Table of Contents

1. Introduction

1.1. Purpose of The Product Design Specification Document

This product design specification document outlines the system design and architecture of SimpleFocals e-commerce web application. This document will be used so that the stakeholders involved in development(project team and project manager), will have a solid understanding of the system architecture. The user interface section of this document will provide the stakeholders who funded the project and insight into current development, leaving open the possibilities of changes depending on the input of the client.

2. General Overview and Design Guidelines/Approach

2.1. Assumptions/Constraints/Standards

The functionalities of the system described in the **Project Specification Document** will be implemented. As outlined in that document the user will require a computing device(Computer, tablet or smartphone), it is assumed that the user's device has an Internet browser connected to the internet. The web application should be able to handle 100 concurrent users at a given time and the system should be responsive to the requested functions of the user. In the event that the system cannot complete the function available, the system shall display appropriate information as to why the function cannot be completed.

The application will be developed in a mobile first approach to design, ensuring that the user experience and features on each page of the web application remain consistent independent of

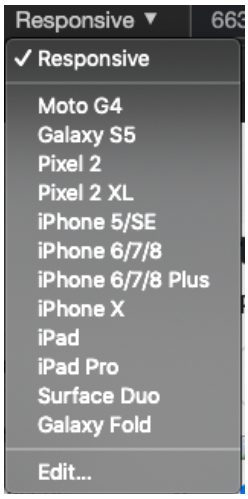

fig2.1.1: options for displaying different screen resolutions on the chrome browser

device. The web application will be developed using Chrome Browser tools where testing for different device screen sizes is possible(*fig2.1.1*).

For desktop browsers the web application will be tested on Brave and Firefox browsers using MacOS. At the completion of the project the web application will be mobile responsive and desktop ready.

3. Architecture Design

3.1. Logical View

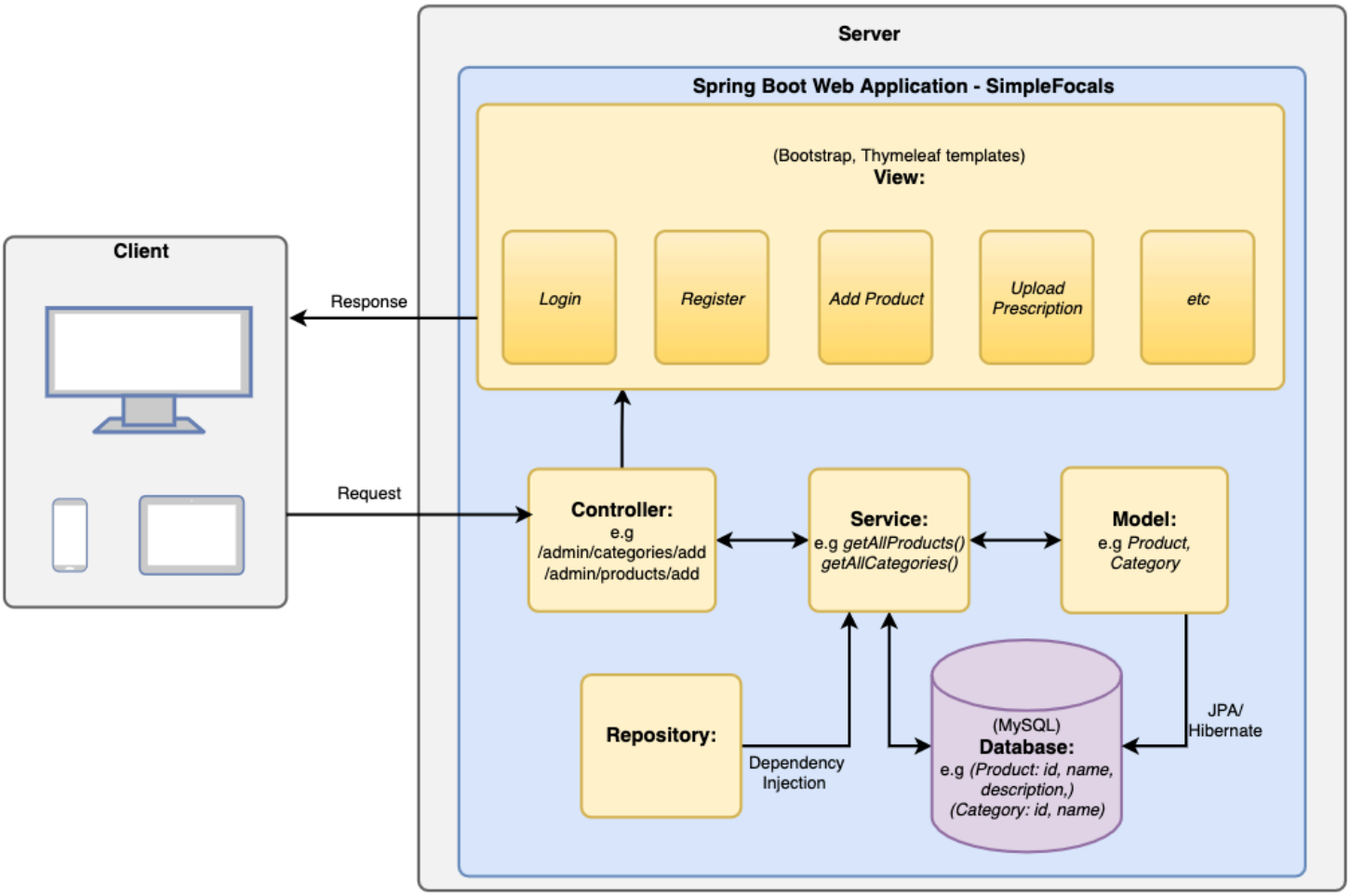

fig3.1.1: System architecture view of SimpleFocals web application

A simple depiction of SimpleFocals system architecture is outlined in in the image above(*fig3.1.1*). The web application will be developed as a spring boot application. The client in the diagram is a representation of the user's device browser. The user connects to the web application by entering a URL corresponding to the SimpleFocals hosted website. The user can navigate to different webpages in the application. When the user requests a specific resource from the web application, it sends a request to the server, the application's controller then deals with that request. The controller communicates with the service layer to handle the application logic before returning the result back to the controller and the view. This communication flow is hidden from the user.

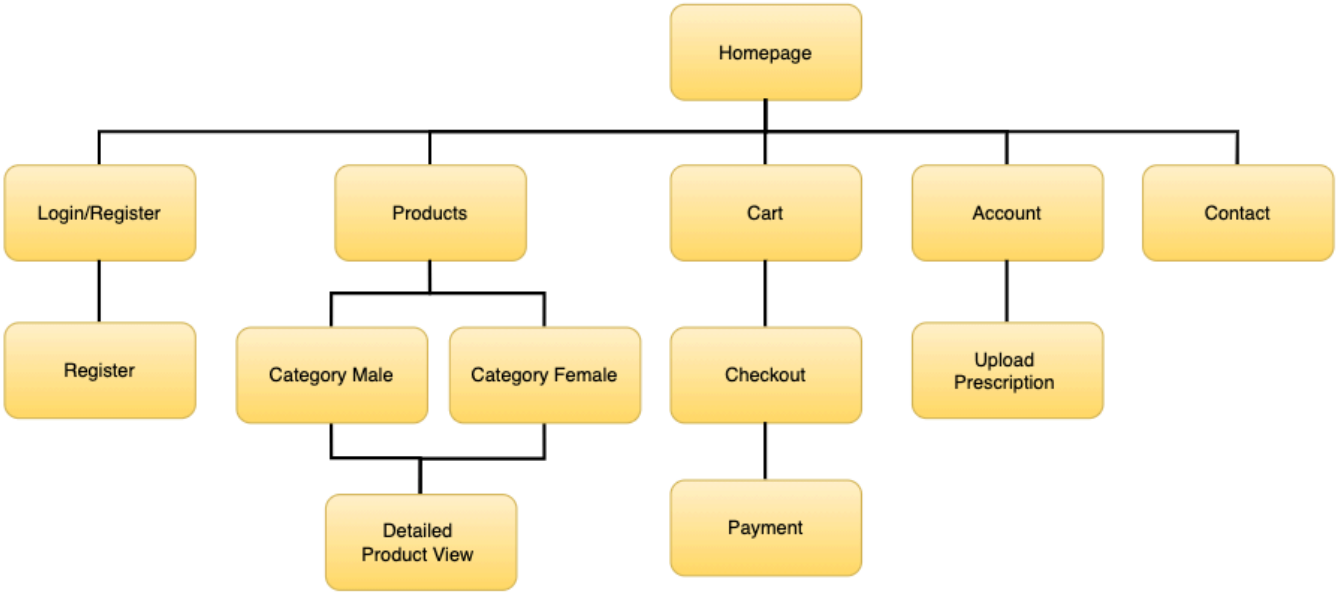

fig3.1.2: Simple site-view for the customer of SimpleFocals

A simple depiction of SimpleFocals site-view for a customer is outlined in in the image above(*fig3.1.2*). The homepage is the entry point of the web application but the customer can access links to the login/register, products, cart, account and contacts from the sticky navigation bar at the top of all of the SimpleFocals pages. If the customer wishes to access their account, registration and logging into the system is required. When logged into the system the user can access their account page to upload a prescription, access their cart page, go to checkout and also payment.

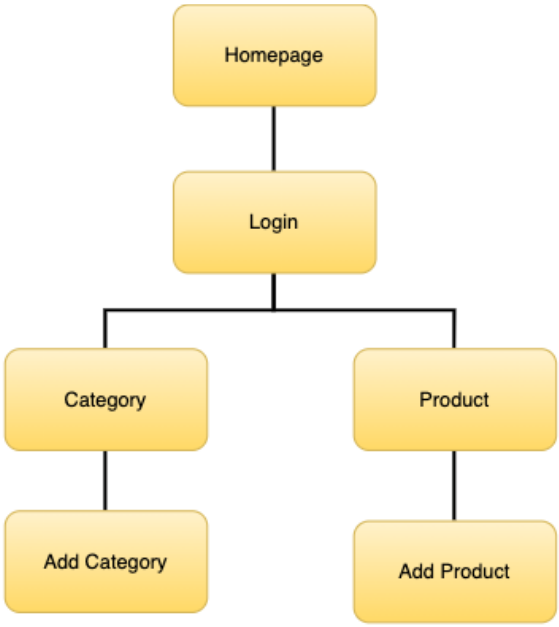

fig3.1.3: Simple site-view for the administrator of SimpleFocals

A simple depiction of SimpleFocals site-view for the administrator is outlined in in the image above(*fig3.1.3*). Once logged into the system, the administrator has access to the content management system pages. A category page is accessible where new categories can be added to SimpleFocals products. A product page is accessible where new products can be added within a specific category.

3.2. Hardware Architecture

The web application will be deployed on Amazon Web Services using Elastic Beanstalk.

"*AWS Elastic Beanstalk is the fastest and simplest way to get web applications up and running on AWS. Developers simply upload their application code and the service automatically handles all the details such as resource provisioning, load balancing, auto-scaling, and monitoring.*" (Amazon, 2021)

AWS Elastic Beanstalk service was chosen as the best option because the spring boot web application can be deployed quickly and without much configuration. Subscribing to AWS Free Tier option the web application can be hosted for 12 months free of charge. Using AWS cloud infrastructure should satisfy all the responsive requirements outlined in the **Specification Requirements Document**.

3.3. Software Architecture

The overall system architecture is displayed in the image(*fig3.1.1*), the technologies that will be used to build this application are listed below.

Java: High-level, class based object orientated programming language.

Spring Boot: Spring Framework is a “comprehensive infrastructure support for developing Java applications.”(Baeldung, 2021) Spring Boot is an extension of Spring Framework where the developers don’t have to spend time dealing with boilerplate configurations but can concentrate on coding their applications for faster development.

Hibernate: JPA (Java Persistence API) is the standard approach for Object Relational Mapping(ORM), ORM is where Java objects are mapped to data in a relational database. Hibernate is one of the most common implementations of JPA.

MySQL DB: MySQL DB is free open-source relational database software that is developed by Oracle.

Bootstrap: Bootstrap is a CSS Framework that can be used in quickly creating responsive mobile first websites.

Stripe API: Stripe API is the application programming interface provided by trusted third party Stripe in order to provide secure payment processing.

Spring-Boot-Starter: Spring boot starter helps reduce the time the developer uses for searching and adding dependancies. Spring-boot-starter groups the require dependancies together in a single package.

-Web: “Is used for building web applications, including RESTful

applications using Spring MVC. It uses Tomcat as the default embedded container."(Javapoint, 2021)

-Data JPA: Data JPA is a data access abstraction that is used to reduce boiler plate code when implementing data access layer for persistence stores. "It is used for Spring Data JPA with Hibernate."(Javapoint, 2021)

-Validation: "Is used for Java Bean Validation with Hibernate Validator." (Javapoint, 2021)

-Security: Spring Security is an access-control and authentication framework that has become the standard for securing Spring applications.

-Thymeleaf: Thymeleaf is a HTML5/XHTML/XML server side java templating engine that works in both web(online) and non-web(off-line) environments. "It is used to build MVC web applications using Thymeleaf views." (Javapoint, 2021)

3.4. Security Architecture

The web application will have secure login for users. Users can be logged in using their email and password combination. A users credentials shall be verified by Spring Security in the application. Spring Security shall be used for role based authorisation. Registered customers shall only be able to access a customer's view of the application including all the functionalities that are available to a registered customer. The administrator will have access to the CMS view of the application and have access to all the functionalities of the CMS. The benefits to using Spring Security is that is can support both authorisation and authentication, it also protects against malicious clients that want to use attacks such as clickjacking, session fixation and cross site request forgery. Spring Security easily integrates into Spring MVC applications.

3.5. Communications Architecture

The basis for SimpleFocals system architecture in the image(*fig3.1.1*), is the model-view-controller(MVC) architecture. Web applications developed using MVC have the benefits of having the presentation, the persistence and control layers loosely coupled and well organised. When the web application is started on the server, a servlet container is created to handle the incoming requests and to serve the responses. In Spring MVC applications, such as SimpleFocals, there is a set flow in the communication structure from the client making a request to them receiving a response. The image(fig3.5.1) outlines the flow of communication in the Spring application.

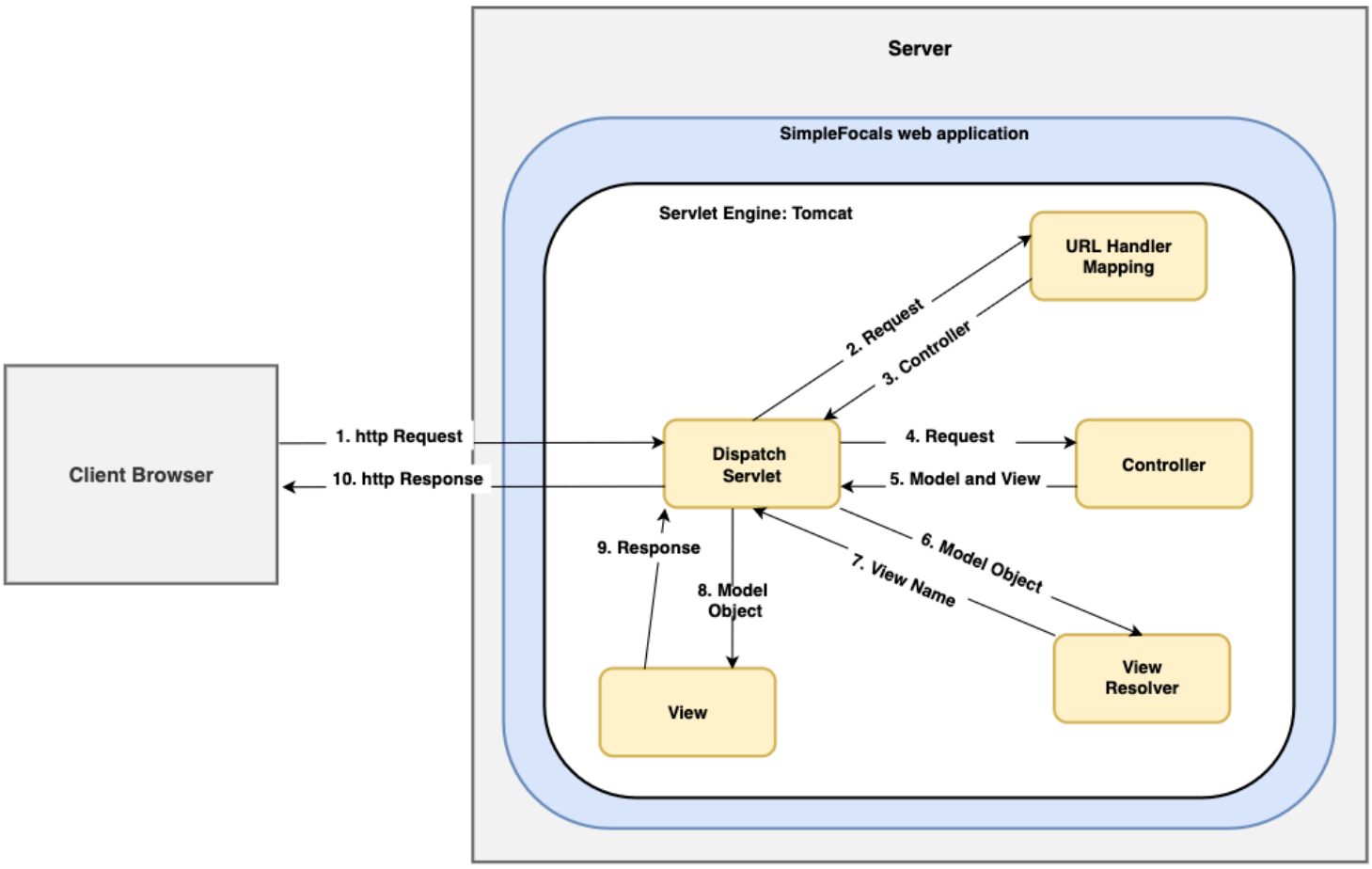

fig3.5.1: How the application handles web requests

1. The client sends a http request specifying SimpleFocals web URL in the browser.
2. The request is received by the Dispatcher Servlet. The Dispatcher Servlet acts as a front controller for the application.
3. The Dispatcher Servlet makes use of Spring's URL Mapping Handler to find the specific controller class that the request should be routed for processing. e.g "/admin/products", would route the request to an admin controller class, where they could access a products page.
4. The Dispatcher Servlet forwards the request to the specific controller class the URL Handler Mapping has pointed at. e.g admin controller class
5. The controller class is where implementations of different methods for http requests reside. Those methods include GET and POST methods for http requests. The controller class will call methods in the service layer to handle the logic(*fig3.1.1*). A Model and View object will be passed back to Dispatcher Servlet.
6. The Dispatcher Servlet now sends the Model object to the View Resolver.
7. The View Resolver now returns a view name for the Model object to the Dispatcher Servlet.
8. The Dispatcher Servlet now can route the Model object to the specific view.
9. The Model object containing the data which needs to be displayed in the view can now be rendered. (In SimpleFocals web application we use Thymeleaf templating engine to embed objects into HTML5 code).
10. The view can now be returned to the clients browser. Passing the view as http response for rendering in the clients browser completes the communication flow.

3.6. Performance

When interacting with the application the client should be able to access all the functions of SimpleFocals e-commerce website without any noticeable delays in performance. The application when deployed to AWS cloud will be able to handle the Performance/Response time requirement outlined in **Requirements Specification Document.**

The web application when deployed should handle 100 users at a given time. The following conditions are expected to be satisfied:

- Login shall not take longer than 3 seconds.
- Loading of pages shall not take longer than 5 seconds

4. System Design

4.1. Use Cases

The use cases for SimpleFocals web application can be found in the **Requirements Specification document.**

4.2. Database Design

The Spring boot application is using a MySQL database to store data. MySQL is a relational database that is deployable for cloud applications. The Spring boot application is using Spring Data JPA and Hibernate for Object relational mapping (ORM) to persist data in the MySQL database. JPA is Java persistence API that defines a specification for reading, persisting and managing data from objects in Java to relations in a database. Hibernate is an implementation of JPA. Spring Data JPA is Spring's layer on top of Java that makes it easier to implement JPA. An example of how it makes things easier would be: if one was to implement Spring and Hibernate without Spring Data JPA, a DAO interface and implementation is needed where CRUD operations are written using Hibernate's SessionFactory. If we take an example, we are writing a DAO for Product_1 class and a Product_2 class is required. A DAO with similar CRUD operations would have to be written for Product_2 class. There is a lot of repeating boilerplate code in this situation.

Spring Data JPA allows the DAO interface to be defined by extended its repositories using JpaRepository this provides a DAO implementation during runtime. A DAO implementation isn't required anymore using Spring Data JPA.

```java
import lombok.Data;
import javax.persistence.*;

//annotation for setting as a table
//also using lombok to reduce boiler plate code getters,setters constructors
@Entity
@Data
public class Category {

    //auto generate primary key
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "category_id")
    private int id;
    private String name;
}
```

fig4.2.1: Category class

Let's take a look at simple implementation in the SimpleFocals project of a Category class using Spring Data JPA and Hibernate. In the image above(*fig4.2.1*), a Category class in Java is defined. The "@Entity" annotation outlines that this class will be a relation in the database. The primary key will be unique and incrementally generated by specifying the "@GeneratedValue" annotation. There are two columns in this table, the primary key "category_id" of type integer and "name" that is of type String. A table representation of the given class is shown in the image(*fig4.2.2*) with some data.

SELECT * FROM CATEGORY;

CATEGORY_ID	NAME
8	Mens Glasses
9	Womens Glasses
41	Children's Glasses

(3 rows, 6 ms)

fig4.2.2:The table representation of the class Category

In the image below(*fig4.2.3*), a CategoryRepository Interface is defined that extends JpaRepository. This allows use of methods to implement CRUD functionality in the Service layer.

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface CategoryRepository extends JpaRepository<Category, Integer> {
}
```

fig4.2.3: CategoryRepository Interface

In the image below(*fig4.2.4*), a CategoryService class is defined. Here the defined CategoryRepository object has many methods to facilitate CRUD functionality.

```java
//the Service is where the business logic should go

@Service
public class CategoryService {
    @Autowired
    CategoryRepository categoryRepository;

    public List<Category> getAllCategory(){
        return categoryRepository.findAll();
    }

    public void addCategory(Category category){
        categoryRepository.save(category);
    }

    public void removeCategoryById(int id){
        categoryRepository.deleteById(id);
    }

    public Optional<Category> getCategoryById(int id){
        return categoryRepository.findById(id);
    }
}
```

fig4.2.4: CategoryService Class

4.3. Data Conversions

4.3.1. Back-End

Using Spring Data JPA and Hibernate, data class objects in Java are mapped to data relational database tables in MySQL. With such mappings the system can perform CRUD operations on data in the database without writing any SQL.

4.3.2 Front-End

When some dynamic function is requested from the front-end, the client sends the server JSON objects(*fig4.3.1*). Once sent to the server the application parses the information into java objects where it can be processed and stored in the database.

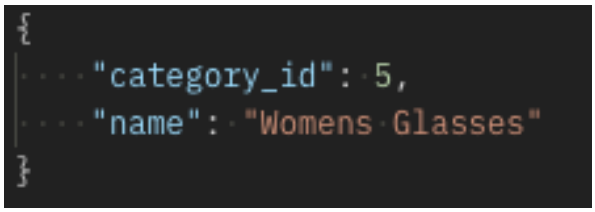

fig4.3.1: JSON object

4.4. Application Programming Interfaces

The application will be integrating Stripe API for payment processing. Customers will be able to pay for SimpleFocals products by credit or debit card. Stripe ensures all payments on orders shall be securely processed on their servers without our web application storing any of the customers sensitive card information. The steps for implementation will require the application to:

1. Add Stripe dependancy to the project.
2. Build the checkout page loading the Stripe.js script. This script is used to access the API keys on the client and the card input field on the form.
3. Complete the payment on the client. The Stripe handles the API response.
4. Test the API using test card numbers supplied by Stripe.

4.5. User Interface Design

The user interface will be developed using frontend technologies HTML, CSS and Javascript. In order to speed up development time, the project shall use Bootstrap to design the pages of the website. Bootstrap is a CSS grid based framework that allows rapid development using Bootstrap's predefined classes that allow for mobile responsive development of websites. The dynamic elements of the user interface will use thymeleaf's templating engine to update information on the webpages. Examples of the wireframes developed for this project are detailed in Interface requirement in the **Requirements Specification Document.**

4.6. Performance

There are a number of reasons why Java is a good choice for backend development.

4.6.1. Why use java for backend development?

- Java is an object orientated high level language that uses a virtual machine to compile programs, this allows it the flexibly to run on many operating systems and platforms.
- Java is scalable and robust, it has the benefits of automatic memory management and garbage collection which can speed up web application development.
- Java has lot of support for open source libraries such XML and JSON parsing libraries.
- Java supports frameworks and tools for quickly developing back-end applications. Spring, Apache and Hibernate being examples of some.

Acronyms, Abbreviations and Terms:

AWS: Amazon Web Services: is the cloud computing service that Amazon provide.

ORM: Object–relational mapping in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

JPA: The Java Persistence API (JPA) is a Java specification for accessing, persisting, and managing data between Java objects / classes and a relational database

API: Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

CMS: A Content Management System is a computer software used to manage the creation and modification of digital content.

MVC: Model–view–controller is a software design pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements.

URL: Uniform Resource Locator is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

GET: GET method requests a representation of the specified resource. Requests using GET should only be used to request data.

POST: A POST request is typically sent via an HTML form and results in a change on the server.

DAO: DAO is an abbreviation for Data Access Object, so it should encapsulate the logic for retrieving, saving and updating data in your data storage (a database, a file-system)

CRUD: Create, Read, Update, and Delete are the four basic operations of persistent storage.

JSON: JSON is an open standard file format and data interchange format that uses

human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.

XML: Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

CSS: Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

HTML: The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser.

MacOS: Linux based 3rd party Operating System that is developed by Apple and native to Mac computers.

References:

Amazon, 2021. *AWS Elastic Beanstalk Features. [online]*

Available at: https://aws.amazon.com/elasticbeanstalk/details/ [Accessed 1 July 2021]

Baeldung, 2021. *A Comparison Between Spring and Spring Boot. [online]*

Available at: https://www.baeldung.com/spring-vs-spring-boot [Accessed 1 July 2021]

Javapoint, 2021. *Spring Boot Starters. [online]*

Available at: https://www.javatpoint.com/spring-boot-starters [Accessed 8 July 2021]