OAuth2 Authorization Server – Maintained Project Documentation

This documentation provides a complete guide for the Django OAuth2 + REST Framework project. It includes setup steps, authentication flows, API usage, OAuth2 server details, security guidelines, and the overall project structure. The goal is to ensure developers and integrators can easily understand, set up, and extend the system.

Setup Instructions

1. Clone the repository:
git clone <repo_url>

2. Create and activate a virtual environment:
python -m venv venv && source venv/bin/activate

3. Install dependencies:
pip install -r requirements.txt

4. Run migrations:
python manage.py migrate

5. (Optional) Seed demo users:
python manage.py seed_demo_users

6. (Optional) Create OAuth client:
python manage.py create_oauth_app

7. Start the server:
python manage.py runserver

Authentication Flows

1. Username/Password Login
Clients can generate tokens directly using username and password:
curl --location 'http://127.0.0.1:8000/api/token/password/' \ --header 'X-Service-Key: <SERVICE_KEY>' \ --form 'username=<USERNAME>' \ --form 'password=<PASSWORD>'

2. OTP-Based Login
Step 1: Request OTP
curl --location 'http://127.0.0.1:8000/api/request-otp/' \ --form 'username=<USERNAME>'

Step 2: Exchange OTP for Token
curl --location 'http://127.0.0.1:8000/api/token/otp/' \ --header 'X-Service-Key: <SERVICE_KEY>' \ --form 'username=<USERNAME>' \ --form 'otp=<OTP>'

Sample APIs (Django REST Framework)

1. Fetch user profile:
GET /api/profile/

2. Update user profile:
PUT /api/profile/

These endpoints demonstrate how authenticated requests can be used to fetch and update data.

Notes

- This project uses Django, Django REST Framework, and JWT authentication.
- Both Username/Password and OTP login flows are supported.
- External integrators do not need client_id and client_secret.
- Architecture is modular, extensible, and production-ready.
- Secure practices (HTTPS, CSRF protection, token expiry, refresh tokens) are recommended.

OAuth2 Authorization Server Details

1. Added oauth2_provider to INSTALLED_APPS.
2. Added o/ routes in urls.py.
3. Run migrations to create required tables.
4. Management command create_oauth_app provisions a default client.

Implemented OAuth2 Flows
- Authorization Code Grant with PKCE.
- Refresh Token flow.
- Confidential clients can exchange tokens securely.

Token-Based Authentication
- Tokens issued at /o/token/ include access & refresh tokens.
- Expiry configurable in .env.
- Refresh tokens are rotated for better security.

Secure Sessions & User Roles
- Custom User model extends AbstractUser with role field.
- Sessions secured with CSRF, HSTS, and secure cookies.
- Roles API available for access control.

Provided APIs
- /o/token/: Token issuance
- /api/userinfo/: Get user info
- /api/logout/: Logout
- /api/roles/: Fetch roles
- /api/validate-token/: Validate token

Setup & Client Integration

Server setup:
```
git clone <repo_url>
cd final
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
python manage.py migrate
python manage.py seed_demo_users
python manage.py create_oauth_app
python manage.py runserver
```

Demo Users
- admin / adminpass
- alice / alicepass

React Client Setup

```
cd react-client
npm install
npm start
```

Security Best Practices

- PKCE enforced for SPA clients.
- HTTPS must be used in production.
- Rotate refresh tokens frequently.
- Manage secrets via .env.
- CSRF middleware enabled.

Project File Structure

```
Oauth-Project/
■■■ auth_server/
■ ■■■ settings.py
■ ■■■ urls.py
■ ■■■ wsgi.py
■■■ users/
■ ■■■ models.py
■ ■■■ views.py
■ ■■■ management/commands/create_oauth_app.py
■■■ client_backend/
■ ■■■ views.py
■■■ react-client/
■ ■■■ src/
■ ■■■ App.js
■ ■■■ config.js
■■■ manage.py
■■■ requirements.txt
■■■ README.md
```