|  |  |
| --- | --- |
| User stories for online book management | Abstract  This document contain different stories and its details for managing an online book store  Rini Kuriyan , gmail : rinikuriyanc@gmail.com ph : 0468105564 |

**Contents**

**Category Page**

**Context ------------------------------------------------------------------------------------------------------------------2**

**User Story 1 ------------------------------------------------------------------------------------------------------------------3**

**User Story 2 ------------------------------------------------------------------------------------------------------------------5**

**User Story 3 ------------------------------------------------------------------------------------------------------------------8**

**User Story 4 ------------------------------------------------------------------------------------------------------------------11**

**User Story 5 ------------------------------------------------------------------------------------------------------------------14**

**Table details ------------------------------------------------------------------------------------------------------------------16**

**Online Bookstore Management**

**Context**

This documentation provides a detailed guide for developing an online bookstore system, covering various user stories and their implementation details.

The system includes frontend and backend components aimed at delivering a good user experience for browsing, purchasing, and managing books.

Each user story represents a specific functionality or feature expected by users, such as browsing books, adding items to the cart, viewing cart contents, modifying cart items, and removing items from the cart. The documentation outlines the steps and technologies required to implement these functionalities, covering both frontend and backend aspects.

By following this documentation, developers can gain insights into the system's architecture, functionality, and implementation details, facilitating effective collaboration and ensuring the successful development of the online bookstore

|  |  |
| --- | --- |
| **User Story** | 1. As a user, I want to view a list of books available in the Online Bookstore   Frontend: Implement a page to display the list of books with details such as title, author, and price.  Backend: Develop an API endpoint to retrieve the list of books from the database and deliver it to the frontend.   1. As a user, I want to add books to my cart   Frontend: A button for users to add books to cart from the book details page.  Backend: Implement API endpoints to add books to the user's cart and update the information in the backend database   1. As a user, I want to view my cart, so that I can see the selected for purchase.   Frontend: Design a page to display the user's cart with the list of items, quantities, and total price.  Backend: Create an API endpoint to retrieve the user's cart information from the backend database.   1. As a user, I want to modify the quantity of items in my cart   Frontend: Allow users to update the quantity of items in their cart .  Backend: Implement API endpoints to update the quantity of items.   1. As a user, I want to remove items from my cart that I no longer wish to purchase.   Frontend: Provide a button for users to remove items from their cart  Backend: Develop API endpoints to remove items from the user's cart and update the cart information |

**Assumptions**

**The user is a validated member, and logined successfully with userid and password.**

**All the databases for this operations are already created.**

**USER STORIES**

1. **As a user, I want to view a list of books available in the Online Bookstore**

**Acceptance Criteria**

**Frontend:**

* **The page display with attractive layout - for display book.**
* **User can navigate to the "Books" section from Home page.**
* **User is given with a list of available books with all relevant details -title, author, and price.**
* **Users should be able to scroll through the list of books .**
* **Each book entry should be clickable, allow users to go through to the book details page for more information.**
* **The page should be responsive and aligh with different screen size.**
* **User can return to the main page or click to other tabs in site.**

**Backend:**

* The API endpoint implemented using the RESTful service.
* The endpoint should be accessible via a URL route, e.g., /api/books, using the HTTP GET method.
* On receiving a request, the endpoint should retrieve the list of books from the database.
* The list of books should be formatted into a JSON response with details such as title, author, and price for each book.
* The JSON response should be delivered to the frontend with an appropriate HTTP status code (both positive and negative)

**Implementation Details**

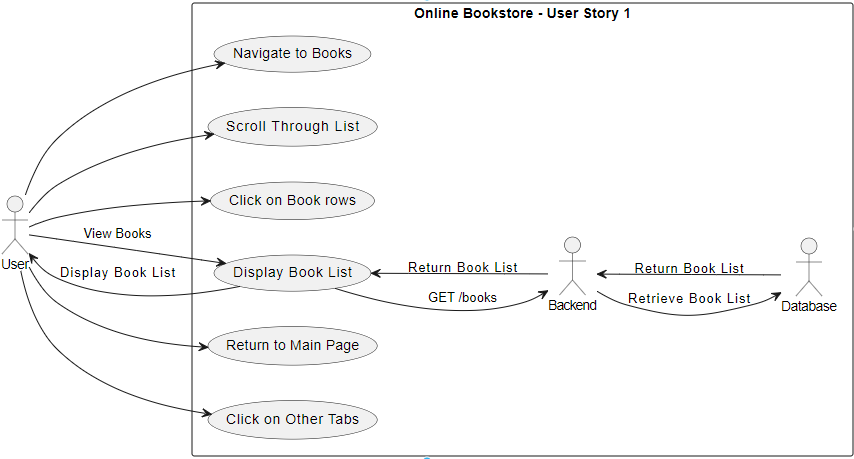
**Front End**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation Detail** | **Technology** |
| Page for Display available books with an attractive layout | Use modern design principles and CSS frameworks | HTML, CSS |
| Navigation to "Books" section from the Home page | Implement a navigation link | HTML, JavaScript |
| List of available books with relevant details | Fetch book data from backend API and render dynamically | JavaScript, AJAX |
| Scrollable list of books | user CSS overflow property to enable scrolling | CSS |
| Clickable book entries for details | Implement event listeners | JavaScript |
| Adaptable for different screen sizes | Utilize CSS media queries to adjust layout | CSS |
| Navigation to other tabs in the site | Implement navigation links or buttons | HTML, JavaScript |

**Backend:**

|  |  |
| --- | --- |
| End Point | /users/{userId}/books |
| Example | http://**sampleurl:9000**/ **api**/users/{userId}/books |
| Swagger |  |
| Method | GET |
| Details | **This endpoint retrieves the list of books from the backend database.**  **Listen for incoming HTTP GET requests**  **Retrieve the list of books from the database.**  **Format the list of books into a JSON response with details such as title, author, and price for each book.**  **Set the appropriate HTTP status code for the response (e.g., 200 OK).**  **Send the JSON response to the frontend to user** |
| Request | Parameters  userId (path**) w**hich specifies the user for whom the book list is being retrieved  **Sample :** <http://sampleurl:9000/api/users/123/books> |
| Response | Display book details with 200 response back  HTTP/1.1 200 OK  Content-Type: application/json  [  {  "id": 1,  "title": "ABC",  "author": "Mr.X",  "price": 9.99  },  {  "id": 2,  "title": "WWWW",  "author": "Mr.Z",  "price": 8.99  },  {  "id": 3,  "title": "123-SAMPLE",  "author": "Philip",  "price": 7.99  }  ] |
| Error Handling | If no books are available, return an empty array with status code 404 Not Found (No books found).  Sample :  {  "message": "No books found”  }  If an unexpected error occurs during data retrieval, return an error response with an appropriate HTTP status code and an error message indicating the issue  For 500 Internal Server Error (Internal server error):  {  "message": "An unexpected error occurred while retrieving book data."  } |

**USE CASE DIAGRAM**

****

1. **As a user, I want to add books to my cart**

**Acceptance Criteria**

**Frontend:**

* **Button - "Add to Cart” displayed on the book details page.**
* **When click "Add to Cart" button, the book should be added to the user's cart.**
* **Alert message when user added book to cart as “Item successfully added to the cart ".**
* **If the user add the same book multiple times, the quantity of the book in the cart should be incremented.**
* **The cart should be updated dynamically to reflect the added book without requiring a page refresh.**

**Backend:**

* **Implement API endpoint(s) to handle “add to cart of books” to the user's cart.**
* **The API endpoint(s) accept parameters such as book ID and quantity.**
* **While receiving the Request add a book to the cart, the backend should update the user's cart information in the database.**
* **If the book is successfully added to the cart, the API should return a success response with an HTTP status code (e.g., 200 OK).**
* **If any errors occur during the addition process (e.g., invalid book ID, database error), the API should return an error response with an HTTP status code (e.g., 400 Bad Request, 500 Internal Server Error).**

**IMPLEMENTAION DETAILS**

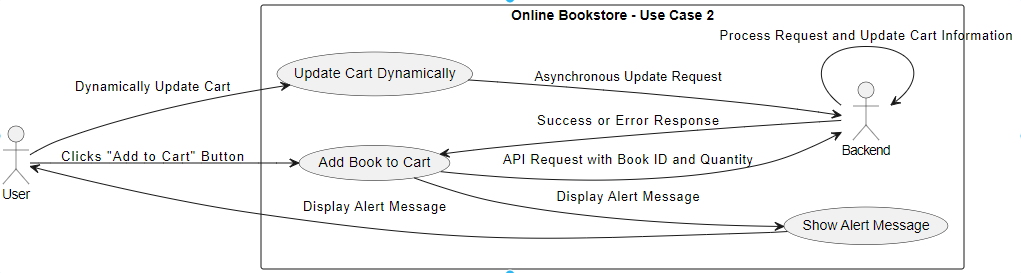
**Front End**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation Details** | **Technology** |
| **Button - "Add to Cart" on book details page** | **Implement a button labeled "Add to Cart" on the book details page.** | **HTML, CSS, JavaScript** |
| **Adding book to cart on button click** | **A event listener to the "Add to Cart" button that trigger an API call to add the selected book to the user's cart.** | **JavaScript, AJAX** |
| **Alert message on addition to cart** | **Display an alert message "Item successfully added to the cart" in success , else "Something went wrong"** | **JavaScript** |
| **Increment quantity for duplicate book additions** | **when same book in the cart is added again, increment the quantity of that book in the cart instead of adding it as a new item.** | **JavaScript** |
| **Dynamic cart update** | **Use asynchronous JavaScript (e.g., AJAX) to update the cart dynamically after successfully adding a book to the cart without requiring a full page refresh.** | **JavaScript, AJAX** |

**Backend**

|  |  |  |  |
| --- | --- | --- | --- |
| **End Point** | /users/{userId}/cart/add |  |  |
| **Example** | http://**sampleurl:9000**/ **api**//users/{userId}/cart/add |  |  |
| **Swagger Link** |  |  |  |
| **Method** | **POST** |  |  |
| **Headers** | **Content-Type: application/json**  **Authorization: Bearer [JWT Token]** |  |  |
| **Request** | **Parameters**  **userId (path): The unique identifier of the user whose cart is being updated.**  **bookId (body): The unique identifier of the book being added to the cart.**  **quantity (body): The quantity of the book to be added to the cart.**  **{**  **"bookId": "string",**  **"quantity": 1**  **} Example**  **POST /api/users/123/** cart/add:  **Request Body:**  **{**  **"bookId": "978-0345339683",**  **"quantity": 2**  **}** |  |  |
| Details | **Update the schema to include a table for storing user carts.**  **Ensure that the user ID is properly indexed for efficient retrieval of cart information for a specific user.**  **Whenever a user adds a book to their cart, insert a new record into the user's cart table with the corresponding book ID and set the quantity to 1.**  **If the same book is added again, update the quantity field accordingly.**  **Validate input parameters to prevent invalid data from being stored in the database.**  **Handle database errors gracefully and provide appropriate error messages in case of failures during database operations** |  |  |
| **Response** | **Status Code: 200 OK**  **{**  **"message": "Item successfully added to the cart",**  **"cart": {**  **"items": [**  **{**  **"bookId": "11",**  **"title": "ABC",**  **"quantity": 1,**  **"price": 9.99**  **}**  **],**  **"totalItems": 1,**  **"totalPrice": 9.99**  **}**  **}** |  |  |
| **Error Handling** | **If the request body is missing the bookId field or the quantity field, the API should return a 400 Bad Request and message.**  **If the specified book ID does not exist in the database, the API should return a 404 Not Found error and message**  **If any database error occurs while updating the cart information, the API should return a 500 Internal Server Error and message**  **Status Code:**  **400 Bad Request**  **{**  **"error": "Invalid request format",**  **"message": "The request is missing the 'bookId' field"**  **}**  **Status Code: 404 Not Found**  **{**  **"error": "Book not found",**  **"message": "The specified does not exist"**  **}**  **Status Code: 500 Internal Server Error**  **Copy code**  **{**  **"error": "Database error",**  **"message": "Unexpected System Error"**  **}** |  |  |

**Use Case Diagram**

****

1. **As a user, I want to display my cart, so that I can see the selected for purchase**

**Acceptance Criteria**

**Frontend:**

* **Design a user interface page for to displaying the user's cart.**
* **The cart page should contain list of items selected for purchase, with as book title, quantity, and price.**
* **The total price of all items in the cart should be displayed.**

**Backend:**

* **Implement an API endpoint, to retrieve the user's cart information.**
* **The endpoint should use HTTP GET method to fetch the cart data from the backend database.**
* **The response should include details of each item in the cart, including the book title, quantity, and price.**
* **Ensure that the API endpoint returns the cart information in a structured format, such as JSON.**

**Implementation Details**

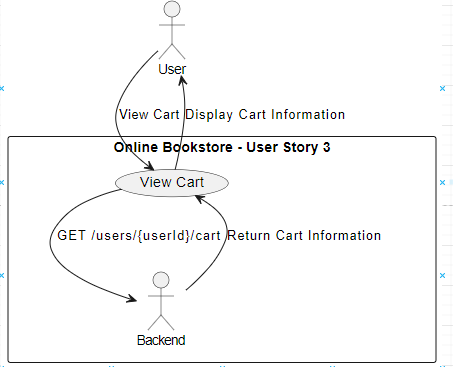
**Front end**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation Details** | **Technology** |
| **User Interface** | **Design a dedicated page for displaying the user's cart.** | **HTML5, CSS3,JAVA Script** |
| **Show List of Items** | **Display a list of items selected for purchase in the cart.Include details such as book title, quantity, and price for each item in the cart.** | **HTML ,CSS styling** |
| **Display Total Price** | **Display the total price of all items in the cart.** | **JavaScript** |
| **Responsive Design** | **Ensure the cart page layout is responsive and adapts to different screen sizes.** | **CSS** |

**Backend**

|  |  |  |  |
| --- | --- | --- | --- |
| End Point | /users/{userId}/cart |  |  |
| Example | http://**sampleurl:9000**/ **api**/users/{userId}/cart |  |  |
| Swagger Link |  |  |  |
| Method | GET |  |  |
| Request | Parameters  **userId: The ID of the user whose cart information is being retrieved. This parameter should be included as a path parameter**  sample :  **GET** /api/users/123456/cart |  |  |
| **Details** | **This endpoint retrieves the user's cart information from the backend database.**  **Parse the userId from the request path to identify the user.**  **Query the backend database to retrieve the user's cart information.**  **Format the cart data into a structured format, such as JSON, including details like book title, quantity, and price.**  **Return the formatted cart data in the HTTP response.** |  |  |
| Response | **The response should include details of each item in the cart**  **bookId: The ID of the book in the cart.**  **title: The title of the book.**  **quantity: The quantity of the book in the cart.**  **price: The price of the book.**  Status Code: 200 OK  {  "cartItems": [  {  "bookTitle": "ABC",  "quantity": 2,  "price": 19.99  },  {  "bookTitle": "WWW",  "quantity": 1,  "price": 24.99  }  ],  "totalPrice": 59.97  } |  |  |
| Error Handling | If user not found then 400 Bad Request and message.  {  "error": "User not found"  }  If any other error occurs, the API should return a 500 Internal Server Error and message {  "error": "Internal server error"  } |  |  |

**Use case Diagram**

****

1. **As a user, I want to modify the quantity of items in my car**

**Acceptance Criteria:**

**Frontend:**

* **Provide a dropdown to allow users to update the quantity of items in their cart.**
* **Ensure that users can easily change the quantity of items.**
* **Display the updated total price dynamically to reflect the changes in quantity.**

**Backend:**

* **Create a API endpoints to handle updating the quantity of items in the user's cart.**
* **Use HTTP methods such as PUT or PATCH to update the quantity of items.**
* **Accept parameters such as item ID and new quantity to identify the item**
* **Validate the input parameters to ensure they are in the correct format /range**
* **Update the backend database with the new quantity for the specified item.**
* **Return an appropriate positive / negative response with an HTTP status code and message**
* **Update the user cart with new data and overall price dynamically**

**Implementation Details**

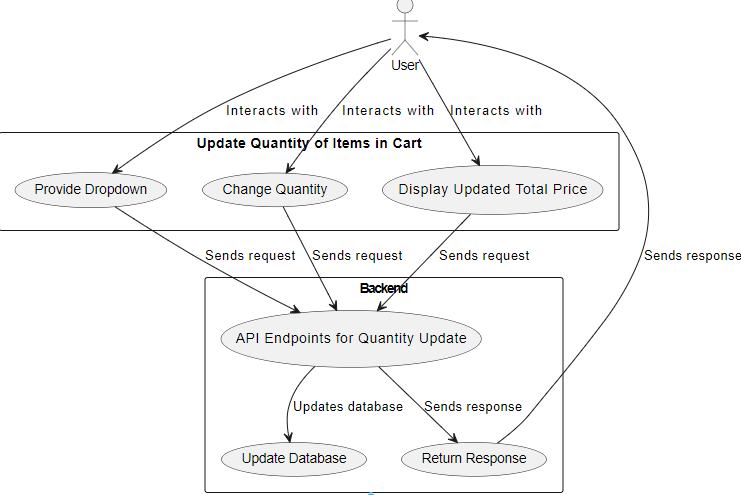
**Frontend**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implementation Detail** | **Technology** |
| **Dropdown to update quantity** | **Implement a dropdown menu allowing users to select the desired quantity for each item in the cart.** | **HTML, CSS, JavaScript** |
| **Ease of quantity modification** | **Ensure intuitive user interaction for modifying the quantity, such as clear labeling and easy access to the dropdown.** | **UI/UX Design Principles** |
| **Dynamic update of total price** | **Implement JavaScript functionality to dynamically update the total price displayed based on the selected quantity changes.** | **JavaScript** |

**Backend**

|  |  |
| --- | --- |
| **End Point** | **/users/{userId}/cart/update , where {userId} is the ID of the user whose cart is being updated.** |
| **Example** | [http://sampleurl:9000/api/users/{userId}/cart/update](http://sampleurl:9000/api/users/%7buserId%7d/cart/update) |
| **Swagger Link** |  |
| **Method** | **PUT** |
| **Details** | **Listen for incoming HTTP PUT**  **Accept parameters such as item ID and new quantity to identify the item specific to the user's cart.**  **Validate the input parameters to ensure they are in the correct format and range, considering the user's cart context**  **Database update : Update the backend database with the new quantity for the specified item in the user's cart. Dynamic Cart Update: After successfully updating the item quantity in the database, update the user's cart with the new data and overall price dynamically. This may involve recalculating the total price of all items in the cart based on the updated quantities.**  **Return an appropriate positive response with an HTTP status code (e.g., 200 OK) and a success message if the update is successful.**  **If any errors occur during the update process (e.g., invalid item ID, database error), return an appropriate error response with an HTTP status code (e.g., 400 Bad Request, 500 Internal Server Error) and an error message indicating the issue.** |
| **Request** | **Parameters  itemId and quantity in the request body- to identify the item and new quantity.**  **Both should be validated. sample : PUT /users/123/cart/update**  **{**  **"itemId": "456",**  **"quantity": 3**  **}** |
| **Response** | **return 200 OK for a successful update**  **Sample**  **HTTP/1.1 200 OK**  **Content-Type: application/json**  **{**  **"message": "Item quantity updated successfully",**  **"cart": {**  **"userId": "123456",**  **"items": [**  **{**  **"itemId": "123456789",**  **"quantity": 3**  **},**  **{**  **"itemId": "987654321",**  **"quantity": 2**  **}**  **],**  **"totalPrice": 59.99**  **}**  **}** |
| **Error Handling** | **400 Bad Request for for validation errors**  **Sample :**  **Content-Type: application/json**  **{**  **"message": "Invalid input parameters"**  **}**  **404 Not Found for other failures**  **Content-Type: application/json**  **{**  **"message": "Cart item not found"**  **}** |

**USE CASE DIAGRAM**

****

1. **As a user, I want to remove items from my cart that I no longer wish to purchase.**

**Frontend:**

* **Create button on the user's cart page to remove items.**
* **Trigger a request to the backend API to remove the selected item from the cart.**
* **Update the cart dynamically .**

**Backend:**

* **Implement API endpoints to handle the removal of items from the user's cart.**
* **Use appropriate HTTP methods - DELETE to perform the delete operation.**
* **Accept parameters such as item ID to identify the item to be removed.**
* **Update the backend database accordigy.**
* **Return an appropriate response to indicate the success or failure of the removal operation.**

**Implementation Details**

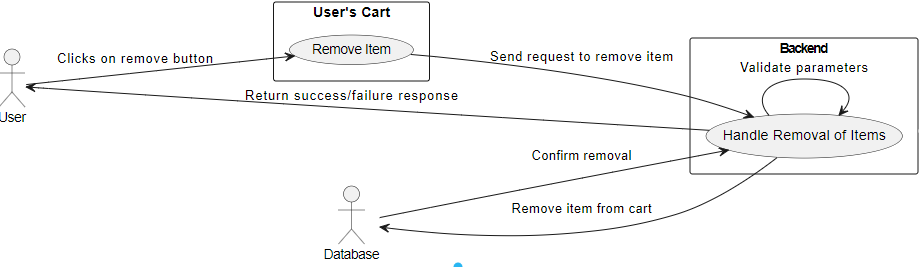
**Front end**

|  |  |  |
| --- | --- | --- |
| Feature | Implementation Detail | Technology |
| Create Button to Remove Items | Design and implement a button element on the user's cart page, name 'Remove Item" to allow users to remove items from their cart. | HTML, CSS, JavaScript |
| Trigger API Request | Trigger an API request to the backend when clicked by the user. | JavaScript, AJAX |
| Update Cart Dynamically | Upon successful removal of an item, dynamically update the cart UI | JavaScript, AJAX |
| Alert Message | Display alert message for success / failiure cases | JavaScript, |

**Backend**

|  |  |
| --- | --- |
| **End Point** | **/cart/{userId}/items/{itemId}** |
| **Example** | **http://example.com/ api/cart/{userId}/items/{itemId}** |
| **Swagger** |  |
| **Method** | **DELETE** |
| **Details** | **Retrieve the user's cart information from the database, including the item to be removed.**  **Remove Item by a DELETE query or method to remove the specified item from the user's cart.**  **Commit Changes** |
| **Request** | **Parameters  userId (path), itemId (path). Both should be validated.** <http://example.com/api/cart/123/items/456> |
| **Response** | **return 200 OK indicating the successful removal of the item.**  **Sample**  **{**  **"status": "success",**  **"message": "Item successfully removed from the cart"**  **}** |
| **Error Handling** | **- If userId or itemId is invalid: 404 Not Found,**  **{ "error": "Item not found" }**  **- If database error occurs: 500 Internal Server Error,**  **{ "error": "Database error" }** |

**USE CASE DIAGRAM**



**Table information**

**Book Table:sss**

**Attributes: book\_id (Primary Key), title, author, price, quantity\_available, genre, publication\_year, description**

**Customer Table:**

**Attributes: user\_id (Primary Key), username, email, password (encrypted), full\_name, shipping\_address, billing\_address, phone\_number,**

**Cart Table:**

**Attributes: cart\_id (Primary Key), user\_id (Foreign Key to User table), creation\_date, total\_price,**

**CartItem Table:**

**Attributes: cart\_item\_id (Primary Key), cart\_id (Foreign Key to Cart table), book\_id (Foreign Key to Book table), quantity, item\_price, subtotal,.**

**Order Table:**

**Attributes: order\_id (Primary Key), user\_id (Foreign Key to User table), order\_date, total\_price, order\_status, shipping\_address, billing\_address, payment\_method,**

**OrderParameter Table:**

**Attributes: order\_item\_id (Primary Key), order\_id (Foreign Key to Order table), book\_id (Foreign Key to Book table), quantity, item\_price, subtotal, etc.**

**Author Table:**

**Attributes: author\_id (Primary Key), author\_name, biography, etc.**

**Review Table:**

**Attributes: review\_id (Primary Key), book\_id (Foreign Key to Book table), user\_id (Foreign Key to User table), rating, review\_text, review\_date, etc.**