🔧 BACKEND TEAM RESPONSIBILITIES

1. API Architecture

Technology Stack Recommendation:

- **Node.js + Express** (JavaScript - matches frontend)
- **OR Python + FastAPI** (Better for AI/ML integration)
- **OR Java Spring Boot** (Enterprise-grade)

API Structure:

```
/api
  /auth
    POST /register
    POST /login
    POST /logout
    POST /refresh-token
    GET  /me

  /users
    GET    /users
    GET    /users/:id
    POST   /users
    PUT    /users/:id
    DELETE /users/:id
    PATCH  /users/:id/password

  /patients
    GET    /patients
    GET    /patients/:id
    POST   /patients
    PUT    /patients/:id
    DELETE /patients/:id
    GET    /patients/:id/medical-profile
    PUT    /patients/:id/medical-profile
    GET    /patients/:id/appointments
    GET    /patients/:id/medical-records
    GET    /patients/:id/prescriptions
    GET    /patients/:id/invoices
```

```
/doctors
  GET    /doctors
  GET    /doctors/:id
  POST   /doctors
  PUT    /doctors/:id
  DELETE /doctors/:id
  GET    /doctors/:id/schedule
  PUT    /doctors/:id/schedule
  GET    /doctors/:id/appointments
  GET    /doctors/:id/availability

/appointments
  GET    /appointments
  GET    /appointments/:id
  POST   /appointments
  PUT    /appointments/:id
  DELETE /appointments/:id
  PATCH  /appointments/:id/status
  GET    /appointments/calendar

/medical-records
  GET    /medical-records
  GET    /medical-records/:id
  POST   /medical-records
  PUT    /medical-records/:id
  DELETE /medical-records/:id
  POST   /medical-records/:id/attachments

/prescriptions
  GET    /prescriptions
  GET    /prescriptions/:id
  POST   /prescriptions
  PUT    /prescriptions/:id
  PATCH  /prescriptions/:id/status
  GET    /prescriptions/:id/pdf

/radiology
  GET    /scans
  GET    /scans/:id
  POST   /scans
```

```
  PUT    /scans/:id
  POST   /scans/:id/upload
  GET    /scans/:id/dicom
  PATCH  /scans/:id/status

/cdss
  GET    /analysis
  GET    /analysis/:id
  POST   /analysis (trigger AI analysis)
  PATCH  /analysis/:id/review

/billing
  GET    /invoices
  GET    /invoices/:id
  POST   /invoices
  PUT    /invoices/:id
  DELETE /invoices/:id
  GET    /invoices/:id/pdf

/payments
  GET    /payments
  POST   /payments
  POST   /payments/process
  POST   /payments/refund

/icd10
  GET    /codes
  GET    /codes/:code
  GET    /codes/search?q=hypertension

/reports
  GET    /reports/dashboard
  GET    /reports/appointments
  GET    /reports/revenue
  GET    /reports/patients
```

---

2. Authentication & Authorization

A. JWT Token System:

```
// Login Response
{
  accessToken: "eyJhbGciOiJIUzI1NiIs...", // 15 min expiry
  refreshToken: "dGhpcyBpcyBhIHJlZn...", // 7 days expiry
  user: {
    id: "123",
    email: "doctor@heartology.com",
    role: "doctor",
    firstName: "John",
    lastName: "Smith"
  }
}
```

B. Role-Based Access Control (RBAC):

```
// Middleware
const authorize = (allowedRoles) => {
  return (req, res, next) => {
    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Forbidden' });
    }
    next();
  };
};

// Usage
app.get('/api/users',
  authenticate,
  authorize(['admin']),
  getUsersController
);
```

C. Password Security:

- **Bcrypt** hashing (cost factor: 12)
- Password requirements: 8+ chars, uppercase, lowercase, number, special char
- Account lockout after 5 failed attempts

---

3. Key API Endpoints Implementation

A. Login Endpoint

```
POST /api/auth/login

Request:
{
  email: "doctor@heartology.com",
  password: "SecurePass123!"
}

Response:
{
  success: true,
  accessToken: "...",
  refreshToken: "...",
  user: {
    id: "123",
    email: "doctor@heartology.com",
    role: "doctor",
    firstName: "John",
    lastName: "Smith"
  }
}

// Frontend will store tokens in localStorage
// Send accessToken in Authorization header: "Bearer <token>"
```

B. Get Appointments (Doctor View)

```
GET /api/appointments?doctorId=123&date=2025-12-06

Response:
{
  success: true,
  data: [
    {
      id: "app_001",
      patient: {
        id: "pat_001",
        name: "John Doe",
        age: 45,
        ssn: "***-**-6789"
      },
      appointmentDate: "2025-12-06",
```

```
      appointmentTime: "09:00",
      type: "Check-up",
      status: "Confirmed",
      reasonForVisit: "Chest pain",
      duration: 30
    }
  ],
  total: 5
}
```

C. Create Medical Record

```
POST /api/medical-records

Request:
{
  patientId: "pat_001",
  doctorId: "doc_123",
  appointmentId: "app_001",
  recordType: "Consultation",
  vitalSigns: {
    bloodPressure: { systolic: 120, diastolic: 80 },
    heartRate: 72,
    temperature: 98.6,
    oxygenSaturation: 98,
    respiratoryRate: 16
  },
  chiefComplaint: "Chest pain",
  subjective: "Patient reports...",
  objective: "Physical examination reveals...",
  assessment: "Suspected angina",
  plan: "ECG ordered, follow-up in 1 week",
  diagnoses: [
    { icd10Code: "I20.9", description: "Angina pectoris", isPrimary: true
  ]
}

Response:
{
  success: true,
  data: {
    id: "rec_001",
    ...requestData,
```

```
    createdAt: "2025-12-06T10:30:00Z"
  }
}
```

D. Process Payment

```
POST /api/payments/process

Request:
{
  invoiceId: "inv_001",
  amount: 350.00,
  paymentMethod: "Credit Card",
  cardDetails: {
    cardNumber: "4111111111111111",
    expiryMonth: "12",
    expiryYear: "2026",
    cvv: "123",
    cardholderName: "John Doe"
  }
}

Response:
{
  success: true,
  data: {
    paymentId: "pay_001",
    transactionId: "txn_abc123",
    status: "Completed",
    receiptNumber: "RCP-2025-001",
    paidAmount: 350.00,
    balanceAmount: 0
  }
}

// Integrate with: Stripe, PayPal, Square, or Authorize.net
```

---

4. File Upload & Storage

A. DICOM File Upload:

```
POST /api/radiology/scans/:id/upload
Content-Type: multipart/form-data

// Use: AWS S3, Azure Blob Storage, or Google Cloud Storage
// Store files with encryption
// Return signed URLs for viewing
```

B. Storage Structure:

```
s3://heartology-medical-files/
  /patients/{patientId}/
    /radiology/{scanId}/
      - scan_001.dcm
      - scan_002.dcm
    /documents/
      - insurance_card.pdf
      - consent_form.pdf
  /prescriptions/{prescriptionId}/
    - prescription.pdf
  /invoices/{invoiceId}/
    - invoice.pdf
```

---

5. CDSS Integration

A. AI Analysis Trigger:

```
POST /api/cdss/analysis

Request:
{
  radiologyScanId: "scan_001",
  studyType: "ECG",
  dicomFileUrls: ["s3://..."]
}

Process:
1. Send DICOM files to AI model endpoint
2. AI processes images (2-5 minutes)
3. Return findings with confidence scores
4. Flag for doctor review
```

```
Response:
{
  success: true,
  data: {
    analysisId: "cdss_001",
    findings: [
      "Left ventricular hypertrophy detected",
      "ST segment elevation in leads V2-V4"
    ],
    confidenceScore: 87,
    severity: "High",
    recommendations: [
      "Immediate cardiology consultation",
      "Consider troponin levels"
    ],
    reviewStatus: "Pending"
  }
}
```

B. AI Model Endpoint:

```
# Separate microservice (Python + TensorFlow/PyTorch)
POST https://ai.heartology.com/analyze

# Models needed:
- ECG Analysis (Arrhythmia detection)
- CT/MRI Analysis (CAD detection)
- Risk Stratification Model
```

---

6. Real-time Features

A. WebSocket for Notifications:

```
// Socket.io implementation
io.on('connection', (socket) => {
  socket.on('subscribe', (userId) => {
    socket.join(`user_${userId}`);
  });
});

// Notify doctor when CDSS analysis completes
io.to(`user_${doctorId}`).emit('cdss_complete', {
```

```
  scanId: "scan_001",
  severity: "High",
  message: "AI analysis flagged critical findings"
});

// Notify patient of appointment confirmation
io.to(`user_${patientId}`).emit('appointment_confirmed', {
  appointmentId: "app_001",
  date: "2025-12-10",
  time: "14:00"
});
```

---

7. Security Requirements

A. Data Encryption:

- **At Rest**: AES-256 encryption for database
- **In Transit**: TLS 1.3 for all API calls
- **PII/PHI**: Additional encryption layer for SSN, medical records

B. HIPAA Compliance:

- Audit logs for all data access
- Automatic session timeout (15 minutes)
- Minimum necessary access principle
- Breach notification system
- Data backup and disaster recovery

C. API Rate Limiting:

```
// 100 requests per 15 minutes per IP
app.use(rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
}));
```

---

8. PDF Generation

A. Prescription PDF:

```
GET /api/prescriptions/:id/pdf

// Use: PDFKit or Puppeteer
// Include:
- Hospital letterhead
- Doctor details and signature
- Patient information
- Medication list
- QR code for verification
```

B. Invoice PDF:

```
GET /api/invoices/:id/pdf

// Include:
- Invoice number and date
- Itemized services
- Insurance breakdown
- Payment history
- QR code for online payment
```

---

9. Email Notifications

A. SendGrid/AWS SES Integration:

```
// Appointment Confirmation
- Send to patient email
- Include calendar invite (.ics file)
- Reminder 24 hours before

// Prescription Ready
- Notify patient when prescription is written
- Include pharmacy instructions

// Invoice Generated
- Send invoice PDF
- Payment link

// Lab Results Available
- Notify patient to log in and view
- Security: Never email actual results
```

---

10. Reporting & Analytics

A. Dashboard Statistics API:

```
GET /api/reports/dashboard?role=admin&period=month

Response:
{
  totalPatients: 1250,
  activePatients: 890,
  newPatients: 45,
  totalAppointments: 320,
  completedAppointments: 280,
  revenue: 125000,
  pendingInvoices: 15,
  averageWaitTime: 12, // minutes
  patientSatisfaction: 4.5
}
```

B. Revenue Reports:

```
GET /api/reports/revenue?startDate=2025-01-01&endDate=2025-12-31

// Group by month, service type, doctor, insurance provider
// Export to CSV/Excel
```

🔄 INTEGRATION CHECKLIST

Frontend-Backend Connection:

1. Update API Base URL:

```
// src/config/api.js
export const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://loca
```

2. Create API Service Layer:

```
// src/services/api.js
import axios from 'axios';

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json'
  }
});

// Add token to requests
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export default api;
```

3. Replace Mock Data:

```
// Before (Mock):
const [patients, setPatients] = useState(mockData);

// After (Real API):
useEffect(() => {
  api.get('/patients')
    .then(response => setPatients(response.data.data))
    .catch(error => console.error(error));
}, []);
```

📋 IMPLEMENTATION TIMELINE

Phase 1 - Foundation (Weeks 1-2)

- Database schema creation
- Basic CRUD APIs for Users, Patients, Doctors
- Authentication system

- Frontend-backend connection

Phase 2 - Core Features (Weeks 3-4)

- Appointments API
- Medical Records API
- Prescriptions API
- File upload system

Phase 3 - Advanced Features (Weeks 5-6)

- Radiology & DICOM handling
- CDSS AI integration
- Billing & payment gateway
- PDF generation

Phase 4 - Production Ready (Weeks 7-8)

- Security hardening
- HIPAA compliance
- Performance optimization
- Testing & bug fixes
- Deployment

🚀 DEPLOYMENT ARCHITECTURE

```
Frontend (React)
  ↓ HTTPS
Load Balancer
  ↓
API Servers (Node.js/Express) [Auto-scaling]
  ↓
Database Cluster (MongoDB/PostgreSQL) [Master-Replica]
  ↓
File Storage (AWS S3)
  ↓
AI Service (Python/TensorFlow) [Separate]
```

Hosting Recommendations:

- **Frontend**: Vercel, Netlify, or AWS S3 + CloudFront
- **Backend**: AWS EC2/ECS, Google Cloud Run, or Azure App Service
- **Database**: MongoDB Atlas, AWS RDS, or Azure Cosmos DB
- **Files**: AWS S3, Azure Blob Storage, Google Cloud Storage