**1.JIRA Board Kanban board**

En bild som visar text, nummer, programvara, Teckensnitt

AI-genererat innehåll kan vara felaktigt.

En bild som visar text, nummer, programvara, skärmbild

AI-genererat innehåll kan vara felaktigt.

**2. Project Description:**

**Name**: "**BookBuddies**" – A Book Rental and Review System  
The platform allows users to rent books, write reviews, and exchange them with others. The system will provide features like user authentication, book catalog browsing, book renting, and review management.

**Background and Purpose:**

With the rising cost of purchasing books and the environmental impact of paper production, renting books provides a sustainable and cost-effective alternative. Many book lovers also struggle to find trustworthy reviews, so a platform that combines rentals with a review system will meet an essential market need.

* **Why this project is important**:
  + Sustainability: Users get access to a wide variety of books without the need to purchase them.
  + Accessibility: More affordable than buying new books.
  + Community: Users can share thoughts and ratings on books they read.

**Clear Goals:**

* **User Authentication**: Allow users to sign up, log in, and manage their profiles.
* **Book Catalog**: Display books available for rent, including search and filter features.
* **Book Rental System**: Enable users to rent books, keep track of rented books, and set rental durations.
* **Review and Rating System**: Allow users to write reviews and rate books they’ve rented.
* **User Interaction**: Enable users to interact with each other through comments, likes, and recommendations.

**Success Metrics:**

* **Feature Completion**: Completion of all key features (Authentication, Catalog, Rental System, Reviews).
* **User Engagement**: Number of active users and their participation in the community (e.g., reviews written, books rented).
* **Platform Performance**: Speed, reliability, and responsiveness of the system.
* **User Satisfaction**: Positive feedback through surveys or user ratings on the platform.

**Stakeholder Mapping**

**Customer:**

* **Who**: The primary customers are readers, students, or anyone interested in books but hesitant to purchase them due to high costs.
* **Role**: They are the end-users of the system who will utilize the rental features and participate in reviews.

**End User:**

* **Who**: Individuals who enjoy reading, borrowing, and sharing thoughts about books.
* **Role**: They will interact with the system by renting books, reading reviews, writing their own, and rating books.

**Other Stakeholders:**

* **Developers**:
  + **Who**: The software engineers responsible for implementing features.
  + **Role**: Design, develop, and maintain the platform.
* **Testers**:
  + **Who**: Quality assurance personnel.
  + **Role**: Ensure the system works as expected, perform testing for bugs, and ensure smooth user experience.
* **Admin**:
  + **Who**: Platform administrators who manage user accounts, books, and platform content.
  + **Role**: Monitor user behavior, approve reviews, and manage book listings.

**3. Requirements Specification**

**Functional Requirements**

These describe the specific behaviours and functions that the system must be able to perform.

1. **User Authentication**
   * The system must allow users to register, log in, and log out.
   * Users should be able to reset passwords and receive email confirmation for account creation.
2. **Book Catalog**
   * The system must display a list of available books, including book titles, authors, categories, and short descriptions.
   * Users must be able to search, filter, and sort the catalog by different parameters (e.g., genre, author, rating, etc.).
3. **Book Rental**
   * Users should be able to rent books for a predefined duration.
   * The system must track the rental status (rented, returned) and notify users about upcoming rental expirations.
4. **Book Review and Rating System**
   * Users must be able to submit ratings (1-5 stars) and write reviews for books they have rented.
   * The system should display the average rating and user reviews for each book in the catalog.
5. **User Profile Management**
   * Users should be able to manage their profiles, including updating personal details, viewing rental history, and managing their reviews.
   * Users should be able to view and edit their account settings.
6. **Admin Management of Books and Users**
   * Admin users must be able to add, update, and delete books in the catalog.
   * Admins should be able to monitor and manage user accounts, including banning abusive users or deleting inappropriate reviews.
7. **Book Availability Tracking**
   * The system must track the availability of books and update the status of books in real time (e.g., available, rented, reserved).
   * The system should notify users when a previously unavailable book is returned and available for rent.

**Non-Functional Requirements**

These describe the qualities of the system, such as performance, security, and usability.

1. **Performance**
   * **Requirement**: The system must load the catalog and book details within 3 seconds under normal load.
   * **Why important**: Quick loading times improve user experience and reduce bounce rates. Slow systems can lead to user frustration and loss of engagement.
   * **How to fulfill**: Optimize database queries, use content delivery networks (CDNs) for static resources, and ensure server-side optimization.
2. **Security**
   * **Requirement**: The system must implement secure login processes, including encryption of passwords and use of HTTPS.
   * **Why important**: Protecting user data and ensuring that sensitive information (like passwords) is securely stored is crucial for trust and to prevent unauthorized access.
   * **How to fulfill**: Use strong encryption algorithms like bcrypt for password hashing, and enforce HTTPS for secure data transfer.
3. **Usability**
   * **Requirement**: The system should provide an intuitive and responsive user interface that works well on both desktop and mobile devices.
   * **Why important**: A user-friendly and responsive design is essential for engaging a broad user base and ensuring the platform can be used easily on various devices.
   * **How to fulfill**: Follow best practices for UI/UX design, ensuring simple navigation, consistent layout, and responsiveness across devices.
4. **Scalability**
   * **Requirement**: The system must be scalable to support a growing number of users and books without a significant decrease in performance.
   * **Why important**: As the platform grows, the system must handle more users and content without slowdowns or downtime.
   * **How to fulfill**: Use cloud infrastructure with auto-scaling capabilities, design a modular architecture, and optimize database design for performance.
5. **Availability**
   * **Requirement**: The system must ensure 99.9% uptime with minimal downtime during maintenance periods.
   * **Why important**: Users expect the system to be available whenever they want to access it. Downtime can lead to frustration and loss of trust.
   * **How to fulfill**: Use a reliable hosting service, implement load balancing, and establish a robust monitoring and alert system.

**Prioritization of Requirements (MoSCoW Method)**

1. **Must Have**: Essential features for the system to be operational and provide basic functionality.
   * User Authentication
   * Book Catalog
   * Book Rental System
   * Book Review and Rating System
2. **Should Have**: Important features that enhance user experience but are not critical for basic operation.
   * Admin Management of Books and Users
   * Book Availability Tracking
   * User Profile Management
3. **Could Have**: Features that are nice to have but not critical for the first release. These can be added if time and resources permit.
   * Enhanced search and filter options (e.g., based on user preferences)
   * Social media sharing options for book reviews
4. **Won't Have (for now)**: Features that will not be included in the current version but could be considered for future updates.
   * Social media integrations (like login with Facebook or Google)
   * Advanced analytics for book recommendations

**User Stories**

1. **User Story 1**:
   * **As a** user,
   * **I want to** register an account,
   * **So that I can** access the system and rent books.
   * **Acceptance Criteria**:
     + The user can sign up with an email and password.
     + The user receives an email confirmation after registration.
     + The user can log in using the registered email and password.
2. **User Story 2**:
   * **As a** user,
   * **I want to** rent books,
   * **So that I can** read them without purchasing them.
   * **Acceptance Criteria**:
     + The user can select a book from the catalog and choose a rental period.
     + The user receives a confirmation of the rental with the return date.
     + The rental status is updated in the user's profile.
3. **User Story 3**:
   * **As a** user,
   * **I want to** write reviews and rate books,
   * **So that I can** share my thoughts with other readers.
   * **Acceptance Criteria**:
     + The user can submit a rating (1-5 stars) and a review for any rented book.
     + Reviews can be edited or deleted by the user.
     + The book's average rating is updated accordingly.
4. **User Story 4**:
   * **As an** admin,
   * **I want to** manage the book catalog,
   * **So that I can** add new books, update existing ones, or remove books from the system.
   * **Acceptance Criteria**:
     + The admin can add books by filling out a form with title, author, category, and description.
     + The admin can edit or delete books from the catalog.

**Use Cases**

1. **Use Case 1: User Registration**
   * **Pre-condition**: The user has access to the registration page.
   * **Main Flow**:
     1. The user provides their email address and password.
     2. The system validates the input and checks for an existing account.
     3. If no account exists, the system creates a new user account and sends a confirmation email.
     4. The user logs in with the registered credentials.
   * **Post-condition**: The user account is created, and the user is authenticated.
   * **Alternative Flow**:
     1. If the email is already registered, the system notifies the user and offers a password recovery option.
2. **Use Case 2: Renting a Book**
   * **Pre-condition**: The user is logged in and browsing the book catalog.
   * **Main Flow**:
     1. The user selects a book and chooses a rental period.
     2. The system checks if the book is available.
     3. If available, the system confirms the rental and deducts the available duration from the user’s rental quota.
     4. The system updates the book status to "rented."
   * **Post-condition**: The user successfully rents the book, and the rental details are recorded in the user's profile.
   * **Alternative Flow**:
     1. If the book is unavailable, the system informs the user and offers to notify them when it becomes available.

**4. Object-Oriented Modeling**

**1. Domain Model / UML Class Diagram**

In this section, we'll focus on the core domain classes for the Book Rental and Review System. The classes will represent different objects in the system and their relationships.

**Classes in the Domain:**

1. **User**
   * **Attributes**:
     + UserID: int
     + Email: string
     + Password: string
     + Name: string
     + Phone: string
     + RentalHistory: List<Rental>
     + Reviews: List<Review>
   * **Methods**:
     + Register()
     + Login()
     + UpdateProfile()
2. **Book**
   * **Attributes**:
     + BookID: int
     + Title: string
     + Author: string
     + Category: string
     + Description: string
     + AverageRating: float
     + AvailableCopies: int
   * **Methods**:
     + GetAvailability()
     + UpdateAvailability()
3. **Rental**
   * **Attributes**:
     + RentalID: int
     + User: User (Association with User)
     + Book: Book (Association with Book)
     + RentalDate: DateTime
     + ReturnDate: DateTime
     + Status: string (e.g., "Active", "Returned")
   * **Methods**:
     + CalculateRentalPeriod()
     + IsOverdue()
4. **Review**
   * **Attributes**:
     + ReviewID: int
     + User: User (Association with User)
     + Book: Book (Association with Book)
     + Rating: int (1-5 scale)
     + Comment: string
     + Date: DateTime
   * **Methods**:
     + EditReview()
     + DeleteReview()
5. **Admin** (Inherits from User)
   * **Attributes**:
     + AdminID: int
   * **Methods**:
     + AddBook()
     + RemoveBook()
     + ManageUserAccounts()
6. **Payment**
   * **Attributes**:
     + PaymentID: int
     + User: User (Association with User)
     + Amount: decimal
     + PaymentDate: DateTime
     + Status: string (e.g., "Completed", "Failed")
   * **Methods**:
     + ProcessPayment()
     + RefundPayment()

**Relationships and Multiplicities:**

* **User ↔ Rental**: A user can have multiple rentals (1..\*), while a rental is linked to exactly one user (0..1).
* **User ↔ Review**: A user can write multiple reviews (1..\*), and each review is written by one user (0..1).
* **Book ↔ Rental**: A book can be rented multiple times (0..\*), and each rental is associated with exactly one book (1..1).
* **Book ↔ Review**: A book can have multiple reviews (0..\*), and each review corresponds to exactly one book (1..1).
* **Admin ↔ Book**: An admin can add/remove many books (0..*), and each book can be managed by multiple admins, although it's typically one admin for each action (0..*).
* **Payment ↔ User**: A user can make multiple payments (1..\*), and each payment is linked to exactly one user (1..1).

1. **UML Class Diagram:**

**A diagram of a computer

AI-generated content may be incorrect.**

**2. Advanced UML Diagram: Sequence Diagram**

To illustrate a core system flow, we'll create a **Sequence Diagram** for the "Renting a Book" use case.

**Sequence Diagram: Renting a Book**

This sequence diagram illustrates the interaction between the user and the system when a book is rented.

A diagram of a system

AI-generated content may be incorrect.

**Description of Sequence Flow:**

1. The user sends a rental request to the system.
2. The system checks if the book is available and responds with the availability status.
3. The user confirms the rental.
4. The system processes the payment by interacting with the payment system.
5. Once the payment is processed, the rental status is updated, and the user receives confirmation.

**1. Timetable / Milestones**

The project is divided into four sprints, each lasting 2 weeks, totalling 8 weeks.

|  |  |  |
| --- | --- | --- |
| **Sprint** | **Tasks / Milestones** | **Duration** |
| **Sprint 1: Planning & Requirements Gathering** | - Define project scope and goals. | Week 1–2 |
| - Gather and document requirements. |
| - Create initial wireframes and ER diagrams. |
| **Sprint 2: System Design & Database Setup** | - Design system architecture and database schema. | Week 3–4 |
| - Create UML diagrams. |
| - Implement authentication (user registration, login). |
| **Sprint 3: Core Feature Development** | - Develop book catalog browsing & search. | Week 5–6 |
| - Implement rental system. |
| - Develop review and rating system. |
| - Build admin panel for book/user management. |
| **Sprint 4: Testing, Deployment & Documentation** | - Perform unit and integration testing. | Week 7–8 |
| - Fix bugs and optimize performance. |
| - Deploy the system to a cloud platform. |
| - Finalize project documentation. |

**2. Work Process**

We will follow a **Scrum-like Agile approach** with the following key components:

* **Sprint-Based Development:**
  + Work is divided into **four sprints**, each focusing on a specific phase of development.
* **Weekly Standups:**
  + If working in a team, quick updates on progress, blockers, and next tasks.
* **Kanban Board (JIRA ):**
  + Tasks are managed with categories like **To Do, In Progress, Review, Done**.
* **Sprint Reviews & Retrospectives:**
  + At the end of each sprint, evaluate progress and adjust the backlog

**3. Change Management**

To handle new or changed requirements, we will use a **backlog and change request process**:

1. **Backlog Management:**
   * New feature requests or changes are added to the backlog.
   * Each request is categorized as **Must Have, Should Have, Could Have** (MoSCoW Method).
   * The backlog is reviewed at the start of each sprint to prioritize changes.
2. **Change Request Handling:**
   * **Minor changes** (UI improvements, bug fixes) can be implemented within ongoing sprints.
   * **Major changes** (new features or significant updates) are evaluated and either:
     + Added to the backlog for future sprints.
     + Integrated into the current sprint if feasible.
     + Deferred to a later release if it impacts deadlines.