MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DISTRIBUTED SYSTEMS

Assignment 3

Web sockets and load balancing

Prof. Tudor Cioara
As. Liana Toderean
As. Gabriel Antonesi

S.l. Marcel Antal
As. Alexandru Rancea

Conf. Cristina Pop
As. Dan Mitrea

2025-2026

1. Requirements

This assignment introduces several new real-time communication and processing components into the Energy Management System (EMS) developed during the previous two assignments. These components will contribute to end-user support via chat, web-socket–based notifications and communication, and scalable device-data processing through a load-balanced monitoring layer.

1.1. Functional requirements

1.1.1 Customer Support (Chat) Microservice

Customer Support Microservice provides the **application-level logic** for interactive communication between clients and system administrators. Its functionality is delivered in collaboration with the WebSocket Microservice, which handles the transport layer (these can also be implemented as one cohesive microservice).

Requirements:

- **Chat Interface Integration**
  The front-end displays a chat interface that allows authenticated users to send messages to the system administrator.
- **Asynchronous Message Delivery**
  Messages are sent to the Customer Support Microservice and forwarded to the administrator, together with the user identifier.
- **Bidirectional Real-Time Messaging**
  Both user and administrator may exchange messages freely within an active session.
- **Rule-Based Customer Support**
  Must implement a simple rule-based system capable of automatically answering common user questions. Rules may be implemented using straightforward conditional logic (e.g., keyword matching or if-else clauses). If a message matches a rule, the microservice returns the predefined response; otherwise, the message is forwarded to the **AI-Driven Customer Support**.
- **AI-Driven Customer Support**
  As an improvement, the microservice may integrate an external LLM API (e.g., Gemini, OpenAI, Mistral, Hugginface) to generate responses when no rule is matched. The response will be displayed in the front end.

1.1.2 WebSocket Microservice

The WebSocket Microservice is responsible for **real-time transport** of chat messages and system notifications. It ensures persistent communication channels between clients and backend microservices.

Its responsibilities include:

- **Real-Time Notifications**
  Deliver overconsumption alerts generated by the monitoring pipeline, to the visualization platform.

- **Forwarding Chat Messages**
  Receive chat events from the Customer Support Microservice and deliver them instantly to the chat session.

1.1.3 Load Balancing Service

The Load Balancing Service introduces a **scalable ingestion pipeline** for distributing device data across multiple Monitoring & Communication Microservice replicas.

Requirements:

- **Single Consumer of Device Data**
  The service consumes all device-generated messages from the central data collection queue in RabbitMQ.
- **Replica Selection Logic**
  Based on a load-distribution strategy (e.g., consistent hashing, load detection, or weighted distribution), the service selects the appropriate monitoring replica.
- **Per-Replica Ingest Queues**
  Forward each message to a dedicated ingest queue (e.g., *Ingest Queue 1 … Ingest Queue N*) corresponding to each monitoring replica.

1.2. Implementation technologies

➢ We recommend RabbitMQ for message brokers, REST for microservices (Java Spring REST or .NET Web API), for storing data we suggest using PostgreSQL or MYSQL, Google Gemini or Hugging Face API for AI driven customer support, WebSockets for real time communication and notifications, and for reverse proxy, we suggest Traefik, especially if you already defined it during the previous assignments. **Docker** with **Swarm** mode enabled is **mandatory** for the deployment of the new microservices with a specific number of replicas.

2. Deliverables

➢ Documentation must consist of:
  a) UML Deployment diagram.
  b) Readme file containing build and execution considerations.

➢ Source files. The source files and the database dump will be uploaded onto the personal *gitlab* account created at the *Lab resources* laboratory work, following the steps:

  - Create a repository on *gitlab* with the exact name: *DS2025_Group_LastName_FirstName_Assignment_Number*
  - Push the source code and the deployment diagram (push the code not an archive with the code or war files)
  - Share the repository with the user *utcn_dsrl*

3. Evaluation

3.1. Assignment Related Basic Questions:

During project evaluation and grading you will be asked details about the following topics:

- Load Balancing
- Docker Swarm
- Sockets, Web sockets
- Security in web applications
- Everything that was discussed during the previous laboratory meetings

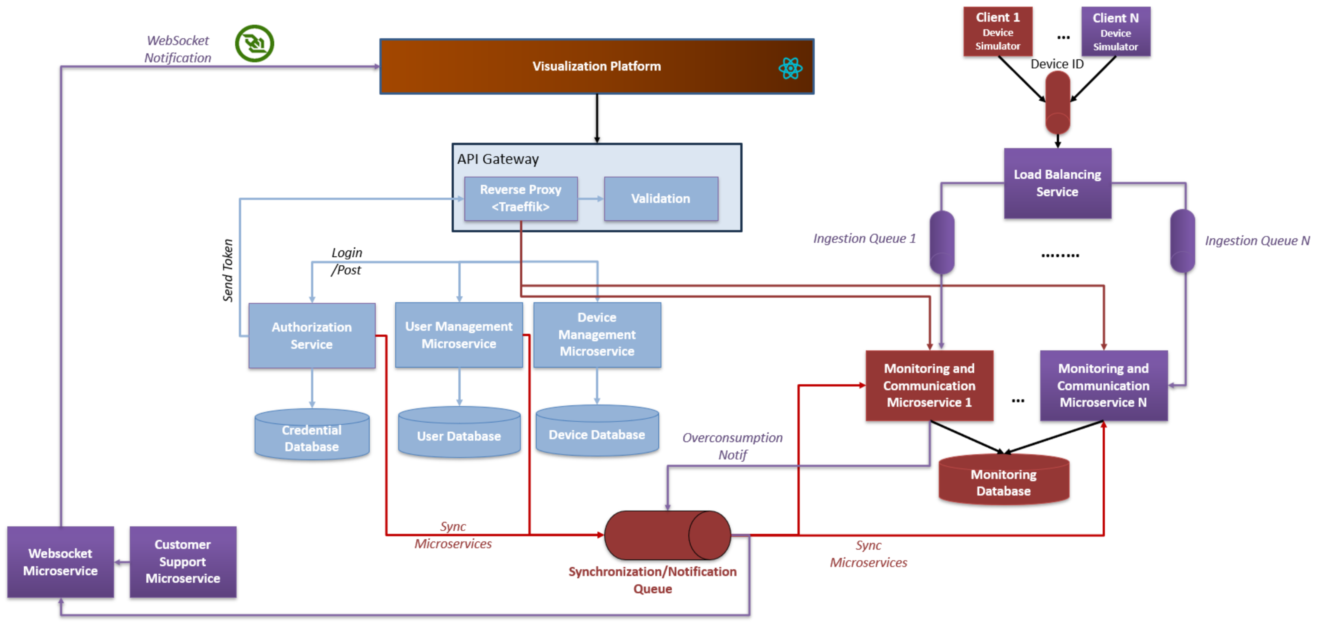
3.2. Grading

- The assignment will be graded as follows:

Points	Requirements
5 p	Minimum to pass: • Websocket Microservice for overconsumption notification (integration with assignment 2) • Rule-based chatbot for customer support (min. 10 rules) • Readme file • Correct answers to 3.1 questions
1 p	AI driven customer support
2 p	Integrate Client to Admin chat into Customer support Microservice using the Websocket Microservice
2 p	Load balancing service to offload device simulator traffic between Monitoring & Communication Service replicas

- The **project** will be graded as follows:

Points	Requirements
10 p	Minimum to pass: • Reverse Proxy • Docker deployment • Deployment diagram



4. Bibliography

1. https://dsrl.eu/courses/sd/
2. Lab Book: I. Salomie, T. Cioara, I. Anghel, T.Salomie, Distributed Computing and Systems: A practical approach, Albastra, Publish House, 2008, ISBN 978-973-650-234-7
3. Lab Book: M. Antal, C. Pop, D. Moldovan, T. Petrican, C. Stan, I. Salomie, T. Cioara, I. Anghel, Distributed Systems – Laboratory Guide, Editura UTPRESS Cluj-Napoca, 2018 ISBN 978-606-737-329-5, 2018, https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/329-5.pdf
4. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
5. https://spring.io/guides/gs/messaging-stomp-websocket
6. https://www.ibm.com/think/topics/data-ingestion