Enomy-Finances Project Analysis

This presentation outlines our approach to ensure the EnomyFinances project meets rigorous quality standards and maintains traceability throughout the software development lifecycle. We'll explore key design principles, behavioral modeling, and data-driven strategies for a robust and reliable solution.

PROJECT GOAL

Develop an e-commerce website using PHP, HTML, CSS, JS server on Laragon to sell clothes, with an intuitive and easy-to-use interface.

1. Build a system to add, edit, delete and display products by category.
2. Develop a user-friendly interface that allows users to easily search for products.
3. Create an administrator account to manage product categories and control orders.

An evaluation of the suitability investigation in relation to the needs of EnomyFinances

1: Suitability of Investigation for Fashion Sales Project

1.1: Software Behavioral Design Techniques:

HTML, CSS, JavaScript, and Bootstrap provide a strong foundation for a responsive and user-friendly website.
Event-driven programming (JavaScript) enables real-time updates (e.g., product search).
Bootstrap ensures responsive design across devices, essential for e-commerce.

1.2: Chosen Tools for Investigating the Problem:

HTML: Structures content.
CSS: Ensures visual appeal.
JavaScript: Manages user interaction (e.g., adding items to cart).
Bootstrap: Simplifies responsiveness across all devices.
Tools help test user experience and how design impacts behavior and conversion rates.

Behavioral Design Techniques

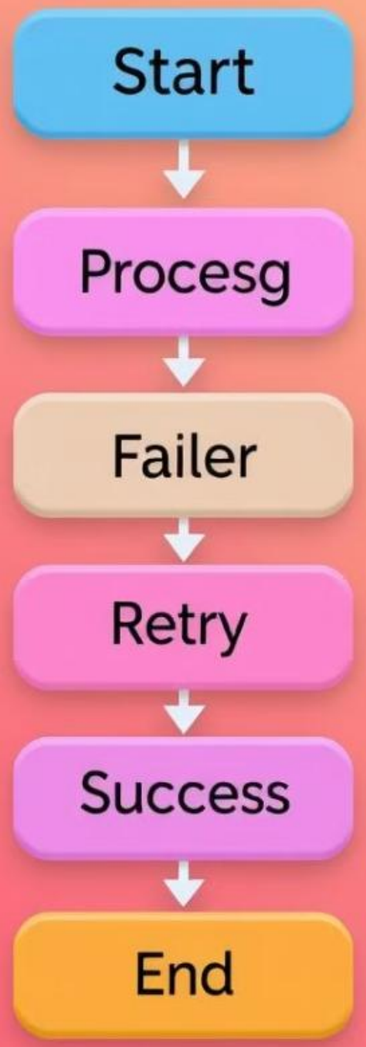
FSM

Definition: An FSM is a mathematical model that represents a system with a finite number of states. It consists of:

- A set of states.
- A set of transitions between states, triggered by events or inputs.
- State transitions are deterministic and do not depend on conditions beyond the event triggering the transition.

Example Application (FSM):
Login Process:

- States: "Logged Out," "Logged In."
- Events: User clicks "Login," enters credentials, or fails to authenticate.
- Transitions:
  - From "Logged Out" to "Logged In" after successful login.
  - Back to "Logged Out" after an unsuccessful attempt.

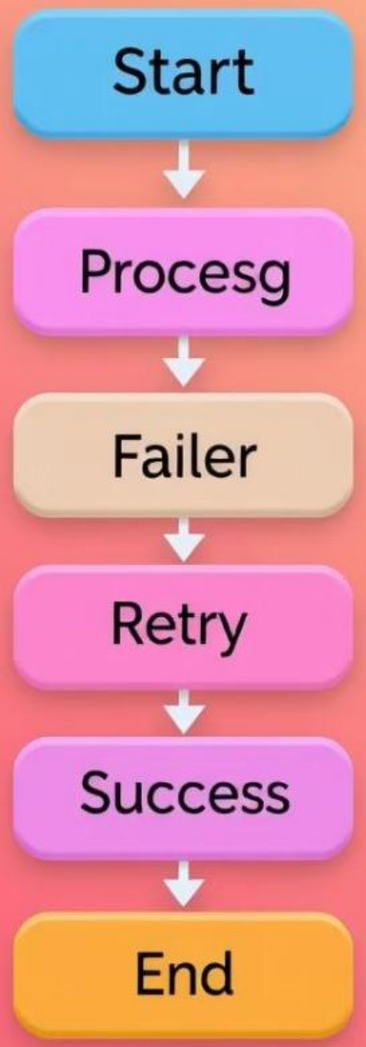


Behavioral Design Techniques

EFSM

Definition: An EFSM is an extension of the FSM that allows for more complex behavior. It includes all the features of an FSM but with additional variables (conditions) or actions associated with state transitions, such as counters, memory, or conditions that depend on external data.

- EFSMs allow for transitions based not just on events, but also on conditions or data values.
- Conditions could involve checking for data or inputs beyond simple events (e.g., numeric values, external system data).

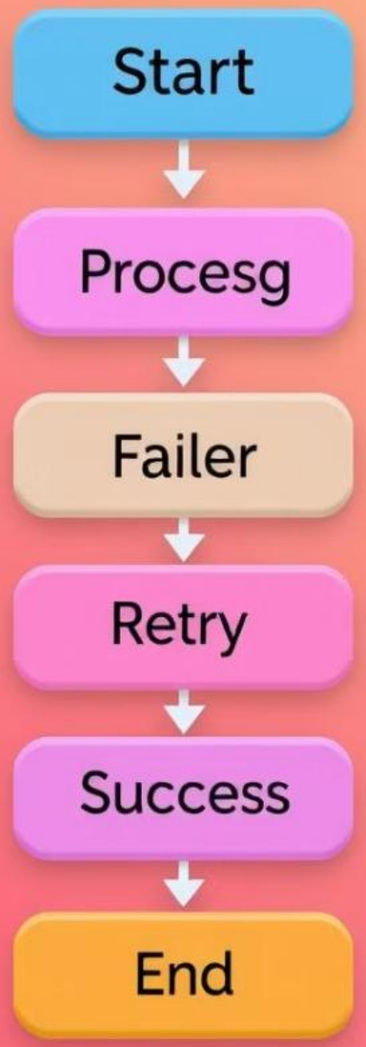


Behavioral Design Techniques

EFSM

Example Application (EFSM):
Shopping Cart System:

- States: "Browsing," "Cart Empty," "Cart Has Items," "Checkout."
- Events: Adding/removing items to/from the cart, clicking "Checkout."
- Conditions:
  - If the cart has more than 5 items, a discount might apply (condition-based transition).
  - If the user has entered a valid promo code, it applies automatically when transitioning from "Cart Has Items" to "Checkout."
- Transitions:
  - From "Cart Empty" to "Cart Has Items" when a product is added.
  - From "Cart Has Items" to "Checkout" when clicking "Proceed to Checkout, " but only if the cart is not empty.



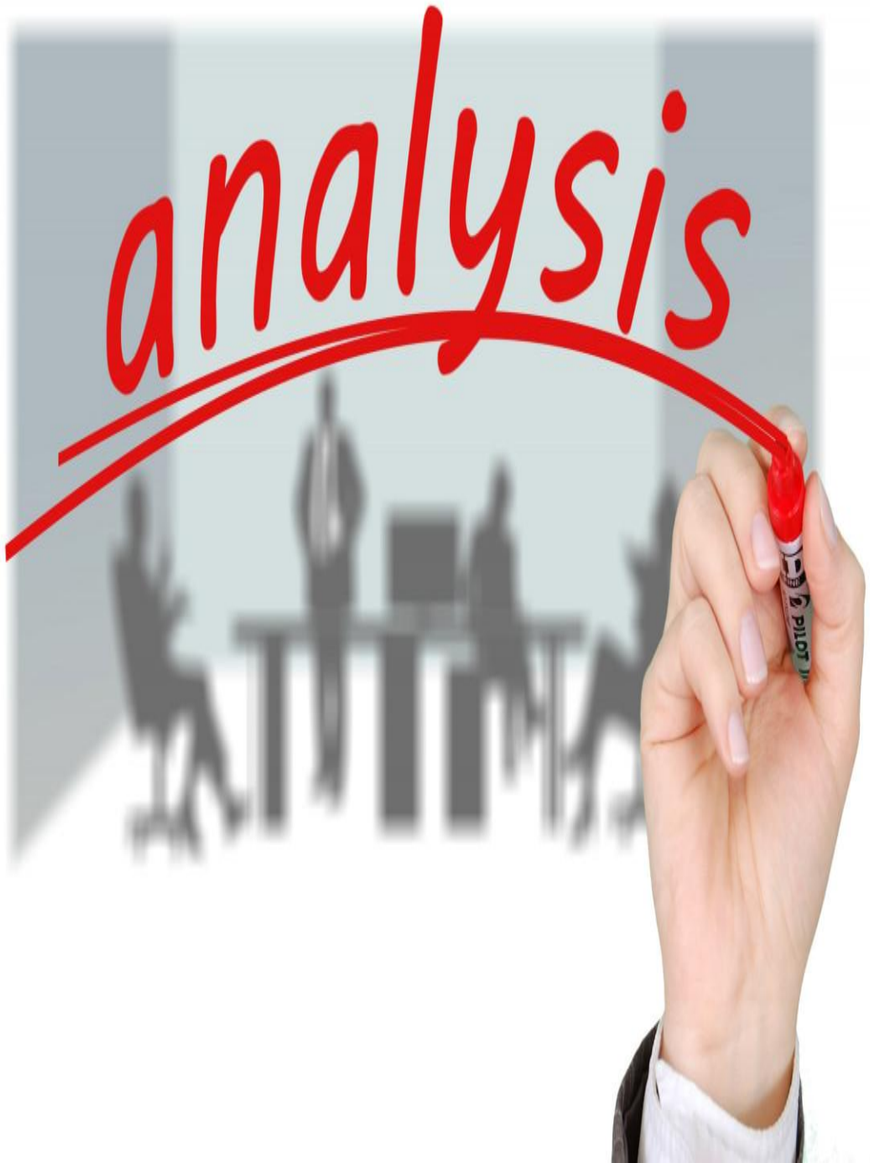
Key Differences Between FSM and EFSM:

Aspect	Finite State Machine (FSM)	Extended Finite State Machine (EFSM)
States	Fixed set of states.	Fixed set of states, but can include **dynamic conditions** or variables.
Transitions	Triggered by events.	Triggered by both events **and conditions** (e.g., data values, user inputs).
Memory	No memory of past states (stateless).	Can have memory (e.g., data variables or conditions to check).
Complexity	Simpler, deterministic transitions.	More complex, allowing for data-driven logic and conditions.
Use Case	Simple processes like user authentication.	More complex systems like shopping cart behaviors with discounts and promotions.

Improving Reliability and Effectiveness:

Area	How Data-Driven Software Improves Reliability and Effectiveness	Impact
User Personalization	- Enhance shopping experience with product recommendations based on user history data.	- **Increase conversion rates** and encourage customer retention.
	- Personalized content boosts user engagement and satisfaction.	
Performance Monitoring and Optimization	- Track real-time data to detect and resolve speed issues or website errors.	- **Improve reliability** by handling high traffic volumes effectively.
	- Optimize server load and performance through data analysis.	
User Behavior Analytics	- Analyze clicks and interactions to improve UI/UX, reducing cart abandonment rates.	- **Enhance effectiveness** by guiding users toward purchase decisions.
	- Identify friction points and areas for improvement in the user flow.	
Predictive Demand and Inventory Analytics	- Forecast product demand to manage inventory better and avoid stockouts.	- **Ensure reliability** by providing products at the right time.
	- Optimize inventory levels based on demand predictions.	
Continuous User Feedback	- Collect and analyze user feedback for quick improvements, feature optimization, and bug fixes.	- **Increase effectiveness** by improving the overall user experience and satisfaction.
	- Adapt the platform based on real-time insights from customers.	

An analysis of how software requirements for the management data analytics module were traced throughout the entire software lifecycle.



User Requirements Analysis

Search for products

Search for product names, prices, and product status.

Secure payments payments

Search for product names, prices, and product status.

Manage accounts

User login, order history, shopping cart

Identifying Stakeholders

1

Customer:

Focus on finding products, viewing product details, making purchases.
Key concerns: User experience, ease of navigation, and secure checkout process.

2

Administrator:

Responsible for inventory management, order fulfillment, and customer account management.
Key concerns: Efficiently managing products and customer data.

3

Developer:

Responsible for developing the website, ensuring it is functional, secure, and user-friendly.
Key concerns: Implementing features, maintaining systems, and ensuring data security.

User Requirements Analysis

Customer Use Cases:

1.Register an Account:

1. Description: Create a new account with name, email, password.
2. Purpose: Enable transactions and order tracking.

2.Login:

1. Description: Log in with email and password.
2. Purpose: Access shopping features and personal info.

3.Search Products:

1. Description: Search by name, category, price, etc.
2. Purpose: Find desired products easily.

4.View Product Details:

1. Description: View detailed info like description, price, images.
2. Purpose: Help in purchase decisions.

5.Add to Cart:

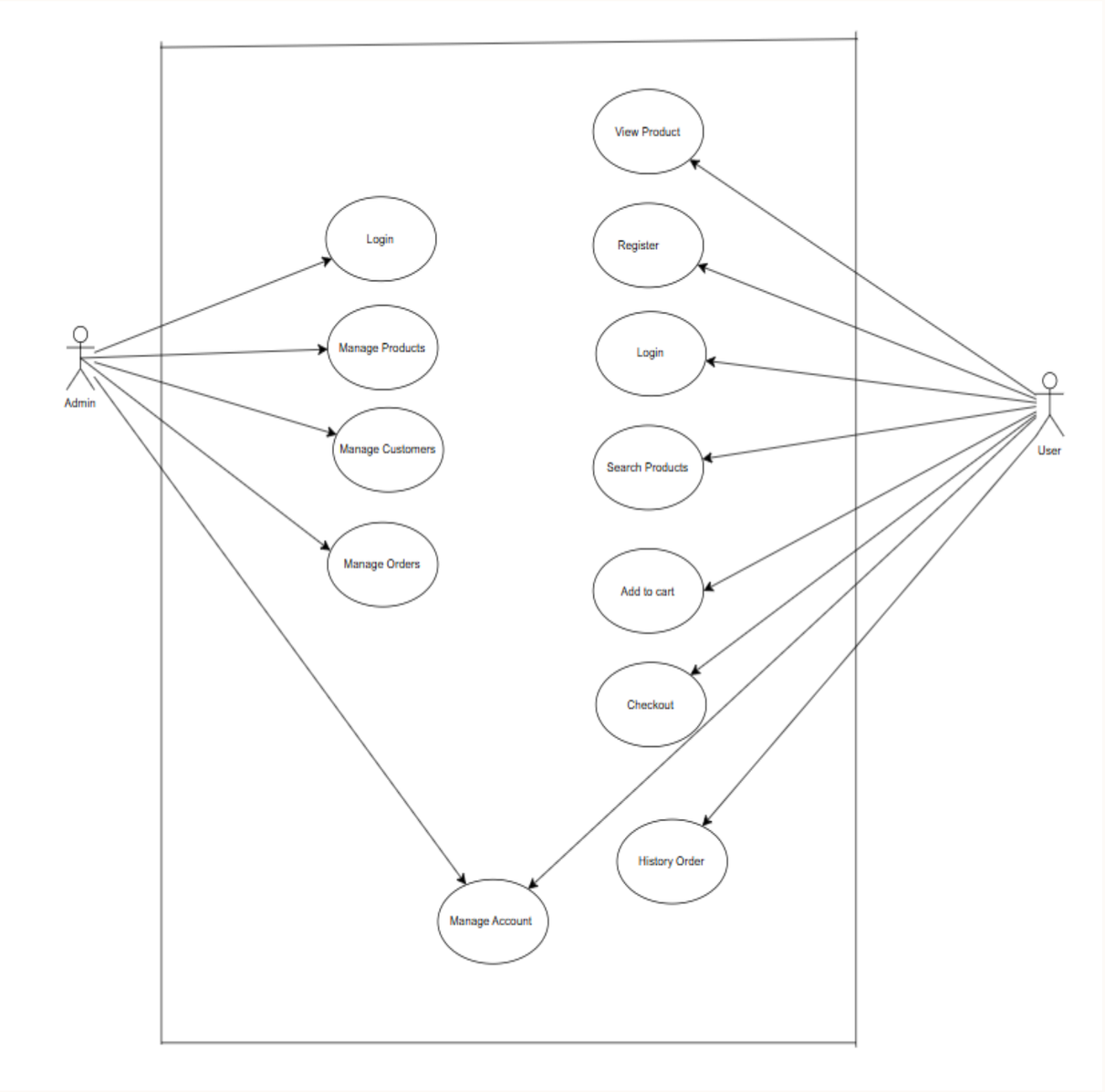
1. Description: Select and add products to cart.
2. Purpose: Save products for later.

6.Checkout:

1. Description: Proceed to payment and shipping.
2. Purpose: Finalize the purchase.

7.View Order History:

1. Description: View past orders, status, and details.
2. Purpose: Track and review past purchases.



User Requirements Analysis

Administrator Use Cases:

1.Login:

1. Description: Admin login for access to system management
2. Purpose: Secure access to admin features.

2.Manage Products:

1. Description: Add, edit, or delete products.
2. Purpose: Keep product catalog updated.

3.Manage Customers:

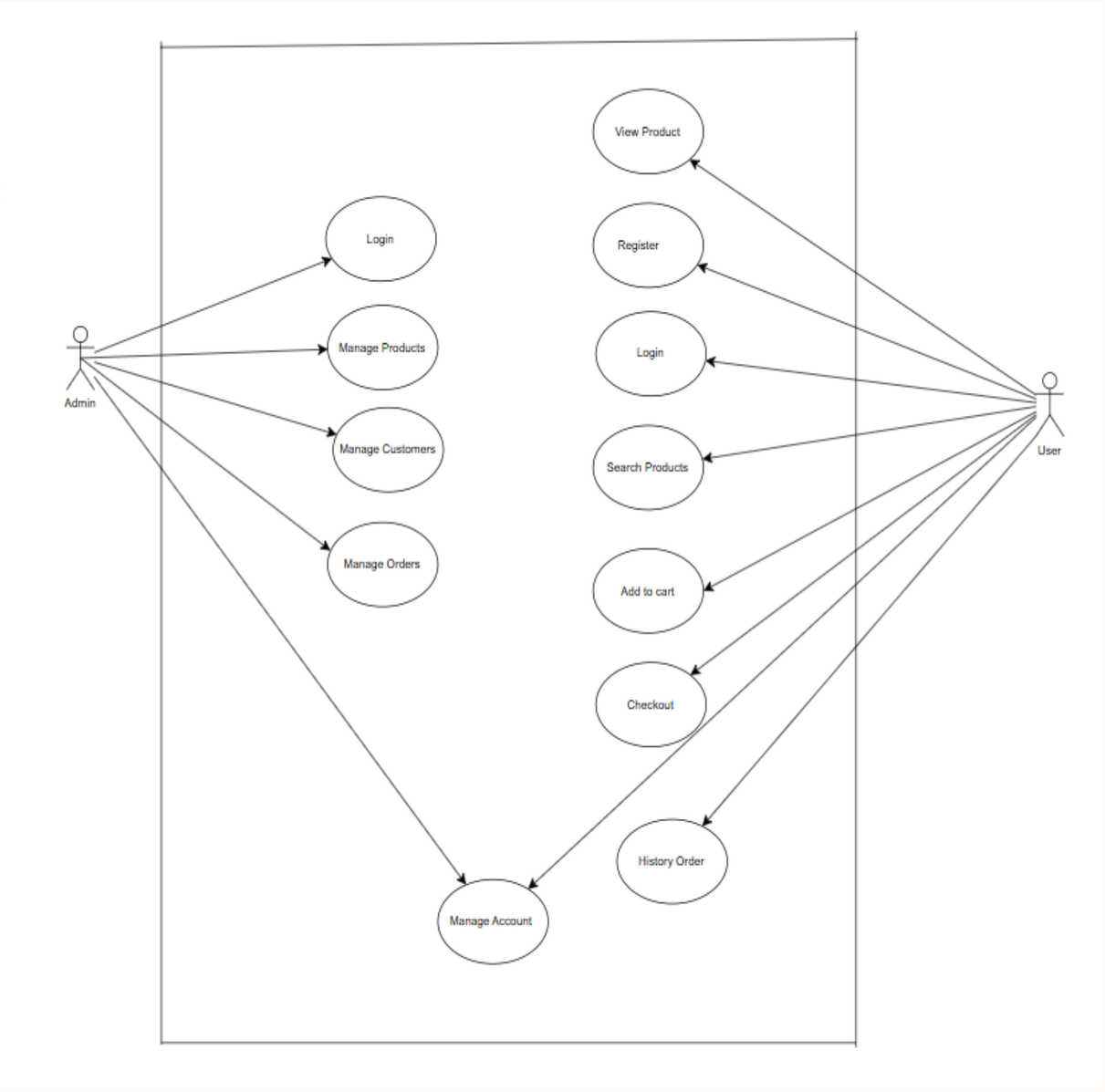
1. Description: View and manage customer information.
2. Purpose: Track and manage customer relationships.

4.Manage Orders:

1. Description: View and process customer orders.
2. Purpose: Ensure timely order processing.

5.View Revenue Reports:

1. Description: Access reports on completed orders.
2. Purpose: Monitor business performance.



Flowchart

Explanation of the Flowchart Steps:

Customer Logs In / Registers:

- Process starts with login or registration.
- Prompt to try again if login/registration fails.

Search and Select Products:

- Customer browses and selects products using search and filters.

View Product Details:

- View product description, price, images, and availability.

Add Product to Cart:

- Customer adds products to the cart for review and modification.

Proceed to Checkout:

- Customer proceeds to checkout when satisfied with the cart.

Enter Payment and Shipping Information:

- Customer provides payment details and shipping info.

Confirm Order:

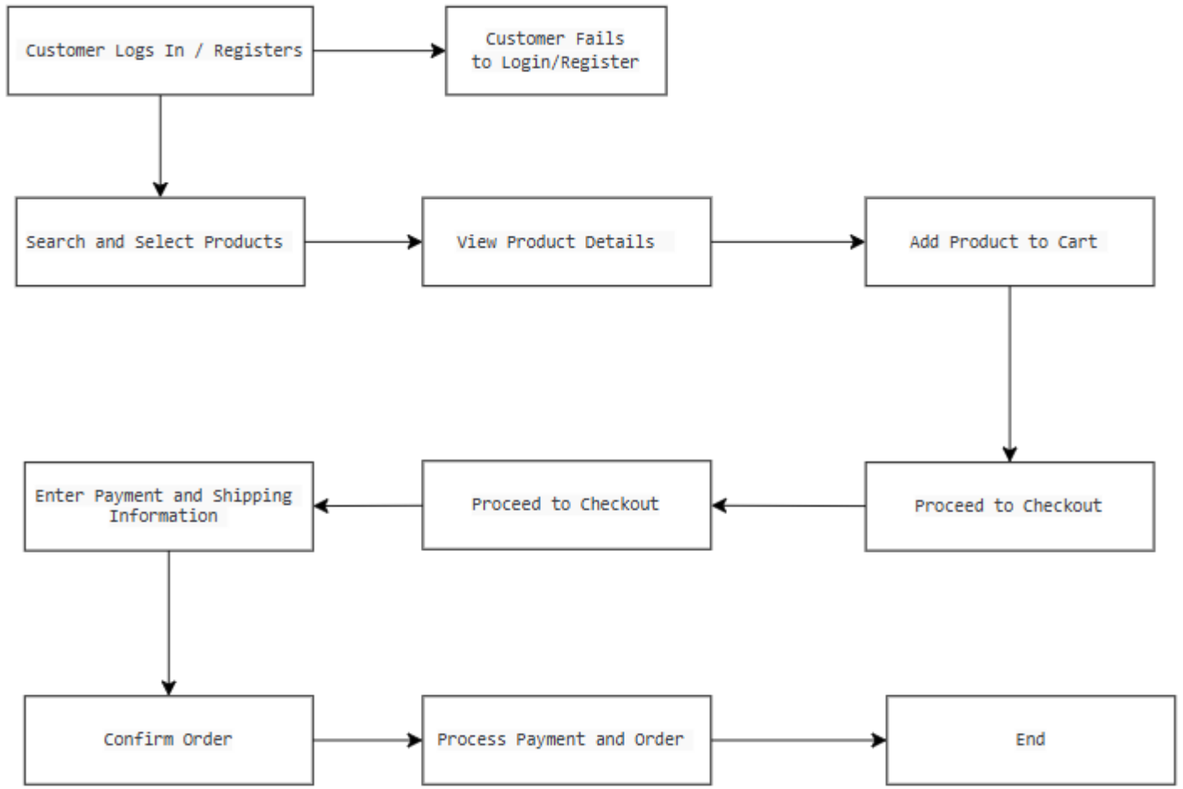
- Customer reviews and confirms the order summary.

Process Payment and Order:

- System processes payment, updates inventory, and confirms order.

End:

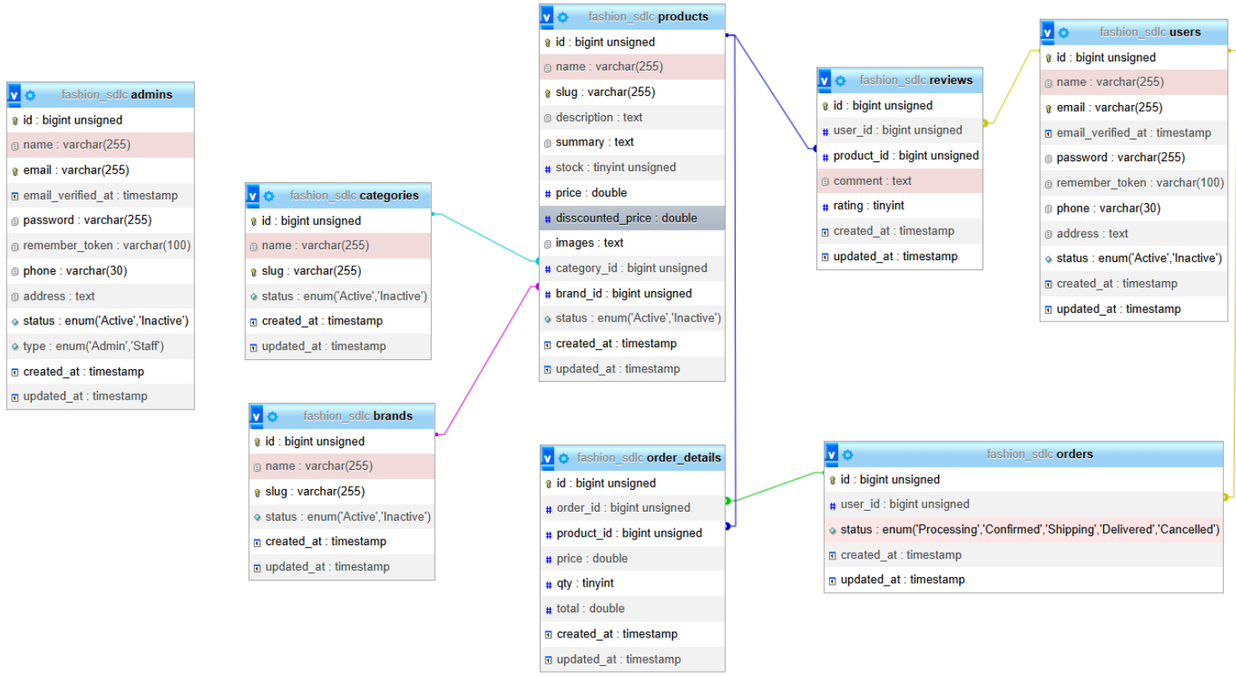
- Process ends with order confirmation sent to the customer.



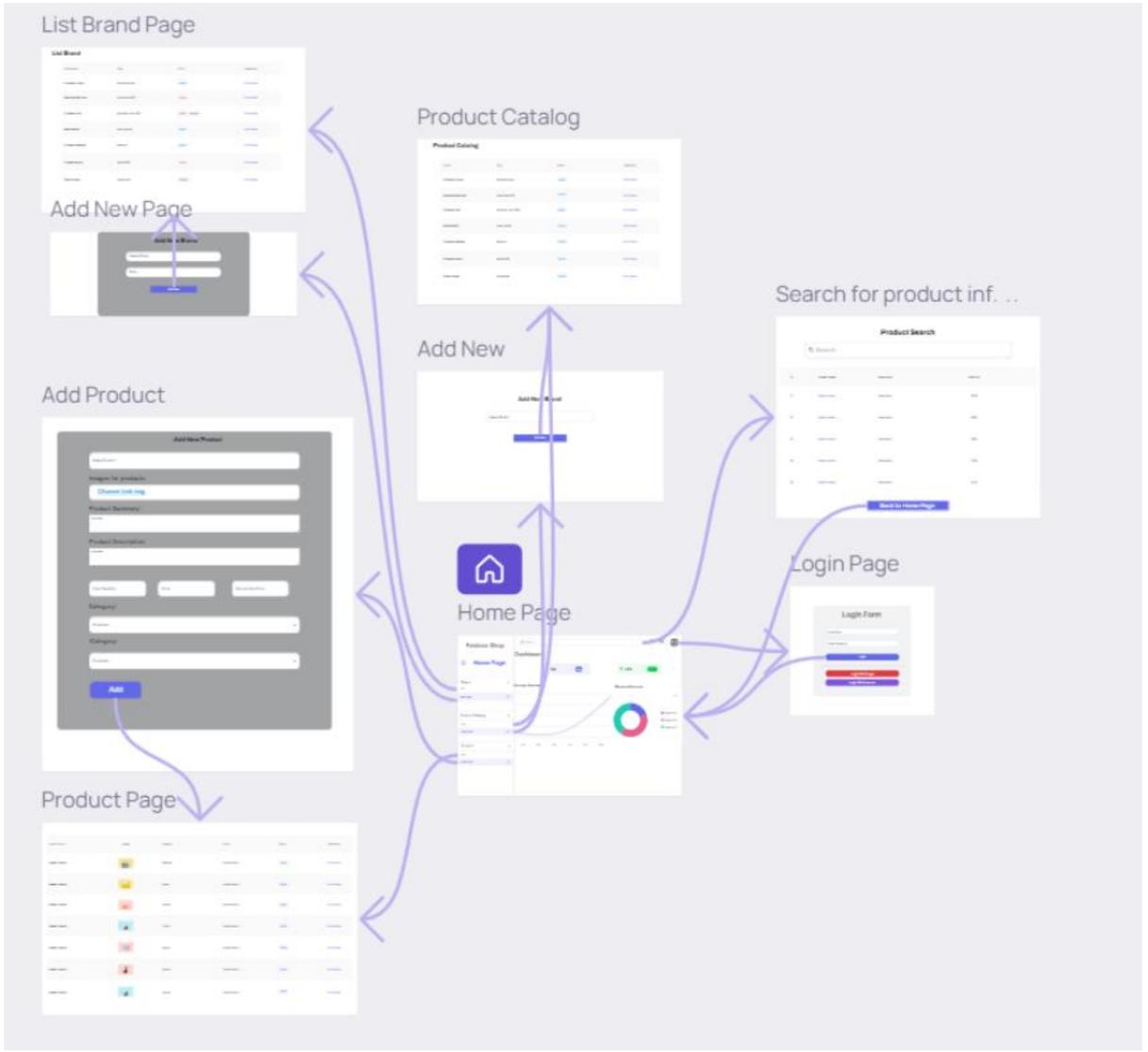
ERD

Relationships Between Tables:

1. **Customer ↔ Order** (1:N):
   - A customer can place many orders. **CustomerID** is a foreign key in the **Order** table to link with the customer.
2. **Order ↔ Product** (N:M):
   - An order can contain many products, and a product can appear in many orders. This **N:M** relationship is managed by the **OrderDetails** table, which holds details of the products in each order (such as quantity, price, etc.).
3. **Product ↔ Category** (N:1):
   - Each product belongs to a category, while a category can contain many products. **CategoryID** is a foreign key in the **Product** table to link products to their respective categories.
4. **Order ↔ Payment** (1:1 or 1:N):
   - Each order can have one or more payments, depending on the payment system. **PaymentID** in the **Order** table is a foreign key linking the order with payment information.

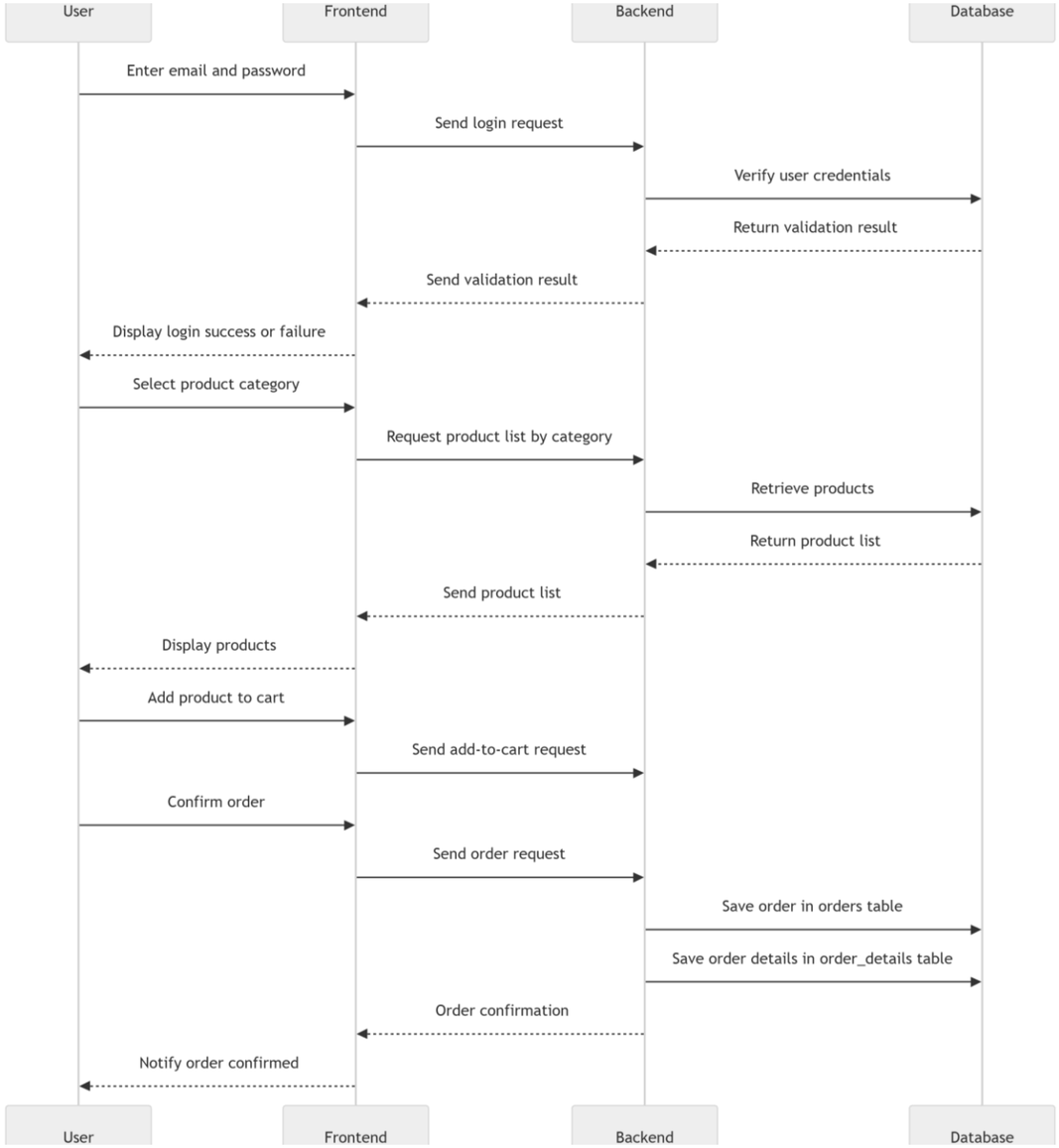


Wireframe


List Brand Page
Product Catalog
Add New Page
Search for product inf. ..
Add New
Add Product
Login Page
Home Page
Product Page

Sequence Diagram

- The sequence diagram illustrates **the process flow of handling user requests** in an e-commerce system for selling car spare parts. It includes the following components:
  - **User:** Performs actions like logging in, searching for products, adding items to the cart, and confirming orders.
  - **Frontend:** Acts as the user interface, sending requests to the backend and displaying results.
  - **Backend:** Processes business logic and serves as the intermediary between the frontend and the database.
  - **Database:** Stores information about users, products, cart, and order details.



Data Flow Diagram

Analysis of Level 0 - General DFD

Main Components:

User:

- Sends orders to the system.
- Receives notifications.

Admin:

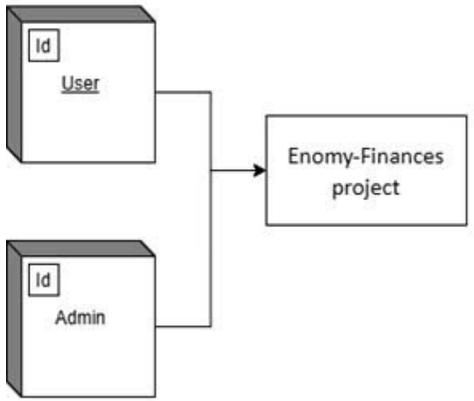
- Interacts with the system to approve actions.
- Receives notifications.

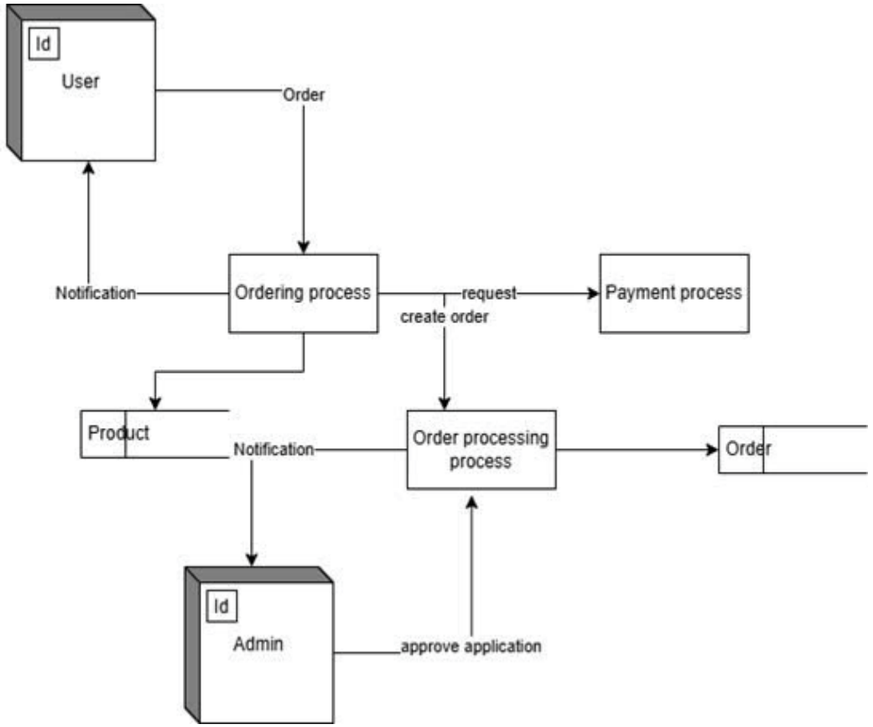
Enomy-Finances Project System:

- Represents internal processes handling data.

Data Flows:

- User → Sends order to the system.
- Admin → Receives and approves notifications.
- System → Sends notifications to User and Admin.





Data Flow Diagram

Analysis of Level 1 - Detailed DFD

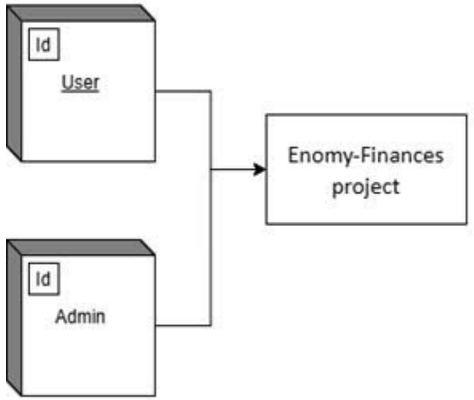
Main Components:
1.Ordering Process:
- Receives order from the User.
- Creates and sends the order to the Order Processing Process.
- Sends notifications to User.

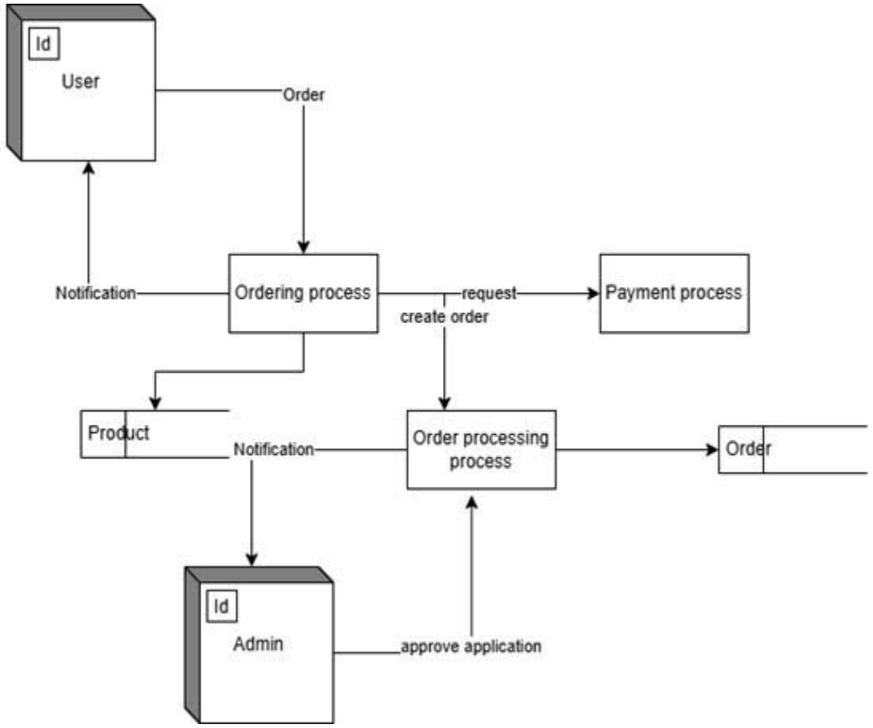
2.Order Processing Process:
- Receives order for processing.
- Sends notifications to Admin for approval.
- Receives Admin's feedback and completes order.

3.Payment Process:
- Handles payment based on the order.

4.Admin:
- Approves orders from Order Processing Process.

Data Flows:
- User → Ordering Process: Sends order request.
- Ordering Process → Payment Process: Sends payment info.
- Ordering Process → Admin: Sends approval request.
- Admin → Order Processing: Approves order.





A discussion of at least two approaches to improving software quality within a software lifecycle.

Automated Testing

Automated testing is the process of using tools and scripts to check the software’s functionality, helping to detect bugs throughout the development lifecycle. Tools like Selenium, JUnit, and Cypress are commonly used.

1 **Early Bug Detection**

Automated testing helps identify bugs as soon as software changes are made.

2 **Time and Cost Savings**

Automating testing reduces the need for manual testing, saving time and resources.

3 **Continuous Testing**

Software is tested regularly with each change, ensuring continuous quality control.

4 **Cross-Platform Testing**

Automated tests can be run across multiple platforms without requiring code modifications.

Code Review

Code review is a collaborative process where developers examine each other's code to identify potential bugs, improve code quality, and ensure adherence to coding standards. This practice is essential for catching errors early and promoting knowledge sharing within the team.

Benefits of Code Review

- Early Bug Detection
- Improved Code Quality
- Cost Savings
- Knowledge Sharing

Examples

GitHub Pull Requests allow developers to review and approve code changes before they are merged into the main branch.

Security Testing

Security testing is crucial for protecting user data and ensuring software is resistant to attacks. It evaluates software for vulnerabilities and weaknesses that could be exploited by attackers.

Security Testing Type	Description
XSS (Cross-Site Scripting)	Protects against malicious code injected by users into web applications.
SQL Injection	Prevents unauthorized access to databases by attackers.
Authentication and Session Management	Ensures secure login processes and handling of user sessions.
Data Protection	Verifies the secure encryption of sensitive data.

Conclusion

Automated testing and security testing are two essential methods for improving software quality. Automated testing helps detect errors early, saving time and costs, while ensuring the software's stability and scalability across multiple platforms. On the other hand, security testing protects user data from threats like SQL injection and XSS, safeguarding sensitive information from attacks. Integrating both methods throughout the software development lifecycle enhances efficiency, strengthens security, and improves the overall quality of the final software product.