**Event Registration**

This document provides an overview of the components, structure, and testing of the project. The project focuses on event management, including user registration,user login, event creation, event registration,and cancel event.  
  
**1.Directory Structure**

The project follows a standard Django structure, organized into a main project directory and an app named "**registration\_app.**"

The project directory is organized as follows:

**event\_registration/**

**|-- registration\_app/**

| |-- \_\_init\_\_.py

| |-- admin.py

| |-- apps.py

| |-- migrations/

| |-- models.py

| |-- permissions.py

| |-- serializers.py

| |-- tests.py

| |-- utils.py

| |-- views.py

**|-- manage.py**

**|-- event\_registration/**

| |-- \_\_init\_\_.py

| |-- asgi.py

| |-- settings.py

| |-- urls.py

| |-- wsgi.py

**|-- templates/**

**|-- …**

**2.Database and Tables**The project utilizes a relational database to store information about users, events, and registrations. Django ORM (Object-Relational Mapping) is employed to interact with the database, providing a high-level and Pythonic interface.The default database engine configured in the project is SQLite, which is a lightweight, serverless, and file-based relational database.

**2.1 Tables**

**1. User Table**

**Model: User(Django User)**

**Fields:**

**username** (CharField): Represents the username of the user.

**email** (EmailField): Represents the email address of the user.

**first\_name** (CharField): Represents the first name of the user.

**last\_name** (CharField): Represents the last name of the user.

**password** (CharField): Represents the hashed password of the user.

**2. Event Table**

**Model: Event**

**Fields:**

**name** (CharField): Represents the name of the event.

**capacity** (IntegerField): Represents the maximum capacity of the event.

**valid\_until** (DateField): Represents the date until which the event registration is valid.

**created\_at** (DateTimeField): Represents the date & time when the event was created.

**updated\_at** (DateTimeField): Represents the datetime when the event was last updated.

**3. Registration Tabl**e

**Model: Registration**

**Fields:**

**user** (ForeignKey to User): Represents the user who is registered for the event.

**event** (ForeignKey to Event): Represents the event for which the user is registered.

**created\_at** (DateTimeField): Represents the date and time when the registration was created.

**updated\_at** (DateTimeField): Represents the date and time when the registration was last updated.

**cancelled** (BooleanField): Represents whether the registration is cancelled.

**2.2 Relationships**

**1.User-Event Relationship:**

The User and Event tables are related through the Registration table.

A user can register for multiple events, and an event can have multiple registrations.

**2.Event-Registration Relationship:**

The Event and Registration tables are related through the foreign key.

Each registration is associated with a specific event.

**2.3 Migrations**

Django migrations are used to manage changes to the database schema. Running **python manage.py makemigrations** creates migration files, and **python manage.py migrate** applies those changes to the database.

**3. API Endpoints**

**1.User Registration (/user-register/):**

Endpoint for user registration, allowing users to provide necessary information(check api doc).

Returns profile , access and refresh tokens upon successful registration.  
  
**2.User Login (/login/):**

Endpoint for user login, requiring a username and password.

Returns profile , access and refresh tokens upon successful login.  
  
**3.Token Refresh (/token/refresh/):**

Allows users to refresh their access token using the refresh token.  
  
**4.Event Listing (/events/):**

Retrieves and returns a list of events available for registration.  
  
**5.Event Detail (/events/<int:pk>/):**

Retrieves and returns detailed information about a specific event based on its ID.

**6.Event Registration (/event-register/):**

Allows authenticated users to register for events by providing the event ID.

Requires a valid JWT token for authentication.

**7.Event Creation (/create-event/):**

Allows authenticated admin users to create events by providing necessary information(check api doc).

Requires a valid JWT token with admin privileges.

**8.Cancel Event Registration (/cancel-event-registration/<int:pk>/):**

Allows users to cancel their own event registration.

Requires a valid JWT token for authentication.  
  
**9.Event Registration List (/all-event-registrations/):**

Retrieves and returns a list of event registrations for the authenticated user.

Requires a valid JWT token for authentication.

**4. Authentication**

* WT (JSON Web Token) authentication is implemented using the rest\_framework\_simplejwt library.
* Certain endpoints are restricted to authenticated users or admin users.

**5. Swagger Documentation**

Swagger documentation(/documentation) is integrated for API endpoints, providing a user-friendly interface to explore and test the APIs.

**6. Serialization**

* Django serializers are used to convert complex data types (models or querysets) to Python data types.
* Serializers like RegisterSerializer, LoginSerializer, EventSerializer, etc., are defined for different purposes.

**7. Permissions**

Custom permissions are implemented, such as IsAdminOrReadOnly, restricting certain actions to admin users.

**8. Event Registration Logic**

The logic for event registration and cancellation is well-implemented, considering event capacities and ownership.

**9. Code Comments and Documentation**

The code includes inline comments, which is beneficial for understanding the purpose of different sections and functions.

**10. Test-Driven Development (TDD)**

A test case is a set of conditions or variables under which a tester will determine whether a system or a specific feature of a system is working as intended. In software development, test cases are written to validate the correctness of code and ensure that it meets the specified requirements. They often consist of a setup, where necessary conditions are prepared, and one or more test methods that define specific scenarios to be tested. Running test cases helps identify and fix issues, ensuring the reliability and functionality of the software.

**10.1 Test cases:**

1. EventListAPITestCase
2. EventDetailAPITestCase
3. CreateEventAPITestCase
4. EventRegistrationAPIViewTestCase
5. CancelRegistrationAPITestCase
6. EventRegistrationListAPITestCase

**10.2 Running Test Cases:**

* To run the test cases, use the Django test runner with the command python manage.py test.
* The test runner will execute all the test cases and display the results, indicating successes and failures.

**11. Conclusion**

This project demonstrates a simple event management system with user registration, event creation, and event registration functionalities. It incorporates Django REST Framework for building APIs and includes comprehensive tests for robustness.