Backend Development - Complete Guide for Beginners

Medicine Recognition System Backend

■ What is a Backend?

A **backend** is the server-side part of an application that:

• Handles business logic and data processing

• Manages databases and file storage

• Provides APIs (Application Programming Interfaces) for frontend/mobile apps

• Handles authentication and security

• Processes user requests and returns responses

Think of it like a restaurant:

• **Frontend** = The dining area (what customers see)

• **Backend** = The kitchen (where food is prepared)

• **API** = The waiter (carries requests and responses)

• **Database** = The pantry (stores ingredients/data)

■■ Our Backend Architecture

Technology Stack

1. **Django** (Web Framework)

• Python-based framework for building web applications

• Handles HTTP requests/responses

• Provides admin panel, ORM, and security features

2. **Django REST Framework** (API Framework)

• Builds RESTful APIs

• Serializes data to JSON format

• Handles API authentication

3. **SQLite** (Database)

• File-based database for storing data

• Stores users, uploads, and authentication tokens

• Located at: `medrec/db.sqlite3`

4. **JWT (JSON Web Tokens)**

• Secure authentication mechanism

• No server-side sessions needed

• Tokens expire automatically

■ Project Structure Explained

```
medrec/
■■■ accounts/                # User authentication app
■   ■■■ models.py            # (Uses default User model)
■   ■■■ serializers.py       # Converts data to/from JSON
■   ■■■ views.py             # API endpoints logic
■   ■■■ urls.py              # URL routes for auth
■
■■■ api/                     # Image upload app
■   ■■■ models.py            # ImageUpload database model
■   ■■■ serializers.py       # Image data serialization
■   ■■■ views.py             # Upload/list endpoints
■   ■■■ urls.py              # URL routes for uploads
■
■■■ core/                    # Shared utilities
■   ■■■ ai_service.py        # AI inference placeholder
■
■■■ medrec/                  # Project settings
■   ■■■ settings.py          # Django configuration
■   ■■■ urls.py              # Main URL router
■   ■■■ wsgi.py              # Server deployment
■
■■■ media/                   # Uploaded images storage
■■■ db.sqlite3               # SQLite database file
■■■ manage.py                # Django command-line tool
```

■ How Does It Work?

Request-Response Flow

```
Mobile App → HTTP Request → Django Backend → Database
                ↓                               ↓
            Process Data                  Store/Retrieve
                ↓                               ↓
            JSON Response ← Format Data ← Return Data
```

Example: User Registration

1. **Mobile sends:**
```
POST /auth/register/
{
"username": "john",
"email": "john@example.com",
"password": "secure123"
}
```

2. **Backend processes:**
• Validates data (checks username uniqueness)
• Hashes password (security)
• Creates user in database
• Generates JWT tokens

3. **Backend responds:**
```json
{
"user": {
"id": 1,
"username": "john",
"email": "john@example.com"
},
"tokens": {
"access": "eyJ0eXAi...",
"refresh": "eyJ0eXAi..."
}
}
```

Authentication System

How JWT Works

1. **Login:**
• User sends username + password

• Backend verifies credentials

• Returns access token (1 hour) + refresh token (7 days)

2. **Authenticated Requests:**

• Client includes token in header:

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGc...
```

• Backend verifies token

• Processes request if valid

3. **Token Refresh:**

• When access token expires

• Send refresh token to get new access token

• No need to re-enter password

Available Auth Endpoints

Endpoint	Method	Purpose
`/auth/register/`	POST	Create new user account
`/auth/login/`	POST	Login and get tokens
`/auth/logout/`	POST	Blacklist refresh token
`/auth/profile/`	GET	Get user info
`/auth/profile/update/`	PATCH	Update user details
`/auth/password/change/`	POST	Change password
`/auth/token/refresh/`	POST	Get new access token

■ Image Upload System

How Image Upload Works

1. **Client sends image:**

• Uses `multipart/form-data` format

• Includes authentication token

• Sends file via `/api/uploads/new/`

2. **Backend processes:**
• Validates image format (JPEG, PNG, etc.)
• Saves file to `media/uploads/` folder
• Calls AI service for analysis
• Stores metadata in database

3. **Database stores:**
```python
ImageUpload {
id: 1
image: "uploads/medicine_abc123.jpg"
uploaded_by: User(john)
result: {"predicted_name": "Aspirin", ...}
created_at: "2025-11-29T10:30:00Z"
}
```

4. **Returns response:**
```json
{
"id": 1,
"image": "http://127.0.0.1:8000/media/uploads/medicine_abc123.jpg",
"result": {
"predicted_name": "Aspirin",
"confidence": 0.95
},
"created_at": "2025-11-29T10:30:00Z"
}
```

Database Explained

What is SQLite?

• **File-based database** (no separate server needed)
• Stores all data in `db.sqlite3` file
• Perfect for development and small projects
• Can be upgraded to PostgreSQL for production

Database Tables

1. **auth_user** (Django built-in)
• Stores user accounts
• Fields: id, username, email, password (hashed), first_name, last_name

2. **api_imageupload** (Custom)
• Stores uploaded images
• Fields: id, image, uploaded_by_id, result, created_at

3. **token_blacklist_outstandingtoken**
• Tracks all issued JWT tokens

4. **token_blacklist_blacklistedtoken**
• Stores invalidated tokens (after logout)

Database Commands

```
# View database in admin panel
python manage.py runserver
# Visit: http://127.0.0.1:8000/admin/

# Query database via shell
python manage.py shell
>>> from django.contrib.auth.models import User
>>> User.objects.all()

# Create database backup
Copy-Item db.sqlite3 db.backup.sqlite3
```

■ Common Backend Operations

Starting the Server

```
cd e:/Temp/GRAD/medrec
python manage.py runserver

# For mobile access:
python manage.py runserver 0.0.0.0:8000
```

Server runs at: `http://127.0.0.1:8000/`

Creating Database Changes

```
# After modifying models.py:
python manage.py makemigrations  # Create migration file
python manage.py migrate         # Apply changes to database
```

Creating Admin User

```
python manage.py createsuperuser
# Username: admin
# Email: admin@example.com
# Password: admin123
```

Accessing Admin Panel

1. Visit: `http://127.0.0.1:8000/admin/`
2. Login with superuser credentials
3. View/edit users and uploads

■ API Basics

What is REST API?

REST = Representational State Transfer

• Uses HTTP methods: GET, POST, PATCH, DELETE

• Exchanges data in JSON format

• Stateless (no server-side sessions)

HTTP Methods Explained

Method	Purpose	Example
GET	Retrieve data	Get list of uploads
POST	Create new data	Register user, upload image
PATCH	Update existing data	Update profile
DELETE	Remove data	Delete account

HTTP Status Codes

Code	Meaning
200	Success
201	Created (new resource)
400	Bad Request (invalid data)
401	Unauthorized (no/invalid token)
404	Not Found
500	Server Error

Testing APIs with curl

```
# Register user
curl -X POST http://127.0.0.1:8000/auth/register/ `
  -H "Content-Type: application/json" `
  -d '{\"username\":\"test\",\"email\":\"test@example.com\",\"password\":\"pass123\",\"password2\":\"pass123

# Login
curl -X POST http://127.0.0.1:8000/auth/login/ `
  -H "Content-Type: application/json" `
  -d '{\"username\":\"test\",\"password\":\"pass123\"}'

# Get profile (replace YOUR_TOKEN)
curl http://127.0.0.1:8000/auth/profile/ `
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"

# Upload image
curl -X POST http://127.0.0.1:8000/api/uploads/new/ `
  -H "Authorization: Bearer YOUR_TOKEN" `
  -F "image=@path/to/image.jpg"
```

■ AI Integration (Placeholder)

Current Implementation

Located in `core/ai_service.py`:

```
def infer(image_path: str) -> dict:
    """
    Placeholder for AI model inference.
    Replace with actual model.
    """
    return {
        'predicted_name': 'example-medicine',
        'confidence': 0.75,
        'description': 'This is a placeholder result',
        'side_effects': ['drowsiness', 'nausea'],
        'dosage': '500mg twice daily'
    }
```

How to Add Real AI Model

Option 1: TensorFlow/Keras

```
import tensorflow as tf

model = tf.keras.models.load_model('path/to/model.h5')

def infer(image_path: str) -> dict:
    img = tf.keras.preprocessing.image.load_img(
        image_path, target_size=(224, 224)
    )
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    predictions = model.predict(img_array)

    return {
        'predicted_name': class_names[np.argmax(predictions)],
        'confidence': float(np.max(predictions))
    }
```

Option 2: PyTorch

```
import torch
from torchvision import transforms

model = torch.load('model.pth')
model.eval()

def infer(image_path: str) -> dict:
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])

    image = Image.open(image_path)
    image_tensor = transform(image).unsqueeze(0)

    with torch.no_grad():
        output = model(image_tensor)

    return {
        'predicted_name': classes[output.argmax()],
        'confidence': float(output.max())
    }
```

Option 3: External API

```
import requests

def infer(image_path: str) -> dict:
    with open(image_path, 'rb') as f:
        response = requests.post(
            'https://api.yourservice.com/predict',
            files={'image': f},
            headers={'Authorization': 'Bearer YOUR_API_KEY'}
        )

    return response.json()
```

■ Security Best Practices

Current Security Features

1. **Password Hashing**
• Uses Django's PBKDF2 algorithm
• Passwords never stored in plain text

2. **JWT Tokens**
• Cryptographically signed
• Cannot be forged
• Auto-expire (1 hour for access tokens)

3. **Token Blacklist**
• Logout invalidates refresh tokens
• Prevents token reuse after logout

4. **CORS (Cross-Origin Resource Sharing)**
• Configured for mobile app access
• Prevents unauthorized domains

For Production

```
# In settings.py:

# Change secret key
SECRET_KEY = os.environ.get('DJANGO_SECRET_KEY')

# Disable debug mode
DEBUG = False

# Specify allowed hosts
ALLOWED_HOSTS = ['yourdomain.com', 'api.yourdomain.com']

# Use PostgreSQL instead of SQLite
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'medrec_db',
        'USER': 'db_user',
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

# Enable HTTPS only
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

■ Mobile App Integration

Flutter Connection

Android Emulator:

```
static const String baseUrl = 'http://10.0.2.2:8000';
```

iOS Simulator:

```
static const String baseUrl = 'http://127.0.0.1:8000';
```

Real Device (same Wi-Fi):

```
static const String baseUrl = 'http://192.168.1.5:8000';
// Use your PC's IP address
```

Making API Calls

```
// Register
final response = await http.post(
  Uri.parse('$baseUrl/auth/register/'),
  headers: {'Content-Type': 'application/json'},
  body: jsonEncode({
    'username': 'john',
    'email': 'john@example.com',
    'password': 'pass123',
    'password2': 'pass123',
  }),
);

// Login
final loginResponse = await http.post(
  Uri.parse('$baseUrl/auth/login/'),
  headers: {'Content-Type': 'application/json'},
  body: jsonEncode({
    'username': 'john',
    'password': 'pass123',
  }),
);

final tokens = jsonDecode(loginResponse.body);
String accessToken = tokens['access'];

// Upload image
var request = http.MultipartRequest(
  'POST',
  Uri.parse('$baseUrl/api/uploads/new/'),
);
request.headers['Authorization'] = 'Bearer $accessToken';
request.files.add(
  await http.MultipartFile.fromPath('image', imageFile.path),
);

final streamedResponse = await request.send();
final response = await http.Response.fromStream(streamedResponse);
```

■ Troubleshooting

Common Issues

1. "Connection refused" error

• **Cause:** Server not running

• **Fix:** Run `python manage.py runserver`

2. "Token has expired"

• **Cause:** Access token older than 1 hour

• **Fix:** Use refresh token to get new access token

3. "CORS error" in browser

• **Cause:** Frontend domain not allowed

• **Fix:** Check `CORS_ALLOWED_ORIGINS` in settings.py

4. "Database is locked"

• **Cause:** Multiple processes accessing SQLite

• **Fix:** Stop all Django processes, restart server

5. "Migrations not applied"

• **Cause:** Database schema out of sync

• **Fix:** Run `python manage.py migrate`

6. Image upload fails

• **Cause:** Missing media folder or permissions

• **Fix:** Check `MEDIA_ROOT` in settings.py

Debug Mode

```
# In settings.py, enable debug mode for detailed errors:
DEBUG = True

# Check logs in terminal where server is running
# Errors will show detailed tracebacks
```

■ Learning Resources

Django Documentation

• Official Docs: https://docs.djangoproject.com/

• Django REST Framework: https://www.django-rest-framework.org/

Tutorials

• **Django Basics:** Django Girls Tutorial
• **REST APIs:** DRF Official Tutorial
• **JWT Auth:** SimpleJWT Documentation

Tools

• **Postman:** Test APIs visually
• **DB Browser for SQLite:** View database contents
• **VS Code Extensions:** Python, Django

■ Key Concepts Summary

Concept	Simple Explanation
Backend	Server that processes requests and manages data
API	Interface for apps to communicate with backend
Database	Organized storage for data (users, uploads, etc.)
Authentication	Verifying user identity (login)
JWT Token	Secure ticket proving user is logged in
Serialization	Converting data to/from JSON format
Migration	Database schema version control
ORM	Write database queries using Python (not SQL)
Endpoint	Specific URL that handles a type of request
HTTP Method	Type of request (GET, POST, etc.)

■ Quick Reference

Start Server

```
cd e:/Temp/GRAD/medrec
python manage.py runserver 0.0.0.0:8000
```

Access Points

• **API Base:** http://127.0.0.1:8000/
• **Admin Panel:** http://127.0.0.1:8000/admin/
• **Admin Credentials:** admin / admin123

Key Files

• **Settings:** `medrec/settings.py`
• **URLs:** `medrec/urls.py`
• **Database:** `db.sqlite3`
• **AI Service:** `core/ai_service.py`

Important Commands

```
python manage.py makemigrations  # Create migrations
python manage.py migrate         # Apply migrations
python manage.py createsuperuser # Create admin
python manage.py shell           # Open Python shell
```

This backend is ready for your graduation project!

All authentication and upload features are implemented.

Just add your AI model and connect your mobile app! ■