Web Applications A.Y. 2024-2025
Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 17 April, 2025

Group Acronym	Bookly	
Last Name	**First Name**	**Badge Number**
Jolani Mameghani	Amirreza	2106526
Saeedidana	Parisa	2142717
Truty	Adam	2142741
Mia	Md Abu Raihan	2142011
Doe	Jane	123456
Doe	John	123456

1 Objectives

The bookstore web application aims to provide an intuitive, feature-rich, and user-friendly platform for browsing, purchasing, and managing books online. The primary objective of this project is to enhance the online book-shopping experience by offering users a well-organized and efficient system where they can explore books, add them to their shopping cart or wishlist, leave reviews, apply discounts, and securely complete their purchases. This application is designed for book enthusiasts who want a convenient way to discover and buy books without visiting a physical store. Users can create an account, browse books by category, author, or publisher, and use additional features like wishlists to save books for future purchases. The wishlist functionality allows users to keep track of books they are interested in without immediately adding them to their shopping cart, enhancing flexibility in decision-making. A key feature of the system is the shopping cart, where users can add multiple books before proceeding to checkout. The ordering system ensures a structured purchasing process, allowing users to place orders and complete payments securely. Payment transactions include details such as the amount, method, and date, ensuring transparency and security. The discount feature provides users with promotional offers through discount codes, enhancing the affordability of books and improving customer satisfaction. To further engage users, the application includes a review system, allowing customers to leave feedback on books they have read. This feature helps other users make informed decisions based on ratings and reviews. Additionally, book categories, authors, and publishers are well-structured, making book discovery easier and more efficient. By integrating essential e-commerce functionalities with interactive features such as wishlists and reviews, this project aims to create a modern, engaging, and accessible online bookstore. The system is designed to improve book accessibility, offer a smooth shopping experience, and provide users with a reliable platform for discovering, saving, reviewing, and purchasing books in an efficient and enjoyable way.

2 Main Functionalities

Bookly is an interactive online bookstore designed to provide a seamless experience for both customers and administrators. The platform allows users to explore and purchase books while ensuring efficient management of payments. At the core of the system, customers can search for books by various criteria, including title, author, and use a filter of category, and publisher, to find a book easier. The book details page is contain of book information, such as ISBN, price, stock availability, and publication year. Additionally, customers can engage with the platform by bookmarking books for future reference, leaving reviews, and receiving personalized recommendations based on their browsing and purchasing history.

To enhance the purchasing experience, customers can add books to their shopping cart and proceed to checkout using a simple and secure payment method. The system ensures that stock availability is updated in real-time, preventing purchasing unavailable books. Customers can also track their orders. Alongside managing their account information.

Bookly is built on a structured system that integrates RESTful APIs to handle book data, user interactions, and order processing. The system includes error-handling mechanisms that provide meaningful feedback for issues such as failed transactions or invalid inputs. By maintaining a clear and organized approach to managing book sales and customer interactions, Bookly ensures a streamlined and user-friendly experience for both customers and administrators, making book discovery, purchase, and management effortless.

3 Data Logic Layer

3.1 Entity-Relationship Schema
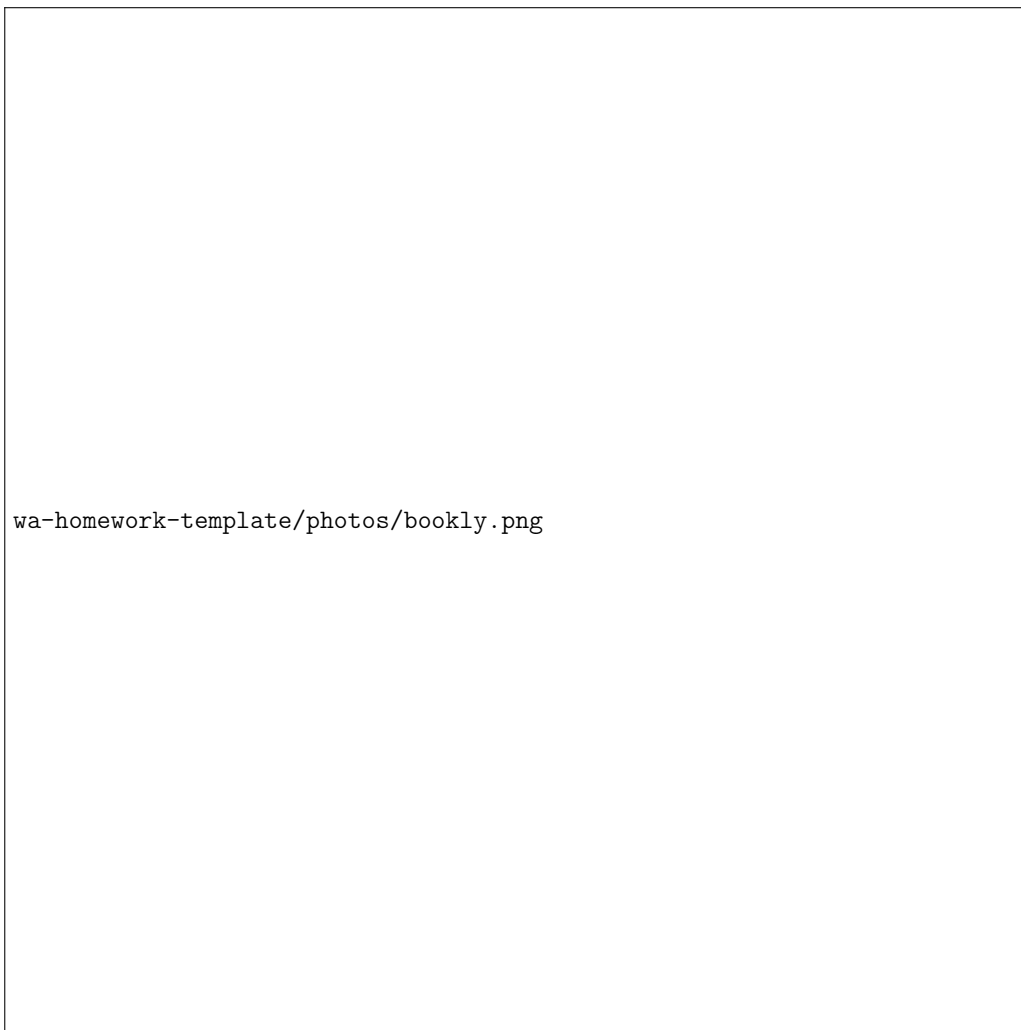

wa-homework-template/photos/bookly.png

Figure 1: ER-Diagram

The ER schema contains the following entities:

User represents individuals who interact with the Bookly platform, such as browsing books, placing orders, writing reviews, and creating wishlists. Each user is uniquely identified by `user_id` (type: SERIAL, primary key). The `username` (type: booklySchema.username_domain, i.e., `VARCHAR(50)` with format constraints) must be unique, and the `password` (type: booklySchema.password_domain, i.e., `VARCHAR(255)`) must have at least 8 characters. Users also have `first_name` and `last_name` (both of type: `VARCHAR(50)`), and a unique `email` (type: `VARCHAR(100)`). The optional `phone` field uses a custom domain (type: phone_domain, i.e., `VARCHAR(20)`) that enforces a valid format, and `address` (type: `TEXT`) stores shipping or billing information. The `role` attribute is defined by an ENUM type `user_role` (values: 'user', 'admin') and defaults to 'user'. Each user is associated with a unique shopping cart via the `shopcart` foreign key (type: INTEGER), linking to the `shopcart` entity (1–1 relationship).

Author captures metadata about individuals who have written books. Each author is identified by a unique `author_id` (type: SERIAL, primary key). Other attributes include `first_name` and `last_name` (both of type: `VARCHAR(100)`), a `biography` field (type: TEXT) containing an optional description of the author's life or achieve-

ments, and `nationality` (type: `VARCHAR(100)`) describing the author's country of origin. Authors can be linked to one or more books through the `writes` relationship, which is modeled as a many-to-many association.

Publisher represents companies or organizations that publish books. Each publisher has a unique `publisher_id` (type: SERIAL, primary key), a `publisher_name` (type: `VARCHAR(100)`), an optional `phone` number (type: `phone_domain`), and an `address` (type: TEXT). Each book can be published by exactly one publisher, and this is modeled by the `published_by` relationship (many-to-one).

Category is used to classify books by genre, theme, or other organizing criteria. It includes a `category_id` (type: SERIAL, primary key), a `category_name` (type: `VARCHAR(50)`), and an optional `description` (type: TEXT). A book can belong to one or more categories, managed by the `category_belongs` relationship (many-to-many).

Book is the core item of the platform, representing all available reading material. Each book is uniquely identified by `book_id` (type: SERIAL, primary key), and includes a `title` (type: `VARCHAR(150)`), optional `language` (type: `VARCHAR(50)`), and a unique `isbn` (type: `VARCHAR(20)`). The `price` (type: REAL) is a required field representing the cost. Additional attributes include `edition` (type: `VARCHAR(50)`), `publication_year` (type: INTEGER) restricted between 1000 and the current year, `number_of_pages` (type: INTEGER, must be positive), and `stock_quantity` (type: INTEGER, default 0, must be non-negative). The `average_rate` (type: REAL) stores the average of user-submitted ratings, and `summary` (type: TEXT) provides an optional overview of the book. Books are connected to categories, publishers, authors, shopping carts, and reviews via appropriate relationships.

ShopCart represents a temporary list of books a user intends to purchase. Each cart has a `cart_id` (type: SERIAL, primary key), a `shippment_method` (type: ENUM `shippment_method`, values: 'in_person', 'credit_card'), and a `user_id` (type: INTEGER) as a unique foreign key referencing `users`, enforcing a 1–1 user-cart relationship. It also includes `created_date` (type: TIMESTAMP, default: current time), `quantity` (type: INTEGER, default: 0), `discount` (type: INTEGER) as a foreign key referencing the `discounts` table, and `order` (type: INTEGER) as a foreign key referencing the `orders` table.

Order represents completed purchases. Each order has a unique `order_id` (type: SERIAL, primary key), a `total_amount` (type: NUMERIC(10,2)) representing the final price, a `payment_method` (type: ENUM `payment_method`), and a `payment_date` (type: TIMESTAMP, default: current timestamp). Orders are linked to shopping carts (1–1), and may include discounts.

Discount defines promotional codes that reduce order totals. It includes a `discount_id` (type: SERIAL, primary key), a unique `code` (type: `VARCHAR(50)`), a `discount_percentage` (type: NUMERIC(5,2)) restricted to 0–100, and an optional `expired_date` (type: DATE). Each discount can be associated with a shopping cart or order to adjust pricing.

Reviews captures feedback from users about books. Each review has a unique `review_id` (type: SERIAL, primary key), and includes `user_id` and `book_id` (both of type: INTEGER) as foreign keys. It also includes `review_text` (type: TEXT), a `rating` (type: INTEGER between 1 and 5), `review_date` (type: TIMESTAMP, default: current timestamp), and optional counters for `number_of_likes` and `number_of_dislikes` (both of type: INTEGER). This entity supports the social and feedback features of the platform.

Wishlist allows users to save books they are interested in for future reference without adding them to the cart. Each wishlist entry has a unique `wishlist_id` (type: SERIAL, primary key), and includes `user_id` and `book_id` (both of type: INTEGER) as foreign keys to link the wishlist to a specific user and book. Additionally, it contains `wishlist_date` (type: TIMESTAMP, default: current timestamp), to track when a book was added to the wishlist. This entity enhances user experience by enabling users to keep track of books they may want to purchase later.

3.2   Other Information

There are also some tests available in the src/test directory which can be used to test some DAOs. But these tests were for development purposes and they may manipulate the database on running on the docker. Moreover, some of the tests are based on the insert queries which will populate the database in the first run of the docker container.

4 Presentation Logic Layer

When a user accesses the website, they will be directed to the home page without needing to sign in. From there, they can navigate to various other sections, including:

- Sign In/Sign Up Page : For user authentication.
- User Profile Page : Displays user details.
- Book Page : Provides details about a specific book.
- Basket Page : Shows the user's selected books for purchase.
- Search Results Page : Displays books based on the user's search query.
- Author Page : Displays information about a specific author.
- Cart Page : Shows the current user's shopping carts and theirs contents.
- Wishlist Page : Displays all the books the user has saved for later.

4.1 Home Page

The Home Page serves as the central hub where all users (whether signed in or not) can browse through the book catalog. It will feature several sections, such as:

- **Most Popular** : Showcasing books with the highest reviews from our users.
- **New Arrivals** : Displaying the latest books added to our stock.
- **Best Sellers** : Highlighting books with the highest sales.

Additionally, the homepage will include:

- A **search bar** allowing users to find books by title, author, or genre within the store's catalog.
- A **basket icon** that provides quick access to the user's shopping cart. Users can click on it to navigate directly to the Basket Page.
- An **account icon** that, when clicked, redirects the user to their User Profile Page if they are signed in. If not, they will be prompted to the Sign In/Sign Up Page.

4.2 Sign In/Sign Up Page

This page allows users to either sign in to an existing account or register for a new one.
The **Sign In section** includes:

- Input fields for **username** and **password**.
- A link for **password recovery** in case the user forgets their credentials.
- A button redirecting new users to the **Sign Up section**.

The **Sign Up section** provides a registration form with the following input fields:

- **First Name** and **Last Name**
- A unique **Username**
- **Password**
- **Phone Number**
- **Address**

4.3   User Profile Page

Users can also access their profile page, which displays their personal details and order history. The page will include the following information:

- **First Name and Last Name**: The full name of the user.

- **Role**: Indicates whether the user is a *Client* or an *Admin*.

- **Username**: A unique identifier for the user within the platform.

- **Email Address**: The registered email associated with the account.

- **Address**: The user's delivery address.

- **Phone Number**: The user's contact number.

The page will also provide:

- **Edit Buttons**: Users will have the option to update their personal information through dedicated buttons.

- **Change Password Link**: A dynamic text link labeled *"Change my password"* that, when clicked, redirects users to a dedicated password update page.

4.4   Book Page

Each book will have its own dedicated page containing the following details: **Title**, **Author**, **Price**, **Edition**, **Genre**, **Summary** and a **cover image** of the book.
In top of that, the page will also have:

- A **Review Section**, where users can:
    - Like or dislike existing reviews.
    - Add their own review by writing a comment and rating the book on a scale from **0 to 5 stars**.

- A **"Add to basket"** button, allowing users to purchase the book.

- A **"Add to wishlist"** button, allowing users to save the book on their wishlist.

4.5   Basket Page

This page will display the books added on the basket by the user, along with options to update quantities. It will also include the following features:

- A **recap of all selected books**, displaying their titles, prices, and quantities.

- The **total price**, dynamically updated based on the selected books, their quantities and the shipping price.

- Several input fields for the user to enter their **billing address**.

- A **payment method selection**, offering the choice between:
    - **Credit/Debit Card Payment**.
    - **Cash on Delivery**.

- An optional input field for entering a **discount code** if the user has one.

- A **"Confirm Order"** button that finalizes the payment and processes the order once all required fields are completed.

4.6 Search Results Page

When a user enters a query in the search bar, they will be redirected to this page, where relevant books will be displayed based on title, author, or genre.

For each book displayed in the search results, the following details will be shown:

- The **book title** and **cover image**, both acting as clickable links that redirect to the corresponding Book Page.

- The **average rating**, displayed as a star rating out of 5, based on user reviews.

- A **"Add to Basket"** button along with the corresponding price of the book, allowing users to quickly add the book to their shopping cart.

4.7 Author Page

Each author will have its own dedicated page displaying their fullname, biography and literary background. This page will help users explore more books from the same author by displaying their best-sellers.

4.8 Cart Page

This page will present the contents of all the user's current shopping carts. It will include all books that the user has added, even if they haven't proceeded to checkout. The cart will persist between sessions for logged-in users. This page will also display the complete user's history order containing all the books the user has already ordered and the dates corresponding.

4.9 Wishlist Page

The wishlist page displays all the books that the user has chosen to save for future reference. From this page, users can move them directly into their basket to initiate a purchase.

5 Business Logic Layer

5.1 Class Diagram

The class diagram contains a selection of the classes used to handle books and users within the Bookly web application. It includes two main servlets: the *BookServlet*, which manages the book resource, and the *UserServlet*, which handles user-related functionality such as login and registration. Each servlet extends the *AbstractDatabaseServlet*, which provides shared logic for acquiring a database connection from the connection pool.

The *BookServlet* implements the `doGet` method to retrieve either a list of all books or the details of a specific book, depending on the URI pattern. These operations are delegated to the DAO classes *GetAllBooksDAO* and *GetBookByIdDAO*, both of which extend the *AbstractDAO<Book>* class and implement the `doAccess()` method to execute SQL statements and return data as *Book* objects.

The *UserServlet* implements both `doGet` and `doPost` methods to support login and registration functionality. Before interacting with the database, it uses the *LoginServices* and *RegisterServices* classes to validate input. If validation succeeds, it calls either *LoginUserDAO* or *RegisterUserDAO*, depending on the operation. These DAO classes extend *AbstractDAO<User>* and are responsible for retrieving or inserting user data, including profile images if provided.

The *User* and *Book* classes represent the core entities in the application. Each class contains multiple fields and standard accessors. The *User* class also references additional entities such as *Order*, *Cart*, and *Wishlist*, which are not expanded in this diagram. Both the *User* and *Book* classes contain an *Image* field to represent profile or cover images using the *Image* class.

All DAO classes inherit from the generic abstract class *AbstractDAO<T>*, which manages consistent database access and exception handling. Each subclass implements the `doAccess()` method to define specific SQL logic. The *AbstractDAO* class also includes mechanisms to control access and return output parameters.

This class diagram focuses on the core application logic behind user authentication, registration, and book browsing functionality. It reflects the architectural layering of the application, with clear separation between controllers (servlets), data access logic (DAOs), business validation (services), and data representation (resources).

5.2 Sequence Diagram

This sequence diagram (see Figure **??**) illustrates the process of retrieving the details of a specific book. The user sends a GET request to the web server with the URI `/books/5`, where 5 represents the book's ID. Upon receiving the request, the web server instantiates the *BookServlet* and calls its `init(ServletConfig)` method to initialize the servlet (if it hasn't been initialized yet). After that, the server calls the `service(HttpServletRequest, HttpServletResponse)` method, which forwards the request to the `doGet(HttpServletRequest, HttpServletResponse)` method implemented in *BookServlet*. Inside the `doGet` method, the servlet obtains a database connection by invoking the `getConnection()` method inherited from *AbstractDatabaseServlet*. With the active connection and the book ID, it creates a new instance of *GetBookByIdDAO*. The servlet then calls the `access()` method on the DAO. This triggers the `doAccess()` method, which prepares and executes an SQL `SELECT` query to retrieve the book with the specified ID. The result is then used to instantiate a *Book* object. The DAO returns this object back to the servlet. Once the servlet receives the book data, it sets it as a request attribute (`book_details`) and forwards the request to the JSP page `bookDetails.jsp`, which is rendered and returned to the user as an HTML page.

Figure **??** shows the sequence diagram for adding a book to the shopping cart. The process starts when the user sends a POST request to the web server at `/cart/add/bookId`. The server passes the request to the *CartServlet*, which runs the *doPost()* method. The servlet gets the *userId* from the current session. If the user is not logged in, they are redirected to the login page. Otherwise, the servlet reads the *bookId* from the request.

Next, the servlet calls the *getOrCreateCartId(userId)* method. It creates a *GetCartByUserIdDAO* to get the user's cart from the database. In this version, we assume the cart already exists and use its *cartId*. Then, the

servlet creates an *AddBookToCartDAO* with the connection, *bookId*, and *cartId*. The DAO adds the book to the cart using an SQL `INSERT` query. Finally, the user is redirected to the `/cart` page.

wa2425-bookly/report/HW1Report/photos/ClassDiagram.png

Figure 2: Class diagram of the Bookly web application

wa2425-bookly/report/HW1Report/photos/dao-class-diagram.png

Figure 3: DAO Classes diagram of the Bookly web application
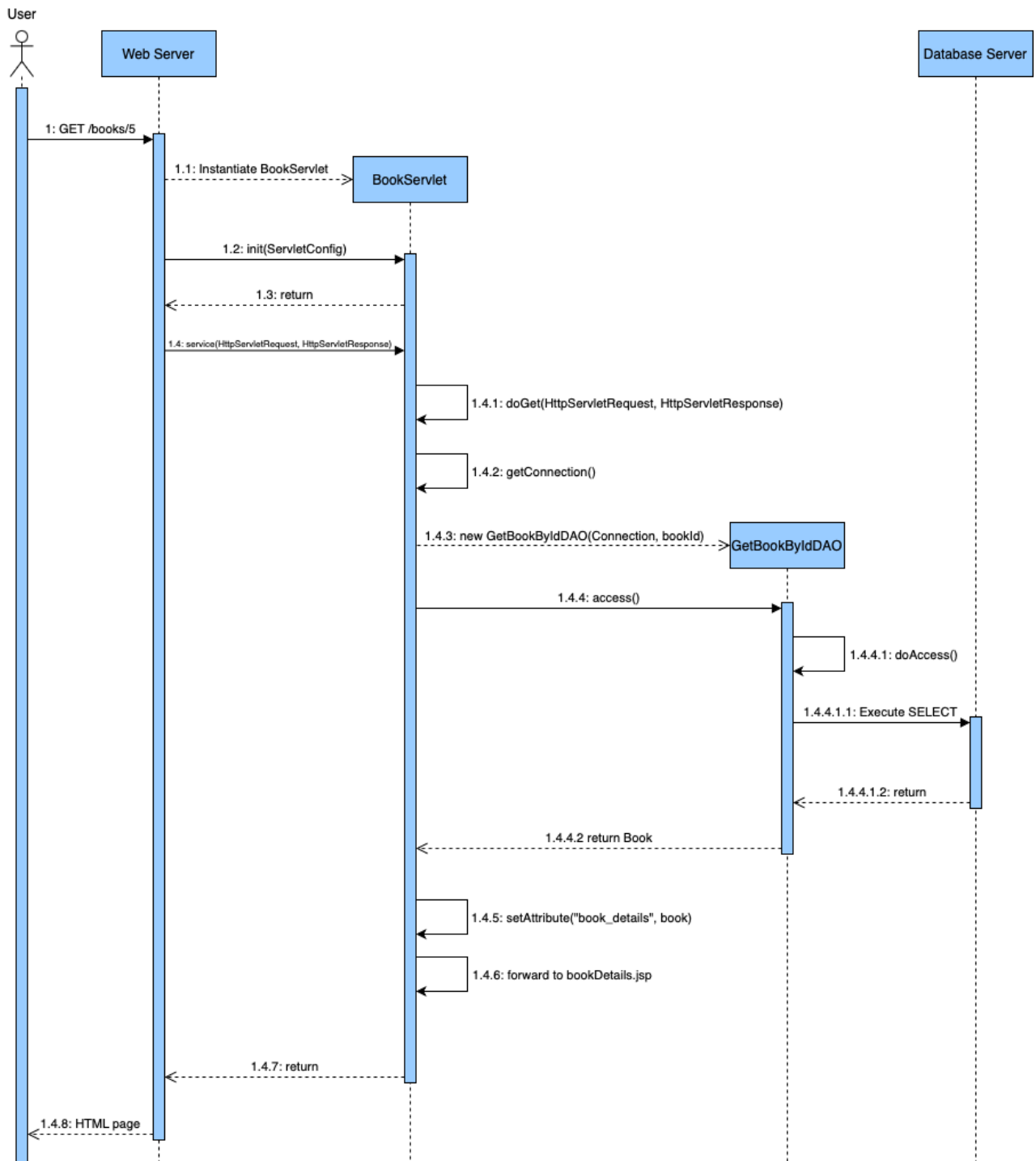
Figure 4: Sequence Diagram for retrieving a book by ID
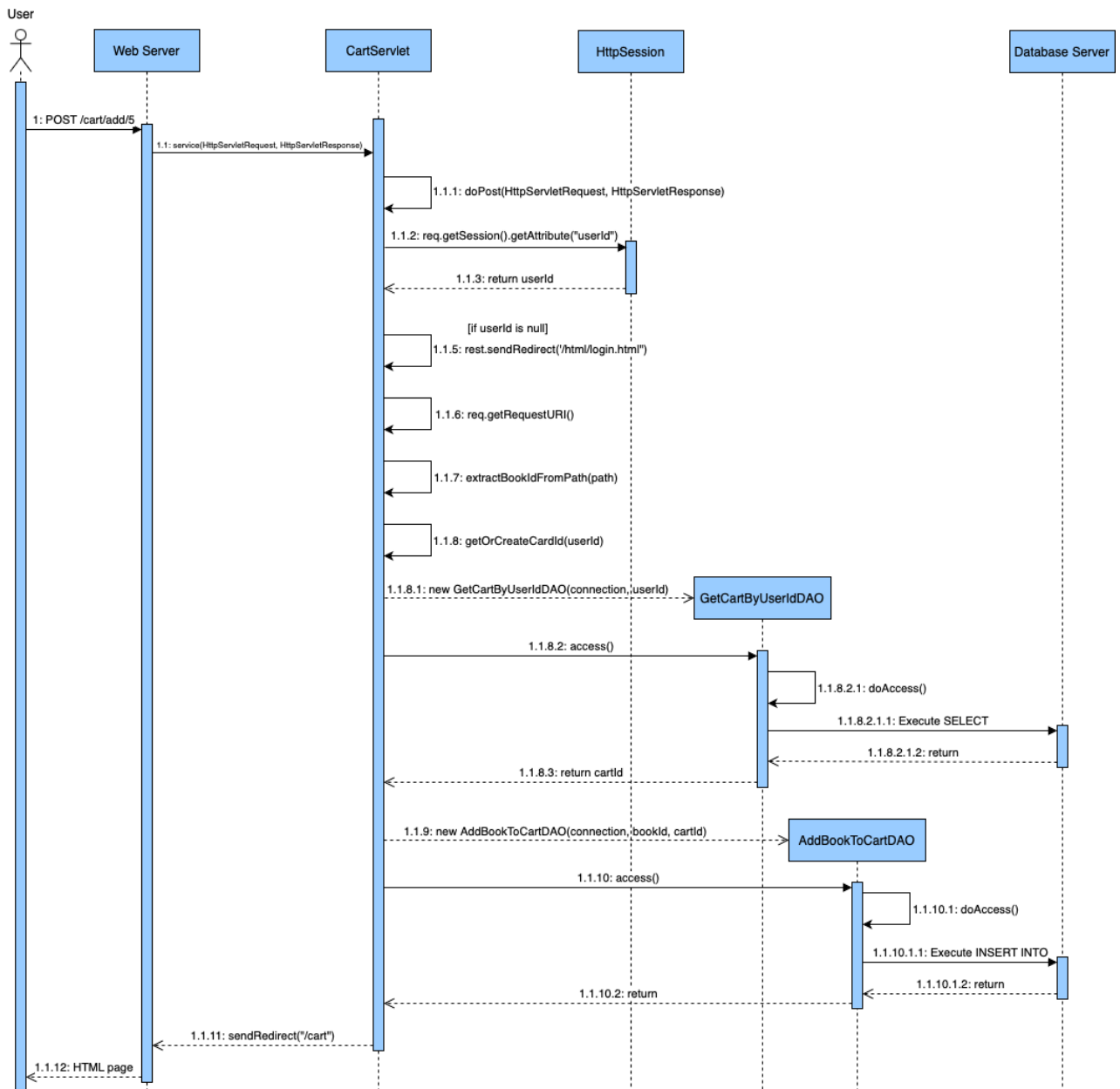
Figure 5: Sequence Diagram for adding a book to the cart

5.3 REST API Summary

This API structure is aligned with the entities in the ER diagram, covering user management, book management, orders, reviews, discounts, and the relationships between them.

URI: The specific endpoint for each action.

Method: The HTTP method used (GET, POST, PUT, DELETE).

Description: A brief explanation of what the endpoint does.

Filter: **A** — User-related   **B** — Book, author, publisher, review   **M** — Discount-related

14

URI	Method	Description
/auth/signup	POST	Register a new user with cr profile info.
/auth/login	POST	Login a user and return auth
/user?username={username}	GET	Get user information by usern
/cart?userId={user_id}	GET	Retrieve a user's cart.
/cart/add?cartId={cart_id}	bookId={book_id}	POST
A		
/cart/remove?cartId={cart_id}	bookId={book_id}	DELETE
A		
/review	POST	Create a new review for a bo
/review/{review_id}	GET	Get a specific review by ID.
/review/{review_id}	DELETE	Delete a review by its ID.
/reviews	GET	Retrieve all reviews in the sys
/review/book/{book_id}	GET	Get all reviews for a specific
/review/user/{user_id}	GET	Get all reviews written by a s
/review/top/book/{book_id}	GET	Get top-rated reviews for a b
/authors	GET	Retrieve all authors.
/authors/{id}	GET	Get details for a specific auth
/authors	POST	Create a new author.
/authors/{id}	PUT	Update an existing author.
/authors/{id}	DELETE	Delete an author by ID.
/authors/assign	POST	Assign an author to a book.
/authors/remove	DELETE	Remove an author from a bo
/wishlist?userId={user_id}	POST	Create a wishlist for a user.
/wishlist/{wishlistId}	DELETE	Delete a wishlist.
/wishlist/user/{userId}	GET	Get all wishlists for a user.
/wishlist/clear/{wishlistId}	POST	Clear all books in a wishlist.
/wishlist/books/add	POST	Add a book to a wishlist.
/wishlist/books/remove	DELETE	Remove a book from a wishli
/wishlist/books/{wishlistId}	GET	Get all books in a wishlist.
/wishlist/books/contains	GET	Check if a book exists in a w
/wishlist/books/count	GET	Get the number of books in a
/publishers	GET	Get all publishers.
/publishers/{id}	GET	Get a publisher by ID.

/publishers	POST	Insert a new publisher.
/publishers/{id}	PUT	Update a publisher.
/publishers/{id}	DELETE	Delete a publisher.
/publishers/book/{bookId}	GET	Get publishers of a book.
/publishers/{publisherId}/books	GET	Get books by publisher.
/publishers/{publisherId}/books/count	GET	Count books by a publisher.
/publishers?bookId={bookId} B	publisherId={publisherId}	POST
/publishers?bookId={bookId} B	publisherId={publisherId}	DELETE
/books	GET	Get all books.
/book?id={book_id}	GET	Get a book by ID.
/book	POST	Insert a new book.
/book	PUT	Update book details.
/book/stock?id={book_id} B	quantity={qty}	PUT
/book?id={book_id}	DELETE	Delete a book.
/books/top-rated	GET	Get top-rated books.
/books/search?title={query}	GET	Search books by title.
/books/author?id={author_id}	GET	Books by author.
/books/category?id={category_id}	GET	Books by category.
/books/publisher?id={publisher_id}	GET	Books by publisher.
/orders	GET	Get all orders.
/order/{id}	GET	Get an order by ID.
/order	POST	Place a new order.
/order/user/{userId}	GET	Get all orders for a user.
/order/user/{userId}/latest	GET	Get latest order for a user.
/discount/all	GET	Retrieve all discounts.
/discount/active	GET	Get all active discounts.
/discount/code?code={discount_code}	GET	Get a discount by code.
/discount/query/valid?code={discount_code}	GET	Get valid discount by code.
/discount?id={discount_id}	DELETE	Delete a discount.

Table 2: Summary of REST API Endpoints categorized by entity and functionality

5.4 REST Error Codes

The table below documents all REST error codes implemented in the Bookly platform. These codes are categorized into general HTTP errors and entity-specific errors such as users, carts, discounts, and orders. Each entry consists of the application-specific error code, the corresponding HTTP status code, and a description for clarity.

Error Code	HTTP Status Code	Description
0	200	Request successful.
1	201	Resource created successfully.
2	202	Request accepted but not yet processed.
3	204	No content available.
-1	500	Internal server error.
-2	405	HTTP method not allowed.
-3	400	Bad request.
-4	401	Unauthorized access.
-5	403	Forbidden action.
-6	404	Resource not found.
-7	409	Request conflict.
-8	415	Unsupported media type.
-100	404	User not found.
-101	400	Username is required.
-102	400	First name is required.
-103	400	Last name is required.
-104	400	Email is required.
-105	400	Invalid email format.
-106	400	Password is required.
-107	400	Password is too weak.
-108	409	Passwords do not match.
-109	400	Username or email is required.
-110	400	Invalid username format.
-111	409	User already exists.
-112	400	User registration input is invalid.
-113	401	Session expired or user not logged in.
-114	401	Invalid login credentials.
-100	400	Invalid discount object.
-101	400	Discount code is required.
-102	400	Discount code format is invalid.
-103	400	Discount must be greater than 0%.
-104	400	Discount cannot exceed 100%.
-105	400	Discount expiration date is required.
-106	400	Discount has already expired.
-107	400	Discount must last at least 1 day.
-108	400	Discount must not exceed 365 days.
-109	400	This discount is expired and cannot be used.
-200	404	Cart not found.
-201	400	Cart operation failed.
-202	400	Cart is empty.
-203	409	Item already in cart.
-204	400	Cart object is null or invalid.

-205	400	User ID is invalid.
-206	400	Cart has too many items.
-207	400	Item quantity must be greater than 0.
-208	400	Cart total must be greater than 0.
-209	400	Cart total exceeds maximum allowed value.
-210	400	Shipping method is missing or empty.
-305	400	Invalid or null order object.
-306	400	Order total must be greater than 0.
-307	400	Shipping address is required.
-308	400	Shipment code format is invalid.
-309	400	Order status is invalid.
-310	400	Order date cannot be in the future.
-311	409	Order has already been shipped or delivered and cannot be cancelled.
-999	400	Unknown operation requested.

Table 3: List of defined REST API error codes

5.5   REST API Details

A resource

A REST API (Representational State Transfer Application Programming Interface) allows communication between a client and a server over the HTTP protocol. It defines a set of standards and conventions to access and manipulate resources (data) via endpoints, typically using CRUD operations (Create, Read, Update, Delete).

Users

Sign Up

- URL: /auth/signup

- Method: POST

- URL Parameters: NONE

- Data Parameters:

  Required:
  username: {String},
  email: {String},
  password: {String}

  Optional:
  first_name, last_name, phone, address, role

- Success Response:

  Code: 201 Created
  Content: {"message": "User created successfully.", "201", "username"}

- Error Response:

18

Code: 400 Bad Request

Content: {"error": "Missing required fields"}

When: Required data fields are missing

Code: 409 Conflict

Content: {"error": "Username or email already exists."}

When: Username or email is not unique

Login

- URL: `/auth/login`

- Method: `POST`

- URL Parameters: NONE

- Data Parameters:

  Required:
  username: {String},
  password: {String}

- Success Response:

  Code: 200 OK

  Content: {"message": "Authentication successful.", "token": "jwtToken"}

- Error Response:

  Code: 400 Bad Request

  Content: {"error": "Missing credentials"}

  When: username or password is not provided

  Code: 401 Unauthorized

  Content: {"error": "Authentication failed."}

  When: Invalid username or password

Get User by Username

- URL: `/user?username={username}`

- Method: `GET`

- URL Parameters:

  username: {String} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK

  Content: {user details in JSON format}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "User not found."}

  When: No user with the given username exists

Cart

Retrieve Cart by User ID

- URL: /cart?userId={user_id}

- Method: GET

- URL Parameters:

  userId: {Integer} — ID of the user whose cart is being retrieved

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {cart_details}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing or invalid userId"}

  When: userId is missing or not a valid integer

  Code: 404 Not Found
  Content: {"error": "No cart found for the given user ID"}

  When: The user has no associated cart

Add Book to Cart

- URL: /cart/add?cartId={cart_id}bookId={book_id}

- Method: POST

- URL Parameters:

  cartId: {Integer} — required
  bookId: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Book ID X was added to cart ID Y"}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing or malformed parameters"}

  When: cartId or bookId is missing or incorrectly formatted

  Code: 500 Internal Server Error
  Content: {"error": "Database failure or unexpected error"}

  When: Server error during addition

Remove Book from Cart

- URL: /cart/remove?cartId={cart_id}bookId={book_id}

- Method: DELETE

- URL Parameters:

  cartId: {Integer} — required
  bookId: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Book ID X was removed from cart ID Y"}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing or invalid parameters"}

  When: cartId or bookId is missing or invalid

  Code: 500 Internal Server Error
  Content: {"error": "Failed to remove book from cart"}

  When: Database error or removal operation failed

Reviews

Create Review

- URL: /review

- Method: POST

- URL Parameters: NONE

- Data Parameters:

  Required:
  user_id: {Integer},

book_id: {Integer},
review_content: {String}

Optional:
plot_rating, style_rating, theme_rating: {Integer}

- Success Response:

  Code: 201 Created
  Content: {... inserted review details ...}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Invalid review payload."}

  When: Missing required fields or incorrect data types

Get Review by ID

- URL: /review/{review_id}

- Method: GET

- URL Parameters:
  review_id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {... review object ...}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "Review not found."}

  When: No review exists with the given ID

Delete Review

- URL: /review/{review_id}

- Method: DELETE

- URL Parameters:
  review_id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Review deleted", "id": review_id}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "Review not found."}

  When: The review to delete does not exist

Get All Reviews

- URL: `/reviews`

- Method: `GET`

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{review_details}, ...]

Get Reviews by Book ID

- URL: `/review/book/{book_id}`

- Method: `GET`

- URL Parameters:
  book_id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{review_details}, ...]

Get Reviews by User ID

- URL: `/review/user/{user_id}`

- Method: `GET`

- URL Parameters:
  user_id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{review_details}, ...]

Top Reviews for a Book

- URL: /review/top/book/{book_id}

- Method: GET

- URL Parameters:
  book_id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{top_rated_review_details}, ...]

Authors

Create Author

- URL: /authors

- Method: POST

- URL Parameters: NONE

- Data Parameters:
  Required:
  name: {String}

- Success Response:

  Code: 201 Created
  Content: {"message": "Author inserted successfully.", "author_name": "John Doe"}

- Error Response:

  Code: 500 Internal Server Error
  Content: {"error": "Failed to insert author."}
  When: Database or server failure

Get All Authors

- URL: /authors

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{author_details}, ...]

Get Author by ID

- URL: /authors/{id}

- Method: GET

- URL Parameters:

  id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {author_details}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "Author not found."}
  When: The specified author does not exist

Update Author

- URL: /authors/{id}

- Method: PUT

- URL Parameters:

  id: {Integer} — required

- Data Parameters:

  Required:
  name: {String}

- Success Response:

  Code: 200 OK
  Content: {"message": "Author updated successfully."}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "Author not found."}
  When: No author with the given ID exists

Delete Author

- URL: /authors/{id}

- Method: DELETE

- URL Parameters:

  id: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Author deleted successfully."}

- Error Response:

  Code: 404 Not Found
  Content: {"error": "Author not found."}

  When: The specified author could not be found

Assign Author to Book

- URL: /authors/assign

- Method: POST

- URL Parameters: NONE

- Data Parameters:

  Required:
  bookId: {Integer},
  authorId: {Integer}

- Success Response:

  Code: 200 OK
  Content: {"message": "Author assigned to book."}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing or invalid parameters."}

  When: bookId or authorId is missing or invalid

Remove Author from Book

- URL: /authors/remove

- Method: DELETE

- URL Parameters: NONE

- Data Parameters:

  Required:
  bookId: {Integer},
  authorId: {Integer}

- Success Response:

  Code: 200 OK
  Content: {"message": "Author removed from book."}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing or invalid parameters."}

  When: bookId or authorId is missing or invalid

Wishlist

Create Wishlist

- URL: /wishlist?userId={user_id}

- Method: POST

- URL Parameters:

  userId: {Integer} — ID of the user

- Data Parameters: NONE

- Success Response:

  Code: 201 Created
  Content: {"wishlistId": 6789, "userId": 123, ...}

- Error Response:

  Code: 400 Bad Request
  Content: {"error": "Missing userId parameter."}

  When: userId is not provided

Delete Wishlist

- URL: /wishlist/{wishlistId}

- Method: DELETE

- URL Parameters:
  wishlistId: {Integer} — ID of the wishlist

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Wishlist deleted successfully."}

Get Wishlists by User

- URL: /wishlist/user/{userId}

- Method: GET

- URL Parameters:
  userId: {Integer} — ID of the user

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"wishlists": [...]}

Clear Wishlist

- URL: /wishlist/clear/{wishlistId}

- Method: POST

- URL Parameters:
  wishlistId: {Integer} — ID of the wishlist

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"message": "Wishlist cleared successfully."}

Add Book to Wishlist

- URL: /wishlist/books/add

- Method: POST

- URL Parameters: NONE

- Data Parameters:

  wishlistId: {Integer} — required
  bookId: {Integer} — required

- Success Response:

  Code: 200 OK
  Content: {"message" : "Book ID: X added to Wishlist ID: Y"}

Remove Book from Wishlist

- URL: /wishlist/books/remove

- Method: DELETE

- URL Parameters: NONE

- Data Parameters:

  wishlistId: {Integer} — required
  bookId: {Integer} — required

- Success Response:

  Code: 200 OK
  Content: {"message" : "Book ID: X removed from Wishlist ID: Y"}

Get Books in Wishlist

- URL: /wishlist/books/{wishlistId}

- Method: GET

- URL Parameters:

  wishlistId: {Integer} — ID of the wishlist

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: [{book_details}, ...]

Check Book in Wishlist

- URL: /wishlist/books/contains

- Method: GET

- Query Parameters:

  wishlistId: {Integer} — required
  bookId: {Integer} — required

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"exists": true/false}

Count Books in Wishlist

- URL: /wishlist/books/count

- Method: GET

- Query Parameters:

  wishlistId: {Integer} — ID of the wishlist

- Data Parameters: NONE

- Success Response:

  Code: 200 OK
  Content: {"count": N}

Publishers

Get All Publishers

- URL: /publishers

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Description: Retrieves all publishers.

- Success Response:
  Code: 200 OK
  Content: [{publisher_details}, ...]

Get Publisher by ID

- URL: /publishers/{id}

- Method: `GET`

- URL Parameters: id: {Integer} — required

- Data Parameters: NONE

- Description: Retrieves a specific publisher by ID.

- Success Response:
  Code: 200 OK
  Content: {publisher_details}

- Error Response:
  Code: 404 Not Found
  Content: {"error" : "Publisher not found."}

Insert Publisher

- URL: /publishers

- Method: `POST`

- URL Parameters: NONE

- Data Parameters:

  Required:
  publisherName: {String}

  Optional:
  phone: {String},
  address: {String}

- Description: Inserts a new publisher.

- Success Response:
  Code: 201 Created
  Content: {"message" : "Publisher inserted successfully." }

- Error Response:
  Code: 500 Internal Server Error

Update Publisher

- URL: /publishers/{id}

- Method: `PUT`

- URL Parameters: id: {Integer} — required

- Data Parameters:

  Required:
  publisherName: {String}

  Optional:
  phone: {String},
  address: {String}

- Description: Updates an existing publisher.

- Success Response:
  Code: 200 OK
  Content: {"message": "Publisher updated successfully."}

- Error Response:
  Code: 404 Not Found

Delete Publisher

- URL: /publishers/{id}

- Method: DELETE

- URL Parameters: id: {Integer} — required

- Data Parameters: NONE

- Description: Deletes a publisher by ID.

- Success Response:
  Code: 200 OK
  Content: {"message": "Publisher deleted successfully."}

- Error Response:
  Code: 404 Not Found

Get Publishers by Book

- URL: /publishers/book/{bookId}

- Method: GET

- URL Parameters: bookId: {Integer} — required

- Data Parameters: NONE

- Description: Gets all publishers assigned to a book.

- Success Response:
  Code: 200 OK
  Content: ["Publisher A", "Publisher B"]

Get Books by Publisher

- URL: /publishers/{publisherId}/books

- Method: GET

- URL Parameters: publisherId: {Integer} — required

- Data Parameters: NONE

- Description: Gets all books published by a given publisher.

- Success Response:
  Code: 200 OK
  Content: [{book_details}, ...]

Count Books by Publisher

- URL: /publishers/{publisherId}/books/count

- Method: GET

- URL Parameters: publisherId: {Integer} — required

- Data Parameters: NONE

- Description: Returns the number of books published by a given publisher.

- Success Response:
  Code: 200 OK
  Content: {"bookCount": 10}

Assign Publisher to Book

- URL: /publishers?bookId={bookId}publisherId={publisherId}

- Method: POST

- URL Parameters:

  bookId: {Integer},
  publisherId: {Integer}

- Data Parameters: NONE

- Description: Assigns a publisher to a book.

- Success Response:
  Code: 200 OK
  Content: {"message": "Assigned"}

- Error Response:
  Code: 400 Bad Request

Remove Publisher from Book

- URL: `/publishers?bookId={bookId}publisherId={publisherId}`

- Method: `DELETE`

- URL Parameters:

  bookId: {Integer},
  publisherId: {Integer}

- Data Parameters: NONE

- Description: Removes a publisher assignment from a book.

- Success Response:
  Code: 200 OK
  Content: {"message" : "Removed"}

- Error Response:
  Code: 400 Bad Request

Books

All Books

- URL: `/books`

- Method: `GET`

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: `[ {"book_id":  "1", "title":  "Title", ...}, ...  ]`

- Error Response: NONE

Get Book by ID

- URL: `/book?id={book_id}`

- Method: `GET`

- URL Parameters: `book_id`: {String} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: { "book_id" : "1", "title" : "Book Title", ... }

- Error Response:
  Code: 404 Not Found
  Content: {"error" : "Book not found."}

Insert Book

- URL: `/book`

- Method: `POST`

- URL Parameters: NONE

- Data Parameters:

  Required:
  title: {String},
  author_id: {Integer},
  publisher_id: {Integer},
  price: {Float}

  Optional:
  language, isbn, edition, publication_year, number_of_pages, stock_quantity, average_rate, summary

- Success Response:
  Code: 201 Created
  Content: { "book_id": "12345", "title": "New Book Title", "author": "Author Name", "publisher": "Publisher Name" }

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing or invalid book details."}

Update Book

- URL: `/book`

- Method: `PUT`

- URL Parameters: NONE

- Data Parameters:

  Required:
  title: {String},
  author_id: {Integer},
  publisher_id: {Integer},
  price: {Float}

  Optional:
  language, isbn, edition, publication_year, number_of_pages, stock_quantity, average_rate, summary

- Success Response:
  Code: 200 OK
  Content: {"message": "Book updated successfully."}

- Error Response:
  Code: 404 Not Found
  Content: {"error": "Book not found."}

Update Stock Quantity

- URL: /book/stock?id={book_id}quantity={qty}

- Method: PUT

- URL Parameters:

  id: {Integer} — ID of the book
  quantity: {Integer} — new stock quantity

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {"message": "Stock updated"}

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing parameters"}

Delete Book

- URL: /book?id={book_id}

- Method: DELETE

- URL Parameters: id: {Integer} — ID of the book

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {"message": "Book deleted"}

- Error Response:
  Code: 404 Not Found
  Content: {"error": "Book with ID not found."}

Top Rated Books

- URL: /books/top-rated

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{ "book_id": 1, "title": "Title", "average_rate": 4.5, ... }, ...]

- Error Response:
  Code: 500 Internal Server Error
  Content: {"error": "Unable to fetch top rated books."}

Search by Title

- URL: /books/search?title={query}

- Method: GET

- URL Parameters: title: {String} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{ "book_id": 1, "title": "Book Title", "author": "Author Name", ... }, ...
  ]

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing 'title' query parameter."}

  Code: 404 Not Found
  Content: {"error": "No books found matching title query."}

Books by Author, Category, or Publisher

- URL: /books/author?id={author_id}, /books/category?id={category_id}, /books/publisher?id={publisher_id}

- Method: GET

- URL Parameters: id: {Integer} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{ "book_id": 1, "title": "Book Title", "author": "Author Name", ... }, ...
  ]

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing or invalid 'id' parameter."}

  Code: 404 Not Found
  Content: {"error": "No books found for the specified ID."}

Orders

Get All Orders

- URL: /orders

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{order_details}, ...]

- Error Response:
  Code: 500 Internal Server Error

Get Order by ID

- URL: /order/{id}

- Method: `GET`

- URL Parameters: id: {Integer} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {order_details}

- Error Response:
  Code: 404 Not Found

Insert New Order

- URL: /order

- Method: `POST`

- URL Parameters: NONE

- Data Parameters:

  JSON body containing order details

- Success Response:
  Code: 201 Created
  Content: {"message": "Order inserted successfully."}

- Error Response:
  Code: 500 Internal Server Error

Get All Orders for User

- URL: /order/user/{userId}

- Method: `GET`

- URL Parameters: userId: {Integer} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{order_details}, ...]

- Error Response:
  Code: 404 Not Found

Get Latest Order for User

- URL: /order/user/{userId}/latest

- Method: GET

- URL Parameters: userId: {Integer} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {order_details}

- Error Response:
  Code: 404 Not Found

Discounts

Retrieve All Discounts

- URL: /discount/all

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{discount_details}, ...]

Retrieve Active Discounts

- URL: /discount/active

- Method: GET

- URL Parameters: NONE

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: [{discount_details}, ...]

Retrieve Discount by Code

- URL: /discount/code?code={discount_code}

- Method: GET

- URL Parameters: code: {String} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {discount_details}

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing code."}

  Code: 404 Not Found
  Content: {"error": "Discount not found."}

Retrieve Valid Discount by Code

- URL: /discount/query/valid?code={discount_code}

- Method: GET

- URL Parameters: code: {String} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {discount_details}

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing or invalid discount code."}

  Code: 404 Not Found
  Content: {"error": "Valid discount not found."}

Delete Discount

- URL: /discount?id={discount_id}

- Method: DELETE

- URL Parameters: id: {Integer} — required

- Data Parameters: NONE

- Success Response:
  Code: 200 OK
  Content: {"message": "Discount deleted successfully."}

- Error Response:
  Code: 400 Bad Request
  Content: {"error": "Missing or invalid discount ID."}

  Code: 404 Not Found
  Content: {"error": "Discount not found."}

6 Group Members Contribution

Parisa Saeedidana Wrote the objective of the project, Drew the ER diagram for the bookstore.

Amirreza Jolani mameghani Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Jane Doe Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

John Doe Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Jane Doe Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

John Doe Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.