SynapseAI - Complete Codebase Context Document

> **Purpose**: This document provides comprehensive technical context about the SynapseAI EMR system for developers, LLMs, and technical stakeholders. It covers architecture, data models, workflows, implementation details, and key system features.

Last Updated: October 7, 2025
Version: 1.0.0 (MVP)
Status: Production-Ready MVP

Executive Summary

SynapseAI is an intelligent Electronic Medical Records (EMR) system designed specifically for **mental health practitioners** in India. It combines AI-powered speech-to-text transcription with automated medical report generation to streamline clinical documentation.

Core Value Proposition

- **Real-time Transcription**: Live audio-to-text during consultations (English, Hindi, Marathi code-mixing supported)
- **AI Report Generation**: Automated medical report creation using Google Gemini 2.5 Flash
- **Privacy-First Architecture**: Field-level encryption for all sensitive data (HIPAA/DISHA compliant)
- **Mental Health Focus**: Specialized prompts and workflows for psychiatric consultations
- **Multi-lingual Support**: Handles code-mixing between Hindi, Marathi, and English

Key Statistics

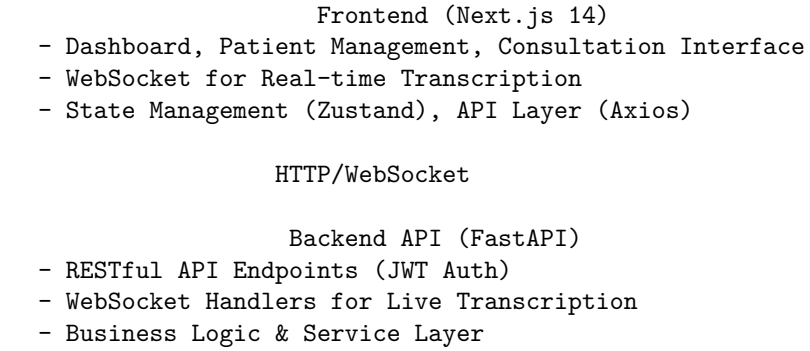
- **Backend**: FastAPI (Python 3.11+) with ~12,000+ lines of production code
- **Frontend**: Next.js 14 (TypeScript/React) with modern UI components
- **Database**: PostgreSQL with field-level AES-256 encryption
- **AI Services**: Google Cloud STT + Gemini 2.5 Flash (Mumbai region)
- **Deployment**: Docker-based with GCP Cloud Run production deployment

---

Table of Contents

System Architecture

High-Level Architecture



Technology Stack

Backend

- **Framework**: FastAPI 0.110+ (async Python web framework)
- **Database**: PostgreSQL 15+ with SQLAlchemy 2.0 ORM
- **Caching/Sessions**: Redis 7.0+
- **Authentication**: JWT (python-jose), bcrypt password hashing

- **Encryption**: AES-256-GCM (Fernet), field-level encryption
- **AI Services**:
  - Google Cloud Speech-to-Text (Vertex AI)
  - Google Gemini 2.5 Flash (via Vertex AI)
- **Database Migrations**: Alembic
- **API Documentation**: OpenAPI/Swagger (auto-generated)
- **Rate Limiting**: slowapi
- **Testing**: pytest, pytest-asyncio

Frontend

- **Framework**: Next.js 14 (App Router, TypeScript)
- **UI Library**: React 18.2, Tailwind CSS 3.3
- **State Management**: Zustand 4.4 (with persist middleware)
- **HTTP Client**: Axios 1.6
- **Forms**: react-hook-form + Zod validation
- **Icons**: Heroicons v2, Lucide React
- **Notifications**: react-hot-toast
- **Animations**: Framer Motion
- **Date Handling**: date-fns

Infrastructure & DevOps

- **Containerization**: Docker + Docker Compose
- **Production Platform**: Google Cloud Run (serverless containers)
- **CI/CD**: Cloud Build (cloudbuild.yaml)
- **Monitoring**: Structured logging with Python logging + Google Cloud Logging
- **Database Hosting**: Cloud SQL (PostgreSQL)
- **Secrets Management**: Google Cloud Secret Manager

System Components

1. API Layer (`backend/app/api/`)

- **RESTful Endpoints**: `/api/v1/*` - All HTTP endpoints
- **WebSocket Handlers**: `/ws/*` - Real-time communication
- **Endpoint Groups**:
  - `/auth` - Authentication (login, logout, token management)
  - `/patients` - Patient CRUD operations
  - `/sessions` - Consultation session management
  - `/consultation` - Active consultation operations
  - `/reports` - Medical report generation & management
  - `/templates` - Report template customization
  - `/admin` - Admin operations (doctor verification)
  - `/doctor` - Doctor registration & profile management
  - `/profile` - User profile management

- `/analytics` - Dashboard analytics

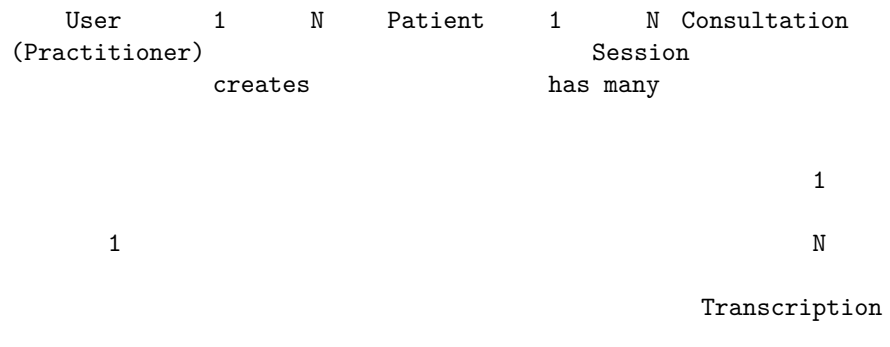
2. **Service Layer (`backend/app/services/`)**
   - **Authentication Service**: User login, token management, password handling
   - **STT Service**: Google Cloud Speech-to-Text integration
   - **Gemini Service**: AI report generation with mental health prompts
   - **Patient Service**: Patient data management
   - **Session Service**: Consultation session lifecycle
   - **Report Service**: Report generation and management
   - **Email Service**: SMTP-based email notifications
   - **Admin Service**: Doctor application review workflow

3. **Data Layer (`backend/app/models/`)**
   - **User Models**: `User`, `UserProfile`, `DoctorProfile`
   - **Patient Models**: `Patient` (fully encrypted demographics)
   - **Session Models**: `ConsultationSession`, `Transcription`
   - **Report Models**: `Report`, `ReportTemplate`
   - **Supporting Models**: `Appointment`, `Bill`, `AuditLog`, `EmailQueue`

4. **Core Infrastructure (`backend/app/core/`)**
   - **Configuration**: Environment-based settings (Pydantic)
   - **Database**: SQLAlchemy session management, connection pooling
   - **Encryption**: Field-level AES-256 encryption utilities
   - **Security**: JWT handling, password hashing, CORS, rate limiting
   - **Middleware**: Request ID tracking, audit logging, error handling
   - **Validation**: Input validation schemas

Data Models & Database Schema

Entity Relationship Overview



1 1

UserProfile Report

1

DoctorProfile
(if role=doctor)

Core Models

1. User & Authentication User Model (users table)

```
{
    "id": "uuid",
    "email": "encrypted_string",       # AES-256 encrypted
    "email_hash": "sha256_hash",       # For lookups
    "password_hash": "bcrypt_hash",     # bcrypt with 12 rounds
    "role": "admin|doctor|receptionist",
    "is_verified": boolean,
    "is_active": boolean,
    "is_locked": boolean,
    "doctor_status": "pending|verified|rejected",  # For doctor role
    "password_reset_required": boolean,
    "mfa_enabled": boolean,
    "mfa_secret": "encrypted_string",
    "last_login": timestamp,
    "failed_login_attempts": int,
    "locked_until": timestamp,
    "created_at": timestamp,
    "updated_at": timestamp
}
```

UserProfile Model (user_profiles table)

```
{
    "id": "uuid",
    "user_id": "uuid (FK → users)",
    "first_name": "encrypted",
```

```
    "last_name": "encrypted",
    "phone": "encrypted",
    "license_number": "encrypted",     # For doctors
    "specialization": "encrypted",
    "clinic_name": "encrypted",
    "clinic_address": "text",
    "logo_url": "string",              # Professional logo for reports
    "address_line1": "encrypted",
    "city": "encrypted",
    "state": "encrypted",
    "postal_code": "encrypted",
    "timezone": "UTC",
    "language": "en"
}
```

DoctorProfile Model (`doctor_profiles` table)

```
{
    "id": "uuid",
    "user_id": "uuid (FK → users)",
    "full_name": "string",
    "medical_registration_number": "string (unique)",
    "state_medical_council": "string",
    "clinic_name": "string",
    "clinic_address": "text",
    "specializations": ["array"],
    "years_of_experience": int,
    "digital_signature_url": "string",
    "application_date": timestamp,
    "verification_date": timestamp,
    "verified_by_admin_id": "uuid",
    "rejection_reason": "text",
    "profile_completed": boolean
}
```

2. Patient Management Patient Model (`patients` table) - **ALL PII fields encrypted**

```
{
    "id": "uuid",
    "patient_id": "PT{RANDOM}",        # Visible ID like PT7A8B9C
    "first_name": "encrypted",
    "last_name": "encrypted",
    "date_of_birth": "encrypted",
    "age": "calculated property",
    "gender": "encrypted",
    "phone_primary": "encrypted",
```

```
    "phone_secondary": "encrypted",
    "email": "encrypted",
    "address_line1": "encrypted",
    "city": "encrypted",
    "state": "encrypted",
    "postal_code": "encrypted",
    "country": "encrypted",
    "emergency_contact_name": "encrypted",
    "emergency_contact_phone": "encrypted",
    "blood_group": "encrypted",
    "allergies": "encrypted (comma-separated)",
    "medical_history": "encrypted",
    "current_medications": "encrypted",
    "insurance_provider": "encrypted",
    "insurance_policy_number": "encrypted",
    "created_by": "uuid (FK → users)",
    # Search indexes (hashed for privacy-preserving search)
    "name_hash": "sha256",
    "phone_hash": "sha256",
    "email_hash": "sha256"
}
```

3. Consultation & Sessions **ConsultationSession Model** (`consultation_sessions` table)

```
{
    "id": "uuid",
    "session_id": "CS-YYYYMMDD-{RANDOM}",  # Like CS-20250107-A1B2C3D4
    "patient_id": "uuid (FK → patients)",
    "doctor_id": "uuid (FK → users)",
    "session_type": "new_patient|followup|consultation|emergency",
    "status": "in_progress|paused|completed|cancelled|error",
    "started_at": "iso_timestamp",
    "ended_at": "iso_timestamp",
    "total_duration": float,              # Minutes
    "chief_complaint": "encrypted",
    "notes": "encrypted",
    "audio_file_url": "gcs_url",          # Google Cloud Storage
    "audio_duration": float,               # Seconds
    "transcription_confidence": float,     # 0-1
    "recording_settings": {"json"},
    "stt_settings": {"json"}
}
```

Transcription Model (`transcriptions` table)

```
{
```

```
    "id": "uuid",
    "session_id": "uuid (FK → consultation_sessions)",
    "transcript_text": "encrypted",        # Full transcript
    "original_transcript": "encrypted",     # Before corrections
    "transcript_segments": [               # JSON array with timing
        {
            "text": "segment text",
            "start_time": 0.0,
            "end_time": 5.2,
            "confidence": 0.95,
            "language": "hi-IN"
        }
    ],
    "processing_status": "pending|processing|completed|failed",
    "stt_service": "google_stt",
    "stt_model": "medical_conversation",
    "stt_language": "hi-IN|en-IN|mr-IN",
    "confidence_score": float,
    "word_count": int,
    "manually_corrected": boolean,
    "corrected_by": "uuid (FK → users)"
}
```

4. Reports & Templates Report Model (`reports` table)

```
{
    "id": "uuid",
    "session_id": "uuid (FK → consultation_sessions)",
    "transcription_id": "uuid (FK → transcriptions)",
    "template_id": "uuid (FK → report_templates)",
    "report_type": "consultation|followup|diagnostic|treatment_plan",
    "status": "pending|generating|completed|failed|signed",
    "version": int,
    "patient_status": "improving|stable|worse",  # For follow-ups
    "generated_content": "encrypted",            # Main report text
    "structured_data": {                         # JSON with extracted data
        "chief_complaint": "...",
        "history_present_illness": "...",
        "mental_status_exam": {...},
        "assessment": "...",
        "plan": {...}
    },
    "ai_model": "gemini-2.5-flash",
    "confidence_score": float,
    "generation_duration": int,                  # Seconds
    "manually_edited": boolean,
```

```
    "reviewed_by": "uuid",
    "signed_by": "uuid",
    "signed_at": "iso_timestamp"
}
```

ReportTemplate Model (`report_templates` table)

```
{
    "id": "uuid",
    "doctor_id": "uuid (FK → users)",
    "template_name": "string",
    "template_description": "text",
    "report_type": "consultation|followup|...",
    "template_structure": {              # JSON schema
        "sections": [
            {
                "id": "chief_complaint",
                "title": "Chief Complaint",
                "required": true,
                "type": "text"
            },
            ...
        ]
    },
    "ai_prompt_template": "text",        # LLM prompt with variables
    "is_default": boolean,
    "is_active": boolean,
    "usage_count": int
}
```

Database Encryption Strategy

Field-Level Encryption: - **Algorithm**: AES-256-GCM (using Python Fernet) - **Key Management**: Environment variables (production: Cloud KMS) - **Encrypted Fields**: All PII data (names, DOB, contact info, medical data) - **Search Strategy**: SHA-256 hashes of sensitive fields for lookups

Example Encryption Flow:

```
# Writing
plaintext = "John Doe"
encrypted = fernet.encrypt(plaintext.encode())
name_hash = sha256(plaintext.lower()).hexdigest()
# Store: encrypted value + hash for search

# Searching
search_term = "John Doe"
search_hash = sha256(search_term.lower()).hexdigest()
```

```
# Query: WHERE name_hash = search_hash

# Reading
encrypted_value = db_query_result
plaintext = fernet.decrypt(encrypted_value).decode()
```

Core Workflows

1. Complete Consultation Workflow

```
PHASE 1: Patient Management

1. Doctor logs in → JWT token issued
2. Search existing patient OR create new patient
3. Patient profile loaded (decrypted from database)

                              ↓

PHASE 2: Session Start

1. POST /api/v1/consultation/start
   {
     "patient_id": "uuid",
     "session_type": "followup",
     "chief_complaint": "Patient complaint"
   }
2. Backend creates ConsultationSession record
3. Initializes STT session with Google Cloud
4. Returns session_id to frontend

                              ↓

PHASE 3: Live Transcription (WebSocket)

1. Frontend establishes WebSocket connection
   ws://localhost:8080/ws/consultation/{session_id}
2. Audio Recording:
   - Browser MediaRecorder captures audio (WebM Opus)
   - Optional: RNNoise noise reduction in browser
   - Chunks sent to backend every 100-500ms
3. Real-time STT Processing:
   - Backend streams audio to Google Cloud STT
   - Interim results sent to frontend (real-time display)
   - Final results saved to database
```

```
4. Multi-language Support:
   - Detects code-mixing (Hindi/Marathi/English)
   - Handles transliteration (Devanagari → Roman)
5. Doctor can pause/resume session

                          ↓

PHASE 4: Session End & Transcription Finalization

1. POST /api/v1/consultation/{session_id}/stop
2. Backend:
   - Stops audio recording
   - Finalizes transcription in database
   - Updates session status to 'completed'
   - Calculates session duration
3. Returns complete transcript to frontend

                          ↓

PHASE 5: AI Report Generation

1. POST /api/v1/reports/generate
   {
     "session_id": "uuid",
     "transcription_id": "uuid",
     "template_id": "uuid (optional)",
     "session_type": "followup"
   }
2. Backend Gemini Service:
   - Loads mental health prompt template
   - Injects transcript with language context
   - Calls Gemini 2.5 Flash API (Mumbai region)
   - Parses structured response
3. Report Structure Generated:
   ## CURRENT SITUATION
   - Present concerns and symptoms
   ## MENTAL STATUS EXAMINATION
   - Mood, affect, thought process
   ## SLEEP & PHYSICAL HEALTH
   - Sleep patterns, appetite
   ## MEDICATION & TREATMENT
   - Current medications, compliance
   ## RISK ASSESSMENT
   - Side effects, suicide risk, protective factors
4. Report saved to database (encrypted)
```

```
                              ↓

PHASE 6: Report Review & Editing

1. Doctor reviews generated report
2. Can make manual corrections
3. PUT /api/v1/reports/{report_id}
   - Updates report content
   - Marks as manually_edited = true
4. Doctor digitally signs report
   - POST /api/v1/reports/{report_id}/sign
   - Updates status to 'signed'
   - Records signed_at timestamp

                              ↓

PHASE 7: Export & Follow-up

1. Export report to PDF
2. Schedule follow-up appointment (optional)
3. Generate bill (optional)
4. Update patient progress status
```

2. Doctor Registration & Verification Workflow

```
STEP 1: Doctor Self-Registration

POST /api/v1/doctor/register
{
  "email": "doctor@example.com",
  "full_name": "Dr. John Smith",
  "medical_registration_number": "MH12345",
  "state_medical_council": "Maharashtra"
}

Backend:
1. Creates User (role=doctor, doctor_status=pending)
2. Creates DoctorProfile with application details
3. Sends email to admin for review
4. Doctor cannot login yet (pending verification)

                          ↓

STEP 2: Admin Reviews Application
```

```
Admin Dashboard:
GET /api/v1/admin/doctor-applications?status=pending

Admin Actions:
A) Approve:
   POST /api/v1/admin/doctor-applications/{id}/approve
   - Sets doctor_status = 'verified'
   - Generates temporary password
   - Sends welcome email with login credentials

B) Reject:
   POST /api/v1/admin/doctor-applications/{id}/reject
   { "reason": "Invalid credentials" }
   - Sets doctor_status = 'rejected'
   - Sends rejection email

                        ↓

STEP 3: Doctor First Login & Profile Completion

POST /api/v1/auth/login
{
  "email": "doctor@example.com",
  "password": "temporary_password"
}

Response includes:
{ "password_reset_required": true }

Frontend redirects to /auth/change-password

After password change:
- Redirect to /doctor/complete-profile
- Doctor adds: clinic info, specializations, logo
- POST /api/v1/profile (updates UserProfile)
- Marks profile_completed = true

                        ↓

STEP 4: Full Access Granted

Doctor can now:
- Access dashboard
- Create/manage patients
- Conduct consultations
```

```
- Generate reports with their logo
```

3. Patient Search & Privacy-Preserving Lookup

```
Frontend Search:
User types: "John Doe"
   ↓
POST /api/v1/patients/search
{
  "query": "John Doe",
  "search_type": "name"  // or "phone", "email"
}
   ↓
Backend Processing:
1. Generate search hash
   search_hash = sha256("john doe").hexdigest()

2. Database query
   SELECT * FROM patients
   WHERE name_hash = search_hash
   AND created_by = current_user.id

3. Decrypt results
   for patient in results:
       patient.first_name = decrypt(patient.first_name)
       patient.last_name = decrypt(patient.last_name)
       # ... decrypt all encrypted fields

4. Return decrypted patient data
   ↓
Frontend displays results
```

Authentication & Security

Authentication Flow

1. Login Process

```
POST /api/v1/auth/login
{
  "email": "doctor@example.com",
  "password": "secure_password"
}
```

```
Backend Process:
1. Hash email: sha256(email.lower())
2. Query User by email_hash
3. Verify password: bcrypt.checkpw(password, user.password_hash)
4. Generate JWT tokens:
   - Access Token (30 minutes)
   - Refresh Token (7 days)
5. Update last_login timestamp
6. Log audit event

Response:
{
  "status": "success",
  "data": {
    "access_token": "eyJ0eXAi...",
    "refresh_token": "eyJ0eXAi...",
    "user_id": "uuid",
    "role": "doctor",
    "password_reset_required": false
  }
}
```

2. Token Structure

```
# Access Token Payload
{
  "sub": "user_id",           # Subject (user ID)
  "email": "encrypted_email",
  "role": "doctor",
  "exp": timestamp,            # Expiration (30 min)
  "iat": timestamp,            # Issued at
  "type": "access"
}

# Refresh Token Payload
{
  "sub": "user_id",
  "exp": timestamp,            # Expiration (7 days)
  "iat": timestamp,
  "type": "refresh"
}
```

3. Protected Endpoint Access

```
# Every protected endpoint uses dependency injection

@router.get("/patients")
async def get_patients(
```

```
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    # current_user is automatically extracted from JWT
    # and validated before this code runs
    ...

# get_current_user dependency:
1. Extract Authorization header
2. Verify token signature (JWT_SECRET_KEY)
3. Check expiration
4. Query user from database
5. Verify user is active
6. Return user object
```

Security Features

1. Field-Level Encryption

- **All PII encrypted**: Patient data, user emails, sensitive fields
- **Encryption at rest**: Data encrypted before database write
- **Transparent decryption**: Automatic decryption on read
- **Key rotation support**: Supports key versioning

2. Password Security

- **Hashing**: bcrypt with 12 rounds
- **No plain text storage**: Never stored unencrypted
- **Password policies**: Enforced at application level
- **Reset tokens**: Secure, time-limited reset links

3. Rate Limiting

```
# Applied globally and per-endpoint
@limiter.limit("100/minute")  # 100 requests per minute
@limiter.limit("1000/hour")   # 1000 requests per hour

# Critical endpoints have stricter limits
@limiter.limit("5/minute")    # Login endpoint
```

4. Audit Logging

```
# Every significant action logged
AuditLog {
  "event_type": "PATIENT_CREATED",
  "user_id": "uuid",
  "resource_type": "patient",
```

```
  "resource_id": "uuid",
  "details": {"patient_id": "PT123"},
  "ip_address": "192.168.1.1",
  "user_agent": "...",
  "timestamp": "2025-01-07T10:30:00Z"
}

# Audit events include:
- LOGIN, LOGOUT
- PATIENT_CREATED, PATIENT_UPDATED
- CONSULTATION_STARTED, CONSULTATION_ENDED
- REPORT_GENERATED, REPORT_SIGNED
- ADMIN_ACTION (approve/reject doctor)
```

5. CORS & Security Headers

```
# CORS Configuration
ALLOWED_ORIGINS = ["http://localhost:3000", "https://yourdomain.com"]
ALLOWED_METHODS = ["GET", "POST", "PUT", "DELETE"]
ALLOW_CREDENTIALS = True

# Security Headers (auto-added to all responses)
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000
Content-Security-Policy: default-src 'self'
```

API Documentation

Base URLs

- **Development**: `http://localhost:8080/api/v1`
- **Production**: `https://your-domain.com/api/v1`

Authentication Endpoints

POST `/auth/login` Authenticate user and receive JWT tokens.

Request:

```
{
  "email": "doctor@example.com",
  "password": "secure_password"
}
```

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
    "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
    "user_id": "123e4567-e89b-12d3-a456-426614174000",
    "role": "doctor",
    "password_reset_required": false
  }
}
```

POST `/auth/logout` Invalidate current session.

Headers: `Authorization: Bearer {access_token}`

Response: (200 OK)

```
{
  "status": "success",
  "message": "Logged out successfully"
}
```

GET `/auth/validate-token` Validate current access token.

Headers: `Authorization: Bearer {access_token}`

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "valid": true,
    "user_id": "uuid",
    "role": "doctor",
    "email": "doctor@example.com"
  }
}
```

Patient Endpoints

POST `/patients` Create new patient.

Request:

```
{
  "first_name": "John",
  "last_name": "Doe",
  "date_of_birth": "1990-05-15",
  "gender": "male",
  "phone_primary": "+919876543210",
```

```
  "email": "john@example.com",
  "address_line1": "123 Main St",
  "city": "Mumbai",
  "state": "Maharashtra",
  "postal_code": "400001",
  "blood_group": "A+",
  "allergies": "Penicillin",
  "medical_history": "Hypertension diagnosed 2015"
}
```

Response: (201 Created)

```
{
  "status": "success",
  "data": {
    "id": "uuid",
    "patient_id": "PT7A8B9C",
    "full_name": "John Doe",
    "age": 34,
    "created_at": "2025-01-07T10:30:00Z"
  }
}
```

GET `/patients/{patient_id}` Get patient details by ID.

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "id": "uuid",
    "patient_id": "PT7A8B9C",
    "full_name": "John Doe",
    "first_name": "John",
    "last_name": "Doe",
    "date_of_birth": "1990-05-15",
    "age": 34,
    "gender": "male",
    "phone_primary": "+919876543210",
    "email": "john@example.com",
    "contact_info": {...},
    "medical_summary": {...},
    "created_at": "2025-01-07T10:30:00Z"
  }
}
```

GET `/patients/list` Get paginated list of patients.

Query Parameters: - `limit`: Number of results (default: 20) - `offset`: Pagination offset (default: 0) - `search`: Search query (optional)

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "patients": [
      {
        "id": "uuid",
        "patient_id": "PT7A8B9C",
        "full_name": "John Doe",
        "age": 34,
        "gender": "male",
        "phone_primary": "+919876543210",
        "last_visit": "2025-01-05T14:20:00Z"
      }
    ],
    "total": 45,
    "limit": 20,
    "offset": 0
  }
}
```

Consultation Endpoints

POST `/consultation/start` Start new consultation session.

Request:

```
{
  "patient_id": "uuid",
  "session_type": "followup",
  "chief_complaint": "Patient reports anxiety symptoms"
}
```

Response: (201 Created)

```
{
  "status": "success",
  "data": {
    "session_id": "CS-20250107-A1B2C3D4",
    "consultation_id": "uuid",
    "patient_id": "uuid",
    "status": "in_progress",
    "started_at": "2025-01-07T10:35:00Z",
    "websocket_url": "/ws/consultation/CS-20250107-A1B2C3D4"
```

```
  }
}
```

POST `/consultation/{session_id}/stop` End consultation session.

Request:

```
{
  "notes": "Session completed successfully. Patient responsive."
}
```

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "session_id": "CS-20250107-A1B2C3D4",
    "status": "completed",
    "duration": 23.5,
    "transcription_id": "uuid",
    "transcript_preview": "Doctor: How are you feeling today?..."
  }
}
```

GET `/consultation/history/{patient_id}` Get consultation history for patient.

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "sessions": [
      {
        "session_id": "CS-20250107-A1B2C3D4",
        "date": "2025-01-07",
        "session_type": "followup",
        "duration": 23.5,
        "has_report": true,
        "status": "completed"
      }
    ],
    "total": 8
  }
}
```

Report Endpoints

POST `/reports/generate` Generate AI-powered medical report from transcription.

Request:

```
{
  "session_id": "uuid",
  "transcription_id": "uuid",
  "template_id": "uuid (optional)",
  "session_type": "followup"
}
```

Response: (201 Created)

```
{
  "status": "success",
  "data": {
    "report_id": "uuid",
    "generated_content": "## CURRENT SITUATION\n- Patient reports...",
    "structured_data": {
      "chief_complaint": "Anxiety symptoms",
      "mental_status_exam": {...}
    },
    "confidence_score": 0.92,
    "generation_duration": 8
  }
}
```

GET `/reports/{report_id}` Get report details.

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "id": "uuid",
    "session_id": "uuid",
    "generated_content": "...",
    "structured_data": {...},
    "status": "completed",
    "manually_edited": false,
    "signed_by": null,
    "created_at": "2025-01-07T11:00:00Z"
  }
}
```

POST `/reports/{report_id}/sign` Digitally sign report.

Response: (200 OK)

```
{
  "status": "success",
  "data": {
    "report_id": "uuid",
    "status": "signed",
    "signed_by": "uuid",
    "signed_at": "2025-01-07T11:15:00Z"
  }
}
```

WebSocket Endpoints

WS `/ws/consultation/{session_id}` Real-time consultation WebSocket for audio streaming and transcription.

Connection:

```
const ws = new WebSocket(
  `ws://localhost:8080/ws/consultation/${sessionId}`,
  ['authorization', accessToken]
);
```

Client → Server Messages:

```
// Send audio chunk (binary)
ws.send(audioBlob);

// Control messages (JSON)
ws.send(JSON.stringify({
  type: "pause_recording"
}));

ws.send(JSON.stringify({
  type: "resume_recording"
}));

ws.send(JSON.stringify({
  type: "stop_recording"
}));
```

Server → Client Messages:

```
// Connection confirmation
{
  "type": "connected",
  "session_id": "CS-20250107-A1B2C3D4",
  "timestamp": "2025-01-07T10:35:00Z"
```

```
}

// Interim transcription (real-time)
{
  "type": "transcription",
  "data": {
    "type": "interim",
    "transcript": "Doctor: How are you",
    "confidence": 0.85,
    "timestamp": "2025-01-07T10:35:05Z"
  }
}

// Final transcription (saved to DB)
{
  "type": "transcription",
  "data": {
    "type": "final",
    "transcript": "Doctor: How are you feeling today?",
    "confidence": 0.95,
    "word_count": 6,
    "timestamp": "2025-01-07T10:35:08Z"
  }
}

// Error
{
  "type": "error",
  "message": "STT service unavailable",
  "timestamp": "2025-01-07T10:35:10Z"
}
```

Frontend Architecture

Application Structure

```
frontend/src/
  app/                          # Next.js 14 App Router
    (auth)/                     # Auth route group
      register/
    admin/                      # Admin dashboard
      applications/             # Doctor applications
      dashboard/
    auth/                       # Auth pages
      login/
```

```
        change-password/
    dashboard/                 # Main doctor dashboard
        patients/              # Patient management
            [id]/              # Patient detail page
            new/               # New patient form
        profile/               # User profile
        settings/
    doctor/                    # Doctor-specific pages
        complete-profile/
    intake/                    # Patient intake forms
    landing/                   # Public landing page
components/                    # React components
    admin/                     # Admin components
        ApplicationCard.tsx
        ApplicationTable.tsx
        ApprovalModal.tsx
        RejectionModal.tsx
    consultation/              # Consultation UI
        AudioRecorder.tsx      # Main recording component
        PatientSelectionModal.tsx
        MedicationModal.tsx
        EditableTranscript.tsx
        MedicalReportDisplay.tsx
    dashboard/                 # Dashboard components
        DashboardHeader.tsx
        DashboardSidebar.tsx
    ui/                        # Reusable UI components
        Button.tsx
        Card.tsx
        Input.tsx
        Select.tsx
        LoadingSpinner.tsx
    landing/                   # Landing page components
services/                      # API & business logic
    api.ts                     # Main API service
    consultationService.ts     # Consultation-specific API
    symptoms.ts                # Symptom search service
    rnnoise.ts                 # Audio noise reduction
store/                         # State management
    authStore.ts               # Zustand auth store
types/                         # TypeScript types
    index.ts
    consultation.ts
    report.ts
    symptom.ts
utils/                         # Utility functions
```

Key Frontend Components

1. AudioRecorder Component Main component for consultation audio recording and transcription.

Features: - Real-time audio capture (WebRTC MediaRecorder) - WebSocket connection for live transcription - Noise reduction (RNNoise WASM module) - Audio device selection - Language selection (English, Hindi, Marathi) - Pause/Resume functionality - Audio level visualization - Code-mixing detection

Usage:

```
<AudioRecorder
  sessionId="CS-20250107-A1B2C3D4"
  isRecording={isRecording}
  selectedAudioDevice="default"
  selectedLanguage="hi-IN"
  onStart={() => handleStartRecording()}
  onStop={() => handleStopRecording()}
  onPause={() => handlePause()}
  onResume={() => handleResume()}
  onTranscriptionUpdate={(text, isFinal) => {
    if (isFinal) {
      setTranscript(prev => prev + " " + text);
    } else {
      setInterimText(text);
    }
  }}
/>
```

2. PatientSelectionModal Modal for selecting patient before starting consultation.

Features: - Search patients by name/ID - Display patient demographics - Recent consultation history - Create new patient inline

3. DashboardHeader & Sidebar Main navigation and layout components.

Features: - User profile dropdown - Logout functionality - Quick actions menu - Responsive mobile menu

State Management (Zustand)

Auth Store (`authStore.ts`):

```
interface AuthState {
  user: User | null;
  profile: UserProfile | null;
  isAuthenticated: boolean;
```

```
  isLoading: boolean;

  // Actions
  login: (credentials: LoginCredentials) => Promise<boolean>;
  logout: () => void;
  checkAuth: () => Promise<void>;
  updateProfile: (data: Partial<UserProfile>) => Promise<boolean>;
  hasRole: (role: string) => boolean;
}

// Usage
const { user, login, logout } = useAuthStore();

// Login
await login({ email, password });

// Check role
if (user?.role === 'admin') {
  // Show admin features
}
```

API Service Layer

Main API Service (`services/api.ts`):

```
class ApiService {
  private api: AxiosInstance;

  constructor() {
    this.api = axios.create({
      baseURL: process.env.NEXT_PUBLIC_API_URL,
      headers: { 'Content-Type': 'application/json' }
    });

    // Auto-attach JWT token
    this.api.interceptors.request.use((config) => {
      const token = localStorage.getItem('access_token');
      if (token) {
        config.headers.Authorization = `Bearer ${token}`;
      }
      return config;
    });

    // Handle 401 unauthorized
    this.api.interceptors.response.use(
      (response) => response,
```

```
      (error) => {
        if (error.response?.status === 401) {
          localStorage.removeItem('access_token');
          window.location.href = '/auth/login';
        }
        return Promise.reject(error);
      }
    );
  }

  // Generic methods
  async get(endpoint: string, params?: any) { ... }
  async post(endpoint: string, data?: any) { ... }
  async put(endpoint: string, data?: any) { ... }
  async delete(endpoint: string) { ... }

  // Specific endpoints
  async getPatients(params: { limit, offset, search }) { ... }
  async createPatient(patientData: any) { ... }
  async startConsultation(data: any) { ... }
  async generateReport(data: any) { ... }
}

export const apiService = new ApiService();
```

Routing & Navigation

Protected Routes (`middleware.ts`):

```
// Middleware runs on every route
export function middleware(request: NextRequest) {
  const token = request.cookies.get('access_token');
  const path = request.nextUrl.pathname;

  // Public routes
  if (path === '/' || path.startsWith('/landing')) {
    return NextResponse.next();
  }

  // Auth routes (redirect if already logged in)
  if (path.startsWith('/auth/login') && token) {
    return NextResponse.redirect(new URL('/dashboard', request.url));
  }

  // Protected routes (require authentication)
  if (!token && path.startsWith('/dashboard')) {
```

```
    return NextResponse.redirect(new URL('/auth/login', request.url));
  }

  return NextResponse.next();
}
```

AI & ML Services

Google Cloud Speech-to-Text

Configuration (backend/app/services/stt_service.py):

```python
class STTService:
    def __init__(self):
        credentials = service_account.Credentials.from_service_account_file(
            'gcp-credentials.json',
            scopes=['https://www.googleapis.com/auth/cloud-platform']
        )
        self.client = speech.SpeechClient(credentials=credentials)

    def get_streaming_config(self, language_code: str = "hi-IN"):
        config = RecognitionConfig(
            encoding=RecognitionConfig.AudioEncoding.WEBM_OPUS,
            sample_rate_hertz=48000,
            language_code=language_code,
            alternative_language_codes=["hi-IN", "mr-IN", "en-IN"],  # Code-mixing
            model="medical_conversation",    # Medical-specific model
            use_enhanced=True,
            enable_automatic_punctuation=True,
            enable_word_confidence=True,
            enable_word_time_offsets=True,
            profanity_filter=False,          # Medical terms may be flagged
            speech_contexts=[{
                "phrases": [
                    "blood pressure", "hypertension", "diabetes",
                    "medication", "symptoms", "diagnosis"
                ],
                "boost": 10.0
            }]
        )

        return StreamingRecognitionConfig(
            config=config,
            interim_results=True,
            single_utterance=False
```

```
        )
```

Streaming Recognition Flow:

```
async def start_streaming_recognition(
    session_id: str,
    audio_stream: AsyncGenerator
):
    streaming_config = self.get_streaming_config()

    # Stream audio to Google STT
    responses = self.client.streaming_recognize(
        config=streaming_config,
        requests=audio_stream
    )

    for response in responses:
        for result in response.results:
            if result.is_final:
                # Final transcription - save to database
                transcript = result.alternatives[0].transcript
                confidence = result.alternatives[0].confidence
                yield {
                    "type": "final",
                    "transcript": transcript,
                    "confidence": confidence,
                    "word_count": len(transcript.split())
                }
            else:
                # Interim result - send to frontend for live display
                yield {
                    "type": "interim",
                    "transcript": result.alternatives[0].transcript,
                    "confidence": result.alternatives[0].confidence
                }
```

Google Gemini 2.5 Flash (Report Generation)

Service Configuration (backend/app/services/gemini_service.py):

```
class GeminiService:
    def __init__(self):
        credentials = service_account.Credentials.from_service_account_file(
            'gcp-credentials.json',
            scopes=['https://www.googleapis.com/auth/generative-language.retriever']
        )
```

```
        genai.configure(credentials=credentials)
        self.model = genai.GenerativeModel("gemini-2.5-flash")
        self.project = "synapse-product-1"
        self.location = "asia-south1"  # Mumbai, India

    async def generate_medical_report(
        self,
        transcription: str,
        session_type: str = "follow_up"
    ) -> dict:
        prompt = self._get_follow_up_prompt(transcription) if session_type == "follow_up" \
                 else self._get_new_patient_prompt(transcription)

        response = self.model.generate_content(prompt)

        return {
            "status": "success",
            "report": response.text,
            "model_used": "gemini-2.5-flash",
            "session_type": session_type,
            "generated_at": datetime.now(timezone.utc).isoformat()
        }
```

Mental Health Prompt Template (Follow-up Session):

```
def _get_follow_up_prompt(self, transcription: str) -> str:
    return f"""
You are an experienced mental health professional reviewing a follow-up consultation.

  CRITICAL LANGUAGE & CONTEXT:
- This transcript contains multilingual content: Hindi, Marathi, and English
- Due to speech-to-text limitations, expect:
  * Misspelled words or incorrect transcriptions
  * Missing words or phrases
  * Code-switching between languages
- UNDERSTAND THE CLINICAL INTENT despite these errors
- Use **bold** for important clinical terms, diagnoses, medications, risk factors
- Minimize hallucinations - only include information clearly present or strongly implied

TRANSCRIPT:
{transcription}

TASK: Generate a CONCISE follow-up mental health assessment report:

## CURRENT SITUATION
- Present concerns and symptoms (brief summary)
- Current stressors and triggers
```

```
- Functional impact on daily life

## MENTAL STATUS EXAMINATION
- Mood and affect
- Thought process and insight
- Risk assessment (if applicable)

## SLEEP & PHYSICAL HEALTH
- Sleep patterns and disturbances
- Appetite and energy levels
- Physical symptoms

## MEDICATION & TREATMENT
- Current medications (if mentioned)
- Treatment compliance and concerns

## RISK ASSESSMENT & SIDE EFFECTS
- Side effects of medications
- Risk of suicide or self-harm
- Risk to others
- Protective factors

GUIDELINES:
1. Keep each section to 2-3 bullet points maximum
2. Use professional but concise language
3. Only include information from the transcript
4. Skip sections if no relevant information
5. Focus on key clinical findings and actionable insights
6. No placeholder text or template language
"""
```

Report Generation API Flow:

```
1. Consultation ends → Transcription finalized
2. Doctor clicks "Generate Report"
3. POST /api/v1/reports/generate
4. Backend loads transcription from database (decrypted)
5. Gemini Service:
   - Selects prompt template based on session_type
   - Injects transcript with language context
   - Calls Gemini 2.5 Flash API
   - Receives structured markdown response
6. Parse and structure report
7. Save to database (encrypted)
8. Return to frontend for review
9. Doctor can edit/sign report
```

Deployment & DevOps

Docker Development Environment

docker-compose.yml structure:

```yaml
services:
  db:                    # PostgreSQL 15
    image: postgres:15-alpine
    ports: ["5432:5432"]
    volumes: [postgres_data]

  redis:                 # Redis 7 (session cache)
    image: redis:7-alpine
    volumes: [redis_data]

  backend:               # FastAPI backend
    build: ./backend
    ports: ["8080:8080"]
    depends_on: [db, redis]
    environment:
      DATABASE_URL: postgresql://emr_user:emr_password@db:5432/emr_db
      REDIS_URL: redis://redis:6379/0

  frontend:              # Next.js frontend
    build: ./frontend
    ports: ["3000:3000"]
    depends_on: [backend]
    environment:
      NEXT_PUBLIC_API_URL: http://localhost:8080/api/v1
```

Quick Start Commands:

```bash
# Start all services
docker-compose up -d

# View logs
docker-compose logs -f backend
docker-compose logs -f frontend

# Restart a service
docker-compose restart backend

# Stop all services
docker-compose down
```

```
# Reset database
docker-compose down -v
docker-compose up -d
```

Production Deployment (Google Cloud Run)

Architecture:

```
      Cloud Load Balancer (HTTPS)




  Backend       Frontend
Cloud Run       Cloud Run
Container       Container




Cloud SQL     Redis   Cloud       Secret
PostgreSQL              STT       Manager
```

cloudbuild.yaml (CI/CD Pipeline):

```yaml
steps:
  # Build backend container
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/backend:$SHORT_SHA', './backend']

  # Build frontend container
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/frontend:$SHORT_SHA', './frontend']

  # Push images
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'gcr.io/$PROJECT_ID/backend:$SHORT_SHA']
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'gcr.io/$PROJECT_ID/frontend:$SHORT_SHA']

  # Deploy to Cloud Run
  - name: 'gcr.io/cloud-builders/gcloud'
    args:
```

```
      - 'run'
      - 'deploy'
      - 'synapseai-backend'
      - '--image=gcr.io/$PROJECT_ID/backend:$SHORT_SHA'
      - '--region=asia-south1'
      - '--platform=managed'
      - '--allow-unauthenticated'
```

Deployment Commands:

```
# Manual deployment
gcloud run deploy synapseai-backend \
  --image gcr.io/synapse-product-1/backend:latest \
  --region asia-south1 \
  --platform managed \
  --allow-unauthenticated \
  --set-env-vars DATABASE_URL="..." \
  --set-secrets=JWT_SECRET_KEY=jwt-secret:latest

# Automatic deployment (push to main branch triggers CI/CD)
git push origin main
# Cloud Build automatically builds and deploys
```

Database Migrations (Alembic)

Create New Migration:

```
cd backend

# Auto-generate migration from model changes
alembic revision --autogenerate -m "Add patient_status field to reports"

# Creates: alembic/versions/abc123_add_patient_status.py
```

Apply Migrations:

```
# Development
alembic upgrade head

# Production (in Cloud Run startup script)
alembic upgrade head && uvicorn app.main:app
```

Migration File Structure:

```
def upgrade():
    # Forward migration
    op.add_column('reports',
        sa.Column('patient_status', sa.String(20), nullable=True)
    )
```

```python
def downgrade():
    # Rollback migration
    op.drop_column('reports', 'patient_status')
```

Configuration & Environment

Environment Variables

Backend (`.env`):

```
# Application
APP_NAME="SynapseAI - Intelligent EMR System"
VERSION="1.0.0"
ENVIRONMENT="development"  # or staging, production
DEBUG="true"

# Database
DATABASE_URL="postgresql://emr_user:emr_password@localhost:5432/emr_db"
DATABASE_POOL_SIZE=20
DATABASE_MAX_OVERFLOW=10

# Redis
REDIS_URL="redis://localhost:6379/0"
REDIS_SESSION_DB=1

# Security & Authentication
SECRET_KEY="your-secret-key-32-chars-minimum"
JWT_SECRET_KEY="your-jwt-secret-32-chars-minimum"
JWT_ALGORITHM="HS256"
JWT_ACCESS_TOKEN_EXPIRE_MINUTES=30
JWT_REFRESH_TOKEN_EXPIRE_DAYS=7
BCRYPT_ROUNDS=12

# Encryption
ENCRYPTION_KEY="your-aes-256-encryption-key-base64"
FIELD_ENCRYPTION_KEY="your-field-encryption-key-base64"

# Google Cloud Platform
GCP_CREDENTIALS_PATH="gcp-credentials.json"
GCP_PROJECT_ID="synapse-product-1"

# Google Cloud STT
GOOGLE_STT_MODEL="latest_long"
GOOGLE_STT_PRIMARY_LANGUAGE="hi-IN"
```

```
GOOGLE_STT_ALTERNATE_LANGUAGES="hi-IN,mr-IN,en-IN"
GOOGLE_STT_SAMPLE_RATE=48000
GOOGLE_STT_ENCODING="WEBM_OPUS"

# Gemini AI
GEMINI_MODEL="gemini-2.5-flash"
VERTEX_AI_LOCATION="asia-south1"
GOOGLE_APPLICATION_CREDENTIALS="gcp-credentials.json"

# API Configuration
API_V1_PREFIX="/api/v1"
API_HOST="0.0.0.0"
API_PORT=8080

# CORS
ALLOWED_ORIGINS="http://localhost:3000,http://localhost:8000"
ALLOWED_METHODS="GET,POST,PUT,DELETE,OPTIONS"

# Rate Limiting
RATE_LIMIT_PER_MINUTE=100
RATE_LIMIT_PER_HOUR=1000

# Email (SMTP)
SMTP_HOST="smtp.gmail.com"
SMTP_PORT=587
SMTP_USER="your-email@gmail.com"
SMTP_PASSWORD="your-app-password"
SMTP_FROM_EMAIL="noreply@synapseai.com"
ADMIN_EMAIL="admin@synapseai.com"

# Frontend URL
FRONTEND_URL="http://localhost:3000"

# Logging
LOG_LEVEL="INFO"
ENABLE_AUDIT_LOGGING="true"
AUDIT_LOG_RETENTION_DAYS=2555  # 7 years (HIPAA requirement)
```

Frontend (`.env.local`):

```
# API Configuration
NEXT_PUBLIC_API_URL="http://localhost:8080/api/v1"
NEXT_PUBLIC_WS_URL="ws://localhost:8080"

# Application
NEXT_PUBLIC_APP_ENV="development"
NEXT_PUBLIC_APP_URL="http://localhost:3000"
```

```
# Feature Flags
NEXT_PUBLIC_ENABLE_ANALYTICS="false"
NEXT_PUBLIC_ENABLE_NOISE_REDUCTION="true"
```

GCP Service Account Setup

1. Create Service Account:

```
gcloud iam service-accounts create synapseai-backend \
  --display-name="SynapseAI Backend Service Account"
```

2. Grant Required Permissions:

```
# Cloud SQL Client
gcloud projects add-iam-policy-binding synapse-product-1 \
  --member="serviceAccount:synapseai-backend@synapse-product-1.iam.gserviceaccount.com" \
  --role="roles/cloudsql.client"

# Speech-to-Text User
gcloud projects add-iam-policy-binding synapse-product-1 \
  --member="serviceAccount:synapseai-backend@synapse-product-1.iam.gserviceaccount.com" \
  --role="roles/speech.client"

# Vertex AI User
gcloud projects add-iam-policy-binding synapse-product-1 \
  --member="serviceAccount:synapseai-backend@synapse-product-1.iam.gserviceaccount.com" \
  --role="roles/aiplatform.user"

# Secret Manager Accessor
gcloud projects add-iam-policy-binding synapse-product-1 \
  --member="serviceAccount:synapseai-backend@synapse-product-1.iam.gserviceaccount.com" \
  --role="roles/secretmanager.secretAccessor"
```

3. Download Credentials:

```
gcloud iam service-accounts keys create gcp-credentials.json \
  --iam-account=synapseai-backend@synapse-product-1.iam.gserviceaccount.com
```

Development Guide

Prerequisites

- **Docker Desktop** (for local development)
- **Python 3.11+** (if running backend outside Docker)
- **Node.js 18+** (if running frontend outside Docker)
- **PostgreSQL 15+** (if running database locally)

- **Redis 7+** (if running cache locally)
- **GCP Account** with billing enabled (for AI services)

Local Development Setup

1. Clone Repository:

```
git clone <repository-url>
cd MVP_v0.5
```

2. Start Docker Services:

```
# Start database and Redis
docker-compose up -d db redis

# Wait for services to be healthy
docker-compose ps
```

3. Backend Setup:

```
cd backend

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Setup environment
cp .env.example .env
# Edit .env with your configuration

# Run migrations
alembic upgrade head

# Start backend
uvicorn app.main:app --reload --host 0.0.0.0 --port 8080
```

4. Frontend Setup:

```
cd frontend

# Install dependencies
npm install

# Setup environment
cp .env.example .env.local
# Edit .env.local with your configuration
```

```
# Start development server
npm run dev
```

5. Access Application: - Frontend: http://localhost:3000 - Backend API: http://localhost:8080 - API Docs: http://localhost:8080/api/v1/docs

Code Style & Standards

Backend (Python): - **Formatter**: Black (88 character line length) - **Linter**: Pylint, mypy (type checking) - **Import sorting**: isort

```
# Format code
black backend/

# Type check
mypy backend/

# Sort imports
isort backend/
```

Frontend (TypeScript): - **Formatter**: Prettier - **Linter**: ESLint (Next.js config) - **Type checking**: TypeScript strict mode

```
# Format code
npm run format

# Lint
npm run lint

# Type check
npm run type-check
```

Testing

Backend Tests:

```
cd backend

# Run all tests
pytest

# Run with coverage
pytest --cov=app --cov-report=html

# Run specific test file
pytest tests/test_auth.py
```

```
# Run with verbose output
pytest -v -s
```

Frontend Tests (to be implemented):

```
cd frontend

# Run tests
npm test

# Run with coverage
npm test -- --coverage
```

Database Management

Reset Database:

```
# Stop services
docker-compose down

# Remove volumes (deletes all data)
docker-compose down -v

# Restart services
docker-compose up -d

# Rerun migrations
cd backend
alembic upgrade head
```

Create Database Backup:

```
# Development
docker exec synapseai_postgres pg_dump -U emr_user emr_db > backup.sql

# Production (Cloud SQL)
gcloud sql export sql synapse-db-instance gs://synapse-backups/backup-$(date +%Y%m%d).sql
```

Restore Database:

```
# Development
docker exec -i synapseai_postgres psql -U emr_user emr_db < backup.sql

# Production
gcloud sql import sql synapse-db-instance gs://synapse-backups/backup-20250107.sql
```

Debugging

Backend Debugging:

```python
# Add breakpoint in code
import pdb; pdb.set_trace()

# Or use built-in debugger
breakpoint()

# View logs
docker-compose logs -f backend

# Structured logging
import logging
logger = logging.getLogger(__name__)
logger.info("User logged in", extra={"user_id": user.id})
```

Frontend Debugging:

```javascript
// Console logging
console.log("API response:", response.data);

// React Developer Tools
// Install browser extension for React inspection

// Network debugging
// Use browser DevTools Network tab
```

Common Development Tasks

Add New API Endpoint:

```python
# 1. Create endpoint file
# backend/app/api/api_v1/endpoints/my_feature.py

from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session

router = APIRouter()

@router.get("/my-endpoint")
async def my_endpoint(db: Session = Depends(get_db)):
    return {"status": "success"}

# 2. Register router
# backend/app/api/api_v1/api.py

from app.api.api_v1.endpoints import my_feature

api_router.include_router(
```

```
    my_feature.router,
    prefix="/my-feature",
    tags=["my-feature"]
)
```

Add New Database Model:

```
# 1. Create model
# backend/app/models/my_model.py

from .base import BaseModel
from sqlalchemy import Column, String

class MyModel(BaseModel):
    __tablename__ = "my_table"

    name = Column(String(100), nullable=False)

# 2. Import in models/__init__.py
from .my_model import MyModel

# 3. Create migration
alembic revision --autogenerate -m "Add MyModel"

# 4. Apply migration
alembic upgrade head
```

Add New Frontend Page:

```
# 1. Create page file
# frontend/src/app/my-page/page.tsx

'use client'

export default function MyPage() {
  return (
    <div>
      <h1>My New Page</h1>
    </div>
  );
}

# 2. Add navigation link
# frontend/src/components/dashboard/DashboardSidebar.tsx

<Link href="/my-page">My Page</Link>
```

Project Statistics

Codebase Metrics

- **Total Lines of Code**: ~35,000+
- **Backend Python**: ~12,000+ lines
- **Frontend TypeScript/React**: ~18,000+ lines
- **Configuration & Infrastructure**: ~2,000+ lines
- **Documentation**: ~3,000+ lines

File Counts

- **Backend**:
  - Models: 12 files
  - API Endpoints: 14 files
  - Services: 8 files
  - Core Utilities: 12 files
- **Frontend**:
  - Pages: 25+ routes
  - Components: 60+ components
  - Services: 4 files

Database Schema

- **Tables**: 15 main tables
- **Indexes**: 25+ performance indexes
- **Foreign Keys**: 30+ relationships
- **Encrypted Fields**: 40+ sensitive fields

API Endpoints

- **Total Endpoints**: 50+
- **Authentication**: 5 endpoints
- **Patients**: 8 endpoints
- **Consultations**: 10 endpoints
- **Reports**: 7 endpoints
- **Admin**: 5 endpoints
- **WebSocket**: 2 endpoints

Key Features Summary

Implemented (MVP Ready)

1. **Complete Authentication System**
   - JWT-based authentication
   - Role-based access control (Admin, Doctor, Receptionist)
   - Doctor registration & verification workflow

- Password reset & forced password change

2. **Patient Management**
   - Full CRUD operations
   - Encrypted demographics
   - Privacy-preserving search
   - Medical history tracking
3. **Consultation Sessions**
   - Real-time audio recording
   - Live transcription (Google Cloud STT)
   - Multi-language support (Hindi, Marathi, English)
   - Code-mixing detection
   - Session pause/resume
4. **AI Report Generation**
   - Automated report creation (Gemini 2.5 Flash)
   - Mental health-specific prompts
   - Structured report format
   - Manual editing & signing
5. **Admin Dashboard**
   - Doctor application review
   - Approve/reject workflow
   - Email notifications
6. **Security & Compliance**
   - Field-level AES-256 encryption
   - Comprehensive audit logging
   - Rate limiting
   - CORS & security headers
   - HIPAA/DISHA compliance considerations

Future Enhancements

1. **Advanced Features**
   - Appointment scheduling with calendar
   - Billing & invoice generation
   - Report templates customization
   - Symptom search & ICD-11 integration
2. **AI Enhancements**
   - Multi-turn conversation analysis
   - Risk prediction models
   - Treatment recommendation engine
3. **Integration**
   - EHR/EMR system integration
   - Pharmacy integration
   - Lab results integration
   - Telemedicine capabilities
4. **Analytics**
   - Patient outcome tracking

- Treatment effectiveness metrics
- Practice analytics dashboard

Troubleshooting

Common Issues

1. Database Connection Error

```
Error: could not connect to server: Connection refused
```

Solution:

```
# Check if PostgreSQL is running
docker-compose ps

# Restart database
docker-compose restart db

# Check database logs
docker-compose logs db
```

2. JWT Token Invalid

```
Error: 401 Unauthorized - Invalid token
```

Solution:

```
# Check if JWT_SECRET_KEY is set correctly
# Logout and login again
# Check token expiration (default 30 minutes)
```

3. STT Service Unavailable

```
Error: Failed to start STT session
```

Solution:

```
# Verify GCP credentials
gcloud auth application-default login

# Check service account permissions
gcloud projects get-iam-policy synapse-product-1

# Verify API is enabled
gcloud services list --enabled
```

4. Frontend API Connection Error

```
Error: Network Error / CORS Error
```

Solution:

```
# Check NEXT_PUBLIC_API_URL in frontend .env.local
# Verify backend is running on correct port
# Check ALLOWED_ORIGINS in backend .env
```

5. Database Migration Error

```
Error: Target database is not up to date
```

Solution:

```
# Check current migration version
alembic current

# View migration history
alembic history

# Upgrade to latest
alembic upgrade head

# If stuck, downgrade and re-upgrade
alembic downgrade -1
alembic upgrade head
```

Support & Resources

Documentation

- **API Docs (Swagger)**: http://localhost:8080/api/v1/docs
- **Data Model**: See `DATA_MODEL_AND_ARCHITECTURE.md`
- **Getting Started**: See `GETTING_STARTED.md`
- **Deployment Guide**: See `DEPLOYMENT_GUIDE.md`

External Resources

- **FastAPI**: https://fastapi.tiangolo.com/
- **Next.js**: https://nextjs.org/docs
- **Google Cloud STT**: https://cloud.google.com/speech-to-text/docs
- **Gemini API**: https://ai.google.dev/docs
- **SQLAlchemy**: https://docs.sqlalchemy.org/
- **Tailwind CSS**: https://tailwindcss.com/docs

Contact

- **GitHub Issues**: For bug reports and feature requests
- **Email**: admin@synapseai.com (for support)

License & Credits

Project: SynapseAI - Intelligent EMR System
Version: 1.0.0 (MVP)
License: Proprietary
Copyright: © 2025 SynapseAI

Technologies Used: - FastAPI (MIT License) - Next.js (MIT License) - PostgreSQL (PostgreSQL License) - Google Cloud Platform Services - Tailwind CSS (MIT License) - And many more open-source libraries (see package files)

Document Changelog

Date	Version	Changes
2025-01-07	1.0.0	Initial comprehensive context document created

---

END OF DOCUMENT

This document provides complete technical context for the SynapseAI EMR system. For specific implementation details, refer to the actual source code and inline documentation.