# Project Plan

Group Name: BGL Ltd


| Name | Email | Student Number |
|---|---|---|
| Gimantha Dissanayake | gimantha.dissanayake@tuni.fi | 151790463 |
| Lakshan Rathnayaka | lakshan.rathnayaka@tuni.fi, | 151788527 |
| Anaththa Pathiranage Bimsara Hirushan | bimsara.anaththapathiranage@tuni.fi, | 151790298 |


GitLab Repo URL: https://course-gitlab.tuni.fi/compcs510-spring2024/bgl-ltd

# Contents

**Timeline of the project** ..... 2

**Responsibilities** ..... 2

**Weekly Time Commitment** ..... 3

**System Architecture Description with Enhanced Features** ..... 3

System Components: ..... 3

Process ..... 6

**User Interfaces** ..... 7

..... 7

**Deployment Instructions** ..... 9

**Lessons Learned** ..... 10

# Timeline of the project

| | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 |
|---|---|---|---|---|---|
| Research & Planning | | | | | |
| Design & Architecture | | | | | |
| Backend Implementation | | | | | |
| Frontend Implementation | | | | | |
| Testing and Documentation | | | | | |

# Responsibilities

| | |
|---|---|
| Anaththa Pathiranage Bimsara Hirushan | • API end point development<br>• Frontend development<br>• Documentation<br>• Frontend implemenation |
| Lakshan Rathnayaka | • Dockerization of server A and server B.<br>• Created docker compose with server A, server B, mongo service, rabbitmq service.<br>• Added get all and get specific order API endpoint.<br>• Added swagger to server A<br>• Created communication between server A and server B through rabbitmq |
| Gimantha Dissanayake | • Create Frontend boilerplate code<br>• Create nginx config file<br>• Create Dockerfile for frontend<br>• Add frontend app service to docker-compose |

# Weekly Time Commitment

- Gimantha Dissanayake: 5 hours per week
- Lakshan Rathnayaka: 5 hours per week
- Anaththa Pathiranage Bimsara Hirushan: 5 hours per week

# System Architecture Description with Enhanced Features

This part outlines the architecture of the sandwich ordering system, incorporating functionalities for managing sandwich types, orders, user registration, and user login. Patterns Applied: Discuss architectural patterns utilized (e.g., microservices, message queues).

## System Components:

### Frontend Functionality:

Provides a user interface for:

- Adding sandwiches types
- View sandwiches types
- Placing sandwich orders.
- Selecting from a list of available sandwich types.
- User registration and login.
- Displaying order status without full page refresh.
- Communicates with Server A's API for order placement, status updates, fetching sandwich types, and user management.

### Backend:

Server A Functionality: Implements a RESTful API for:

- Managing sandwich types (CRUD operations).
- User registration and login.
- Ordering sandwiches (including associating user and chosen sandwich type).
- Retrieving order status information for a specific user.
- Stores and retrieves order details, potentially using a database
- Implements secure user authentication mechanisms
- Publishes received orders to a message queue for delivery to Server B.
- Subscribes to a message queue to receive order status updates from Server B.

Server B Functionality:

- Receives sandwich orders from the message queue.
- Simulates order preparation.
- Publishes a message indicating the order is ready to another message queue.

## Message Broker (RabbitMQ) Functionality:

Hosts two message queues:

- Queue 1: Delivers received orders from Server A to Server B.
- Queue 2: Delivers ready sandwich order notifications from Server B to Server A.

Enables asynchronous communication between backend servers.

## Database:

- Stores information about sandwich types (name, ingredients).
- Stores user registration data (username, hashed password).
- Used by Server A to persist order details including user information and chosen sandwich type.

## Communication Flow:

Admin interaction

- Admin input username and password to log in to the system. There is one particular already configured account for Admin. Credentials: - Username (admin) and Password (password).
- Admin can view the current sandwich types in the application.
- Admin can add new sandwich types to the application by adding toppings and bread types.

Frontend to Server A:

- Frontend sends new sandwich details to Server's API.

Server A processing:

- Server A validates sandwich details and saved in the database.

Customer Interaction:

- Customer can register to the system by entering username and password. After that, user account is created by the system. Then customer can log into the system using username and password.
- After successfully log into the application, customer can view list of sandwiches. Customer can click More Details button to view the more details about the sandwich. Then customer can click Order button to order a sandwich. Then customer can view his orders. After 10 seconds, Customer is notified by that order has been ready

Frontend to Server A:

- Frontend sends order details (user ID, chosen sandwich type) to Server A's API.

Server A Processing:

- Server A validates user credentials.

- Server A stores the order information in the database
- Server A publishes the received order to Queue 1.

Server B Processing:

- Server B receives the order message from Queue 1.
- Server B simulates preparation time with a delay.

Order Completion:

- Server B publishes a message indicating the order is ready to Queue 2.
- Server A receives the order status update from Queue 2.
- Server A updates its internal order status and the database

User Interface Update:

- Frontend retrieves the updated order status from Server A's API and displays it to the user.

## Design Decisions:

- Distributed Backend: Enables modularity and scalability. Backend servers can be scaled independently based on load.
- Message Queue: Enables asynchronous communication between servers, decoupling them and improving fault tolerance.
- RESTful API: Provides a standardized interface for frontend communication with Server A.
- Docker Containers: Facilitate deployment and ensure consistent environment across development, testing, and production.
- Database: Offers persistence for user data and potentially order details, enhancing system reliability.

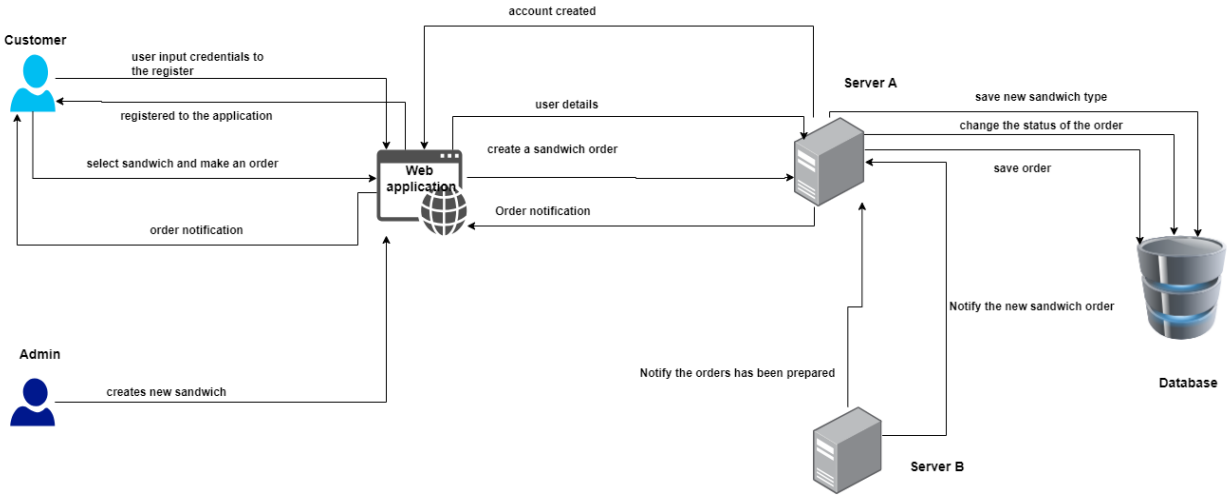
## Security Considerations:

- Implement secure password hashing and storage mechanisms to protect user data.
- Consider user access control for managing sandwich types
- JWTs are signed tokens, allowing verification that the information comes from a trusted source and hasn't been tampered with.

## Technologies used

- Frontend: React
- Backend: Node.js, Express.js
- Database: MongoDB
- Message Broker: RabbitMQ
- Deployment: Docker, Docker Compose
- WebSocket.io
- Version Control: Git, GitLab

# Process


account created
Customer
user input credentials to the register
registered to the application
user details
Server A
save new sandwich type
change the status of the order
select sandwich and make an order
Web application
create a sandwich order
save order
Order notification
order notification
Notify the new sandwich order
Admin
creates new sandwich
Notify the orders has been prepared
Database
Server B

# User Interfaces

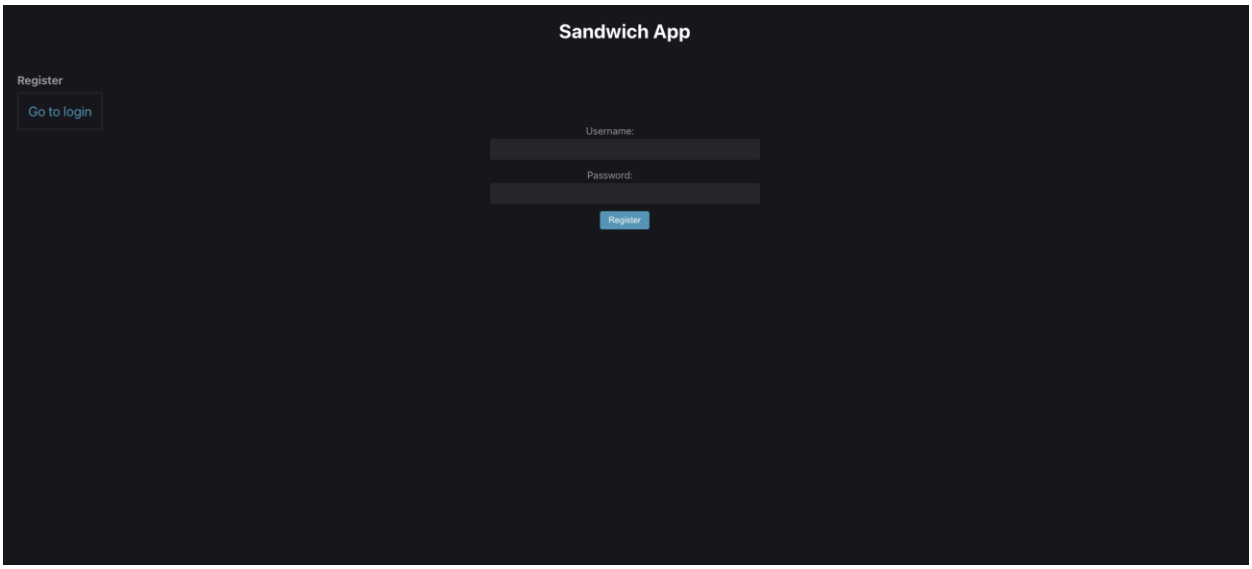



*Figure 1 Register page*

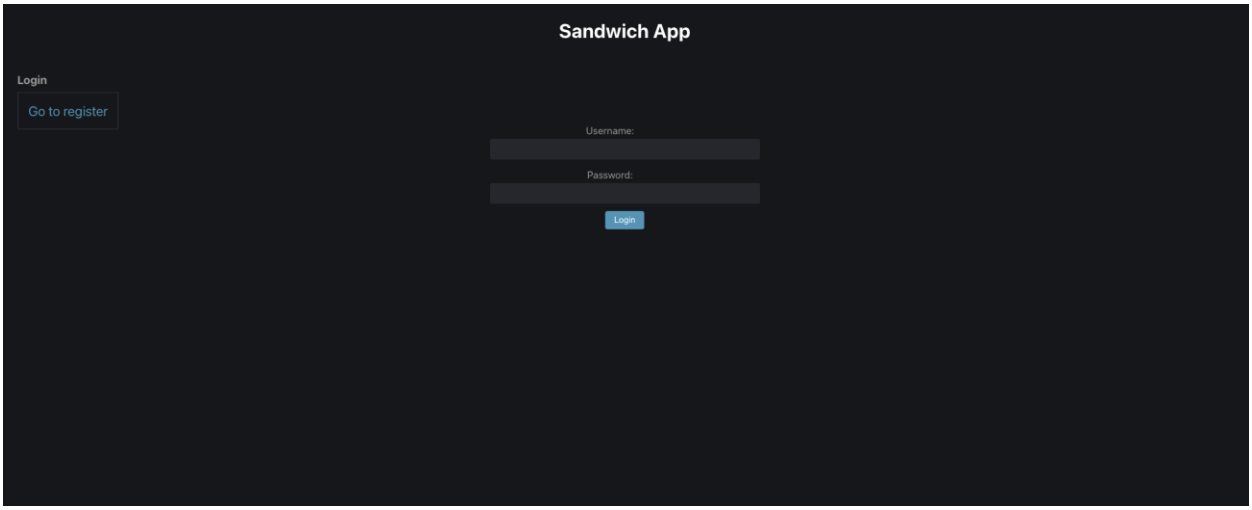



*Figure 2 Login page*

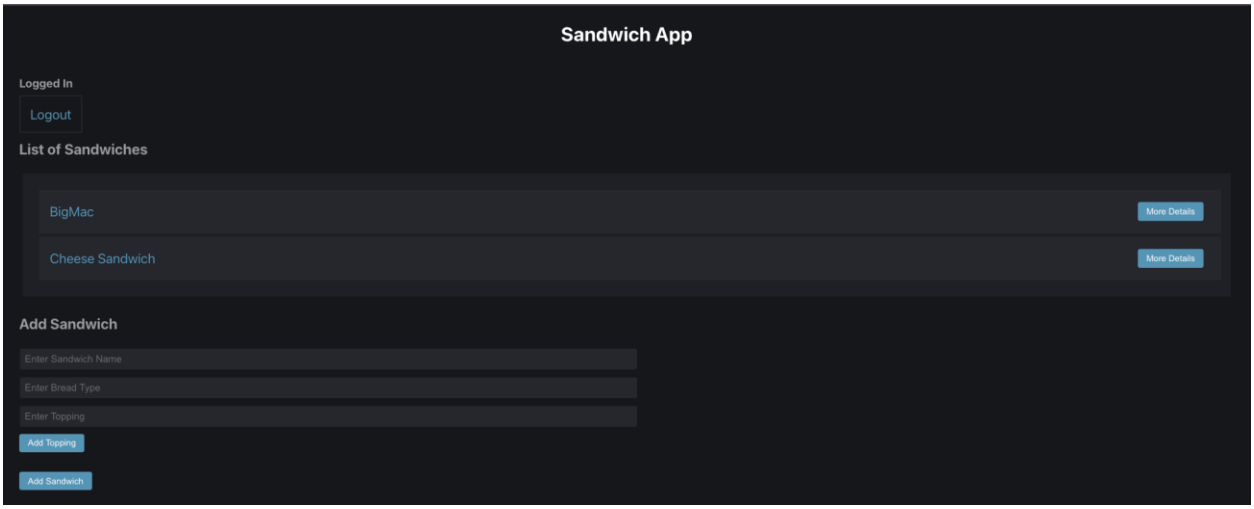



*Figure 3 Home page*

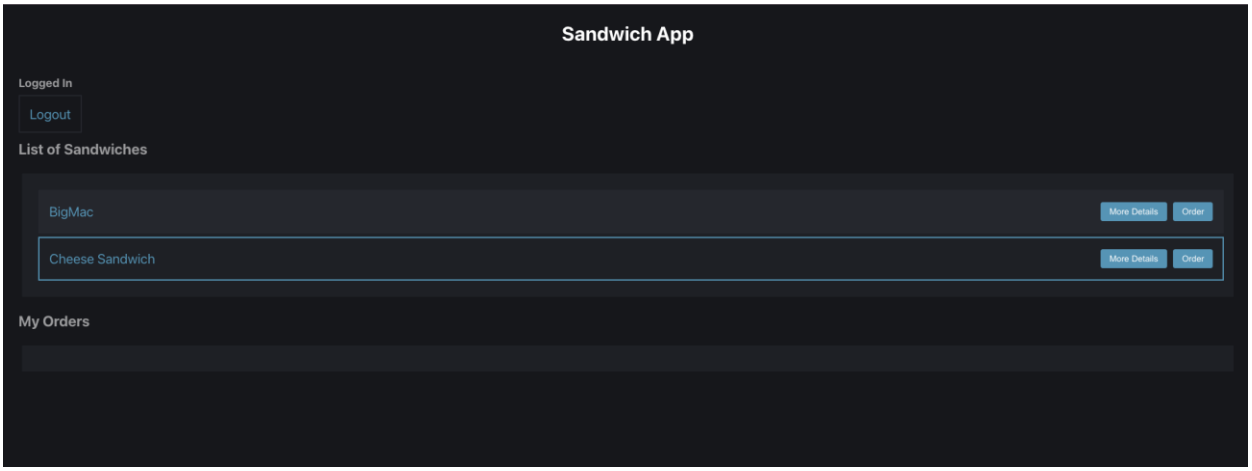



*Figure 4 List of sandwiches page*

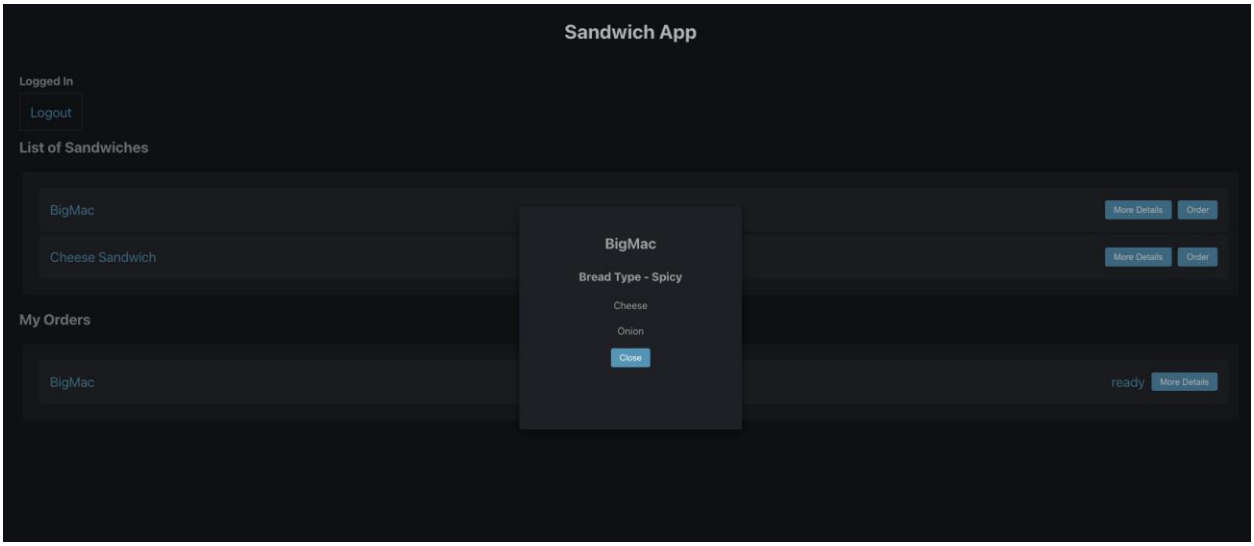



*Figure 5 View one Sandwich page*

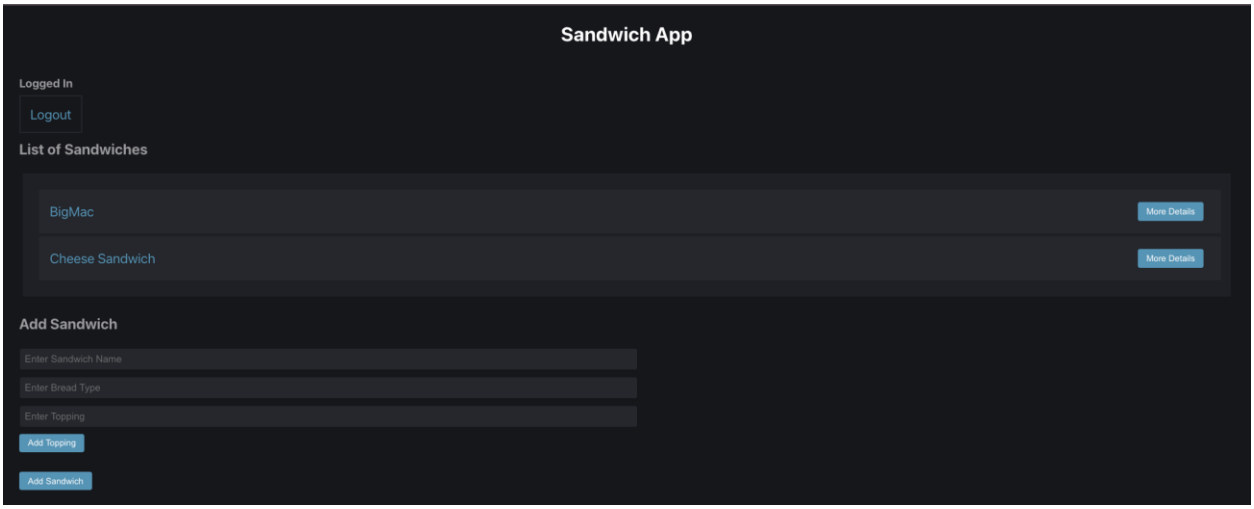



*Figure 6Add new sandwich page*

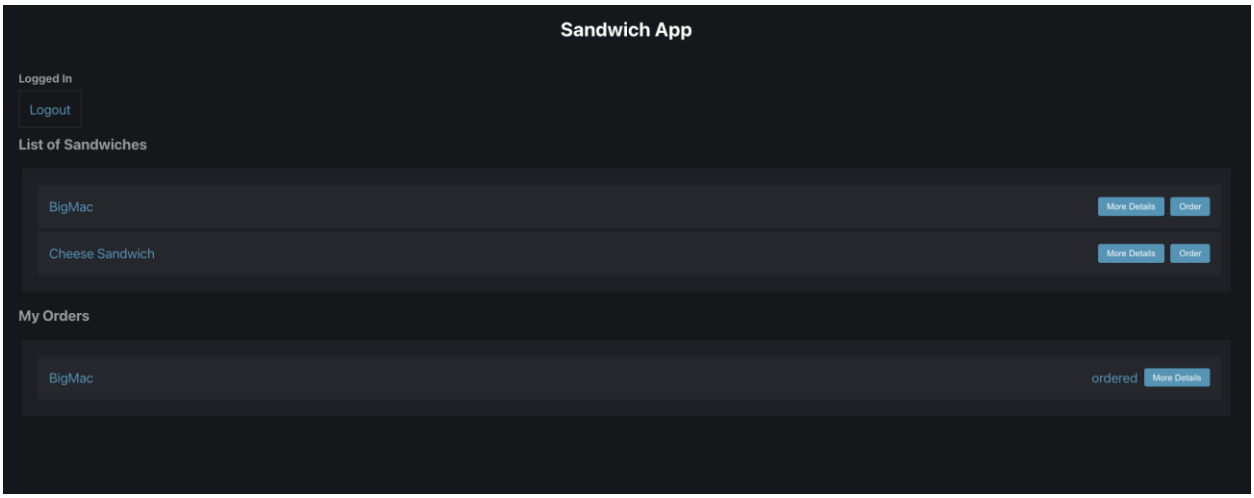



*Figure 7 List of orders page*

# Deployment Instructions

The components of our system are placed as follows in the repository:

- Frontend: Located in the frontend directory.
- Backend (Server A): Located in the backend/server-a directory.
- Backend (Server B): Located in the backend/server-b directory.

To deploy the system, follow these steps:

- Clone the GitLab repository.
- Navigate to the root directory of the repository (BGL-LTD).
- Run “docker compose up” to start the containers.
- Access the frontend application at http://localhost:3000

- Access the swagger API at http://localhost:3001/api-docs/
- Access RabbitMQ dashboard at http://localhost:15672/

# Lessons Learned

Throughout this project, we learned the importance of effective communication, collaboration, and time management in a group setting. We also gained valuable experience in researching and implementing complex software architectures, as well as troubleshooting and debugging issues in a distributed system environment. Overall, this project provided us with hands-on experience in designing and developing real-world applications using modern technologies and best practices. We learned how to containerize our applications using Docker, enabling us to encapsulate each component of our system into lightweight and portable containers. Docker simplified the deployment process and ensured consistency across different environments. Implementing a microservices architecture allowed us to break down our system into smaller, loosely coupled services, each responsible for a specific functionality. This approach improved scalability, flexibility, and maintainability of our system. Integrating RabbitMQ as our message broker facilitated asynchronous communication between the backend servers, enabling seamless interaction and decoupling of components. We learned how to set up message queues, publish and consume messages, and handle message routing efficiently. WebSocket played a crucial role in enabling real-time communication between the frontend and backend components of our system. We leveraged WebSocket to implement features such as live order status updates, enhancing the user experience and responsiveness of our application. Overall, this project provided us with valuable hands-on experience in designing, implementing, and deploying a modern web application using cutting-edge technologies and best practices. We are confident that the knowledge and skills acquired during this project will be valuable assets in our future endeavors as software developers.