# What was implemented

For this project we implemented order and sandwich -routes and frontend functionality. We left out toppings entirely, and sandwich management from frontend although backend supports it. Anything related to users was not implemented.

# Architectural description

## Patterns

The project utilizes multiple different architectural patterns. One of them is the client-server-pattern. This means that the system is divided into two component types, client components and server components. The server component listens to requests from the client component, and provides the requested services, often responding with data.

With layered pattern data passes through the system in layers, and each layer performs a specific role within the application. In this project Server A implements four layers:

- Controllers are responsible for http communication.

- Services are responsible for the business logic.

- Data-controllers are responsible for database operations.

- Database stores the data.

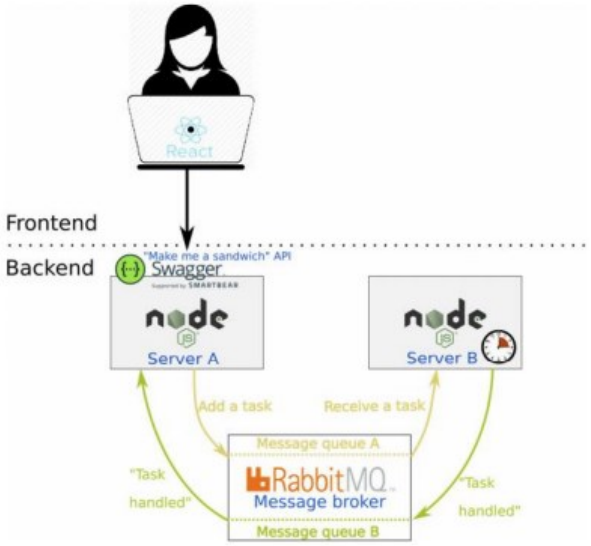


Figure 1. Architectural overview.

Message queue pattern is used for the communication between the two servers. This allows the servers to send messages to the queue without having to know about the receiver of the message.

## Components

The backend consists of four components:

- Server A manages sandwich orders. This means adding new orders, showing the state of earlier orders, and adding sandwich orders to a message queue.
    - Server A implements a pre-defined Swagger API.
- Message broker handles two message queues. One that delivers sandwich orders from Server A to Server B, and another for delivering sandwich status information from Server B to Server A.
- Server B is responsible for the making sandwiches. This means receiving orders from the message queue and publishing sandwich status information for each order when finished.
- Database is used to store all orders and available sandwiches.

Frontend is a single-page-app that enables users to make sandwich orders and view the state of their earlier sandwiches.

## Technologies

For this project we used the recommended technologies:

- ReactJS for frontend
- RabbitMQ for Message broker
- PostgreSQL for database
- NodeJS for servers A and B
    - AMQP -library for connecting to RabbitMQ
    - Pg -library for accessing database
    - Swagger-codegen for Swagger API
- Docker Compose for all components

# How to use

This project requires the latest version of Docker (https://www.docker.com/) and Docker Compose (https://docs.docker.com/compose/). For starting the program simply run

docker-compose up --build

After that the React client should be running on http://localhost:3000. You can access the Swagger UI to see all the routes from http://localhost:8081/docs/.