****

**Shoe Review Software Architecture Document**

**SAD**

**1.Product Overview 2.Architectural Models**

**3.Mapping Between Models 4.Architectural Design Rationale**

**1.0 Product Overview for Shoe Review**

**Product Vision**

The Shoe Review Website is a specialized e-commerce platform designed to provide a seamless experience for shoe enthusiasts and shoppers. It combines the functionality of a traditional e-commerce platform with a review and rating system, allowing users to browse, purchase, and review various types of shoes. The platform also supports personalized wishlist management and administrative controls for shoe review moderation.

**Key Features:**

1. **Comprehensive Shoe Catalog**:
   * Offers a diverse selection of shoes categorized by brand, style, size, and price.
   * Detailed descriptions, stock information, and high-quality images for each product.
2. **Wishlist Management**:
   * Users can create and manage wishlists for desired shoes.
   * Ability to save shoes for future purchase or reference.
3. **Order Placement and Tracking**:
   * Allows users to add shoes to their cart, place orders, and track order status.
   * Features real-time stock updates and dynamic pricing calculations.
4. **User Reviews**:
   * Enables users to rate and write reviews for purchased shoes.
   * Helps other users make informed buying decisions based on reviews and ratings.
5. **Administrative Control Panel**:
   * Allows administrators to moderate shoe reviews and ensure the integrity of the review system.
   * Includes functionality for managing products and user feedback.

**Stakeholders:**

* **End Users**:
  + Primary users who browse, purchase, and review shoes.
  + Create personalized wishlists and place orders.
* **Shoe Brands**:
  + Partners promoting their products through the platform.
  + Benefit from user reviews and visibility.
* **Platform Administrators**:
  + Oversee and manage user interactions, reviews, and products.
* **Developers**:
  + Maintain the platform and enhance its features for scalability and usability.

**Target Market:**

* **General Consumers**: Individuals looking to purchase shoes online conveniently.
* **Shoe Enthusiasts**: People passionate about shoe collections, styles, and brands.
* **Retailers & Brands**: Seeking to enhance their digital presence and reach a broader audience.

**Competitive Advantages:**

* **Focused Niche**: Unlike general e-commerce platforms, this website caters specifically to shoe enthusiasts, creating a tailored user experience.
* **Integrated Review System**: Empowers users with reliable product feedback directly linked to each shoe.
* **Wishlist Flexibility**: Allows for easy organization and management of potential purchases.

**Technical Highlights:**

* **Dynamic Database Design**:
  + Relationships between users, reviews, orders, and shoes ensure consistent and reliable data management.
  + Modular architecture that allows for scalability and easy maintenance.
* **Interactive User Interface**:
  + Designed to be responsive and intuitive, offering a seamless browsing and shopping experience.
  + Integration of modern web technologies for enhanced performance.
* **Secure Transactions**:
  + Uses secure payment gateways and encrypted data storage to protect user information.

### 2. Architectural Models

This section provides the specification of architectural models for the Shoe Review Website, focusing on both static and dynamic aspects.

#### ****2.1 Static Models****

The static models depict the structural elements of the system, including its key entities and relationships.

##### **2.1.1 Entity-Relationship Diagram (ERD)**

The ERD showcases the relationships between the primary entities in the system. Below is a detailed description of the entities and their relationships:

* **User**: Represents customers and administrators.
  + Relationships:
    - Writes reviews (Review).
    - Creates wishlists (Wishlist).
    - Places orders (Order).
* **Review**: Represents feedback submitted by users for shoes.
  + Attributes: reviewID, userID, shoeID, rating, comment.
* **Order**: Tracks user purchases.
  + Attributes: orderID, userID, orderDate, paymentMethod, totalPrice, status, shippingAddress.
  + Relationships:
    - Contains multiple items (OrderItem).
* **Wishlist**: Represents a user's collection of desired shoes.
  + Attributes: wishlistID, userID, name.
  + Relationships:
    - Contains multiple shoes (WishlistItem).
* **Shoe**: The core product in the system.
  + Attributes: shoeID, name, brandID, styleID, categoryID, price, size, stock, description, imageURL.
  + Relationships:
    - Belongs to a Brand, Style, and Category.

##### **2.1.2 Component Diagram**

This diagram highlights the modular architecture of the system.

* **Frontend**:
  + User Interface for browsing, managing wishlists, and placing orders.
  + Admin Interface for moderating reviews and managing products.
* **Backend**:
  + API layer for handling requests from the frontend.
  + Database interaction to manage and query entities.
* **Database**:
  + Centralized storage for all entities: Users, Reviews, Shoes, Orders, Wishlists.

#### ****2.2 Dynamic Models****

The dynamic models illustrate how the system behaves during specific interactions.

##### **2.2.1 Wishlist Interaction Workflow**

* **Actors**: User, System.
* **Steps**:
  1. User navigates to a shoe page and clicks “Add to Wishlist”.
  2. System creates a WishlistItem record linked to the user’s Wishlist and the selected Shoe.
  3. System displays the updated wishlist to the user.

##### **2.2.2 Order Placement Workflow**

* **Actors**: User, System.
* **Steps**:
  1. User selects shoes from the wishlist or catalog and adds them to the cart.
  2. System calculates the total price and checks stock availability.
  3. User confirms the order and provides payment details.
  4. System:
     + Creates an Order record linked to the user.
     + Creates OrderItem records for each shoe in the order.
     + Updates stock levels in the Shoe entity.
  5. Order status is updated, and confirmation is sent to the user.

##### **2.2.3 Review Submission Workflow**

* **Actors**: User, System.
* **Steps**:
  1. User navigates to a purchased shoe and submits a review.
  2. System validates the input (e.g., rating is within the allowed range).
  3. A Review record is created and linked to the user and shoe.
  4. The review is displayed in the shoe’s details page.

**2.3 Simulation Layer Decomposition**

| **Component** | **Functionality** | **Interactions** | **Purpose** |
| --- | --- | --- | --- |
| **User Interaction Simulation** | Simulates user actions like browsing shoes, adding items to wishlist/cart, and placing orders. | - Mimics real user flows (e.g., adding items to wishlist, submitting reviews). - Interacts with Order, Review, and Wishlist entities without affecting live data. | Test user interactions and validate workflows without affecting live data. |
| **Stock Management Simulation** | Simulates real-time stock updates when items are added to cart or purchased. | - Decreases stock count when an item is purchased. - Verifies sufficient stock before purchase. - Simulates stock depletion scenarios. | Prevent overselling by simulating stock management under high traffic. |
| **Order Placement Simulation** | Simulates the order placement workflow (from cart to purchase). | - Simulates adding products to an order. - Processes payment, calculates total, creates Order and OrderItem records. - Updates stock accordingly. | Ensure correct order processing and stock updates during purchase. |
| **Review Simulation** | Simulates the review submission process for purchased shoes. | - Creates and links reviews to users and shoes. - Tests validation rules (e.g., reviewing only purchased items). | Validate the review submission workflow for authenticity and accuracy. |
| **Database Simulation** | Simulates interaction with a mock database. | - Mock data used for testing rather than live data. - Simulates creation, update, and deletion of entities like User, Order, Review, and Shoe. | Safely test database operations without affecting live data. |
| **Error and Exception Handling Simulation** | Simulates various system errors and exceptions. | - Injects error conditions like stock issues, payment failures, and network errors. - Verifies system’s graceful handling of errors. | Test system response to errors and ensure it behaves appropriately under failure conditions. |

**2.4 Simulation Layer Interface Specifications**

| **Interface** | **Component** | **Description** | **Methods/Functions** | **Data Flow** |
| --- | --- | --- | --- | --- |
| **User Interaction Interface** | User Interaction Simulation | Provides methods to simulate user actions like browsing, adding items to wishlist, placing orders, and reviews. | - addToWishlist(shoeID, userID) - placeOrder(userID, cart) - submitReview(userID, shoeID, review) | - Input: shoeID, userID, cart, review - Output: Simulated user interaction (added to wishlist, order placed, review submitted) |
| **Stock Management Interface** | Stock Management Simulation | Interfaces with the stock management system to update stock levels in real-time during order placement. | - updateStock(shoeID, quantity) - checkStock(shoeID) - simulateStockDepletion(shoeID) | - Input: shoeID, quantity - Output: Updated stock levels after purchase or simulation |
| **Order Placement Interface** | Order Placement Simulation | Simulates order processing by adding products to the order, processing payments, and creating OrderItem records. | - createOrder(userID, cart) - calculateTotal(cart) - confirmOrder(orderID) | - Input: userID, cart, orderID - Output: Order and OrderItem records created, updated total |
| **Review Submission Interface** | Review Simulation | Interfaces with the review system, ensuring only purchases are reviewed and associating reviews with Shoe. | - submitReview(userID, shoeID, rating, comment) - validateReview(userID, shoeID) | - Input: userID, shoeID, rating, comment - Output: Created and linked Review record |
| **Database Interface** | Database Simulation | Simulates interactions with the database, creating and updating records for entities like User, Order, etc. | - createRecord(entity, data) - updateRecord(entity, recordID, data) - deleteRecord(entity, recordID) | - Input: entity, data, recordID - Output: Simulated database changes (insert, update, delete) |
| **Error/Exception Handling Interface** | Error and Exception Handling | Provides methods to simulate error conditions like stock issues, payment failures, and network issues. | - simulateStockError(shoeID) - simulatePaymentFailure(userID, orderID) - simulateNetworkError() | - Input: shoeID, userID, orderID - Output: Simulated error conditions for testing |

**2.5 User Interface Layer Structure**

The **User Interface (UI) Layer** of the **Shoe Review Website** is responsible for providing an intuitive and responsive interface for both users and administrators. Below is the structure of the UI Layer, detailing its key components, functionality, and interactions.

**1. Purpose**

The User Interface Layer enables:

* Customers to browse products, manage wishlists, place orders, and submit reviews.
* Administrators to moderate reviews, manage products, and oversee platform operations.

**2. Components of the UI Layer**

| **Component** | **Description** | **Key Features** | **Interactions** |
| --- | --- | --- | --- |
| **Homepage** | The landing page for users, showcasing featured products and categories. | - Display featured shoes, categories, and offers. - Provide search and navigation options. | - Interacts with the Product Management backend to fetch featured shoes and categories. |
| **Product Listing Page** | Displays a list of shoes based on categories, styles, or search queries. | - Supports sorting and filtering options. - Displays product thumbnails, prices, and ratings. | - Fetches products from the backend. - Links to the Product Details Page for more information. |
| **Product Details Page** | Provides detailed information about a selected shoe. | - Displays product images, description, price, stock status, and reviews. - Option to add to wishlist or cart. | - Fetches product details and reviews. - Interacts with the Wishlist and Cart components. |
| **Wishlist Page** | Displays the user’s wishlist and allows them to manage items. | - List of saved items. - Option to move items to cart or remove them from wishlist. | - Interacts with the Wishlist Management backend to fetch and update wishlist data. |
| **Cart Page** | Displays items added to the cart and allows users to proceed to checkout. | - Displays cart summary and total cost. - Option to remove items or update quantities. - Checkout button. | - Interacts with the Order Management system to initiate the checkout process. |
| **Checkout Page** | Allows users to confirm their order and provide payment and shipping details. | - Collects shipping address and payment details. - Displays order summary. - Submit order button. | - Sends order data to the backend for processing. - Updates stock in the Stock Management system. |
| **Review Submission Page** | Enables users to write and submit reviews for purchased products. | - Form to submit ratings and comments. - Displays a list of eligible products for review. | - Validates user’s eligibility to review. - Interacts with the Review Management system. |
| **Admin Dashboard** | Provides an interface for administrators to manage reviews, products, and users. | - Displays admin tools for moderating reviews, adding products, and managing users. | - Interacts with all backend modules for data management. |

**3. Navigation Flow**

* **Homepage**:
  + Users can browse featured shoes or navigate to categories and search results.
* **Product Listing Page**:
  + Users select a shoe, which directs them to the **Product Details Page**.
* **Wishlist Page**:
  + Users can add items to the cart or remove them.
* **Cart Page**:
  + Users proceed to the **Checkout Page** for payment and order confirmation.
* **Admin Dashboard**:
  + Admins can navigate to tools for managing reviews, products, and users.

**4. Interaction with Other Layers**

The User Interface Layer communicates with the backend layers (Business Logic Layer and Data Layer) through APIs to:

1. **Fetch Data**: Retrieves product information, user wishlists, and reviews.
2. **Submit Actions**: Sends user actions such as adding items to cart, placing orders, and submitting reviews.
3. **Receive Responses**: Displays feedback to the user, such as successful order placement or review submission.

**5. Design Principles**

* **Responsiveness**: Optimized for devices of all sizes (desktop, tablet, mobile).
* **User-Centric Design**: Focused on ease of use, minimal clicks, and intuitive navigation.
* **Scalability**: Supports adding new features or pages without disrupting the existing structure.

**3. Mapping Between Models**

This section provides the relationship between the static and dynamic models, illustrating how entities interact within the system during different workflows.

#### ****3.1 Entity Relationship Mapping****

| **Entity** | **Static Model Description** | **Dynamic Model Interaction** |
| --- | --- | --- |
| **User** | Represents customers and administrators. Tracks interactions, orders, and reviews. | The User interacts with various workflows such as placing orders, creating wishlists, and submitting reviews. |
| **Shoe** | Core entity with attributes such as name, price, size, and stock. Associated with brand, style, and category. | Shoes are added to the Wishlist and Order records, and they are reviewed by Users. |
| **Wishlist** | Represents a collection of shoes a user intends to buy. | Users add shoes to their wishlist, and a WishlistItem is created linking the user and shoe. |
| **Review** | Links users to specific shoes and includes attributes like rating, comment, and review date. | Users submit reviews for shoes after purchase. Each review is linked to a User and a Shoe. |
| **Order** | Represents the order a user places, containing items linked to specific shoes. | Once the user confirms a purchase, an Order and related OrderItems are created, linking the user to the shoes they ordered. |

#### ****3.2 Mapping Dynamic Model Workflow to Entities****

| **Workflow** | **Entities Involved** | **Static Model Link** |
| --- | --- | --- |
| **User adds a shoe to the wishlist** | User, Shoe, Wishlist, WishlistItem | The User adds a Shoe to their Wishlist, creating a WishlistItem record. |
| **Order placement process** | User, Order, OrderItem, Shoe | The User places an Order, which contains multiple OrderItem records linked to specific Shoes. |
| **User submits a review for a shoe** | User, Shoe, Review | After purchasing a shoe, the User creates a Review linked to both the Shoe and User. |

### ****4. Architectural Design Rationale****

This section explains the critical design decisions made during the development of the Shoe Review Website. It covers design choices that were difficult to implement, crucial to system performance, and hard to change once made. These decisions are central to the system’s functionality, scalability, and maintainability.

#### ****4.1 Key Design Decisions****

##### **1. Separation of Concerns**

* **Decision**: The system was divided into distinct modules (User, Product Management, Order Management, and Review Management).
* **Rationale**: This separation ensures each module focuses on a specific set of related functions, making the system easier to maintain and extend. It also allows for better scalability, as each module can evolve independently. For example, the Order Management module can be optimized without impacting the Review Management module.
* **Challenges**: Initially, there was concern about potential tight coupling between modules, especially between Order and Review entities. However, maintaining clear module boundaries helps in isolating failures and scaling individual modules.

##### **2. Database Normalization and Entity Relationships**

* **Decision**: The system uses a normalized relational database model to store data, linking entities like User, Shoe, Review, Order, and Wishlist.
* **Rationale**: This approach reduces redundancy and ensures data consistency across the platform. Each entity is linked through foreign keys, which provides a structured way to manage and query data. For example, reviews are linked to both users and shoes, ensuring reviews are always associated with the correct entities.
* **Challenges**: One of the difficulties was ensuring the relationships between User, Shoe, and Review remained consistent when a shoe was removed or updated. This required careful management of foreign key constraints to prevent orphaned records.

##### **3. Real-Time Stock Allocation**

* **Decision**: The system supports dynamic, real-time stock updates when a user places an order.
* **Rationale**: By ensuring that the available stock is updated in real time, the system prevents users from purchasing out-of-stock items. This decision is essential for user trust and operational efficiency.
* **Challenges**: Managing concurrent access to stock data posed a significant challenge, especially during high traffic periods. Implementing an efficient locking mechanism to prevent race conditions in stock updates required additional effort and testing.

#### ****4.2 Crucial Design Decisions****

##### **1. Review System Linked to Purchase**

* **Decision**: Users can only submit reviews for shoes they have purchased through the platform.
* **Rationale**: This ensures that reviews are relevant and trustworthy, as users can only review items they have experience with. It prevents fake or biased reviews, which is crucial for maintaining the credibility of the platform.
* **Challenges**: The complexity of enforcing this rule was a significant challenge, especially when dealing with users who might try to manipulate the system. This required integration between the Order and Review systems to validate that a review could only be submitted after an order had been completed.

##### **2. Flexible Wishlist System**

* **Decision**: Users can create multiple wishlists, and each wishlist can contain multiple shoes.
* **Rationale**: This gives users flexibility in managing their desired products. For example, a user might have one wishlist for summer shoes and another for formal shoes. This decision enhances user experience and increases engagement.
* **Challenges**: A key challenge was designing a system that allowed multiple, dynamically manageable wishlists for each user while maintaining performance and avoiding data redundancy.

#### ****4.3 Hard-to-Change Decisions****

##### **1. Product and Category Hierarchy**

* **Decision**: The system maintains a strict categorization of products, associating each shoe with a Brand, Style, and Category.
* **Rationale**: This hierarchical structure ensures that the system can scale as more products are added, and it helps users filter and search for shoes effectively.
* **Challenges**: The design of the product hierarchy created constraints when attempting to add more categories or change the product structure. For example, adding new product types or modifying the brand structure requires updating the database schema, which could impact system performance and require significant testing.

##### **2. Scalability and Load Balancing**

* **Decision**: The platform was designed to handle a growing number of users and products by distributing traffic across multiple servers (using load balancing techniques).
* **Rationale**: To support potentially thousands of users shopping simultaneously, a robust and scalable infrastructure was necessary. Load balancing helps ensure that traffic is evenly distributed, preventing server overloads and improving overall system performance.
* **Challenges**: Scaling the system while ensuring minimal downtime and consistent performance during traffic spikes required sophisticated infrastructure planning and the implementation of autoscaling policies.

#### ****4.4 Difficult and Crucial Decisions****

##### **1. Security and User Data Protection**

* **Decision**: The platform employs encryption for sensitive user data (e.g., passwords, payment information) and uses secure authentication methods (OAuth, token-based authentication).
* **Rationale**: Protecting user privacy and ensuring secure transactions is paramount. Given the platform's role in handling financial transactions and personal information, security was a priority throughout the design process.
* **Challenges**: Ensuring that the system remained secure against evolving threats required continuous monitoring and periodic updates to the security protocols. The difficulty was in designing a solution that balanced user convenience with robust security measures.

##### **2. Cross-Platform Compatibility**

* **Decision**: The platform is designed to be fully responsive, ensuring a consistent experience across mobile, tablet, and desktop devices.
* **Rationale**: Given the wide variety of devices users may access the platform on, ensuring that the system is compatible with all screen sizes is crucial for user experience. This decision is vital in attracting a broad audience.
* **Challenges**: Implementing a truly responsive design required careful testing across multiple browsers and devices, which added complexity to the design process.