**Funcity Amusement Park Ticketing System Application Documentation**

**Introduction**

Funcity is an online Ticket application developed for ABC Company. The application is designed to provide a simple and convenient platform for customers to browse and purchase tickets and for admins to manage the Activity in the Amusement Park. The Spring Boot application for the amusement park ticket system is a web-based application designed to facilitate the purchase, management, and reporting of tickets for an amusement park. It is built using the Spring Boot framework, which is a popular Java-based framework for building web applications.

The amusement park ticket system allows visitors to the park to purchase tickets online. The system also provides administrators with a set of tools for managing tickets, managing Customers, and managing Activities.

The application is designed to be scalable, reliable, and easy to use. It is built using modern web development practices, including responsive design, RESTful APIs, and a user-friendly interface.

With its rich set of features and capabilities, the amusement park ticket system is an essential tool for any amusement park looking to streamline its ticketing operations and improve the overall visitor experience. Whether you're a park visitor looking to purchase tickets or an administrator looking to manage sales and generate reports, the Spring Boot application for the amusement park ticket system has everything you need to make your experience seamless and efficient.

**Getting Started**

This section provides a guide to getting started with the Funcity Amusement Park Application. Before you begin, please make sure that you have the following prerequisites installed:

1. Java Development Kit (JDK) version 8 or higher

Maven build tool version 3.2 or higher

MySQL relational database management system version 5.6 or higher

1. Installing and Running the Application
2. Clone the FunCity project repository from GitHub using the following command:

“git clone https://github.com/ABCCompany/FunCity.git”

1. Create a MySQL database and configure the database settings in the application.properties file located in the src/main/resources directory. The default database settings are as follows:

“spring.datasource.url=jdbc:mysql://localhost:3306/funcitydb

spring.datasource.username=root

spring.datasource.password=root”

Please make sure to replace the database URL, username, and password with your own database settings.

Please make sure to configure the database settings and other environment-specific settings in the application.properties file before deploying the application to a production environment.

**Architecture**

The Funcity Amusement Park Application is built using the Model-View-Controller (MVC) architecture pattern. The application is developed using Java Core, Spring Boot, and MySQL.

The MVC architecture pattern separates the application into three main components: the Model, the View, and the Controller. This separation of concerns allows for easier development, maintenance, and testing of the application.

Controller

The Controller acts as an intermediary between the Model and the View. It handles user requests, processes input data, and updates the Model and the View accordingly. In the FunCity application, the Controller is implemented using Spring MVC, which provides a robust and flexible framework for building web applications.

Database

The FunCity application uses MySQL as its relational database management system. The database stores and retrieves data related to Customers, Tickets, Activities, and Admins. The database is accessed using the Spring Data JPA framework, which provides a powerful and easy-to-use interface for working with databases.

Overall, the FunCity Application is designed to be a scalable and maintainable web application that follows industry-standard architecture patterns and best practices.

**Modules**

1. Login Module: This module allows users to register, log in, and log out of the application. It includes functionality for secure user authentication and authorization, as well as password reset functionality.
2. Customer Module: This module enables customers to browse, search for, and purchase tickets for different activities in Amusement Park.
3. Admin Module: The admin module for the amusement park ticket system provides administrators with a set of tools for managing tickets, managing Customers, and managing Activity data. The module is designed to be easy to use and highly efficient, allowing administrators to quickly perform tasks and access critical data.
4. Ticket Booking Management Module: The ticket booking management module is a critical component of the amusement park ticket system, allowing visitors to purchase tickets online and manage their bookings. The module is designed to be user-friendly, intuitive, and efficient, providing visitors with a seamless booking experience.
5. Activity Management Module: The activity management module is a key component of the amusement park ticket system, providing administrators with the tools they need to manage and schedule activities and events within the park. The module is designed to be intuitive, efficient, and flexible, allowing administrators to easily create and manage activities based on visitor demand and park capacity.

**Login Module**

To Login into the Application

* Endpoint: POST/login
* RequestBody : JSON object containing login credentials

{

"mobileNumber": "91\*\*\*\*\*\*\*\*",

"password": "abc@1\*\*\*",

"role": "customer"

}

* Response: String sessionId

To logout from the Application

* Endpoint: DELETE/logout
* RequestParam:sessionId
* Response: String “Successfully logged Out”

**Customer Module**

To register new Customer:

* Endpoint: POST /customers
* Request Body: JSON object containing the customer details:

{

"customerId": 0,

"customerName": "string",

"email": "string",

"mobileNumber": "stringstri",

"address": "string",

"dateOfBirth": "2023-05-08",

"username": "string",

"password": "string",

"tickets": []

}

* Response: JSON object containing the saved customer details, including the generated customerID.

To delete a customer from the database:

* Endpoint: DELETE /customer/{customerId}
* Response: HTTP 200 OK with no response body.

To retrieve a specific customer from the database:

* Endpoint: GET /customers/{customerId}
* Response: JSON object containing the customer details.

To retrieve all customers in the database:

* Endpoint: GET /customers
* Response: JSON array containing all customers in the database, each represented as a JSON object.

To update Password of a customer

* Endpoint: GET /customers/{customerId}/{oldpassword}/{password}
* Response: JSON object containing the updated customer details.

To update Details of Customer

* Endpoint: GET /customers/{customerId}
* Request Body: JSON object containing the customer details:

{

"customerName": "string",

"email": "string",

"mobileNumber": "stringstri",

"address": "string",

"dateOfBirth": "2023-05-08",

}

* Response: JSON object containing the updated customer details.

**Admin Module**

To register new Admin:

* Endpoint: POST /admins
* Request Body: JSON object containing the admin details:

{

"admin\_id": 0,

"email": "string",

"username": "string",

"mobileNumber": "string",

"password": "string"

}

* Response: JSON object containing the saved admin details, including the generated adminID.

To retrieve a admin from database:

* Endpoint: GET /{sessionId}/{id}
* Response: JSON object containing the admin details

To delete a admin from database:

* Endpoint:DELETE /{sessionId}/{id}
* Response: JSON object containing the deleted admin details

To update a admin details in database:

* Endpoint:PUT /{sessionId}/{adminId}
* Request Body: JSON object containing the admin details:

{

"admin\_id": 0,

"email": "string",

"username": "string",

"mobileNumber": "string",

"password": "string"

}

* Response: JSON object containing the updated admin details

To get all activity details in database:

* Endpoint:GET /allActivities
* Response: JSON array containing all customers in the database, each represented as a JSON object.

To get a particular customers activities:

* Endpoint:GET /customeractivities/{sessionId}/{customerId}
* Response: JSON array containing all activities of a particular customer in the database, each represented as a JSON object.

To get a particular customers activities by date range:

* Endpoint:GET /activities/{sessionId}/{customerId}/{startDate}/{endDate}
* Response: JSON array containing all activities of a particular customer in Date range in the database, each represented as a JSON object.

**Ticket Booking Module**

To generate a new Ticket:

* Endpoint: POST /tickets/{sessionId}
* Request Body: JSON object containing the ticket details:

{

"ticketId": 0,

"dateTime": "2023-05-08T15:07:51.749Z",

"noOfPersons": 0,

"total": 0,

"customerId": 0,

"activityId": 0

}

* Response: JSON object containing ticket details along with the activity associated

To get all Tickets in the database:

* Endpoint: GET /tickets/
* RequestParameter:sessionId
* Response: JSON object array containing ticket details along with the activity associated ,each array object represent a single ticket

To update no of person in a Ticket in the database:

* Endpoint: PATCH/tickets/{ticketsId}/{noOfPersons}
* RequestParameter:sessionId
* Response: String “”

To delete a Ticket in the database:

* Endpoint: DELETE/{ticketsId}
* RequestParameter:sessionId
* Response: String “”

To get a Ticket by id in the database:

* Endpoint: GET /gettickets/{TicketId}
* RequestParameter:sessionId
* Response: JSON object containing ticket details along with the activity associated

To get all Tickets of a customer in the database:

* Endpoint: GET /ticketsofcutomer/{sessionId}/{customerId}
* Response: JSON JSON object array containing ticket details along with the activity associated ,each array object represent a single ticket of a particular customer

To get total of all tickets of a Customer

* Endpoint: GET /calculateBill/{sessionId}/{customerId}
* Response: Total spending on tickets by particular cuatomer

**Activity Management Module**

To register add new Activity to databse:

* Endpoint: POST /activities/{sessionId}
* Request Body: JSON object containing the activity details:

{

"activityId": 0,

"activityName": "string",

"description": "stringstri",

"imageUrl1": "string",

"imageUrl2": "string",

"charges": 500,

"thrillLevel": 5

}

* Response: JSON object containing the saved activity details, including the generated activityID.

To get all Activity in the database:

* Endpoint: GET /activities
* Response: JSON object array containing activity details ,each array object represent a single Activity

To get all Activity by charges in the database:

* Endpoint: GET /activities/{charges}
* Response: JSON object array containing activity details ,each array object represent a single Activity

To get no of Activities by charges in the database:

* Endpoint: GET /activities/count/{charges}
* Response: count of activities

To delete Activity from the database:

* Endpoint: DELETE/activities/{sessioId}/{activityId}
* Response: JSON object containing the deleted activity details.

To update a Activity details in database:

* Endpoint:PUT /activities/{sessionId}
* Request Body: JSON object containing the Activity details:

{

{

"activityName": "string",

"description": "stringstri",

"imageUrl1": "string",

"imageUrl2": "string",

"charges": 500,

"thrillLevel": 5

}}

* Response: JSON object containing the updated Activity details

**Database**

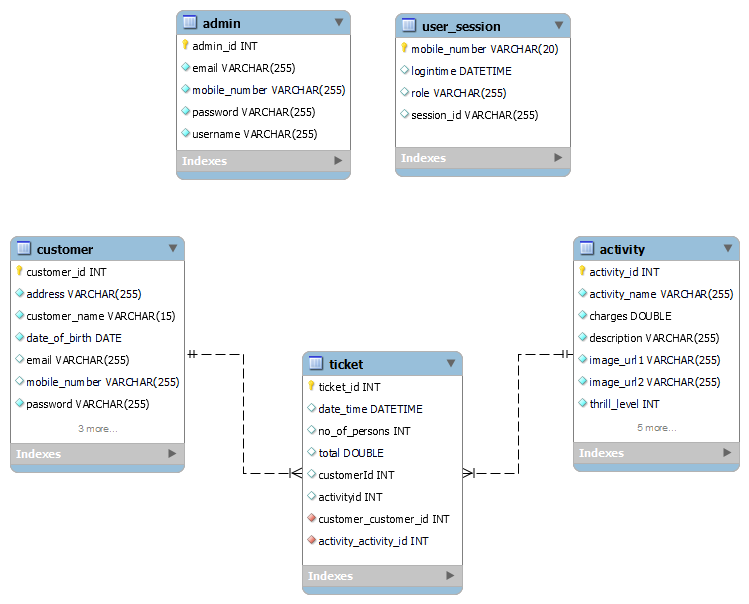
The Customer table stores information about the registered customers including their unique customer ID, name, email address, Date of birth, username, and password. The Admin table stores information about the admins including their unique admin ID, name, email username, mobile number, and password.

The Activity table stores information about the Activities in the Amusement park including the activity ID, name, description, image, charges, and thrill level. The Ticket table stores information about the tickets including the ticket ID, date and time, total, customer id, and activity id.

The database is designed to ensure data consistency and integrity by using foreign key constraints to link the tables together. For example, the ticket table has two foreign key constraints on the Customer and Activity tables.

To interact with the database, the application uses Spring Data JPA which provides an easy-to-use interface for performing CRUD (Create, Read, Update, Delete) operations on the database tables. The application also uses Hibernate as the Object-Relational Mapping (ORM) framework to map the POJO classes to the database tables.

Overall, the use of a relational database in the Amusement Park Application provides a scalable and reliable solution for managing the data of customers, products, orders, and carts.



**Deployment**

This section should provide instructions for deploying the application to a production environment, including any necessary configuration steps and best practices for security and performance.

**Conclusion**

I hope this documentation provides you with a clear understanding of the FunCity Amusement Park Application. If you have any questions or require further clarification, please do not hesitate to reach out. I am here to assist you in any way I can. Thank you for considering this documentation.