Bloque I – S4
Elección de una arquitectura

Ismael Ramos Alonso

Contenido

Supuesto 1: Aplicación de E-commerce para una PYME

Arquitectura recomendada para el Supuesto 1: MVC (Model-View-Controller)

Justificación:

- **Simplicidad y Rapidez:** MVC es una de las arquitecturas más simples y ampliamente adoptadas para el desarrollo de aplicaciones móviles. Dado el tiempo limitado y los recursos humanos disponibles, esta arquitectura permitirá al desarrollador principal estructurar la aplicación de manera eficiente sin una curva de aprendizaje empinada.

- **Separación de Responsabilidades:** MVC separa la lógica de negocio, la interfaz de usuario y los datos. Esto facilita la colaboración entre el desarrollador y el diseñador, ya que pueden trabajar en la vista (View) y el controlador (Controller) de manera independiente.

- **Rendimiento:** Aunque MVC no es necesariamente la arquitectura más eficiente en términos de rendimiento, es más que suficiente. Además, al ser una arquitectura popular, hay muchas optimizaciones y recursos disponibles para mejorar el rendimiento si es necesario.

- **Seguridad:** Aunque la seguridad no es una característica inherente de MVC, el desarrollador puede implementar medidas de seguridad robustas, especialmente al manejar transacciones y datos sensibles. Es esencial asegurarse de que la aplicación siga las mejores prácticas de seguridad, como el uso de HTTPS y la encriptación de datos.

- **Escalabilidad:** Si bien hay arquitecturas más escalables que MVC, se puede manejar el tráfico esperado. Si la aplicación crece en el futuro, se pueden considerar refactoring o migraciones hacia arquitecturas más robustas.

- **Experiencia del usuario:** MVC permite una rápida iteración y pruebas, lo que significa que el desarrollador y el diseñador pueden trabajar juntos para garantizar una experiencia de usuario fluida y atractiva.

En resumen, para una PYME con recursos limitados, tiempos de entrega cortos y expectativas de tráfico moderado, MVC es una elección adecuada. Es esencial, sin embargo, que el desarrollador esté familiarizado con las mejores prácticas de seguridad y rendimiento para garantizar una aplicación exitosa.

Supuesto 2: Aplicación Social Interactiva para una Startup

Arquitectura recomendada para el Supuesto 2: MVVM (Model-View-ViewModel)

Justificación:

- **Separación de Responsabilidades:** MVVM permite una clara separación entre la lógica de la interfaz de usuario y la lógica de negocio. Esto es especialmente útil en aplicaciones complejas como una aplicación social interactiva.

- **Facilita la Colaboración:** Dado que hay un equipo de tres desarrolladores y un diseñador, MVVM permite que múltiples personas trabajen en diferentes partes de la aplicación simultáneamente sin interferir entre sí.

- **Rendimiento y Escalabilidad:** MVVM es conocido por ser altamente escalable, lo que es esencial para una aplicación que espera un alto tráfico. Además, al tener un programador backend en el equipo, se puede garantizar que la lógica del servidor esté optimizada para manejar interacciones en tiempo real, como chats y transmisiones en vivo.

- **Flexibilidad:** MVVM es compatible con una variedad de frameworks y herramientas, lo que da al equipo la flexibilidad de elegir las tecnologías que mejor se adapten a sus necesidades.

- **Experiencia del Usuario:** Dado que MVVM permite una rápida iteración y pruebas, el equipo puede trabajar juntos para garantizar una experiencia de usuario fluida y atractiva, especialmente crucial para una aplicación social donde la retención del usuario es clave.

- **Seguridad:** Aunque la seguridad no es una característica inherente de MVVM, el hecho de tener un programador backend en el equipo garantiza que se puedan implementar medidas de seguridad robustas, especialmente al manejar chats en tiempo real y transmisiones en vivo.

En resumen, para una startup que busca crear una aplicación social interactiva con expectativas de alto tráfico y necesidades de interacción en tiempo real, MVVM es una elección adecuada.

Supuesto 3: Aplicación Financiera para una Gran Empresa

Arquitectura recomendada para el Supuesto 3: Arquitectura Hexagonal

Justificación:

- **Seguridad:** La Arquitectura Hexagonal, al igual que Clean Architecture, promueve una clara separación de responsabilidades. Esta separación permite que los especialistas en seguridad se centren en proteger los puertos y adaptadores.

- **Escalabilidad y Rendimiento:** La Arquitectura Hexagonal es ideal para sistemas que esperan un tráfico muy alto. Al separar la lógica de negocio de los servicios externos y la interfaz de usuario, es posible escalar cada componente de manera independiente según las necesidades.

- **Testabilidad:** Al igual que Clean Architecture, la Arquitectura Hexagonal es altamente testable. La lógica de negocio se encuentra en el núcleo y está completamente desacoplada de las entradas y salidas.

- **Mantenibilidad:** La combinación de Clean Architecture con la Arquitectura Hexagonal resulta en un sistema modular y desacoplado. Esto facilita la adición, modificación o eliminación de componentes sin afectar otras partes del sistema.

- **Independencia de Frameworks y UI:** Ambas arquitecturas promueven la independencia de tecnologías específicas. Esto permite al equipo elegir las herramientas y frameworks que mejor se adapten a sus necesidades, garantizando al mismo tiempo que la lógica de negocio permanezca intacta y sin cambios.

- **Colaboración Eficiente:** Dado el tamaño y diversidad del equipo (desarrolladores, diseñadores, especialistas en seguridad, analistas), la clara separación de responsabilidades permite que cada miembro se concentre en su área de especialidad sin interferir con el trabajo de los demás.

- **Experiencia del Usuario:** Aunque el enfoque principal de estas arquitecturas es la estructura y lógica del código, la capacidad de escalar y adaptar componentes de manera independiente garantiza que la aplicación pueda manejar un alto tráfico sin comprometer la experiencia del usuario.

En resumen, para una gran empresa financiera que busca desarrollar una aplicación robusta, segura, escalable y de alto rendimiento, la combinación de Clean Architecture con la Arquitectura Hexagonal es la elección ideal.

Supuesto 4: Plataforma de Salud y Bienestar para Hospitales

Arquitectura recomendada para el Supuesto 4: Arquitectura Hexagonal (también conocida como "Ports and Adapters") combinada con Clean Architecture

Justificación:

- **Seguridad y Privacidad:** La Arquitectura Hexagonal y Clean Architecture priorizan la seguridad al desacoplar la lógica de negocio de las interfaces externas.

- **Modularidad y Mantenibilidad:** Estas arquitecturas resultan en un sistema modular donde cada componente tiene una responsabilidad única. Esto facilita la adaptación y evolución del sistema a lo largo del tiempo.

- **Testabilidad:** Ambas arquitecturas son altamente testables. La capacidad de probar cada componente de manera aislada garantiza que la aplicación sea confiable y funcione según las expectativas, lo cual es crucial en el ámbito médico.

- **Escalabilidad y Rendimiento:** La Arquitectura Hexagonal, con su estructura desacoplada, permite escalar cada componente de manera independiente según las necesidades. Esto es esencial para manejar el alto tráfico esperado de una gran cantidad de pacientes.

- **Integración con Sistemas Existentes:** La Arquitectura Hexagonal, con su enfoque en puertos y adaptadores, facilita la integración con sistemas existentes o legados sin comprometer la integridad o seguridad de la aplicación.

- **Experiencia del Usuario:** Aunque el enfoque principal de estas arquitecturas es la estructura y lógica del código, la capacidad de adaptar y escalar componentes de manera independiente garantiza que la aplicación pueda manejar un alto tráfico sin comprometer la experiencia del usuario.

En resumen, para un hospital de renombre que busca desarrollar una plataforma de salud y bienestar robusta, segura y escalable, la combinación de Arquitectura Hexagonal con Clean Architecture es la elección ideal. Estas arquitecturas proporcionan un marco que satisface las demandas y desafíos específicos del dominio médico y las expectativas de una institución de salud de alto calibre.

Supuesto 5: Aplicación Prototipo para un Hackathon

Arquitectura recomendada para el Supuesto 5: MVC (Model-View-Controller)

Justificación:

- **Simplicidad y Rapidez:** Dado el corto tiempo disponible, MVC es una de las arquitecturas más rápidas y sencillas de implementar. Permite a los estudiantes estructurar rápidamente la aplicación y centrarse en la funcionalidad principal.

- **Separación de Responsabilidades:** MVC separa la lógica de negocio, la interfaz de usuario y los datos. Esto facilita la colaboración entre los estudiantes, ya que pueden trabajar en diferentes componentes de manera simultánea. El desarrollador puede centrarse en el modelo y el controlador, mientras que el diseñador trabaja en la vista.

- **Recursos Gratuitos y Amplia Documentación**: Dado que MVC es una arquitectura popular, hay una abundancia de recursos, tutoriales y herramientas gratuitas disponibles.

- **Demostrabilidad:** Para un hackathon, es esencial que el prototipo sea demostrable al final del evento. MVC, con su estructura clara, permite a los estudiantes crear una aplicación que puede ser fácilmente presentada y demostrada a los jueces o a la audiencia.

- **Flexibilidad:** Aunque MVC es una arquitectura simple, ofrece suficiente flexibilidad para que los estudiantes implementen características básicas como la búsqueda de compañeros de viaje, la comunicación entre usuarios y la división de gastos.

- **Adaptabilidad:** Si el prototipo resulta ser un éxito en el hackathon y el equipo decide continuar con el desarrollo, MVC proporciona una base sólida sobre la cual se pueden agregar más características o se puede refinar la aplicación.

En resumen, para un hackathon con un tiempo limitado, recursos mínimos y la necesidad de demostrar rápidamente una idea, MVC es la elección ideal. Proporciona una estructura clara y rápida que permite a los estudiantes centrarse en la funcionalidad principal y presentar un prototipo funcional al final del evento.