



Segunda Evaluación

Perfulandia SPA

Curso: DSY_1103

Docente: María
Angeles Robinson

EQUIPO Y ROLES



JEFE DE
PROYECTO

BASTIAN
REYES



FULLSTACK

CRISTIAN
LIZAMA



DEVOPS

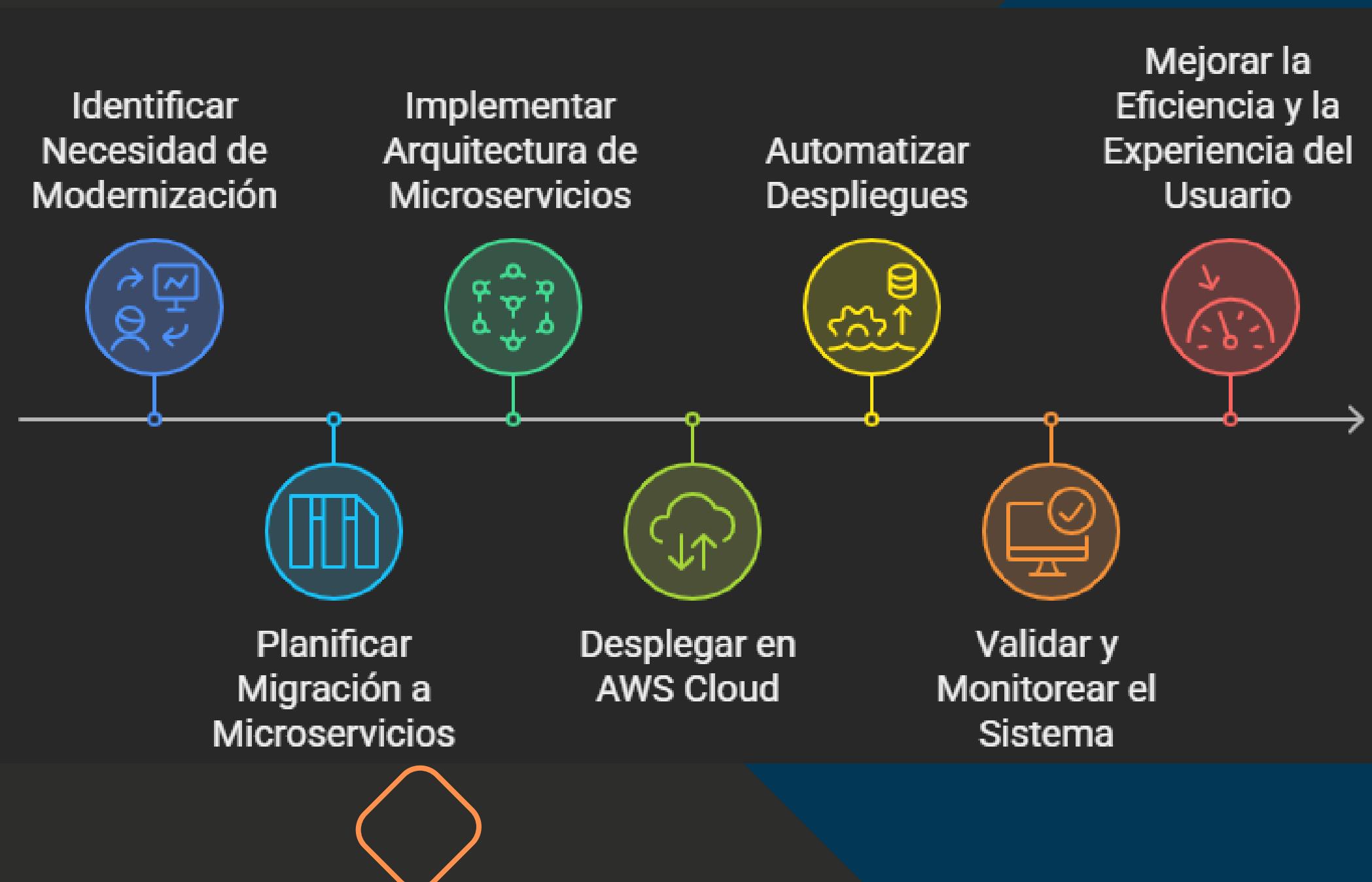
JOSHUA
MARDONES



INTRODUCCIÓN

El proyecto consiste en la modernización del sistema informático de Perfulandia SPA, migrando de una arquitectura monolítica a una basada en microservicios desplegados en la nube de AWS. Esta transformación permite una mayor escalabilidad, rendimiento y facilidad de mantenimiento.

El objetivo principal es mejorar la eficiencia y la experiencia del usuario, implementando un sistema distribuido utilizando tecnologías como Spring Boot, Maven, JPA y herramientas de automatización para despliegues, validación y monitoreo del sistema en tiempo real.



ARQUITECTURA DE MICROSERVICIOS

Capas del Microservicio



Capa de Entidad

Representa la estructura de los datos, mapeando las tablas de la base de datos a objetos Java.

Se encarga del acceso a la base de datos utilizando CrudRepository para operaciones simplificadas.



Capa de Repositorio



Concentra la lógica del negocio, definiendo métodos utilizados por el controlador.

Expone la API REST utilizando anotaciones como RestController.



Capa de Servicio

Capa de Controlador



```
@Controller  
public class perfulandiaController {  
  
    @Autowired  
    private UsuarioRepository usuarioRepository;  
  
    @Autowired  
    private PedidoRepository pedidoRepository;  
  
    @Autowired  
    private ProductoRepository productoRepository;  
  
    @Autowired  
    private GerenteSucursalRepository gerentesucursalRepository;  
  
    @Autowired  
    private SucursalRepository sucursalRepository;
```

```
Entity  
ublic class Producto {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long idProducto;  
    private String nombreProducto;  
    private String descripcionProducto;  
    private double precioProducto;  
    private int cantidadStock;  
    private LocalDateTime fechaIngreso;  
    ...  
    package com.duoc.springboot.api.fullrest.services;  
    ...  
    import java.util.List;  
    import java.util.Optional;  
    ...  
    import com.duoc.springboot.api.fullrest.entities.Producto;  
    ...  
    public interface productoService {  
        ...  
        List<Producto> findAll();  
        Optional<Producto> findById(Long id);  
        Producto save(Producto producto);  
        Optional<Producto> delete(Producto producto);  
    }  
}
```

```
> Users > basti > Desktop > Evaluacion > Perfulandia > src > main > java > com > duoc > springboot > api  
1 package com.duoc.springboot.api.fullrest.repositories;  
2  
3 import org.springframework.data.repository.CrudRepository;  
4 import com.duoc.springboot.api.fullrest.entities.Producto;  
5  
6 public interface ProductoRepository extends CrudRepository<Producto, Long> {  
7  
8 }
```

DISEÑO DEL PROYECTO CON SPRING

1. Spring Boot estructura el backend en capas: Controller, Service, Repository, Entity.

Cada clase cumple una función específica dentro de la arquitectura del sistema:

Controller: recibe las solicitudes HTTP del cliente, define las rutas y delega las tareas al servicio correspondiente. Usa anotaciones como `@RestController` y `@RequestMapping`.

Service: contiene la lógica de negocio. Aquí se procesan los datos antes de enviarlos al repositorio o de responder al controlador. Se anota con `@Service` y puede incluir `@Transactional`.

Repository: interactúa con la base de datos utilizando Spring Data JPA. Extiende interfaces como `CrudRepository` y permite ejecutar operaciones CRUD sin necesidad de escribir SQL.

Entity: representa las tablas de la base de datos como clases Java. Usa anotaciones como `@Entity`, `@Id` y `@GeneratedValue`.

GESTIÓN CON MAVEN

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
```

spring-boot-starter-actuator:

Permite monitorear y administrar la aplicación mediante endpoints como /actuator/health y /actuator/metrics.

spring-boot-starter-data-jpa:

Facilita la conexión entre objetos Java y bases de datos relacionales usando JPA, eliminando la necesidad de escribir consultas SQL.

spring-boot-starter-thymeleaf:

Habilita el uso del motor de plantillas Thymeleaf para generar páginas HTML dinámicas desde el backend.

spring-boot-starter-web:

Incluye todo lo necesario para crear aplicaciones web y APIs REST, incluyendo Spring MVC y el servidor embebido Tomcat.

spring-boot-devtools:

Acelera el desarrollo recargando automáticamente la aplicación cuando se detectan cambios en el código.

mysql-connector-j:

Es el driver JDBC que permite conectar la aplicación con una base de datos MySQL.

spring-boot-starter-test:

Proporciona herramientas de testing como JUnit y Mockito para escribir y ejecutar pruebas automatizadas.



Que es hibernate?

Para que sirve?

CONCLUSIÓN

En la segunda etapa del proyecto de modernización de Perfulandia SPA, se implementaron con éxito tres microservicios: gerente sucursal, pedido, producto, sucursal, usuario. Estos microservicios reemplazan la arquitectura monolítica anterior, aportando flexibilidad, escalabilidad y facilidad de mantenimiento. Cada microservicio cuenta con operaciones CRUD completas y está conectado a una base de datos MySQL.

El microservicio de productos fue desarrollado usando tecnologías como Spring Boot, Spring Data JPA y Maven, siguiendo buenas prácticas de desarrollo backend para garantizar un diseño limpio, reutilizable y preparado para futuras mejoras. Las funcionalidades fueron validadas usando Postman.

Durante el proceso, se aplicaron prácticas de trabajo colaborativo como control de versiones con Git, uso de GitHub y documentación detallada. También se desarrollaron vistas básicas para visualizar los datos desde las APIs, facilitando la interacción con los servicios implementados.



GIT Y GITHUB



DuocUC



GRACIAS
POR SU ATENCIÓN

