Social Network API

This project is a **Django Rest Framework** (DRF) based API for a social networking application. It includes features like user authentication, friend requests, user blocking, user activity logs, and search functionality. The project is fully dockerized for easy deployment and scalability.

Table of Contents:

Installation Steps:

Follow these steps to set up and run the project locally using Docker.

Prerequisites

- **Docker**: Ensure that Docker and Docker Compose are installed on your machine.
- **Git**: Clone the project repository.

Steps:

1. **Clone the Repository**:

   git clone https://github.com/your-repository/social-network-api.git

   cd social-network-api

2. **Set Up Environment Variables**: Create a .env file in the root directory with the following content:

   MYSQL_DATABASE=social_network_db

   MYSQL_ROOT_PASSWORD=your_password

   SQL_HOST=db

   SQL_PORT=3306

3. **Run Docker**: Use Docker Compose to build and run the containers.

   docker-compose up --build

4. **Run Database Migrations**: In a new terminal window, run:

   docker-compose exec web python manage.py migrate

5. **Create a Superuser** (optional): To access the Django admin panel, create a superuser:

   docker-compose exec web python manage.py createsuperuser

6. **Access the Application**: The API will be available at:

   http://localhost:8000

API Documentation

Authentication & JWT Token Management

User Signup

- **URL**: http://127.0.0.1:8000/api/signup/
- **Method**: POST
- **Description**: Registers a new user with username, email, and password.

- **Request Body**:

Json

```
{
  "username": "newuser",
  "email": "newuser@example.com",
  "password": "your_password"
}
```

- **Response**: 201 Created, JSON with user details.

User Login

- **URL**: http://127.0.0.1:8000/api/login/
- **Method**: POST
- **Description**: Logs in the user using username and password.
- **Request Body**:

Json

```
{
  "username": "newuser",
  "password": "your_password"
}
```

- **Response**: 200 OK, returns JWT access and refresh tokens.

JWT Token Management

- **URL**: /token/ (Obtain Tokens), /token/refresh/ (Refresh Tokens)
- **Method**: POST
- **Description**: Manages JWT tokens for authentication.
- **Response**: Access and refresh tokens.

Core Functionalities

User Search

- **URL**: http://127.0.0.1:8000/api/search/?q=<username_or_email>
- **Method**: GET
- **Description**: Search for users by username or email.
- **Authentication**: Required (JWT token).
- **Response**: Paginated list of users matching the query.

Friend Requests

1. **Send Friend Request**
    - **URL**: http://127.0.0.1:8000/api/send-friend-request/
    - **Method**: POST
    - **Request Body**:

Json

```
{
 "to_user": <id_of_user>
}
```

    - **Description**: Sends a friend request to another user.
    - **Authentication**: Required (JWT token).
2. **Accept/Reject Friend Request**
    - **URL**: http://127.0.0.1:8000/api/accept-friend-request/<int:id>/ (or) /reject-friend-request/<int:id>/
    - **Method**: PATCH
    - **Description**: Accept or reject a received friend request.

Blocking Users

- **URL**: http://127.0.0.1:8000/api/block-user/
- **Method**: POST
- **Description**: Block a user from sending further requests or viewing the profile.

- **Request Body**:

json

```
{
 "blocked": <id_of_user>
}
```

Friends List

- **URL**: http://127.0.0.1:8000/api/friends/
- **Method**: GET
- **Description**: Lists all the accepted friends of the logged-in user.

Pending Friend Requests

- **URL**: http://127.0.0.1:8000/api/pending-requests/
- **Method**: GET
- **Description**: Lists all the pending friend requests the user has received.

User Activities

- **URL**: http://127.0.0.1:8000/api/activities/
- **Method**: GET
- **Description**: Retrieves the activity logs for the logged-in user (e.g., friend requests sent/accepted).

Design Choices

1. **JWT Authentication**:
    - JWT ensures secure and stateless user authentication. Access tokens are short-lived, and refresh tokens can be used to re-authenticate without asking for credentials.

2. Rate Limiting:

- Rate limiting is implemented to prevent brute-force attacks on sensitive endpoints like login and friend requests. This ensures security and protects against spamming.

3. Database:

- MySQL is used as the database in this project for relational data storage. The API structure allows for horizontal scaling by design.

4. **Caching**:

- Redis is used to cache frequently accessed data, such as friend lists, to reduce database load and enhance API response times.

5. **Friend Requests**:

- Friend request management includes sending, accepting, and rejecting friend requests with rate limiting to avoid spamming.

6. **Blocking and Privacy**:

- The app allows users to block others, preventing friend requests and profile visibility. This ensures user privacy and control over social interactions.

7. **User Activity Logging**:

- Each user action, such as sending a friend request or blocking someone, is logged in the UserActivity model, making the system auditable and providing transparency to users.

8. **Scalability**:

- Dockerization allows the application to be deployed easily on any cloud platform with multiple containers (e.g., web, database, and cache).

Technologies Used

- **Backend Framework**: Django, Django Rest Framework (DRF)
- **Database**: MySQL
- **Caching**: Redis
- **Authentication**: JWT (JSON Web Token)
- **Rate Limiting**: Django Ratelimit
- **Containerization**: Docker, Docker Compose
- **Security**: Encrypted user data, rate limiting, role-based access control (RBAC)