ASSIGNMENT 2:

Asynchronous Communication

and Real-Time Notification

Tat Teodora

Group 30411

1. **Introduction**

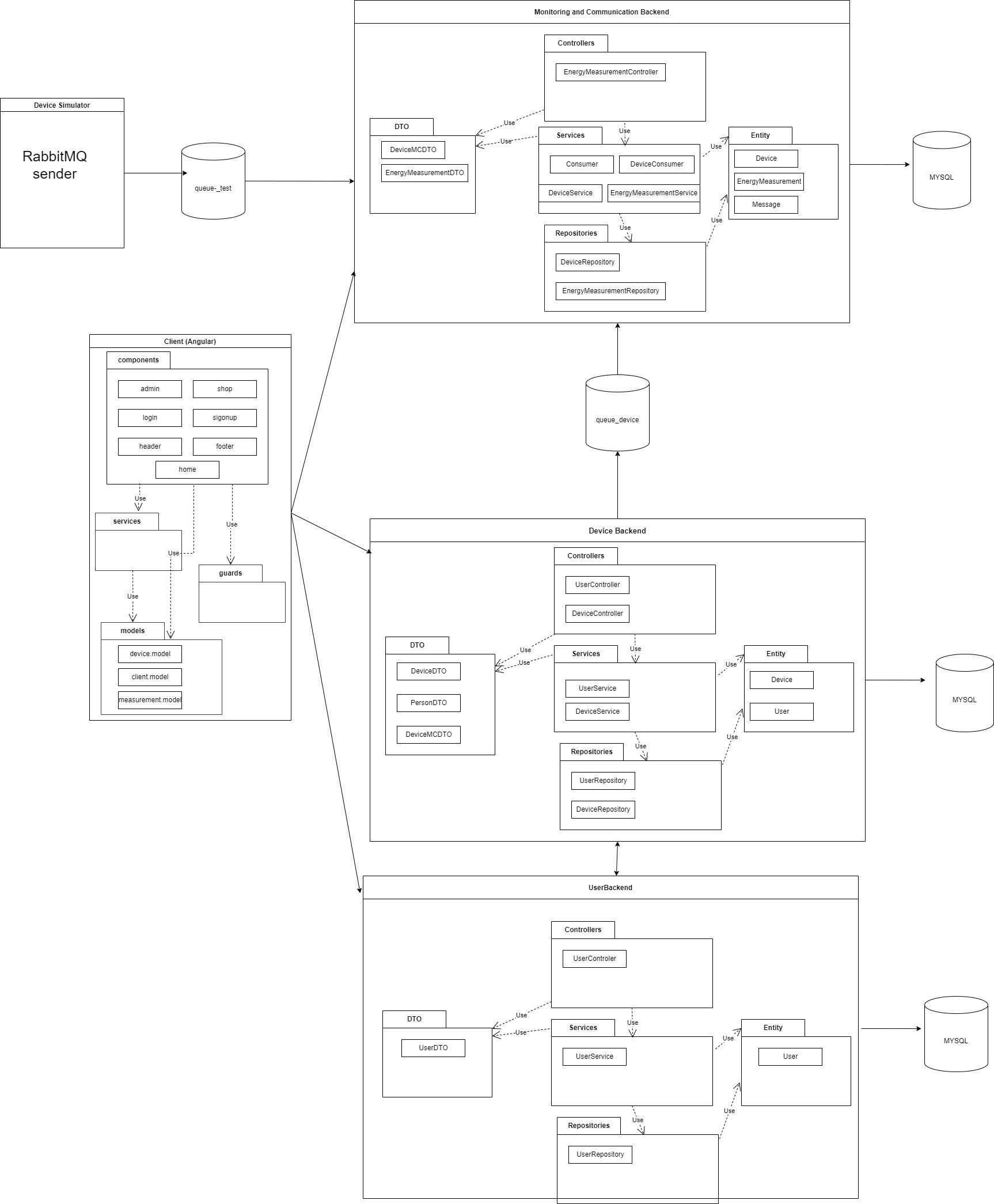
The task involves implementing a Monitoring and Communication Microservice for an Energy Management System. Using RabbitMQ for message-oriented middleware and WebSockets for asynchronous communication, the microservice collects data from smart metering devices, computes hourly energy consumption, and stores it in a database. Synchronization between databases is achieved through an event-based system using a designated topic for device changes.

A Smart Metering Device Simulator, reading energy data from a sensor.csv file, is employed to test the microservice. The Monitoring and Communication Microservice includes a Message Consumer component responsible for processing measurements and triggering real-time notifications to users when energy consumption exceeds predefined limits.

The assignment emphasizes documentation, requiring a solution description document with conceptual architecture, UML deployment diagrams, and essential build considerations.

Evaluation covers key concepts such as Message Oriented Middleware types, Queue vs. Topic, Point-to-Point vs. Publish Subscribe communication, and server pushing data to clients.

1. **Conceptual architecture of the distributed system**



Repositories: This package consolidates components dedicated to database interaction, allowing developers to utilize tailored queries for efficient communication with the database.

Entities: These are representations of tables in a relational database, mapping the structure of the database tables to corresponding objects in the application.

Services: Positioned as the business logic layer in a Spring application, services are responsible for translating Data Transfer Objects (DTOs) into entities and vice versa. They encapsulate the core logic of the application.

DTOs: Specialized objects crafted for external exposure, commonly to the user interface (UI) or APIs. DTOs encapsulate elements of underlying entities or combinations of various entities, often incorporating builders and validators for data integrity.

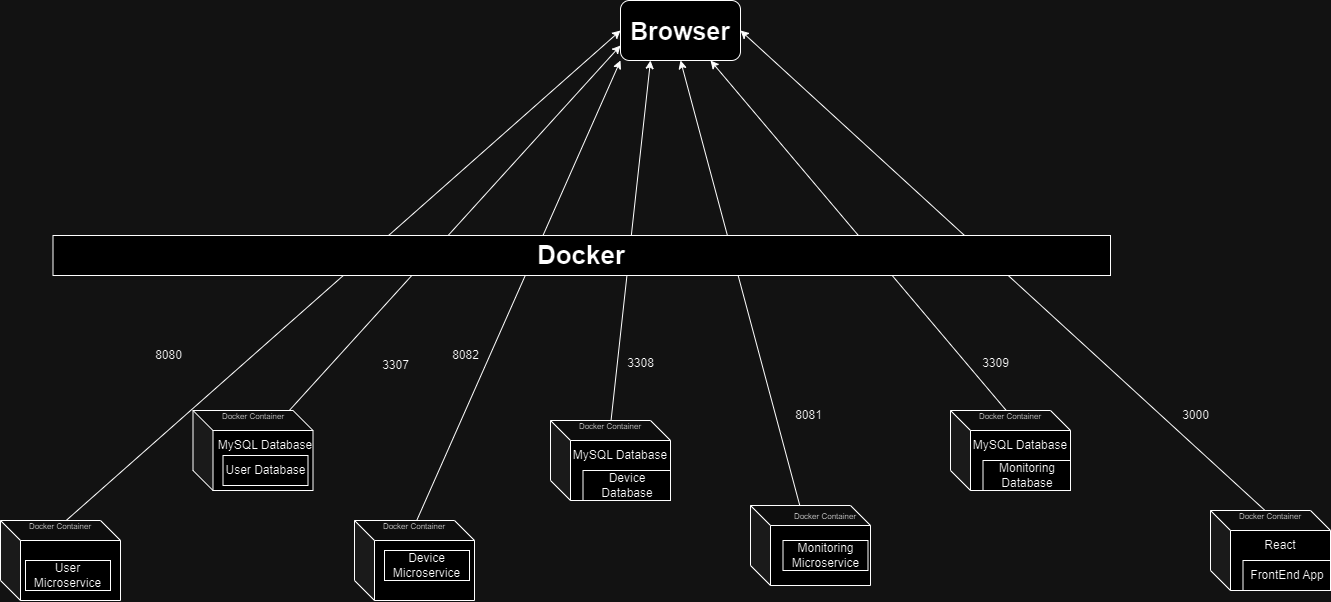
Controller: This layer exposes the application's functionality as an API capable of handling HTTP REST requests. Additionally, it includes handlers for various types of exceptions, facilitating communication between the client and the server.

COMMONS: This section encompasses functionalities shared across all other components, including modules for HTTP request handling, error management components, and other commonly used utilities.

Modules: These are defined for each route in the application, comprising a component container, a set of child components, and an API used for sending HTTP requests to the Spring application. Modules enhance organization and encapsulation within the overall system architecture.

The React application communicates with the Spring application through an API (Application Programming Interface). This API serves as a bridge, enabling seamless interaction and data exchange between the frontend (React) and backend (Spring) components. The communication typically occurs over HTTP, and RESTful APIs are commonly used in this context.

1. **UML Deployment diagram**



The entire application, consisting of the backend, frontend, and database components, will be deployed using Docker, as illustrated in the architecture depicted in the image below.

On the host computer, the Docker runtime is active and responsible for hosting five containers, each dedicated to a specific application:

1. The Docker container for the frontend React application utilizes an NGINX server and is configured to link local port 3000 to port 3000 on the host computer.

2. The Docker container for the user microservice backend applications is configured to link local port 8080 to port 8080 on the host computer.

3. The database container, housing the MySQL server for user-related data, maps local port 3306 to port 3307 on the host computer.

4. Similarly, the Docker container for the device microservice backend applications directs local port 8082 to port 8082 on the host computer.

5. The corresponding database container, containing the MySQL server for device-related data, maps local port 3306 to port 3308 on the host computer.

6. Finally, the Docker container for the management and communication microservice backend applications directs local port 8081 to port 8081 on the host computer.

7. The database container for this microservice, with the MySQL server handling management and communication data, maps local port 3306 to port 3309 on the host computer.

This deployment strategy in Docker provides encapsulation for each application, allowing them to run independently with their required configurations and dependencies, thus ensuring a streamlined and efficient deployment process.