Low Level Design Document

For

Version 1.0

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Thabet M.	19/05/2023	Initial Draft	0.1
Mohaemd A. Thabet	20/05/2023	Changed paragraph formatting, and reformatted cover page.	0.2
Ahmed abdo	22/05/2023	Completing the following parts : ERD Project Provided Services	0.3
Mohaemd alaa Ahmed Abdo	23/05/2023	First version release	1.0

1. System Overview

1.1 Purpose

The primary purpose of EVENTHIVE is to simplify the ticket booking process by offering a user-friendly interface that allows users to browse available events, view seating options, and make secure reservations. The website will serve as a centralized platform, aggregating information from various ticket vendors and event organizers, thereby providing users with a comprehensive selection of options.

1.2 Main Features

- Event Search: Users can search for events
- Seat Selection: Users can view available seating options for each event and select their preferred seats. The organizers can upload their own venues schemas.
- Booking and Payment: Users can reserve seats and proceed with secure online payments. Such as VISA, Fawry, E-wallet.
- User Profiles: Users can create and manage their profiles, including personal information, past bookings, and preferences. User can post a review for each event attended.
- Notifications: Users will receive notifications about upcoming events, booking confirmations, and any changes or cancellations.

1.3 Intended Audience

EVENTHIVE website is intended for a diverse audience, including:

- Individuals and families looking for entertainment options.
- Sports enthusiasts interested in attending games or matches.
- Music lovers seeking tickets for concerts or music festivals.
- Movie enthusiasts wanting to book tickets for the latest films.
- Organizations and event planners who want to list and promote their events on the platform

The website provides a convenient platform for event organizers to showcase and manage their events. By allowing organizations to add their events to the website, it expands the range of offerings and ensures a diverse selection of events for users to choose from. Event organizers can create profiles, list event details, manage ticket inventory, and utilize the website's booking and payment system to streamline their event management process.

The website aims to establish a collaborative environment where event organizers can effectively reach their target audience, maximize ticket sales, and enhance event visibility..

2. System Use Cases

Name:	User Registration		
Number:	1.0		
Author:	Mohamed Alaa	**Date:**	May 20, 2023
Description:	the process of a user registering on EVENTHIVE website. The user enters their personal details, including their name, email address, and password. The system validates the information provided by the user and creates a new user account.		
Actors:	User, System		
Pre-conditions:	The user must have access to the registration page.		
Course of Events:	1. The user enters their personal details. 2. The system validates the information 3. The system sends a confirmation email or WhatsApp message to the user to verify their email address. 4. The user verifies their email address or mobile number by clicking on the verification link. 5. The system confirms the user's email or mobile number and completes the registration process. 6. The system creates a user account.		
Post-conditions:	The user successfully registers an account and is directed to home page. Their personal details is stored in the database.		
Alternatives/ Exceptions:	• If the user enters invalid information, the system displays error messages and prompts the user to correct them. • If the user does not receive the confirmation email or message, they can request a new one or contact support for assistance. • If the email address entered by the user is already associated with an existing account: The system displays an error message indicating that the email address is already registered, prompting the user to use a different email address (loop back to step 3).		
Notes:			

Name:	Event Search and Ticket Booking		
Number:	2.0		
Author:	Mohamed Alaa	**Date:**	May 20, 2023
Description:	This use case allows users to search for events based on event name. After selecting a specific event, the user can view available seating options and choose their preferred seats. The system validates the seat availability and allows the user to proceed with the ticket booking process. Once the user provides the necessary payment details and confirms the booking, the system completes the ticket reservation and generates a booking confirmation, which is sent to the user.		
Actors:	User, System		
Pre-conditions:	The user must be logged into the ticket reservation website.		
Course of Events:	1. The user navigates to the event search page. 2. The user enters search criteria. 3. The system retrieves and displays a list of matching events. 4. The user selects an event from the list to view more details. 5. The system presents the event details, including available seating options. 6. The user selects their preferred seats and specifies the desired quantity. 7. The system validates the seat availability and reserves the selected seats. 8. The user proceeds to the payment page and provides payment details. 9. The system securely processes the payment and completes the booking. 10. The system generates a booking confirmation and sends it to the user via email. 11. The system sends tickets QR code to the user via WhatsApp		
Post-conditions:	The user successfully books tickets for the selected event and receives a booking confirmation and tickets QR code.		
Alternatives/ Exceptions:	• If the selected seats are no longer available, the system notifies the user and prompts them to choose alternate seats. • If the payment processing fails, the system notifies the user and provides instructions to retry the payment or contact support.		
Notes:			

Name:	Event Creation and Management		
Number:	3.0		
Author:	Mohamed alaa	**Date:**	May 21, 2023
Description:	This use case enables event admins to create and manage events on the EVENTHIVE website. After logging in, the event admins accesses the event management dashboard. From there, the admin can create a new event listing by providing event details such as name, date, location, and ticket pricing. The system validates the entered information and creates the event listing. The event admin can then manage the event, including updating event details, adding descriptions and images, and monitoring ticket sales. This use case allows event admins to have full control and visibility over their events within the system.		
Actors:	Event Admin, System		
Pre-conditions:	The event organizer must be logged into the ticket reservation website as an authorized user.		
Course of Events:	1. The event admin navigates to the event management dashboard. 2. The event admin selects the option to create a new event. 3. The system prompts the event organizer to enter event details, including name, date, location, venue scheme and ticket pricing. 4. The event admin provides the necessary information and submits the event details. 5. The system validates the entered information and creates a new event listing. 6. The event admin can manage the event listing, update event information, or add additional details (e.g., event description, images).		
Post-conditions:	The event is available to be booked by users.		
Alternatives/ Exceptions:	• If the entered event details are incomplete or contain errors, the system displays error messages and prompts the event admin to correct them. • If the event organizer wants to make changes to an existing event, they can select the event from the management dashboard and edit the relevant information. • The event admin can monitor ticket sales, view booking reports, and track attendance for their events.		
Notes:			

Name:	Ticket QR Code Validation		
Number:	4.0		
Author:	Mohamed Alaa	**Date:**	May 20, 2023
Description:	the process of the event organizer (gate attendant) scanning users' ticket QR codes via EVENTHIVE mobile applications to validate their tickets and determine if they are genuine and unused, allowing them entry into the event venue.		
Actors:	Event Organizer, User, System.		
Pre-conditions:	• The event organizer must have a scanning device with the EVENTHIVE mobile application installed, and must be logged in. • The user must possess a valid ticket with a QR code.		
Course of Events:	1. Preconditions satisfied. 2. The user presents their ticket with the QR code to the event organizer. 3. The scanning device's camera scans the QR code on the user's ticket. 4. The scanning software decodes the QR code and retrieves the relevant ticket information. 5. The software verifies the authenticity of the QR code and checks if the ticket has been used before or is still valid. 6. If the QR code is valid and the ticket is not used or expired: I. The EVENTHIVE app displays a confirmation message indicating that the ticket is valid. II. The event organizer/gate attendant grants the user permission to enter the event venue. III. Postconditions satisfied.		
Post-conditions:	The user's ticket is validated, and they are granted entry into the event venue.		
Alternatives/ Exceptions:	• If the QR code is invalid or the ticket has been used before or expired: I. The scanning software displays an error message indicating that the ticket is invalid. II. The event organizer/gate attendant denies entry to the user. • If the scanning device fails to scan the QR code successfully: The event organizer may manually enter the ticket information or ask the user to present the ticket for visual inspection. • If the scanning software encounters an error or connectivity issues: The event organizer may contact technical support for assistance or follow a manual ticket validation process.		
Notes:			

Name:	Event Feedback and Rating		
Number:	5.0		
Author:	Mohamed Alaa	**Date:**	May 20, 2023
Description:	This use case allows users who have attended an event to provide their feedback and rating, sharing their experience with other users and event organizers.		
Actors:	User, System.		
Pre-conditions:	• The user must have attended the event and be logged into the ticket reservation website.		
Course of Events:	1. Preconditions satisfied. 2. The user navigates to the event details or their booking history, which lists the events they attended. 3. The user selects the specific event they wish to provide feedback and rating for. 4. The system displays the event details, including information about the venue, performers, and other relevant details. 5. The user finds the option to provide feedback and rating for the event. 6. The user enters their feedback comments, rating, and any additional information as required. 7. The system validates the entered information and saves the feedback and rating. 8. The system may display a confirmation message to the user indicating that their feedback has been successfully submitted.		
Post-conditions:	The user successfully submits their feedback and rating for the event and can be viewed by other users in event's organization page		
Alternatives/ Exceptions:	• If the user has not attended the event: The system displays an error message indicating that only users who attended the event can provide feedback and rating. • If the user submits feedback without rating: The system displays appropriate error messages and prompts the user to input the rating (loop back to step 7).		
Notes:			

3. Database Design

3.1 Database Management System (DBMS)

we have chosen PostgreSQL (psql) as the Database Management System (DBMS). PostgreSQL is a powerful open-source relational database system known for its reliability, scalability, and extensive features. Its support for advanced SQL queries, data integrity, and transaction management makes it a suitable choice for our application.

We have selected PostgreSQL for the following reasons:

- **Reliability:** PostgreSQL has a proven track record of reliability and stability, making it a trusted choice for critical applications. It provides features like transactional integrity, crash recovery, and point-in-time recovery, ensuring data consistency and availability.

- **Scalability:** PostgreSQL offers various scalability options, including horizontal scalability through replication and partitioning. It allows us to distribute the database load across multiple servers and handle a growing number of users and transactions.

- **Rich Feature Set:** PostgreSQL supports advanced SQL features, such as complex queries, window functions, and full-text search. It also provides support for JSON, spatial data, and other data types, enabling us to store and query diverse data efficiently.

- **Data Integrity:** PostgreSQL enforces data integrity constraints, such as unique, foreign key, and check constraints, to maintain data consistency and accuracy. It supports ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring reliable transaction management.

- **Extensibility:** PostgreSQL allows us to extend its functionality through user-defined functions, custom data types, and procedural languages. This flexibility enables us to implement specific business logic and optimize performance as per our requirements.

- **Active Community and Ecosystem:** PostgreSQL has a vibrant and active community that provides regular updates, bug fixes, and security patches. It also has a rich ecosystem with a wide range of tools and frameworks for efficient development and administration.

By choosing PSQL as our DBMS, we can leverage its robustness, scalability, and feature-rich nature to build a reliable and high-performance ticket reservation website.

3.2 Tables and Attributes

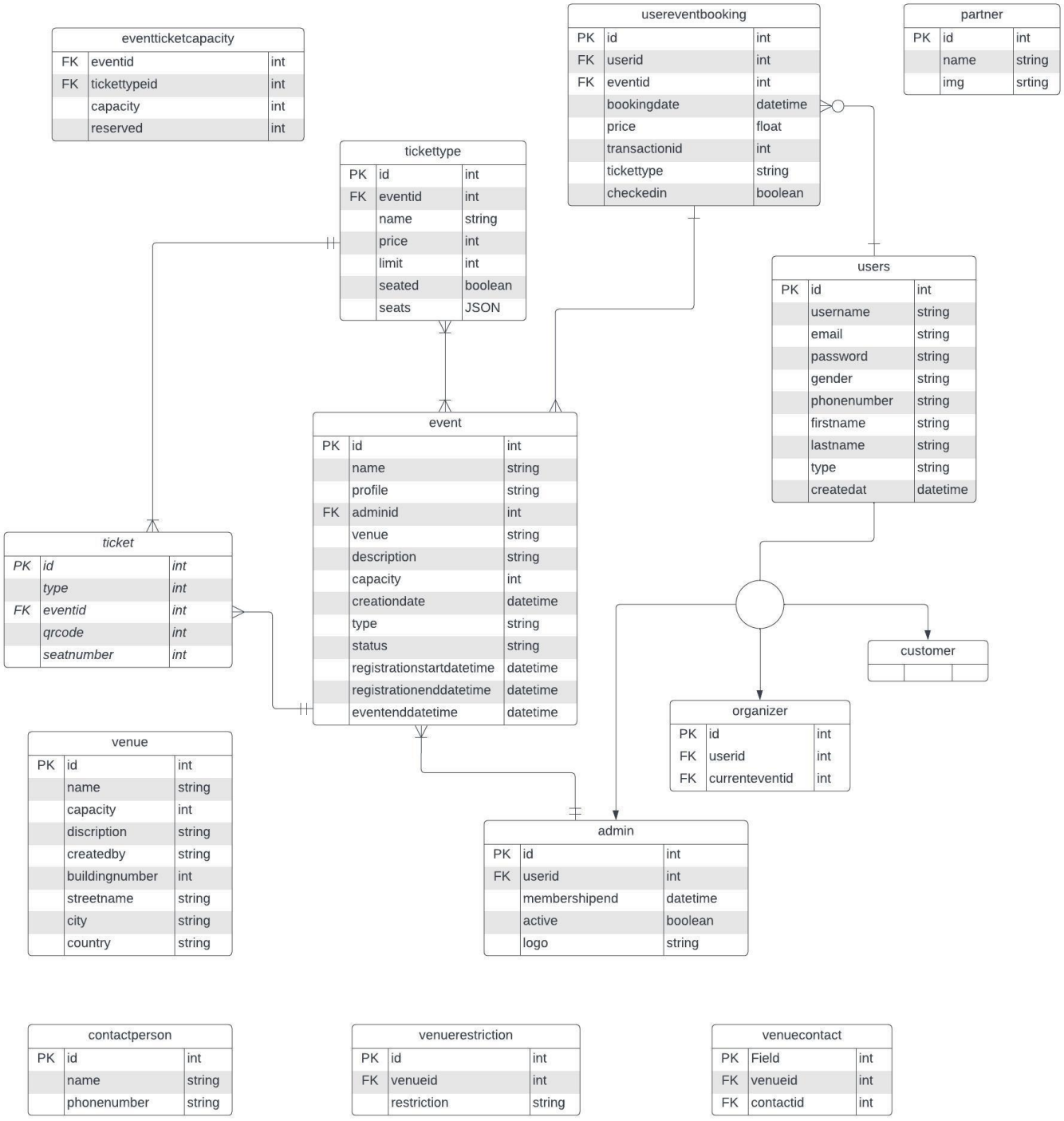
1) Admin Table:
    - Table Name: Admin
    - Attributes:
        - id: Unique identifier for each admin (integer, primary key).
        - userid: Foreign key referencing the User table (integer, unique).
        - membershipend: End date of the admin's membership (datetime).
        - active: Indicates if the admin is active (Boolean).
        - logo: Logo of the admin (string).

2) User Table:
    - able Name: users
    - Attributes:
        - id: Unique identifier for each user (integer, primary key).
        - username: User's login username (string, unique).
        - email: User's email address (string, unique).
        - password: User's password (string).
        - gender: User's gender (string).
        - phonenumber: User's phone number (string, unique).
        - firstname: User's first name (string).
        - lastname: User's last name (string).
        - type: User's type admin, customer or organizer (string).
        - createdat: Timestamp of user’s profile creation (datetime, default : now).
3) Event Table:
    - Table Name: event
    - Attributes:
        - id: Unique identifier for each event (integer, primary key).
        - name: Name of the event (string).
        - profile: Profile of the event (string).
        - adminid: Foreign key referencing the admin table (integer, unique).
        - venue: Venue of the event (string).
        - description: Description of the event (string).
        - capacity: Maximum capacity of the event (integer).
        - creationdate: Timestamp of event creation (datetime).
        - type: Type of the event (string, nullable).
        - status: Status of the event (string, nullable).
        - registrationstartdatetime: Start date and time for event registration (datetime, nullable).
        - registrationenddatetime: End date and time for event registration (datetime, nullable).
        - eventstartdatetime: Start date and time of the event (datetime, nullable).
        - eventenddatetime: End date and time of the event (datetime, nullable).

4) User Event Booking Table:
    - Table Name: usereventbooking
    - Attributes:
        - id: Unique identifier for each user event booking (UUID).

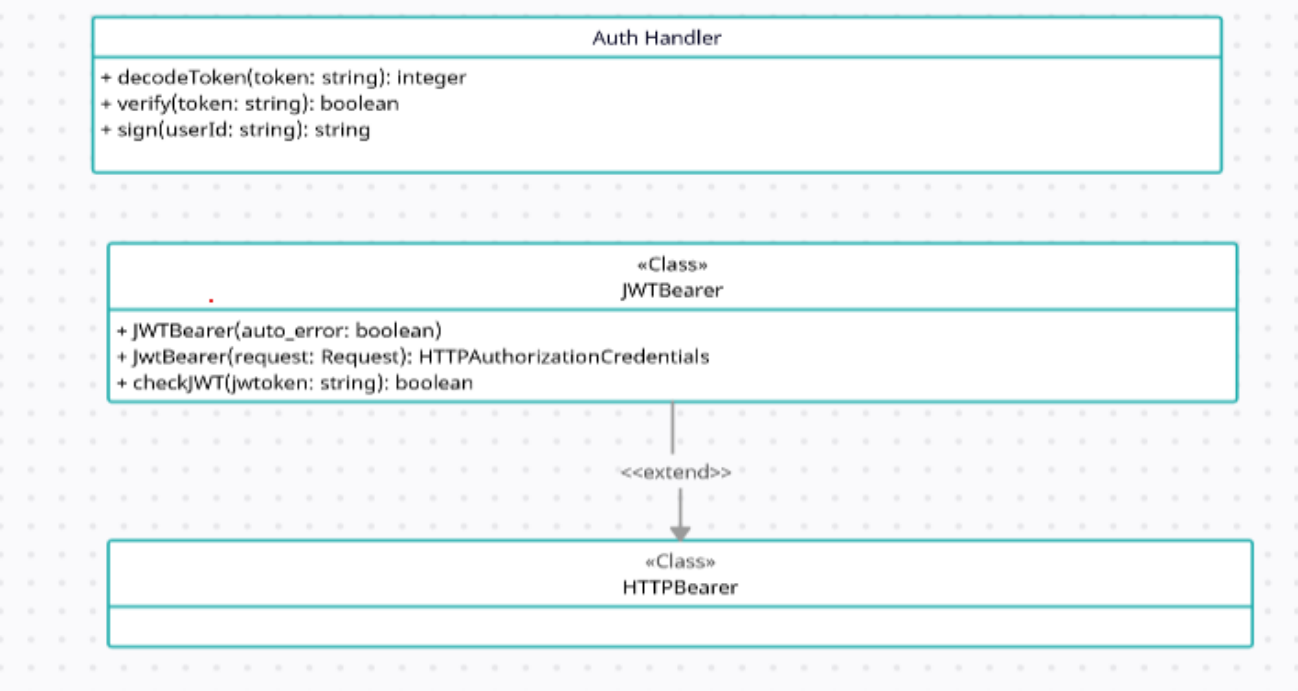
        - userid: Foreign key referencing the User table (integer, unique).
        - eventid: Foreign key referencing the Event table (integer, unique).
        - bookingdate: Timestamp of when the booking was made (datetime, server default).
        - price: Price of the ticket (float).
        - transactionid: ID of the transaction (string).
        - tickettype: Type of the ticket booked (string).
        - checkedin: Indicates if the user has checked-in for the event (boolean, default=False).
5) Partner Table:
    - Table Name: partner
    - Attributes:
        - id: Unique identifier for each partner (integer, primary key).
        - name: Name of the partner (string).
        - img: Image associated with the partner (string).
6) Ticket Type Table:
    - Table Name: tickettype
    - Attributes:
        - id: Unique identifier for each ticket type (integer, primary key).
        - eventid: Foreign key referencing the Event table (integer, unique).
        - name: Name of the ticket type (string).
        - price: Price of the ticket type (integer).
        - limit: Limit on the number of tickets available for this type (integer).
        - seated: Indicates if the ticket is for a seated event or not (Boolean).
        - seats: JSON field storing information about the available seats (JSON).

3.3 Entity-Relationship Diagram (ERD)



4.0 Project Provided Services

4.1 Authentication



Implementing authentication using JSON Web Tokens (JWT).

4.1.1 Structure of the authentication handler:

The authentication handler is responsible for three main functionalities:

1- Signing token: in which the user-id is injected in the JWT payload and encoded.
2- Token verification: decoding tokens and verifying whether the token has expired or not.
3- Token decoding: returning the user-id from the token payload to be used in authorization.

4.1.2 Structure of the token bearer:

A JWTBearer class is defined, and it inherits from HTTPBearer, which is a class provided by FastAPI for handling bearer token authentication.

When calling JWTBearer Class, the credentials are extracted from the request and validated.

Three different outcomes could be resolved:

1- Invalid authentication scheme
2- Invalid credentials due to invalid or expired token
3- Invalid authorization code

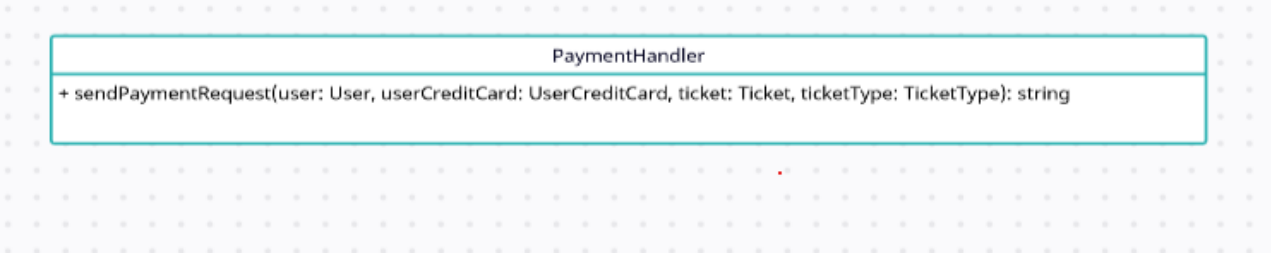
Passing these checks means that the provided credentials are valid and then the token within the credentials is returned.

JWTBearer class acts as a middleware and passes the token to a function to extract the user-id from the payload.

4.1.2 Payment gateway handling:

Constructs and sends the payment request to the **fawry** endpoint.

The payment data containing the user credit card info is sent in a request to the payment gateway which returns the transaction's auth number if successful, otherwise returns an empty string.



4.1.3 QR-code Generation:

A QR-code is generated for each ticket booking by a user. The qr-code image is generated, saved locally, loaded and then uploaded to cloundary cloud storage service.

5.0 Project Data Endpoints:

5.1 User Login-Signup endpoints:

URL	**/user/signup**
Request-Method	**POST**
Request-Body	**User Object**
Query-Params	**None**
Response-Format	**Json indicating correct user creation**
Authentication	**None**
DB-Operations	**Insert and create new user entity if not already exists**
Description	**Sign up endpoint to provide user registration**

URL	**/user/login**
Request-Method	**POST**
Request-Body	**User object containing only username and password**
Query-Params	**None**
Response-Format	**Json object containing user details, success message and created token**
Authentication	**None**
DB-Operations	**Selecting user from database to validate login process**
Description	**Login endpoint [roviding users to login with their credentials**

URL	**/user/verifyToken**
Request-Method	**POST**
Query-Params	**None**
Request-Body	**Token string**
Response-Format	**Json indicating either Valid or invalid authentication**

Authentication	**Verifies user's token**
DB-Operations	**None**
Description	**Checks the passed token using the authentication handler**

5.2 Admin endpoints:

URL	**/admin/event/event_id**
Request-Method	**GET**
Request-Body	**None**
Query-Params	**Event-Id**
Response-Format	**Json containing info of the event of the passed event id**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Select event details**
Description	**Returns the details of the event whom id is passed as query param and whose admin is the user included in the token**

URL	**/admin/event/statistics/event_id**
Request-Method	**GET**
Request-Body	**None**
Query-Params	**Event-Id**
Response-Format	**Json containing Statistics of the event of the passed event id**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Select event details, event ticket-Types details, Event tickets details, User Event Booking details and event admin details**
Description	**Returns the statistics of the event whom id is passed as query param and whose admin is the user included in the token**

URL	**/admin/event/event_id**
Request-Method	**PUT**
Request-Body	**Event object**
Query-Params	**Event-id**
Response-Format	**Json containing status of the event update indicating either success or failure**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Select the event and update its values**
Description	**Receives an Event object and uses its details to update its prior details if the event exists**

5.3 Dashboard endpoints:

URL	**/dashboard/**
Request-Method	**GET**

Request-Body	**None**
Query-Params	**None**
Response-Format	**Json containing information for the user Joined events, Upcoming events and past events**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Select all the events that the user Booked**
Description	**Select all the events that the user registered for either upcoming events or previous ones aside with Selection of some counters to be viewed in dashboard**

URL	**/dashboard/admin**
Request-Method	**GET**
Request-Body	**None**
Query-Params	**None**
Response-Format	**Json containing Dashboard Metrics for the admin including the Joined events, Upcoming events and past events**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Select all the events that the user Booked**
Description	**Select all the events that the user registered for either upcoming events or previous ones aside with Selection of some counters to be viewed in dashboard**

5.4 Contact Person endpoints:

URL	**/contactperson**
Request-Method	**POST**
Request-Body	**Contact Person Object**
Query-Params	**None**
Response-Format	**Json stating successful contact-person registration**
Authentication	**None**
DB-Operations	**Creating a contact person entity**
Description	**Creating a contact person entity based on the values of the passed contact person object**

URL	**/contactperson**
Request-Method	**GET**
Request-Body	**None**
Query-Params	**None**
Response-Format	**List of all contact person entities**
Authentication	**None**
DB-Operations	**Selecting all contact person entities**
Description	**Returning a list of all contact person entities**

URL	/contactperson/id
Request-Method	GET
Request-Body	None
Query-Params	Contact person id
Response-Format	Contact person object
Authentication	None
DB-Operations	Returning specified contact person entity
Description	Returning the contact person entity whom id is the id passed as query param

5.5 Partner endpoints:

URL	/partner
Request-Method	GET
Request-Body	None
Query-Params	None
Response-Format	List of all partner entities
Authentication	None
DB-Operations	Selecting all partner entities
Description	Returning a list of all partner entities

5.6 Ticket Endpoints:

URL	/ticket/verify
Request-Method	POST
Request-Body	Ticket object
Query-Params	None
Response-Format	Json validating user type and ticket check in
Authentication	Verifies user's token and extracts the user-id from the token
DB-Operations	Selecting the user entity and his corresponding ticket booking
Description	check the user type and fetch the booked ticket that matches the passed ticket and check if the ticket is checked in or not

URL	/ticket/type
Request-Method	POST
Request-Body	Ticket type object
Query-Params	None
Response-Format	Json indicating successful ticket type creation
Authentication	None
DB-Operations	Creating ticket type entity
Description	Creating a ticket type entity with the values of the passed object

URL	/ticket/type
Request-Method	GET
Request-Body	None
Query-Params	None
Response-Format	List of all ticket types
Authentication	None
DB-Operations	Selecting all ticket type entities
Description	Returning a list of all ticket type entities

5.7 Venue Endpoints:

URL	/venue
Request-Method	GET
Request-Body	None
Query-Params	None
Response-Format	List of all venue entities
Authentication	None
DB-Operations	Selecting all venue entities
Description	Returning a list of all venue entities

URL	/venue
Request-Method	POST
Request-Body	Venue object
Query-Params	None
Response-Format	Json indicating successful venue creation
Authentication	None
DB-Operations	Inserting new venue entity
Description	Creating new venue entity with the values of the passed object

URL	/venue/id
Request-Method	GET
Request-Body	None
Query-Params	Venue id
Response-Format	Venue object of specific id
Authentication	None
DB-Operations	Selecting venue of certain id
Description	Selecting venue whom id is passed as query param

URL	/venue/restriction
Request-Method	POST
Request-Body	Venue restriction object
Query-Params	None
Response-Format	Json indicating successful restriction creation
Authentication	None
DB-Operations	Creating venue restriction entity
Description	Creating new venue restriction entity with the values of the passed object

URL	/venue/restriction/id
Request-Method	GET
Request-Body	None
Query-Params	Venue id
Response-Format	List of all venue restrictions entities
Authentication	None
DB-Operations	Selecting all venue restrictions entities
Description	Returning all restriction entities of a certain venue whom id is passed as query param

URL	/venue/contact
Request-Method	POST
Request-Body	Venue contact object
Query-Params	None
Response-Format	Json indicating successful venue contact creation
Authentication	None
DB-Operations	Creating venue contact entity
Description	Creating new venue contact entity with the values of the passed object

URL	/venue/contact/id
Request-Method	GET
Request-Body	None
Query-Params	Venue id
Response-Format	List of all venue contact entities
Authentication	None
DB-Operations	Selecting all venue contact entities
Description	Returning all contact entities of a certain venue whom id is passed as query param

5.8 Event endpoints:

URL	/event
Request-Method	GET
Request-Body	None
Query-Params	None
Response-Format	List of all Events
Authentication	None
DB-Operations	Selecting all event entities
Description	Returning a list of all events each with its own details

URL	/event/app
Request-Method	GET
Request-Body	None
Query-Params	None
Response-Format	List of all Events for mobile app
Authentication	Verifies user's token and extracts the user-id from the token
DB-Operations	Selecting all events for mobile app
Description	Returning a list of all events. Each event is included with its relevant details for the mobile app

URL	/event/id
Request-Method	GET
Request-Body	None
Query-Params	Event id
Response-Format	Event of specific id
Authentication	None
DB-Operations	Selecting event entity details
Description	Returning event details if exists with its ticket types

URL	/event
Request-Method	POST
Request-Body	Event object
Query-Params	None
Response-Format	Json indicating successful event creation
Authentication	Verifies user's token and extracts the user-id from the token
DB-Operations	Inserting new event entity
Description	Creating new event entity and creating its own ticket types entities with the values of the passed object

5.9 Payment gateway:

URL	**/payment**
Request-Method	**POST**
Request-Body	**Payment Info object**
Query-Params	**None**
Response-Format	**Json containing ticket links**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Selecting the ticket having the provided transaction id and creating user event booking entity**
Description	**Selecting the booking whom transaction id is provided in the payment info and if the booking isn't yet recorder, a new entity is created containing some event id, user id, transaction number, price and ticket type. A qr code is generated from the booking id and sent via WhatsApp to the user phone number**

URL	**/payment/user**
Request-Method	**GET**
Request-Body	**None**
Query-Params	**None**
Response-Format	**All user booking entities**
Authentication	**Verifies user's token and extracts the user-id from the token**
DB-Operations	**Selecting all the event booking of specified user id**
Description	**Returns all the events booked by the user whom id is obtained from the token**