**Telemedicine Backend Project Documentation**

**✅ Project Overview**

You have developed a microservices backend:

* **auth-service** (authentication & role management)
* **user-service** (profile storage)
* **MongoDB** (shared database)

**✅ Services & Ports**

|  |  |  |
| --- | --- | --- |
| **Service** | **Port** | **Purpose** |
| auth-service | 5001 | Register/Login/Approve Doctors |
| user-service | 5002 | Store & Retrieve profiles |
| MongoDB | 27017 | Database |

**✅ Docker Compose**

version: '3.8'

services:

auth:

build: ./auth

ports:

- "5001:5001"

env\_file:

- ./auth/.env

volumes:

- ./auth:/app

depends\_on:

- mongo

user:

build: ./user-service

ports:

- "5002:5002"

env\_file:

- ./user-service/.env

volumes:

- ./user-service:/app

depends\_on:

- mongo

mongo:

image: mongo

container\_name: mongodb

ports:

- "27017:27017"

volumes:

- mongo\_data:/data/db

volumes:

mongo\_data:

**✅ Environment Files**

**auth/.env**

PORT=5001

MONGO\_URI=mongodb://mongo:27017/authdb

JWT\_SECRET=your\_super\_secret\_key

**user-service/.env**

PORT=5002

MONGO\_URI=mongodb://mongo:27017/userdb

JWT\_SECRET=your\_super\_secret\_key

**✅ Mongoose Models**

**auth-service**

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: { type: String, required: true },

email: { type: String, required: true, unique: true, lowercase: true, validate: { validator: require('validator').isEmail, message: 'Invalid email' } },

password: { type: String, required: true },

role: { type: String, enum: ['user', 'doctor', 'admin'], required: true },

medicalLicenseNumber: { type: String },

isApproved: { type: Boolean, default: true }

});

module.exports = mongoose.model('User', userSchema);

**user-service**

const mongoose = require('mongoose');

const validator = require('validator');

const UserData = new mongoose.Schema({

authUserId: { type: mongoose.Schema.Types.ObjectId, required: true },

firstName: { type: String, required: true },

lastName: { type: String, required: true },

gender: { type: String, required: true },

dob: { type: Date, required: true },

email: { type: String, required: true, unique: true, lowercase: true, validate: { validator: validator.isEmail, message: 'Invalid email' } },

address: { type: String, required: true },

role: { type: String, enum: ['user', 'doctor', 'admin'], default: 'user' }

});

module.exports = mongoose.model('userdata', UserData);

**✅ Auth Service Routes**

**Registration**

**POST /api/auth/register**

Examples:

**User:**

{

"name": "John Doe",

"email": "john@example.com",

"password": "password123",

"role": "user"

}

**Doctor:**

{

"name": "Dr. Alice",

"email": "dr.alice@example.com",

"password": "password123",

"role": "doctor",

"medicalLicenseNumber": "MD-2025-000123"

}

**Login**

**POST /api/auth/login**

Request:

{

"email": "john@example.com",

"password": "password123"

}

Response:

{

"token": "...",

"user": {

"id": "...",

"name": "...",

"role": "user"

}

}

*Note:* Unapproved doctors receive:

{ "message": "Your account is pending approval by the admin." }

**Get Unapproved Doctors**

**GET /api/auth/unapproved-doctors** Requires Admin token.

**Approve Doctor**

**POST /api/auth/approve-doctor/\*\*\*\*\*\*\*\*\*\*\*\*:id** Requires Admin token.

**✅ User Service Routes**

**GET /api/users** - Public health check.

**GET /api/users/\*\*\*\*\*\*\*\*\*\*\*\*:id** - Protected.

**POST /api/users** - Create profile (protected).

**✅ Example Token Usage**

**Headers:**

Authorization: Bearer <token>

**✅ Registration Flow**

1. User or Doctor registers.
2. User is active immediately.
3. Doctor awaits approval.

**✅ Login Flow**

1. Login to get token.
2. Use token in Authorization header.

**✅ Admin Approval Flow**

1. Admin logs in.
2. Admin fetches unapproved doctors.
3. Admin approves doctor.

**✅ Protected Routes**

Any route using:

const protect = require('../middleware/authMiddleware');

Requires Authorization header.

**✅ Docker Commands**

**Build:**

docker-compose build

**Up:**

docker-compose up

**Down:**

docker-compose down -v

\*\*Check Containers:\*\***tainers:**

docker ps

**📄 Appointment Service Documentation**

**✅ Overview**

The **Appointment Service** handles scheduling, updating, cancelling, and rescheduling appointments between patients and doctors.

It exposes REST APIs secured via JWT tokens.

**✅ Service Details**

| **Service** | **Port** |
| --- | --- |
| Appointment Service | 5003 |
| MongoDB | 27017 |

**✅ Environment Variables (.env)**

PORT=5003

MONGO\_URI=mongodb://mongo:27017/appointmentdb

JWT\_SECRET=your\_super\_secret\_key

**✅ Mongoose Schema**

const rescheduleSchema = new mongoose.Schema({

date: { type: Date, required: true },

timeSlot: { type: String, required: true },

reason: { type: String }

});

const appointmentSchema = new mongoose.Schema({

patientId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },

doctorId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },

date: { type: Date, required: true },

timeslot: { type: String, required: true },

status: { type: String, enum: ['pending', 'confirmed', 'completed', 'cancelled'], default: 'pending' },

isFirstTimeVisit: { type: Boolean, required: true },

notes: { type: String },

meetingLink: { type: String },

reasonForVisit: { type: String },

paymentStatus: { type: String, enum: ['pending', 'paid', 'refunded'], default: 'pending' },

durationMinutes: { type: Number, default: 30 },

followUpRequired: { type: Boolean, default: false },

cancellationReason: { type: String },

rescheduleHistory: [rescheduleSchema]

}, { timestamps: true });

**✅ Routes**

**🔹 Create Appointment**

**POST /api/appointment**

Body:

{

"patientId": "ObjectId",

"doctorId": "ObjectId",

"date": "2025-07-15T10:00:00Z",

"timeslot": "09:00-09:30",

"isFirstTimeVisit": true,

"notes": "Some note",

"meetingLink": "https://meetinglink.com",

"reasonForVisit": "Consultation"

}

**🔹 Get Appointment by ID**

**GET /api/appointment/appointment-info/:id**

**🔹 List All Appointments (Paginated)**

**GET /api/appointment/allAppointments**

Query params:

?page=1&limit=10

**🔹 Filter Appointments**

**GET /api/appointment/filter**

Query params example:

?status=confirmed&startDate=2025-07-01&endDate=2025-07-31

**🔹 Update Appointment**

**PUT /api/appointment/update/:id**

Protected (token required).

Body example (doctor):

{

"status": "confirmed",

"meetingLink": "https://updated-link.com"

}

**🔹 Delete Appointment**

**DELETE /api/appointment/delete/:id**

Protected (token required).

**🔹 Mark Appointment Status**

**PUT /api/appointment/status/:id**

Body:

{

"status": "completed"

}

**🔹 Reschedule Appointment**

**PUT /api/appointment/reschedule/:id**

Body:

{

"date": "2025-08-01T09:00:00Z",

"timeSlot": "09:00-09:30",

"reason": "Patient requested reschedule"

}

**✅ Authorization**

Use the Authorization header:

Authorization: Bearer <token>

**✅ Example Tokens**

* Patients and doctors can **create**, **view**, **update** their own appointments.
* Admins can **update or delete** any appointment.

**✅ Docker Commands**

**Build:**

docker-compose build

**Start:**

docker-compose up

**Stop:**

docker-compose down -v

**✅ Next Features (Planned)**

* Notifications on status change.
* Payment integration.
* Automatic reminders.

If you want, I can **generate a Word or Markdown file** of this, or expand further for *frontend usage* and *error examples*. Just tell me!