**Documentation of**

**Assignment 2**

**Asynchronous Communication and Real-Time Notification**

Name: Onea Vlad

Group: 30442

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

**Backend:** For the backend I created a standalone application which is called Sender and which reads the data from csv and publish timestamp, deviceId and measurement value to the rabbitmq queue. At each 5 seconds a message is sent but the timestamp modifies in such a way that 10 minutes have passed. Beside this I added some changes in the already existing EnergyManagementSystem microservice and in Frontend and I added a new microservice called MonitoringAndComunication. As in first project I continued with layered architecture so I devided the project in packages: Controllers, Data, Dtos, Migrations and Services. Controllers: handle incoming requests, interact with the application's business logic, and coordinate the flow of data between the user interface and the underlying services. **Data:** refers to the data access layer, which is responsible for interacting with the database. It often includes entities or models that represent the data structure and a repository pattern for performing CRUD (Create, Read, Update, Delete) operations. **DTOs (Data Transfer Objects):** DTOs are objects used to transfer data between layers of an application. They’re used to encapsulate data and send it between the client and server and between different layers of the application. **Migrations:** Migrations are files that define changes to the database schema over time**. Services:** Services typically contain the business logic of the application. They encapsulate the functionality that doesn't belong directly in the controllers or data access layer.

1.1 **Device Management:** In this project, I made a few changes: I added a RabbitMQPublisher which connects to the queue and writes a message to that queue. In DeviceService I injected this service and with the help of SyncDeviceDto will put on the queue the required fields (userId, deviceId and maxConsumption). I have an API call in controller and when I execute it, all the data will be added to the queue.

**1.2. MonitoringAndCommunication:** This microservice is responsible for consuming the message from both queues: that one coming from Device and that one coming from sender. Here I have 2 models: DeviceConsumption which stores Id, DeviceId, UserId and MaxConsumption, in a database, data which will come from Device microservice queue, and which will help to make the comparison of consumption. Another database is composed of data from EnergyConsumption which will store Id, DeviceId, Consumption and Timestamp. ConsumptionService, IConsumptionService and RabbitMQConsumerService are responsible for data which comes from Sender queue. ConsumptionService inherits IConsumptionService and contains a method which will add data to database. In RabbitMQConsumerService I consider the data at every 60 minutes which simulates an hour and I compare the measurement\_value at every 6 hours with max consumption and if this happens, a notification is send through websocket, whose connection is established through SignalR (at the /chathub endpoint). In SignalRHub I used a dictionary ConnectedClients to keep track of connected clients. The key is a userId, and the value is the connection ID assigned by SignalR to that connection. The SendMessage method allows broadcasting a message to all connected clients. It uses Clients.All.SendAsync("ReceiveMessage", message) to send a message to every client connected to the hub, where "ReceiveMessage" is the method clients should listen for to receive messages. The SendMessageToUser method enables sending a message to a specific user identified by userId. DeviceConsumptionService, IDeviceConsumptionService RabbitMQDevice are responsible for data which comes from device microservice, and all the aspects are pretty similar which what I wrote for the sender queue.

**Frontend:** For the frontend part which is new is that I added a chat service which manages the SignalR hub connection and message communication. Here I added some functions: startConnection which initializes the connection to the SignalR hub, sendMessageToClient which is responsible for sending a message to a specific client connected to the SignalR hub and addTransferChartDataListener which sets up a listener for incoming messages from the SignalR hub, specifically for messages sent with the ReceiveMessage event. The websocket starts its connection at the initialization of user dashboard page where userId is setted.

1. **UML Deployment Diagram**

I used Docker for the deployment of the application. Web app, rabbitmq, services and database are deployed on containers. Web app is using port 4200, database is using port 5432, device microservice port 5050, MonitoringAndCommunication microservice port 7300 and rabbitmq port 5672.

A diagram of a computer

Description automatically generated