Project Documentation

TIE-23526 Web Architectures

Asier Alacaide - Student no. 282679 - asier.alcaidemartinez@tuni.fi
Felix Bernal - Student no. 282957 - felix.bernalsierra@tuni.fi
Raditya Ayu Wirastari - Student no. 287352 - raditya.wirastari@tuni.fi

Introduction

This document explains how the group implements in the work of the project the architecture required for the systems. Backend and frontend components are the components to be implemented in the system. The system would allow users to order sandwiches and track their order status.

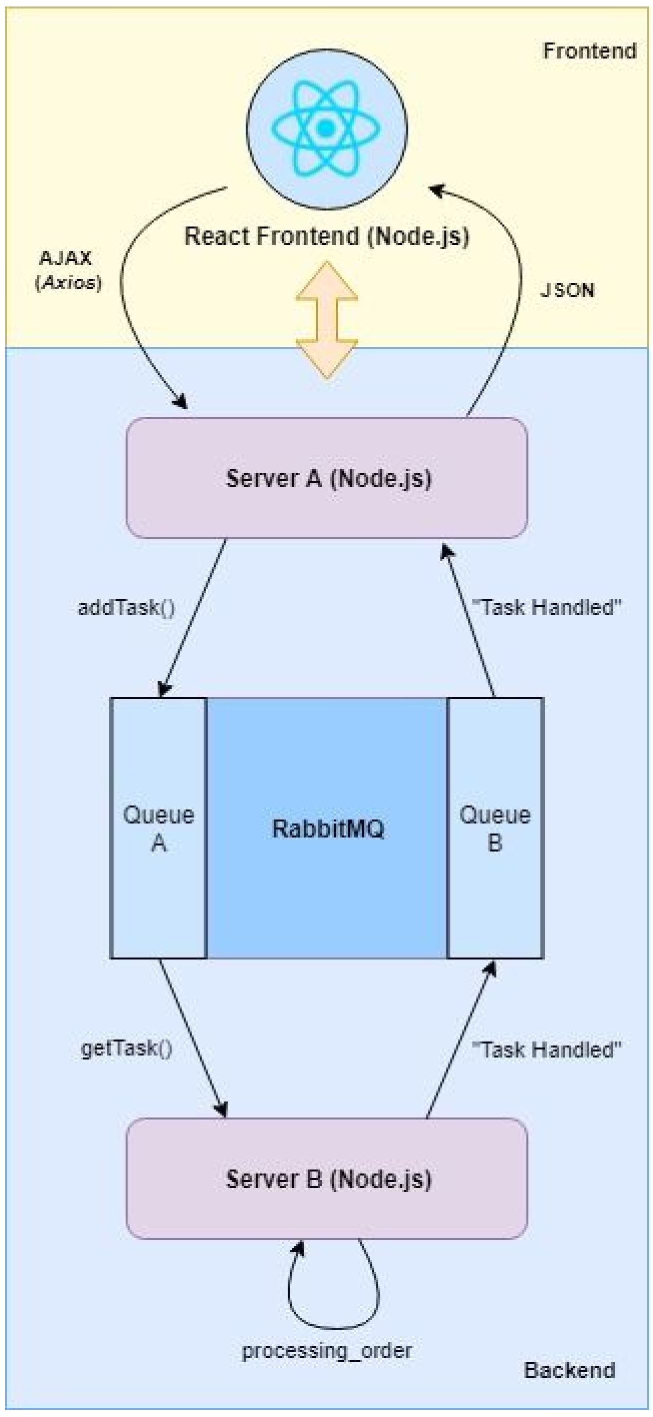
This document outlines the architecture used to implement the system and explains the technologies used for each component. This document would also include project planning and group progress work schedule. As well as explanations on the code-level system and instructions on how to use the system.

System architecture

After the research on what system architecture we wanted use for the web application, we decided to use the same architecture as the one defined on the project instructions. We are using two servers that interacts with the RabbitMQ message-broker component. The two servers are developed under Node.js library, using the RabbitMQ utils given by the Course personnel.

For the data manipulation, all transfer from servers are made by using classes. Due to the simplicity of the web application, we decided not to implement a database for the system.

The following diagram describes the complete web architecture of the system, backend and frontend:



Web Application Architecture Diagram

Used Technologies

This section explains the technologies used to build each component, as well as the tools used to develop the component technology.

Backend

The backend architecture consists of three main components: Server A, Server B, and Message Broker. Server A implements the provided Swagger API and publishes messages to a message queue and provides information on the status of sandwich orders. This server will handle the sandwich orders made by the users. A message broker is used to collect the user order queue in Server A and the status of handled sandwich orders by Server B. Server B will process messages it receives from the message queue and will post a new message to the queue about the status of the order once the sandwich order has been handled. Several technologies have been implemented to deliver these functionalities. The technologies are as follows:

- **NodeJS**

  Node.js is a powerful JavaScript-based platform built on the JavaScript V8 engine of Google Chrome. It is used to develop web-intensive I / O applications such as video streaming sites, single-page applications, and other web applications. Node.js is open source, completely free and used by thousands of developers worldwide.

  Due to its single-threaded nature, Node.js is primarily used for non-blocking event-driven servers. It is used for traditional websites and back-end API services but has been designed with real-time, push-based architectures in mind.

  The reason we chose NodeJS to develop our servers in the backend is because it was built for multiple operating systems, frequently updated, with performance optimization, security patches, and support for modern JavaScript functionality. Node.js' single-threaded, event-driven architecture allows efficient handling of multiple simultaneous connections.

- **RabbitMQ**

  RabbitMQ is an open source message broker software (sometimes referred to as message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended to support Streaming Text-oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols with a plug-in architecture.

  We use RabbitMQ as a message broker to connect servers as components of a larger application. RabbitMQ messaging is asynchronous, decoupling applications by separating sending and receiving messages. RabbitMQ provides a common platform for our applications to send and receive messages.

- **Swagger API**

  The server is communicating and sharing data through the API with the frontend components. We are using Swagger in this project to develop the API and improvise the services ' work. Swagger is an open-source software framework supported by a large tool ecosystem that helps developers design, construct, document, and consume RESTful web services.

  We believe that Swagger is beneficial to our system as Swagger uses a common language that can be understood by everyone, making our API easily comprehensible.

- **Docker**

  Docker is a computer program which performs virtualization at the operating system level. Docker provides container software that is ideal for us to develop the project and to get started and experimenting on the container-based applications.

  We are using Docker container as it allows us to package the application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, we are sure that the application will run on another machine regardless of any custom settings that the machine may have that might differ from the machine that we used to write and test the code.

Frontend

The frontend provides an interface for the end-user to interact with to access the website's functionality. Since the frontend requires several requirements to work accordingly, we built it using the technologies as follows:

- **React**

  ReactJS is essentially an open-source JavaScript library that is used specifically for single-page applications to build user interfaces. It is used in web and mobile apps view layer handling. React also allows us to create UI components that are reusable.

  ReactJS allows us to create web application that can change data without reloading the page. It's also fast, scalable and easy to see in MVC template. We consider these features to be beneficial and through the React Framework can help our website deliver its functionality to the end-user.

- **AJAX**
  AJAX enables asynchronous updating of web pages by exchanging small amounts of data behind the scenes with the server. This means that parts of a web page can be updated without reloading the entire page.

  For this project, we implement AJAX using the Axios library to send and receive sandwich orders and their status without interfering with the display and behavior of the existing page.

Learning during the project

In the course of the project development, we have to admit that we have learned multiple technologies that are very useful for the web development. First of all, we realised that the use of a panning tool like GitLab has been one of the best decisions we can make. We have used the project board to distribute the different tasks in order to optimize the work distribution. Moreover, the git tool that includes Gitlab is the main base to develop all coding scripts.

The first weeks of the project development we worked on the backend of the WebApp. We have learned several tools such as Node.js by reading many documentation and videos, and the swagger implementation to understand how is defined the web structure.

We also have learned asynchronous messaging development between Server A and Server B thanks to RabbitMQ as an easy way to send the different orders made by the client.

Regarding to the frontend part of the project, React and AJAX has been very useful to develop a simple but full Web Application that can interact with the backend without interfering with the display and behavior of the website. Different difficulties has appeared during the implementation, as we have never used React before, but thanks to the React tutorial and the different examples we have found in websites like StackOverflow or GitHub repositories we could manage it.

Project Planning

Timetable

Date	**Activity**
18/03	Researching of Technologies: Getting started with Node.js, RabbitMQ, React and Docker Container
21/03	Start development of basic implementation of Server A
25/03	Start development of Backend: Server A, Message Broker and Server B basic implementations
28/03	Development of Backend: Server A, Message Broker and Server B basic implementations
31/03	Mid project check in, finishing the development of each servers and start development of Frontend
11/04	Development of Frontend
14/04	Development of Frontend and starting implementation of Frontend with Backend
25/04	Implementation of Frontend with Backend, start documenting project
27/04	Implementation of Frontend with Backend, documenting project and running test on system
28/04	Finishing implementation of Frontend with Backend, system evaluation, project submission