Project Write-Up: Causes API Development

Objective:
This project involved building a RESTful API to manage social causes and contributions, enabling users to create, view, update, and delete causes, as well as contribute to specific causes. The API was developed using Django Rest Framework (DRF) and MySQL, with JWT authentication for secure user interactions.

Technologies Used:

- **Django & DRF**: Framework for building the API, serialization, and view handling.
- **MySQL**: Database for storing causes and contributions.
- **JWT Authentication**: Token-based authentication for secure API access.
- **Postman**: For testing and validating API endpoints.

Key Features:

- **User Registration**: A simple user registration API that creates a new user.
- **Cause Management**: API endpoints for creating, updating, retrieving, and deleting causes.
- **Contributions**: Users can contribute to a cause by providing their name, email, and amount.

Key API Endpoints:

1. **POST /api/causes/**:
   Create a new cause. Fields required: title, description, image URL.
2. **GET /api/causes/**:
   Retrieve all causes. No parameters required.
3. **GET /api/causes/{id}/**:
   Retrieve a specific cause by ID. Replace {id} with the cause ID.
4. **PUT /api/causes/{id}/**:
   Update a specific cause by ID. Replace {id} with the cause ID and provide updated fields (title, description, image URL).
5. **DELETE /api/causes/{id}/**:
   Delete a specific cause by ID. Replace {id} with the cause ID.
6. **POST /api/causes/{id}/contribute/**:
   Accept contributions to a cause. Fields required: name, email, amount. Replace {id} with the cause ID.
7. **POST /api/register/**:
   Register a new user with fields: username and password.
8. **POST /api/token/**:
   Obtain a JWT token by passing user credentials (username, password).

9. **POST /api/token/refresh/**:
   Refresh the JWT token.

User Registration & Authentication:

1. **POST /api/register/**
   - **Description**: Register a new user.
   - **Request Body**:

     ```
     {
       "username": "john_doe",
       "password": "password123"
     }
     ```

   - **Response**:

     ```
     {
       "message": "User created successfully"
     }
     ```

2. **POST /api/token/**
   - **Description**: Obtain JWT token by providing username and password.
   - **Request Body**:

     ```
     {
       "username": "john_doe",
       "password": "password123"
     }
     ```

   - **Response**:

     ```
     {
       "access": "your_access_token",
       "refresh": "your_refresh_token"
     }
     ```

Cause Management:

3. **POST /api/causes/**
   - **Description**: Create a new cause.
   - **Request Body**:

     ```
     {
       "title": "Save the Trees",
       "description": "A cause to protect the environment",
     ```

```
  "image_url": "http://example.com/image.jpg"
}
```

- **Response**:

  ```
  {
    "id": 1,
    "title": "Save the Trees",
    "description": "A cause to protect the environment",
    "image_url": "http://example.com/image.jpg"
  }
  ```

4. **GET /api/causes/**
   - **Description**: Retrieve a list of all causes.
   - **Response**:

     ```
     [
       {
         "id": 1,
         "title": "Save the Trees",
         "description": "A cause to protect the environment",
         "image_url": "http://example.com/image.jpg"
       },
       {
         "id": 2,
         "title": "Save the Oceans",
         "description": "A cause to protect the oceans",
         "image_url": "http://example.com/ocean_image.jpg"
       }
     ]
     ```

5. **GET /api/causes/{id}/**
   - **Description**: Retrieve a specific cause by ID.
   - **Request Parameters**: {id} (replace with actual cause ID)
   - **Response**:

     ```
     {
       "id": 1,
       "title": "Save the Trees",
       "description": "A cause to protect the environment",
       "image_url": "http://example.com/image.jpg"
     }
     ```

6. **PUT /api/causes/{id}/**
   - **Description**: Update a specific cause by ID.
   - **Request Parameters**: {id} (replace with actual cause ID)

   - **Request Body**:

     ```
     {
       "title": "Protect the Forests",
       "description": "A new initiative to protect forests",
       "image_url": "http://example.com/new_image.jpg"
     }
     ```

   - **Response**:

     ```
     {
       "id": 1,
       "title": "Protect the Forests",
       "description": "A new initiative to protect forests",
       "image_url": "http://example.com/new_image.jpg"
     }
     ```

7. **DELETE /api/causes/{id}/**
   - **Description**: Delete a specific cause by ID.
   - **Request Parameters**: {id} (replace with actual cause ID)
   - **Response**:

     ```
     {
       "message": "Cause deleted successfully"
     }
     ```

Contribution Management:

8. **POST /api/causes/{id}/contribute/**
   - **Description**: Contribute to a specific cause by providing a name, email, and amount.
   - **Request Parameters**: {id} (replace with actual cause ID)
   - **Request Body**:

     ```
     {
       "name": "John Doe",
       "email": "john.doe@example.com",
       "amount": 100.00
     }
     ```

   - **Response**:

     ```
     {
       "message": "Contribution added successfully",
       "cause_id": 1,
     ```

```
  "amount": 100.00
}
```

9. **GET /api/causes/{id}/contributions/**
   - **Description**: Retrieve all contributions for a specific cause by ID.
   - **Request Parameters**: {id} (replace with actual cause ID)
   - **Response**:

```
[
  {
    "name": "John Doe",
    "email": "john.doe@example.com",
    "amount": 100.00
  },
  {
    "name": "Jane Smith",
    "email": "jane.smith@example.com",
    "amount": 50.00
  }
]
```

10. **GET /api/contributions/**
    - **Description**: Retrieve all contributions across all causes.
    - **Response**:

```
[
  {
    "cause_id": 1,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "amount": 100.00
  },
  {
    "cause_id": 2,
    "name": "Jane Smith",
    "email": "jane.smith@example.com",
    "amount": 50.00
  }
]
```

How to Use the API

1. **Registration:**

    - Send a POST request to /api/register/ with the username and password to register a new user.
2. **Authentication:**
    - Obtain a JWT token by sending a POST request to /api/token/ with the username and password. The token is used for authenticating subsequent requests.
3. **Cause Operations:**
    - **Create a Cause**: Send a POST request to /api/causes/ with the required data (title, description, image URL).
    - **Retrieve All Causes**: Send a GET request to /api/causes/ to get a list of all causes.
    - **Retrieve a Specific Cause**: Send a GET request to /api/causes/{id}/ (replace {id} with the actual cause ID).
    - **Update a Cause**: Send a PUT request to /api/causes/{id}/ with the updated data.
    - **Delete a Cause**: Send a DELETE request to /api/causes/{id}/.
4. **Contribute to a Cause:**
    - Send a POST request to /api/causes/{id}/contribute/ with the name, email, and amount for contributing to a cause.
    - Retrieve contributions for a cause by sending a GET request to /api/causes/{id}/contributions/.

Approach & Solution:

The API is structured to provide users with an intuitive interface for interacting with causes and contributions. It integrates user authentication, cause management, and contribution tracking, all wrapped into a secure, easy-to-use platform for social good.

- **Data Modeling**: The Cause and Contribution models are used to structure the database and represent the core entities of the application.
- **Authentication**: JWT is used for secure user authentication. The /api/token/ endpoint generates access tokens that are required for accessing protected resources, such as contributing to causes.
- **Error Handling**: Each endpoint handles errors gracefully by returning appropriate status codes and messages (e.g., 400 for bad requests, 404 for not found).

Challenges Faced:

- **Token Authentication**: Initially, there were challenges in getting the token to work properly with authentication. After resolving issues with token expiration and renewal, the implementation became more stable.
- **Data Validation**: Ensuring that only valid data (positive contribution amounts) was allowed required extra validation logic.

- **Model Relationships**: Properly linking contributions to causes in the database was a key challenge. Django's ORM made this manageable once the relationships were defined clearly in the models.

Future Considerations:

- **Logging**: Future improvements could include adding logging to monitor activity and errors more efficiently.
- **Rate-Limiting**: Implementing rate-limiting for the contribute endpoint could prevent abuse or accidental spam contributions.
- **Pagination**: Adding pagination for the GET /causes endpoint would be useful if the API scales with many causes.

Conclusion:

This API project enabled the development of a functional system for managing social causes and contributions, leveraging Django and Django Rest Framework. It handles user authentication, CRUD operations for causes, and allows for users to contribute to causes, with a simple and robust architecture.