UNIVERSIDAD NACIONAL DEL SUR

Diseño y Desarrollo de Software

TV Series

Ledesma Candela

Departamento de Ciencias e Ingeniería de la Computación

2024

Índice

1. Introducción

Este informe detalla el proceso de rediseño de la arquitectura de un sistema desde el modelo MVC hacia el modelo MVP, justificando las decisiones de diseño tomadas, explicando la aplicación de principios SOLID y Clean Code, y describiendo la organización del testing, tanto unitario como de integración. Además, se incluye el diagrama de clases actualizado y una descripción de las aclaraciones adicionales implementadas.

2. Rediseño de la Arquitectura a MVP

Para mejorar la modularidad y la escalabilidad del sistema, se rediseñó la arquitectura utilizando el patrón MVP (Model-View-Presenter). Este cambio permite una separación más clara de responsabilidades entre los componentes del sistema:

Creación de paquetes

Presenter

- Dividido en SearchPresenter, StoredPresenter y ScoredPresenter.
- MainPresenter actúa como coordinador, creando y delegando responsabilidades a los presentadores secundarios.
- MainPresenter implementa la interfaz SeriePresenter.

Model

- Dividido en SearchModel, StoredModel y ScoredModel.
- MainModel coordina y delega responsabilidades a los modelos secundarios.
- MainModel implementa la interfaz SerieModel.
- Incluye la base de datos, la clase Serie y un gestor JSON (JsonParser).

View

- Modularizada en clases que extienden JPanel: SearchPanel, ScoredPanel y StoragePanel.
- MainView maneja la creación del frame, configuración y visualización de los paneles.
- Implementa la interfaz SeriesView y delega responsabilidades.
- Incluye el elemento gráfico SerieMenuItem.

Utils

- Contiene la clase HtmlTextFormatter con un método estático textToHtml para formateo HTML.

3. Refactorización para Clean Code y SOLID

SOLID

1. **Single Responsibility Principle**: Se procuro que cada clase tenga una única responsabilidad. Ejemplo: JsonParser se encarga exclusivamente de procesar datos JSON.

2. **Open/Closed Principle** : Se pueden añadir nuevas funcionalidades sin modificar el código existente. Ejemplo: los paneles y modelos pueden extenderse sin afectar a las interfaces principales.

3. **Liskov Substitution Principle** : Las subclases pueden sustituir a sus superclases. Ejemplo: MainPresenter se comunica con SeriePresenter sin depender de la implementación específica.

4. **Interface Segregation Principle** : Las interfaces son pequeñas y específicas. Ejemplo: View solo requiere el método showView.

5. **Dependency Inversion Principle** : Los módulos de alto nivel no dependen de los de bajo nivel, sino de abstracciones. Ejemplo: el modelo utiliza una interfaz para la base de datos.

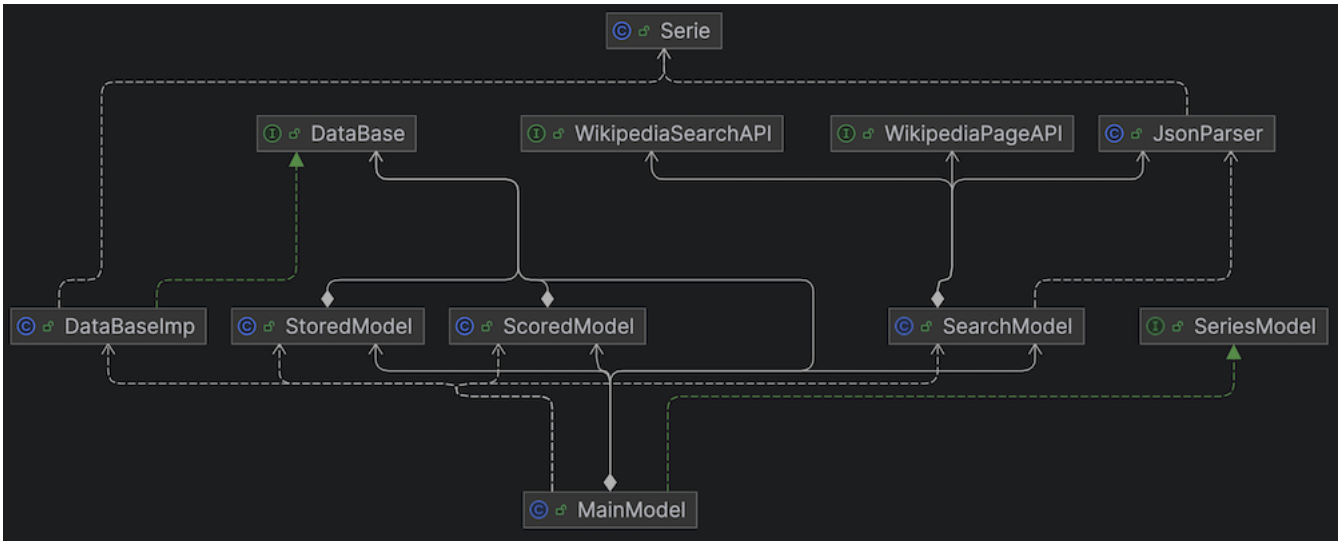
Refactorizaciones

- Se eliminaron métodos estáticos en la base de datos. Ahora el modelo instancia y controla el acceso a la base de datos.

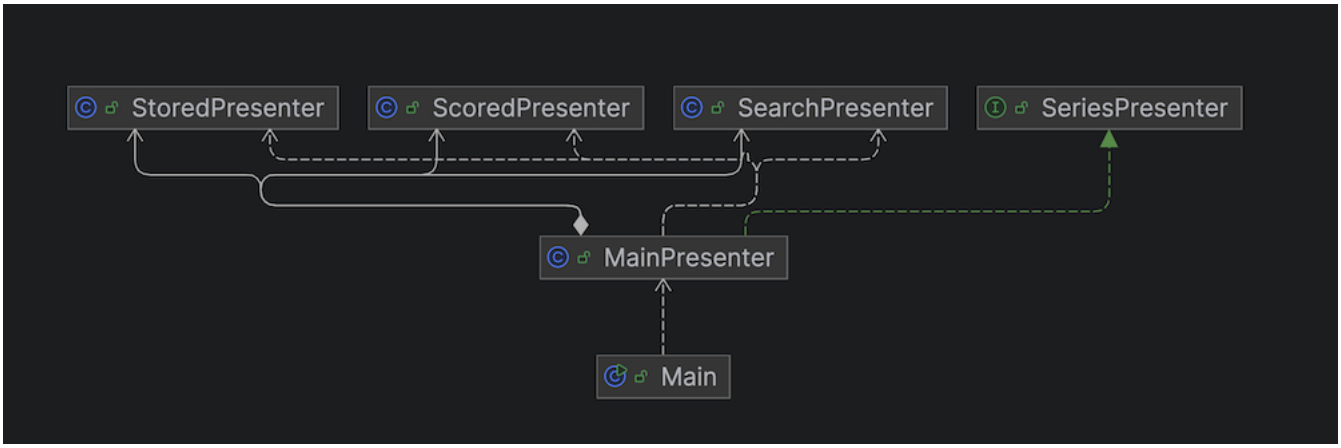
- Manejo de excepciones: Los Presenter capturan las excepciones y notifican a las vistas, mejorando la experiencia del usuario.

4. Diagrama de Clases

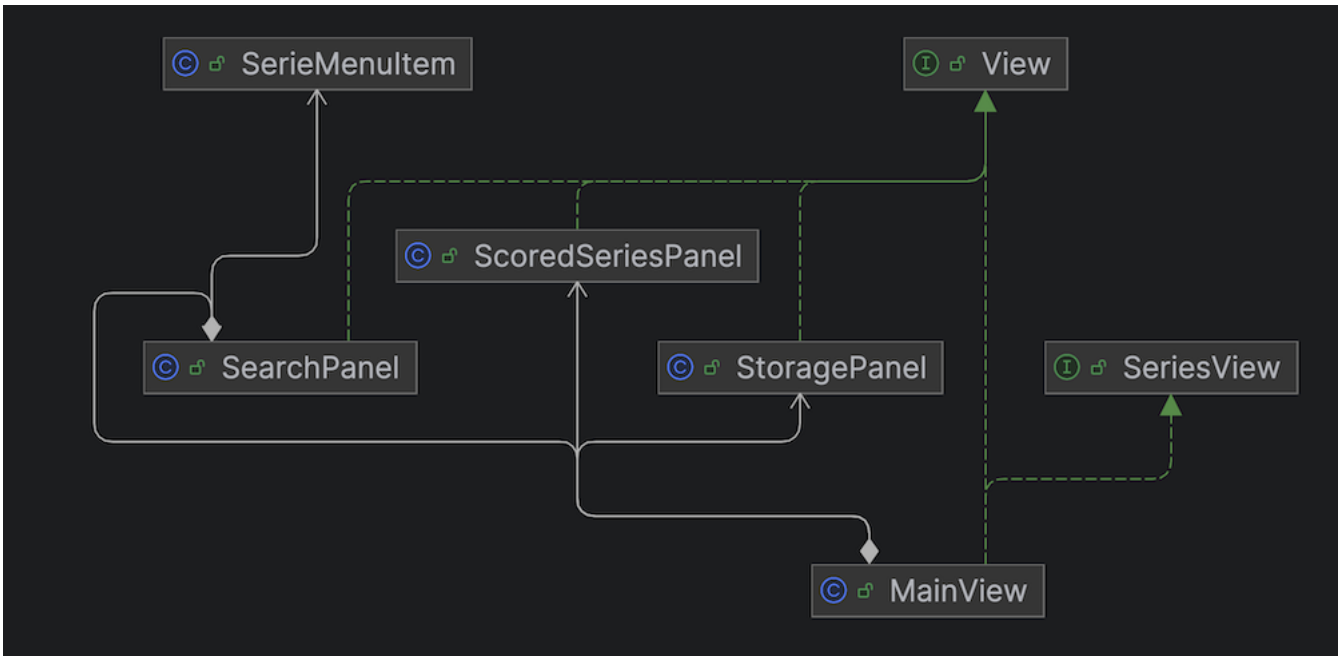
Model



Presenter



View



5. Decisiones de Diseño

Base de Datos

- Se creó una tabla adicional (scored) para almacenar series puntuadas, separadas de las series guardadas.
- Se implementó una interfaz para abstraer el acceso a la base de datos.

JSON Parsing

- Se implementó la clase JsonParser para manejar datos obtenidos de la API con Gson, facilitando el manejo de datos.

Vistas

- Cambio de LookAndFeel para compatibilidad entre sistemas operativos.
- Creación de la clase SerieMenuItem para desacoplar la lógica de la interfaz gráfica.

6. Testing

6.1. Testing Unitario

Se implementaron pruebas unitarias para validar funcionalidades específicas de los modelos:

ScoredModelTest

- testSetGetScore
- testHasScore
- testGetScoredSeries

SearchModelTest

- testSearchByTitle
- testSearchByTitleNotFound
- testSearchByTitleEmpty

StoredModelTest

- testSaveLocallyInfo
- testSavedInfo
- testDeletedInfo
- testGetSavedTitles
- testGetExtract

6.2. Testing de Integración

Se desarrollaron pruebas para validar la interacción entre componentes:

- testSearchSeries: Validación de búsqueda de series.
- testSearchWikiPage: Comprobación de extracción de datos de Wikipedia.
- testShowSavedSeries: Verificación de series almacenadas.
- testShowSavedExtract: Visualización de información guardada.
- testShowScoredSeries: Visualización de series puntuadas.

Se utilizaron stubs para simular dependencias externas como WikipediaSearchAPIStub, WikipediaPageAPIStub y DataBaseStub.

7. Conclusión

El rediseño hacia MVP, junto con la aplicación de principios SOLID y Clean Code, ha permitido obtener un sistema más modular, mantenible y escalable. La organización de pruebas unitarias e integración asegura la calidad del software, mientras que las decisiones de diseño tomadas fortalecen la separación de responsabilidades y la extensibilidad del sistema.