Development of a starting point for a generic open-source event manager back-end

Nicolas Becerra Liñeira

Haaga-Helia University of Applied Sciences
Bachelor of Business Information Technology
Bachelor’s Thesis
2023

Abstract

Author(s)
Nicolas Becerra Liñeira

Degree
Bachelor of Business Administration

Report/Thesis Title
Development of a generic open-source event manager back-end

Number of pages and appendix pages
60 + 1

This thesis aims to develop a starting point for a generic open-source back-end for the creation of an event management application. The project is composed of two parts, the back-end, and the database. The back-end is part of the application that is hidden from the user. It coordinates the actions between the front-end and the database like a conductor. The database will be the place where all the data of the application will be saved.

The report will start by putting forward explanatory and theoretical elements on the subject of the backend. Following the explanation of these primary notions, the author will go on to explain the Spring Framework and the REST API architecture. To conclude this section, the author will present the MariaDB database management system specifically chosen for the development of the project in connection with this thesis. This part will also serve as a justification for the choice of tools and methods for the achievement of the thesis.

Through the empirical part, the author will explain the implementation of the various tools and methodologies used to develop the entire project. This section will focus on a full explanation of the development of the Event Manager project, including the highlights, but also the various obstacles encountered during development.

Finally, through the discussion, the author will return to all the concepts and stages of development that have been addressed throughout this thesis. This will allow him to share with the reader his feelings throughout the writing and programming of the thesis. This part will also allow the author to share his vision for the future of the back-end produced in this thesis.

Keywords
Spring Framework, REST API, Database, Back-end

Table of contents

1 Introduction 1
1.1 Objective 1
1.2 Scope and Limitations 3
1.3 Future 4
2 Theoretical Framework 5
2.1 Difference between the back-end and the front-end 5
2.1.1 Client/Server architecture 5
2.1.2 Framework 6
2.2 Spring Framework 7
2.2.1 History 7
2.2.2 Java EE 9
2.2.3 Spring and Java EE 9
2.2.4 Spring modules 10
2.2.5 Spring Project 12
2.2.6 Spring Boot 12
2.2.7 Alternatives 13
2.2.8 The choice of Spring 14
2.3 Rest API 15
2.3.1 Principles 15
2.3.2 REST and HTTP 16
2.3.3 HTTP Verbs 16
2.3.4 HATEOAS and Spring 17
2.3.5 REST API Security 17
2.4 Databases 18
2.4.1 Relational Database 19
2.4.2 SQL 20
2.4.3 MariaDB 20
2.4.4 MariaDB and MySQL 21
3 Empirical Part 23
3.1 Starting Point 23
3.2 Illustration of the outcome 24
3.3 Methodologies 25
3.4 Project development 26
3.5 Presenting the outcome 40
3.5.1 Creation of a user 41

3.5.2 Create an event ... 43
3.5.3 Participant management ... 51
4 Discussion ... 54
Sources ... 57
Appendices ... 61
Appendix 1. GitHub repository ... 61

1 Introduction

Management is an indispensable part of our way of life. It allows us to control our resources, limit errors and can also be used as a predictive tool. During my exchange in Helsinki, Finland, I had the opportunity to use the **Kide.app** application: it allows students from all over the country to take part in events and sign up to the various associations on the platform. After discovering this application, I wondered why there wasn't a similar one in Switzerland, if only for university campuses.

So I started to find out about the different events available on the campus of my home university and what means of communication were being used. I realised that there were gaps in the organisation and communication of events. The information for some of them was not up to date or events were taking place at the same time and in the same place.

Given these problems, I came up with the idea of creating a generic open-source application for creating, managing and registering for events. As well as aiming for better management, the production of such an open-source application would allow the entity wishing to use it to adapt it to its needs. This would enable university campuses to offer their student community a platform enabling them to keep up to date and take part in the latest events, but it would also give organisers a better overview of activities, which would help them plan future events.

1.1 Objective

The aim of this thesis is to develop a first version of a generic open-source back-end for an event management application. The fact of developing a project under an open-source license imposes constraints on the use of various tools for the development of the project. Only open-source tools can be used.

In order to respect this constraint, the project uses a MariaDB database to store all the data. The relationship between the database and the back-end is managed by the Spring Framework. As well as containing the implementation of the business processes, the use of Spring simplifies the creation of the database and the relationships between entities.

All the business processes developed through the code will be exposed in a secure manner via the implementation of a REST API[1]. This enables communication between the system and another, for example between a back-end and a front-end.

[1] Application Programming Interface Representational State Transfer

The back-end developed through this thesis will aim to provide basic functionality, in order to be able to present the concept to potential stakeholders. The functionality can be divided into three different groups. The first group provides functionality related to event management, allowing organisers to perform the following functions:

- **Create, update and delete an event an event.**
- **Create, update, delete and add an address to an event.**
- **Create and add tags to describe the event.**
- **Remove participants from an event.**
- **Add participants to an event from a waiting list.**

The second groups together the functions intended for interaction between the application and the user:

- **Note and comment on an event.**
- **View available events by system date.**
- **Register for an event on the basis of available places, or else join the event waiting list.**
- **Search for an event by tag, location or name.**

The last group concerns the administration and security of the application. The user registration and authentication functions are absolutely essential, to ensure that the application's security is properly managed. They make it possible to manage access to different methods according to the user's role.

All the functions listed above are based on the vital information required to create and manage an event. To this end, as I mentioned in the section above, I have drawn on my experience as a user of the **kide.app** application to define the functionalities proposed through the back-end produced in this thesis.

Finally, the back-end application must provide a versatile and scalable infrastructure that will provide the necessary tools for the efficient organisation and management of events of all types (festive, sporting, professional, etc.). The creation of such a back-end under an open-source licence aims to develop a turnkey solution that can be adapted to the specific needs of each entity wishing to use or enhance it. The last group concerns the administration and security of the application. User registration and authentication functionalities are absolutely essential to ensure that the application's security is properly managed. They make it possible to manage access to different methods according to the user's role.

All the functions listed above are based on the vital information required to create and manage an event. To this end, as I mentioned in the section above, I have drawn on my experience as a user of the **kide.app** application to define the functionalities offered through the back-end produced in this thesis.

Finally, the back-end application must provide a versatile and scalable infrastructure that will provide the necessary tools for the efficient organisation and management of events of all types (festive, sporting, professional, etc.). The aim of creating such a back-end under an open-source licence is to develop a turnkey solution that can be adapted to the specific needs of each entity wishing to use or enhance it.

1.2 Scope and Limitations

The scope of my Bachelor's thesis focuses on the development of a first version of a generic, open-source back-end for an event management application. The aim of this first version is to provide future stakeholders with a first glimpse of a turnkey solution exhibiting a series of functionalities that I described in **section 1.1, Objectives**. In the scope of this thesis, the term 'back-end' refers to the management of the database, which includes the creation of tables and the manipulation of records within the database. The implementation of all the business processes, as well as their exposure through the implementation of a REST API, are also included.

The completion of this thesis will also allow me, on a personal level, to develop new knowledge in the field of web application development, but also to deepen and demonstrate the knowledge and skills acquired throughout my studies in the field of Java programming, REST API development and server-side programming.

Firstly, I want to take this opportunity to get a fresh look at the elements that define a REST API, as well as its implementation with the Spring framework. Secondly, developing an application that will inspire use by a wider audience than myself involves implementing security. This will allow me to familiarise myself with the different methods and challenges involved in implementing security with Spring and a REST API. Thirdly, I also want to deepen my skills in building applications with the Spring Framework.

Finally, this thesis focuses solely on back-end development. Although, the provision of a REST API is not meant to directly interact with users. The development of a front-end is not included in the scope of this thesis. Indeed, the development of a front-end is a work overload, given the time available for writing the thesis and producing the back-end. This is also due to the fact that I'm developing this project completely independently, without the help of any external party. What's more, I'm keeping the development of a front-end for a side project that will mark the continuity of the project once the thesis has been re-edited.

1.3 Future

As indicated in the introductory part of the introduction, my final objective is to be able to develop a generic open-source application that will enable event management. However, following the submission of my thesis, I'd first like to implement other functionalities to make it more complete. Finally, I want to develop a front-end so that I can meet my final objective, which I mentioned in the introduction of this chapter.

2 Theoretical Framework

2.1 Difference between the back-end and the front-end

The world of web application development is generally distinguished by two elements: a front-end and a back-end. These terms are used to describe the make-up of applications and websites, which are represented at two different levels. As they do not perform the same tasks, it is imperative that these two parts works together, otherwise the fluidity and functioning of the application will be compromised without this essential union of front-end and back-end. (Schaffer, 8 October 2021)

Front-end

The front-end focuses on the user experience and interface of a web application. In other words, the front-end represents all the content intended for the end user. This includes images, text, page design, layout and colour of the elements that make up the application. The main objective of the front-end is to work with the various resources available on the back-end to enable interaction between the user and the program. (Schaffer, 8 October 2021)

Back-end

The back-end concerns all the elements that take place in the background. It groups together all the processes that website and application users can't see, but which enable the application to function properly. It manages the storage, delivery and organization of web project data. Depending on the interactions carried out by the user via the front-end, it returns the data required by these interactions.

The IONOS website explains in its article "back-end and front-end explained simply", that the back-end can be seen as the heart of a web project. I only partially agree with this comparison. It's true that the back-end is indispensable to a web project, but I'd compare it more to a brain: like a brain, the back-end orchestrates the various elements to enable the project to function. (IONOS, 31 Mars 2022)

2.1.1 Client/Server architecture

Democratised at the end of the 80s, client/server architecture is an IT model based on the development of centralised systems. This model distributes functions over the network between the client and the server. The client represents all the processes that make specific requests to the server to access different resources, such as data or functions. The server, on the other hand, represents a process responsible for providing a specific response to the client's request.

In the context of a website, the exchange of resources between the front-end and the back-end is based on this model. The client corresponds to the front-end and the server to the back-end. The exchange of requests is triggered by the user's interaction with the front-end.

In this model, the server is not systematically linked to a single client and can process requests from several clients. This is made possible by providing a permanent, passive service. This structure enables centralised administration of the most important and sensitive resources, such as databases. This will simplify their administration, but also their maintenance, thereby limiting risks. However, this type of structure is not a good solution in the event of a server failure. Indeed, such an incident would lead to a global breakdown of the system, leaving customers without answers and inoperable.

In response to this risk, the client/server architecture allows the client to call on several servers to benefit from other functionalities. In the event of a server failure, they can communicate with a back-up server.

Finally, this IT model is flexible enough to allow a computer to be both client and server, or just one of the two. Its role is determined by whether it sends or receives requests. (IONOS, 31 January 2023)

2.1.2 Framework

In the world of software and application development (web or mobile), a framework is a set of tools and libraries designed to help developers. They are used throughout the application development cycle. This part will serve me to provide a generic, but introductory explanation on the framework-work in the preamble of the chapter on Spring Framework.

Frameworks provide developers with dependencies and functionalities designed to facilitate the management of low-level technical aspects, thus helping to reduce development time. Indeed, these technical aspects, specific to a programming language or library, can slow down project development due to their programming and implementation. Among these technical aspects, a framework can manage and even automate certain tasks, such as database management and connection, data manipulation, security and servlet management.

What's more, the use of frameworks makes it possible to standardize application development. As I explained in the paragraph above, frameworks have pre-established functionalities that enable the management of technical tasks. In this way, code is standardized, because developers no longer need to code methods or functions, but use or reuse what already exists. In my opinion, the fact of being able to standardize certain aspects of an application's programming is a good thing, because, as I mentioned earlier, it speeds up development time, and therefore reduces the associated costs.

Standardization also facilitates the transmission and learning of programming. Indeed, the use of frameworks reduces the need for the developer to add his own personal touches to the code, making it easier for a wider audience to understand, and for it to evolve over time. Having clean, understandable code is all the more important these days, with the evolution of work methods promoting sharing and communication rather than cluster work.

From a personal point of view, learning to code using frameworks enables you to adopt a more simplified approach to code, thanks to the premade functionalities offered by the framework. Where learning traditional code is more akin to solving a puzzle, learning with a framework could be likened to assembling a Lego, as it offers a context focused on both practice and theory. However, the future developer needs to understand the basics of the programming language related to the framework, because without them he can neither understand nor solve problems. Even if the framework offers a layer of abstraction for the management of certain processes such as database management, zero risk is not guaranteed. It's up to the developer to identify a potential error, understand it and fix it. (codecademy, 23 September 2021)

2.2 Spring Framework

The framework market is made up of several players such as Django, Express and Rails, to name but a few. However, for the purposes of this thesis, I'm going to focus on just one: the Spring Framework. This is an open-source framework with its roots in the Java programming language. Born of the desire to facilitate the development of enterprise Java applications, it is now used in all kinds of projects, including microservices, web applications, cloud and servless applications.
(Spring, 13 June 2019)

2.2.1 History

The Spring Framework was created in 2003 by Rod Johnson, as he explained in his 2013 GOTO interview. The framework came onto the market as an inversion of control (IoC) container for Java. Thanks to its simplicity and lightness, it very quickly became a solution to the complexity of developing Java applications with the Java Enterprise Edition (JEE) tool. In fact, Spring integrates selected specifications from the EE umbrella, to make it easier to develop applications with JEE. These specifications include:

- **WebSocket API:**

  This API enables bidirectional, duplex, real-time communication between the server and the web browser. Thanks to this API, the server can send data to clients at any time. (Baeldung 26 Mai 2023)

- **Concurrency Utilities:**

  It is a programming model that allows developers to perform or schedule tasks simultaneously, set up threads and transfer the Java EE context in order to invoke interfaces. (IBM 13 December 2022)

- **JSON Binding API:**

  This API converts Java objects to or from the JSON data format. (Baeldung 24 June 2022)

- **Bean Validation:**

  JavaBeans Validation or Bean Validation is an independent validation specification that intervenes in the application layer. It provides the developer with batches of validation constraints on a domain model, which are then validated across application levels (OWASP s.a.). Taking the form of annotations, they can be linked to a class, a method or a field of a JavaBeans component. (Oracle 2013)

- **JPA:**

  JPA stands for Java Persistence API. This API manages the persistence and mapping of relational objects and functions. Its aim is to simplify the persistence programming model. JPA allows the use of annotations or XML to facilitate the mapping of relational objects. The aim is to standardize the mapping of objects within different databases. (IBM, 2 May 2023)

- **JMS:**

  JMS stands for Java Message Service. This API enables the components of a Java EE application to create, send, receive and read messages asynchronously between applications or components. (Oracle, s.a.)

Today, Spring is one of the most popular frameworks on the market. Supported by an active community, this framework has been able to continue evolving and perfecting itself to offer new features such as security, MVC (model-view-controller), web services, data access and the cloud. (Spring, 13 June 2019)

2.2.2 Java EE

Currently known as Jakarta EE following the transfer of rights from Oracle to the Eclipse Foundation in 2017, Java EE is the set of enterprise specifications revolving around Java SE. Java's enterprise extensions appeared at the same time as the first version of Java, and at that time were an integral part of the JDK kernel. It wasn't until 1999 that they were removed from the standard binary as part of the release of the second version of Java (Java 2). As a result, these extensions became known as J2EE (Java 2 Platform Enterprise Edition).

Through its platform, JEE provides an Api grouping together functionalities such as distributed computing and Web services. Applications developed using Java EE run in reference runtimes or runtime environments, such as microservers or application servers. Thanks to its features and environment, Java EE enables large-scale development and execution of networked, scalable, secure and multi-tiered applications. (javatpoint, s.a.)

2.2.3 Spring and Java EE

Although presented as two competitors, Spring Framework and Java EE are not really in competition: as I explained in the introduction to this section, Spring was created to fill Java EE's gaps in web application development, simplifying and reducing the development costs of Java Enterprise applications. In its documentation, VMware emphasizes that Spring is not a competitor but a complement to Java EE (Spring 13 June 2019). Indeed, the framework uses Java EE functionalities to create Java EE applications, which only supports the concept of complementarity rather than competition. (Juvénal, s.a.)

2.2.4 Spring modules

Spring Framework contains a large number of features offering developers infrastructure support for project development, as well as a guide to application creation. The functionalities are grouped into some twenty groups called modules (adservio, 2011). To make it easier to understand and find information on these different modules, the framework's documentation groups said modules ac-

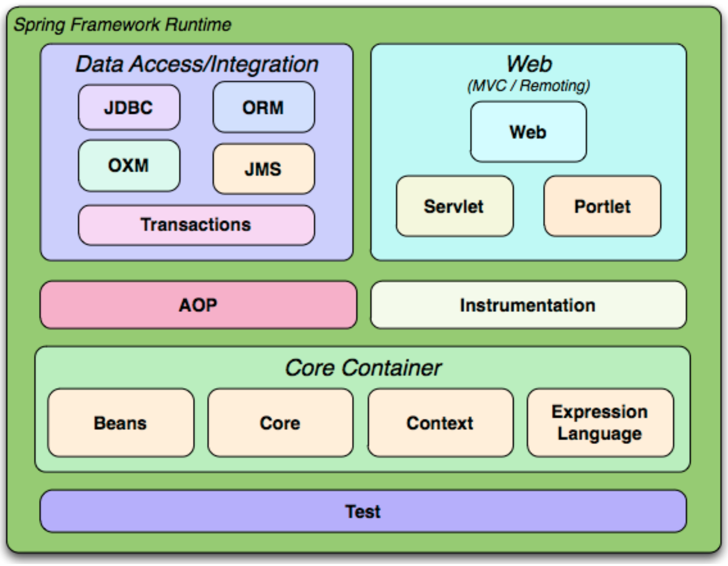


Figure 1. Overview of the Srping Framework (Spring, s.a.)

cording to their primary functionalities in layers, as shown in the image below. (Spring, s.a.). To create this part, I used the official documentation for Spring Framework version 3.0.0.M4 de Spring Framework.

Core Container

The Core Container layer is made up of 4 modules that symbolize the basis of all Spring Framework projects. The Core module provides two functionalities that represent the very nature of Spring: Inversion of Controller (IoC) and Dependency Injection (DI) (Spring, s.a.). The Beans module enables simple, flexible configuration of objects via the IoC container, using the BeanFactory interface. (Baeldung, 2022).

The Context module functions as a JNDI (Java Naming and Directory Interface) registry, enabling access to objects from code and beans.

The Expression Language module is an essential part of the communication between the user and the application. This module enables data from an object graph to be queried and manipulated using the Spring Expression Languge (SpEL). (Spring, s.a.)

Data Access/Integration

The Data Access or Integration layer handles incoming and outgoing data from a Spring application. Data transactions are handled by the five modules making up this layer.

The JDBC (Java database connectivity) module does away with traditional JDBC coding, providing an abstraction layer that manages the database connection, handles SQL exceptions, fetches results and closes the connection. (adservio, 2022)

The ORM (object relation mapping) Module focuses on managing interactions with various object-relation mapping APIs, such as Hibernate and JPA. It also supports multiple ORM APIs, ensuring interaction with multiple data sources. (adservio, 2022)

The Java Message Server (JMS) module handles both incoming and outgoing Java messages. Using this module eliminates the need to use the JMS API. (adservio, 2022)

The final module in the Data Access/Integration layer is Transactions. It manages and propagates transactions within the project. (adservio, 2022)

Web

Spring Framework's Web layer enables the development of highly flexible web applications. The eponymous module provides basic integration functionalities such as multipart file upload, IoC container initialization and a Web-oriented application context. However, the WebServlet module represents the culmination of the Web layer, providing the Model-View-Controller (MVC) implementation. MVC enables a division between code and Web forms, resulting in clearer, more modular code. (Spring, s.a.)

AOP and Instrumentation

AOP is a Spring Framework module that provides an aspect-oriented implementation, allowing cross-cutting concerns such as security, transaction management and caching to be separated. It offers features such as the definition of interceptors, the creation of shortcuts to increase code tenfold by using features that are not necessarily interconnected, and the definition of method interceptors. (Spring, s.a.)

The Instrumentation module is used for class instrumentation and, depending on the application server, is responsible for implementing the class loader. (adservio, 2022)

Test

The Test module includes the various functions needed to create Test Units, using JUnit and TestNG. This module enables code to be isolated and tested before being deployed. (Spring, s.a.)

2.2.5 Spring Project

Spring Project represents the grouping together of several open-source projects developed by the Spring community. Each project is named and created to meet a specific application need. They bring together all existing functionalities related to their fields, as well as additional functionalities and ready-to-use solutions (vmware Open Source Blog, 2020). Among the best-known projects is **Spring Boot**, developed to facilitate development with the Spring Framework by providing default configurations and simplifying development. **Spring Data** provides simplified data access and seamless integration with different database systems. **Spring Security** provides security-focused tools such as authentication, authorization and session management. **Spring Batch** offers efficient processing of large volumes of data for batch applications. Finally, **Spring Cloud** is a project designed to facilitate the creation of applications based on the Microservice concept and the integration of distributed services in a cloud environment. (Spring, 2023)

2.2.6 Spring Boot

Spring Boot is one of the many projects developed by the Spring Framework community. As I introduced in the previous section, Spring Boot is a tool designed to speed up and facilitate the development of Web applications and microservices using the Spring Framework.

Although it's a complete and powerful framework, configuring, installing and deploying Spring Framework can be a daunting task for people who don't have all the knowledge and experience of working with this tool. However, it is on these points that Spring Boot succeeds in setting itself apart with the following three pillars:

The first pillar is Spring Boot's built-in functionality for automatic configuration of Spring Framework and third-party packages according to selected parameters. This makes it possible to start developing quickly, and at the same time helps to reduce human error during project configuration.

The second is the directive approach used by Spring Boot, which relieves the developer of the burden of adding and configuring the dependencies required for the project. Spring Boot itself selects the packages and functionalities to be used, according to the choice of startup dependencies made by the user during the initialization process. Taking IBM as an example, by opting for the Spring Web startup dependency, Spring Boot provides the minimum configuration for creating a Web application by adding the Apache Tomcat server and Spring Security, as well as other dependencies.

The final level is application autonomy. With Spring Boot, developers can develop applications that run directly without the help of an external server. This is made possible by integrating a Web server such as Tomcat or Netty within the project. As a result, the application can be launched at any time and on any platform.

Finally, Spring Boot reduces the complexity and time required to configure and deploy a Spring project, making it easier to learn how to work with the Spring Framework. The advantages offered by Spring Boot are, however, made possible at the expense of the flexibility that users have when working directly with Spring Framework. (IBM, s.a.)

2.2.7 Alternatives

As mentioned in the introduction to this chapter on the Spring Framework, the framework market offers a wide range of alternatives to Spring. These alternatives stand out by offering different approaches to development, but also by using programming languages other than Java. Since the whole of this chapter is devoted to Spring, which is a Java-based framework, we can de-facto divide all the alternatives into 2 different categories: Java-based alternatives and alternatives based on a language other than Java.

Java based alternatives include:

Quarkus:

It is a native Java stack developed by Kubernetes. It was developed with a focus on cloud-native applications and microservices. Compared with Spring, it offers rapid start-up and low memory consumption. (Fol, 26 October 2021)

Micronaut:

It is a framework that provides an approach aimed at simplifying the development of cloud-native and microservice applications. It is characterised by the fact that it is a cloud-native framework, unlike Spring, which works with an embedded server. It is also different in that it offers a native service discovery module and native support for Kubernetes. (Fol, 26 October 2021)

Finally, among the alternatives based on other languages, we find:

Django:

It is a framework based on the Python programming language, which is the main difference between it and Spring. Like Spring, it is open-source and follows the same MVC (Model-View-Controller) architectural model, which is one of the architectural patterns offered by Spring. Finally, Spring works with an embedded server, whereas Django provides a full stack solution. (Fincher and Desmond K, s.a.)

Laravel:

It's a framework based on the PHP programming language. It also has an MVC architectural model like Django. Finally, when it comes to databases, Spring offers greater flexibility than Laravel in terms of security, data management and choice of structure. By offering various choices in the themes outlined above. (Fincher and Desmond, s.a.)

2.2.8 The choice of Spring

My choice to use Spring Framework for the development of my bachelor's thesis was based on three points. The first is that Spring is a fast-growing framework. Through its community and the various companies that use it, Spring has proven over the years to be a framework that evolves to offer complete solutions and tools that suit a wide and varied audience. One example of Spring's rise to prominence is that, in 2019, Netflix is migrating its platform to Spring Boot to ensure stable platform development. (VMware Tanzu, 16 October 2019, min. 1:44 - 7:00)

The second point relates to the fact that Spring Framework has two tools, Spring Framework and Spring Boot. Spring Framework, as I discussed earlier, offers a huge range of features, although the freedom of configuration it provides can paradoxically increase the time and complexity of configuring a project for a novice developer. Like the Spring Framework, Spring Boot has an automatic configuration feature that provides a clearer vision and framework for inexperienced developers, allowing them to focus on business logic rather than understanding and choosing dependencies when configuring. What's more, Spring has a large and very active community, as well as a wealth of documentation, and this offers the feeling of being supported and helped throughout application development.

The last point is based on the fact that throughout my training at the "Haute Ecole de Gestion de Genève (Suisse)", learning the Java programming language was predominant. During the semester prior to my arrival at the "Haaga-Helia University of Applied Sciences", I took an introductory module on creating web services using the Jakarta EE platform. I then took part in a "server programming" module at the University of Haaga-Helia. This module gave me my first contact with the Spring Framework. The fact that Spring is a Java EE based framework, while being more intuitive, won me over straight away.

Finally, for me Spring Framework is a legitimate and fair choice, not only because I've already used it in the past, but also because it's a Framework that's going from strength to strength every year. What's more, its open-source nature enables its community to develop and modulate innovative, high-quality solutions in response to their different needs, making it a modular, scalable framework.

2.3 Rest API

In 2000, a year of doubt and turmoil, Dr. Roy Fielding first evoked the concept of the REST API in his doctoral thesis, bringing a revolution to the API (Application Programming Interface) creation model. Today, the REST (Representation State Transfer) architecture is the common method for connecting components and applications within microservice-based architectures. (IBM, s.a.)

2.3.1 Principles

A REST API, like any other API, enables one application to access another, based on the client/server architectural style I defined earlier. However, for an API to be called REST or RESTfull, it must respect 6 design principles.

The first principle is to have **a uniform interface**. A uniform interface means that all requests for the same resource must have the same identifier. To achieve this, the API must be able to guarantee the uniqueness of an identifier in relation to a URI identifier. Finally, the resources returned by the server must include all the information requested by the client, without being too voluminous. (IBM, s.a.)

The second principle **is client-server decoupling**. This principle is to emphasize the fact that the client and the server must be two separate and independent entities. One cannot and must not change the other. The server only returns the predefined data corresponding to the URI requested by the client. (IBM, s.a.)

The third principle is the principle of **statelessness**. When a client makes a request to the server, it must contain all the information necessary for the server to understand and process the request. The server does not keep any client state between the different requests, which makes the requests independent of each other (future and past). (IBM, s.a.)

The fourth principle is **cacheability**. This is an important mechanism for temporarily storing responses to customer queries. This way, the server is able to quickly send the response back to the client without having to make a new request. This helps to reduce latency and improves performance by avoiding repetition of operations. (IBM, s.a.)

The fifth principle is to provide **a layered system architecture**. This is about containing several layers between the server and the client, which means that one or more servers can be used together to build a response. The use of this architecture also improves security, as resources are divided between the different servers. The layers act as barriers, thus providing a new level of security. (IBM, s.a.)

The sixth principle is the **on-demand code**. This principle, although optional, aims to provide executable code in the client response instead of conventional responses, allowing clients to extend functionality. The code sent in these responses may consist of scripts or calls. (IBM, s.a.)

Finally, the above principles may be subject to slight modifications depending on the source. Some sources present them in 5 different principles. As far as I am concerned, I have based myself on the documentation that IBM offers in relation to the definition of a REST API. However, I find it a pity that the principle of **connectedness or HATEOAS (Hypermedia As The Engine Of Application State)** was not highlighted. This principle aims to provide additional links to other REST APIs in the response formulated by the server. I think this principle is important because it provides a better representation of resources in relation to the data sent by the server. (IONOS 12 July 2023)

2.3.2 REST and HTTP

Architectural style is often associated with the use of the HTTP protocol to create Web APIs. This is because HTTP is the most commonly used choice due to its availability, simplicity and widespread adoption for implementing RESTfull APIs. However, despite its predispositions to offer many REST qualities, it is not the only option.

Indeed, REST is not strictly dependent on HTTP and can be implemented with other communication protocols such as MQTT to name but one. Although HTTP is not the only communication protocol that can be used with REST, it remains the most widespread. Following this principle, I decided to develop my REST API using HTTP. (Pratt, 5 October 2022)

2.3.3 HTTP Verbs

The use of a communication protocol such as HTTP in the implementation of a REST API ensures interaction between the client and the server. These interactions make it possible to perform all the functions called CRUD (Create, Read, Update and Delete) within a database. To perform these different queries, you need to use http verbs. These explicit verbs determine the type of action the client wants to take when sending the request to the server.

The verb **GET** is used to retrieve the state of the resource targeted by the UR by reading the data without modifying it. The **POST** verb submits a new resource to the server, in order to create a new entry in the database. The verb **PUT** is used to change the state of a resource already present in the database and, in other words, it completely updates a data. The penultimate verb is **DELETE** and consists of deleting a data targeted by the server URL. (IBM, s.a.)

Finally, the last verb is **PATCH** and its usage is similar to **PUT** except that **PATCH** only partially modifies the resource. In this way, it only sends data that needs to be specifically updated. (Gupta, 11 December 2021)

2.3.4 HATEOAS and Spring

HATEOAS, as I introduced in the section on REST principles, is a key element in the development of REST APIs. Introduced in 2000 by Dr. Roy Fielding when he defined the REST architecture, this principle aims to self-document APIs. Self-documentation takes shape by forcing the client to navigate through the API only via hyperlinks. These links illustrate all sources related to the URI requested by the client (IONOS, 12 July 2023).

In the case of the event handler that my thesis focuses on, when the client asks the server for data about a particular user, the server generates a response with the user's basic information such as last name, first name, age, etc. But the server also integrates hyperlinks that establish the connection to the various events in which the requested user has participated or will participate.

In addition to providing the client with some freedom and a more dynamic interaction with the application, the HATEOAS principle will also offer greater flexibility and scalability of the API by facilitating the provision of new functionality while avoiding any changes on the client's side. (IONOS, 12 July 2023)

Finally, Spring Framework offers through its ecosystem the Spring HATEOAS project which provides a library of APIs allowing the creation of links and the assembly of representations of the HATEOAS architecture. By integrating this project with the Spring application, developers can benefit from an abstraction that simplifies the implementation of the HATEOAS architecture. (Spring, 11 May 2023)

2.3.5 REST API Security

Creating an API that meets all the constraints of the REST architecture is a good thing. However, as I have mentioned several times, an API allows data to be transferred between the client and the server and when data is included in the equation, it is necessary to put measures in place to secure it.

The first step to secure your REST API is to make sure that all endpoints use only HTTPS internet protocol. The use of this protocol protects the transmission of data by encrypting it and making it unalterable, which is essential if the reply contains confidential information such as passwords or bank details. In addition, HTTPS guarantees client authentication. (OWASP, s.a.)

The second measure is to implement access control at the level of each endpoint, in order to be able to prohibit access to certain endpoints and reduce the risk of saturation between the different services. Access management is made possible by the allocation of access tokens such as JWT tokens (JSON Web Tokens). (OWASP, s.a.)

The third measure is input control. The data sent by the customer must be checked before any use. Without control, the application is exposed to SQL injections that can lead to a data leak or threaten the integrity of the database. To do this, measures can be implemented such as controlling the format and type, imposing a limit on the size of the query, and limiting the number of concurrent queries that the client can make. (OWASP, s.a.)

The fourth measure concerns the handling of error messages. Sometimes error messages contain information and details about the reasons for the failure in addition to the error number. This information can then help an attacker to better understand the architecture of the API, so under no circumstances should technical details be provided to the client. Only generic error messages should be sent. (OWASP, s.a.)

The fifth measure concerns the management endpoints, which are important points of the application, because, as their name suggests, they allow the setting of the application as a whole. In view of their importance, their accessibility via the Internet should be avoided. Where their accessibility is essential, an authentication mechanism such as a dual authentication service should be put in place to ensure the identity of the user. A firewall with an endpoint access list can also be set up. However, the firewall alone cannot guarantee security and must be accompanied by a good authentication service in order to offer the best possible protection. (OWASP, s.a.)

Finally, it should be stressed that there is no such thing as zero risk. However, the implementation of all the above measures can ensure a high level of protection. Most of the security flaws are due to a lack of attention in the development and implementation of the protection package. To avoid these errors, frameworks such as Spring Security offer pre-built features and solutions that limit the impact of the developer on the code, thus mitigating the risk of human error. (OWASP, s.a.)

2.4 Databases

In a world where data is becoming more and more important every day, we need tools that make it possible to store, manage, access, secure, and keep it up to date. Databases work with this in mind. A database in the strict sense is a system used for the structured and logical management of records. The user can interact with the system by using a data control language such as SQL.

A recording is the representation of a physical data or a contextual object, such as a person. Each record is stored in an architecture that can be a table for relational type databases or an object for object-oriented type databases. Once integrated into the environment, the registration is divided into several attributes such as first name, first name, sex and date of birth. Each of these attributes represents a piece of data that can be used separately or in combination with other data as needed.

Databases consist of two components. The first consists of the data described above and the second consists of metadata. The metadata provides a description of the structure of the relevant data and of the database itself. These metadata are very useful for data management and organisation. In addition, metadata provides information on the structure of the data, enabling users and systems to understand how to search for, manipulate, interpret, and use the data. (Taylor 2017, 9-10)

The administration of a database is carried out by a database management system (DBMS), such as MariaDB. The tool consists of a set of programs for the administration, definition and processing of databases. It is also through its use that data can be read, written, modified and deleted. The mode of operation of the DBMS is based on the **ACID** principle (atomicity, coherence, isolation and durability). (IONOS, 13 June 2023)

2.4.1 Relational Database

A database retains data according to its own model. This model describes the logical structure, as well as the constraints and relationships that determine the accessibility of the data and how it is stored within the database. One of the most well-known database models is **the relational model**. Developed in 1970 by IBM researcher Edgar F. Codd, the model was created with a view to minimizing redundancy and making it easier to understand the database structure of the first database model, **the hierarchical model**. (IBM, s.a.)

Functioning

The data within the relational model are organized into different tables governed by rows and columns. Each row in a table represents a single record, and the columns contain a single attribute describing the data contained in it, such as last name, first name, or date of birth. Since this model is entirely based on the relationship principle, the relationships between the tables in the database are ensured using the concepts of primary and foreign keys. Having the particularity of being unique, these two keys make it possible to clearly illustrate and validate the relationship between the tables.

In the case of the database that is the subject of this thesis, we are talking about one table with the data of the participants and another with the data set of an event. Using the primary key of an attendee and the key of an event, it is possible to symbolize that the attendee is going to participate in that event. (IBM, s.a.)

Limitation

Relational databases have many advantages, such as the fact that their use provides a clear picture and understanding of the database architecture. In addition, their use provides processes that reduce redundancy, such as the standardisation process that ensures the uniqueness of information by using primary keys.

However, the relational model has its limitations. Relational databases are put in place to ensure the storage, management and organisation of large amounts of structured data, making a wise choice for the management of structured big data. Unfortunately, the relational model becomes less effective when the data is less structured. This is what happens when data from big data is processed, because it is extremely varied and unstructured. To overcome this problem, it is necessary to use non relational databases, more commonly known as NoSQL databases, such as Neo4j or MangoDB. (Azure, s.a.)

Despite this, the relational model remains one of the most widely used database models around the world. Due to its notoriety, it enjoys a large community that allows its evolution through open-source projects such as MySql and MariaDB. (IBM, s.a.)

2.4.2 SQL

SQL or Structured Query Language is a non-procedural language, which means that you simply ask the system what you want, but without explaining how to get it. Created by Don Chamberlin and Ray Boyce, this language allows interaction between the user and the relational database management system. Using it allows administrators to create, edit, add and delete data from a database. Although it is a non-procedural language, it incorporates procedural statements such as the « **begin** »block, the ability to set conditions with « **IF** » statements, and **functions** and **procedures**. (Taylor 2017, 26)

2.4.3 MariaDB

MariaDB is an open-source relational database system licensed under GPLv2. It is rooted in the source code of MySQL. This DBMS was created by Finnish Michael Widenius following the acquisition of MySQL (also created by Michael Widenius) by Oracle in 2009. MariaDB is one of the most popular DBMS on the market. It is used by Wikipedia, Wordpress and Google. The versions and maintenance of the database are carried out by the MariaDB Foundation. (Mariadb, s.a.)

2.4.4 MariaDB and MySQL

Although MySQL and MariaDB are based on the same software kernel, their respective developments diverge due to two different visions following MySQL's takeover. MySQL's development has taken a more commercial direction. Although Oracle continues to provide a community version, this version does not offer all the features available with the Enterprise version.

In contrast, MariaDB follows a vision that aims to preserve the very essence of the basic MySQL project, an open-source project made for the community and maintained by the community. With this in mind, all new version development has been carried out on new versions of MySQL up to version 7. The aim was to facilitate transitions and exchanges between the two DBMSs. (IONOS, 18 October 2022)

Having set out these two visions, let's now look at the differences between these two protagonists of the relational database market.

The first point on which MariaDB and MySQL differ is their respective licences. MariaDB is based on a completely open-source licence, the GPLv2. MySQL is based on a dual licence, commercial and GPL, which means that customers who do not want to make their projects open-source can choose between the two licences and not have to change DBMS. (Shahzeb, 4 November 2022)

The second point is the ability to pool threads. MySQL allows a cumulative number of 200,000 connections with its enterprise edition, but this does not apply to its Community version, which only supports a lower, static number of threads. MariaDB, on the other hand, can manage more than 200,000 simultaneous connections regardless of its version. (AWS, s.a.)

The third point concerns the database engines compatible with the two DBMSs. MariaDB offers a wider choice of storage engines than MySQL. This list includes XtraDB, Memory Storage Engine, MariaDB ColumnsStore, Aria, Cassandra Storage Engine and Connect. This list also includes engines that are not compatible with MySQL. (Zahra, 6 January 2023)

The fourth point is the management of JSON data, which both DBMSs support, enabling JSON data to be retrieved and stored. However, MariaDB did not originally support JSON data types natively, and it is only since version 10.2 that it supports them, whereas MySQL already did from the outset. Another difference lies in the way JSON forms are stored, since MySQL stores them as binary objects, whereas MariaDB stores them in string format. MariaDB also supports JSON features that MySQL does not, such as **JSON_QUERY** and **JSON_EXIST**, but not the **JSON_TABLE** function, whereas MySQL does. (AWS, s.a.) (Shahzeb, 4 November 2022)

The final point concerns the security offered by the two DBMSs. MySQL uses the SHA-256 algorithm to provide a default authentication solution, while MariaDB uses authentication plugins such as the **mysql_native_password** plugin. MySQL also allows the use of the **validate_password** component natively to reinforce password security and control, as well as the configuration of password rules. MariaDB does not use the same component, but offers three plugins for password control. The first is the **simple_password_check** plugin, which allows you to perform basic checks, such as size checks and checks on specific characters (upper case, lower case, numbers, special characters). The second is the **cracklib_password_check** plugin, which checks the strength of passwords using the **CrackLib** library. The final plugin is the **password_reuse_check** plugin, which prevents users from reusing the same passwords. (Shahzeb, 4 November 2022)

Finally, I decided not to take into account the performance in terms of speed of transactions of the two DBMSs. I made this choice based on the fact that these various tests depend on several factors and that the results can differ from one source to another, although the test results are generally in favour of the MySQL DBMS. (Zahra, 6 January 2023)

3 Empirical Part

3.1 Starting Point

The project described in this thesis aims to create a first version of an open-source back-end for an event management platform. Being an open-source project, I was forced to choose and use technologies in line with the open-source philosophy. As I mentioned in the « Objective » section of the introductory chapter, this version is developed as a starting point for a larger project.

The project is designed independently, is not impacted by the intervention of any external party and does not target any particular company or organization. However, in the long term, I would like to orient the development of this event management platform according to the needs of university campuses for the organization of internal events.

In the context of this thesis, the development was thought to provide functionalities that aim to meet the primary needs required by an event management platform. This includes creating events (type of event, location, age restrictions, etc.), managing events (management of registrations and attendees, deletion of the event, etc.), creating users (type of user, generation of token, etc.) and managing users (modification of personal information, password change service, deletion of the user, etc.).

The limiting factors within any project can be many. Since I am developing this project entirely on my own, it is subject to major interruptions in the event of technical or personal problems, and so it is a factor to be considered. From that point on, I think it's more sensible to focus on quality rather than on the amount of functionality that the project brings.

Finally, to determine whether the whole project developed through this thesis is viable and of quality, it must meet certain criteria:

- The first criterion is based on the management concept, i.e. the various basic functionalities that guarantee the management of an event must be implemented and fully functional.
- The second criterion is to provide CRUD functionality through the API for the address, activity, opinions and tags.
- The third criterion is to make tests for each endpoint of the API, in order to ensure their proper functioning.
- The fourth criterion concerns safety. The project must provide an authentication feature to access the different functionalities of the project, as well as a regulation of access to endpoints depending on the type of user making the request.

These different criteria represent for me a set of basic elements that allow me to determine whether the final result of my thesis is a success or not.

3.2 Illustration of the outcome

Through this thesis, we have so far gone through many subjects from a theoretical point of view. In this practical chapter, we will delve into the explanation of the different phases of the development of this thesis project.

The purpose of this work is to propose a back-end application using the Spring BOOT Framework. In order to allow exchanges between the back-end and a user, the project must provide a set of methods through the REST service. The interaction does not take place directly between the user and the REST API, but with the help of a front-end that communicates with the back-end through the REST API within a client-server architecture.

In the diagram below, we can see schematically the concepts I discussed in the previous paragraph. The user executes an action through the front-end and the action is converted into a query using HTTP verbs. The request is then sent to the server which redirects it to the relevant REST controller using URL routes. Depending on the request, the controller calls repositories or services. Then, the repository executes CRUD requests between the database and the back-end based on the request sent to it by the controller and sends the entity back to the controller. Finally, the

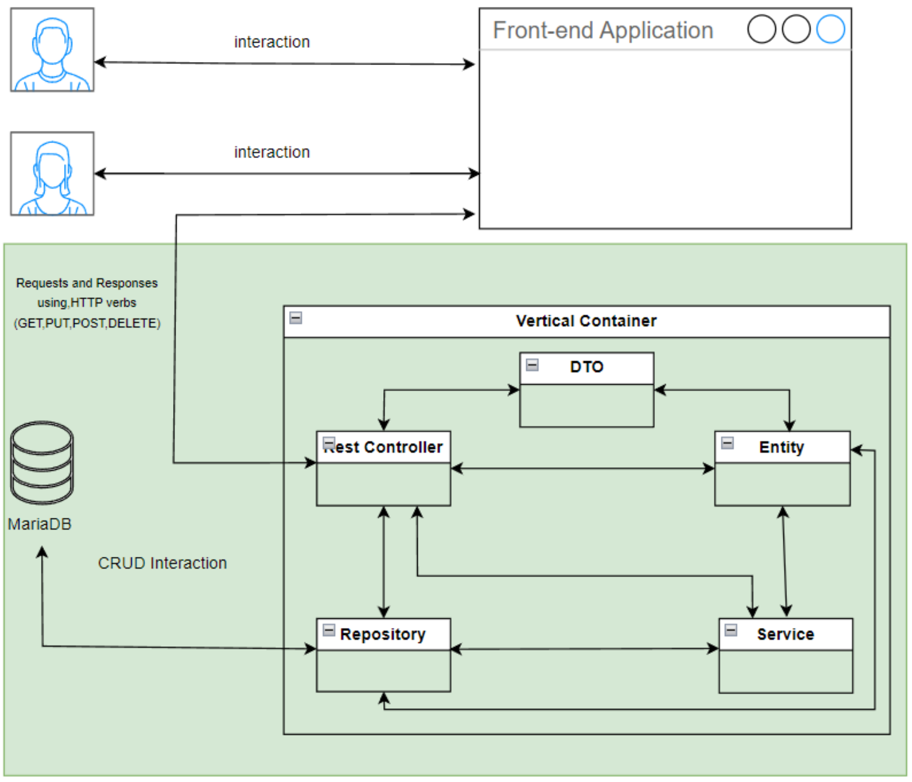


Figure 2. Diagram of the project's final architecture. (self-made)

controller generates the response and sends it to the client by returning a DTO object of the corresponding class to the client. DTO stands for Data Transfer Object. It is a data model used to transfer data between different parts of an application. It is often used between the presentation layer, which corresponds to the user interface, and the data processing layer, which would be the back-end. A DTO object is used to group together a selected set of data in order to avoid exposing the entire internal structure. For example, if you need to return information about a particular user, using a DTO object you can choose to send the user's information, removing sensitive information such as passwords. (baeldung, 22 December 2022)

We can see that the diagram is divided into two distinct parts. The part delimited by the rectangle on the green background, represents the elements addressed by the project. The rest of the diagram outside the black rectangle does not fall within the scope of this thesis.

3.3 Methodologies

Through this section, I will explain the methodology, as well as the tools used to complete my thesis. Being basically a person with a few shortcomings in terms of organizational sense, I have never really used any particular methodologies or tools for the development of my past projects. However, the scope and importance of the writing and development of the Bachelor’s thesis made me aware of the need for seamless management of time, documents and project-related tasks. In order to meet these constraints in the best way, I put into practice what I learned during my studies, used different tools that I already knew, and learned how to use new ones.

To manage my tasks, I used the Trello tool, a web application using the Kanban approach. With this approach, I was able to get a quick and clear overview of what needed to be done, in progress or completed. The flexibility of this tool has allowed me to modify my Kanban so that it adapts to my needs. I subdivided it into three different subjects: Thesis, Spring and Databases. This allowed me to simplify the division of labour, as well as to have a better overview of the progress of the work and to group the completed tasks according to their subjects.

The Trello tool also allowed me to be more agile in my work methodology. Each task I had to perform was prioritized based on an assessment of the level of complexity and the amount of work required. As soon as a task was finished, both in the writing of the thesis and in the development of the code, it was reread, the code tested and everything corrected if necessary. Once I judged the task finished and functional, I proceeded to the next one in the order quoted above. This cascading and iterative method of working has allowed me to move forward in my work with more confidence, especially for the part concerning the development of the code where the slightest mistake may have repercussions for the future of the project.

Finally, file management was an important aspect of the overall methodology used for this project. Throughout the development process, I kept two versions of my thesis permanently in two separate environments: one was stored locally on my computer and the other was stored in a repository on the GitHub web application that I created for the thesis. I used this redundancy for two main reasons: by placing a version of my work on a cloud-based tool like GitHub, I made sure I could access it at any time in the event of an incident and from any device with an internet connection. Finally, it allowed me to have a version history of my work saved in the repository.

3.4 Project development

The development of a project like this cannot be done in a single, long phase, but in several. Dividing the development into different parts is essential to ensure that all the tasks assigned to the step are done and that any mistakes are corrected. This is to prevent the error from being carried over to the next stages. Moreover, this approach avoids a domino effect on the overall development of the project.

First phase

The first phase in the development of the project was to choose the subject of the thesis and establish the working methodology. Selecting the subject was a very important step because the whole project depended on it. The idea of creating a back-end for an event management application had been in my mind long before I started my thesis. I had to make a number of changes with the help of my accompanying teacher, in order to be able to define a subject that met the expectations of creating a Bachelor's thesis. As I explain in section **3.3 Methodology**, I wanted to develop this project using an agile working method. However, given that I was the only person involved in the project, implementing a SCRUM methodology seemed too complicated. So, using the Trello web application, I opted for a working method that combines agile and waterfall. This enabled me to focus on the most urgent tasks and carry out all the necessary checks before moving on to another task.

Second phase

The second phase was to choose the two elements that would be used to produce the code and manage the data, Spring Framework and MariaDB. As I explain in part 2.2.7 of my thesis, the choice of Spring Framework was a natural one because of my academic background and also because of its rich documentation and community. The MariaDB database management system proved to be the wisest choice. I justify my choice by my desire to create an open-source back-end that uses a relational data architecture.

Third phase

The third phase was to create the database architecture for the project. The first step was to think about the different data I would have to store, based on the functionalities I had defined in part **1.2 Scope and Limitation**. The second step was to create the conceptual data model (CDM). The conceptual data model represents the first stage in the design of a database. It is an abstract, high-level representation of a database. Its modelling is necessary in order to understand and ensure that the needs and requirements of the database are met before moving on to the design of a more detailed model. To sum up briefly, in this stage we will define the tables that will be contained in the database according to the project requirements, without looking at the relationships between the tables and the data structure. (Saxena S., 3 May 2023)

Having created the CMD, I then transformed it into a logical data model (LMD). The creation of this model is the intermediate step between the conceptual model and the physical data model. It will be used to define the way in which the data will be organised and managed, by establishing the structure of the data contained in the database tables. In conclusion, this model will provide a clear vision of how the data will be organised and connected, which helped me a lot when creating the Java classes. (TIBCO, s.a.)

Once the LMD has been modelled, the next logical step is to create the physical data model (PDM). This model represents the final phase of the database design. It will describe how the data system will be implemented and with which DBMS. However, I didn't need to create the physical data model (PDM), the database schema is generated automatically, using Hibernate and JPA. (Saxena S., 3 May 2023)

Finally, as I explain in the preamble to this section, the set of tables and relationships illustrated in the diagram below was chosen following an in-depth analysis of the different functionalities and the information they need to provide a logical and functional structure. This is the last phase before the Java Spring project is created.

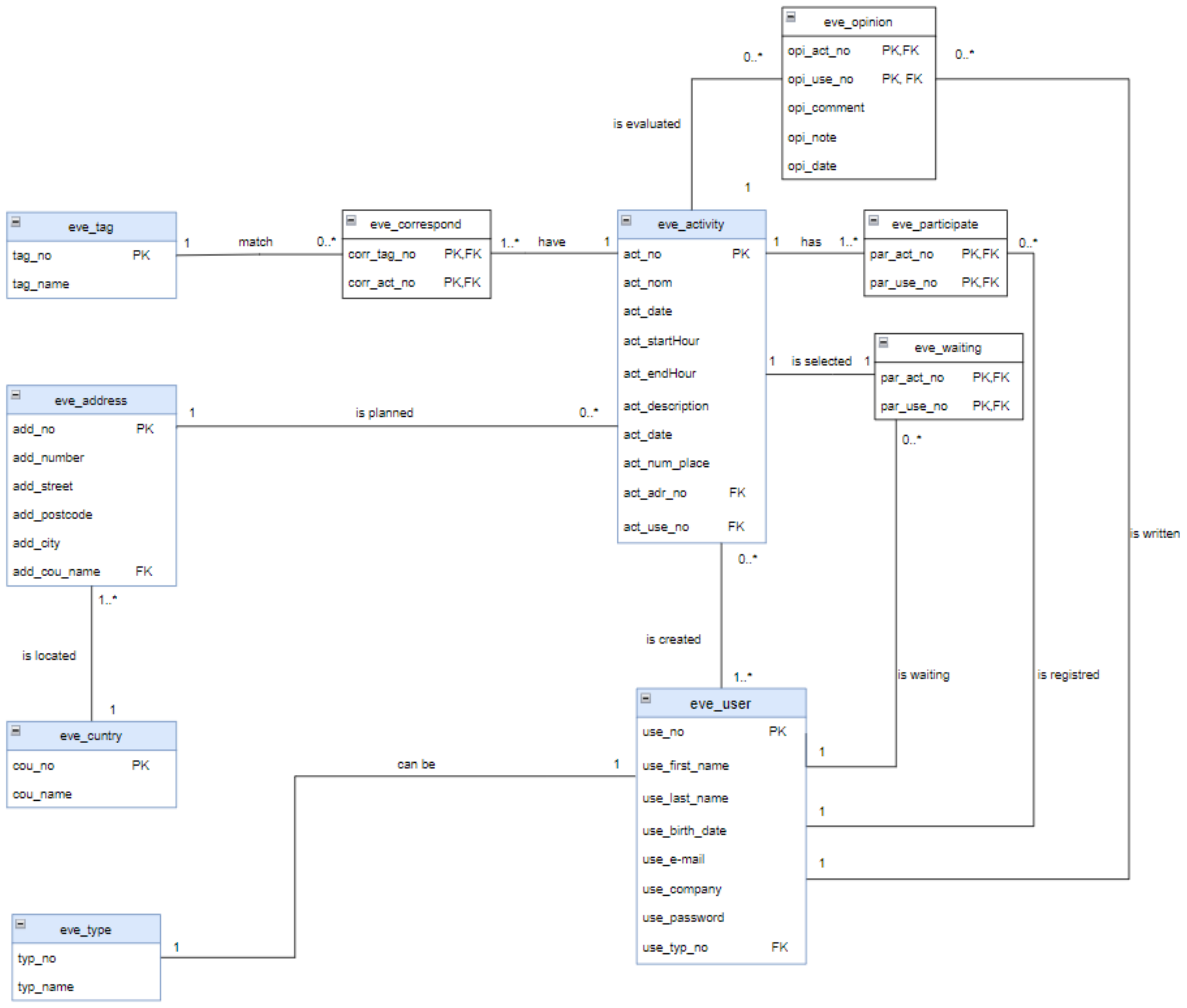
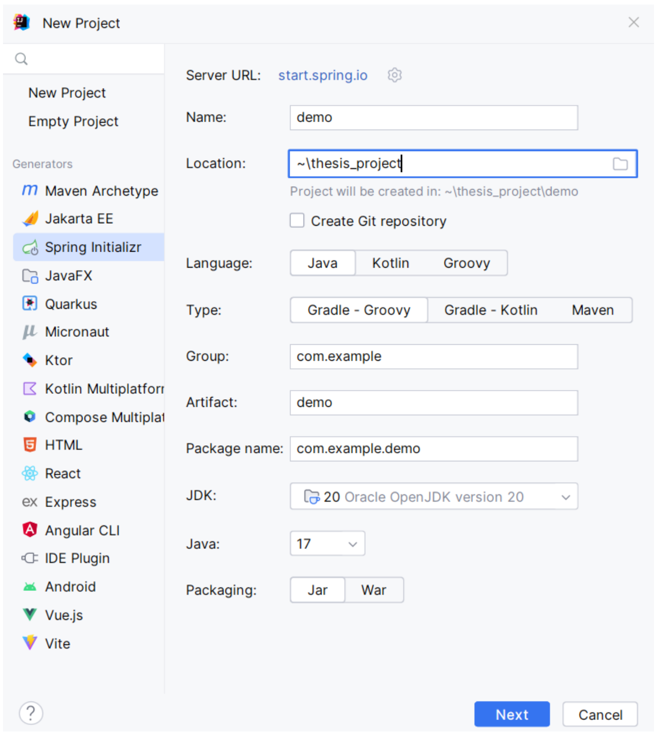


Figure 3. Logical Data Modeling. (self-made)

Fourth phase

The fourth phase was the creation of the Java Spring project. The Spring development teams implemented **spring initializr**, which allows you to generate a Spring Boot project in just a few clicks. Using it, you can choose the type of project, the programming language, the version of Spring Boot, the dependencies and the version of Java. The advantage of this web application is that it offers all the dependencies required for a Spring Boot project, so you don't have to waste precious time looking for dependencies in the Maven repository.

There are two ways of accessing **spring initializr**. The first method is to use the web application: https://start.spring.io/. The second method lies in the fact that some IDEs, such as **IntelliJ IDEA**, offer spring initializr natively when a Spring Boot project is created. This method avoids having to import the project into the IDE after it has been created, so you can start working on the project straight away. I've been using IntelliJ IDEA for 3 years now, so I've opted for the second method, which I find the most comfortable and practical for developing my project.



Server URL: start.spring.io
Name: demo
Location: ~\thesis_project
Project will be created in: ~\thesis_project\demo
Create Git repository
Language: Java Kotlin Groovy
Type: Gradle - Groovy Gradle - Kotlin Maven
Group: com.example
Artifact: demo
Package name: com.example.demo
JDK: 20 Oracle OpenJDK version 20
Java: 17
Packaging: Jar War

Figure 4. Spring intializr IntelliJ IDEA.

Among the dependency choices offered by **spring initializr**, I have selected the following dependencies according to the needs of my project:

- **Spring Web:**

  This dependency provides all the functionality offered by Spring for developing web applications of all types, including RESTful applications. Because my project requires the implementation of a REST service, I had to choose this dependency.

- **Spring Data JPA:**

  This dependency simplifies access to relational databases using the Java JPA tool. This dependency proved essential for transforming classes into entities, which also avoided the creation of the physical data model.

- **Spring Security**

  This dependency offers a set of robust security functions, including user management, authentication and authorisation. The decision to integrate this dependency into my project was based on the fact that I needed it to implement the project's security, but also to implement the JWT[2].

- **Spring HATEOAS**

  Spring HATEOAS is a dependency that simplifies the implementation of the HATEOAS approach. As I explained in section 2.3.4 HATEOAS and Spring, in order to make my project as RESTful as possible, I needed to implement the HATEOAS approach. That's why I chose this dependency.

- **MariaDB Driver**

  In order to be able to use MariaDB as a database for the development of my project, I had to add this dependency. It allows you to add MariaDB JDBC drivers.

Fifth phase

The fifth phase consisted of testing whether the project was working correctly by executing a test class controller which, via a REST method, would return the "Hello World" message. This allowed me to check that all the project's configuration was working and that the Tomcat server was accessible.

Sixth phase

The sixth phase was to implement the project's **application.properties** file. This file is used to configure various elements of the project without having to manipulate the source code directly. Between the various configurations made possible by this file, I focused on using it to configure the database, so that the project could automate the database connection and communicate efficiently with MariaDB via Hibernate. To do this, I used the options linked to the Spring Boot configuration to indicate the following information:

- **The database connection URL the database.**

  The spring.datasource.url property is used to specify the URL for connecting to a specific database. In the context of this project, this URL**: jdbc:mariadb://localhost:3306/event_manager** is used to reach the MariaDB database named **event_manger**. This is the project database. Without this information, communication between the project and the database would not be possible.

[2] JSON Web Token

- **The username**

  The **spring.datasource.username** property is used to specify the username used to access the project database. In the context of this project, the username is **root**. Although the URL is specified, this information must also be provided to access the database.

- **The password**

  The **spring.datasource.password** property is used to specify the password of the user indicated in the **spring.datasouce.username** property. This property is essential for finalising authentication to the database. If there is no password on the database, this property does not need to be specified. However, I have specified it in anticipation of a future developer wanting to use my back-end and needing to set a password.

- **Drivers**

  The **spring.datasource.driver-class-name** property is used to tell JDBC which driver class it should use for MariaDB.

- **The database initialisation mode**

  The **spring.datasource.init-mode** property is used to determine how and when Spring Boot should execute the database initialization SQL scripts. There are three options.

  The first is **always**, which means that each time the database is started, the initialization scripts must be executed. This option is more suited to application development and testing. I have specified this mode in a testing context, so that I can always start from a given database with test data already present.

  The second option is **embedded**. Its use implies that the SQL scripts are only executed if the database is embedded like the H2 data management system.

  Finally, there is the **never** option, which prevents Spring Boot from automatically executing the SQL scripts.

- **Automatic generation of the data schema based on JPA entities.**

  The **spring.jpa.generate-ddl** option is used to tell Spring Boot, if **true**, to automatically generate the database schema based on JPA entities. Thanks to the use of JPA and this option, I didn't need to create the physical data model.

- **Automatic generation of tables following each start-up.**

  The **spring.jpa.hibernate.ddl-auto** option is used to configure Hibernate's behaviour when generating and updating database schemas. This option offers five modes.

The first is **none**. This will prevent Hibernate from automatically generating the database schema. It must be generated manually by the developer.

The second is **validate**. This will tell Hibernate to validate only existing database schemas against JPA entities.

The third is **update**. This tells Hibernate to update the database schema only if the JPA entities change. This means that it will not regenerate the entire database schema each time a table is added, but only add it to the existing schema.

The fourth mode is **create**. This tells Hibernate to create the database schema each time the application is started. This can lead to data loss if the table to be generated already existed in the database.

The fifth is create-drop. It tells Hibernate to perform the same action as the create mode, with the exception of deleting the schema each time the application is stopped. The use of this mode is more suited to running tests or developing the application. I opted to use it so that I could run tests every time I finished programming a controller.

- **The display of different SQL queries, to make it easier to identify a data persistence problem.**

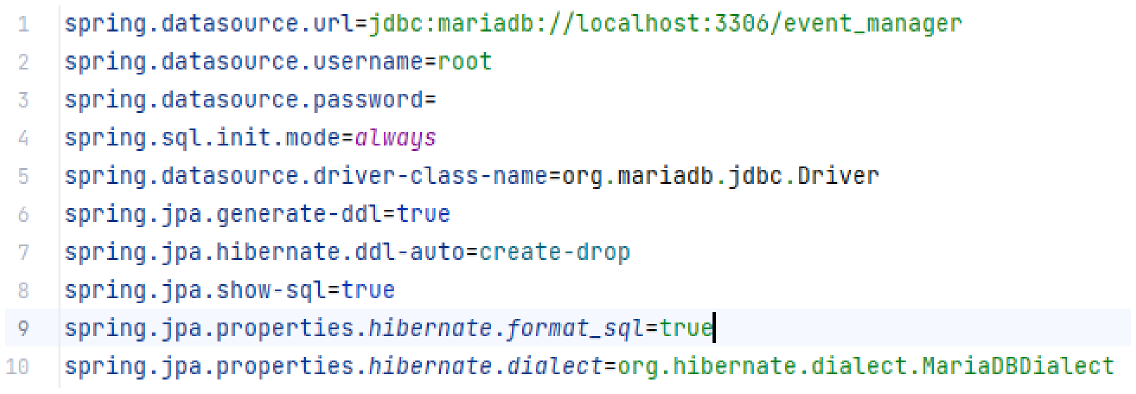
  The **spring.jpa.show-sql** option allows SQL queries made by Hibernate to be displayed in the console, but only if the option is set to **true**. Using this option is useful for debugging a function. In my case, it was useful when creating the controllers to see the transactions between the database and the project.

- **Formatting Hibernates queries in SQL.**

  The **spring.jpa.properties.hibernate.format_sql** option, when set to **true**, formats the SQL queries made by Hibernate, to make them more readable for the user in the console.

- **An indication of the type of dialect Hibernate should use to communicate with MariaDB.**

The **spring.jpa.properties.hibernate.dialect** option is used to specify which dialect Hibernate should use to communicate with the database. More specifically, it allows Hibernate to generate its requests with MariaDB, as shown in the screenshot of my **application.properties** file below.

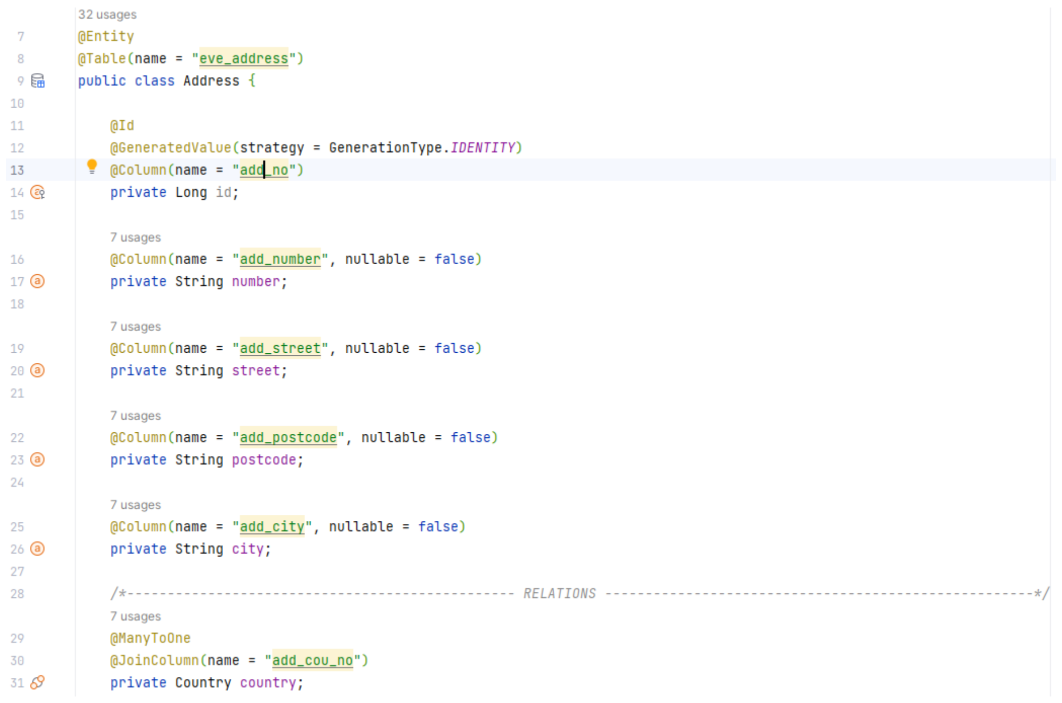


```
spring.datasource.url=jdbc:mariadb://localhost:3306/event_manager
spring.datasource.username=root
spring.datasource.password=
spring.sql.init.mode=always
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
```

Figure 5. Application properties file. (self-made)

Seventh phase

The seventh phase was to implement the different java classes according to my LDM, but without adding the different JPA annotations and relationships. To avoid errors as much as possible, I transformed my different classes into entities by adding the JPA annotations, one after the other. All the while respecting the relationships, I had modelled in the LDM. This enabled me to test the entity each time to see if there were any problems with data persistence and JPA when the server was started. As soon as an entity was tested, I tried out the relationships linking it to other entities, such as the relationship linking a user to a role.



```
32 usages
@Entity
@Table(name = "eve_address")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "add_no")
    private Long id;

    7 usages
    @Column(name = "add_number", nullable = false)
    private String number;

    7 usages
    @Column(name = "add_street", nullable = false)
    private String street;

    7 usages
    @Column(name = "add_postcode", nullable = false)
    private String postcode;

    7 usages
    @Column(name = "add_city", nullable = false)
    private String city;

    /*------------------------------------------------ RELATIONS ------------------------------------------------------*/
    7 usages
    @ManyToOne
    @JoinColumn(name = "add_cou_no")
    private Country country;
```

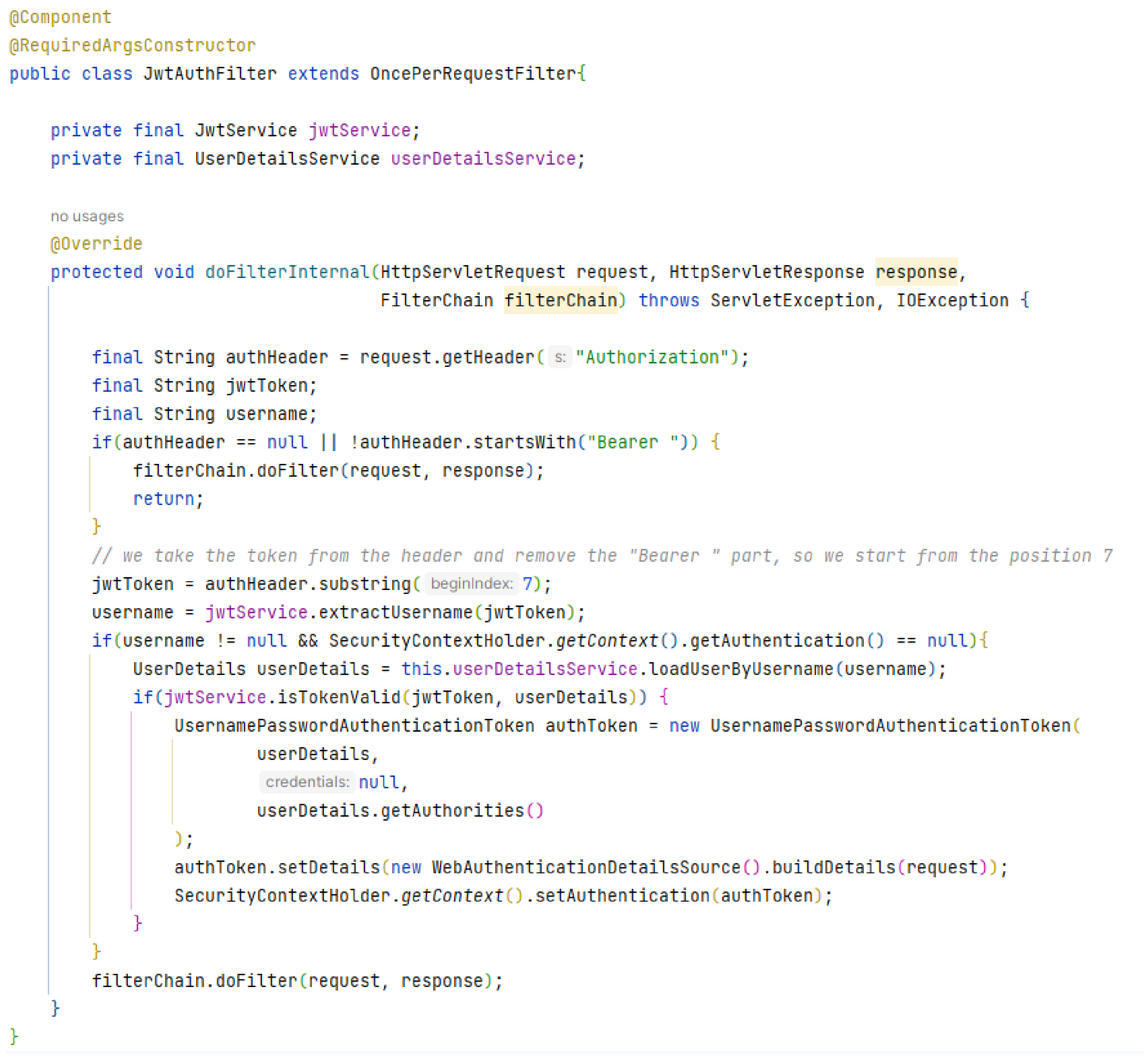
Figure 6. Exemple of a class entity. (self-made)

Eighth phase

With the creation and implementation of the JPA entities complete, I focused on implementing the security for my project, through **the server programming course** I was able to take at the Haaga-Helia University of Applied Sciences in Helsinki. I had the opportunity to develop an application using the Spring Framework. The development of this application enabled me to understand the importance of considering the implementation of security from the outset of the project. Implementing the entire application and leaving the security implementation to the end can lead to security flaws and major changes to the code, which can cause further errors. It was for this reason that security was the element I implemented before developing the various features of the project.

The first step I took was to encrypt sensitive data, such as user passwords. To do this, I used the **BCrypt** hash tool, which provides a set of functions for hashing passwords. I was able to use this tool thanks to the addition of the **Spring Security** dependency. The password is hashed when the user is created, to avoid registering a new user in the database with an unencrypted password.

The second measure was the implementation of Json Web Token (JWT) for user registration and authentication, so that users can access the resources exposed through the REST API. The decision to implement JWT instead of using the pre-established Spring Securtiy functionalities was made in order to respect the REST principle of statelessness. In fact, using JWT will make it possible, via the generation of its token, to provide the user information required for authentication, such as the username. All this without having to retain the state of the session. Implementing JWT required a modification to Spring Security, which does not provide a pre-established tool for generating authentication tokens. To do this, I had to implement a new security filter to authorise users to authenticate with JWT tokens, using the creation of a class to filter the JWT authentication token, which I named **JwtAuthFilter**. This class allows the application's security filter to delegate the task of verifying the token.



```java
@Component
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter{

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    no usages
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        final String authHeader = request.getHeader( s: "Authorization");
        final String jwtToken;
        final String username;
        if(authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        // we take the token from the header and remove the "Bearer " part, so we start from the position 7
        jwtToken = authHeader.substring( beginIndex: 7);
        username = jwtService.extractUsername(jwtToken);
        if(username != null && SecurityContextHolder.getContext().getAuthentication() == null){
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);
            if(jwtService.isTokenValid(jwtToken, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
                        userDetails,
                        credentials: null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

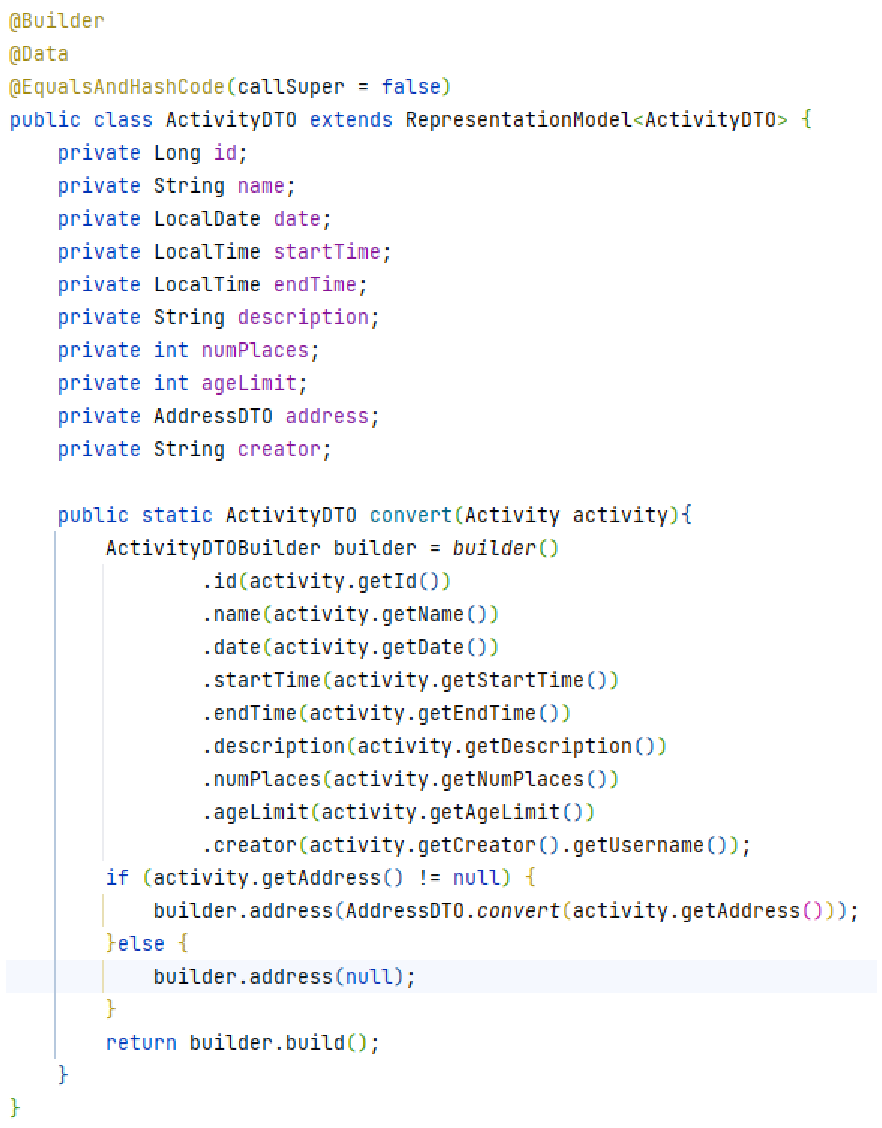
Figure 7. Class JwtAuthFilter. (self-made)

The third step was to set up the management of access authorisations to the endpoints present in the controllers, depending on the role of the user making the request. To implement these authorisations, I had to implement a UserDetailsServiceImpl class, which fetches all the users authenticated to the application. Following this, using the **@PreAuthorise** annotation, provided by Spring Securtity, the application is able, depending on the roles passed in parameter, to authorise access to methods according to the user's role.

Ninth Phase

Creating the REST API for my back-end involved creating 5 elements: DTO classes, requests, re-positories, services and controllers.

After creating the JPA class entities, I modelled a Data Transfer Object (DTO) class based on each of the JPA class entities. This enabled me to choose which data was interesting to return to the user, but also to avoid disclosing sensitive information such as a user's password in the responses returned by the controllers.

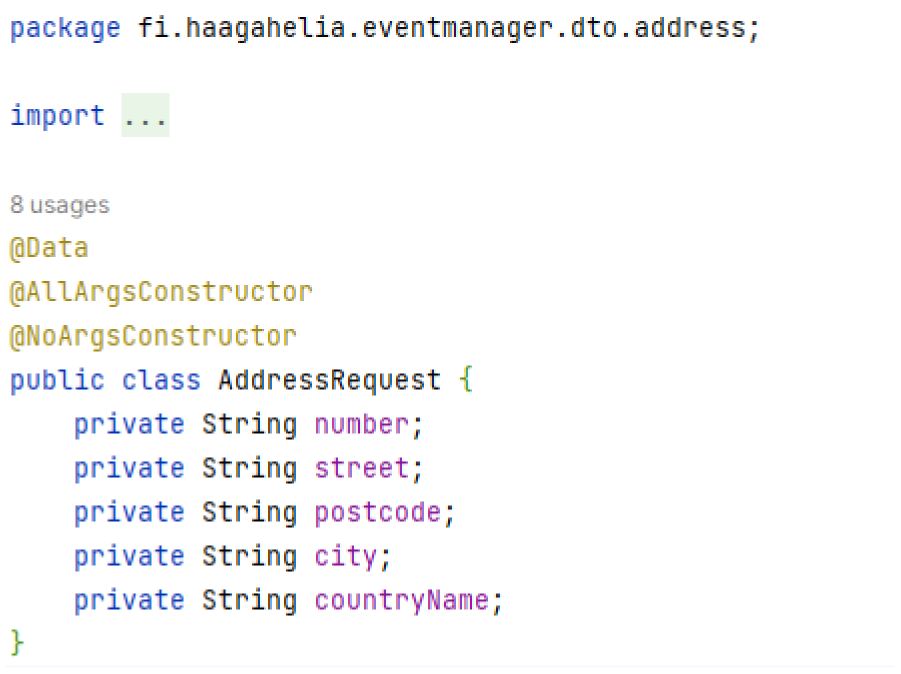


```
@Builder
@Data
@EqualsAndHashCode(callSuper = false)
public class ActivityDTO extends RepresentationModel<ActivityDTO> {
    private Long id;
    private String name;
    private LocalDate date;
    private LocalTime startTime;
    private LocalTime endTime;
    private String description;
    private int numPlaces;
    private int ageLimit;
    private AddressDTO address;
    private String creator;

    public static ActivityDTO convert(Activity activity){
        ActivityDTOBuilder builder = builder()
                .id(activity.getId())
                .name(activity.getName())
                .date(activity.getDate())
                .startTime(activity.getStartTime())
                .endTime(activity.getEndTime())
                .description(activity.getDescription())
                .numPlaces(activity.getNumPlaces())
                .ageLimit(activity.getAgeLimit())
                .creator(activity.getCreator().getUsername());
        if (activity.getAddress() != null) {
            builder.address(AddressDTO.convert(activity.getAddress()));
        }else {
            builder.address(null);
        }
        return builder.build();
    }
}
```

Figure 8. Class ActivityDTO. (self-made)

Some entities require the user to pass various pieces of information to the controller in order to be created. This information is passed to the controller via the request body. However, if these queries are not parameterised, they may present a risk of SQL injection or other attack that could threaten the integrity of the application. To avoid this, I have created parameterised requests for all entities requiring additional information to be created, such as an event or a user. The parameterised requests will contain all the mandatory attributes for creating the object.



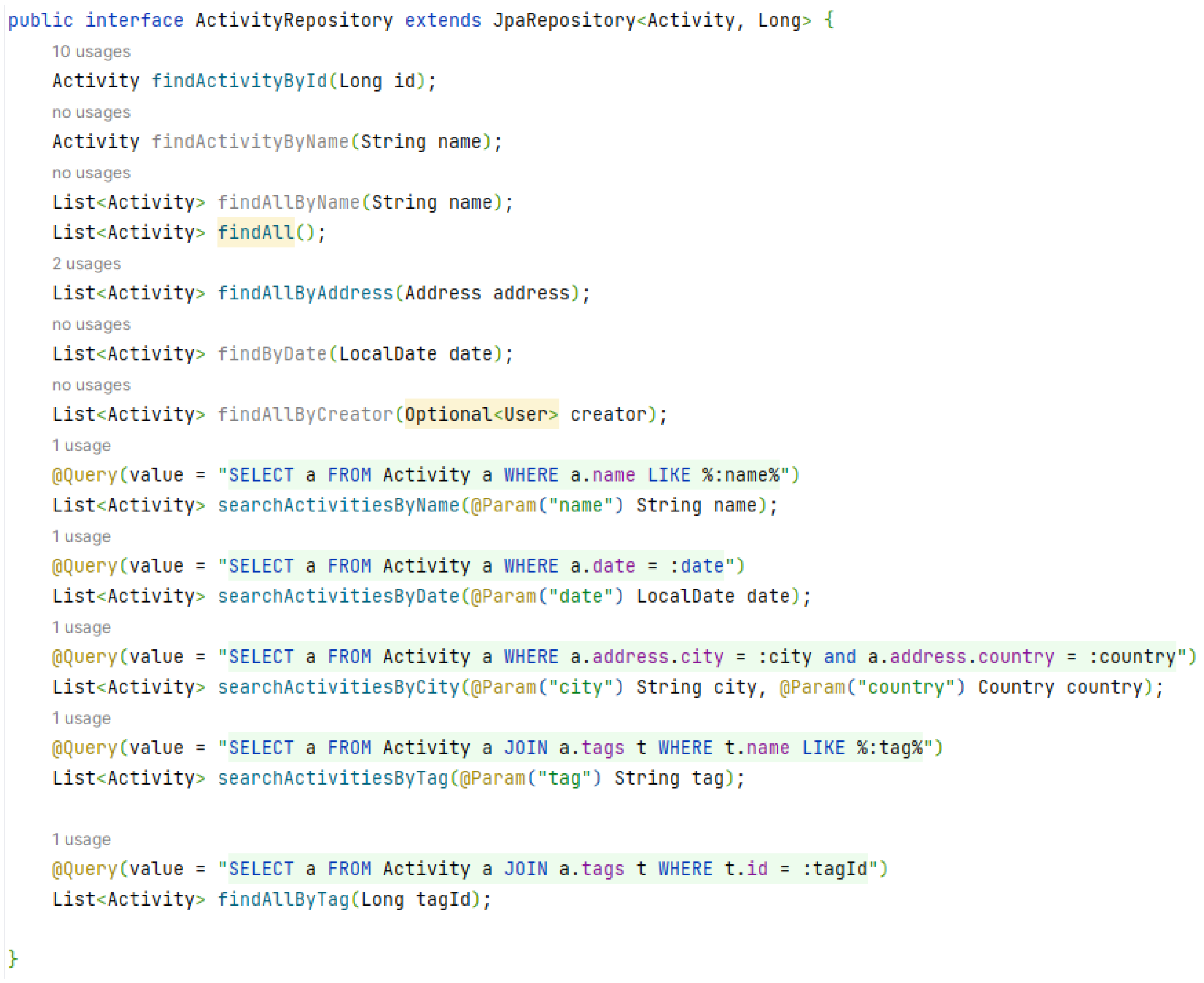
```java
package fi.haagahelia.eventmanager.dto.address;

import ...

8 usages
@Data
@AllArgsConstructor
@NoArgsConstructor
public class AddressRequest {
    private String number;
    private String street;
    private String postcode;
    private String city;
    private String countryName;
}
```

Figure 9. Class AddressRequest. (self-made)

The aim was to provide the necessary tools to the service classes so that they could respond to the various requests sent by the controller. I also implemented repositories linked to the various JPA entities to perform the various actions on the database. The repository is an interface that extends the JpaRepository interface. This will allow real involvement of the various interactions with the database, by providing methods that the service classes can use to persist, retrieve and delete data within the database.



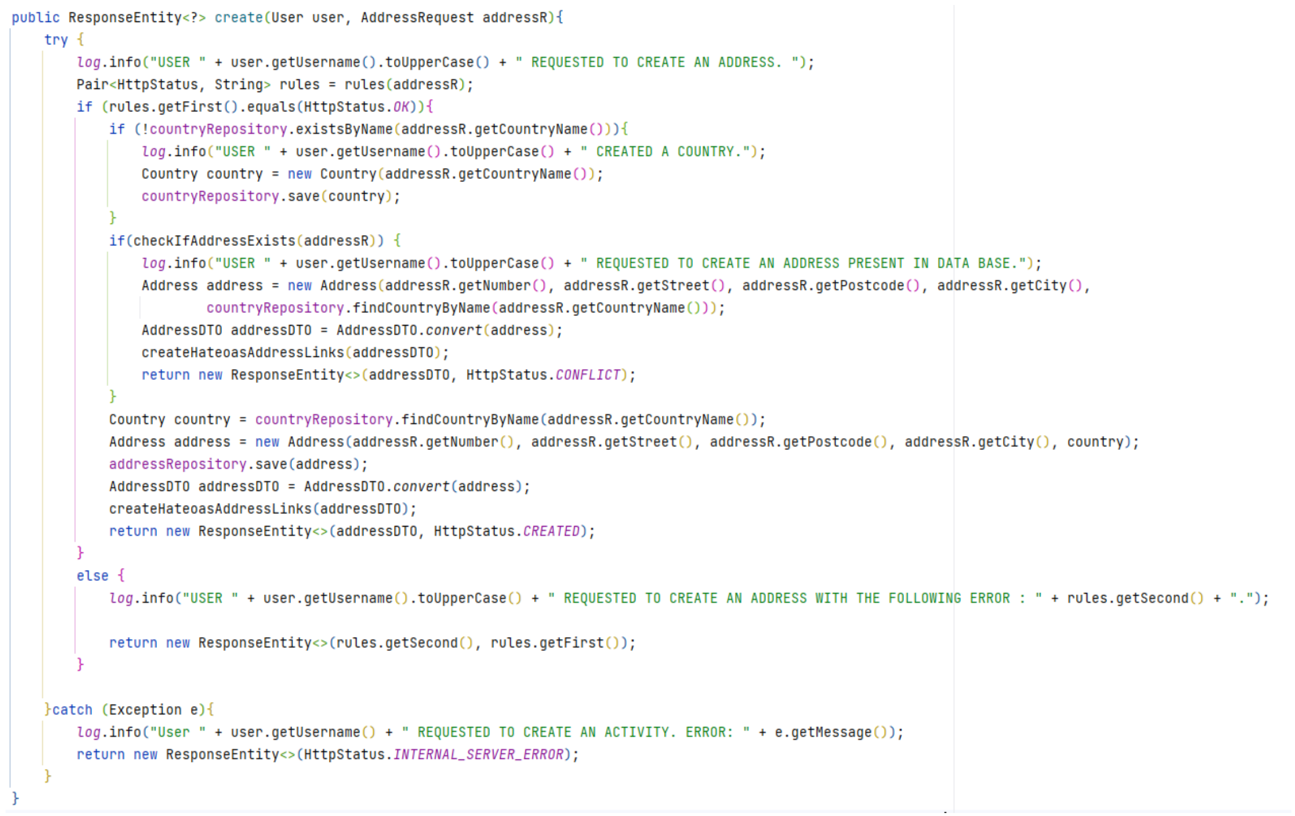
```
public interface ActivityRepository extends JpaRepository<Activity, Long> {
    10 usages
    Activity findActivityById(Long id);
    no usages
    Activity findActivityByName(String name);
    no usages
    List<Activity> findAllByName(String name);
    List<Activity> findAll();
    2 usages
    List<Activity> findAllByAddress(Address address);
    no usages
    List<Activity> findByDate(LocalDate date);
    no usages
    List<Activity> findAllByCreator(Optional<User> creator);
    1 usage
    @Query(value = "SELECT a FROM Activity a WHERE a.name LIKE %:name%")
    List<Activity> searchActivitiesByName(@Param("name") String name);
    1 usage
    @Query(value = "SELECT a FROM Activity a WHERE a.date = :date")
    List<Activity> searchActivitiesByDate(@Param("date") LocalDate date);
    1 usage
    @Query(value = "SELECT a FROM Activity a WHERE a.address.city = :city and a.address.country = :country")
    List<Activity> searchActivitiesByCity(@Param("city") String city, @Param("country") Country country);
    1 usage
    @Query(value = "SELECT a FROM Activity a JOIN a.tags t WHERE t.name LIKE %:tag%")
    List<Activity> searchActivitiesByTag(@Param("tag") String tag);

    1 usage
    @Query(value = "SELECT a FROM Activity a JOIN a.tags t WHERE t.id = :tagId")
    List<Activity> findAllByTag(Long tagId);

}
```

Figure 10. Repository fro the activity entity. (self-made)

The service classes were implemented based on the JPA entity classes developed at the start of the project. They handle all the project's business logic. The service classes provide a set of methods that will be called by the controllers according to the user's request. They are implemented to return DTO objects by marshalling them from a Java class to a JSON object to guarantee an efficient and readable exchange between the back-end and the front-end. In order to make them correspond as closely as possible to the REST philosophy, I implemented a method in each of the service classes, which will be responsible for generating the HATEOAS links according to the DTO object to be returned. The responses are returned to the controller encapsulated in a **ResponseEntity** containing the DTO object and the http status.



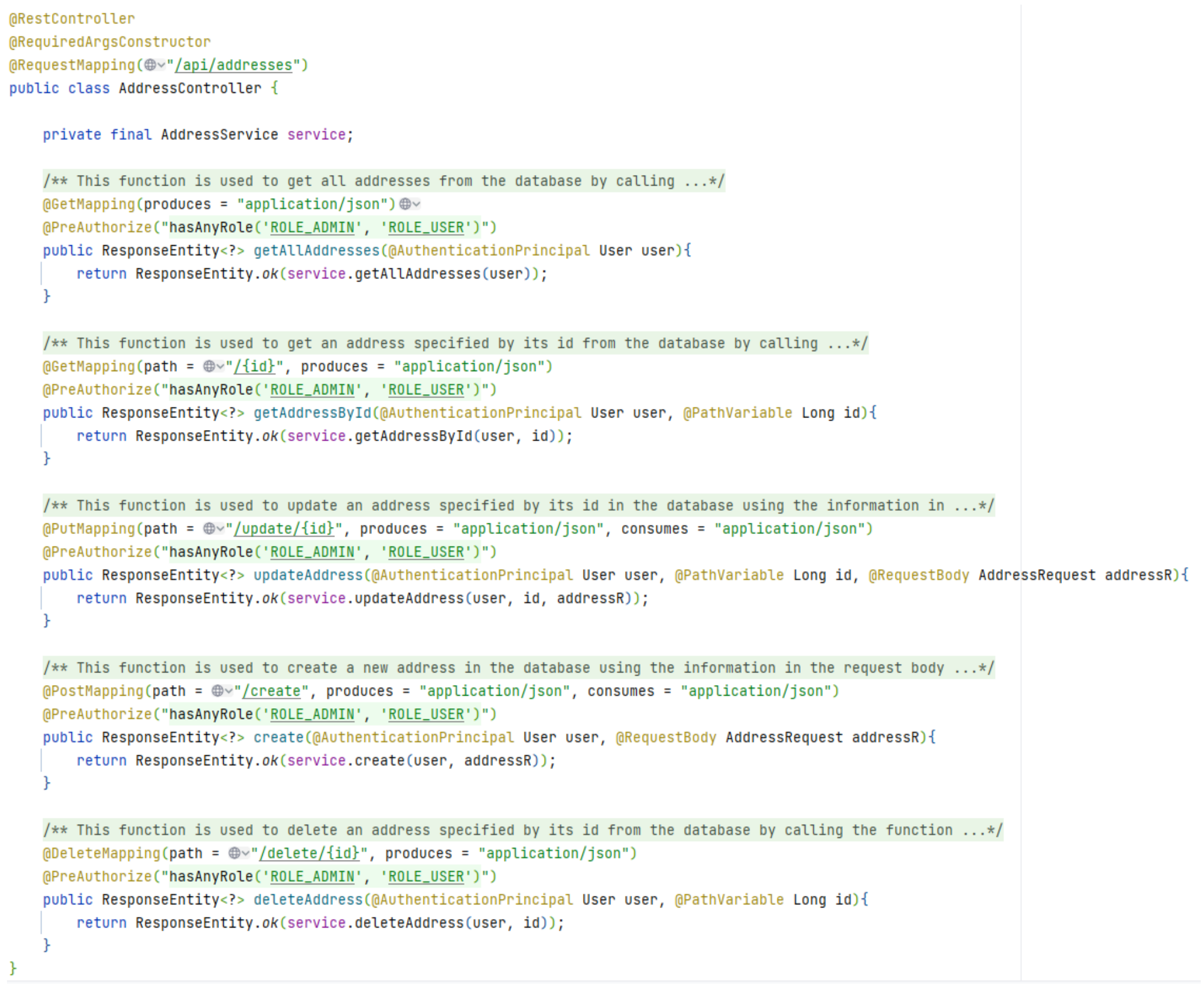
```java
public ResponseEntity<?> create(User user, AddressRequest addressR){
    try {
        log.info("USER " + user.getUsername().toUpperCase() + " REQUESTED TO CREATE AN ADDRESS. ");
        Pair<HttpStatus, String> rules = rules(addressR);
        if (rules.getFirst().equals(HttpStatus.OK)){
            if (!countryRepository.existsByName(addressR.getCountryName())){
                log.info("USER " + user.getUsername().toUpperCase() + " CREATED A COUNTRY.");
                Country country = new Country(addressR.getCountryName());
                countryRepository.save(country);
            }
            if(checkIfAddressExists(addressR)) {
                log.info("USER " + user.getUsername().toUpperCase() + " REQUESTED TO CREATE AN ADDRESS PRESENT IN DATA BASE.");
                Address address = new Address(addressR.getNumber(), addressR.getStreet(), addressR.getPostcode(), addressR.getCity(),
                        countryRepository.findCountryByName(addressR.getCountryName()));
                AddressDTO addressDTO = AddressDTO.convert(address);
                createHateoasAddressLinks(addressDTO);
                return new ResponseEntity<>(addressDTO, HttpStatus.CONFLICT);
            }
            Country country = countryRepository.findCountryByName(addressR.getCountryName());
            Address address = new Address(addressR.getNumber(), addressR.getStreet(), addressR.getPostcode(), addressR.getCity(), country);
            addressRepository.save(address);
            AddressDTO addressDTO = AddressDTO.convert(address);
            createHateoasAddressLinks(addressDTO);
            return new ResponseEntity<>(addressDTO, HttpStatus.CREATED);
        }
        else {
            log.info("USER " + user.getUsername().toUpperCase() + " REQUESTED TO CREATE AN ADDRESS WITH THE FOLLOWING ERROR : " + rules.getSecond() + ".");

            return new ResponseEntity<>(rules.getSecond(), rules.getFirst());
        }

    }catch (Exception e){
        log.info("User " + user.getUsername() + " REQUESTED TO CREATE AN ACTIVITY. ERROR: " + e.getMessage());
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Figure 11. Example of a service method to create an address. (self-made)

Finally, I implemented my controller classes based on my JPA entity classes. However, the controllers in my project didn't implement any business logic. As I indicated above, the controller will be in charge of http requests and interaction with the client, leaving the business logic to the linked service class. This division clearly separates the application's responsibilities, but also makes the code more modular and easier to maintain. Each controller is accessible via a specific URI starting with: /api/{**top-level resource**}/. **The top-level resource** corresponds to the controller in question. Everything after the last '/' will call another controller method.



```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/addresses")
public class AddressController {

    private final AddressService service;

    /** This function is used to get all addresses from the database by calling ...*/
    @GetMapping(produces = "application/json")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
    public ResponseEntity<?> getAllAddresses(@AuthenticationPrincipal User user){
        return ResponseEntity.ok(service.getAllAddresses(user));
    }

    /** This function is used to get an address specified by its id from the database by calling ...*/
    @GetMapping(path = "/{id}", produces = "application/json")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
    public ResponseEntity<?> getAddressById(@AuthenticationPrincipal User user, @PathVariable Long id){
        return ResponseEntity.ok(service.getAddressById(user, id));
    }

    /** This function is used to update an address specified by its id in the database using the information in ...*/
    @PutMapping(path = "/update/{id}", produces = "application/json", consumes = "application/json")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
    public ResponseEntity<?> updateAddress(@AuthenticationPrincipal User user, @PathVariable Long id, @RequestBody AddressRequest addressR){
        return ResponseEntity.ok(service.updateAddress(user, id, addressR));
    }

    /** This function is used to create a new address in the database using the information in the request body ...*/
    @PostMapping(path = "/create", produces = "application/json", consumes = "application/json")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
    public ResponseEntity<?> create(@AuthenticationPrincipal User user, @RequestBody AddressRequest addressR){
        return ResponseEntity.ok(service.create(user, addressR));
    }

    /** This function is used to delete an address specified by its id from the database by calling the function ...*/
    @DeleteMapping(path = "/delete/{id}", produces = "application/json")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_USER')")
    public ResponseEntity<?> deleteAddress(@AuthenticationPrincipal User user, @PathVariable Long id){
        return ResponseEntity.ok(service.deleteAddress(user, id));
    }
}
```

Figure 12. Class AddressController. (self-made)

Test

Throughout the development of my project, I ran tests to make sure that every method I had just implemented worked properly. If there was a problem, I would try to find an appropriate solution to the problem. Given the fact that this project is a starting point, I did not consider it necessary to implement tests using the JUnit5 library. I preferred to conduct all the tests using the Postman app. This allowed me to have a more visual approach, as well as a faster set-up.

3.5 Presenting the outcome

In this section, I'm going to present my outcome. To do this, I'm not going to set out all the methods I've developed throughout my project, but I'm going to focus on managing the registration and authentication of a user, the creation of an event and the management of event participants. My entire back-end is available on a **GitHub** repository, which I have listed in the appendix to this thesis.

The code is commented out in full to make it easier to use and understand the project. The fact that the main aim of the project is to produce a back-end means that I don't have a front-end. To overcome this problem, I'm going to use the Postman application. During the development of the code, I had the opportunity to use it several times to test the different functions I was developing. Finally, before I start presenting the outcome, I'd like to draw your attention to two points.

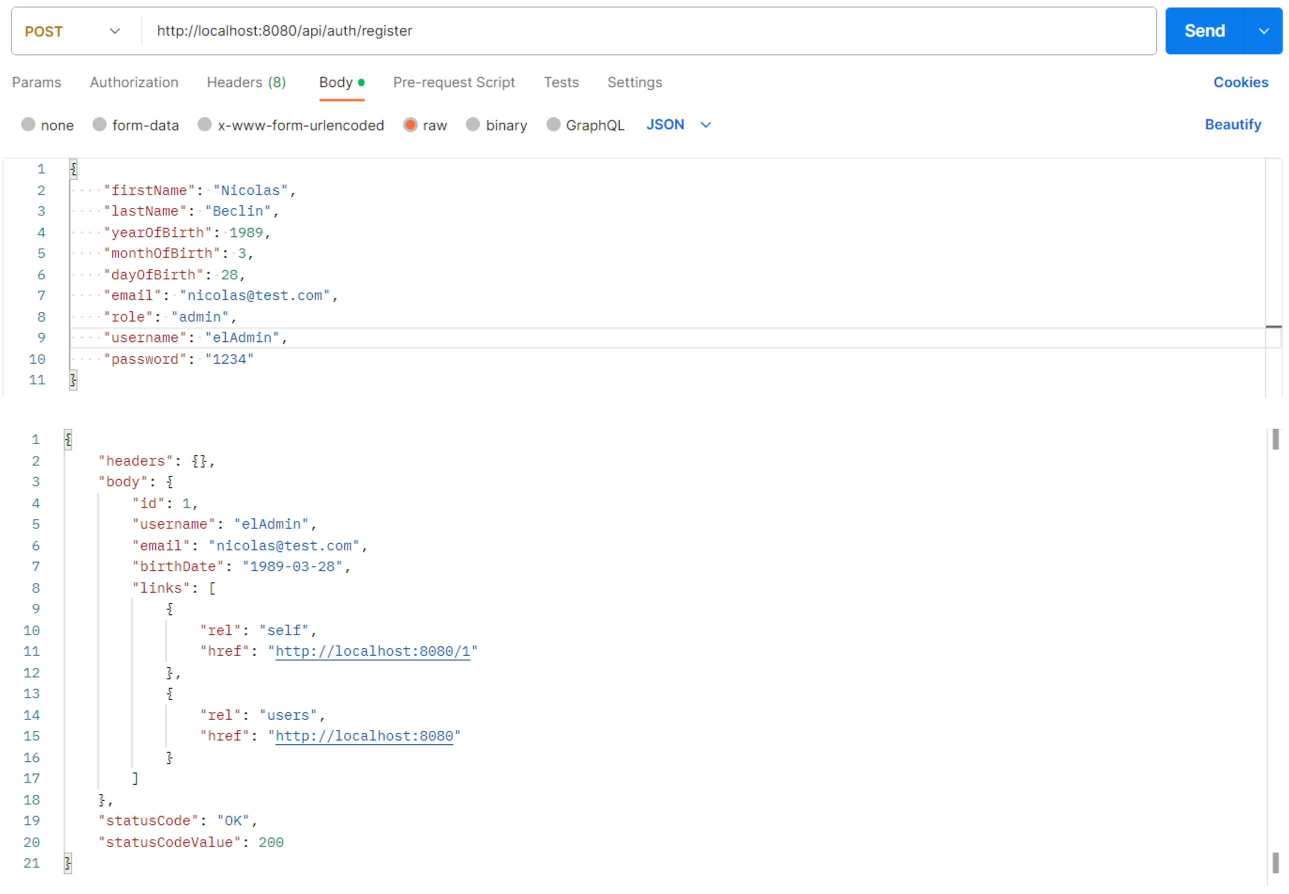
The first point concerns the fact that the database contains virtually no data. Only data linked to user roles are present in the database, namely **ROLE_ADMIN** and **ROLE_USER**. Finally, for the whole presentation, I'll be using an administrator user, to guarantee access to all the endpoints.

The second point concerns access to the project. As I mentioned above, the entire project is available on **GitHub**. The URL is in the appendices of this thesis. After downloading the project, you either need to install the MariaDB DBMS or if you have the **XAMPP** software like I do, you simply need to create an **event_manager** database. Finally, all you have to do is run the application. The project will then be available at the following address: http://localhost:8080/api/.

3.5.1 Creation of a user

When you use the back-end for the first time, you need to register with the application so that you can later log in and access the range of features offered by the back-end. The first thing you need to know is that the highest authority within the back-end is the administrator role. The roles are already present in the database. They are assigned when the user is registered. When registering a user with administrator rights, you must enter admin when filling in the fields. If you leave this field empty, the user will have the user role, which is the lowest. To register, you first need to enter the following information in JSON format in the request body:

- firstName
- lastName
- yearOfBirth
- monthOfBirth
- dayOfBirth
- email
- role
- username
- password



http://localhost:8080/api/auth/register

```
{
    "firstName": "Nicolas",
    "lastName": "Beclin",
    "yearOfBirth": 1989,
    "monthOfBirth": 3,
    "dayOfBirth": 28,
    "email": "nicolas@test.com",
    "role": "admin",
    "username": "elAdmin",
    "password": "1234"
}
```

```
{
    "headers": {},
    "body": {
        "id": 1,
        "username": "elAdmin",
        "email": "nicolas@test.com",
        "birthDate": "1989-03-28",
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:8080/1"
            },
            {
                "rel": "users",
                "href": "http://localhost:8080"
            }
        ]
    },
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

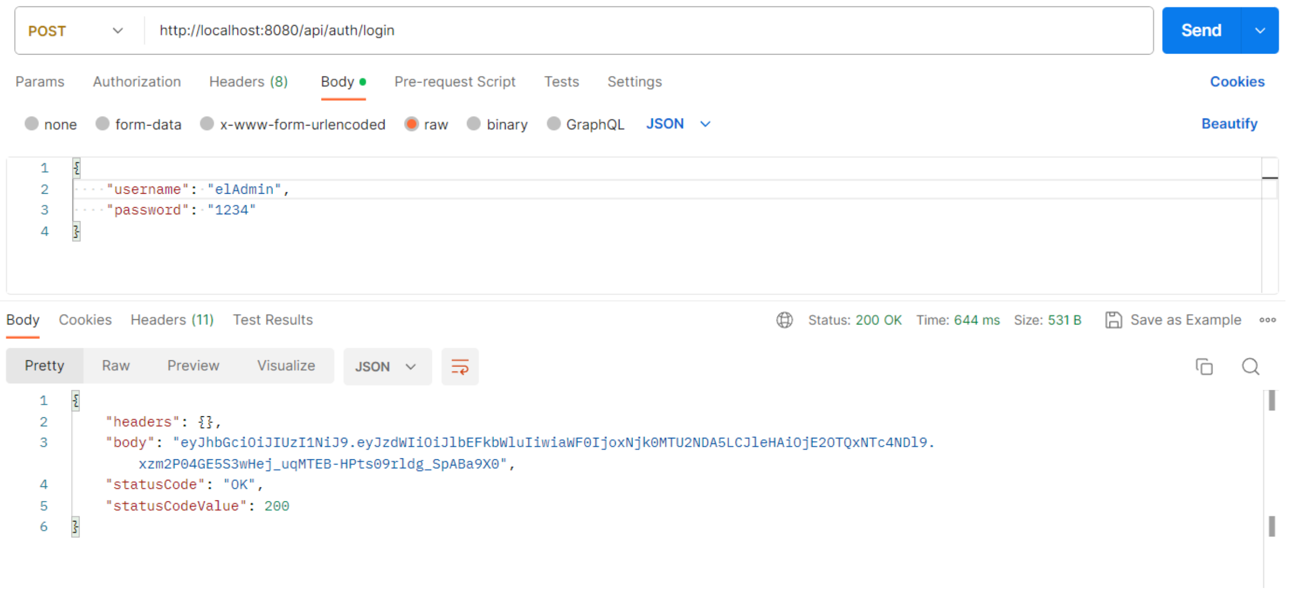
Figure 13. Registration to the application through Postman. (self-made)

You then need to specify that this is a **POST** request and indicate the URL of the request: http://localhost:8080/api/auth/register.

Now that you are registered in the application database, you can connect to the application. The authentication function will first check whether you are registered in the database. If you are, it will generate an access token that you can use to access the various methods. As with the registration functionality, you first need to specify the following information in the request body:

- username
- password

Then you need to specify that it is a **POST** request when making the request to the back-end. To access the functionality you need to specify the URL of the request: http://localhost:8080/api/auth/login.


POST http://localhost:8080/api/auth/login

```
{
    "username": "elAdmin",
    "password": "1234"
}
```

Status: 200 OK Time: 644 ms Size: 531 B

```
{
    "headers": {},
    "body": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlbEFkbWluIiwiaWF0IjoxNjk0MTU2NDA5LCJleHAiOjE2OTQxNTc4NDl9.
        xzm2P04GE5S3wHej_uqMTEB-HPts09rldg_SpABa9X0",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

Figure 14. Authentication to the application through Postman. (self-made)

As you can see from the image above, authentication has generated an access token. You need to copy this token and keep it so that you can access each endpoint in the application, by placing it in the Authorization tab of the request. This will enable you to take advantage of the features offered. If the connection was not possible due to an error when entering the username and password, a **401** error will be returned with an error message: **BAD CREDENTIAL**

3.5.2 Create an event

In order to create an event, I decided to divide the creation process into several distinct stages to make the application more modular and allow greater freedom in the design of the front-end depending on the functionalities at the time of development.

Address

The first step in the process of creating an event is to create an address that can then be assigned to an event. The creation of the address alongside the activity was also motivated by the desire to implement a functionality whereby users of the application would be able to make available a place where an event could take place.

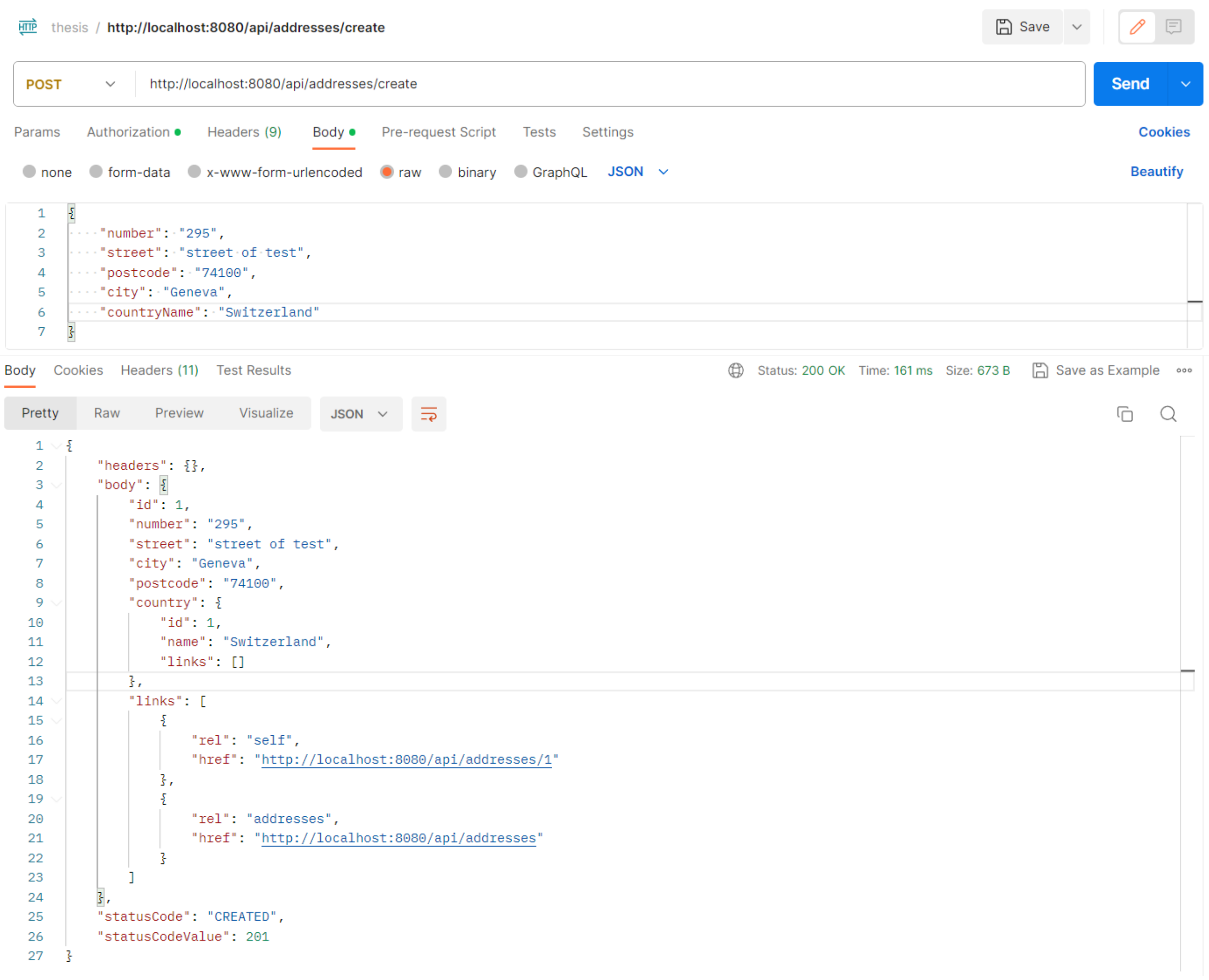
To create an address, the user must provide the following information in the request body:

- number: corresponds to the street number.
- street: street name.
- postcode: corresponds to the postcode.

- city: corresponds to the town where the address is located.
- countryName: corresponds to the country in which the address is located.

Each of the above elements is mandatory. If you do not complete one of the fields, you will be returned an error code **400** with the status **BAD_REQUEST** and an error message corresponding to the error made. However, in the case of a server error, a code **500** will be returned with the status **INTERNAL_SERVER_ERROR**. However, if you create an address that is already in the database, the endpoint will return the address you tried to create with the error code **409** and the status **CONFLICT**. I've decided to return the address of the query to give the user a clearer visual so that they can spot the error more quickly if there is one.

To generate the request, you must first specify that it is a **POST** request. Then, to access the functionality, you need to specify the URL of the request: http://localhost:8080/api/addresses/create.

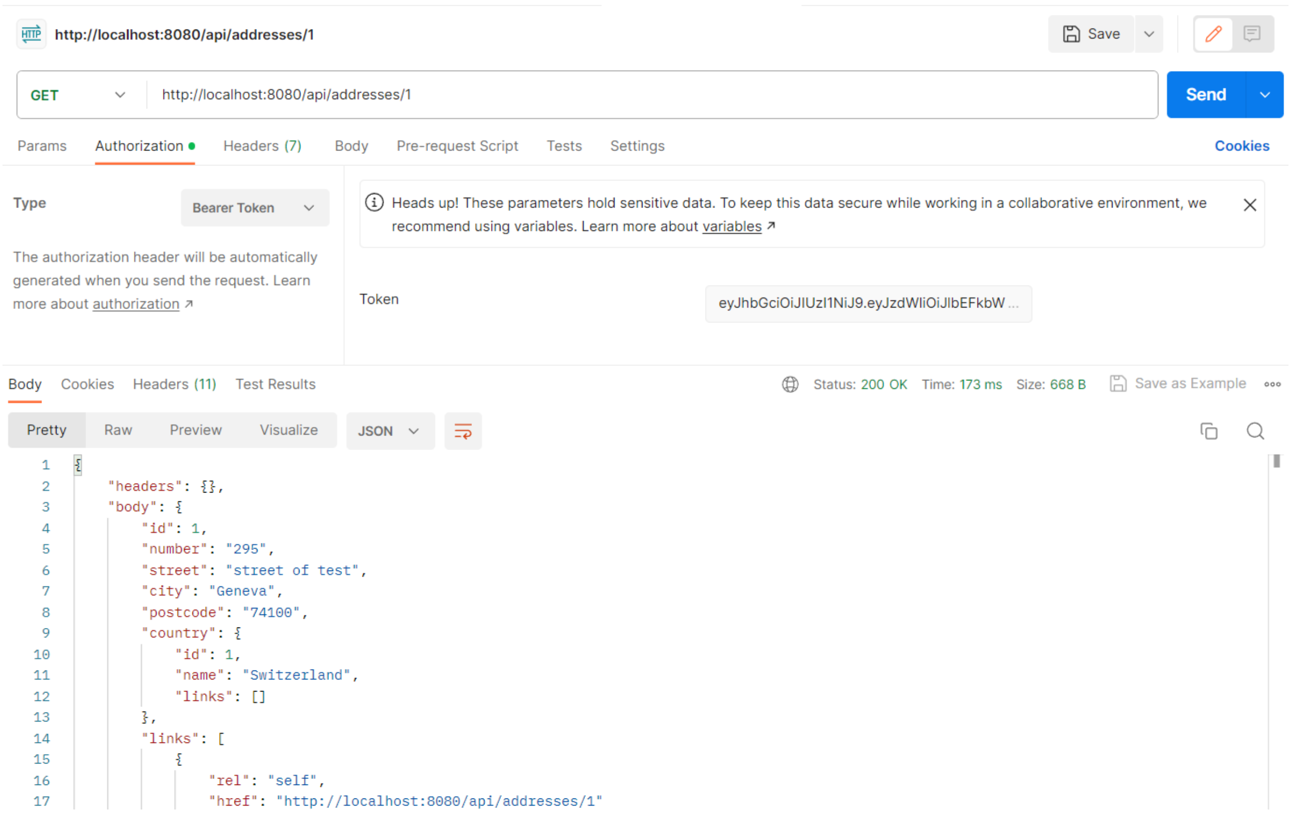


```
{
    "number": "295",
    "street": "street of test",
    "postcode": "74100",
    "city": "Geneva",
    "countryName": "Switzerland"
}
```

```
{
    "headers": {},
    "body": {
        "id": 1,
        "number": "295",
        "street": "street of test",
        "city": "Geneva",
        "postcode": "74100",
        "country": {
            "id": 1,
            "name": "Switzerland",
            "links": []
        },
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:8080/api/addresses/1"
            },
            {
                "rel": "addresses",
                "href": "http://localhost:8080/api/addresses"
            }
        ]
    },
    "statusCode": "CREATED",
    "statusCodeValue": 201
}
```

Figure 15. Create an address. (self-made)

Once the activity has been created, the back-end will return a response containing the address created with the code **201** and the status **CREATED**. You can also see above that the body contains a second element. This second element is the HATEOAS links which redirect to the address itself by making a **GET** request for the first and the second link will return a list of existing addresses in the database, also with a **GET** request.



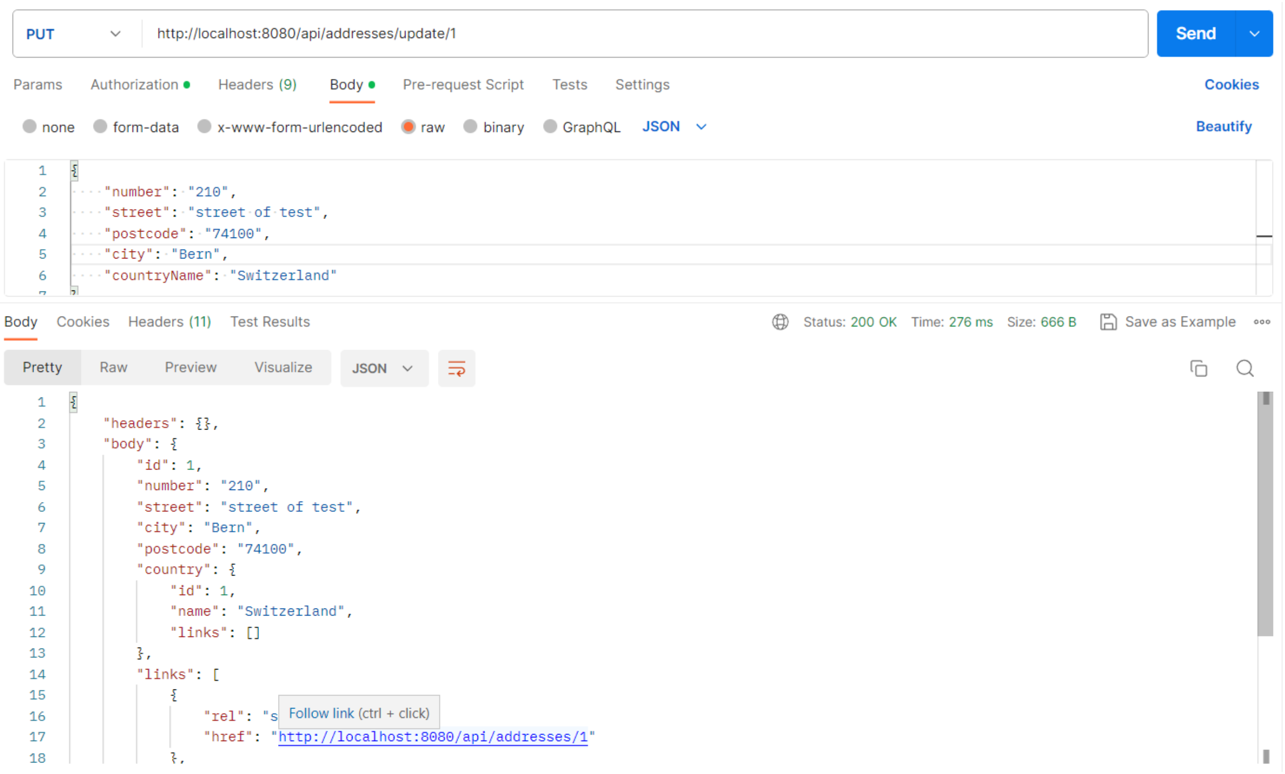
The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

```
{
    "headers": {},
    "body": {
        "id": 1,
        "number": "295",
        "street": "street of test",
        "city": "Geneva",
        "postcode": "74100",
        "country": {
            "id": 1,
            "name": "Switzerland",
            "links": []
        },
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:8080/api/addresses/1"
```

Figure 16. Display an address by using the link HATEOAS (self-made)

To enable these links to work, I had to implement an endpoint that would return an address based on its identifier. This endpoint can be reached by making the following **GET** request: http://localhost:8080/api/addresses/{id}, the id corresponding to the address identifier. I also had to implement an endpoint to return a list of all the addresses available in the database. You can access this endpoint by making the following **GET** request: http://localhost:8080/api/addresses.



PUT http://localhost:8080/api/addresses/update/1

```
{
    "number": "210",
    "street": "street of test",
    "postcode": "74100",
    "city": "Bern",
    "countryName": "Switzerland"
```

Status: 200 OK Time: 276 ms Size: 666 B

```
{
    "headers": {},
    "body": {
        "id": 1,
        "number": "210",
        "street": "street of test",
        "city": "Bern",
        "postcode": "74100",
        "country": {
            "id": 1,
            "name": "Switzerland",
            "links": []
        },
        "links": [
            {
                "rel": "s
                "href": "http://localhost:8080/api/addresses/1"
            },
```

Figure 17. Update of an address with the id. (self-made)

As when an address is created, the endpoint targeted by the update will return the updated address along with code **200** and the **OK** status**.**

Event

The second part of creating an event, as I see it, consists of creating the event itself and adding an address. When modelling the Entity JPA Activity class, which is the class affiliated to an event, I implemented it in such a way as to provide several constructors, so as to be able to create events with no age limit or with, with address and without, with address and with age limit and finally without address and without age limit. I've implemented these different variants in order to directly provide various choices to the future person wishing to use or modify my back-end to implement the creation of an event as they wish. Personally, I've always preferred to see code as a Lego to be built. You don't build it from a single base but by assembling several parts created separately to form a single entity.

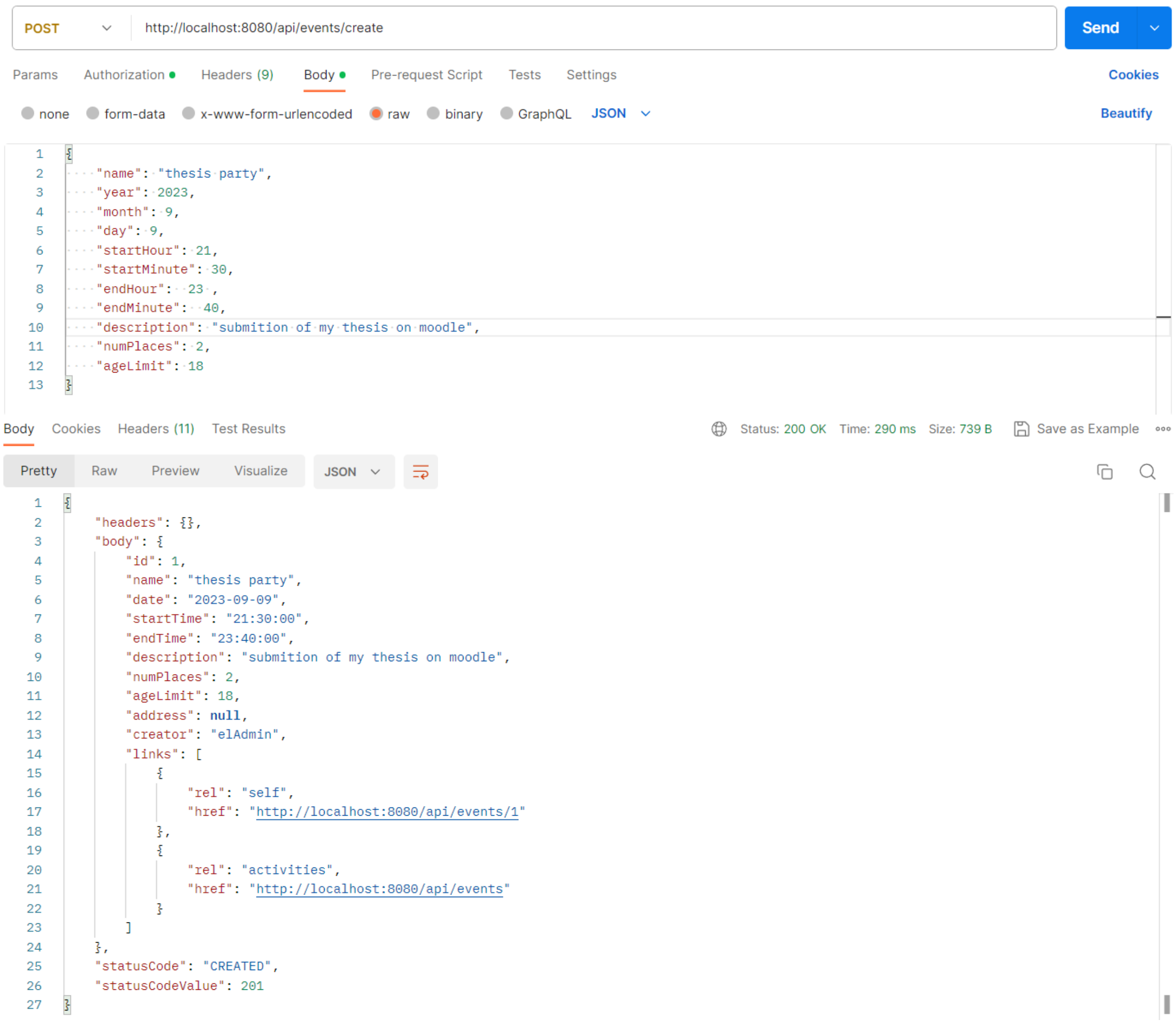
To start setting up an event, you first create the event skeleton, to which you then add an address, tags, participants and notifications. Creating the skeleton will require the following information:

- name: corresponds to the name of the event.
- year: corresponds to the year in which the event takes place.

- month: corresponds to the month in which the event is taking place.
- day: corresponds to the day of the event.
- starthour: corresponds to the time at which the event begins.
- startMinute: corresponds to the minute at which the event begins.
- endHour: corresponds to the hour at which the event ends.
- endMinute: corresponds to the minute at which the event ends.
- description: corresponds to the description of the event.
- numPlaces: corresponds to the number of places available for the event.
- ageLimit: corresponds to the age limit for attending an event the event.

As with the creation of the address, each of the above elements is mandatory. If you do not complete one of the fields, you will be returned an error code **400** with the status **BAD_REQUEST** and an error message corresponding to the error made. However, in the case of a server error, a code **500** will be returned with the status **INTERNAL_SERVER_ERROR**. If you do not wish to impose an age limit on your event, you must indicate that the event is equal to 0. This will allow you to use the correct constructor when generating the response.

To generate the request, you need to place the requested information inside the request body in JSON format. Once the information has been supplied, you can create a **POST** request via the following URL: http://localhost:8080/api/events/create.



http://localhost:8080/api/events/create

```
{
    "name": "thesis party",
    "year": 2023,
    "month": 9,
    "day": 9,
    "startHour": 21,
    "startMinute": 30,
    "endHour":  23 ,
    "endMinute":  40,
    "description": "submition of my thesis on moodle",
    "numPlaces": 2,
    "ageLimit": 18
}
```

Status: 200 OK Time: 290 ms Size: 739 B

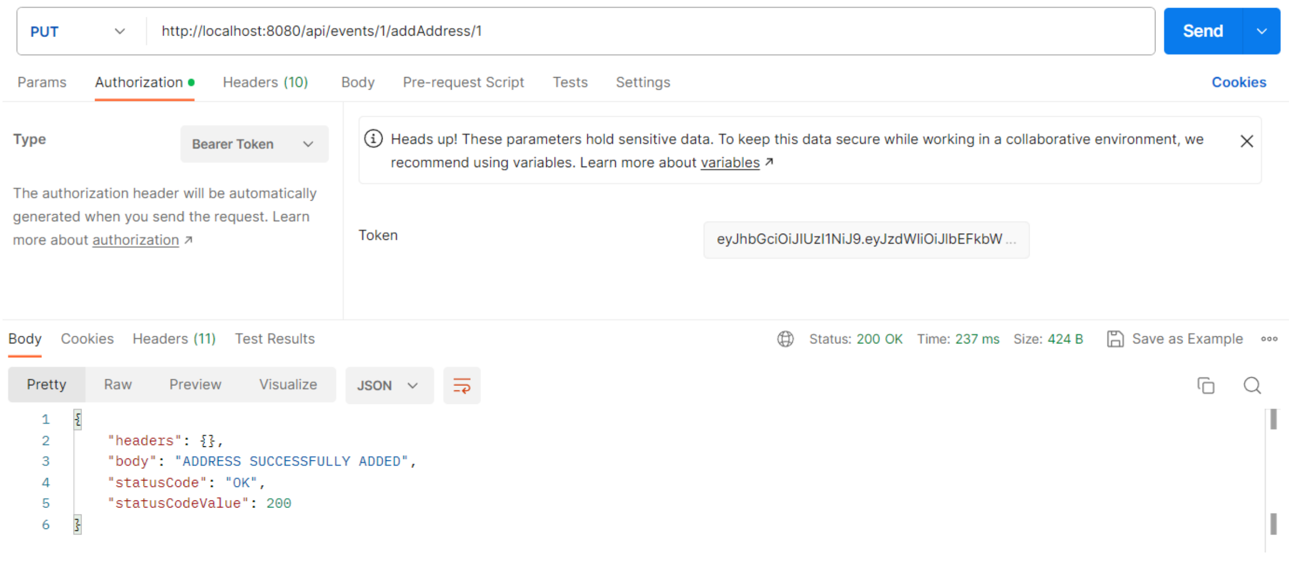
```
{
    "headers": {},
    "body": {
        "id": 1,
        "name": "thesis party",
        "date": "2023-09-09",
        "startTime": "21:30:00",
        "endTime": "23:40:00",
        "description": "submition of my thesis on moodle",
        "numPlaces": 2,
        "ageLimit": 18,
        "address": null,
        "creator": "elAdmin",
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:8080/api/events/1"
            },
            {
                "rel": "activities",
                "href": "http://localhost:8080/api/events"
            }
        ]
    },
    "statusCode": "CREATED",
    "statusCodeValue": 201
}
```

Figure 18. Creation of an event (self-made)

After creating the event, the endpoint will return the event created with the HATEOAS links, along with code **201** and the **CREATE** status.

Now that the event has been created, you can add an address to it. However, to avoid elements running in the same places or duplicates, I've implemented the add method to avoid creating a duplicate activity. It will check whether there is an event with the same name, taking place on the same date and with the same start and end times for the desired address. If this is the case, the event to which the address is to be added will be deleted, to avoid the user generating continuous requests and overloading the server. The endpoint will return the message **ACTIVITY DELETED BECAUSE ALREADY EXISTENT**, with the error code **409** and the status **CONFLICT.**

To add an address to an event, you'll need the event identifier and the identifier of the address you want to add to your event. You can then create a **PUT** request via the following URL: http://localhost:8080/api/events/{id_event}/addAddress/{id_address}.



The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Token eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlbEFkbW ...

```
{
    "headers": {},
    "body": "ADDRESS SUCCESSFULLY ADDED",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

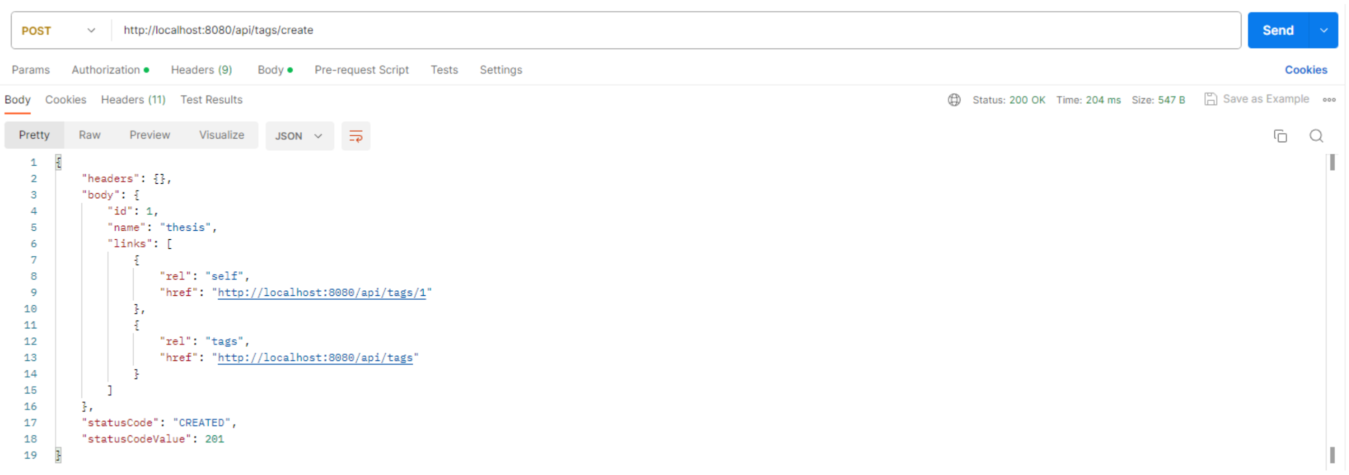
Figure 19. Add an address to an event (self-made)

Following the addition of an address to an event, the endpoint will return the message **ADDRESS SUCCFULLY ADDED**, all accompanied by code **200** and the status **OK**.

TAG

Tags are words used to describe the event. Using the same logic as for addresses, tags must first be created and then added to the event. To create a tag, you need to provide the name of the tag you want to create in the request body. If you execute the request without indicating the name of the tag, or if the tag already exists in the database, the endpoint will return a **400 BAD_REQUEST** error, accompanied by a message describing the error.

To create a tag, you need to create a **POST** request via the following URL: http://localhost:8080/api/tags/create.

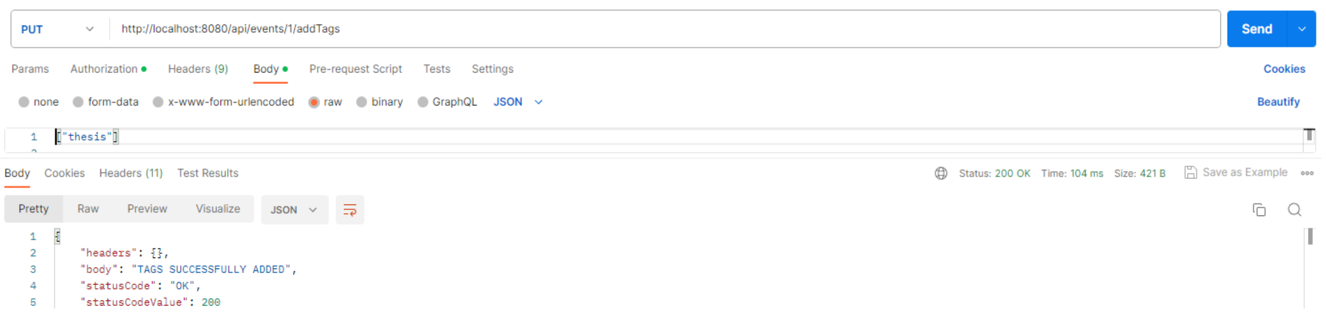


```
{
    "headers": {},
    "body": {
        "id": 1,
        "name": "thesis",
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:8080/api/tags/1"
            },
            {
                "rel": "tags",
                "href": "http://localhost:8080/api/tags"
            }
        ]
    },
    "statusCode": "CREATED",
    "statusCodeValue": 201
}
```

Figure 20. Creation of a tag (self-made)

Once the tag has been created, the endpoint will return the created tag along with code **201** and the status **CREATED**.

Finally, having created the tag, you can now add it to the event we created in the event section. To add it, you need to create a PUT request to the following URL: http://localhost:8080/api/events/{id_event}/addTags.
This request will allow you to supply one or more tags, by sending a list of tags via the body. When tags are received, the method will check whether the tags exist in the database. If the tags do not exist, the endpoint will return a 400 error and the status **BAD_REQUEST**. If they do, it will return the message **TAGS SUCCFULLY ADDED**, along with the code **200** and the status **OK**.



```
["thesis"]
```

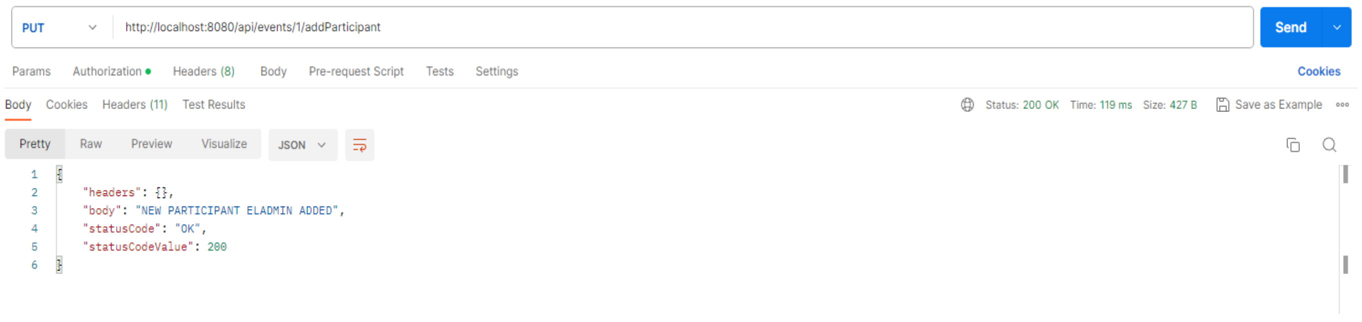
```
{
    "headers": {},
    "body": "TAGS SUCCESSFULLY ADDED",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

Figure 21. Add a tag to an event. (self-made)

3.5.3 Participant management

The back-end produced in this thesis provides three endpoints for managing participants in an event. The first endpoint consists of allowing a user to participate in an event. This method will first check whether the user is already present on the event's participant list or waiting list. If the user is present in one of the two lists, the endpoint will return an error message indicating that the user is already present in one of the two lists. This error message will be accompanied by the error code **409** and the status **CONFLICT**. Otherwise, the user's request will go through a final checkpoint, which is the age limit for taking part in the event, if there is one. If the user is too young, the endpoint will return an error message indicating that the user is too young, as well as a code **403** and the status **FORBIDDEN**.

After passing these checks, the user can request to take part in the event. If there is space available, the user will be added directly to the list of participants; if not, he or she will automatically be put on the waiting list. Users are added according to the user who makes the request to the endpoint. To register for an event, you need to create a **POST** request by indicating the event identifier in the following URL: http://localhost:8080/api/events/{eventId}/addParticipant.

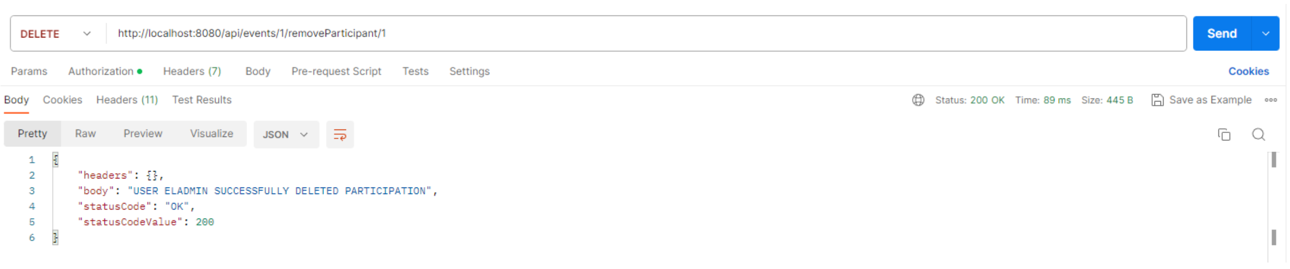


```
{
    "headers": {},
    "body": "NEW PARTICIPANT ELADMIN ADDED",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

Figure 22. Add a participant to an event. (self-made)

The second endpoint is used to remove a participant, specified in the request URL, from the event. Only the event creator can perform this action. Before removing a user from the list of participants, the presence of the user in the list of participants is checked. If the specified user is not a participant, the endpoint will return an error message indicating that the user is not a participant, accompanied by the error code **404** and the status **NOT_FOUND**. Otherwise, the user will be removed from the event. The endpoint will return a confirmation message with code **200** and status **OK**.

If you want to remove a participant from an event you have created, you must create a **DELETE** request by indicating in the following URL the identifier of the participant and the identifier of the event you want to remove him from: http://localhost:8080/api/events/{eventId}/removeParticipant/{participantId}.



DELETE http://localhost:8080/api/events/1/removeParticipant/1 Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (11) Test Results Status: 200 OK Time: 89 ms Size: 445 B Save as Example

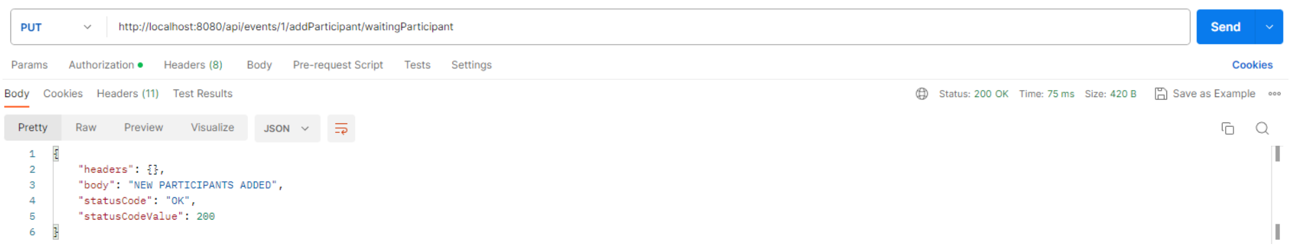
Pretty Raw Preview Visualize JSON

```
{
    "headers": {},
    "body": "USER ELADMIN SUCCESSFULLY DELETED PARTICIPATION",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

Figure 23. Remove participant from an event. (self-made)

Finally, the last endpoint consists of adding a person from an event's waiting list to the list of participants. This action can only be taken by the event creator. However, the new participant is added at random to ensure equal opportunities for all users on the waiting list.

First of all, the available places are checked. If no places are available, a message is returned by the endpoint saying **NO AVAILABLE PLACES** accompanied by the code **409** and the status **CONFLICT**. In the opposite case, a confirmation message saying **NEW PARTICIPANT ADDED** is returned with the same code and status. To be able to add a participant from the waiting list of an event you have created, you must create a **PUT** request by indicating the event identifier in the following URL: http://localhost:8080/api/events/{eventId}/addParticipant/waitingParticipant.



PUT http://localhost:8080/api/events/1/addParticipant/waitingParticipant Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (11) Test Results Status: 200 OK Time: 75 ms Size: 420 B Save as Example

Pretty Raw Preview Visualize JSON

```
{
    "headers": {},
    "body": "NEW PARTICIPANTS ADDED",
    "statusCode": "OK",
    "statusCodeValue": 200
}
```

Figure 24. Add participant from the waiting list. (self-made)

4 Discussion

When I started my thesis in January 2023, I had a single desire and a single objective. To create a generic, open-source application that would enable any Swiss university campus to offer its student community, free of charge, a platform through which students could register, create and find out about the various events organised around them. However, setting up such an application was too much work for a Bachelor's degree and a single developer.

I therefore had to modify my project to adapt it to the requirements of a subject worthy of a Bachelor's dissertation. I therefore concentrated on developing a first version of a generic open-source back-end for event management, which would be the starting point for the project I mentioned at the beginning of this section. As stated in section 1.1 Objectives, the aim of this thesis is to provide basic functionality for event management. The set of functionalities will reflect the manipulation of data according to business logic, stored on a MariaDB database. These same functionalities will be exposed through a REST API also implemented in this project. The development of this thesis also has a personal development objective in terms of learning new working methods, using new tools and deepening the concepts learned throughout my training.

I write this section with a sense of overall achievement in terms of writing the various parts of the thesis, developing the code and the working methodology. Throughout the writing of the thesis, every point I wanted to cover has been dealt with in such a way as to provide clear explanations with enough depth to tackle the more complex and technical topics, such as the explanation of Spring modules and projects. Writing the Spring part was complicated. This is mainly due to the fact that the Spring Framework provides documentation that is rich in information, but not always easy to understand for someone like me who is new to using this tool. What's more, the Spring documentation contains a few out-of-date articles and doesn't offer a clear redirect to up-to-date documentation. Despite these various obstacles, I was able to rely on the various explanations provided by the Framework's large community via internet blogs and explanatory videos.

The development of the project as a whole enabled me to move from theory to practice in creating and implementing a viable working method for completing the project. The time spent reflecting on the choice of an agile or waterfall methodology opened my eyes to the realities in the field. After a great deal of research, which enabled me to learn more about the work methodology. I realised that each approach has its advantages and disadvantages, but that by combining them, we could achieve a work methodology that suited my mentality of wanting to complete one task and check it before moving on to another. However, I think the methodology I used for this project should have been a little more sophisticated. By setting deadlines for the development of tasks and allocating them in a better way. But what I will change with regard to this work for my future projects is to start the project by putting the methodology in place as a first step and not to do it in the middle.

The development of my back-end was dictated by the 3 success criteria that I indicated when writing point 3.1 Starting point of the empirical part. I managed to provide the basic functionality needed to manage an event via my back-end, without encountering many problems. The only major problem I encountered was the implementation of the JSON WEB TOKEN tool. Without this tool, the registration and authentication functions could not have been developed in a RESTful way. The problem was that I found myself confronted with a large number of sources offering various implementations. The problem was that these sources were often incorrect and extremely poorly explained. After a lot of research, I was finally able to find a correct implementation provided by the Spring Framework community, which at the same time allowed me to deepen my knowledge of a new tool and learn how to set up the security of a Spring project in a RESTful way. Following the resolution of this problem, I was able to implement the CRUD functionalities for the Address, Activity, Opinion and Tag classes, without which the basic functionalities could not have been implemented. Throughout the development of the application, I used the Postman application to test the various functions I had just programmed. However, having now finished this project, I realise that implementing unit testing would have enabled me to carry out the tests more quickly and probably in greater detail.

This thesis has enabled me to take the first step towards my goal of creating a generic, open-source event management application. My next objective with regard to this project is to continue to improve the functionalities already present and also to add new ones. Such as adding a feature that will allow users to make locations available for events. I'm also thinking of implementing a payment method to make it possible to create paid events. Not forgetting to add unit tests for old and new features.

However, I think that the work and development carried out during this thesis has enabled the project to reach a sufficiently mature and complete level to be able to create a complete application by creating a front-end. The creation of a front-end will enable entities such as university campuses to see the scope of the project and even contribute to its development so that it can one day be offered to the general public.

Finally, writing and programming this bachelor's thesis has enabled me to acquire and improve a great deal of knowledge in the field of web development. I was able to explore the world of Spring Framework in more detail, focusing on its construction and the functionalities offered through these projects. I also honed my knowledge of the ins and outs of REST APIs, understanding how to design web services that respect the REST philosophy.

Using MariaDB allowed me to immerse myself in the world of relational databases. It gave me the opportunity to deepen my knowledge of the world of data management and its tools and put it into practice.

In conclusion, this work, which symbolised the twilight of my training, gave me a second, more enriching and self-taught approach to web development, with an understanding of the fundamental principles that make it possible to develop web services and applications. Aware that I had only explored part of it, it motivated me to continue to perfect my knowledge and take on new challenges in this vast field that is web development.

Sources

Adservio. 11 October 2022. Spring Modules. adservio. URL: https://www.adservio.fr/post/spring-modules. Accessed: 12 June 2023.

Taylor, Allen G. 2017. SQL For Dummies. 8th ed. First.

Amigoscode. 3 January 2023. Spring Boot 3 + Spring Security 6 – JWT Authentication and Authorisation [NEW] [2023]. Online video. URL: https://www.youtube.com/watch?v=KxqlJblhzfI. Accessed: 26 June 2023.

AWS. s.a. What's the Difference Between MariaDB and MySQL? AWS. URL: https://aws.amazon.com/compare/the-difference-between-mariadb-vs-mysql/?nc1=h_ls. Accessed: 29 June 2023.

Azure. s.a. What is a relational database? Azure. URL: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-a-relational-database/#whatis. Accessed: 29 June 2023.

Baeldung. 13 November 2022. Guide to the Spring BeanFactory. Baeldung. URL: https://www.baeldung.com/spring-beanfactory. Accessed: 13 June 2023.

Baeldung. 24 June 2022. Introduction to the JSON Binding API (JSR 367) in Java. Baeldung. URL: https://www.baeldung.com/java-json-binding-api. Accessed: 6 June 2023.

Baeldung. 22 December 2022. The DTO Pattern (Data Transfer Object). Baeldung. URL: https://www.baeldung.com/java-dto-pattern. Accessed: 2 September 2023.

Clark, J. s.a. Top 10 Backend Frameworks In 2023. Back4app. URL: https://blog.back4app.com/backend-frameworks/. Accessed: 13 April 2023.

Codecademy. 23 September 2021. What is a Framework? Codecademy. URL: https://www.codecademy.com/resources/blog/what-is-a-framework/#:~:text=A%20framework%20is%20a%20structure,to%20different%20types%20of%20tasks. Accessed: 15 April 2023.

Coursera. 17 Mai 2023. Front End vs. Back End: Learning Skills and Tools. Coursera. URL: https://www.coursera.org/articles/front-end-vs-back-end. Accessed: 10 April 2023.

Edwards, R. 26 March 2020. An Introduction to Spring Projects. vmware Open Source Blog. URL: https://blogs.vmware.com/opensource/2020/03/26/spring-open-source/#:~:text=From%20configuration%20and%20security%20to,of%20developers%20around%20the%20world. Accessed: 20 June 2023.

Fincher, J. and Desmond, K. s.a. The 11 Best Backend Frameworks – 2023. CODINGNOMADS. URL: https://codingnomads.co/blog/best-backend-frameworks/. Accessed: 31 August 2023.

Fol, P. 26 October 2021. Spring vs. the World: Comparing Spring Boot Alternatives. JRebel. URL: https://www.jrebel.com/blog/spring-boot-alternatives. Accessed: 31 August 2023.

Grandjean, F. 03 July 2019. Frontend, backend, qu'est-ce que ça veut dire ? Wild Code School. URL: https://www.wildcodeschool.com/fr-FR/blog/differences-backend-frontend-developpement-web. Accessed: 10 April 2023.

Gupta, L. 11 December 2021. HTTP Methods. REST API Tutorial. URL: https://restfulapi.net/http-methods/. Accessed: 17 June 2023.

IBM. 12 December 2023. Concurrency Utilities for Java EE. IBM. URL: https://www.ibm.com/docs/en/was/9.0.5?topic=concurrency-utilities-java-ee. Accessed: 6 June 2023.

IBM. 2 May 2023. Java Persistence API (JPA). IBM. URL: https://www.ibm.com/docs/fr/was-liberty/base?topic=overview-java-persistence-api-jpa. Accessed: 8 June 2023.¨

IBM. 13 December 2022. Bean Validation. IBM. URL: https://www.ibm.com/docs/en/was/9.0.5?topic=validation-bean. Accessed: 8 June 2023.

IBM. s.a. What is Java Spring Boot? IBM. URL: https://www.ibm.com/topics/java-spring-boot. Accessed: 20 June 2023.

IBM. s.a. What is a relational database? IBM. URL: https://www.ibm.com/topics/relational-databases. Accessed: 29 June 2023.

IBM. s.a. What is a REST API? IBM. URL: https://www.ibm.com/topics/rest-apis. Accessed: 29 June 2023.

Ideematic. 02 March 2023. Les avantages de l'utilisation de frameworks pour le développement d'applications web et mobiles. ideematic. URL: https://www.ideematic.com/actualites/2023/03/les-avantages-de-lutilisation-de-frameworks-pour-le-developpement-dapplications-web-et-mobiles/#:~:text=L'utilisation%20des%20frameworks%20pour,c%C3%B4t%C3%A9%20front%20que%20c%C3%B4t%C3%A9%20serveur. Accessed: 14 April 2023.

IONOS. 31 Mars 2022. Backend and frontend: What do we mean by this? Digital Guide IONOS. URL: https://www.ionos.com/digitalguide/websites/website-creation/backend-frontend/. Accessed: 10 April 2023.

IONOS. 13 June 2023. Databases. Digital Guide IONOS. URL: https://www.ionos.com/digital-guide/hosting/technical-matters/databases/. Accessed: 30 June 2023.

IONOS. 12 July 2023. HATEOAS: information on the rest constraint. Digital Guide IONOS. URL: https://www.ionos.com/digitalguide/websites/web-development/hateoas-information-on-the-rest-constraint/. Accessed: 1 September 2023.

IONOS. 14 February 2018. HATEOAS: what does the acronym mean? Digital Guide IONOS. URL: https://www.ionos.com/digitalguide/websites/web-development/hateoas-information-on-the-rest-constraint/. Accessed: 29 June 2023.

IONOS. 18 October 2022. MariaDB vs MySQL. Digital Guide IONOS. URL: https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/mariadb-vs-mysql/. Accessed: 29 June 2023.

IONOS. 31 January 2023. How does the client server model work? Digital Guide IONOS. URL: https://www.ionos.com/digitalguide/server/know-how/client-server-model/. Accessed: 14 April 2023.

Javatpoint. s.a. Java EE. URL: https://www.javatpoint.com/java-ee. Accessed: 24 April 2023.

Juvénal, JVC. s.a. Programmation en Spring Java : le guide complet. Data Transition Numérique. URL: https://www.data-transitionnumerique.com/spring-java/. Accessed: 24 April 2023.

Mahmoud, H. November 2004. Getting Started with Java Message Service (JMS). Oracle. URL: https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html. Accessed: 8 June 2023.

Mariadb s.a. MariaDB in brief. Mariadb. URL: https://mariadb.org/en/. Accessed: 29 June 2023.

OWASP s.a. REST Security Cheat Sheet. OWASP Cheat Sheet Series. OWASP. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html. Accessed: 29 June 2023.

Pratt, M. 5 October 2022. Difference Between REST and HTTP. Baeldung. URL: https://www.baeldung.com/cs/rest-vs-http. Accessed: 1 September 2023.

Saxena, S. 3 May 2023. Data Models in DBMS. GeeksforGeeks. URL: https://www.geeksforgeeks.org/data-models-in-dbms/. Accessed: 4 September 2023.

Schaffer, E. 8 October 2021. Front-end vs back-end development: What's the difference? Educative. URL: https://www.educative.io/blog/frontend-vs-backend-development. Accessed: 10 April 2023.

Shahzeb, A. 4 November 2022. MariaDB vs MySQL: A Detailed Comparison. Cloudways. URL: https://www.cloudways.com/blog/mariadb-vs-mysql/. Accessed: 29 June 2023.

Simmons, L. 14 March 2023. Front-End vs. Back-End: What's the Difference? Computer Science.org. Spring. URL: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/. Accessed: 10 April 2023.

Spring. 11 May 2023. Spring HATEOAS – Documentation de référence. Spring. URL: https://docs.spring.io/spring-hateoas/docs/current/reference/html/. Accessed: 29 June 2023

Spring. s.a. 1.2 Modules. Doc.spring. Spring. URL: https://docs.spring.io/spring-framework/docs/3.0.0.M4/reference/html/ch01s02.html. Accessed: 12 June 2023.

Spring. 13 June 2019. Overview of the Srping Framework. Spring. URL: https://docs.spring.io/spring-framework/docs/3.0.0.M4/reference/html/ch01s02.html. Accessed: 12 June 2023.

Spring. 2023. Projects. Spring. URL: https://spring.io/projects. Accessed: 20 June 2023.

TIBCO. s.a. What is a Logical Data Model? Tibco. URL: https://www.tibco.com/reference-center/what-is-a-logical-data-model. Accessed: 4 September 2023.

VMware Tanzu. 16 October 2019. Taylor Wichsell and Tom Gianos at SpringOne Platform 2019. Online video. URL: https://www.youtube.com/watch?v=mln3_o6qlBo. Accessed: 23 June 2023.

Zahra, F. 6 June 2023. MariaDB vs MySQL – Principales différences, avantages et inconvénients, et plus encore. HOSTFINGER TUTORIELS. URL : https://www.hostinger.fr/tutoriels/mariadb-vs-mysql. Accessed: 29 June 2023.

Appendices

Appendix 1. GitHub repository

Application repository URL: https://github.com/Nicolasbcrrl/event_manager_project.git