

UNIVERSIDAD PONTIFICIA DE SALAMANCA

Máster Universitario Informática Móvil

Trabajo de la asignatura Desarrollo de aplicaciones Android

**Aplicación Android de Los** **Simpson**

Carlos Caño Gómez

Salamanca, mayo 2025

Índice

[Características Generales 5](#_Toc196986060)

[Descripción de la Aplicación 5](#_Toc196986061)

[Esquema de la Aplicación 6](#_Toc196986062)

[:app (Módulo principal) 6](#_Toc196986063)

[remote (Flavor remoto / producción) 12](#_Toc196986064)

[mock (Flavor local / test) 12](#_Toc196986065)

[test (Tests unitarios) 12](#_Toc196986066)

[androidTest (Tests instrumentados / UI / integración) 12](#_Toc196986067)

[screenshotTest  (Tests visuales por Captura) 13](#_Toc196986068)

[Otros ficheros 13](#_Toc196986069)

[Ficheros 15](#_Toc196986070)

[*MainActivity.kt* 15](#_Toc196986071)

[*TheSimpsonPlaceApp.kt* 15](#_Toc196986072)

[**Carpeta utils** 15](#_Toc196986073)

[DateExtensions.kt 15](#_Toc196986074)

[Logger.kt 15](#_Toc196986075)

[LocaleHelper.kt 15](#_Toc196986076)

[**Carpeta assets** 16](#_Toc196986077)

[personajes\_data.json / personajes\_test.json 16](#_Toc196986078)

[imagenes\_data.json / imagenes\_test.json 16](#_Toc196986079)

[episodios\_data.json / episodios\_test.json 16](#_Toc196986080)

[citas\_test.json 16](#_Toc196986081)

[**Carpeta res** 16](#_Toc196986082)

[Carpeta drawable 16](#_Toc196986083)

[Carpeta values/strings 16](#_Toc196986084)

[Arquitectura MVVM 17](#_Toc196986085)

[Carpeta *data* 17](#_Toc196986086)

[**Carpeta *daos (1)*** 17](#_Toc196986087)

[**Carpeta *local (1.1)*** 17](#_Toc196986088)

[**Carpeta datastore (1.1.1)** 17](#_Toc196986089)

[GameDao.kt y UserDao.kt 18](#_Toc196986090)

[GameDaoImpl*.kt y* UserDaoImpl.kt 18](#_Toc196986091)

[**Carpeta room (1.1.2)** 18](#_Toc196986092)

[CharacterDatabaseDao.kt, EpisodeDatabaseDao.kt, QuoteDatabaseDao.kt 18](#_Toc196986093)

[TheSimpsonsConverters.kt 19](#_Toc196986094)

[TheSimpsonsDatabase.kt 19](#_Toc196986095)

[**Carpeta *remote (1.2)*** 19](#_Toc196986096)

[CharacterDao.kt, EpisodeDao.kt, QuoteDao.kt 19](#_Toc196986097)

[**Carpeta *di (2)*** 20](#_Toc196986098)

[DatabaseModule.kt 20](#_Toc196986099)

[DataModule.kt 20](#_Toc196986100)

[DomainModule.kt 21](#_Toc196986101)

[**Carpeta *entities* (3)** 21](#_Toc196986102)

[**Carpeta character** 21](#_Toc196986103)

[**Carpeta episode** 21](#_Toc196986104)

[**Carpeta quote** 21](#_Toc196986105)

[**Carpeta user** 21](#_Toc196986106)

[CharacterDTO.kt 21](#_Toc196986107)

[CharacterEntity.kt 22](#_Toc196986108)

[Gender.kt 22](#_Toc196986109)

[ImageDTO.kt 22](#_Toc196986110)

[**Carpeta *mappers (4)*** 22](#_Toc196986111)

[Carpeta *domain* 22](#_Toc196986112)

[**Carpeta *mappers (1)*** 22](#_Toc196986113)

[**Carpeta *models (2)*** 23](#_Toc196986114)

[**Carpeta *repository (3)*** 23](#_Toc196986115)

[CharacterRepository.kt, etc 23](#_Toc196986116)

[CharacterRepositoryImpl.kt, etc 23](#_Toc196986117)

[**Carpeta *usecases (4)*** 23](#_Toc196986118)

[**Carpeta character (4.1)** 23](#_Toc196986119)

[**Carpeta episode (4.2)** 24](#_Toc196986120)

[**Carpeta quote (4.3)** 24](#_Toc196986121)

[**Carpeta game (4.4)** 24](#_Toc196986122)

[**Carpeta user (4.5)** 24](#_Toc196986123)

[**Carpeta impl (4.6)** 24](#_Toc196986124)

[GetAllCharactersDbUseCase.kt y GetAllCharactersDbUseCaseImpl.kt 24](#_Toc196986125)

[Carpeta presentation 25](#_Toc196986126)

[**Carpeta ui (1)** 25](#_Toc196986127)

[**Carpeta components (1.1)** 25](#_Toc196986128)

[**Carpeta character (1.1.1)** 25](#_Toc196986129)

[**Carpeta episode (1.1.2)** 25](#_Toc196986130)

[**Carpeta quote (1.1.3)** 25](#_Toc196986131)

[**Carpeta game (1.1.4)** 26](#_Toc196986132)

[**Carpeta navigation (1.2)** 26](#_Toc196986133)

[**Carpeta root (1.3)** 26](#_Toc196986134)

[**Carpeta screen (1.4)** 26](#_Toc196986135)

[Sección characterSection (1.4.1) 27](#_Toc196986136)

[Sección episodeSection (1.4.2) 27](#_Toc196986137)

[Sección quoteSection (1.4.3) 29](#_Toc196986138)

[Sección gameQuotes (1.4.4) 29](#_Toc196986139)

[Sección menuSection (1.4.5) 31](#_Toc196986140)

[Sección profileSection (1.4.6) 31](#_Toc196986141)

[**Carpeta theme (1.5)** 32](#_Toc196986142)

[**Carpeta viewmodel (2)** 33](#_Toc196986143)

[**Carpeta character (2.1)** 33](#_Toc196986144)

[**Carpeta charactersList (2.1.1)** 33](#_Toc196986145)

[**Carpeta charactersFilterName (2.1.2)** 33](#_Toc196986146)

[**Carpeta charactersListFav (2.1.3)** 33](#_Toc196986147)

[**Carpeta episode (2.2)** 34](#_Toc196986148)

[**Carpeta episodesList (2.2.1)** 34](#_Toc196986149)

[**Carpeta episodesFilterName (2.2.2)** 34](#_Toc196986150)

[**Carpeta episodesListFav (2.2.3)** 34](#_Toc196986151)

[**Carpeta episodeDetails (2.2.4)** 34](#_Toc196986152)

[**Carpeta profile (2.3)** 34](#_Toc196986153)

[**Carpeta profile (2.4)** 34](#_Toc196986154)

[**Carpeta quotesList (2.4.1)** 35](#_Toc196986155)

[**Carpeta quotesListFav (2.4.2)** 35](#_Toc196986156)

[**Carpeta quotesGame (2.4.3)** 35](#_Toc196986157)

[**Carpeta questionGame (2.4.3.1)** 35](#_Toc196986158)

[**Carpeta resultGame (2.4.3.2)** 35](#_Toc196986159)

[Módulo es.upsa.mimo.thesimpsonplace (remote) 35](#_Toc196986160)

[**Carpeta data/di** 35](#_Toc196986161)

[Módulo es.upsa.mimo.thesimpsonplace (mock) 36](#_Toc196986162)

[**Carpeta data/di** 36](#_Toc196986163)

[Módulo es.upsa.mimo.thesimpsonplace (test) 36](#_Toc196986164)

[**Carpeta domain.usecases.impl.character** 36](#_Toc196986165)

[**Carpeta presentation.viewmodel.character** 36](#_Toc196986166)

[**Subcarpeta charactersFilterName** 36](#_Toc196986167)

[**Subcarpeta charactersListFav** 36](#_Toc196986168)

[**Subcarpeta useCasesMockFake** 37](#_Toc196986169)

[Fichero CharactersFake.kt 37](#_Toc196986170)

[Módulo es.upsa.mimo.thesimpsonplace.presentation.ui (screenshotTest) 37](#_Toc196986171)

[**Carpeta component.character** 37](#_Toc196986172)

[**Carpeta screen.characterSection** 37](#_Toc196986173)

[Mejoras Futuras 37](#_Toc196986174)

[Conclusión 38](#_Toc196986175)

La aplicación **TheSimpsonsPlace** ha sido desarrollada por Carlos Caño Gómez y esta subida a Google Play en fase Alpha, donde se puede descargar a partir de los siguientes enlaces:

* [Unirse en Android](https://play.google.com/store/apps/details?id=es.upsa.mimo.thesimpsonplace) (los testers pueden unirse mediante Google Play en Android)
* [Unirse en la Web](https://play.google.com/apps/testing/es.upsa.mimo.thesimpsonplace) (los testers pueden unirse en la Web)

Tengo habilitado el acceso a los profesores con sus correos electrónicos de la UPSA para que inicien sesión en el Google Play (en caso de no encontrarse en la lista, se lo añadiría si me facilitan otro correo).

* Profesor: Roberto Berjón Gallinas - [rberjonga@upsa.es](mailto:rberjonga@upsa.es)
* Profesor: Antonio Leiva Gordillo [aleivago@upsa.es](mailto:aleivago@upsa.es)
* Profesor: Saúl Díaz González [sdiazgo@upsa.es](mailto:sdiazgo@upsa.es)



# **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, español (predeterminado) 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 Simpson 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. Se organiza en capas: presentation, domain, data.

Incluye viewModels, useCases, UI de Jetpack Compose, DAOs, repositorios y temas.

Además de todos los assets y configuraciones de Gradle. Incluye 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. Los tests de navegación y 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 de mi proyecto al subirlo en GitHub en <https://gitdiagram.com/carloscg2000/thesimpsonsplaceappandroid>)

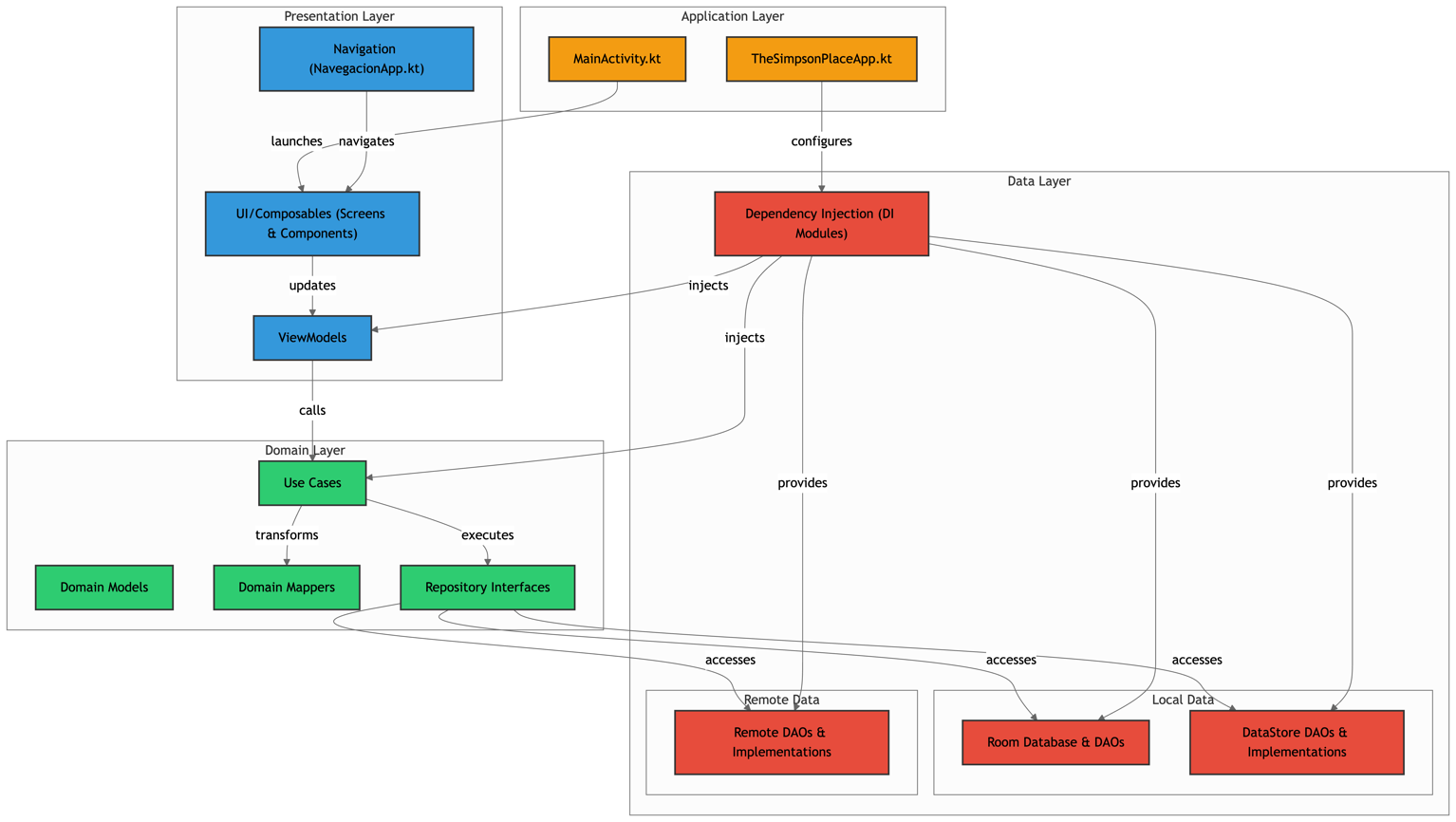


Ilustración 1. De GitHub a GitDiagram

# **Ficheros**

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

### *MainActivity.kt*

Representa el punto de entrada de la aplicación dentro de Android, concretamente con Jetpack Compose. Al tratarse de la actividad principal, es la encargada de iniciar la UI al ejecutarse la app. En este archivo se destaca el uso de @AndroidEntryPoint, lo cual habilita la inyección de dependencias con Hilt. Además, con enableEdgeToEdge() se configura la interfaz para que aproveche toda la pantalla del dispositivo, y finalmente, el método setContent{ MyApp() } establece la raíz composable de toda la interfaz de usuario.

### *TheSimpsonPlaceApp.kt*

Actúa como clase Application, siendo el punto global donde se configura Hilt y también la base de datos local con Room. Mediante la anotación @HiltAndroidApp se habilita el sistema de inyección para toda la aplicación. Dentro de esta clase, se define la instancia principal de la base de datos TheSimpsonsDatabaseRoom, accesible a través de Context.database. También se incluye una lógica de migración (migrationVersion1toVersion2) que permite añadir nuevas tablas, en mi caso episodes y quotes, manteniendo la integridad de los datos existentes. La base de datos se inicializa mediante Room.databaseBuilder().

### **Carpeta utils**

Contiene utilidades generales que facilitan el trabajo con tipos de datos comunes.

#### DateExtensions.kt

Se definen extensiones para convertir String a Date y viceversa, facilitando el formato y la visualización de fechas en la aplicación.

#### Logger.kt

Proporciona una interfaz para registrar logs (Logger) y su implementación (LoggerClass), permitiendo una trazabilidad efectiva del comportamiento de la app, especialmente en las variantes debug configuradas mediante los Build Types. Esta implementación resulta útil durante el desarrollo y depuración, sin interferir en el entorno de producción.

#### LocaleHelper.kt

Centraliza el cambio de idioma en la aplicación. En versiones de Android 13 o superior, se hace uso del LocaleManager de forma segura y moderna, mientras que en versiones anteriores se mantiene la compatibilidad mediante updateConfiguration, aplicando suppress deprecation para evitar errores de compilación.

### **Carpeta assets**

Se encuentran los archivos en formato JSON que actúan como fuente de datos estáticos. Estos archivos son fundamentales tanto para el funcionamiento offline como para los entornos de pruebas definidos mediante Flavors.

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

Contienen los listados de personajes reales y de prueba (por ejemplo, usada con el flavor mock), respectivamente.

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

Contiene las imágenes de los personajes reales y de prueba.

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

Contienen los listados de episodios reales y de prueba.

#### citas\_test.json

Agrupa frases célebres utilizadas exclusivamente en el entorno de pruebas. En producción, las citas provienen directamente de una API externa: <https://thesimpsonsquoteapi.glitch.me/>.

### **Carpeta res**

Contiene los recursos estándares que Android utiliza para componer la interfaz de usuario y soportar múltiples idiomas.

#### Carpeta drawable

Incluye imágenes .webp de los personajes utilizadas en distintas vistas (listados).

#### Carpeta values/strings

Centraliza los textos de la app mediante archivos strings.xml. Estos están localizados para diferentes idiomas (español, inglés y francés), permitiendo que la app se adapte automáticamente al idioma del dispositivo del usuario, garantizando una experiencia multilingüe.

# **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 permite desacoplar de forma clara y eficaz las responsabilidades de persistencia, acceso a red y lógica de transformación de datos. Esto facilita el mantenimiento, el testing y la posibilidad de adaptar la app a distintos entornos mediante el uso de Flavors o Build Types.

### **Carpeta *daos (1)***

La carpeta daos (Data Access Objects) encapsula las interfaces e implementaciones encargadas de obtener los datos. Se divide a su vez en **local** y **remote**, en función de la fuente de origen.

### **Carpeta *local (1.1)***

Organiza dos formas principales de persistencia: **datastore**, para datos simples y clave-valor (como preferencias del usuario), y **room**, para almacenamiento de entidades complejas en una base de datos relacional.

#### **Carpeta datastore (1.1.1)**

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).

La función **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 (1.1.2)**

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** es una interfaz que permite ejecutar operaciones CRUD en **Room** para **CharacterEntity**.

Algunas de las operaciones son:

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

Define el esquema general de la base de datos con @Database, incluyendo versiones y conversores personalizados (como TheSimpsonsConverters.kt) para manejar tipos no soportados nativamente por Room (fechas, listas, URLs, etc.).

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

### **Carpeta *remote (1.2)***

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** está formada por dos funciones:

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

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

**EpisodeDao.kt** está formada por seis funciones:

• **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** está formada por cuatro funciones:

• 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).

### **Carpeta *di (2)***

Centraliza toda la lógica de inyección de dependencias de la aplicación. A través de módulos bien definidos, se proporcionan las implementaciones necesarias para bases de datos (Room, DataStore), acceso a red (Retrofit), repositorios, casos de uso y otros servicios.

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 Qualifierspara diferenciar los **DataStore<Preferences>** de “**game**” y “**user**” permitiendo inyectar dependencias sin que haya conflicto entre ambas instancias.

#### DataModule.kt

Abstrae el origen de los datos remotos y permite elegir entre distintas fuentes (por ejemplo, JSON en modo mock o Retrofit en modo remote) dependiendo del Flavor activo.

#### DomainModule.kt

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* (3)**

Contiene todas las estructuras que representan datos en la aplicación, organizadas por dominio funcional: personajes (character), episodios (episode), citas (quote) y preferencias de usuario (user). Cada módulo define tanto DTOs (fuentes de json o API) como entidades locales (base de datos de Room), y otros modelos auxiliares.

#### **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 (4)***

Contiene las funciones de conversión entre los distintos modelos. Estos mapeadores permiten transformar los DTOs o entidades persistentes en modelos de dominio utilizados en la capa de presentación. Gracias a esta conversión explícita, se garantiza que la UI trabaje con modelos consistentes, independientes del origen de los datos, reforzando los principios de Clean Architecture.

## **Carpeta *domain***

Constituye el núcleo de la aplicación. Aquí es donde se define qué hace realmente la app, desde una perspectiva funcional, sin preocuparse por cómo ni desde dónde se obtienen los datos. Esta capa contiene las reglas de negocio, los modelos puros, los contratos de repositorios y los casos de uso, y se mantiene completamente aislada del resto de capas (presentación y datos), tal como dicta el principio de separación de responsabilidades de Clean Architecture.

### **Carpeta *mappers (1)***

Contiene funciones que transforman objetos entre diferentes formatos. Por ejemplo, se realizan conversiones de DTOs (obtenidos de la API o archivos JSON) a modelos de dominio, o de modelos de dominio a entidades persistibles de Room. Además, se incluyen funciones especializadas como toCharacterDb() o getRandomOptions(), utilizadas en la generación de preguntas del juego. Esta capa de mapeo permite desacoplar el dominio de los detalles de infraestructura y facilita la testabilidad.

### **Carpeta *models (2)***

Dentro de esta capa, los modelos de datos se encuentran organizados en la carpeta models. Estos modelos, como Character, Episode, Quote, Question o EpisodeFilter, representan la información tal y como se utiliza en la lógica de negocio y la UI, sin ningún tipo de dependencia con librerías como Room o Retrofit. Son entidades puras, centradas en la semántica y comportamiento que la app necesita.

### **Carpeta *repository (3)***

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

Se definen interfaces como CharacterRepository, EpisodeRepository o QuoteRepository, que actúan como contratos entre la lógica de negocio y la capa de datos. Estas interfaces no saben de dónde provienen los datos, solo exponen las operaciones necesarias para acceder a ellos.

#### CharacterRepositoryImpl.kt, etc

Se combinan distintas fuentes de datos (como DAOs locales y remotos) para ofrecer una experiencia unificada. Estas implementaciones suelen ejecutarse en Dispatchers.IO y pueden utilizar Flow para emitir datos de forma reactiva y eficiente.

### **Carpeta *usecases (4)***

Encapsulan operaciones específicas que la aplicación puede realizar. Estos casos están organizados por dominio funcional dentro de la carpeta usecases, subdividida en módulos como character, episode, quote, game, user, y impl. Por ejemplo, el módulo character contiene casos como GetAllCharactersUseCase, GetCharacterDbByIdUseCase, o InsertCharacterDbUseCase, cada uno enfocado en una tarea concreta.

#### **Carpeta character (4.1)**

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

#### **Carpeta episode (4.2)**

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.

#### **Carpeta quote (4.3)**

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

#### **Carpeta game (4.4)**

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

#### **Carpeta user (4.5)**

La carpeta user contiene GetUserPreferencesUseCase.kt, UpdateUserUseCase.kt

#### **Carpeta impl (4.6)**

Las interfaces de estos casos de uso definen lo que debe hacer cada acción (por ejemplo, devolver un Flow con la lista de personajes), mientras que las implementaciones delegan la lógica al repositorio correspondiente. Esto se realiza mediante inyección de dependencias con Hilt (@Inject constructor), lo que permite mantener un código limpio, desacoplado y fácilmente testeable.

##### GetAllCharactersDbUseCase.kt y GetAllCharactersDbUseCaseImpl.kt

Por ejemplo, en el caso de GetAllCharactersDbUseCase, su implementación en GetAllCharactersDbUseCaseImpl se limita a llamar al método del repositorio que devuelve todos los personajes guardados localmente. Gracias a esta estructura, la capa de dominio no conoce la fuente de los datos ni cómo se obtienen; simplemente se asegura de que se cumpla el comportamiento esperado.

Este enfoque no solo aporta claridad y separación de responsabilidades, sino que también facilita el mantenimiento, la evolución y la prueba de cada parte de la lógica de negocio. Los casos de uso pueden testearse de forma independiente, utilizando mocks del repositorio, y la UI puede consumirlos sin necesidad de conocer detalles de la infraestructura.

## **Carpeta presentation**

### **Carpeta ui (1)**

Es la encargada de gestionar toda la interfaz de usuario de la aplicación, respetando los principios de arquitectura limpia y buenas prácticas en Jetpack Compose. Está estructurada de manera modular, separando los elementos reutilizables de las pantallas específicas, y centralizando la lógica de navegación y personalización del tema. Esta organización permite mantener una presentación limpia, escalable y fácil de mantener.

#### **Carpeta components (1.1)**

Se agrupan todos los elementos visuales reutilizables que componen la interfaz común de la aplicación. Aquí se encuentran, por ejemplo, las barras de navegación superior e inferior (TopBarComponent, BottomBarComponent), campos de búsqueda personalizados (MySearchTextField), contenedores estilizados (ModifierContainer) o indicadores de ausencia de contenido (NoContentComponent). Además, existen subcarpetas especializadas por módulo como character, episode, quote y game, que definen componentes específicos para cada dominio, como tarjetas de personaje, ítems de episodio o vistas gráficas de estadísticas.

##### **Carpeta character (1.1.1)**

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.

##### **Carpeta episode (1.1.2)**

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.

##### **Carpeta quote (1.1.3)**

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.

##### **Carpeta game (1.1.4)**

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.

#### **Carpeta navigation (1.2)**

Contiene el archivo NavegacionApp.kt. En este archivo se define un NavHost con todas las rutas disponibles en la aplicación, utilizando una sealed class para representar las pantallas (Screen). Esta estructura permite una navegación segura, extensible y tipada, con soporte para el paso de parámetros entre pantallas. En total, la aplicación define 16 pantallas principales dentro del NavHost.

#### **Carpeta root (1.3)**

Configura el idioma y tema del sistema según las preferencias del usuario recuperadas desde ProfileViewModel, aplica el tema mediante TheSimpsonPlaceTheme, y finalmente lanza la navegación principal con NavegacionApp().

#### **Carpeta screen (1.4)**

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.**

Cada pantalla se enfoca en una funcionalidad concreta y su estructura permite una separación clara entre visualización, interacción y lógica.

##### Sección characterSection (1.4.1)



Ilustración 2. 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.

##### Sección episodeSection (1.4.2)



Ilustración 3. Pantallas de episodios 1



Ilustración 4. 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.

##### Sección quoteSection (1.4.3)



Ilustración 5. 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.

##### Sección gameQuotes (1.4.4)

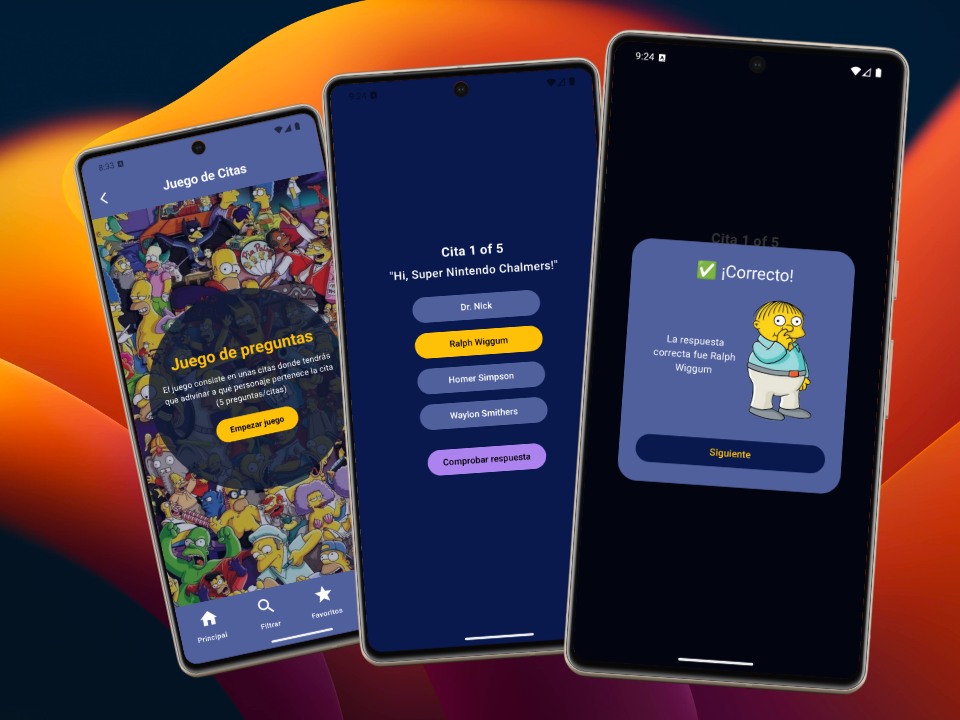


Ilustración 6. Pantallas del juego de citas 1



Ilustración 7. 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.

##### Sección menuSection (1.4.5)



Ilustración 8. 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.

##### Sección profileSection (1.4.6)



Ilustración 9. Pantalla de perfil

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

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

###### Sección profileEdit (1.4.6.1)

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

#### **Carpeta theme (1.5)**

Centraliza todos los elementos relacionados con la apariencia de la aplicación. Aquí se definen la paleta de colores (Color.kt), la tipografía global (Type.kt) y la configuración completa del tema (Theme.kt), que se integra con MaterialTheme y aplica dinámicamente la apariencia clara u oscura según las preferencias del sistema.

### **Carpeta viewmodel (2)**

La lógica de presentación y gestión del estado se encuentra en esta carpeta, estructurada también por módulo. Cada funcionalidad tiene su propio ViewModel y modelo de estado (UI State), lo que permite desacoplar la lógica de negocio de la presentación y mantener la UI reactiva. Se hace uso de StateFlow y viewModelScope.launch para emitir cambios de estado y ejecutar operaciones sin bloquear el hilo principal. Por ejemplo, la funcionalidad de personajes se divide en vistas para listado, filtrado por nombre y gestión de favoritos. Cada una de ellas cuenta con su ViewModel correspondiente (ListCharactersViewModel, ListCharactersFilterViewModel, etc.) y su estado asociado (ListCharactersStateUI, ListCharactersFilterStateUI, etc.), siguiendo el principio de responsabilidad única.

Del mismo modo, en la funcionalidad de episodios se cuenta con ViewModels para listado general, filtros avanzados por fecha o título, gestión de favoritos y vistos, y detalles de episodio. En el caso de las citas, se diferencian las pantallas de listado general, favoritas y el juego de preguntas, cada una con su lógica encapsulada.

#### **Carpeta character (2.1)**

Donde se realiza la gestión de personajes.

##### **Carpeta charactersList (2.1.1)**

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).

##### **Carpeta charactersFilterName (2.1.2)**

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.

##### **Carpeta charactersListFav (2.1.3)**

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…).

#### **Carpeta episode (2.2)**

Donde se realiza la gestión de episodios.

##### **Carpeta episodesList (2.2.1)**

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.

##### **Carpeta episodesFilterName (2.2.2)**

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.

##### **Carpeta episodesListFav (2.2.3)**

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.

##### **Carpeta episodeDetails (2.2.4)**

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.

#### **Carpeta profile (2.3)**

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.

#### **Carpeta profile (2.4)**

Donde se realiza la gestión de citas.

##### **Carpeta quotesList (2.4.1)**

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.

##### **Carpeta quotesListFav (2.4.2)**

Gestión de citas guardadas como favoritas (Room).

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

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

##### **Carpeta quotesGame (2.4.3)**

Gestiona el juego de adivinanzas.

##### **Carpeta questionGame (2.4.3.1)**

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.

##### **Carpeta resultGame (2.4.3.2)**

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 (Flavor remote)

### **Carpeta data/di**

Este módulo representa el entorno de producción real de la aplicación. En la carpeta data/di, se encuentra el archivo NetworkModule.kt, encargado de configurar todo lo relacionado con la red. Se define aquí la instancia de Retrofit, enlazada a la API oficial de citas de Los Simpsons (https://thesimpsonsquoteapi.glitch.me/), acompañada de un OkHttpClient con HttpLoggingInterceptor para facilitar la depuración durante el desarrollo.

Las dependencias están anotadas con @InstallIn(SingletonComponent::class) y @Singleton, garantizando que todas las instancias sean únicas y compartidas durante el ciclo de vida de la aplicación. Esta configuración se activa automáticamente al compilar el proyecto con el Flavor remote, permitiendo que la app funcione en producción utilizando llamadas reales a la red.

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

### **Carpeta data/di**

El módulo mock permite ejecutar la aplicación sin conexión a Internet, utilizando datos simulados cargados desde archivos JSON locales. Esta estrategia es ideal tanto para desarrollo sin red como para pruebas visuales y de funcionalidad con datos predecibles.

Dentro de la carpeta data/di, el archivo MockModule.kt proporciona las implementaciones de CharacterDao y EpisodeDao mediante versiones personalizadas (CharacterDaoImpl, EpisodeDaoImpl). Estas versiones leen directamente desde los archivos .json ubicados en los assets de la app. Para mayor flexibilidad, los nombres de archivo se inyectan utilizando la anotación @Named, permitiendo cambiar fácilmente entre configuraciones según el entorno activo.

Al igual que en el módulo remote, todas las dependencias están centralizadas en un SingletonComponent, asegurando que cada clase inyectada mantenga una única instancia durante toda la ejecución. Este módulo se activa al compilar el Flavor mock.

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

Este módulo incluye las pruebas unitarias locales que no requieren de emuladores ni dispositivos físicos. Está organizado por capas de dominio y presentación, reproduciendo fielmente la estructura de la app para mantener la coherencia y facilitar el mantenimiento de los tests.

Dentro de la carpeta domain.usecases.impl.character se encuentran las pruebas de casos de uso, como DeleteCharacterDbUseCaseImplTest.kt, que verifica la eliminación correcta de personajes en una base de datos simulada, y GetAllCharactersUseCaseImplTest.kt, que evalúa la correcta recuperación de todos los personajes desde el repositorio de prueba.

En la capa de presentación, dentro de presentation.viewmodel.character, se realizan pruebas a los ViewModels. Por ejemplo, en charactersFilterName, el archivo ListCharactersFilterViewModelTest.kt valida la funcionalidad del filtro por nombre, asegurando un comportamiento correcto con listas vacías, indicadores de carga y resultados filtrados. Por otro lado, en charactersListFav, el archivo ListCharactersDBViewModelTest.kt verifica el correcto funcionamiento del sistema de favoritos, asegurando la correcta inserción, eliminación y sincronización de los datos en la base local.

El módulo también incluye una subcarpeta useCasesMockFake, que agrupa implementaciones simuladas de los casos de uso (GetAllCharactersDbUseCaseMock.kt, InsertCharacterDbUseCaseMock.kt, etc.). Estas versiones mock permiten ejecutar tests de integración y pruebas de ViewModels de manera desacoplada, sin necesidad de acceder a la red o base de datos real. La fuente de datos utilizada es CharactersFake.kt, un archivo que contiene una lista mutable de personajes falsos reutilizados en diferentes pruebas.

### **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)

Finalmente, este módulo está dedicado a pruebas visuales de interfaz mediante captura de pantallas (Snapshot Testing) con Jetpack Compose. Permite validar visualmente que los componentes y pantallas de la aplicación se muestran correctamente en diferentes configuraciones.

En la carpeta component.character, el archivo CharacterItemScreenshotPreview.kt genera una captura del componente CharacterItem utilizando datos simulados. En la carpeta screen.characterSection, el archivo CharactersListScreenshotPreview.kt realiza una vista previa de toda la pantalla de listado de personajes (CharactersScreen), garantizando que la composición visual se mantenga estable tras futuras modificaciones en el código.

### **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.