**Models Created for the Project:**

1. **Attendee Model**:
   * Stores basic attendee details like name, email, and phone number.
   * Ensures email addresses are unique to avoid duplication.
2. **Event Model**:
   * Includes all essential fields such as name, description, location, and date.
   * Establishes a Many-to-Many relationship with the Attendee model.
   * Tracks the user who created the event using a ForeignKey to the User model.
   * Contains timestamps to log when events are created or updated.
3. **Task Model**:
   * Features necessary fields such as name, deadline, and status.
   * Links to both the Event and Attendee models using ForeignKey relationships.
   * Includes status options like "Pending" and "Completed."
   * Tracks creation and update times.

**Implementation:**

1. Add 'dashboard' to the INSTALLED\_APPS list in settings.py.
2. Apply database migrations:

bash

Copy code

python manage.py makemigrations

python manage.py migrate

**API Features:**

1. **Serializers**:
   * AttendeeSerializer: Handles serialization for attendee data.
   * TaskSerializer: Includes the name of the assigned attendee for convenience.
   * EventSerializer: Serializes tasks and attendees with nested structures.
2. **ViewSets**:
   * **EventViewSet**:
     + Supports CRUD operations for events.
     + Allows adding/removing attendees through custom actions.
     + Filters events by the currently logged-in user.
   * **AttendeeViewSet**: Provides basic CRUD functionality for attendees.
   * **TaskViewSet**:
     + Supports CRUD operations for tasks.
     + Includes a custom action to update task statuses.
     + Filters tasks based on the event creator.

**Setup Instructions:**

1. Install the Django REST framework:

bash

Copy code

pip install djangorestframework

1. Add 'rest\_framework' to INSTALLED\_APPS in settings.py.

**URL Configuration:**

The following endpoints are configured:

1. **Event API**:
   * List, create, retrieve, update, and delete events (/api/events/).
   * Add or remove attendees from events using /add\_attendee/ and /remove\_attendee/.
2. **Attendee API**:
   * List, create, retrieve, update, and delete attendees (/api/attendees/).
3. **Task API**:
   * List, create, retrieve, update, and delete tasks (/api/tasks/).
   * Update task status with /update\_status/.

**Testing:**

You can test these endpoints using:

* Django's browsable API.
* Postman or curl commands, e.g.:

bash

Copy code

# List all events

curl -H "Authorization: Bearer your\_token" http://localhost:8000/api/events/

# Create a new event

curl -X POST -H "Authorization: Bearer your\_token" -H "Content-Type: application/json" \

-d '{"name":"Team Meeting","description":"Monthly team meeting","location":"Conference Room","date":"2024-12-25T10:00:00Z"}' \

http://localhost:8000/api/events/

**Templates Developed:**

1. **Base Template (base.html)**:
   * Includes a responsive navigation bar with authentication indicators.
   * Uses Tailwind CSS for styling.
   * Employs Axios for dynamic AJAX requests.
2. **Event List Template (event\_list.html)**:
   * Displays all events.
   * Allows adding, editing, and deleting events.
   * Features a modal for creating or editing events.
   * Implements dynamic rendering and CRUD functionality using JavaScript.
3. **Attendee Template (attendee\_list.html)**:
   * Lists all attendees with options to add, edit, or delete.
   * Includes a modal form for attendee management.
4. **Task Template (task\_list.html)**:
   * Lists all tasks and provides options to add, update, and delete them.
   * Includes a status toggle and modal form for selecting events and attendees.

**Implementation Steps:**

1. Create a templates directory within the dashboard app:

bash

Copy code

mkdir -p dashboard/templates/dashboard

1. Place templates in their respective locations:
   * base.html → templates/base.html
   * event\_list.html → templates/dashboard/event\_list.html
   * attendee\_list.html → templates/dashboard/attendee\_list.html
   * task\_list.html → templates/dashboard/task\_list.html
2. Update settings.py to configure template directories:

python

Copy code

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [BASE\_DIR / 'templates'],

'APP\_DIRS': True,

...

},

]

1. Add corresponding views in views.py:

python

Copy code

from django.shortcuts import render

from django.contrib.auth.decorators import login\_required

@login\_required

def attendee\_list(request):

return render(request, 'dashboard/attendee\_list.html')

@login\_required

def task\_list(request):

return render(request, 'dashboard/task\_list.html')

**Authentication & User Management:**

1. Set up authentication views, including login and logout functionality.
2. Update settings.py with the following login-related configurations:

python

Copy code

LOGIN\_URL = 'login'

LOGIN\_REDIRECT\_URL = 'event\_list'

LOGOUT\_REDIRECT\_URL = 'login'

1. Create a superuser for testing:

bash

Copy code

python manage.py createsuperuser

1. Apply migrations and start the server:

bash

Copy code

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

**Summary:**

With this setup, you now have a fully functional event management system that includes:

* User authentication and authorization.
* Management features for events, attendees, and tasks.
* A REST API for external integrations.
* Responsive and dynamic templates for a seamless user experience.