1222·2022 800 ANNI
UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

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**
Ben Yamoune	Ayoub	2163617
Jolani Mameghani	Amirreza	2106526
Mia	Md Abu Raihan	2142011
Saeedidana	Parisa	2142717
Truty	Adam	2142741

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

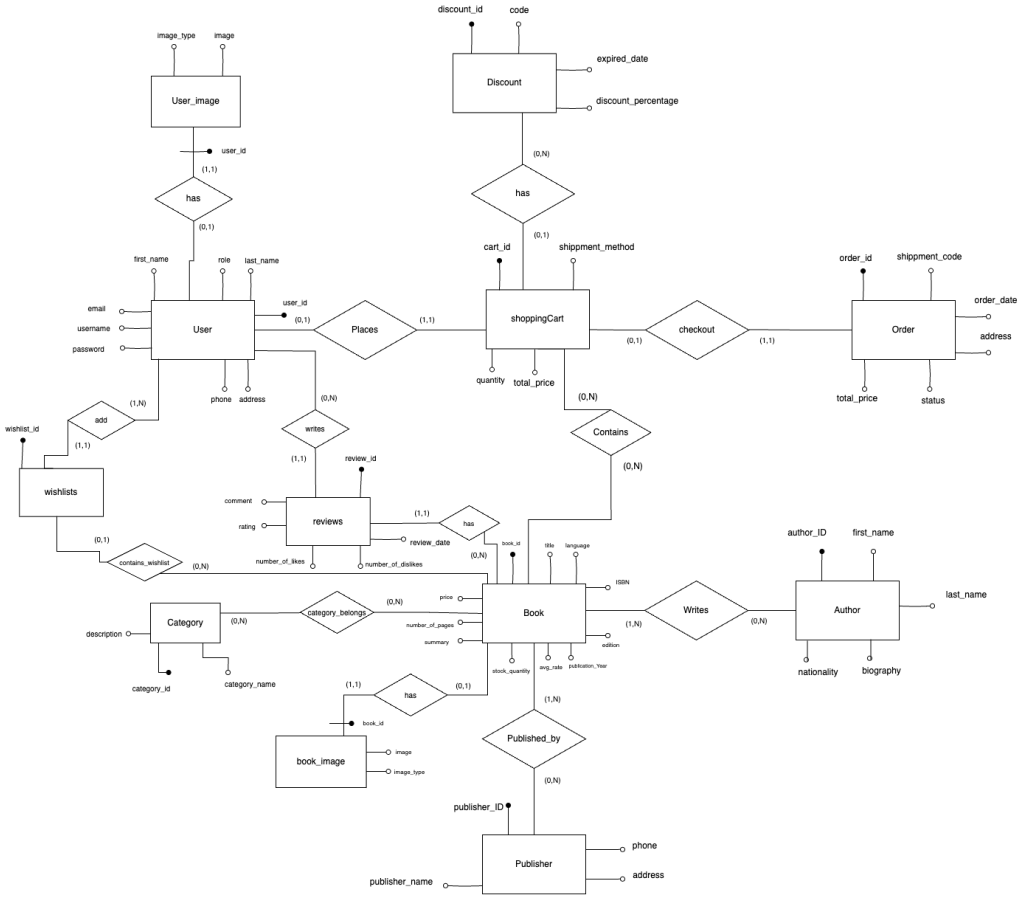

Figure 1: ER-Diagram

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 achievements, 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.
- Order 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

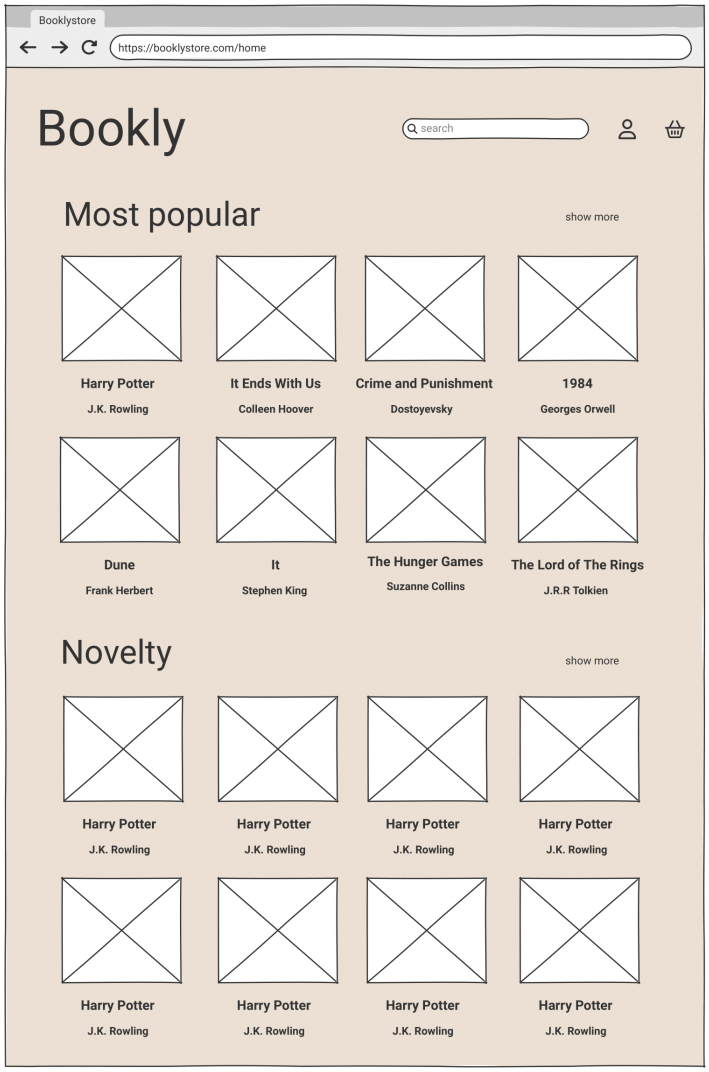

Figure 2: home page

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.

Each section will include a **"Show More"** link that redirects users to the corresponding full section, displaying all books it contains. Additionally, the homepage will also 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

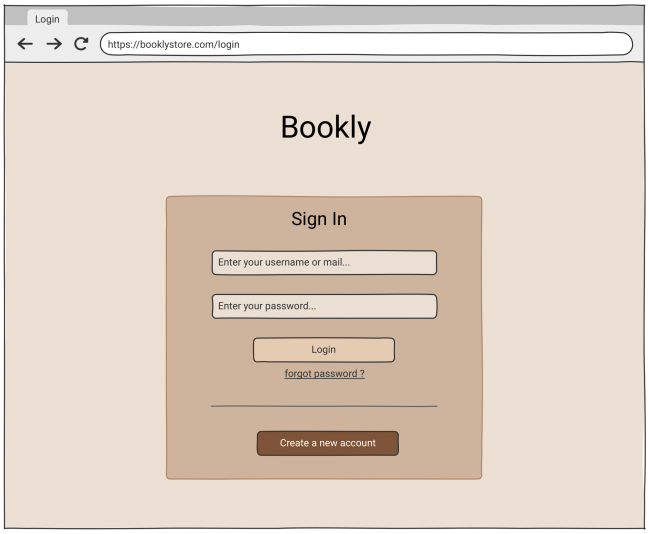

Figure 3: sign in page

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**.

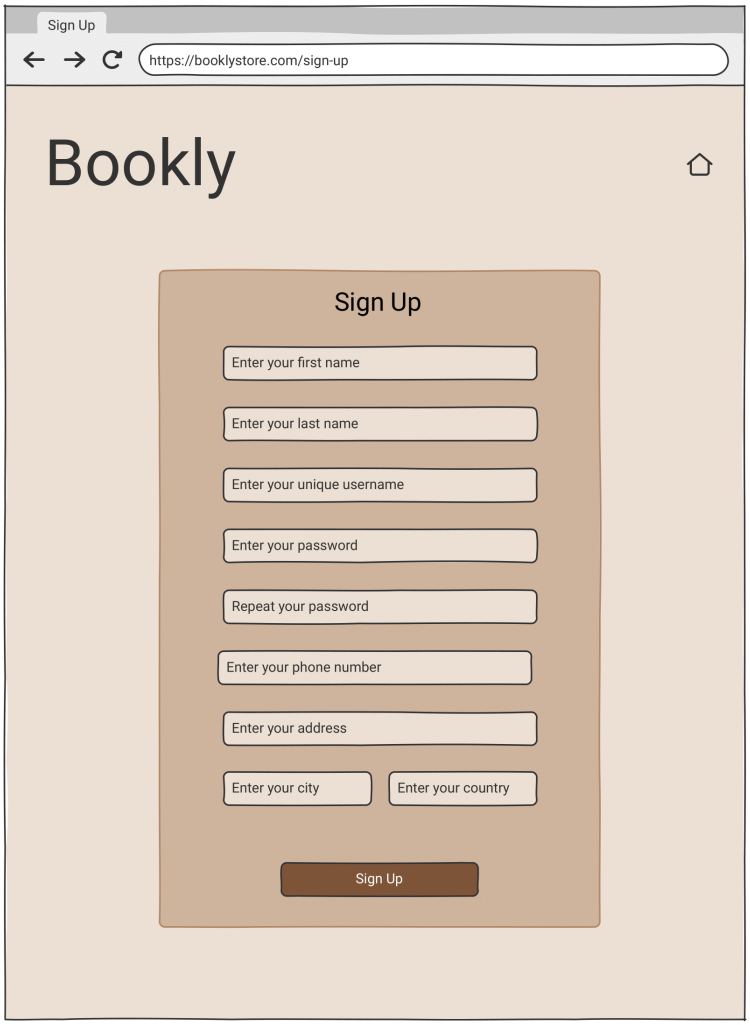

Figure 4: sign up page

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

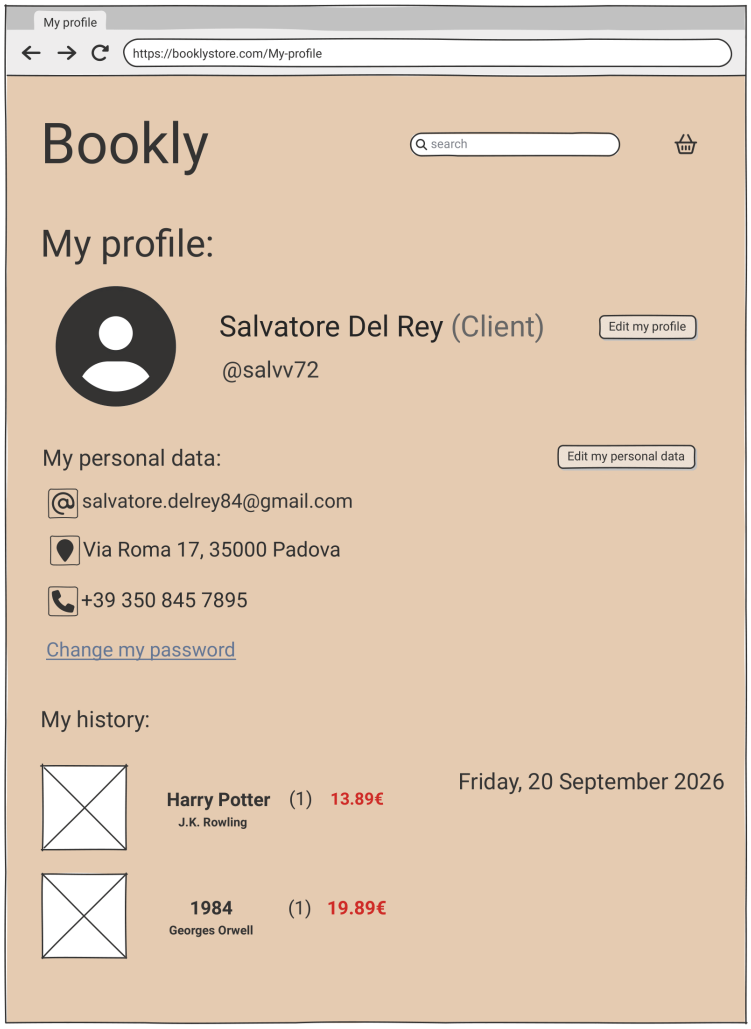

Figure 5: profile page

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

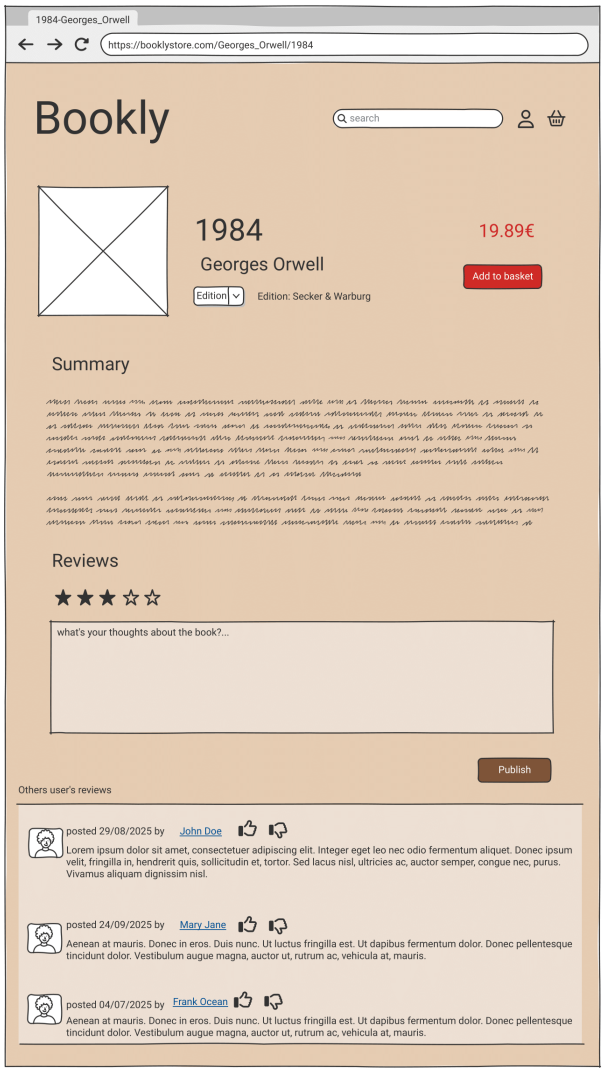

Figure 6: book page

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.

The page will also include:

- An **"Add to Basket"** button, allowing users to add the book to their shopping cart.

- An **Edition picker**, allowing users to choose the edition of the book they are interested in.
- An **"Add to Wishlist"** button, allowing users to save the book to their Wishlist Page.
- A **Review Section**, where users can:
  - Like or dislike existing reviews from other users.
  - Add their own review by writing a comment and rating the book on a scale from **0 to 5 stars**.

4.5 Basket Page

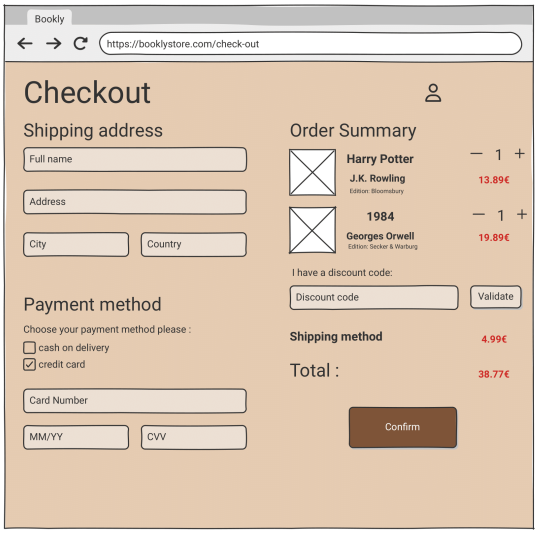

Figure 7: basket page

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, authors, editions, prices, and quantities. Quantities will be editable.
- An optional input field for entering a **discount code** if the user has one.
- 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**.
- A **"Confirm Order"** button that finalizes the payment and processes the order once all required fields are completed.

4.6 Search Results Page

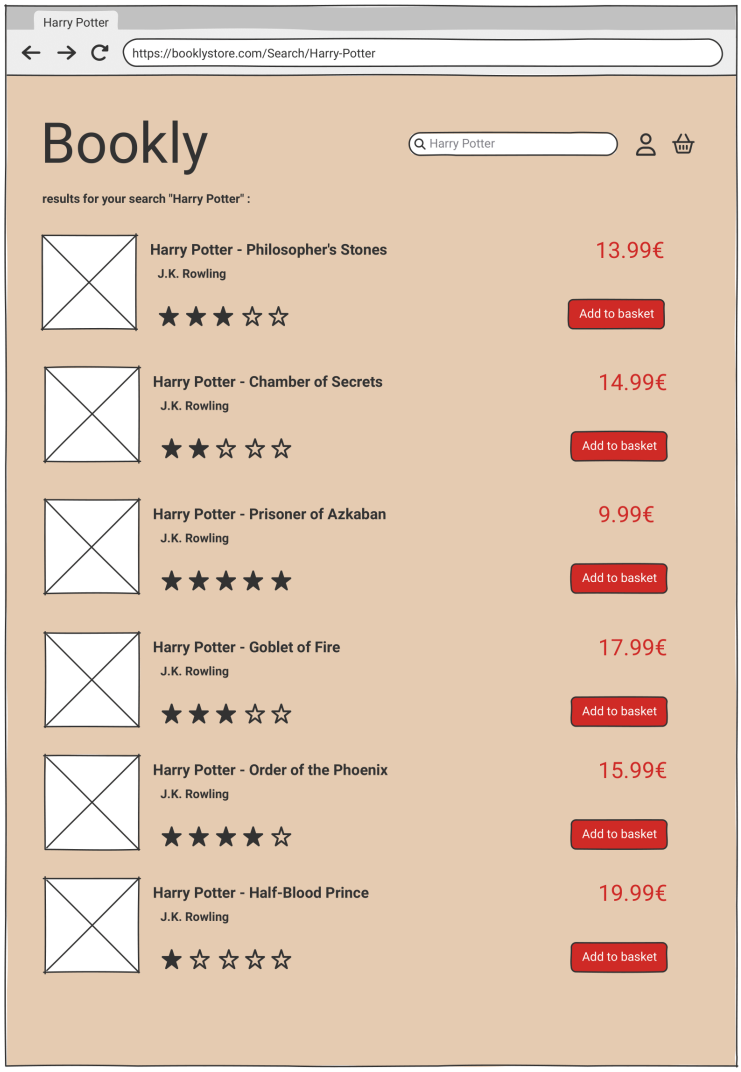

Figure 8: search page

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

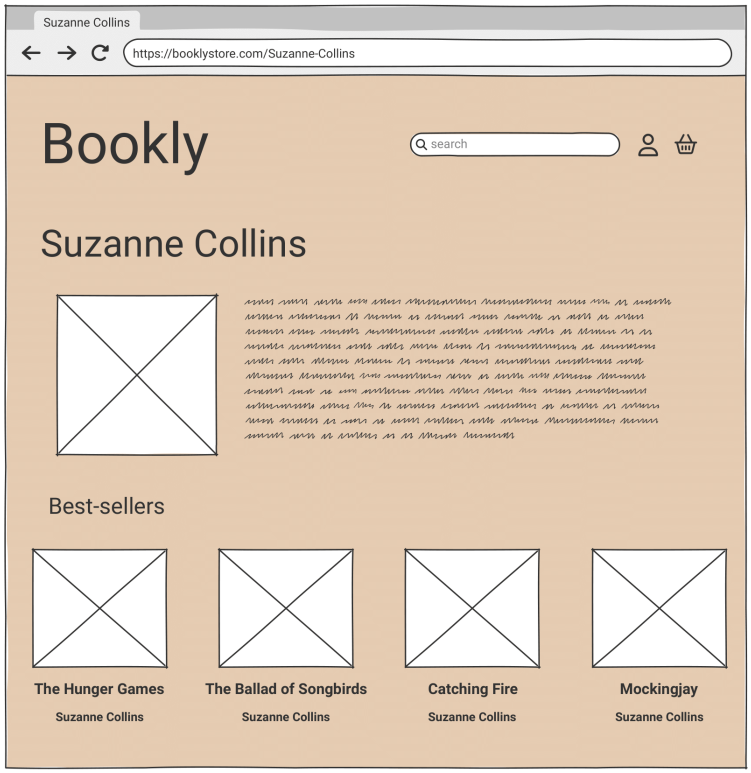

Figure 9: author page

Each author will have their own dedicated page displaying their full name, picture, and a description containing their biography and literary style. The page will also help users to explore more books from the same author by displaying their best-sellers. If a user clicks on one of the book icons, they will be redirected to the corresponding Book Page.

4.8 Order Page

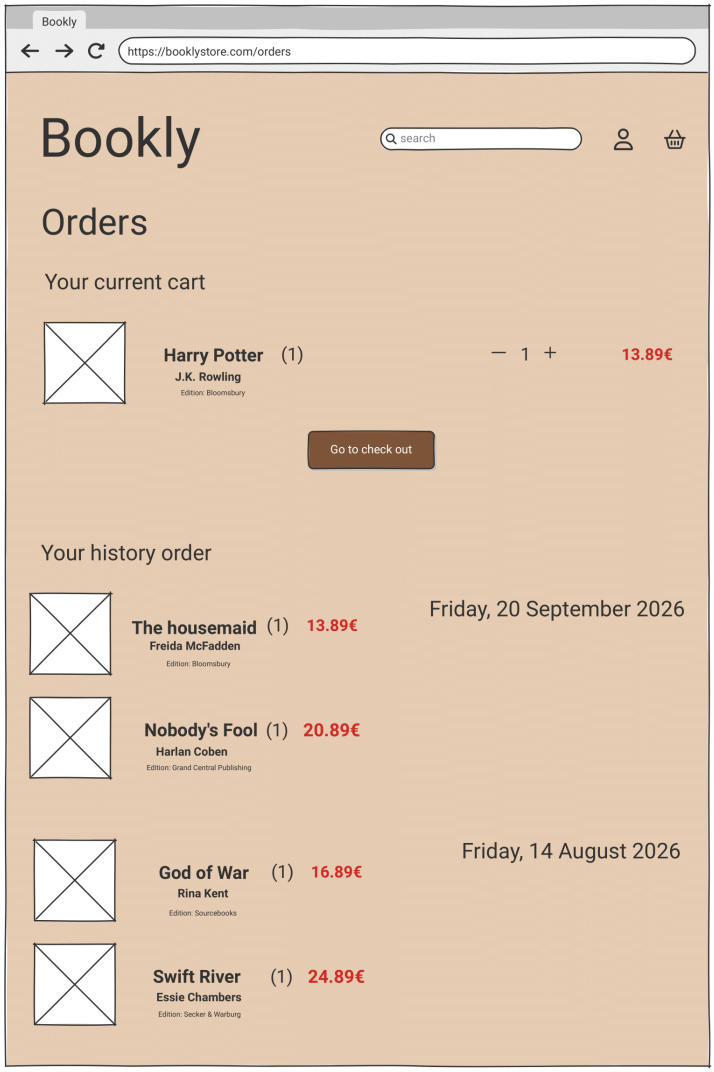

Figure 10: order page

This page will present the contents of 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 carts are valid only for the current session of the logged-in user.

Each cart will include a **"Check Out"** button, allowing the user to directly proceed with the purchase of the corresponding cart.

Also, this page will display the user's complete order history, including all the previously ordered books and their corresponding order dates.

4.9 Wishlist Page

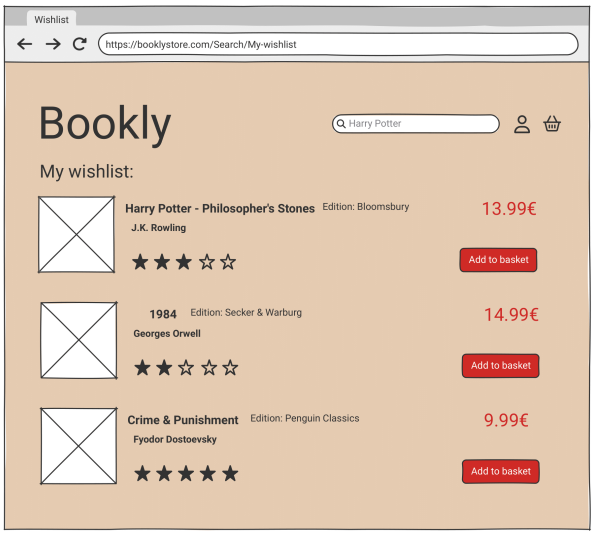

Figure 11: wishlist page

The wishlist page displays all the books that the user has chosen to save for future reference. Users can move them directly into their basket from this page 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).

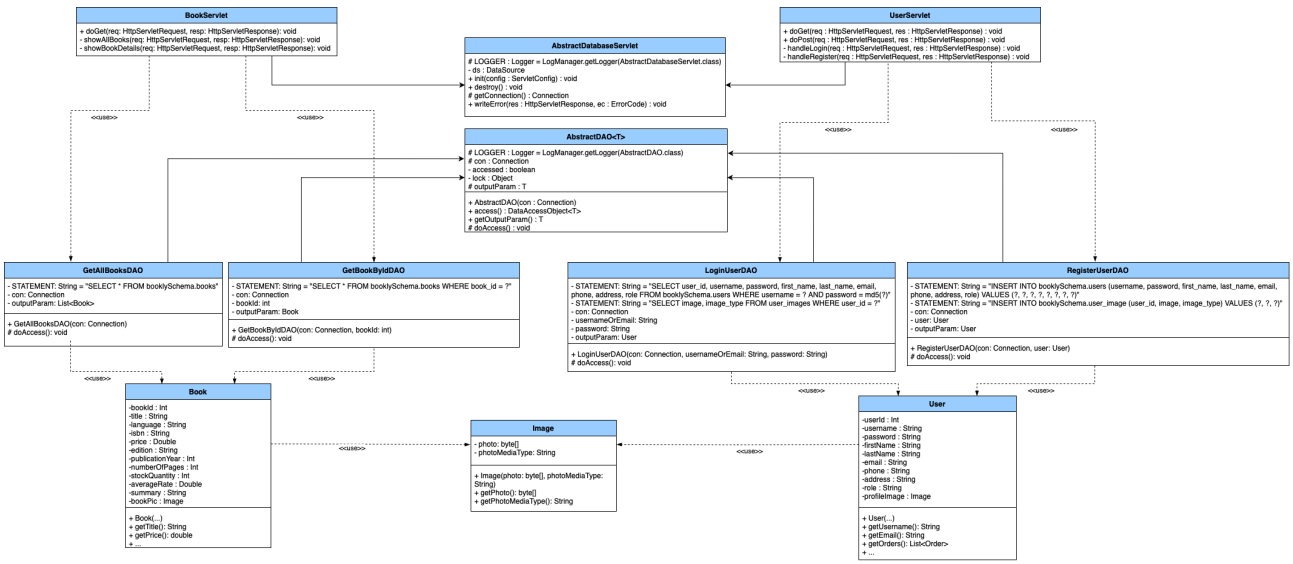

Figure 12: Class diagram of the Bookly web application

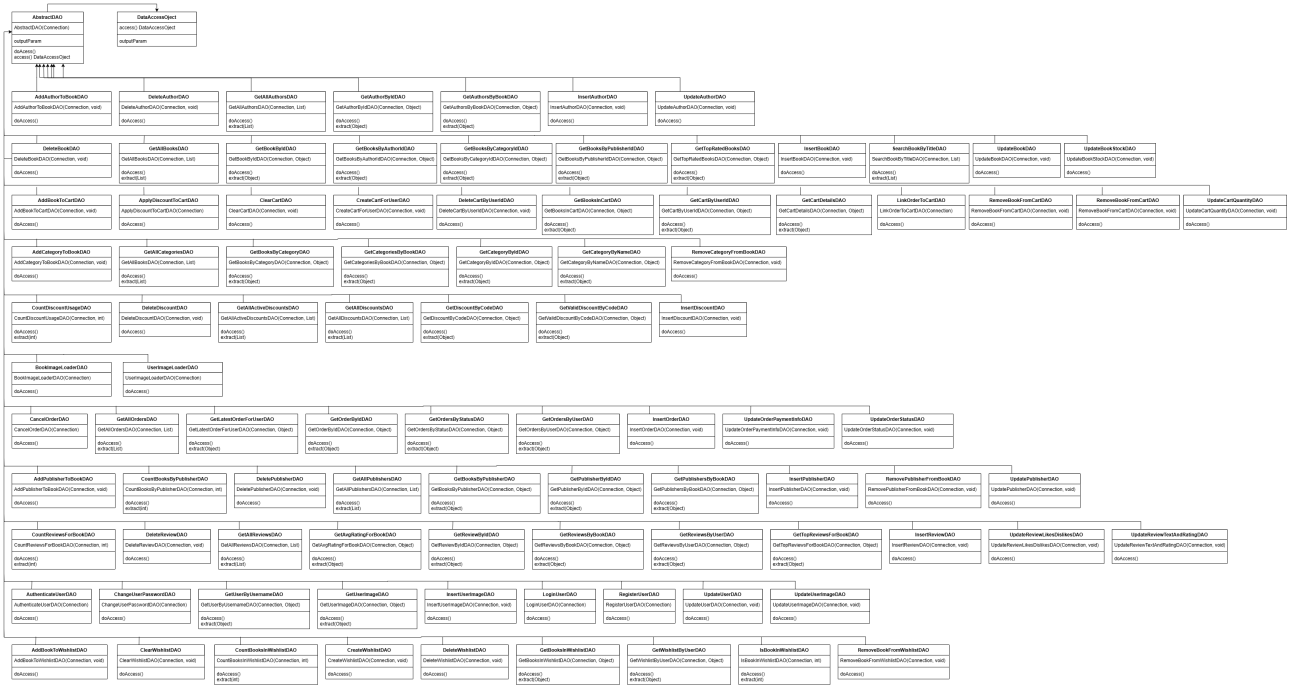

Figure 13: DAO Classes diagram of the Bookly web application

5.2 Sequence Diagram

This sequence diagram (see Figure 14) 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 15 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.

Figure 16 describes the user registration flow in the web application. When a user submits the registration form, the `UserServlet` is initialized by the web server and processes the request through its `doPost()` method.

It extracts user data from the request, generates a JWT token via *JwtManager*, and stores it in the session. The servlet then creates a *RegisterUserDAO* instance to interact with the database, where it performs an SQL *INSERT* to save the new user. Once the user is successfully registered, the servlet saves the user object in the session and forwards the client to the *userProfile.jsp* page, completing the registration process.

The sequence diagram Figure 17 shows what happens when a user sends a `GET` request to the `/author` endpoint. Upon receiving the request, the web server instantiates the `AuthorServlet` (if not already initialized), calls its `init(ServletConfig)` method, and then invokes the `service(HttpServletRequest, HttpServletResponse)` method. The request is then passed to the `doGet()` method of `AuthorServlet`, where logging is set up and a `GetAllAuthorsDAO` object is created using a database connection. The DAO executes an SQL `SELECT` query to fetch all authors and returns the result as a list, which the servlet stores in a request attribute named `"all_authors"` before forwarding the request to the `allAuthors.jsp` page for rendering. In another flow, if an author ID is present in the request path, it is extracted and passed to a `GetBooksByAuthorIdDAO` object to retrieve all books by that author. After executing a `SELECT` query and returning a list of books, the servlet sets additional request attributes such as `"author_books"` and `"author_id"` and forwards the request to `authorBook.jsp`. Optionally, user data may be stored in the session, and the request is redirected to `userProfile.jsp`. Depending on the request context, the servlet dynamically selects the appropriate JSP to generate the final HTML response returned to the user.

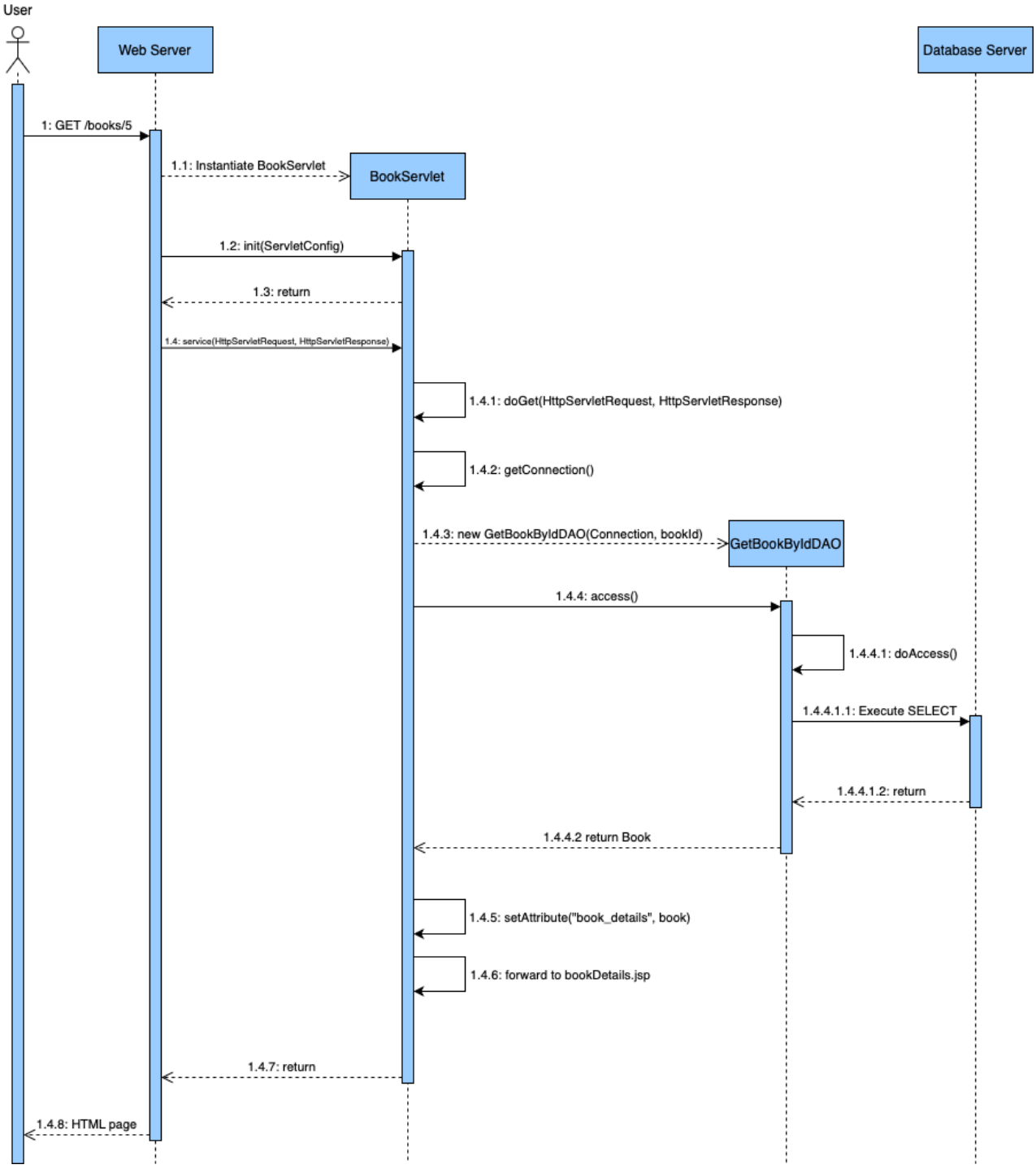

Figure 14: Sequence Diagram for retrieving a book by ID

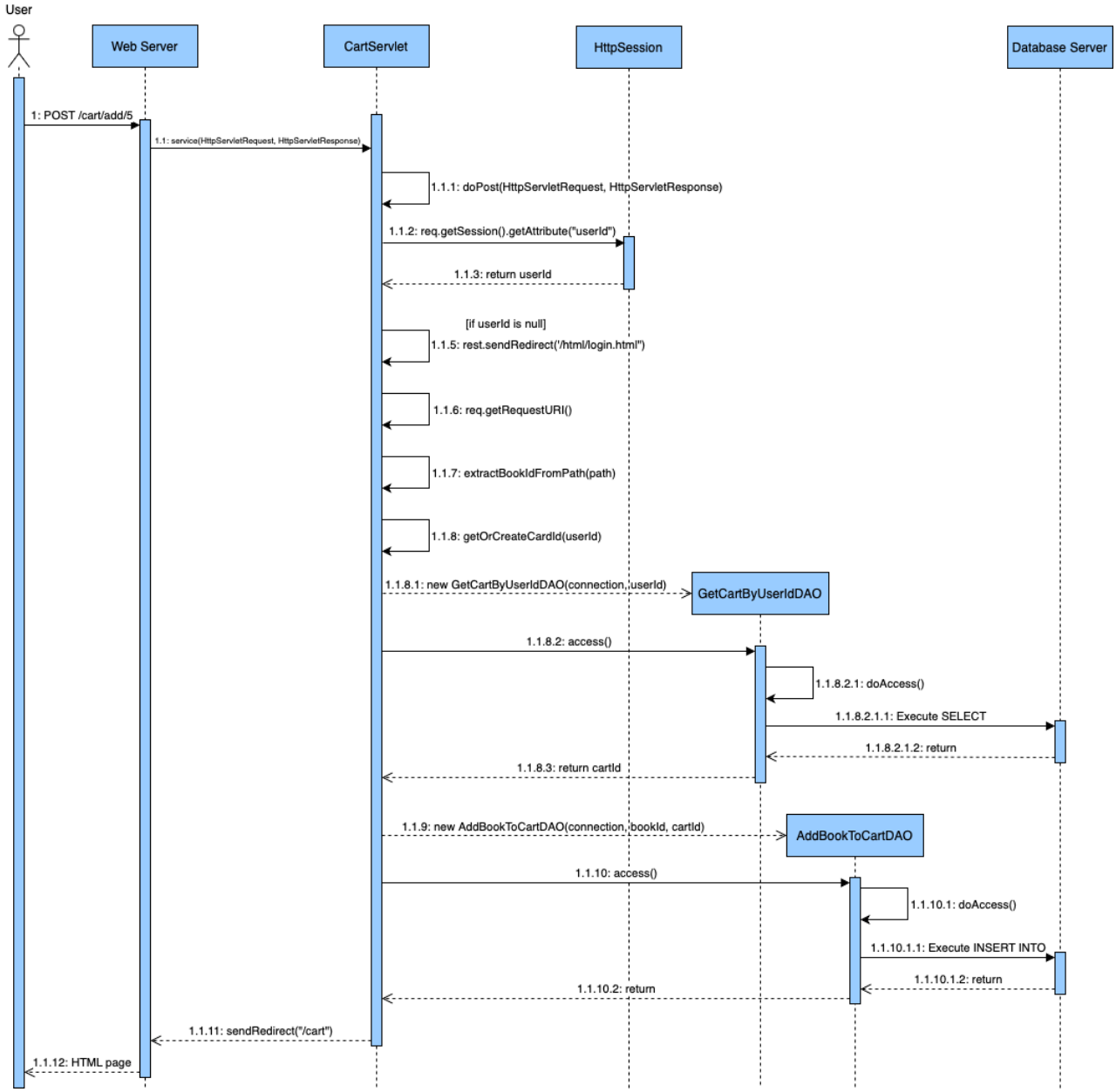

Figure 15: Sequence Diagram for adding a book to the cart

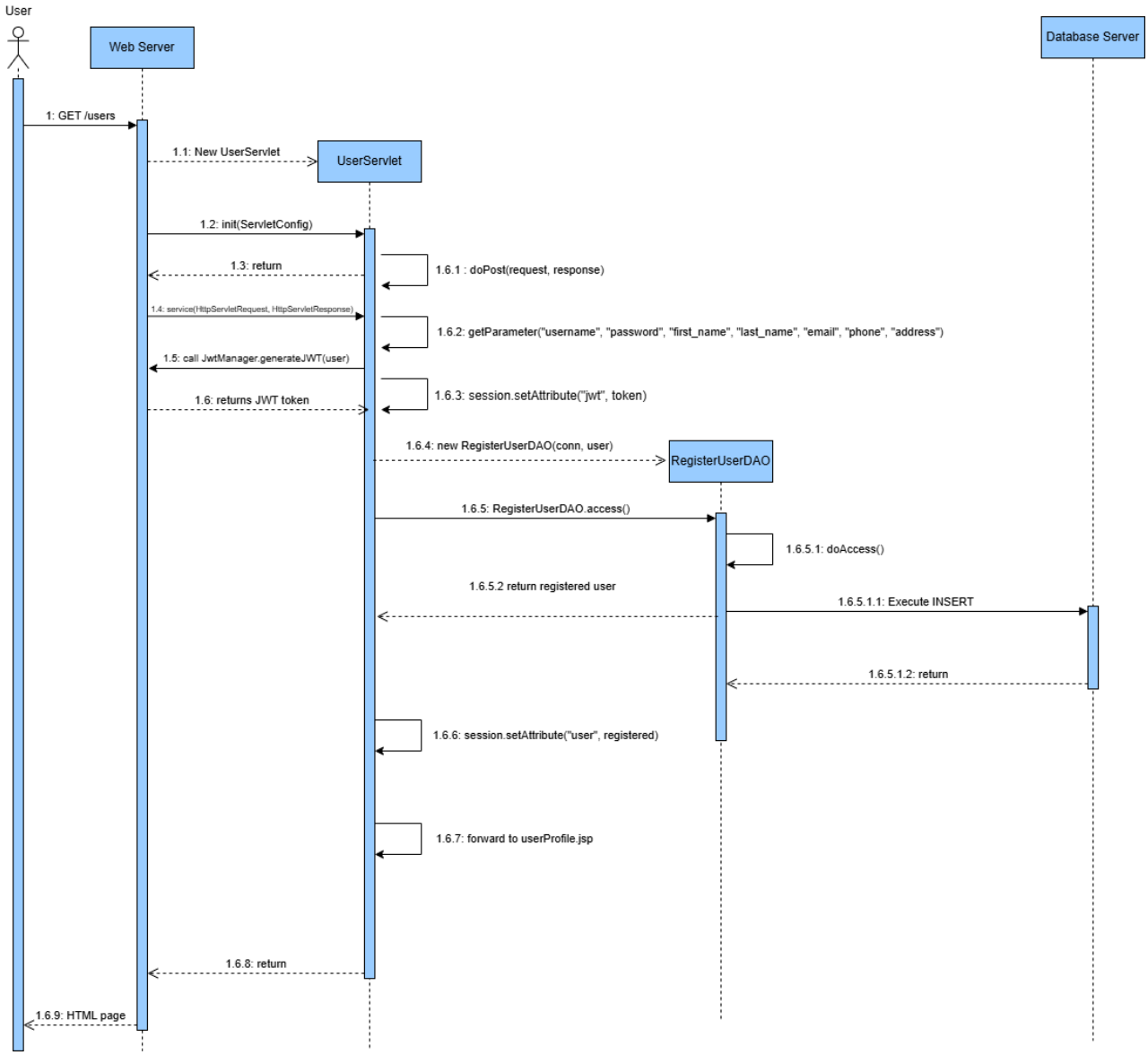

Figure 16: Sequence Diagram for user registration

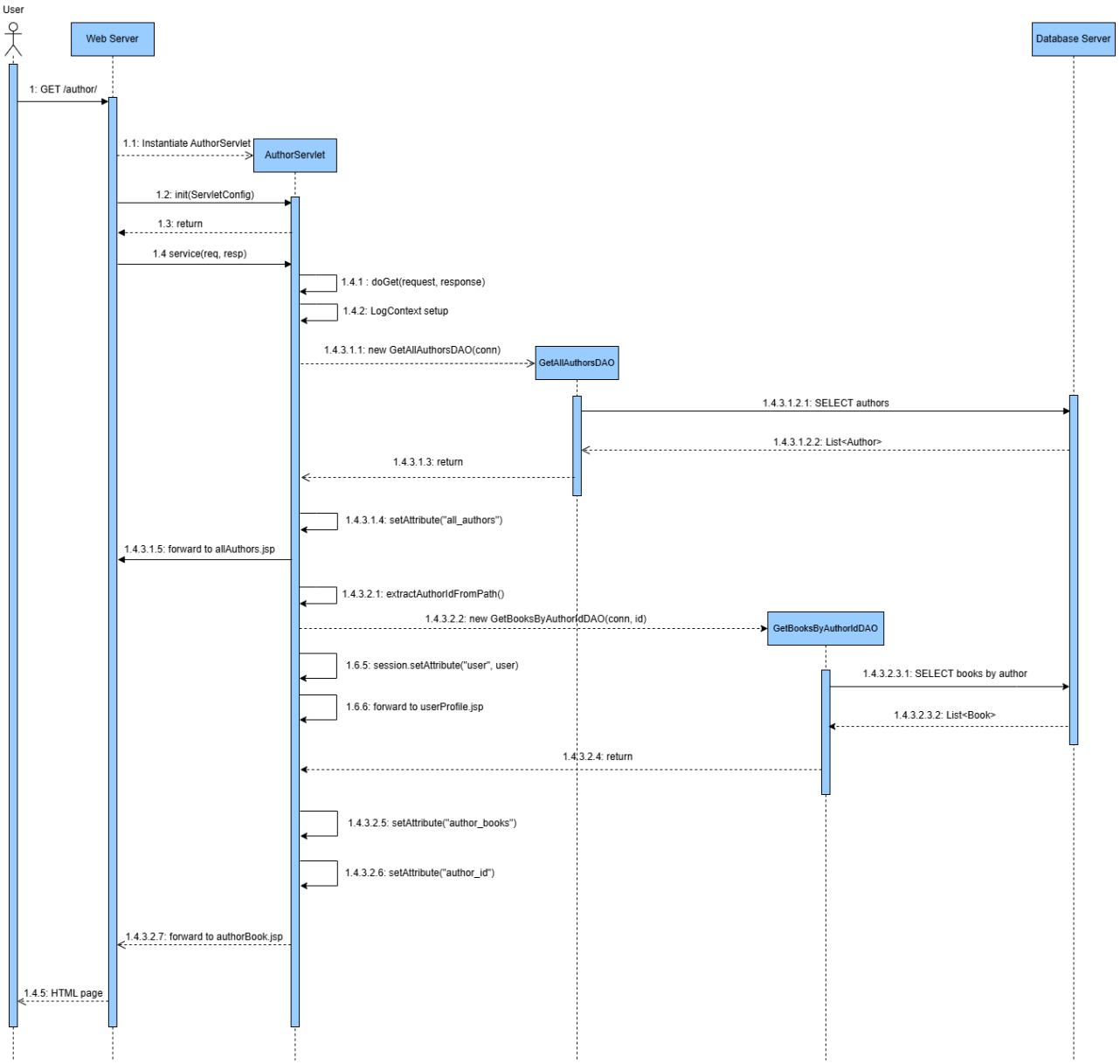

Figure 17: Sequence Diagram for Author servlet

5.3 REST API Summary

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

URI: The specific endpoint for each action.

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

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

URI	Method	Description	F

/user/signup	POST	Register a new user.	A
/user/login	POST	Login a user and return token.	A
/cart?userId={id}	GET	Get user's cart.	A
/cart/add?cartId={id}&bookId={id}	POST	Add book to cart.	A
/cart/remove?cartId={id}&bookId={id}	DELETE	Remove book from cart.	A
/author	GET	Get all authors.	B
/author/{id}	GET	Get author by ID.	B
/admin/authors	POST	Create new author.	B
/admin/authors/{id}	DELETE	Delete author.	B
/wishlist?userId={id}	POST	Create wishlist.	A
/wishlist	GET	Get user wishlists.	A
/wishlist/add/{id}	POST	Add book to wishlist.	A
/wishlist/remove/{id}	DELETE	Remove book from wishlist.	A
/publisher	GET	Get all publishers.	B
/publisher/{id}	GET	Get publisher by ID.	B
/admin/publishers	POST	Create publisher.	B
/admin/publishesr/{id}	DELETE	Delete publisher.	B
/book	GET	Get all books.	B
/book/{id}	GET	Get book by ID.	B
/admin/books	POST	Add new book.	B
/admin/books?id={id}	DELETE	Delete book.	B
/books/top-rated	GET	Get top-rated books.	B
/books/author?id={id}	GET	Books by author.	B
/books/category?id={id}	GET	Books by category.	B
/books/publisher?id={id}	GET	Books by publisher.	B
/admin/discounts	GET	All discounts.	M
/checkout	GET	Display checkout page.	A
/checkout	POST	Submit order from cart.	A
/orders	GET	Get user's orders.	A
/orders	POST	Place new order.	A

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

REST API Details

The following section provides documentation for all RESTful endpoints in the Bookly system. Each entry describes its purpose, HTTP method, required parameters, and both success and error responses.

User Authentication

Sign Up
Registers a new user with profile data.
- URL: /user/signup
- Method: POST
- URL Parameters: NONE
- Data Parameters: username, email, password, firstName, lastName, phone, address
– Success Response:
Code: 201
Content: {"message": "User created successfully."}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Missing required fields"}
When: Required fields are missing
Code: 409 Conflict
Content: {"error": "Username or email already exists."}
When: Username/email already registered

Login
Authenticates a user and returns a session token.
- URL: /user/login
- Method: POST
- URL Parameters: NONE
- Data Parameters: username, password
– Success Response:
Code: 200
Content: {"token": "user-session-token"}
– Error Response:
Code: 401 Unauthorized
Content: {"error": "Invalid login credentials."}
When: Username or password is incorrect
Code: 401 Unauthorized
Content: {"error": "Session expired or user not logged in."}
When: Session has timed out

Get User's Cart
Retrieves the contents of the user's cart.
- URL: /cart?userId={id}

- Method: GET
- URL Parameters: userId (integer)
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"cartId": 123, "items": [...]}
– Error Response:
Code: 404 Not Found
Content: {"error": "Cart not found."}
When: Cart with the given user ID does not exist
Code: 400 Bad Request
Content: {"error": "User ID is invalid."}
When: Provided user ID is not a valid integer

Add Book to Cart
Adds a book to the user's cart using the cart ID and book ID.
- URL: /cart/add?cartId={id}&bookId={id}
- Method: POST
- URL Parameters: cartId (integer), bookId (integer)
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"message": "Book added to cart."}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Cart is full."}
When: Cart has too many items
Code: 400 Bad Request
Content: {"error": "Item already in cart."}
When: The book is already in the cart
Code: 400 Bad Request
Content: {"error": "Item quantity must be greater than 0."}
When: Book quantity is not valid
Code: 404 Not Found
Content: {"error": "Cart not found."}
When: Cart ID does not exist

Remove Book from Cart
Removes a specific book from the user's cart.
- URL: /cart/remove?cartId={id}&bookId={id}
- Method: DELETE
- URL Parameters: cartId (integer), bookId (integer)
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"message": "Book removed from cart."}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Cart operation failed."}
When: The book could not be removed from the cart
Code: 404 Not Found

Content: {"error": "Cart not found."}
When: Cart with the given ID does not exist
Code: 400 Bad Request
Content: {"error": "User ID is invalid."}
When: Provided user ID is invalid

Create Wishlist
Creates a new wishlist for the specified user.
- URL: /wishlist?userId={id}
- Method: POST
- URL Parameters: userId (integer)
- Data Parameters: NONE
– Success Response:
Code: 201
Content: {"message": "Wishlist created successfully."}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Invalid user ID."}
When: Provided user ID is not valid
Code: 409 Conflict
Content: {"error": "User already has a wishlist."}
When: Wishlist already exists for this user

Get User Wishlists
Retrieves all wishlists.
- URL: /wishlist
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"wishlists": [{...wishlist1...}, {...wishlist2...}]}
– Error Response:
Code: 500 Internal Server Error
Content: {"error": "Unable to retrieve wishlists."}
When: Server-side issue occurs

Add Book to Wishlist
Adds a book to the wishlist by wishlist ID.
- URL: /wishlist/add/{id}
- Method: POST
- URL Parameters: id (integer) — ID of the wishlist
- Data Parameters: bookId (integer)
– Success Response:
Code: 200
Content: {"message": "Book added to wishlist."}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Book already in wishlist."}
When: The book is already part of the wishlist
Code: 404 Not Found

Content: {"error": "Wishlist not found."}
When: No wishlist exists with the provided ID
Code: 400 Bad Request
Content: {"error": "Invalid input data."}
When: Missing or malformed parameters

Remove Book from Wishlist
Removes a book from the wishlist by wishlist ID.
- URL: /wishlist/remove/{id}
- Method: DELETE
- URL Parameters: id (integer) — ID of the wishlist
- Data Parameters: bookId (integer)
– Success Response:
Code: 200
Content: {"message": "Book removed from wishlist."}
– Error Response:
Code: 404 Not Found
Content: {"error": "Wishlist not found."}
When: The specified wishlist does not exist
Code: 400 Bad Request
Content: {"error": "Book not in wishlist."}
When: The specified book is not present in the wishlist
Code: 400 Bad Request
Content: {"error": "Invalid input data."}
When: Missing or malformed parameters

Get All Authors
Retrieves a list of all authors available in the system.
- URL: /author
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"authors": [{...author1...}, {...author2...}]}
– Error Response:
Code: 500 Internal Server Error
Content: {"error": "Unable to retrieve authors."}
When: Server-side issue occurs during data retrieval

Get Author by ID
Retrieves detailed information about a specific author.
- URL: /author/{id}
- Method: GET
- URL Parameters: id (integer) — ID of the author
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"id": 1, "name": "Author Name", "bio": "..." }
– Error Response:
Code: 404 Not Found

Content: {"error": "Author not found."}
When: No author exists with the given ID
Code: 400 Bad Request
Content: {"error": "Invalid author ID."}
When: Provided author ID is not valid

Create New Author
Creates a new author entry in the system. Admin access required.
- URL: /admin/authors
- Method: POST
- URL Parameters: NONE
- Data Parameters: name (string), bio (string, optional)
– Success Response:
Code: 201
Content: {"message": "Author created successfully.", "authorId": 12}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Missing required fields."}
When: Required data like author name is not provided
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Delete Author
Deletes an author from the system by ID. Admin access required.
- URL: /admin/authors/{id}
- Method: DELETE
- URL Parameters: id (integer) — ID of the author to delete
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"message": "Author deleted successfully."}
– Error Response:
Code: 404 Not Found
Content: {"error": "Author not found."}
When: No author exists with the given ID
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Get All Publishers
Retrieves a list of all publishers available in the system.
- URL: /publisher
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"publishers": [{...publisher1...}, {...publisher2...}]}
– Error Response:
Code: 500 Internal Server Error

Content: {"error": "Unable to retrieve publishers."}
When: A server-side error occurs during data retrieval

Get Publisher by ID
Retrieves detailed information about a specific publisher.
- URL: /publisher/{id}
- Method: GET
- URL Parameters: id (integer) — ID of the publisher
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"id": 1, "name": "Publisher Name", "country": "..." }
– Error Response:
Code: 404 Not Found
Content: {"error": "Publisher not found."}
When: No publisher exists with the given ID
Code: 400 Bad Request
Content: {"error": "Invalid publisher ID."}
When: Provided publisher ID is invalid

Create Publisher
Creates a new publisher in the system. Admin access required.
- URL: /admin/publishers
- Method: POST
- URL Parameters: NONE
- Data Parameters: name (string), country (string)
– Success Response:
Code: 201
Content: {"message": "Publisher created successfully.", "publisherId": 5}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Missing required fields."}
When: Required data like name or country is not provided
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Delete Publisher
Deletes a publisher by ID. Admin access required.
- URL: /admin/publishesr/{id}
- Method: DELETE
- URL Parameters: id (integer) — ID of the publisher
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"message": "Publisher deleted successfully."}
– Error Response:
Code: 404 Not Found
Content: {"error": "Publisher not found."}
When: No publisher exists with the given ID
Code: 403 Forbidden

Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Get All Books
Retrieves a list of all books.
- URL: /book
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"books": [{...book1...}, {...book2...}]}
– Error Response:
Code: 500 Internal Server Error
Content: {"error": "Unable to retrieve books."}
When: Server error occurs during retrieval

Get Book by ID
Retrieves detailed information about a specific book.
- URL: /book/{id}
- Method: GET
- URL Parameters: id (integer) — ID of the book
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"id": 1, "title": "...", "author": "..."}
– Error Response:
Code: 404 Not Found
Content: {"error": "Book not found."}
When: Book with the given ID does not exist
Code: 400 Bad Request
Content: {"error": "Invalid book ID."}
When: Provided book ID is not valid

Add New Book
Adds a new book to the system. Admin access required.
- URL: /admin/books
- Method: POST
- URL Parameters: NONE
- Data Parameters: title (string), authorId (integer), publisherId (integer), categoryId (integer), price (decimal), description (string)
– Success Response:
Code: 201
Content: {"message": "Book added successfully.", "bookId": 101}
– Error Response:
Code: 400 Bad Request
Content: {"error": "Missing required fields."}
When: Required fields are missing
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Delete Book
Deletes a book from the system by ID. Admin access required.
- URL: /admin/books?id={id}
- Method: DELETE
- URL Parameters: id (integer) — ID of the book
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"message": "Book deleted successfully."}
– Error Response:
Code: 404 Not Found
Content: {"error": "Book not found."}
When: Book does not exist with the given ID
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without admin privileges

Get Top-Rated Books
Retrieves a list of top-rated books.
- URL: /books/top-rated
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"books": [{...book1...}, {...book2...}]}
– Error Response:
Code: 500 Internal Server Error
Content: {"error": "Unable to fetch top-rated books."}
When: Server error occurs

Get Books by Author
Fetches books written by a specific author.
- URL: /books/author?id={id}
- Method: GET
- URL Parameters: id (integer) — Author ID
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"books": [{...book1...}]}
– Error Response:
Code: 404 Not Found
Content: {"error": "Author not found or has no books."}
When: Author exists but has no books or doesn't exist

Get Books by Category
Fetches books within a specific category.
- URL: /books/category?id={id}
- Method: GET
- URL Parameters: id (integer) — Category ID
- Data Parameters: NONE

– Success Response:
Code: 200
Content: {"books": [{...book1...}]}
– Error Response:
Code: 404 Not Found
Content: {"error": "Category not found or has no books."}
When: Category is invalid or empty

Get Books by Publisher
Fetches books from a specific publisher.
- URL: /books/publisher?id={id}
- Method: GET
- URL Parameters: id (integer) — Publisher ID
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"books": [{...book1...}]}
– Error Response:
Code: 404 Not Found
Content: {"error": "Publisher not found or has no books."}
When: Publisher exists but has no books or doesn't exist

Get All Discounts
Retrieves all available discount offers. Manager access required.
- URL: /admin/discounts
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: {"discounts": [{...discount1...}, {...discount2...}]}
– Error Response:
Code: 403 Forbidden
Content: {"error": "Unauthorized access."}
When: Request is made without manager/admin role

Display Checkout Page
Displays the checkout view with cart contents and total.
- URL: /checkout
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: Checkout page with book list and totals.
– Error Response:
Code: 404 Not Found
Content: {"error": "Your cart is empty."}
Code: 401 Unauthorized
Redirect: /user/login

Submit Order from Checkout
Submits an order using cart data and payment info.
- URL: /checkout
- Method: POST
- URL Parameters: NONE
- Data Parameters: address (string), paymentMethod (string)
– Success Response:
Code: 302 Redirect
Redirect: /orders
– Error Response:
Code: 400 Bad Request
Content: {"error": "Please fill in all checkout fields."}
Code: 404 Not Found
Content: {"error": "No active cart to place an order."}
Code: 500 Internal Server Error
Content: {"error": "Failed to place your order."}
Code: 401 Unauthorized
Redirect: /user/login

Get User Orders
Displays a list of orders for the current user.
- URL: /orders
- Method: GET
- URL Parameters: NONE
- Data Parameters: NONE
– Success Response:
Code: 200
Content: Orders page with order list.
– Error Response:
Code: 500 Internal Server Error
Content: {"error": "Unable to load your orders."}
Code: 401 Unauthorized
Redirect: /user/login

Place New Order
Creates a new order based on the user's cart.
- URL: /orders
- Method: POST
- URL Parameters: NONE
- Data Parameters: NONE (uses session/cart data)
– Success Response:
Code: 302 Redirect
Redirect: /orders
– Error Response:
Code: 400 Bad Request
Content: {"error": "No cart available to place an order."}
Code: 500 Internal Server Error
Content: {"error": "Failed to insert order."}
Code: 401 Unauthorized
Redirect: /user/login

6 Group Members Contribution

Parisa Saeedidana

- Defined and updated the project objectives.
- Created the initial ER diagram and wrote the description of the ER schema.
- Developed DAO classes for: Publisher, Reviews, Author, Book, including authentication functionality.
- Implemented REST scripts for: Shopping cart operations and User managemnet.
- Developed servlets for: Book operations, shoppig cart, search and home page.
- Designed and implemented JSP pages for: Author, Book, Search results and Wishlist.
- Added additional test data to the database.
- Created the DAO class diagram and the User Registration sequence diagram for the report.
- Contributed to the overall progress and coordination of the project.

Amirreza Jolani mameghani

- Edited ER diagram and helped to write the ER schema description.
- Wrote the Docker for the project, and created a database folder and write sql code of the project.
- implemented the resource folder of the project.
- developed all the utilites of project ErrorCode and JWT Utils.
- developed the whole services folder(validation) for the project.
- Developed the servlets for admin, image, order.
- developed wishlist, User DAO classes for the project and try to modify the code of other DAO classes.
- Wrote mostly the queries for all DAO classes.
- Implemented REST for classes and modified other teammates' files to fix.
- Contributed to the overall progress and coordination of the project.

Ayoub Ben Yamoune

- Wrote the presentation section of the report and created all associated mockups.
- Developed the DAO classes for Order and User.
- Developed the servlets for User and Book.
- Designed and implemented the JSP pages for User and Registration.
- Contributed to the overall progress and coordination of the project.

Md Abu Raihan Mia:

- Wrote REST API Summary, REST Error Codes and REST API Details for the report.
- Designed and implemented the JSP page for the All Publishers.
- Fixed REST error codes and implemented the `ErrorCode` page.
- Created and designed the AuthorServlet sequence diagram for the report.
- Contributed to the overall progress and coordination of the project.

Adam Truty

- Developed DAO classes for: Discount, Category, and Cart.

- Implemented the servlets for: Cart and Author.
- Designed and implemented the JSP page for the Cart.
- Created and wrote the report sections for the sequence diagrams (CartServlet and BookServlet).
- Created and described the Class Diagram of the application for the report.
- Assisted in debugging and resolving issues during the development process.
- Contributed to the overall progress and coordination of the project.