

Domain – Transport

Online Resort Booking Portal

Overview:

The Online Resort Booking Portal serves as a digital gateway for users seeking unique and immersive accommodations at resorts worldwide. This web-based service provides a user-friendly environment for discovering, selecting, and reserving resort accommodations tailored to individual preferences. Users can explore a diverse catalog of resort listings, complete with detailed descriptions, images, and amenities, empowering them to make informed choices.

Users of the System:

Admin

Roles:

- Admin registration and login
- Add Resort, Edit Resort Details, Delete Resort Details and View All Resorts
- Admin can view bookings and change status of booking,

Customer

Roles:

- Customer registration and login
- Customer can view all Resorts
- Add booking and delete booking can also be done.

Application Flow:

Customer side:

The application flow for the portal begins with customer registration, where prospective customers create accounts by providing personal information. Upon logging in, customers access the customer dashboard and view the available Resorts with pricing details. The customer can book the Resort and can view the booked status, also customer can delete the booking.

Admin side:

The administrative flow within the portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of Resorts details. The admin can view the booking details submitted by the customers and can change the booking status such as 'Accepted' or 'Rejected'.

Modules of the Application:

ADMIN:

- Register

- Login
- Dashboard
 - Add Resort details
 - View Resort details (Edit and Delete)
 - View Booking details (Edit booking status)
 - View All Reviews

Customer:

- Register
- Login
- Dashboard
 - View Resort details (Add Booking)
 - View booking (Delete)
 - Add Review/ Feedback/ Complaint

Technology Stack

Front End	Angular 10+, HTML, CSS
Back End	.Net WebAPI, EF Core, Ms SQLServer Database

Application assumptions:

1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by implementing Auth Guard, utilizing the canActivate interface. For example, if the user enters as <http://localhost:8080/dashboard> or <http://localhost:8080/user> the page should not navigate to the corresponding page instead it should redirect to the login page.
3. Unless logged into the system, the user cannot navigate to any other pages.
4. Logging out must again redirect to the login page.

Project Tasks:

Admin Role:

Action	URL	Method	Response
--------	-----	--------	----------

Login	/api/login	POST	Response code 200 with login object on successful response or else 404
Register	/api/register	POST	Returns response code 201 with User object on successful creation or else 500
Get All Resorts	/api/Resort	GET	Returns response code 200 with List<Resort> object within response body on successful retrieval or else 500
Add Resort	/api/Resort	POST	Returns Response code 201 with Resort object on successful creation or else 500
Update Resort	/api/Resort/{ResortId}	PUT	Response code 200 with Resort data on successful updation or else 500
Delete Resort	/api/Resort/{ResortId}	DELETE	Response code 200 with Resort data deleted on successful deletion or else 500
View All Booking	/api/booking	GET	Response code 200 with List<Resort> objects on successful response or else 500
View Booking by ID	/api/booking/{bookingId}	GET	Response code 200 with Resort object on successful response or else 404
Update Booking	/api/booking/{bookingId}	PUT	Returns Response code 200 with updated booking object on successful updation or else 500
View All Reviews	/api/review	GET	Response code 200 with List<Review> objects on successful response or else 500

Customer Side:

Action	URL	Method	Response
--------	-----	--------	----------

Booking	/api/booking	POST	Returns Response code 201 with booking object within response body on successful creation or else 500
Cancel Booking	/api/booking/{bookingId}	DELETE	Returns Response code 200 with deleted booking object within response body on successful deletion or else 500
Edit Booking	/api/booking/{bookingId}	PUT	Returns Response code 200 with updated booking object within response body on successful updation or else 500
View Booking by Id	/api/booking/{bookingId}	GET	Response code 200 with booking object on successful response or else 404
View Booking by UserId	/api/booking/user/{userId}	GET	Response code 200 with List<Booking> objects on successful response or else 500
Add Review	/api/review	POST	Returns Response code 201 with review object within response body on successful creation or else 500

Backend Requirements:

Create folders named as model, repository, service, controller, exception and configuration inside dotnetapp as mentioned in the below screenshot.

Backend Project Structure:

<<Need to add dotnetapp project folder structure>>

- In exception folder, create the exception classes to handle user defined exceptions.

Model Classes:

Inside model folder create all the model classes mentioned below.

User: This class stores the user role (admin or the customer) and all user information.

- email: String
- userId: Long
- password:String
- username: String

- mobileNumber: String
- userRole: String (ADMIN/CUSTOMER)

Resort: This class stores all Resort information.

- resortId: Long
- resortName: String
- resortImageUrl: String
- resortLocation: String
- resortAvailableStatus: String
- price: Long
- capacity: Integer (number of rooms)
- description: String
- List<Booking>: bookings (OneToMany)

Booking: This class stores all Booking information.

- bookingId: Long
- noOfPersons: Integer
- fromDate: Date
- toDate: Date
- totalPrice: Double
- address: string
- User: user (ManyToOne)
- Resort: Resort (ManyToOne)

Review: This class stores all Review information

- reviewId: Integer
- userId: Long
- subject: String
- body: String
- rating: Integer
- dateCreated: DateTime

Interface to be used:

1. Create an interface **ResortService** inside service folder with the below mentioned abstract methods and provide implementation for these methods in **ResortServiceImpl** class.

```
public interface ResortService {
    Resort addResort(Resort Resort);
    Resort deleteResort(Long ResortId);
    List<Resort> getAllResorts();
    Resort getResortById(Long ResortId);
    Resort updateResort(Resort e, Long ResortId);
}
```

2. Create an interface **BookingService** inside service folder with the below mentioned abstract methods and provide implementation for these methods in **BookingServiceImpl** class.

```
public interface BookingService {  
    Booking createBooking(Booking booking);  
    Booking updateBooking(Long bookingId, Booking booking);  
    Booking deleteBooking(Long bookingId);  
    Booking getBookingById(Long bookingId);  
    List<Booking> getBookingsByUserId(Long userId);  
    List<Booking> getAllBookings();  
}
```

3. Create an interface ReviewService inside service folder with the below mentioned abstract methods and provide implementation for these methods in ReviewServiceImpl class.

```
public interface ReviewService {  
    Review addReview(Review review);  
    List<Review> getAllReviews();  
}
```

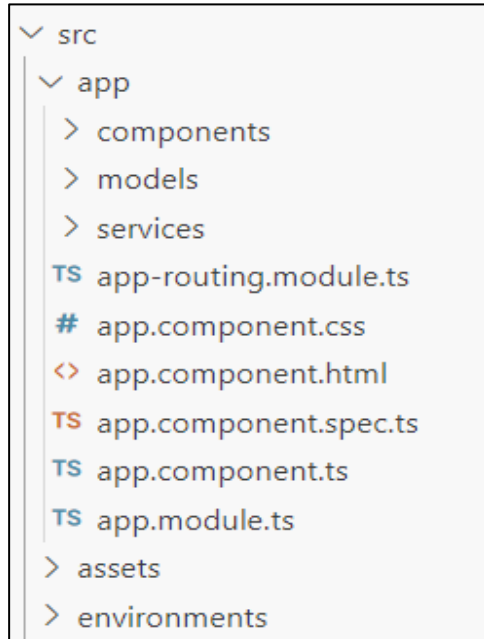
4. Create an interface **UserService** inside service folder with the below mentioned abstract methods and provide implementation for these methods in **UserServiceImpl** class.

```
public interface UserService {  
    public User registerUser(User user);  
    public User loginUser(User user);  
}
```

Frontend Requirements:

- Create a folder named **controllers** inside app to store all the components. (Refer project structure screenshots)
- Create a folder named **model** inside app to store all the model interface.
- Create a folder named as **services** inside app to implement all the services
- Create model interface referring the backend entities (User, Resort and Booking) mentioned in the backend requirements accordingly.
- We have added sample components to be used in the application in the below screenshot. You can create your own components based on the application requirements.
- Import model files, services and components as required.

Project Structure:



Components Structure:

<<Need to add components folder structure>>

Services Structure:

<<Need to add service folder structure>>

Frontend services:

Declare a public property **apiUrl** to store the backend URL in all the services.

For example, **public apiUrl = 'http://localhost:8080'**. Instead of 'localhost', replace it with the URL of your workspace port 8080 URL. For the API's to be used please refer the API Table.

1. AuthService:

- Create a service named **auth** inside app/services folder to implement the following functions.
- **login**(email,password) - returns user model – Sends POST request to `\${this.apiUrl}/api/login`
- **register**(user) - returns user model – Sends POST request to `\${this.apiUrl}/api/register`
- **logout**() - Remove all the localStorage item.

2. ResortService:

- Create service name as **Resort** and implement the following functions in it.
- **addResort**(Resort: any) – Sends POST request to `\${this.apiUrl}/api/Resort`
- **getAllResorts**() – Sends GET request to `\${this.apiUrl}/api/Resort`
- **updateResort**(ResortDetails: any) – UPDATE
- **deleteResort**(ResortId: string) – DELETE

3. BookingService:

- Create service name as **booking** and implement the following functions in it.
- **addBooking**(booking: any) – Sends POST request to `\${this.apiUrl}/api/booking`

- getBookingsByUserId() – Sends GET request to `\${this.apiUrl}/api/booking/user/\${(userId)}`
- updateBooking(booking: any) – PUT
- deleteBooking(bookingId: string) – DELETE
- getAllBookings() – GET

4. ReviewService:

- Create service name as **Review** and implement the following functions in it.
- addReview(Review: any) – Sends POST request to `\${this.apiUrl}/api/Review`
- getAllReviews() – Sends GET request to `\${this.apiUrl}/api/Review`

Validations:

Client-Side Validation:

Implement client-side validation using HTML5 attributes and JavaScript to validate user input before making API requests.

Provide immediate feedback to users for invalid input, such as displaying error messages near the input fields.

Server-Side Validation:

Implement server-side validation in the controllers to ensure data integrity.

Validate user input and API responses to prevent unexpected or malicious data from affecting the application.

Return appropriate validation error messages to the user interface for any validation failures.

Exception Handling

Implement exception handling mechanisms in the controllers to gracefully handle errors and exceptions. Define custom exception classes for different error scenarios, such as API communication errors or database errors.

Log exceptions for debugging purposes while presenting user-friendly error messages to users.

Record all the exceptions and errors handled store in separate table “**ErrorLogs**”.

Error Pages:

Create custom error pages for different HTTP status codes (e.g., **404** Not Found, **500** Internal Server Error) to provide a consistent and user-friendly error experience. Ensure that error pages contain helpful information and guidance for users.

Thus, create a reliable and user-friendly web application that not only meets user expectations but also provides a robust and secure experience, even when faced with unexpected situations.

Something Went Wrong

We're sorry, but an error occurred. Please try again later.

Frontend Sample Screenshots:

User side (Admin and Customer):

1. **Registration:** This page is used for registration purpose for both the roles. Refer the below screenshots for validations.

Register Page:

Registration

Username:

Email:

Password:

Confirm Password:

Mobile Number:

Role:

Select a role

Register

Perform validations for all the form fields as below.

Registration

Username:

*Username is required

Email:

*Email is required

Password:

*Password is required

Confirm Password:

*Confirm Password is required

Mobile Number:

*Mobile number is required

Role:

Select a role

*Role is required

Register

Registration

Username:

Email:

*Please enter a valid email

Password:

*Password must include at least one uppercase letter, one lowercase letter, one digit, and one special character

Confirm Password:

*Passwords do not match

Mobile Number:

*Mobile number must be 10 digits

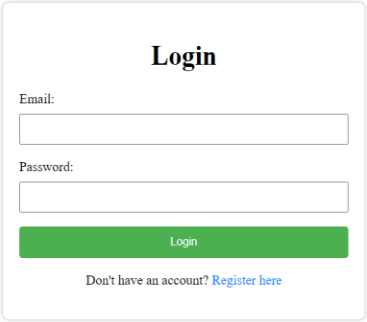
Role:

Select a role

Register

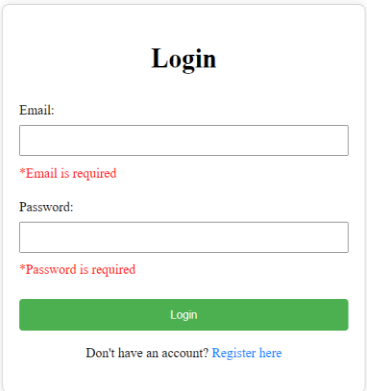
On clicking the ‘Register’ button, the user must be navigated to login page.

- Login:** This page is used for logging in to the application. On providing the valid email and password, the user will be logged in.

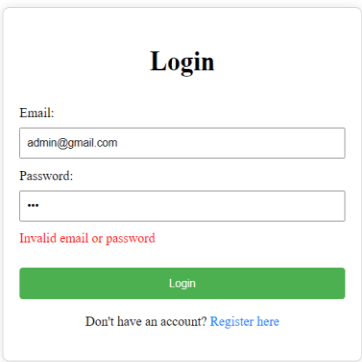


The image shows a login form titled "Login". It has two input fields: "Email:" and "Password:". Below the fields is a green "Login" button. At the bottom, there is a link that says "Don't have an account? [Register here](#)".

Perform validations for email and password fields.



The image shows the login form with validation messages. Below the "Email:" field, there is a red message: "*Email is required". Below the "Password:" field, there is a red message: "*Password is required". The "Login" button and the "Register here" link are still present.



The image shows the login form with a message indicating invalid credentials. The "Email:" field contains "admin@gmail.com" and the "Password:" field contains three asterisks "...". Below the fields, there is a red message: "Invalid email or password". The "Login" button and the "Register here" link are still present.

On Clicking the 'Login' button, user will be navigated to the user dashboard based on their roles.

ADMIN SIDE:

3. **Home Component:** This page is used to display the information about the Resort booking application. On clicking the 'Home' tab, user can view the information about the application.

<<Need to add Home page of Resort Booking>>

User can navigate to other pages, by clicking on the menu available in the navigation bar.

4. **Add Resort:** This page is used to add a new Resort to the system. On Clicking the 'Add Resort' button, admin can view this page.

<<Need to add Add Resort page of Resort Booking>>

Perform validations for all the form fields.

<<Need to add Validation for Add Resort of Resort Booking>>

<<Need to add page Add Resort with data of Resort Booking>>

On Clicking the 'Add Resort' button a Resort is add to the system and user can add a new Resort again. To move to other pages admin can click any of the menus available in the nav bar.

5. **Admin View Resort:** This page is used to display the list of Resorts available. On clicking the 'View Resort' button, admin can view this page.

<<Need to add Admin View Resort Booking>>

On clicking the edit button, the corresponding row should be editable and updated data should be saved.

<<Need to add confirmation page of Resort Booking>>

On clicking the delete button, a pop-up should be displayed with confirmatory message to delete the particular data. After successful deletion, the table should be refreshed as mentioned in the above images.

6. **Admin View Booking:** This page is used to display the list of bookings done by customers. On clicking the 'View Bookings button, admin can view this page.

Admin can accept and reject the bookings of customers.

On clicking the logout button, a pop-up should be displayed with confirmatory message to logout the user.

Add Screenshot for View All Reviews

CUSTOMER SIDE:

This page is used to display the information about the Resort booking application. On clicking the 'Home' tab, user can view the information about the application.

7. **Customer View Resort:** This page is used to display the list of Resorts available. On clicking the 'View Resort' button, customer can view this page.
8. **Add Booking:** This page is used to add booking of Resorts available. On clicking the 'Add Booking' button from 'View Resorts' page, customer can view this page. Resort names will be displayed in dropdown and total price will be displayed based on selection as below.

Perform validations for all the form fields.

On Clicking the 'Submit' button a booking is add to the system and user can add a new booking again. To move to other pages customer can click any of the menus available in the nav bar.

9. **Customer View Booking:** This page is used to display the list of Resorts available. On clicking the 'View Bookings' button, customer can view this page.

Customer can view their booking status in this page.

On clicking the delete button, a pop-up should be displayed with confirmatory message to delete the particular booking. After successful deletion, the table should be refreshed as mentioned in the above images.

Add Screenshot for Add Reviews

On clicking the logout button, a pop-up should be displayed with confirmatory message to logout the user.

Platform Prerequisites (Do's and Don'ts):

- The angular app should run in port 8081.

- The dotnet app should run in port 8080.

To incorporate .Net Security into the application, use JWT authentication within the project workspace.

Key points to remember:

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. Remember to check the screenshots provided with the SRS.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints mentioned in API endpoints section.
5. Don't delete any files in a project environment.

HOW TO RUN THE PROJECT:

FRONTEND:

Step 1:

- Use "cd angularapp" command to go inside the angularapp folder
- Install Node Modules - "npm install"

Step 2:

- Write the code inside src folder
- Create the necessary components
- To create Service: "**npx ng g s <service name>**"
- To create Component: "**npx ng g c <component name>**"

Step 3:

- Click the run test case button to run the test cases

Step 4:

- Please add the following instruction to the projects with DELETE functionality.
- Before running the testcases, drop the database using ***DROP database <DB name>*** command in the terminal.

Note :

- Click PORT 8081 to view the result / output
- If any error persists while running the app, delete the node modules and reinstall them

BACKEND:

API endpoint:
8080

Platform Guidelines:

To run the command, use **Terminal** in the platform.

Dotnet:

Navigate to the dotnetapp directory => **cd dotnetapp** To start/run the application '**dotnet run**' Click on the Run Test Case button to pass all the test cases.