# SISTEME DISTRIBUITE

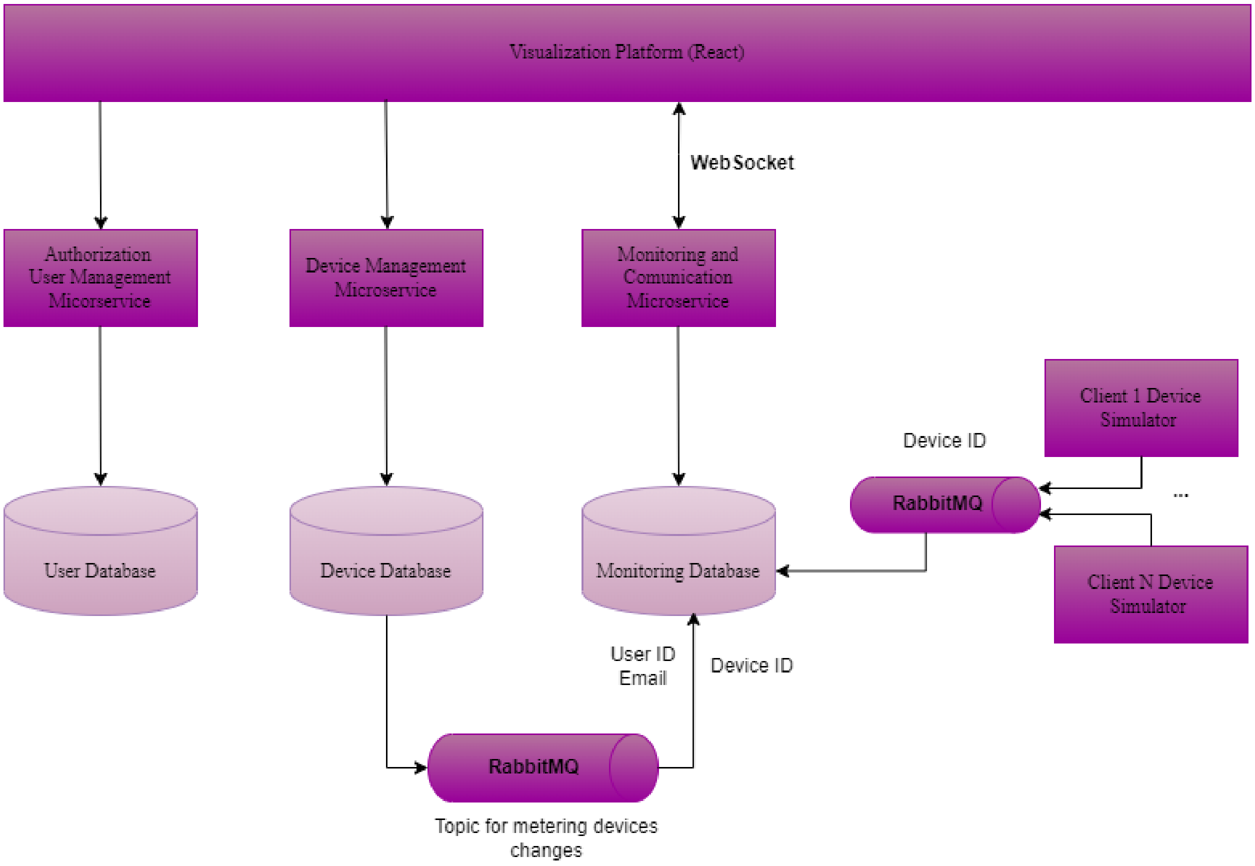
# Assignment 2

# Asynchronous Communication and Real-Time Notification

**Student: Gesica Goron**

**Grupa: 30242**

## 1. Conceptual Architecture of the Distributed System



Visualization Platform (React)
WebSocket
Authorization User Management Micorservice
Device Management Microservice
Monitoring and Comunication Microservice
Client 1 Device Simulator
Device ID
RabbitMQ
...
Client N Device Simulator
User Database
Device Database
Monitoring Database
User ID Email
Device ID
RabbitMQ
Topic for metering devices changes

The distributed system is composed of **three primary microservices** and a **React-based visualization platform**. It is designed to manage users, devices, and monitor energy consumption. Communication between microservices occurs via **HTTP APIs** and **RabbitMQ** for asynchronous messaging.

### Components and Responsibilities

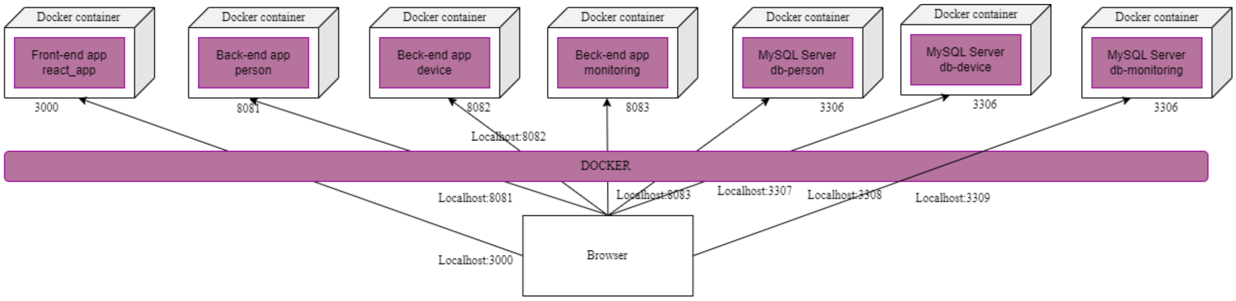
1. **Authorization and User Management Microservice**
    - Handles user authentication, authorization, and user CRUD operations.
    - **Database**: MySQL (User Database).
2. **Device Management Microservice**
    - Manages devices, including adding, editing, and deleting devices.
    - Provides APIs for fetching devices assigned to users.
    - **Database**: MySQL (Device Database).

3. **Monitoring and Communication Microservice**
    - Collects and stores hourly energy consumption data from device simulators.
    - Processes alerts when devices exceed the maximum hourly energy consumption.
    - Publishes device energy consumption changes to **RabbitMQ** for communication with other services.
    - Provides a WebSocket endpoint to send notifications to the React application.
    - **Database**: MySQL (Monitoring Database).
4. **Visualization Platform (React App)**
    - User interface for interacting with the system.
    - Admin users manage devices and users.
    - Client users view their assigned devices and energy consumption data in charts.
    - Receives **real-time alerts** via WebSocket from the monitoring service.

**Communication**

- **Synchronous Communication**: Microservices use RESTful HTTP APIs to interact with each other.
- **Asynchronous Communication**: Device simulators send energy consumption updates to RabbitMQ.
- **WebSocket**: Real-time notifications are pushed to the front end.

## 2. UML Deployment Diagram



The following diagram illustrates the deployment architecture of the distributed system. The system runs inside Docker containers for isolation and scalability.

**Description**

- **Front-end Application**: Runs on port **3000** and communicates with the back-end services via HTTP APIs.
- **Back-end Microservices**:
  - User Management (person) on port **8081**.
  - Device Management (device) on port **8082**.
  - Monitoring Service (monitoring) on port **8083**.
- **Databases**:
  - Each microservice has a dedicated MySQL database.
  - Ports: 3306 for individual databases, exposed as 3307, 3308, and 3309 for external access.
- **Message Broker**: RabbitMQ is used for communication between device simulators and the monitoring service.
- **Device Simulators**: Simulate the energy consumption data and send updates to RabbitMQ.

## 3. Readme file containing build and execution considerations

### 1. Prerequisites

Ensure the following tools are installed on your machine:

1. **Docker** and **Docker Compose** (for container orchestration)
2. **Node.js** (for frontend development, optional for containerized execution)
3. **Java 17** (for local development of Spring Boot services)
4. **RabbitMQ** (included in Docker, no manual setup required)
5. **MySQL Client** (optional, for direct database access and debugging)

### 2. Project Structure

- **frontend/**: React application for managing users, devices, and monitoring energy consumption.
- **person-service/**: Spring Boot microservice for user management.
- **device-service/**: Spring Boot microservice for device management.
- **monitoring-service/**: Spring Boot microservice for monitoring energy consumption.
- **docker-compose.yml**: Orchestrates all containers, including databases and RabbitMQ.
- **device-simulator/**: Simulates devices sending energy consumption data via RabbitMQ.

### 3. Running the System with Docker

#### Step 1: Build and Run Containers

Run the entire system with **Docker Compose** from the project root directory:

```
docker-compose up --build
```

This command will:

- Build and launch all services (React frontend, microservices, databases, and RabbitMQ).
- Expose necessary ports on your localhost.

#### Step 2: Verify Running Services

The following services will be accessible:

1. **Frontend (React App)**:
   - URL: http://localhost:3000

2. **User Management Service (person-service)**:
    - URL: http://localhost:8081
    - Endpoints: /person
3. **Device Management Service (device-service)**:
    - URL: http://localhost:8082
    - Endpoints: /devices
4. **Monitoring Service (monitoring-service)**:
    - URL: http://localhost:8083
    - Endpoints: /devices/{deviceId}/energy-consumption
5. **RabbitMQ Management Interface**:
    - URL: http://localhost:15672
    - Username: guest, Password: guest

### Step 3: Database Configuration

Each microservice uses a separate MySQL database. Ports are mapped as follows:

- **User Service Database**:
    - Port **3307**
    - Database: user_db
- **Device Service Database**:
    - Port **3308**
    - Database: device_db
- **Monitoring Service Database**:
    - Port **3309**
    - Database: monitoring_db

### Step 4: Run Device Simulators

To simulate devices sending energy consumption data:

1. Build the device simulator Docker image:

```
cd device-simulator
```

```
docker build -t device-simulator
```

2. Run the simulator:

```
docker run -d --name device-simulator device-simulator
```

The simulator sends energy data to the **RabbitMQ queue**, which the Monitoring Service processes and stores in the Monitoring Database.

### 4. Inter-Service Communication

- **User-Service** (person) provides user information.
- **Device-Service** (device) retrieves devices and their assignment to users.
- **Monitoring-Service** (monitoring) fetches energy consumption data for devices.
- **RabbitMQ** handles communication between the **Device Simulators** and the **Monitoring Service**.

**Example Workflow**:

1. Devices publish energy consumption data to a RabbitMQ queue.
2. Monitoring Service consumes this data, saves it to the Monitoring Database, and notifies the React Frontend via WebSocket.

### 5. Stopping the System

To stop all containers and clean up volumes:

```
docker-compose down -v
```