BACKEND DEVELOPMENT

2U - Tala Diab, Reina Najdi, Roy Yammine

PROJECT OVERVIEW

2U is a service matching platform connecting users with providers.

Backend responsibilities:

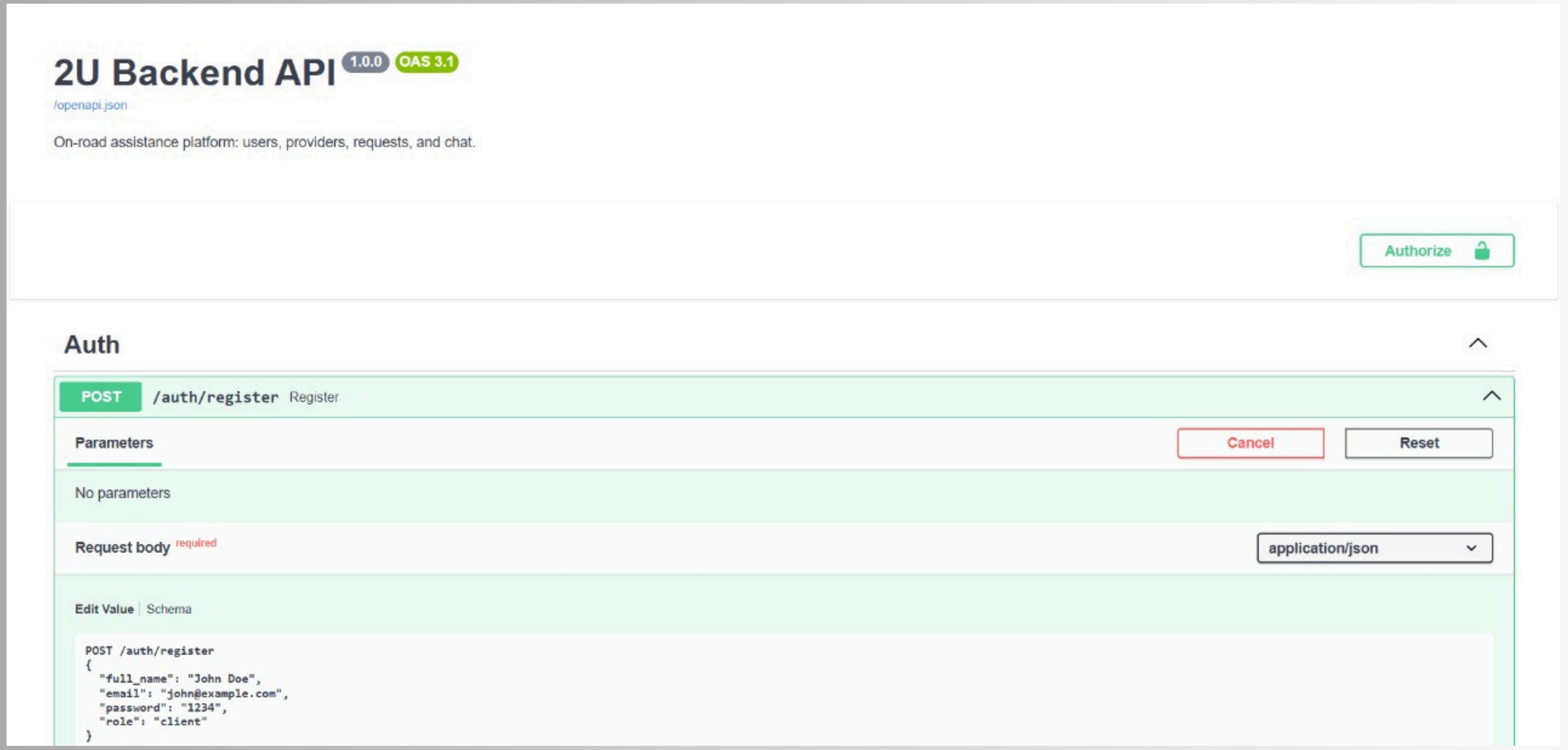
- User authentication and authorization
- Service request management
- Provider matching algorithm
- Database management

TECHNOLOGY STACK

Backend Technologies:
FastAPI - main web framework
SQLite - SQL database
SQLAlchemy - ORM for database operations and model definitions
Pydantic - data validation for request/response schemas
Uvicorn - ASGI server
WebSockets - for real time chat
JWT - authentication
Python-multipart - file uploads

Development Tools:

- GitHub
- Virtual Environment
- Docker



FastAPI Auto generated documentation

DATABASE DESIGN

Data Models (SQLAlchemy):

- User: id, email , password, full_name, role, created_at
- ServiceRequest: id, description, status, user_id, created_at, updated_at
- Message: id, content, sender_id, request_id, timestamp
- FileAttachment: id, filename, file_path, request_id, uploaded_at

API Schemas (Pydantic):

- Login - authentication request
- MsgCreate/MsgOut - chat system
- RequestStatus - request state tracking
- SRCreate/SROut - service request flow
- SRUpdate
- Token, TokenData
- UserCreate/UserOut - user registration/responses
- UserRole - role management

Key Features:

- Role based access (client/provider)
- Request status tracking
- Real time messaging
- Secure data validation

Pydantic schemas

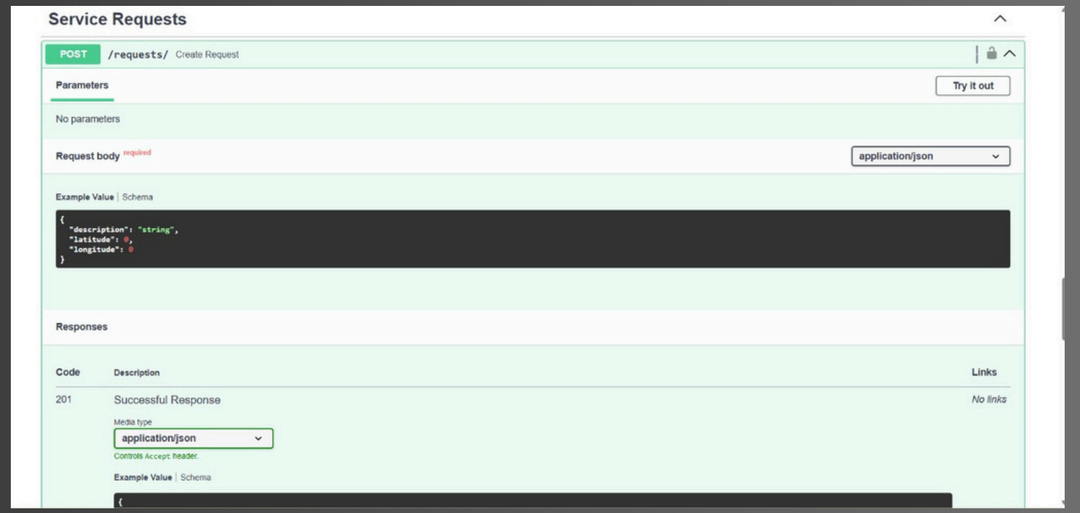
API ENDPOINTS

Authentication:
POST /auth/register - User registration
POST /auth/login - User login

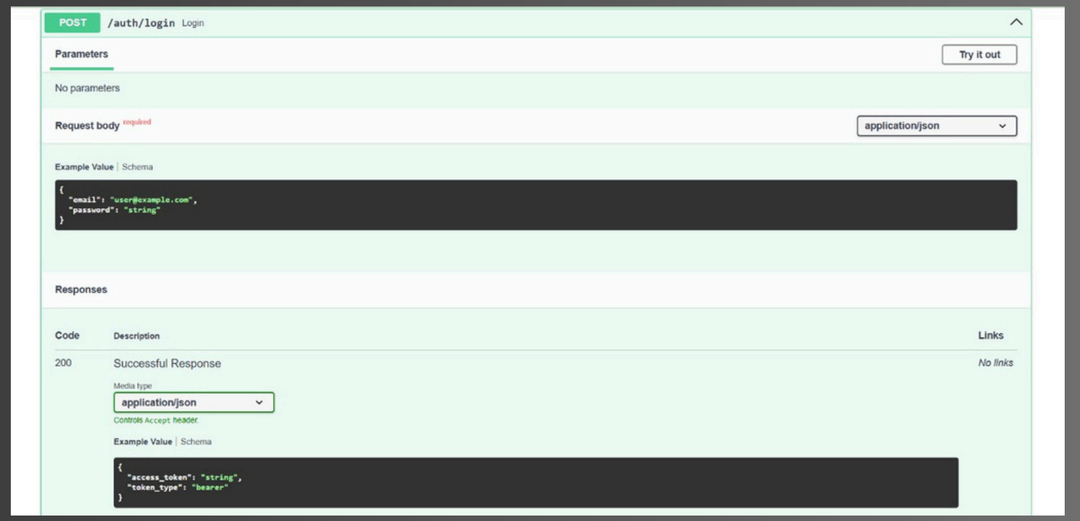
Service Management:
POST /requests/ - Create service request
GET /requests/ - Get user requests
GET /requests/{id} - Get specific request
PUT /requests/{id} - Update request
POST /requests/{id}/accept - Provider accepts requests
POST /request/{id}/complete - Mark request as complete

Messages:
POST /messages/ - Send message
GET /messages/{request_id} - Get chat history
WebSocket /ws/{request_id} - Real time messaging

Files:
POST /requests/{id}/upload - Upload file attachment



API Endpoints - Service requests and authentication



API Access example

Request:
POST /auth/register
{
  "email": "john@example.com",
  "password": "1234",
  “full_name”: “John Doe”,
  "role": "client"
}

Response:
{
  "id": 1,
  "full_name": "John Doe",
  "email": "john@example.com",
  "role": "client"
}
{
 “access_token”:
“eyJ0eXAi0iJKV1QiLCJhbGci0iJIUzl1NiJ9....”,
 “token_type”: “bearer”

Error Handling - Example: Invalid Login

{
  "detail": "Invalid credentials"
}

Features:
• HTTPException for meaningful error messages
• Pydantic automatic data validation
• Proper status codes (400, 401, 404, 500)
• Consistent error response format

Backend
running locally

Containerized with Docker

- Multi-stage Docker build for optimized images
- Docker Compose for easy local development
- Production ready for configuration

Supported Platforms

- Render
- Railway
- Google Cloud Run
- Any Docker-supported platforms

Deployment Setup

docker-compose up —build (single command to build & run)

docker build -t 2u-backend .
docker run -d -p 8000:80 2u-backend

Environment Configuration

DATABASE_URL=sqlite:///./2u.db
JWT_SECRET=jwt-secret-key
JWT_REFRESH_SECRET=refresh_secret
ACCESS_TOKEN_EXPIRE_MINUTES=30

DEPLOYMENT

Development Process

1. Database design - models & relationships
2. API Schema design - request/response format
3. Endpoint implementation
4. Testing and validation (error handling)

GitHub repository: github.com/reinanajdi/2u-backend