**CST-323 Design Report Template**

|  |  |  |
| --- | --- | --- |
| **Topic:** | *CST-323 eCommerce Application* | |
| **Date:** | *11-4-24* | |
| **Revision:** | *This should be the revision, starting at 1.0, for your report.* | |
| **Team:** | 1. *Kaya Nelson* | |
| 1. *Benjamin Lacock* | |
|  | |
|  | |
| **Weekly Team Status Summary:** | |  |  |  |  | | --- | --- | --- | --- | | **User Story** | **Team**  **Member** | **Hours**  **Worked** | **Hours Remaining** | | *Registration form UI completed; backend API for user registration implemented.* | *Benjamin Lacock* | *4* |  | | *Product listings page UI designed; API for fetching products created.* | *Kaya Nelson* | *5* |  | | *Awaiting completion of payment integration* | *Benjamin Lacock* | *4* |  | | *Need to finalize design and functionality details.* | *Kaya Nelson* | *2* |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | | |
| **GIT URL:** | [*https://github.com/Kdeshun/CST-323-Milestone-*](https://github.com/Kdeshun/CST-323-Milestone-) | |
| **Hosting URL:** | *The Hosting URL that I can use to access your application.* | |
| **Peer Review:** | *Y/N* | Y |

**Design Documentation**

**Install Instructions:**

 **Step 1**: Clone the application source code from Bitbucket.

 **Step 2**: Set up your database (MySQL/PostgreSQL).

* Create the necessary tables using the provided DDL scripts.

 **Step 3**: Configure application settings for development/production environments.

* Update configuration files (e.g., application.properties for Spring Boot).

 **Step 4**: Deploy the application to the chosen cloud provider (AWS/Azure).

 **Step 5**: Access the application via the provided hosting URL.

**General Technical Approach:**

 The application will follow an N-Layer architecture, separating presentation, business, and data layers.

 Development will be conducted using Agile methodology, with regular sprints and retrospectives to adapt and improve.

 Collaborative tools (e.g., Slack, Trello) will be used for communication and task management.

**Key Technical Design Decisions:**

* **Cloud Provider**:
  + **Choice**: AWS
  + **Rationale**: Offers a wide range of services and scalability.
* **Web Framework**:
  + **Choice**: Spring Boot
  + **Rationale**: Provides robust support for RESTful API development and is widely used in enterprise applications.
* **Database**:
  + **Choice**: MySQL
  + **Rationale**: A well-supported relational database that is easy to use with Spring Boot.
* **Front-End Framework**:
  + **Choice**: Bootstrap
  + **Rationale**: Facilitates responsive design to enhance user experience on various devices.

**Risks**

* **Technical Risks**:
  + Integration challenges with payment gateways.
  + Potential performance issues under high traffic.
* **Functional Risks**:
  + User experience complexities during the checkout process leading to cart abandonment.
* **Mitigation Strategies**:
  + Conduct thorough testing of integrations and user flows.
  + Implement robust security measures to protect user data.

**ER Diagram:**

*A list of items with text

Description automatically generated*

**DDL Scripts:**

*A screenshot of a computer screen

Description automatically generated*

**Sitemap Diagram:**

*A screen shot of a computer

Description automatically generated*

**Security Design:**

**1. Authentication**

* **User Registration**:
  + Implement a registration process requiring users to provide a unique email, password, and address.
  + Passwords should be hashed using a secure hashing algorithm (e.g., bcrypt) before storing them in the database.
* **Login Process**:
  + Users log in using their email and password.
  + Implement account lockout mechanisms after a specified number of failed login attempts to prevent brute-force attacks.
  + Use JWT (JSON Web Tokens) for session management post-login, allowing stateless authentication.

**2. Authorization**

* **Role-Based Access Control (RBAC)**:
  + Define user roles (e.g., Customer, Admin) and their corresponding permissions.
  + Customers can access product listings, their own order history, and manage their profiles.
  + Admins can manage product listings, view all orders, and manage user accounts.
* **Access Control**:
  + Use middleware to check user roles and permissions before granting access to specific endpoints (e.g., admin routes).

**3. Data Protection**

* **Data Encryption**:
  + Encrypt sensitive data in transit using HTTPS to protect against man-in-the-middle attacks.
  + Consider encrypting sensitive data at rest in the database, such as user addresses.
* **Input Validation**:
  + Validate all user inputs to prevent SQL injection and cross-site scripting (XSS) attacks.
  + Use prepared statements for database queries to ensure safe execution.

**4. Session Management**

* **JWT Tokens**:
  + Use short-lived JWT tokens for user sessions and refresh tokens for longer sessions.
  + Implement token revocation strategies to invalidate tokens on logout or after a certain period of inactivity.
* **Secure Cookies**:
  + Set cookies with the HttpOnly and Secure flags to mitigate risks of XSS and session hijacking.

**5. Auditing and Logging**

* **Logging**:
  + Implement logging for authentication attempts, access to sensitive data, and administrative actions.
  + Use a logging framework (e.g., SLF4J for Java or Monolog for PHP) to capture logs.
* **Monitoring**:
  + Regularly monitor logs for suspicious activities and configure alerts for unusual access patterns.

**6. Third-Party Service Security**

* **API Security**:
  + Use OAuth for securing third-party API integrations (e.g., payment gateways).
  + Ensure that API keys and secrets are stored securely and not hard-coded in the application.

**7. Compliance and Best Practices**

* **GDPR/CCPA Compliance**:
  + Implement features to allow users to manage their data (e.g., delete account, export data).
* **Regular Security Audits**:
  + Regularly review and update security practices and perform penetration testing to identify vulnerabilities.

**Third Party Interface Design:**

**1. Payment Gateway Integration**

* **Provider**: [e.g., Stripe, PayPal]
* **Purpose**: Process secure payments.
* **API Endpoint**: POST /v1/charges
* **Authentication**: API key.
* **Required Parameters**:
  + amount (in cents)
  + currency
  + source (payment token)
* **Example Response**:

json{

"id": "ch\_1GqIC8L6jT8gQX",

"status": "succeeded"

}

**2. Shipping Service Integration**

* **Provider**: [e.g., UPS, FedEx]
* **Purpose**: Calculate shipping rates and track shipments.
* **API Endpoint**: POST /shipping/calculate
* **Authentication**: API key.
* **Required Parameters**:
  + origin
  + destination
  + weight
* **Example Response**:

json{

"rate": {

"service": "Ground",

"cost": 15.99

}

}

**3. Email Service Integration**

* **Provider**: [e.g., SendGrid]
* **Purpose**: Send order confirmations.
* **API Endpoint**: POST /v3/mail/send
* **Authentication**: API key.
* **Required Parameters**:
  + from
  + to
  + subject
* **Example Response**:

json{

"message": "Email sent successfully"

}

**4. Security Considerations**

* Implement rate limiting and data validation.
* Handle errors from APIs appropriately.

**5. Integration Testing**

* Use mock services for testing to avoid costs.

**Flowcharts:**

**1. User Registration**

* **Start** ➔ Input User Details ➔ Validate Inputs
  + **If Valid**: Hash Password ➔ Save User ➔ **End**
  + **If Invalid**: Show Error Message ➔ Return to Input

**2. Login Process**

* **Start** ➔ Input Email and Password ➔ Fetch User
* Validate Credentials
  + **If Valid**: Generate JWT ➔ Redirect to Dashboard ➔ **End**
  + **If Invalid**: Show Error Message ➔ Return to Input

**3. Checkout Process**

* **Start** ➔ View Cart ➔ Proceed to Checkout ➔ Input Shipping & Payment Information
* Process Payment
  + **If Successful**: Create Order ➔ Send Confirmation ➔ **End**
  + **If Failed**: Show Payment Error ➔ Return to Payment

**User Interface Diagrams:**

**1. Home Page**

* Navigation Bar: Links to Home, Products, Cart, Login/Logout
* Search Bar and Featured Products Section

**2. User Registration/Login Page**

* **Registration Form**: Name, Email, Password
* **Login Form**: Email, Password, and Forgot Password link

**3. Product Listings Page**

* Filters Sidebar for categories and price
* Product Cards displaying image, name, price, and Add to Cart button

**4. Product Details Page**

* Large Product Image, Description, Price, and Add to Cart button

**5. Shopping Cart Page**

* List of items with Quantity selectors and Total Price
* Checkout Button

**6. Checkout Page**

* Shipping and Payment Information forms
* Submit Order Button

**Class Diagrams:**

A diagram of a company

Description automatically generated

**Pseudo Code:**

**1. User Registration**

FUNCTION registerUser(name, email, password)

IF isEmailValid(email) THEN

hashedPassword = hashPassword(password)

user = new User(name, email, hashedPassword)

SAVE user TO database

RETURN "User registered successfully"

ELSE

RETURN "Invalid email format"

END FUNCTION

**2. User Login**

FUNCTION loginUser(email, password)

user = FETCH user FROM database WHERE email = email

IF user EXISTS THEN

IF comparePassword(password, user.password) THEN

token = generateJWT(user.id)

RETURN token

ELSE

RETURN "Invalid password"

END IF

ELSE

RETURN "User not found"

END FUNCTION

**3. Add Product**

FUNCTION addProduct(name, description, price, stockQuantity)

product = new Product(name, description, price, stockQuantity)

SAVE product TO database

RETURN "Product added successfully"

END FUNCTION

**4. Get All Products**

FUNCTION getAllProducts()

products = FETCH all products FROM database

RETURN products

END FUNCTION

**5. Create Order**

FUNCTION createOrder(userID, productList, shippingAddress)

totalAmount = 0

FOR EACH product IN productList DO

productDetails = FETCH product FROM database WHERE id = product.id

totalAmount += productDetails.price \* product.quantity

END FOR

order = new Order(userID, totalAmount, shippingAddress)

SAVE order TO database

RETURN "Order created successfully"

END FUNCTION

**6. Get User Orders**

FUNCTION getUserOrders(userID)

orders = FETCH orders FROM database WHERE userID = userID

RETURN orders

END FUNCTION

**Summary**

This pseudocode outlines the basic functions of the eCommerce application, including user registration, login, product management, and order processing. Each function describes the logic without getting into specific programming details, making it easy to understand the flow and structure.

**Other Documentation:**

*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, please explain the rationale for leaving this section as N/A.*