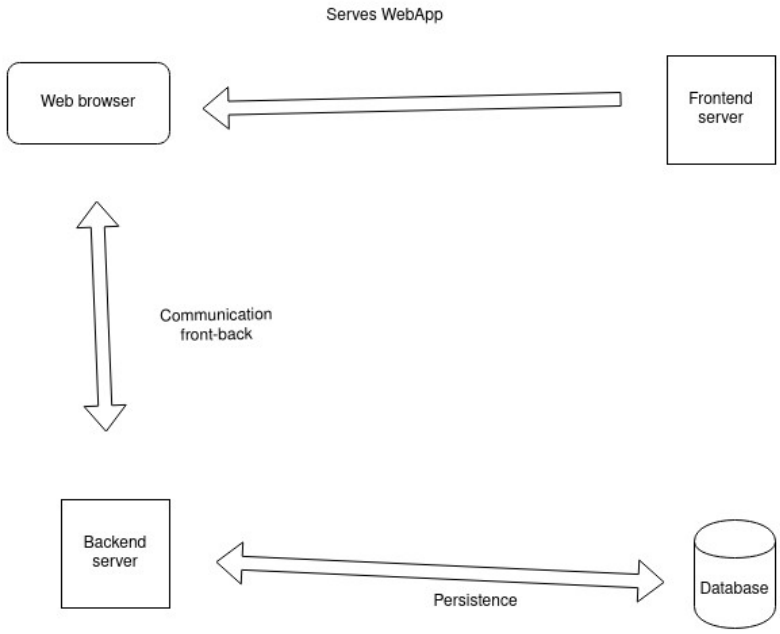
DS Project 1 Documentation

Conceptual Architecture of the distributed system

The architecture of the described distributed system follows the client-server architectural pattern. This implies that the project is divided into two large modules: “Frontend”, which serves the web application to the client and “Backend” which provides API endpoints, handles the logic of the system and ensures connection to the DB. Added to this two systems there are other two components present. The database , which enables persistent storage of the data in the system and the web browser, which hosts.



The Backend server architecture is mainly composed of :

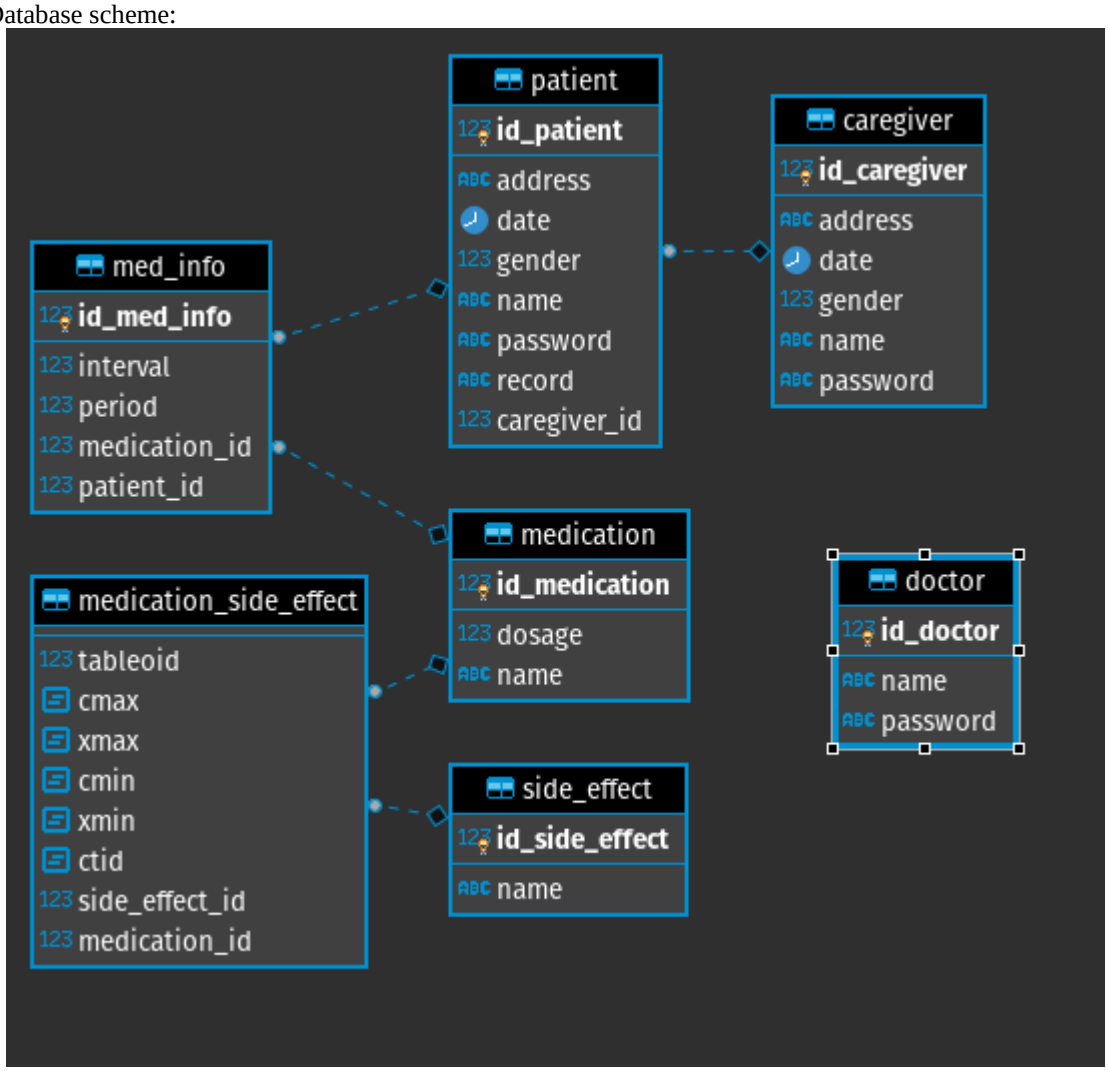
→ Model classes that are mapped to entities in the database through an ORM (Hibernate)
→ Repositories that give access to the entities.
→ REST controllers that establish communication with the Frontend.

The Frontend server architecture is manly composed of :

→ Services that , ensure the communication with the backend
→ Components that provide a graphical interface for the user, through which CRUD operations are made.

Database design

Database scheme:



Doctor:

→ A type of user

→ Has only the attributes required at login.

Patient:

→ A type of user

→ One to many relationship with caregiver (one patient can have multiple caregivers assigned to it)

→ Many to many relationship with medication (multiple medications can be assigned to multiple patients)

→ Has fields needed for login, fields with the information required by other users (personal data).

Caregiver:

→ A type of user

→ Many to one relationship with patient (multiple caregivers can be assigned to one patient)

→ Has fields needed for login, and some personal data.

Medication:

→ Many to many relationship with side-effects (multiple medications can have multiple side effects)

→ Has the availible dosage and a list of side effects attached to it.

Side-effect:

→ Many to many relationship with medication.

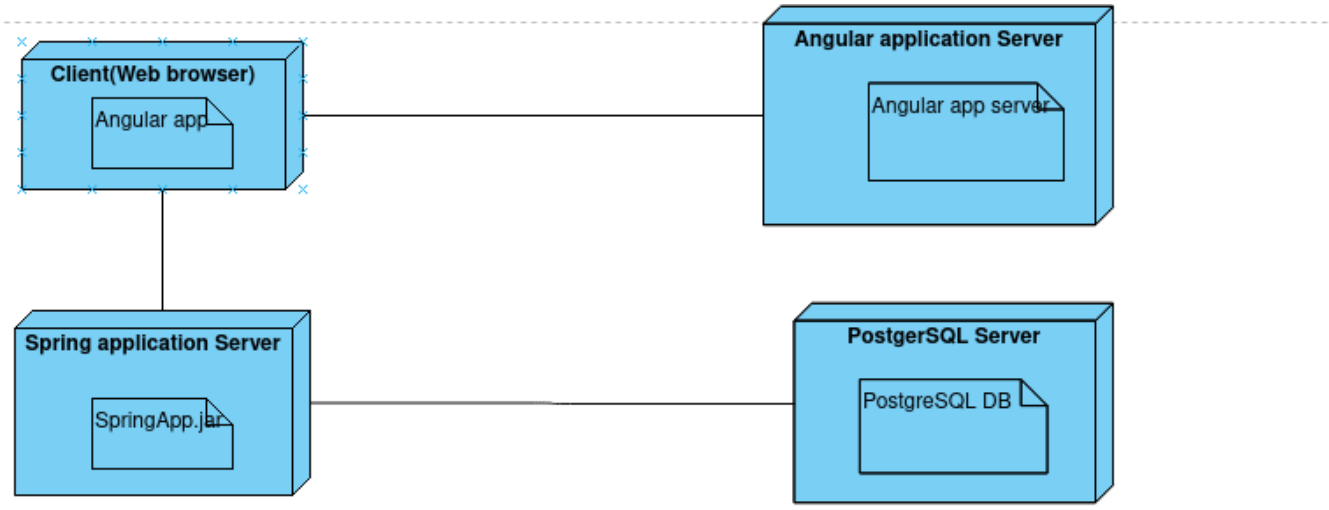
→ Has a name, and a list of medications that have the corresponding side effect.

The Database is created using the Hibernate ORM, which maps classes to entities automatically, and ensures the relationships required in the backend implementation.

I chose to impose a many-to-many relationship between medication and side effects because the most probably side effects are shared between medications, and so at a large-scale system it would be inefficient to have them separated. This also allows the system to store one concrete list of side-effects which is used in the Frontend.

UML Deployment diagram

The UML deployment diagram:



The UML deployment diagram represents the deployment process of our distributed system, through nodes.

→ The database server provided through Heroku hosts a PostgreSQL database.

→ The Application server hosts the Spring application

→ The web app server host the angular web application

Build and execution considerations.

Through maven, the app is built, providing the .jar to the deployment system.

The db configuration is set up through the application.properties config file, to be compatible with the DB provided by Heroku.

The angular application is built using node.

The applications are deployed using ruby.