<Bokify>

Project documentation

Student(s): Borzan Călina-Annemary

Group: 30434

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Contents

I Project specification

The project represents a client-server application designed to simplify the booking process for hotels, flights, and event tickets. Bookify will be a web-based application to ensure accessibility across devices, allowing users to seamlessly plan and manage their trips. The project aims to create an integrated platform that centralizes booking, payment, and travel information, making it easier for users to find and secure reservations.

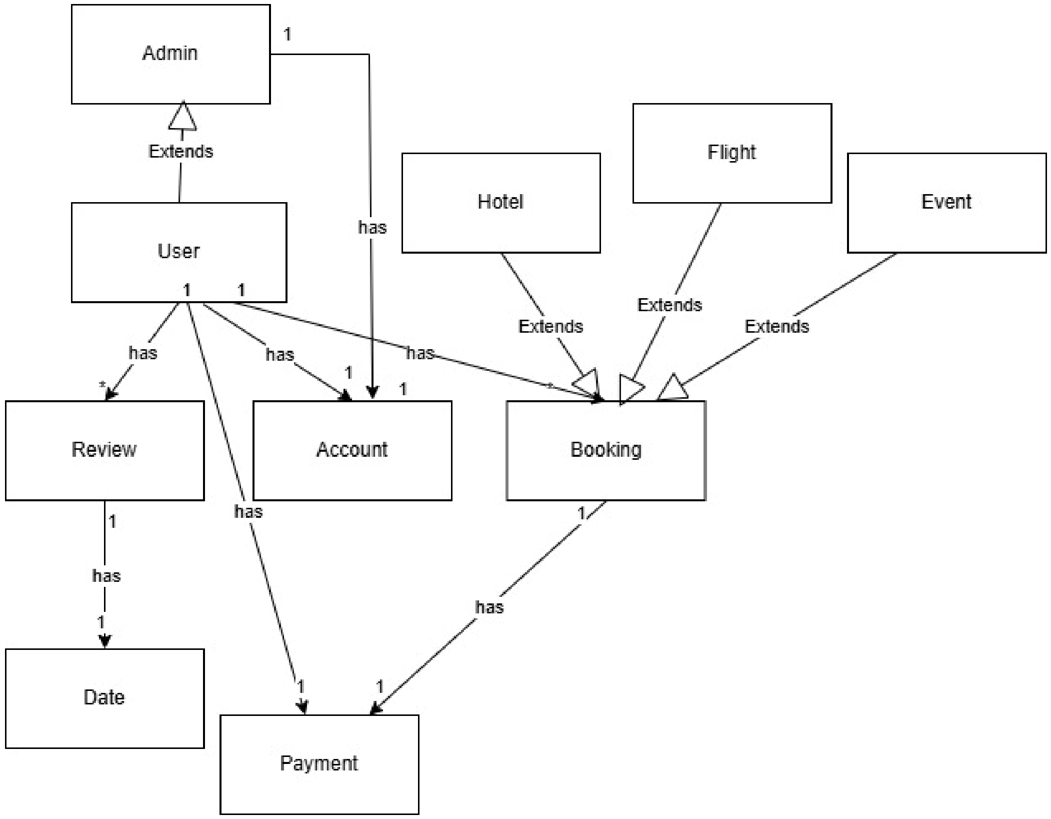
The system allows:

- **Users to log in or create an account** (as an admin or a regular user).
- **An admin to perform CRUD operations** on users, bookings, flights, hotels, and event listings.
- **A user to search and book** hotels, flights, and event tickets based on availability.
- **A user to manage their bookings**, view details, and cancel reservations if needed.
- **A user to leave reviews and rate services**, helping others make informed decisions.
- **Secure payment processing** for booking transactions using third-party payment gateways.

The application will have **two types of users**:

- **Regular users** – Can browse, book, review, and manage their reservations.
- **Admin users** – Can manage listings, monitor transactions, and oversee user activity.

1.1 Domain Model Diagram



II Use-Case model

The use-case model for Bookify defines two main roles: Admin and User. Admins extend User and have additional permissions to manage Users, Bookings, Payments, and Listings (hotels, flights, events). Users can create accounts, browse listings, make bookings, leave reviews, and manage payments. The model establishes one-to-many relationships, such as between Users and Bookings, and Admins with Bookings and Accounts. The goal was to centralize the booking, payment, and review processes while ensuring Admins can oversee and manage system operations effectively.

2.1 Users and stakeholders

- **Users**

The **Bookify** application will initially support two types of users with distinct roles:

- **Admin**: Admins have full control over the platform, allowing them to manage users, bookings, payments, and listings (hotels, flights, events). They can create, update, and delete entries across the application, ensuring the system remains up-to-date and well-maintained.
- **User**: Regular users can create an account, log in, browse available listings (hotels, flights, events), make bookings, view their booking history, leave reviews, and make payments. Their access is limited to interacting with the content rather than managing it.

- **Stakeholders**

The primary target audience for Bookify consists of:

- **Travelers:** People looking to plan their trips, make bookings, and explore available services (hotels, flights, events). This includes both first-time travelers and frequent flyers.
- **Admin Users**: Platform administrators who will manage and maintain listings, user accounts, and ensure smooth platform operation.
- **Businesses/Service Providers:** Hotels, airlines, and event organizers who provide services on the platform. They indirectly benefit from having their listings managed and maintained by Admins, ensuring the accuracy of their offers to potential customers.

2.2 Use-Case identification

Use Case 1: Create Account

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user navigates to the registration page of the application.
  2. The user provides necessary details such as name, email, password, and role (user/admin).
  3. The system validates the input data.
  4. Upon successful validation, the system creates a new user account.
  5. The user is logged in automatically and redirected to their dashboard.
- **Extension**:
  - **Invalid Input**: If the user provides invalid data (e.g., missing required fields or invalid email format), the system shows an error message prompting the user to correct the input.

Use Case 2: Log In

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user navigates to the login page.

2. The user enters their registered email and password.
3. The system validates the credentials.
4. Upon successful login, the user is redirected to their dashboard.
5. The user gains access to their bookings, reviews, and personal information.

- **Extension**:
  - **Invalid Credentials**: If the user enters incorrect login details, the system displays an error message prompting the user to try again.

Use Case 3: Make a Booking

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user logs into the system and browses available hotels, flights, or events.
  2. The user selects the preferred option (e.g., a hotel room, flight, or event).
  3. The user enters the required details, such as dates and payment information.
  4. The system validates the information and processes the booking.
  5. The system confirms the booking and provides a receipt or confirmation number.
  6. The user receives an email with booking details.
- **Extension**:
  - **Unavailable Option**: If the selected option is no longer available (e.g., the hotel is fully booked), the system informs the user and offers alternative choices.

Use Case 4: Leave a Review

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
    1. After completing a booking, the user logs into the system.
    2. The user navigates to the booking history page and selects a completed booking.
    3. The user writes a review, providing a rating and optional feedback about the experience.
    4. The system validates and submits the review.
    5. The review is published, and the user receives a notification confirming the submission.
- **Extension**:
    - **Invalid Review**: If the review contains inappropriate content (e.g., offensive language), the system flags the review for admin moderation before publishing.

Use Case 5: Admin Manage Listings

- **Level:** Subfunction
- **Main actor:** Admin
- **Main success scenario:**
    1. The admin logs into the application and navigates to the admin panel.
    2. The admin selects the option to manage listings (hotels, flights, events).
    3. The admin adds, updates, or deletes listings as needed.
    4. The system validates the changes and updates the platform's content accordingly.
    5. The updated listings are immediately visible to users.

- **Extension:**
  - **Invalid Listing:** If the admin enters incorrect or incomplete data (e.g., missing details about a hotel or flight), the system prompts for correction before saving the changes.

Use Case 6: Search for Listings (Hotels, Flights, Events)

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user logs into the system.
  2. The user enters search criteria (e.g., destination, date, type of service).
  3. The system filters the listings and displays matching options for hotels, flights, or events.
  4. The user can refine the search by applying filters (price range, rating, etc.).
  5. The user selects a listing for more details and proceeds with booking.
- **Extension**:
  - **No Matches**: If no listings match the search criteria, the system informs the user and suggests alternative search options.

Use Case 7: Manage Booking (View, Modify, Cancel)

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user logs into the system and navigates to their booking history.

2. The user selects a booking to view the details.
3. The user can modify the booking (e.g., change dates, cancel booking).
4. If modifying, the system updates the booking and sends a confirmation.
5. If canceling, the system asks for confirmation and processes the cancellation.
6. The user receives an email confirmation of the change or cancellation.

- **Extension**:
  - **Late Cancellation**: If the user tries to cancel a booking close to the date of the service, the system may impose a fee and display a warning before proceeding.

Use Case 8: Make Payment

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user selects a booking to finalize.
  2. The system redirects the user to the payment page.
  3. The user enters payment details (credit card, PayPal, etc.).
  4. The system processes the payment through a third-party gateway.
  5. If the payment is successful, the system confirms the booking and sends a receipt.
- **Extension**:
  - **Payment Failure**: If the payment fails (e.g., insufficient funds), the system prompts the user to try again or use a different payment method.

Use Case 9: Admin View User Activity

- **Level**: Subfunction
- **Main actor**: Admin
- **Main success scenario**:
    1. The admin logs into the system and navigates to the user management section.
    2. The admin selects a user from the list of registered users.
    3. The system displays the user's activity, including bookings, payments, and reviews.
    4. The admin can view detailed booking history, including canceled or modified bookings.
- **Extension**:
    - **Suspicious Activity**: If the admin notices suspicious behavior (e.g., frequent cancellations), they may flag the account for further review.

Use Case 10: Admin Manage User Accounts

- **Level**: Subfunction
- **Main actor**: Admin
- **Main success scenario**:
    1. The admin logs into the admin panel.
    2. The admin selects the option to manage user accounts.
    3. The admin can add, remove, or update user accounts.
    4. The admin can change the role of a user (e.g., from regular user to admin).
    5. The system confirms the changes and updates the user database accordingly.
- **Extension**:

- **Role Change**: If the admin changes a user's role, the system checks if the user has permission to perform the tasks allowed by the new role.

Use Case 11: Admin Manage Listings (Hotels, Flights, Events)

- **Level**: Subfunction
- **Main actor**: Admin
- **Main success scenario**:
    1. The admin logs into the system and navigates to the "Manage Listings" section.
    2. The admin can add, edit, or delete listings for hotels, flights, or events.
    3. For adding or editing, the admin enters details like price, location, dates, and other required fields.
    4. The system validates the information.
    5. The admin confirms and the system updates the listings available to users.
- **Extension**:
    - **Missing Information**: If the admin fails to provide required information (e.g., missing hotel room details), the system prompts the admin to complete the missing fields before saving the listing.

Use Case 13: Admin Monitor Payments

- **Level**: Subfunction
- **Main actor**: Admin
- **Main success scenario**:
    1. The admin logs into the admin panel and navigates to the payments section.

2. The system displays a list of all payments made by users for bookings.
3. The admin can filter and view the payment status, such as successful, pending, or failed transactions.
4. The admin can view details of each payment and take necessary actions if needed (e.g., refund requests).

- **Extension**:
  - **Refund Request**: If a payment is marked as "failed" and the user requests a refund, the admin can process the refund through the payment system.

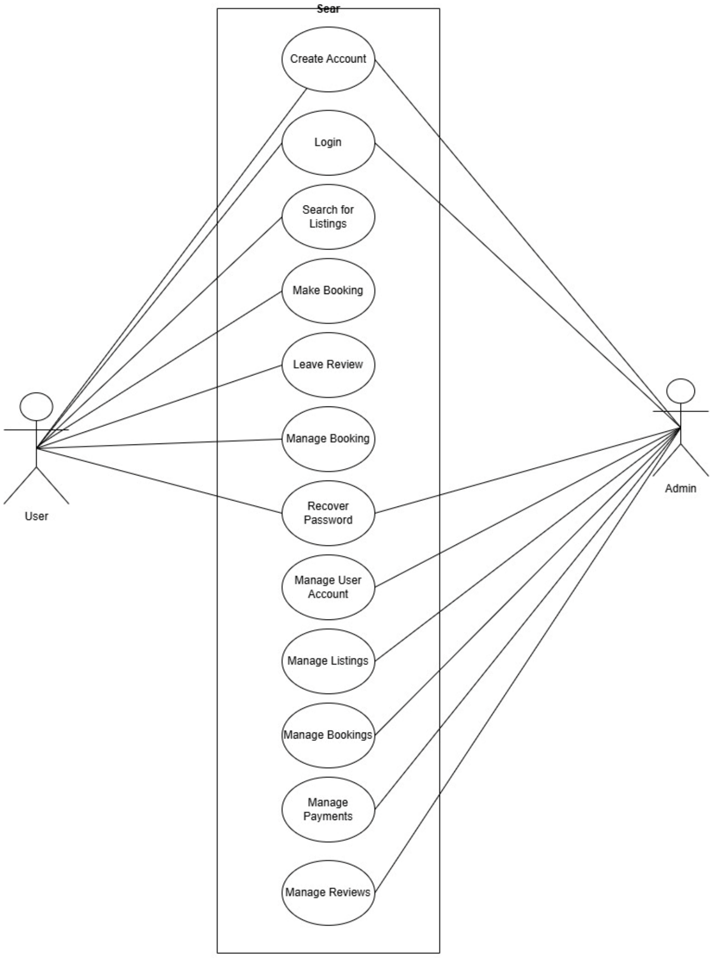
Use Case 14: Password Recovery

- **Level**: User-Goal
- **Main actor**: User
- **Main success scenario**:
  1. The user clicks on the "Forgot Password" link on the login page.
  2. The system asks the user to enter their registered email address.
  3. The system sends a password reset link to the email.
  4. The user clicks the link, resets their password, and is redirected to the login page with a success message.
- **Extension**:
  - **Invalid Email**: If the user enters an email that is not registered, the system displays an error message.

Use Case 15: Admin Manage Bookings

- **Level**: Subfunction
- **Main actor**: Admin
- **Main success scenario**:
  1. The admin logs into the admin panel and navigates to the bookings section.
  2. The system displays a list of all bookings made by users.
  3. The admin can view, modify, or cancel bookings on behalf of users.
  4. The system updates the booking status and notifies the user of any changes.
- **Extension**:
  - **Booking Modification**: If the admin changes the booking details (e.g., dates, services), the system revalidates the new details and updates the user's booking.

2.3 UML Use-Case diagram



III Architectural design

The architectural design of my application represents the structure, key components and interactions of my application. I chose to implement my project with layered architecture because it ensures a more organized code, enhancing maintainability and code reusability. It will also help me debug the errors with more ease and also test little parts of my program to make sure we have total functionality.

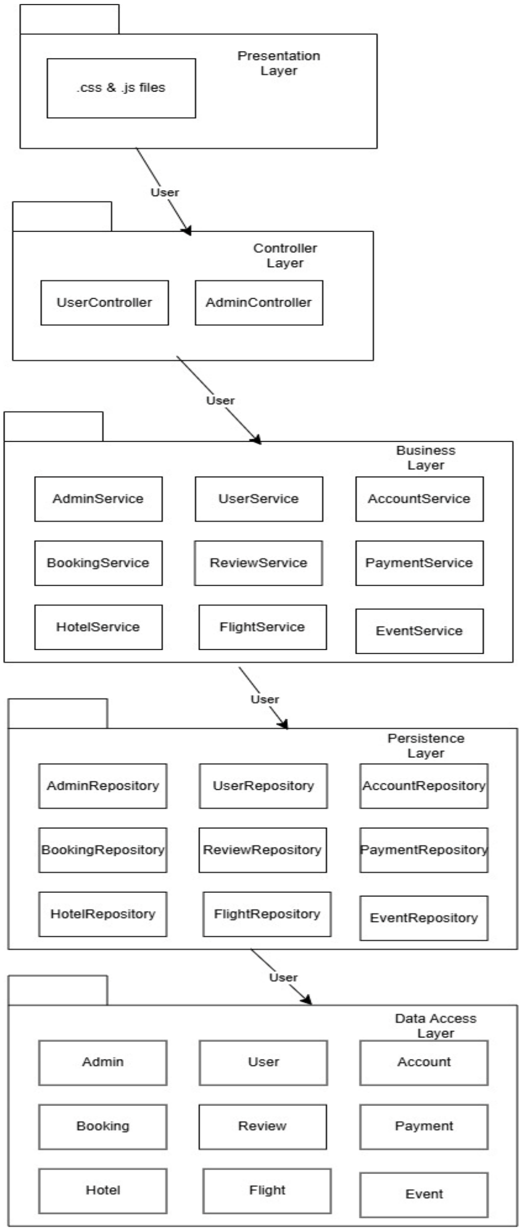
3.1 Conceptual architecture

Bookify is a web-based client-server application designed for booking hotels, flights, and event tickets. The system follows a layered architectural pattern to ensure modularity and scalability, and it also will use a database (MySql) in which all the data will be stored. The architecture consists of the following layer components:

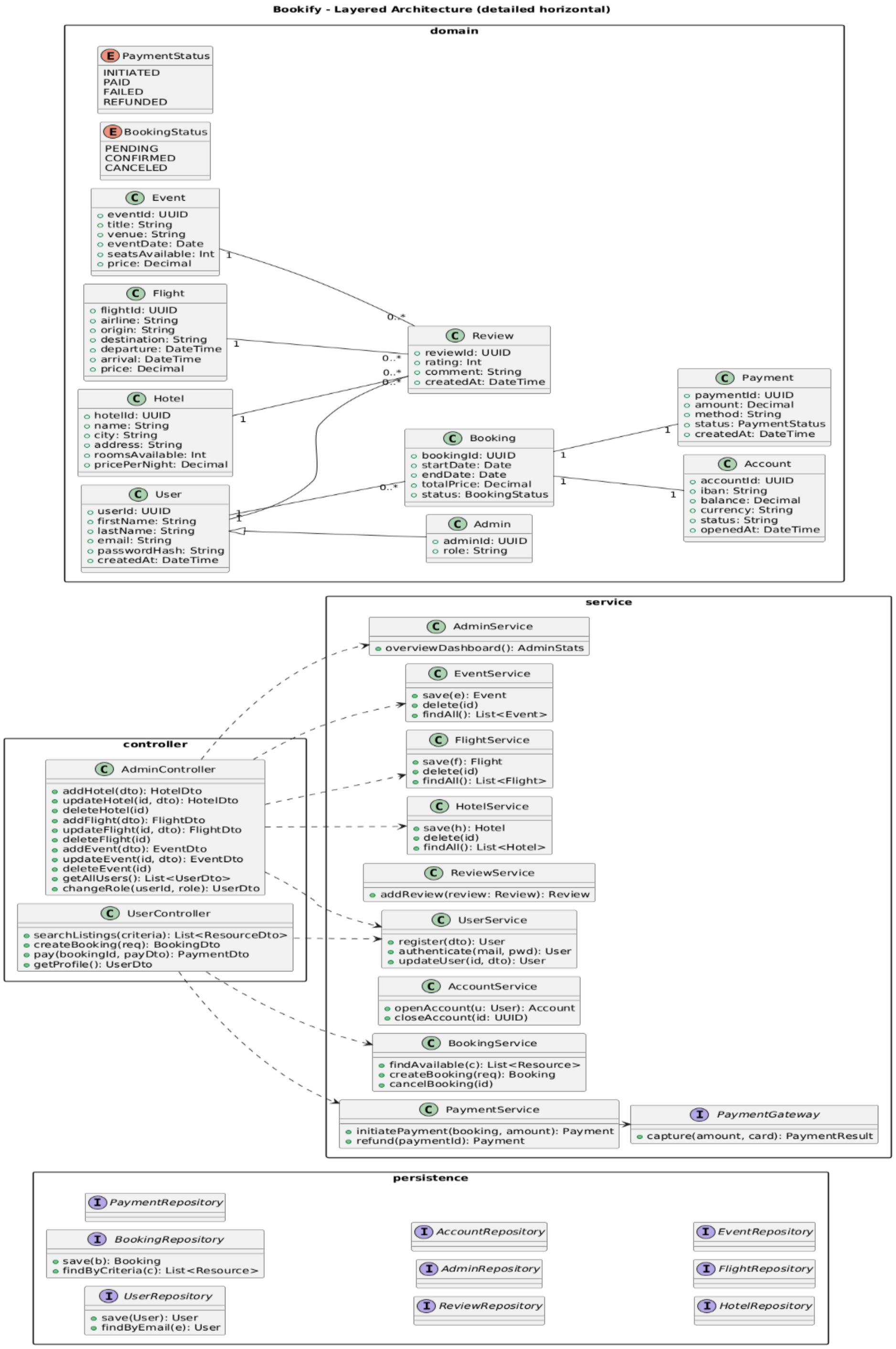
- **Presentation Layer**: the frontend is built with modern technologies, declaring components necessary to have an intuitive user interface
- **Controller Layer:** this represents the connection between UI and business logic, handling requests and responses
- **Business Layer:** it contains the core logic of the application, taking care of validation, booking management and payment processing
- **Persistence Layer:** this layer is responsible for the interaction of the application with the database, making the operations in the form of queries
- **Database Layer:** this layer consists of the classes we work with, which will also be the tables existing in the database, where we store user accounts, booking, payments and reviews

The system also integrates third-party payment gateways for secure transactions and utilizes authentication mechanisms to protect user data.

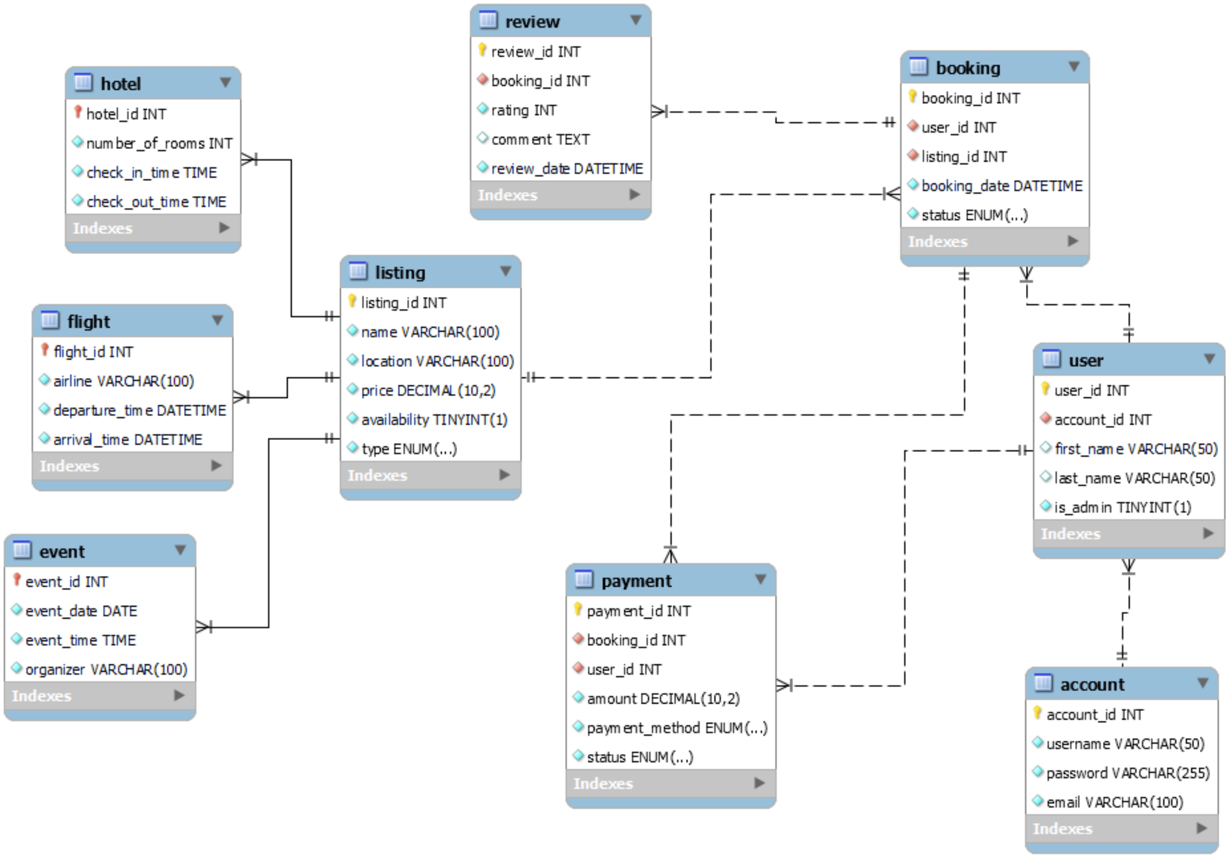
3.2 Package diagram



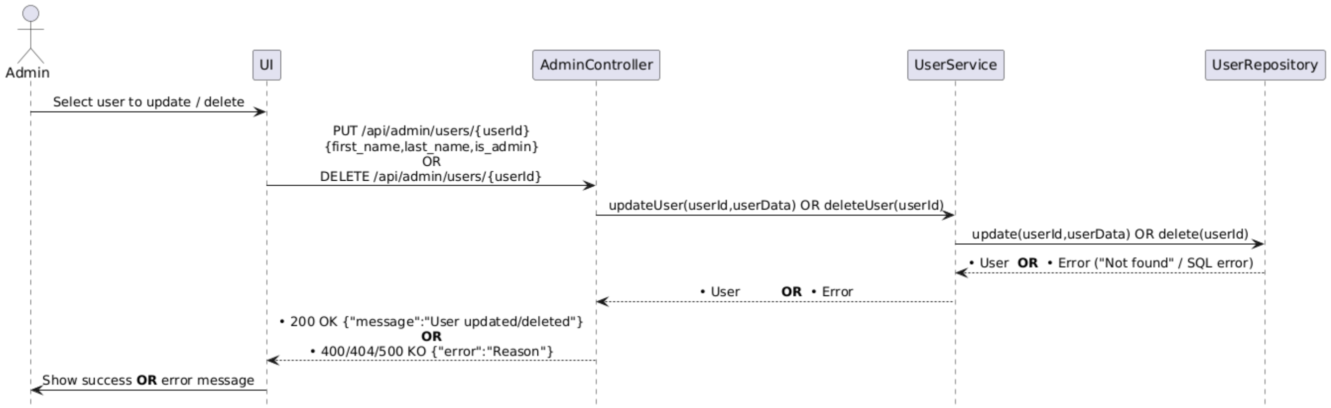
3.3 Class diagram



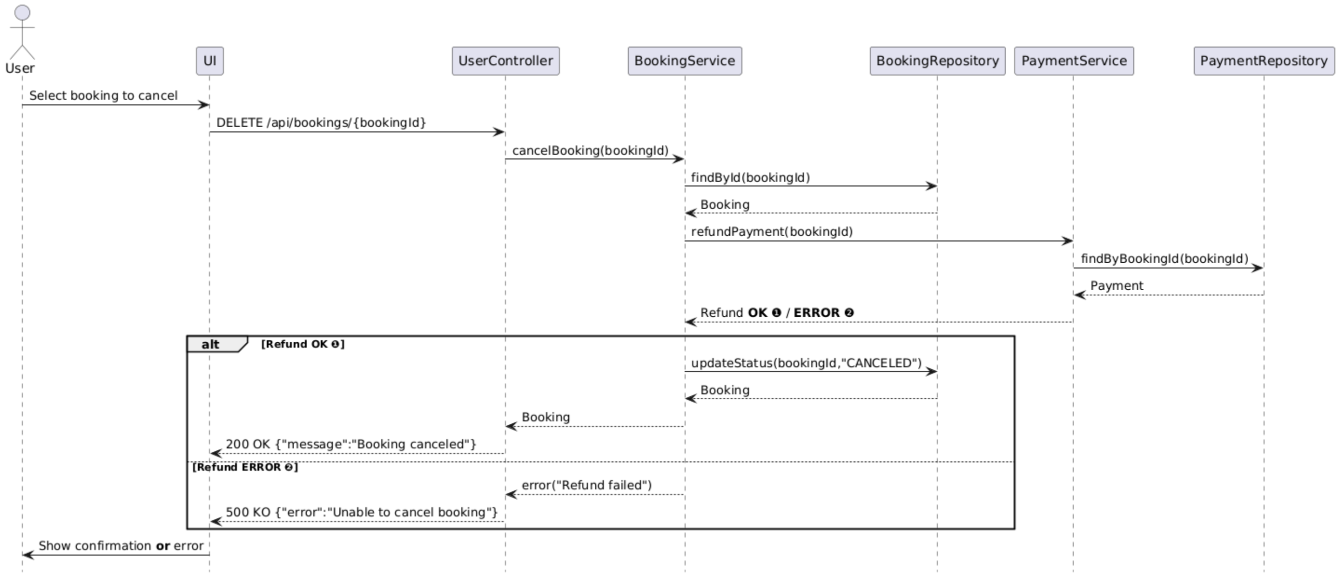
3.4 Database (E-R/Data model) diagram



3.5 Sequence diagram

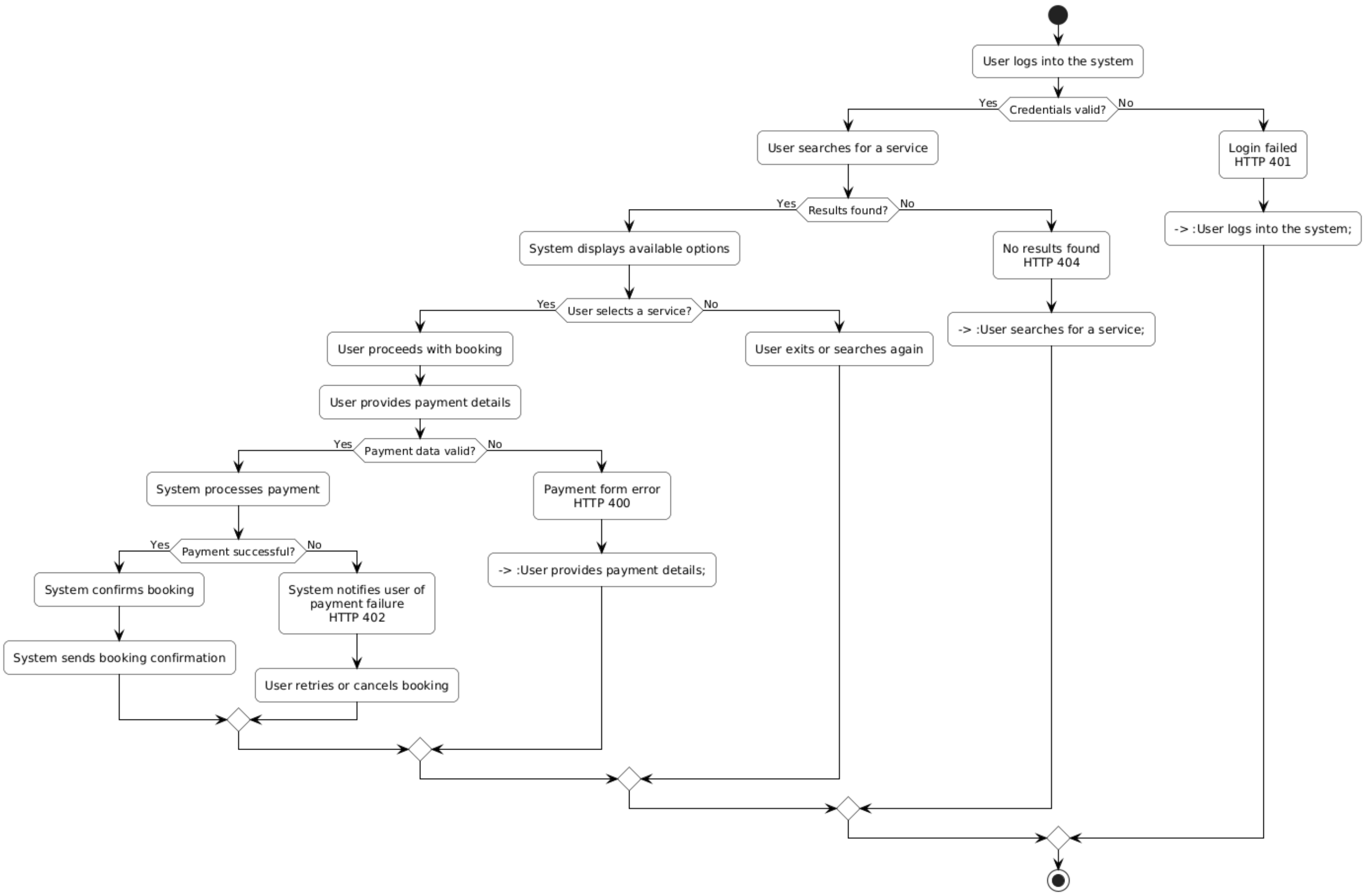

Admin
UI
AdminController
UserService
UserRepository
Select user to update / delete
PUT /api/admin/users/{userId}
{first_name,last_name,is_admin}
OR
DELETE /api/admin/users/{userId}
updateUser(userId,userData) OR deleteUser(userId)
update(userId,userData) OR delete(userId)
• User OR • Error ("Not found" / SQL error)
• User OR • Error
• 200 OK {"message":"User updated/deleted"}
OR
• 400/404/500 KO {"error":"Reason"}
Show success OR error message

This diagram illustrates how an **Admin** updates *or* deletes a user account and how the system reacts when the operation fails.

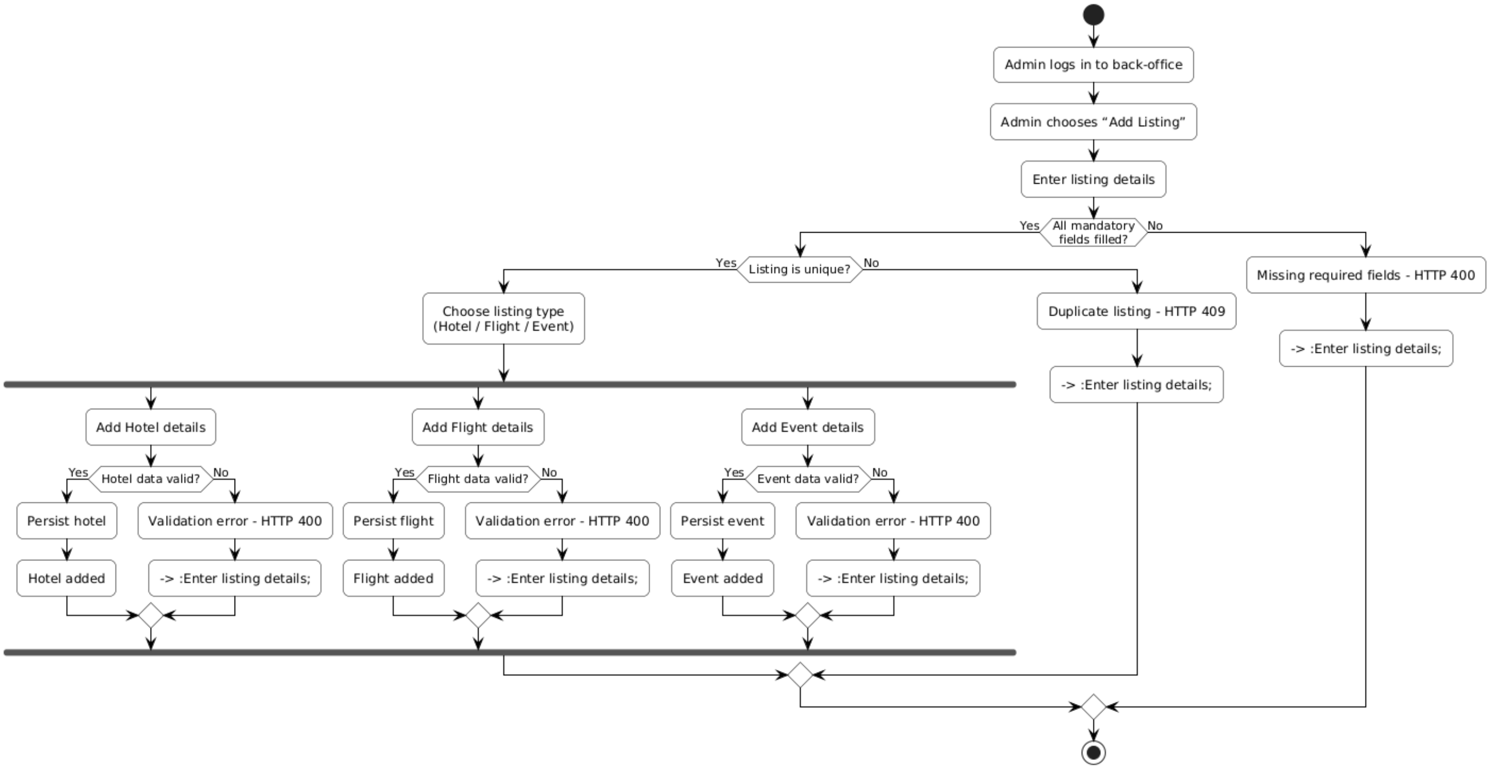


The sequence describes how a **User** cancels an existing booking:

3.6 Activity diagram



User logs into the system
Credentials valid?
Yes
No
User searches for a service
Login failed
HTTP 401
Results found?
Yes
No
-> :User logs into the system;
System displays available options
No results found
HTTP 404
User selects a service?
Yes
No
-> :User searches for a service;
User proceeds with booking
User exits or searches again
User provides payment details
Payment data valid?
Yes
No
System processes payment
Payment form error
HTTP 400
Payment successful?
Yes
No
-> :User provides payment details;
System confirms booking
System notifies user of
payment failure
HTTP 402
System sends booking confirmation
User retries or cancels booking



Admin logs in to back-office
Admin chooses "Add Listing"
Enter listing details
Yes
All mandatory fields filled?
No
Yes
Listing is unique?
No
Missing required fields - HTTP 400
Choose listing type (Hotel / Flight / Event)
Duplicate listing - HTTP 409
-> :Enter listing details;
-> :Enter listing details;
Add Hotel details
Add Flight details
Add Event details
Yes
Hotel data valid?
No
Yes
Flight data valid?
No
Yes
Event data valid?
No
Persist hotel
Validation error - HTTP 400
Persist flight
Validation error - HTTP 400
Persist event
Validation error - HTTP 400
Hotel added
-> :Enter listing details;
Flight added
-> :Enter listing details;
Event added
-> :Enter listing details;

IV Supplementary specifications

The **Supplementary Specifications** section captures the non-functional requirements and design constraints for the **Bookify** application.

4.1 Non-functional requirements

1. Availability

- **Requirement**: The application should be available **24/7**, with minimal downtime for maintenance.
- **Details**:
  - The system should have a **99.9% uptime** guarantee.

2. Performance

- **Requirement**: The application should handle a **large number of concurrent users** and respond to requests within **2 seconds**.
- **Details**:
  - The system should support **at least 10,000 concurrent users** without performance degradation.
  - Database queries and API responses should be optimized to ensure fast response times.

3. Security

- **Requirement**: User data, especially payment information, must be securely stored and transmitted using encryption (e.g., HTTPS, SSL). Also the account details should be secure, that is why only the admin can see and verify the personal information of any user.
- **Details**:
  - All sensitive data (e.g., passwords, payment information) must be encrypted **both in transit and at rest**.
  - The application must use **HTTPS** with up-to-date SSL/TLS certificates to secure communication between the client and server.

4. Usability

- **Requirement**: The application should have an **intuitive and user-friendly interface**, accessible to users with varying levels of technical expertise.
- **Details**:
  - The user interface (UI) should follow **modern design principles** (e.g., clean layout, consistent navigation, responsive design).
  - The application should provide **clear feedback** for user actions (e.g., success messages, error messages, loading indicators).

5. Accessibility

- **Requirement**: The application should be **accessible across devices** (desktop, mobile, tablet) and browsers.
- **Details**:
  - The application should be **fully responsive**, ensuring a seamless experience on devices of all screen sizes.
  - The application should be fully responsive when there is a good network connection.

6. Reliability

- **Requirement**: The application should be **reliable** and provide consistent performance under normal and peak loads.
- **Details**:
  - The system should have **automatic retry mechanisms** for failed requests (e.g., payment processing, API calls).
  - Database transactions should be **atomic** to ensure data consistency (e.g., if a booking fails, the payment should be rolled back).

4.2 Design constraints

- The application will be developed using **Java with Spring Boot** for the backend and **React.js** for the frontend.
- **MySQL** will be used as the primary database.
- **Hibernate ORM** will be used to manage database interactions and simplify object-relational mapping.
- The frontend and backend will communicate via **RESTful APIs**.
- The system must integrate with **third-party payment gateways** to facilitate transactions.

- **OAuth-based authentication** will be implemented to ensure secure access control.

V Testing

5.1 Testing methods/frameworks

The Bookify application was tested using JUnit 5, a robust testing framework for Java, combined with Mockito for mocking dependencies. The testing strategy ensured the reliability, correctness, and robustness of core functionalities, including booking management, user authentication, event handling, and payment processing. Tests covered both success scenarios and edge cases to validate the application's behavior under various conditions.

- **Unit Testing**: Focused on individual components, such as services (BookingService, AuthService, EventService, PaymentService), to verify their logic in isolation. Mocks were used to simulate dependencies like repositories and external services, ensuring methods performed as expected.

- **Integration Testing**: Validated interactions between components, such as service-to-repository communication and third-party integrations (e.g., payment gateways and email services), to ensure seamless data flow and system consistency.

- **Edge Case Testing**: Tested boundary conditions, such as invalid inputs, non-existent IDs, and unavailable resources (e.g., fully booked hotels), to ensure the application handles errors gracefully and throws appropriate exceptions.

- **Security Testing**: Focused on authentication and authorization, verifying secure user login, role-based access control, and password reset functionality to protect user data and ensure secure access.

5.2 Future improvements

To enhance the Bookify application, several new features are proposed to improve user experience and functionality:

- **Chat Functionality**: Introduce a real-time chat feature allowing users to communicate directly with accommodation providers (e.g., hotel staff or hosts). This will facilitate inquiries about bookings, special requests, or clarifications, improving customer service and satisfaction.
- **Enhanced Hotel and Room Details**: Expand hotel and room listings to include detailed descriptions, amenities (e.g., Wi-Fi, breakfast options), and additional attributes like room size or view type. This will help users make more informed booking decisions.
- **Photo Gallery**: Add support for uploading and displaying high-quality photos for hotels, rooms, and event venues. A gallery view will allow users to visually explore accommodations and venues, enhancing trust and engagement.
- **Discount and Promotion System**: Implement a discount system where admins can create promotional offers or seasonal discounts for hotels, flights, or events. Users will see available discounts during booking, and the system will automatically apply reductions to eligible transactions, improving affordability and competitiveness.

VI API Documentation

The following section details the RESTful API endpoints for the Bookify application, implemented using Spring Boot and React. Each endpoint includes the HTTP method, request body (in JSON format where applicable), and possible response codes to indicate success or failure.

POST API Calls

1. User Login

• http://localhost:8080/bookify/api/auth/login

Request:

Body(JSON):

~~~
{
  "username": "calinaborzan",
  "password": "Calina2003"
}
```
~~~

Response:

- 200 OK – Login successful, returns JWT token

- 400 Bad Request – Invalid credentials

2. User Signup

• http://localhost:8080/bookify/api/auth/signup

Request:

Body(JSON):

~~~
{
~~~

```
  "username": "calinaborzam",
  "email": "calina.borzan18@yahoo.com",
  "password": "Calina2003",
  "firstName": "Calina",
  "lastName": "Borzan",
  "age": 22,
  "role": "ROLE_ADMIN"
}
```

__Response:__

- 201 Created – Account successfully created

- 400 Bad Request – Account cannot be created (e.g., email or username already exists)

3. Forgot Password

• http://localhost:8080/bookify/api/auth/forgot-password

__Request:__

Body(JSON):

```
{
  "email": "calina.borzan18@yahoo.com"
}
```

\`\`\`

Response:

- 200 OK – Password reset email sent

- 400 Bad Request – Email not found

4. Reset Password

• http://localhost:8080/bookify/api/auth/reset-password

Request:

Body(JSON):

```
{
  "email": "calina.borzan18@yahoo.com",
  "token": "reset-token-123",
  "newPassword": "Calina1234"
}
```

Response:

- 200 OK – Password reset successfully

- 400 Bad Request – Invalid or expired token

5. Create Booking

• http://localhost:8080/bookify/api/bookings

Request:

Body(JSON):

```
{
  "listingId": 1,
  "checkIn": "2025-06-01",
  "checkOut": "2025-06-02",
  "numGuests": 2,
  "payNow": false
}
```

Response:

- 201 Created – Booking successfully created

- 400 Bad Request – Booking cannot be created (e.g., invalid input or unavailable listing)

6. Create Hotel Listing

• http://localhost:8080/bookify/api/listings/hotels

Request:

Body(JSON):

~~~
{
  "title": "Grand Hotel",
  "description": "Luxury hotel in city center",
  "price": 150.00,
  "country": "Romania",
  "city": "Cluj-Napoca",
  "totalRooms": 100
}
```
~~~

Response:

- 201 Created – Hotel successfully added

- 400 Bad Request – Hotel cannot be added (e.g., invalid input)

7. Create Flight Listing

• http://localhost:8080/bookify/api/listings/flights

Request:

Body(JSON):

~~~
{
  "title": "Flight to Paris",
  "description": "Direct flight to Paris",
  "price": 200.00,
~~~

```
  "departureCity": "Cluj-Napoca",
  "destinationCity": "Paris",
  "flightDate": "2025-06-01T10:00:00"
}
```

__Response:__

- 201 Created – Flight successfully added

- 400 Bad Request – Flight cannot be added (e.g., invalid input)

8. Create Event Listing

- http://localhost:8080/bookify/api/listings/events

__Request:__

Body(JSON):

```
{
  "title": "Concert in Cluj",
  "description": "Live music event",
  "price": 50.00,
  "country": "Romania",
  "venue": "Main Hall",
```

~~~
  "eventDate": "2025-12-31T20:00:00",

  "ticketCapacity": 500

}

```
~~~

__Response:__

- 201 Created – Event successfully added

- 400 Bad Request – Event cannot be added (e.g., invalid input)

9. Create Payment

- http://localhost:8080/bookify/api/payments

__Request:__

Body(JSON):

~~~
{

  "bookingId": 100,

  "amount": 250.00,

  "transactionId": "tx-789"

}

```
~~~

__Response:__

- 201 Created – Payment successfully created

- 400 Bad Request – Payment cannot be created (e.g., booking not found)

10. Create Review

- http://localhost:8080/bookify/api/reviews

__Request:__

Body(JSON):

~~~
{
  "bookingId": 100,
  "listingId": 1,
  "rating": 5,
  "comment": "Great experience!"
}
```
~~~

__Response:__

- 201 Created – Review successfully added

- 400 Bad Request – Review cannot be added (e.g., user already reviewed this booking)

11. Moderate Review

- http://localhost:8080/bookify/api/reviews/{id}/moderate

__Request:__

Body(JSON):

~~~
{
~~~

"approved": true,

"remarks": "Review meets guidelines"

}

```

*Response:*

- 200 OK – Review successfully moderated

- 400 Bad Request – Review not found

## **GET API Calls**

### **1. Get Current User**

### • http://localhost:8080/bookify/api/users/me

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – Current user details retrieved

- 401 Unauthorized – No authenticated user

2. Get All Users

- http://localhost:8080/bookify/api/users

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – All users retrieved

- 403 Forbidden – Unauthorized access (non-admin user)

## **3. Get User by ID**

- http://localhost:8080/bookify/api/users/{id}

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – User found

- 404 Not Found – User ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

4. Get Booking by ID

• http://localhost:8080/bookify/api/bookings/{id}

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Booking found

- 404 Not Found – Booking ID does not exist

## **5. Get Bookings by User ID**

• http://localhost:8080/bookify/api/bookings/user/{userId}

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – Bookings for user retrieved

- 404 Not Found – User ID does not exist

6. Get Current User’s Bookings

• http://localhost:8080/bookify/api/bookings/me

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Current user’s bookings retrieved

- 401 Unauthorized – No authenticated user

## **7. Get Bookings for Current User’s Listings**

## • http://localhost:8080/bookify/api/bookings/admin/bookings

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – Bookings for current user’s listings retrieved

- 403 Forbidden – Unauthorized access (non-admin user)

8. Get Hotel by ID

• http://localhost:8080/bookify/api/listings/hotels/{id}

Request:

Body(JSON):

{}

```

***Response:***

- 200 OK – Hotel found

- 404 Not Found – Hotel ID does not exist

## **9. Get All Hotels**

• http://localhost:8080/bookify/api/listings/hotels

***Request:***

Body(JSON):

{}

```

Response:

- 200 OK – All hotels retrieved

- 404 Not Found – No hotels found

10. Get Flight by ID

• http://localhost:8080/bookify/api/listings/flights/{id}

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Flight found

- 404 Not Found – Flight ID does not exist

## **11. Get All Flights**

### • http://localhost:8080/bookify/api/listings/flights

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – All flights retrieved

- 404 Not Found – No flights found

12. Get Event by ID

• http://localhost:8080/bookify/api/listings/events/{id}

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Event found

- 404 Not Found – Event ID does not exist

## **13. Get All Events**

## • http://localhost:8080/bookify/api/listings/events

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – All events retrieved

- 404 Not Found – No events found

14. Get Payment by ID

• http://localhost:8080/bookify/api/payments/{id}

Request:

Body(JSON)

{}

```

*Response:*

- 200 OK – Payment found

- 404 Not Found – Payment ID does not exist

## **15. Get Payments by Booking ID**

### • http://localhost:8080/bookify/api/payments/booking/{bookingId}

*Request:*

Body(JSON)

{}

```

Response:

- 200 OK – Payments for booking retrieved

- 404 Not Found – Booking ID does not exist

16. Download Receipt for Booking

- http://localhost:8080/bookify/api/payments/booking/{bookingId}/receipt

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Receipt downloaded in specified format (e.g., JSON)

- 400 Bad Request – Unknown format specified

- 404 Not Found – No payment found for booking

## **17. Get Review by ID**

- http://localhost:8080/bookify/api/reviews/{id}

*Request:*

Body(JSON):

{}

```

Response:

- 200 OK – Review found

- 404 Not Found – Review ID does not exist

18. Get Reviews by Booking ID

• http://localhost:8080/bookify/api/reviews/booking/{bookingId}

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Reviews for booking retrieved

- 404 Not Found – Booking ID does not exist

## **19. Get Reviews by Listing ID**

## • http://localhost:8080/bookify/api/reviews/listing/{listingId}

*Request:*

Body(JSON)

{}

```

Response:

- 200 OK – Reviews for listing retrieved

- 404 Not Found – Listing ID does not exist

20. Get Pending Reviews

• http://localhost:8080/bookify/api/reviews/pending

Request:

Body(JSON):

{}

```

*Response:*

- 200 OK – Pending reviews retrieved

- 404 Not Found – No pending reviews

- 403 Forbidden – Unauthorized access (non-admin user)

# **PUT API Calls**

## **1. Update Current User**

## • http://localhost:8080/bookify/api/users/me

*Request:*

Body(JSON):

{

"username": "calina1234",
```

~~~
  "email": "calina.borzan18@yahoo.com",

  "firstName": "Calina",

  "lastName": "Borzan",

  "age": 22,

  "role": "ROLE_USER"

}

```
~~~

Response:

- 200 OK – User details updated

- 400 Bad Request – Invalid input

- 401 Unauthorized – No authenticated user

2. Update User by ID

• http://localhost:8080/bookify/api/users/{id}

Request:

Body(JSON):

~~~
{

username": "calina1234",

  "email": "calina.borzan18@yahoo.com",

  "firstName": "Calina",

  "lastName": "Borzan",
~~~

~~~
  "age": 22,
  "role": "ROLE_ADMIN"
}
```
~~~

Response:

- 200 OK – User details updated

- 404 Not Found – User ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

3. Update Hotel

- http://localhost:8080/bookify/api/listings/hotels/{id}

Request:

Body(JSON):

```
{
  "title": "Grand Hotel Updated",
  "description": "Updated luxury hotel description",
  "price": 175.00,
  "country": "Romania",
  "city": "Cluj-Napoca",
  "totalRooms": 120
```

~~~
}
```
~~~

Response:

- 200 OK – Hotel successfully updated

- 404 Not Found – Hotel ID does not exist

- 400 Bad Request – Invalid input

- 403 Forbidden – Unauthorized access (non-admin user)

4. Update Flight

• http://localhost:8080/bookify/api/listings/flights/{id}

Request:

Body(JSON):

~~~
{
 "title": "Flight to Paris Updated",
 "description": "Updated direct flight details",
 "price": 220.00,
 "departureCity": "Cluj-Napoca",
 "destinationCity": "Paris",
 "flightDate": "2025-06-02T10:00:00"
}
```
~~~

Response:

- 200 OK – Flight successfully updated

- 404 Not Found – Flight ID does not exist

- 400 Bad Request – Invalid input

- 403 Forbidden – Unauthorized access (non-admin user)

5. Update Event

• http://localhost:8080/bookify/api/listings/events/{id}

Request:

Body(JSON):

~~~
{
  "title": "Concert in Cluj Updated",
  "description": "Updated music event details",
  "price": 60.00,
  "country": "Romania",
  "venue": "Main Hall",
  "eventDate": "2025-12-31T21:00:00",
  "ticketCapacity": 600
}
```
~~~

Response:

- 200 OK – Event successfully updated

- 404 Not Found – Event ID does not exist

- 400 Bad Request – Invalid input

- 403 Forbidden – Unauthorized access (non-admin user)

6. Update Booking

• http://localhost:8080/bookify/api/bookings/admin/bookings/{id}

Request:

Body(JSON):

{

  "status": "CANCELLED"

}

```

*Response:*

- 200 OK – Booking successfully updated

- 404 Not Found – Booking ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)
```

DELETE API Calls

1. Delete Current User

• http://localhost:8080/bookify/api/users/me

Request:

Request:

Body(JSON):

{}

```

Response:

- 204 No Content – User successfully deleted

- 401 Unauthorized – No authenticated user

## **2. Delete User by ID**

• http://localhost:8080/bookify/api/users/{id}

*Request:*

Body(JSON):

{}

```

Response:

- 204 No Content – User successfully deleted

- 404 Not Found – User ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

3. Delete Booking

- http://localhost:8080/bookify/api/bookings/admin/bookings/{id}

Request:

```json
{}
```

Response:

- 204 No Content – Booking successfully deleted

- 404 Not Found – Booking ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

4. Delete Hotel

- http://localhost:8080/bookify/api/listings/hotels/{id}

Request:

Body(JSON):

{}

```

*Response:*
```

- 204 No Content – Hotel successfully deleted

- 404 Not Found – Hotel ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

5. Delete Flight

• http://localhost:8080/bookify/api/listings/flights/{id}

Request:

Body(JSON):

{}

```

*Response:*

- 204 No Content – Flight successfully deleted

- 404 Not Found – Flight ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

## **6. Delete Event**

• http://localhost:8080/bookify/api/listings/events/{id}

*Request:*

Body(JSON):

{}

```

Response:

- 204 No Content – Event successfully deleted

- 404 Not Found – Event ID does not exist

- 403 Forbidden – Unauthorized access (non-admin user)

VII Bibliography

1. https://www.geeksforgeeks.org/rest-api-introduction/
2. https://www.geeksforgeeks.org/spring-boot-integrating-hibernate-and-jpa/
3. https://www.w3schools.com/react/
4. https://www.geeksforgeeks.org/json-web-token-jwt/