ShopIT: Enhancing Local Businesses Through Secure E-Commerce

Senior Capstone Proposal

Ahmad Bishara

Fall 2024

Contents

1 Introduction and Background **5**

2 Requirements **8**

3 Architecture **12**

3.1 Client-Side (Front End) 12

3.2 Server-Side (Back End) 13

4 Interfaces **15**

4.1 User Interfaces 15

4.1.1 Login and Registration Page 15

4.1.2 Product Listing Page 15

4.1.3 Product Detail Page 16

4.1.4 Shopping Cart Page 17

4.1.5 Cart drop down menu 18

4.2 Backend APIs 18

4.2.1 Authentication API 18

4.2.2 Product Retrieval API 19

4.2.3 Product Detail API 20

4.2.4 Order Processing API 21

4.2.5 Design Choices 22

5 Data Model **23**

5.1 Data Layer (Database and Data Model) . . . 23
5.1.1 Design Choices . . . 23
5.1.2 Data Storage . . . 24
5.1.3 Data Model Overview . . . 24
5.1.4 MySQL Justification for E-Commerce . . . 24
5.2 Design Choices and Alternatives . . . 27
5.2.1 MongoDB vs. Relational Databases . . . 27
5.2.2 JWT vs. Session-Based Authentication . . . 28
5.2.3 RESTful API Design . . . 29
5.3 Data Model Illustration . . . 29
5.4 Collections and Relationships . . . 29
5.4.1 Users Collection . . . 29
5.4.2 Products Collection . . . 30
5.4.3 Orders Collection . . . 30
5.4.4 Cart Collection . . . 30
5.4.5 Payments Collection . . . 30
5.5 Data Storage and Security . . . 31

6 Testing Plan . . . 32

6.1 Secure Checkout and Payment Processing . . . 32
6.1.1 Automated Testing . . . 32
6.1.2 User Testing . . . 32
6.2 Product Browsing and Search Functionality . . . 33
6.2.1 Automated Testing . . . 33
6.2.2 User Testing . . . 33
6.3 Shopping Cart Functionality . . . 33
6.3.1 Automated Testing . . . 33
6.3.2 User Testing . . . 34

6.4 User Authentication and Authorization 34
6.4.1 Automated Testing 34
6.4.2 User Testing 34

7 Prototype 35

8 Timeline 38

9 Summary 40

Abstract

This project aims to develop *ShopIT*, a comprehensive and secure e-commerce platform designed to facilitate online shopping, with a special emphasis on enhancing local businesses' global reach. The website will provide a seamless user experience, integrating features such as product browsing, shopping cart functionality, and secure checkout. It will ensure enhanced security protocols through the implementation of JSON Web Token (JWT) authentication [13], role-based control, and Payment Card Industry Data Security Standard (PCI-DSS) [20] compliant payment processing. The back end will be developed using Java and Spring Boot to run the server, while the front end will utilize React/JavaScript to deliver a responsive user interface. This platform will prioritize user privacy and data protection, handling sensitive information such as credit card details and shipping addresses securely. Furthermore, the project encompasses the design of APIs for managing user authentication, product retrieval, and order processing. By integrating advanced cybersecurity measures and a user-friendly interface, *ShopIT* aims to enhance the online presence of small businesses while ensuring customer trust and security.

Chapter 1

Introduction and Background

I have decided to build an e-commerce site called *ShopIT*, an online shopping platform that mirrors the offerings of a local store. My initial motivation for this project was to help my parents establish a better online network for their business, allowing their store to become active and visible globally. In today's competitive digital marketplace, businesses need an online presence to reach a broader audience and thrive. By creating this e-commerce site, I aim to extend the reach of their physical store beyond its geographic limitations, offering products to customers anywhere in the world.

As a computer science major, I have worked on several projects involving developing both front-end and back-end systems. This hands-on experience has given me the technical expertise required to manage the complexities of web application development. The front end, which will serve as the user interface, is what customers will interact with when browsing products, placing orders, and managing their accounts. This portion of the website must be intuitive and responsive, providing a seamless experience across different devices. On the other hand, the back end—what customers do not see—handles the critical data transactions between the site and the server. This part of the system will be responsible for tasks such as retrieving product information from a database, processing orders, handling payments, and managing user authentication.

Building an e-commerce site of this nature requires not only powerful functionality but also a high level of security. Since the platform will facilitate online transactions, it is crucial to protect sensitive customer information such as credit card numbers, mailing addresses, and personal details. The potential risks associated with handling this type of data include hacking, data breaches, and identity theft. To mitigate these risks, I will be integrating advanced security measures, including encryption and secure authentication methods. Drawing from my coursework in cybersecurity, I have learned about various vulnerabilities that e-commerce platforms face and how to safeguard against them. For example, implementing secure HTTPS connections [23], utilizing token-based authentication like JWT (JSON Web Tokens) [13], and following best practices for data encryption will be key components of this project.

Considerable e-commerce platforms available today offer a variety of features related to different business needs. For instance, Shopify [24] is a widely-used platform that enables businesses to create online stores, manage products, and process payments seamlessly. Its user-friendly tools make it an attractive option for entrepreneurs. Still, the subscription fees and limited customization options can be a drawback for small local businesses seeking a cost-effective solution. Similarly, WooCommerce [2], an open-source e-commerce plugin for WordPress, offers significant customization and flexibility. Although powerful, WooCommerce requires advanced technical expertise for optimal use, which may not be feasible for local businesses particularly those lacking IT resources.

On the other hand, platforms like Magento [1] and BigCommerce [3] provide medium to large businesses with features like scalability and multi-channel selling. Magento is known for its strength and adaptability but involves higher operational costs and complexity. BigCommerce provides a comprehensive set of tools, including SEO optimization, but its feature set might be overwhelming for small-scale retailers focusing on their immediate market.

Lastly, Squarespace [25] stands out for its aesthetically engaging templates and integrated e-commerce functionalities, making it ideal for style-focused users. However, like

other platforms mentioned, it charges subscription fees and imposes limitations on feature customization. These factors highlight a market gap for a customized e-Commerce solution for small, local businesses that balances cost, simplicity, and control.

Unlike these existing platforms, *ShopIT* aims to provide a customized and secure e-commerce solution specifically designed to address the unique challenges faced by small local businesses. Emphasizing flexibility, enhanced security, and seamless integration with existing business processes, *ShopIT* offers a customized approach that aligns with the specific needs of its target audience. This project empowers local businesses to establish a strong online presence without the constraints of high subscription fees or technical barriers.

Chapter 2

Requirements

In this section, I will describe the key features and actions that are essential for the successful implementation of the e-commerce platform. Each feature will be evaluated in terms of its complexity (low, medium, high) and assigned a priority level (with 1 being the highest priority). Additionally, I will distinguish between "must-have" features that are critical for the project's core functionality and "want-to-have" features that can be implemented later as enhancements.

Product Browsing and Search Functionality

One of the most important actions is allowing users to browse and search for products by category or specific keywords. The system will fetch product data from a database and render it dynamically on the front end using React. This is a key feature that contributes to an intuitive and efficient shopping experience.

- **Feature:** Product browsing and search functionality.
- **Complexity:** Medium (due to dynamic data rendering and search optimization).
- **Priority:** 1 (must-have).

Shopping Cart

A shopping cart function is essential for allowing users to manage their selected items before checking out. Users must be able to add and remove products, adjust quantities, and view the total cost. This functionality will be managed on the server side to ensure data consistency and security.

- **Feature:** Shopping cart with item management.
- **Complexity:** Low (standard functionality with established patterns).
- **Priority:** 2 (must-have).

Secure Checkout and Payment Processing

One of the most critical actions is providing a secure checkout process, including form validation, shipping and billing addresses, and various payment options (e.g., PayPal [19], credit cards). Payment data must be encrypted, and the system must comply with the Payment Card Industry Data Security Standard (PCI-DSS) [20] to protect users' sensitive information.

- **Feature:** Secure checkout with payment processing and encryption.
- **Complexity:** High (due to security compliance and integration with payment gateways).
- **Priority:** 1 (must-have).

Product Detail Page

Each product needs a detailed page where users can view relevant information, such as price, availability, and specifications. This feature ensures customers can make informed decisions before adding items to their cart.

- **Feature:** Product detail page with product information.
- **Complexity:** Low (straightforward data display).
- **Priority:** 3 (must-have).

User Authentication and Authorization

While many e-commerce sites allow users to browse and shop without requiring authentication, *ShopIT* will require users to create accounts before completing a purchase. This requirement is not strictly for security purposes but rather to provide a personalized shopping experience and enable features such as order history, saved addresses, and faster checkouts for repeat customers.

JWT-based authentication will be used for managing user sessions, as it allows for lightweight, stateless session management [13]. With a focus on usability, the tokens will be configured to expire after a reasonable time and can be refreshed as needed to balance convenience with security.

- **Feature:** JWT-based authentication and role-based authorization.
- **Complexity:** Medium (due to security considerations).
- **Priority:** 4 (must-have).

Explanation of Complexity Levels:

The complexity levels are assigned based on the estimated effort required to implement each feature, considering factors like security, integration, and user experience design. Secure checkout and payment processing are rated high due to the need for compliance with security standards and integration with external payment systems.

Additional Features

The following features are considered "want-to-have" and can be implemented to enhance the user experience in later phases of development:

- **Advanced Product Filtering:** Allows users to filter products by additional criteria such as price, brand, and customer reviews.
- **Customer Reviews:** Enables users to leave reviews and ratings for products.
- **Wish List:** Allows users to save products to a wish list for future reference.

Chapter 3

Architecture

The architecture of *ShopIT* is structured using a three-tier design, consisting of the presentation layer (front end), the business logic layer (back end), and the data layer (database). Each layer is essential to ensure secure and efficient communication between the client and the server.

3.1 Client-Side (Front End)

The front end will be built using React [11], a JavaScript library that allows for the creation of dynamic and responsive user interfaces. Customers interact with the platform through this layer, where they can browse products, add items to their shopping cart, and proceed through checkout.

Key front-end components include:

- **Login Page:** Validates user credentials and communicates with the backend for authentication.
- **Product Listing and Detail Pages:** Display products fetched from the server with detailed information.
- **Shopping Cart:** Manages items selected by the user before final checkout.

The interaction between the front-end layer and the rest of the system is illustrated in Figure 3.1. This figure highlights how the front-end components communicate with the back-end APIs and database for data retrieval and processing.

3.2 Server-Side (Back End)

The back end will be developed using Spring Boot with Java [21], which handles the core business logic. It processes user requests, validates inputs, and communicates with the database.

Key back-end services:

- **User Authentication:** Manages user login and role-based access control through JWT.
- **Product Retrieval:** Handles product data and serves it to the front end via API calls.
- **Cart Retrieval:** Handles cart data and serves it to the front end via API calls.
- **Order Management:** Processes order details, including payment validation and shipping status.

As shown in Figure 3.1, the back-end APIs interact with the database to fetch, validate, and process information. These APIs serve as the core communication layer, enabling seamless data transfer between the front-end components and the MongoDB database.

For detailed API endpoints and descriptions, refer to Chapter 4.

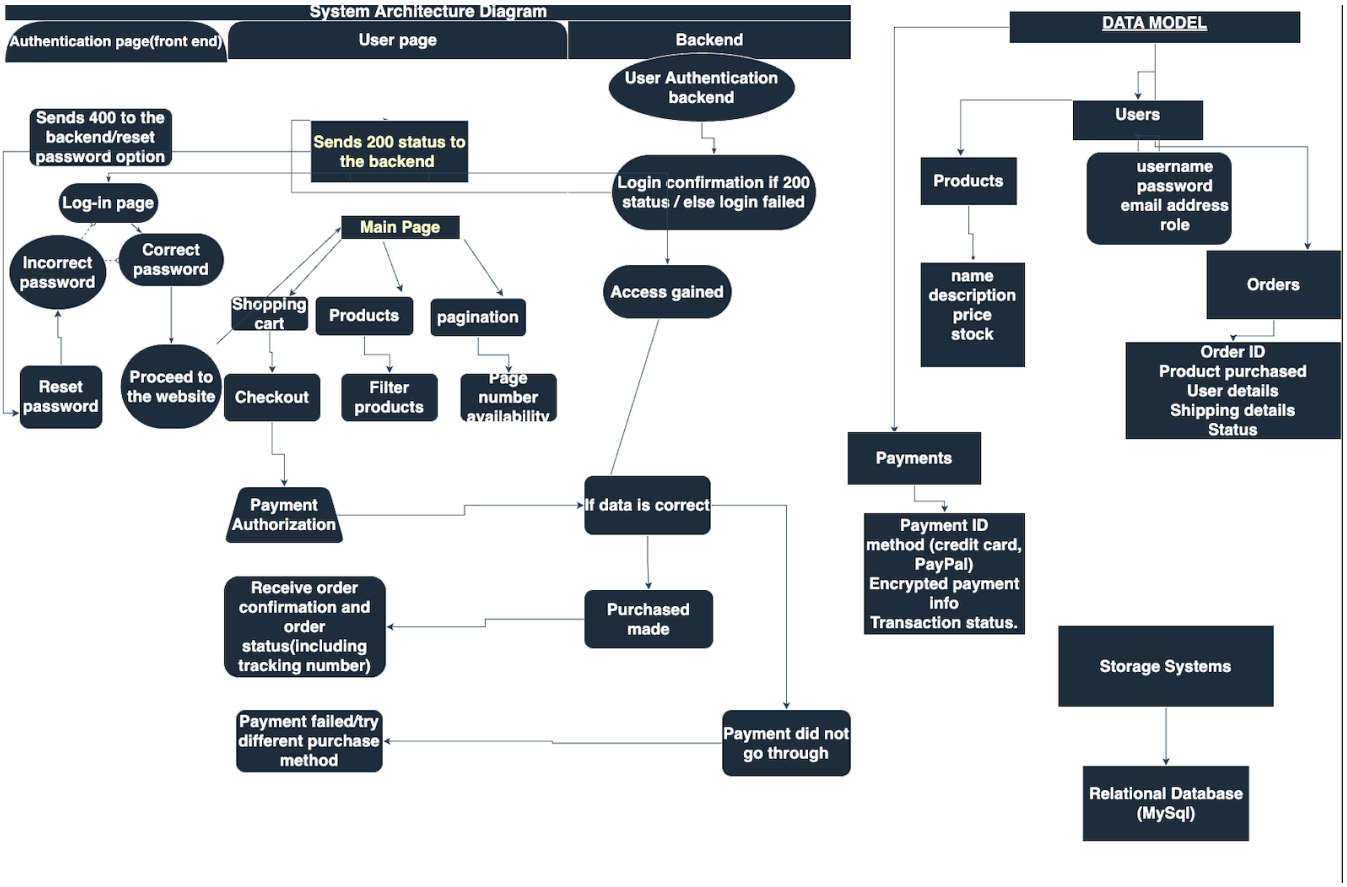

Figure 3.1: System Architecture Diagram showing interactions between the front end, back end, and data model.

Chapter 4

Interfaces

The major interactions customers will have when browsing, buying, and managing their accounts are represented by various screens. Every interface has been designed with safety and ease of use in mind to ensure a smooth shopping experience for both customers and administrators. Additionally, this section outlines the APIs that define the backend operations, such as order management, user authentication, and product retrieval.

4.1 User Interfaces

4.1.1 Login and Registration Page

Users can log into their accounts or create new ones. After successful authentication, users are redirected to the product listing page. Figure 4.1 shows the login page interface. The product listing page will be the first that appears to customers, the login/registration is optional. However, Customers will be asked to log in or register before placing an order.

4.1.2 Product Listing Page

Displays all available products, categorized into different sections. Includes a search bar and filter options for easy navigation. See Figure 4.2.

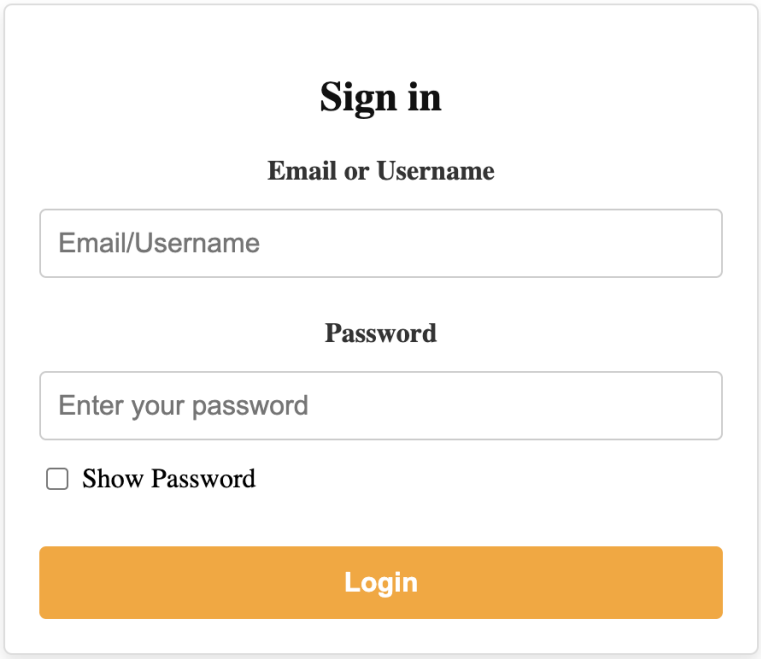

Figure 4.1: Login and Registration Page

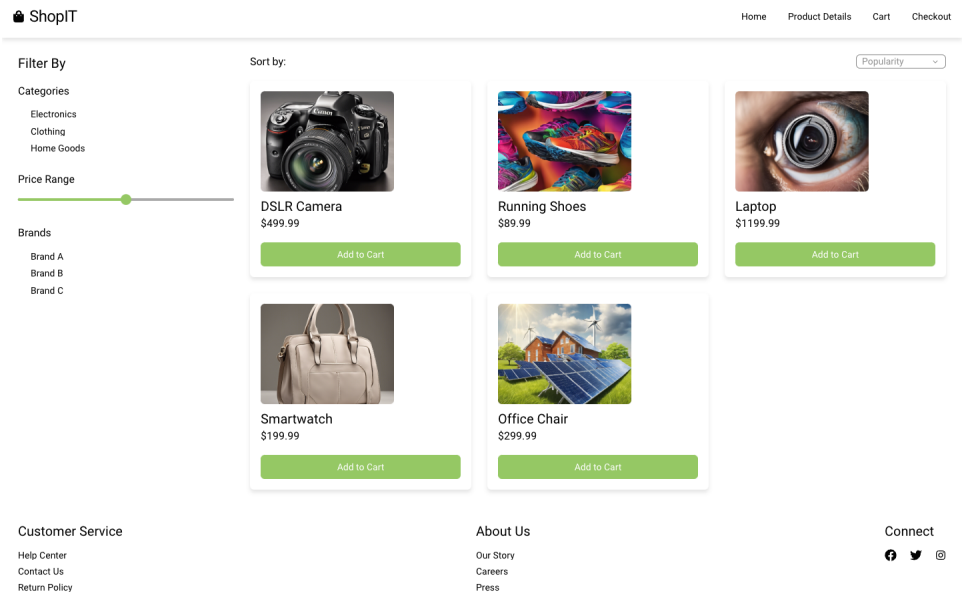

Figure 4.2: Product Listing Page with Search and Filter Options

4.1.3 Product Detail Page

Provides detailed information about a selected product, including price, availability, specifications, and customer reviews. Refer to Figure 4.3.

Nolita Large Nubuck Hobo Shoulder Bag

The Nolita hobo bag looks relaxed yet refined in supple nubuck with structured edges that define its shape. It features an adjustable shoulder strap with polished buckles and a large open interior for stashing everything you need.

Price: $59.99

Figure 4.3: Product Detail Page

4.1.4 Shopping Cart Page

Allows users to manage their selected items, adjust quantities, and view the total price before proceeding to checkout. See Figure 4.4.

Cart for User: test

Total Items: 3

Total Price: $199.97

Nolita Large Nubuck Hobo Shoulder Bag

Quantity: 2

Price per Item: $59.99

Total for this Item: $119.98

Leather Purse

Quantity: 1

Price per Item: $79.99

Total for this Item: $79.99

Figure 4.4: Shopping Cart Page

4.1.5 Cart drop down menu

Allows users to view what they have stored in the cart without navigating to cart page. Additional featured are available, such as view item quantity, price, remove, and checkout button. See Figure 4.5

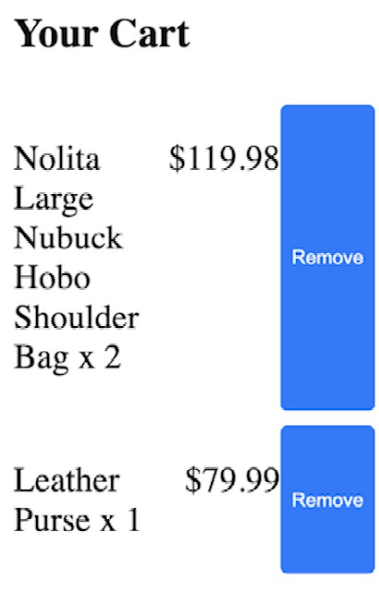

Figure 4.5: Cart drop-down menu

4.2 Backend APIs

The backend APIs in *ShopIT* enable communication between the frontend and backend layers of the system. Each API is designed with a specific purpose, supporting functionalities such as user authentication, product retrieval, and order management. All APIs follow RESTful principles to ensure scalability, reliability, and maintainability.

4.2.1 Authentication API

The Authentication API is responsible for user login and token issuance. It validates user credentials and generates a JSON Web Token (JWT) for authenticated users. This token is used to authorize subsequent requests to secure endpoints.

- **Purpose:** To securely authenticate users and provide token-based session management.
- **Endpoint:** `/api/auth`
- **Method:** POST
- **Usage:** Invoked when a user logs in via the login page. The frontend sends the `email` and `password` to this endpoint.
- **Error Handling:** Returns a `401 Unauthorized` error for invalid credentials.
- **Security:** Ensures that user credentials are never exposed, and only secure tokens are returned.

The detailed structure of this API is presented in **Table 4.1**.

Endpoint	`/api/auth`
Method	POST
Description	Handles user login and JWT token-based authentication.
Parameters	`email` (string), `password` (string)
Returns	`token` (string), `userRole` (string)
Error Codes	401 Unauthorized - Incorrect login credentials

Table 4.1: Authentication API Endpoint

4.2.2 Product Retrieval API

The Product Retrieval API is designed to fetch and display products based on user-defined search and filter criteria. This API ensures that customers can easily explore available products.

- **Purpose:** To retrieve and display product information based on search queries and filters.

- **Endpoint:** `/api/products`
- **Method:** GET
- **Usage:** Invoked when a user performs a search or navigates through product categories on the frontend. Accepts optional parameters like `search`, `category`, and `page`.
- **Error Handling:** Returns a `500 Internal Server Error` if the product data cannot be retrieved.
- **Performance:** Optimized for handling paginated requests to support a large inventory.

The detailed structure of this API is presented in **Table 4.2**.

Endpoint	`/api/products`
Method	GET
Description	Retrieves and displays products based on search and filter criteria.
Parameters	`search` (string), `category` (string), `page` (integer)
Returns	`products` (array), `totalPages` (integer)
Error Codes	500 Internal Server Error - Unable to retrieve products

Table 4.2: Product Retrieval API Endpoint

4.2.3 Product Detail API

The Product Detail API provides in-depth information about a specific product when selected by the user.

- **Purpose:** To deliver detailed information about a single product.
- **Endpoint:** `/api/products/:productId`
- **Method:** GET

- **Usage:** Invoked when a user clicks on a product from the product listing page. Requires the `productId` as a parameter.
- **Error Handling:** Returns a `404 Not Found` error if the specified product does not exist.
- **Data Integrity:** Ensures accurate and up-to-date information is retrieved from the database.

The detailed structure of this API is presented in **Table 4.3**.

Endpoint	`/api/products/:productId`
Method	GET
Description	Retrieves detailed information for a specific product.
Parameters	`productId` (string)
Returns	`product` (object)
Error Codes	404 Not Found - Product does not exist

Table 4.3: Product Detail API Endpoint

4.2.4 Order Processing API

The Order Processing API handles all operations related to customer orders, including payment and order confirmation.

- **Purpose:** To process customer orders and manage order-related information.
- **Endpoint:** `/api/orders`
- **Method:** POST
- **Usage:** Invoked during the checkout process. The frontend sends the `userId`, `cartItems`, and `paymentInfo` to this endpoint.
- **Error Handling:** Returns a `400 Bad Request` error for invalid payment information or incomplete order details.

- **Security:** Ensures that sensitive payment information is encrypted and complies with PCI-DSS standards.

The detailed structure of this API is presented in **Table 4.4**.

Endpoint	`/api/orders`
Method	POST
Description	Processes the data of each order and sends it to the backend.
Parameters	`userId` (string), `cartItems` (array), `paymentInfo` (object)
Returns	`orderStatus` (string), `trackingNumber` (string)
Error Codes	400 Bad Request - Incorrect payment information

Table 4.4: Order Processing API Endpoint

4.2.5 Design Choices

The APIs are designed to be RESTful to ensure scalability and ease of integration with the front end. RESTful APIs provide a standardized way of communication, making it easier to maintain and extend the system [12].

Chapter 5

Data Model

The data model for *ShopIT* is designed to efficiently store and manage all necessary information while maintaining data integrity and security.

5.1 Data Layer (Database and Data Model)

The data model consists of several core entities, stored in a MySql database [18]. MySQL is a database that stores data in tables and enforces rules between data fields. It's based on structured query language (SQL), a common programming language for accessing databases The data model consists of several core entities, stored in a MongoDB database [16]. MongoDB is a NoSQL, document-oriented database that stores data in JSON-like documents, providing flexibility and scalability.

5.1.1 Design Choices

MySQL was chosen for its reliability, strong relational capabilities, and support for ACID-compliant transactions. These features make it well-suited for applications like e-commerce platforms, where data consistency, integrity, and transactional safety are crucial.

5.1.2 Data Storage

All sensitive data, such as user passwords and payment information, will be securely stored on the server side. Passwords will be hashed using algorithms like bcrypt [22], and payment details will be encrypted to comply with Payment Card Industry Data Security Standard (PCI-DSS) standards [20].

Client-side storage is limited to JWT tokens stored in secure HTTP-only cookies to manage user sessions. No sensitive data is stored on the client side to prevent security vulnerabilities.

5.1.3 Data Model Overview

5.1.4 MySQL Justification for E-Commerce

The relational data model for the *ShopIT* application is based on structured tables representing core entities, ensuring data consistency, integrity, and efficient handling of transactions. The entities and their roles include:

- **Users Table:** Stores user information such as credentials, roles, and profile details.
- **Products Table:** Contains product details, including name, description, price, and stock levels.
- **Orders Table:** Records order transactions, linking users to purchased products.
- **Order Items Table:** Tracks the individual items within an order to normalize data storage.
- **Payments Table:** Stores payment information and transaction details.
- **Cart Table:** Manages the shopping cart for each user, including items and quantities.

These tables are interconnected through primary and foreign keys to enforce referential integrity and eliminate data anomalies. The relational structure is particularly advantageous for maintaining transactional consistency in e-commerce scenarios.

Why MySQL for E-Commerce?

MySQL was chosen for *ShopIT* because of its robust support for ACID-compliant transactions, relational modeling, and scalability, all of which are critical for an e-commerce platform that requires secure and reliable data management.

- **Transactional Integrity:** E-commerce platforms require robust transaction handling to prevent issues such as double charges or inventory inconsistencies. MySQL ensures transactional integrity with its ACID compliance, making it ideal for handling critical operations like payment processing and inventory updates [5].
- **Relational Modeling:** MySQL's table-based schema is well-suited for the structured relationships between users, products, orders, and payments. This structure supports complex queries and maintains data integrity through foreign key constraints [8].
- **Performance Optimization:** MySQL provides features like indexing, caching, and query optimization, which enhance performance for read-heavy operations such as product browsing, while efficiently managing write-heavy tasks like updating stock levels [7].
- **Concurrency Control:** E-commerce systems often experience high-concurrency scenarios where multiple users interact with the same data (e.g., purchasing the same product). MySQL's support for row-level locking and isolation levels helps prevent race conditions and maintains data consistency [6].
- **Proven Use Cases:** MySQL powers many successful e-commerce platforms, including Shopify and WordPress-based stores, demonstrating its reliability and scalability for handling high traffic and large datasets [10].

- **Scalability and Flexibility:** MySQL supports vertical and horizontal scaling, allowing *ShopIT* to grow as user demand increases. Features like replication and sharding enable the database to handle larger workloads efficiently [9].

Limitations of NoSQL (MongoDB) for E-Commerce

While NoSQL databases like MongoDB offer flexibility, they lack some critical features that make MySQL a better fit for e-commerce:

- **Transactional Safety:** MongoDB's document-based model can encounter race conditions during concurrent writes, such as when multiple users purchase the same product. This can lead to data inconsistencies, which are mitigated in MySQL through ACID compliance [5].
- **Complex Relationships:** Relational databases like MySQL naturally handle complex relationships and queries using joins and foreign keys. While MongoDB can model relationships, it requires additional effort and compromises query simplicity [4].
- **Concurrency Challenges:** In high-concurrency scenarios, MongoDB's eventual consistency model may lead to temporary data inconsistencies, which are unacceptable in e-commerce transactions [15].

MySQL's combination of reliability, performance, and flexibility makes it an optimal choice for *ShopIT*, ensuring that the platform can manage the complexities of e-commerce operations while scaling with demand.

An illustration of the data model is provided in Chapter 5. The key collections in MongoDB include:

- **Users Collection:** Stores user documents containing credentials and roles.
- **Products Collection:** Contains product documents with details and inventory information.

- **Orders Collection:** Records order documents with purchase transactions and statuses.
- **Cart Collection:** Stores an array of items, in addition to count (items in cart), and total(total price of items in cart).
- **Payments Collection:** Manages payment documents with transaction details.

5.2 Design Choices and Alternatives

5.2.1 MongoDB vs. Relational Databases

While MongoDB is a popular choice for its flexibility and scalability, relational databases like MySQL offer specific advantages for e-commerce platforms due to the nature of transactional operations and the structured relationships between entities such as users, orders, and products.

Advantages of MongoDB

MongoDB offers several benefits in scenarios requiring flexibility and scalability:

- **Schema Flexibility:** MongoDB's schema-less design allows for dynamic and schema-free data storage, making it easier to adapt to changing requirements.
- **Horizontal Scalability:** MongoDB supports sharding, enabling horizontal scaling for applications with large volumes of data or high traffic [17].
- **Optimized for Read-Heavy Workloads:** Its ability to store and retrieve nested and hierarchical data makes MongoDB effective for analytics and read-heavy applications.

Reasons to Choose MySQL

Relational databases like MySQL provide critical features for e-commerce platforms:

- **Transactional Integrity:** MySQL ensures strong ACID compliance, which is essential for e-commerce operations requiring consistency in transactions such as inventory updates and payments [5].

- **Relational Data Modeling:** MySQL's structured schema using foreign keys ensures data consistency and integrity, making it ideal for handling complex relationships between entities such as products, users, and orders [8].

- **Concurrency Control:** MySQL supports row-level locking and isolation levels to prevent race conditions, ensuring data integrity when multiple users interact with the same data simultaneously (e.g., purchasing the same product) [6].

- **Query Optimization:** MySQL provides indexing and advanced query optimization, enhancing the performance of read-heavy operations such as product searches and filtering [7].

Final Decision: Why MySQL?

For *ShopIT*, MySQL was selected due to its strong support for transactional integrity, relational modeling, and mature ecosystem, which align better with the operational requirements of e-commerce platforms. While MongoDB is a viable option for scenarios demanding schema flexibility and horizontal scalability, the critical need for ACID compliance and reliable relational modeling makes MySQL the better choice for this project.

5.2.2 JWT vs. Session-Based Authentication

JWT was chosen over traditional session-based authentication due to its stateless nature, which improves scalability and reduces server load [14]. JWT tokens allow for secure transmission of user claims, enabling efficient authentication across distributed systems.

5.2.3 RESTful API Design

The APIs are designed to be RESTful to provide a standardized and scalable way for the front end and back end to communicate. RESTful APIs are stateless, cacheable, and provide uniform interfaces [12].

5.3 Data Model Illustration

An illustration of the schema is provided in Figure 5.1, showing the relationships between the key collections in MongoDB.

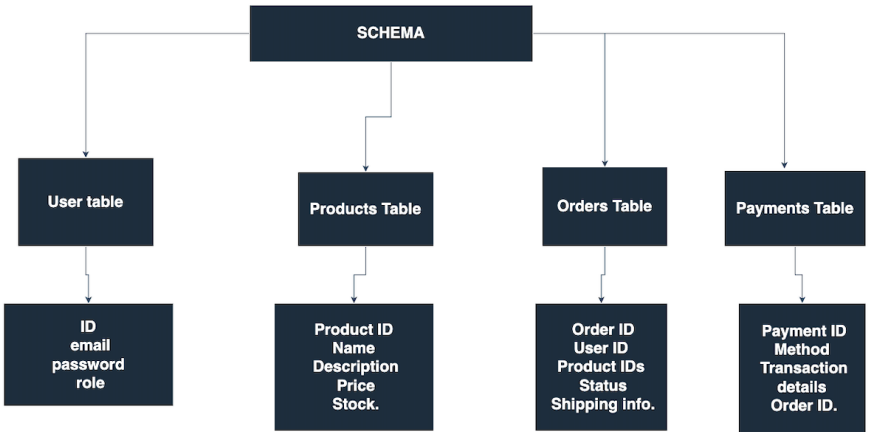

Figure 5.1: Data Model Illustration for *ShopIT* using MongoDB

5.4 Collections and Relationships

5.4.1 Users Collection

Each user document contains:

- **Fields:** `_id` (ObjectId), `email`, `passwordHash`, `role`, `firstName`, `lastName`, `address(object)`
- **Description:** Stores user credentials and roles.

5.4.2 Products Collection

Each product document includes:

- **Fields:** `_id` (ObjectId), `name`, `description`, `price`, `stock`, `category`, `attributes`
- **Description:** Contains product details and inventory information.

5.4.3 Orders Collection

Each order document contains:

- **Fields:** `_id` (ObjectId), `userId` (reference to Users), `orderDate`, `totalAmount`, `status`, `items` (array of products)
- **Description:** Records purchase transactions and order statuses.

5.4.4 Cart Collection

Each cart document includes:

- **Fields:** `_id` (ObjectId), `userId`, `items(Array)`, `count`, `total`
- **Description:** Manages stored items in cart.

5.4.5 Payments Collection

Each payment document includes:

- **Fields:** `_id` (ObjectId), `orderId` (reference to Orders), `paymentMethod`, `transactionId`, `paymentDate`
- **Description:** Manages payment transactions and methods.

5.5 Data Storage and Security

All sensitive data is stored on the server side with appropriate encryption and hashing mechanisms. Passwords are hashed using algorithms like bcrypt [22], and payment information is encrypted in compliance with PCI-DSS standards [20].

Client-side storage is limited to JWT tokens stored in secure HTTP-only cookies to manage user sessions. No sensitive data is stored on the client side to prevent security vulnerabilities.

Chapter 6

Testing Plan

To ensure that the e-commerce project meets the requirements outlined in Chapter 2, I will implement a comprehensive strategy combining both automated testing and user testing. This plan covers unit testing, integration testing, and user interface (UI) testing to validate the functionality, performance, and usability of the website.

6.1 Secure Checkout and Payment Processing

6.1.1 Automated Testing

- **Framework:** JUnit for back-end testing.
- **Unit Testing:** Validate that the payment processing service correctly handles transactions, encrypts data, and complies with PCI-DSS requirements.
- **Integration Testing:** Test the interaction between the payment service and external payment gateways like PayPal [19].

6.1.2 User Testing

- **Protocol:** Users will perform a complete purchase using test payment information.

- **Success Criteria:** Transactions are processed securely, and users receive confirmation of their orders without any data breaches.

6.2 Product Browsing and Search Functionality

6.2.1 Automated Testing

- **Framework:** JUnit for back-end testing, Selenium for UI testing.
- **Unit Testing:** Ensure that the product retrieval service correctly fetches and filters products based on search criteria.
- **UI Testing:** Automate search queries and validate that the results match expected outputs.

6.2.2 User Testing

- **Protocol:** Users will search for specific products and use filters.
- **Success Criteria:** The search and filtering functions return accurate results, and the interface is intuitive.

6.3 Shopping Cart Functionality

6.3.1 Automated Testing

- **Framework:** JUnit and Selenium.
- **Unit Testing:** Verify that items can be added, updated, and removed from the cart on the server side.
- **UI Testing:** Simulate user interactions with the cart and ensure the UI reflects the correct state.

6.3.2 User Testing

- **Protocol:** Users will add specific items to the cart, adjust quantities, and verify the total price matches expected values.
- **Success Criteria:** Cart operations work seamlessly, and pricing calculations are accurate.

6.4 User Authentication and Authorization

6.4.1 Automated Testing

- **Framework:** JUnit for unit and integration testing.
- **Unit Testing:** Test the authentication service to ensure JWT tokens are issued correctly and invalid credentials are rejected.
- **Integration Testing:** Verify that role-based access controls are enforced throughout the application.

6.4.2 User Testing

- **Protocol:** Users will attempt to log in with both valid and invalid credentials and access different sections based on their roles.
- **Success Criteria:** Authentication works correctly, and access controls are properly enforced.

Chapter 7

Prototype

A functional prototype of *ShopIT* has been developed to demonstrate the core features. The prototype includes:

- **User Interface:** Interactive pages for product browsing, detailed views, shopping cart, and checkout.
- **Backend Services:** API calls to retrieve products, cart, and registration.
- **Security Measures:** Implementation of JWT authentication and HTTPS.

The following figures provide an overview of the prototype design:

The product browsing page (Figure 7.1) serves as the starting point for users to explore available products. This page includes dynamic product listings fetched from the backend and features intuitive navigation.

Once a user selects a product, the system displays its details on the product detail page (Figure 7.2). Here, users can view key information such as the product description, price, and stock availability.

To promote easy management of selected items, the shopping cart page (Figure 7.3) allows users to adjust quantities, remove items, and view the total cost before proceeding to checkout.

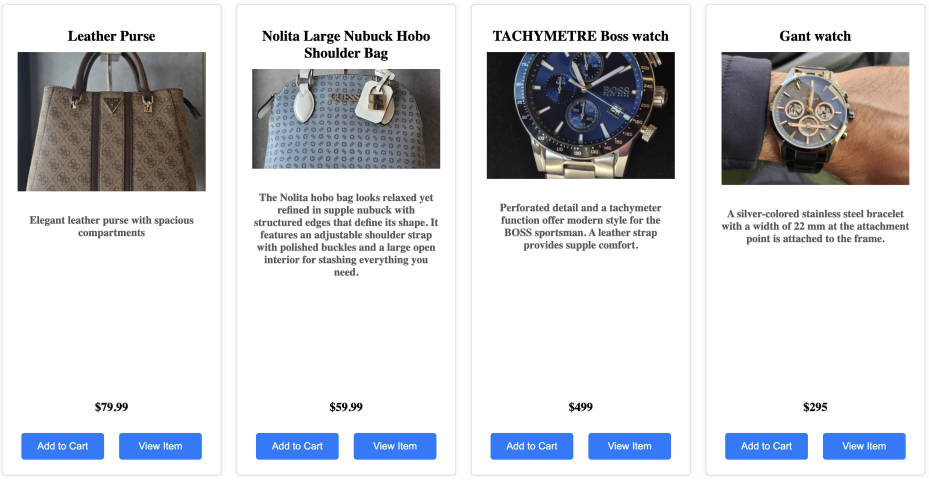

Figure 7.1: Prototype Screenshot 1: Product Browsing Page

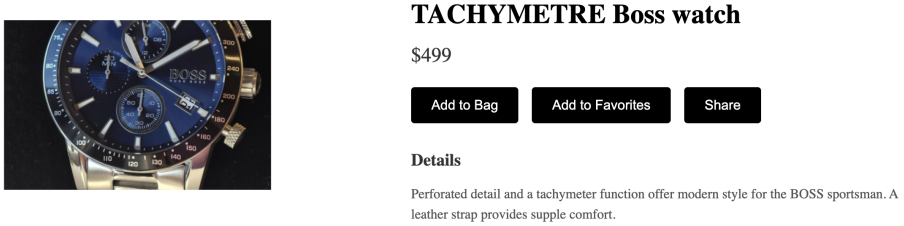

Figure 7.2: Prototype Screenshot 2: Product Detail Page

Figure 7.3: Prototype Screenshot 3: Shopping Cart Page

Finally, the landing page (Figure 7.4) welcomes users to the platform, offering options to log in, register, or start browsing products immediately.

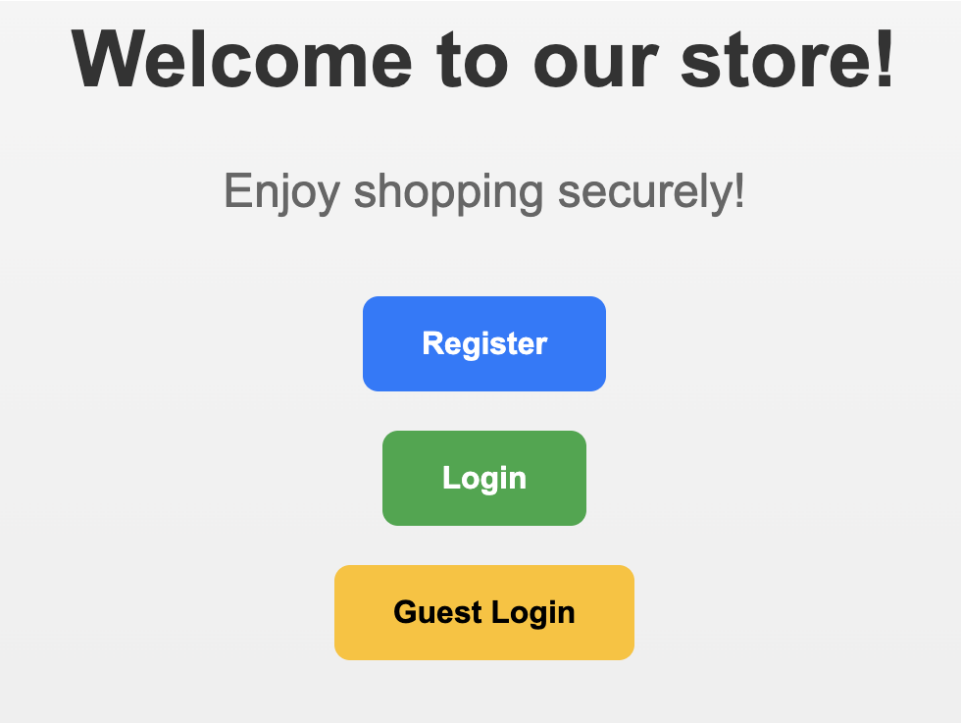

Figure 7.4: Prototype Screenshot 4: Landing Page

User feedback on the prototype has been positive, highlighting the intuitive design and responsiveness of the interface. Areas for improvement include enhancing the search functionality and adding more payment options.

Chapter 8

Timeline

A detailed timeline outlining the project milestones:

- **Phase 1 (September):** *Project Idea and Planning.* During this phase, I conceptualized the project, decided on the technologies to use, and finalized the system design. This included selecting Java with Spring Boot for the back end, React for the front end, and MongoDB for the database.
- **Phase 2 (October - November):** *Initial Development and Core Functionalities.* In this phase, I began developing core back-end and front-end features. Key milestones included implementing JWT authentication, designing and developing the product retrieval and user login endpoints, and building a functional user interface. CRUD operations (add, delete, update) for items were also completed. The current prototype demonstrates these functionalities successfully.
- **Phase 3 (December):** *Search and Filtering Features.* This phase will focus on enhancing the user experience by adding a search bar and data filtering functionalities to help users locate products efficiently. Additionally, a new feature will be developed, allowing customers to add items to a wishlist for future reference. Testing and refinement of the existing APIs, such as product retrieval and login, will continue alongside these enhancements. Minor errors in the prototype will also be addressed.

- **Phase 4 (January - February):** *Implementation of Advanced Features and Payment Processing.* Building on the foundation of the tested APIs, I will implement the order management and payment processing system, ensuring compliance with PCI-DSS standards. Advanced features such as order tracking and detailed user account management will also be added during this phase.

- **Phase 5 (March):** *User Feedback and Testing.* Comprehensive user testing will be conducted during this phase. The goal will be to gather feedback from real users to identify usability issues and refine the application accordingly. Additional automated testing, such as load testing and security testing, will also be performed to ensure reliability.

- **Phase 6 (April - May):** *Final Integration and Deployment Preparation.* This final phase will focus on integrating all components of the system into a seamless experience. Performance optimizations and bug fixes will be completed, and the application will be prepared for deployment. A final review will ensure all requirements are met and the system is ready for launch.

Chapter 9

Summary

This project focuses on building a secure and user-friendly e-commerce website, *ShopIT*, designed to help local businesses expand their reach online. The platform allows users to browse products, add items to their shopping cart, and complete purchases using a secure checkout process. Key features include product search functionality, user authentication using JWT, and compliance with industry security standards like PCI-DSS to protect customer payment information.

The website is built using Java and Spring Boot for the back end, which handles tasks such as user login, product management, and order processing. The front end, created using React and JavaScript, ensures the user interface is responsive and easy to navigate. Security is a top priority in this project, with encryption and secure data handling measures in place to protect sensitive customer information like credit card details.

In addition to the core functionalities, the platform will support enhancements such as advanced product filtering, customer reviews, and wish lists, which will be implemented later. The project is designed to help small businesses grow by creating a reliable and secure online store that users can trust and easily interact with.

References

[1] Adobe Inc. *Magento Commerce*. https://magento.com. Version 2.4.8. Accessed: 2024-12-12. Oct. 8, 2024.

[2] Automattic Inc. *WooCommerce*. https://woocommerce.com. Version 9.4.3. Accessed: 2024-12-12. Dec. 3, 2024.

[3] BigCommerce Pty. Ltd. *BigCommerce*. https://www.bigcommerce.com. Accessed: 2024-11-22. Nov. 22, 2024.

[4] IBM Corporation. *RDBMS vs NoSQL Databases: A Detailed Comparison*. https://www.ibm.com/cloud/blog/rdbms-vs-nosql. Accessed: 2024-12-15. 2024.

[5] Oracle Corporation. *ACID Compliance in MySQL*. https://dev.mysql.com/doc/refman/8.0/en/acid.html. Version 8.0.34. Accessed: 2024-12-15. 2024.

[6] Oracle Corporation. *Concurrency Control in MySQL*. https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-model.html. Version 8.0.34. Accessed: 2024-12-15. 2024.

[7] Oracle Corporation. *MySQL Performance Optimization and Best Practices*. https://dev.mysql.com/doc/refman/8.0/en/performance.html. Version 8.0.34. Accessed: 2024-12-15. 2024.

[8] Oracle Corporation. *Relational Database Design and Structure*. https://dev.mysql.com/doc/refman/8.0/en/innodb-data-model.html. Version 8.0.34. Accessed: 2024-12-15. 2024.

[9] Oracle Corporation. *Scaling MySQL with Replication and Sharding*. https://dev.mysql.com/doc/refman/8.0/en/replication.html. Version 8.0.34. Accessed: 2024-12-15. 2024.

[10] Oracle Corporation. *Use Cases for MySQL in Modern Applications*. https://www.mysql.com/why-mysql/. Version 8.0.34. Accessed: 2024-12-15. 2024.

[11] Facebook, Inc. *React - A JavaScript library for building user interfaces*. https://react.dev/. Version 19.0. Accessed: 2024-12-12. Dec. 12, 2024.

[12] R. T. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Accessed: 2024-11-22. PhD thesis. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[13] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519, https://tools.ietf.org/html/rfc7519. Accessed: 2024-11-22. 2015.

[14] J. Martin. *JWTs vs Sessions - What's the Right Authentication Method?* Accessed: 2024-11-22. 2019. URL: https://auth0.com/blog/jwt-vs-session-cookies/.

[15] Inc. MongoDB. *Limitations of NoSQL Databases*. https://www.mongodb.com/nosql-explained. Version 8.0.4. Accessed: 2024-12-15. 2024.

[16] MongoDB, Inc. *MongoDB*. https://www.mongodb.com. Version 8.0.4. Accessed: 2024-12-12. Dec. 11, 2024.

[17] MongoDB, Inc. *Transactions*. https://www.mongodb.com/docs/manual/core/transactions/. Accessed: 2024-11-22. 2024.

[18] Oracle Corporation. *MySQL*. https://www.mysql.com/. Version 8.0.34. Accessed: 2024-12-12. Dec. 12, 2024.

[19] PayPal Holdings, Inc. *PayPal Developer Documentation*. https://developer.paypal.com/. Accessed: 2024-11-22. Nov. 22, 2024.

[20] PCI Security Standards Council. *Payment Card Industry Data Security Standard v3.2.1*. https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss. Accessed: 2024-11-22. 2018.

[21] Pivotal Software, Inc. *Spring Boot*. https://spring.io/projects/spring-boot. Version 3.2.0. Accessed: 2024-11-23. Nov. 23, 2024.

[22] N. Provos and D. Mazieres. "A Future-Adaptable Password Scheme". In: *USENIX Annual Technical Conference* (1999). Accessed: 2024-11-22. URL: https://www.usenix.org/legacy/events/usenix99/provos.html.

[23] E. Rescorla. *HTTP Over TLS*. RFC 2818, https://tools.ietf.org/html/rfc2818. Accessed: 2024-11-22. 2000.

[24] Shopify Inc. *Shopify*. https://www.shopify.com. Version 3.47.3. Accessed: 2024-11-22. Nov. 22, 2024.

[25] Squarespace, Inc. *Squarespace*. https://www.squarespace.com. Accessed: 2024-11-22. Nov. 22, 2024.