**Project Title: *Bookstore Management System*Group 3**

**Created by:   
Latherio Kidd,**

**Johnathan Kavanaugh,**

**Cody Peterson,**

**Larry Hezekiah**

**03/28/2025**

**Last Edit: 04/06/2025**

Table of Contents

[1. Project Vision and Description 3](#_Toc71466583)

[2. Team Roles and Assignment 3](#_Toc71466584)

[3. Team Collaboration Methodology 4](#_Toc71466585)

[4. The Definition of “Done” 4](#_Toc71466586)

[5. Product Design 5](#_Toc71466587)

[6. Sprint 1 Retrospective Summary Report 5](#_Toc71466588)

[7. Sprint 2 Retrospective Summary Report 6](#_Toc71466589)

[8. References 8](#_Toc71466590)

# Project Vision and Description

**Project Vision and Description**

The Bookstore Management System is designed to automate and streamline the operations of a rare bookstore. The system replaces the paper-based process with a digital platform to improve efficiency and accuracy.

**Core Features Include:**

* Inventory Management
* Sales Record Management
* Customer Order Tracking
* Order Placement to Suppliers
* Real-Time Low-Stock Alerts
* User Authentication (Customers & Staff)
* Personalized Recommendations
* Secure Payment Gateway Integration
* Mobile Compatibility for Staff

**Expanded Scope Includes:**

* Notification System
* Role-Based Access Control
* Data Encryption
* Cloud Deployment for Scalability

# Team Roles

Each team member has been assigned a **specific Scrum role** to ensure the project runs efficiently:

## Team Roles and Assignment

Each team member is assigned a distinct role based on their strengths and contributions to the overall system scope. Below is a detailed breakdown of individual responsibilities as they relate to the Bookstore Management System:

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Responsibilities** |
| Latherio Kidd | Backend Developer | • Develop and maintain the backend architecture (Node.js or Django).  • Design and manage the database schema in PostgreSQL or MySQL.  • Implement CRUD operations and ensure backend logic aligns with business rules.  • Perform unit and integration testing.  • Coordinate with front-end for API consumption. |
| Latherio Kidd | Frontend Developer | • Integrate login forms and dynamic elements tied to backend services.  • Work closely with backend to ensure smooth front-end/back-end integration.  • Build responsive UI components using React.js or Angular.  • Help Document front-end development guidelines. |
| Johnathan Kavanaugh | Hybrid –  Frontend Dev & Product owner | • Create wireframes and design layouts for the web interface.  • Help Document front-end development guidelines.  • Assist with team collaboration to translate business needs into user stories.  • Assist in prioritizing backlog items for each sprint. |
| Cody Peterson | Product Owner | • Define and maintain the product backlog.  • Collaborate with team members to translate business needs into user stories.  • Prioritize backlog items for each sprint.  • Serve as the primary liaison for requirements clarification.  • Approve or reject deliverables based on acceptance criteria. |
| Larry Hezekiah | Scrum Master | • Facilitate Scrum ceremonies (planning, daily standups, retrospectives).  • Remove impediments to progress and ensure agile best practices.  • Monitor sprint velocity and guide workflow efficiency.  • Support communication across the team and manage collaboration tools.  • Provide quality assurance support for early sprint reviews. |
| ------------------ | ----------------- | ------------------ |
| Latherio Kidd | Developer | Frontend/backend development, database setup, unit testing |
| Johnathan Kavanaugh | Hybrid –  Developer & Product Owner | Documentation, wireframes, clarifies requirements, maintains backlog |
| Cody Peterson | Product Owner | Maintains backlog, prioritizes features, clarifies requirements |
| Larry Hezekiah | Scrum Master | Facilitates ceremonies, ensures Agile practices, supports team operations |

# Collaboration Methodology

Our team follows the Scrum framework, emphasizing iterative development and continuous feedback.

**Processes:**

* Sprint Planning (bi-weekly)
* Daily Standups (async updates in Teams/Discord)
* Sprint Reviews & Retrospectives

**Tools:**

* Trello for task tracking
* GitHub for version control
* MS Teams / Discord for communication
* Google Drive for document collaboration
* CTU Email and Messenger for academic communication

# The Definition of “Done”

## ****4.**** A task is considered "done" when:

## It meets all functional/non-functional requirements

## It passes all forms of testing (unit, integration, system)

## It is peer-reviewed and merged

## It includes supporting documentation

## It is approved by the Product Owner

# Product Design

**Architecture:**

* Front-End: React.js or Angular
* Back-End: Node.js or Django
* Database: PostgreSQL or MySQL

**Design Artifacts:**

* Use Case Diagram: Details user interactions across the system (e.g., logging in, managing inventory, placing orders).
* Activity Diagram: Displays the flow of activities from user actions to system processing.
* Class Diagram: Shows relationships between entities such as Book, Order, Customer, and Employee.

**Security & Non-Functional Requirements:**

* Authentication & Authorization: Role-based access for employees and customers
* Data Protection: Encrypt sensitive user and payment data
* Scalability: Enable cloud-based auto-scaling
* Performance: Implement caching for fast responses

**Use Case Diagram**

The Use Case Diagram represents the core functionalities of the Bookstore Management System and how users interact with them. Below are detailed actors and their corresponding use cases:

**Actors:**

* Customer: Interacts with the storefront, places orders, creates accounts.
* Employee: Manages inventory, fulfills orders, updates book entries.
* Manager/Admin: Oversees sales reports, user access, and restock levels.
* System (internal actor): Handles notifications, authentication, and payment processes.

**Primary Use Cases:**

* Register/Login (Customer, Employee): Authenticate using secure credentials.
* Browse Inventory (Customer): View available books, filter by category.
* Add to Cart (Customer): Queue books for purchase.
* Place Order (Customer): Submit orders using integrated payment gateway.
* View Order Status (Customer): Check shipping status, history.
* Manage Inventory (Employee): Add, edit, or delete book entries.
* Generate Reports (Manager): View sales data, stock levels, order frequency.
* Receive Notifications (System): Alert managers on low stock or failed transactions.
* Authorize Access (System): Enforce RBAC policies depending on user roles.

**Activity Diagram**

This diagram outlines the high-level business processes within the system and how they flow from user interaction to backend processing.

**Primary Activities:**

1. **Customer Browses Inventory** → Filters/searches for books → Adds to cart → Initiates checkout.
2. **System Validates Cart** → Applies promotions/taxes → Sends data to payment gateway.
3. **Customer Completes Purchase** → System sends order to backend → Inventory updated.
4. **Employee Receives Order** → Prepares package → Marks order as shipped.
5. **Manager Reviews Reports** → Views sales data, out-of-stock alerts → Initiates restock orders.
6. **System Sends Notifications** (email/SMS) to customers and suppliers based on triggers.

**Decision Points:**

* Is the item in stock?
* Was payment successful?
* Does the order require manual approval?

**Endpoints:**

* Confirmation page for customers
* Admin dashboards
* REST API logs for analytics

**Class Diagram**

This diagram identifies and defines the core entities of the system and their relationships.

**Classes:**

* **User** (Customer, Employee, Admin) → Attributes: userID, role, email, passwordHash
* **Book** → Attributes: ISBN, title, author, stock, price, category
* **Order** → Attributes: orderID, userID, bookList[], totalPrice, status, timestamp
* **Cart** → Attributes: cartID, userID, items[], subtotal
* **Payment** → Attributes: paymentID, orderID, amount, status, method
* **Notification** → Attributes: type, targetUser, messageContent, triggerEvent

**Relationships:**

* One-to-many: User → Orders
* Many-to-many: Books ↔ Orders (via OrderItem or BookList)
* One-to-one: Order → Payment
* One-to-many: User → Notifications

**Architecture:**

* Front-End: React.js or Angular
* Back-End: Node.js or Django
* Database: PostgreSQL or MySQL

**Design Artifacts:**

* Use Case Diagram: Details user interactions across the system (e.g., logging in, managing inventory, placing orders).
* Activity Diagram: Displays the flow of activities from user actions to system processing.
* Class Diagram: Shows relationships between entities such as Book, Order, Customer, and Employee.

**Security & Non-Functional Requirements:**

* Authentication & Authorization: Role-based access for employees and customers
* Data Protection: Encrypt sensitive user and payment data
* Scalability: Enable cloud-based auto-scaling
* Performance: Implement caching for fast responses

# Sprint 1 Retrospective Summary Report

|  |
| --- |
| **Things That Went Well** |
| * Clear role assignments * Communication channels established * Initial setup completed on schedule * Backend completed ahead of time * Front-end development made strong progress on layout and wireframes * Team members were responsive and proactive in offering help |
| **Things That Could Have Gone Better** |
| * The onboarding process for new team members took extra time * Some features underestimated in scope * Front-end/backend handoff was not clearly defined * Some tasks lacked clear definitions or acceptance criteria * Some issues with understanding team member capabilities * Some issues with connecting front and backends |
| **Things That Surprised Us** |
| * Backend development moved faster than anticipated * Integration issues arose due to unclear file handoffs * Not all planned test coverage was achieved by the end of the sprint * Having to re-work team roles & responsibilities to best fit the project needs |
| **Lessons Learned** |
| * Weekly check-ins should be formalized to maintain sync * Break large user stories into smaller tasks to improve estimations * Ensure clearer alignment on integration responsibilities between frontend and backend * Start testing earlier in the sprint to avoid end-loading QA work * Finalize and share UI components early to reduce integration delays |

# Sprint 2 Retrospective Summary Report

|  |
| --- |
| **Things That Went Well** |
|  |
| **Things That Could Have Gone Better** |
|  |
| **Things That Surprised Us** |
|  |
| **Lessons Learned** |
|  |

# References

**Flewelling, P. (2017). *The Agile Developer's Handbook: Get More Value from Your Software Development: Get the Best Out of the Agile Methodology*. Packt Publishing.**

**Nutcache. (n.d.). Agile - Scrum Sprint Definition, Roles, and Planning. Retrieved from** [**https://www.nutcache.com/blog/agile-scrum-sprint/**](https://www.nutcache.com/blog/agile-scrum-sprint/)

**Scrum.org. (n.d.). Scrum Value: Respect. Retrieved from** [**https://www.scrum.org/resources/scrum-value-respect**](https://www.scrum.org/resources/scrum-value-respect)

**Scrum.org. (n.d.). What is a Scrum Development Team? Retrieved from** [**https://www.scrum.org/resources/what-is-a-scrum-development-team**](https://www.scrum.org/resources/what-is-a-scrum-development-team)

**Simplilearn. (n.d.). Top Scrum Values and Principles. Retrieved from** [**https://www.simplilearn.com/scrum-values-and-principles-article**](https://www.simplilearn.com/scrum-values-and-principles-article)

**Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Retrieved from** [**https://scrumguides.org/scrum-guide.html**](https://scrumguides.org/scrum-guide.html)

**Scrum.org. (n.d.). The Importance of Product Backlog Transparency. Retrieved from** [**https://www.scrum.org/resources/importance-product-backlog-transparency**](https://www.scrum.org/resources/importance-product-backlog-transparency)