GUÍA PRÁCTICA DE DISEÑO DE SOFTWARE

Para estudiantes de 6to nivel de la Carrera de Ingeniería de Software

Unidad III : Conceptos de diseño, principios, estándares y lineamientos de calidad

Grupo 2 : Lugmaña Matias, Rea Denise , Orrico Camilo, Viche Julio

Documento base: U3_Conceptos_Diseno_Principios_Estandares_Lineamientos.pdf

1. Objetivo y proyección

Esta guía nos ayuda ha a aterrizar en los conceptos claves para el diseño de software que puede ser utiliza en cualquier nivel de estudio o en ámbito ya profesional el cual incluye listas de verificación de la ISO 25010 así como principios prácticos para el diseño

2. Principios prácticos de diseño

- Modularidad:
    - ¿Puedo entender/cambiar una parte sin romper las demás?
      Sí, ya que el proyecto es modular, de esta forma no rompemos las estructuras, se tienen módulos independientes, teniendo la posibilidad de realizar cambios a futuro.

- Cohesión y Acoplamiento
    - Cada módulo hace una cosa clara? Depende lo mínimo de otros?
      Se lo ha logrado parcialmente, ya que se mantiene una alta cohesión delegando toda la persistencia de datos a MongoDB Atlas de forma externa. Sin embargo, para mejorar el acoplamiento, estamos trabajando en mover las configuraciones de conexión a archivos independientes en lugar de dejarlas fijas en el código de las rutas, evitando así dependencias innecesarias

- Abstracción y encapsulación
    - ¿Oculto detalles internos y expongo solo contratos?
      Sí, porque el backend actúa como una caja negra y los endpoints se ocultan ante un find() o aggregate() exponiendo solo el JSON

- Separación de responsabilidades
    - ¿UI, lógica de negocio y persistencia están separadas?
      Se ha implementado de manera efectiva la separación de responsabilidades trabajamos en un entorno web donde tenemos un front-end que es la parte visual de nuestro sistema donde las personas y nuestro cliente pueden interactuar además de utilizar las funciones que esta presenta y un backend donde tenemos todo lo que es logica del negocio así como su conexión con la base de datos que es encuentra en la nube.

- Reutilización y patrones
    - ¿Reutilizar componentes y patrones conocidos cuando conviene?
      Sí, ya que se lo ha implementado al separar las rutas de la lógica de negocio, se aplica el patrón MVC (Modelo-Vista-Controlador). Además, aprovechamos las capacidades de la librería Mongoose para reutilizar la lógica de validación de esquemas en diferentes partes de la aplicación sin reescribir código.

- Diseño para pruebas
    - ¿Puedo probar funciones y componentes de forma aislada?
    - Por definir, para facilitar la ejecución de pruebas unitarias en el futuro, necesitamos ajustar el código para que la conexión a la base de datos no se inicie automáticamente al importar los archivos. Esto, junto con un entorno limpio, nos permitirá probar cada componente de forma aislada

- Seguridad por diseño
    - ¿Válido entradas, gestionar permisos y registrar auditoría?
      Por definir, aun no han validado las rutas por ello se tiene pendiente implementar aquella validacionesn sen las rutas de acceso. Tenemos que reforzar la validación en los campos de registro para prevenir ataques externos o sobrecargas en el sistema.

Adecuación funcional

- Todos los casos de uso que se acordarón fueron implementados?

Si cumple parcialmente, de momento están implementados unos, otros en proceso y otros planeados para su implementación

- Los resultados son correctos para entradas válidas y límites?

Cumple de forma parcial. se debe realizar pruebas para validar las entradas existentes en los campos

- Las funciones ayudan a lograr objetivos del usuario sin pasos extra?

Sí, cumple, ya que se cuenta con un CRUD, y una implementación visual adecuada para los usuarios

Eficiencia de desempeño

- ¿Responde dentro del tiempo objetivo bajo carga esperada?

Sí, cumple parcialmente, de momento la aplicación no ha eperimentado fallos o demoras, pero aun no se han realizado pruebas de carga para que sea medible.

- ¿Es razonable y monitoreable el uso de CPU/RAM/BD?

Cumple de forma parcial,se debe implementar pruebas para tomar métricas y medir la eficiencia

- ¿Soporta el volumen esperado (N registros, concurrencia)?

Si, cumple parcialmente, se debe realizar pruebas de carga y concurrencia

Compatibilidad

- ¿Convive con otros sistemas/servicios sin interferir?

  Si cumple, ya que al ser un servicio en la web no interfiere con otros sistemas

- ¿Integra/expone APIs con contratos?

  No.

Usabilidad

- ¿El usuario entiende si el sistema le es util?

Sí cumple, ya que el sistema muestra alertas, tiene distintos colores y un diseño usable para el usuario

- ¿Un usuario nuevo aprende a usarlo rápidamente?

Sí cumple, ya que el sistema es similar a otros sistemas en su diseño

- ¿Se puede operar con facilidad sin necesidad de muchos pasos?

Sí, cumple de forma parcial, se debe realizar pruebas de interacción con el usuario.

- ¿Previene errores (validación, confirmaciones)?

Sí, cumple de forma parcial, las validaciones implementadas son básicas

- ¿Diseño consistente y legible?

Si cumple, se usan colores que hacen contraste y permitiendo al usuario entender el diseño

- ¿Soporta accesibilidad mínima (labels, contraste, teclado)?

Fiabilidad

- ¿Tiene pocas fallas en condiciones normales?

  No, ya que falta correr pruebas para verificarlo

- ¿Maneja fallos sin colapsar?

  No, no se han implementado soluciones a los fallos

- ¿Se recupera ante fallos?

  No, debido a que la tecnología que usamos es limitada en este aspecto por su plan gratuito

Seguridad

- ¿Controla acceso a datos (authz/authn)?

Se cumple, ya que se manejan los permisos en la base de datos

- ¿Protege contra modificación no autorizada y valida entradas?

No, aun no se implementa seguridad

- ¿Se registra acciones para evidenciar autoría?

No tiene auditoria

- ¿Se puede rastrear quién hizo qué?

No se pueden rastrear no se hay auditoria en el sistema

Mantenibilidad

- ¿Módulos separados y dependencias controladas?

Si cumple, los modulos estan separados en capas con las siguientes capas:

controller, database, factories, models, repositories, routes.

- ¿Componentes reutilizables?

No, se espera implementarlo en un futuro

- ¿Es fácil diagnosticar defectos?

Si, porque el flujo es facil de entender

- ¿Cambios locales con bajo riesgo?

Si cumple

- ¿Hay pruebas unitarias/integración y CI?

No existen pruebas todavía

Portabilidad

- ¿Puede ejecutarse en distintos entornos sin cambios mayores?

Si , nuestro aplicativo como es web está pensado para ordenadores , pero también se lo puede utilizar en dispositivos móviles mediante un diseño responsivo previamente analizado y desarrollado con el equipo de trabajo.

- ¿Instalación reproducible (scripts, Docker, README)?

  Parcialmente

  Como tal ocupamos algunos frameworks tanto para el front y back depende en el entorno (servidor) en donde se quiera reproducirlo , por el lado de la base de datos el uso de docker es una buena alternativa para tener un sistema distribuido hasta el momento nuestra BD se encuentra en la nube con MongoDB Atlas , pero si se lo podría reproducirlo en un entorno dock erizado creando las instrucciones de como hacerlo con un docker-compose.yaml y todo especificado en un README en nuestro repositoria de git.

- ¿Puede reemplazar un componente por otro equivalente?

Si, se puede realizar en nuestro caso sería el uso de librerías que nos ayudan o mejoran en algunos aspectos de funcionalidad del aplicativo ya sea para funciones específicas o temas de rendimiento y escalabilidad de dicho sistema en el entorno web.