Table of Contents

**Versioning Table1**

**C4 Style Architecture**2

C1

C23

C34

**Entity Relationship Diagram5**

**CI Configuration Diagram6**

**Framework & Database Choice Justification7**

**Authentication & Authorization Setup Explanation8**

Backend Implementation

Frontend Implementation9

**Wireframes10**

Homepage

Login11

Register12

**Quality Assurance Metrics13**

**Quality Assurance Metrics14**

**Burndown Chart15**

**UX Feedback16**

**DOT Framework Research Methods17**

**DOT Framework Research Methods18**

**Video per finished story19**

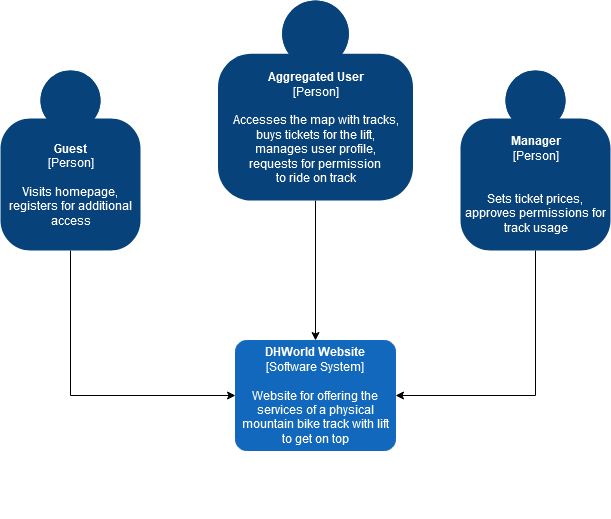
**References20**

**Versioning Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision history | Revision class | Comments |
| Sprint 2 | 1.0 | Major | Initial content |
| Sprint 3 | 1.1 | Minor | New diagrams, updates on old diagrams |
| Sprint 4 | 2.0 | Major | Updated to APA style; added new points, new diagrams |
| Sprint 5 | 3.0 |  |  |
| Sprint 6 | 4.0 |  |  |

1. **Architecture using C4 style**

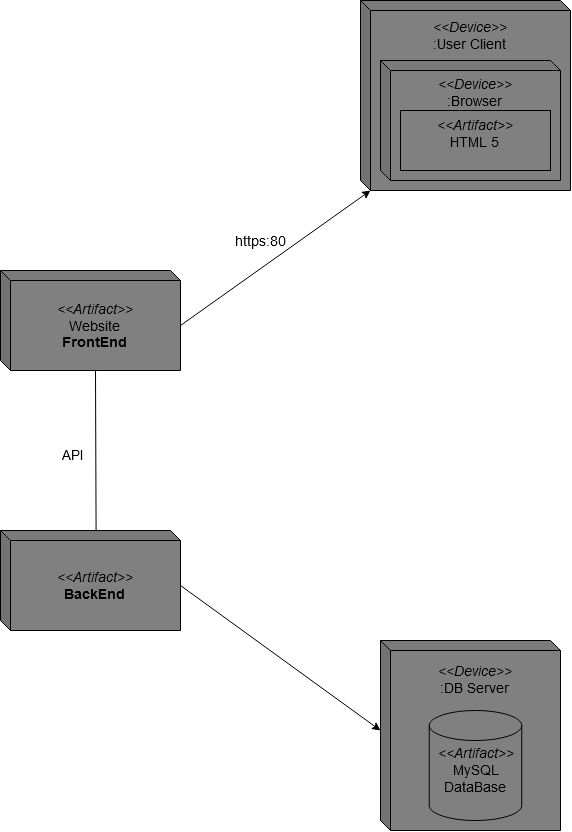
**C1**

****

**Clarification**

A **guest** is a regular visitor that is not logged in/registered. An **aggregated user** is a logged in user. A **manager** has administration role and managing role.

**C2**

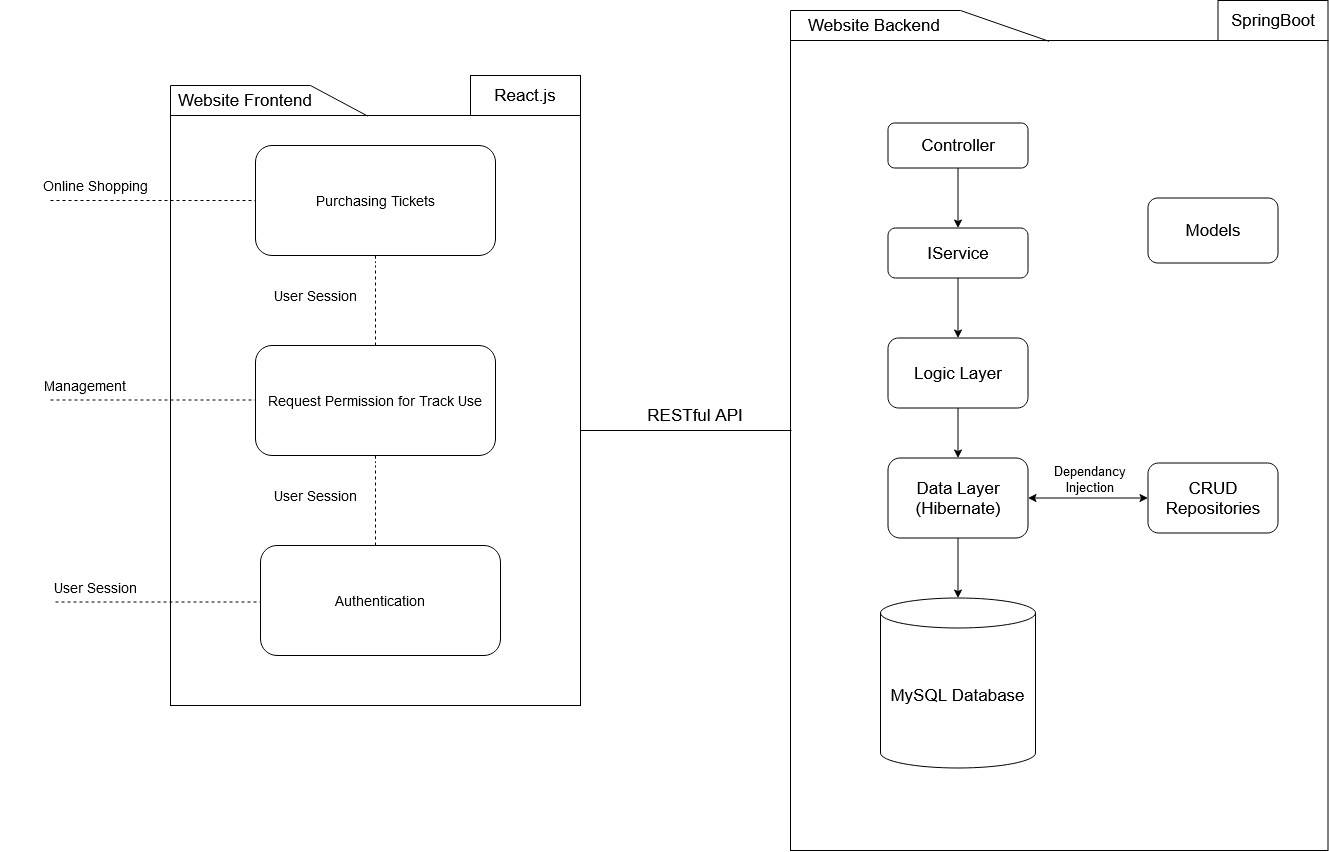
****

**Clarification**

The above diagram showcases the connectivity between the **back** and **front end** through a RESTful API. They are linked via association relationship.

The backend sends **data flow** to the data layer, which is displayed by an arrow pointer.

**C3**

****

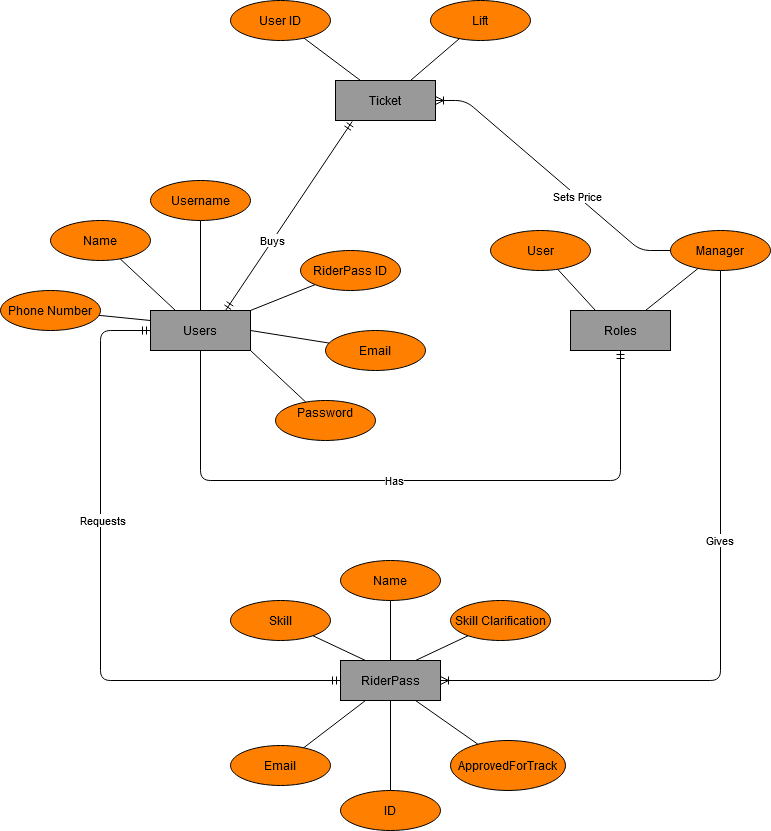
**Clarification**

Layered Architecture design means that everything related to a certain layer, stays in that very layer. This allows for a clean separation between the different layers, e.g. logic & data layer. It means that the whole code has to be grouped by each different functional roles in “layers”.

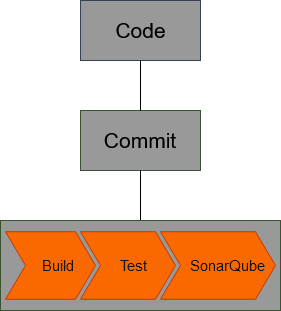
My layers are separated as displayed in the diagram:

* I have a controller layer, otherwise known as business layer.
* I have a service interface, which separates the business layer from the logic & data layer.
* The data layer uses dependency injection to connect to the native CRUD repository, and is connected to the logic layer.

**2. Entity Relationship Diagram**

****

**CI Configuration Diagram**

****

**Clarification**

In order to setup continuous integration for my project I had to get familiar with how it works, starting with DOT framework *“Library”* research. For the setup I’ve used two yml files in my repository. The first one is in the root folder of the repository and it just points to the second yml file, which is located in the backend folder. The yml backend folder has the functions explained in the diagram above.

**3. Back-end Framework Choice Justification**

I decided to go with **Spring Boot** instead of Jersey mainly because the documentation of Jersey is written in a strongly technical and theoretical way, thus it’s more difficult to understand the explanations. Moreover, Spring Boot’s documentation has detailed examples and it is written in a simple way. Also, Spring Boot seems to be the more popular choice.

**4. Front-end Framework Choice Justification**

For the front-end part I picked **React.js** given the great features it provides for building interactive UIs. It is fairly simple to use, with an architectural design consisting of encapsulated components that manage their own state, which is very appealing to me. Furthermore, it’s the most popular choice amongst the front-end frameworks, with a lot of accessible educational materials, guides and tutorials out there on the internet. Besides, I like React’s syntax style.

**4. Database Choice Justification**

For the database of my project I picked **MySQL**. The main reasoning behind this is the fact that it is the most widely used open-source database. Therefore, there are a lot of easily accessible educational materials for using MySQL on the internet. I find it easy to use and feel comfortable using it. It is very fast and satisfies perfectly the needs of my project.

For testing purposes of my application I chose to use the **H2** **database**. It can be configured to run either as a disk-based, or an in-memory database, hence the data doesn’t persist on the disk. Being open-source, with support for SHA-256 password encryption and other encryption functions; very lightweight, and therefore extremely fast, it is a go-to database for development and testing. There are many accessible options to learn and get used to work with H2, which helped my choice over another in-memory database.

**Authentication & Authorization Setup Explanation**

I use the OAuth protocol. It enables websites or applications (Consumers) to access Protected Resources from a web service (Service Provider) via an API, without requiring Users to disclose their Service Provider credentials to the Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication.

**The backend uses Spring Security for the Authentication & Authorization as follows:**

The interface *WebSecurityConfigurerAdapter* is the main feature of the security implementation. It provides *HttpSecurity* configurations for CORS, ESRF, Session Management and rules for protected resources.

*UserDetailsService* interface has a method *LoadUserBy*username that returns a *UserDetails* object, which contains necessary information (such as: username, password, authorities) that Spring Security can to build an Authentication object.

*UsernamePasswordAuthenticationToken* gets the username and password of the user from LoginRequest, *AuthenticationManager* will use it to authenticate a login profile.

The *AuthenticationManager* has a *DaoAuthenticationProvider* (with help of *UserDetailsService* & *PasswordEncoder*) to validate *UsernamePasswordAuthenticationToken* object. If successful, *AuthenticationManager* returns a fully populated Authentication object (including granted authorities).

The public abstract class *OncePerRequestFilter* makes a single execution for each request to the API. It provides the *doFilterInternal()* method that is implemented parsing & validating JWT, loading User details (using *UserDetailsService*), checking Authorizaion (using *UsernamePasswordAuthenticationToken*).

*AuthenticationEntryPoint* is used to catch eventual authentication errors.

I use *UserRepository* & *RoleRepository* to work with the database, they’re imported into **the *AuthController***.

**AuthController** receives and handles a request after it was filtered by *OncePerRequestFilter*.

– *AuthController* handles signup/login requests

– *TestController* has accessing protected resource methods with role based validations.

**The frontend implementation is as follows:**

*Login* & *Register* components have a form for data submission (with support of react-*validation* library). They call methods from *auth.service* to make a login/register request.

*auth.service* methods use *axios* to make HTTP requests. It stores/gets the **JWT** from the Browser Local Storage inside these methods.

The *Home* component is intended to be public for guest users.

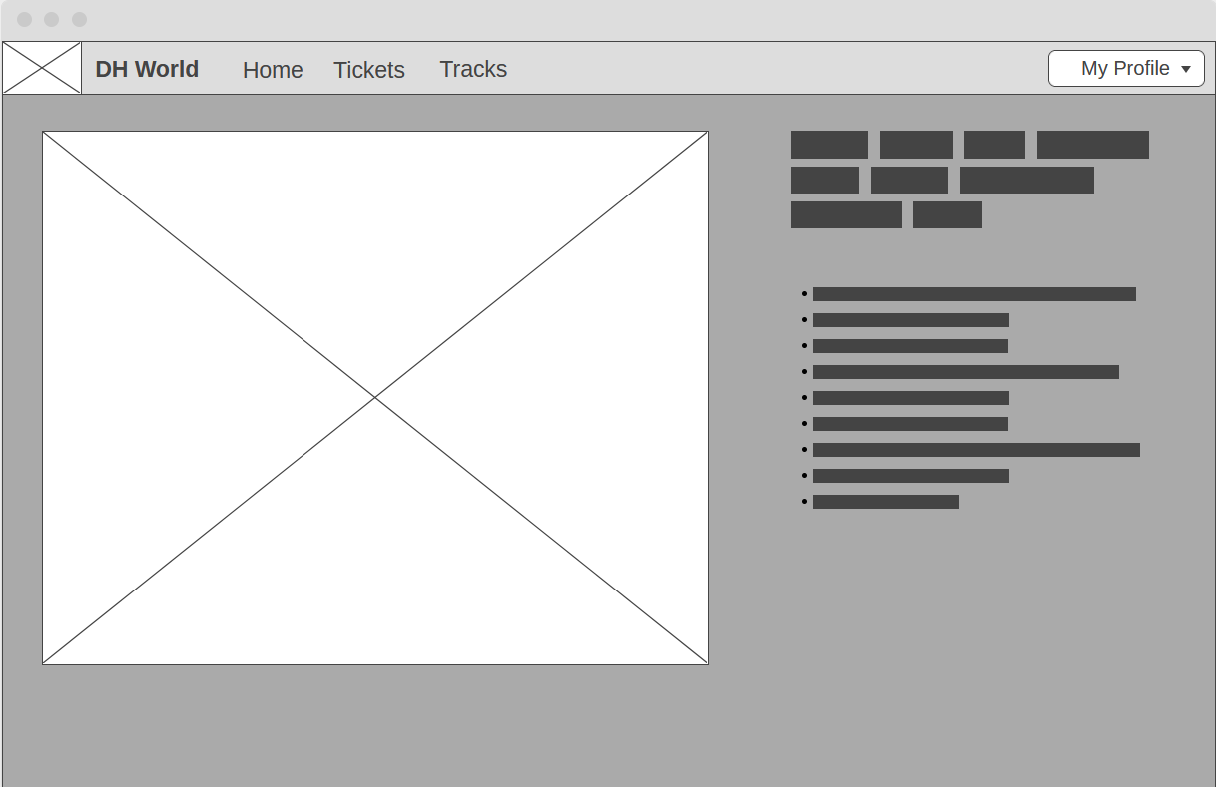
The *Profile* component displays user information provided that a user is logged in.

*BoardUser* and *BoardModerator* components will be displayed based on the state of *user.roles*. In these components, *user.service* is used to access protected resources from the Web API.

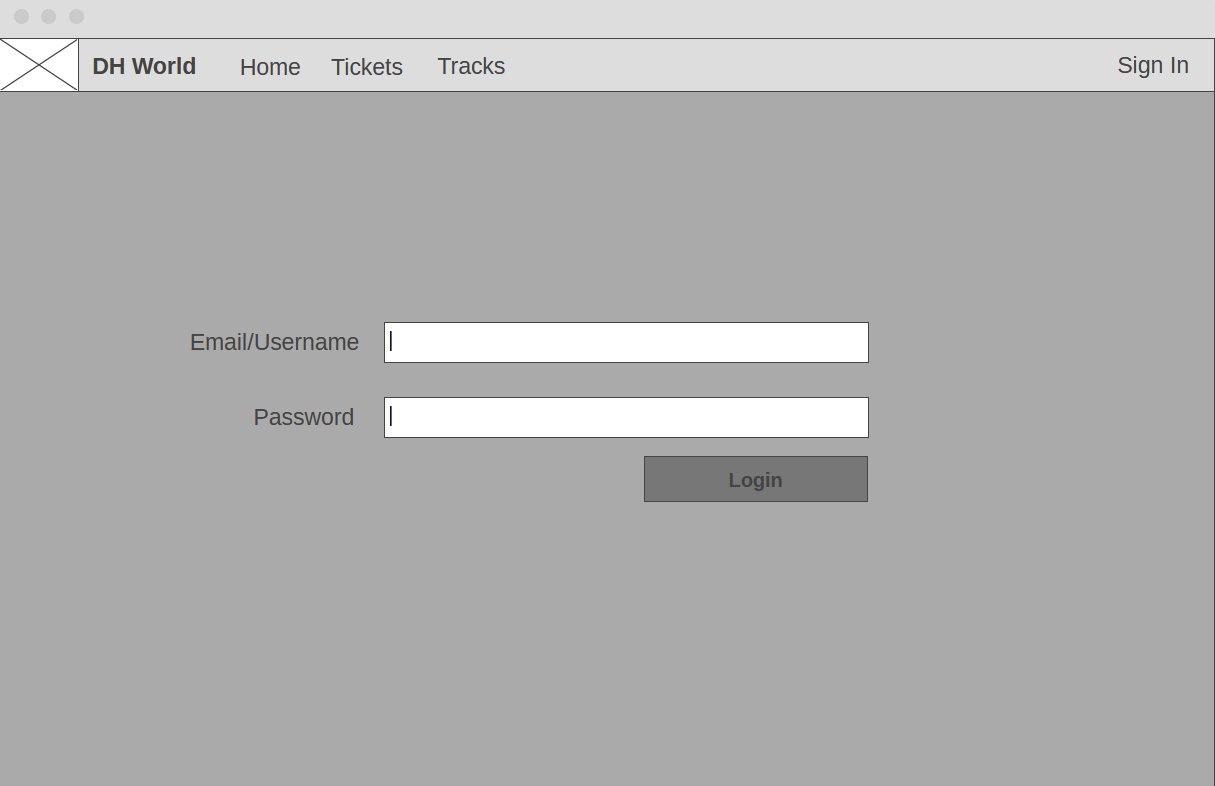
This *user.service* uses *auth-header()* helper function to add the JWT to HTTP header. *auth-header()* returns an object containing the JWT of the currently logged in user from the Local Storage.

**5. Wireframe**

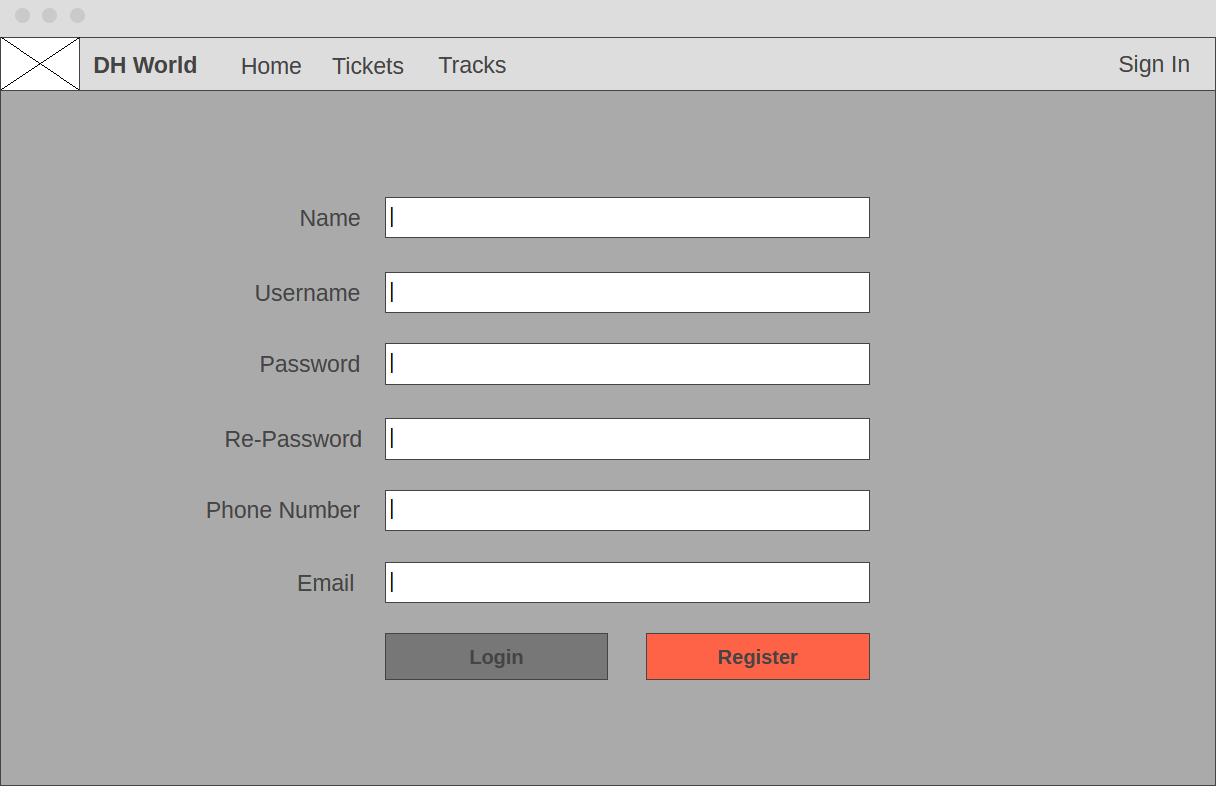
* **Homepage**



* **Login**

****

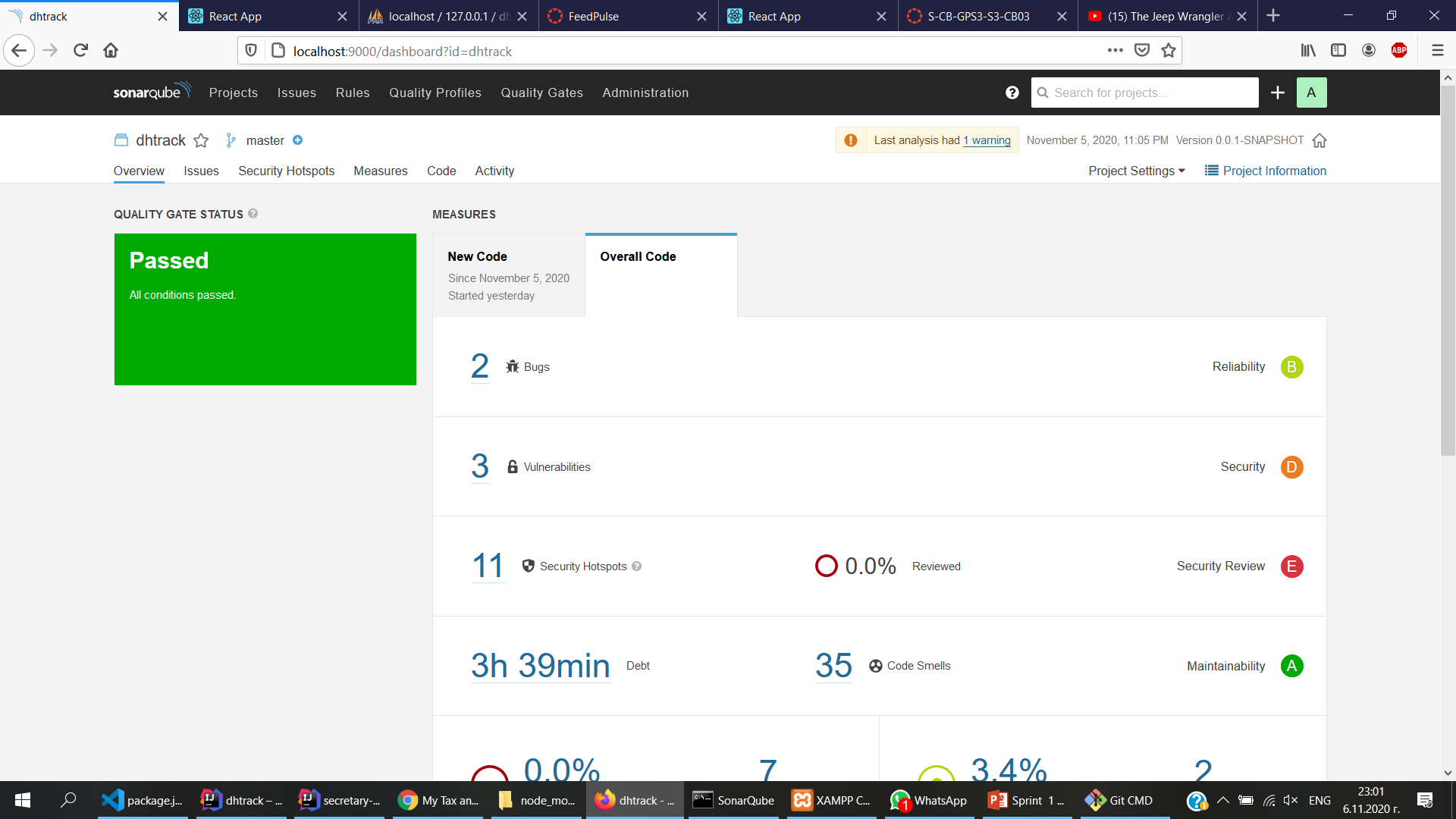
* **Register**

****

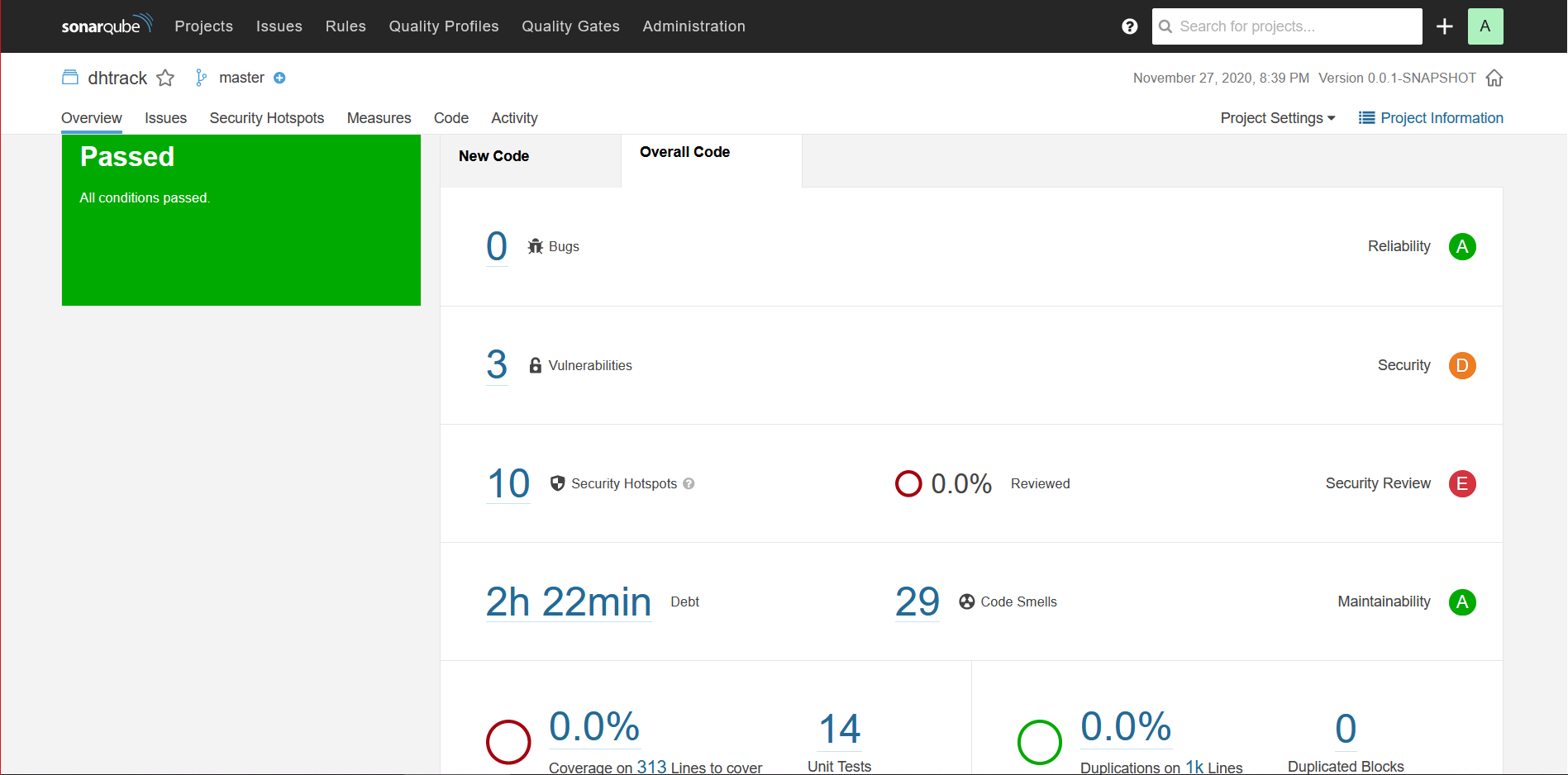
**6. Quality Assurance Metrics**

Screenshots attached from the SonarQube analysis of the project’s code quality.

**Pre**:



**Post**:



**7. Burndown Chart**

**8. UX Feedback**

In the research conducted I’ve contacted both men and women. The first feedback is made by a female with some background in the ICT sector. The second feedback was given by a male with no special knowledge in IT.

**First Feedback:**

* The tester liked the design, the fact that it has all the CRUD functionalities, and gave me a feedback that I should work on the design further and that I should include a footer, and a homepage matching the website’s context.
* While trying the login function, the tester came upon a bug: When a person successfully logs in, an error pops up and they have to manually go to the homepage to continue browsing the website.

**What was the follow-up from the given feedback**

* I took the feedback into consideration and fixed the login bug; I started working on the design of the website and made a homepage.

**Second Feedback:**

* The tester complimented the design, saying that it’s minimalistic and matches their taste.
* The tester noted that the bug on the login feature ruins the otherwise flawless design

**What was the follow-up from the given feedback**

* I am actively working on a possible solution for the issue noted by both

**DOT Framework Research Methods**

**The What:** Research is being conducted on both the specific application context of the project and the more general knowledge which could be useful for the project itself. Both types of research are necessary and we can express this by three domains. The first one is the “application domain”, which is the domain of the specific context on which the application takes place. The second one is “available work”, which in a nutshell is all the available theory, models and etc that could be used. The third one is the “innovation domain”, where all the research is done and the actual innovation takes place.

**The Why:** An explanation of the research goals help to better structure the actual research. If an application context research is conducted, the reason would probably be to obtain a product which is relevant to the stakeholders. Most of the times an assurance that the product is up to contemporary quality standards. In these case we want to use all the expertise available for creating the product. This is done by doing a research in the “available work” domain.

Often times we want to gain a good overview over what is needed or what is available. At other times we might want to test specific aspects of the work to make sure it works. In these cases we try to configure the research to optimize “certainty” about the hypothesis or goals.

**Which Methods I’ve applied for my research:**

* **Library** - it is performed in order to explore what is already done and what guidelines and theories exist that could be useful for the design.
* I have used video tutorials and publicly available online documentations.
* **Field Research** is done to explore the application context. It’s applied to get to know the end users, their needs, desires and limitations as organizational and physical contexts in which they will use your product.
* I have contacted two actual people to test my web application to better get to know my flaws by getting different opinions about the user experience of an end user, and to allocate possible bugs and fix them.
* **Workshop Research** is done to explore opportunities. Prototyping, designing and co-creation activities are all ways to gain insights in what is possible and how things could work.
* I created wireframes so that I could visualize how the application would look like before making the design itself.

**Video per finished story**

Hereby I provided a link with a video I’ve created showcasing authentication & authorization plus basic crud functionalities on my website.

https://vimeo.com/484555504

# References

(n.d.). Retrieved from guru99: https://www.guru99.com/postgresql-vs-mysql-difference.html

(11/2020). Retrieved from bezkoder: https://bezkoder.com/spring-boot-jwt-authentication/

(03/09/2020). Retrieved from bezkoder: https://bezkoder.com/spring-boot-jwt-authentication/

(2012). Retrieved from databasequest: http://www.databasequest.com/index.php/product-service/mysql-dbquest