ASSIGNMENT 3

ENERGY MANAGEMENT APPLICATION

Tat Teodora Group 30433

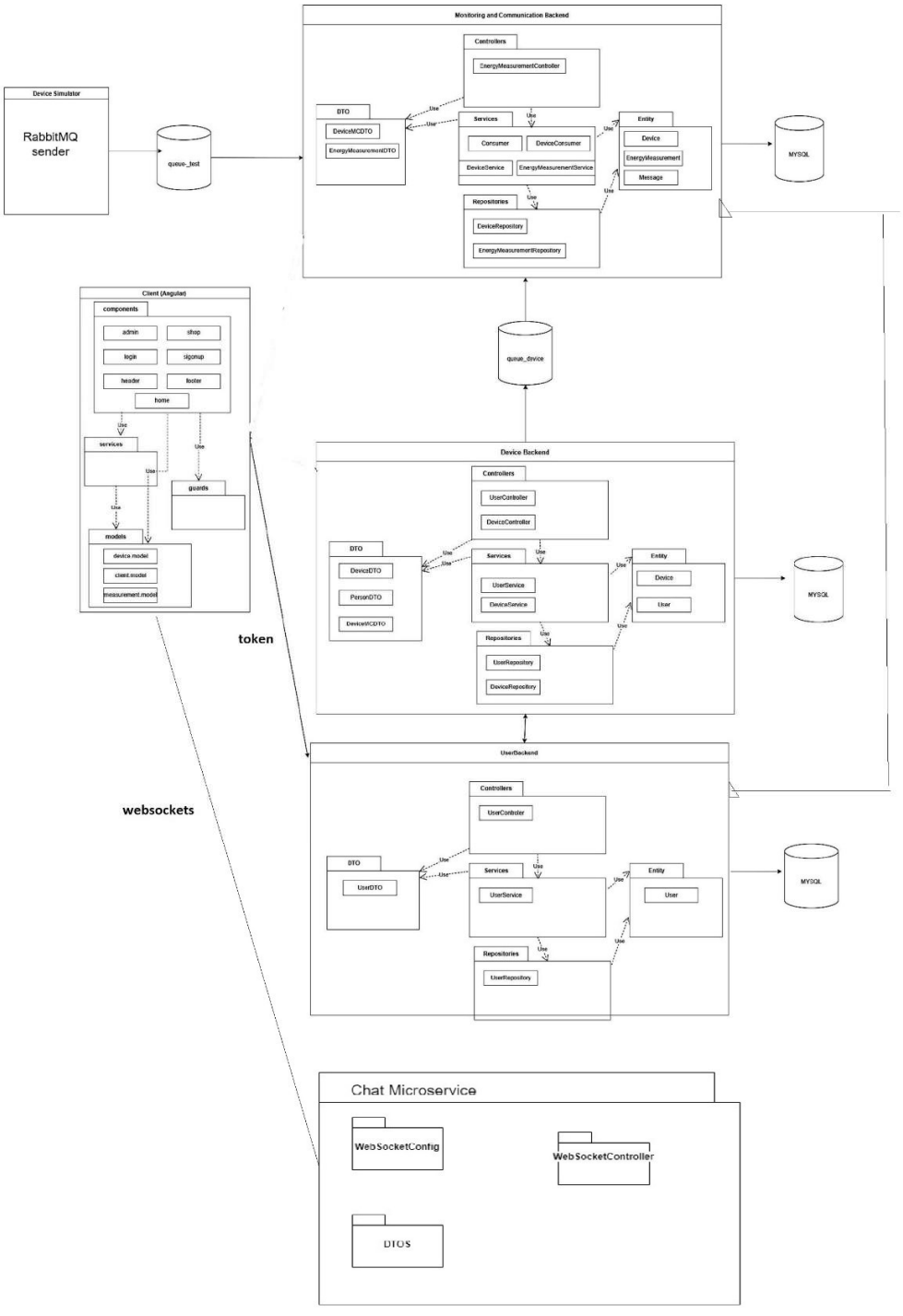
CONTENTS

1. Conceptual architecture of the distributed system

- Repositories : The package encompasses the repositories and classes designed to streamline database access, enabling developers to employ customized queries for effective communication with the database.
- Entities: Corespond to a table from the relational database
- Services: This layer serves as the business logic layer in the Spring application, responsible for translating Data Transfer Objects (DTOs) into entities and vice versa.
- DTOs: pecialized object designed for external exposure, typically to the user interface (UI) or APIs. It encapsulates elements of the underlying Entities or combinations of various entities and may also include builders and validators.
- Controller: layer is responsible for exposing the application's functionality as an API capable of processing HTTP REST requests. Additionally, it includes handlers for various types of exceptions.

- COMMONS: includes functionality that is used by all the other components such as functionality performing HTTP requests, implementing error handling components, etc.
- Modules: defined for each route consisting of a component container,a set of child components and an API used to send HTTP requests to the spring app.

The Angular application communicates with the spring applications through the User Microservice.

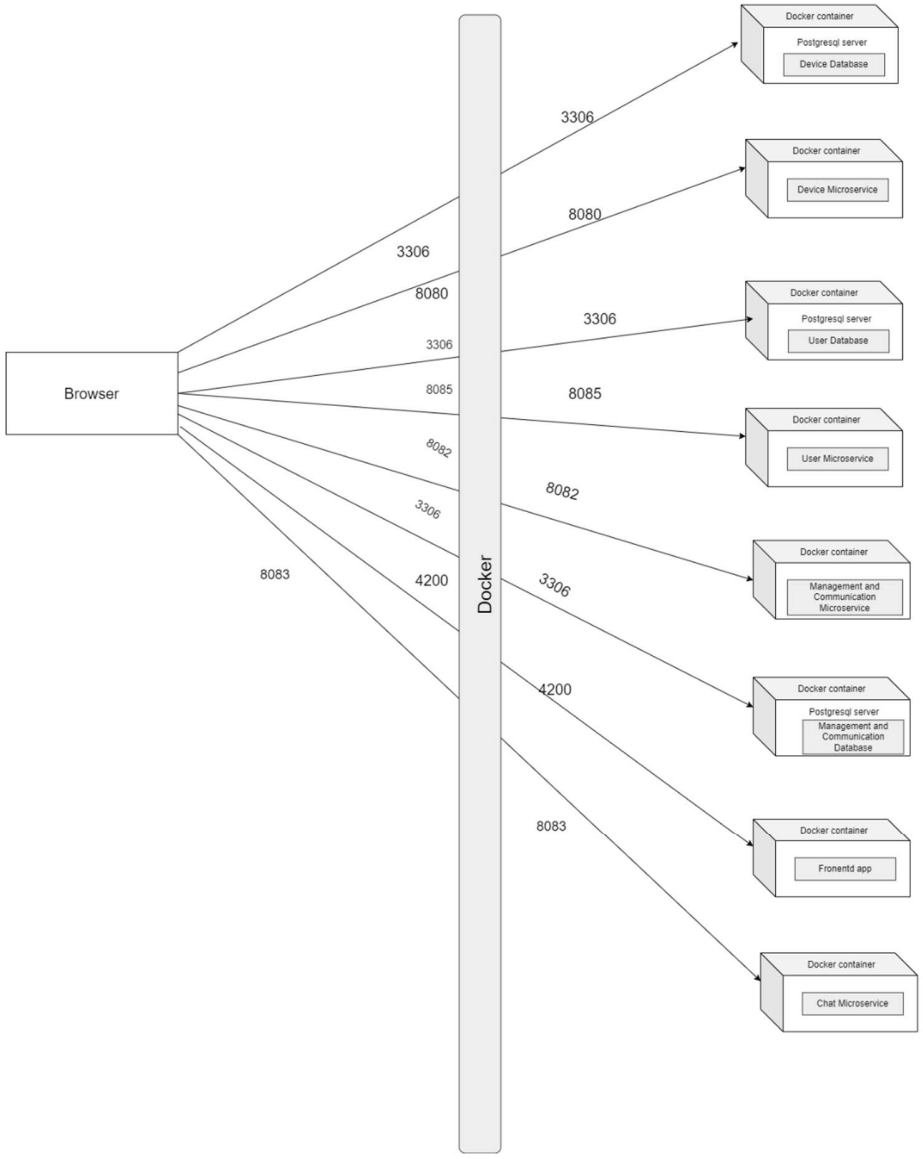

Monitoring and Communication Backend
Controllers
EnergyMeasurementController
DTO
DeviceMCDTO
EnergyMeasurementDTO
Services
Consumer
DeviceConsumer
DeviceService
EnergyMeasurementService
Entity
Device
EnergyMeasurement
Message
Repositories
DeviceRepository
EnergyMeasurementRepository
MYSQL
Device Simulator
RabbitMQ sender
queue_test
queue_device
Client (Angular)
components
admin
shop
login
signup
header
footer
home
services
guards
models
device.model
client.model
measurement.model
token
Device Backend
Controllers
UserController
DeviceController
DTO
DeviceDTO
PersonDTO
DeviceMCDTO
Services
UserService
DeviceService
Entity
Device
User
Repositories
UserRepository
DeviceRepository
MYSQL
UserBackend
Controllers
UserController
DTO
UserDTO
Services
UserService
Entity
User
Repositories
UserRepository
MYSQL
websockets
Chat Microservice
WebSocketConfig
WebSocketController
DTOS

2. UML Deployment diagram

The application composed of the backend, frontend and database will be deployed in docker as showed by the architecture in the image below.

On the host computer runs the docker runtime that will host five containers, one for each application:

- The Docker container for the frontend react application utilizes an NGINX server and is configured to route local port 4200 to port 4200 on the host computer.
- The Docker container for the user microservice backend applications and directs local port 8085 to port 8085 on the host computer.
- The database container, containing the MySQL server,for the user, maps local port 3306 to port 3315 on the host computer.
- The Docker container for the device microservice backend applications and directs local port 8080 to port 8080 on the host computer.
- The database container, containing the MySQL server,for the device, maps local port 3306 to port 3309 on the host computer.
- The Docker container for the management and communication microservice backend applications and directs local port 8082 to port 8082 on the host computer.
- The database container, containing the MySQL server,for the device, maps local port 3306 to port 3310 on the host computer.
- The Docker container for the device microservice backend applications and directs local port 8083 to port 8083 on the host computer.



3. Read me file

The Energy Management Application allows users to manage devices and associated information.

To begin, users can sign up and create their own unique accounts. Once an account is created, they can log in to access a dashboard that provides an overview of all the devices linked to their account. However, the capabilities

available to standard "client" users are carefully secured to ensure security and maintain the integrity of the system; they are limited to basic operations and viewing functions.

In contrast, the application designates special roles such as "admin" and "client." Admins have a broader range of permissions, enabling them to perform a variety of operations on devices across all accounts. This includes adding new devices, viewing all devices registered by any user, updating device details, and even removing devices as necessary. Managers share similar privileges, but their focus is on user management, allowing them to modify user information or remove user access when required.

Upon successful authentication, each user type is directed to a home page for their own role, with a navigation menu at the top bar to access different categories of functionality specific to their permissions. There is also a side menu for the admin, which includes crud operations he can perfom. This user interface design simplifies the user experience, ensuring that each user encounters only the features relevant to their role. Additionally, for added security and convenience, a logout option is readily accessible within this top bar menu.

If the user tries to access a page he is not allowed to, he will be redirected to a page wich contains a message that explains why he cannot access this page.

If someone who is not logged in tries to access a page other than home, he will be redirected to the login page, in order to login to access that page.

Architecturally, the application is compartmentalized into three primary components:

- User Microservice: This backend service handles all user-related operations, including account creation, authentication, and user management. It operates independently, interfacing with the other components via well-defined APIs.

- Device Microservice: A separate backend service dedicated to managing device information. It allows for the addition, update, and deletion of device details and can handle large volumes of device data efficiently.

- Angular Frontend: A modern, responsive user interface built with Angular. It communicates with both the User and Device microservices to provide a seamless user experience.

I also developed a Monitoring and Communication Microservice for the Energy Management System, employing a message broker middleware to collect data from smart metering devices. This microservice processes the gathered data, computes hourly energy consumption, and stores the results in its database. Synchronization between the databases of the Device Management Microservice and the new Monitoring and Communication Microservice is achieved through an event-based system utilizing a topic for device changes. This system sends device information through a queue to the Monitoring and Communication Microservice.

To simulate smart metering devices, a Smart Metering Device Simulator application will be implemented as the Message Producer. This simulator reads energy data from a sensor.csv file (with one value every 10 minutes) and sends data in the form of <timestamp, device_id, measurement_value> to the Message Broker queue. The timestamp is derived from the local clock, and the device_id is unique to each instance of the simulator, corresponding to the device_id of a user from the database (as defined in Assignment 1)

There is also a chat microservice. The "Chat microservice" allows for asynchronous communication between users and an administrator. Users can send messages through a front-end chat box, which the administrator

receives along with the user's identifier. This enables two-way communication during a chat session. The administrator has the capability to chat with multiple users simultaneously. Both parties receive notifications when their messages are read and also when the other party is typing a message.

The user microservice is also considered a authorization microservice. One authorization service that generates tokens and is configured as reverse proxy for all services. The services behind the reverse proxy do not need authorization,since they are protected from outside access by the LAN

Deployment and runtime management of these components are done by Docker, a platform that enables the application to be containerized. Each component of the application is packaged into its own Docker container, allowing for easy deployment, scaling, and management. Docker ensures that the application runs consistently across different environments by encapsulating the application and its environment. This means developers can focus on building features rather than worrying about inconsistencies between development, staging, and production environments.

Docker is run by first building the images for each component using the command “docker build -t image-name .” for each component. After each image is created, the command “docker compose up” is called in order to start the containers.