TheCouchPotato.com: An E-commerce Platform for Comfort Items

Christopher Markel

CST-451-452 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Donna Jackson

Revision: 3

Date: 9/19/2023

**ABSTRACT**

TheCouchPotato.com is an e-commerce platform designed to provide a one-stop solution for consumers seeking products that enhance their comfort and relaxation. The project aims to offer a user-friendly interface, secure transactions, a diverse range of comfort products, and promote sustainable and ethically sourced products. This document serves as the final architectural plan and design specification, transforming the requirements analysis into a complete, detailed system design specification. It includes an overview of the project, final technical requirements, design planning summary, detailed solution architecture, and revision and sign off sheet. The document provides sufficient information for a developer to produce the system, ensuring that the platform meets the functional, non-functional, and technical requirements necessary to provide a secure, efficient, and user-friendly experience.

|  |
| --- |
| History and Sign Off Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 8/30/2023 | Chris Markel | Initial draft for review/discussion |
| 9/2/2023 | Chris Markel | Created Gantt chart for project schedule using Placker planning tool |
| 9/2/2023 | Chris Markel | De-scoping recommendation engine due to time constraints and complexity. |
| 9/10/2023 | Chris Markel | Changing frontend design from React to Thymeleaf templates in Spring Boot. |
| 9/17/2023 | Chris Markel | De-scoping product review/ ratings due to time constraints and complexity. |
| 917/2023 | Chris Markel | De-scoping manage brands (admin) due to time constraints and complexity. |
| 9/17/2023 | Chris Markel | De-scoping manage orders (admin) due to time constraints and complexity. |
| 9/17/2023 | Chris Markel | De-scoping view order history due to time constraints and complexity. |
| 9/17/2023 | Chris Markel | De-scoping track order status due to time constraints and complexity. |
| 9/19/2023 | Chris Markel | Changed position of functional requirements in report to make it more intuitive to the reader. |

|  |
| --- |
| **Overall Instructor Feedback/Comments**   * Clarification needed on device compatibility (desktops and mobile devices). * Clarification needed on the implementation of the recommendation engine (use of ML engine). * Recommendation to use a graphical representation like a Gantt chart for the schedule instead of text. |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

xYes ☐ No

**Useful Links**

Github Repository: <https://github.com/cylon-raider/TheCouchPotato>

UML diagram can be found [**here**](https://drive.google.com/file/d/1Sr3Z6cgfuYXntoKBngekzmW0VIfUT-JO/view?usp=sharing).

Data Dictionary Template can be found [**here**](https://docs.google.com/spreadsheets/d/1f5zjZkVCaFE3XZJvRgb-NlB3Osxb4YCK/edit?usp=sharing&ouid=103213121461926363767&rtpof=true&sd=true).

Project Schedule Gantt Chart can be found [**here**](https://placker.com/app#/gantt?e=b&s=p499a9).

**Response to Instructor Feedback/Comments**

1. Clarification needed on device compatibility (desktops and mobile devices).

*Device Compatibility*:

The application is being designed as a web application, which will be responsive and compatible with both desktop and mobile devices. I will be using a responsive design framework to ensure that the application adjusts to different screen sizes and devices.

1. Clarification needed on the implementation of the recommendation engine (use of ML engine).

*Implementation of the Recommendation Engine*:

After careful consideration of the project's time constraints, complexity, and my current skill level, I have decided to descope the recommendation engine from the project. This decision was made to ensure that the core functionalities of the application are developed and tested thoroughly. While a recommendation engine would be a valuable feature, it is not essential for the initial launch of the application. It may be considered for a future enhancement once the application is up and running.

1. Recommendation to use a graphical representation like a Gantt chart for the schedule instead of text

*Use of Gantt Chart for Schedule*:

Thank you for the recommendation to use a graphical representation like a Gantt chart for the schedule. I agree that a Gantt chart would provide a clearer visual representation of the project schedule and I will update the project plan to include a Gantt chart. I am also including a screenshot of it on this report, under the section “Other Documentation” on page 27. You can also find a link on the previous page that will take you directly to the chart on the Placker website.

**TABLE OF CONTENTS**

**Design Overview 4**

**Detailed High-Level Solution Design 5**

**Detailed Technical Design 6**

**Appendix A – Technical Issue and Risk Log 7**

**Appendix B – References 8**

**Appendix C – External Resources 9**

**Design Introduction**

1. Provide the high-level design of the proposed solution or business case with supporting narrative text. This design should include mock-up screen shots for the proposed user interface, pseudocode, or flowcharts that show the logic for the program, as well as the anticipated process flow. The purpose of the solution/business case design is to allow the stakeholder to approve the concepts before committing resources to the technical design.

The high-level design of the proposed solution is structured to address the key requirements of the business case, ensuring a seamless and efficient user experience. The design includes mock-up screen shots of the proposed user interface, flowcharts depicting the logic of the program, and the anticipated process flow.

The user interface is designed to be intuitive and user-friendly, with clear navigation and a clean layout. The mock-up screen shots included in the report illustrate the key screens that users will interact with, providing a visual representation of the user interface design.

The logic of the program is illustrated through flowcharts included in the report. These flowcharts outline the key processes and decision points within the application, providing a clear visual representation of the program's logic and anticipated process flow.

The purpose of this high-level design is to provide stakeholders with a clear and comprehensive overview of the proposed solution, allowing them to approve the concepts before committing resources to the technical design. This approach ensures alignment between stakeholder expectations and the final solution, minimizing the risk of costly changes later in the development process.

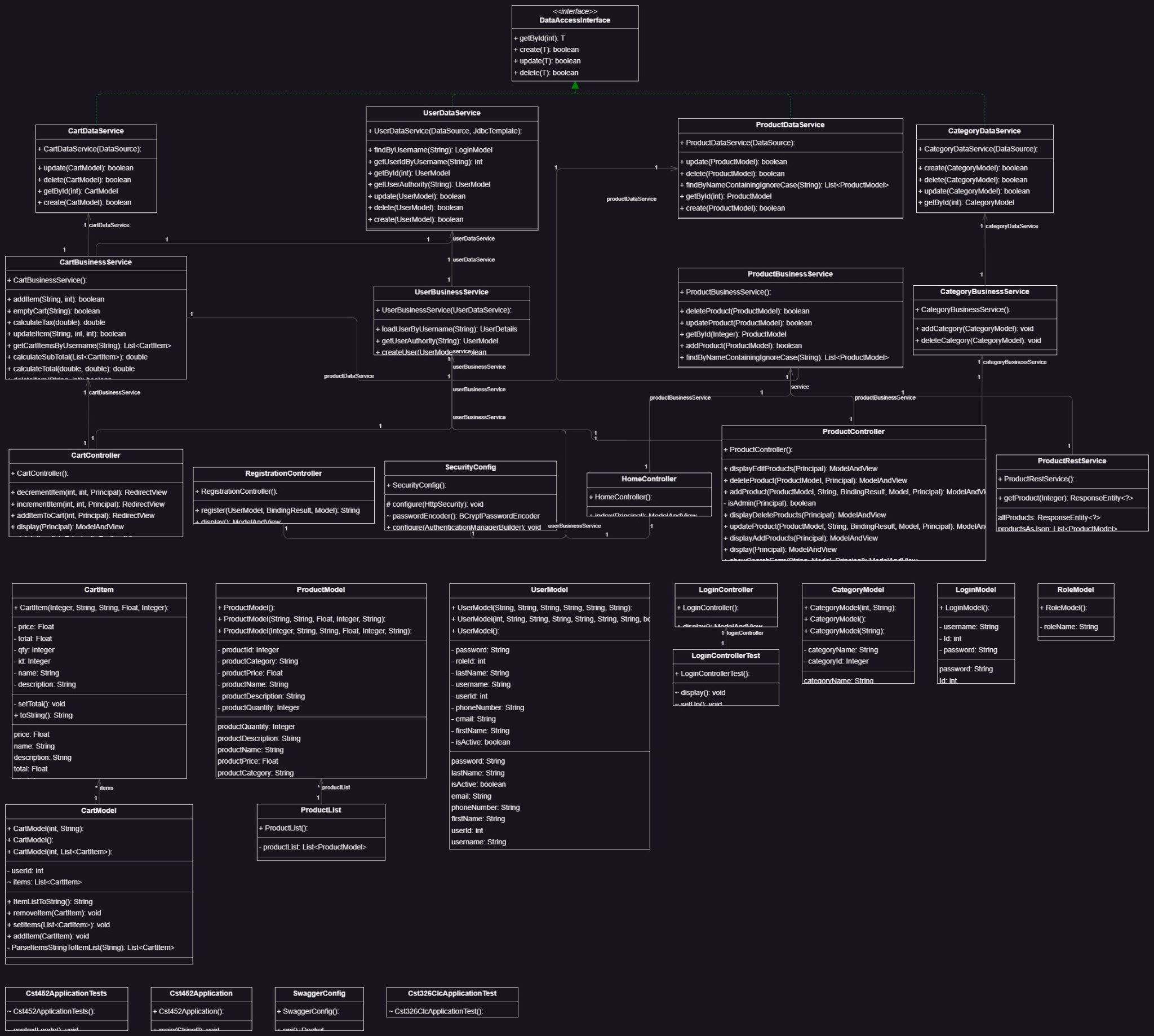
1. Use the template to list the project deliverables that are to be included external to this Design Specification (Data Dictionary, API Design, etc.).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deliverable Acceptance Log | | | | | |
| ID | Deliverable Description | Comments | Evaluator (internal or external as applicable) | Status | Date of Decision |
| 1 | CST-451 Data Dictionary Template | Data dictionary in excel format | External | Submitted | 9/3/2023 |
| 2 | CST-451-452 Capstone Project API Documentation | Not a retrievable API yet. Currently only sample documentation. | External | Submitted | 9/3/2023 |
| 3 | CST-451\_UMLdrawio.png | UML Diagram created using draw.io | External | Submitted | 9/3/2023 |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |

***NOTE: If necessary, you may add subsections to those listed in order to match the requirements in the assignment description. Do not remove any top level sections (those that are listed in the Table of Contents).***

**Detailed High-Level Solution Design**

1. Provide a detailed overview of how the proposed design fits into the overall solution.
   1. Create diagrams to logically and physically depict the system. This can be illustrated using UML Component, UML Deployment, and UML Activity diagrams or simply block diagrams done in a drawing tool such as Visio.



*Figure 1: UML diagram depicting class structure, interfaces, constructors, methods, and properties.*

* 1. This section should also include any solution configuration changes that will be required to develop and implement the proposed solution.

To develop and implement the proposed solution, several configuration changes will be required:

*Database Configuration*: The MySQL database will need to be set up and configured to match the structure defined in the Data Definition Language (DDL) scripts. This includes creating tables for cart, category, product, role, and user.

*Server Configuration*: The server will need to be configured to handle the API requests. This includes setting up the server to handle different routes, requests, and responses as defined in the API documentation.

*Security Configuration*: Security measures will need to be implemented to ensure the safety of the data. This includes setting up authentication and authorization for the API, and ensuring data is encrypted during transmission.

*Performance Configuration*: The system will need to be configured to handle the expected load. This includes setting up load balancing, caching, and optimizing database queries. (Future Implementations)

* 1. Describe the approach and resources required to assure non-functional requirements (such as security and performance) will be met with this solution.

**Security:**

*Approach:* Implement authentication and authorization for the API to ensure only authorized users can access certain endpoints. Use HTTPS to encrypt data during transmission.

*Resources:* Use Spring Security for authentication and authorization. Use an SSL certificate to enable HTTPS.

**Performance:**

*Approach:* Optimize database queries to reduce response times. Implement caching for frequently accessed data. Use load balancing to distribute traffic across multiple servers.

*Resources:* Use a database profiling tool to optimize queries. Use a caching library like EhCache. Use a load balancer like Nginx.

**Scalability:**

*Approach:* Design the application to be stateless so it can be easily scaled horizontally. Use a load balancer to distribute traffic across multiple instances of the application.

*Resources:* Use a load balancer like Nginx. Use a cloud computing service like AWS or Azure to easily scale the application.

**Reliability:**

*Approach:* Implement proper error handling and logging in the application. Use a monitoring tool to monitor the health of the application and the server.

*Resources:* Use a logging library like Logback. Use a monitoring tool like Prometheus and Grafana.

**Maintainability:**

*Approach:* Follow best practices for coding and design. Write unit tests for the code. Use version control to track changes and collaborate with other developers.

*Resources:* Use a version control system like Git. Use a unit testing framework like JUnit.

By following this approach and using these resources, the non-functional requirements of security, performance, scalability, reliability, and maintainability will be met with this solution.

The purpose of the detailed solution architecture is to provide sufficient information for a developer to produce the system.

1. Provide a detailed inventory of hardware and software technologies that will be used in the solution:
   1. List any Frameworks or third party libraries that will be used.
   2. List any Proof of Concepts to be completed (POC) to ensure that the technologies and frameworks selected are the best fit for purpose, cost effective, and proper to solve the problem. This section should also be updated with the purpose/rationale for the POC and the results of the POC.

Use the templates below to list the Proof of Concepts, hardware, and technologies.

|  |  |  |
| --- | --- | --- |
| Proof of Concepts | |  |
| **Description** | **Rationale** | **Results** |
| 1 - Spring Boot and MySQL Integration  Description: A proof of concept to test the integration of Spring Boot and MySQL. | To ensure that Spring Boot can successfully connect to the MySQL database and perform CRUD operations. | Testing |
| 2 - Authentication and Authorization  Description: A proof of concept to test the implementation of authentication and authorization. | To ensure that only authorized users can access certain endpoints and that the user's data is secure. | Testing |
| 3 - Load Balancing  Description: A proof of concept to test the implementation of load balancing. | To ensure that the application can handle a large number of requests and distribute the load across multiple servers. | TBD - Out of scope |
| 4 - Caching  Description: A proof of concept to test the implementation of caching. | To ensure that frequently accessed data is cached to improve the performance of the application. | TBD - Out of scope |

**Final Technical Requirements**

**Use Cases**

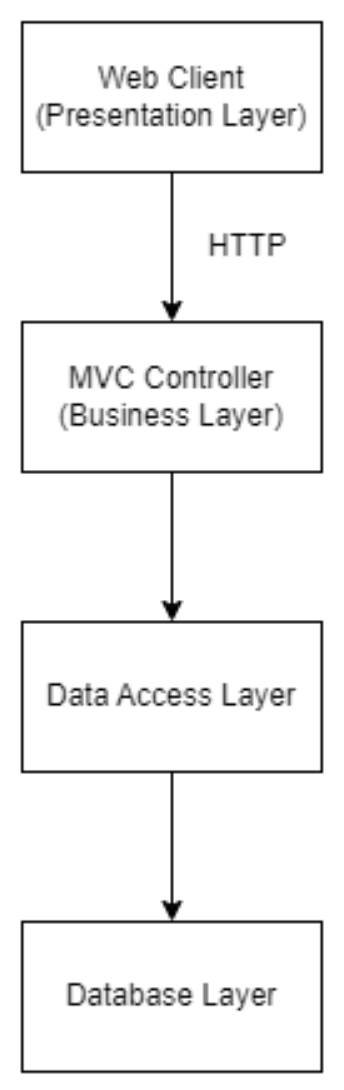
Describe the tools and technologies used in the project.

NOTE: Once the technical requirements have been completed, there may be situations where technologies or tools may need to be taken out of scope or changed, possibly due to technical challenges or timeline challenges. Any technologies or tools that are taken out of scope or changed once the project development has started must be approved by the mentor and instructor with justification as to why the functionality is being removed from the project. The following must be updated if any technologies or tools are taken out of scope or changed:

|  |  |  |
| --- | --- | --- |
| **Technology or Tool** | **Approval Date** | **Justification** |
| React.js | 9/3/2023 | Connectivity issues with backend Spring boot application. Integrating Thymeleaf templates for frontend instead. |
| AWS (Amazon Web Services) | 9/3/2023 | De-scoping requirement due to time constraints. Locally hosting application for this course. |

**Logical Solution Design:**

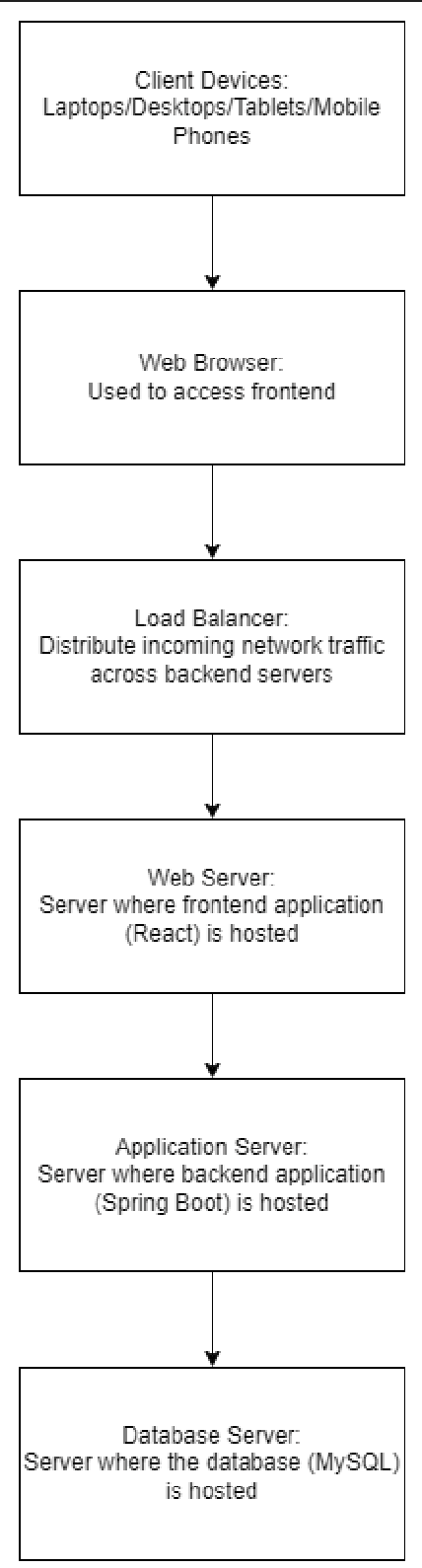
Provide the proper diagrams and drawings that represent the high-level logical solution design.



*Figure 2: In this diagram, the web client communicates with the MVC controller layer through HTTP. The controller layer handles business logic and communicates with the data access layer to retrieve or update data from the database layer*

**Physical Solution Design:**

Provide the proper diagrams and drawings that represent the high-level physical solution design.



*Figure 3: High-Level Physical Solution Design of the CouchPotato Application: A visual representation of the hardware, software, and network components that will interact to form the complete CouchPotato system.*

**Detailed Technical Design**

**General Technical Approach:**

You should, in words, describe your approach and design here. You should also summarize any meeting notes, brainstorming sessions, etc. that you want to retain through the design of your project.

The technical approach for this project involves developing a robust and scalable e-commerce application using a combination of modern technologies and best practices. The application will be developed using a microservices architecture, which will allow for flexibility, scalability, and maintainability.

*Frontend*: The frontend of the application will be developed using Spring Boot Thymleaf templates that will act as the view in the MVC pattern.

*Backend*: The backend of the application will be developed using Spring Boot, a popular Java-based framework for building production-ready applications. Spring Boot provides a lot of out-of-the-box features such as security, data access, and messaging which will help in speeding up the development process. The backend will expose RESTful APIs that will be consumed by the frontend.

*Database*: The database used for this application will be MySQL, a popular open-source relational database management system. MySQL provides a robust and scalable solution for storing and retrieving data.

*Version Control*: Git will be used for version control. Git will help in tracking changes, reverting to previous stages, and working on different branches simultaneously.

*API Documentation*: The API documentation will be created using Postman. Postman is a popular tool for API development and testing. It provides features to create, share, test, and document APIs.

*Deployment*: The application will be deployed on a cloud-based server. This will allow for scalability and flexibility in terms of resources.

*Testing*: The application will be thoroughly tested using a combination of unit tests, integration tests, and end-to-end tests. This will ensure that the application is robust and free of bugs.

*Security*: Security will be a top priority in the development of this application. Best practices such as data encryption, user authentication, and authorization will be implemented to ensure the security of the application.

**Key Technical Design Decisions:**

Any final technical design decisions, such as framework decisions, etc., should be documented here. This should list the technology/framework, its purpose in the design, and why it was chosen. If necessary, the proper Proof of Concepts should be defined and implemented to validate the technical decision.

**Technology/Framework: Spring Boot**

*Purpose*: To develop the backend of the application and expose RESTful APIs.

*Reason for Choosing*: Spring Boot is a popular Java-based framework for building production-ready applications. It provides a lot of out-of-the-box features such as security, data access, and messaging, which will help in speeding up the development process.

**Technology/Framework: MySQL**

*Purpose*: To store and retrieve data for the application.

*Reason for Choosing*: MySQL is a popular open-source relational database management system. It provides a robust and scalable solution for storing and retrieving data.

**Technology/Framework: HTML Basic Authentication**

*Purpose*: For user authentication.

*Reason for Choosing*: HTML Basic Authentication is a simple authentication scheme built into the HTTP protocol. The client sends HTTP requests with an Authorization header containing the word 'Basic' followed by a space and a base64-encoded string 'username:password'. It is a widely used method for controlling access to web resources.

**Tool: Postman**

*Purpose*: For API development, testing, and documentation.

*Reason for Choosing*: Postman is a popular tool for API development and testing. It provides features to create, share, test, and document APIs.

**Deployment Platform: MAMP**

*Purpose*: To locally host the application during the initial development phase.

*Reason for Choosing*: MAMP is a free, local server environment that can be installed under macOS and Windows with just a few clicks. It provides an easy-to-use environment to develop and test the application locally before deploying it to a cloud-based server.

**Deployment Platform: Cloud-based server (Future Implementation)**

*Purpose*: To deploy the application and make it accessible over the internet.

*Reason for Choosing*: Cloud-based servers provide flexibility and scalability in terms of resources. It also provides a level of abstraction as the developer does not have to worry about the underlying hardware and infrastructure.

**Version Control System: Git**

*Purpose*: For version control of the application code.

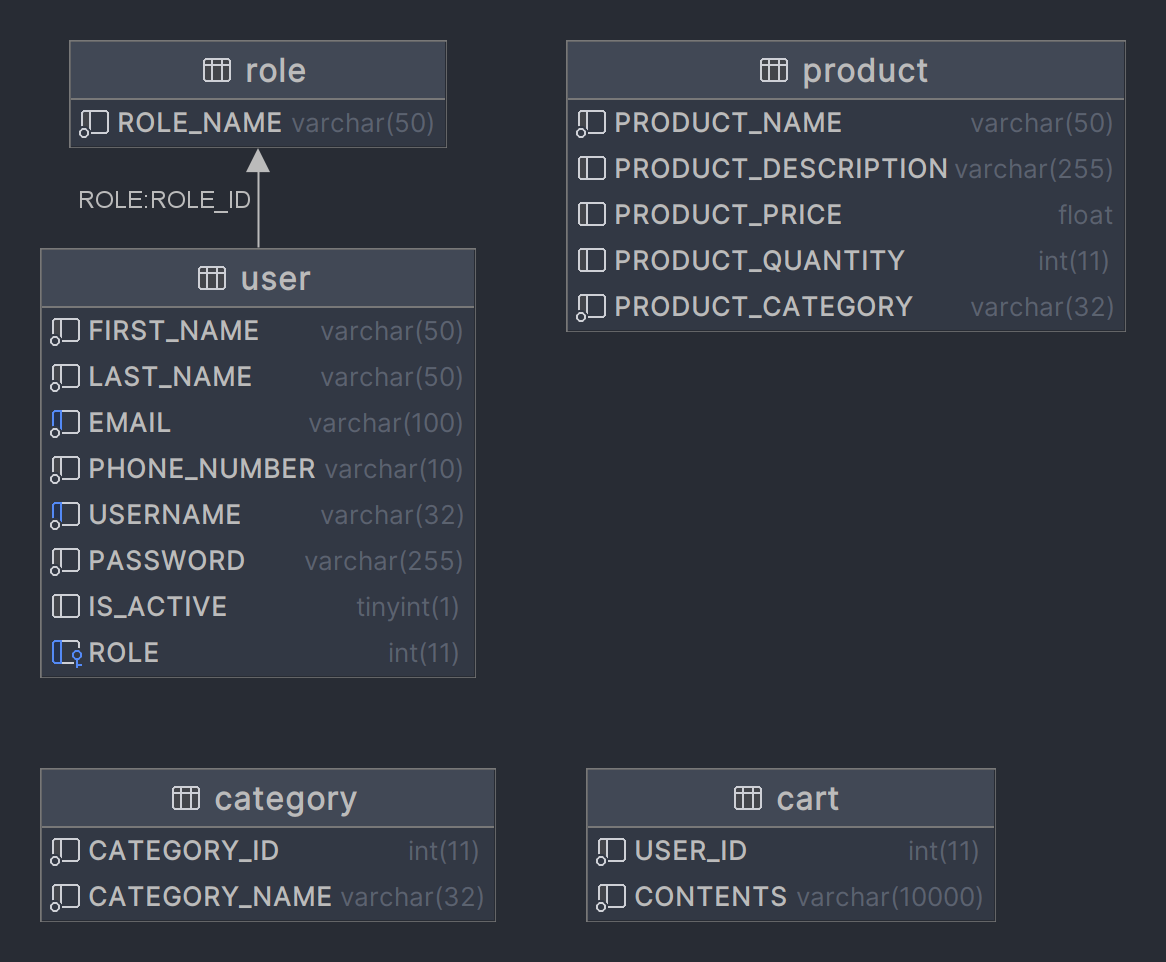
*Reason for Choosing*: Git is a widely used version control system that helps in tracking changes, reverting to previous stages, and working on different branches simultaneously.

**Proof of Concepts (POC):**

No POCs were required for this project as all the technologies and frameworks selected are widely used, well-documented, and have a large community of developers. Additionally, the team (myself) has prior experience working with these technologies and frameworks.

**Database ER Diagram:**

Image file of your ER database diagram.



*Figure 4: Database ER Diagram of the CouchPotato Application: A visual representation of the entities, relationships, and attributes that make up the CouchPotato application's database schema.*

**Database DDL Scripts:**

This should contain the DDL script showing all database constraints, etc.

create table cart

(

USER\_ID int not null,

CONTENTS varchar(10000) not null

);

create table category

(

CATEGORY\_ID int not null,

CATEGORY\_NAME varchar(32) not null

);

create table product

(

PRODUCT\_ID int auto\_increment

primary key,

PRODUCT\_NAME varchar(50) not null,

PRODUCT\_DESCRIPTION varchar(255) null,

PRODUCT\_PRICE float default 0 null,

PRODUCT\_QUANTITY int default 0 null,

PRODUCT\_CATEGORY varchar(32) not null

);

create table role

(

ROLE\_ID int auto\_increment

primary key,

ROLE\_NAME varchar(50) not null

);

create table user

(

USER\_ID int auto\_increment

primary key,

FIRST\_NAME varchar(50) not null,

LAST\_NAME varchar(50) not null,

EMAIL varchar(100) not null,

PHONE\_NUMBER varchar(10) not null,

USERNAME varchar(32) not null,

PASSWORD varchar(255) not null,

IS\_ACTIVE tinyint(1) default 1 null,

ROLE int default 4 null,

constraint USER\_EMAIL\_uindex

unique (EMAIL),

constraint USER\_USERNAME\_uindex

unique (USERNAME),

constraint USER\_ROLE\_ROLE\_ID\_fk

foreign key (ROLE) references role (ROLE\_ID)

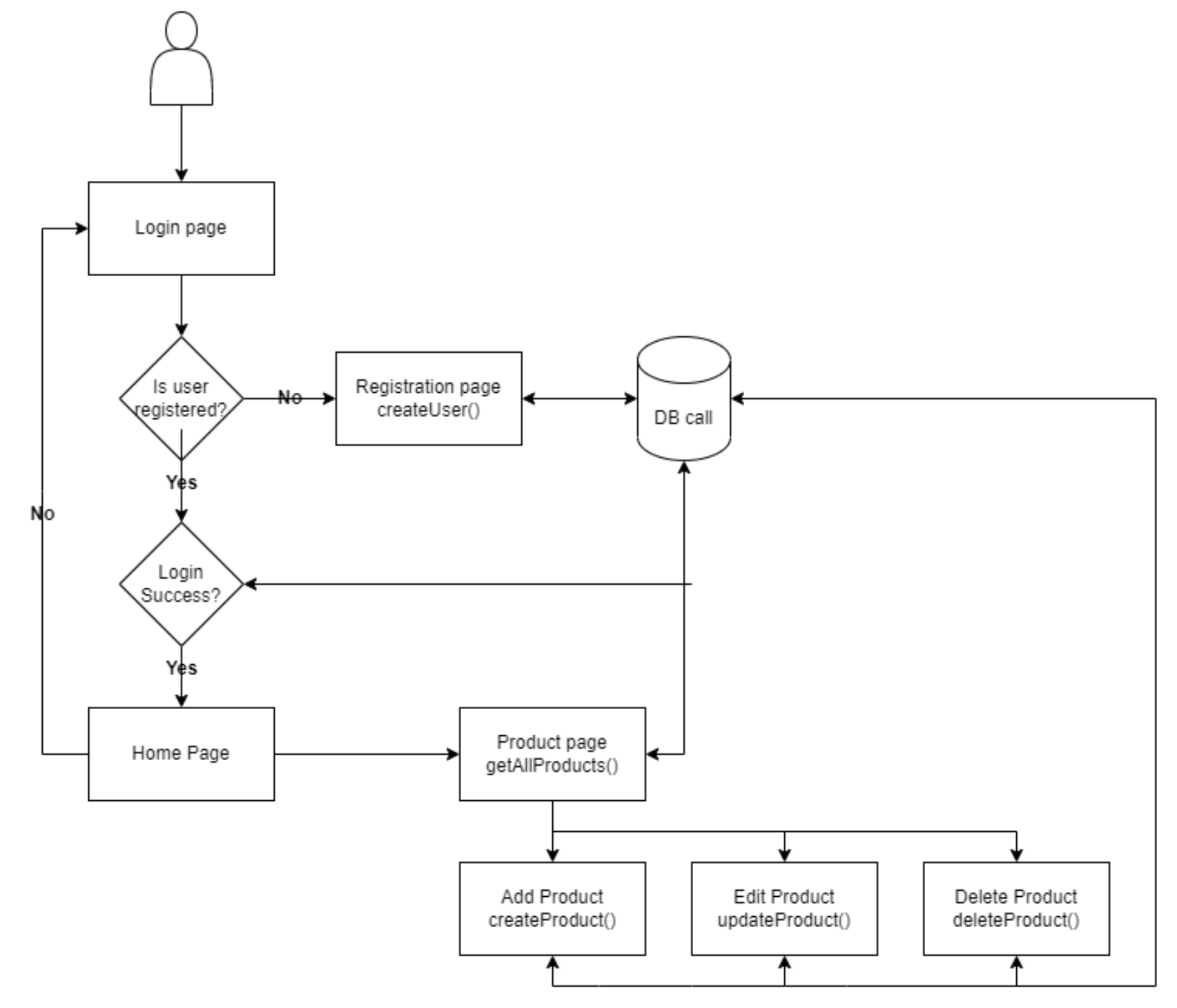
on update cascade on delete cascade

);

**Flow Charts/Process Flows:**

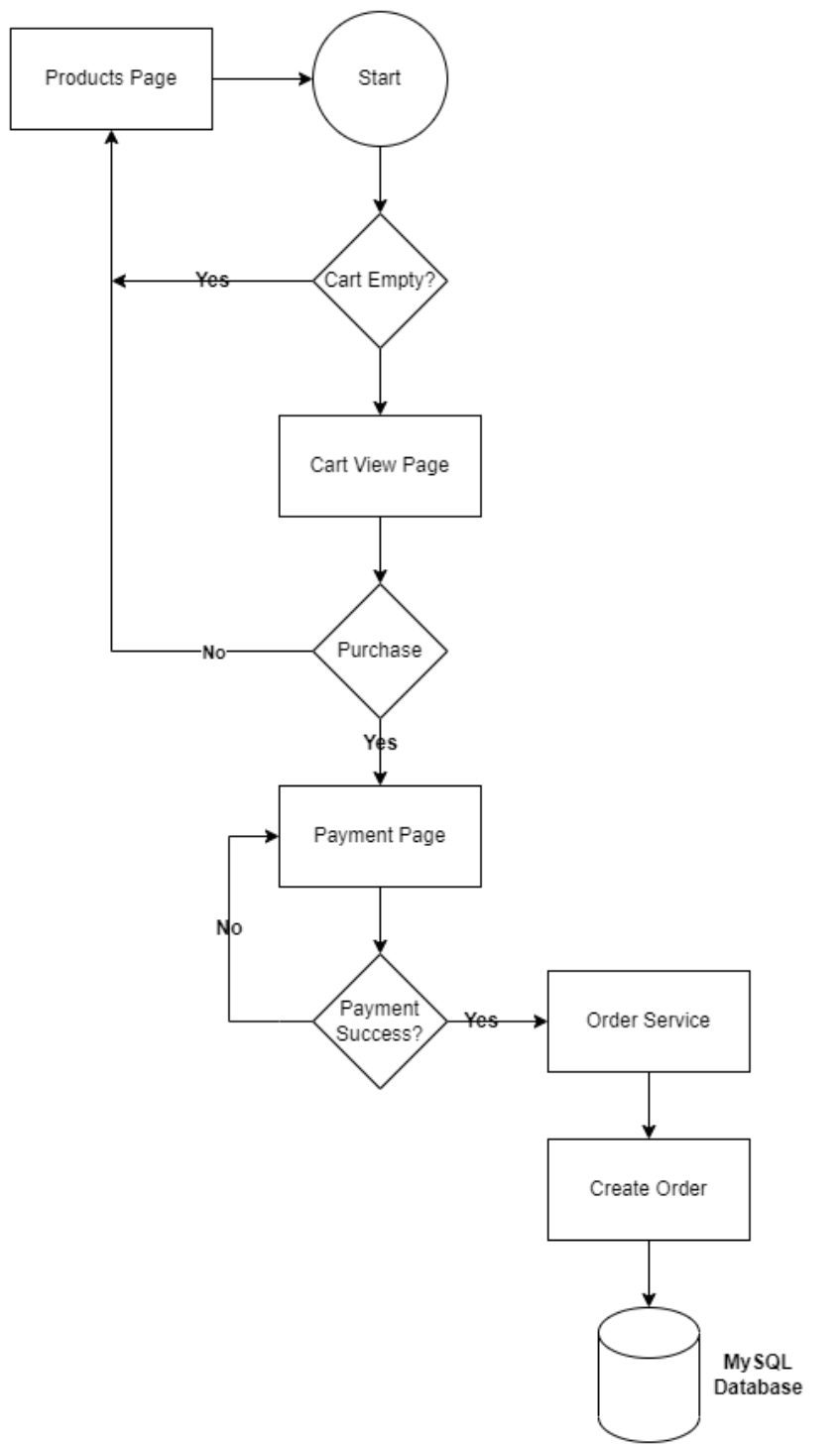
You should insert any flow charts or UML Activity diagrams here. Flow charts should document algorithms or workflows that will be implemented in your program.

*Logical System Diagram*

**

*Figure 5:This logical system diagram shows a high-level overview of the system, how the components are interconnected, and how they contribute to the system.*

*Shopping Cart Logical System Diagram*

**

*Figure 6:The shopping cart logical user diagram shows a high-level overview of the shopping cart system, how the components are interconnected, and how they contribute to the system.*

**Sitemap Diagram:**

Image file of your Sitemap diagram.

**A diagram of a product

Description automatically generated**

*Figure 7: This site map provides a clear and logical structure for the website, allowing users to easily navigate and find the information they need.*

**User Interface Diagrams:**

You should insert any wireframe drawings or white board concepts that were developed to support your application. If you have no supporting documentation, explain the rationale why you are able to leave this section as N/A.

*Index/Home Page*

A screenshot of a web page

Description automatically generated

*Figure 8: A wireframe of the Index, or Home page*

*Products Page*

A screenshot of a web page

Description automatically generated

*Figure 9: A wireframe of the Products page*

*Login*

A screenshot of a login page

Description automatically generated

*Figure 10:A wireframe of the Login page*

*Registration*

A screenshot of a login form

Description automatically generated

*Figure 11: A wireframe of the Registration page*

*Add Product*

A screenshot of a web page

Description automatically generated

*Figure 12: A wireframe of the Add Product page*

*Update Product*

A screenshot of a web page

Description automatically generated

*Figure 13: A wireframe of the Update Product page*

*Delete Product*

A screenshot of a web page

Description automatically generated

*Figure 14: A wireframe of the Delete Product page*

**UML Diagrams:**

You should insert any UML Class diagrams and UML Sequence diagrams here. If you have no supporting documentation, explain the rationale why you are able to leave this section as N/A.

*UML Diagram submitted as an external file*

**Service API Design:**

This section should fully document any third party Service Interface APIs being consumed or application specific Service API’s being published, how to access the service, what parameters are required by the API, and the detailed JSON data format specification that could be used by a third party developer to integrate with the service and API.

*Service API Design submitted as an external file. It is currently inaccessible and only a high-level example as there is currently no working code*

**NFR’s (Security Design, etc.):**

This section should outline how non-functional requirements will be supported by the design.

**Security:**

*Authentication*: Users will be authenticated using a username and password. The password will be hashed and stored securely in the database.

*Authorization*: Different users will have different roles and permissions. For example, an admin user will have more permissions than a regular user.

*Data Encryption*: Sensitive data like passwords will be encrypted before being stored in the database.

*Secure Communication*: All data transferred between the client and server will be encrypted using HTTPS.

**Performance:**

*Caching*: Frequently accessed data will be cached on the server-side to reduce the load on the database and improve response times.

*Load Balancing*: If the application experiences high traffic, a load balancer will be used to distribute the load across multiple servers.

*Database Optimization*: The database will be optimized by using proper indexing, partitioning, and normalization techniques.

**Scalability:**

The application will be designed to handle a large number of users and requests. The system will be able to scale horizontally by adding more servers if necessary.

*Availability*: The application will be designed to be highly available. Multiple instances of the application will be running, and if one instance fails, the traffic will be redirected to another instance.

**Usability:**

The user interface will be intuitive and easy to use. It will be designed with the end-user in mind and will be tested with real users to ensure usability.

**Maintainability:**

The code will be well-documented and follow best practices. This will make it easier for other developers to understand and maintain the code in the future.

**Portability:**

The application will be designed to be portable and will be able to run on different operating systems and devices.

**Responsiveness:**

The application will be designed to be responsive and will work well on both desktop and mobile devices.

By addressing these NFRs in the design of the application, you can ensure that the application is secure, performant, scalable, available, usable, maintainable, portable, and responsive.

**Operational Support Design:**

This section should fully document how your design supports monitoring and logging.

**Monitoring**: (Future implementation)

*Application Performance Monitoring (APM):* Tools like New Relic or Dynatrace will be used to monitor the application's performance in real-time. These tools will provide insights into the application's response times, error rates, and throughput.

*Infrastructure Monitoring*: Tools like AWS CloudWatch or Nagios will be used to monitor the infrastructure, including the servers, databases, and network. These tools will provide insights into the CPU usage, memory usage, disk usage, and network activity.

*Error Trackin*g: Tools like Sentry or Rollbar will be used to track errors in the application. These tools will provide detailed information about the errors, including the stack trace, the affected users, and the frequency of the errors.

**Logging**:

*Application Logs*: SLF4J with Logback:

Advantages: Logback is the successor to the popular Log4j framework and is designed to be faster and have a smaller footprint. It's also the default logging implementation for Spring Boot.

**Alerting**: (Future implementation)

*Threshold-based Alerts*: Alerts will be set up for different metrics, like response times, error rates, and CPU usage. If these metrics cross a certain threshold, an alert will be triggered, and the operations team will be notified.

*Anomaly Detection*: Machine learning algorithms will be used to detect anomalies in the application's behavior. If an anomaly is detected, an alert will be triggered, and the operations team will be notified.

**Backup and Recovery**:

*Database Backup*: The database will be backed up regularly to a secure location. In case of a failure, the database can be restored from the backup.

*Data Replication:* The data will be replicated across multiple servers to ensure availability in case of a server failure.

**Documentation**:

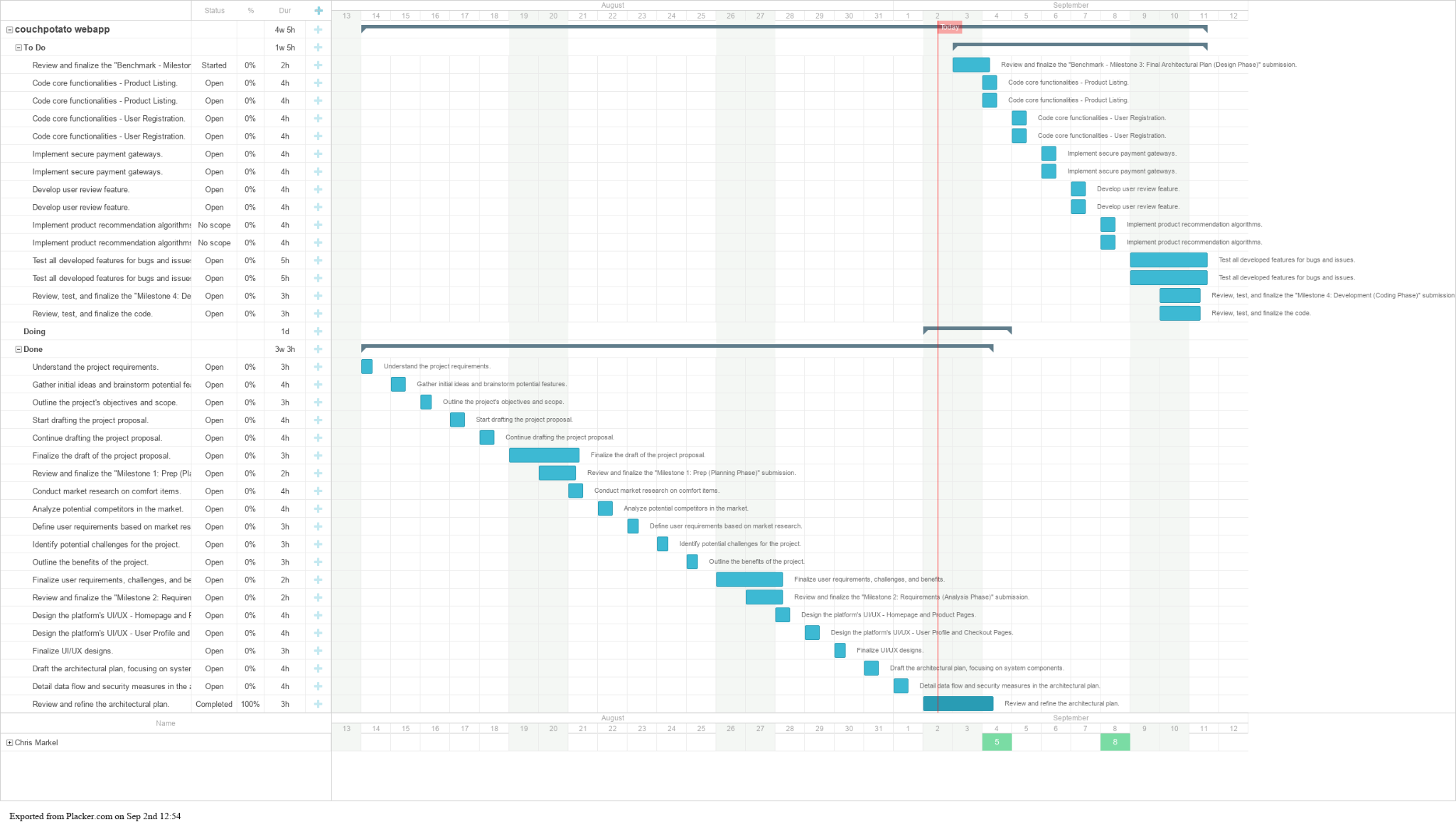
Comprehensive documentation will be created for the operational support of the application. This will include documentation for the monitoring tools, logging tools, alerting systems, backup and recovery procedures, and troubleshooting guides.

By implementing these strategies and tools, the design will support comprehensive monitoring and logging of the application, ensuring its smooth operation and facilitating quick resolution of any issues that may arise.

**Other Documentation:**

You should insert any additional drawings, storyboards, white board pictures, project schedules, tasks lists, etc. that support your approach, design, and project. If you have no supporting documentation, explain the rationale why you are able to leave this section as N/A.

*Gantt Chart for Project Schedule*



*Figure 15: Project Schedule Gantt Chart: A visual timeline that outlines the various tasks, their estimated duration, and their dependencies throughout the CouchPotato application development process*

**Functional Requirements**

*User Registration*

| ID | Requirement |
| --- | --- |
| FR-1.1 | The application shall allow users to register for an account by providing their name, email address, and a password. |
| FR-1.2 | The application shall validate that the email address provided by the user is not already associated with an existing account. |
| FR-1.3 | The application shall store user account information in a MySQL database. |
| FR-1.4 | The application shall display a confirmation message to the user upon successful registration. |

*User Login*

| ID | Requirement |
| --- | --- |
| FR-2.1 | The application shall allow registered users to log in by providing their email address and password. |
| FR-2.2 | The application shall verify the user's credentials before granting access to protected areas of the site. |
| FR-2.3 | The application shall display an error message if the user's credentials are invalid. |

*Product Browsing*

| ID | Requirement |
| --- | --- |
| FR-3.1 | The application shall display a list of available products on the home page. |
| FR-3.2 | The application shall allow users to filter products by category, price, and other relevant attributes. |
| FR-3.3 | The application shall display detailed information about each product, including price, description, and images. |
| FR-3.4 | The application shall provide users with the ability to add products to a shopping cart. |

*Product Purchasing*

| ID | Requirement |
| --- | --- |
| FR-4.1 | The application shall allow users to view the contents of their shopping cart and update quantities as necessary. |
| FR-4.2 | The application shall allow users to enter their shipping and billing information during the checkout process. |
| FR-4.3 | The application shall display a summary of the order before processing payment. |
| FR-4.4 | The application shall process payments securely using a third-party payment gateway. |
| FR-4.5 | The application shall send a confirmation email to the user upon successful order completion. |

*MySQL Database*

| ID | Requirement |
| --- | --- |
| FR-5.1 | The database should allow users to register an account and store their personal information securely. |
| FR-5.2 | The database should allow users to login to their account securely. |
| FR-5.3 | The database should store product information including name, description, price, and availability. |
| FR-5.4 | The database should store order information including products ordered, quantity ordered, and total price. |
| FR-5.5 | The database should allow users to view their order history. |

**Non-functional Requirements**

| ID | Requirement |
| --- | --- |
| NFR-1.1 | Security: All user data should be protected from unauthorized access, and the system should be protected against attacks such as SQL injection, cross-site scripting, and cross-site request forgery. |
| NFR-1.2 | Performance: The system should be able to handle a large number of concurrent users, and response times for user registration, login, product browsing, and purchasing should be fast and consistent. |
| NFR-1.3 | Availability: The system should have a high level of uptime, with minimal downtime for maintenance and upgrades. |
| NFR-1.4 | Scalability: The system should be able to handle growth in user traffic and data storage requirements. |
| NFR-1.5 | Compatibility: The system should be compatible with a wide range of web browsers and operating systems. |
| NFR-1.6 | Usability: The user interface should be intuitive and easy to use, with clear instructions and error messages. |
| NFR-1.7 | Reliability: The system should be reliable, with minimal errors or crashes. |

|  |  |
| --- | --- |
| NFR-1.8 | Maintainability: The system should be easy to maintain, with clear documentation and well-organized code. |
| NFR-1.9 | Accessibility: The system should be accessible to users with disabilities, complying with web accessibility guidelines. |
| NFR-1.10 | Data integrity: The database should maintain the integrity of data, with proper data validation and error handling. |
| NFR-1.11 | Backup and recovery: The database should be regularly backed up and have a plan for disaster recovery. |
| NFR-1.12 | Data privacy: The system should comply with relevant data privacy laws and regulations, with user consent and proper handling of personal data. |

**Appendix A – Technical Issue and Risk Log**

1. Use the template to identify and monitor project issues and risks.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Issues and Risk Log | | | | | | | | |
| **Issue or Risk** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date to Review** | **Date Resolved** |
| I/R | What is the issue or risk? | How will this impact scope, schedule, and cost? | How do you intend to deal with this issue? | Who manages this issue? |  |  |  |  |
| R | Integration issues with third-party services | This could impact the functionality of the application, and potentially the schedule and cost if significant rework is required | Conduct thorough testing of the integration points early in the development process. Have a backup plan for alternative services if necessary. | Development Team | Medium | 9/3/2023 | 9/10/2023 |  |
| I | Budget overrun due to unexpected costs (only applicable after cloud hosting begins) | This impacts the cost of the project. | Regularly monitor the budget and update the project plan and stakeholders if any unexpected costs arise. Look for areas where costs can be reduced. | Project Manager | High | 9/3/2023 | 10/1/2023 |  |

**Appendix B – References**

*Shen-Guan Shih. (2014). Notes on Generative Modeling, Procedural Symmetry, and Constructability of Architectural Design. Computer-Aided Design & Applications, 11(5), 518–525.* [*https://doi-org.lopes.idm.oclc.org/10.1080/16864360.2014.902681*](https://doi-org.lopes.idm.oclc.org/10.1080/16864360.2014.902681)

*Software Architecture & Design Introduction. (2023). Retrieved from* [*https://www.tutorialspoint.com/software\_architecture\_design/introduction.htm#*](https://www.tutorialspoint.com/software_architecture_design/introduction.htm)

*Trello project management: Power-Up! (2023). Retrieved from* [*https://placker.com/trello-project-management.html*](https://placker.com/trello-project-management.html)

**Appendix C – External Resources**

|  |  |
| --- | --- |
| **GIT URL:** | *https://github.com/cylon-raider/TheCouchPotato* |
| **Hosting URL:** | *Initial development will only include local hosting- localhost:8080* |