

## 2\_BACKEND\_DOCUMENTATION —

Project structure, all API endpoints, demo users, how JWT auth works, how video session works, complete setup and run after git pull steps, useful commands for logs and restart.

# Backend Documentation

## ChourangiHealth Video Consultation Platform

---

### Technology Stack

- Python 3.12
  - FastAPI 0.110.0
  - Uvicorn 0.29.0
  - PyJWT 2.8.0
  - Requests 2.31.0
  - Python-dotenv 1.0.1
- 

### Project Structure

```
openvidu-fastapi-poc/
  └── app/
      ├── main.py          # FastAPI app entry point, CORS, routers
      ├── config.py        # Loads environment variables
      └── dependencies.py # JWT token verification middleware

      └── routers/
          ├── auth.py       # Login endpoint
          ├── health.py     # Health check (empty placeholder)
          └── video.py      # Video session creation endpoint

      └── services/
          ├── auth_service.py # User authentication and JWT creation
          └── openvidu_service.py # OpenVidu REST API calls

      └── schemas/
          └── auth_schema.py # Login request and response models
```

```
|   └── video_schema.py    # Video session request and response models  
|  
|   ├── .env              # Environment variables (never commit secrets in production)  
|   ├── requirements.txt  # Python dependencies  
|   ├── Dockerfile         # Docker build file  
|   └── docker-compose.yml # Docker compose configuration
```

---

## Environment Variables

The .env file at the root of the project contains:

```
OPENVIDU_URL=https://chourangi.duckdns.org  
OPENVIDU_USERNAME=OPENVIDUAPP  
OPENVIDU_SECRET=openvidu-v2-32  
JWT_SECRET=supersecretjwtkey  
JWT_ALGORITHM=HS256
```

- OPENVIDU\_URL must point to your OpenVidu server domain (not raw IP when using SSL)
  - OPENVIDU\_USERNAME is always OPENVIDUAPP for OpenVidu v2
  - OPENVIDU\_SECRET must match the secret configured in /opt/openvidu/.env on the server
  - JWT\_SECRET is used to sign and verify JWT tokens — use a strong random string in production
  - JWT\_ALGORITHM is the signing algorithm — HS256 is standard
- 

## API Endpoints

### Health Check

GET /health

Response: { "status": "ok" }

Authentication: None required

### Login

POST /api/auth/login

Body: { "username": "doctor", "password": "123456" }

Response: { "access\_token": "...", "token\_type": "bearer", "role": "admin" }  
Authentication: None required

## Create Video Session

POST /api/video/session  
Body: { "session\_id": "1" }  
Headers: Authorization: Bearer YOUR\_JWT\_TOKEN  
Response: { "sessionId": "1", "token": "...", "openviduUrl": "https://..." }  
Authentication: JWT token required

## Protected Test Route

GET /protected  
Headers: Authorization: Bearer YOUR\_JWT\_TOKEN  
Response: { "message": "Protected route accessed", "user": {...} }

## Test OpenVidu Connection

GET /test-openvidu  
Response: OpenVidu session object  
Authentication: None required

---

## Demo Users

The application uses hardcoded in-memory users for this POC:

Username	Password	Role
e	d	
doctor	123456	admin
user	123456	user

Note: In a production system these should be stored in a real database with hashed passwords.

---

## How Authentication Works

1. User sends username and password to POST /api/auth/login
  2. Backend checks against the hardcoded USERS\_DB dictionary
  3. If valid, creates a JWT token containing username, role, and expiry of 60 minutes
  4. Returns the token to the frontend
  5. Frontend stores the token in localStorage
  6. All subsequent requests to protected endpoints include the token in the Authorization header as Bearer token
  7. The dependencies.py verify\_token function decodes and validates the token on each request
- 

## How Video Session Works

1. Frontend sends POST /api/video/session with the Calling ID as session\_id and JWT token in header
  2. Backend calls OpenVidu REST API to create a session with customSessionId equal to the Calling ID
  3. If OpenVidu returns 409 meaning the session already exists that is handled gracefully
  4. Backend calls OpenVidu connection API to generate a fresh token for that session
  5. Returns the token and session details to the frontend
  6. Both doctor and patient using the same Calling ID receive tokens for the same session
- 

## Setup and Run After Git Pull

### Step 1 — Clone the repository

```
git clone -b master  
https://github.com/AshuuuPatil/openvidu_videocall_microservice_backend.git backend  
cd backend
```

### Step 2 — Install dependencies

```
pip install -r requirements.txt --break-system-packages --ignore-installed
```

### Step 3 — Configure environment variables

```
nano .env
```

Update OPENVIDU\_URL to point to your OpenVidu server domain:

```
OPENVIDU_URL=https://chourangi.duckdns.org
```

## Step 4 — Run the backend

For development with auto-reload:

```
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

For production on server (runs in background, no Ctrl+C needed):

```
nohup uvicorn app.main:app --host 0.0.0.0 --port 8000 > /tmp/backend.log 2>&1 &
```

## Step 5 — Verify it is running

```
curl http://localhost:8000/health
```

Expected response:

```
{"status":"ok"}
```

---

## Useful Commands

Check if backend is running:

```
ps aux | grep unicorn
```

View backend logs when running with nohup:

```
tail -f /tmp/backend.log
```

Stop the backend:

```
pkill -f unicorn
```

Restart the backend after code changes (pull from GitHub then restart):

```
cd /home/ubuntu/test-1/backend
```

```
git pull origin master  
pkill -f unicorn  
nohup unicorn app.main:app --host 0.0.0.0 --port 8000 > /tmp/backend.log 2>&1 &
```

---

## CORS Configuration

In `app/main.py` the CORS middleware is configured to allow requests from the frontend domain. Update this when deploying to a new domain:

```
app.add_middleware(  
    CORSMiddleware,  
    allow_origins=["https://chourangi.duckdns.org"],  
    allow_credentials=True,  
    allow_methods=["*"],  
    allow_headers=["*"],  
)
```

---

## Dependencies

All dependencies are listed in `requirements.txt`:

```
fastapi==0.110.0  
unicorn==0.29.0  
python-dotenv==1.0.1  
requests==2.31.0  
PyJWT==2.8.0
```