**FACULTY OF COMPUTER AUTOMATION AND COMPUTER SCIENCE**

**DISTRIBUTED SYSTEMS**

Online Energy Utility Platform

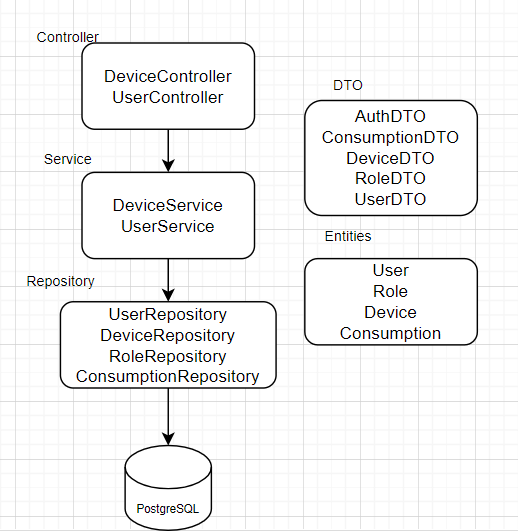
Dunca Denisa-Mihaela, gr: 30643

**Conceptual architecture of the online platform**

The project “Online Energy Platform” is a full stack application that represents a useful way for users to access information about their devices and about the energy that every device consumes by accessing the energy consumption charts and it provides an easy way of adding new users and devices for the administrators. The project is made in principle using Java and SpringBoot on backend, React Js on frontend and PostreSQL for the data base. The project is at the state of local deployment by using Docker.

The application has a login page that provides access to users and administrator by inserting the credentials, it also provides a registration for the users that don’t have yet an account. This application has two kinds of users, the clients that can easily see their devices and their energy consumptions by device, on a specific chosen date, and the administrators which can see, modify, edit, delete, and add users and devices and they can also map devices to users.

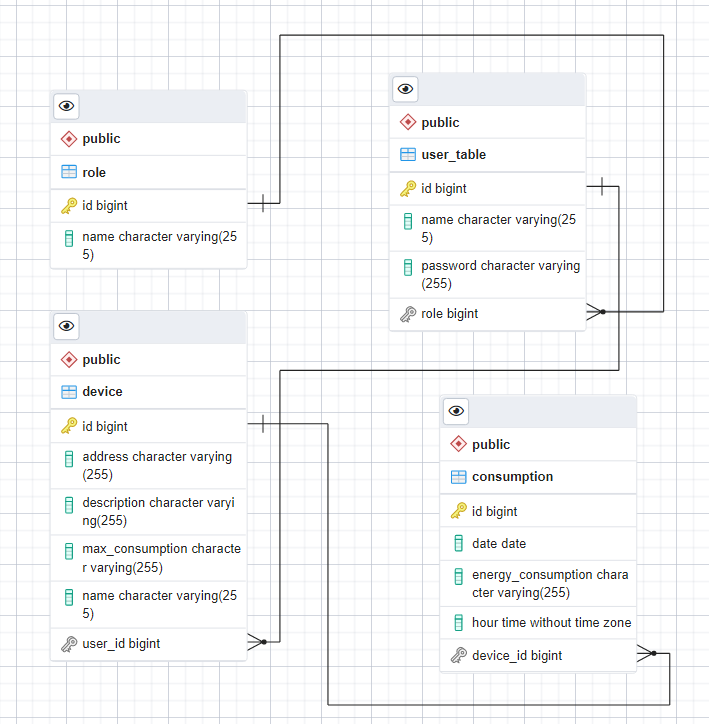
In order to create the project, I used the Layered Architecture which is a way the project is organized and is made out of four big categories of layers: presentation layer, application layer, domain layer and infrastructure layer. The presentation layer contains all the classes responsible for presenting the user interface and sends the response back to the client (Controller layer). The application layer contains all the logic in the application, all the functionalities (Service layer). The domain layer represents the entities and the business rules (Repository layer), and the infrastructure layer contains the classes responsible for the technical stuff, in this project we have DAO classes, and entities form Model.



**Data Base Design**

The database contains four tables, two of them are related to the users and the other two are related to the devices. The Role table contains the name of the role (user, admin) and the id for each role so that it can be mapped to the users. The User table contains the minimum information about the users (name and password) and the id of the role that each user has. Between Role table and User table there is a “one to many” relationship.

The Devices table contains all the information related to the devices such as the name, description, address, the maximum energy consumption and the most important the user ID, which is by default null so that the admin can map the device to the user. And the last table, the Consumption table contains all the energy consumption mapped by a timestamp (date and hour) and the device ID. Between Devices table and Consumption table there is a “one to many” relationship and also between User and Device table.

****

**RabbitMQ**

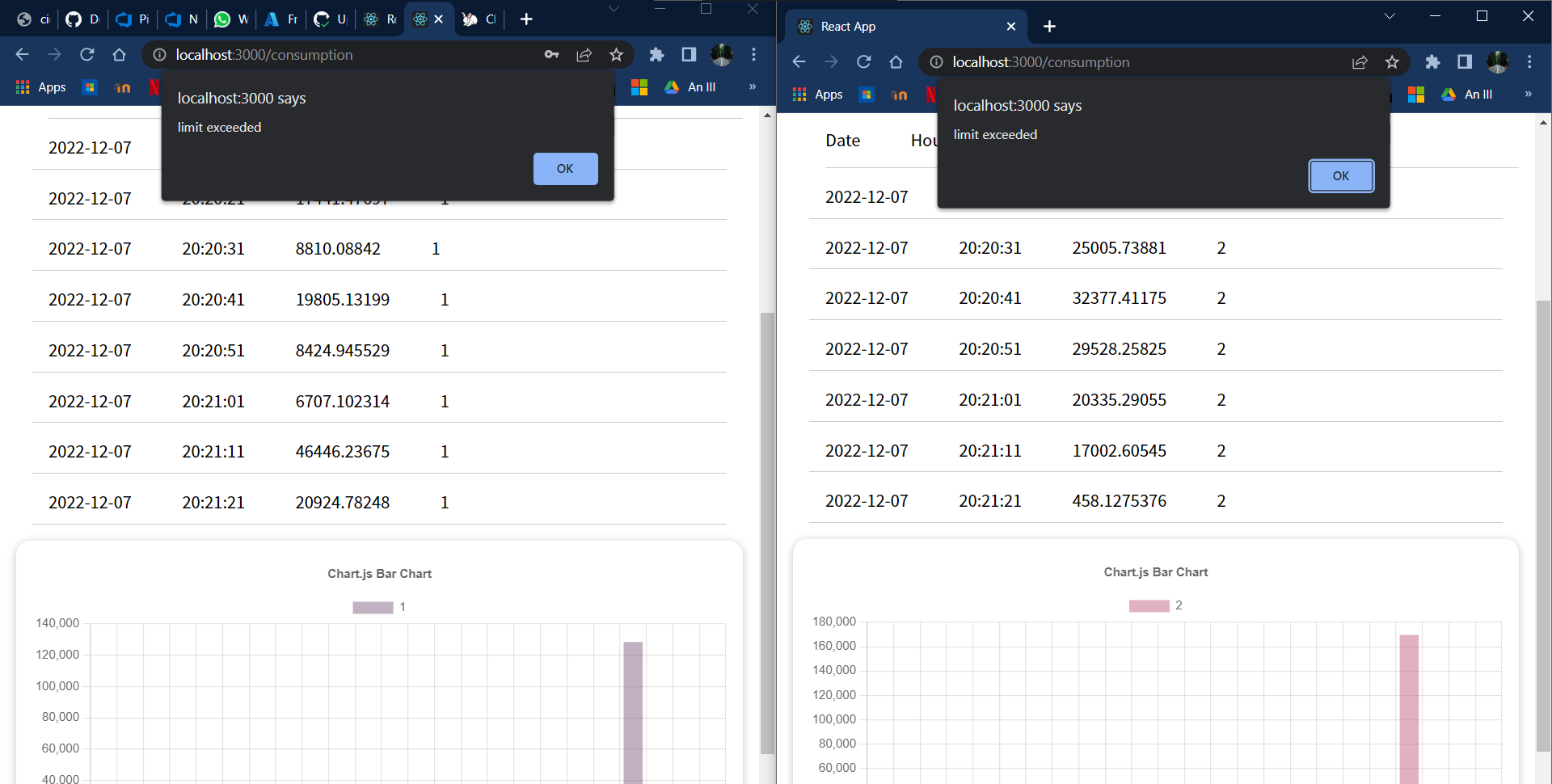
For the RabbitMQ part I created a new Java project Message Produces. In this application I made a method to read the given csv with the energy data and I made the connection to the Cloud RabbitMQ using the AMQP credentials. I also created a string in JSON format that is being send as the message in the queue and a configuration file from which I take the device Id to be send to the queue. In the Backend application I created a new Class called Message Consumer which connects to the Cloud RabbitMQ and gets the message from the queue. That message is being transformed and saved in the data base by the Message Broker in the consumption table.

**WebSocket**

In the backend I created a class called WebSocket config that implements WebSocket Message Broker Configurer which send a notification to the frontend when someone is connected.

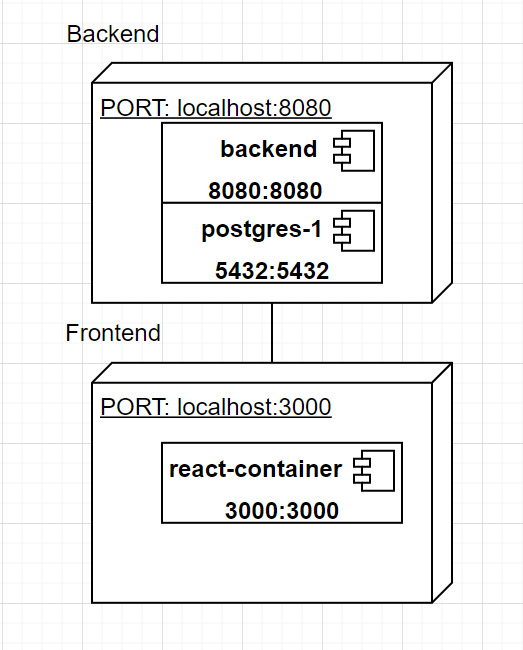
O imagine care conține text

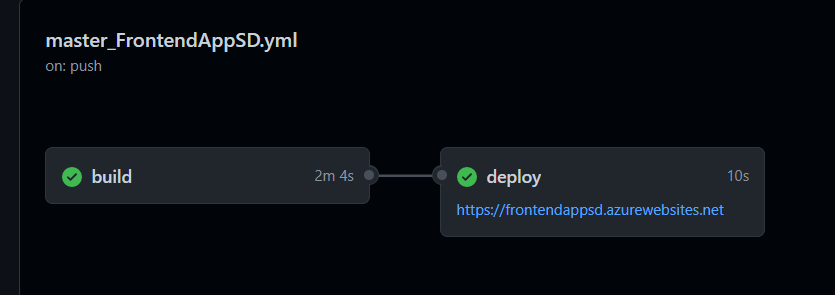
Descriere generată automat

 In the frontend I added in the useEffect for the consumption page the SockJS connection. The consumption is being updated every time when the Message Produces sends a message to the queue, and when the maximum energy consumption is passed an alert is showing on the page. I managed to do this for two users in parallel using threads on the Message producer application.

**UML Deployment diagram**

For the deploy I used Docker, and I created two containers, one for the backend and data base and one for the frontend. The backend was implemented using Java and Spring Boot and the database used was PostgreSQL and for the frontend I used React JS.

****

****

**O imagine care conține text

Descriere generată automat** For the second Assignment, I managed to deploy the frontend part and backend part on Azure DevOps using Github Repository Actions. But because I could not manage to deploy the data base in order for the app to work, I still had to access the backend from the local docker deploy. ☹

**Chat**

For the third Assignment I implemented a chat between users and admin. The admin interface contains a list with the names of the users that sent a message. He can select from that list the user to chat and a chat box opens to see the message received from user. When the opens the chat box the admin receive a notification that says that the user seen the message and when the user is typing, the admin receives a notification that says that the user is typing.

The interface for the user contains a chat box that sends messages directly to the admin When the admin opens the chat box, the user receives a notification that the admin has seen the message. When the admin is typing, the user will receive a notification that says that the admin is typing.

Text

Description automatically generatedThe chat was entirely implemented using web-sockets in order to send and receive real-time notifications. I used the before explained Web-socket configuration and I created a series of methods in the UserServiceImp class.

Graphical user interface, application

Description automatically generatedMethod implemented for sending the message to admin or user using web-sockets.

**JWT Security**

Text

Description automatically generatedI also implemented a JWT Security (JSON Web Token), so that every Rest Call can be called only when the user is authenticated by using the token that is generated by the authorization call. Firstly, I added the spring-boot-starter-security dependency which made the authentication with username and password default for the spring application. Secondly, I created a class AuthenticationController that contains the only request that does not require a token because it is creating it.

After that I created three classes for configurations:

* The JwtAuthFilter class extends OncePerRequestFilter and using the doFilterInternal method to intercept every request that is coming on the server and verifies if the bearer token that comes with the call is valid and then proceeds with the actual call.
* The JwtUtils class generate the token for the application and contains the method that verifies if the token valid.
* Text

  Description automatically generatedThe SecurityConfig matches the http request, handles cors and csrf, allows security.

I also modified the model for UserTable to import UserDetails Interface and overrides its methos such as getUsername and getPassword in order to use the security service.

Text

Description automatically generated For the client side, in the frontend, I added to each request an Authorization Token header.

**Build and execution considerations**

I used firstly the Intellij IDE to implement the code on the backend and PGAdmin4 to create the data base, I also used Postmen to test the API calls and for the frontend I used WebStorm. You can start the application by running these two projects and by creating a specific database. After creating all the functionalities needed to this project, I did the deployment locally by using Docker desktop.

For both, frontend and backend I created a Dockerfile and a docker-compose.yaml. Firstly, I built the docker image by running docker-compose build command, and finally I run the docker-compose up command to create the container and run the application. After that you can also run the application from Docker Desktop.

For the second Assignment in order to access the application you need to access the following link: <https://frontendappsd.azurewebsites.net> and run the backend from Docker.