TECHNISCHE UNIVERSITÄT SOFIA

Fakultät für deutsche Ingenieur- und Betriebswirtschaftsausbildung

FDIBA

Master Thesis

On the topic:

Social Network for people sharing similar outdoors hobbies.

Student: **Lyubomir Ivanov Hristozov**

Faculty Number: **201315014**

Subject area: **Informatics**

Sofia

2018

Table of Contents

List of Abbreviations

Acronym	Meaning
IT	Information Technology
UI	User Interface
SDK	Software Development Kit
API	Application Programming Interface
GUI	Graphic User Interface
XML	Extensible Markup Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheet
IDE	Integrated Development Environment
JSP	JavaServer Pages
JS	JavaScript
DB	Database
JVM	Java Virtual Machine
MVC	Model View Controller
POJO	Plain Old Java Object
RMI	Remote Method Invocation
EJB	Enterprise Java Beans
URI	Uniform Resource Identifier
AJAX	Asynchronous JavaScript and XML
ORM	Object Relational Mapping
JPA	Java Persistence API
SQL	Structured Query Language
HQL	Hibernate Query Language
URL	Uniform Resource Locator
SPA	Single page application
DOM	Document Object Model

Introduction

All around the world people have various outdoors hobbies that require more than one person to practice or simply it is more fun and entertaining if there is a group of people with which you can do these activities with. More often than not people find themselves lacking friends and family members that like the same activities as them and this hinders them from practicing their favorite hobbies. The solution for this problem comes in the form of social network sites, sites where people sharing common point of interest can interact with each other. With these social networks people can easily find new groups of people and friends with which they can practice their favorite activities. But with these websites come a few challenges. First is the security of the data of each user. While we live in a fast growing digital world, the cybercrime grows as well. The next challenge is the popularity of the web platform – if no one uses it, it is not very attractive to the users, but if a lot of people use it, its efficiency is improved. And thirds but not last is the variety of the social networks – currently there are not so many platforms of this kind which limits the user's choices.

Considering all of the above, I have chosen as a main objective of this diploma thesis the design and implementation of social network for people sharing similar outdoors hobbies. For the purpose of completing the objective of the thesis I have identified a few main processes. The first ordeal will be the research and design of a database that will contain all of the necessary information that will be used throughout the web system. The second will be to implement the back-end of the application, meaning implementing the database and the business logic of the site. And last but not least will be the implementation of the GUI for the social network site.

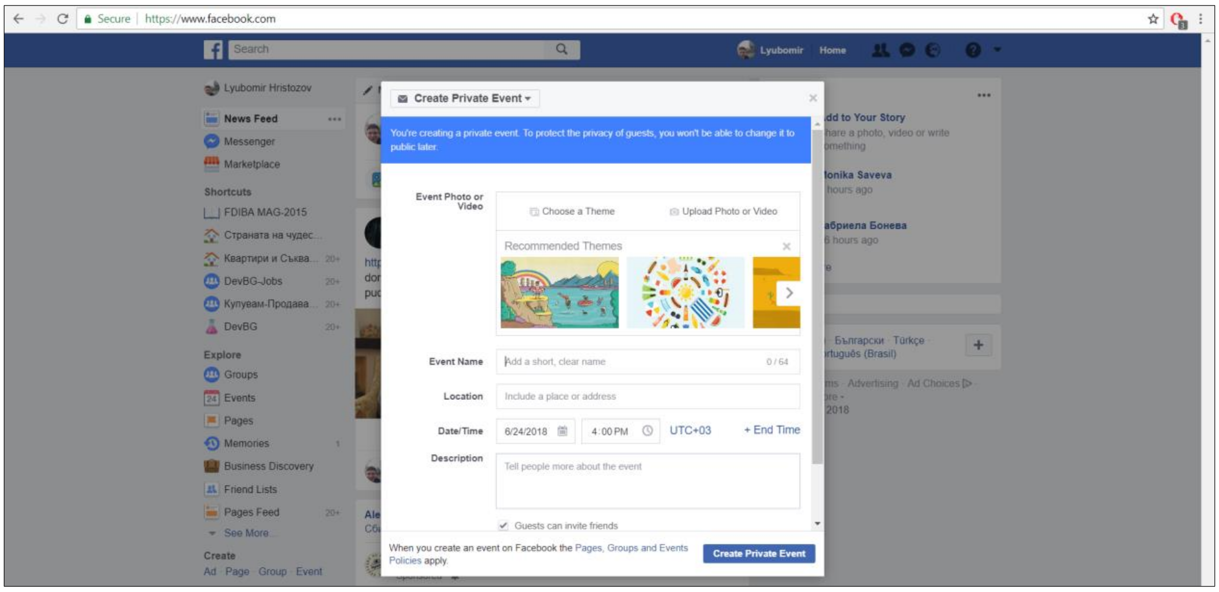
In the next few chapters of the thesis a detailed overview of the web application will be given. First a comparison with other web sites with the same functionality will be made. After that the technologies that have been used to create the application will be presented and a short explanation of them will be given. An explanation of the architecture of the application and how the front-end and back-end work with each other will also will be given. At the end of the thesis a user guide that shows the complete social network will be presented.

1. Problem analysis

1.1. Overview of already existing solutions

The concept of social networks that emphasize on outdoors hobbies is not very common, but there are similar sites that offer the functionality for creating various events. In this section we will try to examine and compare different solutions to the solution of this thesis.

1.1.1. Facebook



Create Private Event
You're creating a private event. To protect the privacy of guests, you won't be able to change it to public later.
Event Photo or Video
Choose a Theme
Upload Photo or Video
Recommended Themes
Event Name
Add a short, clear name
0 / 64
Location
Include a place or address
Date/Time
6/24/2018
4:00 PM
UTC+03
+ End Time
Description
Tell people more about the event
Guests can invite friends
When you create an event on Facebook the Pages, Groups and Events Policies apply.
Create Private Event

Figure 1: Facebook creation of events

As a well know and highly used social network Facebook without a doubt has an event creation functionality. The facebook event creation feature in itself contains the following:

- Clean and intuitive design
- Ability to add theme/picture of the event
- Ability to add information about the event such as name, description, date and location
- Ability to switch event type – private or public

The facebook implementation of the feature is simple, intuitive and with a clean UI and it is sufficient in almost all of the cases when a user wants to create an event.

In the web application, that is the objective of this thesis work, are incorporated some of the functionality from the facebook event creation feature. But some of them are brought a step further such as the location of the event. In the web application that is the object of this thesis the location can be selected directly form a map. Another enhancement is the ability to use predefined destinations. The following table (table 1) compares the facebook feature and the feature of the web application that is the object of the master thesis.

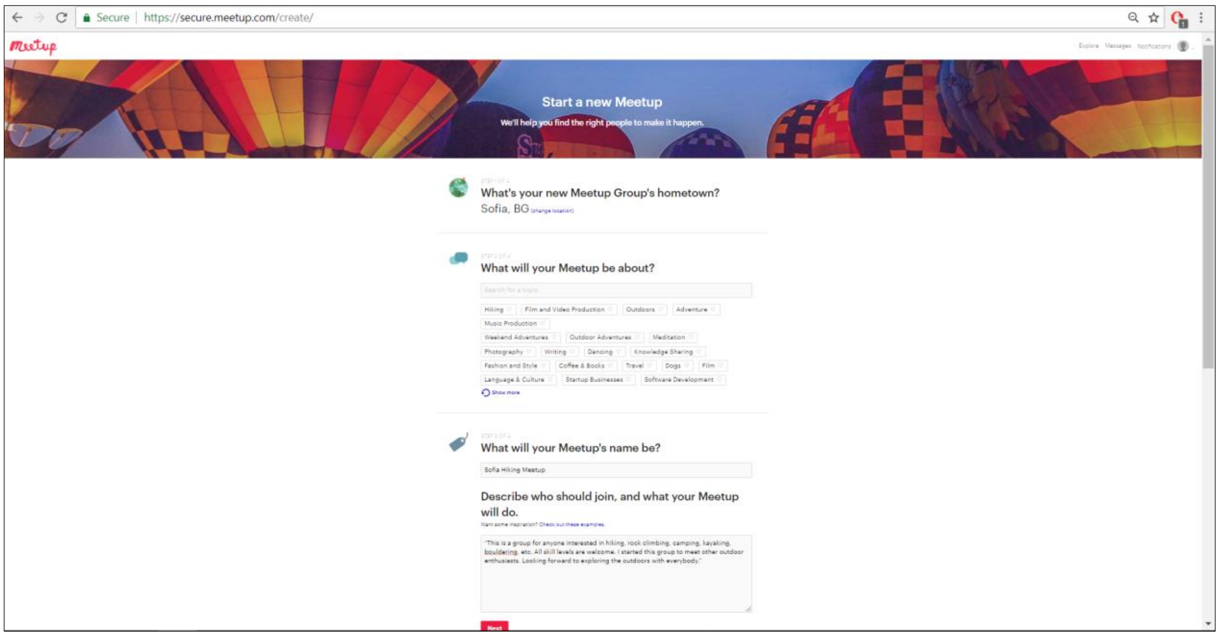
	Facebook	**OutdoorsHobbies**
Target Group	Users of the web site	Users of the website
Platform	It is web-based i.e. it can be used on any platform that supports internet	It is web-based i.e. it can be used on any platform that supports internet
Additional information about the meeting point	No	Yes
Map on which you can select coordinates	No	Yes
Ability to write street name for the meeting point	Yes	No
Ability to change event type (private/public)	Yes	No
Use predefined Destination	No	Yes
Validation for user actions.	Yes	Yes
Supports Tags for events	No	No
Free to use	Yes	Yes

Table 1: Comparison between OutdoorsHobbies and Facebook event creations

In conclusion both of the features are similar with minor differences of the information that the user can write/use for the event, with the outdoorshobies project giving the user more freedom to write additional information.

1.1.2. Meetup

As the name of the site implies this social network is used for creating and joining various meetings.



Start a new Meetup

We'll help you find the right people to make it happen.

What's your new Meetup Group's hometown?
Sofia, BG (change location)

What will your Meetup be about?

Hiking · Film and Video Production · Outdoors · Adventure · Music Production · Weekend Adventures · Outdoor Adventures · Meditation · Photography · Writing · Dancing · Knowledge Sharing · Fashion and Style · Coffee & Books · Travel · Dogs · Film · Language & Culture · Startup Businesses · Software Development

Show more

What will your Meetup's name be?
Sofia Hiking Meetup

Describe who should join, and what your Meetup will do.
Want some inspiration? Check out these examples.

"This is a group for anyone interested in hiking, rock climbing, camping, kayaking, bouldering, etc. All skill levels are welcome. I started this group to meet other outdoor enthusiasts. Looking forward to exploring the outdoors with everybody."

Next

Fig 2: Meetup meeting creation page

The Meetup meeting creation page offers the following features:

- Setting up meeting group hometown
- Selecting tags for the meet up
- Writing name of the meet up and description

As it can be seen both from the described features and figure 2 this web application doesn't offer that much in terms of event creation and they go a step further – to create a new event you have to pay. The plus side of this site is that there are a lot less fake events, because of the payment of creation.

	Meetup	**OutdoorsHobbies**
Target Group	Users of the web site	Users of the website
Platform	It is web-based i.e. it can be used on any platform that supports internet	It is web-based i.e. it can be used on any platform that supports internet
Additional information		

about the meeting point	No	Yes
Map on which you can select coordinates	No	Yes
Ability to write street name for the meeting point	No	No
Ability to change event type (private/public)	No	No
Use predefined Destination	No	Yes
Validation for user actions.	Yes	Yes
Support Tags for events	Yes	No
Free to use	No	Yes

Table 2: Comparison between OutdoorsHobbies and MeetUp event creations

1.2.Conclusion

In conclusion we can say that there are a lot of similar web sites as the one from this thesis. Every application has added different functionality depending on the needs of the clients, but their primary function is the fastest, easiest and the most efficient way of creating events for the users of the sites.

1.3.Goal and tasks of the master thesis

The goal of the master thesis is to create a social network that has the main functionality that is defined in the previous section (1.2.). The point of social network is not to be overcomplicated but to be a simple, clean, intuitive and free to use for all the users of the web application.

The tasks are:

- Design of the DB that will be used.
- Implementation of the back end of the web site using Spring Framework
- Implementation of the front end of the web site using Angular2/5 Framework

2. Architecture and design

2.1. Programing language, platforms, libraries, IDE

This chapter of the thesis describes the technologies used in to implement the module that is the objective of the thesis.

2.1.1. Java Enterprise Edition

"Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals"[1]. Even though I am not going to use a lot of these features Java EE includes a large variety of APIs such as RMI, e-mail, web services and XML. There are several unique specifications to Java EE: EJB, connectors, servlets, JSP and several web service technologies. If we keep in mind the fact that JDK is free to use and 2 of the best IDE that can be found use Java, Java is one of the main contributors to the open source community. With that been said I would like to point out that there are a lot of free to use libraries, some of them have event become an irreplaceable part of the programming community. Another advantage of Java is the large community that is always ready to help with algorithms, ideas and solutions to various problems.

2.1.2. TypeScript

With the rapid development of JavaScript (from now called JS for short) emerged a lot of "deficits" that halt the developers to use the language for application-scale development. Features such as object orientation, compile time error handling and strong type checking are highly valued and sought after in today's programming world. And with that comes TypeScript from now being called TS for short. As said above TS is an object oriented language that supports strong typing and compile time errors with the basic building blocks of JS. With other words TS is an enhanced version of JS (it is often said that TS is typed superset of JS that is compiled to JS).

Some of the important features of the language are exactly that the developers can easily switch from JavaScript to TS because of the similar syntaxes of both languages;

TS can be consumed by any JS library and vice versa, meaning JS is TS and TS is JS. TS encompasses most of the basic features of ECMAScript5 (official specification for JS) and also incorporates features such as Modules, classes, generics and type annotations. So to sum up why I am going to use TS instead of JS for the implementation of the web based system:

- Compile time check – JS is interpreted language that needs to be ran to see if the code works, if everything is alright we see the result instantaneous but if there is error somewhere we have to spend a lot of time to investigate where and why our code doesn't work correctly. TS with its compile time check don't have such problems and we can save some investigation time.
- Strong Typing – JS doesn't have types for the variables which inevitably leads to using variables that you are not sure exactly what they are hence increasing the possibility of making errors in your code. Once again TS has a solution for this problem – strong variable type.
- OOP – this is one of the most common and used principle in the programing. Like the back-end language that I am using for the project(Java) TS also supports concepts like classes, interfaces that will in its most insignificant part boost the readability of the code for outside personnel and that is never to be underestimated for future development.

2.1.3. Spring Framework / SpringBoot

When you start a project one of the things you have to consider are the frameworks that you are going to use. The Spring Framework is one of the most popular and used frameworks for java enterprise applications. Spring is open source, lightweight, high performing framework with which you can develop any java application but it is mostly use for its extensions for building web application on top of J2EE platforms. Some of benefits of using Spring are but not only:

- The design of the framework is modular, meaning even thou there are a lot of packages, classes and interfaces the developer doesn't need to manage all of them, he or she only needs to tend to the ones he/she is using.

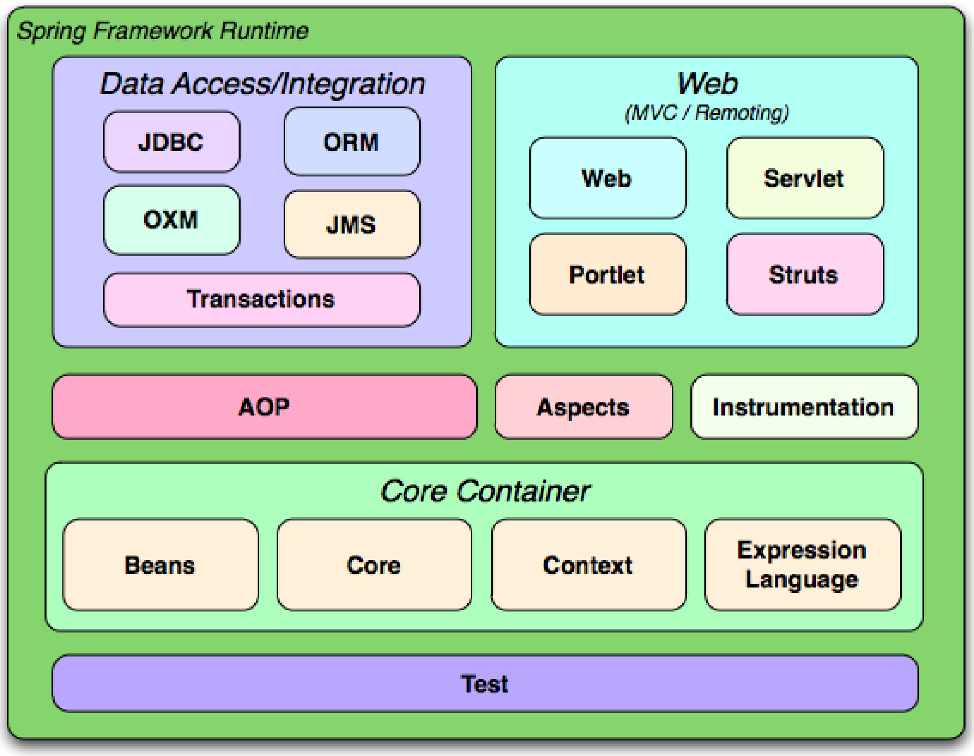
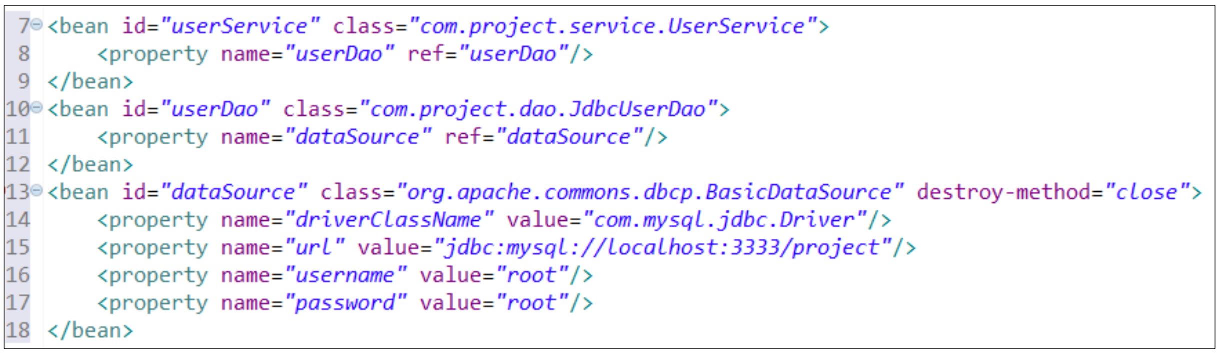


Figure 3: Spring Modules

- The framework incorporates in itself a lot of well know technologies such as ORM frameworks, logging frameworks and J2E.
- Transaction management interface – you can easily scale your transaction up and down when needed.
- Dependency Injection – one of the fundamental concepts in developing application is the ability to test and reuse your code, meaning you don't want your classes to be tightly coupled with each other. Here comes the Dependency Injection into play – it helps us supply the dependency between two classes. A simple example of that is when you have an object car and you know that the car needs another object engine – the DI can give you the already made instance of the engine object so you don't need to create the engine manually.
- POJOs – you can use plain old java objects for better testing or simply to get whole objects from forms (send from the client side).

Spring also provides frameworks for security and data that can easily be used in your project to secure your application better or to simplify database access. Another one of the benefits of the framework is the flexibility to configure beans: currently there are 3 types for that – xml based approach, annotations and JavaConfig approach.

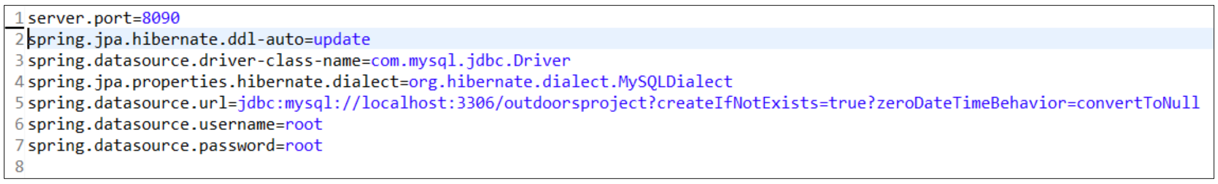


```xml
<bean id="userService" class="com.project.service.UserService">
    <property name="userDao" ref="userDao"/>
</bean>
<bean id="userDao" class="com.project.dao.JdbcUserDao">
    <property name="dataSource" ref="dataSource"/>
</bean>
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3333/project"/>
    <property name="username" value="root"/>
    <property name="password" value="root"/>
</bean>
```

Figure 4: Example of XML approach

The flexibility here comes from the fact that you can use one or more meaning you can combine approaches) of the approaches to configure your beans depending on the situation at hand.

But one of the down sides of the Spring Framework is also the configuration part. Oftentimes when developers new to the framework try to build a project from the ground up, they have numerous problems with the setup. When you try to develop a typical spring/hibernate web application usually the processer of the configuration consists of the following: configure dependencies (using build tools such as maven or gradle), configure the various beans in spring framework (service/dao) – using the latest approach with JavaConfigs ypu can omit some of the lightly descriptions of configurations and using simple properties files (see figure below)



```
server.port=8090
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.datasource.url=jdbc:mysql://localhost:3306/outdoorsproject?createIfNotExists=true?zeroDateTimeBehavior=convertToNull
spring.datasource.username=root
spring.datasource.password=root
```

Figure 5: Properties File Figure

And last but not least you have to configure front controller of this Spring Web application, responsible for handling all application requests i.e. the Spring DispatcherServlet which in itself can be configure in multiple ways. As you can already figure out configuring the framework is not very fast and easy if you have not done it multiple times:

- You are not always sure which libraries you need for your project to run, so you can miss some
- You have to spend some extra time while configuring the DB connections (EntityManager, TransactionManager)
- Configuring the various Spring Beans
- Configuring a web application server

All of these configurations are simplified and even in some cases made automatic by SpringBoot. For example the dependency/library management - in case of the most basic application instead of adding numerous dependencies, with SpringBoot you only need to add 2: spring-boot-starter-web and spring-boot-starter-data-jpa. These two dependencies are easily managed and pull every necessary library that you need such as spring-webmvc, spring-data-jpa, hibernate, tomcat and more. Springboot also creates default configurations for some of the most commonly used beans – front controllers and database connections. Spring-boot-starter-web very conveniently also pulls a servlet container (tomcat) on which serves as an embedded container where you can deploy your application without needing to install an external server/container.

2.1.4. Hibernate (spring-boot-starter-data-jpa)

As already mention above the spring-boot-starter-data-jpa is responsible for pulling a lot of useful libraries that help us with the development and one of those is Hibernate. Hibernate is an Object/Relational Mapping (ORM) framework. With that the framework helps the developers to write applications whose data persist after the applications process. In simpler words, that means that the objects that we use can be available for later usage. While Hibernate has its own API it also uses the Java Persistence API (JPA) which further simplifies and eases its use in any environment supporting JPA. Following the natural object-oriented idioms (inheritance, polymorphism, association, composition and java collections framework) Hibernate allows the developers to create persistent classes with ease. With no requirements for a special database tables and SQL generation at the system initialization time – Hibernate has very high performance. Other reasons for the high performance are the numerous fetching strategies and the usage of eager and lazy initialization. [8]

There is no need to fully understand SQL to use Hibernate, but the understanding of table joins is a must. As the core of relational database, the joins define the relationship between two (or more) tables (When we use the term tables, we also mean objects and vice versa. That is possible because Hibernate is ORM - we create objects with the information from the tables). The need to understand the relationships comes from the necessity to define the fetch type. Their role is simple but important. The fetch types decide whether to load all of the relationships of an object or not when it is initially fetched.

Whenever you specify a relationship (OneToOne, OneToMany, ManyToMany) you also have to specify the fetch type. By default the fetch type is set to "Lazy". That means that whenever you load up something with Hibernate query, Hibernate will not load the information related to other tables (unless explicitly asked) and if you try to access it you will get a NullPointerException. If the "Lazy" fetch type doesn't allow Hibernate to load the relationships of an object then the "Eager" fetch type does the opposite. It load all of the relationships related to the object. If the "Lazy" fetch type can (and possibly will) throw a NullPointerException why use it? The reason is simple – it's faster. Imagine that you want access to an object but the object has 10 relationships to other objects. If you use "Eager" fetch method you have to load all of the objects and that's time consuming. And what happens if every one of those 10 objects has other 10 more relationships to other objects? In conclusion the "Eager" fetch method is more convenient in some ways but can lead to massive delay in the application. With "Lazy" fetch type the developer has to write more code but in the end it's faster and more efficient.

To ease the usage of Hibernate, it uses a HQL - special query language that's close to SQL but is fully object-oriented and can understand the OOP principles like inheritance, polymorphism and association. HQL is also case-insensitive (exceptions are the java classes and properties and column names). That means that sElect is the same as SELECT.

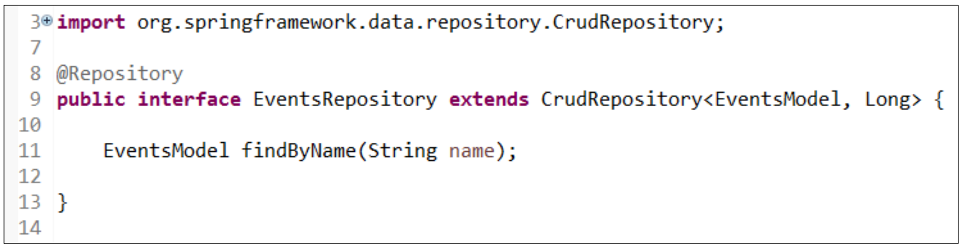
SQL	HQL
SELECT * FROM users	from Users

SELECT *
FROM accounts a, users u
WHERE
a.owner_id = u.user_id
and u.user = '18'
	from site.Account account
	where
	account.owner.id.age = ‘18’

SELECT distinct CONT_NAME
FROM CONTINENT
INNER JOIN COUNTRY on
CONTINENT.CONT_ID =
COUNTRY.CONT_ID
WHERE AREA > 10000
	select distinct cont.name
	from Continent cont joincont.countries
	ctry
	where ctry.area > 10000

Table 3: Comparison between SQL and HQL

With all those features of Hibernate, there is no reason not to use it. And the people from the Spring community come one step further in simplifying its use. With the inclusion of the spring boot starter jpa you automatically get the jpa specification and hibernates implementation as a default, auto reading the data base configurations from the application.properties file and auto creation of entities as tables. Another big improvement is the Spring CrudRepository which builds up on top of the ordinary Hibernate and provides default implementations for the basic CRUD operations i.e. you don’t have (even thou you can) write hql/sql queries.



```
import org.springframework.data.repository.CrudRepository;

@Repository
public interface EventsRepository extends CrudRepository<EventsModel, Long> {

    EventsModel findByName(String name);

}

```

Figure 6: Example of crud repository

2.1.5. CSS3

CSS3 is the latest standard for CSS. While it supports all of the previous versions of CSS it also adds a lot of new features such as rounded corners, animations, shadows and a few new layouts – multi-columns, grid layout. Due to holdbacks of secondary features CSS level 2 needed 9 years to reach the Recommendation status. Thus the CSS Working Group of the W3C took a different approach. They divided CSS in smaller components called modules. Each module is now independent part of CSS and it is developed at its own pace. Some of the most important CSS3 modules are: Selectors, Box Model, Text Effect, 2D/3D Transformations, Animations, Multiple Column Layout and User Interface.

Sadly a big part of the functionality of CSS3 is not supported by some of the older versions of the web browsers. This leads to the additional use of JS to recreate the missing functionality of the unsupported tags. [5]

2.1.6. Bootstrap 3, PrimeNg, Semantic UI

Bootstrap (originally named as Twitter Blueprint) is free, open-source framework created for the purpose of easing the creation of websites and web applications. It contains HTML and CSS based design templates for various elements such as: typography, forms, tables, buttons, navigation and many other interface components. One of the main features of bootstrap is the responsive design. With its flexible grid structure the framework is perfect for web applications that look good on all of the different devices. Bootstrap defines 4 display resolutions categories:

- Xs – extra small (under 768px width, for phones)
- Sm – small (over 768px width, for tablets)
- Md – medium (over 992px width, for desktops)
- Lg – large (over 1200px width, for larger desktops)

The categories above can be combined to create more dynamic and flexible layouts. [6]

“Semantic is a development framework that helps create beautiful, responsive layouts using human-friendly HTML”. [7]. Semantic UI is a modern front-end framework that helps the developers to create good looking and responsive layouts. It is powered by LESS and jQuery and enables the users to use various already made styles for buttons, inputs, fields, containers and many more.

PrimeNg is one of the riches UI libraries for Angular2. It contains in itself vast collections of UI components that can be used together with Angular. Some of the most useful components include dropdown menus, Growl i.e. notification messages, calendars, accordions, confirm dialogs and data table grids that are easy to use and implement. And another big bonus is that it is also opensource which means it is always up to date with the latest's trends and all of the bugs are fixed in a timely manner.

2.1.7. Angular 2/5

As previously mention the choice of the frameworks that are being used is really important when you start a new project. For the back-end of the application there is Spring and as for the front-end there is Angular2. A few years back AngularJS was without a doubt the most popular and preferred choice for framework when dealing with front-end but with the coming of Angular2 and TypeScript that change. Currently Angular2 together with TS is the most desired combination for front-end development amongst the front-end developers. TypeScript is the recommended language for development of Angular2 application but as the language is a superset of JS you can just as easily use JavaScript (both ECMAScript 5 and 6) for your application. When we talk about web development one of the first architectures that come to mind is the MVC but in fact Angular2 is not a MVC framework. It is a component-based framework. What that mean is that the application is a tree of loosely coupled components that interact with each other. This is very useful when building a single page application (SPA) – you don't have to reload the whole page just to update some information inside the page, you just should reload the component that contains the said information. Also, like Spring, Angular2 presents us with a convenient dependency injection so that developer can easily use shared service classes that contain the commonly shared business logic. In Spring, there are a few ways to use dependency injection but in Angular2 there is only one - objects can be injected only via constructors. When we talk about the architecture of the framework its worth to mention that it has 8 major blocks – see the figure below.

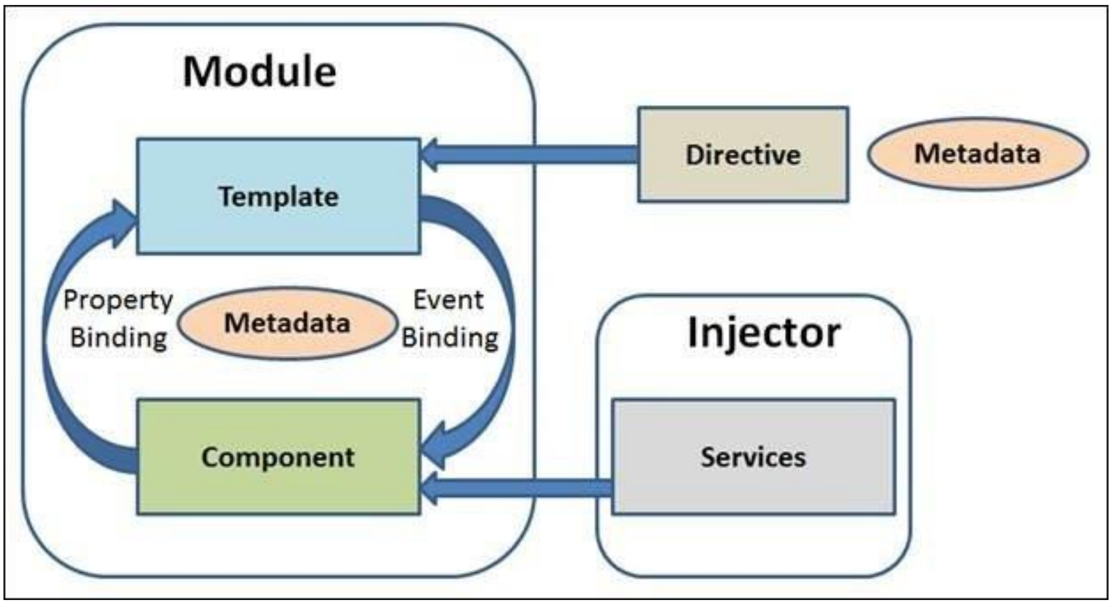
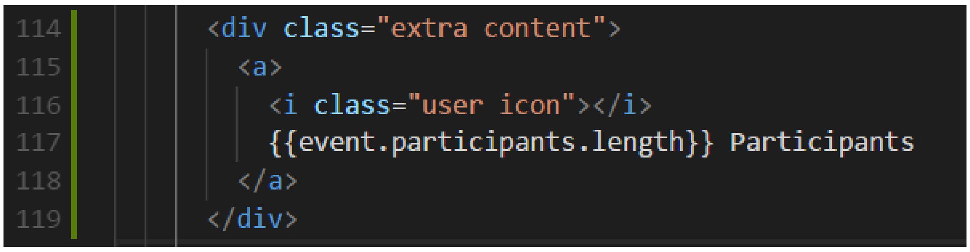


Figure 7: 8 major blocks of Angular2

- Module – the modules can be seen as classes, which perform a single, particular task. To define a class, you must use not only the key word class but also the key word export.
- Component – View of the application and logic on the page. It must be annotated with @Component so it is register as component and there can be only one component per DOM element.
- Template – This is the actual view of the component or with other words said it is the HTML.



```
<div class="extra content">
  <a>
    <i class="user icon"></i>
    {{event.participants.length}} Participants
  </a>
</div>
```

Figure 8 : Part of the template

- Metadata – it is used to extend the functionality of the class. An example is the @Component decorator (annotation).
- Data Biding – this is the feature that connects the view and the model. There are 4 ways supported in Anuglar2 for binding data: Property Binding, Event Binding, Interpolation and Two-Way Binding. Later in the thesis the four way

will be further discussed and showed how will be used in the context of the web application that is the object of the thesis.

- Directive – Directives are used to add behavior to the DOM elements. There are three types: Directive-with-a-template, Structural, Attribute. They will also be discussed further in later chapters.
- Service – class with containing any function, feature with specific purpose. Often used with functionality that can be shared between multiple modules in the application.
- Dependency Injection - It allows injecting a dependency as a service throughout the web application. It also increases the maintainability, reusability and testability.

2.1.9 Maven

Maven is a software management tool that simplifies and makes the building of the projects faster. With Maven, you can easily manage the builds, the documentation, the releases and the dependencies of every application. Maven project structure and contents are declared Project Object Model (POM) file in the form of an xml – pom.xml. Maven uses convention over configuration what that means is that the developers doesn’t need to create the build process themselves – the tools automatically creates a sensible default structure. Some of the features of maven include the following, but also many more:

- Very simple project setup.
- Dependency management.
- A large repository of libraries.
- Release management and distribution publication – can be integrated with version control systems.
- Backward Compatibility.

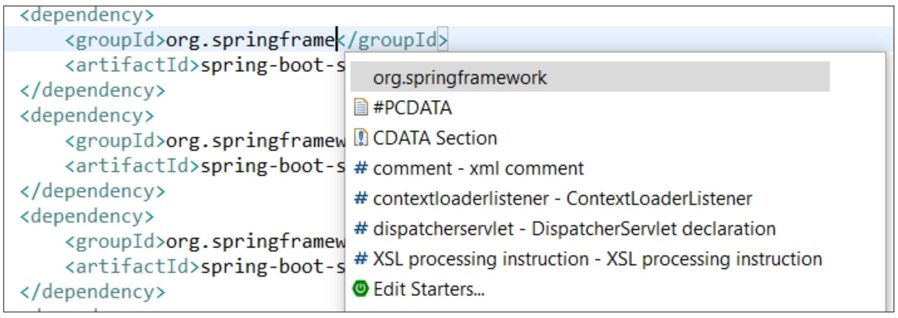
2.1.10 IDEs

Today there is a large variety of IDEs and the right choice of IDE can influence the development of the project throughout the whole project lifecycle. The web based application that is the focus of this thesis can be split in two – front-end part and back-end part. Taking in to account the technologies and languages that are going to be used the best IDEs for the job is without a doubt IntelliJ IDEA that gives us the chance to write both the front-end and back-end in the same place, but sadly it has one major flaw and that it is commercial and you have to pay for it. A good alternative is to use STS for the back-end and Visual Studio Code for the front-end.

The Spring Tool Suite can be said that is built on top of is an Eclipse and that is customized for developing Spring applications. It provides a ready-to-use environment to implement, debug, run, and deploy your Spring applications, including integrations for other technologies that are going to be used in the project such as Git and Maven and a build-in tomcat that is optimized for Spring.

Some of the key features include:

- Recognition and understanding of Spring Projects - parses the configuration files and displays detailed information about the beans.
- Comprehensive Validations for your Spring Configuration – with combination of the previous feature the STS apples automatically validation in the configuration that show errors directly in the IDE.
- Extensive Refactoring Support – the IDE supports not only the usual well know Java refactoring ways, but also new Spring elements refactoring.
- Code Assists – no matter where you are in the project, the STS provides a meaningful content assist and quick fixes for common problems.



```
<dependency>
    <groupId>org.springframe</groupId>
    <artifactId>spring-boot-s
</dependency>
<dependency>
    <groupId>org.springframew
    <artifactId>spring-boot-s
</dependency>
<dependency>
    <groupId>org.springframew
    <artifactId>spring-boot-s
</dependency>
```

org.springframework
#PCDATA
CDATA Section
# comment - xml comment
# contextloaderlistener - ContextLoaderListener
# dispatcherservlet - DispatcherServlet declaration
# XSL processing instruction - XSL processing instruction
Edit Starters...

Figure 9: Code Assist in an XML file

- Graphical Views – The STS provides various relevant to the projects views.
- And many, many more.

Developed by Microsoft, Visual Studio Code is a source code editor with powerful developer tooling such as code completion and debugging. Some of the key features include, but are not limited to:

- Easy editing, building, and debugging – increases productivity by syntax highlighting, bracket-matching, auto-indentation, box-selection. Includes built-in support for IntelliSense code completion for more serious programing logics, interactive debugger and supports Git for fast working with source control.
- Highly Customizable – the IDE is very customizable; you can add any third-party extension that you want with almost no extra configuration. Being an open source project, you can also join the community and contribute to the improvement of the code base.
- Build for web – the IDE comes with enriched built-in support for Node.js development with JavaScript and TypeScript.

2.1.11 Git

Git is a widely-used version control system which tracks changes in computer files and coordinating work on those files among multiple people. One of the main selling points of Git is its architecture distributed architecture. Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes. Other advantages of Git are the enhanced performance, security and flexibility.

2.2. Web Application Architecture

2.2.1. MVC

“Software Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”[ANSI/IEEE Std 1471-2000]. Being a vital part of the web application a poorly selected architecture leads to low performance and availability, insufficient maintainability and expandability. On the other hand, if the right architecture is being used it makes the system more understandable and flexible.

One of the most commonly used architectures (design pattern) is MVC. It is a 3-layer architecture with data layer, presentation layer and business layer. The data layer (Model) represents the core functionality of the system. It is also responsible to the DB manipulation. The presentation layer (View) renders the results of the requests in the desired format. The business layer (Controller) is responsible to connect the Model and the View. It’s also responsible for the actions of the users.

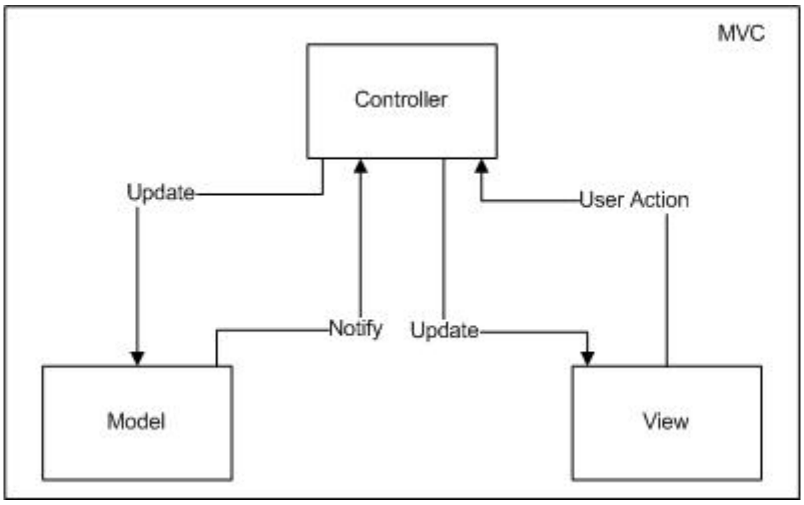


Figure 14: Simple representation of MVC

The MVC design pattern has 3 major benefits which make it one of the most popular and used design patterns in the web programming. The first one is the separation of concerns. The separation in different parts allow the developers to reuse the business logic and on a different units of the program without affection others. The second one is the specialization and focus of the developers. With clearly defined front-end and back-end the developers can focus solely on their tasks of the project i.e. UI developers work on the UI without caring how the back and works and visa versa – the back-end developers don’t need to know how the front-end works. The third benefit is a combination of the first two – parallel development. Without being bound the presentation logic and the business logic can be developed and updated separately from each other and without slowing each other down. With that being said, MVC is simple but very effective design pattern. [12][13]

2.2.2. Architecture of the back-end

As shown above MVC fits perfectly for the needs of the back-end of the site that is the objective of the master thesis because of its separation of concerns.

The Model layer is composed of the following components:

- MySql Database - object-relational database management system that stores the information
- Spring Data – ORM that uses JPA to manage the communication between the module and the DB
- Entities – Java classes that show the exact structure of the DB and store the information that is pulled from the DB.

The Presentation layer is not the typical presentation of most MVC architectures that use technologies such as:

- JSP – Technology that allows the developers to create dynamic web pages.
- Servlets – Java Classes that have doGet and doPost methods that respond to the clients requests.
- Struts2 Actions – Separate methods and classes that transfer date from the request through to the view and determine what result should be rendered in the view.

In the architecture that will be used for the application that is the object of this thesis, the presentation layer will consist of JSON objects that are consumed by another project that will be written in Angular2.

The Controller layer is composed of the following components:

- Spring Beans – Java classes that encapsulate a set of objects in one object.
- RestController – this is a specialized version of the controller in the usual MVC, this controller simply populates and returns an object and the object data will be written directly to the HTTP response as JSON.

Figure 15 shows graphically, how all of the above mention components are positioned in the MVC design pattern.

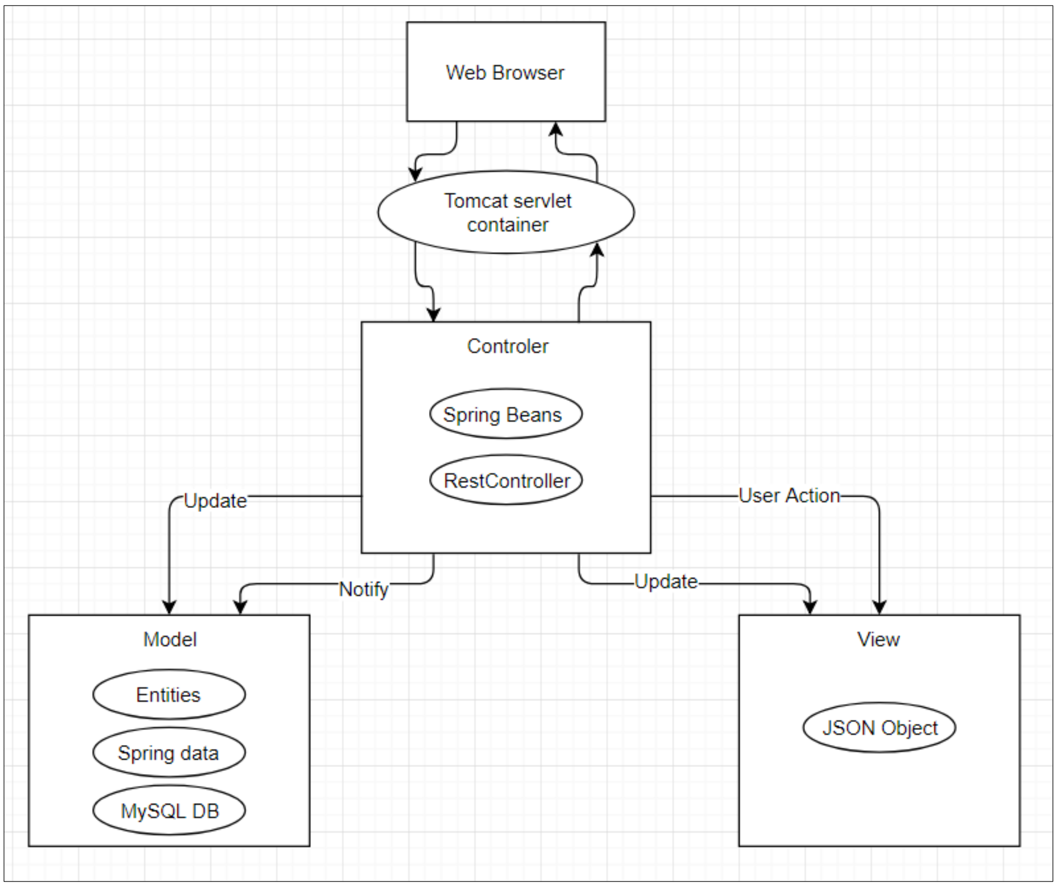


Figure 15: MVC for back-end

2.3.3 Architecture of the front-end

As previously mentioned the front end will be written in angular2 and Typescript. The architecture of the project will simple and it will follow the following requirements:

- Modular design – this means that we should be able to plug in and out of modules very easily. This will make the application more testable and easily managed. This approach decouples the application into blocks of functionalities.
- Unidirectional data flow – Being a modern SPA application, everything will be a component and these components will be the main building blocks for creating and controlling various interfaces. And since Angular organizes the components in a hierarchical tree, we can easily achieve inheritance if needed.

2.3.Data organization

To meet the requirements of the object of the master thesis the web based system needs a place to store the information so a database is need. . For that purpose, is used a local database MySQL. This database is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons like:

- MySQL is released under an open-source license. So it is free for everyone who want to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but can be increased (if the operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments. [15]

2.3.1. ER model of the DB

The presented relationships between the different DB tables show the database structure for the module. For clarity of the ER model the types of the attributes are not included.

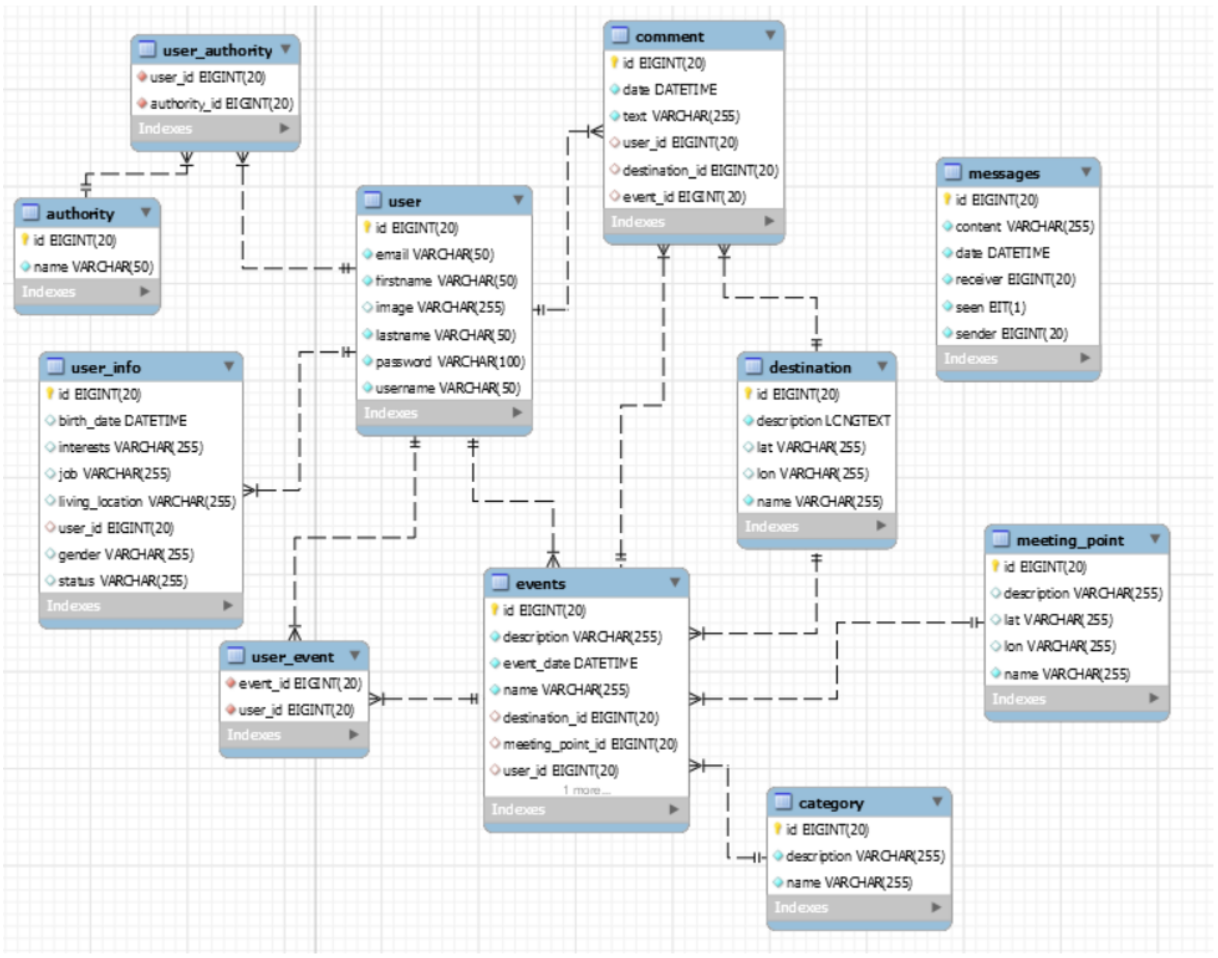


Figure 16: ER model of the DB

On figure 16 can be seen the logical structure of the database of the application. This diagram is visualizing how the various data in the application is related to each other. The name of database of the application is “outdoorsproject”. It contains 11 different tables with data about the users, the events and the destinations and everything necessary for the needed functionalities to work correctly.

A relationship between tables is an important aspect of a good relational database. Well defined relationship means that developer will not facing problems when he try to insert, update, or delete records in related tables. This chapter discusses in detail the tables, with which the database is structured, as well the relationships between them.

In the followed tables are represented all database table structures. Each row of these tables represented each column of the database table. Each table has five columns. The first column only shows the number order of the properties. The Second column contains information about names of the table columns. The third column shows the type of each table column. From the fourth column can be understood if the table column can have null data or not. And the last column shows if the column have some extra configurations.

Table 1 shows the structure of the database table – events. The primary key is "id" (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns "Description", "Event_date" and "Name" contain the information concerning the event directly. "Description" and "Name" are both self-explanatory and are of type varchar (255) meaning that they will contain string information. The "event_date" column will contain the date of the event and is of type datetime. All of these columns are marked as non-null because they are necessary for the event.

#	Name	Type	Null	Extra
1	id (primary key)	Bigint(20)	No	Auto Increment
2	Description	Varchar(225)	No	-
3	Event_date	datetime	No	-
4	Name	Varchar(255)	No	-
5	Destination_id	Bitint(20)	No	-
6	Meeting_point_id	Bigint(20)	No	-
7	User_id	Bitint(20)	No	-

8	Category_id	Bigint(20)	No	-

Table 1: Events

This table can be considered as one of the main tables in the application that is the object of this thesis. It contains all of the necessary information (both directly and "indirectly") needed to visualize the events. The "indirect" information that was mention previously is the last three columns of the "events" table. They represent relationships with other tables and also cannot be null.

"Destination_id" is a relationship with the "Destination" table, it is a foreign key. The relationship here is many-to-one. A many-to-one association is one of the most common kinds of association where an Object can be associated with multiple objects. Many-to-one mapping means that one row in a table is mapped to multiple rows in another table. In this case the event has one Destination, but this same Destination can be also associated with another Event as well.

"Meeting_point_id" marks the relationship with the "Meeting_point" table, it is a foreign key. The relationship here is one-to-one. A one-to-one association is similar to many-to-one association with a difference that the column will be set as unique. In this case an event record (object) can be associated with a single meeting point_ record (object).

"User_id" marks the relationship with the "User" table. The relationship here is many-to-one and represents the "owner" of an event. So in this case the Event can have one owner but this same owner (user) can be associated with another Events. The column is also part of a many-to-many relationship again with the User table to define the participants that join the event. In this case many events can have many participants or with simpler words any user can join any event and any event can be joined by any user. To achieve the desired affect the two tables – "User" and "Events" are joined with a third table called "User_events".

"Category_id" is a relationship with the "Category" table, it is a foreign key. The relationship here is many-to-one. So in this case the Event can have one category but this same category can be associated with another event.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	-
2	Description	Varchar(225)	No	-
3	Name	Varchar(225)	No	-

Table 2: Category

Table 2 shows the structure of the database table – Category. The primary key is "id" (also marked in the table below). The column is not marked as auto increment because there are a limited number of categories that are inserted in the database and the user will not be able to change that, making the auto increment unnecessary. Also the column is marked as Not Null, meaning that this column can never be null. The columns "Description" and "Name" contain the information concerning the category directly and are self-explanatory. Both columns are of type varchar (they will contain strings) and are also marked as not null.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	text	Varchar(225)	No	-
3	date	datetime	No	-
5	Destination_id	Bitint(20)	Yes	-
6	Event_id	Bigint(20)	Yes	-
7	User_id	Bitint(20)	No	-

Table 3: Comment

Table 3 shows the structure of the database table – Comment. The primary key is "id" (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns "text" and "date" contain the information concerning the event directly. "Text" is the column that contains the content of every comment a user makes in the site. It is of type varchar and it cannot be null. The column “date” represents the time at which point the comment is made by the users. It is of type datetime and also cannot be null.

This table can also be considered as one of the main tables in the application that is the object of this thesis. It contains all of the necessary information (both directly and “indirectly”) needed for all of the comments in the site. The “indirect” information that was mention previously is the last three columns of the “Comment” table. They represent relationships with other tables and also cannot be null.

“Destination_id” is a relationship with the “Destination” table, it is a foreign key. The relationship here is many-to-one. As previously mentioned a many-to-one association is one of the most common kinds of association where an Object can be associated with multiple objects. In this case here the comment is associated with one destination, but this same destination can be also associated with other comments as well. The field is not marked as not null, because one comment can be either associated with a destination or an event – it cannot be associated with both.

“Event_id” is a relationship with the “Events” table, it is a foreign key. The relationship here is many-to-one. In this case here the comment is associated with one event, but this same event can be also associated with other comments as well. The field is not marked as not null, because one comment can be either associated with a destination or an event – it cannot be associated with both.

“User_id” marks the relationship with the “User” table. The relationship here is many-to-one and represents the creator of the comment. So in this case the comment can have one creator (user) but this same creator (user) can be associated with other comments.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	Description	Varchar(225)	No	-
4	Lat	Varchar(255)	No	-
5	Lon	Varchar(255)	No	-

6	name	Varchar(255)	No	-

Table 4: Destination

Table 4 shows the structure of the database table – Destination. The primary key is "id" (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns "Description", "Lat", "Lon" and "name" contain the information concerning the destination. All of these columns are marked as not null and are of type varchar. "Lon" and "Lat" represent the exact coordination of the destination. This table can also be considering as main table for the application because of all the relationships that it has with other tables.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	Description	Varchar(225)	No	-
4	Lat	Varchar(255)	No	-
5	Lon	Varchar(255)	No	-
6	name	Varchar(255)	No	-

Table 5: Meeting_Point

Table 5 shows the structure of the database table – Meeting_Point. The primary key is "id" (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. Also the column is marked as Not Null, meaning that this column can never be null. Similar to the columns in the Destination table, the columns here are also "Description", "Lat", "Lon" and "name". They contain the information concerning the meeting point of every event. All of these columns are marked as not null and are of type varchar. "Lon" and "Lat" represent the exact coordination of the destination. It is important to mention that because of the one-to-one relationship with the "Events" table, every time a new event is being made a new record will be inserted in the table.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	Content	Varchar(225)	No	-
3	date	datetime	No	-
4	receiver	Varchar(255)	No	-
5	sender	Varchar(255)	No	-
6	seen	Bit(1)	No	Default value 0

Table 6: Messages

Table 6 shows the structure of the database table – Messages. The primary key is "id" (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. Also the column is marked as Not Null, meaning that this column can never be null. The other columns in the table are "Content", "date", 'receiver", "sender" and "seen". The "Content" column is self-explanatory – it contains the content of the messages. It cannot be null and it is of type varchar. "Date" represents the date and time at which the massage is being send. It's of type datetime and also cannot be null. The "seen" column is of type bit and it also cannot be null. The column represents if a message has been seen by a user. It has a default value of 0, meaning that it has not been read yet. The columns "receiver" and "sender" represent the user that have send the massage and receive it. To limit the unnecessary relationships from and to the Users table these two columns are not foreign keys but instead are the usernames of users.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	email	Varchar(225)	No	-
3	firstname	Varchar(225)	No	-
4	Lastname	Varchar(225)	No	-
5	image	Varchar(225)	Yes	-
6	Password	Varchar(225)	No	-
7	username	Varchar(225)	No	Unique

Table 7: User

Table 7 shows the structure of the database table – User. The primary key is “id” (also m arked in the table below). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The other columns in the table are “email”, “firstname”, ‘lastname”, “image” and “password” and “username”. All of the columns are of type varchar and all of them except “image” cannot be null. The image column represents a string that is the name of the profile picture of the user. The “username” column has an extra configuration and that is that it should be unique – meaning that there can be only unique usernames in the site. This table can be considered as main table, because it contains the login information for each user. When user log in to the site, the data entered by user will be compare with the “username” and “password” columns from this table.

#	Name	Type	Null	Extra
1	User_id	bigint(20)	No	-
2	Authority_id	bigint(20)	No	-

Table 8: User_authority

Table 8 shows the structure of the database table – User_authority. The idea behind this table is that it helps with the many-to-many relationship between the “User” table and the “Authority” table. It contains two columns – “user_id” and “Authority_id”. They are both of type bigint and also cannot be null.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	name	Varchar(50)	No	-

Table 9: Authority

Table 9 shows the structure of the database table – Authority. The primary key is “id” (also marked in the table above). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The other column in the table is “name” – also marked that it cannot be null. The table is used to set security roles for the application. It’s worth mentioning that currently there is only one active role in the application and that’s for normal users, but if the need arises an admin role can be easily added.

#	Name	Type	Null	Extra
1	User_id	bigint(20)	No	-
2	Event_id	bigint(20)	No	-

Table 10: User_event

Table 10 shows the structure of the database table – User_event. The idea behind this table is that it helps with the many-to-many relationship between the “User” table and the “Event” table. It contains two columns – “user_id” and “event_id”. They are both of type bigint and also cannot be null.

The last table shows the structure of the database table – User_info. This table can also be considered as one of the main tables in the application because it contains all of the extra information about the users. The primary key is “id” (also marked in the table above). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The columns “job”, “intrests”, “living_location”, “gender”, “status” are all of type varchar and can be null as they are not obligatory for the site to function correctly. The column birth_date is used to store the users birth day, it’s of type datetime and also have null value.

“User_id” marks the relationship with the “User” table. The relationship here is one-to-one because one user can have only one specific user information about itself and vice versa.

#	Name	Type	Null	Extra
1	id (primary key)	bigint(20)	No	Auto Increment
2	job	Varchar(225)	Yes	-

3	birth_date	datetime	Yes	-
4	interests	Varchar(255)	Yes	-
5	Living_location	Varchar(255)	Yes	-
6	gender	Varchar(255)	Yes	-
7	User_id	Bitint(20)	No	-
8	Status	Varchar(255)	Yes	-

Table 1: User_Info

It is also worth mentioning that there are some auto-generated tables that help with the security of the site, but I'll not go into more details about them because all of the information in them is not managed by the code directly.

3. Practical solution

As mention earlier in the thesis the web application consists of two separate projects – one for the back-end i.e. the server side and one for the front-end i.e. the client side. All the technologies, libraries and programing languages listed in the architecture and design part of the thesis are being used for the implementation of both projects. In this chapter the implementation will be thoroughly reviewed and explained. Because of the unusual architecture – having two separate projects for the front-end and the back-end, a special attention will be given to the setup of the environment. Also, each of the layers in both projects will be reviewed and it will be explained how they interact with one another.

3.1.Solution to the problem

To complete the objective of the bachelor thesis we have decided to use the technologies, libraries and programing languages that are described in the previous section of the thesis. To achieve a nice, clean and intuitive module for templates, everything has to be kept as simpler as it can be while keeping as much of the functionality.

When the site is loaded in front of the user will be display a login form that prompts the user to log in the site using his/her profile, if the user is new to the site he/or she can also create a new account. It’s important to mention that the content of the site should not be visible without being logged in the site. To make the user experience as best as it can be the registration form should not include all of the fields that can be written by the user, but only the essentials. Also there should be basic validation for the fields in the form. After every field is correctly written for the registration and the user clicks the register button, a new record should be written in the data base containing the said information. If record is successfully saved the user should be redirected back to the logging page where he or she can immediately, without needing to confirm the registration through email or needing to wait for administrator’s approval, log in the site and begin using its functionalities. The log in form should be simple username/password login form that checks the database for existing user. If the username or the password is incorrect the logging in should not be successful. If they are correct the use should be redirected to the content of the site.

After successful login the user should be greeted by a nice looking but in the same time simple interface to all of the main functionalities of the site:

- Home page
- Event page
- Destination page
- Chat page
- Messages page
- Search functionality
- Drop down menu to control user

The home page should display the information of the user. This information includes personal information, profile picture (if there is a set on), interests, a status input, list of all the events that the user has been to or is going to be, and the next event that the user is going to go to.

The event page should contain lists with the past events and upcoming events. These events should have some kind of information about them displayed such as date, picture, number of participants etc. and functionality to redirect to the said event when clicked on it. There should also be a functionality that enables the user to create a new event. When the button is pressed, the user should be redirected to a new page where he or she would be prompt to type the necessary information for the creation of a new event such as title, description, date, meeting point and destination. Basic validation of the input should also be included here. When a user clicks on one of the events, he or she should be redirected to the information page of the said event. Here should be displayed the most important information about the event such as where and when will be held the event and who is going to the event. The user should have the ability to join the said event and/or write comments for the event.

In the Destination page the user should be able to see all of the available destination as well he or she should be able to create new destinations. When creating new destinations the user should be able to write the necessary information for the creation of a new destination such as title, description, location and a picture. Similar to the events page, when the user clicks on a destination he or she should be redirected to the destination information page, where all of the information for the destination can be found. In the destination information page the user should be able to review the information as well as write comments and add more pictures of the destination.

Chat page – this should be a global chat where all of the users can come together and chat with each other.

The Messages page should enable the user to write new massages to other user and receive/ read messages sent from other users.

The search functionality should enable the user to search other users, events and destinations and redirect the user to them.

The Drop down menu should be a standard dropdown that shows two additional functionalities – logging out of the site and editing the personal data of the user. The editing personal data functionality should be done in a new page where the user should be able to add or change his photo or personal information.

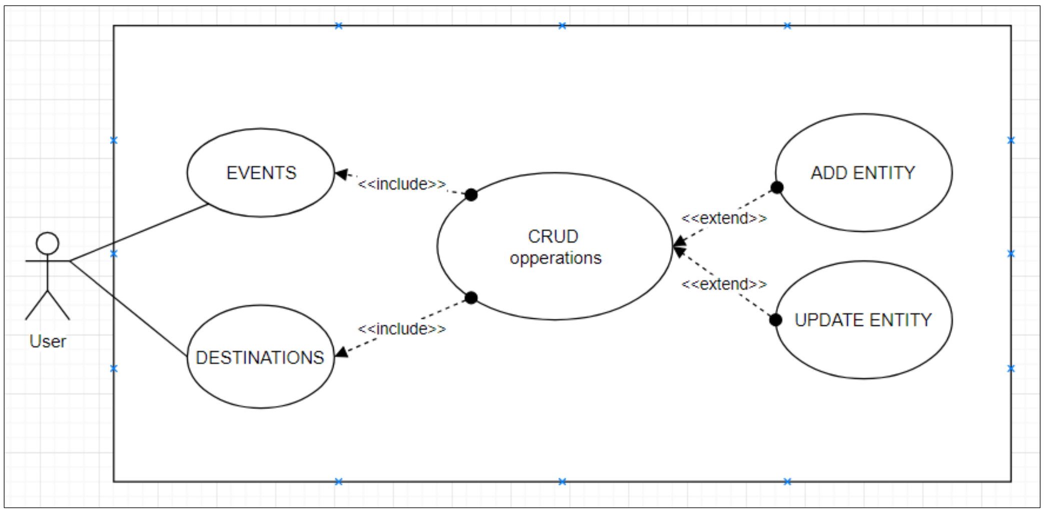


Figure 10: UML Use Case diagram for the main

The presented Use Case Diagram shows a basic representation of the needed functionality for creating and editing the destinations and the events. The functionalities are supported by the operations that they use.

As for the functionality of the site, the following flow char diagrams show some of the algorithms that help to achieve the functionalities described at the beginning of the chapter.

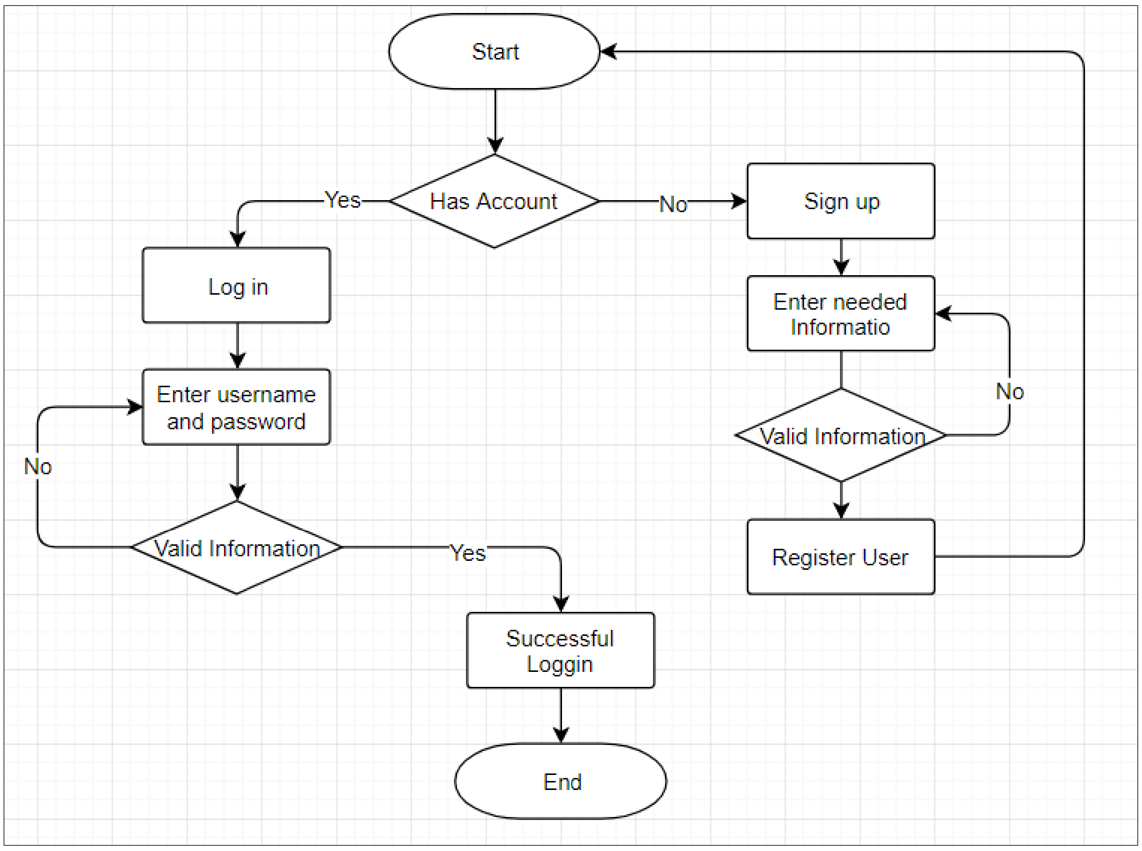


Figure 11: Flow Char diagram of the algorithm for sign up and sign in the site.

The main goal of the algorithm above will be to check if a user has an account. If the user doesn’t have one, one must be created to enter the site. To do that the correct information should be verified and stored in the database at which point if the information is successfully recorded the user will be ask to write the username and password for the account. If the username and the password are correct the user will proceed in entering the site. It is worth mentioning that there should be no way possible for unregistered user to enter the site.

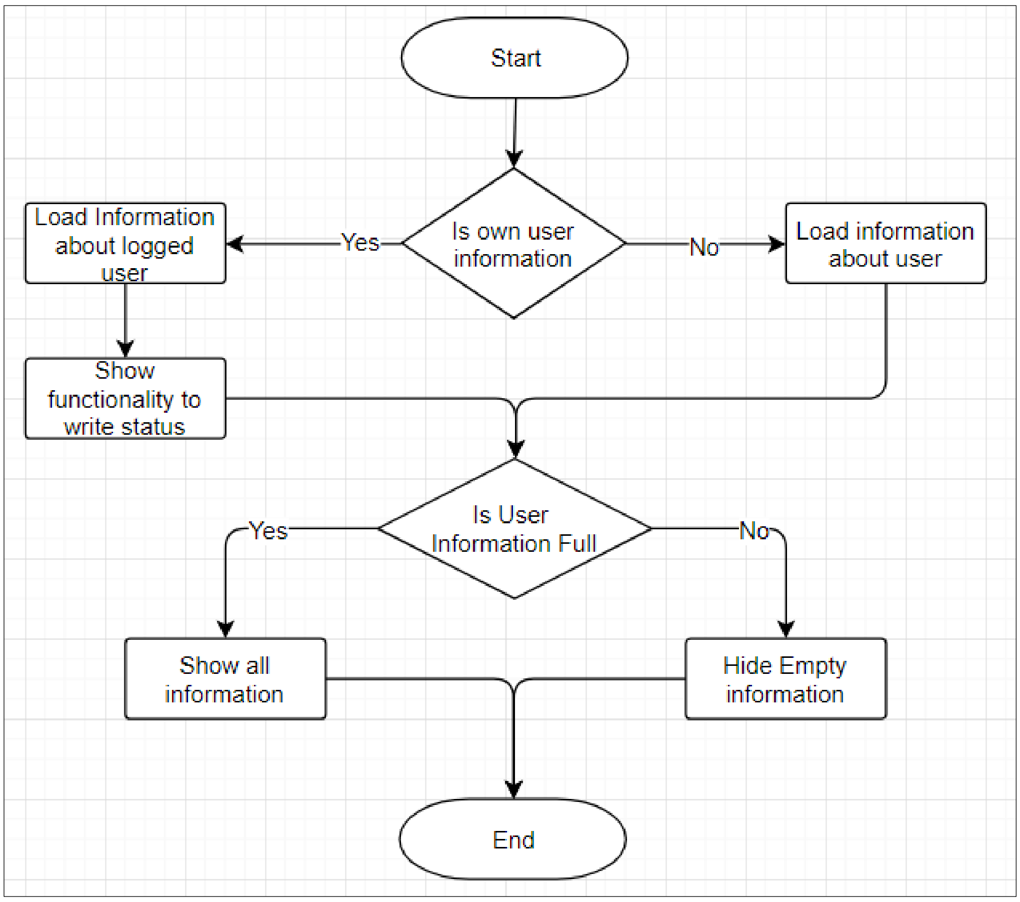


Figure 12: Flow Char diagram of the algorithm for loading the user information page.

The main goal of the algorithm above will be to load the correct information in the user information page. The first check that it should make is if the logged user is trying to view his/hers own profile information or the information of another user. If the logged user is trying to look at the information page of his account, the correct info should be loaded and an additional functionality for writing statuses should be displayed. In both cases, when the user is trying to see his own information or when he/she is trying to see another users information if the information in the DB is not full i.e. the said user hasn’t written all his information some of the fields in the web page will be omitted and not displayed.

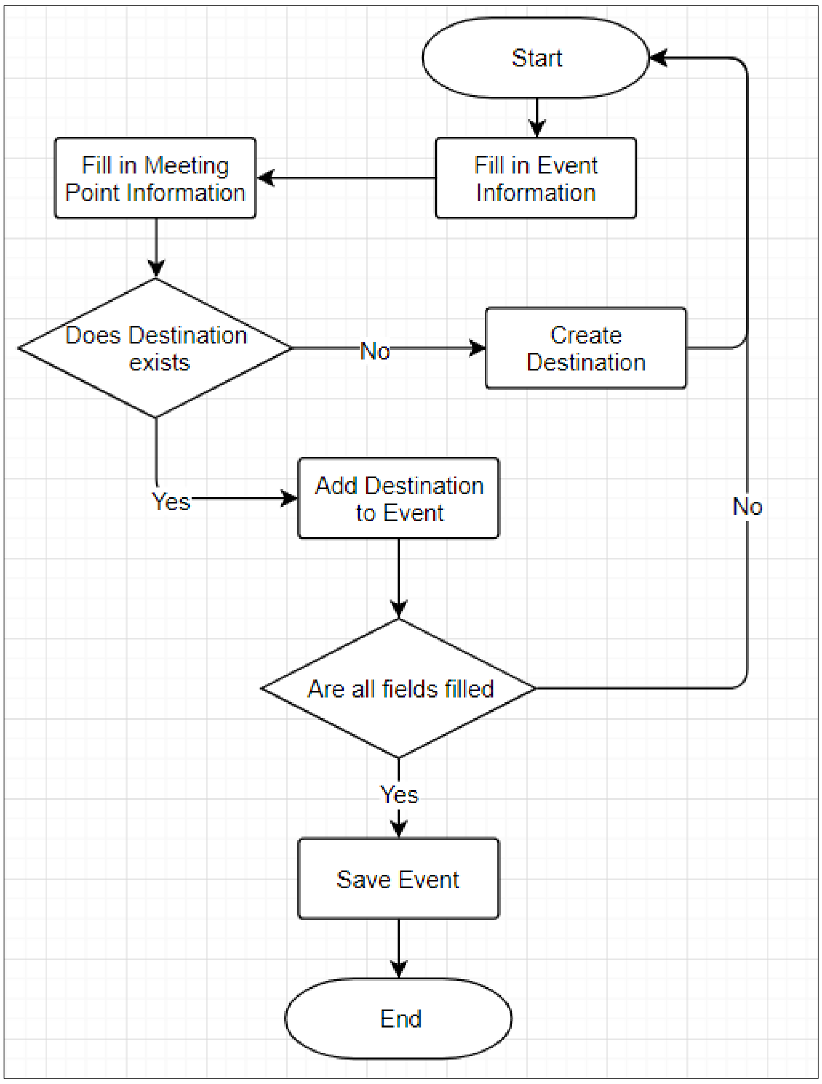
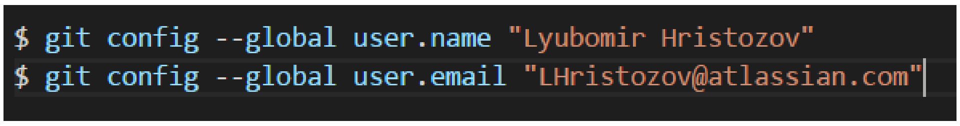


Figure 13: Flow Char diagram of the algorithm for creating new Events.

The main goal of the algorithm above will be to allow the user to create a new Event. First all the fields should be filled, that includes the event information, meeting point information and destination. If the user wants to create an event that has a destination that has yet to be added to the data base of the application, the said destination should be added to the database. Before the saving of the event, verification must be made that all of the fields are not empty; if any are empty the event should not be saved in the database.

3.2. Setting up the Workspace

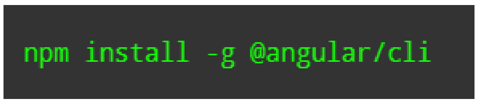
Since the web site is being split in two separate projects the first step in the development is the correct configuration and installation of all of the necessary software that was mention above and of course all of the necessary helper software for those technologies. In this part of the thesis, a short walkthrough will be given on the necessary steps to set up the workspace. Since we are planning to use Java as a main back-end technology first we need to have the Java Development Kit (JDK) installed on the computer that we will be using to develop the web based system. The installation of JDK is free and anybody can download it from the Oracle official site and install it. The next step is the installation of a software versioning and revision control system. We will be using Git (Github) because it offers an easy to use repository of current and historical versions of the projects files. The installation itself is very straightforward and easy. For windows user there is a convenient to use installer – for other OS there is a step by step installation guides. The only configuration that you have to do is to configure your username and email – see figure 17.



```
$ git config --global user.name "Lyubomir Hristozov"
$ git config --global user.email "LHristozov@atlassian.com"
```

Figure 17: Bash commands to configure username and email

After this is done you can use it as such and only use it as a local repository, but that is not very effective because if you go to another computer you cannot download and use your code very easy. So to avoid this you have to make yourself a repository. For this particular project I have selected Github. In the official site everything is described step by step how to start so I'll not be going in details. The next step in the setting up of the workstation is the installation of an IDEs. As said in the previous section we will be using STS for the server side project (Spring project) and Visual Studio Code for the client side (Angular2/5 project). As both of these IDEs are free to use, you can download them from their official sites and use the installation guides there to install them. The next step in the setup is the installation of Angular. To install Angular we need to install the Angular CLI. But to do this we actually need to install something else first and that is npm – this is a package manager for JavaScript and the world's largest software registry. Luckily this is very easy because npm is distributed with Node.js - which means that when you download Node.js, you automatically get npm installed on your computer. So we simply have to download node.js and install it. After all of this is done the installation of angular itself is very easy with a singular command:



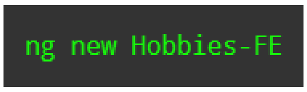
```
npm install -g @angular/cli
```

Figure 18: Command to install Angular CLI

The next important thing that is needed for the project is the software management tool. As mention previously the project will use Maven. As a free to use Maven can be directly download from the official Apache site. The only configuration that has to be made is to create 2 new variables (M2_HOME and MAVEN_HOME) in the Windows environment and point them to the installation folder of Maven and of course them to the PATH variable. If you already have maven, but you are not sure what version you are using or if you doubt, that the installation is done - you can simply run "mvn –version" in the command prompt. If everything is right you will see information about your maven version, the maven directory, the version of the Java you are using and the location of the java.

So now we have all the necessary software installed so we can proceed with the setup of the project themselves. First is the Spring project. Luckily the Spring Tool Suite (STS) is design to help its user with such task so we can simply create a Spring Boot Maven project directly from it - File > New > Spring Starter Project. This will open the creation wizard where you will be prompt to select the standard configurations of the project such as Group ID, Artifact ID, Root package, build tool – in our case Maven and etc. On the next page you will be asked to add dependencies that you will be using. It is not necessary to select all any dependencies – they can be added at any time from the Pom.xml file. With this a new project will be created and if everything is correctly set, this is all for the setting up of the Spring Boot project. In the next chapter the file stricter and all of the dependencies will be reviewed.

The setting up of the Angular 2/5 project is fairly simpler. Since we are using the Angular CLI we only need to input a command in the command line and everything will be done for us automatically (See figure 19). The next chapter will also include the structure and dependencies of the front-end project.



```
ng new Hobbies-FE
```

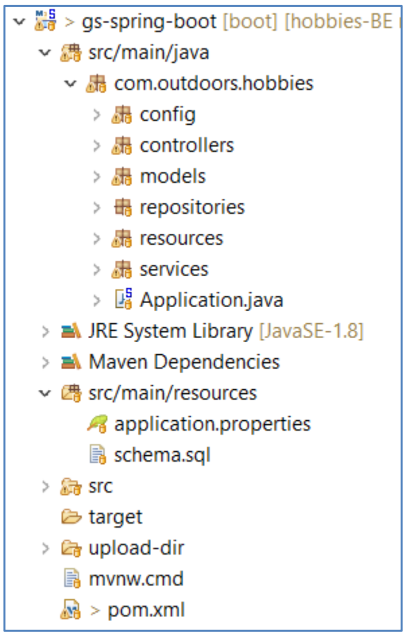
Figure 19: Setting up of Angular project

3.3. File Structure and dependencies of the application

In this section a short explanation of the file structure of both projects will be given. First will be the back-end. Being a Spring Boot project the files are organized in the standard application assembly (packaging) way. The packages can be divided in 2 types- module and full-fledged applications. A Spring module is a collection of components of the same type such as service or repository or etc. The module also has to have the necessary type of deployment descriptors. If there are no deployment descriptors, the module can be deployed as a stand-alone module. So in the project there are 6 packages:

- Config – as the name implies in this package will be stored all of the configuration files of the application such as the SecurityConfig, CorsConfig and AuthorizationServerConfig files.
- Controllers – again self-exploratory name, this package will contain all of the controllers of the application for example the EventsController, the DestinationConroller and CommentController.
- Models – this package contains the Entities of the application. Examples are the EventsModel, the DestinationModel and the CommentModel.
- Repositories – as we will be using hibernate and spring data we need repositories for the various Entities. Examples for repositories are the EventsRepository, the DestinationRepository and the CommmentRepository.

- Resources – the resource package contains plain old java objects (POJOs). We need them for easier manipulation of data. Examples are the EventsResource, the DestinationResource and the CommentResource.
- Services – as the name implies the package contains the service classes of the application – exmaples are the EventsService, the DestinationService, and the CommentService.



```
> gs-spring-boot [boot] [hobbies-BE
  src/main/java
    com.outdoors.hobbies
      > config
      > controllers
      > models
      > repositories
      > resources
      > services
      > Application.java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  src/main/resources
    application.properties
    schema.sql
  > src
  target
  > upload-dir
  mvnw.cmd
  > pom.xml
```

Figure 20: Structure of the Spring

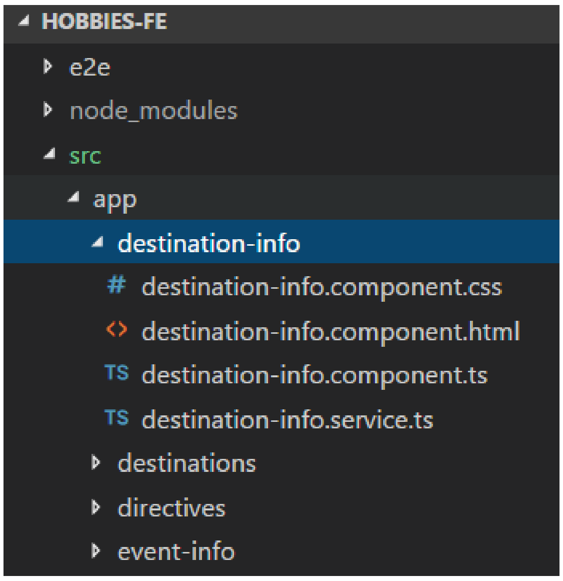
All of the file in these packages share a common good naming practice – the first word of the name is the name of the entity and the last word is the name of the type i.e. service/repository etc.

First it is important to mention that when the creation wizard was creating files it made one file that defines the project as a spring boot project – Application.java. This file can be viewed as the starting point of the application – it contains the main method and also there is the @SpringBootApplication annotation that marks the class and the project. All of the dependencies in the project are defined in the pom.xml file. Here are auto-defined from the creation wizard the spring-boot-starter-web and spring-boot-starter-data-jpa dependencies. Another important dependency that must be added is the dependency for the database that the application is going to interact with - mysqlconnector-java. Of course there are other dependencies that are added but they will be explained and reviewed later in the thesis. Another equally important file is the application.properties file. As the name suggests this file contains the properties of the project such as the server port that the application will be using, the database dialect, driver, url, username and password. Here can also be defined some extra properties such as “spring.jpa.show-sql = true“, that enables the sql queries to be showed in the server console. All of these files can be seen in Figure 20.

When using the angular cli to start a new project, there are a lot of folders and files that are auto created and most of them can be ignored. The structure can be split in 2 main categories: inside and outside of the src and e2e folders. Inside the two folders are the files that most of the developers are concerned about – such as the angular application files or the end-to-end tests of the application. Outside of these folders are the files that condition the development environment. Once setup they rarely change and are not of any big concern throughout the development. Some of the more essential files are:

- index.html – application host page, it loads scripts in set order and boots the application.
- package.json – contains package dependencies and command scripts for running the application and more.
- styles.css – as the name implies here are the global styles of the application.

As previously mentioned all of the application files are in the src folder. All of the features are separated in their own folders meaning each component that has multiple files accompanying it (those are the .ts, .html and .css files) has its own folder named after the component. With this separation, a nice clean, readable and easily scalable structure is achieved. If it is not done correctly the multiples files of every component can easily clutter the src folder, making it hard to work with. All of the feature (component) files use consistent names that first describe the component and then show its type – this way it’s easier to find whatever you are looking for.



```
HOBBIES-FE
  e2e
  node_modules
  src
    app
      destination-info
        destination-info.component.css
        destination-info.component.html
        destination-info.component.ts
        destination-info.service.ts
      destinations
      directives
      event-info
```

Figure 21: Part of the project structure

As showed on the figure above the feature/component “destination-info” can be found in a folder named after it. In the folder there are all of the necessary files that accompany the component such as the css file that contains the styles that are only applied to the component, the html file that is the visual representation of the component, the component itself and the service file that contains the connection with the backend project.

3.4. Implementation

Since the web based system is split in two separate projects for the server side and the client side, the implementation will as well be split in two for the both sub-projects.

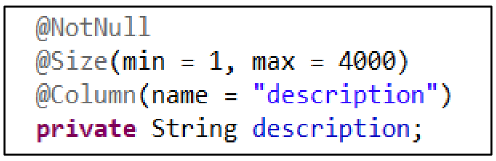
3.4.1. Back-end Implementation

To start the implementation of the back-end the first thing we have to do is create the entities that we will be working with. The entity classes (JPA entities) are simply a representation of the database tables. The Entity classes are plain POJOs and as such they have to follow some simple rules such as:

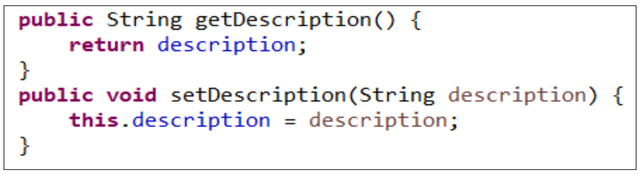
- Persisted classes need a default constructor – luckily java provides us with it automatically and we don’t need to create it manually.
- Persisted attributes should be encapsulated and should have the appropriate getters and setters.
- All classes should contain ID for easy identification of objects within JPA and the database.

Because the Entity classes follow the same rules they all have similar form and an explanation only to one of those classes will be given. Since the scope of the project is very wide only one feature will be reviewed and discussed. The feature that will be reviewed will be one of the main features of the site – the events. So the first class that will be viewed is called “EventsModel” and as the name suggests it represents the table Events from the Database. As a POJO that follows the rules above. As said before we don’t need to write a default constructor because java creates one for us automatically. The reason why we need to have a constructor is because to instantiate a persistent class Hibernate must have no-argument constructor because it uses Constructor.newInstance(). If there are other constructors in the entity class, the default constructor is not needed - it is provided by the JVM.

Every column in the table should be represented by a field with the same name as the name of the column. The access level of the fields must be set to private. As private fields they must also have the appropriate getters and setters methods.



```
@NotNull
@Size(min = 1, max = 4000)
@Column(name = "description")
private String description;
```



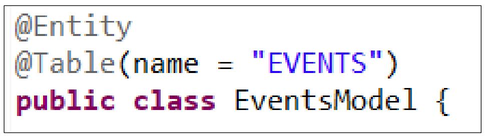
```
public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
}
```

Figure 22: Code Snippet for declaration of a field and its setter and getter methods

There are two ways to transfer information from the entity classes to the database tables. The first one is with the use of XML mapping file and the second is with the use of JPA Annotations. We have decided to use the later. The JPA Annotations provide a better understanding of the table structure and entity classes simultaneously during the development. There are 2 things that are necessary to use the JPA Annotations. The first being the need for JDK version 5.0 or higher and second is the JPA annotations distribution package.

To connect the entity class to the table the annotation @Table is being used above the class. This annotation provides some additional attributes such as:

- Override the name of the table
- Override the catalogue of the table
- Override the schema of the table
- Enforce unique constraints on columns



```
@Entity
@Table(name = "EVENTS")
public class EventsModel {
```

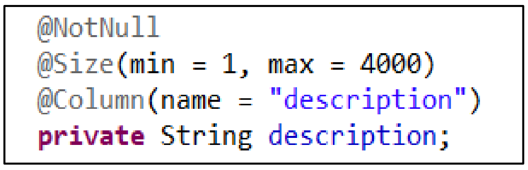
Figure 23: Code Snippet for connecting the entity class to the database table.

The @Entity annotation (see figure above) marks the class as a persistence class and must have a no-argument constructor, fields that represent the columns of the database table and the appropriate getters and setters.

After connecting the name of the class to the name of the table we need to connect every field in the class to the desired column from the table. This is achieved through the use of the @Column annotation (see figure 24 above). This annotation is used to specify details about the column. Some of the most commonly used attributes are:

- Name – explicitly specify the name of the column.
- Length – explicitly specify the size of a column value (used mostly for strings).
- Nullable – allows a column to be marked as NOT NULL.
- Unique – allows a column to be marked as containing only unique values.

JPA offers a large variety of annotations. One of the most useful ones are the annotation that offers validation:



```
@NotNull
@Size(min = 1, max = 4000)
@Column(name = "description")
private String description;
```

Figure 24: Code Snippet for annotations that offer validation.

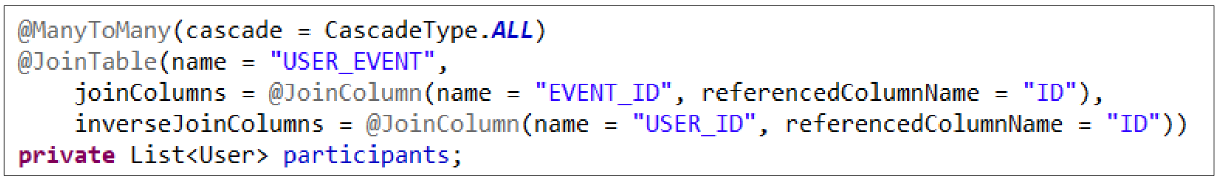
On the figure above can be seen two validations. These validations are on the application level – they are faster than the database validations. The first two annotations in figure 24 validate the field before a query to the database is made. The first validation checks if the field has a value of null (i.e. its empty). The second validation checks if the size of the field is between one and four thousand.

As every table has a primary key (it can also have more than one), that primary key also should be noted as such in the entity class. The annotation that represents that a field (column) is a primary key is @Id.

To create a relationship between the persistent classes jpa uses 3 different degrees of relationship annotations:

- @ManyToOne / @OneToMany - one of the most commonly used relationships. We use one of those depending on the perspective we are looking at the relationship. An Object can be associated with multiple objects.
- @OneToOne – similar to @ManyToOne but the column will be set as a unique.
- @ManyToMany – usually implemented with a Set Collection.

The annotation @JoinTable indicates that there will be an interaction with another table. To further develop the relationship between the tables the @JoinColumns annotations can be used. The name attribute holds the foreign key identifier for the relationship – specifies the column name in the join table. The referenceColumnName attribute holds the name of the primary key identifier for this relationship [10] [11].



```
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "USER_EVENT",
    joinColumns = @JoinColumn(name = "EVENT_ID", referencedColumnName = "ID"),
    inverseJoinColumns = @JoinColumn(name = "USER_ID", referencedColumnName = "ID"))
private List<User> participants;
```

Figure 25: Example for joining tables with JPA

It is worth mentioning that every entity class has a helper class called "resource", also known as DTO class. In this case the Events entity has the helper class EventsResource. The class is also a POJO that contains all of the fields from the database table and two important methods that help using the entity object:

- toResource (EventsModel eventsModel) – This method gets an entity object of type EventsModel and converts it to resource.
- toModel () – this method is being called in a resource object and it converts the resource to an entity object.

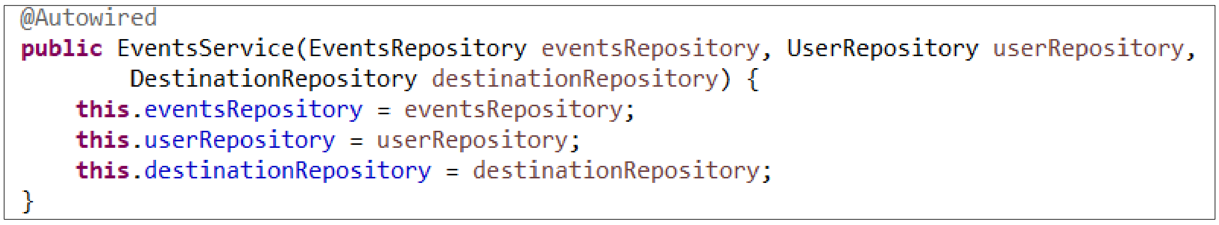
The idea behind the two methods is that they allow us to return json objects to the view – in our case that would be the angular application. If we try to make an entity object in a json object it is possible that we will get an error when it is being serialized. The other method – toModel (), helps us with the CRUD operations.

The service classes play another major role in the implementation of the project. The service classes contain all of the business logic of the application. In the example for the events the service class is called EventsService. In the Spring framework to mark a class as a service class you only need to add the @Service annotation above the class declaration. The EventsService contains the following methods:

- getAllEvents () – gets all of the events from the database.
- saveEvents (EventsResource eventsResource) – saves a new event in the database, can also be used for updates.
- registerForEvent (EventsResource eventsResource) – register a user for a specific event.
- getEventParticipants (String name) – returns a list of all of the users that are registered to a specific event.

- getEventsByUser (String name) – gets all of the events that a specific user has been registered to.
- getNextEventByUser (String name) - gets the next upcoming event of a specific user.
- getEventById (Long id) – gets an event by its id.
- getEventByName (String name) – gets an event by its name.
- deleteEventBy (Long id) – deletes an event by id.

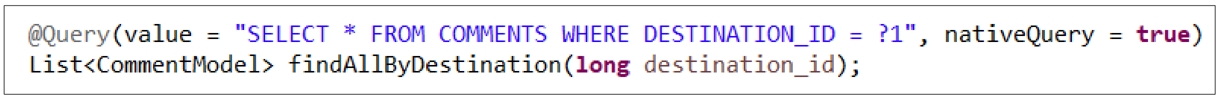
The service classes are one of the places that Spring shines. To achieve an easier access to the database a repository objects are being injected in the service class (see Figure 26).



```
@Autowired
public EventsService(EventsRepository eventsRepository, UserRepository userRepository,
        DestinationRepository destinationRepository) {
    this.eventsRepository = eventsRepository;
    this.userRepository = userRepository;
    this.destinationRepository = destinationRepository;
}
```

Figure 26: Constructor injection of repositories

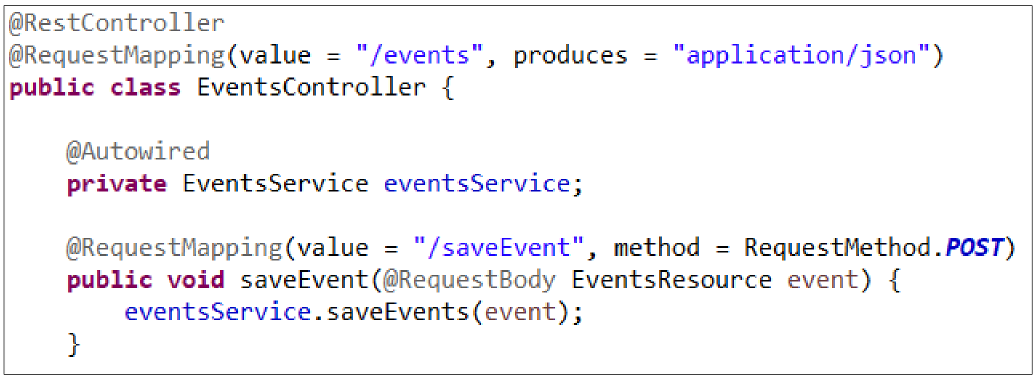
The repository interfaces are another point of interest of the back-end implementation. Being a spring boot application these interfaces must be marked as @Repository they must extend the CrudRepository. As the name implies the crud repository interface provides us with additional crud methods that we can user without additional implementation. The repository interface can work both with the crud repository methods as well as custom queries (see the figure below).



```
@Query(value = "SELECT * FROM COMMENTS WHERE DESTINATION_ID = ?1", nativeQuery = true)
List<CommentModel> findAllByDestination(long destination_id);
```

Figure 27: Custom query added in the repository class

As said before the controllers are responsible for processing user requests and building an appropriate model to return to the client. In our example the controller is name EventsController. The class itself contains a lot of methods and not all of them will be reviewed, but a class diagram of the class will be included in the thesis.



```
@RestController
@RequestMapping(value = "/events", produces = "application/json")
public class EventsController {

    @Autowired
    private EventsService eventsService;

    @RequestMapping(value = "/saveEvent", method = RequestMethod.POST)
    public void saveEvent(@RequestBody EventsResource event) {
        eventsService.saveEvents(event);
    }
```

Figure 28: Code snippet of events controller class.

To mark a class a controller it is only needed to annotate the class with @RestController or @Controller. The difference between the two is that the controller annotation only marks the class as a spring mvc controller and the restcontroller annotation adds the controller annotation as well as the @ResponseBody annotation that is used to automatically serialize the return object into JSON and passed back into the HttpResponse object. The figure above also shows 3 more annotation: the @Autowired annotation that is used to inject the service object, the @RequestMapping annotation that is used to map web requests to specific class or method. In this case we have 2 @RequestMapping annotation so if we want to make a call to the saveEvent() method we have to use the “/events/saveEvent” url the path; the annotation also points the type of the web request – in this case it is a POST request. The third annotation used is the @RequestBody annotation – it enables spring to try converting the content of the incoming request body to specific parameter object. The figure below represents a class diagram of the classes described above.

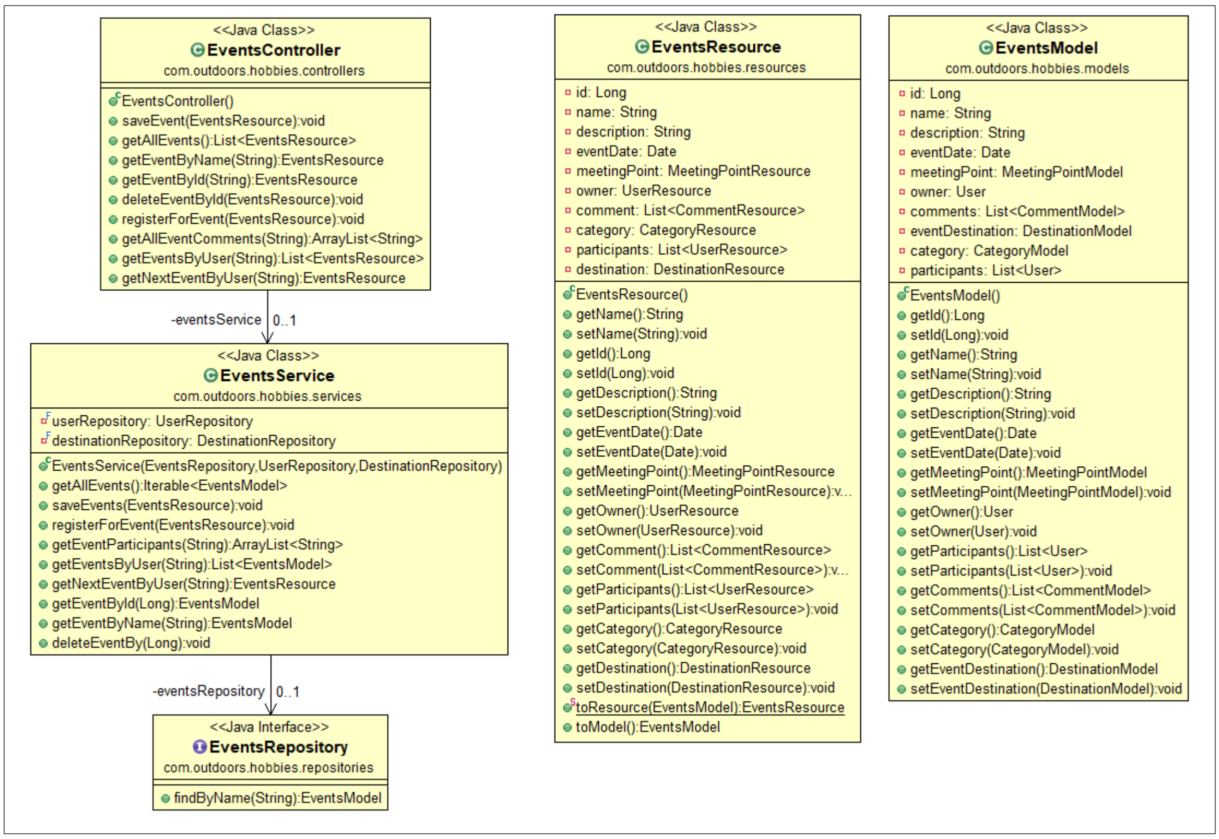

<<Java Class>>
EventsController
com.outdoors.hobbies.controllers
EventsController()
saveEvent(EventsResource):void
getAllEvents():List<EventsResource>
getEventByName(String):EventsResource
getEventById(String):EventsResource
deleteEventById(EventsResource):void
registerForEvent(EventsResource):void
getAllEventComments(String):ArrayList<String>
getEventsByUser(String):List<EventsResource>
getNextEventByUser(String):EventsResource
-eventsService
0..1
<<Java Class>>
EventsService
com.outdoors.hobbies.services
userRepository: UserRepository
destinationRepository: DestinationRepository
EventsService(EventsRepository,UserRepository,DestinationRepository)
getAllEvents():Iterable<EventsModel>
saveEvents(EventsResource):void
registerForEvent(EventsResource):void
getEventParticipants(String):ArrayList<String>
getEventsByUser(String):List<EventsModel>
getNextEventByUser(String):EventsResource
getEventById(Long):EventsModel
getEventByName(String):EventsModel
deleteEventBy(Long):void
-eventsRepository
0..1
<<Java Interface>>
EventsRepository
com.outdoors.hobbies.repositories
findByName(String):EventsModel
<<Java Class>>
EventsResource
com.outdoors.hobbies.resources
id: Long
name: String
description: String
eventDate: Date
meetingPoint: MeetingPointResource
owner: UserResource
comment: List<CommentResource>
category: CategoryResource
participants: List<UserResource>
destination: DestinationResource
EventsResource()
getName():String
setName(String):void
getId():Long
setId(Long):void
getDescription():String
setDescription(String):void
getEventDate():Date
setEventDate(Date):void
getMeetingPoint():MeetingPointResource
setMeetingPoint(MeetingPointResource):v...
getOwner():UserResource
setOwner(UserResource):void
getComment():List<CommentResource>
setComment(List<CommentResource>):v...
getParticipants():List<UserResource>
setParticipants(List<UserResource>):void
getCategory():CategoryResource
setCategory(CategoryResource):void
getDestination():DestinationResource
setDestination(DestinationResource):void
toResource(EventsModel):EventsResource
toModel():EventsModel
<<Java Class>>
EventsModel
com.outdoors.hobbies.models
id: Long
name: String
description: String
eventDate: Date
meetingPoint: MeetingPointModel
owner: User
comments: List<CommentModel>
eventDestination: DestinationModel
category: CategoryModel
participants: List<User>
EventsModel()
getId():Long
setId(Long):void
getName():String
setName(String):void
getDescription():String
setDescription(String):void
getEventDate():Date
setEventDate(Date):void
getMeetingPoint():MeetingPointModel
setMeetingPoint(MeetingPointModel):void
getOwner():User
setOwner(User):void
getParticipants():List<User>
setParticipants(List<User>):void
getComments():List<CommentModel>
setComments(List<CommentModel>):void
getCategory():CategoryModel
setCategory(CategoryModel):void
getEventDestination():DestinationModel
setEventDestination(DestinationModel):void

Figure 29: Class diagram of the events feature

3.4.2. Front-end implementation

For the different features of the site, the angular application has various components. Because it is not practical they will not be all reviewed. The feature that will be reviewed and described is the same as the back-end feature – the events. To achieve the events feature 3 components have been created:

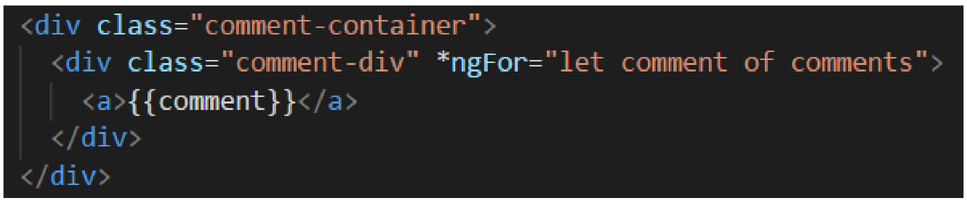
- Events-info – this component display all of the information about each of the events from the database. It also provides the functionalities for registering for events and writing comments. It also contains links to various important entities that are connected to the event – participants, destination.
- Events – this component displays three columns of events: the ones that are already over, the incoming events and the recommended events. Each event showed has short information about itself and is also a direct link to the event information page.
- New-events – as the name implies this component is used for creating new events.

Of those 3 components only the event-info will be reviewed. As described above the event info feature (folder) contains: the css file that contains all of the styles used in the template (the view part of the component), the html file, that is the view itself, the service file that contains the methods used to make calls to the back end project and of course the component file is also in this folder.

The Component file contains the business logic. For this specific page the business logic (i.e. viewing all of the information about the event, writing comments and joining an event) is achieved though the following methods:

- ngOnInit () – this method is being called when the page is first initialized. This method makes several calls to the back end application to get the information about the event, the event participants and the events comments.
- registerForEvent() – this method as the name implies is responsible for registering the user to the event.
- saveDestinationComment() – once again as the name implies the method is responsible for saving new comment for the event.

After the first initializing of the page and after a successful execution of the ngOnInit method, the necessary information has already been loaded in various objects in the component. Once this is done to access the needed information in the view part of the component tanks to Angular is very easy – using the various binding styles.



```
<div class="comment-container">
  <div class="comment-div" *ngFor="let comment of comments">
    <a>{{comment}}</a>
  </div>
</div>
```

Figure 30: Angular binding

On the figure above can also be seen one of the powerful tools of angular – the ngFor directive. This is used to easily iterate over a collection – in this case it will iterate over all of the comments for the event.

3.5.User guide for the module

The purpose of this section is to show how the web application that is the object of this thesis functions. The next figure shows the first thing that a user sees when he tries to access the web site.

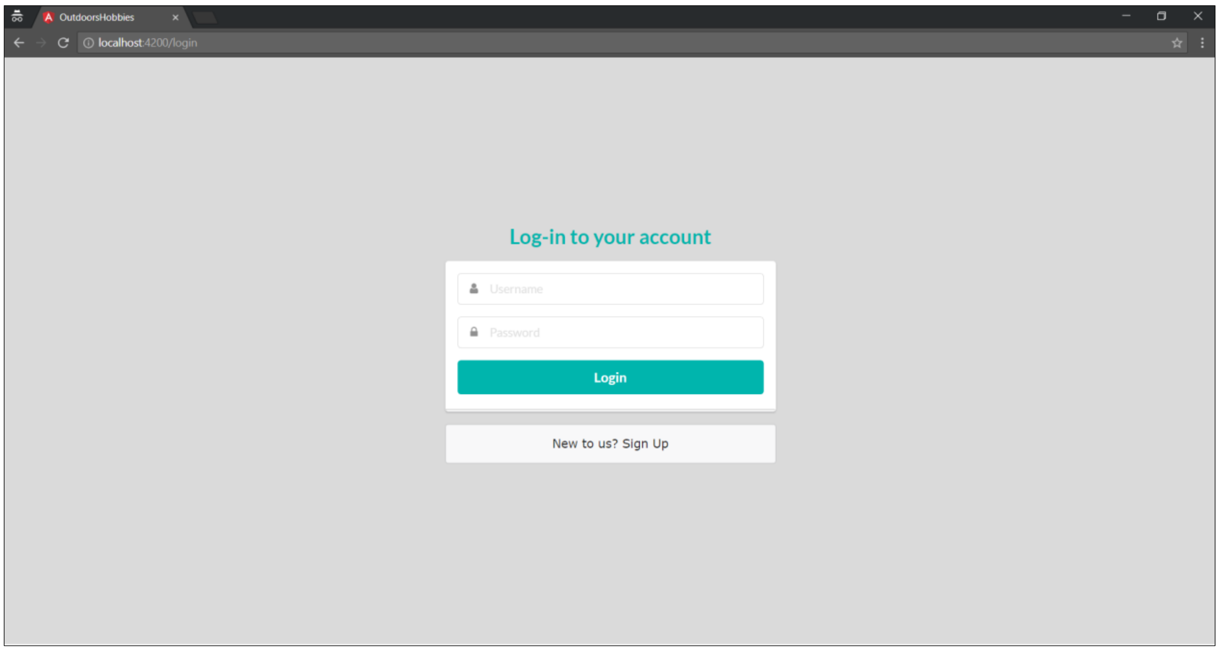


Figure 31: Log in page of the site

As seen on the figure above the user must log in order to use the site. Nothing in the site is usable without an account. If the user has an account he/she can simply type his or hers username or password and hit the Login button. If they are correct he/she will proceed to the site. If the user is new and has no account he or she can create it from the “Sign up” URL. When the link is clicked the user will be redirected to a different page (see figure below).

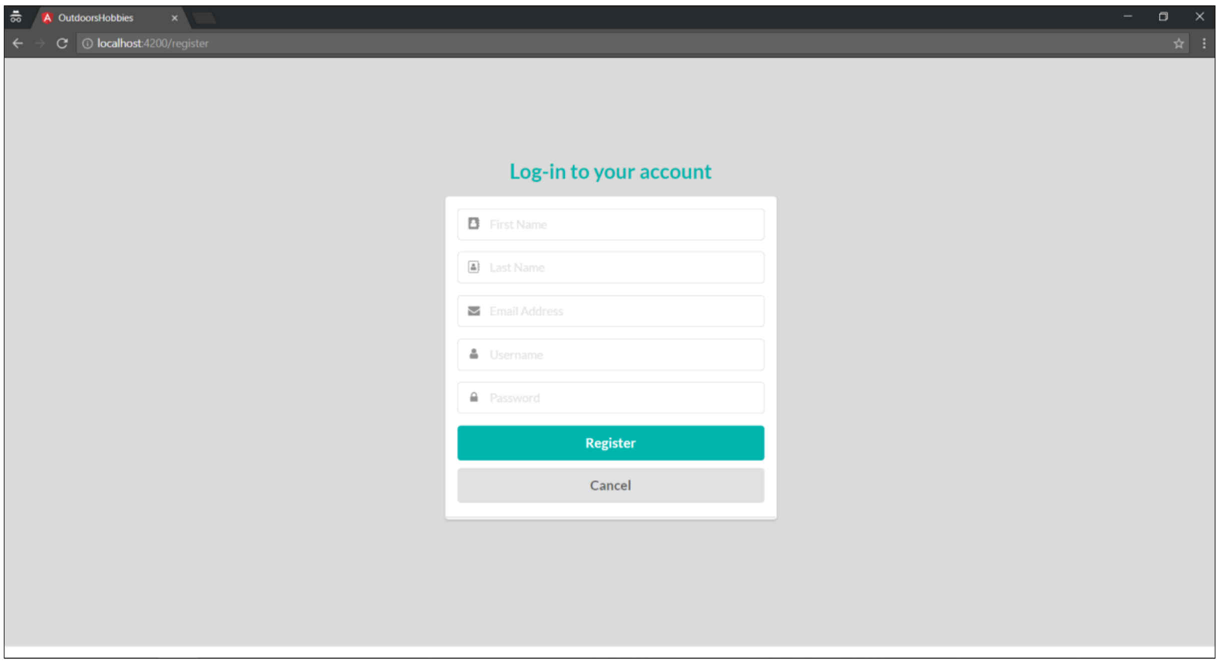
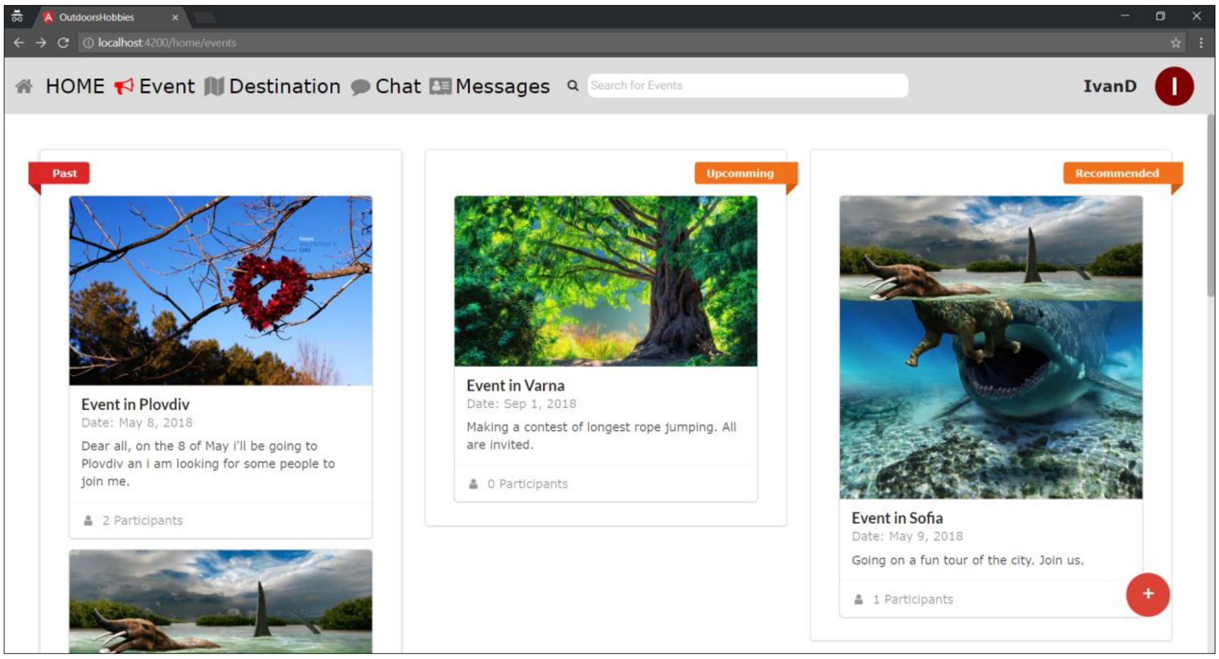


Figure 32: Registration page of the application

The registration page has 5 fields that are obligatory and cannot be left empty. This is ensured by a front-end validation that shows when a field is empty and the Register button is clicked. When the user has filled all of the fields and has pressed the Register button his or hers new account will be created and he will be redirected to the login page once again. If the user changes his mind mid-registration he or she can return to the login page by simply clicking on the Cancel button. After the user logs in the web application he/she will be redirected to the Events page of the site. The figure below shows exactly that. The first thing that is noticeable is the header. It contains 5 tabs, 1 search field and a drop down menu with additional functionalities – editing user personal information and login off.



Event in Plovdiv
Date: May 8, 2018
Dear all, on the 8 of May i'll be going to Plovdiv an i am looking for some people to join me.
2 Participants

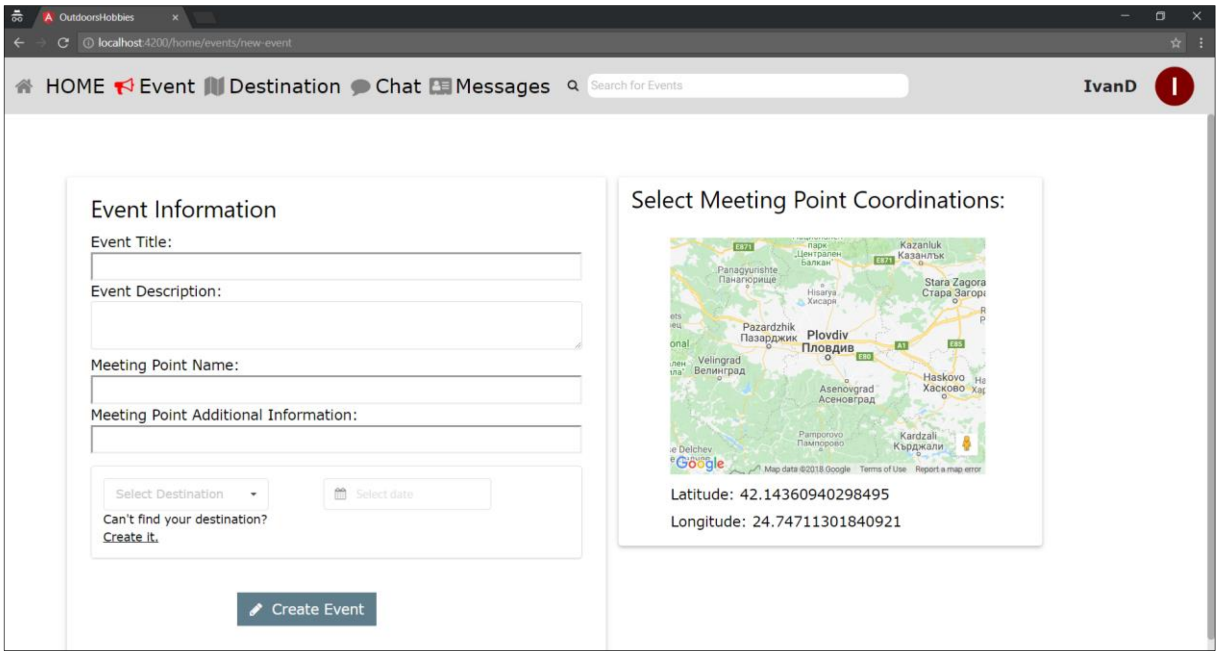
Event in Varna
Date: Sep 1, 2018
Making a contest of longest rope jumping. All are invited.
0 Participants

Event in Sofia
Date: May 9, 2018
Going on a fun tour of the city. Join us.
1 Participants

Figure 33: Events Page of the web site

When the user is interested in one of the events showed on the page he can simply click on one of the events he or she will be redirected to the corresponding events information page. If the user wants to create a new event he or she can achieve this through the round + button at the bottom right corner of the screen. This button redirects to the new event creation page.

On the figure bellow can be seen the above mention event creation page.

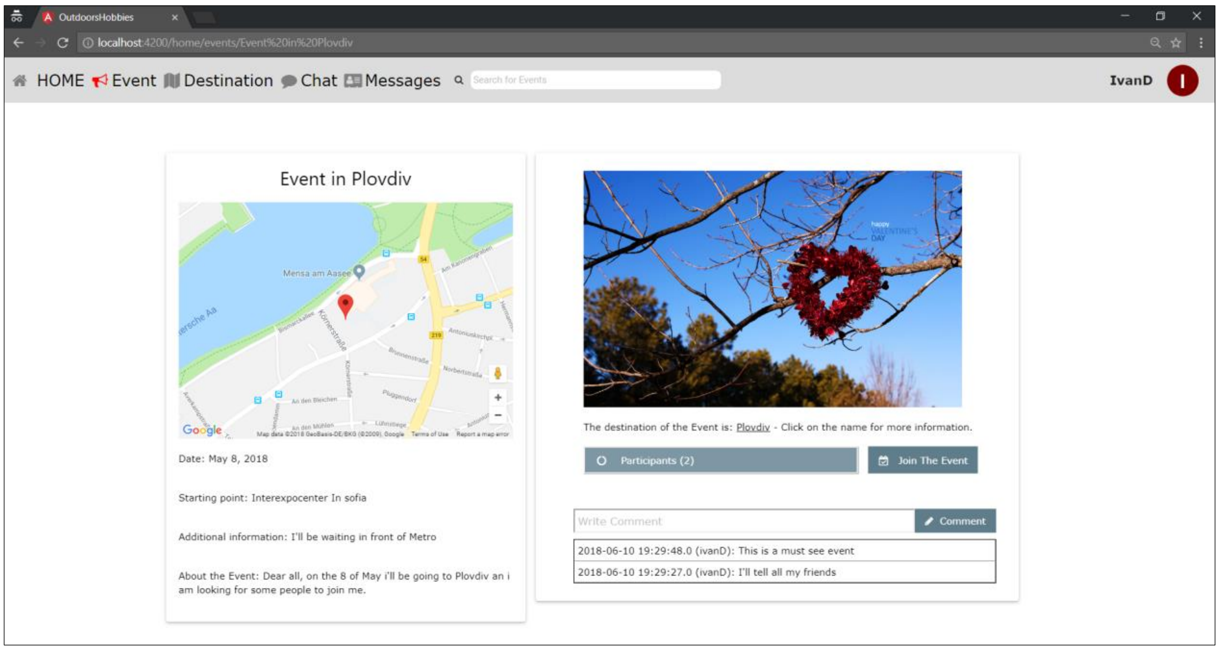


Latitude: 42.14360940298495
Longitude: 24.74711301840921

Figure 34: New event creation page

Here the user is prompt to input the necessary information of the event such as the title, description, meeting point information, destination and date. One of the perks here is the map, on which the user can directly select the coordinates of the meeting. Another perk here is the drop down menu of already created destinations from which the user can select destinations. But if the user cannot find the desired destination he/or she has a shortcut to creating it.

If the user takes a step back and returns to the events page and there is an event that he or she is interested in and wants to learn more about it, it is as simple as clicking on it to open the page from the figure below.


OutdoorsHobbies
localhost:4200/home/events/Event%20in%20Plovdiv
HOME Event Destination Chat Messages
Search for Events
IvanD
Event in Plovdiv

Date: May 8, 2018

Starting point: Interexpocenter In sofia

Additional information: I'll be waiting in front of Metro

About the Event: Dear all, on the 8 of May i'll be going to Plovdiv an i am looking for some people to join me.

The destination of the Event is: Plovdiv - Click on the name for more information.

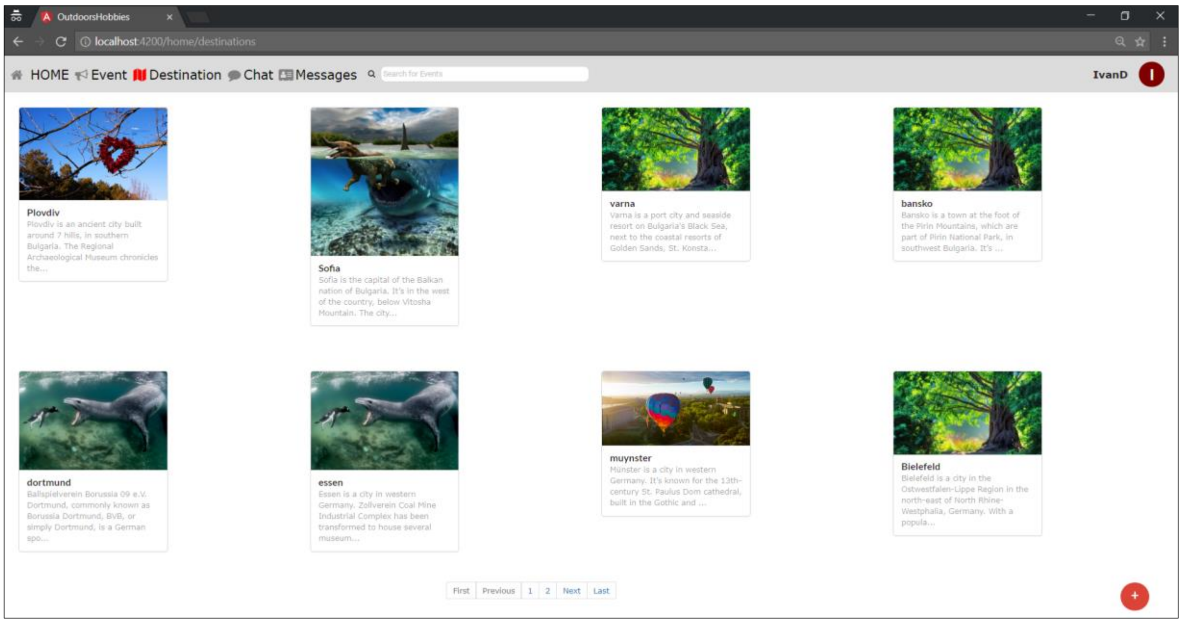
2018-06-10 19:29:48.0 (ivanD): This is a must see event

2018-06-10 19:29:27.0 (ivanD): I'll tell all my friends

Figure 35: Event information page

In this page the user can see all of the information about an event. From the description to the coordinates (nicely showed in a map) of the event, the number of the participants (conveniently stored in a drop down menu), the main photo of the destination and last but not least the comments of the said event. Here can be found and two functionalities for joining an event and writing comments.

If the user gets back to the header of the page and clicks on the Destination tab he or she will be met with a new page that contains all of the available destinations in the site (see figure below).


OutdoorsHobbies
localhost:4200/home/destinations
HOME Event Destination Chat Messages
Search for Events
IvanD

Plovdiv
Plovdiv is an ancient city built around 7 hills, in southern Bulgaria. The Regional Archaeological Museum chronicles the...

Sofia
Sofia is the capital of the Balkan nation of Bulgaria. It's in the west of the country, below Vitosha Mountain. The city...

varna
Varna is a port city and seaside resort on Bulgaria's Black Sea, next to the coastal resorts of Golden Sands, St. Konsta...

bansko
Bansko is a town at the foot of the Pirin Mountains, which are part of Pirin National Park, in southwest Bulgaria. It's ...

dortmund
Ballspielverein Borussia 09 e.V. Dortmund, commonly known as Borussia Dortmund, BVB, or simply Dortmund, is a German spo...

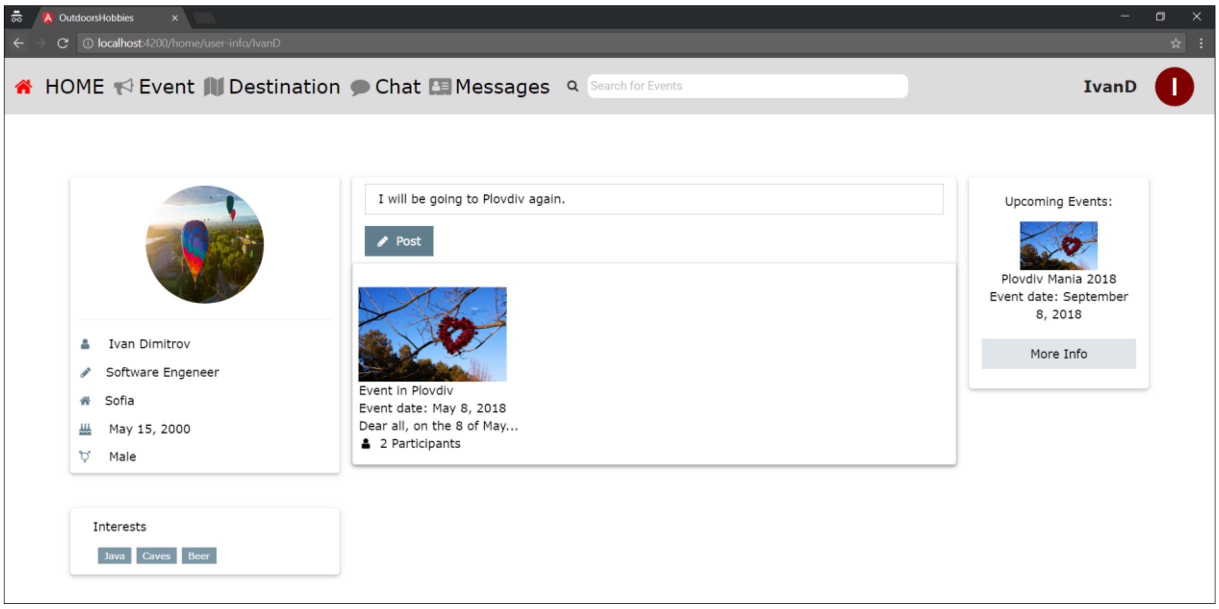
essen
Essen is a city in western Germany. Zollverein Coal Mine Industrial Complex has been transformed to house several museum...

muynster
Münster is a city in western Germany. It's known for the 13th-century St. Paulus Dom cathedral, built in the Gothic and ...

Bielefeld
Bielefeld is a city in the Ostwestfalen-Lippe Region in the north-east of North Rhine-Westphalia, Germany. With a popula...

Figure 36: Destinations page

This page has a clean and simple UI. It contains a list of all of the destinations nicely ordered in 2 rows of 4 destinations. At the bottom of the page there is a page navigator to easily scroll through the destinations. At the bottom right corner can be seen once again a creation Button that redirects to the Destination creation page. And if the user clicks on a specific destination the destination information page opens. These two pages are similar to the events information page and the event creation pages that's why no figures representing them will be added to the thesis. The only difference is that the destination creation page has a functionality to upload picture to the destination and the destination info page has a carousel where the user can review all of the pictures for the said destination and if he or she wants new pictures of the events can be added.



Ivan Dimitrov
Software Engeneer
Sofia
May 15, 2000
Male

I will be going to Plovdiv again.

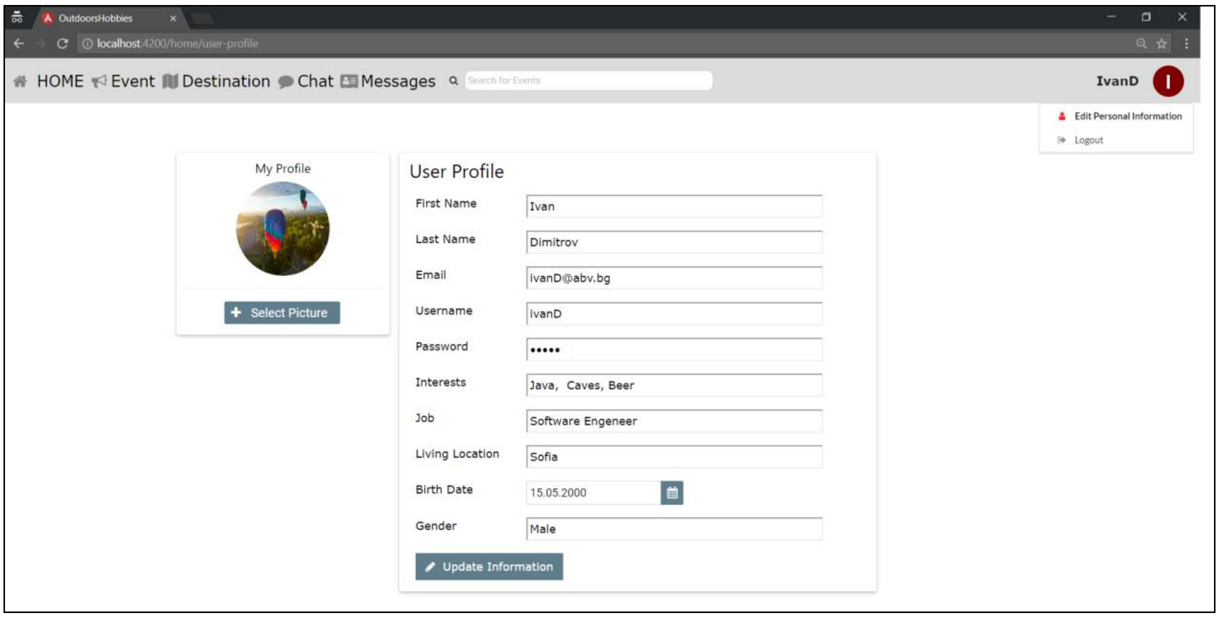
Event in Plovdiv
Event date: May 8, 2018
Dear all, on the 8 of May...
2 Participants

Upcoming Events:
Plovdiv Mania 2018
Event date: September 8, 2018

Figure 37: User information page

On the figure above is shown the user information page. This page is used both on the user that is logged in and a specific user selected beforehand (for example a user that has joined an event). The difference between the two is that when the logged user goes to his information page he or she can write a status and post it for others to see it. If the user goes to the profile information of another user, he or she will not be able to post statuses. The user information page shows the personal information of the user, his interests, profile picture and information about all of the events that the user has participated in and which is the next event that the user is going to visit. It is worth mentioning here that if there is no information to be displayed, the container that held the information will not be showed. What that means is that if the user hasn't written his job on the information panel in the left part of the screen, it will not show any job.

The same is valid for the profile picture but instead of not showing anything a circle with initials will be shown – similar to the one at the top right corner.



User Profile

First Name	Ivan
Last Name	Dimitrov
Email	IvanD@abv.bg
Username	IvanD
Password	•••••
Interests	Java, Caves, Beer
Job	Software Engeneer
Living Location	Sofia
Birth Date	15.05.2000
Gender	Male

Figure 38 : User profile page

The user profile page is the page where the user can edit his or hers personal information as well as change password or username. The user can also upload profile picture from this page. To enter this page the user must click on the drop down menu from the header – the circular button with the initials of the user at the top right corner.

4. Conclusion

This section has the goal to analyze the development process and draw conclusions about the results:

4.1. Summary of the results

The goal of the master thesis is the creation of an easy to use social network that enables the users of the site to create events for their favorite hobbies. This goal has been achieved through the completion of three main tasks:

- To achieve the creation of the web application, a place to store the information is needed. That is done by designing and implementing a database. Several tables have been created and the appropriate relationships between them have been created.
- A nice clean Spring Boot project has been created for the back end of the social network. The code can be easily maintained and reused as well as extended in newer functionalities.
- An Angular 2/5 project has been created for the front side of the social network. The user interface of the site has been kept as cleaner and simpler as it can be so the use of the site to be easy and intuitive.

4.2. Ideas for future research and development

The social network can be further developed to improve the user experience of the site. One of the new functionalities that can be added is an administrator account that can manage more efficiently the content of the web application. Another great functionality that can be added is the ability the user to connect to other user in the form of a friendship request. This will allow them easier access to their friends accounts and a shortcut for writing messages. Another great feature that can be added is a direct one to one messaging or chatting.

Literary sources

1. Java Platform, Enterprise Edition (Java EE), http://www.oracle.com/technetwork/java/javaee/overview/index.html
2. Mandala, R., JSP, http://java-brain.blogspot.com/2011/10/jsp.html May 7, 2006
3. JavaServer Pages Overview, http://www.oracle.com/technetwork/java/overview-138580.html
4. Java Platform, Enterprise Edition (Java EE), http://www.oracle.com/technetwork/java/javaee/overview/index.html
5. CSS3, https://developer.mozilla.org/en/docs/Web/CSS/CSS3
6. Bootstrap Get Started, http://www.w3schools.com/bootstrap/bootstrap_get_started.asp
7. Semantic UI, https://semantic-ui.com/
8. Hibernate ORM, http://hibernate.org/orm/
9. Application Server 8 Developer's Guide, Chapter 3. Assembling and Deploying J2EE Applications, http://docs.oracle.com/cd/E19253-01/817-6087/dgdeploy.html
10. Hibernate - Persistent Class, http://www.tutorialspoint.com/hibernate/hibernate_persistent_classes.htm
11. Bernard, E., Chapter 2. Mapping Entities, https://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html, September 15, 2010
12. Dalling, T., Model View Controller Explained, http://www.tomdalling.com/blog/software-design/model-view-controller-explained/ May 31, 2009
13. Design Patterns MVC Pattern, http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
14. jQuery – Overview, http://www.tutorialspoint.com/jquery/jquery-overview.htm
15. MySQL, https://www.tutorialspoint.com/mysql/mysql-introduction.htm
16. Angular 2/5, https://angular.io/guide/quickstart
17. Spring Modules figure , https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch01s02.html

18. Anuglar 8 major block , https://qph.ec.quoracdn.net/main-qimg-feec1638f923cdce67e5935f3d626f18-c

Attachments

Full code can be found here:

https://github.com/LHristozov/hobbies-FE

https://github.com/LHristozov/hobbies-BE

```
package com.outdoors.hobbies.config;

import java.util.Arrays;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

import
org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfi
gurerAdapter;

import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServe
r;

import
org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpoi
ntsConfigurer;

import
org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurity
Configurer;

import org.springframework.security.oauth2.provider.token.TokenEnhancerChain;

import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;


@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {


        @Value("${security.jwt.client-id}")
        private String clientId;


        @Value("${security.jwt.client-secret}")
        private String clientSecret;


        @Value("${security.jwt.grant-type}")
        private String grantType;
        @Value("${security.jwt.scope-read}")
        private String scopeRead;
        @Value("${security.jwt.scope-write}")
        private String scopeWrite = "write";
        @Value("${security.jwt.resource-ids}")
        private String resourceIds;
        @Autowired
        private TokenStore tokenStore;
        @Autowired
        private DataSource dataSource;
        @Autowired
        private JwtAccessTokenConverter accessTokenConverter;
        @Autowired
```

```
	private AuthenticationManager authenticationManager;

	@Override

	public void configure(ClientDetailsServiceConfigurer configurer) throws Exception {


	configurer.jdbc(dataSource).withClient(clientId).secret(clientSecret).authorizedGrantTypes(gr
antType)

				.scopes(scopeRead, scopeWrite).resourceIds(resourceIds);

	}

	@Override

	public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

		TokenEnhancerChain enhancerChain = new TokenEnhancerChain();

		enhancerChain.setTokenEnhancers(Arrays.asList(accessTokenConverter));


	endpoints.tokenStore(tokenStore).accessTokenConverter(accessTokenConverter).tokenEnha
ncer(enhancerChain)

				.authenticationManager(authenticationManager);

	}

	@Override

	public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {

		oauthServer.allowFormAuthenticationForClients();

	}

}

package com.outdoors.hobbies.config;

import org.springframework.boot.web.servlet.FilterRegistrationBean;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.Ordered;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
@Configuration
public class CorsConfig {
  @Bean
  public FilterRegistrationBean corsFilter() {
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowCredentials(true);
    config.addAllowedOrigin("*");
    config.addAllowedHeader("*");
    config.addAllowedMethod("*");
    source.registerCorsConfiguration("/**", config);
    FilterRegistrationBean bean = new FilterRegistrationBean(new CorsFilter(source));
    bean.setOrder(Ordered.HIGHEST_PRECEDENCE);
    return bean;
  }
}
package com.outdoors.hobbies.config;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.encoding.ShaPasswordEncoder;
```

```
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuil
der;

import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.oauth2.provider.token.DefaultTokenServices;

import org.springframework.security.oauth2.provider.token.TokenStore;

import org.springframework.security.oauth2.provider.token.store.JdbcTokenStore;

import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Value("${security.signing-key}")

  private String signingKey;

  @Value("${security.encoding-strength}")

  private Integer encodingStrength;

  @Value("${security.security-realm}")

  private String securityRealm;

  @Autowired

  private UserDetailsService userDetailsService;

        @Autowired
```

```
          private DataSource dataSource;
@Bean
@Override
protected AuthenticationManager authenticationManager() throws Exception {
  return super.authenticationManager();
}
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
  auth.userDetailsService(userDetailsService);
//         .passwordEncoder(new ShaPasswordEncoder(encodingStrength));
}
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
          .sessionManagement()
          .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
          .and()
          .httpBasic()
          .realmName(securityRealm)
          .and()
          .csrf()
          .disable();
}
@Override
public void configure(WebSecurity web) throws Exception {
 web.ignoring()
   .antMatchers(HttpMethod.OPTIONS);
  }
 @Bean
  public JwtAccessTokenConverter accessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
    converter.setSigningKey(signingKey);
    return converter;
  }
  @Bean
  public TokenStore tokenStore() {
    return new JdbcTokenStore(dataSource);
  }
  @Bean
  @Primary //Making this primary to avoid any accidental duplication with another token service
instance of the same name
  public DefaultTokenServices tokenServices() {
    DefaultTokenServices defaultTokenServices = new DefaultTokenServices();
    defaultTokenServices.setTokenStore(tokenStore());
    defaultTokenServices.setSupportRefreshToken(true);
    return defaultTokenServices;
  }
}
package com.outdoors.hobbies.controllers;
import org.springframework.web.bind.annotation.RestController;
import com.outdoors.hobbies.resources.DestinationResource;
import com.outdoors.hobbies.resources.EventsResource;
import com.outdoors.hobbies.services.EventsService;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
@RestController
@RequestMapping(value = "/events", produces = "application/json")
public class EventsController {
	@Autowired
	private EventsService eventsService;
	@RequestMapping(value = "/saveEvent", method = RequestMethod.POST)
	public void saveEvent(@RequestBody EventsResource event) {
		eventsService.saveEvents(event);
	}
	@RequestMapping(method = RequestMethod.GET)
	public List<EventsResource> getAllEvents() {
		List<EventsResource> events = new ArrayList<>();
		eventsService.getAllEvents().forEach(d -> {
			events.add(EventsResource.toResource(d));
		});
		return events;
	}
	@RequestMapping(value = "event-info/{name}", method = RequestMethod.GET)
	public EventsResource getEventByName(@PathVariable String name) {
		return EventsResource.toResource(eventsService.getEventByName(name));
	}
	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
```

```
public EventsResource getEventById(@PathVariable String id) {
        return EventsResource.toResource(eventsService.getEventById(Long.getLong(id)));
}
@RequestMapping(method = RequestMethod.DELETE)
public void deleteEventById(@RequestBody EventsResource event) {
        eventsService.deleteEventBy(event.getId());
}
@RequestMapping(value = "/registerForEvent", method = RequestMethod.POST)
public void registerForEvent(@RequestBody EventsResource event) {
        eventsService.registerForEvent(event);
}
@RequestMapping(value = "/getEventParticipants/{name}", method = RequestMethod.GET)
public ArrayList<String> getAllEventComments(@PathVariable String name) {
        return eventsService.getEventParticipants(name);
}
@RequestMapping(value = "/getEventsByUser/{name}", method = RequestMethod.GET)
public List<EventsResource> getEventsByUser(@PathVariable String name) {
        List<EventsResource> userEvents = new ArrayList<>();
        eventsService.getEventsByUser(name).forEach(d -> {
                userEvents.add(EventsResource.toResource(d));
        });
        return userEvents;
}
@RequestMapping(value = "/getNextEventByUser/{name}", method = RequestMethod.GET)
public EventsResource getNextEventByUser(@PathVariable String name) {
        return eventsService.getNextEventByUser(name);
}
```

```
	}
package com.outdoors.hobbies.models;
import java.util.Date;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
@Table(name = "EVENTS")
public class EventsModel {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	@NotNull
	private String name;
```

```
        @NotNull
        @Size(min = 1, max = 4000)
        @Column(name = "description")
        private String description;
        @NotNull
        private Date eventDate;
        @OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "meeting_point_id")
        private MeetingPointModel meetingPoint;
        @ManyToOne()
@JoinColumn(name = "user_id")
        private User owner;
@OneToMany(mappedBy = "event", cascade = CascadeType.ALL)
private List<CommentModel> comments;
@ManyToOne()
@JoinColumn(name = "destination_id")
private DestinationModel eventDestination;
@ManyToOne()
@JoinColumn(name = "category_id")
private CategoryModel category;
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "USER_EVENT",
        joinColumns = @JoinColumn(name = "EVENT_ID", referencedColumnName = "ID"),
        inverseJoinColumns = @JoinColumn(name = "USER_ID", referencedColumnName = "ID"))
        private List<User> participants;
        public Long getId() {
                return id;
```

```
}

public void setId(Long id) {

        this.id = id;

}

public String getName() {

        return name;

}

public void setName(String name) {

        this.name = name;

}

public String getDescription() {

        return description;

}

public void setDescription(String description) {

        this.description = description;

}

public Date getEventDate() {

        return eventDate;

}

public void setEventDate(Date eventDate) {

        this.eventDate = eventDate;

}

public MeetingPointModel getMeetingPoint() {

        return meetingPoint;

}

public void setMeetingPoint(MeetingPointModel meetingPoint) {

        this.meetingPoint = meetingPoint;
```

```
}

public User getOwner() {

        return owner;

}

public void setOwner(User owner) {

        this.owner = owner;

}

public List<User> getParticipants() {

        return participants;

}

public void setParticipants(List<User> participants) {

        this.participants = participants;

}

public List<CommentModel> getComments() {

        return comments;

}

public void setComments(List<CommentModel> comments) {

        this.comments = comments;

}

public CategoryModel getCategory() {

        return category;

}

public void setCategory(CategoryModel category) {

        this.category = category;

}

public DestinationModel getEventDestination() {

        return eventDestination;
```

```
	}
	public void setEventDestination(DestinationModel eventDestination) {
		this.eventDestination = eventDestination;
	}
}
package com.outdoors.hobbies.resources;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;
import com.outdoors.hobbies.models.DestinationModel;
import com.outdoors.hobbies.models.EventsModel;
import com.outdoors.hobbies.models.MeetingPointModel;
import com.outdoors.hobbies.models.User;
public class EventsResource {
	private Long id;
	private String name;
	private String description;
	private Date eventDate;
	private MeetingPointResource meetingPoint;
	private UserResource owner;
	private List<CommentResource> comment;
	private CategoryResource category;
	private List<UserResource> participants;
	private DestinationResource destination;
	public String getName() {
		return name;
```

```
}
public void setName(String name) {
            this.name = name;
}           public Long getId() {
            return id;
}
public void setId(Long id) {
            this.id = id;
}
public String getDescription() {
            return description;
}
public void setDescription(String description) {
            this.description = description;
}
public Date getEventDate() {
            return eventDate;
}
public void setEventDate(Date eventDate) {
            this.eventDate = eventDate;
}
public MeetingPointResource getMeetingPoint() {
            return meetingPoint;
}
public void setMeetingPoint(MeetingPointResource meetingPoint) {
            this.meetingPoint = meetingPoint;
}
```

```
public UserResource getOwner() {
		return owner;
}
public void setOwner(UserResource owner) {
		this.owner = owner;
}
public List<CommentResource> getComment() {
		return comment;
}
public void setComment(List<CommentResource> comment) {
		this.comment = comment;
}
public List<UserResource> getParticipants() {
		return participants;
}
public void setParticipants(List<UserResource> participants) {
		this.participants = participants;
}
public CategoryResource getCategory() {
		return category;
}
public void setCategory(CategoryResource category) {
		this.category = category;
}
public DestinationResource getDestination() {
		return destination;
}
```

```
	public void setDestination(DestinationResource destination) {

		this.destination = destination;

	}

	public static EventsResource toResource(EventsModel eventsModel) {

		EventsResource eventsResource = new EventsResource();

		eventsResource.setDescription(eventsModel.getDescription());

		eventsResource.setName(eventsModel.getName());

		eventsResource.setId(eventsModel.getId());

		// if (eventsModel.getComments() != null) {

		// eventsResource.setComment(

		//
eventsModel.getComments().stream().map(CommentResource::toResource).collect(Collectors.toList
()));

		// }

		if (eventsModel.getCategory() != null) {
	eventsResource.setCategory(CategoryResource.toResource(eventsModel.getCategory()));

		}

		if (eventsModel.getEventDate() == null ||
eventsModel.getEventDate().toString().equals("0000-00-00 00:00:00")) {

			// TODO: its possible this line here to mess up dates!

			eventsResource.setEventDate(new Date());

		} else {

			eventsResource.setEventDate(eventsModel.getEventDate());

		}


	eventsResource.setMeetingPoint(MeetingPointResource.toResource(eventsModel.getMeeti
ngPoint()));
	eventsResource.setOwner(UserResource.toResource(eventsModel.getOwner()));

		eventsResource.getOwner().setUserInfoResource(null);

	eventsResource.setDestination(DestinationResource.toResource(eventsModel.getEventDesti
nation()));
```

```
		if (eventsModel.getParticipants() != null) {
			List<UserResource> participants = new ArrayList<UserResource>();

			for(User u : eventsModel.getParticipants()) {
				UserResource us = UserResource.toResource(u);
				us.setUserInfoResource(null);
				participants.add(us);
			}
			eventsResource.setParticipants(participants);


//			eventsResource.setParticipants(
//	eventsModel.getParticipants().stream().map(UserResource::toResource).collect(Collectors.to
List()));
		}


		return eventsResource;
	}
	public EventsModel toModel() {
		EventsModel eventsModel = new EventsModel();
		eventsModel.setDescription(description);
		eventsModel.setName(name);
		eventsModel.setEventDestination(destination.toModel());
		eventsModel.setOwner(owner.toModel());
		eventsModel.setEventDate(eventDate);
		eventsModel.setId(id);
		if (meetingPoint != null) {
			eventsModel.setMeetingPoint(meetingPoint.toModel());
		}
```

```
			if(participants != null && !participants.isEmpty()) {
				ArrayList<User> userList = new ArrayList<User>();
				for (UserResource participant : participants) {
					userList.add(participant.toModel());
				}
				eventsModel.setParticipants(userList);
			}
			// TODO: add all fields
			return eventsModel;
		}
	}
package com.outdoors.hobbies.services;


import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.outdoors.hobbies.models.CommentModel;
import com.outdoors.hobbies.models.EventsModel;
import com.outdoors.hobbies.models.User;
import com.outdoors.hobbies.models.UserInfoModel;
import com.outdoors.hobbies.repositories.DestinationRepository;
import com.outdoors.hobbies.repositories.EventsRepository;
import com.outdoors.hobbies.repositories.UserRepository;
import com.outdoors.hobbies.resources.DestinationResource;
import com.outdoors.hobbies.resources.EventsResource;

import com.outdoors.hobbies.resources.UserResource;

@Service

public class EventsService {

	private final EventsRepository eventsRepository;

	private final UserRepository userRepository;

	private final DestinationRepository destinationRepository;

	@Autowired

	public EventsService(EventsRepository eventsRepository, UserRepository userRepository,

			DestinationRepository destinationRepository) {

		this.eventsRepository = eventsRepository;

		this.userRepository = userRepository;

		this.destinationRepository = destinationRepository;

	}

	public Iterable<EventsModel> getAllEvents() {

		return eventsRepository.findAll();

	}

	public void saveEvents(EventsResource eventsResource) {

		if (eventsRepository == null) {

			return;

		}			eventsResource.setOwner(
	UserResource.toResource(userRepository.findByUsername(eventsResource.getOwner().getU
sername())));

		User user =
userRepository.findByUsername(eventsResource.getOwner().getUsername());

		eventsResource.setOwner(UserResource.toResource(user));

		UserInfoModel uim = eventsResource.getOwner().getUserInfo();

		uim.setUser(user);

		eventsResource.getOwner().setUserInfoResource(uim);
		eventsResource.setDestination(DestinationResource

	.toResource(destinationRepository.findByName(eventsResource.getDestination().getName())
));
		eventsRepository.save(eventsResource.toModel());
	}
	public void registerForEvent(EventsResource eventsResource) {
		if (eventsRepository == null) {
			return;
		}
		ArrayList<UserResource> userList = new ArrayList<UserResource>();

	userList.add(UserResource.toResource((userRepository.findByUsername(eventsResource.get
Owner().getUsername()))));
		for (UserResource participant : eventsResource.getParticipants()) {
			userList.add(participant);


		}
		eventsResource.setParticipants(userList);
		eventsResource.setOwner(
	UserResource.toResource(eventsRepository.findOne(eventsResource.getId()).getOwner()));
		eventsResource.setDestination(DestinationResource

	.toResource(destinationRepository.findByName(eventsResource.getDestination().getName())
));
		eventsResource.setMeetingPoint(null);
		eventsRepository.save(eventsResource.toModel());
	}
	public ArrayList<String> getEventParticipants(String name) {
		EventsModel event = eventsRepository.findByName(name);
```

```
        List<User> participants =  event.getParticipants();

        ArrayList<String> participantsNames = new ArrayList<String>();

        for (User participant : participants) {

                participantsNames.add(participant.getUsername());

        }

        return participantsNames;


}

public List<EventsModel> getEventsByUser(String name) {

        List<EventsModel> allEvents = (List<EventsModel>) eventsRepository.findAll();

        User user = userRepository.findByUsername(name);

        List<EventsModel> userEvents = new ArrayList<EventsModel>();

        for(EventsModel em : allEvents) {

                if(em.getParticipants().contains(user)) {

                        userEvents.add(em);

                }

        }

        return userEvents;

}

public EventsResource getNextEventByUser(String name) {

        List<EventsModel> events =  getEventsByUser(name);

        EventsModel currentEvent = null;

        for(EventsModel event : events) {

                if(currentEvent == null) {

                        currentEvent = event;

                        continue;

                }
```

```
                    if(event.getEventDate().compareTo(currentEvent.getEventDate()) < 0) {
                        currentEvent = event;
                    }


                }
                return EventsResource.toResource(currentEvent);
        }
        public EventsModel getEventById(Long id) {
                return eventsRepository.findOne(id);
        }
        public EventsModel getEventByName(String name) {
                return eventsRepository.findByName(name);
        }


        public void deleteEventBy(Long id) {
                eventsRepository.delete(id);
        }
}
/* .event-comments .ui-scrollpanel-wrapper {
  border-right: 9px solid #f4f4f4;
}
.event-comments .ui-scrollpanel-bar {
  background-color: #1976d2;
  opacity: 1;
  transition: background-color .3s;
}
.event-comments .ui-scrollpanel-bar:hover {
  background-color: #135ba1;
}
.chat-container{
  height: 180px;
  width: 65vw;
}
.chat{
  text-align: left;
  margin: 0 auto;
  width: 50%;
  height: 100%;
  border: 1px solid grey;
}
.footer{
  background: #cccccc;
  margin: 0 auto;
  position: absolute;
  bottom: 0;
  height: 10vh;
  width: 100vw;
}
.send-button{
  margin: 0 auto;
  height: 70%;
  width: 50%;
}
input{
  float: right;
  height: 30px;
  border-radius: 15px;
  font-size: 1.5em;
}
.send{
  float: right;
  background: url("../../assets/images/send.png");
  background-size: contain;
  background-repeat: no-repeat;
  height: 30px;
  width: 30px;
  margin-left: 5px;
} */
:host {
  height: 100%;
  width: 98%;
}
.destination-comments .ui-scrollpanel-wrapper {
  border-right: 9px solid #f4f4f4;
}
.destination-comments .ui-scrollpanel-bar {
  background-color: #1976d2;
  opacity: 1;
  transition: background-color .3s;
}
.destination-comments .ui-scrollpanel-bar:hover {
  background-color: #135ba1;
}
agm-map {
  height: 375px;
  width: 530px;
}
.chat-container{
  height: 180px;
  width: 65vw;
}
.chat{
  text-align: left;
  margin: 0 auto;
  width: 50%;
  height: 100%;
  border: 1px solid grey;
}
.footer{
  background: #cccccc;
  margin: 0 auto;
  position: absolute;
  bottom: 0;
  height: 10vh;
  width: 100vw;
}
.send-button{
  margin: 0 auto;
  height: 70%;
  width: 50%;
}
input{
  float: right;
  height: 30px;
  border-radius: 15px;
  font-size: 1.5em;
}
.send{
  float: right;
  background: url("../../assets/images/send.png");
  background-size: contain;
  background-repeat: no-repeat;
  height: 30px;
  width: 30px;
  margin-left: 5px;
}
.imgStyle{
  /* cursor:pointer;
  display: block;
  width: 100%;
  height: auto; */
  display: block;
  max-width:230px;
  max-height:95px;
  width: auto;
  height: auto;
}

.width{
  width: auto!important;
  /* margin-right: 4em; */
}

.center {
  margin: auto;
  width: 32%;
  padding: 10px;
}
.padding {
  padding: 20px;
}
.paddig-top-bottom {
  padding-top: 20px;
  padding-bottom: 20px;
}
.mybutton {
  margin: 2px;
  position:relative;
  text-decoration: none;
  display:inline-block;
  left: 75%;
  padding: 13px !important;
  z-index: 15;
  text-align: center;
  font-size: 26px;
  line-height: 26px;
  color: white;
  border-radius: 50px;
  border-color: #08806c;
  left: 30px;
  top: 0.5em;
  background-color: #db4437;
  width: 56px;
  height: 56px;
  transition-property: background-color,box-shadow;
}
.send-container {
  display: flex;
  padding-bottom: 10px;
  width: 100%;
}
.input-field{
  float: right!important;
  height: 38px!important;
  border-radius: 0px!important;
  font-size: 17px!important;
  width: 100%!important;
  padding-left: 5px;
}
```

```
.comment-div{
  border-style: outset;
  border-width: 1px;
  padding: 5px;
  margin: 13  px;
}
.comment-container{
  border-style: solid;
  border-width: 1px;
}
.participants-padding-container{
padding-top: 5px;
padding-bottom: 5px;
padding-left: 0px;
padding-right: 0px;
}
.user-style{
  border: solid 1px gray;
  padding: 1px;
  margin-right: 3px;
  margin-left: 4px;
}
<div class="w3-container w3-content" style="max-width:1400px;margin-top:80px">
  <!-- The Grid -->
  <div class="w3-row">

    <!-- Left Column -->
```

```
<div class="w3-col m5">
  <!-- Profile -->
  <div class="w3-card w3-round w3-white">
    <div class="w3-container padding">
      <h2 style="text-align: center;padding-bottom: 10px;">{{event.name}}</h2>
      <div class="container height">
        <div class="map-padding ">
          <agm-map [latitude]="lat" [longitude]="lng" [zoom]="zoom">
            <agm-marker [latitude]="lat" [longitude]="lng"></agm-marker>
          </agm-map>
        </div>
        <div class="form-group paddig-top-bottom">
          <a>Date:</a>
          <a>{{event.eventDate | date}}</a>
        </div>
        <div *ngIf="event.meetingPoint.name" class="form-group paddig-top-bottom">
          <a>Starting point:</a>
          <a>{{event.meetingPoint.name}}</a>
        </div>
        <div *ngIf="event.meetingPoint.description" class="form-group paddig-top-bottom">
          <a>Additional information:</a>
          <a>{{event.meetingPoint.description}}</a>
        </div>
        <div class="form-group paddig-top-bottom">
          <a>About the Event:</a>
          <a>{{event.description}}</a>
        </div>
        </div>
      </div>
    </div>
    <!-- End Left Column -->
  </div>
  <!-- Middle Column -->
  <div class="w3-col m7">
    <div class="w3-row-padding">
      <div class="w3-col m12">
        <div class="w3-card w3-round w3-white">
          <div class="w3-container w3-padding">
            <div class="col-md-7 padding" style="padding-left: 44px;">
              <div class="w3-container" id="destination">
                <div class="w3-display-container mySlides">
                  <img [src]=[currentImg] style="display: block;max-width: 600px;max-height: 400px;width:
auto;height: auto;">
                </div>
                <div class="form-group paddig-top-bottom">
                  <a>The destination of the Event is:</a>
                  <a style="text-decoration: underline;" [routerLink]="['/home/destinations',
event.destination.name]" routerLinkActive="active">{{event.destination.name}}</a>
                  <a>- Click on the name for more information.</a>
                </div>
                <div style="display: flex;">
                  <div style="width: 70%;height: 100%;border-style: ridge;">
                    <button onclick="myFunction('Demo1')" class="w3-button w3-block w3-theme-l1 w3-
left-align">
                      <i class="fa fa-circle-o-notch fa-fw w3-margin-right"></i> Participants
({{participants.length}})</button>
```

```
          <div id="Demo1" class="w3-hide w3-container participants-padding-container">
           <a [routerLink]="['/home/user-info', user]" class="user-style" routerLinkActive="active"
*ngFor="let user of participants">{{user}}</a>
          </div>
         </div>
         <div style="width: 30%;margin-left: 15px;">
          <button style="height: 43px;" [disabled]="loading" class="w3-button w3-theme"
(click)="registerForEvent()">
           <i class="calendar check icon"></i> &nbsp;Join The Event
          </button>
         </div>
         <!-- <button pButton type="button" (click)="registerForEvent()" lable="Register for
Event">Register for Event</button> -->
        </div>
       </div>
       <div style="padding-top: 55px;">
        <div class="send-button send-container">
         <!-- <div class="send" (click)="saveDestinationComment(input.value)"></div> -->
         <input class="input-field" id="input" placeholder="Write Comment" type="text" #input>
         <div>
          <button [disabled]="loading" class="w3-button w3-theme"
(click)="saveDestinationComment(input.value)">
           <i class="fa fa-pencil"></i> &nbsp;Comment
          </button>
         </div>
        </div>
        <div class="comment-container">
         <div class="comment-div" *ngFor="let comment of comments">
          <a>{{comment}}</a>
          </div>
         </div>
        </div>
       </div>
      </div>
     </div>
    </div>
    <!-- End Middle Column -->
   </div>
   <!-- End Grid -->
  </div>
  <!-- End Page Container -->
 </div>
import { Component, OnInit } from '@angular/core';
import { EventInfoService } from './event-info.service';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs/Subscription';
import { OnDestroy } from '@angular/core/src/metadata/lifecycle_hooks';
import { DestinationsService } from '../destinations/destinations.service';
import { Destination } from '../destination';
import { User } from '../user/user';
import { CustomComment } from '../custom-comment';
import { Event } from '../event';
@Component({
 selector: 'app-event-info',
 templateUrl: './event-info.component.html',
 providers: [EventInfoService],
 styleUrls: ['./event-info.component.css']
})
export class EventInfoComponent implements OnInit, OnDestroy {
 name: String;
 event: Event;
 imgs: String[] = [];
 destination: Destination = new Destination();
 comments: String[] = [];
 participants: String [] = [];
 owner: User = new User();
 model: CustomComment = new CustomComment();
 lat: Number = 0;
 lng: Number = 0;
 zoom = 16;
 currentImg: any = "";
 private _routeSubscription: Subscription = new Subscription();
 constructor(
  private eventInfoService: EventInfoService,
  private route: ActivatedRoute
 ) {}
 ngOnInit(): void {
  this._routeSubscription = this.route.params.subscribe(data => {
   this.name = data['name'];
  });
  const currentUser = JSON.parse(localStorage.getItem('currentUser'));
  this.owner.username = currentUser.username;
  this.eventInfoService.getEventInfo(this.name).subscribe(
  (res: any) => {
    this.event = res;
    debugger;
    this.lat = Number(res.meetingPoint.lat);
    this.lng = Number(res.meetingPoint.lon);

    // Event img is the first img from the destination
    this.eventInfoService.getEventImgs(res.destination.name).subscribe(
      (resu: any) => {
        debugger;
        this.imgs.push('../assets/upload-dir/' + res.destination.name + '/' + resu[0]);
        this.currentImg = this.imgs[0];
      },
      err => {
        console.error(err);
      }
    );
  },
  err => {
    console.error(err);
  }
);
this.eventInfoService.getEventParticipants(this.name).subscribe(
  (res: any) => {
    for (const participant of res) {
      this.participants.push(participant);
    }
    },
    err => {
      console.error(err);
    }
  );

  this.eventInfoService.getEventComments(this.name).subscribe(
    (res: any) => {
      for (const comment of res) {
        this.comments.push(comment);
      }
      this.comments = this.comments.reverse();
    },
    err => {
      console.error(err);
    }
  );
}
ngOnDestroy() {
  this._routeSubscription.unsubscribe();
}
registerForEvent() {
  console.log('REGISTER FOR EVENT');
  this.event.owner = this.owner;
  this.eventInfoService.registerForEvent(this.event).subscribe(
    (res: any) => {
    },
    err => {
      console.error(err);
    }
  );
}
saveDestinationComment(comment) {
  this.model.user_id = this.owner;
  this.destination.name = this.name;
  this.model.event_id = this.event;
  this.model.text = comment;
  console.log(this.model);
  this.eventInfoService.saveEventComment(this.model).subscribe(
    (res: any) => {
      this.eventInfoService.getEventComments(this.name).subscribe(
        (ress: any) => {
          this.comments = ress.reverse();
        },
        err => {
          console.error(err);
        }
      );
    },
    err => {
      console.error(err);
    }
  );
}
```

```
 myFunction(id) {
   var x = document.getElementById(id);
   if (x.className.indexOf("w3-show") == -1) {
      x.className += " w3-show";
      x.previousElementSibling.className += " w3-theme-d1";
   } else {
      x.className = x.className.replace("w3-show", "");
      x.previousElementSibling.className =
      x.previousElementSibling.className.replace(" w3-theme-d1", "");
   }
}
}
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { environment } from '../../environments/environment';
import { Observable } from 'rxjs/Observable';
import { CustomComment } from '../custom-comment';
import { Event } from '../event';
const SERVER_DOMAIN = environment.serverDomain;

@Injectable()
export class EventInfoService {
 constructor(private httpClient: HttpClient) { }
 getEventInfo(name: String): Observable<any> {
    return this.httpClient.get(`${SERVER_DOMAIN}/events/event-info/${name}`);
 }
```

```
  getEventImgs(name: String): Observable<any> {
    return this.httpClient.get(`${SERVER_DOMAIN}/getDestinationImgs/${name}`);
  }
  getEvents(): Observable<any> {
    return this.httpClient.get(`${SERVER_DOMAIN}/event/getevents`);
  }
  saveEventComment(comment: CustomComment): Observable<any> {
    return this.httpClient.post(`${SERVER_DOMAIN}/saveEventComment`, comment);
  }
  getEventComments(name: String): Observable<any> {
    return this.httpClient.get(`${SERVER_DOMAIN}/getEventComments/${name}`);
  }
  getEventParticipants(name: String): Observable<any> {
    return this.httpClient.get(`${SERVER_DOMAIN}/events/getEventParticipants/${name}`);
  }
  registerForEvent(event: Event): Observable<any> {
    return this.httpClient.post(`${SERVER_DOMAIN}/events/registerForEvent`, event);
  }
  // saveComment(): Observable<any>{
  //   return
  // }

}
```