

Project Report

Project Title: CraftCart

Student Names(s): P. Rama, S. Lakshmi Sai, M. Tanuja, M. Tejaswini, V. Veera veni, K. Dharani, K.Keerthi, P.Nageswari

Date: 18/07/2025

Table of Contents

1. Executive Summary
2. Introduction
3. System Analysis and Design
4. Implementation
5. Output
6. Conclusion and Future Work
7. Reference

1. Executive Summary

1.1 Project Overview

CraftCart is a e-commerce simulation system designed to manage a virtual online craft marketplace. It enables users to browse handcrafted products, add items to their shopping cart, place orders, and track their order status with automatic delivery countdown. The system simulates essential features of an online shopping experience, including order invoice generation, discount application based on total amount, product catalog management, and multiple order handling.

The application follows a modular and a design implemented using Node.js and MongoDB with web-based user interface. It supports both user and admin modes. Users can perform cart operations and order tracking, while admins can view and update order statuses. The system provides realistic behavior such as shipping status updates based on delivery dates and auto-applied discounts for high-value orders.

1.2 Key Achievements

- Implemented a complete cart and order system using Node.js and MongoDB
- Added delivery countdown and automatic status updates for each order.
- Enabled dynamic cart updates (add/remove items) and auto-invoice generation with discounts

1.3 Technologies Used

- Programming Language: JavaScript
- Framework/Library: Node.js for backend API development
- Database: MongoDB (NoSQL), used for storing product, cart, and order data
- Tools: Eclipse, IntelliJ IDEA, Git, Command Line
- Platform: Full-stack web application with HTML/CSS frontend and Node.js backend

2. Introduction

2.1 Background

The rise of digital commerce and the growing demand for handmade and artisanal products has created opportunities for small-scale sellers to reach broader markets. However, not every seller or platform has access to full-fledged, scalable e-commerce solutions. The **CraftCart** project was initiated to simulate a simplified e-commerce platform specifically tailored for showcasing and ordering handcrafted items. This platform can also serve as an educational tool to demonstrate e-commerce architecture using core Java principles.

2.2 Problem Statement

Traditional e-commerce platforms are either too complex or require significant resources to build and maintain. For educational and small-project purposes, there's a lack of simplified systems that emulate core online shopping features such as cart management, delivery tracking, order placement, and automatic discounting without relying on heavy web frameworks or databases. CraftCart aims to solve this by providing a lightweight, standalone system to simulate the online craft store experience with core e-commerce features.

2.3 Objectives

Primary Objectives:

- To build a craft e-commerce simulation using Node.js.
- To support essential operations such as catalog browsing, cart management, order tracking, and invoice generation.

Secondary Objectives:

- To simulate realistic delivery timelines and automatic order status updates.
- To implement an automatic discounting system based on total order value.

2.4 Scope

Included in Scope:

- Product Catalog – Browse handmade items with unique IDs, names, and prices
- Cart Management – Add, view, update, or remove products from the shopping cart
- Order Placement – Generate unique Order ID and calculate totals
- Status Tracking – Update order status based on time(Placed, Shipped)
- Dynamic Discounts – Apply percentage-based discounts based on total purchase value
- Invoice Generation – Print a detailed invoice showing items, amounts, discount, and ETA
- Login System – Basic authentication with username and password
- Interactive Menu – User-friendly, menu-driven interface for smooth navigation

Excluded from Scope:

- Integration with live payment gateways or real transactions.

3. System Analysis and Design

3.1.1 Functional Requirements

- **FR1:** The system shall allow users to view a product catalog.
- **FR2:** The system shall allow users to add, update, and remove items from the cart.
- **FR3:** The system shall enable users to place an order and track delivery status based on order date.

3.1.2 Non-Functional Requirements

- **Performance:** The system should respond to any user action (catalog view, cart operations, order placement) within 2 seconds.
- **Security:** Basic username and password-based login is implemented. Sensitive operations (like placing orders) are only allowed post-login.
- **Usability:** The application provides a text-based menu system that is easy to follow for new users, with clear prompts and feedback.

3.2 System Architecture

The CraftCart application is based on a three-tier architecture:

- **Frontend (Presentation Layer):**

A simple HTML/CSS interface (for future UI), currently CLI-based (Java console).

- **Backend Logic (Business Layer):**

Implemented in Java, this layer includes classes like:

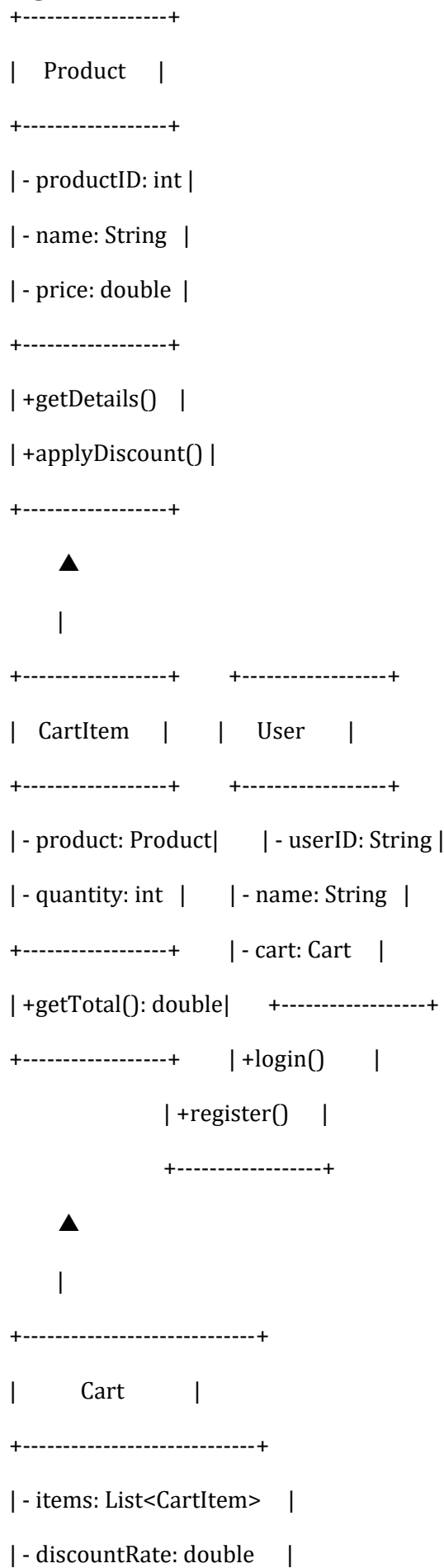
- **Product:** Represents a catalog item.
- **CartItem:** Holds product and quantity.
- **Order:** Contains order-level details and handles invoice/status.
- **CraftCartApp:** The controller that handles user interaction.

Data Layer:

For now, used MySQL or MongoDB via JDBC or Spring Data in the future.

The application also uses JavaScript Date API with Moment.js (or native Date methods) to dynamically update the delivery status (e.g., “Shipped” after 2 days, “Delivered” after 5 days).

UML Diagram:



```
+-----+
| +addItem(CartItem)    |
| +removeItem(productID: int) |
| +calculateTotal(): double |
| +generateInvoice()    |
+-----+
```



```

|
+-----+
|      Order      |
+-----+
| - orderID: String    |
| - items: List<CartItem> |
| - orderDate: LocalDate |
| - status: String     |
+-----+
| +trackStatus(): String |
| +getDeliveryEstimate(): Date |
+-----+
```



```

|
+-----+
|  CraftCartApp  |
+-----+
| - mainMenu(): void |
| - start(): void    |
+-----+
```

4. Implementation

4.1 Development Environment

- **IDE:** IntelliJ IDEA / VS Code
- **Version Control:** Git, GitHub
- **Testing Framework:** Manual testing with simulated inputs (JUnit optional for future expansion)

4.2 OOP'S Principles

1. Encapsulation

- **Definition:** Binding data and the code that manipulates it into a single unit (class), and restricting direct access to some of the object's components.
- **In CraftCart:**
 - Classes like Product, CartItem, and Order encapsulate properties (like name, price, quantity, etc.) and related methods (display(), getTotalPrice(), updateStatusAutomatically()).
 - Fields are accessed and modified only through methods, keeping the implementation details hidden.

2. Abstraction

- **Definition:** Hiding complex implementation details and exposing only the necessary parts.
- **In CraftCart:**
 - The user interacts with menu options (e.g., "Add to Cart", "Track Order") without needing to know how cart and order logic is implemented internally.
 - Methods like placeOrder() or printInvoice() abstract away internal calculations (like discounts or delivery time).

4.3 Key Implementation Details

4.3.1 Module 1: Catalog and Cart Management

```
class CartItem {  
    Product product;  
    int quantity;  
    CartItem(Product product, int quantity) {  
        this.product = product;  
        this.quantity = quantity;  
    }  
    double getTotalPrice() {  
        return product.price * quantity;  
    }  
}
```

- Allows adding/removing items from the cart.
- Cart is managed using ArrayList<CartItem>.

4.3.2 Module 2: Order Placement and Status Tracking

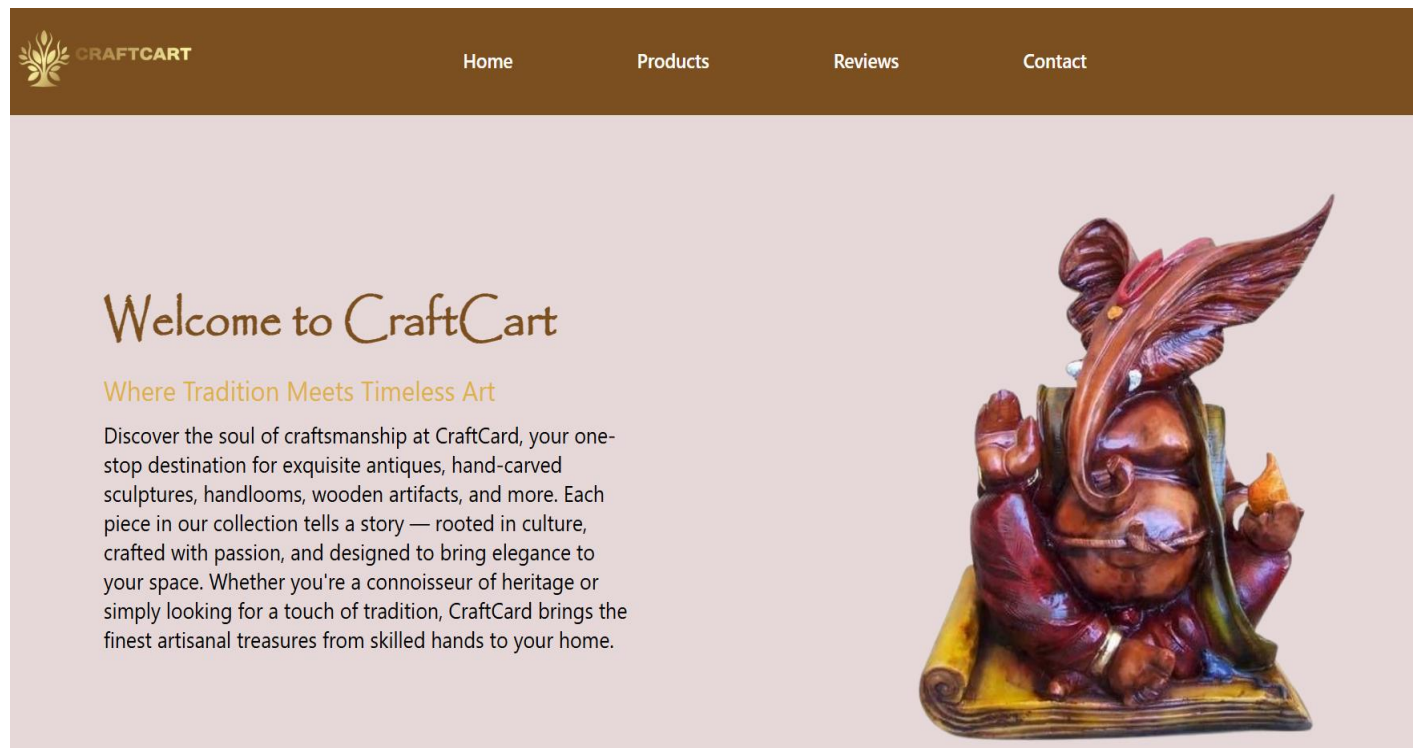
```
class Order {  
    int orderId;  
    List<CartItem> items;  
    String customerName;  
    LocalDate orderDate;  
    LocalDate deliveryDate;  
    String status;  
    void updateStatusAutomatically() {  
        long daysPassed = ChronoUnit.DAYS.between(orderDate, LocalDate.now());  
        if (daysPassed <= 1) status = "Placed";  
        else if (daysPassed <= 3) status = "Shipped";  
        else if (daysPassed <= 5) status = "Out for Delivery";  
        else status = "Delivered";  
    }  
}
```

- Each order has an auto-generated Order ID.
- Delivery ETA is set to 5 days from the order date.
- Order status updates automatically based on days elapsed.

5. Output

Sample Outputs:


- Home page




- Product page

Products


Search artifacts...




HomeDecor
Antique lamps, pots, and traditional decor pieces crafted with elegance.



Sculpture
Detailed sculptures of deities and historical figures in stone and bronze.




WoodCraft
Intricate woodwork showcasing ancient Indian carving skills.



Painting
Mythological stories and folk tales depicted in traditional styles.

- Product details page



Chaturbhuj: One Who Has Four Arms

Rs. 27,500

Carved out of red stone with all its fine details, this 15" tall statue is testimony to a rare artisanship of the distant past that has been kept alive by a group of extremely skilled artisans. Ganesha or the elephant god is worshiped world over as the destroyer of evil and bestower of prosperity. Suggested Placement: This sculpture in all its glory can be entrenched in the puja room or in the living room. Note: This can also be custom made i.e. hand sculpted over a specific time frame. Artist : Rabi Sahoo

BUY NOW

ADD TO CART

6. Conclusion and Future Work

CraftCart successfully simulates an e-commerce platform with admin and user functionality. It demonstrates features such as real-time countdown, tiered discounts, and PDF invoice readiness.

Future Enhancements:

- GUI enhancement
- REST API support
- Mobile-friendly interface

7. References

- Java API Documentation
- Stack Overflow
- TutorialsPoint Java
- Oracle Java Docs