DISTRIBUTED SYSTEMS

ASSIGNMENT 1

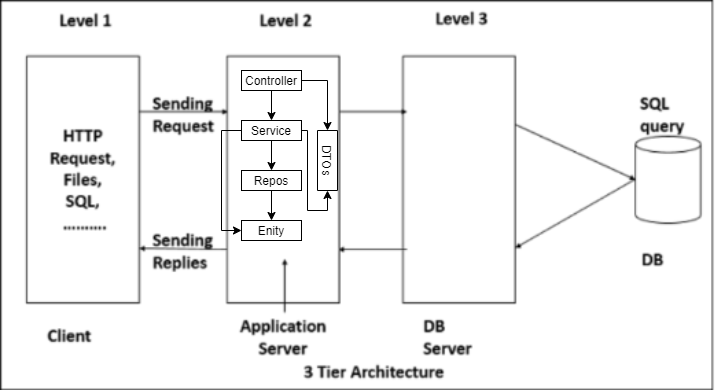
Pașc Andreea-Mădălina

Group 30441

1. Conceptual architecture of the distributed system

The assignment is built using a 3-Tier REST architecture which decouples the layers (presentation, business, database). The communication between the layers is done using the TCP/IP network stack and HTTP protocol.

The application has been implemented in IntelliJ in a Gradle project. It uses a server-side backend and a client-side application. The server application is implemented using Java and the Spring Framework with Hibernate and it communicates with the client-side and the database (MySQL). The client application is implemented using JavaScript and Vue.js frontend framework. The conceptual architecture of the distributed system is the following:



The communication between the application sides is done through GET, POST, PUT and DELETE requests in the following way: we make a request from the frontend to the backend which will be handled in the controller; it will call the service in order to execute the desired functionality; the methods implemented in the services use repositories which are needed in order to access the information from the database; we also have entities and dtos which are responsible for data transferring between different parts of the application. We have the following components: User, Device, Sensor (as well as Role and ERole used for the User), Consumption where we stock the monitored values and a Security packet where we tackle the security between the admin and clients. Each one has an entity class, repository, service and controller.

1. DB Design

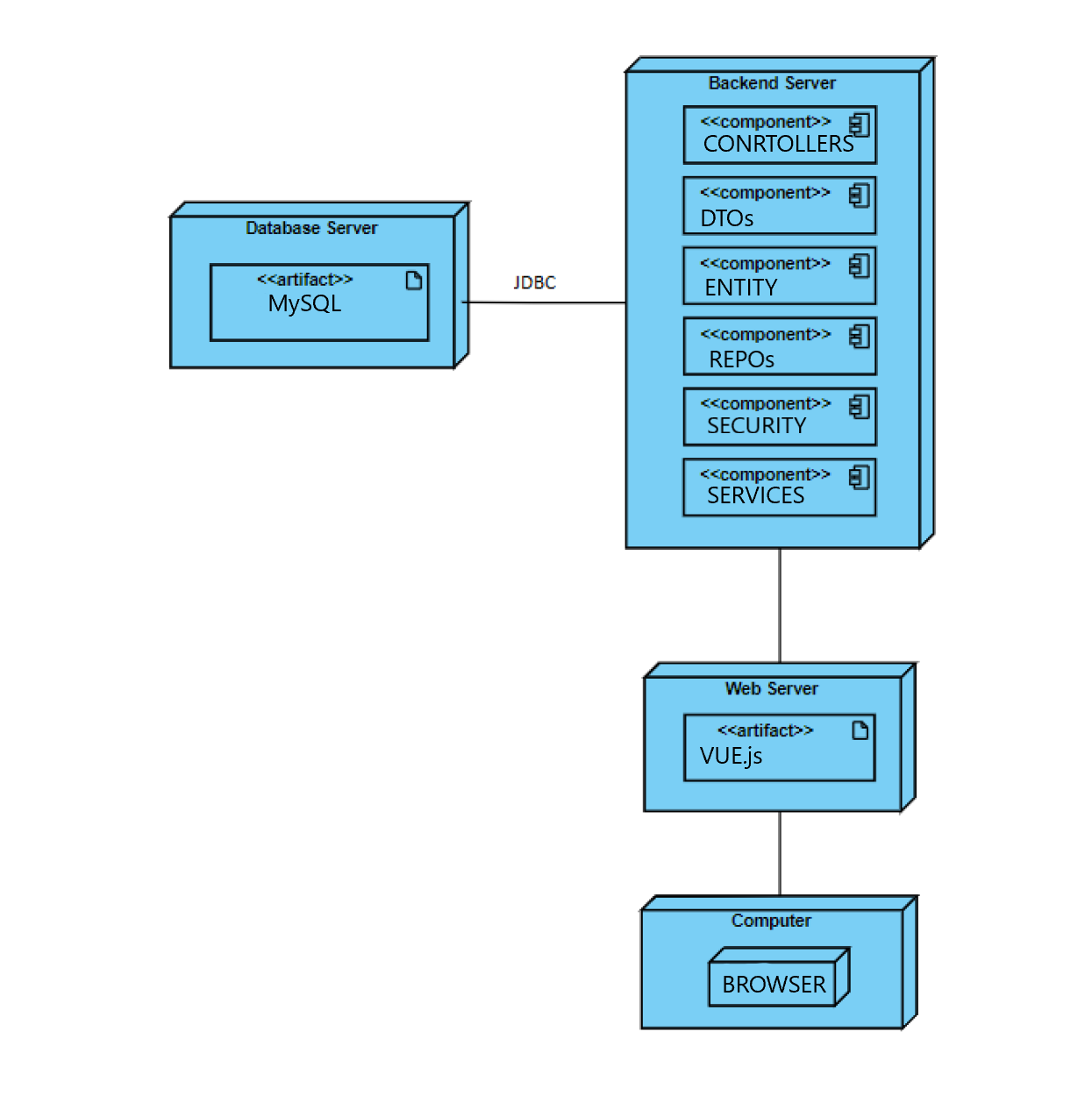
The database is generated by Java Spring and Hibernate using the definitions of the entity classes and it consists of 5 tables (user, role, device, sensor, consumption), plus the relation table user\_roles between the roles and the users table. The user table contains the id, address, age, email, password and username. The devices table contains id, average\_energy\_consumption, description, location, max\_energy\_consumption, sensor\_id, user\_id. The role table contains the role\_is and the name of the role. The sensor table contains id, description, max\_value. The consumption table contains id, energy\_consumption, sensor\_id and timestamp. The relationships are:

* Many-To-Many relationship between user and role => many roles can be taken by many users.

A picture containing graphical user interface

Description automatically generated

1. UML deployment diagram



1. Readme file containing build and execution considerations

The project can be run both locally and on the Heroku server but we need to make a few modifications. For the local part, we need the MySQL database, we need to update the application.properties with the right credentials for user, password and name of the database. The host.js file needs to be ‘http://localhost:8088/api’. After these changes, we can run the backend using the run button and the frontend with npm run serve/npm start command.

For Heroku, we have to set up the pipeline and the projects as in the tutorial presented in the laboratory, change the database information for both backend and frontend and update the gitlab-ci.yml files so that it points to the correct application in Heroku. In the Gitlab repository we need to create the API key obtained from Heroku. Also, we need to add the backend’s url in the hosts.js file from the frontend. The source code is pushed on the master branch and it will trigger continuous deployment on Heroku. Using the “Open App” buttons we can start the backend and frontend applications and the login page should appear.