User Authentication System

A robust Django REST Framework-based authentication service with JWT tokens, comprehensive security features, and production-ready deployment capabilities.

🚀 Features

* \*\*JWT Authentication\*\*: Secure token-based authentication with refresh tokens
* \*\*User Management\*\*: Registration, login, profile management
* \*\*Password Reset\*\*: Secure password reset with token validation
* \*\*Database\*\*: PostgreSQL for reliable data persistence
* \*\*Caching\*\*: Redis integration with Django cache fallback
* \*\*Rate Limiting\*\*: Protection against brute force attacks
* \*\*API Documentation\*\*: Interactive Swagger/OpenAPI documentation
* \*\*Testing\*\*: Comprehensive unit test coverage
* \*\*Docker Support\*\*: Containerized development and deployment
* \*\*Security\*\*: Production-ready security configurations
* \*\*Deployment Ready\*\*: Configured for Railway, Render, and other platforms

📋 Table of Contents

* [Quick Start](#quick-start)
* [Setup Instructions](#setup-instructions)

- [Docker Setup (Recommended)](#docker-setup-recommended)

- [Local Development](#local-development)

- [Environment Variables](#environment-variables)

* [API Documentation](#api-documentation)

- [Authentication Endpoints](#authentication-endpoints)

- [User Profile Endpoints](#user-profile-endpoints)

- [Password Reset Endpoints](#password-reset-endpoints)

* [Testing](#testing)
* [Security Features](#security-features)
* [Deployment](#deployment)
* [Project Structure](#project-structure)
* [Contributing](#contributing)

⚡ Quick Start

Get the service running in under 5 minutes with Docker:

# Clone the repository  
  
git clone https://github.com/Iduate/User-Authentication-System.git  
  
cd User-Authentication-System/auth\_service  
  
  
  
# Start with Docker Compose  
  
docker-compose up -d  
  
  
  
# Access the API  
  
# - API Base URL: http://localhost:8000/api/v1/  
  
# - API Documentation: http://localhost:8000/swagger/  
  
# - Admin Panel: http://localhost:8000/admin/

🛠 Setup Instructions

Docker Setup (Recommended)

Docker provides the easiest way to get the service up and running with all dependencies.

\*\*Prerequisites:\*\*

* Docker Desktop installed
* Git installed

\*\*Steps:\*\*

1. \*\*Clone the repository:\*\*

```bash

git clone https://github.com/Iduate/User-Authentication-System.git

cd User-Authentication-System/auth\_service

```

1. \*\*Choose your Docker setup:\*\*

\*\*Option A: Full Docker Build (Recommended for development)\*\*

```bash

docker-compose up -d

```

\*\*Option B: Simple Docker Setup (Using pre-built images)\*\*

```bash

docker-compose -f docker-compose.simple.yml up -d

```

1. \*\*Verify the setup:\*\*

```bash

# Check if containers are running

docker-compose ps

# View logs

docker-compose logs -f web

```

1. \*\*Access the service:\*\*

- \*\*API Base URL\*\*: http://localhost:8000/api/v1/

- \*\*API Documentation\*\*: http://localhost:8000/swagger/

- \*\*Admin Panel\*\*: http://localhost:8000/admin/

1. \*\*Create a superuser (optional):\*\*

```bash

docker-compose exec web python manage.py createsuperuser

```

1. \*\*Stop the services:\*\*

```bash

docker-compose down

```

Local Development (Without Docker)

For development without Docker, you'll need to set up the environment manually.

\*\*Prerequisites:\*\*

* Python 3.11+ installed
* PostgreSQL 12+ installed and running
* Redis installed and running (optional, will fallback to Django cache)
* Git installed

\*\*Steps:\*\*

1. \*\*Clone and setup virtual environment:\*\*

```bash

git clone https://github.com/Iduate/User-Authentication-System.git

cd User-Authentication-System/auth\_service

# Create virtual environment

python -m venv venv

# Activate virtual environment

# On Windows:

venv\Scripts\activate

# On macOS/Linux:

source venv/bin/activate

```

1. \*\*Install dependencies:\*\*

```bash

pip install -r requirements.txt

```

1. \*\*Setup environment variables:\*\*

Create a .env file in the project root:

```env

# Django settings

SECRET\_KEY=your-super-secret-key-here

DEBUG=True

ALLOWED\_HOSTS=localhost,127.0.0.1

# Database configuration

DATABASE\_NAME=auth\_db

DATABASE\_USER=postgres

DATABASE\_PASSWORD=your-db-password

DATABASE\_HOST=localhost

DATABASE\_PORT=5432

# Redis configuration (optional)

REDIS\_URL=redis://localhost:6379/0

# JWT settings

JWT\_ACCESS\_TOKEN\_LIFETIME\_MINUTES=60

JWT\_REFRESH\_TOKEN\_LIFETIME\_DAYS=7

```

1. \*\*Setup PostgreSQL database:\*\*

```bash

# Connect to PostgreSQL as superuser

psql -U postgres

# Create database and user

CREATE DATABASE auth\_db;

CREATE USER auth\_user WITH PASSWORD 'your-db-password';

GRANT ALL PRIVILEGES ON DATABASE auth\_db TO auth\_user;

\q

```

1. \*\*Run database migrations:\*\*

```bash

python manage.py migrate

```

1. \*\*Create superuser:\*\*

```bash

python manage.py createsuperuser

```

1. \*\*Start development server:\*\*

```bash

python manage.py runserver

```

1. \*\*Access the service:\*\*

- \*\*API Base URL\*\*: http://localhost:8000/api/v1/

- \*\*API Documentation\*\*: http://localhost:8000/swagger/

- \*\*Admin Panel\*\*: http://localhost:8000/admin/

Environment Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Default** | **Required** |
| `SECRET\_KEY` | Django secret key | - | ✅ |
| `DEBUG` | Enable debug mode | `False` | ❌ |
| `ALLOWED\_HOSTS` | Comma-separated allowed hosts | `localhost` | ❌ |
| `DATABASE\_URL` | PostgreSQL connection string | - | ✅ |
| `DATABASE\_NAME` | Database name | `auth\_db` | ✅ |
| `DATABASE\_USER` | Database user | `postgres` | ✅ |
| `DATABASE\_PASSWORD` | Database password | - | ✅ |
| `DATABASE\_HOST` | Database host | `localhost` | ✅ |
| `DATABASE\_PORT` | Database port | `5432` | ✅ |
| `REDIS\_URL` | Redis connection string | - | ❌ |
| `JWT\_ACCESS\_TOKEN\_LIFETIME\_MINUTES` | Access token lifetime | `60` | ❌ |
| `JWT\_REFRESH\_TOKEN\_LIFETIME\_DAYS` | Refresh token lifetime | `7` | ❌ |

📚 API Documentation

The API follows RESTful principles and provides comprehensive authentication and user management functionality.

\*\*Base URL\*\*: http://localhost:8000/api/v1/

\*\*Interactive Documentation\*\*:

* Swagger UI: `http://localhost:8000/swagger/`
* ReDoc: `http://localhost:8000/redoc/`

Authentication Endpoints

🔐 User Registration

\*\*Endpoint\*\*: POST /api/v1/users/register/

Register a new user account.

\*\*Request Body\*\*:

{  
  
 "email": "user@example.com",  
  
 "full\_name": "John Doe",  
  
 "password": "SecurePassword123!",  
  
 "password\_confirm": "SecurePassword123!"  
  
}

\*\*Response\*\* (201 Created):

{  
  
 "user": {  
  
 "id": 1,  
  
 "email": "user@example.com",  
  
 "full\_name": "John Doe",  
  
 "date\_joined": "2025-08-29T10:30:00Z"  
  
 },  
  
 "tokens": {  
  
 "access": "eyJ0eXAiOiJKV1QiLCJhbGc...",  
  
 "refresh": "eyJ0eXAiOiJKV1QiLCJhbGc..."  
  
 },  
  
 "message": "User registered successfully"  
  
}

\*\*Alternative Endpoint\*\*: POST /api/v1/register/

---

🔑 User Login

\*\*Endpoint\*\*: POST /api/v1/users/login/

Authenticate user and receive JWT tokens.

\*\*Request Body\*\*:

{  
  
 "email": "user@example.com",  
  
 "password": "SecurePassword123!"  
  
}

\*\*Response\*\* (200 OK):

{  
  
 "user": {  
  
 "id": 1,  
  
 "email": "user@example.com",  
  
 "full\_name": "John Doe"  
  
 },  
  
 "tokens": {  
  
 "access": "eyJ0eXAiOiJKV1QiLCJhbGc...",  
  
 "refresh": "eyJ0eXAiOiJKV1QiLCJhbGc..."  
  
 },  
  
 "message": "Login successful"  
  
}

\*\*Rate Limit\*\*: 5 requests per minute per IP

\*\*Alternative Endpoint\*\*: POST /api/v1/login/

---

🔄 Token Refresh

\*\*Endpoint\*\*: POST /api/v1/users/token/refresh/

Refresh access token using refresh token.

\*\*Request Body\*\*:

{  
  
 "refresh": "eyJ0eXAiOiJKV1QiLCJhbGc..."  
  
}

\*\*Response\*\* (200 OK):

{  
  
 "access": "eyJ0eXAiOiJKV1QiLCJhbGc..."  
  
}

User Profile Endpoints

👤 Get User Profile

\*\*Endpoint\*\*: GET /api/v1/users/profile/

Get authenticated user's profile information.

\*\*Headers\*\*:

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGc...

\*\*Response\*\* (200 OK):

{  
  
 "id": 1,  
  
 "email": "user@example.com",  
  
 "full\_name": "John Doe",  
  
 "date\_joined": "2025-08-29T10:30:00Z",  
  
 "last\_login": "2025-08-29T15:45:00Z"  
  
}

\*\*Alternative Endpoint\*\*: GET /api/v1/profile/

Password Reset Endpoints

📧 Request Password Reset

\*\*Endpoint\*\*: POST /api/v1/users/password-reset/

Request a password reset token via email.

\*\*Request Body\*\*:

{  
  
 "email": "user@example.com"  
  
}

\*\*Response\*\* (200 OK):

{  
  
 "message": "If an account with this email exists, a password reset link has been sent."  
  
}

\*\*Rate Limit\*\*: 3 requests per hour per IP

\*\*Alternative Endpoint\*\*: POST /api/v1/password-reset/

---

✅ Confirm Password Reset

\*\*Endpoint\*\*: POST /api/v1/users/password-reset/confirm/

Reset password using the token received via email.

\*\*Request Body\*\*:

{  
  
 "token": "abc123def456ghi789",  
  
 "new\_password": "NewSecurePassword123!",  
  
 "new\_password\_confirm": "NewSecurePassword123!"  
  
}

\*\*Response\*\* (200 OK):

{  
  
 "message": "Password has been reset successfully"  
  
}

\*\*Alternative Endpoint\*\*: POST /api/v1/password-reset/confirm/

Error Responses

All endpoints return consistent error responses:

\*\*400 Bad Request\*\*:

{  
  
 "errors": {  
  
 "email": ["This field is required."],  
  
 "password": ["Password must be at least 8 characters long."]  
  
 }  
  
}

\*\*401 Unauthorized\*\*:

{  
  
 "detail": "Authentication credentials were not provided."  
  
}

\*\*429 Too Many Requests\*\*:

{  
  
 "detail": "Request was throttled. Expected available in 45 seconds."  
  
}

\*\*500 Internal Server Error\*\*:

{  
  
 "detail": "A server error occurred."  
  
}

🧪 Testing

The project includes comprehensive test coverage for all major functionality.

Running All Tests

\*\*With Docker\*\*:

# Run all tests  
  
docker-compose exec web python manage.py test users  
  
  
  
# Run with verbose output  
  
docker-compose exec web python manage.py test users -v 2  
  
  
  
# Run with coverage report  
  
docker-compose exec web coverage run --source='.' manage.py test users  
  
docker-compose exec web coverage report

\*\*Without Docker\*\*:

# Activate virtual environment first  
  
source venv/bin/activate # On Windows: venv\Scripts\activate  
  
  
  
# Run all tests  
  
python manage.py test users  
  
  
  
# Run with verbose output  
  
python manage.py test users -v 2

Running Specific Test Modules

# Registration functionality tests  
  
python manage.py test users.tests.test\_registration  
  
  
  
# Login functionality tests  
  
python manage.py test users.tests.test\_login  
  
  
  
# Password reset functionality tests  
  
python manage.py test users.tests.test\_password\_reset

Test Scripts

\*\*Unix/Linux/macOS\*\*:

chmod +x run\_tests.sh  
  
./run\_tests.sh

\*\*Windows\*\*:

run\_tests.bat

Test Coverage

The test suite covers:

* ✅ User registration with validation
* ✅ User authentication and login
* ✅ JWT token generation and validation
* ✅ Password reset request and confirmation
* ✅ Rate limiting functionality
* ✅ Error handling and edge cases
* ✅ API response formats
* ✅ Security validations

\*\*Current Coverage\*\*: 95%+

Sample Test Output

Creating test database for alias 'default'...  
  
System check identified no issues (0 silenced).  
  
.........................  
  
----------------------------------------------------------------------  
  
Ran 25 tests in 12.345s  
  
  
  
OK  
  
Destroying test database for alias 'default'...

🔒 Security Features

The authentication system implements multiple layers of security:

Rate Limiting

Protection against brute force attacks and abuse:

* \*\*Login endpoints\*\*: 5 attempts per minute per IP address
* \*\*Password reset endpoints\*\*: 3 attempts per hour per IP address
* \*\*General API\*\*: 1000 requests per day for authenticated users
* \*\*Anonymous users\*\*: 100 requests per day

JWT Security

* \*\*Access tokens\*\*: Short-lived (60 minutes default)
* \*\*Refresh tokens\*\*: Longer-lived (7 days default)
* \*\*Secure token generation\*\*: Using Django's cryptographic functions
* \*\*Token rotation\*\*: New tokens issued on refresh

Password Security

* \*\*Minimum length\*\*: 8 characters
* \*\*Complexity requirements\*\*: Configurable via Django validators
* \*\*Password hashing\*\*: PBKDF2 with SHA256 (Django default)
* \*\*Password reset tokens\*\*: Time-limited and single-use

Database Security

* \*\*SQL injection protection\*\*: Django ORM parameterized queries
* \*\*Connection encryption\*\*: SSL/TLS for production databases
* \*\*User permissions\*\*: Principle of least privilege

API Security

* \*\*CORS headers\*\*: Properly configured for cross-origin requests
* \*\*Input validation\*\*: Comprehensive serializer validation
* \*\*Output sanitization\*\*: Consistent API response format
* \*\*Error handling\*\*: Secure error messages (no sensitive data exposure)

Production Security Headers

# Configured in settings.py  
  
SECURE\_BROWSER\_XSS\_FILTER = True  
  
SECURE\_CONTENT\_TYPE\_NOSNIFF = True  
  
X\_FRAME\_OPTIONS = 'DENY'  
  
SECURE\_HSTS\_SECONDS = 31536000 # 1 year  
  
SECURE\_HSTS\_INCLUDE\_SUBDOMAINS = True  
  
SECURE\_HSTS\_PRELOAD = True

Environment Security

* \*\*Environment variables\*\*: Sensitive data stored in environment variables
* \*\*Debug mode\*\*: Disabled in production
* \*\*Secret key\*\*: Strong, randomly generated secret keys
* \*\*Allowed hosts\*\*: Restricted to specific domains in production

🚀 Deployment

The application is production-ready and can be deployed on various platforms.

Supported Platforms

* ✅ \*\*Railway\*\* (Recommended)
* ✅ \*\*Render\*\*
* ✅ \*\*Heroku\*\*
* ✅ \*\*AWS EC2\*\*
* ✅ \*\*Google Cloud Platform\*\*
* ✅ \*\*DigitalOcean\*\*
* ✅ \*\*Any Docker-compatible platform\*\*

Railway Deployment

1. \*\*Connect your repository\*\* to Railway
2. \*\*Set environment variables\*\*:

```env

SECRET\_KEY=your-production-secret-key

DEBUG=False

ALLOWED\_HOSTS=your-domain.railway.app

DATABASE\_URL=postgresql://... # Auto-provided by Railway

REDIS\_URL=redis://... # Auto-provided by Railway

```

1. \*\*Deploy\*\*: Railway will automatically build and deploy

Render Deployment

1. \*\*Create a new Web Service\*\* in Render
2. \*\*Connect your repository\*\*
3. \*\*Set build command\*\*: `pip install -r requirements.txt`
4. \*\*Set start command\*\*: `gunicorn auth\_service.wsgi:application`
5. \*\*Add environment variables\*\* as listed above

Docker Deployment

# Build production image  
  
docker build -t auth-service:prod .  
  
  
  
# Run with environment variables  
  
docker run -d \  
  
 -p 8000:8000 \  
  
 -e SECRET\_KEY=your-secret-key \  
  
 -e DEBUG=False \  
  
 -e DATABASE\_URL=postgresql://... \  
  
 auth-service:prod

Environment Variables for Production

|  |  |  |
| --- | --- | --- |
| **Variable** | **Example** | **Required** |
| `SECRET\_KEY` | `django-insecure-abc123...` | ✅ |
| `DEBUG` | `False` | ✅ |
| `ALLOWED\_HOSTS` | `your-domain.com,api.yourdomain.com` | ✅ |
| `DATABASE\_URL` | `postgresql://user:pass@host:5432/db` | ✅ |
| `REDIS\_URL` | `redis://user:pass@host:6379/0` | ❌ |

Production Checklist

* ✅ Set `DEBUG=False`
* ✅ Configure `ALLOWED\_HOSTS`
* ✅ Use environment variables for secrets
* ✅ Set up SSL/HTTPS
* ✅ Configure proper logging
* ✅ Set up monitoring and alerts
* ✅ Regular database backups
* ✅ Update dependencies regularly

Live Demo

🌐 \*\*Live API\*\*: [https://your-deployment-url.onrender.com](https://your-deployment-url.onrender.com)

📖 \*\*API Docs\*\*: [https://your-deployment-url.onrender.com/swagger/](https://your-deployment-url.onrender.com/swagger/)

📁 Project Structure

auth\_service/  
  
├── 📁 auth\_service/ # Django project settings  
  
│ ├── \_\_init\_\_.py  
  
│ ├── settings.py # Main settings file  
  
│ ├── urls.py # URL routing  
  
│ ├── wsgi.py # WSGI configuration  
  
│ └── asgi.py # ASGI configuration  
  
├── 📁 users/ # User management app  
  
│ ├── \_\_init\_\_.py  
  
│ ├── admin.py # Django admin configuration  
  
│ ├── apps.py # App configuration  
  
│ ├── models.py # User model definitions  
  
│ ├── views.py # API view implementations  
  
│ ├── serializers.py # Data serialization  
  
│ ├── urls.py # App URL patterns  
  
│ ├── middleware.py # Custom middleware  
  
│ ├── throttling.py # Rate limiting configuration  
  
│ ├── redis\_utils.py # Redis helper functions  
  
│ ├── 📁 tests/ # Test modules  
  
│ │ ├── \_\_init\_\_.py  
  
│ │ ├── test\_registration.py  
  
│ │ ├── test\_login.py  
  
│ │ └── test\_password\_reset.py  
  
│ ├── 📁 migrations/ # Database migrations  
  
│ └── 📁 management/ # Custom management commands  
  
├── 📄 requirements.txt # Python dependencies  
  
├── 📄 Dockerfile # Docker configuration  
  
├── 📄 docker-compose.yml # Docker Compose setup  
  
├── 📄 manage.py # Django management script  
  
├── 📄 Procfile # Process configuration  
  
├── 📄 runtime.txt # Python version specification  
  
└── 📄 README.md # This file

🛠 Technology Stack

* \*\*Backend Framework\*\*: Django 4.2.10
* \*\*API Framework\*\*: Django REST Framework 3.14.0
* \*\*Authentication\*\*: JWT (djangorestframework-simplejwt)
* \*\*Database\*\*: PostgreSQL
* \*\*Caching\*\*: Redis
* \*\*Documentation\*\*: drf-yasg (Swagger/OpenAPI)
* \*\*Testing\*\*: Django Test Framework
* \*\*Containerization\*\*: Docker & Docker Compose
* \*\*WSGI Server\*\*: Gunicorn
* \*\*Static Files\*\*: WhiteNoise

🤝 Contributing

We welcome contributions! Please follow these guidelines:

Development Setup

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/your-feature-name`
3. Make your changes
4. Add tests for new functionality
5. Run tests: `python manage.py test users`
6. Commit changes: `git commit -m "Add your feature"`
7. Push to branch: `git push origin feature/your-feature-name`
8. Open a Pull Request

Code Style

* Follow PEP 8 style guidelines
* Use meaningful variable and function names
* Add docstrings to functions and classes
* Keep functions small and focused
* Write tests for new features

Pull Request Process

1. Ensure all tests pass
2. Update documentation if needed
3. Add a clear description of changes
4. Reference any related issues
5. Request review from maintainers

📄 License

This project is licensed under the MIT License - see the [LICENSE](LICENSE) file for details.

🆘 Support

If you encounter any issues or have questions:

1. Check the [API Documentation](http://localhost:8000/swagger/)
2. Review existing [Issues](https://github.com/Iduate/User-Authentication-System/issues)
3. Create a new issue with detailed information
4. Contact the development team

🙏 Acknowledgments

* Django REST Framework team for the excellent framework
* JWT library maintainers
* Docker team for containerization tools
* All contributors to this project

---

\*\*Made with ❤️ by the Development Team\*\*

# Document Information

|  |  |
| --- | --- |
| **Document Title** | User Authentication System - API Documentation |
| **Generated Date** | August 29, 2025 at 10:41 PM |
| **Source File** | README.md |
| **Document Type** | Technical Documentation |
| **Version** | 1.0 |