Color Catalogue Digital shop

Capstone Project-2

By Group-6

Names:

Sivabalan.S

Sai Teja.T

Sadhika.K

Sai Sahithi.M

Capstone Evaluator:

Prof. Abhishek Sengupta

Revision Sheet:

S.NO	DATE	AUTHOR	DESCRIPTION
1	8/7/2024	S.SIVABALAN	Back-end,Database
2	9/7/2024	K.SADHIKA	Front-End
3	9/7/2024	M.SAI SAHITHI	Front-End,PPT
4	9/7/2024	T.SAI TEJA	Documenation,Testing
5	10/7/2024	TEAM	Final Draft

Table of Contenets

1.Introduction

1.1 Purpose:

The primary purpose of the Color Catalog Digital Shop project is to develop an efficient, user-friendly, and comprehensive web-based application to manage and streamline the operations of an online shop for colors. This system is intended to achieve the following key objectives:

- Improve Shopping Experience
- Enhance User Engagement
- Ensure Data Integrity and Security
- Support Efficient Color Management
- Optimize Shopping Processes
- Facilitate Customer Administration
- Support Agile Development and Deployment

1.2 Scope of the Project:

The scope of the Color Catalog Digital Shop project encompasses the development and deployment of a comprehensive web-based application that covers all aspects of online color shopping. The project includes the following key components and features:

- User Management
- Color Catalog Management
- Shopping Cart Management
- Security and Access Control
- Reporting and Analytics

1.3 Project Overview:

The Color Catalog Digital Shop is a comprehensive web-based application designed to streamline and enhance the shopping experience for customers looking to purchase colors. The system offers a range of features, including user management, color cataloging, shopping cart functionality, search functionality, and detailed reporting. It leverages modern technologies such

as Java/J2EE for backend development, HTML/CSS for frontend design, and SQL for data management. The system is built following Agile methodology and is deployed using containerization (Docker) and Kubernetes for scalability and reliability. The goal is to provide a user-friendly, secure, and efficient platform for both shop administrators and customers.

2. System Requirements:

2.1 Functional Requirements

2.1.1 Login with Username and Password:

- Users must be able to log in to the Color Catalogue Digital Shop using their unique username and password.
- The login process should be secure, using encryption protocols to protect user credentials.

2.1.2 Color Selection Page:

- The color selection page should allow users to browse and select up to three colors.
- Users should be able to view color names.
- Users should be able to add selected colors to a comparison list for further evaluation.

2.1.3 Selected Colors Page:

- The selected colors page should display the colors that the user has chosen.
- Users should be able to compare the selected colors side-by-side.
- The page should show detailed information about each selected color.
- User can logout from the selected color page.

These functional requirements ensure that the Color Catalogue Digital Shop provides a seamless and user-friendly experience for customers to log in, select, and compare colors, enhancing their overall shopping experience.

2.2 Non-Functional Requirements

2.2.1 Security:

- The system must ensure secure authentication and authorization mechanisms to protect user data.
- All sensitive information, including user credentials and personal details, must be encrypted both in transit and at rest.
- Implement HTTPS for secure communication between the client and server.
- Regular security audits and vulnerability assessments should be conducted to identify and mitigate potential threats.
- Access controls and role-based permissions should be enforced to restrict access to sensitive data and functionalities.

2.2.2 Performance:

- The application should provide fast response times for all user interactions, ensuring a smooth and efficient user experience.
- The system must handle concurrent users efficiently, with minimal latency.
- Database queries should be optimized to reduce load times and improve overall performance.
- Implement caching strategies where appropriate to enhance performance and reduce server load.

2.2.3 Usability:

- The user interface must be intuitive and easy to navigate, providing a seamless experience for users of all technical levels.
- Provide clear and concise error messages and validation feedback to guide users through the application.
- Ensure that the application is accessible to users with disabilities, following WCAG (Web Content Accessibility Guidelines) standards.

- Regular usability testing should be conducted to gather feedback and make necessary improvements.

2.2.4 Scalability:

- The system must be designed to handle an increasing number of users and data without compromising performance.
- Utilize scalable infrastructure, such as Kubernetes, to manage containers and scale resources dynamically based on demand.
- Implement load balancing to distribute traffic evenly across multiple servers, ensuring high availability and reliability.
- The architecture should support the addition of new features and functionalities with minimal disruption.

2.2.5 Maintainability:

- The codebase should adhere to clean code principles and follow the SOLID design principles to ensure modularity and readability.
- Comprehensive documentation should be maintained for all aspects of the system, including architecture, design, and implementation details.
- Implement version control practices using Git, with separate branches for development, testing, and production.
- Regular code reviews and refactoring should be conducted to maintain code quality and address technical debt.
- Automated testing and CI/CD pipelines should be in place to streamline the deployment process and ensure continuous integration and delivery.

3.Design

3.1 Architecture

The architecture of the color catalog digital shop is designed to ensure a seamless, efficient, and scalable system. The key components of the architecture include:

1. **Client-Side (Frontend):**
    - **Technologies:** HTML, CSS, JavaScript
    - **Responsibilities:**
        - Rendering the user interface.
        - Handling user interactions, such as logging in, selecting colors, and displaying results.
        - Making API calls to the backend to fetch and display data.
2. **Server-Side (Backend):**
    - **Technologies:** Java/J2EE, Design Patterns (e.g., MVC, Singleton, DAO)
    - **Responsibilities:**
        - Handling business logic and data processing.
        - Managing user authentication and authorization.
        - Serving API endpoints for frontend interactions.
        - Interacting with the database to perform CRUD operations.
3. **Database:**
    - **Technologies:** SQL (e.g., PostgreSQL, MySQL)
    - **Responsibilities:**
        - Storing user data, color catalogue
        - Ensuring data integrity and security.
        - Providing efficient querying mechanisms.
4. **Deployment and Environment:**
    - **Technologies:** Docker, Kubernetes, OpenShift
    - **Responsibilities:**
        - Containerizing the application for consistent deployment across environments.

- Using Kubernetes for container orchestration and scaling.
- Implementing load balancing to distribute traffic and ensure high availability.
- Maintaining different environments (Development, Testing, Production) with proper Git branching strategies.

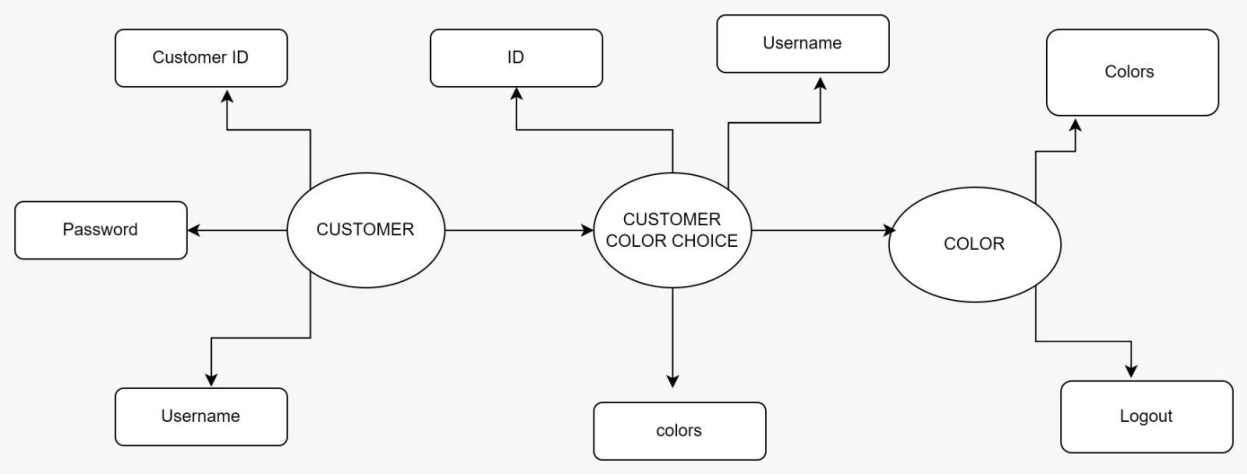
5. **Security and Performance:**
    - Implementing HTTPS for secure communication.
    - Using encryption for sensitive data.
    - Conducting regular performance optimization and security audits.

3.2 Database Design

The database design is crucial for storing and managing the data efficiently. The key tables include:

1. **Users Table:**
    - **Fields:** user_id (Primary Key), username, password (hashed)
    - **Responsibilities:** Storing user credentials and registration details.
2. **Colors Selection Table:**
    - **Fields:** color_id (Primary Key), color_name, username.
    - **Responsibilities:** Storing details of selected colors.

3.2 ER Diagram

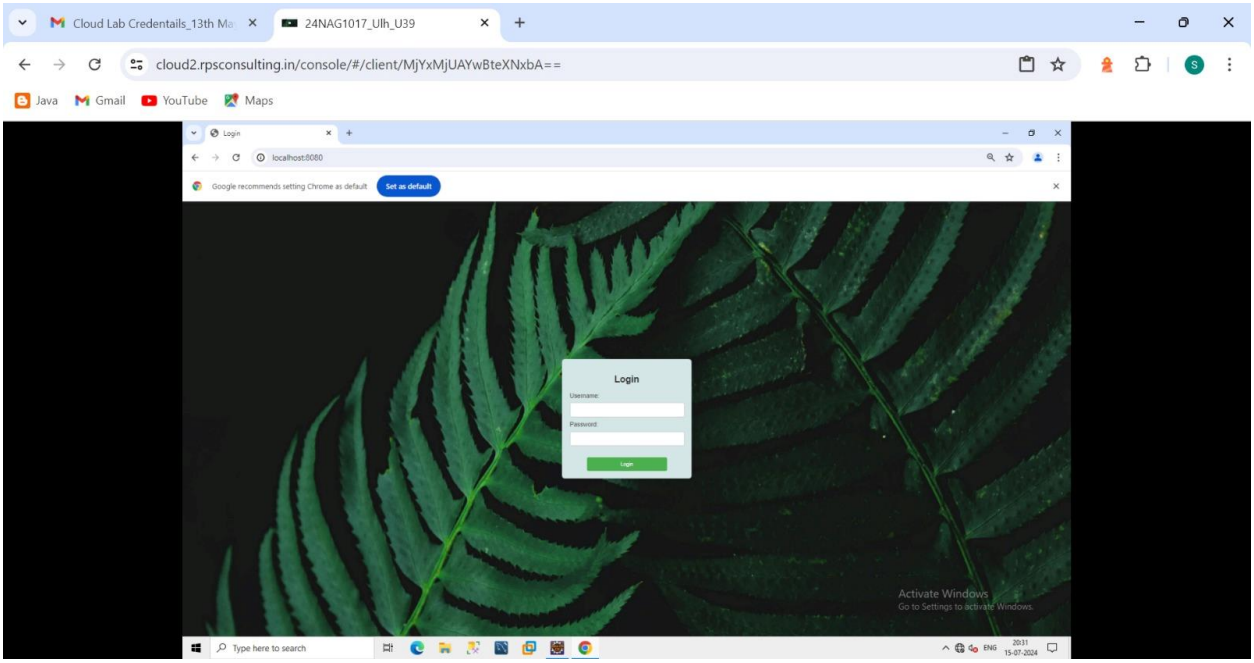


4.Implementation

4.1 Login Page

4.1.1 Username and Password

- **Frontend:**
    - **HTML/CSS:** Create a form with input fields for the username and password, styled for usability and aesthetics.
    - **JavaScript:** Validate the input fields to ensure they are not empty before submitting the form.
- **Backend:**
    - **API Endpoint:** Create a POST endpoint /api/login to handle login requests.
    - **Authentication Logic:** Verify the username and password against the stored hashed passwords in the database.
    - **Response Handling:** Return a success response with a session token or an error message for invalid credentials.



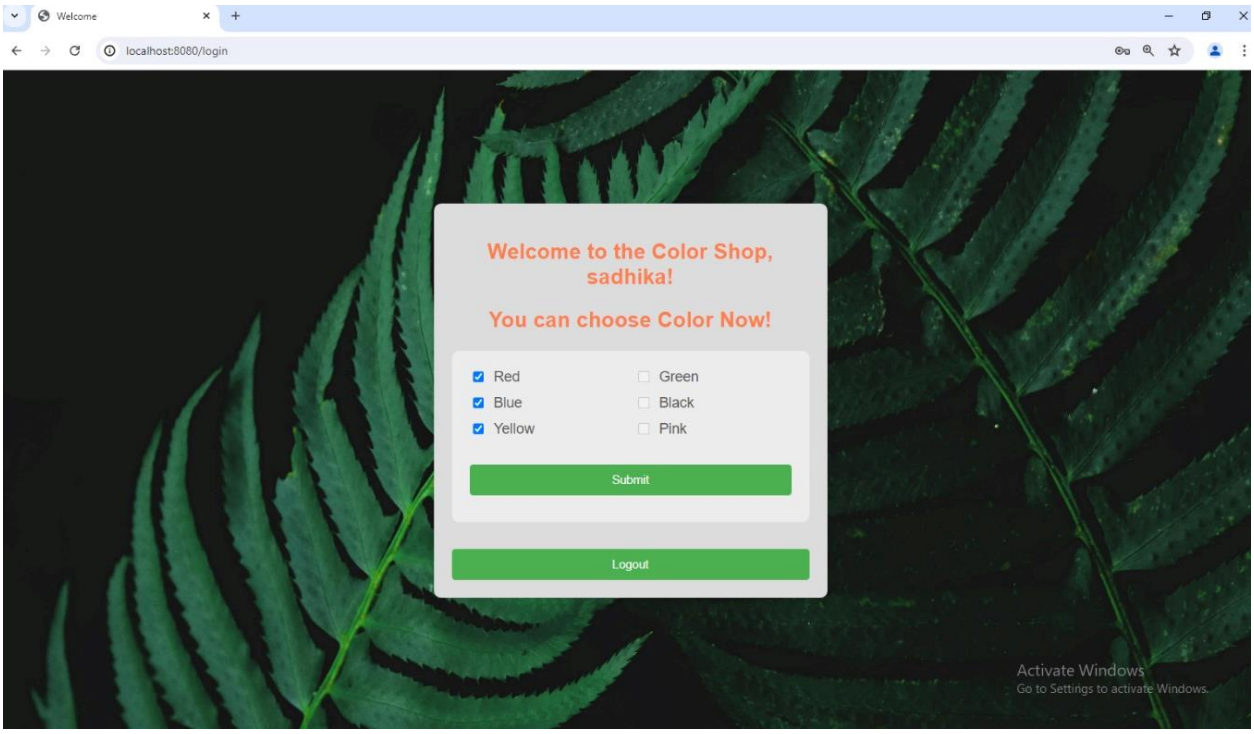
4.2 Color Selection Page

4.2.1 Colors Display

- **Frontend:**
  - **HTML/CSS:** Create a grid or list layout to display available colors, each represented by a color swatch and name.
  - **JavaScript:** Fetch the available colors from the backend API and dynamically populate the color selection grid.
- **Backend:**
  - **API Endpoint:** Create a GET endpoint /api/colors to fetch the list of available colors.
  - **Data Retrieval:** Query the Colors table in the database to get the available colors and send the data to the frontend.

4.2.2 Selection Handling

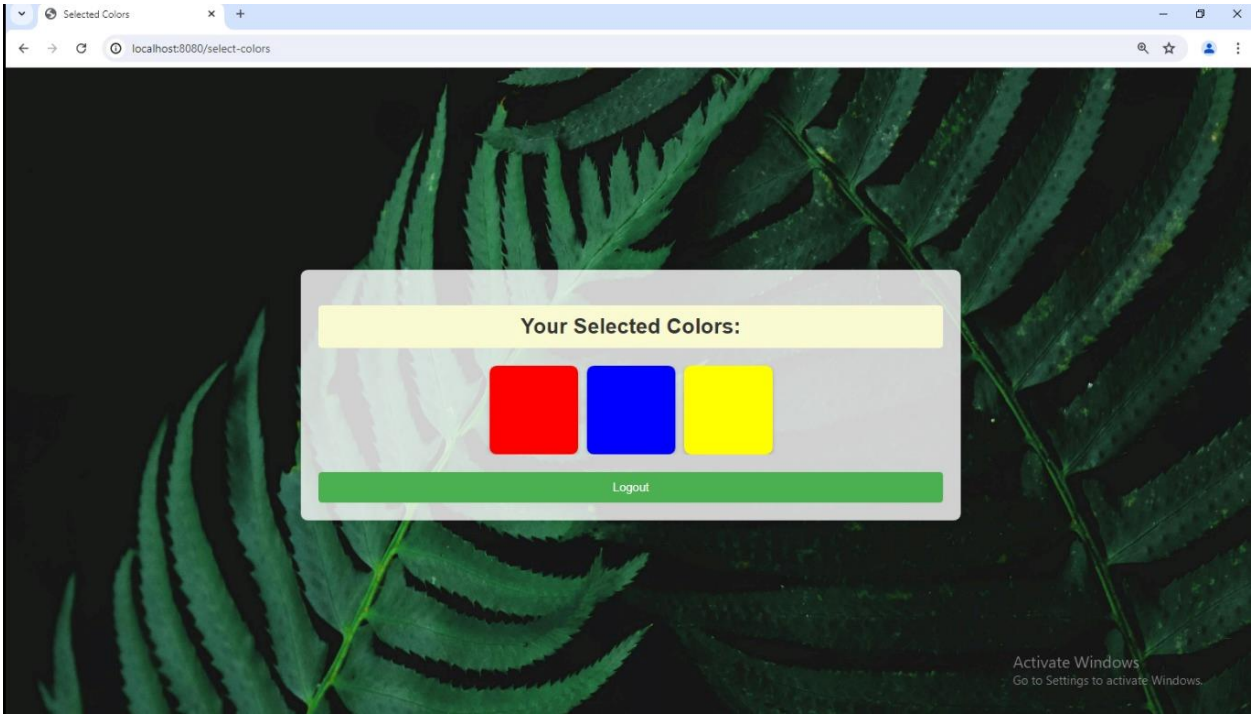
- **Frontend:**
  - **JavaScript:** Allow users to select up to 3 colors by clicking on the color swatches. Highlight the selected colors.
  - **Form Submission:** Create a form to submit the selected colors to the backend.
- **Backend:**
  - **API Endpoint:** Create a POST endpoint /api/selections to handle the submission of selected colors.
  - **Data Handling:** Store the selected colors in the Selections table, linking them to the current user's ID.



4.3 Selected Colors Page

4.3.1 Display Selected Colors

- **Frontend:**
  - **HTML/CSS:** Create a page layout to display the selected colors in a visually appealing manner.
  - **JavaScript:** Fetch the user's selected colors from the backend API and dynamically display them on the page.
- **Backend:**
  - **API Endpoint:** Create a GET endpoint /api/user/selections to fetch the current user's selected colors.
  - **Data Retrieval:** Query the Selections table to get the colors selected by the current user and send the data to the frontend.



The implementation section covers the key functionalities of the color catalog digital shop, including user authentication, color selection, and displaying the selected colors. This structured approach ensures that each part of the application is developed systematically, ensuring a smooth and cohesive user experience.

5.Testing

5.1 Unit Testing

Unit testing ensures that individual components or units of code function correctly in isolation. For the Color catalogue digital shop, unit tests will validate functions such as user authentication, color selection algorithms, and database interactions. Test cases will cover various scenarios to verify the expected behavior of each unit.

5.2 Integration Testing

Integration testing checks how different modules or components work together as a whole. In the context of the system, integration tests will focus on testing interactions between the frontend and backend systems, ensuring smooth communication and data flow. Tests will cover scenarios such as login processes, color category displays, and selected color functionalities.

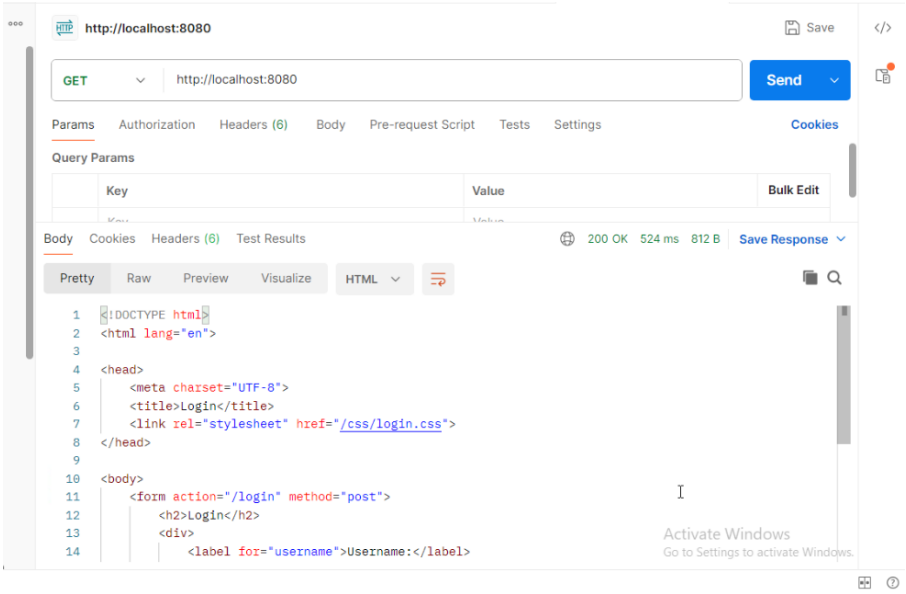
5.3 System Testing

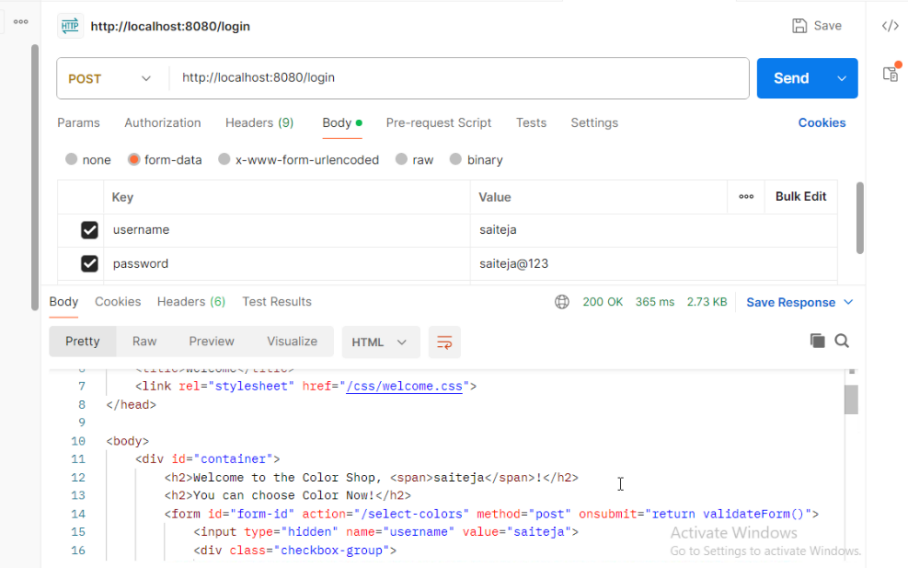
System testing evaluates the complete system against specified requirements. It ensures that the Color catalogue digital shop meets functional and non-functional requirements as a cohesive unit. Tests will validate end-to-end workflows, including user journeys from login to color selection page and selected colors page. Performance, security, and usability aspects will also be assessed during system testing.

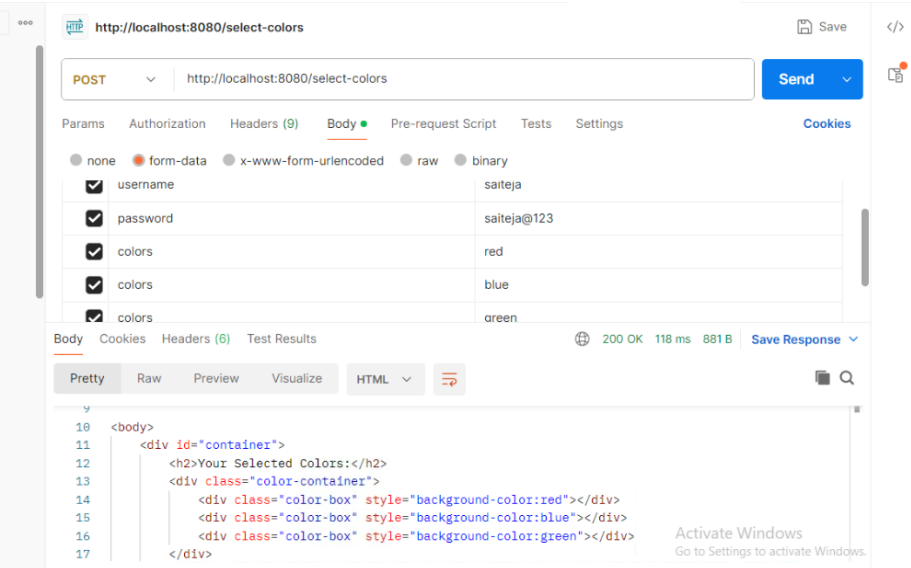
5.4 Acceptance Testing

Acceptance testing validates whether the system meets business requirements and user expectations. It involves testing the system with real users or stakeholders to ensure it fulfills its intended purpose effectively. For the Color catalogue digital shop, acceptance tests will confirm that features like color selection and selected colors align with user needs and organizational goals.

Testing By Using Postman







6.Deployment

6.1 Environment Setup

- **Development Environment:**
  - **Operating System:** Linux/Ubuntu
  - **Tools:**
    - IDE: IntelliJ IDEA or Eclipse for Java development.

        - Docker: Containerization of applications.
        - Kubernetes: Orchestrating containers.
        - OpenShift: Managing Kubernetes clusters.
        - GitHub/GitLab: Version control.
    - **Database:** SQL-based database setup (e.g., MySQL or PostgreSQL).
    - **Configuration:** Local configurations for testing and development purposes.
- **Testing Environment:**
    - **Operating System:** Linux/Ubuntu
    - **Tools:**
        - Continuous Integration (CI) tools like Jenkins or GitHub Actions for automated testing.
        - Docker and Kubernetes for container management.
        - OpenShift for managing Kubernetes clusters.
        - Load Balancers: Ensuring high availability and performance.
    - **Database:** SQL-based database with test data.
    - **Configuration:** Environment-specific configurations for integration and system testing.
- **Production Environment:**
    - **Operating System:** Linux/Ubuntu
    - **Tools:**
        - Docker and Kubernetes for container management.
        - OpenShift for managing Kubernetes clusters.
        - Load Balancers: To distribute traffic evenly and ensure reliability.
        - Monitoring tools: Prometheus, Grafana for performance monitoring.
    - **Database:** SQL-based database with production data.
    - **Configuration:** Production-specific configurations for deployment.

6.2 Deployment Strategy

- **Branch Management:**
    - **Development Branch:** For ongoing development work.
    - **Testing Branch:** For integration and system testing.

    - **Production Branch:** For the live application.
    - **Pull Requests:** Required from Dev to Test and Test to Prod for code promotion.
- **Containerization:**
    - **Docker Files:** Creation and maintenance for different environments (development, testing, production).
    - **Kubernetes Pods:** Utilize Kubernetes for deploying and managing containerized applications.
    - **Clustered Environment:** Use clustering for load balancing and failover support.
- **Deployment Pipeline:**
    - **Continuous Integration (CI):** Automatically build and test the code on each commit using tools like Jenkins or GitHub Actions.
    - **Continuous Deployment (CD):** Automated deployment to the testing environment upon passing tests.
    - **Manual Approval:** Required for deployment from testing to production environment to ensure code quality and stability.
    - **Rolling Updates:** Gradual deployment to production to minimize downtime and allow easy rollback if issues are detected.
- **Environment-Specific Deployments:**
    - **Development:** Frequent deployments for developer testing and early feedback.
    - **Testing:** Regular deployments for integration and system testing.
    - **Production:** Controlled and scheduled deployments to ensure stability and minimize disruptions.
- **Monitoring and Maintenance:**
    - **Monitoring:** Continuous monitoring of application performance and health using tools like Prometheus and Grafana.
    - **Logging:** Centralized logging for debugging and issue tracking.
    - **Alerts:** Set up alerts for critical issues to ensure quick response and resolution.

By following this deployment strategy, the color catalogue digital shop can ensure smooth, reliable, and efficient deployment processes across all environments, maintaining high standards of code quality, performance, and user satisfaction.

7.Maintainance and support

7.1 Monitoring and Logging

- **Monitoring:**
  - Use tools like Prometheus and Grafana for real-time monitoring of system performance, resource usage, and application health.
  - Set up alerts for critical metrics to ensure prompt response to issues.
- **Logging:**
  - Implement centralized logging using tools like ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk.
  - Ensure that all application logs are collected, stored, and analyzed for troubleshooting and auditing purposes.

7.2 Bug Fixes and Updates

- **Bug Tracking:**
  - Use issue tracking systems like JIRA or GitHub Issues to manage and prioritize bug fixes.
- **Regular Updates:**
  - Schedule regular updates and patches to fix bugs, enhance features, and improve security.
  - Ensure backward compatibility and thorough testing before deploying updates to the production environment.

8. Security Considerations

8.1 Authentication and Authorization

- **Authentication:**
  - Implement secure login mechanisms using best practices such as hashing passwords with bcrypt.
  - Support multi-factor authentication (MFA) for additional security.

- **Authorization:**
    - Use role-based access control (RBAC) to manage user permissions and ensure users have appropriate access to resources.

8.2 Data Protection

- **Encryption:**
    - Encrypt sensitive data both in transit (using TLS/SSL) and at rest (using database encryption mechanisms).
- **Data Privacy:**
    - Ensure compliance with data protection regulations like GDPR by implementing data anonymization and consent mechanisms.
- **Backup:**
    - Regularly back up critical data and ensure that backup processes are secure and reliable.

9. User Documentation

9.1 User Guide

- **Introduction:**
    - Provide an overview of the color catalog digital shop and its features.
- **Getting Started:**
    - Detailed instructions on how to register, log in, and use the application.
- **Feature Descriptions:**
    - Explain how to select colors, view selected colors, and any additional features available to users.
- **FAQ:**
    - Include frequently asked questions and their answers to help users troubleshoot common issues.
- **Support:**
    - Provide contact information or a support portal for users to seek help with any issues.

10. Conclusion

In conclusion, the color catalogue digital shop is designed to offer a user-friendly platform for customers to select and purchase colors on the go. By adhering to Agile with Scrum methodology, utilizing Java/J2EE for the backend, and ensuring robust deployment and maintenance processes, the application aims to deliver a high-quality, secure, and scalable solution. The integration of comprehensive security measures, thorough documentation, and proactive maintenance strategies will ensure the system remains reliable and user-focused, ultimately driving customer satisfaction and business success.