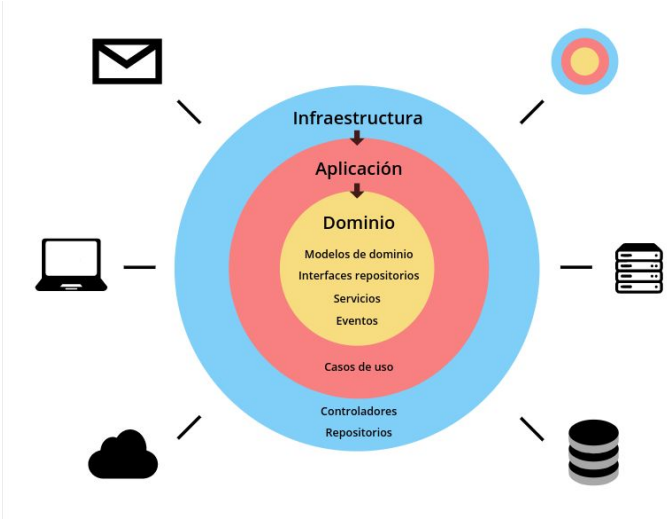
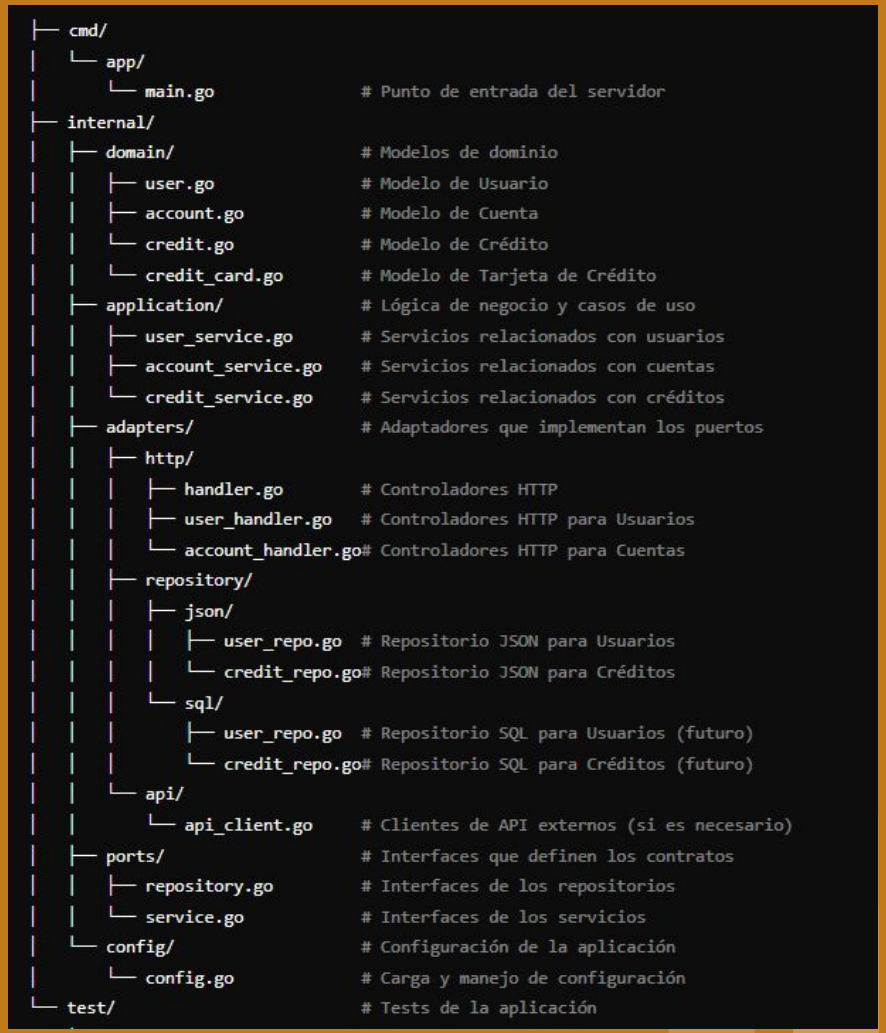
Arquitectura Hexagonal

La **arquitectura hexagonal**, también conocida como **arquitectura de puertos y adaptadores**, es un patrón de diseño que busca separar la lógica de negocio del código que maneja la entrada y salida de datos



Manejo de carpetas



Ventajas

Independencia de la infraestructura:

- **Descripción**: La lógica de negocio no depende de detalles técnicos como bases de datos, sistemas de archivos, APIs externas o frameworks.
- **Ventaja**: Esto facilita cambiar la infraestructura subyacente sin afectar el núcleo de la aplicación.

Facilita las pruebas:

- **Descripción**: Los componentes del núcleo pueden ser probados independientemente de la infraestructura externa.
- **Ventaja**: Esto permite hacer pruebas unitarias más confiables y rápidas, ya que se pueden usar mocks o stubs para las dependencias externas.

Flexibilidad y extensibilidad:

- **Descripción**: La arquitectura fomenta una clara separación de responsabilidades, lo que facilita agregar nuevas funcionalidades o cambiar las existentes.
- **Ventaja**: Hace que la aplicación sea más fácil de mantener y escalar a medida que crece.

Enfoque en el dominio:

- **Descripción**: El modelo de dominio y la lógica de negocio son el centro de la arquitectura.
- **Ventaja**: Permite un diseño más limpio y enfocado en resolver los problemas del negocio, en lugar de estar atado a decisiones tecnológicas.

Desventajas

Curva de aprendizaje:

- **Descripción**: Puede ser complicado para desarrolladores novatos entender y aplicar correctamente esta arquitectura.
- **Desventaja**: Puede requerir tiempo adicional de formación y adaptación, especialmente en equipos acostumbrados a arquitecturas más tradicionales.

Complejidad adicional:

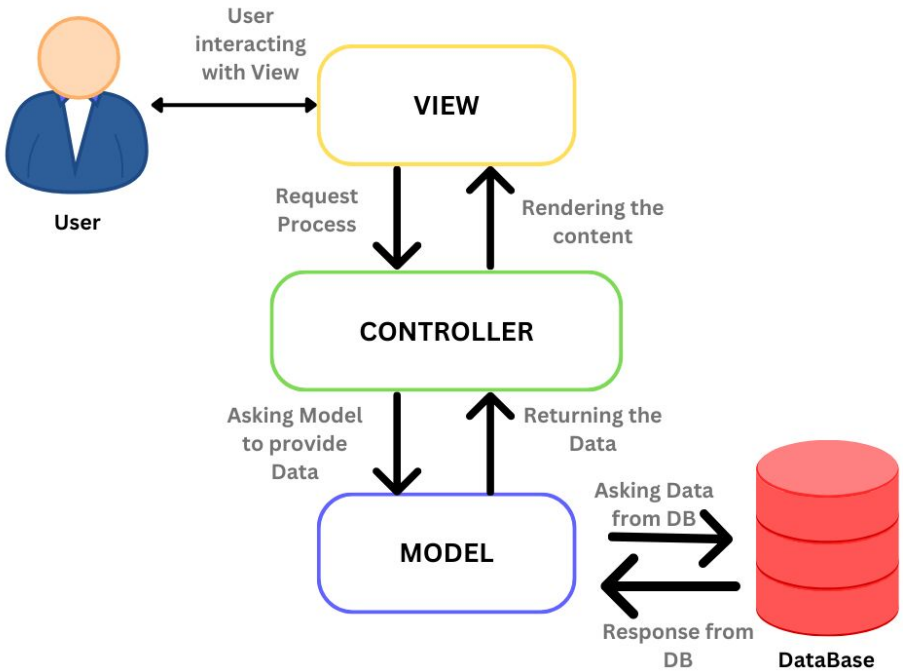
- **Descripción**: La arquitectura hexagonal introduce capas adicionales (puertos, adaptadores), lo que puede aumentar la complejidad del proyecto.
- **Desventaja**: En proyectos pequeños o simples, esta complejidad podría no estar justificada y podría ralentizar el desarrollo.

Aislamiento de módulos:

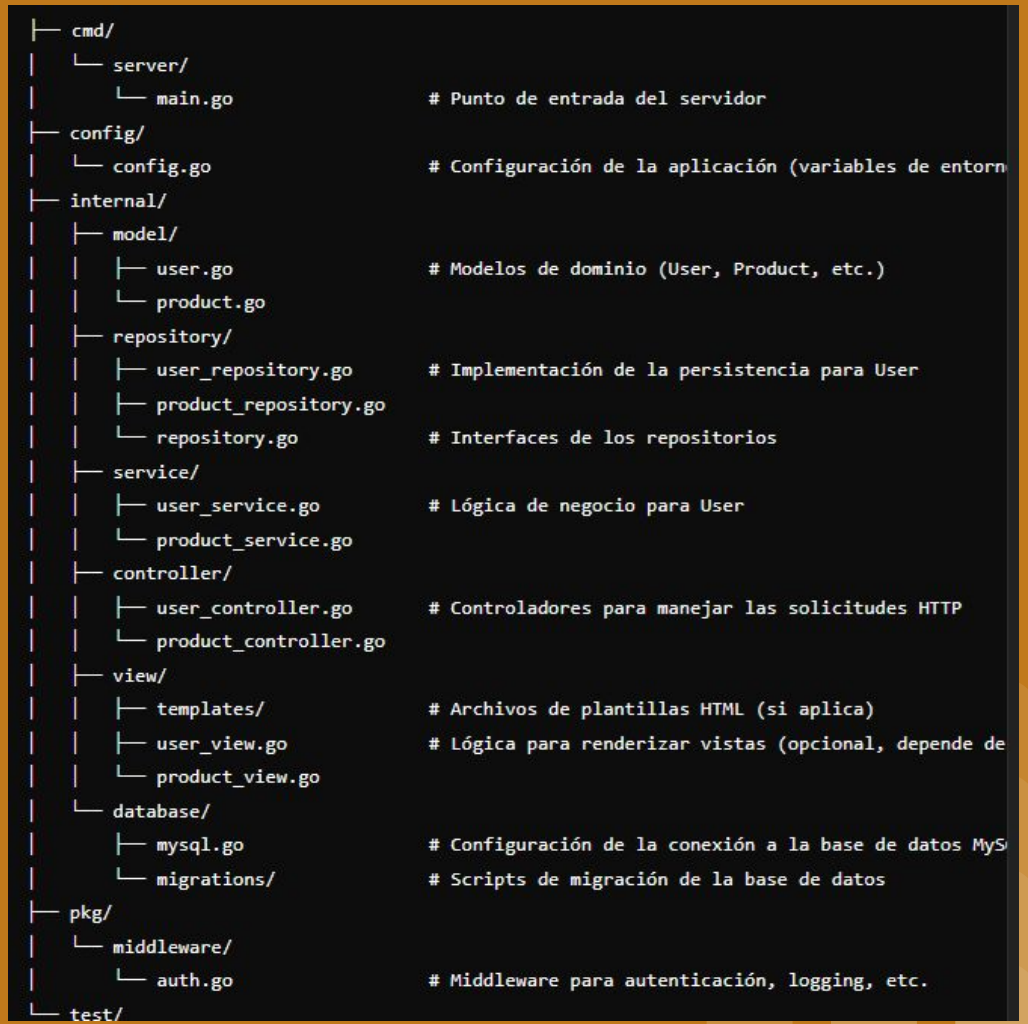
- **Descripción**: La arquitectura fomenta un aislamiento estricto de los módulos.
- **Desventaja**: Si no se gestiona adecuadamente, esto puede llevar a duplicación de código o esfuerzo adicional para mantener consistencia entre los módulos.

Modelo Vista Controlador

El patrón **MVC (Modelo-Vista-Controlador)** es un patrón de diseño arquitectónico utilizado en el desarrollo de software para separar las preocupaciones y organizar el código de manera más estructurada.



Manejo de carpetas



Ventajas

Separación de preocupaciones:

- El MVC divide la aplicación en tres componentes distintos, lo que facilita la organización del código y la colaboración entre diferentes equipos (por ejemplo, desarrolladores de backend pueden trabajar en el modelo, mientras que los diseñadores de UI trabajan en la vista).

Mantenibilidad:

- Como cada componente tiene una responsabilidad clara, el código es más fácil de mantener y extender. Si necesitas cambiar la lógica de negocio, puedes modificar el modelo sin afectar la vista.

Reutilización del código:

- El modelo puede reutilizarse en diferentes vistas, y las vistas pueden actualizarse o cambiarse sin alterar la lógica del modelo.

Facilidad de pruebas:

- La separación de responsabilidades facilita la realización de pruebas unitarias. Los modelos y controladores pueden probarse de manera aislada sin necesidad de incluir la lógica de la interfaz de usuario.

Escalabilidad:

- Debido a su estructura organizada, las aplicaciones basadas en MVC pueden escalarse más fácilmente. Se pueden agregar nuevos módulos sin afectar el funcionamiento existente.

Desventajas

Complejidad inicial:

- Implementar el patrón MVC puede añadir complejidad inicial al proyecto, especialmente en aplicaciones pequeñas o simples donde la separación de preocupaciones puede parecer excesiva.

Curva de aprendizaje:

- Los desarrolladores nuevos en el patrón MVC pueden necesitar tiempo para entender completamente cómo interactúan los diferentes componentes.

Sobrecarga:

- En aplicaciones muy simples, el uso de MVC puede introducir una sobrecarga innecesaria, haciendo que el desarrollo sea más complejo de lo necesario.

Interacción entre componentes:

- Aunque la separación de responsabilidades es una ventaja, también puede ser una desventaja en términos de comunicación entre componentes. Puede ser necesario un esfuerzo adicional para coordinar correctamente las interacciones entre el modelo, la vista y el controlador.