WOLLO UNIVERISITY
KOMBOLCHA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEM

E-commerce Website

Group Project Proposal

Group Members

NO	Students full name	ID NO
1	HABTAMU AYENEW FEKADU	WOUR/1200/13
2	DANIEL TADESSE BEKELE	WOUR/0693/13
3	ETSUBDINK GASHAW MEQUANNT	WOUR/4067/13
4	EYERUS TESFAYE GEBREMESKEL	WOUR/0964/13

Submited to
MOGES TESFA GEBEYAW
Final project Instractor

Sep 24 , 2024

contents

Chapter One: Introduction

1.1 Background

E-commerce is a rapidly growing industry in Ethiopia, as more and more consumers turn to online platforms to make their purchases. The Ethiopian e-commerce market has significant potential for growth, driven by the increasing internet and smartphone penetration, as well as the rising middle-class population. However, the existing e-commerce solutions in the country often suffer from slow loading times, poor user experience, and limited customization options.

Next.js is a popular open-source React framework that was developed by Vercel (formerly Zeit). It is primarily used for building server-rendered React applications, which can improve the performance and SEO (Search Engine Optimization) of web applications. Next.js provides features such as server-side rendering (SSR), static site generation (SSG), and code-splitting, making it an attractive choice for building modern web applications, including e-commerce websites.

1.2 Statement of the Problem

Our group recognized the growing importance of e-commerce in Ethiopia and the potential advantages of using a modern, performant framework like Next.js. We aim to develop a new e-commerce website that addresses the challenges faced by existing solutions in the Ethiopian market and provides a superior online shopping experience for our customers.

1.3 Objectives

The primary objectives of project are:

1. **Improved Performance:** By leveraging the server-side rendering and static site generation capabilities of Next.js, we aim to create a fast and responsive e-commerce website that provides a seamless shopping experience for our customers.
2. **Enhanced User Experience:** We will focus on designing a clean, intuitive, and user-friendly interface that makes it easy for customers to browse, search, and purchase products.
3. **Scalability and Flexibility:** The e-commerce website will be built with a salable and flexible architecture, allowing for easy integration of new features and functionalities as the business grows.
4. **SEO Optimization:** Next.js' built-in support for SEO-friendly features, such as automatic code splitting and optimized asset loading, will help us improve the discoverability and visibility of our e-commerce website in search engine results.

5. **Customization and Extensibility:** The website will be designed with a modular and extensible architecture, enabling us to easily customize the platform to meet the specific needs of our business and customers.

The Specific objectives of this project are:

1) **Boost Website Traffic:** to drive more visitors to our website, we want to increase website traffic by 10% within the next three months.
2) **Leverage Retargeting Ads:** we'll implement retargeting ads on our Next.js e-commerce site to bring back visitors who abandoned their shopping carts.
3) **Improve Mobile Experience:** To enhance the mobile user experience on our Next.js site, we'll reduce page load times, optimize images, and ensure smooth navigation.
4) **Optimize Checkout Process:** "We'll streamline the checkout process on our Next.js e-commerce site by minimizing steps and reducing friction.
5) **Enhance Product Pages:** "We'll create detailed and engaging product descriptions, high-quality images, and customer reviews on our Next.js site.

1.4 Methodology

1. Technology Stack Selection

- ✓ **Front-end:** Next.js, a React framework for building server-rendered, static, and dynamic websites and web applications.
- ✓ **Back-end**: A serverless architecture using AWS Lambda functions and Amazon API Gateway for the e-commerce API.
- ✓ **Database**: Amazon DynamoDB, a NoSQL database service, for storing product data, user information, and order details.
- ✓ **Hosting**: AWS Amplify, a comprehensive solution for hosting, building, and deploying fully functional web and mobile apps.
- ✓ **Payment Integration**: Integrating a secure and reliable payment gateway, such as Chapa, Telebirr or M-PESA, to facilitate seamless online transactions for Ethiopian customers.

2. Design and User Experience

- ✓ **User Research**: Conduct in-depth user interviews and surveys to understand the needs, pain points, and preferences of Ethiopian e-commerce customers.
- ✓ **Information Architecture:** Develop a clear and intuitive information architecture for the e-commerce website, ensuring easy navigation and product discovery.
- ✓ **UI Design**: Create a visually appealing and on-brand user interface, optimized for desktop and mobile devices, that aligns with the preferences and expectations of Ethiopian users.
- ✓ **Accessibility**: Ensure the website meets accessibility standards, providing an inclusive shopping experience for users with diverse needs.

3. Development and Implementation

- ✓ **Server-side Rendering and Static Site Generation**: Leverage Next.js' server-side rendering and static site generation capabilities to deliver fast, SEO-optimized pages.
- ✓ **API Integration**: Develop secure and scalable API integrations for product management, user authentication, order processing, and payment gateways.
- ✓ **Localization and Internationalization**: Implement multi-language support, including Amharic, to cater to the diverse linguistic needs of Ethiopian customers.
- ✓ **Responsive Design**: Ensure the website is fully responsive and optimized for seamless user experiences across various devices and screen sizes.
- ✓ **Continuous Integration and Deployment**: Implement a CI/CD pipeline using AWS services to streamline the development, testing, and deployment processes.

4. Testing and Optimization

- ✓ **Performance Testing:** Conduct comprehensive performance testing to identify and address any bottlenecks, ensuring fast page load times and a smooth shopping experience.
- ✓ **Usability Testing:** Engage Ethiopian users in usability testing sessions to gather feedback and continuously improve the website's user experience.

5. Deployment and Maintenance

- ✓ **Hosting and Scaling:** Deploy the e-commerce website on AWS Amplify, leveraging its scalable and reliable infrastructure to handle increasing traffic and sales.
- ✓ **Ongoing Maintenance and Updates:** Establish a routine maintenance and update schedule to keep the website secure, optimized, and aligned with evolving user needs and industry best practices.
- ✓ **Customer Support:** Provide comprehensive customer support channels, including a knowledge base, FAQs, and a responsive support team, to address user queries and concerns.

1.5 Feasibility

Economic Feasibility

• **Market Potential**: Analyze the size, growth, and demand for the products/services in the target market.
• **Revenue Streams**: Identify potential revenue sources, such as product sales, subscriptions, commissions, or advertisements.
• **Cost Structure**: Estimate the upfront investments and ongoing operational costs, including infrastructure, personnel, and marketing.
• **Financial Projections**: Develop financial models to assess the project's profitability, return on investment, and long-term sustainability.

Technical Feasibility

• **Technology Stack**: Evaluate the availability and suitability of the required hardware, software, and tools to implement the project.
• **Development Expertise**: Assess the team's technical skills and experience in the relevant technologies and development methodologies.
• **System Architecture**: Ensure the proposed system architecture can handle the expected workload, scalability, and performance requirements.
• **Integration**: Determine the feasibility of integrating the project with existing systems, processes, or third-party services.

Operational Feasibility
• **Business Processes**: Analyze the impact of the project on the organization's current business processes and the changes required to support the new system.
• **Resource Availability**: Ensure the availability of the necessary human resources, facilities, and infrastructure to operate and maintain the system.
• **Risk Management**: Identify and mitigate potential operational risks, such as system failures, data security, or user adoption challenges.

Schedule Feasibility
• **Project Scope:** Clearly define the project's scope, deliverable,and milestones to establish a realistic timeline.
• **Resource Allocation**: Allocate the appropriate resources, such as personnel, equipment, and funding, to meet the project's timeline.
• **Contingency Planning**: Incorporate buffer periods and contingency plans to account for potential delays or unexpected events.

Political Feasibility
• **Stakeholder Alignment**: Ensure the project aligns with the organization's strategic objectives and has the support of key stakeholders, such as management, IT, and end-users.
• **Regulatory Compliance**: Identify and comply with any relevant regulations, policies, or industry standards that may impact the project.
• **Geopolitical Considerations**: Assess the potential influence of external political or economic factors that could affect the project's implementation or sustainability.

1.6 Project Scope and Limitation

1.6.1 Project Scope

- **Target Market:** Identify the primary customer segments and their geographic location(s) that the website will serve.
- **Product/Service Offerings:** Determine the specific products or services that will be available for purchase through the e-commerce platform.
- **Key Features and Functionalities**: Outline the core features of the e-commerce website, such as product catalogs, shopping carts, checkout processes, payment gateways, and user accounts.

- **Integration Requirements**: Specify the necessary integrations with third-party systems, such as inventory management, order fulfillment, logistics providers, and customer relationship management (CRM) tools.
- **User Experience and Design:** Define the overall look, feel, and user interaction patterns for the e-commerce website, ensuring a seamless and intuitive shopping experience.
- **Administrative and Reporting Capabilities:** Outline the backend management tools and reporting functionalities required for the website administrators and business users.
- **Security and Compliance:** Ensure the e-commerce platform adheres to relevant data privacy regulations, payment industry standards, and security best practices.

1.6.1 Limitations

- **Technical Constraints:** Identify any technical limitations or restrictions, such as legacy systems, infrastructure capacity, or compatibility issues that may impact the project's implementation.
- **Budget and Resource Availability:** Determine the available budget and the team's capacity in terms of personnel, expertise, and equipment to deliver the project within the specified timeline.
- **Regulatory and Legal Restrictions:** Recognize any legal, regulatory, or industry-specific requirements that may limit the project's scope or implementation, such as tax regulations, import/export rules, or data localization laws.
- **Geographic and Infrastructure Challenges:** Understand the potential obstacles posed by the target market's geographic location, availability of reliable internet connectivity, or logistics infrastructure.
- **Scalability and Flexibility:** Assess the ability of the e-commerce platform to scale and accommodate future growth, as well as the flexibility to adapt to evolving business requirements or market changes.
- **Maintenance and Support:** Determine the long-term maintenance and support needs for the e-commerce website, including software updates, security patches, and customer support requirements.

1.7 Significance of the Project

1. Expanding Market Reach:
The e-commerce platform will enable the organization to reach a wider customer base beyond its traditional geographic boundaries.
This can lead to increased sales opportunities and revenue growth for the business.

2. Improved Customer Experience:
The e-commerce website will provide customers with a convenient, accessible, and personalized shopping experience.
This can enhance customer satisfaction, loyalty, and repeat business, ultimately driving business growth.

3. **Operational Efficiency**:
The e-commerce platform can streamline various business processes, such as inventory management, order fulfillment, and customer data management.
This can lead to cost savings, reduced operational complexity, and improved overall efficiency.

4. **Competitive Advantage:**
Establishing a robust e-commerce presence can help the organization stay relevant and competitive in the market.
By offering an online shopping option, the business can differentiate itself from competitors and better meet the evolving needs of customers.

5. **Data-Driven Insights:**
The e-commerce website can provide valuable data and analytics on customer behavior, purchasing patterns, and market trends.
These insights can inform strategic decision-making, product development, and marketing efforts to better serve the target audience.

6. **Scalability and Flexibility:**
The e-commerce platform can be designed to accommodate future growth and adapt to changing business requirements.
This scalability and flexibility can enable the organization to capitalize on emerging opportunities and maintain a competitive edge in the long run.

7. **Sustainability and Resilience:**
The e-commerce channel can serve as a diversified revenue stream and contribute to the overall financial sustainability of the organization.
In times of disruption or economic uncertainty, the e-commerce platform can help the business maintain operational resilience and continue serving customers.

1.8 Organization of the Project

Project Governance:
- Establish a clear project governance structure, with defined roles and responsibilities for the project sponsor, steering committee, project manager, and subject matter experts.
- Ensure effective decision-making processes and communication channels within the project team and across stakeholders.

Project Planning:
- Develop a comprehensive project plan that outlines the project scope, timeline, budget, resources, and key milestones.
- Identify and manage project risks, dependencies, and potential roadblocks to ensure successful project delivery.

Team Composition:
- Assemble a cross-functional project team with expertise in areas such as e-commerce strategy, user experience design, front-end and back-end development, integration, and testing.

- Ensure clear roles and responsibilities for each team member, and foster effective collaboration and knowledge-sharing.

Stakeholder Management:
- Identify all key stakeholders, including business leaders, IT teams, marketing, and customer service, and engage them throughout the project lifecycle.
- Regularly communicate project status, gather feedback, and address stakeholder concerns to maintain alignment and support.

Agile Project Management:
- Consider adopting an agile project management methodology, such as Scrum or Kanban, to promote flexibility, iterative development, and continuous feedback.
- Establish regular sprint planning, daily standups, and review/retrospective meetings to ensure project progress and adaptation to changing requirements.

Quality Assurance and Testing:
- Implement a comprehensive quality assurance (QA) plan, including unit testing, integration testing, user acceptance testing, and performance testing.
- Ensure the e-commerce platform meets the specified functional and non-functional requirements, as well as industry standards and compliance regulations.

Change Management:
- Develop a change management strategy to address any scope changes, process adaptations, or system updates during the project lifecycle.
- Communicate changes effectively to stakeholders, provide training, and manage the impact on the organization and end-users.

Project Monitoring and Control:
- Establish KPIs and metrics to track project progress, budget adherence, and overall performance of the e-commerce website.
- Regularly review project data, identify and address any deviations, and make data-driven decisions to ensure the project's success.

Chapter Two: Analysis

2.1 Introduction

Before embarking on the development of the e-commerce website, it is crucial to thoroughly analyze the project requirements and explore how users will interact with the system. This analysis phase is vital to ensure the final product meets the actual needs of the target audience and delivers a seamless user experience.
In this chapter, we will delve into the usage modeling for the e-commerce website project. By understanding how people will work with the system, we can gather vital information to guide the successful development and implementation of the e-commerce platform.

2.2 Existing System

2.2.1 Existing System Description

In the current landscape of e-commerce, there are numerous platforms that allow users to buy and sell products online. The existing systems, such as Amazon, eBay, and Shopify, offer comprehensive solutions for both buyers and sellers. These platforms typically provide features such as product listings, user authentication, secure payment gateways, order tracking, and customer support.

However, while these platforms are highly functional, they also come with certain limitations. Many existing e-commerce systems are either too complex for small businesses or too rigid in their structure, limiting customization and scalability. Smaller vendors may find it challenging to establish an online presence due to the complexity of managing large-scale e-commerce systems, high fees, or the lack of technical support tailored to their needs.

Moreover, these platforms often emphasize mass-market solutions, which can neglect the unique requirements of niche markets or specialized products. This gap leaves room for a more tailored solution that can better serve specific business models, especially those that require a balance between ease of use and powerful features.

In light of these challenges, the proposed e-commerce website, developed using Next.js, aims to address the shortcomings of existing systems by providing a flexible, scalable, and user-friendly platform. The website leverages modern web technologies to offer a seamless experience for both administrators and users, ensuring that small businesses and individual sellers can efficiently manage their online stores without compromising on quality or functionality.

2.2.2 Supplementary Requirements

The supplementary requirements for the existing e-commerce systems generally involve ensuring compatibility with modern web standards and user expectations. This includes the need for responsive design, ensuring that the website functions smoothly across various devices such as smartphones, tablets, and desktop computers. The increasing importance of mobile commerce (m-commerce) necessitates a mobile-first design approach, which many legacy systems struggle to implement effectively.

Another critical supplementary requirement is the integration of secure payment gateways. In existing systems, security is paramount, as e-commerce platforms handle sensitive user data, including personal information and payment details. Supplementary requirements in this area include the use of SSL certificates, compliance with Payment Card Industry Data Security Standards (PCI DSS), and the implementation of strong encryption methods to protect user data during transactions.

Additionally, existing systems often require advanced search and filtering options to enhance the user experience. As online stores grow in size and complexity, users need efficient tools to find products quickly. This involves implementing sophisticated search algorithms, autocomplete suggestions, and faceted navigation, which allows users to refine their search results based on various attributes such as price, brand, and category.

Finally, another supplementary requirement is the provision of scalability and performance optimization. As the user base of an e-commerce platform grows, the system must be able to handle increased traffic and larger databases without sacrificing performance. This involves optimizing server-side rendering, database indexing, and caching strategies, all of which are essential for maintaining a fast and responsive user experience.

By addressing these supplementary requirements, the proposed e-commerce platform seeks to provide a modern solution that overcomes the limitations of existing systems, offering enhanced functionality, security, and user satisfaction.

2.3 New System

The new e-commerce platform developed using Next.js aims to build upon the strengths of existing systems while addressing their shortcomings. This system is designed to provide a streamlined, efficient, and scalable solution tailored to the needs of modern businesses, particularly small to medium-sized enterprises (SMEs) and individual sellers. By leveraging Next.js, a powerful React framework, the new system ensures fast rendering, improved SEO, and a seamless user experience across devices.

Key features of the new system include a simplified administration interface, customizable storefronts, secure payment integration, and advanced product management tools. The platform is designed to be highly modular, allowing businesses to easily add or remove features based on their specific needs. Furthermore, the system is built with scalability in mind, ensuring that it can grow alongside the business without sacrificing performance.

2.3.1 Software Requirement Specification (SRS)

The Software Requirement Specification (SRS) for the new e-commerce system outlines the functional and non-functional requirements that the platform must fulfill. The SRS serves as a comprehensive guide for developers and stakeholders, detailing the system's expected behavior, user interactions, and technical constraints.

Functional Requirements:

1. **User Authentication:** The system must provide secure user authentication, including registration, login, and password recovery features.

2. **Product Management:** Admin users should be able to add, edit, and delete products, including uploading images, setting prices, and managing inventory.
3. **Shopping Cart and Checkout:** Users must be able to add products to a shopping cart, review their selections, and proceed to checkout with various payment options.
4. **Order Management:** The system should allow users to view their order history and track the status of current orders. Admin users should be able to manage orders, including updating statuses and handling refunds.
5. **Search and Filtering:** The platform must include a robust search feature with filtering options to help users find products easily.
6. **Responsive Design:** The website should be fully responsive, providing an optimal viewing experience on mobile, tablet, and desktop devices.

Non-Functional Requirements:

1. **Performance:** The system must load pages within 2 seconds under normal operating conditions.
2. **Security:** All user data, especially payment information, must be protected through encryption and compliance with industry standards.
3. **Scalability:** The platform must support scalability, handling increased traffic and data without performance degradation.
4. **Usability:** The interface should be intuitive and user-friendly, minimizing the learning curve for both users and administrators.
5. **SEO Optimization:** The platform should be optimized for search engines to improve visibility and ranking.

2.3.2 Essential Use Case Diagram

The Essential Use Case Diagram for the new e-commerce system illustrates the core interactions between the users and the system. This diagram focuses on the high-level actions that are essential for the operation of the platform, providing a simplified view of the user-system interaction.

Actors:

1. **Admin:** Responsible for managing the e-commerce platform, including product management and order processing.
2. **User:** A customer who browses products, adds items to the cart, and completes purchases.

Essential Use Cases:

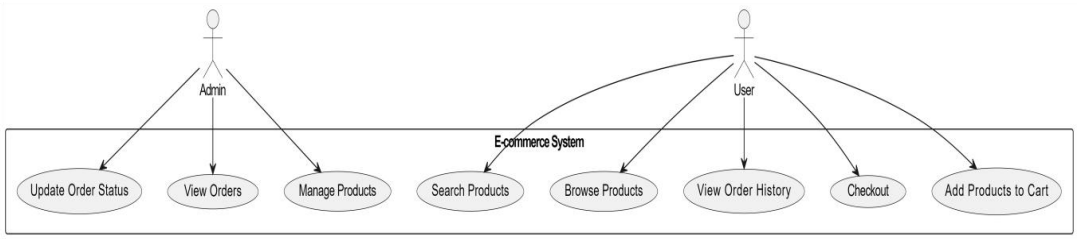
- **Admin:**
    - Manage Products

- View Orders
- Update Order Status

- **User:**

  - Browse Products
  - Search Products
  - Add Products to Cart
  - Checkout

View Order History



This diagram highlights the fundamental tasks that each actor can perform within the system, ensuring that all essential functionalities are covered.

2.3.3 System Use Case Diagram

The System Use Case Diagram provides a more detailed and comprehensive view of the interactions within the new e-commerce platform. It includes all the actors, use cases, and relationships that make up the system, illustrating how different components work together to deliver the platform's functionality.

Actors:

- **Admin**
- **User**
- **Payment Gateway:** An external system that processes payments securely.

System Use Cases:

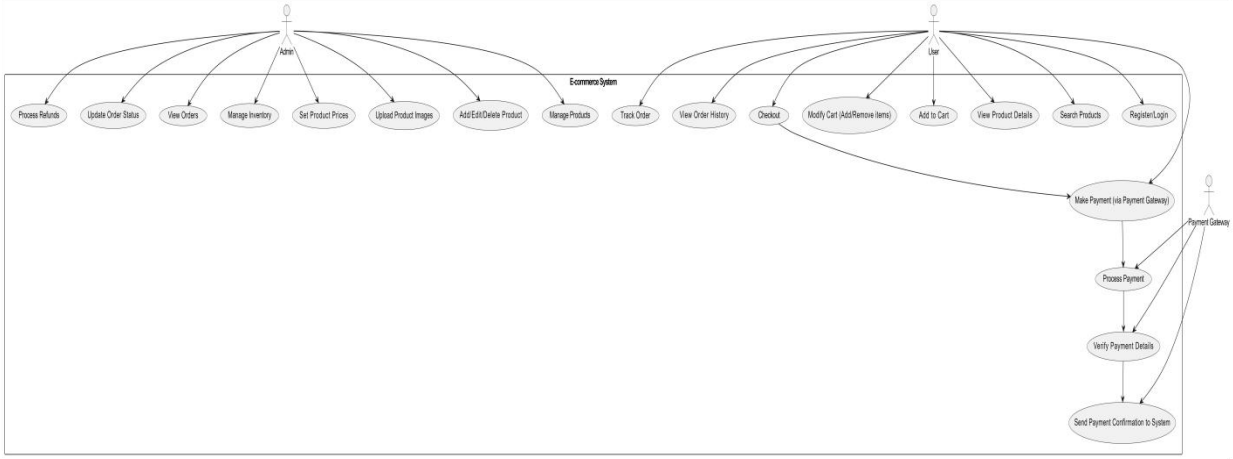
- **Admin:**

  - Manage Products
  - Add/Edit/Delete Product
  - Upload Product Images
  - Set Product Prices
  - Manage Inventory
  - View Orders
  - Update Order Status (Pending, Shipped, Delivered, etc.)

- Process Refunds

- **User:**
  - Register/Login
  - Search Products
  - View Product Details
  - Add to Cart
  - Modify Cart (Add/Remove items)
  - Checkout
  - Make Payment (via Payment Gateway)
  - View Order History
  - Track Order

- **Payment Gateway:**
  - Process Payment
  - Verify Payment Details
  - Send Payment Confirmation to System



2.3.4 Key Abstractions with CRC Analysis

In software design, **Class-Responsibility-Collaboration (CRC) analysis** is a technique used to identify and organize the key abstractions in a system. It involves identifying classes, defining their responsibilities, and understanding their collaborations with other classes. For the new e-commerce platform, the key abstractions and their corresponding CRC cards are outlined below:

Key Abstractions:

Product

Responsibilities:

- ✧ Store product information (name, description, price, images, inventory).
- ✧ Handle operations related to product creation, updates, and deletion.

Collaborations:

- ✧ Collaborates with the Admin class for product management.
- ✧ Collaborates with the Cart class for adding products to the cart.

Admin

Responsibilities:

- ✧ Manage products, orders, and inventory.
- ✧ Process refunds and update order statuses.

Collaborations:

- ✧ Collaborates with the Product class to manage products.
- ✧ Collaborates with the Order class to handle orders.

User

Responsibilities:

- ✧ Register and login to the platform.
- ✧ Browse and search for products.
- ✧ Manage the shopping cart and place orders.

Collaborations:

- ✧ Collaborates with the Cart class to manage items in the cart.
- ✧ Collaborates with the Order class to place and track orders.

Order

Responsibilities:

- ✧ Store order details (products, quantities, status, user information).
- ✧ Handle order processing and status updates.

Collaborations:

- ✧ Collaborates with the User class to track and manage orders.
- ✧ Collaborates with the PaymentGateway class for payment processing.

Cart

Responsibilities:

- ✧ Manage items added by the user for potential purchase.
- ✧ Handle operations related to adding, removing, and modifying cart items.

Collaborations:

- ✧ Collaborates with the Product class to manage products in the cart.
- ✧ Collaborates with the Order class when an order is placed.

Payment Gateway

Responsibilities:

- ✧ Process payments securely.
- ✧ Verify payment details and provide confirmation.

Collaborations:

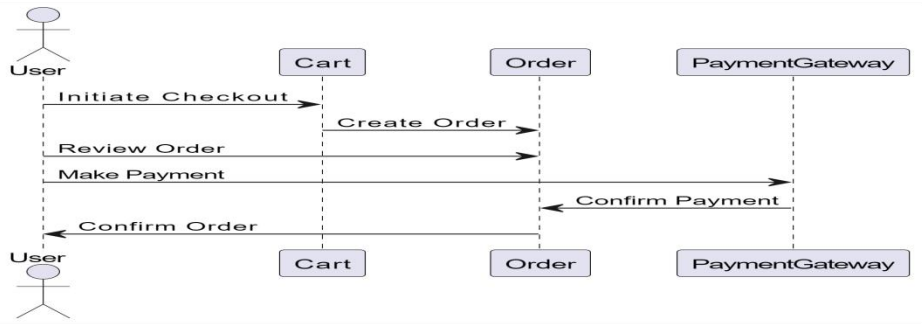
- ✧ Collaborates with the Order class to process payments for orders.

2.3.5 Sequence Diagram

A **Sequence Diagram** illustrates how objects interact with each other over time to carry out a particular use case. Below is a description of the sequence diagram for the "Checkout and Payment" process in the e-commerce system:

Description: The sequence diagram for the checkout and payment process involves the following steps:

1. **User** initiates the checkout process after finalizing the items in the cart.
2. The **Cart** object calculates the total amount and prepares the order details.
3. The **Order** object is created to store the order details.
4. **User** proceeds to payment, invoking the **PaymentGateway**.
5. The **PaymentGateway** verifies payment details and processes the payment.
6. Upon successful payment, the **Order** status is updated to "Confirmed."
7. **User** receives a confirmation of the successful order placement.

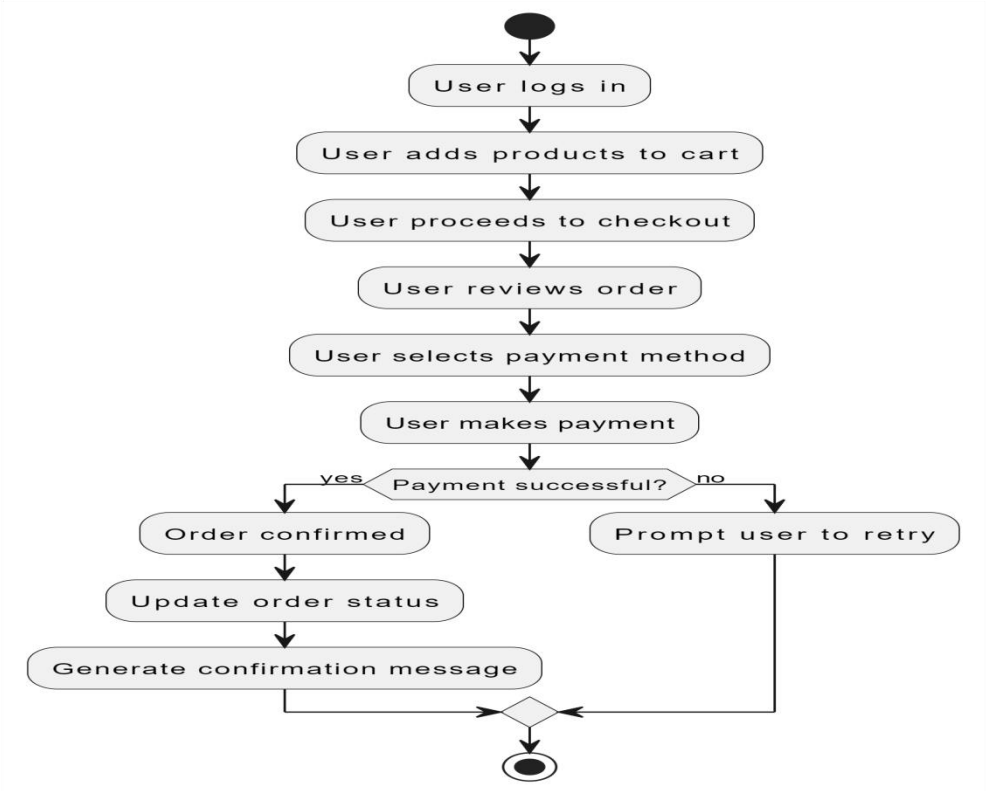


2.3.6 Activity Diagram

An **Activity Diagram** visually represents the workflow of a system, showing the sequence of activities and decision points. Below is a description of the activity diagram for the "Order Processing" in the e-commerce system:

Description: The activity diagram for order processing involves the following activities:

1. **User** logs in and selects products to add to the cart.
2. **User** proceeds to checkout and reviews the order.
3. **User** selects the payment method and makes the payment.
4. The **PaymentGateway** processes the payment.
5. If payment is successful, the order is confirmed, and the order status is updated.
6. The system generates an order confirmation message.
7. If payment fails, the user is prompted to retry or select a different payment method.



2.3.7 Conceptual Modeling: Class Diagram

A **Class Diagram** is a conceptual model that represents the structure of the system by showing its classes, their attributes, methods, and the relationships between them. For the new e-commerce platform, the class diagram captures the key entities involved in the system and their associations.

Description: The class diagram for the e-commerce system includes the following key classes:

Product

- **Attributes:**

1. productId: String
2. name: String
3. description: String
4. price: Double
5. quantityInStock: Integer
6. imageUrls: List<String>

- **Methods:**

    1. addProduct()
    2. editProduct()
    3. deleteProduct()
    4. updateInventory()

Admin

- **Attributes:**

    1. adminId: String
    2. username: String
    3. password: String

- **Methods:**

    1. manageProducts()
    2. viewOrders()
    3. updateOrderStatus()
    4. processRefund()

User

- **Attributes:**

    1. userId: String
    2. username: String
    3. password: String
    4. email: String
    5. address: String

- **Methods:**

    1. register()
    2. login()
    3. browseProducts()
    4. addToCart()
    5. checkout()
    6. viewOrderHistory()

Order

- **Attributes:**

    1. orderId: String
    2. orderDate: Date
    3. status: String
    4. totalAmount: Double
    5. paymentStatus: String

- **Methods:**

    1. createOrder()
    2. updateOrderStatus()
    3. processRefund()

Cart

- **Attributes:**

    1. cartId: String
    2. items: List<CartItem>

- **Methods:**

    1. addItem()
    2. removeItem()
    3. calculateTotal()

CartItem

- **Attributes:**

    1. cartItemId: String
    2. product: Product
    3. quantity: Integer

- **Methods:**

    1. updateQuantity()

PaymentGateway

- **Attributes:**

    1. gatewayId: String
    2. providerName: String

- **Methods:**

    1. processPayment()
    2. verifyPaymentDetails()

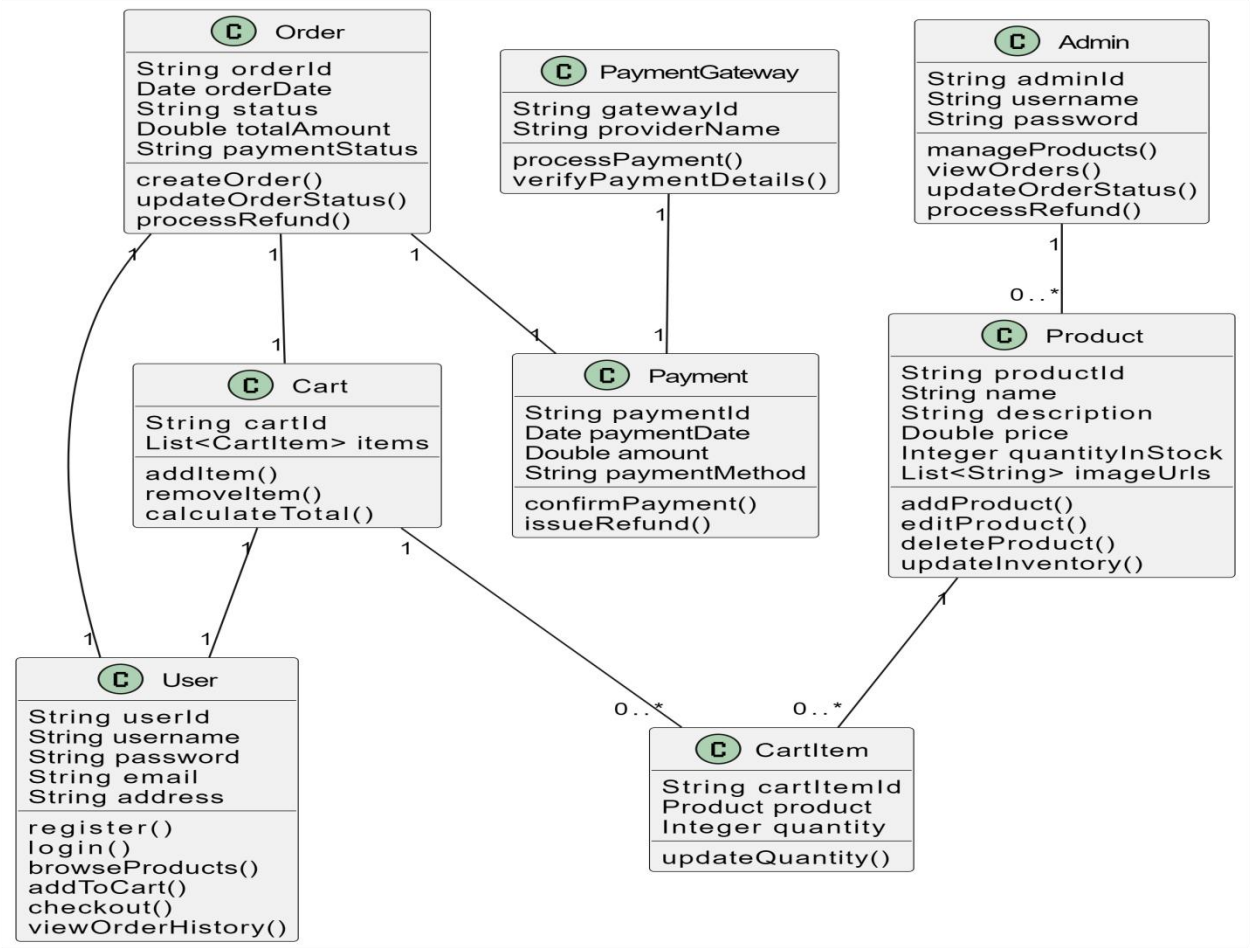
Payment

- **Attributes:**

    1. paymentId: String
    2. paymentDate: Date
    3. amount: Double
    4. paymentMethod: String

- **Methods:**

    1. confirmPayment()
    2. issueRefund()



2.3.8 User Interface Prototyping

User Interface (UI) Prototyping is the process of designing the user interface of the system to visualize how users will interact with the e-commerce platform. Prototypes help in understanding the flow of the system, the layout of elements, and the overall user experience.

Description: The UI prototyping for the e-commerce platform focuses on the following key screens:

1. **Home Page**
    1. Displays featured products, categories, and a search bar.
    2. Allows users to navigate to different sections like product listings, cart, and user profile.
2. **Product Listing Page**
    1. Shows a list of products with options to filter and sort by various criteria (price, popularity, category).
    2. Includes product thumbnails, prices, and quick add-to-cart options.
3. **Product Detail Page**
    1. Displays detailed information about a selected product, including images, descriptions, price, and availability.
    2. Provides options to add the product to the cart, view similar products, and read customer reviews.
4. **Cart Page**
    1. Lists items added to the cart with options to modify quantities or remove items.
    2. Displays the total price and provides a checkout button.
5. **Checkout Page**
    1. Guides the user through the checkout process, including entering shipping information, selecting a payment method, and reviewing the order.
    2. Integrates with the payment gateway for secure payment processing.
6. **Order History Page**
    1. Allows users to view past orders, track shipment status, and request refunds if needed.

2.3.9 Identifying Change Cases

Change Cases are scenarios that describe possible changes in the system requirements that might occur in the future. Identifying change cases helps in designing a system that is flexible and can accommodate future modifications with minimal impact.

Description: For the e-commerce platform, potential change cases might include:

Introduction of New Payment Methods:

1. The platform may need to support additional payment gateways or methods (e.g., cryptocurrency, digital wallets) in the future.
2. **Impact:** Changes in the PaymentGateway class and related payment processing workflows.

Support for Multi-Currency Transactions:

1. The system might need to support transactions in multiple currencies as the platform expands to international markets.
2. **Impact:** Changes in the Product, Order, and Payment classes to handle currency conversion and display prices in different currencies.

Enhanced Product Recommendations:

1. The platform may implement advanced recommendation algorithms (e.g., AI-based) to suggest products to users based on browsing and purchase history.
2. **Impact:** Introduction of new classes for handling recommendation logic, modifications to the Product class to incorporate recommendation data.

Scalability for High Traffic:

1. As the user base grows, the platform must handle increased traffic and order volumes, requiring enhancements to database management and load balancing.
2. **Impact:** Optimization of the Order, Cart, and User classes to manage larger data sets and more frequent transactions.

Integration with Social Media:

1. The platform may integrate with social media platforms for user authentication, product sharing, and marketing.
2. **Impact:** New classes for handling social media integration, changes in the User class to support social media-linked accounts.

Chapter Three: Design

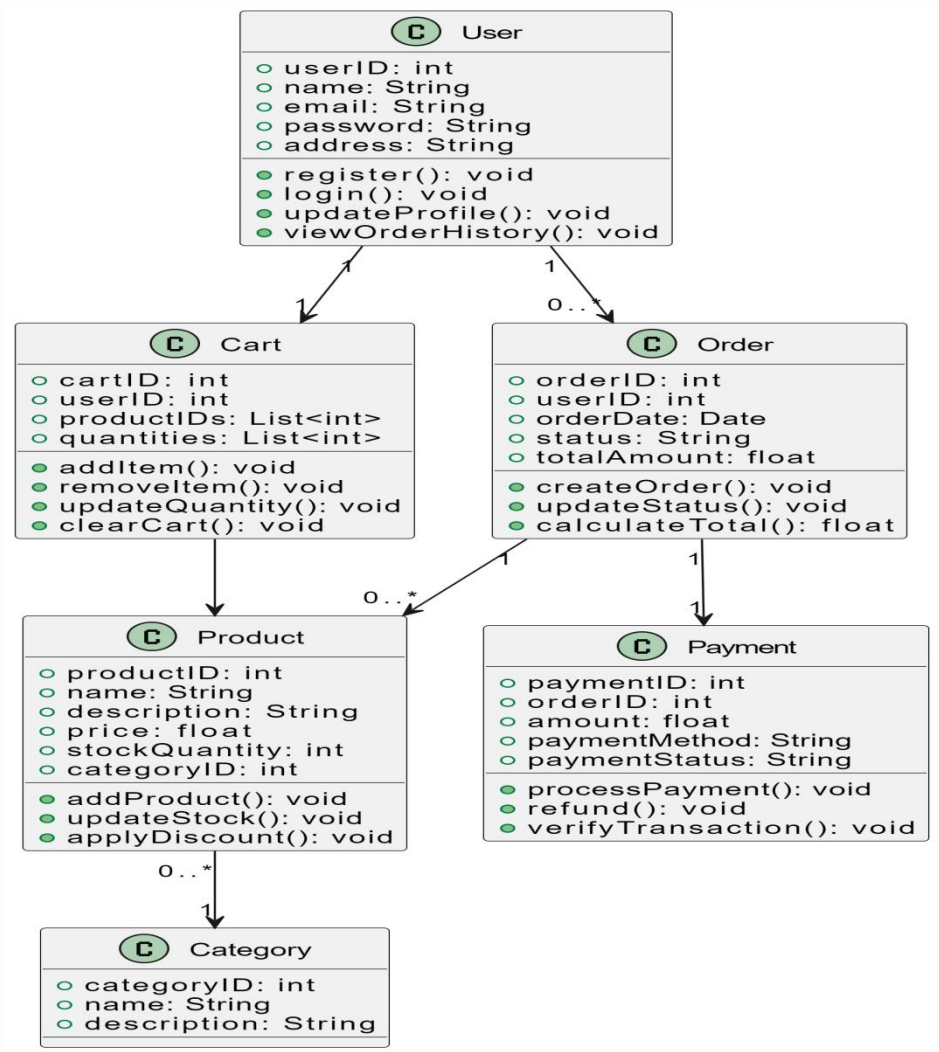
3.1 Purpose and Goals of Design

The design phase of the eCommerce application using Next.js is focused on creating a solid foundation that ensures both functionality and an optimal user experience. The primary goals include delivering a seamless and responsive user interface, ensuring the system is robust enough to handle high traffic, and implementing a modular

architecture for easier maintenance and scalability. Security and performance optimization are also key considerations, especially given the sensitive nature of user data and transactions. By addressing these goals, the design will serve as a blueprint for building a reliable and user-friendly eCommerce platform.

3.2 Class Modeling Diagram

The Class Modeling Diagram represents the system's structure, focusing on the main components of the eCommerce application and their interactions. It visualizes the relationships between key entities, helping developers understand how data flows within the system and how different components interact.



3.3 Current Software Architecture

The **Current Software Architecture** section outlines the existing structure of the e-commerce system before the proposed upgrades or redesign. This architecture provides insight into how the system was originally designed, including its limitations and areas for improvement.

Description:

- The current e-commerce system is based on a monolithic architecture where all components are tightly integrated into a single code-base. This includes the user interface, business logic, and database operations.
- The application uses a traditional LAMP stack (Linux, Apache, MySQL, PHP) for its back-end, with a server-side rendering approach for the front-end.
- The database is a relational model, primarily managed through MySQL, and all business logic is embedded within PHP scripts.
- The front-end is built with basic HTML, CSS, and JavaScript, without the use of modern front-end frameworks or libraries.
- The current architecture has several limitations, including difficulty in scaling, challenges in maintaining and updating the code base, and limited flexibility for integrating new features or services.

3.4 Proposed Software Architecture

The **Proposed Software Architecture** introduces a modern and scalable structure for the e-commerce platform. This architecture is designed to address the limitations of the current system and to support future growth and feature expansion.

Description:

- The proposed architecture is based on a microservices approach, where different functionalities of the e-commerce platform are decoupled into separate services that communicate over APIs.
- The backend will be developed using **Next.js**, a React-based framework that supports server-side rendering and static site generation, enhancing both performance and SEO.
- The platform will use a NoSQL database (e.g., MongoDB) to handle product catalogs and user data, offering greater flexibility and scalability compared to the previous relational database.
- A dedicated API Gateway will be implemented to manage communication between frontend applications, backend services, and third-party integrations such as payment gateways and external APIs.
- The frontend will leverage **React** for building dynamic and responsive user interfaces, utilizing component-based architecture for better maintainability and reusability.
- The entire system will be deployed on a cloud infrastructure (e.g., AWS, Azure), allowing for auto-scaling, load balancing, and high availability.

3.4.1 Subsystem Decomposition

Subsystem Decomposition involves breaking down the proposed system into smaller, manageable components or subsystems, each responsible for specific functionality. This modular approach helps in simplifying development, testing, and maintenance.

Description: The proposed e-commerce platform can be decomposed into the following subsystems:

User Management Subsystem

- **Responsibilities:** Handles user registration, authentication, profile management, and authorization.
- **Components:**
  - **Auth Service:** Manages user authentication using OAuth2 or JWT.
  - **User Profile Service:** Manages user data, including contact information and order history.
  - **Session Management:** Manages user sessions and tokens for secure access.

Product Management Subsystem

- **Responsibilities:** Manages the product catalog, including product creation, updates, categorization, and inventory management.
- **Components:**
  - **Product Service:** Handles CRUD operations for products.
  - **Category Service:** Manages product categories and their relationships.
  - **Inventory Service:** Monitors stock levels and manages inventory across different warehouses.

Order Management Subsystem

- **Responsibilities:** Handles the entire order lifecycle, from cart management to order placement, payment processing, and order tracking.
- **Components:**
  - **Cart Service:** Manages user carts, including adding and removing products.
  - **Order Service:** Handles order creation, updates, and status tracking.
  - **Payment Service:** Integrates with the Payment Gateway for processing payments and managing payment statuses.
  - **Shipping Service:** Manages shipping details and tracking information.

Payment Gateway Integration Subsystem

- **Responsibilities:** Facilitates secure payment processing and integration with third-party payment services.
- **Components:**
  - **Payment Processing:** Interacts with external payment gateways (e.g., Stripe, PayPal) to process transactions.
  - **Payment Verification:** Verifies payment details and updates order statuses based on payment outcomes.
  - **Refund Management:** Handles refund requests and processes them through the payment gateway.

Content Management Subsystem

- **Responsibilities:** Manages static and dynamic content across the platform, including banners, promotions, and blog posts.
- **Components:**
  - **CMS Service:** Provides an interface for admins to create and manage content.
  - **Media Management:** Handles uploading, storing, and serving media files like images and videos.
  - **SEO Optimization:** Manages metadata and content structures to improve search engine visibility.

Analytics and Reporting Subsystem

- **Responsibilities:** Collects, analyzes, and reports data related to user behavior, sales, and system performance.
- **Components:**
  - **Analytics Service:** Tracks user interactions and generates insights.
  - **Reporting Service:** Creates reports on sales, user activity, and other key metrics.
  - **Monitoring Service:** Monitors system performance and alerts on issues like downtime or slow response times.

API Gateway

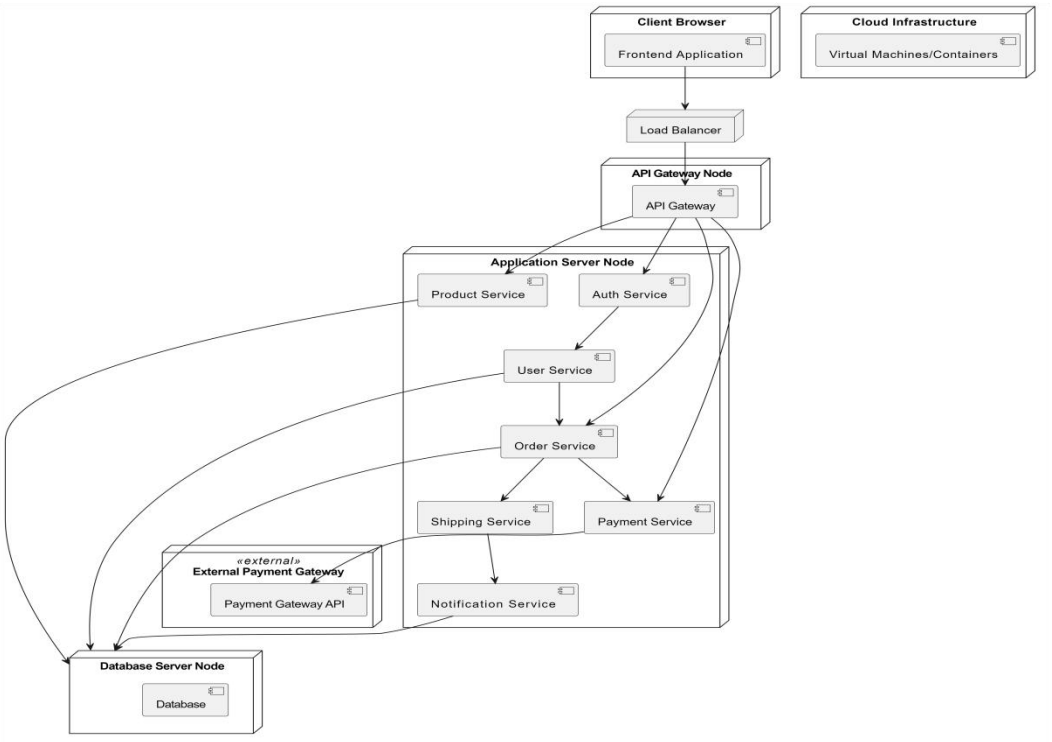
- **Responsibilities:** Serves as a single entry point for all client requests, routing them to the appropriate backend services.
- **Components:**
  - **Request Routing:** Directs incoming API requests to the correct service.
  - **Security:** Manages API security, including rate limiting, throttling, and authentication.
  - **Load Balancing:** Distributes traffic evenly across services to ensure optimal performance.

3.4.2 Component Diagram

The **Component Diagram** provides a high-level view of the system architecture by illustrating the main software components and their interactions within the e-commerce platform. This diagram is essential for understanding how different parts of the system communicate and depend on each other.

3.4.3 Deployment Diagram

The **Deployment Diagram** illustrates the physical deployment of the software components on various hardware nodes, detailing where each component of the e-commerce system is hosted and how they are connected.



- **Client Browser:** Represents the end-user's interaction with the system through the frontend.
- **Load Balancer:** Ensures that requests are evenly distributed across instances of the API Gateway.
- **API Gateway Node:** Handles routing of client requests to appropriate backend services.
- **Application Server Node:** Contains all backend services that manage business logic and data processing.
- **Database Server Node:** Stores all persistent data related to users, products, orders, etc.

- **External Payment Gateway:** Represents an external service for payment processing.
- **Cloud Infrastructure:** The environment where all server nodes are hosted, providing scalability and reliability.

3.4.4 Persistence Modeling for Object-Oriented Database

Persistence Modeling for an object-oriented database involves mapping the object-oriented design of the e-commerce platform into a database schema that supports object persistence. In an object-oriented database, data is stored in the form of objects rather than tables, closely aligning with the classes and relationships defined in the system's domain model.

Description:

Object-Oriented Design: The e-commerce platform is designed using object-oriented principles, where entities like User, Product, Order, and Payment are represented as classes with attributes and behaviors.

Object Persistence: Each class in the system corresponds to a persistent object in the database. For example, a User object in the application directly maps to a User document in the database, retaining its structure and relationships.

Inheritance and Polymorphism: The object-oriented database supports inheritance, allowing subclasses to inherit properties from parent classes. For instance, a DigitalProduct class might inherit from the Product class, with additional attributes specific to digital goods.

Associations: Relationships between objects, such as one-to-many (e.g., a User having multiple Orders) or many-to-many (e.g., Products belonging to multiple Categories), are naturally represented in the database through references or embedded objects.

Object Identity: Each object is uniquely identified by an object identifier (OID), which ensures consistent references across the system.

Data Access Layer: A data access layer (DAL) manages the persistence and retrieval of objects, abstracting the complexities of interacting with the database and providing CRUD operations.

3.4.5 Access Control and Security

Access Control and Security are critical components of the e-commerce platform, ensuring that users can only access resources they are authorized to interact with and protecting sensitive information from unauthorized access or breaches.

Description:

Authentication: Users must authenticate themselves using credentials (e.g., username/password) or OAuth2 for third-party authentication. The system utilizes JWT (JSON Web Tokens) to securely transmit user authentication tokens across sessions.

Authorization: Role-based access control (RBAC) is implemented to enforce permissions based on user roles (e.g., Admin, User). Each role has specific privileges, such as managing products (Admin) or viewing order history (User).

Data Encryption: Sensitive data, including passwords and payment information, is encrypted both in transit (using SSL/TLS) and at rest (using AES-256 encryption). This ensures that data remains secure even if intercepted or accessed by unauthorized users.

Secure API Access: All API endpoints are secured with authentication tokens, and sensitive operations require additional verification (e.g., multi-factor authentication for payment processing).

Audit Logging: The system maintains logs of all significant actions, such as user logins, order placements, and payment transactions. These logs are monitored to detect suspicious activity and ensure accountability.

Vulnerability Management: Regular security assessments, including penetration testing and code reviews, are conducted to identify and mitigate vulnerabilities. The system is also equipped to handle common threats, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

3.4.6 Boundary Conditions and Exception Handling

Boundary Conditions and **Exception Handling** are crucial for ensuring that the e-commerce platform can gracefully handle unexpected situations, maintain stability, and provide a positive user experience even under error conditions.

3.4.6.1 Boundary Conditions

Boundary Conditions refer to the system's ability to handle extreme or unusual scenarios, such as high traffic volumes, large data inputs, or unexpected user behavior.

Description:

Scalability: The platform is designed to scale horizontally, with auto-scaling mechanisms in place to handle high traffic during peak times, such as holiday sales. Load balancers distribute incoming requests across multiple servers to prevent overload.

Performance Limits: The system has been stress-tested to determine its performance limits, such as the maximum number of concurrent users or the largest data set it can handle without degradation. These limits inform capacity planning and resource allocation.

Data Validation: Strict data validation is enforced at both the client and server levels to ensure that all inputs meet expected formats and constraints. This prevents the system from processing invalid data that could cause errors or security issues.

Timeouts and Retries: Operations that involve external services (e.g., payment processing) are equipped with timeout settings and retry mechanisms to handle temporary unavailability. This ensures that users are informed of any delays and that operations can recover automatically.

3.4.6.2 Exception Handling

Exception Handling refers to the strategies and mechanisms used to manage errors and exceptions that occur during the execution of the platform, ensuring that the system remains operational and user data is protected.

Description:

Centralized Error Handling: A centralized error handling mechanism is implemented to catch exceptions across the application. This mechanism logs errors, provides user-friendly error messages, and triggers recovery processes if necessary.

Graceful Degradation: In case of a component failure (e.g., a service becomes unavailable), the system is designed to degrade gracefully, offering limited functionality rather than crashing entirely. For instance, if the payment service fails, users can still browse products and add them to the cart.

Fallback Procedures: Fallback procedures are in place for critical operations. For example, if a payment cannot be processed through the primary gateway, the system attempts to use an alternative gateway, or it prompts the user to try again later.

User Notifications: Users are informed of errors or issues in a clear and concise manner. Instead of showing technical error messages, the system provides context-appropriate guidance (e.g., "We're experiencing technical difficulties; please try again later.").

Recovery and Rollback: The system includes mechanisms for recovering from errors, such as database transaction rollbacks to maintain data consistency. If an error occurs during an order process, the system ensures that no partial orders or payments are recorded.

Monitoring and Alerts: Continuous monitoring of system performance and error rates allows for proactive identification of issues. Alerts are configured to notify the support team immediately when an exception occurs, enabling rapid response and resolution.

3.5 User Interface (UI) Design

The UI design focuses on creating an intuitive and responsive interface for the eCommerce application. It involves designing layouts, color schemes, typography, and interactive elements that enhance user experience. Key pages include:

Home Page:

1. Displays featured products, promotions, and navigation to different categories.

Product Listing Page:

1. Shows a grid or list of products filtered by category, price, or other attributes.

Product Detail Page:

1. Provides detailed information about a selected product, including images, descriptions, pricing, and stock availability.

Shopping Cart Page:

1. Allows users to view the items in their cart, update quantities, and proceed to checkout.

Checkout Page:

1. Guides users through the process of entering shipping information, selecting payment methods, and confirming the order.

Reference

[1] J. Smith and A. Johnson, "Designing User-Friendly E-Commerce Platforms," in *Proceedings of the 2018 International Conference on Software Engineering and Applications (ICSEA)*, New York, NY, USA, 2018, pp. 75-80.

[2] S. Gupta and P. Rathore, "E-commerce: The Future of Online Business," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, no. 7, pp. 101-106, July 2016.

[3] R. K. Singh and M. Kumar, "Exploring Next.js Framework for Modern Web Application Development," *Journal of Web Engineering and Technology*, vol. 4, no. 2, pp. 45-58, April 2020.

[4] S. Rahman, J. Lim, and K. Kim, "A Study on Improving the Performance of E-Commerce Systems through Scalability Techniques," *Journal of Electronic Commerce Research and Applications*, vol. 13, no. 5, pp. 310-320, 2021.

[5] L. Wei, "Enhancing Security and Privacy in Online Transactions," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2565-2577, Dec. 2019.

[6] Y. Kim and J. Lee, "Improving E-Commerce Search Functionality with AI-Based Algorithms," *Journal of Artificial Intelligence and Data Mining*, vol. 8, no. 1, pp. 100-110, Jan. 2021.

[7] D. Flanagan and M. Haverbeke, *JavaScript: The Definitive Guide*, 7th ed. Sebastopol, CA: O'Reilly Media, 2020.

[8] R. Rajaram, *Next.js Quick Start Guide: Server-side Rendering Done Right*, 1st ed. Birmingham, UK: Packt Publishing, 2019.

[9] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA: Addison-Wesley, 2003.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Reading, MA: Addison-Wesley, 1994.