Impulse Pathology Lab - Full Backend Guide

1. Project Overview

Tech Stack:
- Frontend: Next.js + TypeScript
- Backend: Next.js API Routes + TypeScript
- Database: MongoDB
- Payment: Stripe & Razorpay
- Reports: AWS S3 pre-signed uploads

Key Features:
- User registration/login with JWT authentication
- Role-based access (user/admin)
- Services management (CRUD)
- Online test booking & payment
- Patient report upload & secure access
- Admin dashboard stats

2. Project Structure

impulse-backend/
% % % lib/
% % % % db.ts
% % % % auth.ts
% % % % middleware.ts
% % % % rateLimit.ts
% % % models/
% % % % User.ts
% % % % Service.ts
% % % % Booking.ts
% % % % Report.ts
% % % pages/
% % % % api/
% % % % auth/
% % % % % login.ts
% % % % % register.ts
% % % % services/

```
%       %    % % %  index.ts
%       %    % % %  [id].ts
%       % % %  bookings/
%       %    % % %  index.ts
%       %    % % %  [id].ts
%       % % %  reports/
%       %    % % %  upload.ts
%       % % %  payments/
%       %    % % %  stripe.ts
%       %    % % %  razorpay.ts
%       % % %  admin/
%          % % %  stats.ts
% % %  .env.local
% % %  package.json
% % %  tsconfig.json
```

3. Environment Variables (.env.local)

```
MONGODB_URI=your_mongodb_connection_string
JWT_SECRET=your_jwt_secret
STRIPE_SECRET_KEY=your_stripe_secret_key
RAZORPAY_KEY_ID=your_razorpay_key_id
RAZORPAY_KEY_SECRET=your_razorpay_secret
AWS_ACCESS_KEY_ID=your_aws_access_key
AWS_SECRET_ACCESS_KEY=your_aws_secret_key
AWS_REGION=your_aws_region
S3_BUCKET_NAME=your_s3_bucket_name
```

4. Models - User.ts

```
import mongoose from "mongoose";

const UserSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String,
  role: { type: String, default: "user" },
}, { timestamps: true });

export default mongoose.models.User || mongoose.model("User", UserSchema);
```

5. Models - Service.ts

```
import mongoose from "mongoose";

const ServiceSchema = new mongoose.Schema({
  name: String,
  price: Number,
  description: String,
}, { timestamps: true });

export default mongoose.models.Service || mongoose.model("Service",
ServiceSchema);
```

6. Models - Booking.ts

```
import mongoose from "mongoose";

const BookingSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
  services: [{ type: mongoose.Schema.Types.ObjectId, ref: "Service" }],
  total: Number,
  status: { type: String, default: "pending" },
  sampleDate: Date,
  payment: {
    paid: { type: Boolean, default: false },
    provider: String,
    providerPaymentId: String,
  },
}, { timestamps: true });

export default mongoose.models.Booking || mongoose.model("Booking",
BookingSchema);
```

7. Models - Report.ts

```
import mongoose from "mongoose";

const ReportSchema = new mongoose.Schema({
  booking: { type: mongoose.Schema.Types.ObjectId, ref: "Booking" },
  fileUrl: String,
  notes: String,
  uploadedBy: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
}, { timestamps: true });

export default mongoose.models.Report || mongoose.model("Report", ReportSchema);
```

8. lib/db.ts

```
import mongoose from "mongoose";

const dbConnect = async () => {
  if (mongoose.connection.readyState >= 1) return;
  return mongoose.connect(process.env.MONGODB_URI!);
};

export default dbConnect;
```

9. lib/auth.ts

```
import jwt from "jsonwebtoken";

export const signToken = (payload: object) =>
  jwt.sign(payload, process.env.JWT_SECRET!, { expiresIn: "7d" });

export const verifyToken = (token: string) => {
  try {
    return jwt.verify(token, process.env.JWT_SECRET!);
  } catch {
    return null;
  }
};
```

10. lib/middleware.ts

```
import type { NextApiRequest, NextApiResponse } from "next";
import { verifyToken } from "./auth";

export const authMiddleware = (roles: string[] = []) => {
  return (handler: any) => async (req: NextApiRequest, res: NextApiResponse) => {
    const authHeader = req.headers.authorization;
    if (!authHeader) return res.status(401).json({ message: "Unauthorized" });

    const token = authHeader.split(" ")[1];
    const decoded = verifyToken(token);
    if (!decoded) return res.status(401).json({ message: "Invalid token" });

    if (roles.length && !roles.includes((decoded as any).role))
      return res.status(403).json({ message: "Forbidden" });

    (req as any).user = decoded;
    return handler(req, res);
  };
};
```

11. lib/rateLimit.ts

```
import rateLimit from "express-rate-limit";

export const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 10,
  message: "Too many requests, please try again later.",
});
```

12. Deployment Steps

1. Clone backend project
2. Install dependencies: npm install

3. Configure .env.local
4. Run locally: npm run dev
5. Deploy to Vercel or other Next.js hosting
6. Use MongoDB Atlas for production
7. Stripe/Razorpay: live keys for production
8. AWS S3: set proper bucket policy