Gym Store Project

Gym Store – Project Proposal

Overview

Gym Store is an online platform for selling fitness equipment, supplements, apparel, and accessories. The goal is to provide a smooth, secure, and efficient shopping experience for fitness enthusiasts.

Objectives

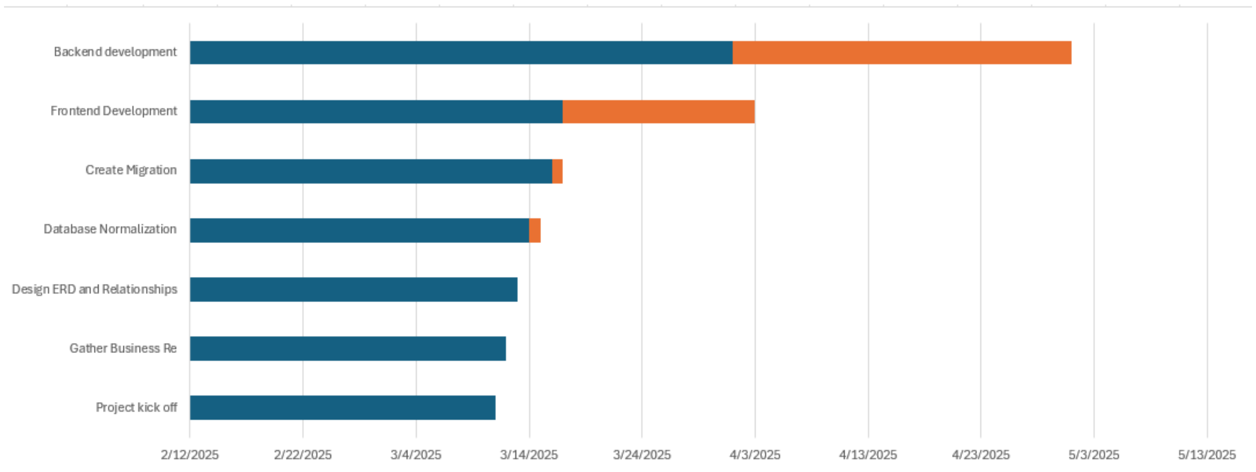
- Build a responsive and user-friendly e-commerce site.
- Enable product browsing, cart, and secure checkout.
- Improve customer satisfaction with reviews and promotions.

Scope

Develop a website with an easy-to-use interface, product management system.

Project Plan

- Timeline: period from March 9 to May 9.



Task Assignment & Roles

- Backend Developer and Frontend Developer: Mohamed Ali
- Backend Developer and Frontend Developer: Mohamed Talal
- Tester and Backend Developer: Ahmed Mohamed
- Database Designer and Frontend Developer: Belal Mohamed

Risk Assessment & Mitigation Plan

Potential Risks:

Data breaches or cyberattacks
Encrypt sensitive data using SSL and perform regular security audits to prevent unauthorized access.

Payment gateway failures
Support multiple payment methods and provide fallback options like cash on delivery.

Server downtime
Use reliable cloud hosting with auto-restart features and ensure 99.9% uptime through monitoring.

Inaccurate inventory levels
Link inventory updates to purchases and set alerts for low-stock items.

Poor user experience (UX)
Design a clean, intuitive UI and gather user feedback to continuously improve usability.

Scalability issues during high traffic
Use scalable cloud infrastructure and run stress tests to ensure system stability under load.

Shipping and delivery delays
Partner with reliable couriers and provide users with real-time tracking and estimated delivery dates.

Negative product reviews
Monitor and respond to reviews professionally; encourage satisfied customers to leave positive feedback.

Lack of regular backups
Enable daily automated backups and store them securely for quick recovery if needed.

Mitigation Plan:

1. Data breaches or cyberattacks
Use SSL encryption, validate user input, and conduct regular security audits.

2. Payment gateway failures
Support multiple payment methods and offer backup options like cash on delivery.

3. Server downtime
Use reliable cloud hosting with auto-scaling and uptime monitoring.

4. Inaccurate inventory levels
Automatically update stock after each purchase and set low-stock alerts.

5. Poor user experience (UX)
Design a simple, mobile-friendly UI and improve based on user feedback.

6. Scalability issues during high traffic
Use scalable cloud infrastructure and perform stress testing.

7. Shipping and delivery delays
Work with trusted courier services and offer real-time tracking.

8. Negative product reviews
Monitor reviews, respond professionally, and encourage happy users to rate products.

9. Lack of regular backups
Schedule daily automatic backups and store them securely.

Key Performance Indicators (KPIs)

Website Load Time: less than 4 seconds

Customer Satisfaction Score (CSAT): 95% or higher

Literature Review

Research best practices in e-commerce design

Use Cases & user experience

- As a customer I want to search for gym equipment.
- As a customer I want to add products to the shopping cart.
- As a customer I want user friendly interface.

Functional Requirements

User Management

Customers should be able to register, log in, and manage their profiles.

Admins should be able to manage users (add, update, delete).

Product Management

Admins should be able to add, update, and delete products (gym equipment, supplements, and activewear).

Products should have details like name, description, price, stock availability, and images.

Product Search & Filtering

Users should be able to search for products using keywords.

Users should be able to filter products by category, price, brand, and rating.

Shopping Cart & Checkout

Users should be able to add/remove products to/from the shopping cart.

The system should calculate the total cost, including discounts and shipping fees.

Users should be able to proceed to check out and select a payment method.

Payment Processing

The system should support multiple payment options (credit/debit cards, PayPal, cash on delivery, etc.).

The system should generate invoices for completed orders.

Order Management

Users should be able to view their order history and track order status.

Admins should be able to manage orders (approve, ship, cancel, refund).

Reviews & Ratings

Users should be able to leave reviews and rate products after purchase.

Admins should have the ability to moderate reviews.

Wishlist Feature

Users should be able to add products to their wishlist for future purchases.

Inventory Management

The system should update stock levels automatically after each purchase.

Admins should receive alerts when stock is low.

Shipping & Delivery Tracking

Users should be able to select a shipping method at checkout.

The system should provide estimated delivery times and tracking information.

Non-Functional Requirements

Performance

The website should load within 3 seconds for an optimal user experience.

The system should handle at least 500 concurrent users.

Security

All user data and transactions should be encrypted using SSL.

The system should comply with PCI DSS standards for secure payments.

Scalability

The system should support future expansion (more products, categories, and users).

Cloud hosting should be considered for scaling.

Usability

The interface should be simple, intuitive, and mobile-friendly.

Users should be able to complete a purchase within 3-5 clicks.

Availability & Reliability

The website should have 99.9% uptime.

The system should provide automated backups to prevent data loss.

Maintainability

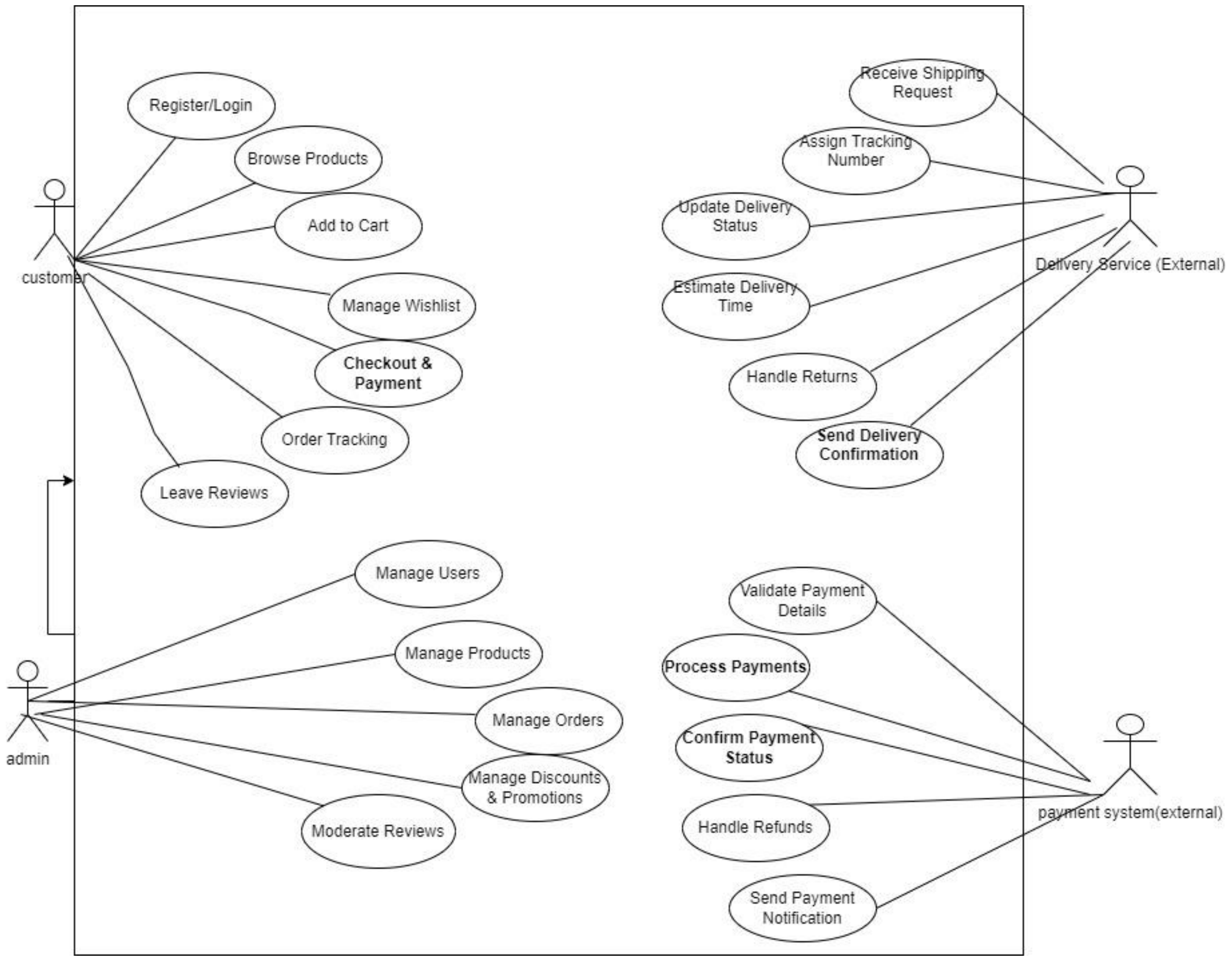
The system should have a modular code structure for easy updates and bug fixes.

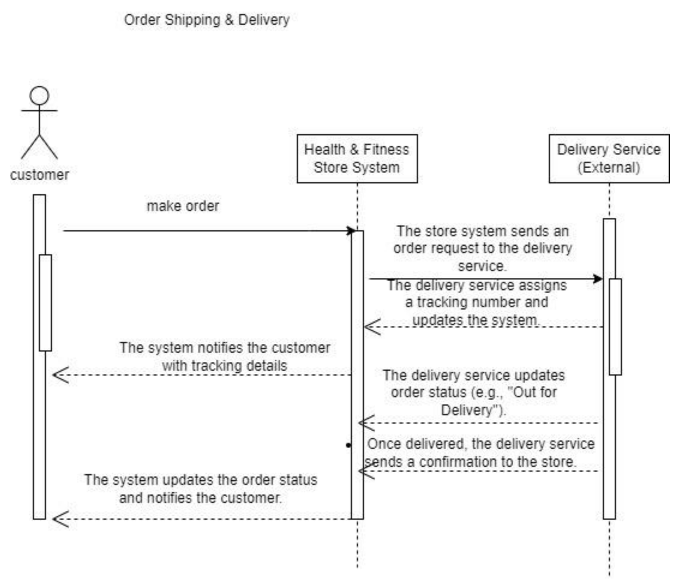
Problem Statement & Objectives

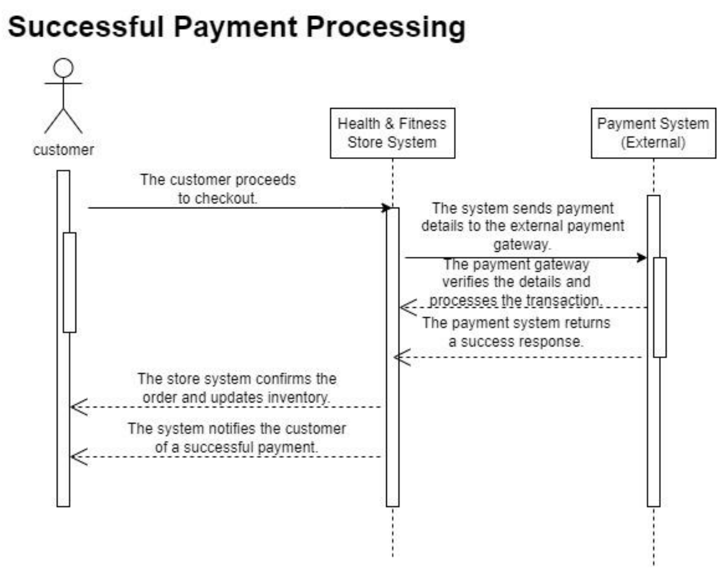
Problem: users want reliable platform.

Objective: Provide Seamless user experience and customer support.

Use Case Diagrams




Order Shipping & Delivery
customer
Health & Fitness Store System
Delivery Service (External)
make order
The store system sends an order request to the delivery service.
The delivery service assigns a tracking number and updates the system.
The system notifies the customer with tracking details
The delivery service updates order status (e.g., "Out for Delivery").
Once delivered, the delivery service sends a confirmation to the store.
The system updates the order status and notifies the customer.

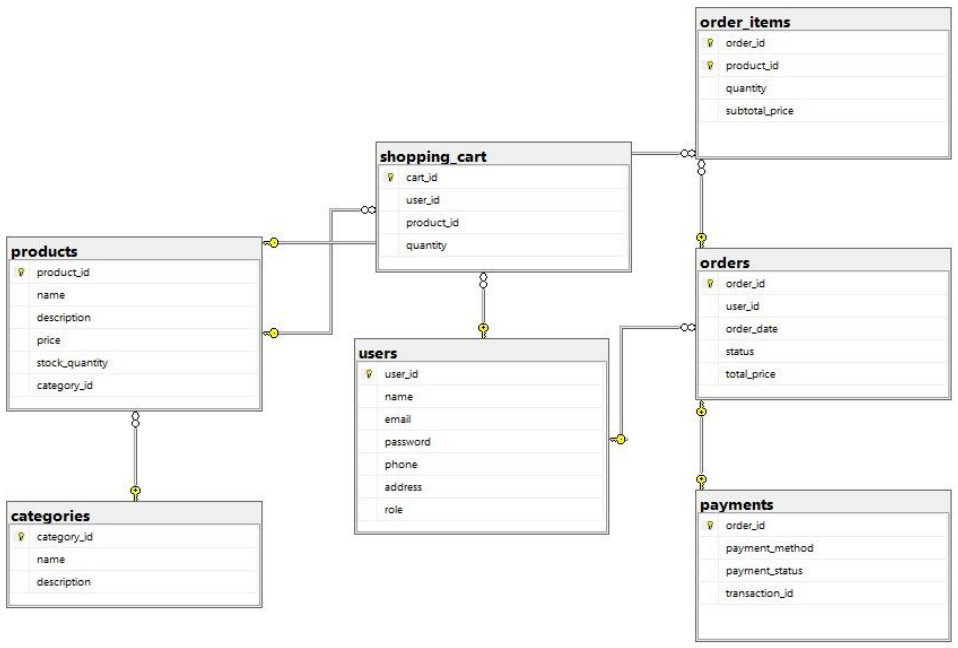

Successful Payment Processing
customer
Health & Fitness Store System
Payment System (External)
The customer proceeds to checkout.
The system sends payment details to the external payment gateway.
The payment gateway verifies the details and processes the transaction.
The payment system returns a success response.
The store system confirms the order and updates inventory.
The system notifies the customer of a successful payment.

Software Architecture

- Design system based on MVC architecture within an n-tier structure.

Database design

ERD and Logical Shema



Database Schema:

Parent Table	**Child Table**	**Relationship Type**
users	orders	**1:M** (One user can have multiple orders)
orders	payments	**1:1** (One order has only one payment)
orders	order_items	**1:M** (One order has multiple order items)
order_items	products	**M:1** (Each order item is for a single product, but a product can be in multiple orders)
categories	products	**1:M** (One category has multiple products)
users	shopping_cart	**1:M** (One user can add multiple products to the cart)
products	shopping_cart	**M:1** (A product can be in multiple carts)

Wireframes & Mockups

Design search interface, product page.

Source code

You can access the project repository on github via this link:

Security Measures

encrypt payment data.

Unit Testing

Each component (like user login, product add/edit, or cart logic) is tested in isolation to ensure it works correctly. For example, testing the AddToCart() method to confirm it adds items properly without relying on other parts of the system.

Integration Testing

Verifies that different module (e.g., frontend, backend, and database) work together seamlessly. For example, testing the full purchase flow from adding a product to the cart, checking out, and recording the order in the database.