Universite Hassan II de Casablanca
ENSET Mohammedia
Master's Program in Software Engineering (SDIA)

Design and Implementation of a Conference Management Microservices System

Distributed Systems Laboratory

Prepared by:
Otmane TOUHAMI

Academic Year:
2024 – 2025

Academic Year 2024 – 2025

Abstract

This report details the design and implementation of a scalable, microservices-based Conference Management System. The project aims to provide a comprehensive platform for managing academic conferences, keynote speakers, and participant reviews.

The architecture leverages the Spring Cloud ecosystem to solve distributed system challenges such as service discovery, configuration management, and intelligent routing. The backend comprises specialized microservices: a **Conference Service** for event management and reviews, and a **Keynote Service** for speaker profiles. These services communicate securely using OpenFeign and are protected by OAuth2/OIDC authentication via Keycloak.

A modern frontend application, built with Angular 19 and a reactive signals-based architecture, provides an intuitive user interface. The system ensures high availability and resilience through patterns like Circuit Breakers (Resilience4j) and is containerized using Docker for reproducible deployment.

Keywords: Microservices, Spring Boot, Spring Cloud, Angular, Keycloak, Docker, Resilience4j, Feign Client.

Contents

1 Introduction

In the domain of academic event management, systems must handle complex data relationships and varying loads while maintaining data consistency and user accessibility. Monolithic architectures often struggle to scale specific components independently or adopt new technologies for isolated features.

This laboratory project addresses these challenges by implementing a decoupled microservices architecture. The system facilitates the creation and management of conferences, the assignment of keynote speakers, and the submission of attendee reviews.

The primary objectives are:

1. To implement a robust microservices infrastructure using Spring Cloud (Gateway, Eureka, Config).
2. To secure the application using industry-standard OAuth2/OIDC protocols with Keycloak.
3. To demonstrate effective inter-service communication and fault tolerance.
4. To develop a responsive, user-friendly frontend using Angular.

2 System Architecture

The system is designed as a set of loosely coupled services that collaborate to fulfill business requirements.

2.1 Architecture Overview

The architecture diagram below illustrates the interactions between the client (Angular), the infrastructure services, and the core business microservices.

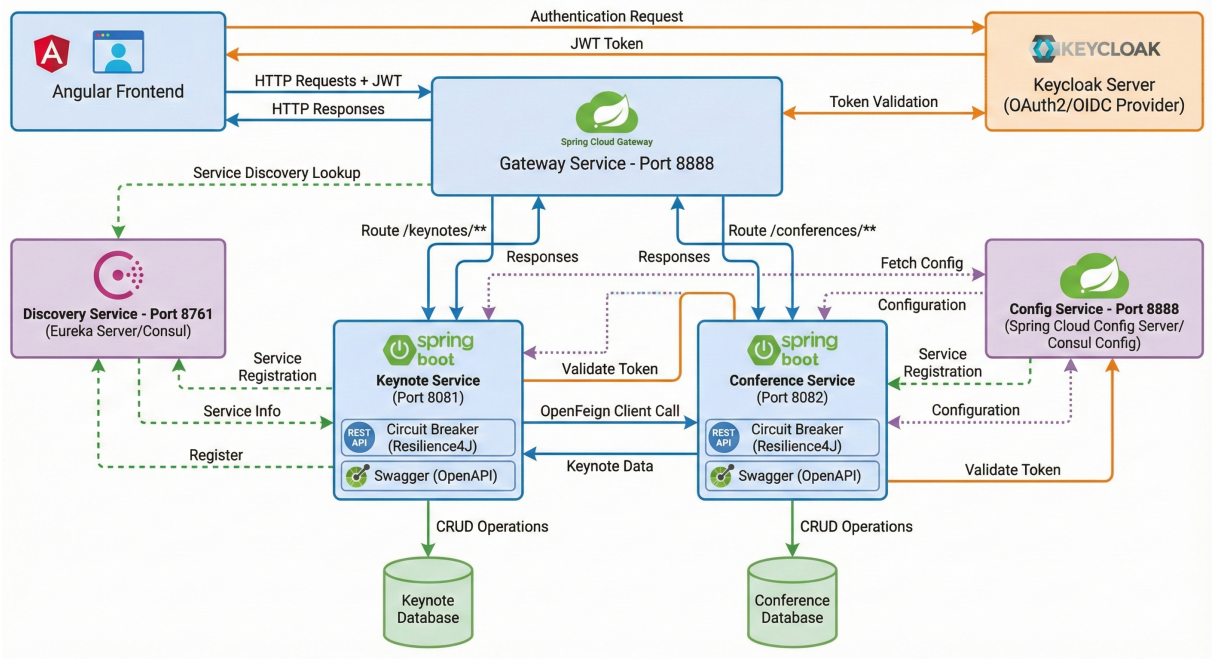

Figure 1: Global Architecture of the Conference Management System

Data flows from the client through the **API Gateway**, which handles routing and initial security checks. The gateway discovers service locations via the **Discovery Service** (Eureka). All services retrieve their configuration from the **Config Service**. The **Keycloak** server acts as the Identity Provider (IdP), issuing JWT tokens for authentication.

2.2 Technology Stack

- **Backend Language**: Java 21
- **Framework**: Spring Boot 3.4.1, Spring Cloud 2024.0.0
- **Database**: H2 (In-memory for development/testing)
- **Security**: Spring Security 6, OAuth2 Resource Server, Keycloak
- **Frontend**: Angular 19 (TypeScript, Signals)
- **Containerization**: Docker, Docker Compose

3 Infrastructure Services

The infrastructure layer provides the operational foundation for the microservices.

3.1 Service Discovery (Netflix Eureka)

The Discovery Service allows microservices to register themselves and discover other services dynamically. This eliminates the need for hardcoded hostnames and ports.

3.2 Configuration Management (Spring Cloud Config)

The Config Service centralizes application configuration. It serves properties from a git repository (or local filesystem), allowing configuration updates without recompiling services.

3.3 API Gateway (Spring Cloud Gateway)

The Gateway Service acts as the single entry point. It routes requests to the appropriate microservice based on the URL path (e.g., `/api/v1/conferences` → `conference-service`) and handles Cross-Origin Resource Sharing (CORS).

3.4 Security & Identity (Keycloak)

Keycloak manages users, roles (ADMIN, USER), and authentication. The backend services act as Resource Servers, validating the JWT tokens attached to incoming requests.

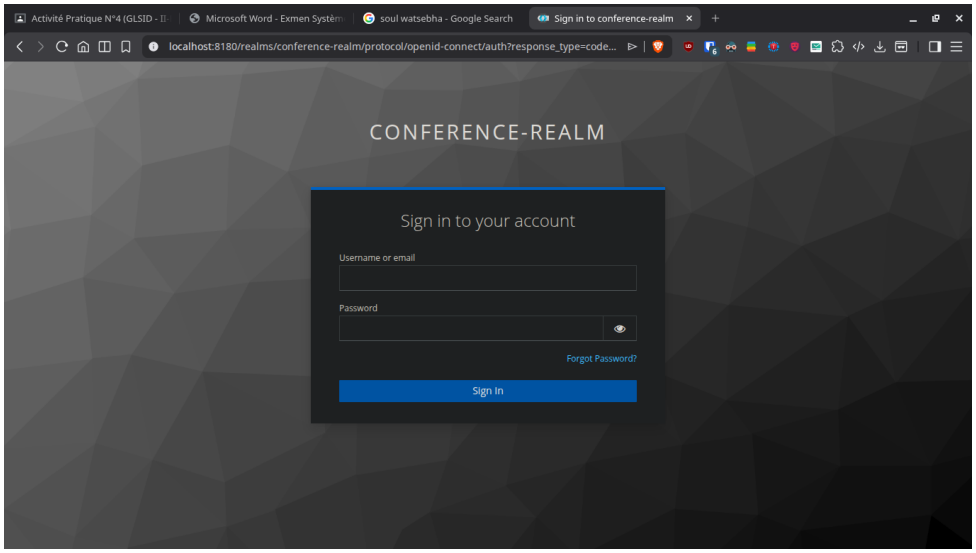

Figure 2: Keycloak Login Interface

4 Business Services Implementation

The core logic is divided into two main services.

4.1 Conference Service

This service manages conference details and reviews. It owns the `Conference` and `Review` entities.

4.1.1 Entity Model

The `Conference` entity includes a reference to a Keynote speaker (`keynoteId`) stored as a UUID, adhering to the database-per-service pattern where foreign keys to other services' tables are avoided.

```java
@Entity
@Getter @Setter @AllArgsConstructor @NoArgsConstructor @Builder
public class Conference {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(nullable = false)
    private String title;

    @Enumerated(EnumType.STRING)
    private ConferenceType type;

    @OneToMany(mappedBy = "conference", cascade = CascadeType.ALL)
    private List<Review> reviews = new ArrayList<>();

    @Column(name = "keynote_id")
    private UUID keynoteId; // Reference to remote Keynote entity
}
```

Listing 1: Conference Entity

4.2 Keynote Service

This service manages the profiles of keynote speakers (name, email, function). It exposes a REST API for CRUD operations on speakers.

4.3 Inter-Service Communication

The Conference Service needs to display speaker details. It uses **Spring Cloud OpenFeign** to call the Keynote Service. To handle network failures, **Resilience4j** is used as a circuit breaker.

```java
@FeignClient(name = "keynote-service")
public interface KeynoteClient {
    Logger log = LoggerFactory.getLogger(KeynoteClient.class);

    @GetMapping("/api/v1/keynotes/{id}")
    @CircuitBreaker(name = "keynoteService", fallbackMethod = "getKeynoteByIdFallback")
    KeynoteResponseDto getKeynoteById(@PathVariable UUID id);

    default KeynoteResponseDto getKeynoteByIdFallback(UUID id, Exception e) {
        log.error("Fallback triggered for id " + id, e);
        return KeynoteResponseDto.builder()
                .firstName("Unknown")
                .lastName("Speaker")
```

```
14                      .build();
15      }
16  }
```

Listing 2: Keynote Feign Client with Circuit Breaker

4.3.1 Security Context Propagation

Since Feign clients run in a protected environment, the JWT token must be forwarded to the downstream service. We implemented a `RequestInterceptor` to propagate the `Authorization` header. Additionally, `SecurityContextHolder` strategy was set to `MODE_INHERITABLETHREADLOCAL` to ensure the context is available in Resilience4j's thread pool.

5 Frontend Application

The Angular frontend provides a seamless user experience, interacting with the backend services via the API Gateway.

5.1 User Interface Design

The application features a modern, responsive design.

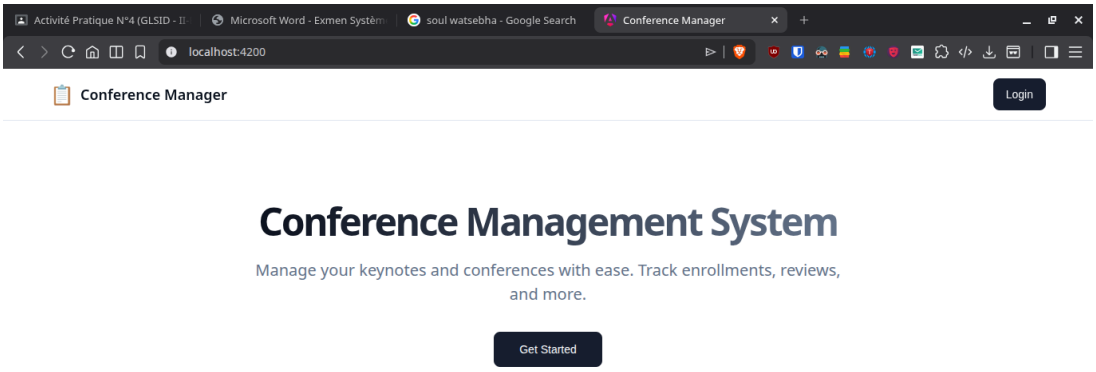

Figure 3: Application Landing Page

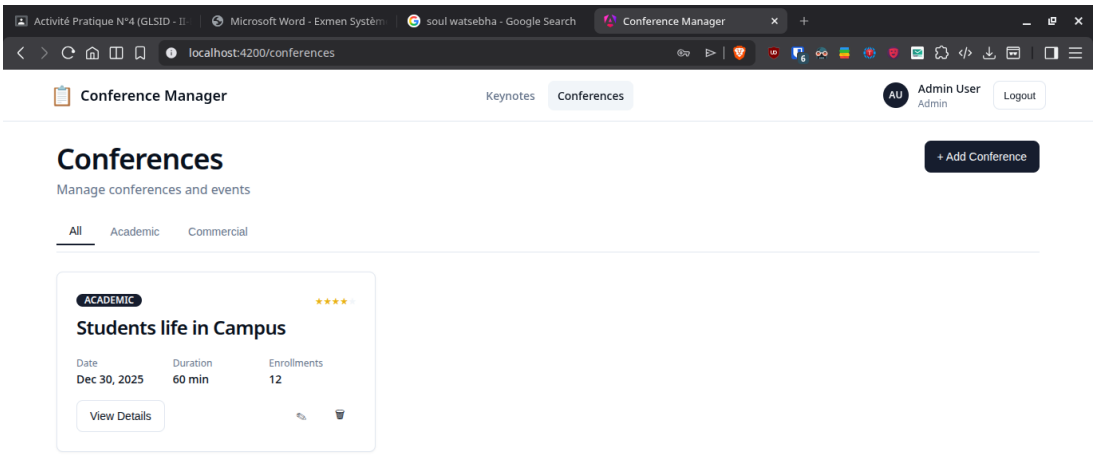

Figure 4: Conference Listing Page

5.2 Detailed Views

The Conference Detail page aggregates information. It displays the conference metadata, the fetched Keynote Speaker profile, and a grid of participant reviews.

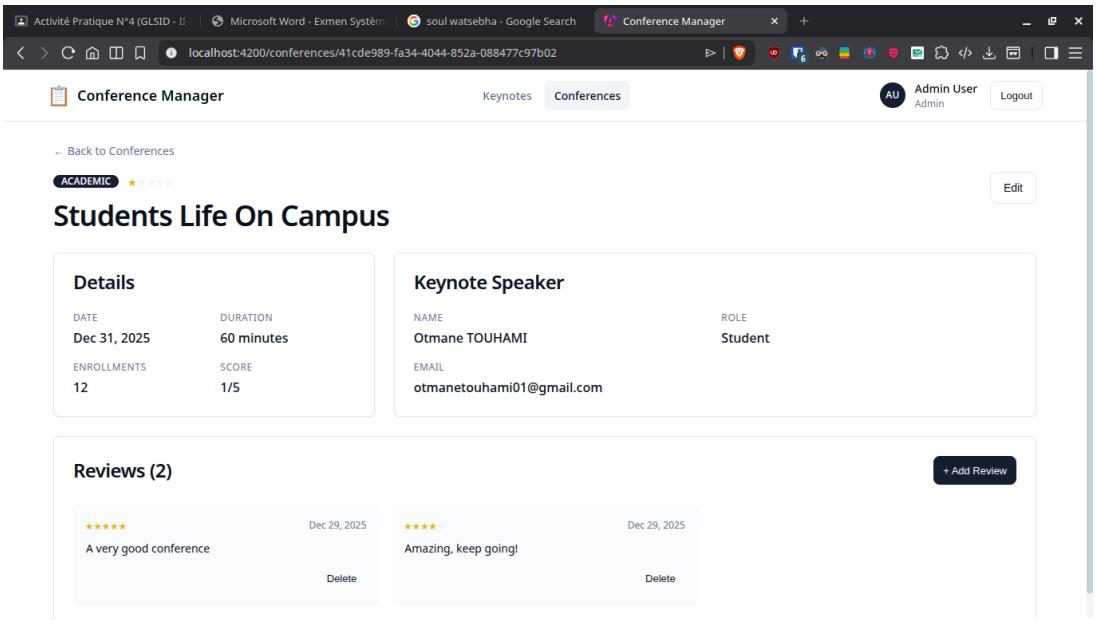

Figure 5: Conference Details with Keynote and Reviews Grid

The "Reviews" section demonstrates the 3-column grid layout implemented using CSS Grid, which responsive adapts to mobile and tablet screens.

6 Deployment

The entire system is containerized using Docker. A `docker-compose.yml` file orchestrates the deployment of all microservices, databases, Keycloak, and the frontend (served via Nginx).

```
services:
  gateway-service:
    build: ./backend/gateway-service
    ports:
      - "8888:8888"
    depends_on:
      - discovery-service
      - keycloak

  frontend:
    build: ./frontend
    ports:
      - "4200:80"
    depends_on:
      - gateway-service

  conference-service:
    build: ./backend/conference-service
    environment:
      - SPRING_CONFIG_IMPORT=configserver:http://config-service
    :9999/
```

Listing 3: Docker Compose Extract

This setup allows for a single-command deployment using `docker-compose up -d -build`, significantly simplifying the development and testing workflow.

7 Conclusion

The Conference Management System successfully demonstrates the principles of microservices architecture. By decomposing the application into focused services, we improved modularity and maintainability. Implementing OpenFeign and Resilience4j ensured robust inter-service communication, while Keycloak provided a secure and standardized authentication mechanism.

The project highlights the complexity of distributed systems—specifically regarding data consistency, security context propagation, and configuration management—and presents effective solutions using the Spring Cloud ecosystem. The modern Angular frontend complements this robust backend, delivering a professional and responsive user experience.