Backlog del Proyecto Java Backend - Arka

1. Introducción

Este backlog contiene las historias de usuario que se desarrollarán a lo largo del curso para la implementación del backend del sistema de Arka. Su objetivo es garantizar que la empresa pueda gestionar eficientemente su inventario, órdenes de compra y ventas, optimizando así sus operaciones.

---

2. Historias de Usuario

Módulo 1: Gestión de Inventario y Abastecimiento

HU1 - Registrar productos en el sistema
Como administrador, quiero registrar nuevos productos con sus características para que los clientes puedan comprarlos.

- **Criterios de aceptación:**
  - Se debe permitir la carga de nombre, descripción, precio, stock y categoría.
  - Validaciones de datos requeridos.
  - Mensaje de confirmación tras el registro exitoso.

HU2 - Actualizar stock de productos
Como administrador, quiero actualizar la cantidad de productos en stock para evitar sobreventas.

- **Criterios de aceptación:**
  - El sistema debe permitir modificar el stock de un producto.
  - No se deben permitir valores negativos.
  - Historial de cambios en el stock.

HU3 - Generar reportes de productos por abastecer
Como administrador, quiero recibir reportes de productos con bajo stock para tomar decisiones de abastecimiento.

- **Criterios de aceptación:**
  - El reporte debe generarse automáticamente cada semana.
  - Debe incluir productos con stock menor a un umbral configurable.
  - Exportación en formato CSV o PDF.

Módulo 2: Gestión de Órdenes de Compra

HU4 - Registrar una orden de compra
Como cliente, quiero poder registrar una orden de compra con múltiples productos para realizar mi pedido.

- **Criterios de aceptación:**
  - Se debe validar la disponibilidad del stock.
  - Registro de fecha y detalles del pedido.
  - Mensaje de confirmación con resumen del pedido.

HU5 - Modificar una orden de compra
Como cliente, quiero modificar mi pedido antes de su confirmación para corregir errores o agregar productos.

- **Criterios de aceptación:**
  - Solo se pueden modificar pedidos en estado 'pendiente'.
  - Se debe actualizar el stock en caso de eliminación de productos.

HU6 - Notificación de cambio de estado de pedido
Como cliente, quiero recibir notificaciones sobre el estado de mi pedido para estar informado de su progreso.

- **Criterios de aceptación:**
  - Notificación por correo o en la plataforma.
  - Estados: pendiente, confirmado, en despacho, entregado.

Módulo 3: Reportes y Análisis de Ventas

HU7 - Generar reportes de ventas semanales
Como administrador, quiero generar reportes semanales de ventas para analizar el rendimiento del negocio.

- **Criterios de aceptación:**
  - El reporte debe incluir total de ventas, productos más vendidos y clientes más frecuentes.
  - Exportación en formato CSV o PDF.

HU8 - Identificar carritos abandonados
Como administrador, quiero visualizar los carritos abandonados para contactar a los clientes y recuperar ventas.

- **Criterios de aceptación:**
  - Listado de carritos abandonados con fecha y productos.
  - Opción de enviar recordatorio por correo al cliente.

3. Priorización

Se recomienda priorizar las historias de usuario de la siguiente manera:

Historia de Usuario	Prioridad
HU1 - Registrar productos	Alta
HU2 - Actualizar stock	Alta
HU4 - Registrar orden de compra	Alta
HU6 - Notificación de cambio de estado	Alta
HU7 - Reporte de ventas semanales	Media
HU8 - Carritos abandonados	Media
HU5 - Modificar orden de compra	Baja
HU3 - Reporte de productos por abastecer	Baja

4. Conclusión

Este backlog proporciona una estructura clara para el desarrollo del backend del sistema de Arka. Con estas historias de usuario y su priorización, se garantizará la entrega de un producto funcional que atienda las necesidades principales del negocio.

A continuación, se desarrollarán los diagramas gráficos requeridos.

He creado el backlog con las historias de usuario organizadas en módulos, sus criterios de aceptación y la priorización para su desarrollo. Ahora procederé a diseñar los **diagramas gráficos**:

1. **Diagrama de base de datos** (estructura de información).
2. **Diagrama de infraestructura** (despliegue y componentes).
3. **Diagrama de arquitectura** (modelo de capas y servicios).

Diagrama de Base de Datos del sistema de Arka. Representa las entidades principales y sus relaciones, incluyendo productos, clientes, órdenes, carritos de compras y detalles de pedidos.

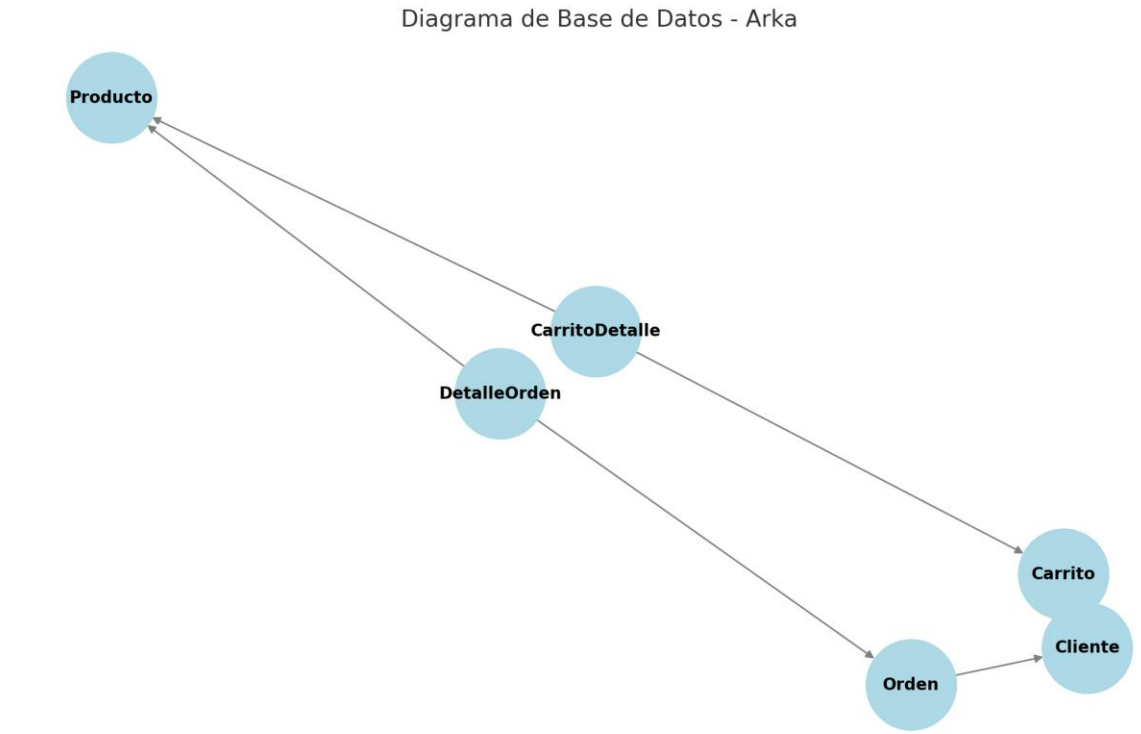


Diagrama de Infraestructura del sistema de Arka. Muestra la interacción entre los diferentes componentes del sistema, incluyendo el **Frontend, API Gateway, Servicios de Backend, Base de Datos y Proveedores Externos**.

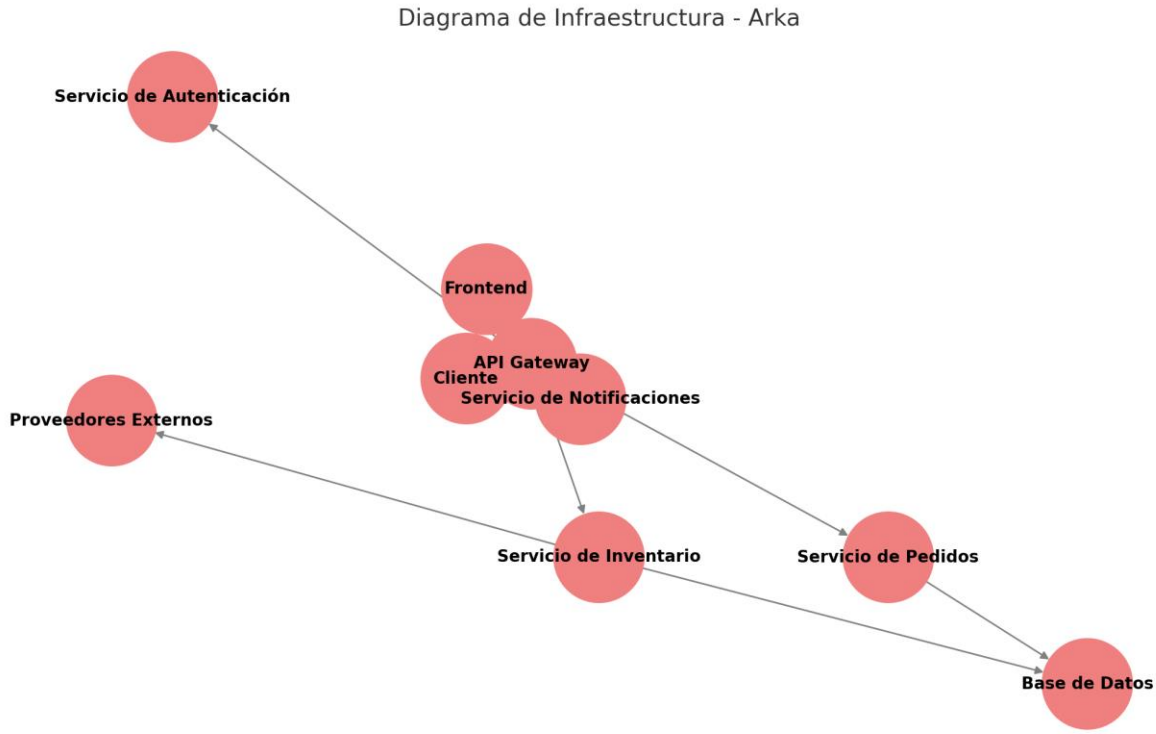
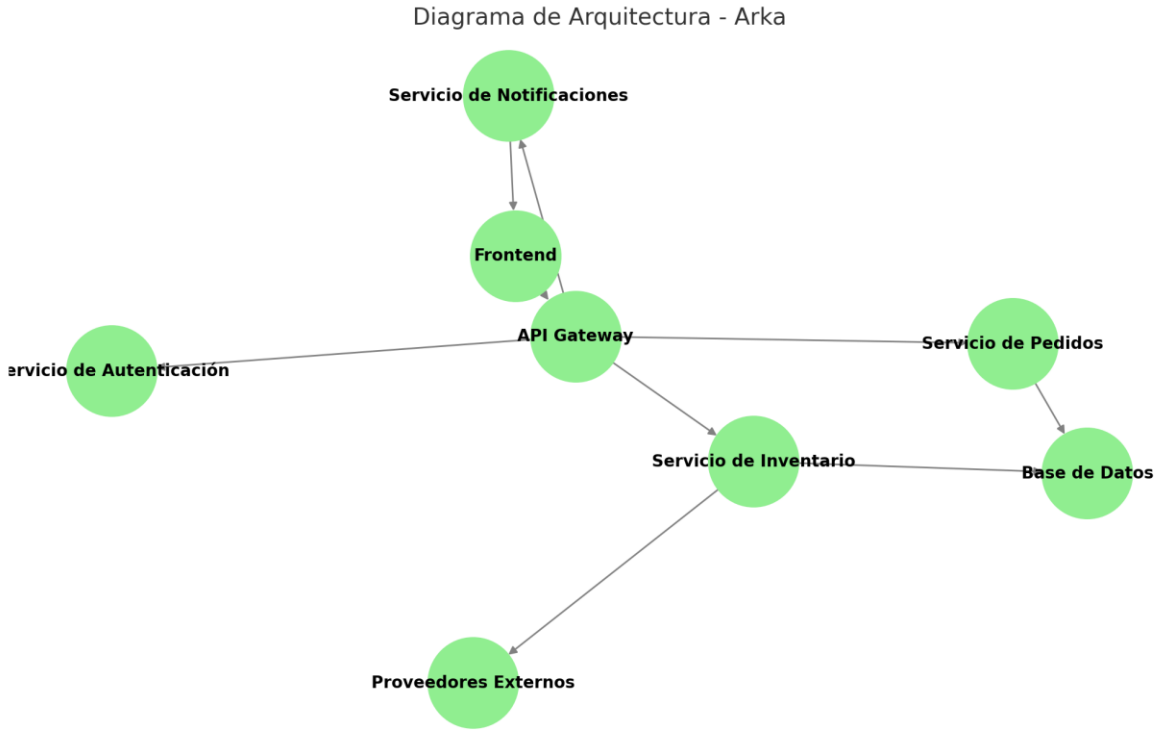


Diagrama de Arquitectura del sistema de Arka. Representa las diferentes capas del sistema:

1. **Capa de Presentación**: Donde se encuentra el **Frontend**.

2. **Capa de Aplicación**: Incluye el **API Gateway** y los diferentes **Servicios Backend** como autenticación, inventario, pedidos y notificaciones.
3. **Capa de Datos**: Contiene la **Base de Datos** y la conexión con **Proveedores Externos**.



Plan de acción

Para reducir la incertidumbre y facilitar el trabajo de los estudiantes, podemos proporcionarles una **base de datos preconfigurada** que ya esté alineada con la arquitectura de microservicios y lista para su despliegue en AWS. Aquí hay algunas acciones concretas que podemos tomar:

1. Entregar una Base de Datos Lista para Usar

- **Opción 1: Archivo SQL predefinido**
  - Crear un **script SQL** con las estructuras de tablas, relaciones y datos de prueba.
  - Asegurar que sea compatible con **PostgreSQL o MySQL**, que son comunes en AWS RDS.
- **Opción 2: Base de Datos en AWS RDS**
  - Configurar una instancia de **Amazon RDS** con la estructura de datos ya creada.
  - Proporcionar credenciales temporales para que los estudiantes puedan conectarse fácilmente.

2. Explicar Cómo Se Relaciona con Microservicios

- Crear una guía o diagrama explicando **cómo cada microservicio interactúa con la base de datos**.
- Implementar **una estrategia de bases de datos por microservicio**:
    - Opción 1: **Base de datos única con esquemas separados** para cada servicio.
    - Opción 2: **Cada microservicio tiene su propia base de datos** y se comunican vía APIs.

3. Proporcionar Código Base y Ejemplos

- Subir un **repositorio en GitHub con un microservicio base** que ya incluya:
    - Conexión a la base de datos.
    - Consultas básicas (crear, leer, actualizar, eliminar - CRUD).
    - Ejemplo de integración con **Spring Boot y AWS RDS**.

4. Sesión de Re-alineación

- Organizar una **sesión en vivo** donde expliquemos cómo usar la base de datos y resolver dudas.
- Mostrar **cómo conectar los microservicios a la base de datos** en AWS.

Voy a generar el **script SQL inicial** y el **código de conexión a una base de datos en AWS** para que los estudiantes tengan una base sólida.

https://chatgpt.com/canvas/shared/67bdeb0ff61481919444022090ba9c5b

```
package com.arka.config;


import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;


import javax.persistence.EntityManagerFactory;

import javax.sql.DataSource;
```

```
import org.springframework.boot.jdbc.DataSourceBuilder;


@Configuration
@EnableJpaRepositories(basePackages = "com.arka.repository")
public class DatabaseConfig {


    @Value("${aws.db.url}")
    private String dbUrl;


    @Value("${aws.db.username}")
    private String dbUsername;


    @Value("${aws.db.password}")
    private String dbPassword;


    @Bean
    public DataSource dataSource() {
        return DataSourceBuilder.create()
                .url(dbUrl)
                .username(dbUsername)
                .password(dbPassword)
                .driverClassName("org.postgresql.Driver")
                .build();
    }


    @Bean
```

```
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {

        LocalContainerEntityManagerFactoryBean factoryBean = new
LocalContainerEntityManagerFactoryBean();

        factoryBean.setDataSource(dataSource());

        factoryBean.setPackagesToScan("com.arka.model");

        factoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

        return factoryBean;

    }


    @Bean

    public JpaTransactionManager transactionManager(EntityManagerFactory
entityManagerFactory) {

        return new JpaTransactionManager(entityManagerFactory);

    }

}
```