SERVICE ARCHITECTURE AND SOFTWARE ENGINEERING

- **Cours magistraux (CM) :** 6
- **Travaux Dirigés (TD) : 10**
- **Travaux pratiques (TP) :** 4

Teacher: Nawal Guermouche

General information

The "Service Architecture" course allowed me to explore and put into practice key concepts for the design of distributed systems, particularly those based on REST architecture and microservices. These concepts are essential for developing modular, scalable and secure applications, capable of managing complex data while ensuring their integrity and interoperability. Through concrete projects, I was able to apply these concepts to real cases and understand their challenges in modern application development.

Part A: Background and Objectives

The main objective of the project was to develop an application to manage volunteers in an associative environment. This application had to take into account several criteria, such as the management of volunteers' contact details, their availability and the centralization of data. The REST architecture, coupled with microservices, was chosen to meet the modularity, security and scalability needs of the project. In addition, the management of data security, especially that of volunteers, was a priority. This involved the implementation of authentication and authorization mechanisms via REST APIs. Another important consideration was the scalability of the system. Thanks to the microservices architecture, each component of the system could evolve independently to manage a growing number of users without disrupting the entire application.

Part B: Requirements and Architecture Analysis

Analysis of functional and technical needs
The project made it possible to clearly distinguish between functional needs, such as user and request management, and technical needs, such as data security and system load management. The REST architecture helps meet these needs by providing clear and standardized communication between services.

Main components of the system
The application was split into several microservices, each with a specific responsibility:

- **VolunteerService** : Volunteer data management (creation, update, deletion).
- **UserService** : User authentication and role management.
- **RequestService** : Volunteer Request Management.

Each of these services has been designed to be independent, facilitating maintenance and updates without disrupting other services.

Part C: Technical Implementation

Technologies used
Spring Boot was used to implement REST APIs, which is a common solution for building robust microservices. This framework makes API management easier, providing tools to ensure their security and performance. In parallel, relational databases were used to structure and store user, role, and request data efficiently.

Communication between microservices
Communication between the different microservices was ensured via RestTemplate, a tool that allows HTTP calls to be made between services, respecting the stateless principle of the REST architecture. This approach ensures that each service remains independent and does not need to know the state of the other services.

Part D: Analysis and reflections on architecture

Advantages of REST architecture and microservices
One of the main advantages I have seen through this project is modularity. Since each microservice is autonomous, it can be updated or replaced independently without affecting the entire system. This also allows for easier error handling, as each service can be fixed without disrupting the others.

Scalability is another strong point of microservices. Depending on the demand, it is possible to deploy additional services to handle a larger number of users without rewriting the entire application. This model also facilitates maintenance, since each service can be updated or improved independently.

Interoperability is also a key feature of REST architecture. It allows independent services to easily communicate with each other, but also to integrate other external systems, which is essential for the evolution of a project in the long term.

Possible improvements

Beyond the implementation of the architecture, this course also allowed me to understand the importance of predicting the evolution of the system. For example, adding automated tests could improve the quality of the code with each modification, and the integration of advanced monitoring tools would allow to better track the performance of the system and detect anomalies in real time.

Conclusion

In conclusion, this course allowed me to develop a deep understanding of service architecture, especially REST architectures and microservices. Through this project, I gained practical skills on creating, managing and securing a distributed architecture, while mastering the challenges related to scalability and maintenance. The concepts learned prepared me to meet today's challenges of software development, especially in a service-oriented environment.

<u>Matrice d'évaluation</u>

Service Oriented Architecture	**Expected**	**Estimated**
Know how to define a Service Oriented Architecture	4	4
Deploy an SOA with web services	4	3
Deploy and configure an SOA using SOAP	4	3
Deploy and configure an SOA using REST	4	3
Integrate a process manager in an SOA	4	3