MedVision: Healthcare Platform

Developed by StratX

May 11, 2025

Documentation

For support, contact: vijitverma1023@gmail.com

Contents

1 Project Overview

HealthSync is a comprehensive healthcare platform designed to enhance user interactions with medical services. Users can log in, verify their accounts, access a personalized dashboard, monitor health status, book doctor appointments, engage in real-time chatting and video con- ferencing with doctors, and receive professional health consultancy. The platform leverages AI-based suggestions and exclusive offers to provide a modern, user-centric experience.

1.1 Purpose

This documentation serves as a detailed guide for the HealthSync platform, outlining its features, user flow, technical architecture, and setup instructions. It is intended for developers, administrators, and stakeholders to understand, deploy, and maintain the system.

1.2 Scope

The platform supports two user roles: Patients and Doctors. Patients manage health profiles, book appointments, and access consultancy, chatting, and video conferencing. Doctors provide consultations, manage appointments, and communicate via chat or video. The system uses the MERN stack for authentication and Spring Boot for dashboard, chatting, and video conferencing functionalities.

1.3 Key Features

- **Secure Authentication**: Users sign up or log in with email, password, and role (Patient or Doctor), powered by the MERN stack.
- **OTP Verification**: Doctors undergo email-based OTP verification for enhanced security.
- **Personalized Dashboard**: Displays health status, appointment history, and consultancy options, built with Spring Boot and React.
- **Appointment Booking**: Patients schedule appointments with doctors.
- **Chatting with Doctors**: Real-time text-based communication with doctors via the dashboard.
- **Video Conferencing**: Secure video consultations with doctors, integrated into the plat- form.
- **Health Consultancy**: Expert guidance and AI-based health suggestions.
- **Premium Plans**: Subscription plans (Basic, Advanced, Business) with Razorpay payment integration.
- **Responsive UI**: White-and-blue-themed interface with Framer Motion animations.

2 User Flow

2.1 Account Creation and Login

Users navigate to the login page and select “Sign Up” or “Login”.

- **For Sign Up**:
  - Enter name, email, password, and role (Patient or Doctor).

  - Submit the form to register via the /api/auth/register endpoint (MERN stack).
- **For Login**:
  - Enter email, password, and role.
  - Submit the form to authenticate via the /api/auth/login endpoint (MERN stack).

2.2 OTP Verification (Doctors Only)

After successful login, Doctors receive an OTP via email.

- Enter the OTP and submit via the /api/auth/verify-otp endpoint (MERN stack).
- Upon verification, Doctors are redirected to the dashboard.

2.3 Dashboard

Displays user-specific data:

- Health status (e.g., recent vitals, medical history).
- Appointment history and upcoming appointments.
- AI-based health suggestions (Advanced and Business plans).
- Options to book appointments, initiate chats, or start video

conferences. Powered by Spring Boot backend with React frontend.

2.4 Appointment Booking

Patients select a doctor, view availability, and choose a time slot.

- Submit the booking request via a Spring Boot endpoint (e.g., /api/appointments/book).
- Receive confirmation and view the appointment in the dashboard.

2.5 Chatting with Doctors

From the dashboard, Patients select a doctor and initiate a chat session.

- Real-time messaging handled via a Spring Boot endpoint (e.g., /api/chat/messages).
- Chat history is saved and accessible in the dashboard.

2.6 Video Conferencing

Patients schedule or start a video consultation from the dashboard.

- Join a secure video call via a Spring Boot-integrated service (e.g., /api/video/start).
- Doctors and Patients communicate in real-time, with session details logged.

2.7 Health Consultancy

- Access expert consultancy through video meetings or chat (all plans).
- Receive AI-based suggestions (Advanced and Business plans).
- View exclusive offers for health packages (Best Offers feature).

3 Flowcharts

3.1 User Authentication Flow

The user authentication flow involves users signing up by entering their name, email, password, and role, or logging in with their email, password, and role. Doctors require additional OTP verification post-login to access the dashboard.

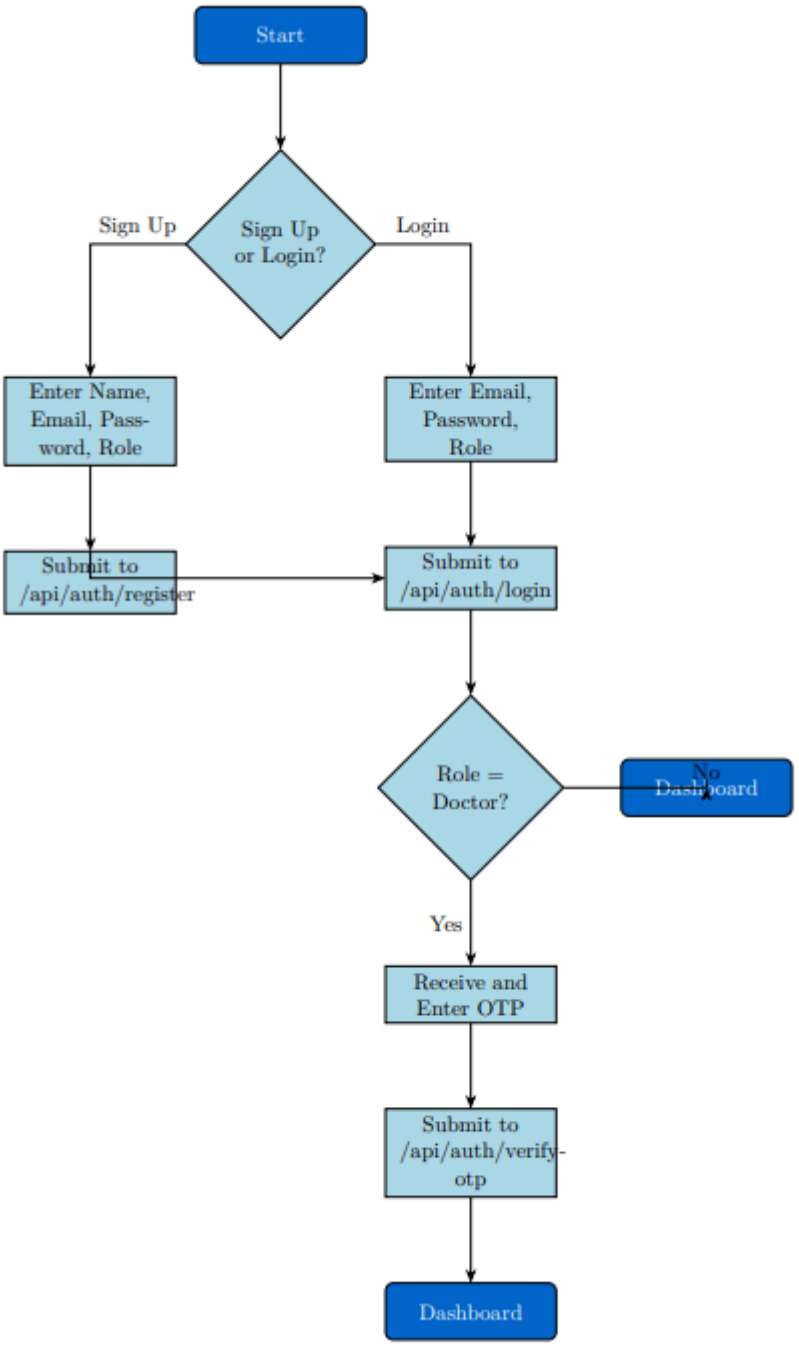
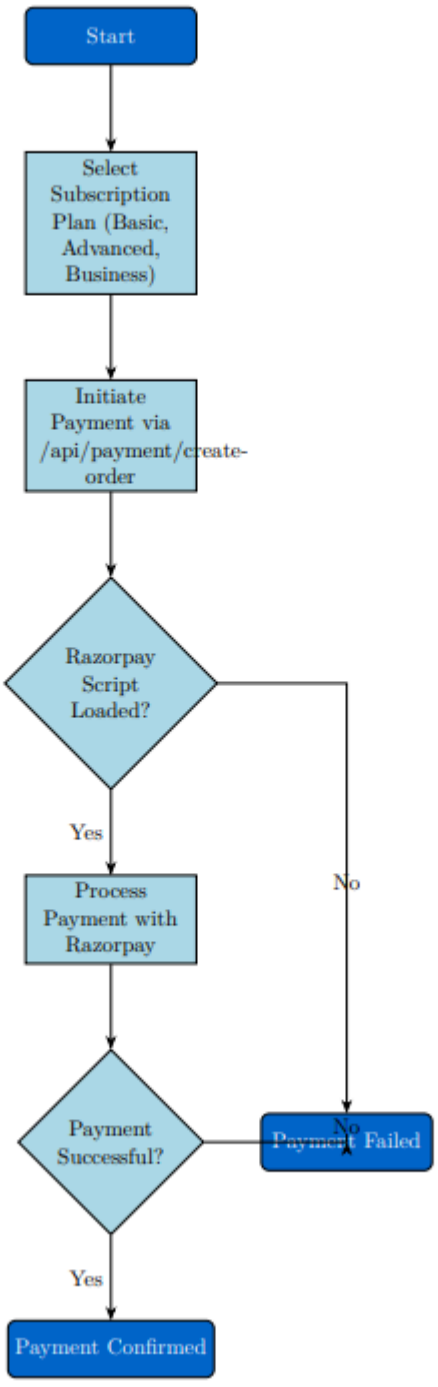


Figure 1: User Authentication Flow



3.2 Appointment and Consultancy Flow

The appointment and consultancy flow allows patients to select a doctor, book an appointment, and engage in consultancy via chat or video conferencing, with confirmations and details accessible in the dashboard.

4 Technical Architecture

4.1 Tech Stack

4.1.1 MERN Stack (Authentication)

- **MongoDB**: NoSQL database for storing user credentials and OTPs.
- **Express.js**: Node.js framework for authentication API routes.
- **React**: Frontend for login and verification UI.
- **Node.js**: Runtime for authentication backend.

4.1.2 Spring Boot (Dashboard, Chatting, Video Conferencing)

- Java-based framework for dashboard, appointment booking, real-time chatting, and video conferencing.
- Integrates with MongoDB for data persistence.
- Uses WebSocket (e.g., STOMP over SockJS) for chatting and WebRTC or third-party services (e.g., Twilio) for video conferencing.
- Exposes RESTful APIs for the React frontend.

4.2 Frontend

- **Framework**: React with Vite for fast builds.
- **Styling**: Tailwind CSS for responsive, white-and-blue-themed UI.
- **Animations**: Framer Motion for card and button transitions.
- **Icons**: Font Awesome via react-icons and Lucide React.
- **Components**:
  - Login.jsx: Handles sign-up, login, and role selection (MERN stack).
  - DoctorLogin.jsx: Manages Doctor login with OTP verification (MERN stack).
  - WhatWeOffer.jsx: Showcases services with animated cards.
  - BuyAPlan.jsx: Displays subscription plans with Razorpay integration.
  - Navbar.jsx: Navigation bar for site-wide access.
  - Dashboard.jsx: Displays user data, chat, and video options (Spring Boot).
  - Chat.jsx: Real-time chat interface (Spring Boot).
  - VideoConference.jsx: Video call interface (Spring Boot).
- **Context**: AppContext.js manages global state (e.g., login status, user data).

–

4.3 Backend

4.3.1 MERN Stack (Authentication)

- **Framework**: Node.js with Express.js.
- **Database**: MongoDB for user data and OTPs.
- **Endpoints**:
  - POST /api/auth/register: Register a new user.
  - POST /api/auth/login: Authenticate a user.
  - POST /api/auth/verify-otp: Verify Doctor OTP.
  - GET /api/user/me: Fetch user data.
- **Authentication**: Session-based with cookies (withCredentials: true).

4.3.2 Spring Boot (Dashboard, Chatting, Video Conferencing)

- **Framework**: Spring Boot with Java.
- **Database**: MongoDB for dashboard data, appointments, chat history, and video session logs.
- **Endpoints**:
  - GET /api/dashboard/status: Fetch health status and appointment history.
  - POST /api/appointments/book: Book an appointment.
  - GET /api/consultancy/options: Fetch consultancy options.
  - POST /api/chat/messages: Send/receive chat messages (WebSocket).
  - POST /api/video/start: Initiate video call.
  - POST /api/payment/create-order: Initiate Razorpay payment (shared).
- **Authentication**: JWT or session-based, integrated with MERN stack.
- **Real-time Features**: WebSocket for chatting, WebRTC or third-party service for video.

4.4 Third-Party Services

- **Razorpay**: Handles payment processing for subscription plans.
- **Email Service**: Sends OTPs for Doctor verification (e.g., SendGrid for MERN, Spring Mail for Spring Boot).
- **Video Conferencing**: WebRTC or third-party service (e.g., Twilio, Zoom SDK) integrated via Spring Boot.

5 Setup Instructions

5.1 Prerequisites

- Node.js (v20.12.0 or later)
- Java (JDK 17 or later) for Spring Boot
- MongoDB (local or cloud-based, e.g., MongoDB Atlas)
- Razorpay account with API keys
- Email service credentials (e.g., SendGrid API key)
- WebSocket support (included in Spring Boot)
- WebRTC or third-party video conferencing service (e.g., Twilio API keys)

5.2 MERN Stack Setup (Authentication)

1. Clone the repository: `git clone <repository-url>`.
2. Navigate to the authentication backend directory: `cd server/auth`.
3. Install dependencies: `npm install`.
4. Create a `.env` file with:

```
PORT=4000
MONGODB_URI=<your-mongodb-uri>
RAZORPAY_KEY_ID=<your-razorpay-key-id>
RAZORPAY_KEY_SECRET=<your-razorpay-key-secret>
EMAIL_SERVICE_API_KEY=<your-email-service-key>
```

5. Start the server: `npm run dev` (uses Nodemon).

5.3 Spring Boot Setup (Dashboard, Chatting, Video Conferencing)

1. Navigate to the dashboard backend directory: `cd server/dashboard`.
2. Ensure Maven is installed.
3. Configure `application.properties` in `src/main/resources`:

```
spring.data.mongodb.uri=<your-mongodb-uri>
server.port=8080
razorpay.key.id=<your-razorpay-key-id>
razorpay.key.secret=<your-razorpay-key-secret>
spring.mail.host=smtp.<your-email-service>.com
spring.mail.username=<your-email-username>
spring.mail.password=<your-email-password>
video.service.api.key=<your-video-service-key>
```

4. Build and run: `mvn spring-boot:run`.

5.4 Frontend Setup

1. Navigate to the frontend directory: `cd client`.
2. Install dependencies: `npm install`.
3. Create a `.env` file with:

```
VITE_RAZORPAY_KEY_ID=<your-razorpay-key-id>
VITE_AUTH_BACKEND_URL=http://localhost:4000
VITE_DASHBOARD_BACKEND_URL=http://localhost:8080
```

4. Start the development server: `npm run dev`.

6 Testing

- Access the frontend at http://localhost:5173.
- Test login (MERN stack), OTP verification (MERN stack), dashboard access, appointment booking, chatting, and video conferencing (Spring Boot).
- Monitor backend logs for errors, especially `ERR_HTTP_HEADERS_SENT` in MERN stack.

7 Troubleshooting

7.1 ERR_HTTP_HEADERS_SENT (MERN Stack)

Issue: Backend sends multiple responses.
Solution: Ensure endpoints send one response. Use `return` with `res.json`. Check `userController.js` (e.g., `getUserData`).

7.2 Razorpay Issues

Issue: Razorpay script fails to load.
Solution: Verify connectivity and `VITE_RAZORPAY_KEY_ID`.

7.3 Spring Boot Dashboard Errors

Issue: Dashboard, chat, or video not loading.
Solution: Verify MongoDB connection, WebSocket configuration, and video service API keys in `application.properties`.

8 Future Enhancements

- Implement JWT-based authentication across MERN and Spring Boot.
- Add real-time notifications for appointments and messages.
- Enhance video conferencing with screen sharing and recording.
- Improve AI suggestions with machine learning models.

9 Contact

For support, contact the development team at vijitverma1023@gmail.com

10 Screen Shots