System Documentation

Project EWA

Thimo Soet	500825251
Floris van Stal	500828298
Marcel Hettinga	500789255
Jari Roossien	500823316
Bram Mes	500829896

Karthik Srinivasan
Tom Scholten

Monday January 10th

Hogeschool van Amsterdam

Introduction

This web application was developed to assist the starting general practitioner Zonnevelt. The application will serve multiple purposes, such as managing patients and doctors and having a live chat to let a patient question a doctor about any inquiries they might have. Such questions might differ from discussing a diagnosis to making appointments. Sensitive data, like medical records, must be anonymized when saved to maintain the privacy of a patient.

In this document, several aspects of the product made will be discussed. Starting with product vision, which will feature a description of the product. This will be followed by an explanation of some of the most important user stories which are implemented in the product. Furthermore, this document will contain a Layered Architecture Package Diagram, which will show the layered architecture of the product, as well as a Navigable Class Diagram, an elaboration of some challenges that were encountered, a Deployment Diagram and an analytical reflection of the design respectively.

All of this info will give an insight in what the product consists of and what purpose it serves. If there is a need to further improve the product, the analytical reflection will help future workers to realise those improvements, as well as show what flaws might be present. The Deployment Diagram serves to show how the product can be deployed at its current state. However, this might be altered later on by future architects if the product migrates to a different environment. The Navigable Class Diagram explains the structure of the product, accompanied by a class diagram that shows the structure and dependencies of a single repository in the JPA persistence layer, as well as a class diagram that clarifies the use of an external interface.

Product Vision

Our product vision was to make a simple, easy to use website. The patient will be able to create an appointment much faster and come into contact with their doctor. For the practitioners side, it will help them by removing some workload. For example, an appointment can be created online instead of calling the doctor. For questions a patient does not need to come all the way to the practitioners office, they can now send a message.

A patient can create an account and then a doctor will need to add this person to their list. A person will then be able to make an appointment with their doctor.
A patient also has the ability to talk to a doctor via a live chat.

A Doctor account can only be created by an admin account through the admin panel. When a doctor account has been created he or she can start accepting patients through the unlinked patients tab. Then the doctor can see her patients in an overview and there they can start a chat or change details.

Our target demographic are patients. With this site patients are able to schedule their own appointment and they can consult with their doctor via chat. Especially during corona it’s very handy that all the appointment making is handled through the site instead of the need of calling the practitioner's office.This will reduce some workload on the people working there.

The chat is also very handy. Because, if you don’t want to go all the way to the practitioners office or don’t want to wait on the phone. You could send a message to your doctor, that he can answer when he has the time.

A doctor will now be able to see all his patients and see their medical information. This is handy because when chatting with a patient, he will be able to look through their history to see if there is an explanation for their problem.

Epic Stories

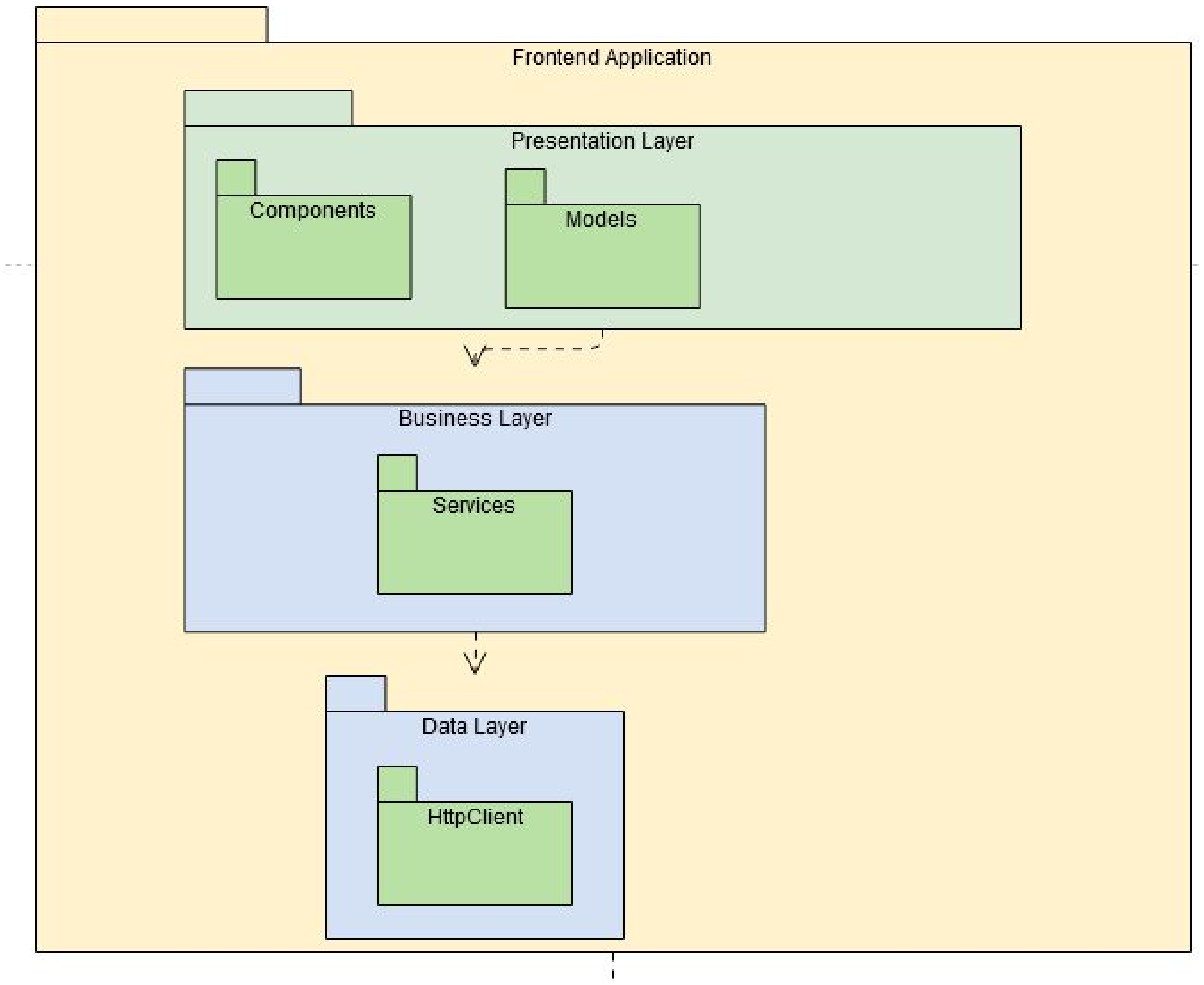
The first epic was the ability to communicate between patients and doctors through the website itself. We split this up into the ability to chat live between 2 users, and an overview of patients a doctor had the ability to chat with.

For the user story chat the requirements we'd set were that it had to be updated on the spot, rather than forcing a user to refresh or wait for a certain period, the chat for private for each user between the patient and the doctor, meaning no other users could read with them, and the history of the chat should be able to read back if a user would go back to the chat.

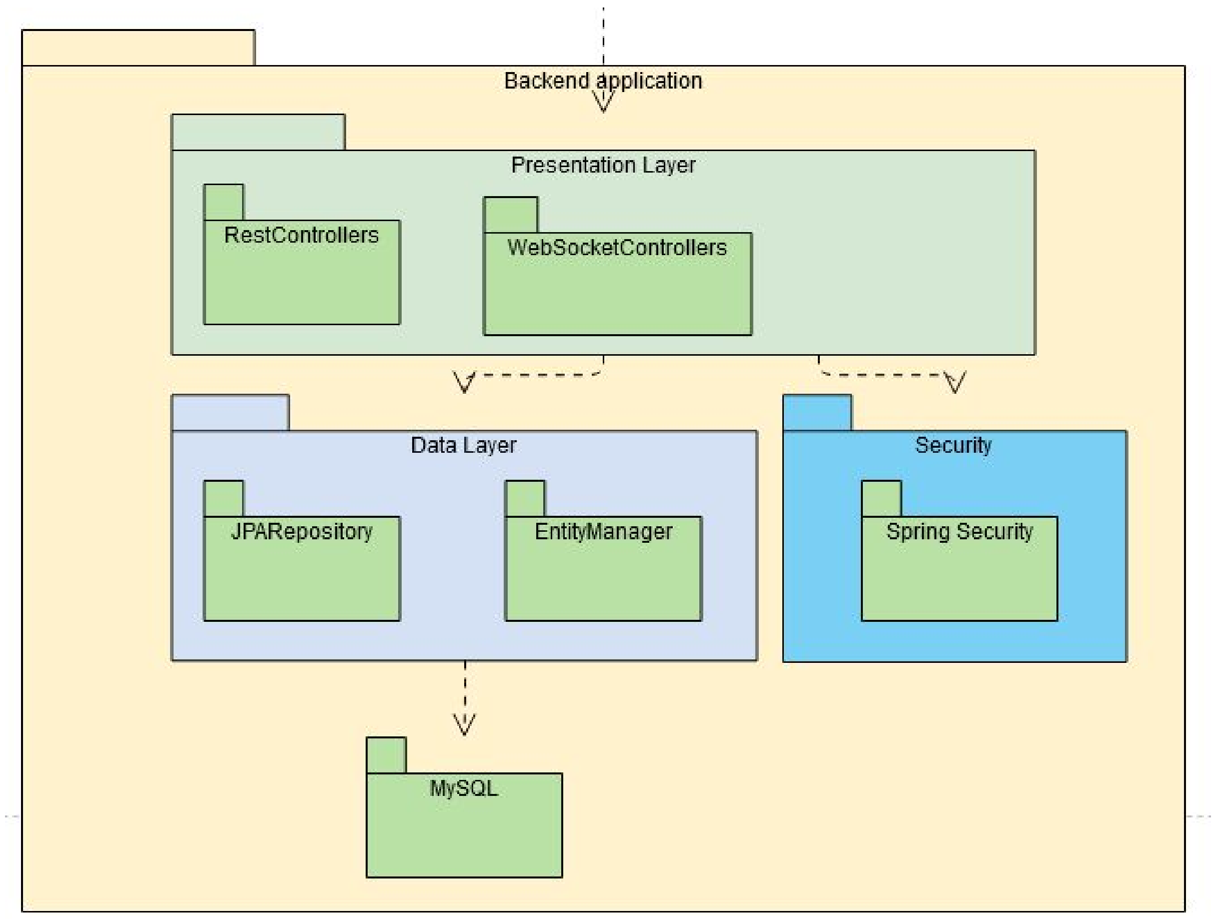
The second epic was that the data of patients should be saved and edited in our application in a secure manner. They got split into the user stories of that a patient can see but can't change their own data, a doctor can see an overview of their own patients but not other patients, a doctor can change the data of a patient, and the data of a patient isn't linked directly to their account, but only referenced by their user id.

Another epic was the way we needed to deliver our system. Instead of using our Heroku service, we were asked to deliver it in docker containers.
So after a private lesson and some self study we created a private repo with our frontend and backend image. So you could easily pull and run our project.

Layered Architecture Package Diagram

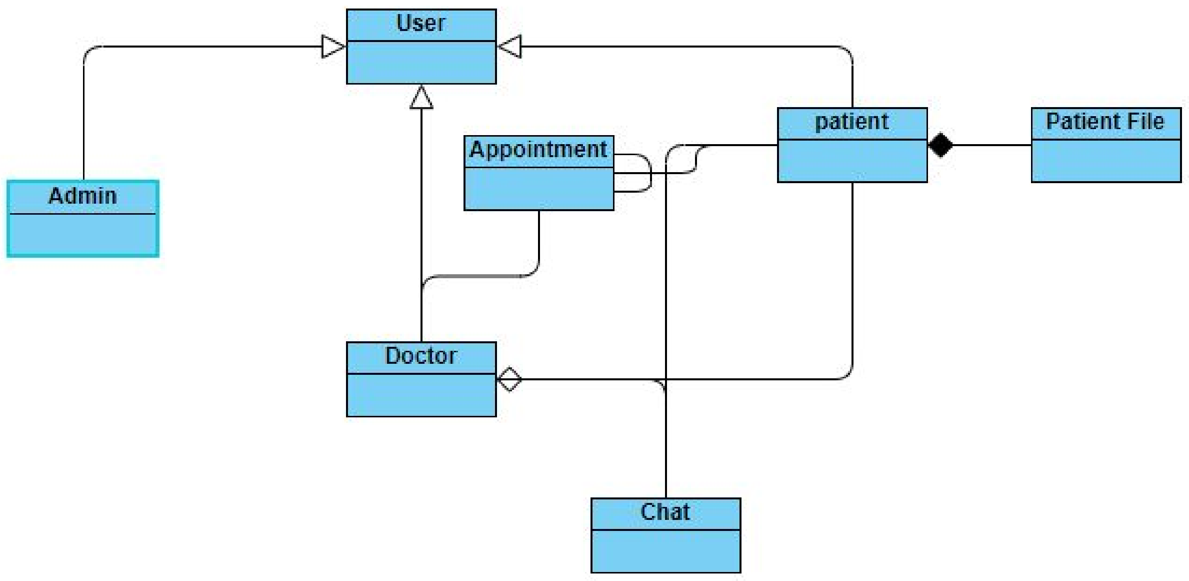


In the frontend we're working with the Angular Framework, so everything in the presentation layer is based on components interacting with each other. We inject the services in the components to communicate with the backend. By using HttpClient we can make the REST request and through the services we can display it on the components.

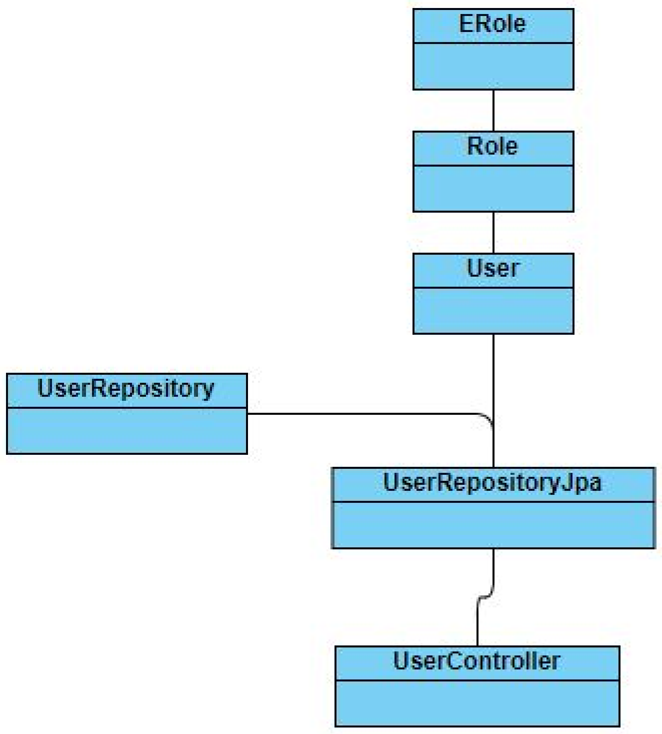


In the backend we are working with Spring Boot. The presentation layer is the controllers that open up the endpoints for the frontend to communicate through. In the endpoints we then get the Repositories managed with JPA to communicate with our MySQL database. We also have a security layer to manage incoming requests and make sure they are authenticated.

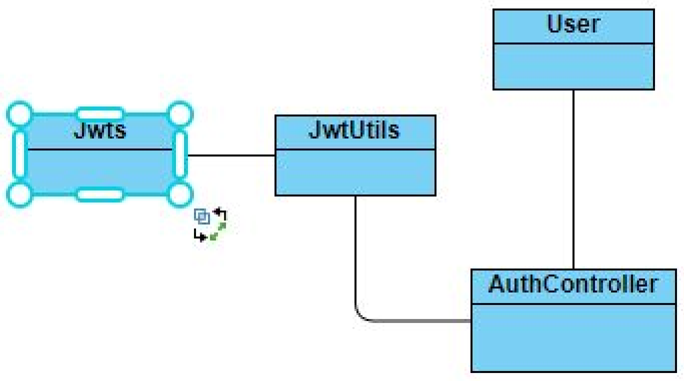
Navigable Class Diagram



Here is our domain model. Our solution is built around our user that can have three different roles. Those roles can be: admin, doctor or patient. When a person is a doctor or patient, he or she can create a chat between a doctor or patient. This person is also able to create an appointment. A patient will also have a patient file with all their medical information.



Here is an example from one of our repositories in the JPA persistence layer. We use an enum for roles that we can set in a user, so we know if a user is a doctor, patient or admin. then we have our UserRepositoryJpa that uses our UserRepository. The JpaRepository is connected to our Oege SQL database. Then we have our UserController that accepts our endpoints and does the processing.



Here is an example of an external interface. We use JWT for our authentication, together with Spring security. When a user logs in correctly, the user will get a JWT back to access the rest of our backend. This is needed because every endpoint except our authentication controller is protected until logged in.

Design theme and choices

An example of a choice made in the flow of using the product is the sign-up for a role. At first, a new user would register at the registration page and mark the right role there. A user can be a patient, a doctor or an admin.

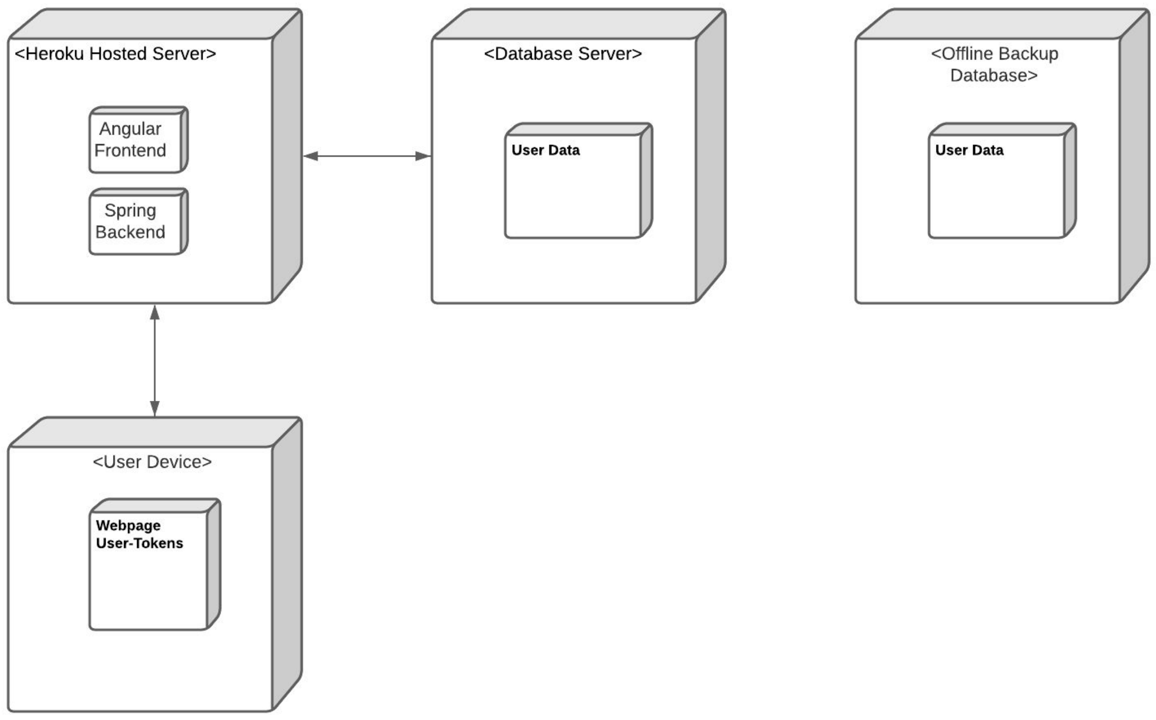
However, a quite significant security concern appeared. Because in this case, how do we determine if the person signing up as a doctor is actually a doctor? A doctor has different rights than a patient, such as viewing the information from other patients. Therefore, a user can register as a doctor, while not actually being one. This ‘doctor’ is then able to view the private medical information of other patients, harming privacy of other users.

To prevent this case, the team has decided to remove the option to check a role during sign-up. Instead, a separate admin page was created, which is able to modify the role of other users. A person has to be manually marked admin in the database at first. Afterwards, this person can mark other users as a doctor while at the admin page, making sure this is not done by the person himself. This prevents the situation described at the beginning by using a third party to be able to view the other information of other patients.

One of the other problems we had was that when a patient was registered he or she needed to be connected to a doctor. So we had two ideas, one would have the patient needing to fill in his or her doctor so she could immediately be coupled and then we knew for sure that the person creating this account is a patient from our practice. But then you would have the problem that people could create thousands of accounts and spam this doctor. Also, for new patients that have never visited the office and don’t have a doctor yet won’t be able to use our service. Therefore they need to come to the practice to get a doctor and we want to avoid that as much as possible.

That's why we chose for option two. When a patient creates an account he will not have the ability to chat or create appointments, until a doctor has added them to their patient list. A doctor can do this from their account and see all the patients without a doctor. Of course we do now run into the problem that a user won’t be able to use their account to their fullest until they have been added, but that is a choice we had to make.

Deployment Diagram



This diagram shows a proposal of a physical deployment of the production version of our application. The diagram is pretty common and as simplistic as possible. It only uses the necessities for our web application.

It comes down to a main server; hosted on Heroku. This will have our frontend and backend application where the user can connect to through their web browser. Connected to that server is a database, to store the necessary information about the user such as the doctors of Zonnevelt, Patients and their corresponding login credentials. Lastly we have the user device itself, be it a laptop, pc or smartphone. This will connect to the Heroku Server to connect to the web application.

An important addition though, is the fact that we added a backup database. This is because the data on this server will be of very high importance and a company as Zonnevelt can not afford to lose that data in for example a hack or a crash. Because of these reasons it is an offline database, which will be kept up to date regularly. We chose an offline database because of the fact that this will be less vulnerable to hacks.

Analytical Reflection

Overall our design aims to be simple and easy to use. This is due to the fact that many people need to be able to use our web application, such as elderly people or people with a vision problem. Because of this, we use a lot of images and simple buttons that state their corresponding action.

The website is also divided simplistic. Each user has their own navigation bar depending on their role with Zonnevelt. For example a doctor has a different navigation bar than a patient. This is done because a doctor needs to access different web pages and functionalities then a patient, and thus we can keep the navigation bar as clean as possible for each user.

We have not used a color scheme. We have used standard colors available and chose colors that fit well with a GP such as blue and grey. This could be an improvement for next time, where we discuss a certain color scheme for more matching pages.

At the start we had created a mockup and from that we worked our way to the final design. We experienced slight problems with that however, due to mostly the fact that much changed; functionality wise and thus design wise. Because of those changes it was a difficult challenge to keep our design as close to our mockup as possible. Our solution for that was to improvise a bit and see how it would turn out. This would be a big area of improvement for next time; create a mockup for the design and keep it up to date with the latest achievements and challenges.

An important feature we focussed on is the fact that our web application is compatible for all kinds of devices. This makes it easier for users to use the web application on mobile devices for example. In this day and age we find compatibility for multiple devices important, the chances that users will access our web application through their mobile device is big.

Our design is mainly focussed on functionality and overall we are pleased with the outcome; it’s practical and easy to use for young people but also elderly. It also scales well with most devices.