I've created the models according to your requirements. Here's a breakdown of the models:

1. Attendee model:
   * Basic information about attendees (name, email, phone)
   * Email is unique to prevent duplicates
2. Event model:
   * All required fields (name, description, location, date)
   * ManyToMany relationship with Attendees
   * ForeignKey to User for tracking who created the event
   * Timestamps for creation and updates
3. Task model:
   * All required fields (name, deadline, status)
   * ForeignKey relationships to both Event and Attendee
   * Status choices for Pending/Completed
   * Timestamps for tracking

To implement these models:

1. Make sure your settings.py has 'dashboard' in INSTALLED\_APPS
2. Create and run migrations:

bash

Copy

python manage.py makemigrations

python manage.py migrate

I've created:

1. **Serializers**:
   * AttendeeSerializer: Handles attendee data serialization
   * TaskSerializer: Includes the assigned attendee's name for convenience
   * EventSerializer: Includes nested serialization for tasks and attendees
2. **ViewSets**:
   * EventViewSet:
     + CRUD operations for events
     + Custom actions for adding/removing attendees
     + Filters events by the current user
   * AttendeeViewSet:
     + Basic CRUD operations for attendees
   * TaskViewSet:
     + CRUD operations for tasks
     + Custom action for updating task status
     + Filters tasks by the event creator

To implement these, you'll need to:

1. Install Django REST framework if you haven't:

bash

Copy

pip install djangorestframework

1. Add to INSTALLED\_APPS in settings.py:

python

Copy

INSTALLED\_APPS = [

...

'rest\_framework',

]

This URL configuration sets up the following API endpoints:

1. Events endpoints:
   * GET /api/events/ - List all events
   * POST /api/events/ - Create new event
   * GET /api/events/{id}/ - Retrieve specific event
   * PUT /api/events/{id}/ - Update specific event
   * DELETE /api/events/{id}/ - Delete specific event
   * POST /api/events/{id}/add\_attendee/ - Add attendee to event
   * POST /api/events/{id}/remove\_attendee/ - Remove attendee from event
2. Attendees endpoints:
   * GET /api/attendees/ - List all attendees
   * POST /api/attendees/ - Create new attendee
   * GET /api/attendees/{id}/ - Retrieve specific attendee
   * PUT /api/attendees/{id}/ - Update specific attendee
   * DELETE /api/attendees/{id}/ - Delete specific attendee
3. Tasks endpoints:
   * GET /api/tasks/ - List all tasks
   * POST /api/tasks/ - Create new task
   * GET /api/tasks/{id}/ - Retrieve specific task
   * PUT /api/tasks/{id}/ - Update specific task
   * DELETE /api/tasks/{id}/ - Delete specific task
   * PATCH /api/tasks/{id}/update\_status/ - Update task status

You can test these endpoints using tools like:

1. The Django REST framework browsable API interface (available at each endpoint when you visit in a browser)
2. Postman
3. curl commands

Example curl commands for testing:

bash

Copy

*# List all events*

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

*# Create 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/

I've created the base template and the events list page with the following features:

1. Base Template (base.html):
   * Responsive navigation bar
   * User authentication status
   * Tailwind CSS for styling
   * Axios for AJAX requests
   * Common layout structure
2. Events List Page (event\_list.html):
   * List of all events
   * Add new event functionality
   * Edit and delete capabilities
   * Modal form for creating/editing events
   * Dynamic rendering using JavaScript
   * CRUD operations using the REST API

To implement these templates:

1. Create a templates directory in your project:

bash

Copy

mkdir -p dashboard/templates/dashboard

1. Copy these templates into their respective locations:
   * base.html → templates/base.html
   * event\_list.html → templates/dashboard/event\_list.html
2. Update your settings.py to include the templates directory:

python

Copy

TEMPLATES = [

{

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

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

'APP\_DIRS': True,

...

},

]

I've created both the attendee and task management templates with full functionality. These templates include:

1. Attendee Management (attendee\_list.html):
   * List all attendees
   * Add new attendees
   * Edit existing attendees
   * Delete attendees
   * Modal form for creating/editing
2. Task Management (task\_list.html):
   * List all tasks
   * Add new tasks
   * Update task status
   * Delete tasks
   * Modal form with event and attendee selection
   * Status toggle functionality

To implement these templates:

1. Place them in your templates directory:

bash

Copy

cp attendee\_list.html task\_list.html dashboard/templates/dashboard/

1. Update your views.py to render these templates:

python

Copy

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')

This setup provides:

1. View Functions:
   * Event list view
   * Attendee list view
   * Task list view
   * Login view with authentication
   * Logout view
2. URL Configuration:
   * API endpoints (/api/...)
   * Template views (/, /attendees/, /tasks/)
   * Authentication URLs (/login/, /logout/)
3. Login Template:
   * Clean, responsive design
   * Error message display
   * CSRF protection
   * Form validation

To implement this:

1. Update your project's settings.py to include login-related settings:

python

Copy

LOGIN\_URL = 'login'

LOGIN\_REDIRECT\_URL = 'event\_list'

LOGOUT\_REDIRECT\_URL = 'login'

1. Create a superuser to test the login:

bash

Copy

python manage.py createsuperuser

1. Run migrations if you haven't:

bash

Copy

python manage.py makemigrations

python manage.py migrate

1. Start the development server:

bash

Copy

python manage.py runserver

Now you should have a fully functioning event management system with:

* User authentication
* Event management
* Attendee management
* Task management
* RESTful API endpoints
* Responsive UI