w

TrueConf

* conference management simplified -

***Table of Content:***

1. Description
2. Domain
3. Functional Design
4. System architecture
5. Frameworks Used

1 Description:

“*Why TrueConf?”*

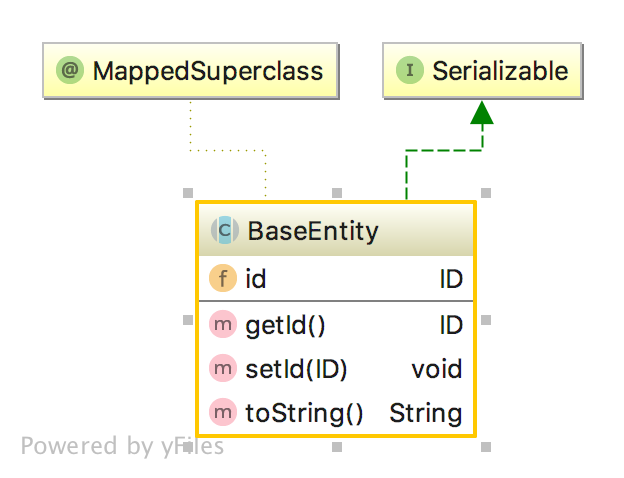
In our modern era and specifically in the domain of scientific research, meetings and events of important nature require high amounts of planning and organizational work. This can be done in many ways, with the help of on-line social platforms or via e-mail. But this is not enough, as on-line social platforms don't offer *specialized solutions* and e-mail is just not enough. This is where ***TrueConf***, our conference management service fills the bill. A free and fast website capable of handling *event planning* complete with *e-mail integration*, *attendance & reviewing management*.

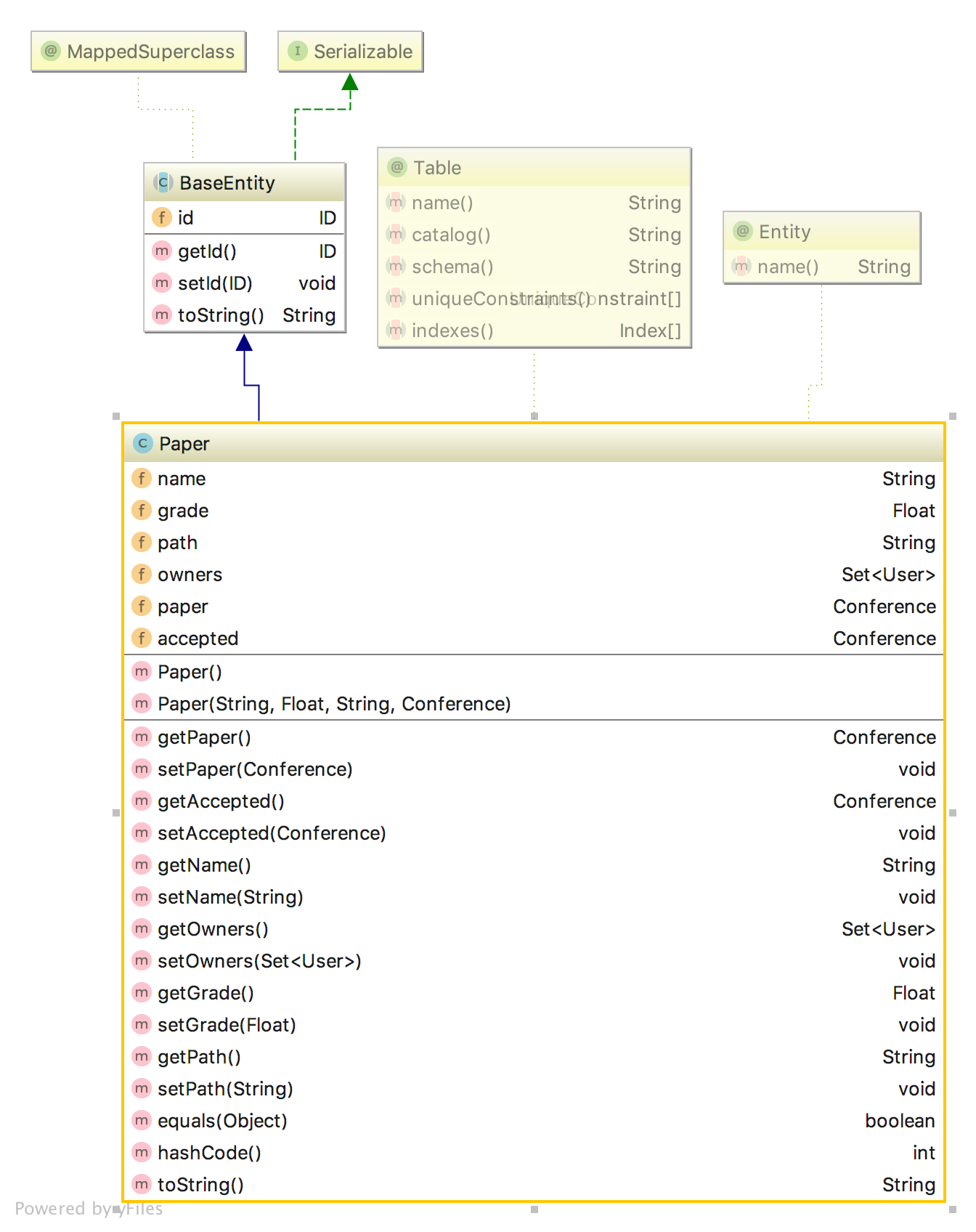
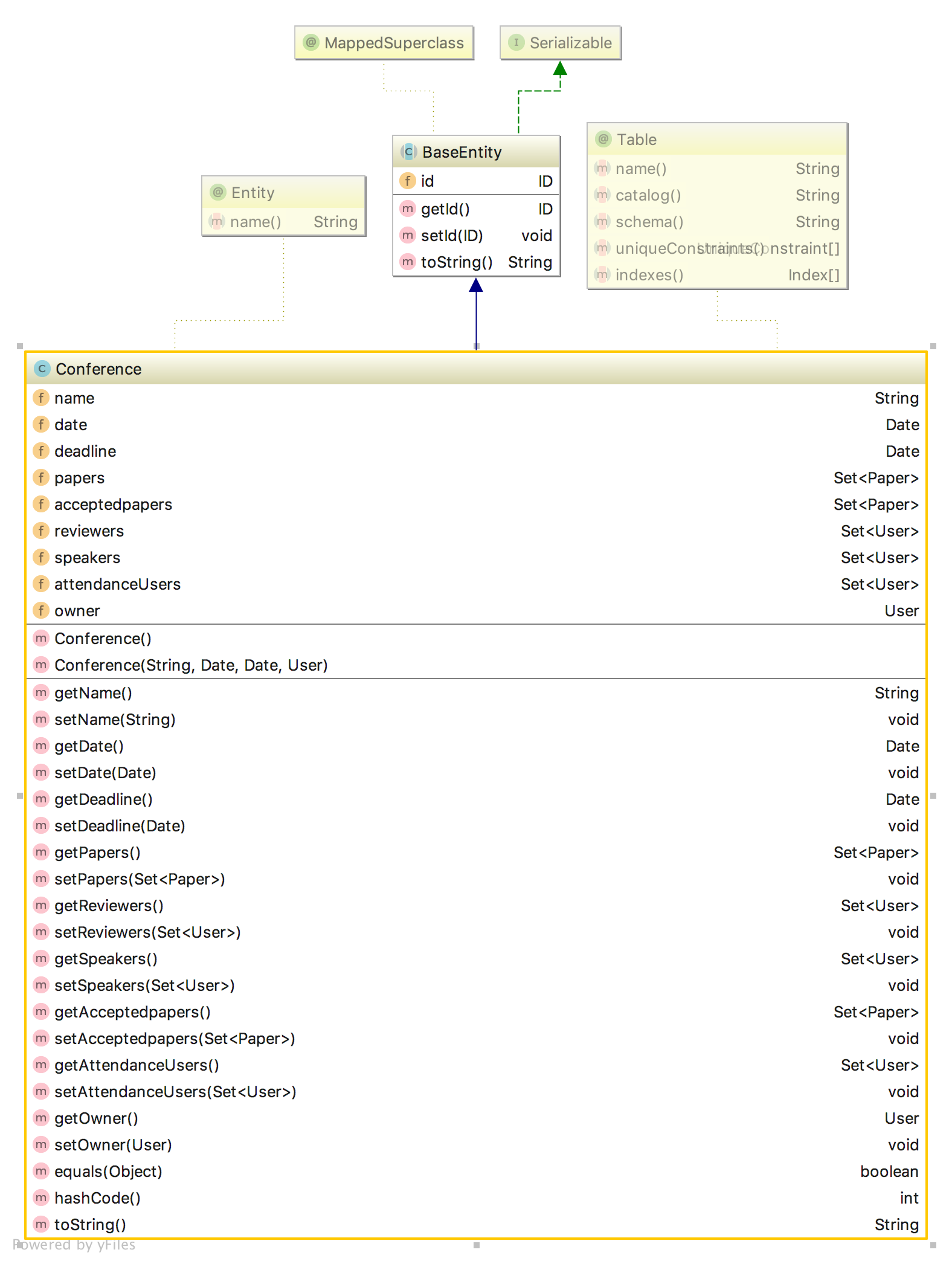
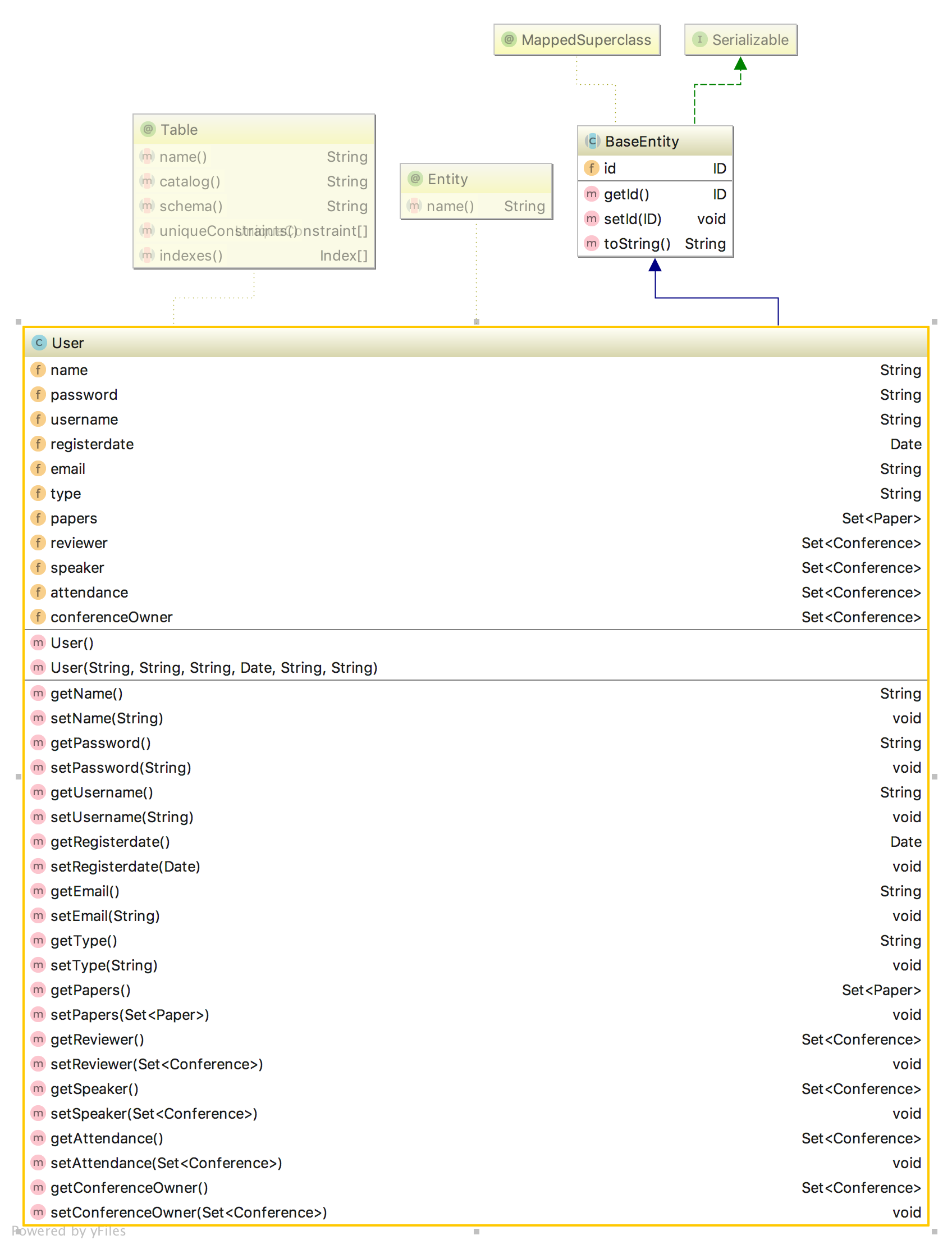
Managing conferences as a chair is not an easy task! Announcing reviewers for new papers, keeping all attenders in touch with the event and other domain specific tasks are a hassle to deal with. Solutions to this problem usually involve stand-alone applications or clunky alternatives. Although our system relies on a more complex back-end service, our users need to worry only about *connecting to a website* *and registering an account!* This simplifies your task to only creating an event and assigning a reviewing board! The rest is dealt with by our system, which will be further detailed bellow under the *System Architecture*.

2 Conference Domain

Building software for conference attenders and managers, we had to look at real life cases and model our domain classes accordingly. For the sake of usefulness and delivering a straight-forward experience, our domain model consists of three main classes. These are some of our mappings, with some reasoning behind *why* we chose them.

* **User** - as an integral part of the conference, the user receives different statuses in the *back-end services.* This design decision simplified the CRUD operations regarding users, allowing the Hibernate framework to handle more strenuous filtering.
* **Conference** – conferences were designed as modular events, holding data regarding the real-life events and being directed exclusively by conference owners and associated reviewers.
* **Paper** – the design of papers was quite straight forward, the data stored being the files themselves as well as important metadata (author, conference, status).

Another very important part of this model was the **BaseEntity**, a *serializable* and *Spring* *compatible* class, serving as a generalization of the aforementioned classes.

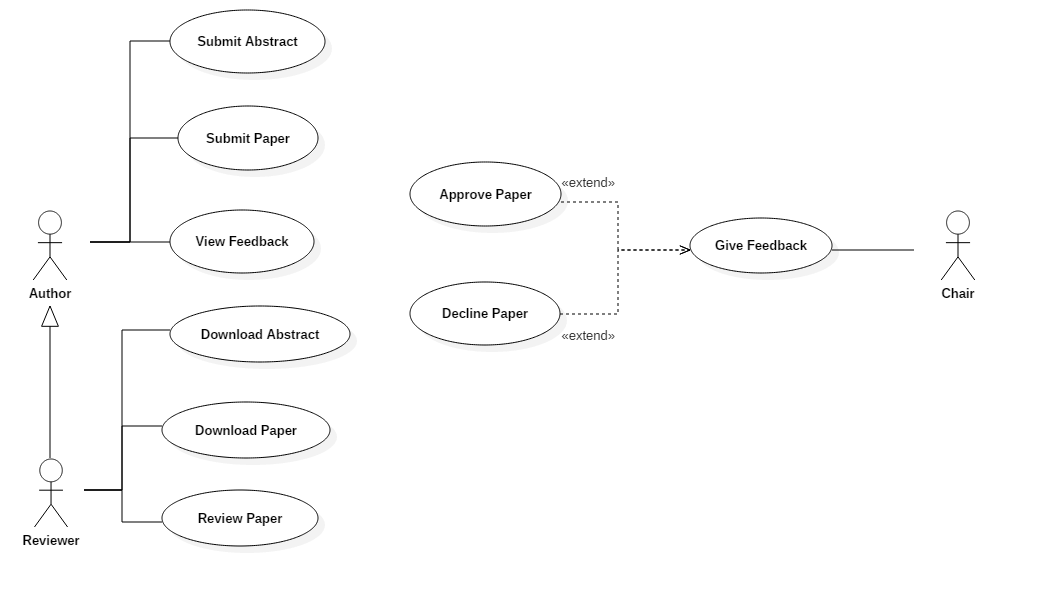


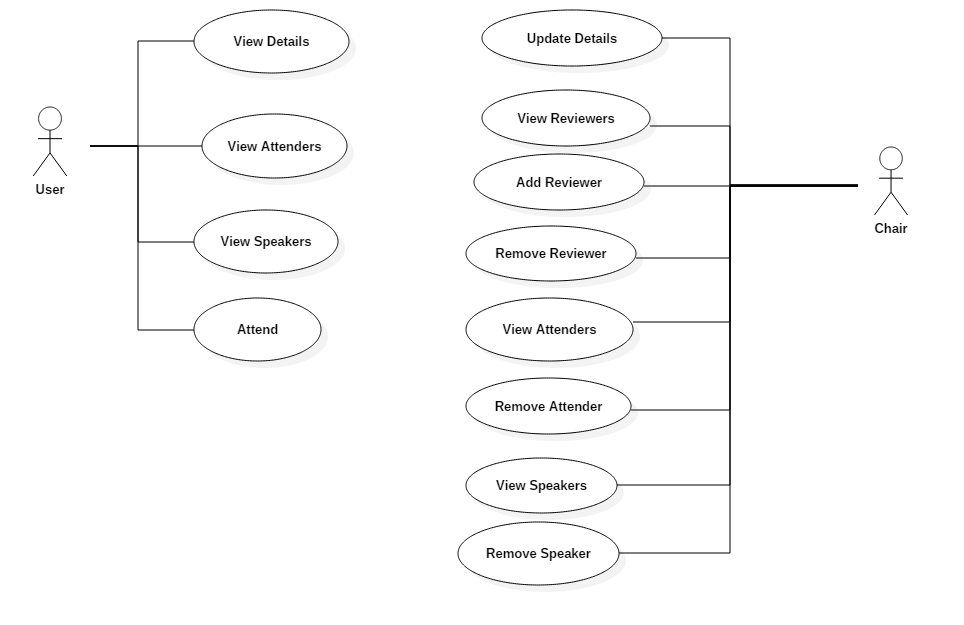
3 Functional design:

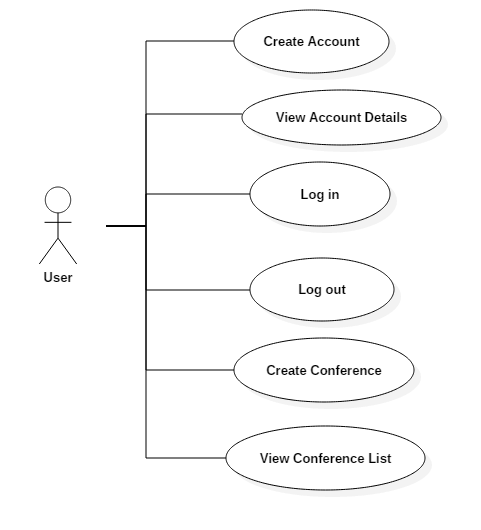
*“Filling a need”*

Our functionalities are based around three main entities: Papers, Users and Conferences. Each of these was designed with the essential needs of the users in mind. The first thing we decided upon was an authentication system which lets you handle other entities in different ways based on your *account type*. Therefore, regular users have access to their own papers, they can improve upon them if they wish to do so, but any functionalities related to infrastructure are locked.

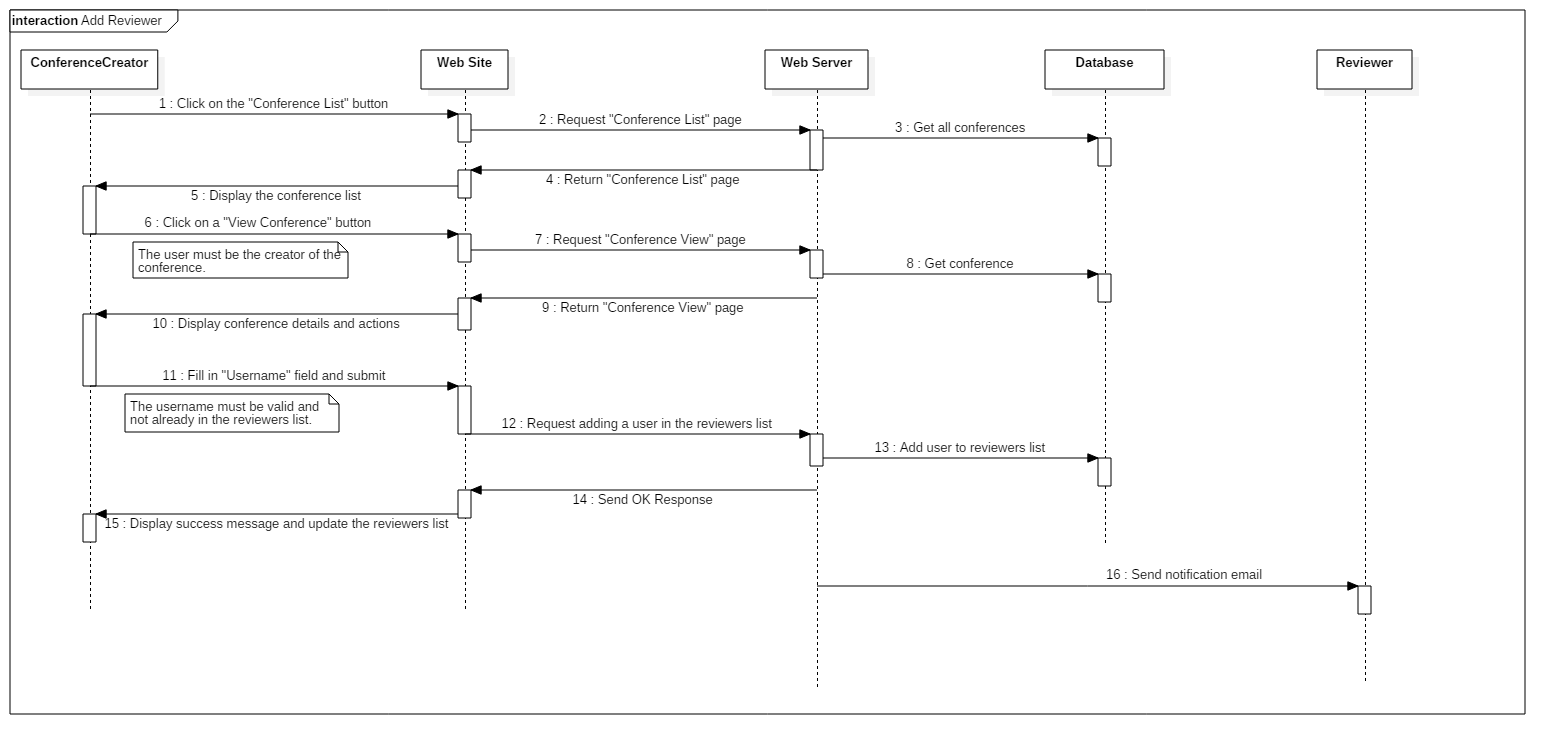
A more privileged user will have access to grading functionalities and deadlines, as well as assigning people to the reviewer committee. Conferences are also pretty straight forward and intuitive in their design. A user can create or edit a conference, invite people to it or simply choose to participate to an existing one. Some of these specific functionalities regarding users interacting with the system are shown in the following diagrams:

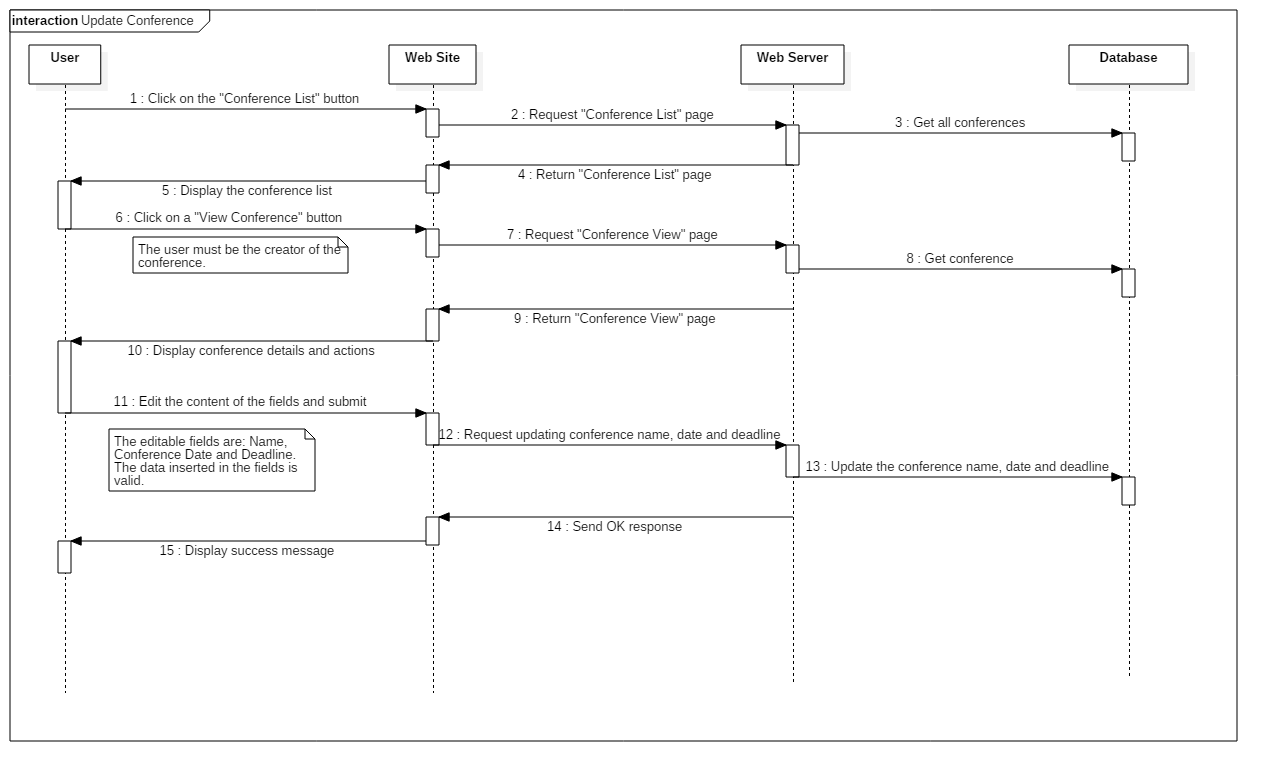
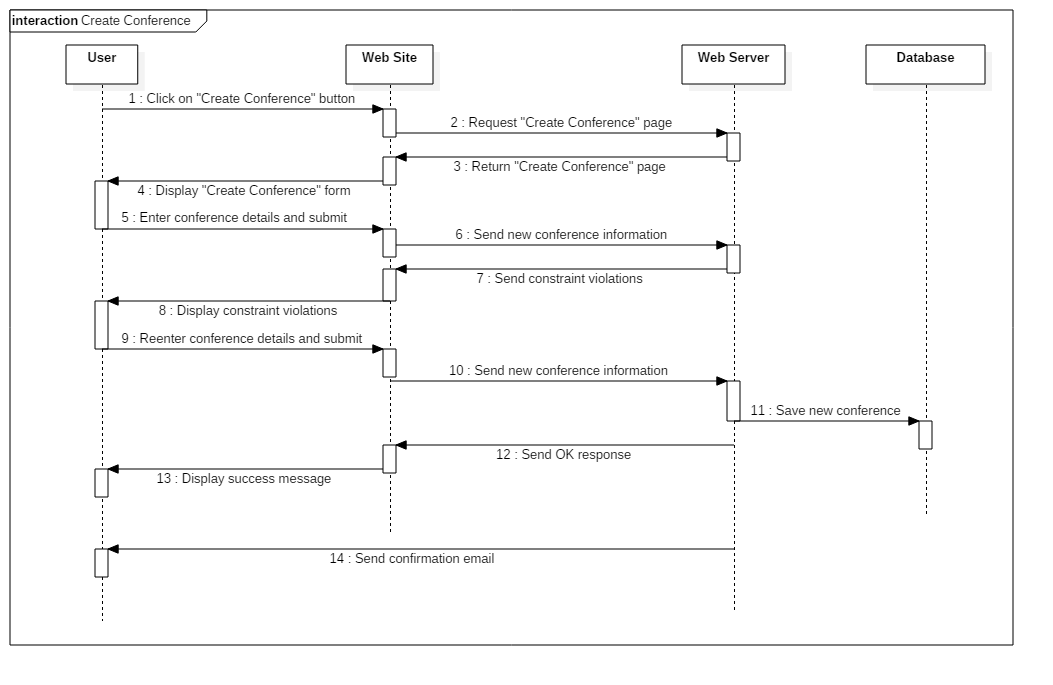
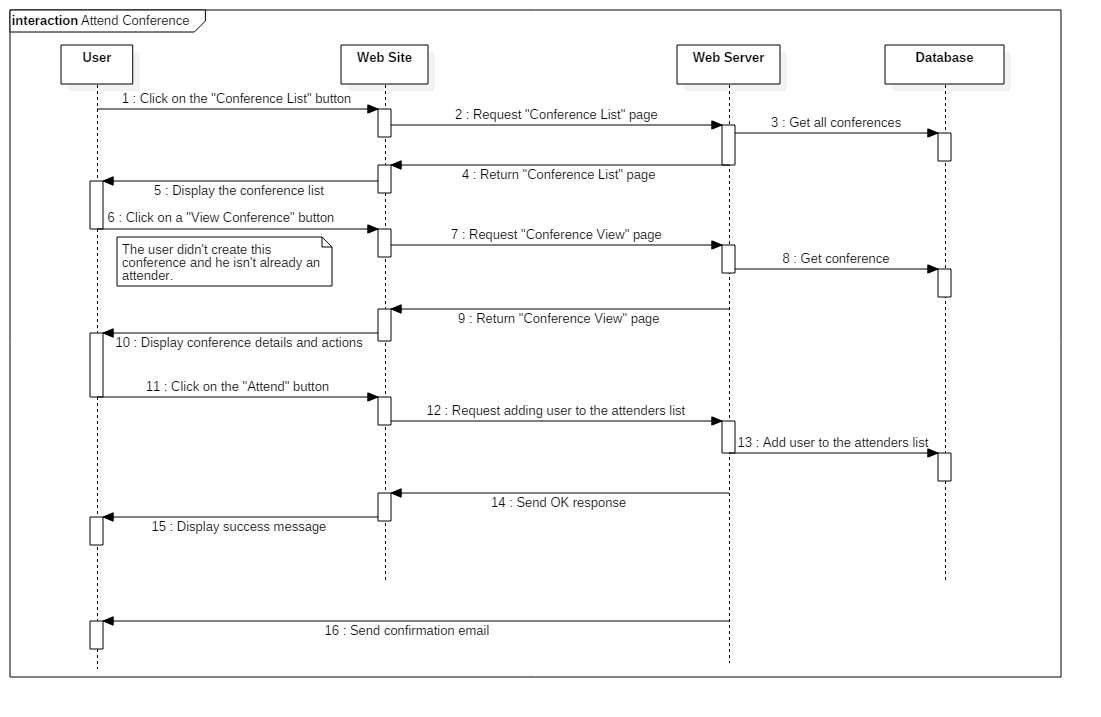






Apart from these use-case diagram, there was also the need to model the real-time behavior of the system, as depicted in the diagrams bellow:





4 Architecture:

Our system was comprised of multiple subsystems, dealing with user issued tasks in an asynchronous manner, whilst also dealing with server issued requests. By integrating the Client-Server model into our application via the Angular framework together with the Spring and Hibernate frameworks, we managed to design the application in a straight-forward manner. The relevant subsystems that emerge from this approach are as following: **client (website)**, **server**, **database and model.** These two were inevitably split into web (containing the client and an angular service) and the core component (containing the rest).

**Client**

The client component represents the website. Using TypeScript and Angular service calls the application can communicate to the back-end server.

**Server**

The server deals with incoming Angular requests and ensures real-time connectivity to the web-app component, at the same time making sure that concurrent client sessions cannot lead to conflicting actions.

By use of the Spring Framework all the components of the server (mainly the class-related services) are enabled and managed.

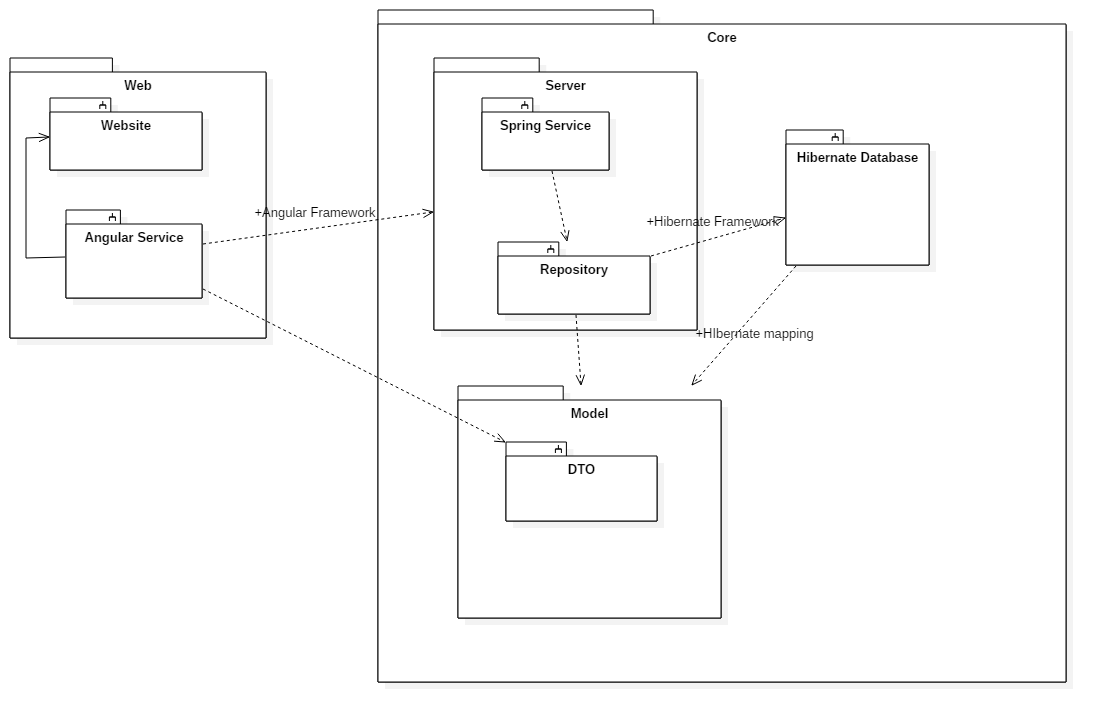
**Database**

The database component is made up of the actual database generated by Hibernate and it’s class analog present in the core of the application.

**Model**

The Model implements the business logic of the application and makes sure that the data is correct.

Bellow we’ve attached a diagram to describe the major interaction between the subsystems, as they delegate and accomplish tasks.



5)Frameworks used

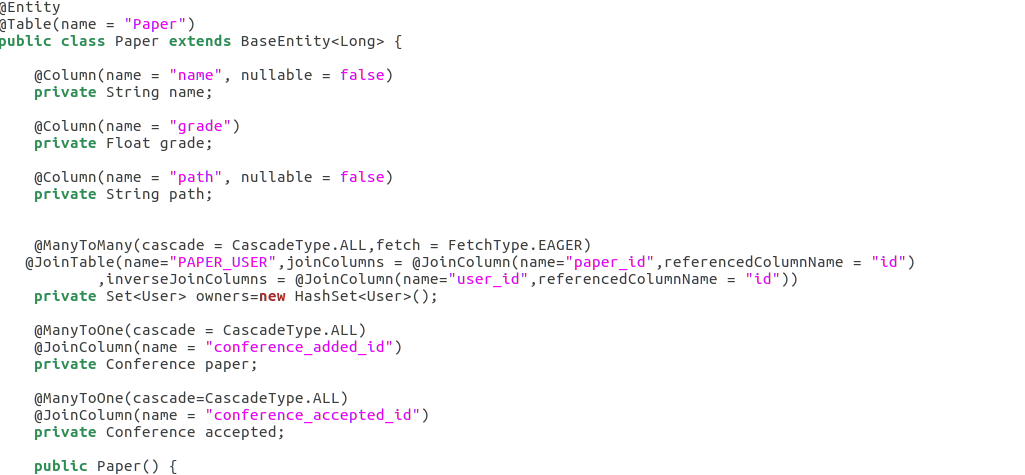
*Hibernate:*

Object-Relational Mapping It is a programming technique for converting object-type data of an object oriented programming language into database tables.

Hibernate it is a open It is open source object-relational mapping (ORM) for Java.

It’s used to convert object data in JAVA to relational database tables and is responsible for making data persistent by storing it in a database.

In our project we used Hibernate to generate the database for the model and for the relations between the entities.



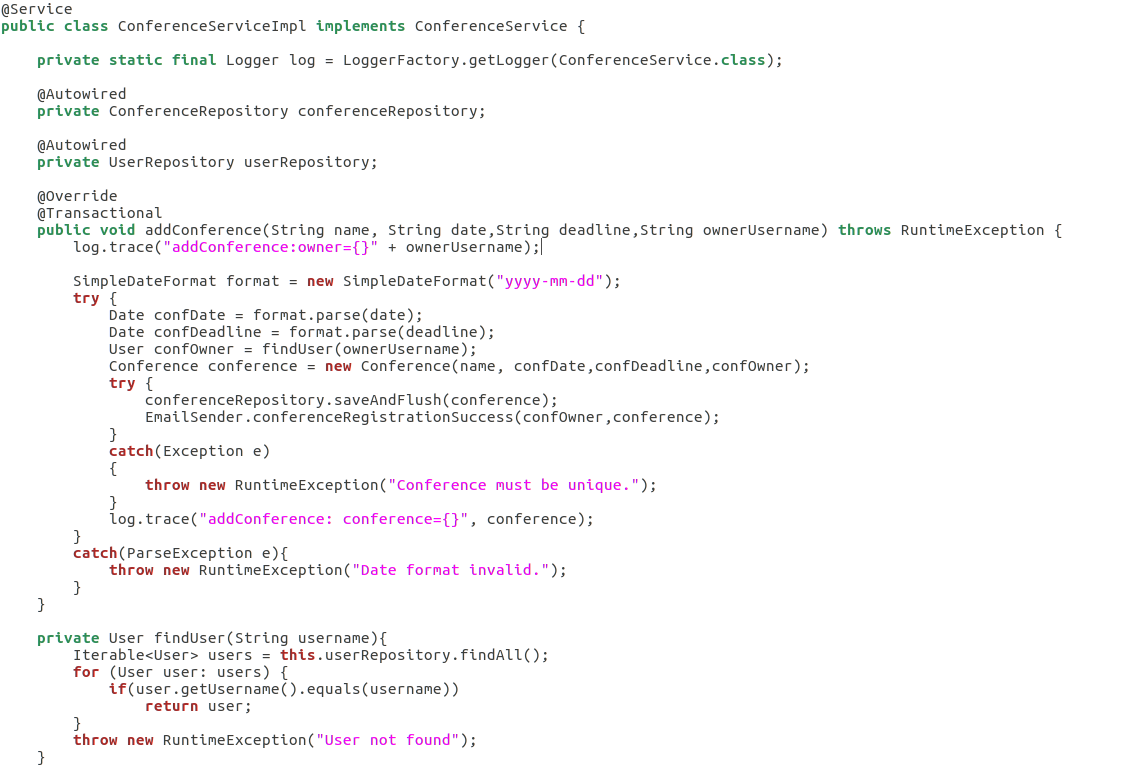
In the example above we described an entity for Paper .Using the @Table annotation we had specified the table name from the database. For the class properties that is the corespondent of a column we used the annotation @Column and we specified the name of the column and if can be NULL or not (also here we ca decide if is unique or not).

For the relationships we used annotations like @ManyToMany, @OneToMany and @ManyToOne. To map the authors (users) we used a many to many relationship to show that a author can have multiple papers and also a paper can have multiple authors, also with the annotation @ManyToMany we specified the table in which hold the relation and the columns that are involved in this relation. Using @ManyToOne we mapped the Conference in which the paper was submitted, having in the Conference part a @OneToMany relation.

We used Hibernate to speed up the process of creating the database and the model, also being very easy to change things later in the project.

*Spring:*

The Spring Framework is an application framework (open and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an alternative to, replacement for, or even addition to the Enterprise JavaBeans (EJB) model.



The application is using Spring Beans to link the server services to the client side controllers, so it can be accessed remotely. With the @Service we specify that this si a public service and can be accessed from a remote controller. @Autowired will link the declaration of the repository with a instance of that repository.



On the client side is a controller notated with @RestController. In this controller we link the service form the server side so we can use it as a method of calling functions remotely from server. To access the functions from the web side we map the functions using @RequestMapping and call them using http request with parameters.