**Module 7 Activity 1: Project Design and Implementations**

Saurabh Kale

IFT 458/554: Middleware Programming & Database Security

Dinesh Sthapit

November 12th, 2023

**Assignment: Comprehensive Project Documentation for Book Exchange Application**

1. **Domain Definitions:**

* **Book Exchange:** This realm thrives on enabling users to lend, trade, or share their books with others, fostering a vibrant space for book enthusiasts to connect and exchange their favorite reads.
* **User:** Individuals actively engaged in the book exchange platform. They maintain distinct profiles, empowering them to offer their own books, seek reads from others, and execute various tasks within the system.
* **Book:** Encompasses both physical and digital copies available for exchange. Each book carries key details such as title, author, description, genre, price, and its current availability status.
* **Exchange Type:** Classifies whether a book is available for borrowing or trading, providing clear guidelines for the exchange process. It's categorized as either 'borrow' or 'trade'.
* **Authentication:** The process that verifies user identities during sign-up, log-in, or when accessing secure resources on the platform.
* **Authorization:** The system’s capacity to delineate a user’s permissible actions based on their role and permissions, ensuring appropriate access control.
* **Token:** A security measure, commonly a JSON Web Token (JWT), utilized for user authentication and authorization. Tokens are issued upon successful login and are pivotal components within secured requests.
* **Middleware:** Functions responsible for handling incoming requests, executing specific tasks like logging, parsing request data, and managing authentication and authorization processes.
* **View:** The user interface elements that present information to users. Views dynamically adapt based on routes and user interactions, shaping the visual and interactive aspects of the platform.

1. **REST API Definitions-**
2. **Books Endpoint:**
   * **Endpoint:** **/books**
   * **HTTP Methods:**
     + **GET**: Retrieves a list of all available books.
     + **POST**: Adds a new book to the collection.
   * **Request Format (POST):**
     + **Request Body:** JSON object with book details (title, author, description, genre, owner, etc.).
   * **Response Format (GET):**
     + **Response Body:** JSON array containing book objects with their respective details.
3. **Single Book Endpoint:**
   * **Endpoint:** **/books/{bookID}**
   * **HTTP Methods:**
     + **GET**: Retrieves details of a specific book by its ID.
     + **PUT**: Updates the details of a specific book by its ID.
     + **DELETE**: Removes a specific book from the collection.
   * **Request Format (PUT):**
     + **Request Body:** JSON object with updated book details to be modified.
   * **Response Format (GET/PUT):**
     + **Response Body:** JSON object containing the details of the requested book.
   * **Response Format (DELETE):**
     + **Response Body:** Confirmation message on successful deletion.
4. **User Endpoint:**
   * **Endpoint:** **/users/{userID}**
   * **HTTP Methods:**
     + **GET**: Retrieves details of a specific user by their ID.
   * **Response Format (GET):**
     + **Response Body:** JSON object containing the user's profile information.
5. **User Books Endpoint:**
   * **Endpoint:** **/users/{userID}/books**
   * **HTTP Methods:**
     + **GET**: Retrieves a list of books associated with a specific user.
   * **Response Format (GET):**
     + **Response Body:** JSON array containing book objects owned by the user.
6. **UI Components**
7. **Navigation Bar:**
   * **Role:** Provides navigation and access to different sections of the application.
   * **Operation:** Displays links for home, book listings, user profile, login, registration, and logout.
8. **Book Listings View:**
   * **Role:** Showcases available books for exchange or borrowing.
   * **Operation:** Displays book titles, authors, genres, and descriptions. Enables users to view book details, request exchanges, or add new books.
9. **User Profile View:**
   * **Role:** Presents user-specific information and actions.
   * **Operation:** Displays user details, such as name, email, and possibly a profile picture. Allows users to manage their uploaded books, view their borrowing history, and modify personal information.
10. **Login and Registration Forms:**
    * **Role:** Facilitate user authentication and onboarding.
    * **Operation:** Enables users to log in using email and password or register by providing necessary details like name, email, and password. Handles user authentication and account creation.
11. **Book Details View:**
    * **Role:** Provides in-depth information about a specific book.
    * **Operation:** Displays detailed descriptions, additional information, and possibly user reviews for a particular book. Allows users to request exchanges or add the book to their collection.
12. **Error Pages/Alerts:**
    * **Role:** Notifies users about errors or invalid actions.
    * **Operation:** Displays informative messages when errors occur, such as invalid login credentials, missing fields, or unsuccessful transactions.
13. **Dashboard/Home View:**
    * **Role:** Serves as the central landing page.
    * **Operation:** Showcases a personalized collection of recommended books, latest additions, or user-specific content, providing an overview of the platform's offerings.
14. **Concept of Wireframe:**

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated

Successful Login page is the same page for Successful Sign up

The Home page consists of time, IP and welcome page. Below which there are 3 buttons for Book Exchange, Register and Login.

The Registration page consists of Time and IP address and input boxes for the user to type in the name, email, password and confirmation of password.

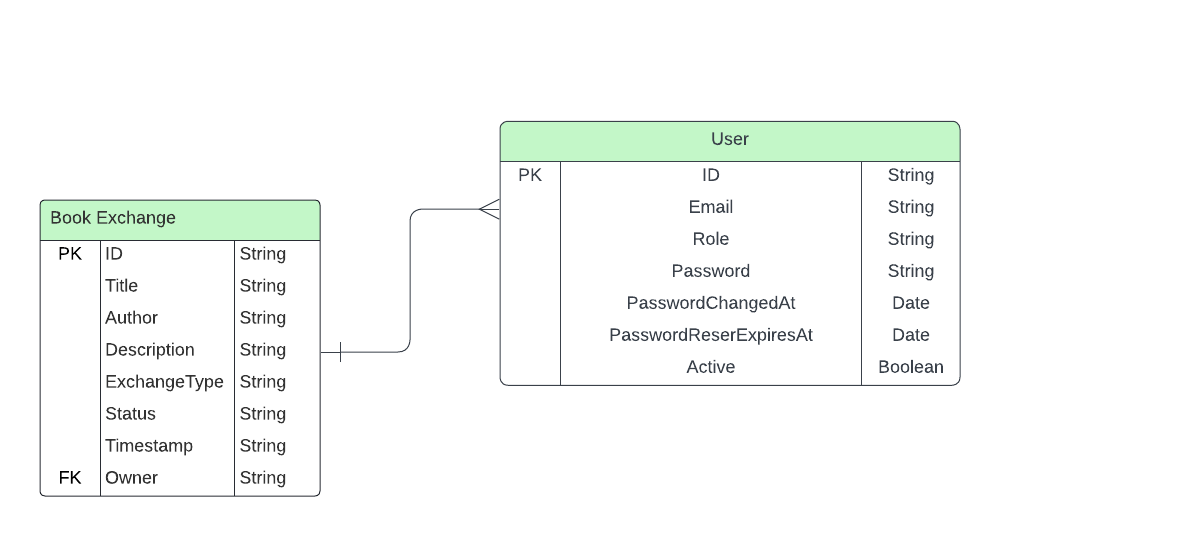
The successpage gives the token and saves it in cookies for 10 days.

The login page gives the Input boxes for the user to write email and password for the user and button to click login

The successpage comes after login to save the token for 10 days

Book Exchange Page consists of time, IP and add new book for exchange which also has options for register and Login button first.  
The Book exchange have Input boxes for Title, Author, Genre, Description, Book cover is a button to upload file and below is the button to upload book add book.

1. **Data Modeling:**

****

**Users:**

1. Account ID: Uniquely identifies each user profile in the system.
2. Email: The user's contact email.
3. Access Level: Specifies the user's role, determining system permissions.
4. Passcode: Encrypted password for secure authentication.
5. Last Passcode Change: Denotes the date of the most recent password update.
6. Passcode Reset Deadline: Specifies the expiration date for password reset.
7. Activation Status: A Boolean value indicating the user's active or inactive status.

**Book Exchange:**

1. Entry ID: Identifies each book exchange record uniquely.
2. Title: The title of the book available for exchange.
3. Author: The writer of the book.
4. About: A brief summary of the book and exchange particulars.
5. Exchange Mode: Indicates whether it's for borrowing or trading; simplified as a Boolean for processing.
6. State: Shows the present condition of the book exchange request.
7. Date and Time Stamp: Records when the entry is added to the exchange.
8. Originator: Connects to the user database, indicating the requester of the exchange.
9. **Model-View-Controller**

**Model (M):**

* **User Model:** Represents the structure of user data, including attributes like name, email, password, and role. Handles user authentication, authorization, and interactions with the database.
* **Book Model:** Defines the structure of book data, encompassing attributes such as title, author, description, owner details, and status. Manages book-related operations and interactions with the database.
* **ExchangeRequest Model:** Structures the exchange request data, including sender, receiver, book details, and request status. Manages the interaction and storage of exchange request-related data.

**View (V):**

* **EJS Templates:** Used to create the user interface for various screens like the home page, book listings, user profile, login, and registration forms. Renders the visual presentation of data retrieved from the server.
* **UI Components:** HTML, CSS, and JavaScript elements utilized to design and structure the user interface, ensuring a seamless and intuitive experience for users.

**Controller (C):**

* **AuthController:** Manages user authentication and authorization processes, handling user sign-up, login, logout, and user-related operations.
* **BooksController:** Controls book-related functionalities, such as retrieving book listings, adding, updating, and deleting books, as well as managing book exchanges.
* **ViewsController:** Handles the logic for rendering different views, providing functionalities for user actions like registering, logging in, logging out, and accessing various application screens.

The controller acts as an intermediary, receiving user inputs from the view and interacting with the model to fetch data and apply business logic. It then sends the processed data back to the view for display. The model represents the data and database-related operations, ensuring data integrity and consistency. The view presents the data to the user, allowing interaction and providing a visually appealing interface. This segregation of concerns helps maintain a clean and organized codebase, making the application more maintainable and scalable.

1. **Data Storage (NoSQL)**

**Motivation for Document-Based NoSQL:**

1. **Schema Flexibility**: Documents allow for storing different types of data without a rigid schema. This flexibility is ideal for projects where the data structure evolves over time.
2. **Scalability**: Document-based databases can horizontally scale with relative ease, making them great for distributed architectures and handling large volumes of data.

**Data Storage:**

* **Documents**: Data is stored in documents, which are collections of key-value pairs, usually encoded in JSON or BSON.
* **Collections**: Documents are grouped into collections, which act as containers for related documents.
* **Indexes**: These databases use indexes to quickly locate documents. Indexes are usually built on specific fields for faster retrieval.

**Data Retrieval:**

* **Querying**: Documents are retrieved through queries, which can target specific fields within the documents. These databases offer a variety of querying methods.
* **Aggregation**: Some document-based databases provide aggregation pipelines to perform complex data manipulation and retrieval.

**Data Updates:**

* **Atomic Operations**: Updates in document-based databases are often atomic, meaning that they are either fully completed or not done at all. This ensures data integrity.
* **Partial Updates**: Documents can be updated partially, adding or modifying specific fields without needing to update the entire document.

**Considerations:**

* **Data Structure**: While the flexibility of schema can be an advantage, it might require careful management to avoid inconsistencies in the data structure.
* **Query Performance**: In some cases, complex queries or large data sets might impact performance, and thus, indexing and data organization become crucial for optimizing performance.

1. **Security Flow-**

**Authorization Sequence Diagram-**

A diagram of a user access

Description automatically generated

1. **User Requests Access:** The process begins with the user requesting access to a resource through the client application.
2. **User Authentication:** The client redirects the user to the Authorization Server, prompting the user to log in and provide credentials.
3. **Authorization Request:** The client sends an authorization request to the Authorization Server, initiating the authentication process.
4. **Credentials Verification:** The user enters login credentials, and the Authorization Server verifies them. If the credentials are valid, an authorization code is sent back to the client.
5. **Access Token Retrieval:** The client requests an access token using the authorization code obtained from the Authorization Server.
6. **Resource Access:** The Authorization Server provides an access token, which the client uses to request access to a resource from the Resource Server.
7. **Resource Response:** If the token is valid, the Resource Server grants access to the requested resource.

**Authentication Sequence Diagram-**

A diagram of a user

Description automatically generated

1. **User Request**: The user initiates a request for a resource.
2. **Client Request Authorization**: The client application requests authorization for the user from the Authorization Server.
3. **Authorization Server Prompts User**: The Authorization Server prompts the user to provide login credentials (username, password, etc.).
4. **User Provides Credentials**: The user enters their credentials, which are sent back to the Authorization Server.
5. **Credential Validation**: The Authorization Server validates the provided credentials.
6. **Authorization Code**: Upon successful validation, the Authorization Server issues an authorization code to the client.
7. **Access Token Request**: The client requests an access token using the authorization code.
8. **Access Token Issuance**: The Authorization Server, after validating the code, provides an access token to the client.
9. **Resource Access**: The client uses the access token to access the protected resource on the Resource Server.
10. **Resource Server Response**: The Resource Server provides the requested resource back to the client.
11. **Incomplete Requirements-**

**Pending/Uncertain Requirements:**

* **User Roles and Authorization:** The exact definitions and setups for user roles and their corresponding access levels remain ambiguous, lacking explicit details.
* **Error Handling and Logging:** The specifics of error handling and logging mechanisms are vaguely outlined, potentially resulting in inconsistent error messages or incomplete logs.
* **Front-End Interface Specifications:** Specific design and functionality elements of certain UI components might lack clarity, impacting the integration of related views.
* **Data Validation Rules:** Detailed validation requirements for certain data inputs or constraints might be incomplete, risking potential inconsistencies in the data.
* **Scalability and Future Requirements:** Clarity on how the application should handle future scaling and large user volumes is needed, as scalability requirements are not explicitly defined.

**Impacts:**

* **Resource Allocation:** Ambiguous requirements might lead to the misallocation of resources, potentially causing rework as developers interpret features differently.
* **Increased Iteration:** Unclear or incomplete specifications may lead to multiple iterations, impacting the project's duration and overall efficiency.
* **Functional Discrepancies:** Inadequate specifications might cause discrepancies between developed features and actual user requirements, affecting functionality and usability.
* **Testing Complexity:** Uncertain requirements can complicate the testing process, creating challenges in defining test cases and ensuring comprehensive test coverage.
* **Post-Development Adaptations:** Incomplete requirements might necessitate post-development modifications, requiring additional time and effort to align the application with actual user needs.

1. **Risks-**  
   Potential challenges related to the application's creation and deployment include:
2. **Complex Integration:** Integrating various modules could pose challenges due to their individual dependencies and interactions.
3. **Scalability Concerns:** Without clear requirements for scalability, the application might face challenges in handling increased loads or future expansion.
4. **Security Vulnerabilities:** Inadequate user authorization definitions can lead to security gaps or unauthorized access.
5. **Inconsistent Error Handling:** Vaguely defined error handling mechanisms might lead to inconsistent error messages or, worse, missed critical logs.

Mitigation strategies could involve:

1. **Detailed Planning:** Create a comprehensive plan outlining integration strategies, defining scalability points, and identifying potential security loopholes.
2. **Modular Development:** Use a modular approach, which will facilitate easier integration and testing of individual units before combining them into a whole.
3. **Scalability Preparation:** Create an architecture capable of scaling based on future demands, regardless of the current size, anticipating increased users and data load.
4. **Authorization Audits:** Conduct regular security audits and access control checks to ensure that users have appropriate permissions and data access rights.
5. **Error Handling Protocols:** Define clear error handling protocols to ensure comprehensive logging, with contingency plans for crucial system errors.
6. **Comprehensive Test Specifications:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Test Input** | **Expected Output** | **Testing Method** |
|  | Register with valid credentials. | Successful registration | Functional Testing |
|  | Register with existing email. | Error: User already exists | Functional Testing |
|  | Invalid login credentials | Error: Incorrect email or password | Functional Testing |
|  | Get all books requests. | Array of books | API Testing |
|  | Retrieve a specific book by ID. | Book details | API Testing |
|  | Add a new book. | New book added | API Testing |
|  | Update book details | Book details updated | API Testing |
|  | Delete a book by ID. | Success message | API Testing |
|  | Concurrent user registrations | Handle multiple registrations concurrently | Performance Testing |
|  | Stress testing with high user load | System remains responsive | Performance Testing |
| 11. | Data integrity check after transactions | No loss or corruption of data | Security & Integrity Testing |