Back-end Deployment Documentation

---

1. Prerequisites

1.1 Environment Setup

Ensure the following tools and software are installed on the deployment environment:

- **JDK**: Version 17 or later.
- **Build Tool**: Maven.
- **Database**: PostgreSQL.
- **Application Server**: Ubuntu-based server hosted on Vultr.

Vultr Server Details:

- **IP**: 155.138.237.5
- **Username**: root
- **Password**: g.B3=,qruS5-L6TN

Access the server via SSH or SFTP using the above credentials.

1.2 Environment Variables

Set up the following environment variables on the server:

- `JAVA_HOME`: Path to the JDK installation.
- `DB_URL`: JDBC URL for the database. Example: `jdbc:postgresql://155.138.237.5:5432/quick_escape`
- `DB_USERNAME`: `postgres`
- `DB_PASSWORD`: `QuickEsc@pe@Em0ry`

1.3 Database Configuration

Ensure the PostgreSQL database is set up with the following credentials:

- **Host**: 155.138.237.5
- **Port**: 5432
- **Database Name**: quick_escape
- **Username**: postgres
- **Password**: QuickEsc@pe@Em0ry

1.4 Server Access Instructions

To access the application server:

1. Use an **SSH client** (e.g., PuTTY, OpenSSH) with the following command:
`ssh root@155.138.237.5`

2. Enter the password: `g.B3=,qruS5-L6TN.`

To transfer files, use an **SFTP client** (e.g., FileZilla, WinSCP) with the same credentials.

---

2. Database Configuration

Update the database configurations in the `application.properties` file to match the target environment:

```
spring.datasource.url=jdbc:postgresql://155.138.237.5:5432/quick_escape
spring.datasource.username=postgres
spring.datasource.password=QuickEsc@pe@Em0ry
```

MyBatis Configuration

Add the following MyBatis settings in `application.properties`:

```
mybatis.mapper-locations=classpath:mapper/*.xml
mybatis.configuration.map-underscore-to-camel-case=true
```

---

3. Deployment Steps

3.1 Connect to the Server

Access the server using SSH:
Use the following command to connect to the target server:
`ssh root@155.138.237.5`
Navigate to the deployment directory:
`cd QuickEscapes-0.0.1-SNAPSHOT.jar`

3.2 Build the Project

On your local machine, build the project using Maven:
`mvn clean package`

The generated JAR file will be located in the `target` directory. Example:
`target/quickescapes.jar`

3.3 Transfer the JAR File to the Server

Use the `scp` command to transfer the JAR file to the server:

```
scp target/quickescapes.jar root@155.138.237.5:~
```

3.4 Configure Application Properties

Update the `application.properties` file or create an environment-specific configuration file (e.g., `application-prod.properties`) in the deployment directory.

Ensure the `spring.profiles.active` property is set to the correct profile. Example:
`spring.profiles.active=prod`

3.5 Run the Application

Start the application using the following command:
`java -jar quickescapes.jar`

Verify the application is running by checking the logs:
`tail -f app.log`

---

4. CI/CD Deployment

Build the project:
`mvn clean package`

1. **Transfer** the generated JAR file to the target server.
2. **Restart** the application:

Stop the current instance if necessary:
`pkill -f quickescapes.jar`

Start the new instance using the commands listed above.

---

5. Post-Deployment Verification

5.1 Verify Logs

Check the application logs for any errors:

```
tail -f app.log
```

5.2 Verify Application Health

Use the `/actuator/health` endpoint to check the application status:

```
curl http://<server-ip>:<port>/actuator/health
```

Back-end API Documentation

1. Overview

Provide a brief description of the backend APIs, their purpose, and intended use.

- **Project Name**: QuickEscapes API
- **Version**: v1.0
- **Base URL**: `https://api.quickescapes.com`

---

2. Authentication

Describe the authentication method required to access the APIs.

- **Authentication Type**: Bearer Token

Header:
`Authorization: Bearer <access_token>`

---

3. API Endpoints

3.1 User Management

3.1.1 Register a User

- **Endpoint**: `POST /user/register`
- **Description**: Registers a new user.
- **Request**:

Body:

```
{
 "username": "String",
 "password": "String",
 "email": "String"
}
```

- **Response**:

Success:

```
{
  "message": "User registered successfully.",
  "userId": "string"
}
```

Error:

```
{
  "code": 101
  "msg": "User already exists."
}
```

3.1.2 Login a User

- **Endpoint**: `POST 8080/users/login`
- **Description**: Logs in an existing user and returns a JWT token.
- **Request**:

Headers:

```
{
  "Authorization": Bearer "application/json"
}
```

Body:

```
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

- **Response**:

Success:

```
{
  "data": "string"
}
```

Error:

```
{
  "error": "Invalid email or password."
}
```

3.2 Destination Management

3.2.1 Get All Destinations

- **Endpoint**: `GET /itinerary/find`
- **Description**: Retrieves a list of all available destinations.
- **Request**:

Headers:

```
{
  "Authorization": "Bearer <token>"
}
```

- **Response**:

Success:

```
[
  "code": "0",
  "data":
  {
    "cityCode": "string",
    "roundTrips": "List<RoundTrip>",
    "hotels": "List<Hotel>",
  }
  "msg":
]
```

Returned Sample output

```
RoundTrip : {
                    "cityCode": "Milwaukee",
                    "outboundDepartureDate": "2024-12-15T14:30:00",
                    "outboundArrivalDate": "2024-12-15T18:38:00",
                    "outboundPrice": 279.98,
```

```
                    "outboundFlightNumbers": "AA1757",
                    "outboundCabin": "COACH",
                    "returnDepartureDate": "2024-12-18T07:20:00",
                    "returnArrivalDate": "2024-12-18T11:21:00",
                    "returnPrice": 279.98,
                    "returnFlightNumbers": "AA2934",
                    "returnCabin": "COACH",
                    "totalPrice": 559.96,
                    "attributes": "city, culture, foodie"
                }
```

```
Hotel:{             "destination": "Minneapolis",
                    "hotelName": "Days Hotel by Wyndham SE",
                    "totalPrice": 475.84,
                    "finalCheckInDate": "2024-12-15",
                    "finalCheckOutDate": "2024-12-20",
                    "photoUrl":
"https://cf.bstatic.com/xdata/images/hotel/square500/135451216.jpg?k=48acba1e39
a0a499b59e9d3d8702539838df785c493529987a04ebc8f0d59d98&o=",
                    "attributes": "city, nature, culture"
}
```


Error:


```
{
  "error": "Unauthorized access."
}
```

3.4 Error Codes

Provide a list of standard error codes and their meanings:

Code	Description
400	Bad Request
401	Unauthorized Access

403	Forbidden
404	Resource Not Found
500	Internal Server Error

---

5. Testing the APIs

Test the APIs using tools like **Postman** or **cURL**:

- **Postman**:
    1. Import the collection provided at `<link>`.
    2. Set up environment variables (e.g., `base_url`, `token`).

cURL Example:

```
curl -X GET https://api.quickescapes.com/api/v1/destinations \

-H "Authorization: Bearer <token>"
```

Back-end Technical Documentation

Directory Structure

1. src/main/java/com/example/quickescapes/

This is the root package for the backend application. Below are the sub-packages and their purposes:

1.1 config/

Purpose:
Contains configuration classes for the application, including Spring Boot and third-party library configurations.
Examples:
Security configuration (e.g., user authentication/authorization setup).
Database connection settings.
CORS or other global application settings.

1.2 controller/

Purpose:
Acts as the entry point for handling HTTP requests (the "Controller" in MVC). Controllers define the RESTful endpoints for the application.
Responsibilities:
Mapping URLs to service methods.
Handling request validation.
Returning responses to the client.
Examples:
UserController: Handles endpoints for user registration and login.
HotelController: Handles endpoints for hotel search and related operations.

1.3 dao/ (Data Access Objects)

Purpose:
Provides an abstraction for database operations. Contains interfaces or classes that interact with the database using SQL queries or ORM tools.
Responsibilities:
CRUD operations for database entities.
Querying the database based on business needs.
Examples:
UserDAO: Contains methods for fetching and saving user details.
HotelDAO: Contains methods for fetching hotel-related information.

1.4 interceptors/

Purpose:
Contains interceptor classes for handling cross-cutting concerns like logging, authentication, or request/response pre-processing.
Responsibilities:
Modifying or validating incoming requests or outgoing responses.
Managing session or token-based authentication.
Examples:
AuthenticationInterceptor: Validates user authentication tokens.

1.5 mappers/

Purpose:
Provides mapping between database tables and application objects.
Responsibilities:
Translate database query results into Java objects (DTOs or entities).
Map objects back to database queries for insert/update operations.
Examples:
UserMapper: Maps user database fields to User class attributes.
HotelMapper: Maps hotel database fields to corresponding Java objects.

1.6 service/

Purpose: Contains interfaces defining business logic and core functionalities.
Responsibilities:
Abstract the core logic of the application.
Expose high-level methods for the controller to call.
Examples:
UserService: Interface for user registration, login, and account management logic.
HotelService: Interface for hotel search and booking logic.

1.7 serviceImpl/

Purpose: Implements the service interfaces in the service/ package.
Responsibilities:
Provide concrete implementations of business logic.
Call DAOs to interact with the database as needed.
Handle any intermediate processing or error handling.
Examples:
UserServiceImpl: Implements UserService with logic for user authentication and registration.
HotelServiceImpl: Implements HotelService with logic for searching hotels and returning results.

1.8 util/

Purpose: Contains utility classes or methods used across the application.
Responsibilities:
Provide reusable helper methods or constants.
Encapsulate non-business-specific logic (e.g., string manipulation, date formatting).
Examples:
JwtUtil: Handles JWT token creation and validation.
DateUtil: Provides helper methods for parsing and formatting dates.

1.9 QuickEscapesApplication.java

Purpose: The main entry point for the Spring Boot application.
Responsibilities:
Bootstraps the Spring application context.
Starts the embedded web server.
Loads all configurations and dependencies.

2. resources/

Purpose: Contains non-Java resources such as configuration files, SQL scripts, or static assets.
Examples:
application.properties: Stores configuration settings (e.g., database URLs, server ports).
data.sql: Contains SQL scripts for initializing or populating the database.

3. Build and Dependency Management

The project uses Maven for dependency management and build automation. Key files:
pom.xml:
Lists dependencies such as Spring Boot, MyBatis, PostgreSQL drivers, etc.
Configures build plugins and project metadata.

Library Dependencies

Core Dependencies

Spring Boot Starter Web
Artifact: spring-boot-starter-web
Group: org.springframework.boot
Purpose: Provides the core web functionality, including embedded Tomcat server, RESTful services, and Spring MVC framework.

Spring Boot Starter Thymeleaf
Artifact: spring-boot-starter-thymeleaf
Group: org.springframework.boot
Purpose: Integrates Thymeleaf as a templating engine for rendering HTML pages.

Spring Boot Starter Security
Artifact: spring-boot-starter-security
Group: org.springframework.boot
Purpose: Provides authentication and authorization mechanisms for securing the application.
Database and Persistence

PostgreSQL JDBC Driver
Artifact: postgresql
Group: org.postgresql
Version: 42.7.4
Purpose: Provides JDBC support for connecting and interacting with a PostgreSQL database.

MyBatis Spring Boot Starter
Artifact: mybatis-spring-boot-starter
Group: org.mybatis.spring.boot
Version: 3.0.3
Purpose: Simplifies integration of MyBatis with Spring Boot, supporting database persistence and SQL mapping.

Utilities

Lombok
Artifact: lombok
Group: org.projectlombok
Purpose: Reduces boilerplate code by generating getters, setters, constructors, and more through annotations.

Android JSON
Artifact: android-json

Group: com.vaadin.external.google
Version: 0.0.20131108.vaadin1
Purpose: Provides JSON parsing and manipulation utilities.

Java JWT
Artifact: java-jwt
Group: com.auth0
Version: 4.4.0
Purpose: Enables JSON Web Token (JWT) creation, parsing, and validation for secure API authentication.

Testing

Spring Boot Starter Test
Artifact: spring-boot-starter-test
Group: org.springframework.boot
Scope: test
Purpose: Provides libraries for testing Spring Boot applications, including JUnit, Mockito, and Spring-specific test utilities.

JUnit Jupiter API
Artifact: junit-jupiter-api
Group: org.junit.jupiter
Version: 5.10.0
Scope: test
Purpose: Enables unit testing using the JUnit 5 framework.

Build and Plugin

Spring Boot Maven Plugin
Artifact: spring-boot-maven-plugin
Group: org.springframework.boot
Purpose: Supports packaging and running Spring Boot applications. Provides integration with Maven for building the application.

Project Properties

Java Version: 17
The project uses Java 17, ensuring compatibility with the latest features and security updates.