TRABAJO FINAL UNIDAD #2

PATRONES DE DISEÑO DE SOFTWARE

ESTUDIANTES:

ING. JERSON RODRÍGUEZ

ING. ANA MARTINEZ

ING. JEAN PEDROZO

C.P DANIEL CONTRERAS

DOCENTE:

ING. JAIRO SEOANES

UNIVERSIDAD POPULAR DEL CESAR

FACULTAD DE INGENIERÍAS Y TECNOLOGÍAS

ESPECIALIZACIÓN INGENIERÍA DEL SOFTWARE

2025-I

TABLA DE CONTENIDO

INTRODUCCIÓN 3
OBJETIVOS 4
OBJETIVO GENERAL 4
OBJETIVOS ESPECÍFICOS 4
ANÁLISIS PRELIMINAR 4
PLANTEAMIENTO DEL PROBLEMA: 4
ELICITACIÓN DE LOS REQUERIMIENTOS 5
- Requisitos funcionales: 5
- Requisitos no funcionales 6

DESCRIPCIÓN DE LA SOLUCIÓN 7
DETALLE DE LA APLICACIÓN 7
DISEÑO DE LA ARQUITECTURA DE LA SOLUCIÓN 7
Esta división facilita: 7
Esta estructura permite: 8
Ejemplo de microservicios definidos: 8
APLICACIÓN DE PATRONES DE DISEÑO EN LA SOLUCIÓN PROPUESTA 8
Patrones creacionales: 8
Patrones estructurales: 9
Diagrama de clases 10
Diagrama de componentes 11
IMPLEMENTACION DE LA SOLUCION 12
Frontend: 12
Backend: 12
Componentes adicionales: 13

INTRODUCCIÓN

Nos encontramos en una época en donde la tecnología sigue estando en auge y el no hacernos con ella nos garantiza un retraso a nivel general en nuestras vidas. Lo mismo sucede con las empresas, si no se adaptan a los cambios y empiezan a cambiar su mentalidad y a adoptar la tecnología dentro de su propósito de su actividad comercial, y a generar esa cultura digital dentro de su ambiente y su entorno, muy probablemente esa empresa está condenando su futuro al fracaso.

Este proyecto, se enfoca en un emprendimiento que ha surgido en la ciudad de Aguachica, donde su actividad económica se basa en la gestión de domicilios entre empresas y consumidores finales. Se analiza desde una perspectiva estructural la actividad comercial de este negocio y se evidencia que la forma de operar es muy metódica y manual, haciendo que su actividad a diario pueda estar propensa a errores y fallos humanos, además de la poca seguridad y la falta de organización al momento de ofrecer sus servicios.

Se plantea una solución a la problemática de esta empresa, proponiendo el desarrollo de una aplicación web que se encargue de gestionar el proceso y otorgue una mayor facilidad operativa. Este desarrollo se compone de los requerimientos que necesita la aplicación y que son claves para el negocio, la arquitectura de software más ajustada al contexto de la aplicación, modelos y patrones de diseño de software para que el desarrollo garantice que la aplicación sea escalable y mantenible en el tiempo; todo esto con el fin de que la empresa pueda desarrollarse en el tiempo y obtenga una mejor posición frente a su competencia.

OBJETIVOS

OBJETIVO GENERAL

Proponer una solución a la problemática actual que surge del no acoplamiento de la empresa de domicilios a desarrollos tecnológicos que ayuden a mejorar la calidad de su servicio y la forma de operar.

OBJETIVOS ESPECÍFICOS

- Analizar la operación actual de la empresa para identificar falencias y necesidades.
- Realizar la elicitación de los requerimientos fundamentales que necesita la aplicación para que la empresa pueda operar.
- Proponer técnicas (como la arquitectura y los patrones de diseño) que ayuden al desarrollo óptimo de la aplicación y promuevan su escalabilidad y mantenibilidad.
- Determinar el stack de tecnologías más acorde para el desarrollo de la aplicación.

ANÁLISIS PRELIMINAR

PLANTEAMIENTO DEL PROBLEMA:

Hoy en día, el surgir y el éxito de una empresa va de la mano con su acople a la tecnología y a las herramientas tecnológicas que incluyan en su labor comercial, ya que resulta beneficioso en esfuerzo y dinero y puede ayudar a que la compañía tenga gran potencial a futuro.

Actualmente, en la región se está dando el surgimiento de una nueva empresa de domicilios, donde su actividad principal es la gestión y la distribución de domicilios actuando como puente entre los comerciantes y el público en general. Pero a diferencia de lo mencionado anteriormente, esta empresa se ve que utiliza métodos “arcaicos” para llevar a cabo la realización de su actividad económica; toda su gestión y operación logística la realizan mediante chats de WhatsApp, no existe una gestión de los datos tanto de los pedidos, como de los clientes, ni tampoco de los domiciliarios; algunos domiciliarios tienen que pagar los domicilios a los comercios y luego esperar a que el usuario les reintegre el dinero (se ve que se abre una brecha de seguridad aquí), los domiciliarios no poseen un sistema para poder consultar la ruta y los clientes no pueden tampoco consultar el estado del domicilio, entre otras cosas.

Dado que la empresa no posee una infraestructura tecnológica adecuada, se hace indispensable e indiscutible que esta empresa empiece a abordar este problema lo antes posible, así podrá mejorar el servicio que presta, se hará más fácil la realización de su actividad financiera, y le podrá generar una ventaja frente a la competencia.

ELICITACIÓN DE LOS REQUERIMIENTOS

- Requisitos funcionales:

ID	RF01
Nombre	Gestión de usuarios.
Desc.	El sistema debe permitir la gestión básica de usuarios (crear, listar, actualizar y desactivar) para que se pueda llevar un control.
Prio.	Alta

ID	RF02
Nombre	Gestión de roles.
Desc.	El sistema debe permitir la gestión de roles de los usuarios cuando se están creando/registrando para que su interacción dentro del sistema sea acorde con el usuario.
Prio.	Alta

ID	RF03
Nombre	Gestión de rutas.
Desc.	El sistema debe permitir la asignación, modificación y eliminación de rutas para que los usuarios de tipo domiciliarios puedan guiarse y los usuarios generales puedan saber dónde se encuentra su pedido.
Prio.	Alta

ID	RF04
Nombre	Gestión de pedidos.
Desc.	El sistema debe permitir la creación, eliminación, modificación y consulta de los pedidos que se realizan para que se puedan procesar correctamente.
Prio.	Alta

ID	RF05
Nombre	Gestión de reportes.
Desc.	El sistema debe permitir la generación, modificación y eliminación de reportes en base a la información contenida dentro del mismo, para que los administradores y analistas puedan trabajar sobre estos y tomar decisiones.
Prio.	Baja

ID	RF06
Nombre	Pasarela de pagos.
Desc.	El sistema debe integrar una pasarela de pagos para extender sus opciones de pago a los clientes y que no solo sea en efectivo.
Prio.	Baja

ID	RF07
Nombre	Mecánica de créditos.
Desc.	El sistema debe de integrar una mecánica de créditos para las personas que decidan pagar al recibir el pedido.
Prio.	Baja

ID	RF08
Nombre	Inicio de sesión.
Desc.	El sistema debe permitir a los usuarios identificarse mediante un inicio de sesión para que puedan acceder a las funcionalidades.
Prio.	Alta

- Requisitos no funcionales

ID	RFN01
Nombre	Disponibilidad operativa del sistema.
Desc.	El sistema debe estar disponible los 7 días de la semana con una taza de tiempo operativo o funcional del 98%
Prio.	Alta

ID	RFN02
Nombre	Disponibilidad funcional del sistema.
Desc.	El sistema debe soportar el acceso concurrente de usuarios a los servicios y peticiones, como mínimo de 100 peticiones simultáneas por segundo.
Prio.	Alta

ID	RFN03
Nombre	Seguridad en los datos.
Desc.	El sistema debe garantizar que los datos son almacenados correctamente y que los datos sensibles son encriptados utilizando alguna técnica o algoritmo de cifrado para evitar que sean aprovechados en caso de una filtración.
Prio.	Alta

ID	RFN04
Nombre	Facilidad de uso.

Desc.	El sistema debe de permitir que los usuarios puedan utilizarlo de manera autónoma después de 2 días de uso, además debe de incorporar pequeñas descripciones sobre cada botón o funcionalidad que este posea.
Prio.	Alta

DESCRIPCIÓN DE LA SOLUCIÓN

DETALLE DE LA APLICACIÓN

Es una aplicación de tipo web, en donde la empresa de domicilio podrá gestionar su operación sin importar el dispositivo (PC o smartphone) ya que está enfocada a un diseño responsivo, además contando con diferentes módulos y roles que permiten interactuar de manera dinámica con las funcionalidades; un cliente puede solicitar un servicio de domicilio, mientras que un administrador puede gestionar los pedidos, y un domiciliario puede procesar los pedidos y ver las rutas.

DISEÑO DE LA ARQUITECTURA DE LA SOLUCIÓN

Dado el contexto y las necesidades específicas de la empresa de domicilios descritas en el planteamiento del problema, se propone una arquitectura basada en microservicios con enfoque hexagonal (Ports and Adapters). Esta decisión responde a criterios de escalabilidad, mantenibilidad, independencia tecnológica y facilidad para la integración con servicios externos.

Arquitectura de microservicios: La elección de una arquitectura de microservicios permite dividir el sistema en módulos independientes que se comunican entre sí mediante API REST. Cada microservicio se encarga de una responsabilidad específica, como la gestión de usuarios, pedidos, rutas, pagos, reportes o autenticación.

Esta división facilita:

- Escalabilidad horizontal: Se puede aumentar la capacidad de un módulo específico sin afectar el resto del sistema.
- Mantenimiento y despliegue independiente: Cada servicio puede actualizarse, probarse y desplegarse de forma aislada.
- Tolerancia a fallos: Un fallo en un microservicio no afecta directamente a los demás, lo que mejora la disponibilidad general del sistema.

Arquitectura hexagonal (Ports and Adapters): Sobre cada microservicio se adopta el enfoque de arquitectura hexagonal, que propone una separación clara entre:

- El núcleo del dominio (reglas de negocio),

- Los puertos (interfaces de entrada y salida),
- Y los adaptadores (implementaciones concretas como APIs REST, bases de datos o servicios externos).

Esta estructura permite:

- Alta cohesión en la lógica de negocio: Las reglas del sistema permanecen independientes de detalles tecnológicos.
- Facilidad para pruebas automatizadas: Al desacoplar la lógica del sistema de las tecnologías externas.
- Sustitución sencilla de tecnologías externas: Por ejemplo, cambiar de proveedor de pagos sin afectar el núcleo del sistema.

Ejemplo de microservicios definidos:

- Servicio de autenticación y gestión de usuarios
- Servicio de gestión de pedidos
- Servicio de rutas y logística
- Servicio de reportes
- Servicio de integración de pasarela de pagos

Cada uno puede contar con su propia base de datos, respetando el principio de encapsulamiento, y todos estarán coordinados por una API Gateway o fachada (Facade) que sirve como único punto de entrada para clientes móviles y web.

APLICACIÓN DE PATRONES DE DISEÑO EN LA SOLUCIÓN PROPUESTA

Para el diseño e implementación del sistema de gestión para la empresa de domicilios, se han considerado únicamente aquellos patrones de diseño creacionales y estructurales que aportan un valor directo y funcional a las necesidades específicas del proyecto, evitando complejidad innecesaria. A continuación, se detallan los patrones seleccionados junto con su respectiva justificación técnica.

Patrones creacionales:

1. **Factory Method:** Este patrón se utilizará para la generación de distintos tipos de reportes (por ejemplo, PDF, Excel o JSON), permitiendo que el sistema pueda crear objetos sin acoplarse a clases concretas. Esta abstracción facilita la incorporación de nuevos formatos de reporte en el futuro sin necesidad de modificar el núcleo de la lógica de negocio.
2. **Singleton:** Se aplicará este patrón en componentes que deben tener una única instancia compartida dentro del sistema, como el gestor de configuración o el servicio de registro de logs. Esto garantiza un acceso global y coherente a servicios que no requieren múltiples instancias, mejorando el rendimiento y la integridad de los datos.

3. **Builder:** Será implementado en la construcción de pedidos complejos, los cuales pueden incluir múltiples elementos como productos, dirección de entrega, opciones de pago y seguimiento. El patrón Builder permite crear estos objetos paso a paso, de forma flexible y controlada, manteniendo la legibilidad del código y facilitando su mantenimiento.

Patrones estructurales:

1. **Adapter:** Este patrón permitirá la integración con servicios externos, como pasarelas de pago (ej. Stripe, PayU o MercadoPago), adaptando sus interfaces para que sean compatibles con la lógica interna del sistema. De este modo, se garantiza una comunicación fluida sin comprometer el diseño de la aplicación.
2. **Facade:** Se empleará para encapsular la complejidad de las operaciones internas de los distintos microservicios, exponiendo una interfaz única y simplificada para el consumo del frontend (aplicaciones móviles o web). Esto mejora la experiencia del usuario y reduce la dependencia entre capas del sistema.
3. **Decorator:** Será utilizado para extender dinámicamente la funcionalidad de los pedidos sin alterar su estructura base. Por ejemplo, se podrá añadir seguimiento en tiempo real, confirmaciones personalizadas o seguros adicionales, manteniendo la flexibilidad del sistema ante nuevos requerimientos.

Diagrama de clases

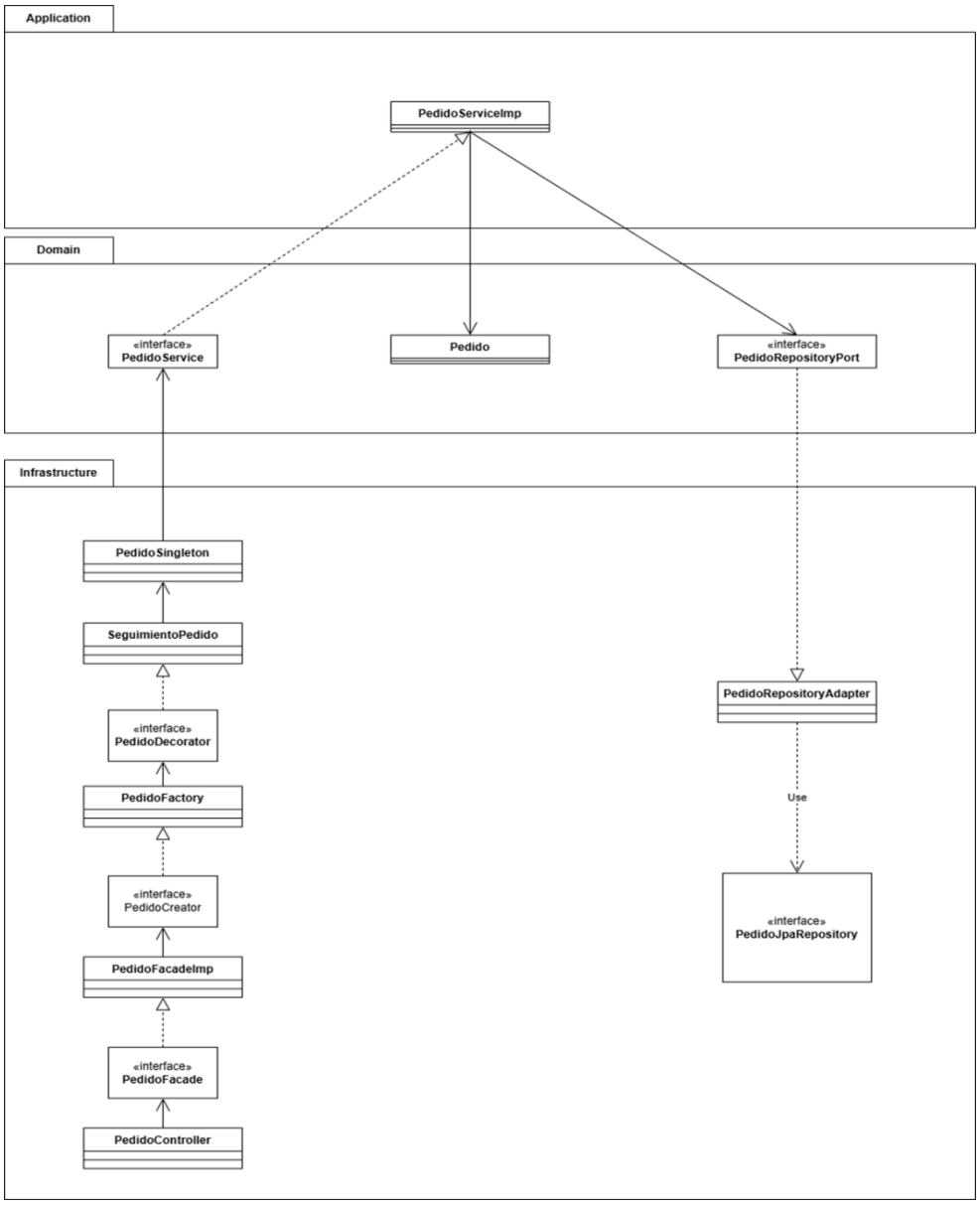
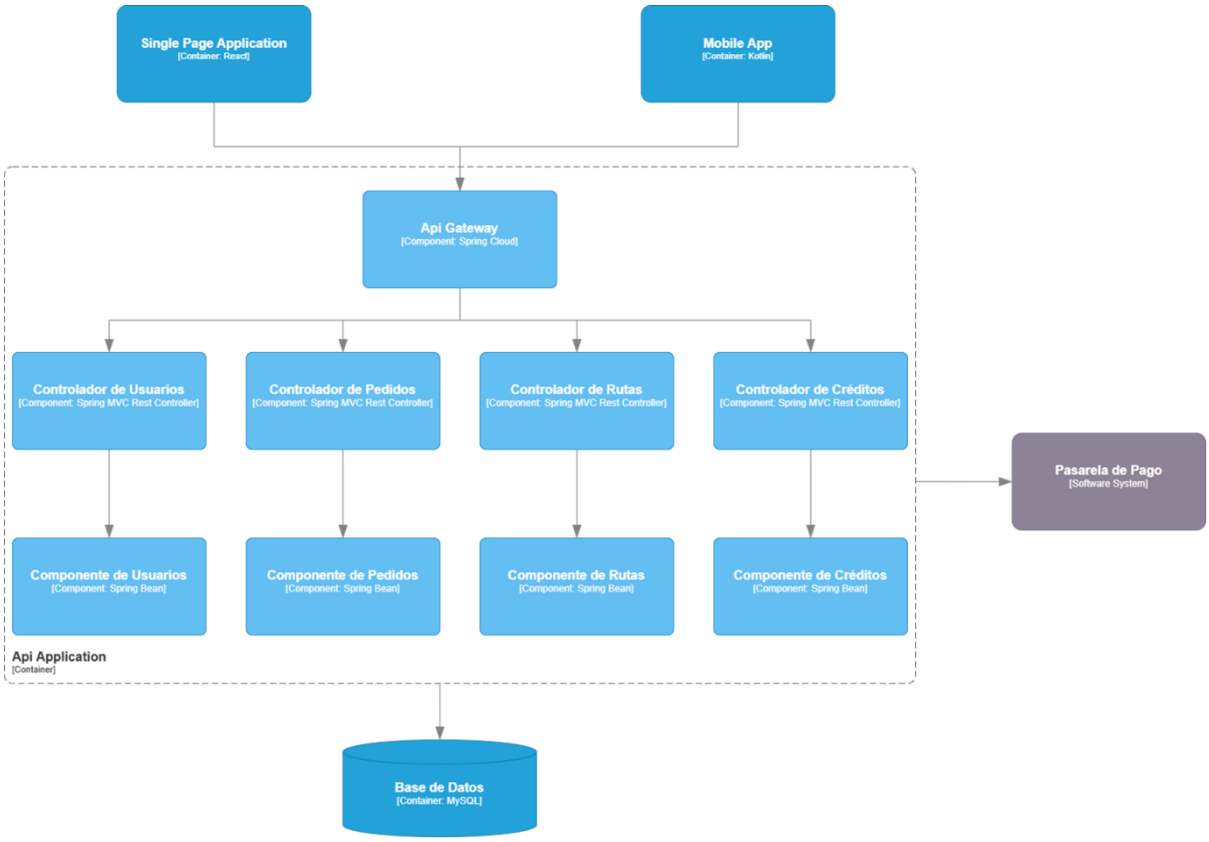


Diagrama de componentes



IMPLEMENTACION DE LA SOLUCION

Descripción: Para el desarrollo de esta aplicación se ha seleccionado un stack escalable y moderno que ofrece una experiencia de usuario fluida en múltiples dispositivos, así como una arquitectura que facilite la gestión de roles y módulos. El stack combina tecnologías frontend y backend que destacan por su rendimiento y soporte con patrones de diseño.

Frontend:

- **Lenguaje:** Typescript
  Agrega una sintaxis adicional a javascript agregando tipado y ayuda a reducir errores en tiempo de ejecución.

- **Framework:** Next.js 14
  Permite SSR (Server-Side Rendering) y SSG (Static Side Generation), importantes para mejorar el SEO y su rendimiento.

- **Gestor de dependencias:** npm
  Funciona como un repositorio con una amplia compatibilidad con el ecosistema Javascript/Typescript.

- **Framework de estilos:** Tailwind CSS
  Permite construir aplicaciones responsivas de manera rápida y flexible.

- Manejo de estado: Redux Toolkit
  Permite gestionar el estado global de la aplicación, importante para manejar múltiples roles y módulos.

Backend:

- **Lenguaje:** Typescript
  Mejora la seguridad del código, su mantenibilidad y permite detectar errores de forma temprana gracias a su tipado estático reduciendo la posibilidad de errores en ambientes productivos

- **Framework:** NestJs
  Permite crear aplicaciones eficientes y escalables del lado del servidor. Su soporte para controladores, servicios, inyección de dependencias, middlewares y uso de decoradores lo hacen ideal para proyectos escalables y mantenibles.

- **Gestor de dependencias:** npm

Facilita la administración de librerías externas

- **ORM:** TypeORM
  Es el ORM (Object Relational Mapping) mas completo disponible para Typescript. Al estar escrito en Typescript se integra perfectamente con el framework NestJs.

- **Base de datos:** PostgreSQL:
  Es una base de datos robusta, confiable y con una gran cantidad de funciones y extensiones para crear bases de datos altamente escalables y fáciles de administrar.

Componentes adicionales:

- **Autenticación y autorización:** Keycloak
  Ideal para manejar múltiples roles y permisos diferenciados.

- **Control de versiones:** Git y GitHub
  Es el estándar de la industria para trabajo colaborativo y versionamiento del código.