Universidad Católica Andrés Bello

Facultad de Ingeniería

Escuela de Ingeniería Informática

Periodo Académico 2024-2025

Sistema de Reservas de Laboratorio con Arquitectura de Microservicios

Asignatura: Computación en la Nube (CEN)

Profesor: Nombre del Profesor

Estudiante: Nombre del Estudiante

Diciembre 2024

Índice

1. Introducción

El Sistema de Reservas de Laboratorio es una aplicación web desarrollada con arquitectura de microservicios que permite a los estudiantes y profesores gestionar las reservas de computadoras en los laboratorios de la universidad. Este sistema ha sido diseñado siguiendo los principios de la computación en la nube y las mejores prácticas de desarrollo de software moderno.

El proyecto implementa una arquitectura distribuida basada en microservicios, donde cada servicio tiene una responsabilidad específica y bien definida. La comunicación entre servicios se realiza a través de APIs REST, y todo el sistema está containerizado usando Docker y orquestado con Kubernetes.

1.1. Objetivos del Proyecto

- **Objetivo Principal**: Desarrollar un sistema completo de reservas de laboratorio utilizando arquitectura de microservicios.
- **Objetivos Específicos**:
  - Implementar una arquitectura de microservicios con separación de responsabilidades
  - Utilizar tecnologías de containerización con Docker
  - Implementar orquestación con Kubernetes
  - Desarrollar APIs REST bien documentadas
  - Garantizar la persistencia de datos con PostgreSQL
  - Implementar un API Gateway para la gestión centralizada de peticiones

1.2. Tecnologías Utilizadas

- **Backend**: Node.js con Express.js
- **Base de Datos**: PostgreSQL
- **Containerización**: Docker y Docker Compose
- **Orquestación**: Kubernetes
- **API Gateway**: Node.js con Express.js
- **Documentación**: Postman Collections
- **Control de Versiones**: Git

2. Análisis del Problema

2.1. Problemática Identificada

La gestión manual de reservas de laboratorios presenta varios desafíos:

1. **Falta de Centralización**: No existe un sistema unificado para gestionar las reservas de todos los laboratorios.
2. **Conflictos de Horarios**: Es común que se produzcan dobles reservas o conflictos de horarios.
3. **Información Desactualizada**: La información sobre disponibilidad de computadoras no está actualizada en tiempo real.
4. **Proceso Manual**: El proceso de reserva requiere intervención manual, lo que genera demoras.
5. **Falta de Trazabilidad**: No existe un registro histórico de las reservas realizadas.

2.2. Justificación de la Arquitectura de Microservicios

La elección de una arquitectura de microservicios se justifica por:

1. **Separación de Responsabilidades**: Cada servicio maneja un dominio específico
2. **Escalabilidad Independiente**: Cada servicio puede escalarse según sus necesidades
3. **Desarrollo Paralelo**: Equipos pueden trabajar independientemente en cada servicio
4. **Tolerancia a Fallos**: El fallo de un servicio no afecta a todo el sistema
5. **Facilidad de Mantenimiento**: Cambios en un servicio no impactan a otros

3. Diseño de la Solución

3.1. Arquitectura del Sistema

El sistema está compuesto por los siguientes microservicios:

1. **API Gateway - Backend (Puerto 3001)**:
   - Punto de entrada único para todas las peticiones
   - Enrutamiento a los microservicios correspondientes
   - Manejo de CORS y políticas de seguridad
   - Expuesto externamente en NodePort 30001

2. **Backend-Persons (Puerto 3002)**:
   - Gestión de usuarios y perfiles
   - Validación de datos de usuarios
   - Operaciones CRUD para usuarios
3. **Backend-Computers (Puerto 3004)**:
   - Gestión de laboratorios y computadoras
   - Control de disponibilidad de equipos
   - Relaciones entre laboratorios y computadoras
4. **Backend-Reservations (Puerto 3003)**:
   - Gestión del sistema de reservas
   - Validación de disponibilidad
   - Control de estados de reserva

3.2. Diagrama de Arquitectura

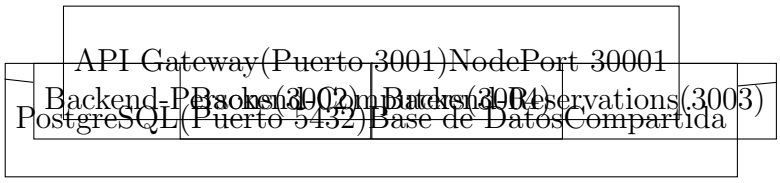

Figura 1: Arquitectura del Sistema de Microservicios

3.3. Modelo de Datos

3.3.1. Entidad User (Backend-Persons)

```
{
  id: number,
  nombre: string,
  email: string,
  password: string,
  cedula: string,
  created_at: timestamp,
  updated_at: timestamp
}
```

Listing 1: Modelo de Usuario

3.3.2. Entidad Lab (Backend-Computers)

```
{
  id: number,
  nombre: string,
  created_at: timestamp,
  updated_at: timestamp
}
```

Listing 2: Modelo de Laboratorio

3.3.3. Entidad Computer (Backend-Computers)

```
{
  id: number,
  nombre: string,
  lab_id: number,
  created_at: timestamp,
  updated_at: timestamp
}
```

Listing 3: Modelo de Computadora

3.3.4. Entidad Reservation (Backend-Reservations)

```
{
  id: number,
  user_id: number,
  computer_id: number,
  fecha: date,
  hora: time,
  duracion: number,
  estado: string,
  created_at: timestamp,
  updated_at: timestamp
}
```

Listing 4: Modelo de Reserva

3.4. Endpoints del Sistema

3.4.1. API Gateway Endpoints

Users Management:

- `GET /api/users` - Obtener todos los usuarios
- `GET /api/users/:id` - Obtener usuario por ID
- `POST /api/users` - Crear nuevo usuario
- `PUT /api/users/:id` - Actualizar usuario

- `DELETE /api/users/:id` - Eliminar usuario

Labs Management:

- `GET /api/labs` - Obtener todos los laboratorios
- `GET /api/labs/:id` - Obtener laboratorio por ID
- `POST /api/labs` - Crear nuevo laboratorio
- `PUT /api/labs/:id` - Actualizar laboratorio
- `DELETE /api/labs/:id` - Eliminar laboratorio
- `GET /api/labs/:labId/computers` - Obtener computadoras por laboratorio

Computers Management:

- `GET /api/computers` - Obtener todas las computadoras
- `GET /api/computers/:id` - Obtener computadora por ID
- `POST /api/computers` - Crear nueva computadora
- `PUT /api/computers/:id` - Actualizar computadora
- `DELETE /api/computers/:id` - Eliminar computadora

Reservations Management:

- `GET /api/reservations` - Obtener todas las reservas
- `GET /api/reservations/details` - Obtener reservas con detalles
- `GET /api/reservations/:id` - Obtener reserva por ID
- `POST /api/reservations` - Crear nueva reserva
- `PUT /api/reservations/:id` - Actualizar reserva
- `DELETE /api/reservations/:id` - Eliminar reserva
- `GET /api/reservations/user/:userId` - Obtener reservas por usuario
- `GET /api/reservations/computer/:computerId` - Obtener reservas por computadora
- `GET /api/reservations/status/:status` - Obtener reservas por estado

4. Implementación

4.1. Estructura del Proyecto

```
lab-reservation-system/
          backend/                    # API Gateway
                src/
                      computers.controller.ts
                      reservations.controller.ts
                      users.controller.ts
                      server.ts
                package.json
                Dockerfile
          backend-persons/           # Microservicio de Usuarios
                src/
                      routes.ts
                      types.ts
                      models.ts
                      server.ts
                package.json
                Dockerfile
          backend-computers/         # Microservicio de Computadoras
                src/
                      routes.ts
                      types.ts
                      models.ts
                      server.ts
                package.json
                Dockerfile
          backend-reservations/      # Microservicio de Reservas
                src/
                      routes.ts
                      types.ts
                      models.ts
                      server.ts
                package.json
                Dockerfile
          docker-compose.yml
          k8s/                       # Configuraciones Kubernetes
                configmaps.yaml
                deployments.yaml
                services.yaml
                postgres.yaml
          postman-collections.json
          endpoints.md
```

Listing 5: Estructura de Directorios

4.2. Configuración de Kubernetes

4.2.1. ConfigMaps

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: backend-config
data:
  PORT: "3001"
  DB_HOST: postgres
  DB_PORT: "5432"
  DB_USER: admin
  DB_NAME: mydb
  DB_PASS: admin
  BACKEND_PERSONS_SERVICE_PORT: "3002"
  BACKEND_COMPUTERS_SERVICE_PORT: "3004"
  BACKEND_RESERVATIONS_SERVICE_PORT: "3003"
  BACKEND_PERSONS_SERVICE_URL: "http://backend-persons:3002"
  BACKEND_COMPUTERS_SERVICE_URL: "http://backend-computers:3004"
  BACKEND_RESERVATIONS_SERVICE_URL: "http://backend-reservations:3003"

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-persons-config
data:
  PORT: "3002"
  DB_HOST: postgres
  DB_PORT: "5432"
  DB_USER: admin
  DB_NAME: mydb
  DB_PASS: admin
```

Listing 6: Configuración de ConfigMaps

4.2.2. Implementación de Microservicios

Backend-Persons (Gestión de Usuarios):

```
interface User {
  id: number;
  nombre: string;
  email: string;
  password: string;
  cedula: string;
  created_at: Date;
  updated_at: Date;
}

interface CreateUserRequest {
  nombre: string;
  email: string;
  password: string;
  cedula: string;
}
```

Listing 7: Interfaces de Usuario

Backend-Computers (Gestión de Laboratorios y Computadoras):

```
interface Lab {
  id: number;
  nombre: string;
  created_at: Date;
  updated_at: Date;
}

interface Computer {
  id: number;
  nombre: string;
  lab_id: number;
  created_at: Date;
  updated_at: Date;
}
```

Listing 8: Interfaces de Laboratorio y Computadora

Backend-Reservations (Gestión de Reservas):

```
interface Reservation {
  id: number;
  user_id: number;
  computer_id: number;
  fecha: string;
  hora: string;
  duracion: number;
  estado: "pendiente" | "confirmada" | "cancelada";
  created_at: Date;
  updated_at: Date;
}
```

Listing 9: Interface de Reserva

4.3. Configuración de Docker

4.3.1. Docker Compose

```
version: '3.8'

services:
  api-gateway:
    build: ./backend
    ports:
      - "3001:3001"
    environment:
      - NODE_ENV=production
      - BACKEND_PERSONS_URL=http://backend-persons:3002
      - BACKEND_COMPUTERS_URL=http://backend-computers:3004
      - BACKEND_RESERVATIONS_URL=http://backend-reservations:3003
    depends_on:
      - backend-persons
      - backend-computers
      - backend-reservations

  backend-persons:
    build: ./backend-persons
```

```
    ports:
      - "3002:3002"
    environment:
      - NODE_ENV=production
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_NAME=mydb
      - DB_USER=admin
      - DB_PASSWORD=admin
    depends_on:
      - postgres

  backend-computers:
    build: ./backend-computers
    ports:
      - "3004:3004"
    environment:
      - NODE_ENV=production
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_NAME=mydb
      - DB_USER=admin
      - DB_PASSWORD=admin
    depends_on:
      - postgres

  backend-reservations:
    build: ./backend-reservations
    ports:
      - "3003:3003"
    environment:
      - NODE_ENV=production
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_NAME=mydb
      - DB_USER=admin
      - DB_PASSWORD=admin
    depends_on:
      - postgres

  postgres:
    image: postgres:15
    environment:
      - POSTGRES_DB=mydb
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=admin
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Listing 10: Configuración Docker Compose

4.4. Patrones de Diseño Implementados

1. **API Gateway Pattern**: Centralización del punto de entrada
2. **Shared Database**: Base de datos PostgreSQL compartida entre servicios
3. **Health Check**: Monitoreo de estado de servicios
4. **Service Communication**: Comunicación HTTP/REST entre servicios

4.5. Estrategia de Comunicación

- **Comunicación Síncrona**: HTTP/REST para operaciones en tiempo real
- **Formato de Datos**: JSON para todas las comunicaciones
- **Validación**: Esquemas Joi para validación de datos de entrada
- **Manejo de Errores**: Códigos HTTP estándar y mensajes descriptivos

5. Pruebas – Ejecución

Esta sección será completada con los resultados de las pruebas realizadas.

6. Conclusiones

El desarrollo del Sistema de Reservas de Laboratorio con arquitectura de microservicios ha demostrado la viabilidad de implementar sistemas distribuidos utilizando tecnologías modernas como Docker y Kubernetes. La separación de responsabilidades en diferentes servicios permite una mejor organización del código y facilita el mantenimiento futuro del sistema.

La implementación de un API Gateway centralizado simplifica la gestión de las peticiones y proporciona un punto único de entrada para todas las operaciones del sistema. La utilización de PostgreSQL como base de datos compartida garantiza la consistencia de los datos entre los diferentes microservicios, aunque presenta desafíos en términos de acoplamiento que podrían ser abordados en futuras iteraciones del proyecto.

La experiencia obtenida en el desarrollo de este sistema proporciona una base sólida para la implementación de arquitecturas de microservicios más complejas y distribuidas en el futuro.

7. Bibliografía

No hay bibliografías.