JamCity - Tech Challenge

Adrián Bayardi - 29 de noviembre del 2024

Introducción

Esta aplicación fue desarrollada para resolver el desafío técnico planteado, cuyo objetivo es agrupar, ordenar y calcular incrementos salariales para los empleados de una empresa multinacional de 251 trabajadores. Además de las funcionalidades solicitadas, se añadieron varias características adicionales que mejoran la interacción y flexibilidad del sistema, como la posibilidad de seleccionar exactamente qué empleados deben ser calculados para el salario o el ordenamiento de la lista de empleados según el ID, rol o seniority.

Arquitectura General

La aplicación utiliza una arquitectura basada en MVP (Model-View-Presenter) para desacoplar la lógica de negocio de la interfaz de usuario. Esto permite reutilización de código, modularidad y testeo aislado de cada componente.

El sistema se compone de los siguientes módulos principales:

1. **Carga Dinámica de Datos:** La lista de empleados se carga desde un archivo CSV utilizando un repositorio.
2. **Configuraciones de Salarios:** Los datos relacionados con salarios y posiciones se gestionan mediante `ScriptableObjects`.
3. **MVP:** Implementado para asegurar una separación clara entre la interfaz de usuario y la lógica de negocio.
4. **Factories:** Se utilizan para crear y gestionar las dependencias de los componentes.
5. **Services:** Gestionan la lógica de negocio, como cálculos y manipulación de datos.
6. **AppManager:** Controla el ciclo de vida de la aplicación y las dependencias.

Componentes

Carga de la lista de empleados

La lista de empleados se carga dinámicamente desde un archivo CSV mediante el módulo `EmployeeService`. Este servicio utiliza el patrón de repositorio para separar la lógica de acceso a datos.

- Se encarga de calcular los salarios y sus incrementos mediante la configuración cargada
- `IEmployeeRepository` define los métodos necesarios para obtener los empleados.

El uso de un CSV permite flexibilidad para modificar los datos sin necesidad de recompilar la aplicación.

Configuraciones de Sueldos y Posiciones

Se implementaron configuraciones de salarios y posiciones con `ScriptableObjects`. Esta decisión asegura:

- Los datos pueden ser configurados desde el editor de Unity sin modificar el código.
- Permite agregar nuevas configuraciones sin alterar la lógica existente.
- Se diferencian de la lista de empleados ya que se suelen modificar menos.

Promotion Panel

El **Promotion Panel** es el componente encargado de mostrar y gestionar la lista de empleados. Fue diseñado con el patrón MVP:

1. **Modelo:** Contiene los datos de los empleados.
2. **Vista:** Un prefab configurado en el `AppManager` que representa la interfaz gráfica.
3. **Presenter:** Maneja la lógica de negocio, incluyendo el ordenamiento por ID, rol y seniority.

El **Promotion Panel** es creado mediante un factory dentro del `AppManager`, asegurando que todas las dependencias sean inyectadas correctamente.

EmployeeCard

Cada empleado en la lista se representa con una tarjeta individual (**EmployeeCard**), también diseñada con MVP.

- **Factory:** El `AppManager` inyecta una factory para la creación de vistas y presenters de cada tarjeta.

AppManager

El `AppManager` es un Singleton responsable de:

- Gestionar el ciclo de vida de la aplicación.
- Inyectar dependencias necesarias en las factories.
- Registrar prefabs utilizados para las vistas.

Gestión de Eventos

Los cambios en el modelo son comunicados a las vistas mediante un sistema de eventos:

- Cada presenter se suscribe a eventos específicos del modelo.
- Esto garantiza que las vistas reflejen cambios en tiempo real, como actualizaciones en los incrementos salariales o modificaciones en el orden.

Lógica de Negocio

La lógica de cálculo de incrementos salariales y agrupamiento de empleados se gestiona en el `EmployeeService`. Este utiliza las configuraciones de `ScriptableObjects` para realizar:

1. Obtencion de los datos de los empleados mediante el repositorio
2. Cálculo de incrementos salariales basados en posición y seniority.

Patrones de Diseño Implementados

1. **Factory Method:** Para la creación de componentes como `PromotionPanel` y `EmployeeCard`.
2. **MVP:** Separación clara entre modelo, vista y presenter para el panel principal y cada carta
3. **Dependency Injection:** Facilita la implementación de pruebas unitarias.
4. **Repository:** Para el manejo de datos desde el CSV.
5. **Singleton:** Gestión centralizada de configuraciones y dependencias mediante `AppManager`.

Pruebas Unitarias

Se desarrollaron pruebas unitarias utilizando TDD (Test Driven Development) para los siguientes módulos clave:

1. **EmployeeService:** Verificación de la lógica de cálculo y agrupamiento.
2. **PromotionPanel:** Verificación del funcionamiento del modelo y el presentador
3. **EmployeeCard**: Verificación del funcionamiento del modelo y el presentador

---

Comentarios:

Tal vez haya hecho un poco de sobreingeniería en este desafío, pero lo hice con la intención de mostrar mi capacidad para implementar patrones de diseño como **MVP**, **Factory Method** y **Dependency Injection**, los cuales mejoran la escalabilidad y mantenibilidad de la aplicación. Aunque no era necesario hacerlo tan complejo, creí que era importante demostrar cómo se pueden aplicar estos patrones para crear un sistema modular y fácilmente ampliable. Esto también facilita la implementación de pruebas unitarias, algo clave en proyectos a largo plazo.

Quise replicar tambien lo que recuerdo de la arquitectura de **Panda Pop** intentando que la lógica de negocio, la presentación y los datos estuvieran completamente desacoplados. Esto permite un mantenimiento más sencillo y un testing más eficiente.

Además, incluí algunas funcionalidades extras que no fueron requeridas, pero que creo que enriquecen la aplicación y mejoran la experiencia del usuario. Estas incluyen:

- Ordenar la lista de empleados por ID, rol o seniority.
- Seleccionar empleados específicos para realizar cálculos salariales.
- Carga dinamica de la lista de empleados a traves de un CSV
- Configuracion mediante ScriptableObjects