

## Department of Computing

# Project (Technical Computing) [55-604708] 2019/20

Author:	Zoltan Nagy
Student ID:	27027696
Year Submitted:	2020
Supervisor:	Dr Soumya Basu
Second Marker:	Dr Hsi-Ming Ho
Degree Course:	BENG HONS SOFTWARE ENGINEERING
Title of Project:	Implementation of a Containerized Microservice Architecture

Confidentiality Required?

NO ☒

YES ☐

---

## ABSTRACT

Software containerization and the microservice architecture have been gaining ground over the traditional monolithic approach ever since Netflix (considered to be a major pioneer in the microservice world) decided to change their system architecture, having made a lot of their hard work open source. It has since been adopted by other giant companies as well, like Amazon, Uber, and eBay.

The microservice architecture revolves around separating a system into individual, loosely coupled components, each responsible for different tasks, while containerization means building and deploying applications regardless of host environments.

This project endeavors to explore and combine these two concepts through the construction of one such system, learning about and applying the most up-to-date methods and technologies in the process.

## CONTENTS

Abstract.....	2
1. Introduction .....	6
1.1. Project Overview.....	6
1.2. Project Aims.....	6
2. Tools & Technologies .....	7
2.1. Main Languages .....	7
2.1.1. Java .....	7
2.1.2. JavaScript.....	7
2.2. Frontend Framework & Dependencies.....	8
2.2.1. React.....	8
2.3. Backend Framework & Dependencies.....	9
2.3.1. Spring / Spring Boot.....	9
2.3.2. PostgreSQL Database .....	9
2.3.3. Eureka Server & Eureka Discovery Client & Zuul.....	10
2.3.4. Spring Security / JWT.....	10
2.4. Containerization.....	11
2.4.1. Docker.....	11
3. Design.....	12
3.1. Requirements Specification .....	12
3.1.1. User Categories .....	12
3.1.2. Functional Requirements .....	12
3.1.3. Non-functional Requirements.....	12
3.2. Design Specification.....	13
3.2.1. Use Cases.....	13
3.2.2. System Architecture .....	13
3.2.3. Structure of Backend Applications .....	14
3.2.4. Endpoints.....	15
3.2.5. User Interface .....	16
4. Development.....	17
4.1. Strategy .....	17
4.2. Setup .....	17
4.3. The Backend Projects.....	17
4.3.1. Eureka Server & Clients .....	17
4.3.2. Package Structure of Backend Applications.....	18
4.3.3. ID Generation Strategy .....	19

4.3.4. Common Project.....	19
4.3.5. Authorization Service .....	20
4.3.6. Gateway Service .....	20
4.4. The Frontend.....	21
4.5. Containerization.....	21
4.5.1. Backend Containerization.....	22
4.5.2. Frontend Containerization .....	22
4.5.3. Container Orchestration.....	22
4.6. Testing.....	23
4.6.1. Login (landing) Page Tests .....	23
4.6.2. Registration Page Tests .....	23
4.6.3. Navigation Bar / Top Bar Tests .....	24
4.6.4. Employee Overview Page Tests.....	24
4.6.5. Customer Overview Page Tests.....	25
4.6.6. Profile Page Tests .....	25
4.6.7. Categories Page Tests.....	26
4.6.8. Ticket Browser Page Tests.....	26
4.6.9. New Ticket Page Tests .....	26
4.6.10. Ticket View Page Tests .....	27
4.6.11. My Assigned / Pending / Closed Tickets (Employees only) .....	28
4.6.12. Active Tickets / Archived Tickets Pages (Customers Only) .....	29
4.6.13. Responsiveness Tests .....	29
5. Evaluation.....	30
5.1. Critical Reflection.....	30
5.2. Future Enhancement Opportunities.....	31
5.3. Personal Reflection .....	31
6. References.....	32
7. Appendices.....	33
7.0. Project Specification .....	33
7.1. Visitor Use Case Diagram .....	40
7.2. Customer Use Case Diagram.....	41
7.3. Employee Use Case Diagram .....	42
7.4. MVC Design Pattern Diagram .....	43
7.5. WordSpot's Microservice Architecture Diagram .....	43
7.6. Structure of Backend Applications.....	44
7.7. UI Wireframe – Login Screen .....	44
7.8. UI Wireframe – Customer Overview.....	45

7.9.	UI Wireframe – Employee Overview .....	45
7.10.	UI Wireframe – Profile .....	46
7.11.	UI Wireframe – Categories .....	46
7.12.	UI Wireframe – Ticket Browser.....	47
7.13.	UI Wireframe – Ticket View .....	48
7.14.	Eureka Server – application.properties .....	48
7.15.	Ticket Service – Package Structure .....	49
7.16.	Ticket Service – TicketController.java GET Example .....	49
7.17.	Ticket Service – TicketService.java GET Example.....	49
7.18.	Ticket Service – Ticket.java .....	50
7.19.	Ticket Service – IDGenerator.java.....	51
7.20.	Common Project – JWT Config .....	52
7.21.	Authorization Service – SecurityConfig.java .....	53
7.22.	Gateway Service – application.properties .....	54
7.23.	Gateway Service – SecurityConfig.java.....	55
7.24.	Dashboard Service – Dependencies.....	56
7.25.	Dashboard Service – Folder Structure .....	57
7.26.	Backend Builder Dockerfile.....	58
7.27.	BATCH_CreateBuilder.bat.....	58
7.28.	Ticket Service – Dockerfile .....	59
7.29.	Dashboard Service - Dockerfile.....	59
7.30.	docker-compose.yml .....	60

# 1. INTRODUCTION

## 1.1.PROJECT OVERVIEW

Microservices are on the rise, offering many advantages over the traditional monolithic approach in certain scenarios. The focus of this project is to develop an application using a microservice architecture and deploy using a containerized approach. The system was modelled around a suitably relevant example, which is as follows:

WordSpot is a fictional company that provides a range of writing services, such as translation, report writing, contract preparation and proofreading.

The company employs Project Managers (PMs) and Subject Matter Experts (SMEs). PMs are responsible for creating tickets out of phone calls, assigning tickets to SMEs, carrying out post-work quality assurance and dealing with complaints. SMEs are responsible for completing their assigned tickets in a timely manner.

All customers are served on an ad-hoc basis; services can be requested through an online interface or by phone. This creates a service request ticket in the system that contains the details of the customer, the service required, any source files and optional additional information.

A newly created ticket has an “unassigned” status. A PM then assigns the ticket to a suitable SME, who carries out the relevant work and updates the ticket accordingly, assigning it back to the customer who requested the service.

Then, the customer can either accept or reject the work. If the work is rejected, the ticket goes back to “Assigned” status, and the SME is expected to examine the ticket once again.

## 1.2.PROJECT AIMS

- ❖ Research & learn suitable technologies for the project
- ❖ Learn how to implement and work with microservices
- ❖ Learn about general microservice security
- ❖ Research & learn about containerization
- ❖ Produce an application to satisfy the requirements described in the elaboration
- ❖ Create a suitably attractive online frontend user interface
- ❖ Learn the advantages of the microservice architecture compared to the monolithic approach

## 2. TOOLS & TECHNOLOGIES

*This section intends to list and describe the main languages, tools and technologies used in the project, while also justifying their selection against other similar items, where appropriate.*

### 2.1. MAIN LANGUAGES

#### 2.1.1. JAVA



Not long after being introduced to the object-oriented Java language in the second year of my studies, it had quickly become a main focus and the preferred language to work with. Java has been around for a long time now and it is one of the most popular object-oriented languages of today, thanks to its adaptability, independence, and active community of developers behind it.

Because of this, there is also a vast array of libraries available for it, many of which are intended to make developers' lives easier. For example, Lombok is one such library, created to reduce boilerplate code through various annotations, which write code automatically at compile-time. Consequently, it is also used in project. (The Project Lombok Authors, n.d.)

This project uses Java version 11 to take advantage of a smaller JDK (Java Development Kit) size, local-variable type inference (introduced in version 10), and its extension to lambda expressions.

#### 2.1.2. JAVASCRIPT



While Java is a personal preference for all manners of programming tasks and backend implementations, JavaScript's excellence is undeniable when used as part of a modern framework like React. Its inferred and loosely typed nature works well with frontend development and it was used as a direct result of choosing React as the frontend framework for this project (explained in the next section).

### 2.2.1. REACT



React is the most popular member of the holy trinity of JavaScript frontend frameworks of today, the other two being Vue.js and Angular. Created and maintained by Facebook along with a large community of individual and corporate developers, React is an excellent framework for building responsive single-page web applications. (React, n.d.)

All three options were viable for the purposes of this project. However, having worked with each of them to some degree previously, a personal preference was acquired for React, which is the main reason why it was chosen. Its main advantage is its lightweight component-based system, which allows developers to only bring in libraries that are actually required for a given project.

Among the libraries used are React Router, Axios and React Bootstrap.

Integrating React Router into the interface has allowed the frontend to give the illusion of having multiple pages, when in reality, React Router allows us to render different components for different, pre-defined URL paths. The fact that the browser does not reload the whole page during navigation gives a smooth feeling to the frontend. (React Router, n.d.)

Axios is a Promise-based library that allows simple communication with the application's back-end APIs through http requests. These requests have the potential to be executed asynchronously, which further adds to the smooth feeling of navigating a properly constructed React application. (Axios, n.d.)

Bootstrap is a tried and tested tool for web applications to produce clean and responsive layouts. The React Bootstrap library allows us to directly use component-based versions of Bootstrap's classes, which integrates very well with React. (Bootstrap, n.d.) (React Bootstrap, n.d.)



## 2.3.BACKEND FRAMEWORK & DEPENDENCIES

### 2.3.1. SPRING / SPRING BOOT



During the university course, several frameworks / technologies suitable for backend development were introduced, including the JavaScript-based Express and PHP. However, resolute about working with Java, more research had to be conducted to find the perfect fit, during which the Spring and Struts frameworks were discovered. Spring was chosen because it is the more popular of the two, and the Struts framework has encountered severe security breaches in the past and seems to be poorly maintained. (Singh, Chawla, Singh, & Singh, 2018)

Moreover, it provides an excellent framework for web applications, first-class support for the MVC (Model-View-Controller) design pattern, and a stable, structured and highly configurable base to build on. (Spring | Web Applications, n.d.)

Familiar with how backend applications function but having no prior experience with this framework meant some trial and error when it came to learning it. This is where Spring Boot came into the picture, which is essentially a “framework for a framework”. Spring Boot eliminates the need for developers to manually configure Spring with XML files and allows the use of a clever annotation system to dynamically and automatically configure a Spring application. Moreover, the Spring Initializr tool allows developers to quickly create and bootstrap a Spring application from a large pool of available dependencies. (Spring Boot, n.d.) (Spring Initializr, n.d.)

### 2.3.2. POSTGRESQL DATABASE



When it came to selecting a database, two options were familiar: MySQL and MongoDB, but it was decided that alternatives were to be looked at as well. PostgreSQL was discovered during the research phase, which is a relational database engine similar to MySQL. However, the main reason for choosing it over the other two options was its concurrency capability. (PostgreSQL: About, n.d.)

When it comes to microservices, the concurrency capabilities of the backend databases have to be top notch, and the first-class support for this feature is one of the main selling points of PostgreSQL, along with excellent read & write speeds and support for large systems. (PostgreSQL vs MySQL, n.d.)

---

### 2.3.3. EUREKA SERVER & EUREKA DISCOVERY CLIENT & ZUUL

With microservices came the need for some technology that was able to keep track of individual services and act as a “phone book”, where each service was aware of and could look up every other service as well.

In this regard, two options were discovered: Eureka and Consul, both of which are intended to accomplish the above. The former was chosen over the latter due to the fact that Eureka can be easily and automatically used in a Spring Boot application through minimal manual configuration.

Eureka is a tool made, maintained and open-sourced by Netflix that serves to locate individual microservices and help with load balancing (with Zuul). In the context of Spring, Eureka can be bootstrapped to an application in two ways: either a Eureka Server or a Eureka Client.

The server operates as a service registry; each individual application with the client dependency can register itself with the server and from that point onward, they each have easy access to one another through the said registry. This allows for the dynamic calling of one service from another by simply using its name instead of having to hard-code an address, which would have become a massive problem in a containerized environment. (Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka, n.d.)

Zuul, also developed and made open source by Netflix, is essentially a gateway that allows the controlling and routing of incoming requests by working in harmony with Eureka. It is also highly configurable, and an individual service equipped with Zuul could serve as the main entry point into the backend system of this project with the appropriate security checks. (Netflix/zuul, n.d.)

---

### 2.3.4. SPRING SECURITY / JWT

Security should always be an important consideration, and to this end, Spring’s own customizable security framework was used, Spring Security. (Spring Security, n.d.) Furthermore, a lightweight and easily managed token-based approach was also found, so the decision was made to also bring JWT (JSON Web Token) into the mix.

JWTs are a modern way to secure an application’s API endpoints. In and of itself, it is simply an encoded string that is made up of three parts: the header, the payload and the signature. The header usually contains information about the encoding used and the type of a given token. The payload is the actual data that the token contains, which can be anything we want it to be, but usually includes at least the subject (user) of the token, and a timestamp of its date of issue. (JSON Web Token Introduction, n.d.)

However, the most important part is the signature, where a given token’s validity stems from. When a token is issued, the encoded header, encoded payload and a secret string of characters (known only to the developers) are encrypted using the algorithm specified in the header, and the result becomes the signature. This can later be used to verify that the content carried by the token wasn’t modified along the way. (JSON Web Token Introduction, n.d.)

## 2.4.CONTAINERIZATION

### 2.4.1. DOCKER



Containerization is the ultimate solution to adapt an application to run on any environment imaginable. Docker does this by allowing developers to combine the application with all its necessary dependencies into a Docker image. Docker was chosen because it seems to be the industry standard when it comes to containerization, with similar alternatives being much less popular.

Well-built Docker images can be deployed regardless of environment, because when an image is ran, a kind of virtual machine is created on the host machine (i.e. a container), which runs the application in its own pre-defined environment, isolated from the host environment, but still accessible from it. The Docker Hub provides many base images upon which developers can base their own Docker images. Consequently, the only dependency required on the host machine is the Docker engine itself. (Docker Inc., n.d.)

However, Docker is also useful when it comes to compiling / building an application, especially because development environments tend to be different and larger in size than runtime environments. Imagine that we have a Maven project that uses Java version 11, but our system only has the runtime environment of Java version 8 and no Maven installed. This is an example where a so-called builder image might come in handy. They work similar to the description above, but their only purpose is to create a container with an environment suitable for compiling and building a given project, which, in this example, would be a JAR or a WAR file. The compiled application could then be Dockerized in a different image and ran in the manner described above. (Docker Inc., n.d.)

The Docker engine also provides a way to manage and configure multiple containers within a Dockerized environment through the use of the Docker Compose functionality. (Docker Inc., n.d.)

## 3. DESIGN

### 3.1. REQUIREMENTS SPECIFICATION

#### 3.1.1. USER CATEGORIES

Defining the functional requirements is made easier by separating our users into distinct groups. The specification already provides three different kinds of users (Project Manager, Subject Matter Expert and Customer), but for the purposes of requirements, these can be reduced even further to just **Employees** and **Customers**.

The two user groups need to have a different experience on the website. While a Customer might visit to request jobs or review completed ones, an Employee might want to work on a ticket, view information like the number of tickets assigned to them or the number of tickets in the system, but these pieces of information are irrelevant to Customers.

#### 3.1.2. FUNCTIONAL REQUIREMENTS

Firstly, Employees should be able to easily:

- ❖ Create tickets
- ❖ Update existing tickets
- ❖ Track a ticket's history
- ❖ Identify unassigned tickets
- ❖ Identify if a ticket is assigned to them
- ❖ See an overview of tickets related to them
- ❖ Find any ticket in the system by ID, status, Customer name or the name of the Employee the ticket is assigned to
- ❖ View, create, edit and delete ticket categories

Consequently, the tickets should house all the necessary information required to identify its target customer, the job that needs to be done and the history of the ticket. Moreover, they should also provide the opportunity to attach files where necessary.

Secondly, Customers should be able to easily:

- ❖ Request a job (i.e. create a ticket)
- ❖ Identify the status of their active tickets
- ❖ Update their tickets with more information if necessary
- ❖ Accept or refuse tickets where the work has been completed
- ❖ View all their active & archived tickets at any time

Finally, both user groups should be able to view and modify their profile data, like email address, password and username.

#### 3.1.3. NON-FUNCTIONAL REQUIREMENTS

The first – and perhaps the most important – non-functional requirement of the application is having a user-friendly frontend interface that is easy to navigate, which plays a vital role in

helping users feel comfortable while using the website. A responsive, mobile-friendly design and a visually appealing look both play a huge part in this.

Secondly, instant feedback and unambiguous alerts in response to user actions. The website should be quick to load and communicate clear feedback on the actions of the user where this is justified. For example, if an error is encountered, the reason for it needs to be told explicitly, with suggestions on what could be done to prevent it.

Lastly, due to the nature of this project, some data must be stored about each user, including email addresses, phone numbers and passwords. Security needs to be an important consideration in every online application, and when storing sensitive information cannot be avoided, measures are required to make sure that the data is safe. Passwords should be hashed and the backend should be secured using a fitting, modern approach.

## 3.2.DESIGN SPECIFICATION

---

### 3.2.1. USE CASES

*Related appendix items: 7.1, 7.2, 7.3*

When it comes to use cases, a third category of users comes into play in addition to the two described in the functional requirements section, which is Visitors.

Visitors are a type of user that is either not registered on the website, or is not logged in. When a visitor logs in, he or she becomes either a Customer or an Employee in the context of use cases. Appendix 7.1 shows the use case diagram for Visitors.

The second group of users is Customers. Customers use the website to request services or view their requests. They can only see tickets associated with them and cannot modify ticket status or make significant changes after creating them, but they can update the ticket with an additional piece of description if necessary. Appendix 7.2 shows the use case diagram for Customers.

The third and final group of users is Employees, who use the website to work and have absolute control over the tickets in the system. They can see and filter through all the tickets in the system by several criteria and are able to modify all aspects of a given ticket. Appendix 7.3 shows the use case diagram for Employees.

---

### 3.2.2. SYSTEM ARCHITECTURE

*Related appendix items: 7.4, 7.5*

The application is based on the MVC (Model-View-Controller) design pattern, which serves to separate an application's operations into distinct, interconnected parts. The basic MVC design pattern diagram is shown in Appendix 7.4. Users get served the View part through the Controller, the Controller can manipulate the Model (which contains the business logic and the persistence layer) and in turn, View is changed through the Controller according to the changes to the Model.

To show a clearer picture of the overall architecture of the system, Appendix item 7.5 was created, which shows an overview of how the individual services are connected. Each named rectangular item is a service (application) of its own.

The planned microservices shown in the diagram are:

- ❖ Dashboard Service
  - Serves as the frontend of the system
  - A web application through which the system can be interacted with
- ❖ Eureka Server
  - Backend service registry
  - Keeps track of all online backend services and allows them to easily access one another
- ❖ Gateway Service
  - Provides an entry point into the backend, controls incoming requests
  - Responsible for the authentication and routing of incoming requests
- ❖ Authorization Service
  - Responsible for the generation and issuing of authorization tokens after the appropriate checks
- ❖ User Service
  - Provides endpoints to access and manipulate the users stored in its respective database
- ❖ Ticket Service
  - Provides endpoints to access and manipulate tickets and ticket categories.
- ❖ File Service
  - Responsible for storing and keeping track of files uploaded and associated with tickets.
  - Also provides endpoints to access and manipulate these files and their metadata.

---

### 3.2.3. STRUCTURE OF BACKEND APPLICATIONS

*Related appendix item: 7.6*

Having established a clear overview of the system architecture as a whole, it was now possible to go one level deeper and design the structure of the individual backend services, which resulted in Appendix 7.6. This diagram adheres to the dependency inversion principle, which states that high-level and low-level modules should not depend on each other; they should both depend on abstractions. (Thorben Janssen, 2018)

The first point of contact for requests is the Controller layer, which defines the endpoints of a given application, and is responsible for logging and forwarding the request to the Service layer, where the business logic is contained.

If the given application has a data source, then it also has a Data Access layer that the Service layer can interact with. The Data Access layer implements an interface through which the data source can be accessed and manipulated.

Once a request has been dealt with by the Service layer (having used the Data Access layer to do so, if necessary), the response is sent back to the Controller, which then forwards the response out of the application's context.

---

### 3.2.4. ENDPOINTS

The endpoints used by the system can be categorized according to the service which they provide access to. These endpoints are defined in the following list (Eureka Server and Gateway Service don't have explicit endpoints, more on this in the Development section).

#### ❖ Dashboard Service

- Not logged in (Visitor)
  - (GET) / – Landing page login screen
  - (GET) /register – Customer registration page
- Logged in as Customer
  - (GET) / – Customer-specific overview page
  - (GET) /profile – Profile information view / edit page
  - (GET) /tickets/new-ticket – New service request page
  - (GET) /tickets/active – My active tickets page
  - (GET) /tickets/history – My archived tickets page
  - (GET) /tickets/view/{ID} – Single ticket view page
- Logged in as Employee
  - (GET) / – Employee-specific overview page
  - (GET) /profile – Profile information view / edit page
  - (GET) /tickets/browse – Ticket browser & search page
  - (GET) /tickets/new-ticket – New ticket page
  - (GET) /tickets/assigned – My assigned tickets page
  - (GET) /tickets/pending – My pending tickets page
  - (GET) /tickets/closed – My closed tickets page
  - (GET) /tickets/view/{ID} – Single ticket view / edit page

#### ❖ Authorization Service

- Public endpoints
  - (POST) /auth – Authorization token request

#### ❖ User Service

- Public endpoints
  - (POST) /public/register – Customer registration
- Secured endpoints
  - (GET) /get/all – Get all users
  - (GET) /get/by-username/{username} – Get user by username
  - (GET) /get/by-id/{ID} – Get user by ID
  - (POST) / – Create new user
  - (PUT) /{ID} – Modify existing user
  - (DELETE) /{ID} – Delete existing user

#### ❖ Ticket Service

- Secured endpoints
  - (GET) / – Get all tickets (optional filters can be passed through params)
  - (GET) /id/{ID} – Get ticket by ID
  - (POST) /add – Create new ticket
  - (PUT) /update/{ID} – Update existing ticket
  - (GET) /categories – Get all ticket categories
  - (GET) /categories/{ID} – Get ticket category by ID
  - (POST) /categories – Create new category

- (PUT) /categories/{ID} – Modify existing category
  - (DELETE) /categories/{ID} – Delete category
- ❖ File Service
  - Secured endpoints
    - (GET) /getAll – Get metadata for all files stored
    - (GET) /getAllFor/{ticketID} – Gets all file metadata for files associated with {ticketID}
    - (GET) /getMetadata/{ID} – Gets file metadata by file ID
    - (GET) /getResource/{ID} – Gets the file itself by its recorded ID
    - (POST) /upload – Stores a file and its metadata

---

### 3.2.5. USER INTERFACE

*Related appendix items: 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13*

Wireframes were created for the most important parts of the frontend user interface. These can be found from appendix item 7.7 to 7.13. They were designed using Balsamiq, which is a specialized tool for creating wireframes. (Balsamiq Studios, LLC, n.d.)



## 4. DEVELOPMENT

*This section intends to highlight the most important aspects of development.*

### 4.1. STRATEGY

The chosen development strategy for this project was an Agile method called feature-driven development (FDD). This strategy revolves around defining the work that needs to be done based on planned functionalities, so that development can be divided into small, well-defined sprints, each one focused on a particular functionality.

Having already defined all the aspects needed to start development in the Design section, it was easy to identify the functionalities to implement with this approach.

### 4.2. SETUP

Some important steps needed doing before development could start.

Firstly, setting up version control. To this end, GitHub and its supporting desktop application were used to create and push a new repository called WordSpot, after which a development branch was created to keep my production environment isolated.

Secondly, the backend projects were created using Spring Initializr. This tool allows developers to skip manually doing boilerplate configurations, easily include dependencies, and get straight to developing. Eureka Server, Authorization Service, Gateway Service, User Service, Ticket Service and File Service were all created and initialized with the Spring Initializr tool and the chosen project management tool was Maven. (Spring Initializr, n.d.)

Lastly, the frontend project was created with Node Package Manager (NPM). The basis for this project was the so-called “npx create-react-app” command, which created a blank React application to get started on.

### 4.3. THE BACKEND PROJECTS

---

#### 4.3.1. EUREKA SERVER & CLIENTS

*Related appendix item: 7.14*

When it came to the Eureka Server project, most of the required configurations could be done automatically with Spring Boot’s clever annotation system. Including Netflix’s Eureka Server dependency in the project has provided an `@EnableEurekaServer` annotation to use on the main entry point of the application, next to the `@SpringBootApplication` annotation, which serves to automatically configure and start the application in a default embedded Tomcat server.

Apart from the few lines of configuration in the application.properties file (shown in appendix 7.14), this application was now configured and ready to use.

With Eureka Server functional, the only thing that was left to do in this regard was to make use of the Eureka Discovery Client dependency in every other backend project and point to the address of Eureka Server in their application.properties files.

This has made sharing data and communicating between services easy because now a service could refer to another service that was also part of the Eureka service registry by name.

---

#### 4.3.2. PACKAGE STRUCTURE OF BACKEND APPLICATIONS

*Related appendix items: 7.15, 7.16, 7.17, 7.18*

Appendix item 7.15 shows the package structure of Ticket Service, but this general structure was applied to every backend application, where appropriate. Using Ticket Service as an example, the following is a description of what each package houses and what they are responsible for.

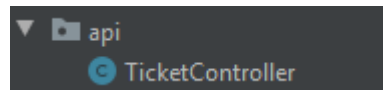


Figure 1

The API package contains the Controller class. Marked with the `@RestController` annotation, this class describes the endpoints and their types for a given backend application. Once a request is received, it logs this fact and then calls the Service layer to deal with the request, passing all the received arguments through. Every method of this class is annotated with the corresponding type of request it accepts, using the `@GetMapping`, `@PostMapping`, `@PutMapping` or `@DeleteMapping` annotations. Appendix item 7.16 shows an example, which is a GET endpoint in Ticket Service.

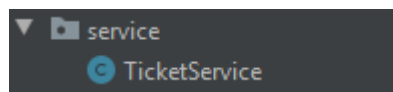


Figure 2

The service package acts as a middleware between the Controller and the Data Access layers to isolate the operational roles within the application context. Annotated with `@Service`, the `TicketService` class contains the business logic of a given application. Appendix 7.17 shows the function of `TicketService` called by the Controller GET mapping in Appendix 7.16.

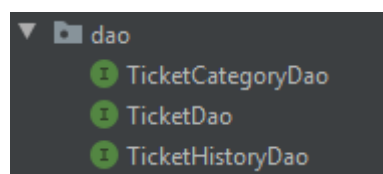


Figure 3

The DAO (Data Access Object) package contains interfaces that extend the JPA (Java Persistence API) Repository interface. These interfaces, accessible only from the Service layer, allow us to perform CRUD (create, read, update, delete) operations on the tables in the data source connected to the application (configured in the `application.properties` file). They make use of Models to do this:

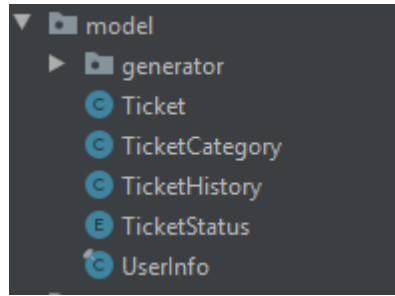


Figure 4

Models are Java classes that define the tables in our connected data source; each class file can be considered as a Java interpretation of a table structure, also usable as an object. Appendix item 7.18 shows the implementation for the Ticket class. The `@Data` (generates getters and setters), `@AllArgsConstructor` (generates a constructor with all class attributes as arguments), `@NoArgsConstructor` (generates a constructor with no arguments) and `@Builder` (allows builder-based initialization of objects) annotations are provided by the Lombok library. The `@Entity` and `@Table` annotations are provided by JPA and their purpose is to define this class as a table in our data source.

---

#### 4.3.3. ID GENERATION STRATEGY

*Related appendix item: 7.19*

One of the requirements for this system is for entities (e.g. tickets) to be easily searchable by ID as well. Since a simple incremental ID generation strategy is not generally considered to be a good idea, another method was required that would still allow human-readable IDs.

First, adhering to the IDs used in the wireframes, a prefix was used for each category of entities that would benefit from a special generation strategy:

- ❖ “WST-” for tickets (short for WordSpot Ticket)
- ❖ “WSTH-” for ticket history items (short for WordSpot Ticket History)
- ❖ “WSU-” for users (short for WordSpot User)
- ❖ “WSF-” for files (short for WordSpot File)

Then, the next, variable part of the ID is generated by taking random digits from Base36, which contains all numbers and all capital letters. However, this strategy was prone to generating (potentially vulgar) vocabulary words and easily confused characters like zero and O. In response, all vowels and some similar characters were removed from the pool, ending up with essentially “Base28”.

Furthermore, for tickets, users and files 8 characters were used for the variable part of the ID, which means that 377,801,998,336 unique combinations could be generated before another character had to be added. For ticket history items, the character string length was set to 10, which resulted in 296,196,766,695,424 different possible combinations.

As an example, the ID Generator class file for tickets can be found under Appendix 7.19.

---

#### 4.3.4. COMMON PROJECT

*Related appendix item: 7.20*

As a prelude to introducing security to the system, a project titled Common was created, which would house the JWT configurations of the system. This project could then be included as a dependency in other projects; ensuring that the JWT configuration remained uniform across the projects that made use of it. Appendix item 7.20 shows the JWT configuration, which defines the following things:

- ❖ URI – The path (endpoint) through which tokens are issued
- ❖ HEADER – The key of the key-value pair that contains the token in request headers
- ❖ PREFIX – The string that precedes the token in the value of the key-value pair
- ❖ EXPIRATION – The expiration date of every issued token is set to be one day after it was issued
- ❖ SECRET – The secret string of characters used to sign & encrypt the token.

---

#### 4.3.5. AUTHORIZATION SERVICE

*Related appendix item: 7.21*

After including the Common project as a dependency in the Authorization Service project, the Spring Security framework had to be configured to work with JWT. To reiterate, the job of Authorization Service is to hand out access tokens after confirming that the received username and password pair is correct, after which this token can be used to send requests to secured endpoints.

A class named SecurityConfig was created (shown in Appendix 7.21) in the config package of the Authorization Service project, extending the Spring Security framework's WebSecurityConfigurerAdapter class and annotated with `@EnableWebSecurity`. In this class, two important parent methods are overwritten: one that configures our HTTP security and one that configures the authentication manager.

A custom filter had to be created and plugged in to our HTTP security configuration, which was named JwtUsernameAndPasswordAuthenticationFilter (extending the base UsernameAndPasswordAuthenticationFilter class) and placed in the same package. This class dictates that when a POST request arrives to the /auth endpoint, the supplied username and password are checked against a matching pair (using the username to get a user from User Service). If a user is found and the password can be translated to be equal to the stored password hash for that user, a token is generated. Then, the response sent back by Authorization Service will contain a header with the issued token and will have a status of 200 (OK).

---

#### 4.3.6. GATEWAY SERVICE

*Related appendix item: 7.22, 7.23*

With Authorization Service now issuing tokens, the next order of business was to set up a gateway that all requests had to go through. The first step was to map routes to other services in the application.properties file (shown in Appendix 7.22), which ensured that all those services were accessible through the gateway. For example, a PUT "/update/{ID}" endpoint in Ticket Service was now accessible through Gateway Service as "/tickets/update/{ID}".

After including the Common project as a dependency in this project too, the WebSecurityConfigurerAdapter class had to be configured in this project as well, and a

SecurityConfig class was created, similar to what was done in Authorization Service. However, the filter that had to be plugged in as a middleware was different in this case. The gateway was configured to do the following:

- ❖ Make sure that the incoming request originates from the system's frontend, also known as CSRF (Cross-site request forgery) protection
- ❖ Allow the request through if its target is a public endpoint
- ❖ Require a token if the destination is a secured endpoint and check its validity before forwarding the request. If no token is supplied or it is invalid, the request is denied and the sent response will have a status of 403 (UNAUTHORIZED).

#### 4.4. THE FRONTEND

*Related appendix item: 7.24, 7.25*

The frontend was created using the dependencies shown in Appendix 7.24. As per the requirements, the aim was to create a clean and responsive user interface for the system.

The current logged in user is stored (without its password field) in the uppermost parent component and passed as a prop to every component that makes use of it. Once logged in, the token received from the backend is stored along with the user so that it can be passed as authorization in a header when sending requests to the backend. Moreover, when a user successfully logs in, the received token is also stored locally in the browser, so that the login session can be restored when closing & opening the website again (provided that the token is still valid upon re-opening the website).

The website's operation was separated into three different modes, based on the requirements and design specifications:

- ❖ No user is logged in
  - Accessible pages include a login form and a customer registration form
- ❖ Customer is logged in
  - Customer-specific pages with less control but more explanations present in the form of tooltips and plain text
- ❖ Employee is logged in
  - Employee-specific pages with more control and minimal explanations

The responsiveness of the website can be largely attributed to Bootstrap's respective classes. When the viewport shrinks to tablet / mobile dimensions:

- ❖ All items on the pages are transformed to a single-column layout
- ❖ The sidebar (present when a user is logged in) transforms to a burger menu with a toggle

#### 4.5. CONTAINERIZATION

There were two important aspects to deal with when it came to containerization: building and running. Both of these lifecycle operations had to be isolated in a containerized environment to fully enable the application to deploy anywhere regardless of environment. At this point, I had a Maven project used as a dependency in other projects, six Spring-Maven projects and one Node project.

---

#### 4.5.1. BACKEND CONTAINERIZATION

*Related appendix items: 7.26, 7.27, 7.28*

Containerization of the backend projects came first. To achieve this, a base image had to be created that was based on a Maven-ready environment and included all necessary dependencies of all the backend projects to avoid having to download them separately when building each project. Appendix 7.26 shows the Dockerfile that is used to create a base builder image. I also created a batch file (Appendix 7.27) to easily execute and build the image described in the builder Dockerfile.

Comments are provided to explain what the Dockerfile is doing, but the important part is that the resulting image is used as a basis to compile and build the six backend projects in a containerized environment.

With the builder image ready, individual Dockerfiles had to be created for all six backend Spring projects. Appendix item 7.28 shows the Dockerfile for Ticket Service as an example, but all the other Dockerfiles are similar, the only differences being the port exposed and the name of the JAR files used.

In these individual Dockerfiles, Docker is told to use the previously created builder image as a base to compile and produce a JAR file from the source files of a given project. This JAR is then transferred to another environment which is based on a version 11 Java Runtime Environment base image. Finally, the ENTRYPOINT command tells Docker how to start the application inside the image. This process of giving building and running instructions in the same Dockerfile is known as a multi-stage build. (Docker Inc., n.d.)

---

#### 4.5.2. FRONTEND CONTAINERIZATION

*Related appendix item: 7.29*

Containerizing the frontend did not require a separate base builder image, because it is a one-of-a-kind project in this system (i.e. it is the only one that uses a Node environment). A multi-stage build process similar to the backend projects' individual Dockerfiles could be applied here by itself. Appendix item 7.29 shows the Dockerfile for Dashboard Service.

The image for Dashboard Service is based on a Node environment (that runs on Alpine, a lightweight operating system based on Linux). After copying the source files and the package JSON files inside the image, the "npm install" command is used to install the project's dependencies within the container and the final command tells Docker how to start the application inside the image.

---

#### 4.5.3. CONTAINER ORCHESTRATION

*Related appendix item: 7.30*

To synchronize and run the created images together, Docker Compose was used, which is a native functionality provided as part of the Docker engine. Creating a "docker-compose.yml" file allows developers to define the containers (and their configurations) that make up a Dockerized system. It can also be used to build all the images with just one command (provided that the backend builder image already exists in this project's case).

The lengthy docker-compose.yml (shown in Appendix 7.30) defines 10 containers in total: the six backend projects, three individual database containers for the three backend projects that use them and the frontend project. Additionally, it also defines four volumes, which are folders on the host machine mapped to a container. Three of these four volumes are used to persist the data in our databases, and one is used by file-service to store uploaded files.

The container system is set up so that only two containers are accessible from the host machine: Dashboard Service (port 3000) and Gateway Service (port 8762). These are the only two services that have their used ports mapped to the host system. Dashboard Service is accessed from any browser by navigating to “localhost:3000” and it makes requests to the backend through Gateway Service, which is “localhost:8762”.

## 4.6. TESTING

*Testing was realized on a “per every page” basis, covering all possible interactions on a given screen or element. Every item in the lists of the following sections indicate a test case with an expectation, all of which have passed.*

---

### 4.6.1. LOGIN (LANDING) PAGE TESTS

- ✓ Navigating to “localhost:3000” in the browser with no prior logins loads the login screen
- ✓ Clicking on “Customer Registration” loads the customer registration screen
- ✓ Clicking on “Forgot Your Password?” does nothing, as this feature is outside the scope of this project
- ✓ Clicking “Login” with a missing username or password field shows an error alert with the appropriate message (“Missing username or password!”)
- ✓ Clicking “Login” with invalid user credentials shows an error alert with the appropriate message (“Invalid username or password!”)
- ✓ Clicking “Login” after entering valid user credentials loads the overview page specific to the role of the user logging in and displays a success alert (“Successful login!”)

---

### 4.6.2. REGISTRATION PAGE TESTS

- ✓ Clicking “Back to Login” loads the login screen
- ✓ Clicking “Submit” with no fields filled out shows two error alerts with the appropriate messages (“Missing required fields!” and “Please enter a valid email address!”)
- ✓ Entering a valid email address into the appropriate field and leaving all other fields empty before clicking “Submit” shows an error alert with the appropriate message (“Missing required fields!”)
- ✓ Filling out the entire form except for one field at a time before clicking “Submit” shows an error alert with the appropriate message (“Missing required fields!”)
- ✓ If the inputs in the “Password” and “Confirm Password” fields don’t match after clicking “Submit”, an error alert is shown (“Passwords don’t match!”)

- ✓ Filling out the entire form correctly, but giving an email address that is already in use before pressing “Submit” displays an error alert (“Email address already in use!”)
- ✓ Filling out the entire form correctly, but giving a username that is already in use before pressing “Submit” displays an error alert (“Username already in use!”)
- ✓ Filling out the form correctly and then pressing “Submit” navigates back to the login screen and displays a success alert (“Successful registration!”)

---

#### 4.6.3. NAVIGATION BAR / TOP BAR TESTS

- ❖ Logged in as an Employee (PM or SME)
  - ✓ Clicking on “Overview” shows the employee overview screen
  - ✓ Clicking on “Navigation” shows the employee overview screen
  - ✓ Clicking on the WordSpot logo or the WordSpot title shows the employee overview screen
  - ✓ Clicking on “Profile” shows the profile screen
  - ✓ Clicking on “Categories” shows the categories screen
  - ✓ Clicking on “Browse Tickets” shows the ticket browser screen
  - ✓ Clicking on “New Ticket” shows the ticket creation screen
  - ✓ Clicking on “My Assigned Tickets” shows the own assigned tickets screen
  - ✓ Clicking on “My Pending Tickets” shows the own pending tickets screen
  - ✓ Clicking on “My Closed Tickets” shows the own closed tickets screen
  - ✓ Navigating to a ticket view screen adds the ticket ID as a link under the “Recently Viewed” section in the sidebar
  - ✓ Clicking on a ticket ID under “Recently Viewed” shows the ticket view screen for that ticket
  - ✓ Clicking on “Logout” navigates back to the login screen and shows a success alert (“Successful logout!”)
- ❖ Logged in as a Customer
  - ✓ Clicking on “Overview” shows the customer overview screen
  - ✓ Clicking on “Navigation” shows the customer overview screen
  - ✓ Clicking on the WordSpot logo or the WordSpot title shows the customer overview screen
  - ✓ Clicking on “Profile” shows the profile screen
  - ✓ Clicking on “Request Job” shows the customer service request screen
  - ✓ Clicking on “Active Tickets” shows the customer active tickets screen
  - ✓ Clicking on “Archived Tickets” shows the customer archived tickets screen
  - ✓ Navigating to a ticket view screen adds the ticket ID as a link under the “Recently Viewed” section in the sidebar
  - ✓ Clicking on a ticket ID under “Recently Viewed” shows the ticket view screen for that ticket
  - ✓ Clicking on “Logout” navigates back to the login screen and shows a success alert (“Successful logout!”)

---

#### 4.6.4. EMPLOYEE OVERVIEW PAGE TESTS

- ✓ Navigating to “localhost:3000” in the browser with an active login session stored for this employee loads the employee overview screen



- ✓ The “My Assigned Tickets” area shows the number of tickets assigned to the logged in employee with the status “assigned”
- ✓ The “My Pending Tickets” area shows the number of tickets assigned to the logged in employee with the status “pending”
- ✓ The “My Closed Tickets” area shows the number of tickets assigned to the logged in employee with the status “closed”
- ✓ The “Total Active Tickets” area shows the total number of tickets in the system that have “unassigned”, “assigned” or “pending” statuses.
- ✓ The “My Recently Updated Tickets” area shows some information about at most three tickets assigned to the logged in user, ordered by their last updated date.
- ✓ The “My Oldest Assigned Tickets” area shows some information about at most three tickets assigned to the logged in user with the status “assigned”, ordered by their age
- ✓ Clicking on a ticket in the “My Recently Updated Tickets” or “My Oldest Assigned Tickets” areas shows the ticket view screen for that ticket

---

#### 4.6.5. CUSTOMER OVERVIEW PAGE TESTS

- ✓ Navigating to “localhost:3000” in the browser with an active login session stored for this customer loads the customer overview screen
- ✓ The “Active Tickets” area shows the number of tickets that were requested by the logged in customer (applicable statuses are “unassigned” and “assigned”) along with a description of what ‘active’ means
- ✓ The “Archived Tickets” area shows the number of tickets that were requested by the logged in customer (applicable status is “closed”) along with a description of what ‘archived’ means
- ✓ The Pending Tickets” area shows the number of tickets that were requested by the logged in customer (applicable status is “pending”) along with a description of what ‘pending’ means

---

#### 4.6.6. PROFILE PAGE TESTS

- ✓ The “Base Data” area shows the following information about the current user: user ID, date of birth, account creation date and user role.
- ✓ Fields in the “Base Data” section cannot be changed
- ✓ The “Changeable Data” section shows the following information about the current user: email address, username, first name(s) and last name.
- ✓ Fields in the “Changeable data” section can be altered
- ✓ The “Password Change” section shows two empty fields: new password and confirm new password
- ✓ Clicking “Discard Changes” resets all changeable data to their initial values and clears the password fields
- ✓ Clicking “Confirm Changes” when a required field is empty shows an error alert asking the user to fill in affected fields
- ✓ Clicking “Confirm Changes” with valid or unchanged fields shows a success alert (“All changes saved!”)
- ✓ Clicking “Confirm Changes” with a valid, but already taken username / email fields results in an error alert being shown

- ✓ Clicking “Confirm Changes” with non-matching password fields results in an error alert being shown

---

#### 4.6.7. CATEGORIES PAGE TESTS

- ✓ The table displays all the categories stored in the Ticket Service category database
- ✓ Clicking the green plus icon in the last row in the table brings up the form to add a new category and changes the controls in that row to a green checkmark and a red cross
- ✓ Clicking the checkmark to add the new category when the name given in the appropriate field is less than three characters long results in an error alert (“Name must be longer than 3 letters!”)
- ✓ Clicking the checkmark when the name given is longer than 3 letters results in the creation of the new category, followed by a success alert (“Category created!”). The table is refreshed to display the new category
- ✓ Clicking the cross to cancel adding a new category hides the form
- ✓ Clicking the pencil icon next to an existing category changes that row into a form and changes the controls to a green checkmark and a red cross
- ✓ Clicking the checkmark to save changes to the existing category when the name is less than three characters long results in an error alert (“Name must be longer than 3 letters!”)
- ✓ Clicking the checkmark to save changes to the existing category when the name is longer than three letters results in the modification of the existing category, followed by a success alert (“Category changes saved!”). The table is refreshed to display the changed category
- ✓ Clicking the cross to cancel the editing of an existing category returns that row to plain text view
- ✓ Clicking the bin icon in an existing category’s row deletes the category and displays a success alert (“Category deleted!”). All tickets that had that category are changed to have the default, first category.

---

#### 4.6.8. TICKET BROWSER PAGE TESTS

- ✓ The table displays all tickets in the system initially.
- ✓ Clicking the “Toggle Search Filters” button hides or shows the search filters.
- ✓ Changing the “Status Filter” dropdown updates the table to only show tickets with the selected status.
- ✓ Changing the “Ticket ID” search filter updates the table to only show tickets that have the search string in their ID as a substring (case-insensitive)
- ✓ Changing the “Assigned To” search filter updates the table to only show tickets that have the search string in their ‘assigned to’ field as a substring (case-insensitive)
- ✓ Changing the “Customer” search filter updates the table to only show tickets that have the search string in their ‘customer’ field as a substring (case-insensitive)
- ✓ Clicking the “View” button in a row of the table shows the ticket view screen for the selected ticket

---

#### 4.6.9. NEW TICKET PAGE TESTS

- ❖ Employee-specific page (New Ticket)

- ✓ On page load, the “Ticket ID” field shows “Assigned after creation” and the “Ticket Author” field shows the name and ID of the logged in employee. Neither field is changeable
  - ✓ Typing into the “Customer” field continuously brings up results from the customer pool for selection
  - ✓ The “Category” dropdown shows all categories defined in the “Categories” section
  - ✓ Typing into the “Assigned To” field continuously brings up results from the employee pool for selection
  - ✓ The “Status” dropdown shows the “Unassigned”, “Assigned”, “Pending” and “Closed” options
  - ✓ Clicking the “Choose file” button under the “Attachment” section brings up a file chooser window. After selection, the file name is shown next to the button instead of “No file chosen”
  - ✓ Clicking “Reset Form” resets the form to its initial state
  - ✓ Clicking “Create Ticket” without a valid, selected “Customer” field shows an error alert (“Tickets must have a targeted customer!”)
  - ✓ Clicking “Create Ticket” without typing anything into the “Description” field shows an error alert (“Tickets must have a description!”)
  - ✓ Clicking “Create Ticket” after typing into the “Description” field and selecting a valid customer in the “Customer” field, a success alert is shown (“Ticket created!”), and the ticket view page is shown for the newly created ticket
- ❖ Customer-specific page (Request Job)
- ✓ On page load, the “Ticket ID” field shows “Assigned after creation” and the “Requester” field shows the name and ID of the logged in customer. Neither field is changeable
  - ✓ The “Category” dropdown shows all categories present in the system
  - ✓ Clicking the “Choose file” button under the “Attachment” section brings up a file chooser window. After selection, the file name is shown next to the button instead of “No file chosen”
  - ✓ Clicking “Cancel” navigates away to the customer overview screen
  - ✓ Clicking “Request Job” without giving a description in the appropriate field, an error alert is shown (“Tickets must have a description!”)
  - ✓ Clicking “Request Job” with a description filled out shows a success alert (“Ticket created!”) and loads the ticket view screen for the newly created ticket.

---

#### 4.6.10. TICKET VIEW PAGE TESTS

- ✓ Employee-specific view page
  - ✓ On page load, the “Ticket ID” and “Ticket Author” fields show their values for this ticket but are not changeable.
  - ✓ On page load, the “Customer”, “Category”, “Assigned To” and “Status” fields show their current values for this ticket and are changeable.
  - ✓ The “Category” dropdown shows all categories present in the system
  - ✓ The “Status” dropdown shows the “Unassigned”, “Assigned”, “Pending” and “Closed” options

- ✓ The “Attachments” section shows the files associated with this ticket. If there are no related files, it shows “No files associated with this ticket.”
- ✓ Clicking on a file link in the “Attachments” section initiates the download of that file
- ✓ The “History” section shows all related history items for this ticket
- ✓ Clicking “Update Ticket” shows a success alert (“Ticket updated!”) and updates to ticket to the values set before pressing the button
- ✓ Customer-specific view page
  - ✓ On page load, the “Ticket ID”, “Ticket Author”, “Customer”, “Category”, “Assigned To” and “Status” fields show their current values for this ticket but are not changeable by the customer
  - ✓ The “Category” dropdown shows all categories present in the system
  - ✓ The “Status” dropdown shows the “Unassigned”, “Assigned”, “Pending” and “Closed” options
  - ✓ The “Attachments” section shows the files associated with this ticket. If there are no related files, it shows “No files associated with this ticket.”
  - ✓ Clicking on a file link in the “Attachments” section initiates the download of that file
  - ✓ The “History” section shows all related history items for this ticket
  - ✓ If the ticket status is “Unassigned” or “Assigned”, clicking the “Update Ticket” button without a description given shows an error alert (“Updates must have a description!”)
  - ✓ If the ticket status is “Unassigned” or “Assigned”, clicking the “Update Ticket” button with a description given updates the ticket, refreshes the “History” section and shows a success alert (“Ticket updated!”)
  - ✓ If the ticket status is “Pending”, clicking the “Accept” button changes the ticket status to “Closed”, removes the “Update Ticket” section and shows a success alert (“Ticket closed!”)
  - ✓ If the ticket status is “Pending”, clicking the “Refuse” button without giving a description shows an error alert (“Please provide a reason in the description field!”)
  - ✓ If the ticket status is “Pending”, clicking the “Refuse” button with a description given updates the ticket status back to “Assigned” and refreshes the “History” section.
  - ✓ If the ticket status is “Closed”, no controls are present and no data can be changed, however, the ticket’s history and all associated files are still accessible.

---

#### 4.6.11. MY ASSIGNED / PENDING / CLOSED TICKETS (EMPLOYEES ONLY)

These pages work on the same principle as the Ticket Browser, the only difference being that the filtered status is determined by the page itself instead of being variable in a user input field.

---

#### 4.6.12. ACTIVE TICKETS / ARCHIVED TICKETS PAGES (CUSTOMERS ONLY)

These pages are yet more restricted versions of the Ticket Browser page. Only tickets in which the logged in customer is indicated as the “Customer” are shown. The difference is that search filters are not available here, and the ticket statuses have icons next to them that explain what the status means if the mouse is hovered over them. This was the only unique test case on these pages in addition to some already existing ones.

---

#### 4.6.13. RESPONSIVENESS TESTS

- ✓ When the screen width shrinks below 991px, the sidebar automatically disappears, and a burger icon appears in the top bar before the WordSpot logo
- ✓ When the burger is clicked, the main screen gets slightly greyed out and the sidebar menu appears on the left
- ✓ Clicking anywhere off the menu causes it to disappear
- ✓ On mobile devices, swiping from the right from the left edge of the screen drags the sidebar menu out
- ✓ On mobile devices, swiping left when the menu is open drags the sidebar menu off the screen
- ✓ When the screen width shrinks below 991px, all “items” immediately transform to a single-column layout

## 5. EVALUATION

### 5.1. CRITICAL REFLECTION

Firstly, looking at the project aims, most of them have been achieved. Suitable technologies were researched and chosen, including Spring / Spring Boot, PostgreSQL, Docker and Netflix's open source microservice-focused tools like Eureka to list the most important ones. These tools were then investigated in-depth and used as a means to gain a deeper understanding into how microservices and containerization work by producing the main deliverable. However, being biased towards Java might have excluded some otherwise potentially more fitting tools to use for this project.

Secondly, the requirements specification was looked at, which was compared to the finished deliverable to validate it. Most of the functional requirements were met, although files can only be attached to tickets one at a time (i.e. one per update). Apart from this, the non-functional requirement that relates to unambiguous alerts could have been implemented more robustly, as request timeouts were not taken into account when preparing possible error messages. For example, when a user tries to log in, but the request sent to the backend times out, the displayed error message is "Invalid username or password!", which is not necessarily correct.

Moreover, security was implemented to a publishable standard (as per the corresponding non-functional requirement), however, it would have been beneficial to take it one step further and make checks for the user role included in the authorization tokens. At present, no such checks are made, and, as an example, a user with a "Customer" role could make a request to change ticket categories. Of course, the frontend doesn't provide the opportunity for Customers to do this, but the fact remains that they would be able to make such a request if they could.

Thirdly, the design specification was examined. All the use cases in the use case diagrams were implemented and the system architecture was realized as per the designs. However, the endpoints could have been made more uniform in the design section to begin with, as some endpoints used camelCase and some endpoints used hyphen-separated wording. Additionally, some endpoints had unnecessary paths in them, like the GET requests for User Service.

Finally, the project's scenario (i.e. the elaboration on how WordSpot works in the specification) has been changed slightly as the project progressed because of time constraints. These changes include:

- ❖ Removed maintained skills and skill levels from SMEs
- ❖ Removed the three levels of contracts that correlated to service quality (Gold, Silver and Bronze)
- ❖ Removed contracts and corporate customers; every customer gets served on an ad-hoc basis
- ❖ Removed the PM work review from a ticket's lifecycle. PMs only follow up on closed tickets where the customer was not happy with the result.
- ❖ A rejected ticket goes back to the SME who did the work instead of a PM

These were not significant changes in the context of what the project's main focus was, but they could have been implemented as well with better time management.

All in all, the project has successfully provided insight into how microservices work and how applications can benefit from containerization.

## 5.2. FUTURE ENHANCEMENT OPPORTUNITIES

An exciting modification to the deliverable would be to generalize all aspects of it. Instead of branding it as a tool for WordSpot to deal with writing-related requests, it could be modified to be a general, highly customizable ticketing system. This would also mean that parts of the system that are taken for granted (like ticket statuses and user roles) could be made configurable, similar to how ticket categories can be added, modified and deleted.

After making every such aspect customizable, the addition of an appropriate administrator user interface would enable the end-users of the product to set up their own roles, statuses, categories, branding and so on. The application could then be used by individual companies across multiple business sectors as their own ticketing system, be it public or internal use only.

Another interesting enhancement would be the addition of a mobile application to the frontend, along with the browser-based user interface. Because of the REST API based nature of the backend services, no changes would need to be implemented there, as the native mobile application could simply use the existing endpoints.

## 5.3. PERSONAL REFLECTION

One of the obstacles encountered during the realization of this project was the relative newness of the microservice architecture. There is a fair amount of resources available that deal with it, but there does not seem to be a general “valid” way of implementing them, so various aspects were up for interpretation.

The containerization aspect of the project has probably been the hardest part. Not because of the process itself, but because of the need to completely rethink how building and running an application (let alone a microservice-based one) works in a containerized environment, as this whole concept was new to me prior to starting this project.

If I had to do this project all over again, I would probably focus more on keeping in-line with the chosen development approach and pay more attention to keeping track of the work, because even though an Agile approach (FDD) was adopted, I feel like it did not play as big a role as it should have.

However, as a result of facing the many challenges this project has entailed, a lot has been learned, but not just about microservices and containerization. My planning, designing, development and management skills have all improved as a direct result of this undertaking. Most of all, I am glad to have had the opportunity to broaden my knowledge of the Java domain through implementing such a relatively complex project that is focused on fairly recent trends in the development world.

## 6. REFERENCES

- Axios. (n.d.). Retrieved from GitHub: <https://github.com/axios/axios>
- Balsamiq Studios, LLC. (n.d.). *Balsamiq*. Retrieved from Balsamiq: <https://balsamiq.com/>
- Bootstrap. (n.d.). Retrieved from Bootstrap: <https://getbootstrap.com/>
- Docker Inc. (n.d.). *Overview of Docker Compose*. Retrieved from Docker Docs: <https://docs.docker.com/compose/>
- Docker Inc. (n.d.). *Use multi-stage builds*. Retrieved from Docker Docs: <https://docs.docker.com/develop/develop-images/multistage-build/>
- Docker Inc. (n.d.). *What is a Container?* Retrieved from Docker: <https://www.docker.com/resources/what-container>
- JSON Web Token Introduction. (n.d.). Retrieved from JWT: <https://jwt.io/introduction/>
- Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka. (n.d.). Retrieved from Spring Blog: <https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka>
- Netflix/zuul. (n.d.). Retrieved from GitHub: <https://github.com/Netflix/zuul>
- PostgreSQL vs MySQL. (n.d.). Retrieved from 2ndQuadrant: <https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/>
- PostgreSQL: About. (n.d.). Retrieved from PostgreSQL: <https://www.postgresql.org/about/>
- React. (n.d.). Retrieved from React: <https://reactjs.org/>
- React Bootstrap. (n.d.). Retrieved from React Bootstrap: <https://react-bootstrap.github.io/>
- React Router. (n.d.). Retrieved from React Training: <https://reacttraining.com/react-router/web/guides/philosophy>
- Singh, A., Chawla, P., Singh, K., & Singh, A. K. (2018). Formulating an MVC Framework for Web Development in JAVA. *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 926-929). Tirunelveli: IEEE.
- Spring | Web Applications. (n.d.). Retrieved from Spring: <https://spring.io/web-applications>
- Spring Boot. (n.d.). Retrieved from Spring: <https://spring.io/projects/spring-boot>
- Spring Initializr. (n.d.). Retrieved from Spring Initializr: <https://start.spring.io/>
- Spring Security. (n.d.). Retrieved from Spring: <https://spring.io/projects/spring-security#overview>
- The Project Lombok Authors. (n.d.). *Project Lombok*. (Project Lombok) Retrieved 2020, from Project Lombok: <https://projectlombok.org/>
- Thorben Janssen. (2018, May 7). *SOLID Design Principles Explained*. Retrieved from Stackify: <https://stackify.com/dependency-inversion-principle/>



## 7. APPENDICES

### 7.0. PROJECT SPECIFICATION

#### PROJECT SPECIFICATION - Project (Technical Computing) 2019/20

Student:	Zoltán Nagy
Date:	24/10/2019
Supervisor:	Dr Soumya Basu
Degree Course:	BENG HON SOFTWARE ENGINEERING
Title of Project:	Implementation of a Containerised Microservice Architecture

##### Elaboration

Microservices are on the rise, offering many advantages over the traditional monolithic approach in certain scenarios.

The focus of this project is to develop an application using a microservice architecture and deploy using a containerized approach.

The system will be modelled around a suitably relevant example, which is as follows:

WordSpot is a *fictional* company that provides a range of writing services, such as translation, report writing, contract preparation and proofreading. It has individual and corporate customers. Corporate customers sign a contract for a defined duration and individual customers sign task-based contracts. The contracts can be one of three categories: Gold, Silver and Bronze, with corresponding levels of service quality. (e.g. Completion time, language standard, indicative error rate)

The company employs Project Managers (PMs) and Subject Matter Experts (SMEs). Skill levels and skill categories are maintained for each SME (Proficient, Expert, Familiar and e.g. English to French Translation, Report writing, etc.)

Customers can request a service through an online interface. This creates a service request ticket in the system that contains the details of the customer, the service required, any source files and optional additional information.

A newly created ticket goes to a PM. The PM determines the suitable SMEs to undertake the ticket and broadcasts it to them. Interested SMEs show their interest by applying to complete the ticket. The PM then assigns the ticket to an available SME based on experience and skill level. The SME carries out the relevant work and updates the ticket, which is then reviewed by the PM who originally received the ticket.

The completed work is then made available to the customer, who can either accept or reject the work. If the work is rejected, the ticket goes back to the PM, who can choose to repeat the above process or assign the ticket back to the SME who carried out the work.

##### Project Aims

- ☐ Research & learn suitable technologies for the project
- ☐ Learn how to implement and work with microservices
- ☐ Learn how to secure microservice endpoints & general microservice security
- ☐ Produce the individual microservices that will make up the application
- ☐ Produce an application to satisfy the concepts given in the elaboration
- ☐ Create a suitably attractive online front-end user interface for the application
- ☐ Learn the advantages of a microservice architecture compared to a monolithic one

**Project deliverable(s)**

The main deliverable of the project will be an application to implement the concepts outlined in the elaboration.

I will be creating and deploying a microservice-based application that will be made up by several smaller applications (microservices) using an agile approach.


Other deliverables include a requirement specification, a design specification, the codebase and test cases.

**Action plan**

<b>Submit Project Specification</b>	<b>On 25 October 2019</b>
Research & learn to use useful dependencies for the application	By 11 November 2019
Learn & practice constructing simple applications with microservice architectures	By 18 November 2019
Plan the individual microservices that will make up the application for the project	By 25 November 2019
Sprint 0: Set up version control & development environment	2 December 2019 - 8 December 2019
Sprint 1: Requirement & Design Specifications	9 December 2019 – 22 December 2019
Sprint 2: Back-end Implementation (Specifics TBD)	6 January 2020 – 19 January 2020
Sprint 3: Back-end Implementation (Specifics TBD)	20 January 2020 – 2 February 2020
Sprint 4: Front-end Implementation (Specifics TBD)	3 February 2020 – 16 February 2020
Sprint 5: Front-end Implementation (Specifics TBD)	17 February 2020 – 1 March 2020
Sprint 6: Containerization	2 March 2020 – 15 March 2020
Sprint 7: TBD, depends on previous sprints	16 March 2020 – 29 March 2020
Sprint 8: Testing	30 March 2020 – 12 April 2020
Refine & Finish the Project Report	13 April 2020 – 20 April 2020
Turnitin Submission	On 22 April 2020
Submit Bound Copy + Electronic Version	On 23 April 2020
Demo to Supervisor & Second Marker	Before 12 May 2020

#### BCS Code of Conduct

I confirm that I have successfully completed the BCS code of conduct on-line test with a mark of 70% or above. This is a condition of completing the Project (Technical Computing) module.

Signature: 

#### Publication of Work

I confirm that I understand the "Guidance on Publication Procedures" as described on the Bb site for the module.

Signature: 

#### GDPR

I confirm that I will use the "Participant Information Sheet" as a basis for any survey, questionnaire or participant testing materials. This form is available on the Bb site for the module.

Signature: 

#### Ethics

Complete the SHUREC 7 (research ethics checklist for students) form below. If you think that your project may include ethical issues that need resolving (working with vulnerable people, testing procedures etc.) then discuss this with your supervisor as soon as possible and comment further here.

Both you and your supervisor need to sign the completed SHUREC 7 form.

Please contact the project co-ordinator if further advice is needed.

### RESEARCH ETHICS CHECKLIST FOR STUDENTS (SHUREC 7)

This form is designed to help students and their supervisors to complete an ethical scrutiny of proposed research. The SHU [Research Ethics Policy](#) should be consulted before completing the form.

Answering the questions below will help you decide whether your proposed research requires ethical review by a Designated Research Ethics Working Group.

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements for keeping data secure and, if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management.

The form also enables the University and Faculty to keep a record confirming that research conducted has been subjected to ethical scrutiny.

For student projects, the form may be completed by the student and the supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader, and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in their research projects, and staff should keep a copy in the student file.

Please note if it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the Faculty Safety Co-ordinator.

# General Details

Name of student	Zoltán Nagy
SHU email address	zoltan.nagy@student.shu.ac.uk
Course or qualification (student)	BENG HON SOFTWARE ENGINEERING
Name of supervisor	Dr Soumya Basu
email address	soumya.basu@shu.ac.uk
Title of proposed research	Individual Research on Microservices
Proposed start date	25 October 2019
Proposed end date	23 April 2020
Brief outline of research to include, rationale & aims (250-500 words).	<p>The aim of this project is to develop an application built on microservices, deploy said application using containerisation and learn the differences &amp; advantages of the microservice architecture.</p> <ul style="list-style-type: none"> <li>▪ Research &amp; learn suitable technologies for the project</li> <li>▪ Learn how to implement and work with microservices</li> <li>▪ Learn how to secure microservice endpoints &amp; general microservice security</li> <li>▪ Produce the individual microservices that will make up the application</li> <li>▪ Produce an application to satisfy the concepts given in the elaboration</li> <li>▪ Create a suitably attractive online front-end user interface for the application</li> <li>▪ Learn the advantages of a microservice architecture compared to a monolithic one</li> </ul> <p>This project will not include external participants.</p>
Where data is collected from individuals, outline the nature of data, details of anonymisation, storage and disposal procedures if required (250-500 words).	No data will be collected from individuals.



**1. Health Related Research Involving the NHS or Social Care / Community Care or the Criminal Justice Service or with research participants unable to provide informed consent**

Question	Yes/No
<p>1. Does the research involve?</p> <ul style="list-style-type: none"> <li>• Patients recruited because of their past or present use of the NHS or Social Care</li> <li>• Relatives/carers of patients recruited because of their past or present use of the NHS or Social Care</li> <li>• Access to data, organs or other bodily material of past or present NHS patients</li> <li>• Foetal material and IVF involving NHS patients</li> <li>• The recently dead in NHS premises</li> <li>• Prisoners or others within the criminal justice system recruited for health-related research*</li> <li>• Police, court officials, prisoners or others within the criminal justice system*</li> <li>• Participants who are unable to provide informed consent due to their incapacity even if the project is not health related</li> </ul>	No
<p>2. Is this a research project as opposed to service evaluation or audit?</p> <p><i>For NHS definitions please see the following website</i>  <a href="http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf">http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf</a></p>	-

If you have answered **YES** to questions **1 & 2** then you **must** seek the appropriate external approvals from the NHS, Social Care or the National Offender Management Service (NOMS) under their independent Research Governance schemes. Further information is provided below.

NHS <https://www.myresearchproject.org.uk/Signin.aspx>

\* All prison projects also need National Offender Management Service (NOMS) Approval and Governor's Approval and may need Ministry of Justice approval. Further guidance at: <http://www.hra.nhs.uk/research-community/applying-for-approvals/national-offender-management-service-noms/>

**NB** FRECs provide Independent Scientific Review for NHS or SC research and initial scrutiny for ethics applications as required for university sponsorship of the research. Applicants can use the NHS pro-forma and submit this initially to their FREC.

**2. Research with Human Participants**

Question	Yes/No
Does the research involve human participants? This includes surveys, questionnaires, observing behaviour etc.	No
Question	Yes/No
1. <i>Note If YES, then please answer questions 2 to 10</i> <i>If NO, please go to Section 3</i>	-
2. Will any of the participants be vulnerable? <i>Note: Vulnerable people include children and young people, people with learning disabilities, people who may be limited by age or sickness, etc. See definition on website</i>	-
3. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	-
4. Will tissue samples (including blood) be obtained from participants?	-
5. Is pain or more than mild discomfort likely to result from the study?	-

6. Will the study involve prolonged or repetitive testing?	-
7. Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants? <i>Note: Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, etc.</i>	-
8. Will anyone be taking part without giving their informed consent?	-
9. Is it covert research? <i>Note: 'Covert research' refers to research that is conducted without the knowledge of participants.</i>	-
10. Will the research output allow identification of any individual who has not given their express consent to be identified?	-

If you answered **YES only** to question 1, the checklist should be saved and any course procedures for submission followed. If you have answered **YES** to any of the other questions you are **required** to submit a SHUREC8A (or 8B) to the FREC. If you answered **YES** to question 8 and participants cannot provide informed consent due to their incapacity you must obtain the appropriate approvals from the NHS research governance system. Your supervisor will advise.

### 3. Research in Organisations


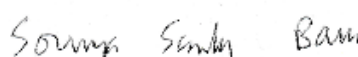
Question	Yes/No
1. Will the research involve working with/within an organisation (e.g. school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? <i>If YES, students please show evidence to your supervisor. PI should retain safely.</i>	-
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access.  <i>Note: You will only be able to start the research when you have been granted access.</i>	-

### 4. Research with Products and Artefacts

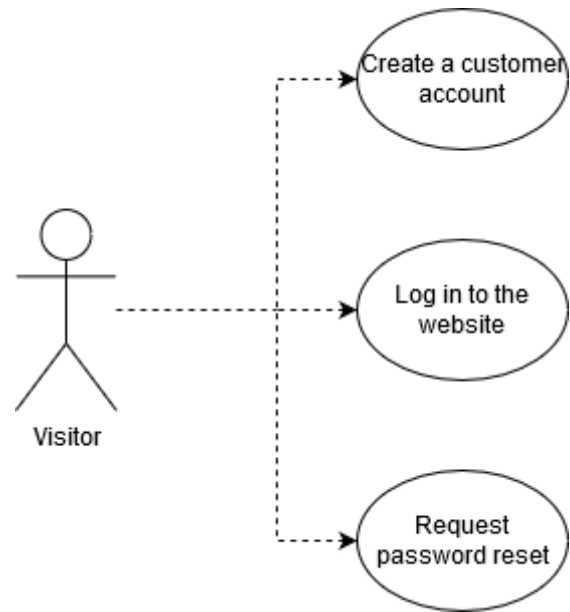
Question	Yes/No
1. Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes, existing datasets or secure data?	Yes

<p>2. If you answered YES to question 1, are the materials you intend to use in the public domain?</p> <p><i>Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.</i></p> <ul style="list-style-type: none"> <li>Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission.</li> <li>Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.</li> </ul> <p><i>If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British Educational Research Association.</i></p>	Yes
<p>3. If you answered NO to question 2, do you have explicit permission to use these materials as data?</p> <p><i>If YES, please show evidence to your supervisor.</i></p>	-
<p>4. If you answered NO to question 3, is it because:</p> <p>A. you have not yet asked permission</p> <p>B. you have asked and not yet received an answer</p> <p>C. you have asked and been refused access.</p> <p><i>Note: You will only be able to start the research when you have been granted permission</i></p>	-

#### Adherence to SHU policy and procedures

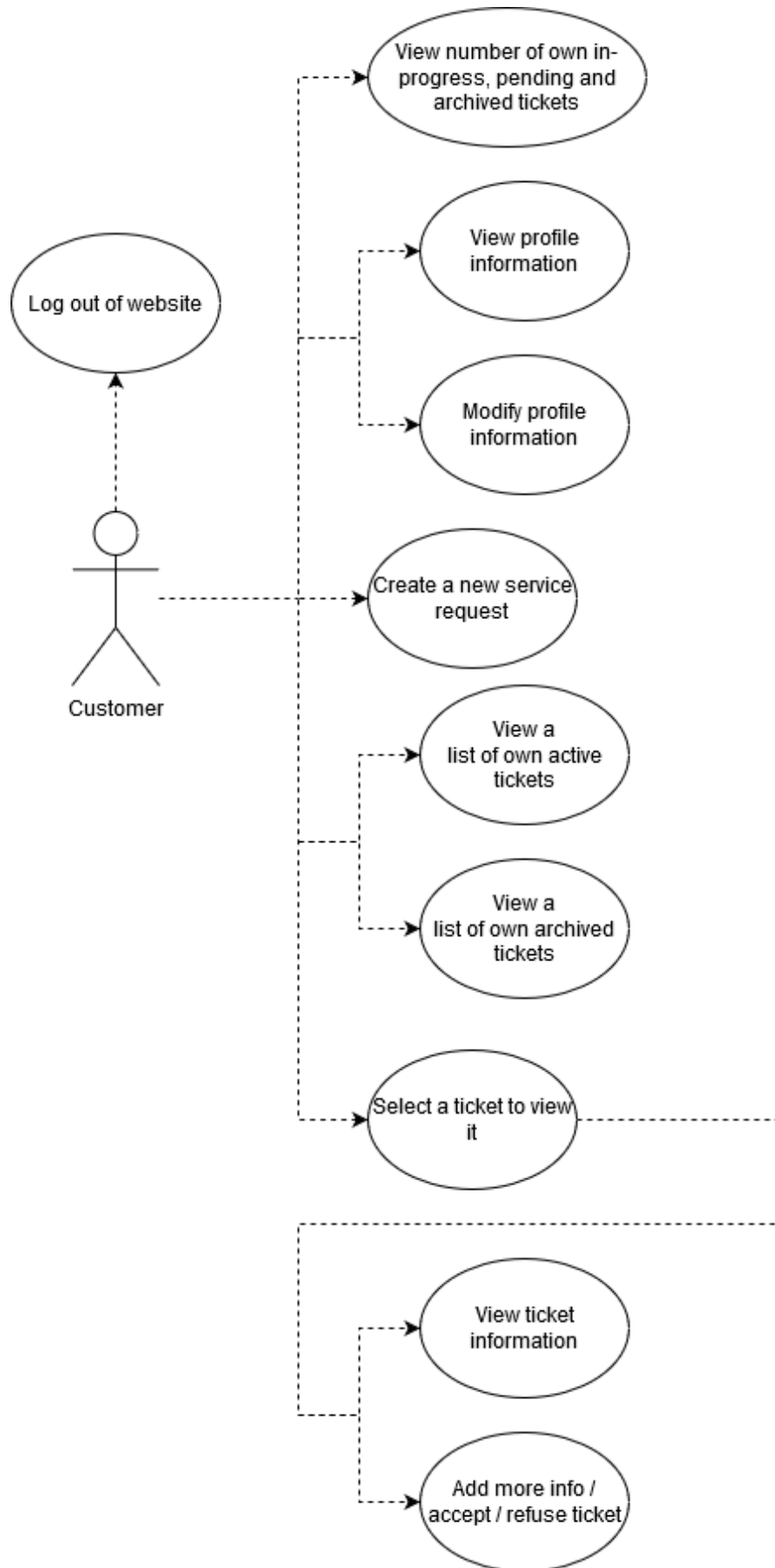
<b>Personal statement</b>	
I can confirm that:	
<input type="checkbox"/> I have read the Sheffield Hallam University Research Ethics Policy and Procedures <input type="checkbox"/> I agree to abide by its principles.	
<b>Student</b>	
Name: Zoltán Nagy	Date: 24/10/2019
Signature: 	
<b>Supervisor or other person giving ethical sign-off</b>	
I can confirm that completion of this form has not identified the need for ethical approval by the FREC or an NHS, Social Care or other external REC. The research will not commence until any approvals required under Sections 3 & 4 have been received.	
Name: Dr Soumya Basu	Date: 24/10/2019
Signature: 	

## 7.1. VISITOR USE CASE DIAGRAM

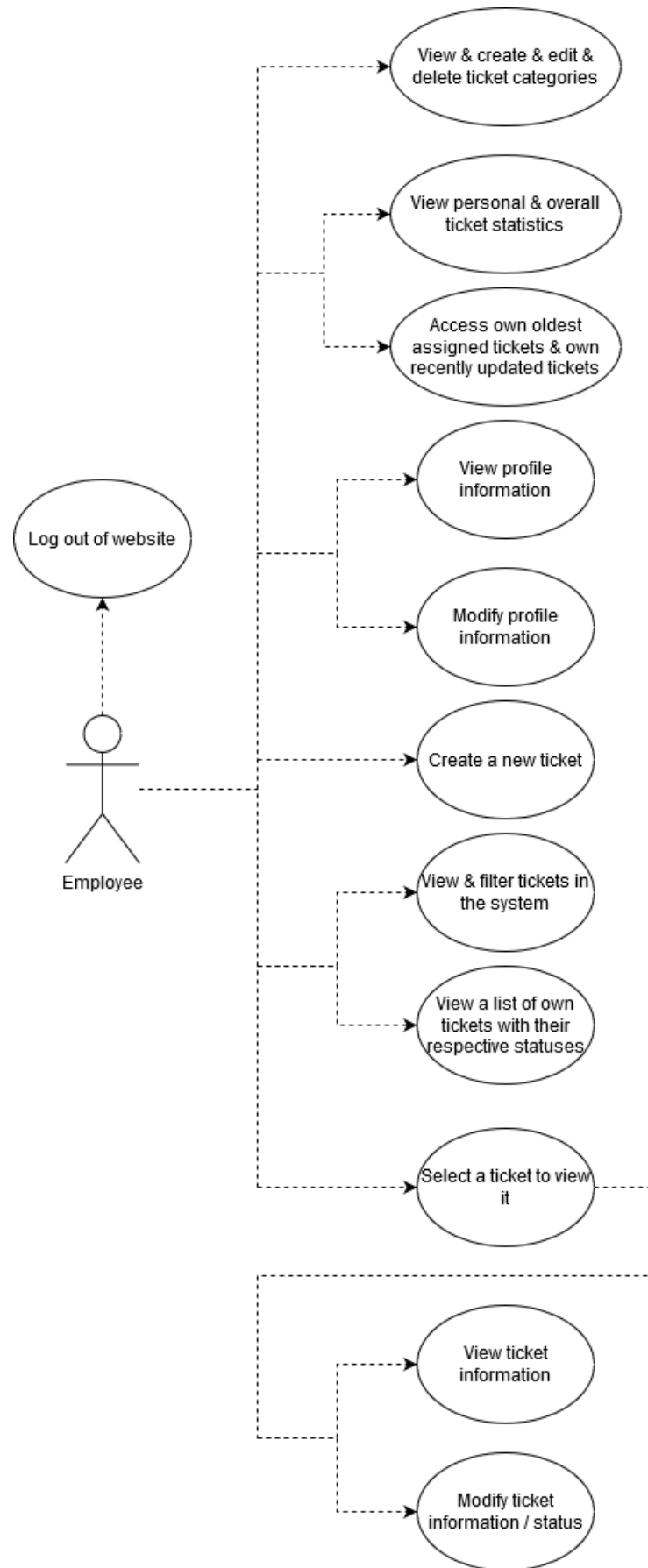




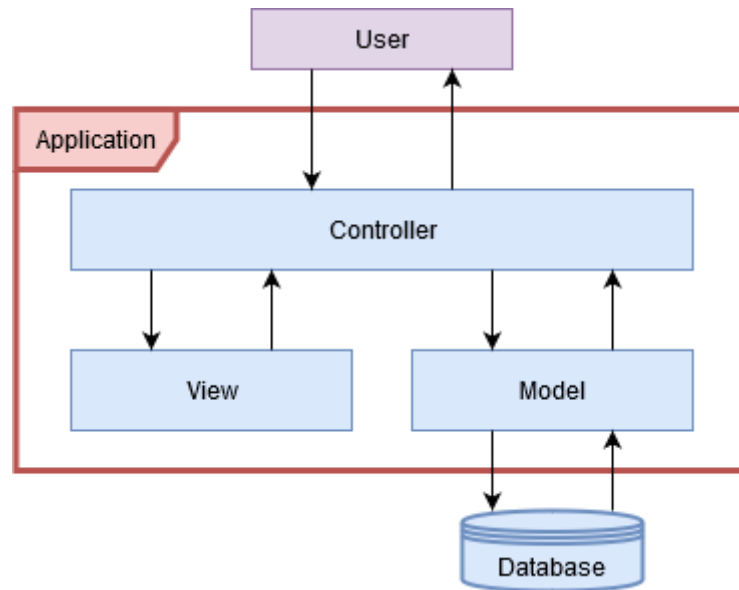
## 7.2. CUSTOMER USE CASE DIAGRAM



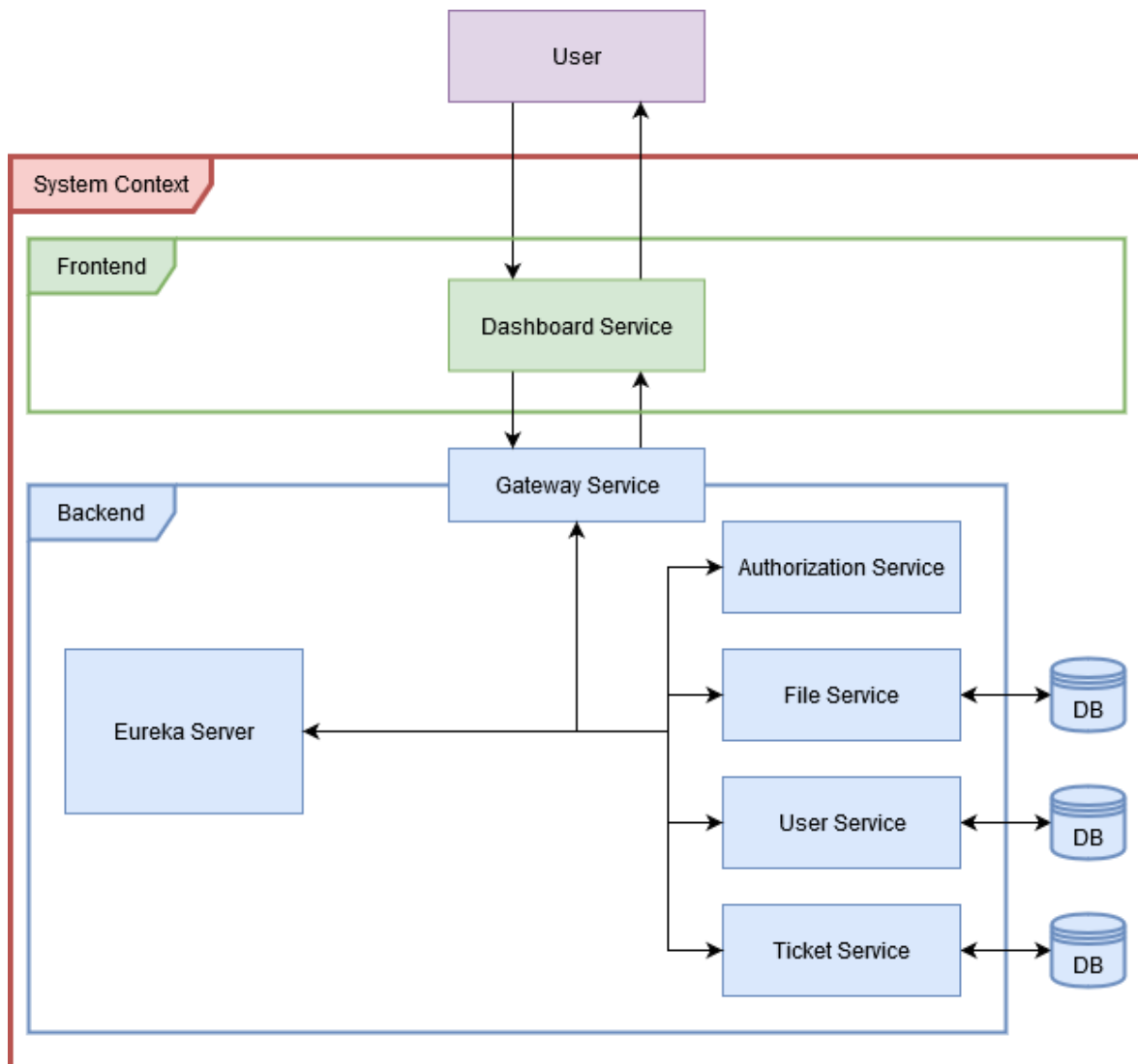
### 7.3. EMPLOYEE USE CASE DIAGRAM



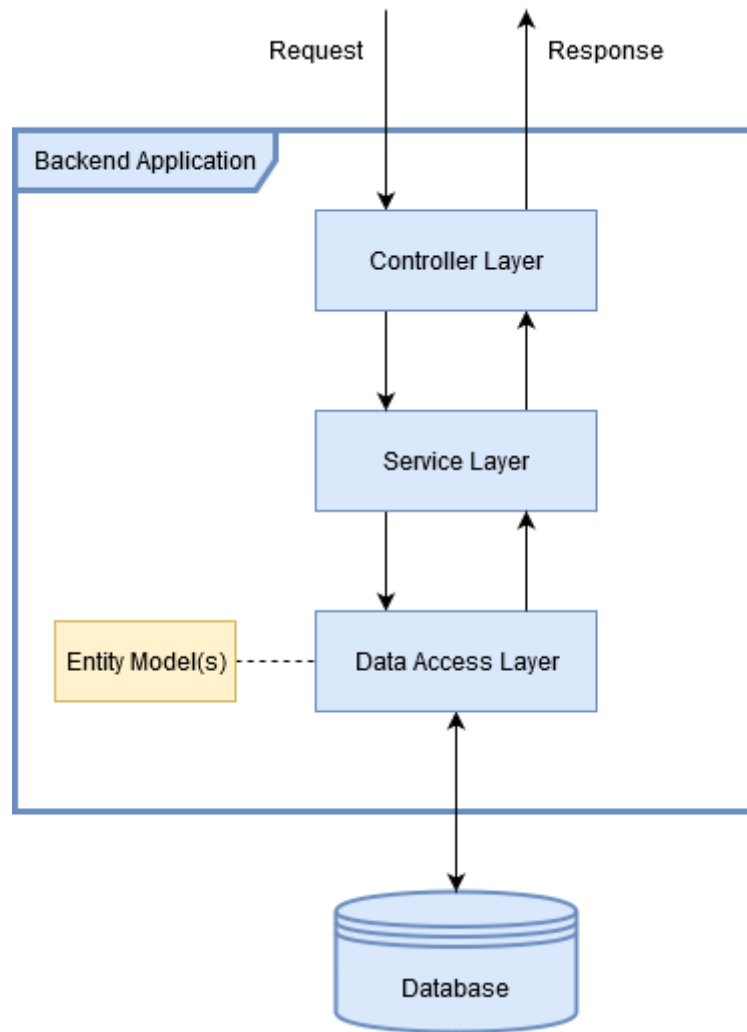
#### 7.4. MVC DESIGN PATTERN DIAGRAM



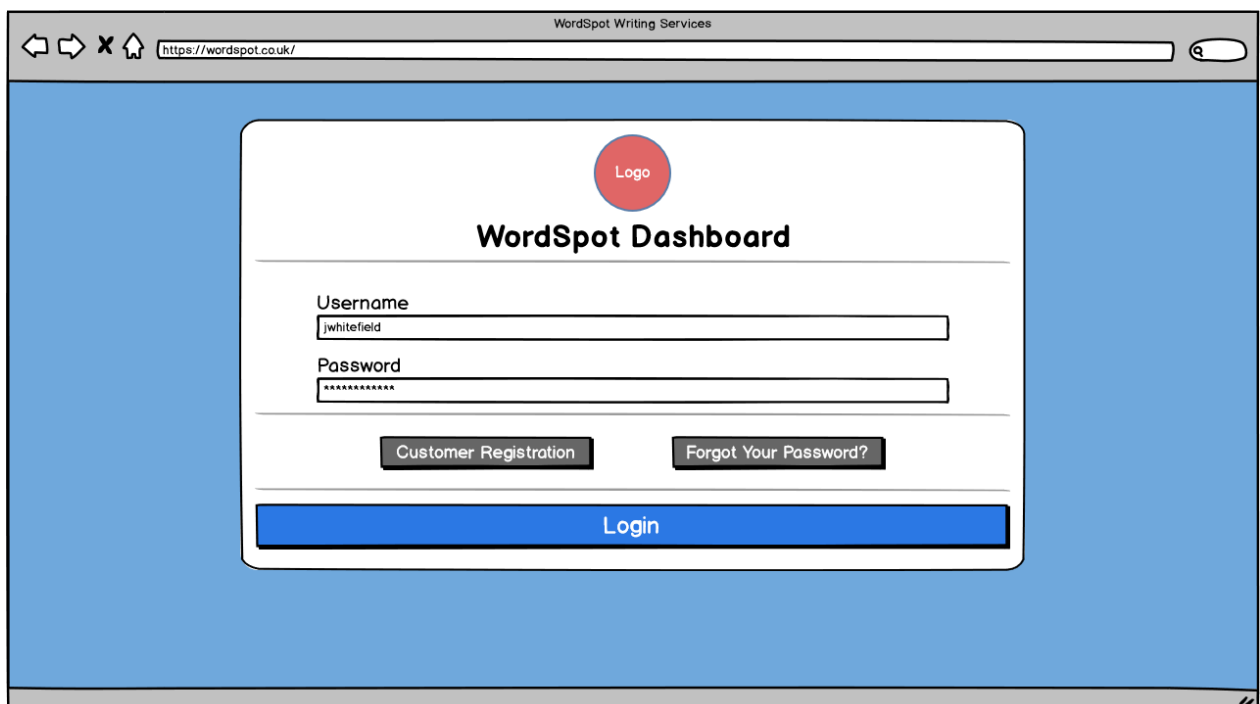
#### 7.5. WORDSPOT'S MICROSERVICE ARCHITECTURE DIAGRAM



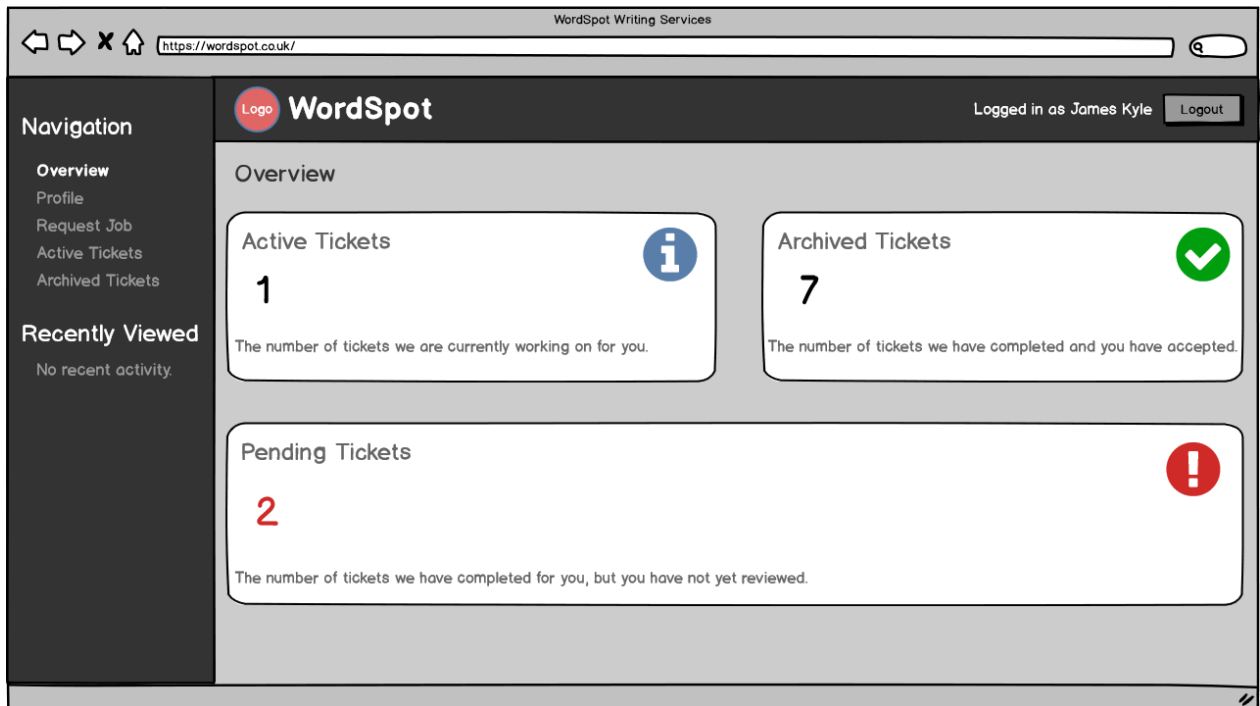
## 7.6. STRUCTURE OF BACKEND APPLICATIONS



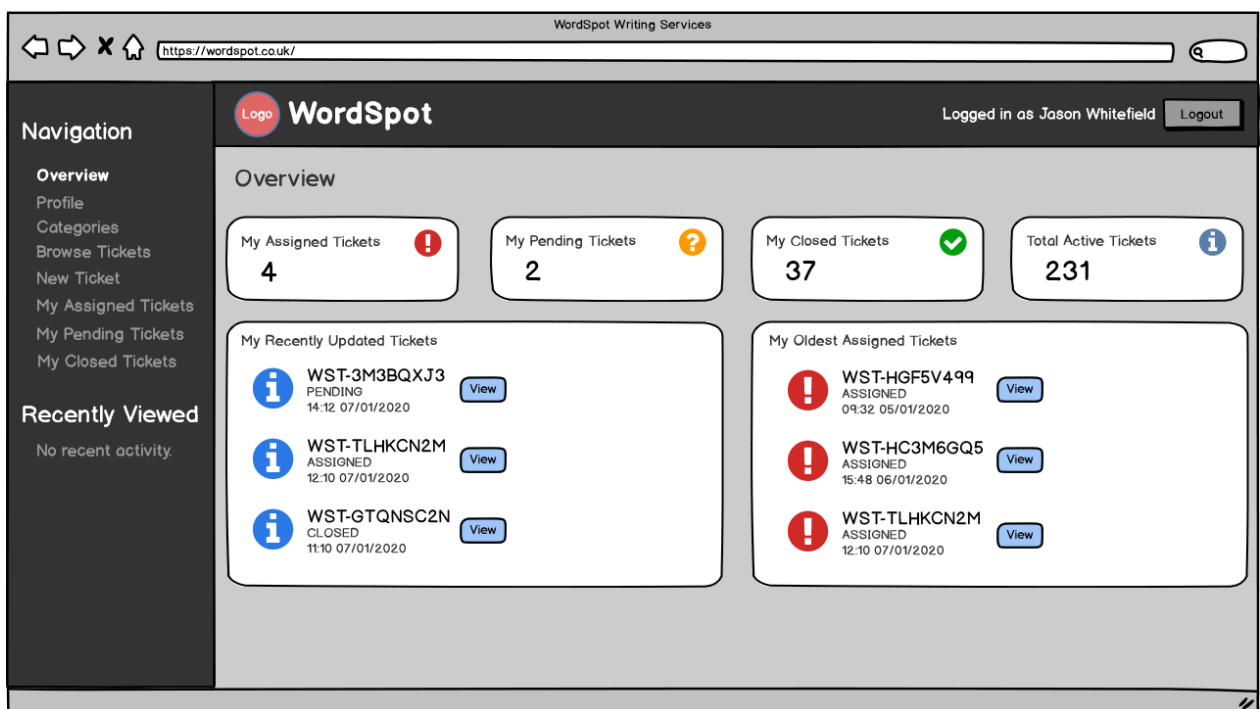
## 7.7. UI WIREFRAME – LOGIN SCREEN



## 7.8. UI WIREFRAME – CUSTOMER OVERVIEW



## 7.9. UI WIREFRAME – EMPLOYEE OVERVIEW



## 7.10. UI WIREFRAME – PROFILE

WordSpot Writing Services
https://wordspot.co.uk/

WordSpot
Logged in as Jason Whitefield
Logout

Navigation
Overview
Profile
Categories
Browse Tickets
New Ticket
My Assigned Tickets
My Pending Tickets
My Closed Tickets
Recently Viewed
No recent activity.

### Profile

#### Base Data

ID

WSU-ZPFKZ6J6

Date of Birth

21/03/1991

Account Created

08:48 12/05/2019

Role

Project Manager

#### Changable Data

Email Address

j.whitefield@wordspot.co.uk

Username

jwhitefield

First Name(s)

Jason

Last Name

Whitefield

#### Password Change

New Password

Enter new password

Confirm New Password

Enter new password again

Discard Changes

Confirm Changes

## 7.11. UI WIREFRAME – CATEGORIES

WordSpot Writing Services
https://wordspot.co.uk/

WordSpot
Logged in as Jason Whitefield
Logout

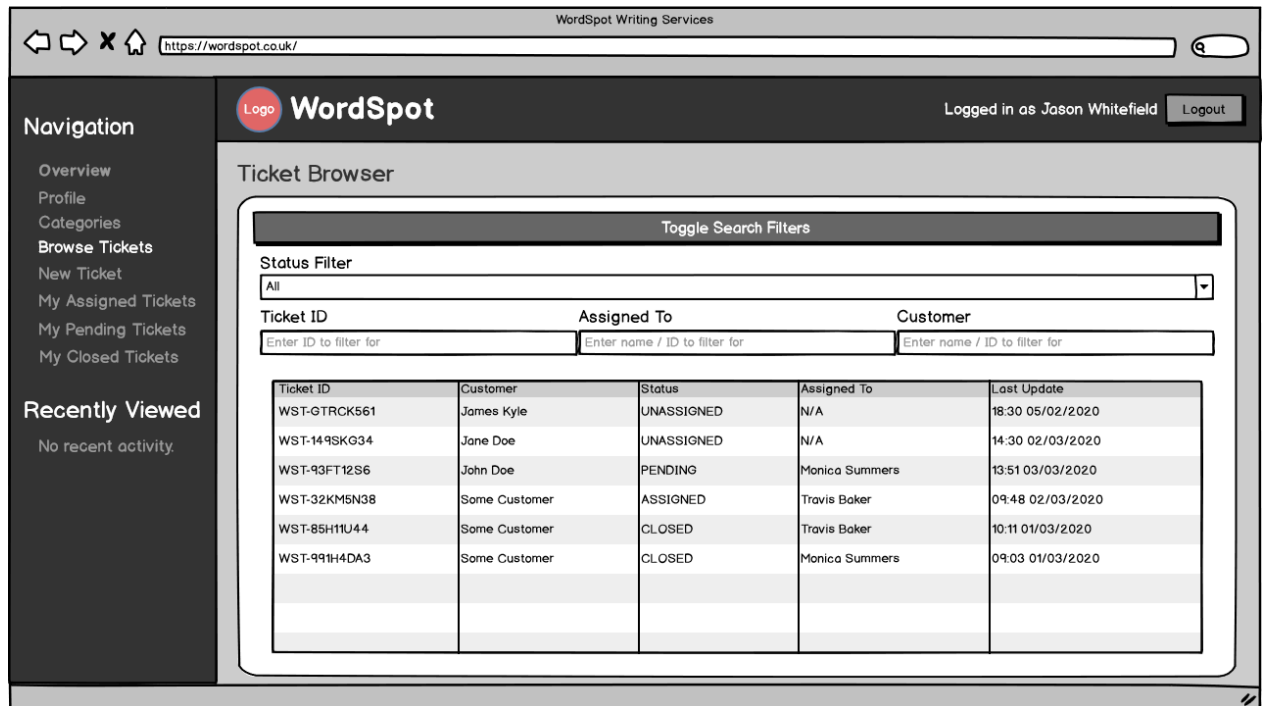
Navigation
Overview
Profile
Categories
Browse Tickets
New Ticket
My Assigned Tickets
My Pending Tickets
My Closed Tickets
Recently Viewed
No recent activity.

### Ticket Categories

ID	Name	Base Cost	Cost / 100 Words	Actions
1	Unset	£5.90	39p	Edit   Delete
2	Translation	£5.89	29p	Edit   Delete
3	Report Writing	£9.90	79p	Edit   Delete
4	Contract Preparation	£9.90	49p	Edit   Delete
5	Proofreading	£3.90	19p	Edit   Delete

New Category

## 7.12. UI WIREFRAME – TICKET BROWSER



## 7.13. UI WIREFRAME – TICKET VIEW

WordSpot Writing Services

https://wordspot.co.uk/

Q

WordSpot

Logged in as Jason Whitefield Logout

Navigation

Overview

Profile

Categories

Browse Tickets

New Ticket

My Assigned Tickets

My Pending Tickets

My Closed Tickets

Recently Viewed

WST-G98FT1K5

Profile

Base Information

Ticket ID

WST-G98FT1K5

Ticket Author

James Kyle

Customer

James Kyle

Category

Translation

Ticket Status

Assigned To

Status

Unassigned

Attachments

English Transcript.docx (~0.01 MB)

History

Update #1

Author	Assigned To	Date Updated
James Kyle	N/A	02/02/2020

Description Given

Hi,  
I would like to have the attached document translated from English to Hungarian please.  
Thank you in advance.

Update Ticket

Description

Attachment

Select File

No file selected.

Update Ticket

## 7.14. EUREKA SERVER – APPLICATION.PROPERTIES

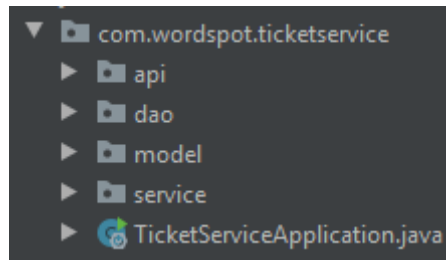
```
# Application name
spring.application.name=eureka-server

# Port for eureka server
server.port=8761

# Stop eureka server from registering itself as a client
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```



## 7.15. TICKET SERVICE – PACKAGE STRUCTURE



## 7.16. TICKET SERVICE – TICKETCONTROLLER.JAVA GET EXAMPLE

```
@GetMapping("id/{id}")
public ResponseEntity<Ticket> getTicket(@PathVariable String id) {
    log.info("Request to get ticket number " + id);
    return ticketService.getTicket(id);
}
```

## 7.17. TICKET SERVICE – TICKETSERVICE.JAVA GET EXAMPLE

```
// Get single ticket by id
public ResponseEntity<Ticket> getTicket(String id) {
    Optional<Ticket> t = ticketDao.findById(id);

    return t.map(response -> ResponseEntity.ok().body(response))
        .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
}
```

## 7.18. TICKET SERVICE – TICKET.JAVA

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "tickets")
public class Ticket {

    @Id
    @GenericGenerator(name = "id", strategy = "com.wordspot.ticket.service.model.generator.IDGenerator")
    @GeneratedValue(generator = "id")
    private String id;

    private String createdBy;

    private String customerId;

    private String assignedTo;

    @Column(columnDefinition = "TEXT")
    private String description;

    private TicketStatus status;

    @ManyToOne(fetch = FetchType.EAGER, targetEntity = TicketCategory.class, cascade = CascadeType.ALL)
    @JoinColumn(name = "CATEGORY_ID", referencedColumnName = "CATEGORY_ID")
    private TicketCategory category;

    @ElementCollection(fetch = FetchType.EAGER)
    private List<TicketHistory> history;

    // Constructor for optional matcher filtering
    public Ticket(String assignedTo, TicketStatus status, String customerId) {
        this.assignedTo = assignedTo;
        this.status = status;
        this.customerId = customerId;
    }
}
```

## 7.19. TICKET SERVICE – IDGENERATOR.JAVA

```
public class IDGenerator implements IdentifierGenerator {

    // Base36, but Vowels and potentially confusing characters removed: "Base28"
    // Possible combinations until length needs to be increased: 377,801,998,336
    private static final int LENGTH = 8;
    private static final char[] BASE28CHARS =
        "23456789BCDFGHJKLMNPQRSTVWXZ".toCharArray();
    private static Random _random = new Random();

    @Override
    public Serializable generate(SharedSessionContractImplementor session, Object o) throws HibernateException {
        String prefix = "WST-";
        Connection connection = session.connection();

        try {
            boolean valid;
            String newId = "";
            do {
                valid = true;
                var sb = new StringBuilder(LENGTH);
                for (int i = 0; i < LENGTH; i++) {
                    sb.append(BASE28CHARS[_random.nextInt(BASE28CHARS.length)]);
                }

                Statement st = connection.createStatement();
                ResultSet rs = st.executeQuery("select id from tickets");

                newId = prefix + sb.toString();

                while (rs.next()) {
                    if (rs.getString("columnIndex: 1").equals(newId)) {
                        valid = false;
                        break;
                    }
                }
            } while (!valid);
            return newId;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

## 7.20. COMMON PROJECT – JWT CONFIG

```
package jwt;

import org.springframework.beans.factory.annotation.Value;

public class JwtConfig {

    @Value("${security.jwt.uri:/auth/**}")
    private String URI;

    @Value("${security.jwt.header:Authorization}")
    private String HEADER;

    @Value("${security.jwt.prefix:Bearer }")
    private String PREFIX;

    @Value("${security.jwt.expiration:#{24*60*60}}")
    private int EXPIRATION;

    @Value("${security.jwt.secret:D63DDA162AF51ECED10FE2D7689E254471D5D8E758165F02B0651CABED0FFAF5856EFE0AA4C1EAE44B" +
        "1E6360AC283E5575986DC414E910308771AB3FB4EB0C5A}")
    private String SECRET;

    public String getURI() { return URI; }
    public String getHEADER() { return HEADER; }
    public String getPREFIX() { return PREFIX; }
    public int getEXPIRATION() { return EXPIRATION; }
    public String getSECRET() { return SECRET; }
}
```

## 7.21. AUTHORIZATION SERVICE – SECURITYCONFIG.JAVA

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Qualifier("userDetailsServiceImpl")
    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtConfig jwtConfig;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .cors().disable()

            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

            .and()

            .exceptionHandling().authenticationEntryPoint((req, res, e) -> res.sendError(HttpServletResponse.SC_UNAUTHORIZED))

            .and()

            .addFilter(new JwtUsernameAndPasswordAuthenticationFilter(authenticationManager(), jwtConfig))

            .authorizeRequests()

            .antMatchers(HttpMethod.POST, jwtConfig.getURI()).permitAll()

            .anyRequest().authenticated();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    public JwtConfig jwtConfig() { return new JwtConfig(); }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

## 7.22. GATEWAY SERVICE – APPLICATION.PROPERTIES

```
# Application name
spring.application.name=gateway-service

# Port for gateway-service
server.port=8762

# Point to eureka server
eureka.client.service-url.defaultZone=http://eureka-server:8761/eureka
eureka.instance.prefer-ip-address=true

# Zuul Configuration
zuul.ribbon.eager-load.enabled=true
ribbon.eager-load.enabled=true
ribbon.ReadTimeout=10000
zuul.ignored-services=*

# ! Mapping paths to services !

# Auth Service
zuul.routes.auth-service.path=/auth/**
zuul.routes.auth-service.service-id=auth-service
# Don't remove /auth/ from the request url
zuul.routes.auth-service.strip-prefix=false
# Exclude authorization from sensitive headers
zuul.routes.auth-service.sensitive-headers=Cookie, Set-Cookie
#-----

# Ticket Service
zuul.routes.ticket-service.path=/tickets/**
zuul.routes.ticket-service.service-id=ticket-service
zuul.routes.ticket-service.sensitive-headers=Cookie, Set-Cookie

# User Service
zuul.routes.user-service.path=/users/**
zuul.routes.user-service.service-id=user-service
zuul.routes.user-service.sensitive-headers=Cookie, Set-Cookie

# File Service
zuul.routes.file-service.path=/files/**
zuul.routes.file-service.service-id=file-service
zuul.routes.file-service.sensitive-headers=Cookie, Set-Cookie

# File size limits
spring.servlet.multipart.max-file-size=200MB
spring.servlet.multipart.max-request-size=200MB
```

## 7.23. GATEWAY SERVICE – SECURITYCONFIG.JAVA

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtConfig jwtConfig;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .cors().and()

            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

            .and()

            .exceptionHandling().authenticationEntryPoint((req, res, e) -> res.sendError(HttpServletResponse.SC_UNAUTHORIZED))

            .and()

            .addFilterAfter(new JwtTokenAuthenticationFilter(jwtConfig), UsernamePasswordAuthenticationFilter.class)

            .authorizeRequests()

            .antMatchers(HttpMethod.POST, jwtConfig.getURI()).permitAll()

            .antMatchers(HttpMethod.POST, ...antPatterns: "/users/public/**").permitAll()

            .anyRequest().authenticated();
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList("http://localhost:3000"));
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        config.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        config.setExposedHeaders(Arrays.asList("Authorization", "Content-Type"));
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( path: "/*", config);
        return source;
    }

    @Bean
    public JwtConfig jwtConfig() { return new JwtConfig(); }
}
```

```
"dependencies": {  
  "@testing-library/jest-dom": "^4.2.4",  
  "@testing-library/react": "^9.4.1",  
  "@testing-library/user-event": "^7.2.1",  
  "axios": "^0.19.2",  
  "bootstrap": "^4.4.1",  
  "jwt-simple": "^0.5.6",  
  "moment": "^2.24.0",  
  "query-string": "^6.11.0",  
  "react": "^16.12.0",  
  "react-alert": "^6.0.0",  
  "react-alert-template-basic": "^1.0.0",  
  "react-bootstrap": "^1.0.0-beta.16",  
  "react-bootstrap-typeahead": "^4.0.0-rc.2",  
  "react-dom": "^16.12.0",  
  "react-icons": "^3.9.0",  
  "react-router-dom": "^5.1.2",  
  "react-scripts": "3.4.0",  
  "react-sidebar": "^3.0.2",  
  "react-spinners": "^0.8.0",  
  "react-tooltip": "^4.0.3"  
},
```



## 7.25. DASHBOARD SERVICE – FOLDER STRUCTURE

```

✓ components
  ✓ customers
    JS ListJobs.js
    JS RequestJob.js
    JS ViewJob.js
  ✓ elements
    ✓ overview
      JS CustomerOverview.js
      JS RecentActivity.js
      JS TicketOverview.js
    ✓ profile
      JS ProfileDisplay.js
    ✓ ticket
      JS NewTicketForm.js
      JS TicketViewForm.js
  JS Header.js
  JS Loading.js
  JS Pagination.js
  JS SidebarContent.js
  JS Browse.js
  JS Categories.js
  JS Login.js
  JS MyTickets.js
  JS Overview.js
  JS Profile.js
  JS Register.js
  JS ViewTicket.js
```

## 7.26. BACKEND BUILDER DOCKERFILE

```
# Use a Maven-ready image as a base
FROM maven:3.6.3-jdk-11-slim

# Copy the common project's source files and its pom.xml
COPY common/src /common/src
COPY common/pom.xml /common/pom.xml

# Create a separate 'poms' folder for our other projects' pom files
RUN mkdir /poms

# Copy the projects' pom.xml files to the created 'poms' folder
COPY file-service/pom.xml /poms/file-service/pom.xml
COPY user-service/pom.xml /poms/user-service/pom.xml
COPY ticket-service/pom.xml /poms/ticket-service/pom.xml
COPY gateway-service/pom.xml /poms/gateway-service/pom.xml
COPY auth-service/pom.xml /poms/auth-service/pom.xml
COPY eureka-server/pom.xml /poms/eureka-server/pom.xml

# Build and install the common project. The install command also adds
# the .jar to Maven's local repository, so that it can be used by
# other projects.
RUN mvn -f /common/pom.xml clean install

# Download the dependencies of other projects one by one
RUN mvn -f /poms/file-service/pom.xml dependency:go-offline
RUN mvn -f /poms/user-service/pom.xml dependency:go-offline
RUN mvn -f /poms/ticket-service/pom.xml dependency:go-offline
RUN mvn -f /poms/gateway-service/pom.xml dependency:go-offline
RUN mvn -f /poms/auth-service/pom.xml dependency:go-offline
RUN mvn -f /poms/eureka-server/pom.xml dependency:go-offline

# Now, with all dependencies downloaded, the image is ready to be
# used as a base for building our projects in a containerized environment
```

## 7.27. BATCH\_CREATEBUILDER.BAT

```
docker build -t ws-builder -f Dockerfile-builder .
PAUSE
```

## 7.28. TICKET SERVICE – DOCKERFILE

```
# Use our own builder image as a base for building
FROM ws-builder:latest AS build

# Copy application source code and pom.xml to the image
COPY src /app/src
COPY pom.xml /app/pom.xml

# Tell Maven to build our application into a .jar file
RUN mvn -f /app/pom.xml clean package

# -----

# Set up runtime environment
FROM openjdk:11-jre-slim
VOLUME /tmp

# Expose port 8000 - the port used by Ticket Service
EXPOSE 8000

# Create app folder and log folder within it
RUN mkdir -p /app/
RUN mkdir -p /app/logs/

# Copy the built .jar file from the builder image
COPY --from=build /app/target/WS-Ticket-Service.jar /app/app.jar

# Tell the image how to start our application
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-Dspring.profiles.active=container", "-jar", "/app/app.jar"]
```

## 7.29. DASHBOARD SERVICE - DOCKERFILE

```
# Use a Node base image
FROM node:12.14.1-alpine

# Set up environment variable and /app working directory
ENV PATH /app/node_modules/.bin:$PATH
WORKDIR /app

# Copy package JSON files and the application's source code
COPY package.json package.json
COPY package-lock.json package-lock.json
COPY src src
COPY public public

# Install dependencies from the NPM repository
RUN npm install

# Expose port 3000 - the port used by Dashboard Service
EXPOSE 3000

# Tell the image how to start our application
CMD ["npm", "start"]
```

```
version: '3'
services:
  eureka-server:
    hostname: eureka-server
    container_name: eureka-server
    image: ws-eureka-server:latest
    build: ./eureka-server
    restart: unless-stopped
  gateway-service:
    container_name: gateway-service
    image: ws-gateway-service:latest
    build: ./gateway-service
    depends_on:
      - eureka-server
    links:
      - eureka-server
    ports:
      - "8762:8762"
    restart: unless-stopped
  auth-service:
    container_name: auth-service
    image: ws-auth-service:latest
    build: ./auth-service
    depends_on:
      - eureka-server
    links:
      - eureka-server
    restart: unless-stopped
  ticket-postgres:
    image: postgres:alpine
    container_name: ticket-postgres
    volumes:
      - ticket-postgres-data:/var/lib/postgresql/data
    expose:
      - "5432"
    environment:
      - POSTGRES_PASSWORD=password
      - POSTGRES_USER=postgres
      - POSTGRES_DB=ticketdb
    restart: unless-stopped
```

```

ticket-service:
  image: ws-ticket-service:latest
  build: ./ticket-service
  container_name: ticket-service
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://ticket-postgres:5432/ticketdb
    - SPRING_DATASOURCE_PASSWORD=password
  restart: unless-stopped
  depends_on:
    - eureka-server
    - ticket-postgres
  links:
    - eureka-server
    - ticket-postgres
user-postgres:
  image: postgres:alpine
  container_name: user-postgres
  volumes:
    - user-postgres-data:/var/lib/postgresql/data
  expose:
    - "5432"
  environment:
    - POSTGRES_PASSWORD=password
    - POSTGRES_USER=postgres
    - POSTGRES_DB=userdb
  restart: unless-stopped
user-service:
  image: ws-user-service:latest
  build: ./user-service
  container_name: user-service
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://user-postgres:5432/userdb
    - SPRING_DATASOURCE_PASSWORD=password
  restart: unless-stopped
  depends_on:
    - eureka-server
    - user-postgres
  links:
    - eureka-server
    - user-postgres

```

```

file-postgres:
  image: postgres:alpine
  container_name: file-postgres
  volumes:
    - file-postgres-data:/var/lib/postgresql/data
  expose:
    - "5432"
  environment:
    - POSTGRES_PASSWORD=password
    - POSTGRES_USER=postgres
    - POSTGRES_DB=filedb
  restart: unless-stopped
file-service:
  image: ws-file-service:latest
  build: ./file-service
  container_name: file-service
  volumes:
    - ./wordspotfiles:/wordspotfiles
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://file-postgres:5432/filedb
    - SPRING_DATASOURCE_PASSWORD=password
  restart: unless-stopped
  depends_on:
    - eureka-server
    - file-postgres
  links:
    - eureka-server
    - file-postgres
dashboard-service:
  image: ws-dashboard-service:latest
  build: ./dashboard-service
  container_name: dashboard-service
  ports:
    - "3000:3000"
  restart: unless-stopped
  depends_on:
    - eureka-server
    - gateway-service
    - auth-service
    - user-service
    - ticket-service
    - file-service
volumes:
  ticket-postgres-data:
  user-postgres-data:
  file-postgres-data:

```