**Guía 3 Principios SOLID**

**Arquitectura de Software**

**Carlos Eduardo Mujica Reyes**

**Julián Mauricio Montoya Rodríguez**

**Bryan Andres Puentes Vargas**

**Universidad Manuela Beltrán**

**Bogotá D.C**

**2 de marzo de 2025**

1. **Modelo de Arquitectura**

* **Descripción**

El proyecto consiste en desarrollar un Sistema de Planificación de Recursos Empresariales (ERP) basado en HTML y CSS para gestionar los procesos clave de una fábrica de ampollas médicas aisladas. Este ERP permitirá administrar la producción, el inventario, los pedidos, la calidad del producto, la gestión de clientes y proveedores, además de generar reportes de desempeño. Dentro de los módulos que se podrán encontrar dentro del ERP están:

1. Módulo De Inventario
2. Módulo De Contabilidad
3. Módulo De Financiera
4. Módulo De Recursos Humanos
5. Módulo De producción

* **Trade Offs**

Para el proyecto que vamos a desarrollar este semestre, se nos han ocurrido varios “trade offs” que aplican y varían dependiendo del tipo de tecnología que se va a usar, por esa razón se va a hacer una pequeña tabla con 5 trade offs que se van a tener en cuenta mientras se desarrolla este ERP

|  |  |  |
| --- | --- | --- |
|  | **Lo bueno** | **Lo malo** |
| **HTML, CSS y JavaScript con Node.js** | Es moderno, flexible y permite una buena escalabilidad a futuro. | Tiene una gran curva de aprendizaje y, por lo tanto, más esfuerzo en el desarrollo. |
| **MySQL (Base de datos relacionales)** | Hay una mejor integridad de los datos y también ayuda a la consistencia. | Llega a ser lento cuando se manejan grandes volúmenes de datos que no están estructurados. |
| **Firebase (NoSQL en la nube)** | Es fácil de escalar, no hay necesidad de hacer una administración a los servidores | Llega a ser menos eficiente cuando se trata de consultas complejas o datos relacionales |
| **Mas seguridad (Autenticaciones, controles de acceso etc.)** | Se protege la información sensible de la empresa o de la fabrica que haga uso del ERP | Puede hacer más compleja la implementación y es probable que también aumenten los costos |
| **Mas facilidad de uso (Interfaz sencilla)** | Reduce la curva de aprendizaje de los empleados y aumenta el gusto por el ERP | Es probable que sea mas vulnerable a errores o accesos no autorizados a los datos de la fábrica o empresa |
| **Enfoque Modular** | Permite agregar funciones que dependen del crecimiento de la empresa | Puede retrasar la entrega del ERP porque requiere más planificación. |
| **Enfoque Minimalista** | Reduce el tiempo de implementación y permite una validación rapida | Llega a ser insuficiente si la empresa crece rápido y si llega a necesitar de mas funcionalidades |

* **Tecnologías seleccionadas**

Para la parte de Frontend se va a hacer uso de HTML, CSS y JavaScript gracias a que con ellas podemos llegar a tener una interfaz de usuario más intuitiva, también por el diseño responsivo y también porque llega a ser escalable en caso de necesitar mejoras a futuro. Para la parte de Backend se va a hacer uso de Node.js con Express, la base de datos que va a hacer uso el ERP va a ser Firebase por su conectividad en la nube y por su fácil configuración. Herramientas de desarrollo extra que se van a usar serán Github para el control de versiones y Jira o Trello para tener una mejor gestión del proyecto

* **Modelo C4**

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Diagrama, Aplicación

El contenido generado por IA puede ser incorrecto.

1. **Diagrama

   El contenido generado por IA puede ser incorrecto.Patrones de Diseño**

* **Patrones De Diseño**

A continuación, se va a hacer una pequeña lista de los patrones de diseño que se van a usar para el proyecto seguido de una breve definición y descripción

1. Arquitectura de capas: Lo que hace es separar la aplicación en capas, entre los beneficios que tiene se encuentran que facilita la escalabilidad y el mantenimiento y permite modificar una capa sin modificar las demás.
2. Patrón de Singleton: Garantiza que una clase solo tenga una instancia y proporciona de ella un punto de acceso global, entre los beneficios que tiene es que hay un control centralizado de los recursos compartidos y ella evita que se creen múltiples conexiones que no son necesarias.
3. Modelo-vista-controlador: Separa los datos (modelo), la interfaz de usuario (vista) y la lógica de control (controlador) entre los beneficios que tiene es que facilita la reutilización del código y hace que la UI sea más fácil de modificar sin afectar la lógica del negocio como tal.

* **Principios Solid**

Los principios solid que se van a aplicar a nuestro proyecto son los siguientes

1. Principio de responsabilidad única: La manera en como se puede aplicar es por ejemplo separar la lógica de la gestión del inventario, pedidos y producción en módulos independientes, seria algo como “ControladorProduccion” Encargado de manejar los procesos del área de fabricación, “ControladorInventario” Gestiona el inventario de materias primas y productos terminados
2. Principio de abierto-cerrado: Como se puede aplicar al proyecto es usar una herencia o una inyección de dependencias en el Backend para extender funcionalidades sin modificar el código base, por ejemplo puede haber una clase llamada “Reportes” y que de ellas salgan 2 subclases “ReporteProduccion” y “ReporteInventario” que extienden de la clase “Reportes” sin modificarla