470840)

[1.4.5. Carpeta menuSection 30](#_Toc195470841)

[1.4.6. Carpeta profileSection 30](#_Toc195470842)

[1.4.6.1. Carpeta profileEdit 31](#_Toc195470843)

[**2. Carpeta *viewmodel*** 32](#_Toc195470844)

[Módulo es.upsa.mimo.thesimpsonplace (remote) 34](#_Toc195470845)

[**Carpeta data/di** 34](#_Toc195470846)

[Módulo es.upsa.mimo.thesimpsonplace (mock) 35](#_Toc195470847)

[**Carpeta data/di** 35](#_Toc195470848)

[Módulo es.upsa.mimo.thesimpsonplace (test) 35](#_Toc195470849)

[**Carpeta** domain.usecases.impl.character 35](#_Toc195470850)

[**Carpeta** presentation.viewmodel.character 35](#_Toc195470851)

[**Subcarpeta** charactersFilterName 35](#_Toc195470852)

[**Subcarpeta** charactersListFav 35](#_Toc195470853)

[**Subcarpeta** useCasesMockFake 36](#_Toc195470854)

[**Fichero** CharactersFake.kt 36](#_Toc195470855)

[Módulo es.upsa.mimo.thesimpsonplace.presentation.ui (screenshotTest) 36](#_Toc195470856)

[**Carpeta** component.character 36](#_Toc195470857)

[**Carpeta** screen.characterSection 36](#_Toc195470858)

[Mejoras Futuras 36](#_Toc195470859)

[Conclusión 37](#_Toc195470860)

# **Desarrollo de la Aplicación**

La aplicación ha sido desarrollada por Carlos Caño Gómez. Se ha subido a Google Play con el nombre: **The Simpsons Place.**

## **Características Generales**

1. Desarrollada en **Kotlin** con **JeckPack Compose** con un diseño personalizado.

2. Basada en la **arquitectura** **MVVM** siguiendo los principios de **Clean Architecture**.

3. Realiza llamadas tanto a **mocks de datos en JSON** como a **APIs** a través de **Retrofit**.

4. Uso de inyección de dependencias automatizadas con **Hilt**.

5. Implementa persistencia de datos local con **DataStore** y **ROOM**.

6. Uso de **Coroutines** y **Flows**.

7. Ofrece soporte multilingüe mediante el archivo **Localizable**, con versiones en inglés (predeterminado), español y francés.

8. Fuentes de datos desacopladas. Uso de **variantes de gradle** combinando **Build Types** y **Flavors**.

9. Cuenta con una estrategia de **testing** y líneas **logs**, incluyendo: **tests unitarios** y **tests Snapshot.**

## **Descripción de la Aplicación**

La aplicación está completamente centrada en el universo de Los Simpsons y se estructura en cuatro secciones principales, cada una diseñada para ofrecer una experiencia completa y personalizada a los usuarios.

La primera sección está dedicada a los personajes de la serie. Aquí, los usuarios pueden explorar un extenso listado de personajes, aplicar filtros para encontrar rápidamente un personaje específico y marcar sus favoritos. Estos personajes favoritos se almacenan de manera persistente, lo que permite acceder a ellos fácilmente desde una sección específica sin necesidad de buscarlos nuevamente en la lista general.

La segunda sección se enfoca en los episodios y temporadas. Los usuarios pueden explorar el catálogo completo de episodios, visualizar información detallada de cada uno, como su año de estreno, sinopsis, guionistas, directores y apariciones especiales de celebridades. Además, es posible marcar episodios como vistos o favoritos, asegurando que esta información quede guardada de manera persistente. Para facilitar la navegación, se incluyen filtros que permiten buscar episodios por fecha de emisión, si han sido vistos, por título y otros criterios.

La tercera sección está dedicada a las citas icónicas de los personajes. Aquí, los usuarios pueden visualizar citas, generar nuevas y guardar sus favoritas para tenerlas siempre disponibles. Dentro de esta sección también se incluye un mini-juego en el que se presentan cinco citas aleatorias y el usuario debe adivinar qué personaje las dijo, eligiendo entre cuatro opciones. Al finalizar la partida, se muestran los resultados y se almacenan en un historial para llevar un registro de estadísticas de forma persistente.

Por último, en la página principal se encuentra un botón para acceder a la sección del perfil del usuario. Aquí se puede personalizar el nombre del usuario, el tema de la interfaz (en color claro u oscuro) y el lenguaje (español, inglés o francés) en la que se almacena de manera persistente, y visualizar un resumen con el recuento de personajes, episodios y citas favoritas, además de las estadísticas del mini-juego de citas.

## **Esquema de la Aplicación**

### :app (Módulo principal)

Contiene toda la lógica de negocio, presentación, navegación y dependencias principales. Organizado en capas: presentation, domain, data.

Incluye ViewModels, UseCases, UI de Jetpack Compose, DAOs, Repositorios y temas.

Contiene todos los assets y configuraciones de Gradle. Tiene soporte para Hilt, Room, Retrofit y Flavors.

* MainActivity.kt
* TheSimpsonPlaceApp.kt
* 📂 utils
* DateExtensions.kt
* Logger.kt
* LocaleHelper.kt
* 📂 data
* 📂 daos
* 📂 local
  + 📂 datastore
  + GameDao.kt
  + UserDao.kt
  + 📂 impl

- GameDaoImpl.kt

- UserDaoImpl.kt

* 📂 room
* CharacterDatabaseDao.kt
* EpisodeDatabaseDao.kt
* QuoteDatabaseDao.kt
* TheSimpsonsConverters.kt
* TheSimpsonsDatabase.kt
* 📂 remote
* CharacterDao.kt
* EpisodeDao.kt
* QuoteDao.kt
* 📂 impl
* CharacterDaoImpl.kt
* EpisodeDaoImpl.kt
* 📂 di
* DatabaseModule.kt
* DataModule.kt
* DomainModule.kt
* 📂 entities
* 📂 character
* CharacterDTO.kt
* CharacterEntity.kt
* imageDTO.kt
* Gender.kt
* 📂 episode
* EpisodesDTO.kt
* EpisodeDTO.kt
* EpisodeEntity.kt
* 📂 quote
* QuoteDTO.kt
* QuoteEntity.kt
* 📂 user
* Language.kt
* UserPreference.kt
* 📂 mappers
* CharacterDtoMapper.kt
* EpisodeDtoMapper.kt
* QuoteDtoMapper.kt
* 📂 domain
* 📂 mappers
* CharacterMapper.kt
* EpisodeMapper.kt
* QuestionMapper.kt
* QuoteMapper.kt
* 📂 models
* Character.kt
* Episode.kt
* EpisodeFilter.kt
* Question.kt
* Quote.kt
* 📂 repository
* CharacterRepository.kt
* EpisodeRepository.kt
* QuestionRepository.kt
* QuoteRepository.kt
* GameRepository.kt
* UserRepository.kt
* 📂 impl
* CharacterRepositoryImpl.kt
* EpisodeRepositoryImpl.kt
* QuestionRepositoryImpl.kt
* QuoteRepositoryImpl.kt
* GameRepositoryImpl.kt
* UserRepositoryImpl.kt
* 📂 usecases
* 📂 character
* GetAllCharactersUseCase.kt
* GetAllCharactersDbUseCase.kt
* GetCharacterDbByIdUseCase.kt
* GetFilterNameCharactersUseCase.kt
* InsertCharacterDbUseCase.kt
* DeleteCharacterDbUseCase.kt
* 📂 episode
* GetAllEpisodesUseCase.kt
* GetEpisodeByIdUseCase.kt
* GetEpisodesByTitleUseCase.kt
* GetEpisodesByDateUseCase.kt
* GetEpisodesBySeasonUseCase.kt
* GetEpisodesByChapterUseCase.kt
* GetEpisodesByViewUseCase.kt
* GetEpisodesOrderUseCase.kt
* GetAllEpisodesDbUseCase.kt
* GetEpisodeDbByIdUseCase.kt
* GetWatchedEpisodesUseCase.kt
* IsEpisodeDbWatchedUseCase.kt
* IsEpisodeDbFavoriteUseCase.kt
* InsertEpisodeDbUseCase.kt
* UpdateEpisodeDbStatusUseCase.kt
* 📂 quote
* GetQuotesUseCase.kt
* GetAllQuotesDbUseCase.kt
* GetQuoteDbByCitaUseCase.kt
* InsertQuoteDbUseCase.kt
* DeleteQuoteDbUseCase.kt
  + 📂 game
* GetGameStatsUseCase.kt
* UpdateStatsUseCase.kt
* ResetStatsUseCase.kt
* 📂 user
* GetUserPreferencesUseCase.kt
* UpdateUserUseCase.kt
  + 📂 impl
* 📂 character
* GetAllCharactersUseCaseImpl.kt
* `GetAllCharactersDbUseCaseImpl.kt
* GetCharacterDbByIdUseCaseImpl.kt
* GetFilterNameCharactersUseCaseImpl.kt
* InsertCharacterDbUseCaseImpl.kt
* DeleteCharacterDbUseCaseImpl.kt
* 📂 episode
* GetAllEpisodesUseCaseImpl.kt
* GetEpisodeByIdUseCaseImpl.kt
* GetEpisodesByTitleUseCaseImpl.kt
* GetEpisodesByDateUseCaseImpl.kt
* GetEpisodesBySeasonUseCaseImpl.kt
* GetEpisodesByChapterUseCaseImpl.kt
* GetEpisodesByViewUseCaseImpl.kt
* GetEpisodesOrderUseCaseImpl.kt
* GetAllEpisodesDbUseCaseImpl.kt
* GetEpisodeDbByIdUseCaseImpl.kt
* GetWatchedEpisodesUseCaseImpl.kt
* IsEpisodeDbWatchedUseCaseImpl.kt
* IsEpisodeDbFavoriteUseCaseImpl.kt
* InsertEpisodeDbUseCaseImpl.kt
* UpdateEpisodeDbStatusUseCaseImpl.kt
* 📂 quote
* GetQuotesUseCaseImpl.kt
* GetAllQuotesDbUseCaseImpl.kt
* GetQuoteDbByCitaUseCaseImpl.kt
* InsertQuoteDbUseCaseImpl.kt
* DeleteQuoteDbUseCaseImpl.kt
* 📂 game
* GetGameStatsUseCaseImpl.kt
* UpdateStatsUseCaseImpl.kt
* ResetStatsUseCaseImpl.kt
* 📂 user
* GetUserPreferencesUseCaseImpl.kt
* UpdateUserUseCaseImpl.kt
* 📂 presentation
* 📂 ui
  + 📂 components
* BottomBarComponent.kt
* TopBarComponent.kt
* NoContentComponent.kt
* MySearchTextField.kt
* ModifierContainer.kt
* 📂 character
* CharacterList.kt
* CharacterItem.kt
* 📂 episode
* ListEpisodes.kt
* EpisodeItem.kt
* 📂 quote
* BottomBarQuoteComponent.kt
* ListQuotes.kt
* QuoteItem.kt
* 📂 game
* HistoryGameStatistics.kt
* PieChartData.kt
* 📂 navigation
  + NavegacionApp.kt
* 📂 root
  + MyApp.kt
* 📂 screen
  + 📂 characterSection
    - CharactersScreen.kt
    - CharacterFilterScreen.kt
    - CharactersFavScreen.kt
* 📂 episodeSection
* EpisodesScreen.kt
* EpisodesFilterScreen.kt
* EpisodesFavScreen.kt
* EpisodeDetailScreen.kt
* 📂 quoteSection
* QuotesScreen.kt
* QuotesFilterScreen.kt
* QuotesFavScreen.kt
* 📂 gameQuotes
* QuotesGameScreen.kt
* QuotesQuestionScreen.kt
* QuotesResultScreen.kt
* 📂 menuSection
* MenuScreen.kt
* ItemMenuComponent.kt
* TopBarMenuComponent.kt
* 📂 profileSection
* ProfileScreen.kt
* TopBarProfileComponent.kt
* 📂 profileEdit
  + - ProfileEditScreen.kt
* 📂 theme
* Color.kt
* Theme.kt
* Type.kt
* 📂 viewmodel
* 📂 character
* 📂 charactersList
* ListCharactersViewModel.kt
* ListCharactersStateUI.kt
  + 📂 charactersFilterName
* ListCharactersFilterViewModel.kt
* ListCharactersFilterStateUI.kt
  + 📂 charactersListFav
* ListCharactersDBViewModel.kt
* ListCharactersDbStateUI.kt
* 📂 episode
* 📂 episodesList
* ListEpisodesViewModel.kt
* ListEpisodesStateUI.kt
* 📂 episodesFilterName
* ListEpisodesFilterViewModel.kt
* ListEpisodesFilterStateUI.kt
* 📂 episodesListFav
* ListEpisodesDBViewModel.kt
* ListEpisodesDbStateUI.kt
* 📂 episodeDetails
* DetailsEpisodeViewModel.kt
* DetailsEpisodeStateUI.kt
* 📂 profile
* ProfileStateUI.kt
* ProfileViewModel.kt
* 📂 quote
* 📂 quotesListFav
* ListQuotesDBViewModel.kt
* ListQuotesDbStateUI.kt
* 📂 quotesList
* ListQuotesViewModel.kt
* ListQuotesStateUI.kt
* 📂 quotesGame
* 📂 questionGame
  + QuotesGameViewModel.kt
  + QuotesGameUI.kt
* 📂 resultGame
  + ResultGameViewModel.kt
  + ResultGameUI.kt

### remote (Flavor remoto / producción)

Ruta: src/remote/java/es/upsa/mimo/thesimpsonplace/data/di/NetworkModule.kt Proporciona las dependencias para conectarse a la API real (Retrofit, OkHttp, QuoteDao, etc).

Se activa cuando se selecciona el Flavor remote.

* 📂 es.upsa.mimo.thesimpsonplace (remote)
* 📂 data
* 📂 di
* NetworkModule.kt

### mock (Flavor local / test)

Ruta: src/mock/java/es/upsa/mimo/thesimpsonplace/data/di/MockModule.kt

Proporciona los DAOs de prueba que leen de los JSON locales (citas\_test.json, etc).

Se activa cuando se selecciona el Flavor mock.

* 📂 es.upsa.mimo.thesimpsonplace (mock)
* 📂 data
* 📂 di
* MockModule.kt

### test (Tests unitarios)

Ruta: src/test/java/es/upsa/mimo/thesimpsonplace/

Incluye:

Tests de lógica de negocio (use cases)

Tests de ViewModels con Fake o Mock de base de datos

Tests de DAOs que acceden a JSON de test

* 📂 es.upsa.mimo.thesimpsonplace (test)
* 📂 data.daos.remote.impl
* CharacterDaoImplTest.kt
* 📂 domain.usescases.impl.character
* DeleteCharacterDbUseCaseImplTest.kt
* GetAllCharactersUseCaseImplTest.kt
* 📂 presentation.viewmodel.character
* 📂 charactersFilterName
* ListCharactersFilterViewModelTest.kt
* 📂 charactersListFav
* ListCharactersDBViewModelTest.kt
* 📂 useCasesMockFake
* DeleteCharacterDbUseCaseMock.kt
* GetAllCharactersDbUseCaseMock.kt
* GetCharacterDbByIdUseCaseMock.kt
* InsertCharacterDbUseCaseMock.kt
* CharactersFake.kt

### androidTest (Tests instrumentados / UI / integración)

Ruta: src/androidTest/java/es/upsa/mimo/thesimpsonplace/

Aquí irían los tests que usan un emulador o dispositivo real:

UI Tests con Espresso o Compose Test

Tests de navegación

Tests de integración con Room real

### screenshotTest  (Tests visuales por Captura)

Ruta: src/screenshotTest/java/es/upsa/mimo/thesimpsonplace/presentation/ui/

Contiene Previews renderizados como capturas de pantalla para comprobar el aspecto visual de los componentes Compose.

Se puede ejecutar con ./gradlew recordScreenshotTest o verifyScreenshotTest.

* 📂 es.upsa.mimo.thesimpsonplace.presentation.ui (screenshotTest)
* 📂 component.character
* CharacterItemScreenshotPreview.kt
* 📂 screen.characterSection
* CharactersListScreenshotPreview.kt

### Otros ficheros

* 📂 manifest
* AndroidManifest.xml
* 📂 assets
* personajes\_data.json
* personajes\_test.json
* imagenes\_data.json
* imagenes\_test.json
* episodios\_data.json
* episodios\_test.json
* citas\_test.json
* 📂 res
* 📂 drawable
* Imágenes webp de personajes de los Simpsons.
* 📂 values
* 📂 strings
* strings.xml
* strings.xml (en)
* strings.xml (es)
* strings.xml (fr)

🐘 Gradle Scripts

* build.gradle.kts (project)
* build.grade.kts (module :app)
* libs.versions.toml
* settings.gradle.kts

(Gráfica obtenida en <https://gitdiagram.com/carloscg2000/thesimpsonsplaceappandroid>)

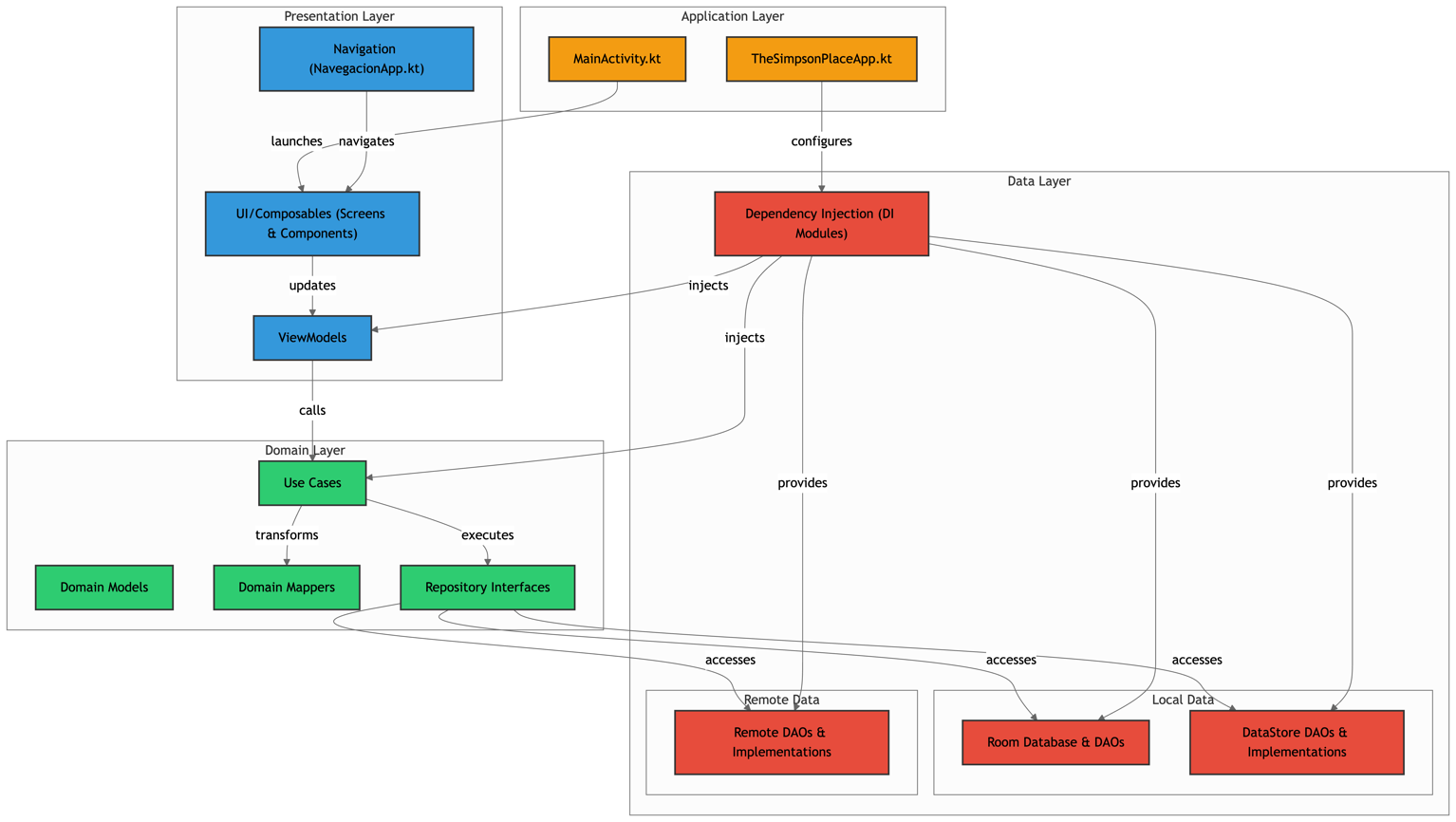


Ilustración . De GitHub a GitDiagram

## **Ficheros**

Antes de detallar cada sección de la arquitectura MVVM, se describen varios ficheros clave utilizados en la aplicación.

### *MainActivity.kt*

Este es el punto de entrada de tu aplicación en Jetpack Compose (a nivel de Android, porque es la Activity principal que se lanza cuando se inicia la aplicación).

Destacar:

• **@AndroidEntryPoint** permite la inyección de dependencias con Hilt en esta actividad.

• **enableEdgeToEdge()** configura la UI para usar toda la pantalla.

• **setContent { MyApp() }** inicia la interfaz de usuario con Compose, renderizando MyApp().

### *TheSimpsonPlaceApp.kt*

Este es el Application de la app, donde configuras Hilt y la base de datos Room.

Destacar:

• **@HiltAndroidApp** habilita Hilt para la inyección de dependencias en toda la app.

• **database** es una instancia de TheSimpsonsDatabaseRoom, accesible desde Context.database.

• **migrationVersion1toVersion2** define una migración de Room, agregando las tablas episodes y quotes.

• **initDatabase()** inicializa la BD usando Room.databaseBuilder() y registra la migración.

### **Carpeta utils**

### *DateExtensions.kt*

Extiende String con toDate(), convirtiendo un texto en formato "yyyy-MM-dd" a Date.

Extiende Date con toFormattedString(), devolviendo una fecha en "dd/MM/yyyy".

Facilita el manejo de fechas en la app.

### Logger.kt

Logger es una interfaz con métodos para registrar logs en diferentes niveles (verbose, debug, info, etc.) aplicable en la variante debug (mockDebug o remoteDebug) de la app configurado con **Build Types.**

LoggerClass la implementa, permitiendo su uso en cualquier parte de la app, si no se extiende. Útil para depuración y seguimiento del flujo de ejecución.

### LocaleHelper.kt

Se encarga del cambio de idioma de forma ordenada y eficiente.

Android 13+ (API 33): se usa LocaleManager para cambiar el idioma de forma segura y recomendada.

applicationLocales = LocaleList.forLanguageTags(...) es la forma correcta en Android 13+.

Android 12 y anteriores: se usa la forma antigua (setLocale(locale)) con updateConfiguration, pero con suppress deprecation para evitar errores de compilación.

### **Carpeta assets**

Contiene archivos de datos en formato JSON utilizados como fuente de información en la app, especialmente útil en modo offline o para testing que he realziado con Flavors

### *personajes\_data.json / personajes\_test.json*

Listado de personajes de la serie.

• data: producción (versión real).

• test: versión de prueba (por ejemplo, usada con el flavor mock).

### *imagenes\_data.json / imagenes\_test.json*

Mapeo de personajes con sus imágenes.

• data: imágenes asociadas reales.

• test: usadas para testing o desarrollo.

### *episodios\_data.json / episodios\_test.json*

Información estructurada de los episodios de la serie.

Útil para filtrar por temporada, fecha, etc.

### *citas\_test.json*

Frases célebres de personajes. Solo usado en modo test/mock.

En producción se usa una API REST externa: <https://thesimpsonsquoteapi.glitch.me/>

### **Carpeta res**

Recursos estándar utilizados por Android para construir la UI y mantener internacionalización. Contiene a la carpeta **drawable y values/strings.**

### Carpeta *drawable*

• Contiene imágenes webp de los personajes.

Usadas para ilustrar los personajes tanto en listado como en detalle.

### Carpeta *values/strings*

Archivos strings.xml para internacionalización (i18n) de la app.

• strings.xml: idioma por defecto (generalmente inglés o español).

• strings.xml (en): inglés.

• strings.xml (es): español.

• strings.xml (fr): francés.

Esto permite que la app se adapte al idioma del sistema automáticamente.

## **Arquitectura MVVM**

Para el desarrollo de mi aplicación, he optado por una arquitectura MVVM (Model-View-ViewModel) combinada con Clean Architecture. Esta elección me permite lograr un código modular, escalable y fácil de mantener, asegurando un desacoplamiento claro entre capas y facilitando la evolución del proyecto sin afectar su funcionalidad principal.

Al integrar Clean Architecture, la aplicación sigue el principio de separación de responsabilidades, donde cada capa tiene un propósito específico:

* Capa de Presentación (View + ViewModel): Gestiona la UI y la lógica de presentación sin depender de la lógica de negocio.
* Capa de Dominio (Casos de Uso): Contiene la lógica de negocio central y sigue el principio de responsabilidad única.
* Capa de Datos (Repositorios y Base de Datos): Maneja la persistencia de datos y las llamadas a la API y ficheros jsons, manteniendo independencia respecto a otras capas.

Además, esta arquitectura me ha permitido:

* Facilitar el testing, ya que cada capa es independiente y puede probarse por separado.
* Reutilizar código y mejorar la escalabilidad, permitiendo cambios o mejoras sin afectar otras partes de la aplicación.
* Aplicar principios SOLID, especialmente la inversión de dependencias mediante protocolos, para desacoplar implementaciones concretas.

## Carpeta *data*

Esta capa actúa como puente entre el dominio y las fuentes de datos (API, JSON, Room, DataStore). Su estructura modular garantiza una separación clara entre la obtención de datos remotos, locales y la lógica de persistencia.

### **1. Carpeta *daos***

La carpeta daos (Data Access Objects) contiene las clases e interfaces encargadas de acceder a los datos, ya sea desde una base de datos local o una fuente remota para aislar la lógica de acceso a los datos del resto de la aplicación.

Se encuentra a su vez la carpeta local y remote.

### **Carpeta *local***

Contiene todo lo necesario para acceder y gestionar datos almacenados localmente en el dispositivo del usuario.

Se compone a su vez de la carpeta **datastore** y **room**.

### **Carpeta *datastore***

Para datos clave-valor persistentes (preferencias del usuario, estadísticas de juego, etc.).

Se encuentra los ficheros **GameDao.kt** y **UserDao.kt** y la carpeta **impl** con los ficheros a su vez **GameDaoImpl.kt** y **UserDaoImpl.kt**

### *GameDao.kt* y *UserDao.kt*

Son las interfaces DAO para DataStore.

Definen las operaciones para leer/escribir datos de preferencias del usuario (UserDao) y estadísticas de juego (GameDao). Usan **Flow<>** para obtener cambios en tiempo real.

**updateUser() y updateStats()** para actualizar datos en DataStore.

### GameDaoImpl*.kt y* UserDaoImpl.kt

**GameDaoImpl** es la implementación de GameDao. Usa **DataStore<Preferences>** para guardar estadísticas de juego (aciertos y preguntas respondidas).

El **gameStatsFlow** es el flujo que devuelve un **Pair<Int, Int>** con aciertos y preguntas.

La función **updateStats()** suma nuevos valores a los existentes.

La función **resetStats()** elimina los datos almacenados.

**UserDaoImpl** es la implementación de UserDao. Usa **DataStore<Preferences>** para guardar datos del usuario (nombre, tema oscuro, idioma).

La función **userPreferencesFlow** tiene el flujo que devuelve un UserPreference.

La función **updateUser()** guarda los nuevos valores en DataStore.

Utilizo dataStore en lugar de SharedPreferences porque es más moderno, seguro y eficiente. Lo implemento con Hilt (@Inject) para inyección de dependencias. Uso de edit {} en DataStore para modificar datos atómicamente.

### **Carpeta *room***

A diferencia Datastore se usa para entidades complejas persistidas en una base de datos SQLite mediante Room. Implementa la lógica de acceso a una base de datos relacional local mediante Room (una abstracción de SQLite en Android).

Se encuentra los ficheros **CharacterDatabaseDao.kt, EpisodeDatabaseDao.kt, QuoteDatabaseDao.kt, TheSimpsonsConverters.kt, TheSimpsonsDatabase.kt**.

### CharacterDatabaseDao.kt, EpisodeDatabaseDao.kt, QuoteDatabaseDao.kt

Misma explicación en los otros ficheros para Episode (EpisodeDatabaseDao) y Quote (QuoteDatabaseDao), cambiando solo su contexto.

**CharacterDatabaseDao** se define las operaciones CRUD en **Room** para **CharacterEntity**.

Usa **@Query** para obtener todos los personajes o buscar por ID.

Usa **Flow<List<CharacterEntity>>** para obtener actualizaciones en tiempo real.

**@Insert(onConflict = OnConflictStrategy.REPLACE)** inserta o actualiza si ya existe.

**@Delete** elimina un personaje de la BD.

### *TheSimpsonsConverters.kt*

Contiene unos conversores de tipos para Room.

Convierte tipos complejos (que Room no maneja nativamente) en tipos primitivos:

• Date ↔ Long

• List<String> ↔ String (JSON)

• URL ↔ String

### TheSimpsonsDatabase.kt

Es la base de datos de Room.

Usa **@Database** para definir las entidades (CharacterEntity, EpisodeEntity, QuoteEntity).

Tiene la versión 2 de la BD esto indica indica que ha habido un cambio en la estructura (se añadieron las entiendades de Episodes y Quotes).

**@TypeConverters(TheSimpsonsConverters::class)** habilita los conversores para manejar tipos complejos.

**abstract fun characterDbDao()** expone los DAOs para interactuar con la BD.

### **Carpeta *remote***

En esta carpeta se encuentran los DAO que acceden a fuentes de datos remotas, como archivos JSON en assets o llamadas a APIs a través de Retrofit. El nombre remote hace referencia a que los datos no residen en el dispositivo, sino que provienen de internet (API REST) o archivos externos distribuidos con la app. Esta separación facilita desacoplar la lógica de acceso a datos locales y remotos y permite alternar entre ambas según el tipo de Build Type o Flavor.

Compuesto por tres ficheros **CharacterDao.kt, EpisodeDao.kt, QuoteDao.kt** y una carpeta **impl** con dos ficheros a su vez **CharacterDaoImpl.kt** y **EpisodeDaoImpl.kt**.

### CharacterDao.kt, EpisodeDao.kt, QuoteDao.kt

Son las Interfaces DAO. Estas interfaces definen las funciones para obtener datos remotos (desde una API o un archivo JSON en assets (según el Flavor usado)).

**CharacterDao.kt** contiene:

• **getAllCharacters()** retorna todos los personajes disponibles.

• **getCharactersByName(name: String)** filtra personajes por nombre.

**EpisodeDao.kt** contiene:

• **getAllEpisodes()** obtiene todos los episodios.

• **getEpisodeById(id: String)** Busca un episodio por ID.

• **getEpisodesByTitle(title: String)** filtra episodios por título.

• **getEpisodesByDate(minDate: Date?, maxDate: Date?)** filtra episodios por fecha.

• **getEpisodesBySeason(season: Int)** filtra por temporada.

• **getEpisodesByChapter(chapter: Int**) filtra por capítulo.

**QuoteDao.kt** contiene:

• Usa Retrofit para obtener frases de la API externa.

• **getQuotes(numElementos, textPersonaje)** hace una petición HTTP a https://thesimpsonsquoteapi.glitch.me/quotes.

**• @GET("/quotes")** define el endpoint.

**• @Query("count")** es el número de frases a recuperar.

**• @Query("character")** opcionalmente, filtra por personaje.

**CharacterDaoImpl.kt** y **EpisodeDaoImpl.kt** usa JSONs en assets en lugar de API.

Recibe el Context y los nombres de los archivos JSON (dataJson, imageJson).

• Usa **Kotlinx Serialization (Json { ignoreUnknownKeys = true })** para parsear el JSON (en vez de Gson).

### **2. Carpeta *di***

La carpeta di agrupa todos los módulos de inyección de dependencias utilizando Hilt.

DI (Dependency Injection) permite desacoplar la creación de objetos de su uso, facilitando la reutilización, testeo y mantenimiento del código.

Aquí se definen los proveedores **(@Provides, @Binds**) para: bases de datos locales (Room y Datastore), repositorios, casos de uso, Retrofit/API remota (en remote) o JSON (en mock).

Se compone de 3 ficheros **DatabaseModule.kt, DataModule.kt, DomainModule.kt**. Antes también contenia **NetworkModule.kt** (borrado y pasado para usar en **es.upsa.mimo.thesimpsonplace (remote)** o **es.upsa.mimo.thesimpsonplace (mock),** debido al uso de **Flavors** definido en **Gradle Scripts**).

### DatabaseModule.kt

Provee DAOs de **Room** y **DataStore**. Este módulo se encarga de proporcionar los DAOs de Room, permitiendo la interacción con la base de datos. Y la inyección de Room y la DataStore.

Funciones **@Provides** en DAOs:

• **provideCharacterDatabaseDao**() retorna **CharacterDatabaseDao**.

• **provideEpisodeDatabaseDao()** retorna **EpisodeDatabaseDao**.

• **provideQuoteDatabaseDao(**) retorna **QuoteDatabaseDao**.

**TheSimpsonsDatabaseRoom** para Room formada por:

• **provideDatabaseRoom()** que es una función **@Provides** de **Room Database** asegura que Room solo tenga una instancia de la base de datos.

**GameDataStore** y **UserDataStore** para **DataStore** que contiene Qualifiers:

• Se usa **@Qualifier** para diferenciar los **DataStore<Preferences>** de “**game**” y “**user**” permitiendo inyectar dependencias sin que haya conflicto entre ambas instancias.

### DataModule.kt

Fuentes de **datos**. Este módulo proporciona **CharacterDao** y **EpisodeDao**, permitiendo elegir entre modo producción o test con archivos JSON.

Permite cambiar entre archivos de producción y prueba sin modificar el código en otros lugares, utilizo **Flavors** para cambiar la variable para test de producción o de testing.

*DomainModule.kt*

Contiene **los casos de uso y repositorios**.

Este módulo proporciona los repositorios y casos de uso mediante **@Binds**.

**@Binds** se usa en lugar de **@Provides** cuando ya existe una implementación de una interfaz (con **@Inject constructor**). Separa el acceso a datos del dominio, mejorando la escalabilidad y mantenibilidad.

**@Singleton** se usa en los repositorios (repository) para mantener una única instancia de **CharacterRepositoryImpl**, etc.

### **Carpeta *entities***

La carpeta entities almacena todas las clases que representan modelos de datos en sus distintas formas:

• **DTOs** (Data Transfer Objects) estructuras que representan datos tal y como llegan desde una API o JSON (CharacterDTO, EpisodeDTO, QuoteDTO, etc.)

• **Entities**: Estructuras que se usan para guardar datos en la base de datos local Room (CharacterEntity, EpisodeEntity, QuoteEntity).

• **Otros**: Elementos auxiliares como enumeraciones (Gender.kt) o clases de imagen (ImageDTO.kt).

Está organizada por módulos funcionales (**character, episode, quote, user**) para mantener una separación lógica del dominio de cada tipo de dato.

### Carpeta *character*

Contiene los ficheros **CharacterDTO.kt, CharacterEntity.kt, imageDTO.kt, Gender.kt.**

### Carpeta *episode*

Contiene los ficheros **EpisodesDTO.kt, EpisodeDTO.kt, EpisodeEntity.kt**

### Carpeta *quote*

Contiene los ficheros **QuoteDTO.kt y QuoteEntity.kt**

### Carpeta *user*

Contiene los ficheros **Language.kt y UserPreference.kt**

Partimos de solo un ejemplo el de Character (ya que los demás son lógicamente muy parecidos).

*CharacterDTO.kt*

Representa la estructura de los datos que llegan desde la API. Uso **@Serializable** para poder deserializar JSON.

**@SerialName("campo\_api")** permite mapear nombres de la API a nombres más manejables en el código. Contiene una función **getIdAsInt()** para asegurar que id se convierta a Int.

*CharacterEntity.kt*

Entidad para Room. Es el modelo que se almacena en la base de datos local.

Usa **@Entity** para definir la tabla en Room. Define **TABLE\_NAME** en un **companion object** para centralizar el nombre. Usa **@PrimaryKey(autoGenerate = false)** para definir la clave primaria.

*Gender.kt*

Enumerado para género. Define los valores posibles de género (Male, Female, Undefined). Tiene una función **fromAbbreviation()** para mapear los valores de la API ("m" y "f") al enum.

*ImageDTO.kt*

Representa imágenes asociadas a un personaje de uno de los JSONs.

Usa **@Serializable** y **@SerialName** para definir el mapeo JSON.

### **Carpeta *mappers***

Contiene estos archivos **CharacterDtoMapper.kt, EpisodeDtoMapper.kt, QuoteDtoMapper.kt.**

Estos archivos contienen funciones de extensión para convertir entre diferentes modelos de datos:

• DTO → Modelo de dominio (datos de la API/JSON a la app).

• ROOM → Modelo de dominio (datos almacenados localmente a la app).

## Carpeta *domain*

La capa domain representa el corazón de la aplicación. Define qué hace la app (reglas de negocio, lógica, casos de uso), sin preocuparse por cómo ni desde dónde se obtienen los datos (API, BD, etc.).

### **1. Carpeta *mappers***

Estos mapeadores (mappers) convierten datos entre diferentes modelos de la aplicación. Son utilizables en diferentes capas del proyecto (BD, lógica de negocio y UI).

• De DTO (API) a Model (Dominio).

• De Model a Entity (Room).

**• toCharacterDb(), toQuoteDb()** – Convierte un modelo de dominio a formato guardable en Room.

• **toQuestion()** – Genera una pregunta desde una cita.

• **getRandomOptions()** – Devuelve opciones aleatorias incorrectas para el juego.

• ... demás funciones de mapeo.

Contiene los ficheros **CharacterMapper.kt, EpisodeMapper.kt, QuestionMapper.kt, QuoteMapper.kt.**

### **2. Carpeta *models***

Define los modelos de datos independientes de la infraestructura (la base de datos (Entity) o de la API/JSON (DTO)). Aquí están las entidades “puras” que usa la lógica de negocio y la UI. Las clases de dominio, que representan los datos tal y como los usa la lógica de negocio de la aplicación. No dependen de Room, Retrofit ni Compose.

Contiene los ficheros **Character.kt, Episode.kt, EpisodeFilter.kt, Question.kt, Quote.kt**

### **3. Carpeta *repository***

Aplica el patrón Repository, que desacopla la lógica de negocio del acceso a datos.

Ficheros que contiene **CharacterRepository.kt, EpisodeRepository.kt, QuestionRepository.kt, QuoteRepository.kt, GameRepository.kt, UserRepository.kt** y la carpeta **impl**.

La carpeta **impl** a su vez contiene **CharacterRepositoryImpl.kt, EpisodeRepositoryImpl.kt, QuestionRepositoryImpl.kt,**

**QuoteRepositoryImpl.kt, GameRepositoryImpl.kt, UserRepositoryImpl.kt.**

El patrón Repository para separar la lógica de acceso a datos y proporcionar una abstracción sobre las fuentes de datos.

Vere solo el **Character**:

*CharacterRepository.kt, etc*

Son interfaces quedefine los métodos que la capa de dominio puede usar para acceder a los datos. No sabe de dónde vienen los datos, solo define las operaciones disponibles.

*CharacterRepositoryImpl.kt, etc*

Implementa la interfaz usando CharacterDao (fuente remota) y CharacterDatabaseDao (base de datos local). Se ejecuta en Dispatchers.IO para evitar bloquear el hilo principal. Usa Flow para emitir cambios en la BD en tiempo real.

### **Carpeta *usecases***

Contiene la lógica de aplicación dividida en casos de uso específicos.

Contiene las carpetas **character, episode, quote, game, user, impl.**

4.1. Carpeta *character*

Contiene los ficheros GetAllCharactersUseCase.kt, GetAllCharactersDbUseCase.kt, GetCharacterDbByIdUseCase.kt, GetFilterNameCharactersUseCase.kt, InsertCharacterDbUseCase.kt, DeleteCharacterDbUseCase.kt

4.2. Carpeta *episode*

La carpeta episode contiene los ficheros GetAllEpisodesUseCase.kt, GetEpisodeByIdUseCase.kt, GetEpisodesByTitleUseCase.kt, GetEpisodesByDateUseCase.kt, GetEpisodesBySeasonUseCase.kt, GetEpisodesByChapterUseCase.kt, GetEpisodesByViewUseCase.kt, GetEpisodesOrderUseCase.kt, GetAllEpisodesDbUseCase.kt, GetEpisodeDbByIdUseCase.kt, GetWatchedEpisodesUseCase.kt, IsEpisodeDbWatchedUseCase.kt, IsEpisodeDbFavoriteUseCase.kt, InsertEpisodeDbUseCase.kt, UpdateEpisodeDbStatusUseCase.kt.

4.3. Carpeta *quote*

La carpeta quote contiene GetQuotesUseCase.kt, GetAllQuotesDbUseCase.kt,GetQuoteDbByCitaUseCase.kt, InsertQuoteDbUseCase.kt, DeleteQuoteDbUseCase.kt.

4.4. Carpeta *game*

La carpeta game contiene GetGameStatsUseCase.kt, UpdateStatsUseCase.kt, ResetStatsUseCase.kt.

4.5. Carpeta *user*

La carpeta user contiene GetUserPreferencesUseCase.kt, UpdateUserUseCase.kt

4.6. Carpeta *impl*

La carpeta impl contiene los mismos archivos nombrados antes implementado casos de uso para dichas interfaces.

Estoy aplicando el principio de separación de responsabilidades (SRP) dentro de la capa de dominio al introducir casos de uso (Use Cases). Estos encapsulan reglas de negocio y evitan que la UI acceda directamente a los repositorios.

Ejemplo con Character.

*GetAllCharactersDbUseCase.kt y GetAllCharactersDbUseCaseImpl.kt*

Define un contrato que dice que el caso de uso debe devolver un **Flow<List<Character>>.** La UI solo usa esta interfaz, sin saber qué repositorio usa internamente.

La implementación (Impl) simplemente delega la llamada al repositorio. Usa inyección de dependencias **(@Inject constructor**) para recibir el CharacterRepository.

Gracias a esto hay mayor separación de responsabilidades. La capa de dominio se encarga solo de la lógica de negocio. La capa de datos (repositorios) gestiona el acceso a la API y BD.

Adicionalmente esto facilita las pruebas unitarias, los usecases se pueden testear individualmente con mocks de Repository y evita dependencias directas en la UI, si la UI (o un ViewModel) necesita personajes de la BD, usa GetAllCharactersDbUseCase, sin tocar directamente el repositorio.

## Carpeta presentation

### **1. Carpeta *ui***

Esta carpeta se encarga de la interfaz de usuario de la aplicación. Está estructurada para seguir buenas prácticas de diseño en Jetpack Compose, dividiendo responsabilidades entre componentes reutilizables, pantallas y navegación.

#### **Carpeta components**

Contiene componentes visuales reutilizables que se emplean en distintas pantallas de la app.

Ficheros **BottomBarComponent.kt, TopBarComponent.kt, NoContentComponent.kt,**

**MySearchTextField.kt, ModifierContainer.kt** y carpetas **character, episode, quote, game.**

• **BottomBarComponent.kt**: barra inferior de navegación principal, permite moverse entre secciones como personajes, episodios y citas.

• **TopBarComponent.kt**: barra superior con título y botón de navegación (flecha atrás).

• **NoContentComponent.kt**: componente que se muestra cuando no hay resultados (por ejemplo, al aplicar filtros sin coincidencias).

• **MySearchTextField.kt**: textField personalizado para búsquedas con diseño consistente.

• **ModifierContainer.kt**: wrapper de Modifier reutilizable para aplicar paddings o estilos comunes a las pantallas.

1.1.1. Carpeta *character*

Componentes específicos para representar personajes.

Con los ficheros **CharacterList.kt y CharacterItem.kt**

• **CharacterList.kt**: lista vertical de personajes renderizados con Compose.

• **CharacterItem.kt**: ficha individual de personaje con imagen, nombre y botón de favorito.

1.1.2. Carpeta *episode*

Componentes específicos para la sección de episodios.

Con los ficheros **ListEpisodes.kt y EpisodeItem.kt**

• **ListEpisodes.kt:** lista de episodios con soporte para scroll, filtros y favoritos.

• **EpisodeItem.kt:** representación visual de un episodio individual con información básica.

1.1.3 Carpeta *quote*

Componentes visuales usados para la gestión de citas.

Con los ficheros **BottomBarQuoteComponent.kt, ListQuotes.kt y QuoteItem.kt**

• **BottomBarQuoteComponent.kt**: barra de navegación inferior específica del juego de citas.

• **ListQuotes.kt:** lista de frases célebres de los personajes.

• **QuoteItem.kt**: componente que representa una cita con opciones como favorito o jugar.

1.1.4. Carpeta *game*

Visualización y resumen de estadísticas del juego de citas.

Con los ficheros **HistoryGameStatistics.kt y PieChartData.kt**

• **HistoryGameStatistics.kt**: muestra los aciertos y errores pasados del usuario en formato gráfica.

• **PieChartData.kt**: componente de tipo gráfico circular que representa visualmente los resultados del juego.

#### **1.2. Carpeta navigation**

Define todas las rutas y composables que representan las pantallas principales.

Actúa como el router central de la app.

Contiene el fichero **NavegacionApp.kt**. Este archivo gestiona la navegación de la app utilizando NavHost y NavController.

Se estructura en:

• Clase **Screen** que define las rutas de la app con **sealed class Screen**, permitiendo pasar parámetros en las rutas (ej. EpisodeDetail(id)).

• **NavHost** con **NavController** donde se define 16 pantallas en total y sus transiciones.

#### **1.3. Carpeta root**

Esta el fichero **MyApp.kt** que contiene el punto de entrada de la aplicación (a nivel de la UI en Jetpack Compose, el primer Composable que se ejecuta cuando la aplicación se carga).

Obtiene las preferencias del usuario (userState) desde **ProfileViewModel**.

Aplica el idioma seleccionado con **LocaleHelper.updateLocale().**

Configura el tema oscuro/claro con **TheSimpsonPlaceTheme**.

Inicia la navegación con **NavegacionApp().**

#### **1.4. Carpeta screen**

Esta carpeta organiza todas las pantallas de la aplicación, agrupadas por funcionalidad o módulo (personajes, episodios, citas, juego, menú, perfil…). Cada subcarpeta contiene los Compose Screens responsables de renderizar la UI para esa funcionalidad específica.

Contiene las carpetas **characterSection, episodeSection, quoteSection, gameQuotes, menuSection, profileSection.**

### 1.4.1. Carpeta characterSection



Ilustración . Contiene las pantallas de personajes

• **CharactersScreen.kt**: muestra el listado completo de personajes. Permite marcarlos como favoritos.

• **CharacterFilterScreen.kt**: pantalla para buscar personajes por nombre.

• **CharactersFavScreen.kt**: pantalla que muestra únicamente los personajes favoritos guardados en la base de datos local.

### 1.4.2. Carpeta episodeSection



Ilustración . Pantallas de episodios 1



Ilustración . Pantallas de episodios 2

• **EpisodesScreen.kt**: lista completa de episodios disponibles.

• **EpisodesFilterScreen.kt**: permite aplicar filtros avanzados por título, fecha, temporada, vistos, etc.

• **EpisodesFavScreen.kt**: muestra únicamente los episodios marcados como favoritos por el usuario.

**• EpisodeDetailScreen.kt:** pantalla detallada de un episodio seleccionado, mostrando toda su información.

### 1.4.3. Carpeta quoteSection



Ilustración . Pantallas de citas

**• QuotesScreen.kt**: pantalla principal que muestra citas célebres aleatorias.

**• QuotesFilterScreen.kt**: permite filtrar las citas por nombre del personaje.

**• QuotesFavScreen.kt**: muestra las citas guardadas como favoritas en la base de datos local.

### 1.4.4. Carpeta gameQuotes

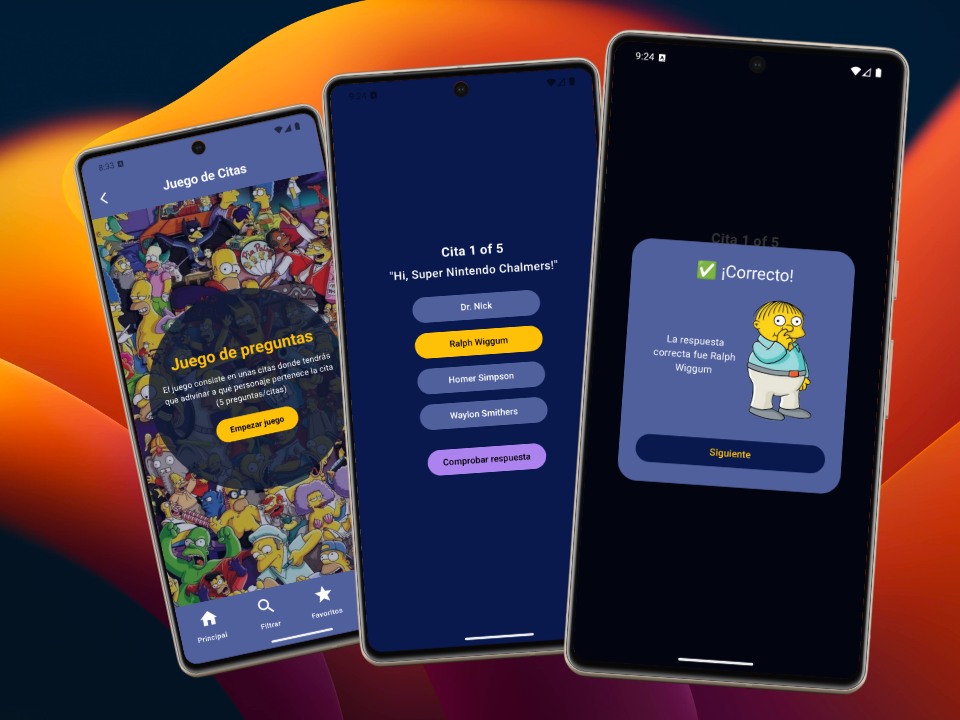


Ilustración . Pantallas del juego de citas 1



Ilustración . Pantallas del juego de citas 2

• **QuotesGameScreen.kt**: pantalla de bienvenida al juego de frases: explicación y botón para comenzar.

• **QuotesQuestionScreen.kt**: muestra cada pregunta del juego con la frase y opciones de personajes.

• **QuotesResultScreen.kt**: resultados finales del juego: número de aciertos y fallos + estadísticas.

### 1.4.5. Carpeta menuSection



Ilustración . Pantalla de menú

• **MenuScreen.kt**: pantalla principal del menú con accesos rápidos a secciones.

• **ItemMenuComponent.kt**: componente que representa cada item del menú (botones de acceso).

• **TopBarMenuComponent.kt**: barra superior personalizada para esta sección.

### 1.4.6. Carpeta profileSection



Ilustración . Pantalla de perfil

• **ProfileScreen.kt:** muestra la información del usuario actual (idioma, estadísticas, etc.).

• **TopBarProfileComponent.kt**: barra superior del perfil con diseño personalizado.

### 1.4.6.1. Carpeta profileEdit

**• ProfileEditScreen.kt:** permite al usuario cambiar idioma, restablecer estadísticas, etc.

#### **1.5. Carpeta theme**

Esta carpeta contiene los ficheros encargados de centralizar los colores, tipografías y estilos de la interfaz de usuario en Jetpack Compose. Todo lo relacionado con la apariencia (tema claro/oscuro, paleta de colores, fuentes…) está aquí.

**• Color.kt:** define la paleta de colores personalizada usada en la app.

**•** Type.kt: define la tipografía global usada en los textos de la app.

• Theme.kt: es la pieza clave que conecta Color.kt y Type.kt. Es el fichero que se llama en MainActivity (MyApp()) para aplicar los colores y tipografías definidos al resto de composables de la app.

Contiene:

• Configuración de MaterialTheme.

• Soporte para tema claro u oscuro automático con **isSystemInDarkTheme().**

• Asignación de colors, typography, shapes.

### **2. Carpeta *viewmodel***

Aquí se concentran todos los ViewModels y estados UI usados en la capa de presentación. Cada funcionalidad tiene su propia carpeta para mantener la separación por dominio de forma clara y escalable.

Esta parte de la arquitectura se encarga de manejar la lógica de presentación para la pantalla de lista de personajes, episodios, citas, juego de citas y perfil de usuario.

Uso StateFlow y update {} en lugar de modificar valores directamente. Separo el estado y la lógica, con StateUI solo almacena datos y el ViewModel los gestiona.

Y por ejecuto llamadas a los jsons, API o base de datos dentro de **viewModelScope.launch** para evitar bloqueos.

#### **2.1. Carpeta character**

Donde se realiza la gestión de personajes.

#### 2.1.1. Carpeta charactersList

ViewModel y estado para mostrar todos los personajes.

• **ListCharactersViewModel.kt**: carga los personajes desde el repositorio remoto y expone el estado de carga.

• **ListCharactersStateUI.kt**: modelo de estado que contiene la lista de personajes y bandera de carga (isLoading).

#### 2.1.2. Carpeta charactersFilterName

Lógica para filtrar personajes por nombre.

• **ListCharactersFilterViewModel.kt**: gestiona el filtro de búsqueda por nombre. Aplica el uso del caso de uso GetFilterNameCharactersUseCase.

• **ListCharactersFilterStateUI.kt**: contiene el estado del filtro aplicado y los resultados obtenidos.

#### 2.1.3. Carpeta charactersListFav

Control de los personajes marcados como favoritos (base de datos local).

**• ListCharactersDBViewModel.kt**: accede a la base de datos de ROOM para insertar o eliminar favoritos, y exponerlos.

**• ListCharactersDbStateUI.kt**: guarda el listado de favoritos y su estado (lista, conjunto de IDs, carga…).

#### **2.2. Carpeta episode**

Donde se realiza la gestión de episodios.

#### 2.2.1. Carpeta episodesList

Pantalla principal de todos los episodios.

• **ListEpisodesViewModel.kt**: se encarga de obtener todos los episodios y exponerlos a la UI.

**• ListEpisodesStateUI.kt**: contiene el listado y el estado de carga.

#### 2.2.2. Carpeta episodesFilterName

Filtro avanzado de episodios (por fecha, título, vistos…).

• ListEpisodesFilterViewModel.kt: gestiona todos los filtros y ejecuta los casos de uso según se apliquen.

**• ListEpisodesFilterStateUI.kt**: modelo de estado con los episodios filtrados y bandera de carga.

#### 2.2.3. Carpeta episodesListFav

Favoritos de episodios en base de datos.

• **ListEpisodesDBViewModel.kt:** inserta o elimina episodios como vistos o favoritos. Obtiene también los marcados.

• **ListEpisodesDbStateUI.kt:** contiene la lista de episodios favoritos o vistos, y sus IDs.

#### 2.2.4. Carpeta episodeDetails

Vista de detalle de episodio.

**• DetailsEpisodeViewModel.kt:** recupera el episodio por ID para mostrar todos sus datos.

• **DetailsEpisodeStateUI.kt:** estado del episodio detallado, control de carga, error, etc.

#### **2.3. Carpeta profile**

Donde se realiza la gestión del usuario.

**• ProfileViewModel.kt:** lee y actualiza las preferencias del usuario (idioma, estadísticas…).

**• ProfileStateUI.kt:** modelo de estado para la pantalla de perfil.

#### **2.4. Carpeta profile**

Donde se realiza la gestión de citas.

#### 2.4.1. Carpeta quotesList

Manejo de todas las citas (remoto o local según variant).

**• ListQuotesViewModel.kt:** llama al caso de uso para obtener citas desde la API o archivo local.

**• ListQuotesStateUI.kt:** contiene la lista de citas y el estado de carga.

#### 2.4.2. Carpeta quotesListFav

Gestión de citas guardadas como favoritas (Room).

**• ListQuotesDBViewModel.kt:** inserta o elimina citas favoritas.

**• ListQuotesDbStateUI.kt**: almacena las citas favoritas y sus IDs.

#### 2.4.3. Carpeta quotesGame

Gestiona el juego de adivinanzas.

#### 2.4.3.1. Carpeta questionGame

Lógica del juego (preguntas).

**• QuotesGameViewModel.kt:** genera preguntas con citas y posibles respuestas. Lleva la lógica principal del juego.

**• QuotesGameUI.kt:** estado del juego: puntuación, preguntas actuales, etc.

#### 2.4.3.2. Carpeta resultGame

Lógica para resultados del juego.

**• ResultGameViewModel.kt:** calcula los aciertos/fallos y guarda estadísticas del usuario.

• **ResultGameUI.kt:** modelo de estado para los resultados y estadísticas finales.

## Módulo es.upsa.mimo.thesimpsonplace (remote)

### **Carpeta data/di**

**• NetworkModule.kt:** este módulo proporciona la configuración de red real de la aplicación para producción o entornos conectados a la API real.

Incluyeretrofit + okHttpClient.

Con **@Provides fun provideRetrofit()** configuro la instancia de Retrofit con la BASE\_URL oficial: <https://thesimpsonsquoteapi.glitch.me/>

Incluye un **HttpLoggingInterceptor** para depuración.

El módulo está anotado con **@InstallIn(SingletonComponent::class)** para asegurar que las dependencias sean únicas en toda la app.

Se utiliza cuando el **Flavor** activo es remote. Esto permite que la app funcione con datos reales en producción.

## Módulo es.upsa.mimo.thesimpsonplace (mock)

### **Carpeta data/di**

**• MockModule.kt: e**ste módulo proporciona las dependencias necesarias para ejecutar la app con datos mock desde archivos **.json locales.**

Incluye inyección de CharacterDao y EpisodeDao con **@Provides**. En ambos daos se devuelven implementaciones personalizadas (CharacterDaoImpl, EpisodeDaoImpl) que leen desde archivos \*.json en assets/.

Los nombres de archivos (dataJson, imageJson, etc.) se inyectan con @Named, lo que permite alternar entre mock y producción fácilmente.

También usa **@InstallIn(SingletonComponent::class)** y **@Singleton** para una única instancia.

Se activa automáticamente cuando se compila la app con el **Flavor** mock.

## Módulo es.upsa.mimo.thesimpsonplace (test)

Este módulo incluye pruebas unitarias locales (sin necesidad de emulador/dispositivo).

### **Carpeta** domain.usecases.impl.character

• **DeleteCharacterDbUseCaseImplTest.kt**: testea que al ejecutar el caso de uso se borre correctamente un personaje en la BD simulado.

• GetAllCharactersUseCaseImplTest.kt: verifica que la lista de personajes se obtenga correctamente desde el repositorio simulado.

### **Carpeta** presentation.viewmodel.character

### **Subcarpeta** charactersFilterName

**• ListCharactersFilterViewModelTest.kt:** prueba el ViewModel de filtrado por nombre.

Evalúa comportamiento con listas vacías, loading y resultados esperados.

### **Subcarpeta** charactersListFav

• **ListCharactersDBViewModelTest.kt:** testea el ViewModel que gestiona favoritos.

Verifica inserción, eliminación, carga inicial y reactividad.

### **Subcarpeta** useCasesMockFake

Contiene mocks de los casos de uso para tests de integración y UI desacoplados del repositorio real.

**• DeleteCharacterDbUseCaseMock.kt**

**• GetAllCharactersDbUseCaseMock.kt**

**• GetCharacterDbByIdUseCaseMock.kt**

**• InsertCharacterDbUseCaseMock.kt**

Estos se usan como reemplazo de la lógica real para pruebas rápidas sin acceso a la BD o red.

### **Fichero** CharactersFake.kt

Contiene una lista mutable de personajes simulados.

Se usa como fuente de datos en mocks y tests, evitando dependencias externas.

## Módulo es.upsa.mimo.thesimpsonplace.presentation.ui (screenshotTest)

Módulo para tests de captura de pantalla con Jetpack Compose.

### **Carpeta** component.character

• **CharacterItemScreenshotPreview.kt**: captura cómo se ve un personaje en la lista (componente CharacterItem) con datos de prueba.

### **Carpeta** screen.characterSection

• **CharactersListScreenshotPreview.kt**: muestra una vista previa de toda la pantalla de personajes (CharactersScreen) con datos simulados.

## Mejoras Futuras

1. **Tests instrumentados (androidTest) y E2E**

Añadir pruebas reales de navegación y comportamiento en dispositivo físico para mejorar la cobertura.

1. **Autenticación de usuario**

Posibilidad de introducir login con Firebase/Auth0 y asociar estadísticas/favoritos a un usuario real.

1. **Sincronización offline**

Implementar lógica para mantener datos sincronizados entre la API y Room (mediante RemoteMediator de Paging 3, por ejemplo).

1. **Uso de Jetpack Compose Navigation avanzado**

Soporte para deep links, animated transitions, y navegación más dinámica.

1. **Modularización avanzada del proyecto**

Separar cada feature en un módulo independiente (por ejemplo: :character, :episode, :game, :core, etc).

1. **Mejoras visuales.**

Añadir animaciones suaves, mejora de feedback visual, y detección de modo oscuro de sistema.

1. **Analytics y crash reporting**

Integrar herramientas como Firebase Analytics o Sentry para seguimiento y mejora continua.

## Conclusión

Los puntos fuertes del proyecto:

1. **Arquitectura limpia y escalable (Clean Architecture + MVVM)**

La app sigue una separación clara por capas: presentation, domain y data, favoreciendo la mantenibilidad, testabilidad y evolución del proyecto.

1. **Inyección de dependencias con Hilt**

Permite un manejo eficiente de dependencias, mejora el modularidad y facilita el testing.

1. **Fuentes de datos desacopladas mediante Flavors y Build Types**

Uso avanzado de productFlavors (mock y remote) y buildTypes (debug, release, beta), lo que permite alternar entre datos reales (API) o simulados (JSON local) y activar/desactivar funcionalidades como logs.

1. **Persistencia local moderna con Room + DataStore**

Soporte completo para favoritos, estadísticas y preferencias del usuario. Uso de Flow para reactividad en tiempo real.

1. **Interfaz UI moderna con Jetpack Compose**

Diseño intuitivo, mantenible y con componentes reutilizables. Uso de StateFlow y LaunchedEffect para control del estado y side-effects de forma eficiente.

1. **Testing sólido**

• Tests unitarios para ViewModels y UseCases con mocks.

• Tests de integración con base de datos en memoria.

• Pruebas de UI mediante Screenshot Testing.

• Arquitectura preparada para pruebas instrumentadas (androidTest).

1. **Soporte multi-idioma (internacionalización)**

Implementación de strings.xml en varios idiomas (es, en, fr), con lógica adaptada a Android 13+.

1. **Diseño adaptado a buenas prácticas**

• Uso de Theme.kt, Color.kt y Typography.kt para estilo unificado.

• Código modular, reutilizable y documentado.

• Funcionalidad completa: filtros, detalles, favoritos, juego de citas, estadísticas.

Este proyecto no solo demuestra una comprensión de Jetpack Compose y arquitectura moderna en Android, sino también buenas prácticas de ingeniería de software, separación de responsabilidades, testing efectivo y enfoque en experiencia de usuario. Está preparado para escalar y ser mantenido a largo plazo.