Software Technical Requirements Document

For: Doompause Emporium Ecommerce Application

Daniel Rumfelt, Ryan Coon, and Christa Moncrief

Grand Canyon University: CST-326

09/24/2023

Document Version 1.3

09.24.2023

**AUTHORS**

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Department** |
| Daniel Rumfelt | Backend and IoT Site Security | Software Engineering |
| Christa Moncrief | Frontend, GUI and Database Development | Web Design, GUI |
| Ryan Coon | Backend, Frontend, and Database Development | Software Engineering |

**DOCUMENT HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Document Revision Description** | **Document Author** |
| 09/19/23 | 1.0 | Creation of Document | Daniel Rumfelt |
| 09/19/23 | 1.1 | Added file structure for app.  Performance expectations. | Daniel Rumfelt |
| 09/23/23 | 1.2 | Added Technical Requirements | Daniel Rumfelt |
| 09/24/23 | 1.3 | Update and proofread | Ryan Coon |
|  |  |  |  |

**APPROVALS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Approval Date** | **Approved Version** | **Approver Role** | **Approver** |
| 09/24/23 | 1.3 | Backend, Frontend, and Database Development | Ryan Coon |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc134441706)

[1.1. Purpose of the Document 4](#_Toc134441707)

[1.2. Related Documents 5](#_Toc134441708)

[1.3. Terms/Acronyms and Definitions 5](#_Toc134441709)

[2. Risks and Assumptions 6](#_Toc134441710)

[3. Architecture 7](#_Toc134441711)

[3.1. System Architecture 7](#_Toc134441712)

[3.2. Application Architecture 11](#_Toc134441713)

[4. Requirements 12](#_Toc134441714)

[4.1. Hardware Requirements 13](#_Toc134441715)

[4.2. Software Requirements 14](#_Toc134441716)

[4.3. Documentation/Comments 15](#_Toc134441717)

[4.4. Error Checking/Handling 16](#_Toc134441718)

[4.5. Software Versioning and Storage 16](#_Toc134441719)

[4.6. Data Requirements 17](#_Toc134441720)

[4.7. File Structure 17](#_Toc134441721)

[4.8. Schema Objects 19](#_Toc134441722)

[5. Integration Requirements 19](#_Toc134441723)

[6. Performance 19](#_Toc134441724)

[7. Software Quality 20](#_Toc134441725)

[8. Technical Requirements 21](#_Toc134441726)

[9. Open Issues 22](#_Toc134441727)

[10. Appendix 22](#_Toc134441728)

# Introduction

This document will address the **development of a web application for online sales**. The company, with its rich history in the trade of medieval weapons, is seeking to expand its reach to customers globally through an online platform.

**Background Information:** The company has been in business for hundreds of years, offering medieval weapons for both defensive and offensive combat. Over the years, it has built a reputation for quality and authenticity. However, with changing times and the rise of digital commerce, the company has found it increasingly difficult to reach customers.

The proposed web application will allow users to view items, read weapon descriptions, and see the prices of the items. It will also offer an online shopping cart where customers can add, remove, and edit the quantity of products. This digital transformation aims to make shopping more convenient for customers and expand the company’s customer base.

By addressing these needs, the company hopes to continue its tradition of excellence while adapting to the modern marketplace.

## Purpose of the Document

A Software Technical Requirements Document, also known as a technical specification document, is a comprehensive guide that defines the requirements and specifications for a particular software product or system. It functions as a guide for developers and other participants in the development process. Here are its primary functions:

1. Define Requirements: This section defines the software's functional and nonfunctional requirements in detail. This includes system characteristics, performance criteria, security measures, and other particular functionalities.
2. It provides developers with a distinct comprehension of what must be built and how it should operate, thereby facilitating the planning and execution of the development process.
3. It functions as a point of reference for all project stakeholders (developers, project managers, testers, etc.), ensuring that everyone has a consistent comprehension of the project.
4. It assists evaluators in confirming that the software fulfills the specified requirements.
5. It provides important data for future updates, maintenance, and troubleshooting.

This document is intended for software developers, project supervisors, testers, system analysts, and occasionally even end-users. Depending on their position within the project, each audience utilizes the document differently. Developers use it to comprehend what to construct, while testers use it to ensure the software meets the requirements.

## Related Documents

1. **Functional Specification Document (FSD)**: This document provides detailed descriptions of the system’s functionalities. It describes what the system should do and how it should behave from the user’s perspective. It would detail features like product display, shopping cart, user-friendly interface, and secure payment gateway function in the web application. [Link to Document](https://1drv.ms/w/s!Ag2O9AWDF9dPk4Q1SenFi9_mr1tViQ?e=zhdnIg) This document includes:
   1. Project Scope Document: This document outlines the project’s objectives, deliverables, milestones, and timeline. It would detail the aim of developing a web application for selling medieval weapons to online customers.
   2. Design Document: This document provides a comprehensive architectural layout of the application. It includes wireframes, database design, workflow diagrams, and other design-related information.
   3. Test Plan: This document outlines the testing strategies, test cases, testing schedule, and responsible persons for each test case.
   4. User Manual: This document provides instructions on how to use the system. It is usually written in non-technical language for end-users.
   5. Maintenance Manual: This document provides detailed information about how to troubleshoot common issues and perform routine maintenance tasks on the system.

## Terms/Acronyms and Definitions

|  |  |  |
| --- | --- | --- |
| **Term/Acronym** | **Definition** | **Description** |
| FSD | Functional Spec Document | A Functional Specification Document (FSD) is a formal document that details all features and functionalities of a software system that is to be developed. It serves as a guide for the development team and a reference for stakeholders to understand what the system will do. |
| TSD | Software Technical Requirements Document, also known as a technical specification document. | A comprehensive guide that defines the requirements and specifications for a particular software product or system. It functions as a guide for developers and other participants in the development process. |

# Risks and Assumptions

Assumptions, risks, and constraints are an integral part of the technical design process. Here are some that could affect the technical design of the system:

**Assumed Factors**:

1. **Reliable Third-Party Services**: It’s assumed that third-party services such as the payment gateway are reliable and secure.
2. **Stable Internet Connection**: The system assumes users have a stable internet connection for seamless browsing and transactions.
3. **User Tech Savviness**: It’s assumed that users have a basic understanding of how to navigate web applications.

**Identified Risks**:

1. **Third-Party Service Failure**: There’s a risk that third-party services might experience downtime or other issues, affecting the application’s functionality.
2. **Security Breaches**: Despite using secure payment gateways, there’s always a risk of security breaches and data theft.
3. **Technical Debt**: Rapid development to meet deadlines might lead to shortcuts and workarounds, accumulating technical debt.

**Constraints**:

1. **Budget**: The budget could limit the number of features that can be implemented or the technologies that can be used.
2. **Time**: The project timeline might not allow for extensive testing or the implementation of all desired features.

**Potential Solutions**:

1. **Backup Third-Party Services**: Have backup third-party services in case the primary ones fail.
2. **Regular Security Audits**: Conduct regular security audits to identify and fix potential security vulnerabilities.
3. **Prioritize Features**: Prioritize features based on their impact on user experience and business value to manage time and budget constraints effectively.

The actual factors will depend on the specific circumstances of the project and should be identified through thorough analysis and discussion with all stakeholders.

# Architecture

The web application for Doompause Emporium will have a two-tier architecture:

1. Frontend (Client Tier): The frontend of the application will be built using JavaScript, HTML, and CSS. These technologies are standard for web development and will ensure a user-friendly interface and seamless user experience. The front end will be responsible for presenting data to users and handling user interactions.
2. Backend (Server Tier): The backend will be powered by Spring Boot and Spring Boot Security. Spring Boot simplifies the setup of standalone Spring applications, while Spring Boot Security will handle secure transactions. MySQL will be used for database management, ensuring efficient data storage and retrieval. The backend will handle business logic, data storage, security, and communication with the frontend.

The purpose of this web application is to expand the company’s reach to customers globally, adapting to the changing shopping habits of customers in the digital age. The benefits include increased reach, improved customer experience, and efficient operations. The main objectives are to develop a user-friendly web application with an intuitive interface, ensure secure online transactions with Spring Boot Security, efficiently manage data with MySQL, and launch the web application within the specified timeline and budget.

## System Architecture

A diagram of a flowchart

Description automatically generated

A diagram of a company

Description automatically generated

A diagram of a company

Description automatically generated

A diagram of a product

Description automatically generated

A diagram of a product

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a company

Description automatically generated

## Application Architecture

The design and construction of the software application for Doompause Emporium will involve several patterns and techniques:

1. **Model-View-Controller (MVC)**: This design pattern separates the application into three interconnected components. The Model represents the application data and business rules, the View is the user interface and presentation of data, and the Controller handles the input from users. This pattern is commonly used in web applications and will help in maintaining a clean separation of concerns among the components.
2. **Responsive Web Design**: This technique ensures that the web application’s user interface adjusts dynamically, considering the characteristics of the device used (desktop, mobile, tablet, etc.). It enhances user experience as it provides optimal viewing across a wide range of devices.
3. **Object-Relational Mapping (ORM)**: ORM is a technique that lets you interact with your database, like MySQL in this case, in an object-oriented manner. With ORM tools, you can manipulate data as objects, making it easier to create, retrieve, update, and delete records without having to write SQL queries.
4. **Spring Security**: Spring Security is a powerful and highly customizable authentication and access-control framework. It is used to secure Spring-based applications and provides protection against attacks like session fixation, clickjacking, cross-site request forgery, etc.
5. **RESTful API Design**: Representational State Transfer (REST) APIs provide a way for the client-side of an application to communicate with the server-side. This architectural style allows different components of the application to communicate over HTTP in a way that mimics the behavior of web browsers and servers.
6. **Agile Development**: Agile development is a project management methodology that involves continuous iteration of development and testing throughout the software development lifecycle of the project. This approach promotes adaptive planning, evolutionary development, early delivery, and continuous improvement.

# Requirements

The hardware and software requirements for the Doompause Emporium’s web application are as follows:

**Hardware Requirements**:

1. **Server**: A robust server with sufficient processing power and memory is required to host the web application. The exact specifications would depend on the expected load and traffic.
2. **Storage**: Adequate storage space is necessary to store the application data, user information, product images, and other related data.
3. **Network**: A reliable and high-speed internet connection is required for smooth operation of the web application.

**Software Requirements**:

1. **Operating System**: A stable operating system like Linux, Windows, or macOS to host the server.
2. **Backend**: Spring Boot and Spring Boot Security for robust, secure, and scalable server-side operations.
3. **Database Management System**: MySQL for efficient data storage and retrieval.
4. **Frontend**: JavaScript, HTML, and CSS for creating a user-friendly interface and seamless user experience.
5. **Web Server**: A web server like Apache or Nginx to serve the web application.
6. **Browser Compatibility**: The application should be compatible with all major browsers like Chrome, Firefox, Safari, and Edge.

## Hardware Requirements

The hardware requirements for the Doompause Emporium’s web application are as follows:

**Client-Side Requirements**:

* **Operating System**: The application should be compatible with all major operating systems like Windows, macOS, and Linux.
* **Browser**: The application should be compatible with all major browsers like Chrome, Firefox, Safari, and Edge.
* **Hardware**: The minimum hardware requirements for the client-side would be a computer with at least 4GB RAM and a stable internet connection. However, these requirements can vary based on the specific tasks the user is performing.

**Server-Side Requirements**:

* **Operating System**: A stable operating system like Linux is recommended for hosting the server due to its robustness and compatibility with various software.
* **Processor**: A multi-core processor (like Intel i5 or better) is recommended for efficient processing of multiple requests.
* **Memory**: At least 16GB RAM is recommended for smooth operation of the server-side applications.
* **Storage**: Adequate storage space (SSD preferred) is necessary to store the application data, user information, product images, and other related data. The exact amount would depend on the size of the data.
* **Network**: A high-speed internet connection with high uptime is required for hosting the web application.

**Hosting Environment**:

* The hosting environment should support Spring Boot and MySQL. It should also provide sufficient computational resources to handle expected traffic and data volume. Options could include cloud platforms like AWS, Google Cloud, Azure, or a dedicated server depending on the company’s needs and budget.

## Software Requirements

The software requirements for the Doompause Emporium’s web application are as follows:

**Design and Development Software**:

1. **Integrated Development Environment (IDE)**: An IDE like Visual Studio Code, Sublime Text, or Atom for writing and managing code.
2. **Version Control System**: Git for version control to track changes and coordinate work between different team members.
3. **Frontend Development**: JavaScript, HTML, and CSS for creating the user interface. You might also use libraries or frameworks like React or AngularJS to simplify the development process.
4. **Backend Development**: Spring Boot and Spring Boot Security for server-side operations.
5. **Database Management System**: MySQL for managing application data.

**Server-Side Software Requirements**:

1. **Operating System**: A stable operating system like Linux is recommended for hosting the server.
2. **Web Server**: A web server like Apache or Nginx to serve the web application.
3. **Backend Framework**: Spring Boot and Spring Boot Security for handling server-side operations.
4. **Database**: MySQL for data storage and retrieval.

**Client-Side Software Requirements**:

1. **Web Browser**: The application should be compatible with all major web browsers like Chrome, Firefox, Safari, and Edge.

**Hosting Platform Software Requirements**:

1. **Operating System**: The hosting platform should support the chosen operating system (like Linux).
2. **Web Server**: The hosting platform should support the chosen web server (like Apache or Nginx).
3. **Backend Framework**: The hosting platform should support Spring Boot.
4. **Database**: The hosting platform should support MySQL.

## Documentation/Comments

The Documentation/Comments requirements for the Doompause Emporium’s web application are as follows:

1. **UI/UX Designers**: They should create detailed design documents that include wireframes, user flows, and mockups. These documents should explain the design decisions and how the design meets the users’ needs and business goals.
2. **Frontend Developers**: They should comment on every complex function in their code. Comments should explain what the function does, its inputs and outputs, and any side effects. They should also document any dependencies, such as external libraries or APIs.
3. **Backend Developers**: Similar to frontend developers, they should comment on their code, especially for complex algorithms and database operations. They should also document the structure of the database, relationships between tables, stored procedures, triggers, etc.
4. **QA Testers**: They should document test cases, expected results, and actual results. Any bugs found should be reported in detail so that developers can understand the issue and fix it.

.

## Error Checking/Handling

The Error Checking/Handling requirements for the Doompause Emporium’s web application are as follows:

1. **Data Format Errors**: These occur when the data is being sent to a database, or another system is in the wrong format. The strategy to handle this scenario includes checking and validating the data before sending it. If the data is not in the correct format, try to convert it. If an error or exception still occurs, notify the user of the error.
2. **Network Errors**: These occur when there’s a problem with the network connection, such as timeouts or connection failures. The strategy to handle this scenario includes implementing retry logic and using circuit breakers to prevent system failure.
3. **System Outages**: These occur when an external system or service is unavailable. The strategy to handle this scenario includes implementing fallback methods that provide default behavior when the external system is down.
4. **Concurrency Errors**: These occur when multiple processes try to access and manipulate shared data simultaneously. The strategy to handle this scenario includes using locks, semaphores, or other synchronization techniques.
5. **Resource Limit Errors**: These occur when the system runs out of resources, such as memory or disk space. The strategy to handle this scenario includes monitoring resource usage and optimizing code to use resources efficiently.

Lastly, detailed error logs should be maintained, including information like the nature of the exception, error ID, root cause of the error, timestamp, etc. These logs can be invaluable for debugging and preventing future errors.

.

## Software Versioning and Storage

The software will be hosted for all team members to collaborate on GitHub. The repository is located at: <https://github.com/AtlasWriter/Doompause_Eporium_WebApp>

The team will make use of forking, cloning, pushing, and pull requests to update the software over time.

## Data Requirements

A screenshot of a computer error

Description automatically generated

## File Structure

The file structure for the web application for Doompause can be organized as follows:

* **Root Directory**: *Doompause-Emporium:* This is the main folder that contains all other files and folders related to the project.

**src**: *Doompause-Emporium/src/main/java*. This folder contains all the source code files.

**main**: com.ecommerce

*/Doompause-Emporium/src/main/java/com/ecommerce/DoompauseEmporiumApplication.java.* This folder contains the main application code.

* + - * **java**: This folder contains all the Java source files (for a Spring Boot application).
      * com.ecommerce.dao

/Doompause-Emporium/src/main/java/com/ecommerce/dao/ProductCategoryRepository.java

/Doompause-Emporium/src/main/java/com/ecommerce/dao/ProductRepository.java

* + - * com.ecommerce.entity

/Doompause-Emporium/src/main/java/com/ecommerce/entity/Product.java

/Doompause-Emporium/src/main/java/com/ecommerce/entity/ProductCategory.java

* + - * **resources**: This folder contains resources like templates, static assets, property files, etc.
        + /Doompause-Emporium/src/main/resources
        + /Doompause Emporium/src/main/resources/application.properties
        + **static**: This folder contains static files like CSS, JavaScript, images, etc.
        + **templates**: This folder contains template files (like Thymeleaf or JSP files in a Spring Boot application).
      * **test**: This folder contains all the test code and resources. /Doompause-Emporium/src/test/java
        + com.ecommerce
  + **target**: /Doompause-Emporium/target/generated-sources/annotations This folder is generated when you build your project. It contains all the compiled .class files, .jar file, etc.
  + **pom.xml**: /Doompause-Emporium/target/generated-test-sources/test-annotations This is the Project Object Model (POM) file for Maven-based projects. It includes project information and configuration details including dependencies, plugins, goals, etc.
  + **.gitignore**: /Doompause-Emporium This file specifies which files and directories to ignore in git version control.
  + **README.md**: /Doompause-Emporium This file provides important information about the project, including how to set up and run the application.

## Schema Objects

A screenshot of a computer

Description automatically generated

# Integration Requirements

The client makes an HTTP request (GET, PUT, POST, DELETE, etc.). The HTTP request is forwarded to the Controller. The controller maps the request, processes the handles, and calls the server logic. The business logic is performed in the Service layer.

To run a Spring Boot application, you would need a suitable runtime environment that includes JDK 17 or above. You would also need a suitable web server such as Tomcat or Jetty. For database interactions, you would need a database management system like MySQL or PostgreSQL.

Spring Boot REST APIs is required to render formats such as JSON or XML. RESTful APIs by creating controllers and routes and handling HTTP requests. Our software must be integrated using annotations like @RestController, @RequestMapping, @GetMapping, @PostMapping, etc.

# Performance

The performance requirements for the Doompause Emporium’s web application

**Response Time**: The application should respond quickly to user interactions. For example, page load times should ideally be under 2 seconds.

1. **Scalability**: The application should be able to handle an increasing number of users and data volume without a significant impact on performance. This is particularly important as the company aims to reach a global audience.
2. **Availability**: The application should be available for use as much as possible, aiming for 99.9% uptime or higher.
3. **Concurrency**: The application should be able to handle multiple users simultaneously without any degradation in performance.
4. **Data Integrity**: All transactions should be processed reliably and accurately, ensuring data integrity.
5. **Security**: The application should effectively resist malicious attacks and ensure secure transactions, without compromising on performance.
6. **Resource Utilization**: The application should make efficient use of server resources (CPU, memory, disk space, etc.) to optimize performance.

# Software Quality

Doompause Emporium’s web application:

1. **User Experience (UX)**: The application should provide a seamless and enjoyable user experience. This can be quantified by measuring user satisfaction through surveys or user feedback.
2. **Accessibility**: The application should be accessible to all users, including those with disabilities. This can be verified by conducting accessibility audits and ensuring compliance with standards like the Web Content Accessibility Guidelines (WCAG).
3. **Search Engine Optimization (SEO)**: The application should be easily discoverable on search engines to attract more users. This can be quantified by the website’s ranking on search engine result pages for relevant keywords.
4. **Cross-Browser Compatibility**: The application should function correctly across all major web browsers. This can be verified through cross-browser testing.
5. **Mobile Responsiveness**: Given the increasing use of mobile devices, the application should be responsive and function correctly on various screen sizes. This can be verified through responsive design testing on different devices.
6. **Load Time**: The application should load quickly to provide a good user experience and retain users. This can be quantified by measuring the load time of the application’s pages.
7. **Scalability**: As the user base grows, the application should be able to handle increased traffic without performance issues. This can be verified by conducting load testing or stress testing.
8. **Security**: The application should ensure secure transactions and protect user data. This can be verified through security audits and penetration testing.
9. **Maintainability**: The codebase should be clean, well-organized, and easy to maintain. This can be quantified by measuring technical debt or code complexity metrics.
10. **Analytics Integration**: The application should integrate with analytics tools to track user behavior and gather insights for improvement. This can be verified by checking the integration with tools like Google Analytics or similar platforms.

# Technical Requirements

| **FR**  **ID** | **TR**  **ID** | **Technical Requirement** |
| --- | --- | --- |
| FR | 001 | Lombok to reduce boiler plate code. Lombok.jar - Backend |
|  | 002 | Upgrade to Java Development Kit 21 |
|  | 003 | import jakarta.persistence.Entity;  import jakarta.persistence.Table;  import java.math.BigDecimal;  import java.util.Date; |
|  | 004 | Custom REST Resource: @RepositoryRestResource(collectionResourceRel = "productCategory", path = "product-category") in DAO product category interface. Name of JSON entry path is the /product-category. |
|  | 005 | MySQL57Dialect has been deprecated; use org.hibernate.dialect.MySQLDialect instead |
|  | 006 | Springboot 4, JpaRepositorys, JDBC Driver, WebDev Tools, Crud Repository. |

# Open Issues

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Issue ID** | **Issue** | **Raised By** | **Raised On** | **Solution/ Decision** | **Resolved By** | **Resolved On** | **Status** |
| 001 | javax.persistence API version issue. | Daniel Rumfelt | 09/21/2023 | Update version and remove deprecated issues.  <version>2.2</version | Daniel Rumfelt | 09/22/2023 | Closed |
| 002 | Starting with MySQL 8.0.4, the MySQL team changed the default authentication plugin for MySQL server from mysql\_native\_password to caching\_sha2\_password. | Daniel Rumfelt | 09/21/2023 | The command below will make the appropriate updates for your user account.See the MySQL Reference Manual for details:  https://dev.mysql.com/doc/refman/8.0/en/caching-sha2-pluggable-authentication.html | Daniel Rumfelt | 09/22/2023 | Closed |
| 003 | Known Bug with Lombok. When using the JPA entities, the one-to-one or many-to-one @Data. | Daniel Rumfelt | 09/22/2023 | import lombok.Getter;  import lombok.Setter; | Daniel Rumfelt | 09/22/2023 | Closed |

# Appendix

Reference:

Weapons and description: <https://en.wikipedia.org/wiki/List_of_medieval_weapons>