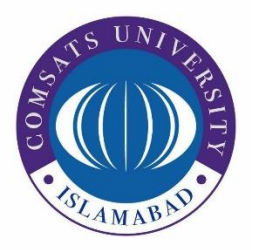


COMSATS University Islamabad (CUI)

Software Design Description
(SDS DOCUMENT)

for

CardioLink

Version 1.0

By

Muhammad Bilal **CIIT/SP22-BCS-052/ISB**
Faizan Maqbool **CIIT/SP22-BCS-022/ISB**

Supervisor

Mr. Tanveer Ahmed

Bachelor of Science in Computer Science (2022-2026)

Table of Contents

Table of Figures

Revision History

Name	Date	Reason for changes	Version

Application Evaluation History

Comments (by committee) ***include the ones given at scope time both in doc and presentation**	**Action Taken**

Supervised by

Mr. Tanveer Ahmed

Signature_____

1. Introduction

Managing cardiovascular health in Pakistan remains a critical challenge, especially in remote or underserved areas. CardioLink revolutionizes this space by offering an all-in-one digital platform for heart disease management. Unlike traditional systems, CardioLink integrates AI-powered diagnosis from ECG/ECHO images, instant video consultations with cardiac specialists, and real-time emergency support. Patients can securely store and share medical records, receive AI-based lifestyle recommendations, and order prescribed medicines through the built-in pharmacy. What sets CardioLink apart is its centralized, patient-centric interface that connects diagnosis, treatment, and emergency services seamlessly. With encrypted data handling and 24/7 chatbot support, CardioLink ensures timely, confidential, and accessible cardiac care—empowering both doctors and patients to take informed, efficient action toward better heart health.

1.1 Modules

1.1.1 Module 1: Electronic Health Records

This module integrates patient profile management and a repository of comprehensive medical records.

FE-1: *Implement a secure user registration system that ensures smooth onboarding.*
FE-2*: Design a robust login feature ensuring safe and fast access for authorized users while preventing unauthorized intrusions.*
FE-3: *Develop an advanced file upload system to store diverse patient history formats with metadata tagging for efficient retrieval.*
FE-4*: Implement a streamlined history viewing interface for healthcare providers and patients, offering quick access to past medical records and reports for informed decision-making*

1.1.2 Module 2: AI ECG Interpreter

This module specializes in leveraging AI-driven ECG analysis to diagnose arrhythmias and myocardial infarction.

FE-1: *Create a secure interface for uploading ECG files.*
FE-2: *Implement advanced AI algorithms to identify arrhythmias, abnormal hearbeat, as well as myocardial infarction markers.*
FE-3*: Generate detailed diagnostic summaries with visual annotations for detected abnormalities.*
FE-4: *Provide healthcare professionals with tools to validate AI diagnoses, attach recommendations, and initiate consultations if needed.*

1.1.3 Module 3: AI EchoVision

This module focuses on analyzing ECHO videos to diagnose structural and functional heart diseases.

FE-1: *Implement a secure and user-friendly upload system for ECHO video files, enabling fast and efficient processing for cardiovascular health assessments.*

FE-2: *Generate individualized diagnostic reports focused on structural and functional heart parameters, identifying key markers such as Ejection Fraction, End Systolic Volume and End Diastolic Volume for assessing heart function.*
FE-3: *Provide actionable insights on ECHO findings and strategies to mitigate risks of progression.*
FE-4: *Facilitate expert review features for cardiologists to validate AI-generated findings, add professional observations, and provide patient-specific recommendations.*

1.1.4 Module 4: AI Heartbeat Classifier

This module focuses on utilizing AI to process and classify heartbeat audio recordings, aiding in the detection of cardiac irregularities.

FE-1: *Develop a user-friendly interface for uploading heartbeat audio files.*

FE-2: *Integrate advanced AI models to analyze acoustic features and categorize the recordings as normal or abnormal, detecting if a murmur is present or not.*
FE-3: *Generate intuitive diagnostic outputs, determine possible heart issues on the basis of whether a murmur was detected or not and at which region if detected.*

1.1.5 Module 5: Teleconsultation

Patients can get medical consultation from specialists in cardiology in their homes through video consultations.

FE-1: *Schedule appointments with a cardiac specialist.*
FE-2: *Initiate video consultation calls from within the app to get immediate medical opinion and consultation.*
FE-3: *After consultation digital prescriptions and diagnostic reports are sent to patient which can be stored in the app.*
FE-4: *Doctors can access a comprehensive patient history, including prior diagnoses and treatment records, and lifestyle insights before each consultation.*
FE-5: *Every new prescription and/or diagnosis will update the patient medical history as predefined.*
FE-6: *Review and rate doctors to assist future patient decisions*

1.1.6 Module 6: PharmaLink

Enables patients to effortlessly place an order for an online drug prescription and pharmacy service incorporating the prescribing physician’s details.

FE-1: *Allows Medication Automation such that after each consultation, upon a doctor's prescription, the system automatically checks medication availability and, if in stock, dispatches it directly to the patient's address without requiring any further action by the user.*
FE-2: *Patients can also browse and order medication and lifestyle products related to cardiovascular health directly on our app.*
FE-3: *Provide a dedicated pharmacy reorder tab that includes price comparisons and availability, allowing patients to reorder medications easily.*

1.1.7 Module 7: HeartWise AI

This module features a virtual assistant that offers continuous support, education, and personalized advice focused on cardiovascular health.

FE-1: *Provide information on common cardiac conditions, risk factors, and the importance of professional diagnosis over self-diagnosis.*
FE-2: *Offer personalized advice on preventing cardiovascular risks based on individual health profiles.*
FE-3: *Provide practical guidance for living with heart disease, including lifestyle and emotional support.*
FE-4: *Share the latest trends, research, and practices in cardiovascular care.*
FE-5: *Engage users with interactive tools to assess and improve their understanding of heart health.*

1.1.8 Module 8: CardioHealth Tracker

This module helps patients actively track their daily physical activity, dietary intake, and key health metrics to monitor cardiovascular health.

FE-1: *The user would be able to track the number of steps taken or total exercises done like walking, jogging, cycling, etc. as physical activity metric.*
FE-2: *The user will keep a record of food intakes (breakfast, lunch, supper, and snacks), from which the application provides a summary of saturated fats, cholesterol, and calories ingested concerning the users' heart health.*
FE-3: *The patients measure and log their blood pressure and heart rates along with the medication intake and monitor the vital signs for changes over periods.*
FE-4: *Patients can schedule alerts for taking their medications so that they follow their treatment as prescribed.*
FE-5: *At the end of each month, the application will prepare report which includes summary of activities performed, meals taken, health metrics recorded, and medications compliance.*

1.1.9 Module 9: AI-Powered Heart Disease Prediction

This module leverages machine learning algorithms to predict the likelihood of heart disease, assisting in early detection and fostering informed decision-making. The predictions are based on user-provided health data and validated with scientific rigor.

FE-1: *Collect user health, including metrics such as age, gender, cholesterol levels, blood pressure, heart rate, BMI, smoking status, and exercise habits.*
FE-2: *Use machine learning model to analyze the user's data and predict the probability of heart disease, presenting results in a clear and user-friendly format.*
FE-3: *Offer actionable insights and recommendations based on prediction results, emphasizing the importance of consulting healthcare professionals.*
FE-4: *Provide a library of resources explaining the predictive features and how each contributes to heart disease risk, fostering transparency and user trust.*

1.1.10 Module 10: Ambulance-Connect

Enables patients to summon an ambulance by simply clicking a button, which in turn will inform the closest available ambulance for an effective response.

FE-1: *Allow patients to request an ambulance with a single click, automatically signaling the nearest available ambulance for quick response.*
FE-2: *Enable patients to track the ambulance arrival time and route to their location.*
FE-3: *Allow ambulance personnel to receive patient requests and navigate to the patient's location efficiently.*
FE-4: *Provide communication options between ambulance personnel and the patient for status updates.*

1.1.11 Module 11: Physician Gateway

This module aims at introducing the doctor's system where they will have the ability to register to our app, login, manage their schedules.

FE-1: *The doctors can Register, by forwarding their requests containing their Information and Documents to Administration.*
FE-2: *Login to their account using correct credentials.*
FE-3: *Edit Personal Information, adding educational credentials and Password change.*
FE-4: *Specify and record the capacity for individual consultations up to 30 days in advance.*
FE-5: *A personalized dashboard which has a calendar where one can view appointments, edit them and confirm them while adjusting one's availability whenever it is necessary.*

1.1.12 Module 12: Admin Control Panel

Gives admins control over user registrations, particularly for doctors and ambulance providers, as well as system notifications.

FE-1: *Review and approve registration requests from doctors and ambulance services.*

FE-2: *Manage system-wide notifications, data audits, and error reporting to ensure seamless app functionality.*

2. Design Methodology and Software Process Model

We are following the Object-Oriented Programming (OOP) design methodology for the development of CardioLink. OOP is ideal for this project as it allows us to represent real-world healthcare entities—such as patients, doctors, and medical records—as individual objects. This approach improves code modularity, reusability, and maintainability, which are crucial for a complex and evolving healthcare system. By encapsulating both data and behavior within objects, OOP makes the system easier to manage, extend, and debug as new features or updates are introduced.

For the development process, we have selected the Incremental Model. This model divides the project into smaller, manageable parts that are developed and delivered in increments. It enables early delivery of key functionalities, which is beneficial for gathering timely feedback from stakeholders and adapting to any changes in healthcare regulations or user requirements. The incremental approach also helps in minimizing risks and ensuring that the system evolves in alignment with actual user needs, making it a practical and flexible choice for a healthcare application like CardioLink.

3. System Overview

3.1 Architectural Design

CardioLink is built on a modular, MVC-based architecture that ensures scalability, flexibility, and maintainability. The platform is organized into key modules: User Management, Diagnostics & AI Analysis, Telemedicine, Pharmacy, Emergency Dispatch, and Health Tracking & Education.

The system follows a multi-tiered architecture:

- The Model Layer (MongoDB with Mongoose) defines schemas for users, medical data, visits, diagnostics, and more.
- The Controller Layer (Express.js) handles business logic, routes, and integration with AI services.
- The View Layer includes mobile UIs (Flutter) and web interfaces (React), delivering tailored experiences to patients, doctors, and admins.

An integrated AI Service Layer supports ECG, echo, and heartbeat analysis via deep learning models. A secure Cloud Data Layer manages storage of records, imaging, and results.

This clean separation of concerns ensures CardioLink remains robust, extensible, and focused on delivering intelligent, end-to-end cardiovascular care

3.2 Box and Line Diagram

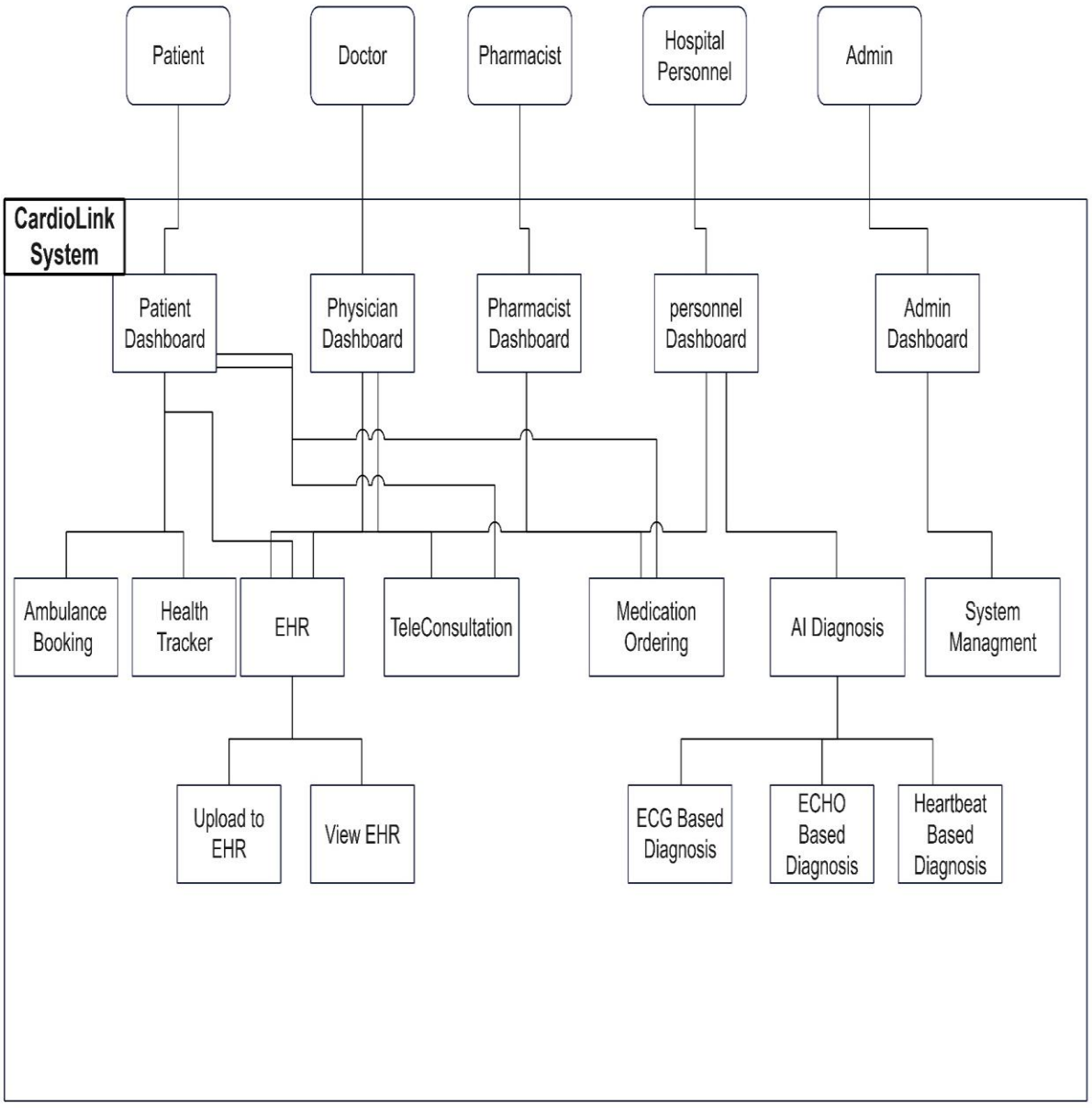


Figure 1: Box and Line Diagram of System

3.3 Architectural Diagram

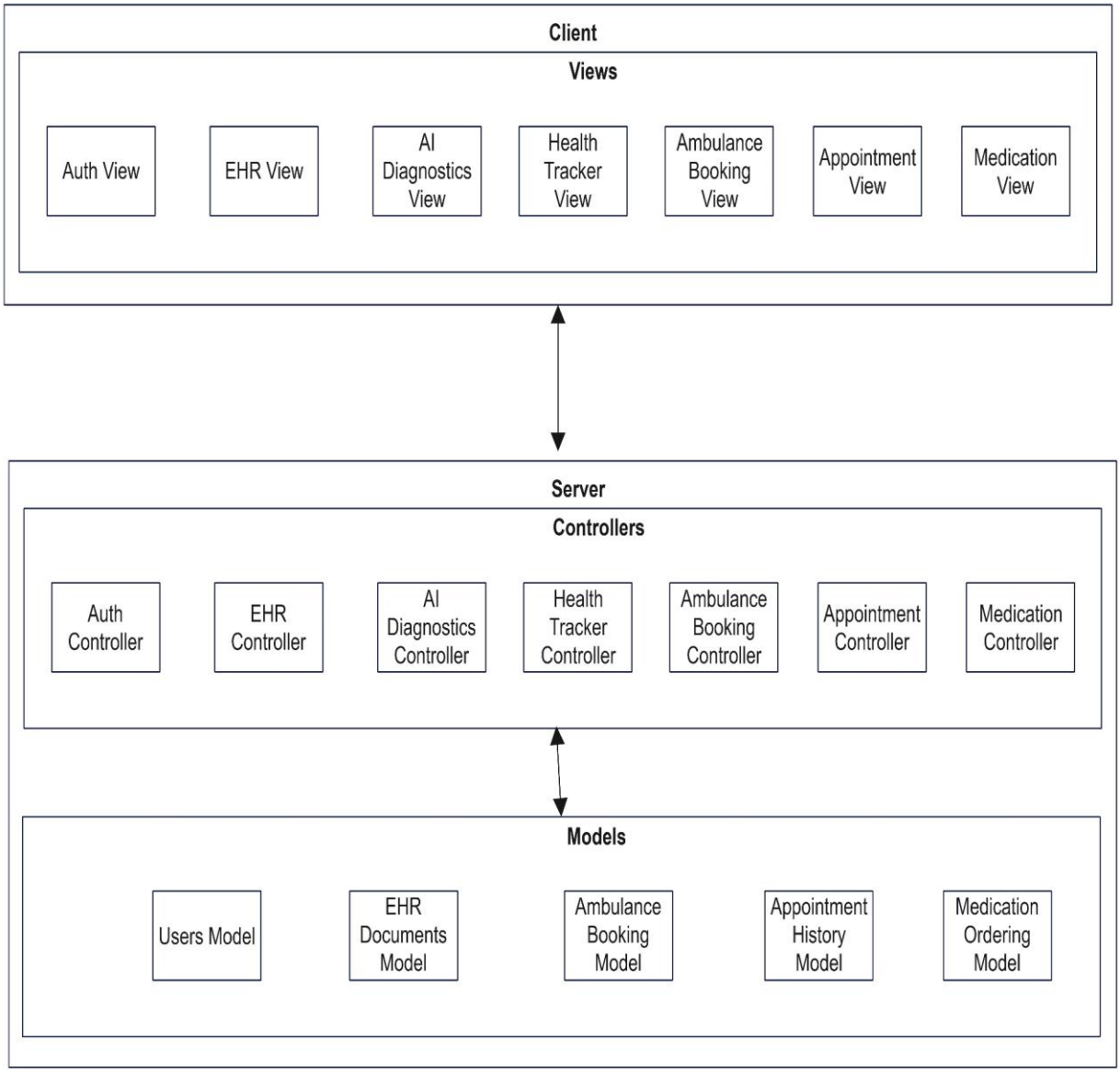


Figure 2: Architecture Diagram of System

4. Design Models

This section provides information about the CardioLink design models.

4.1. Activity Diagrams:

This section shows the activity diagrams of major use cases of each module of CardioLink.

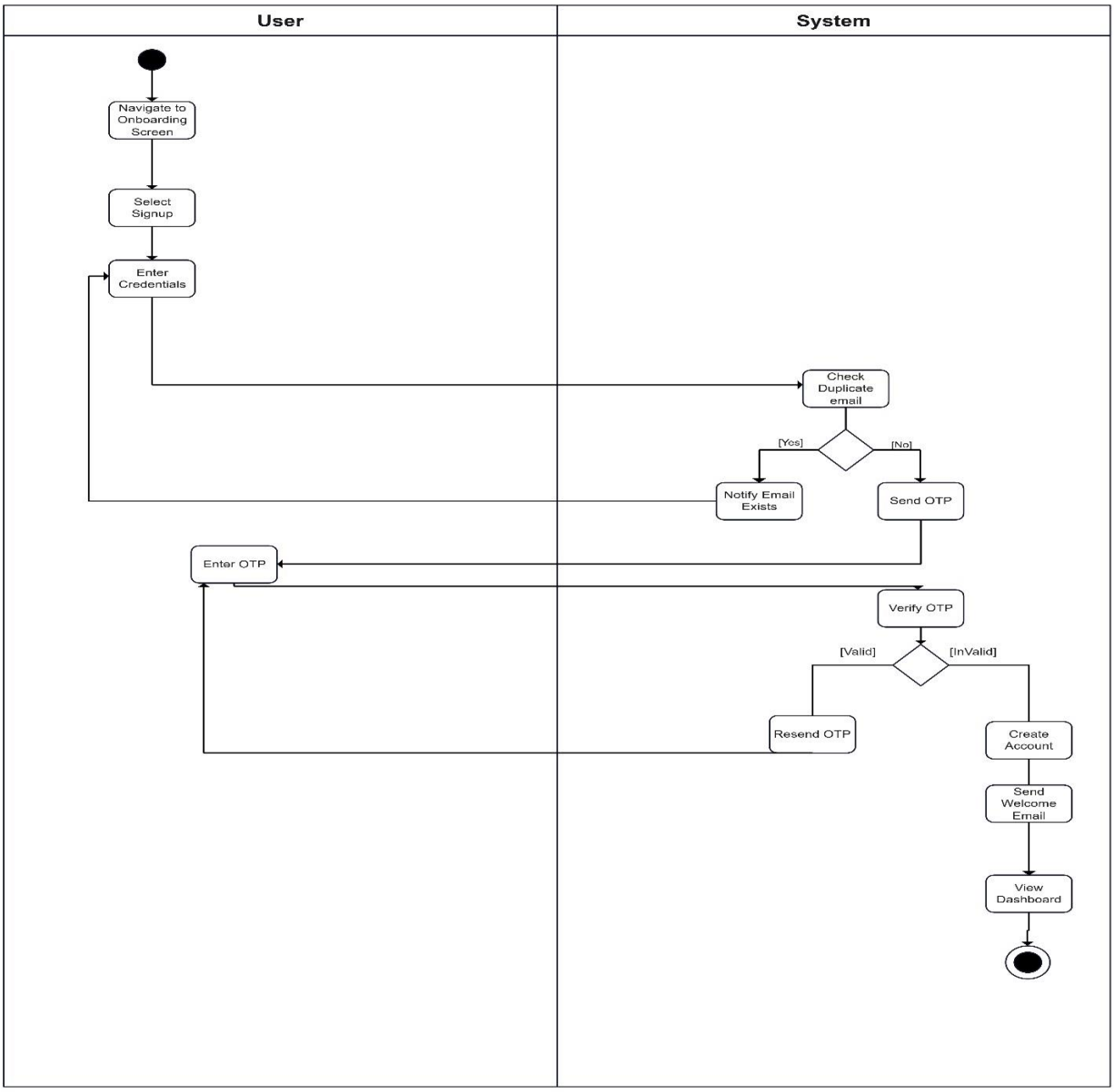


Figure 3: Activity Diagram for User Registration

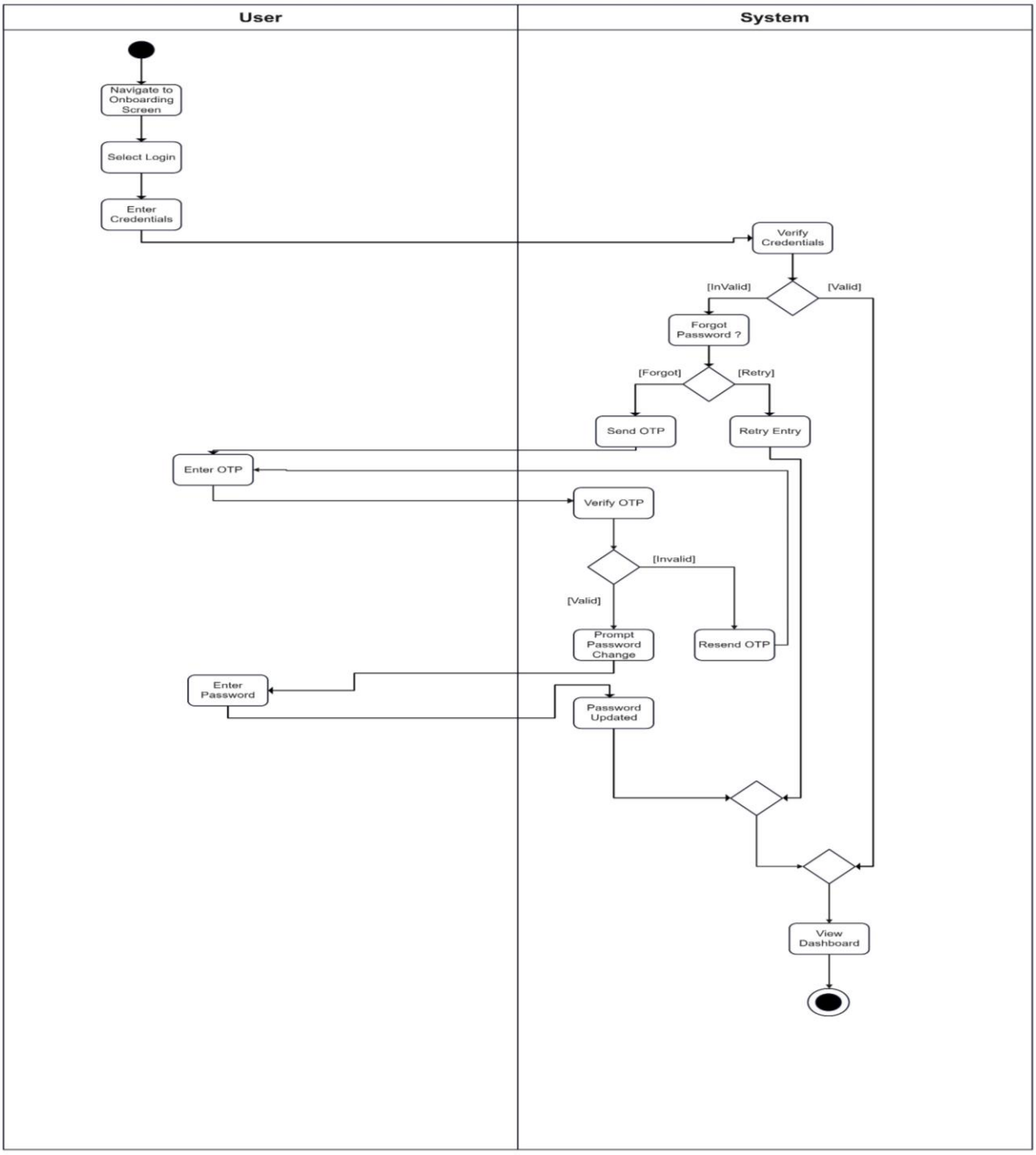


Figure 4: Activity Diagram for User Login

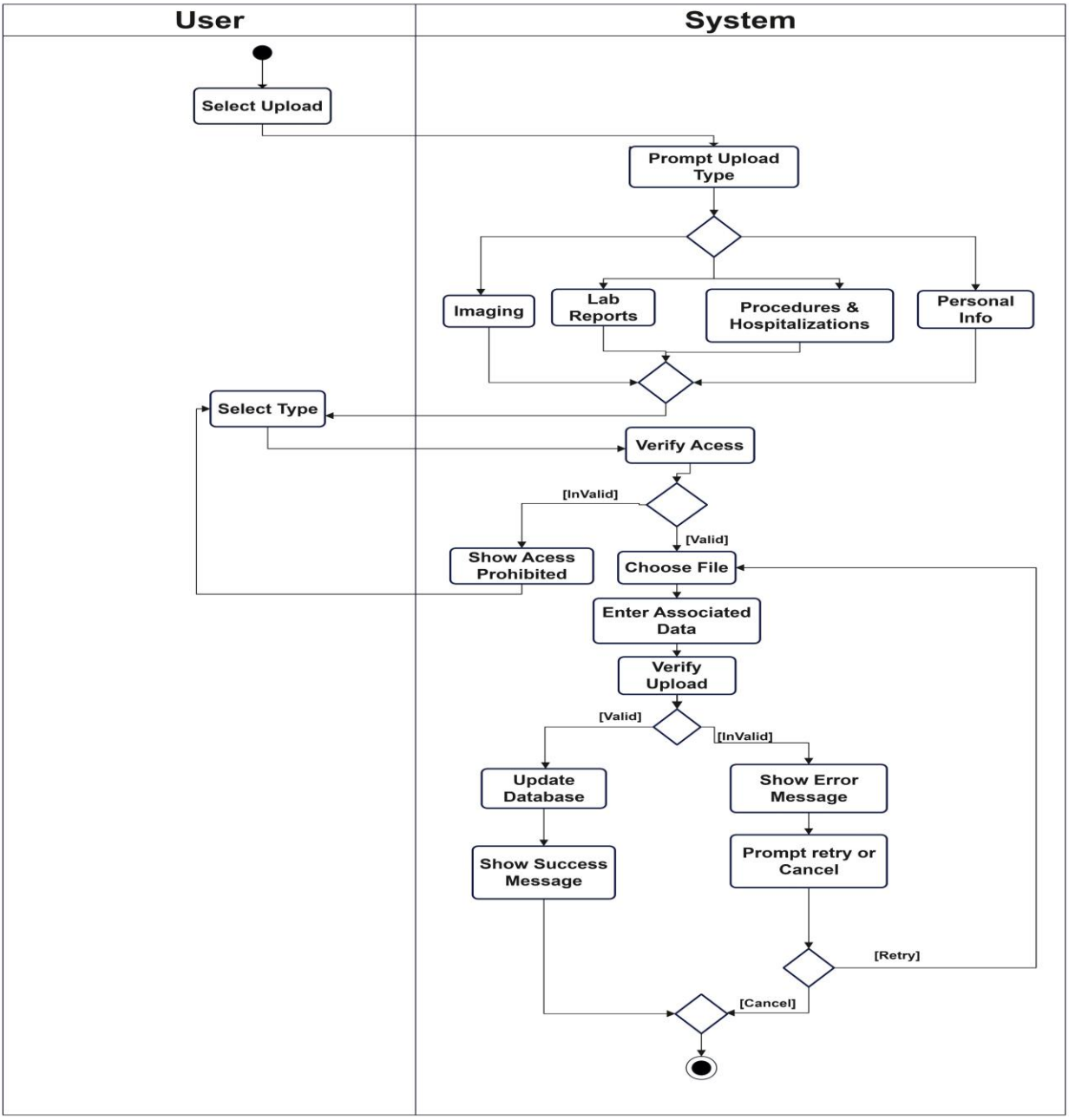


Figure 5: Activity Diagram for Uploading to EHR

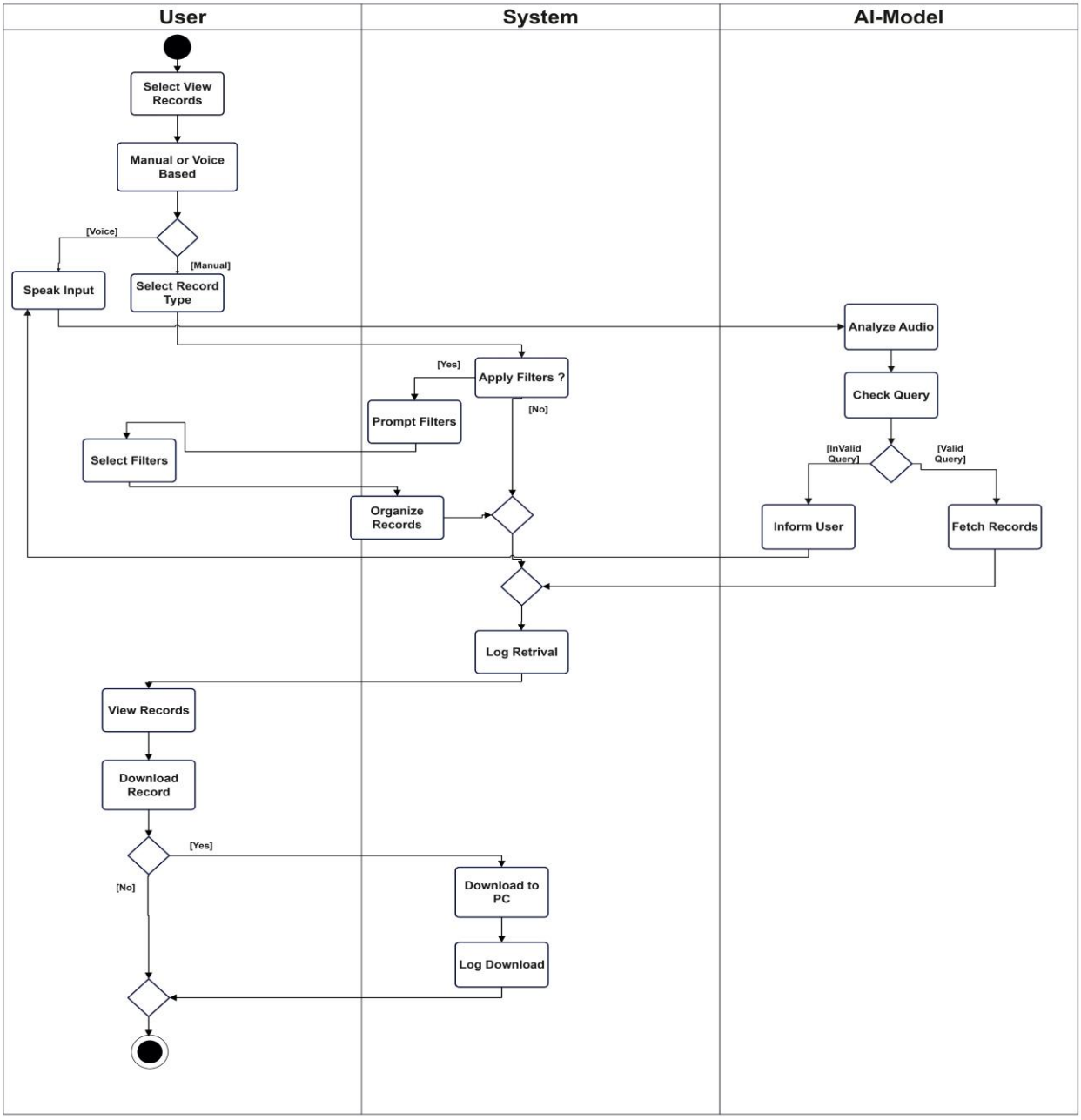


Figure 6: Activity Diagram for Viewing EHR

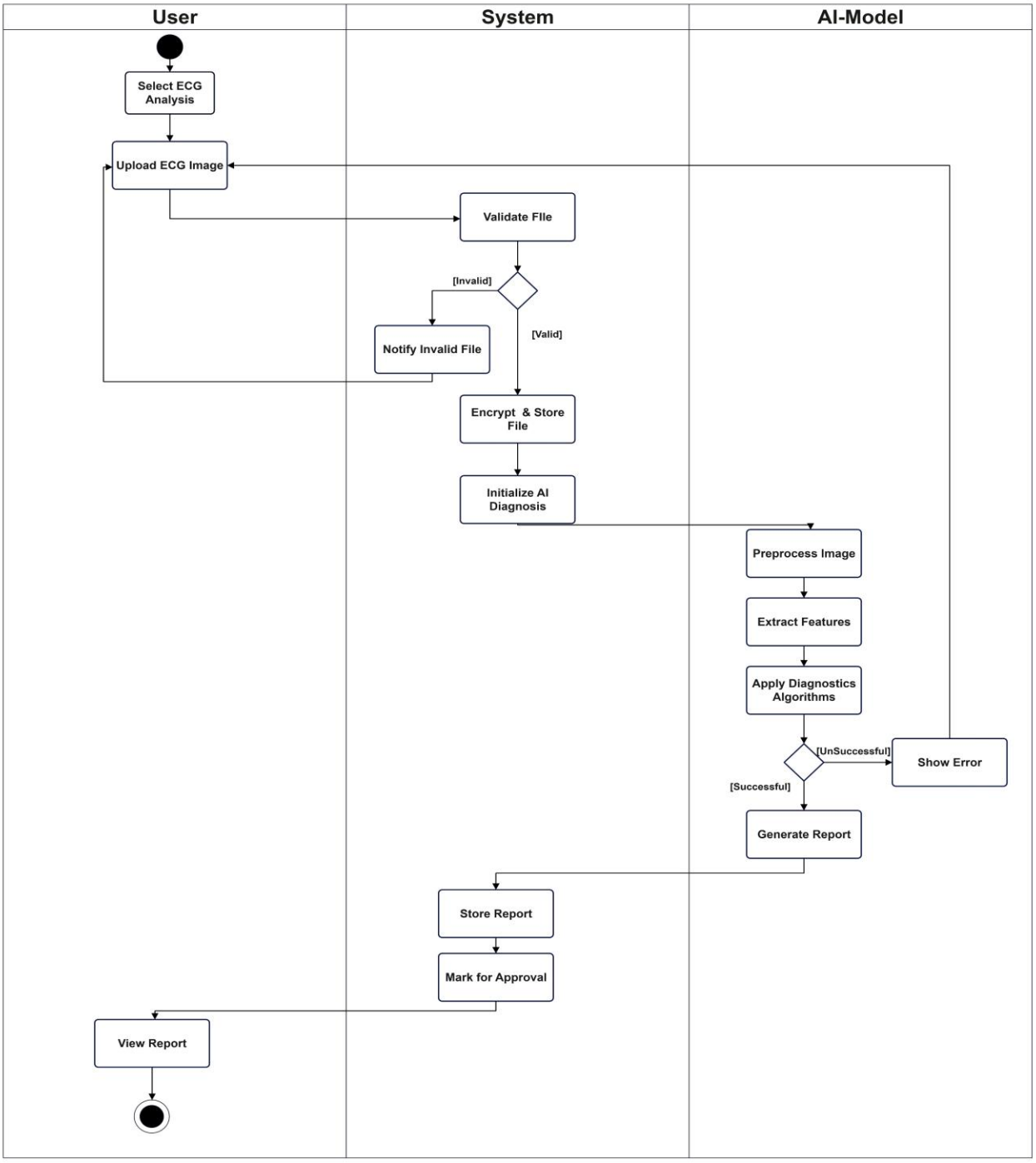


Figure 7: Activity Diagram for ECG based Diagnosis

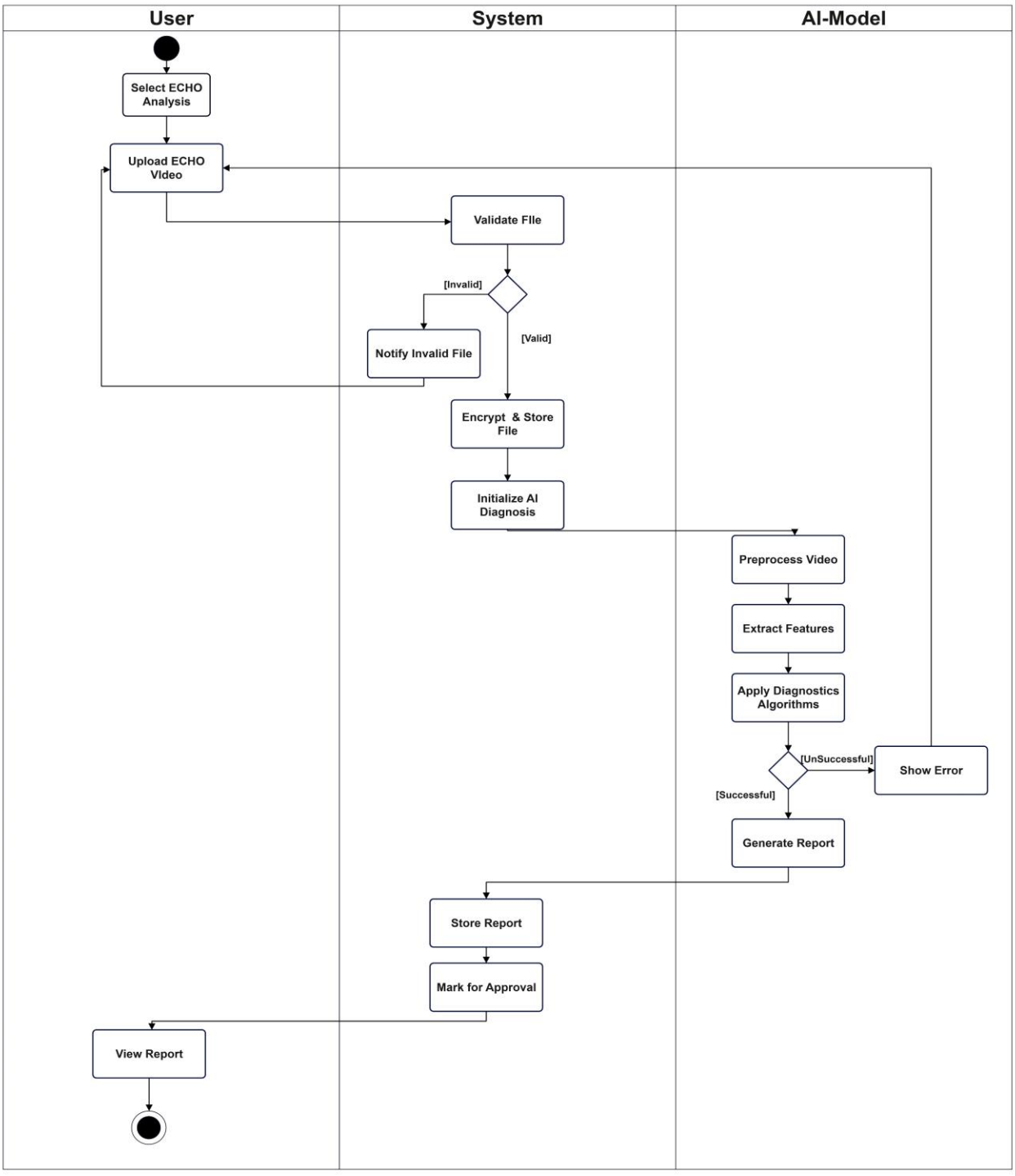


Figure 8: Activity Diagram for ECHO based Diagnosis

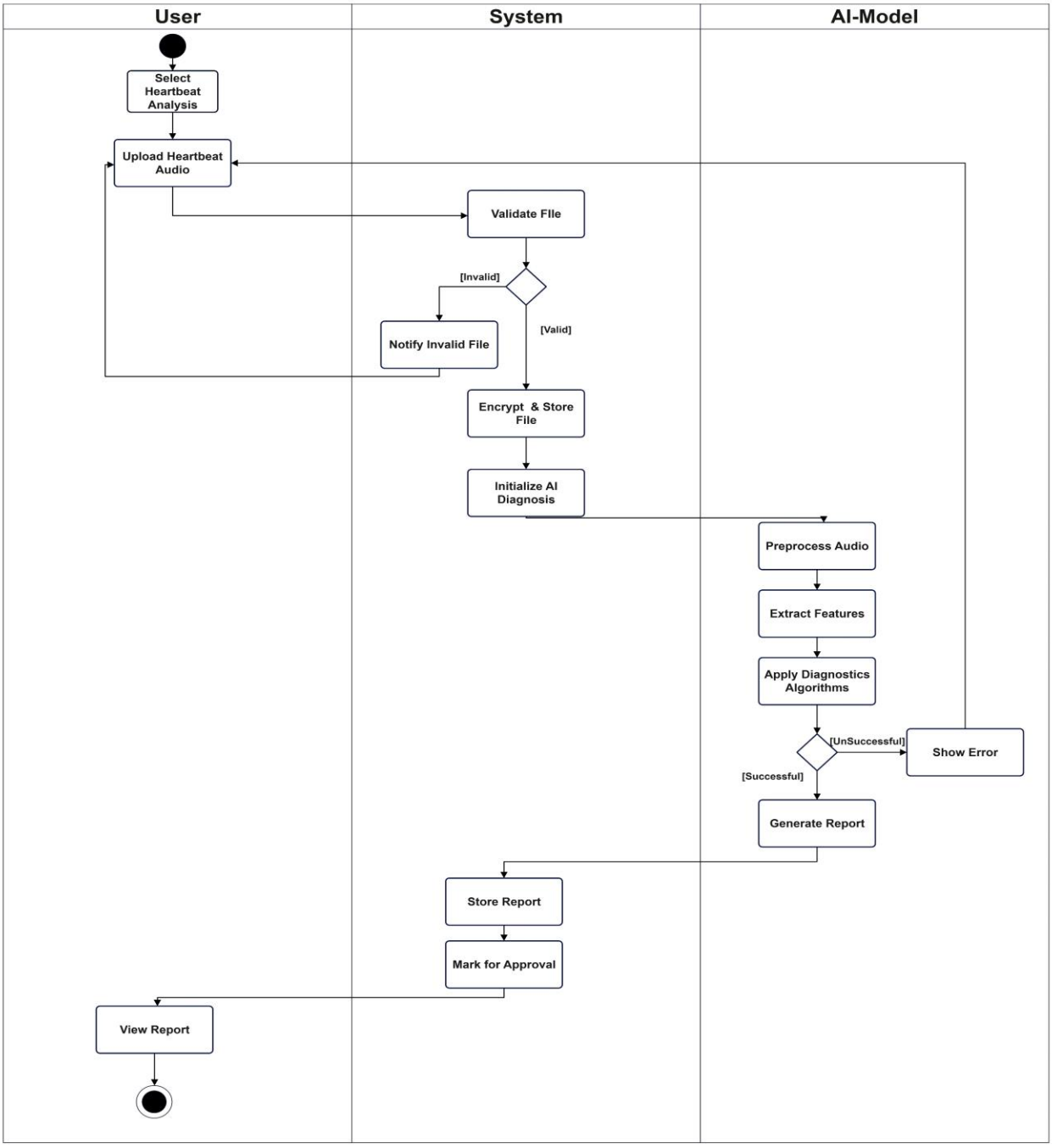


Figure 9: Activity Diagram for Audio Based Murmur Diagnosis

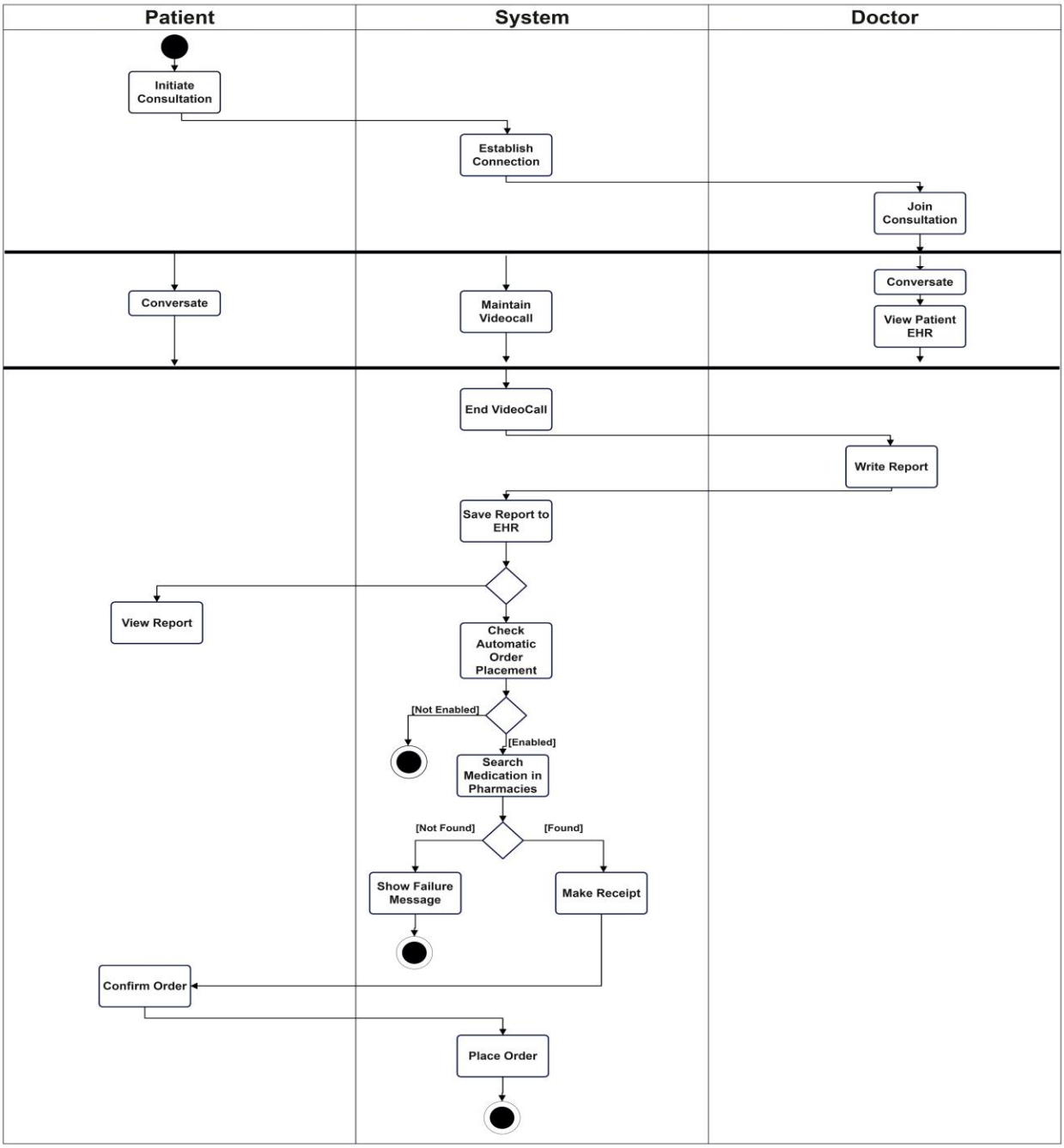


Figure 10: Activity Diagram for Video Consultation

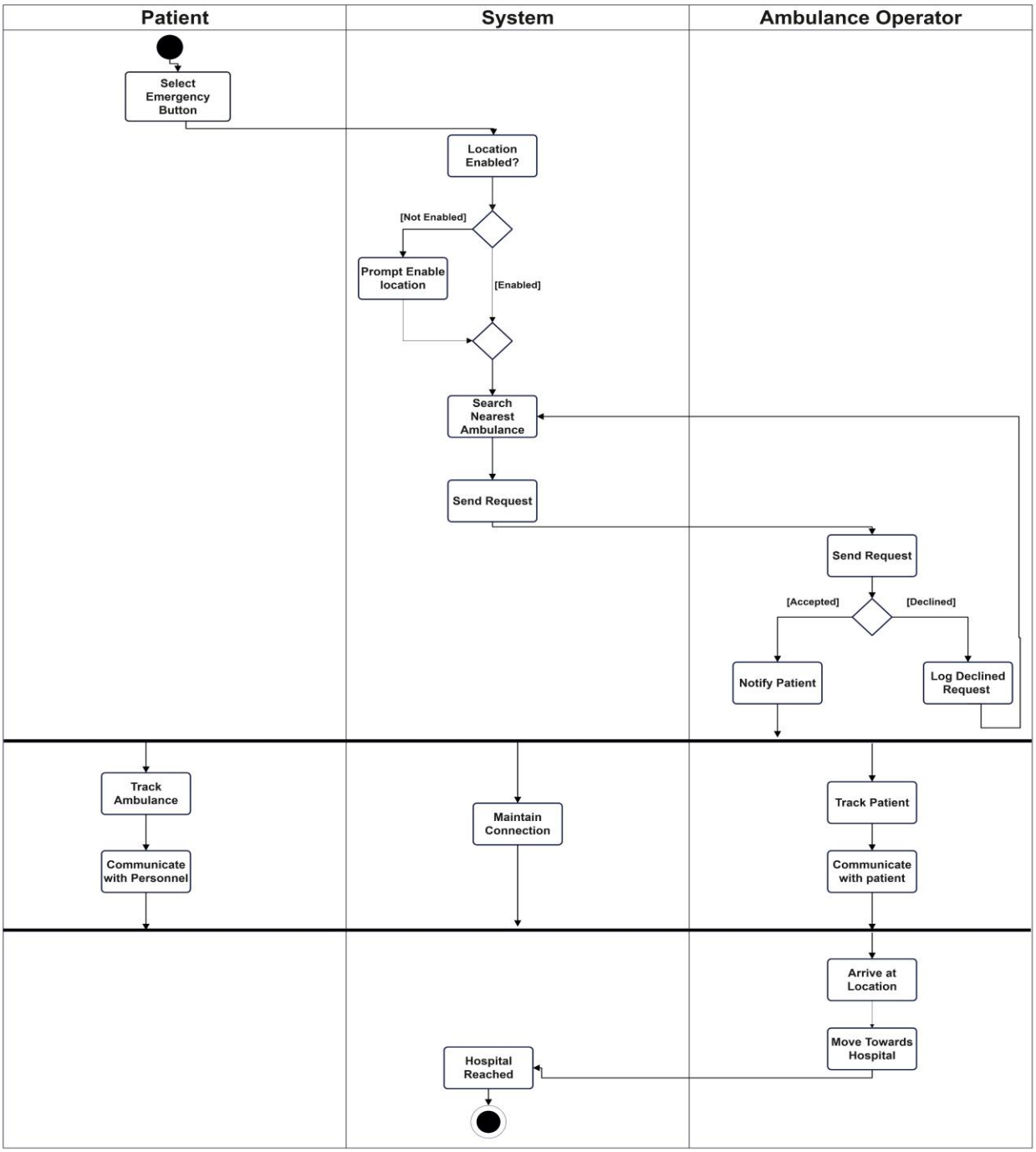


Figure 11: Activity Diagram for Emergency Ambulance Booking

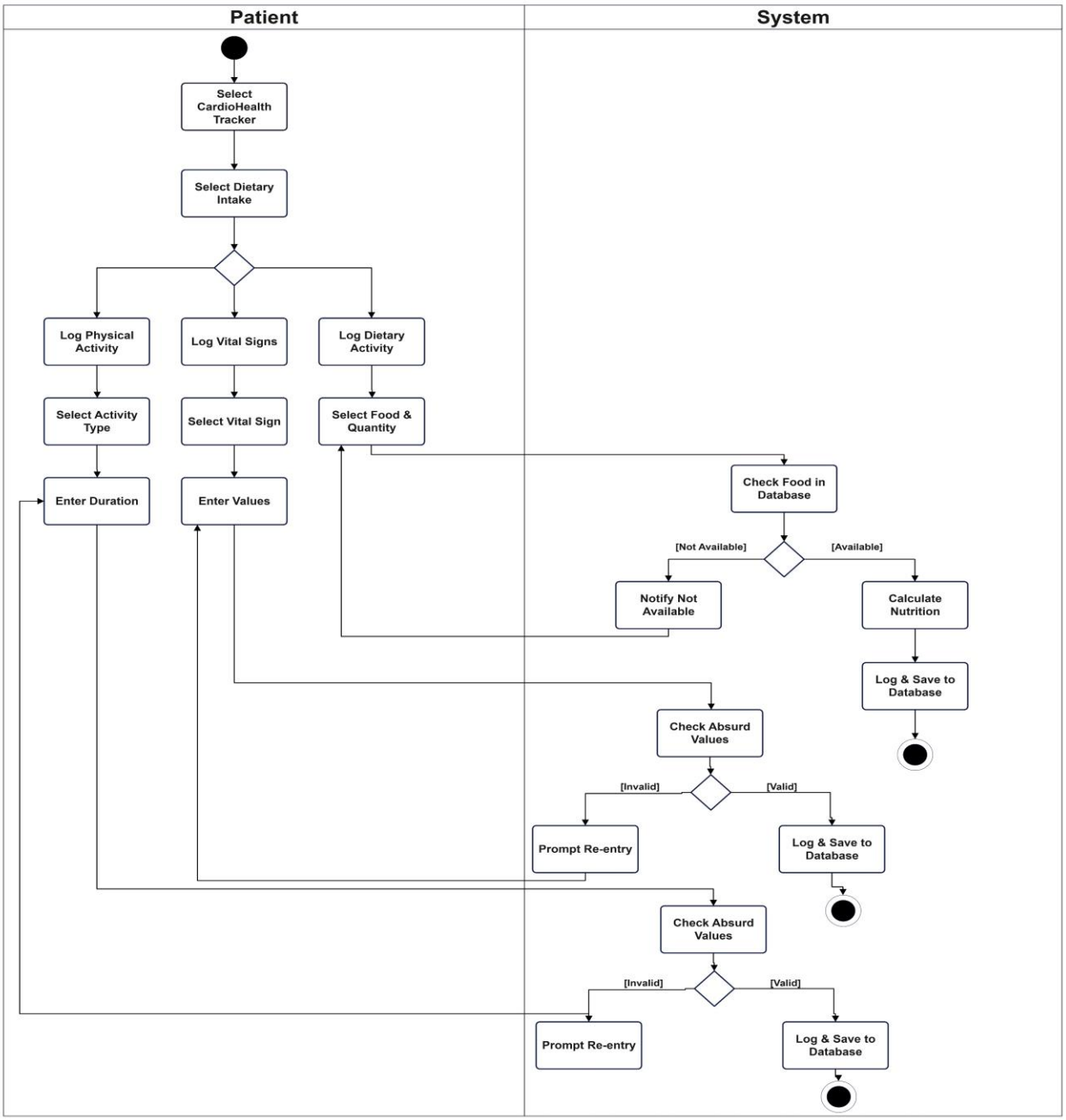


Figure 12: Activity Diagram for Health Tracker

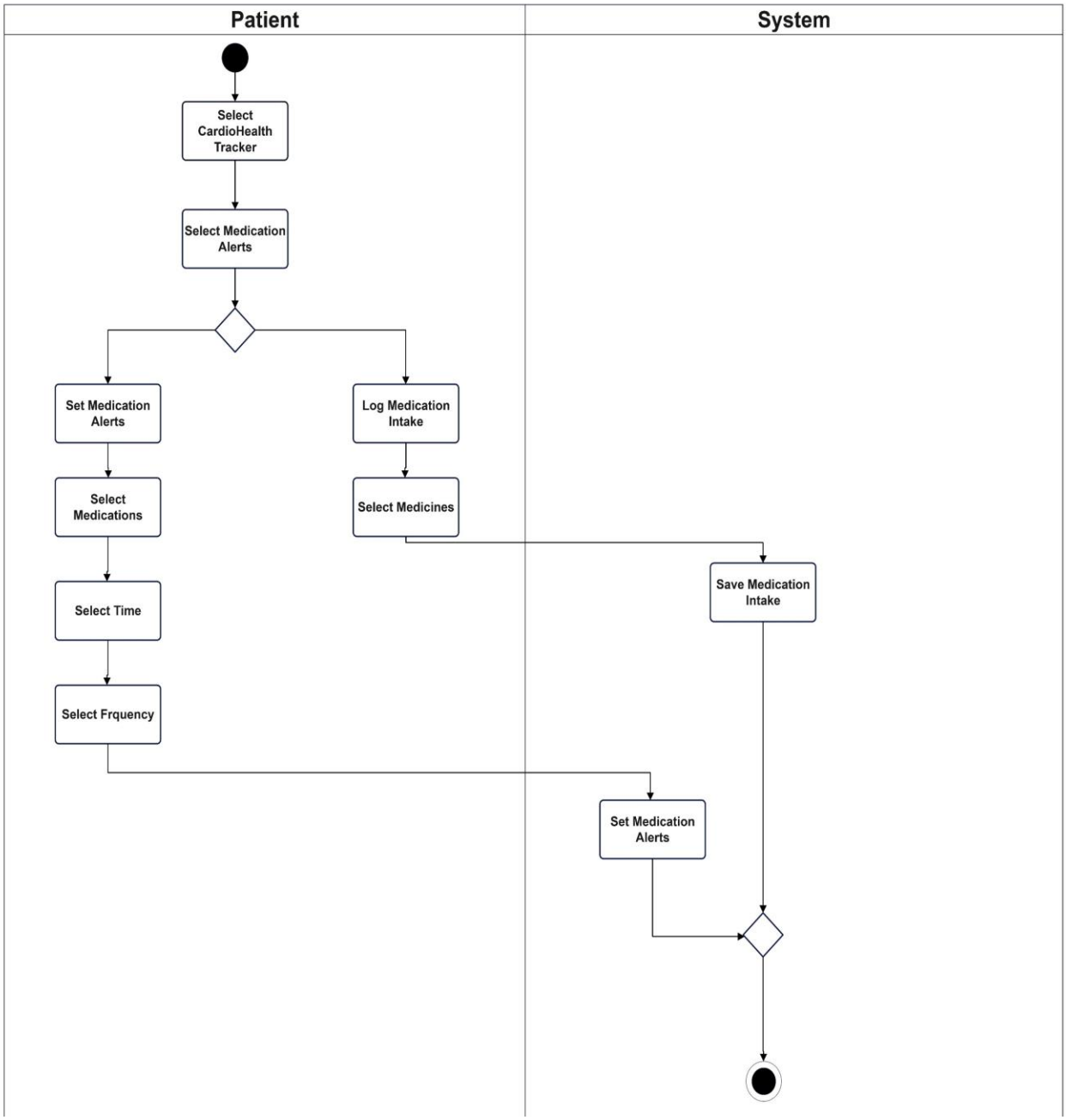


Figure 13: Activity Diagram for Medication Alerts

4.2 Class Diagram

The following figure shows the CardioLink class diagram.

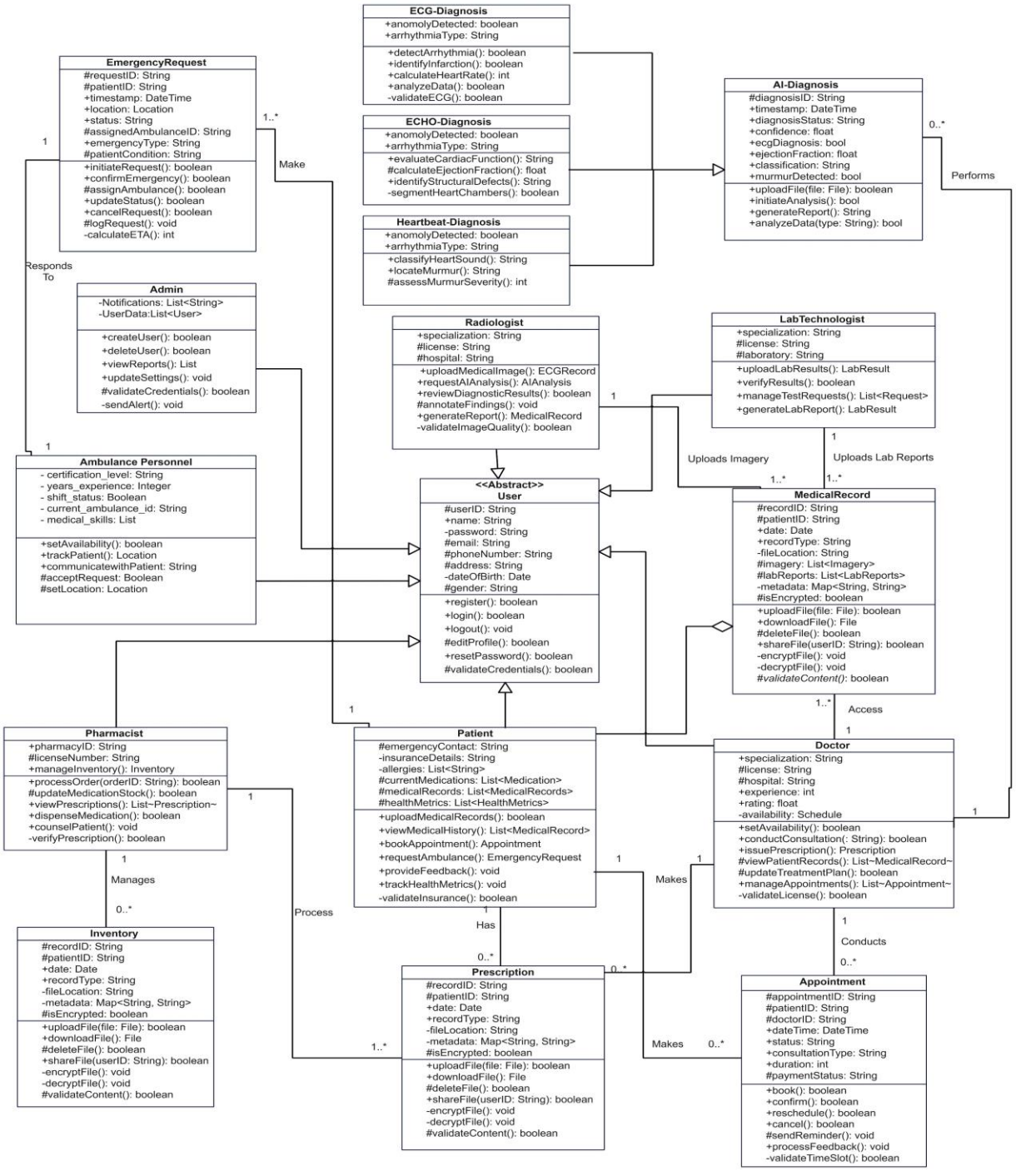


Figure 14: Class Diagram of the System

4.3. Sequence Diagrams:

This section shows the sequence diagrams of major use cases of main modules of CardioLink

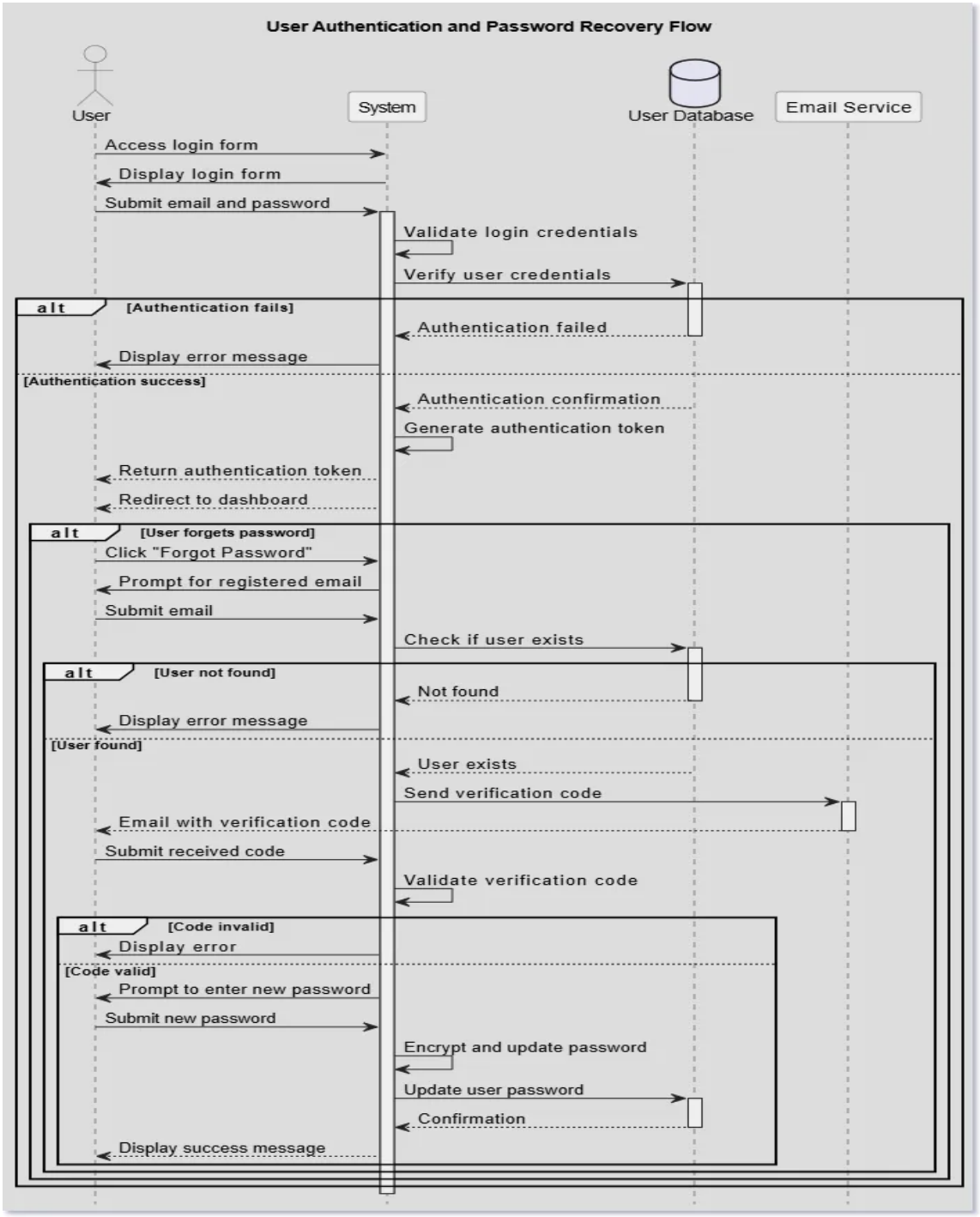


Figure 15: Sequence Diagram for User Authentication and Password Recovery

.

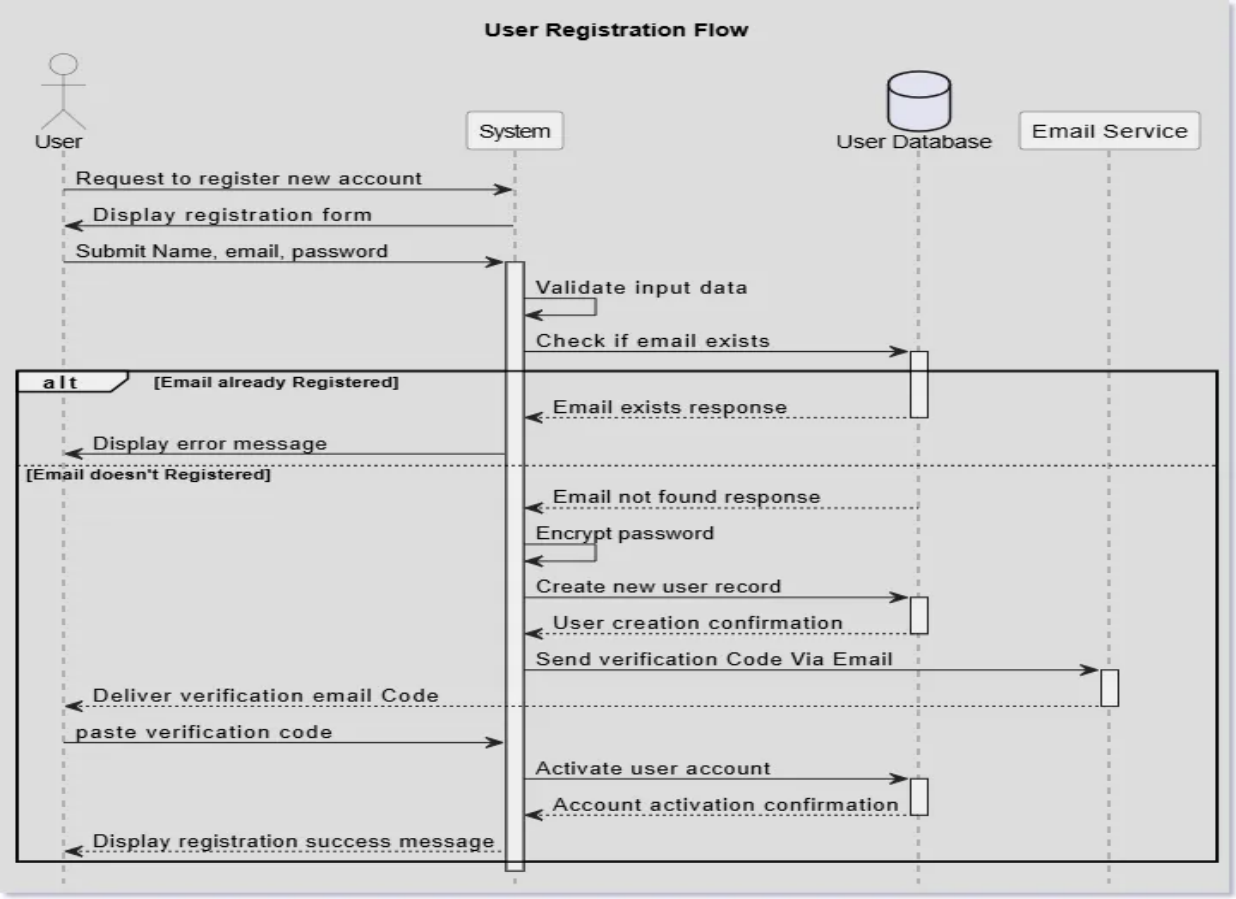


Figure 16: Sequence Diagram for User Registration

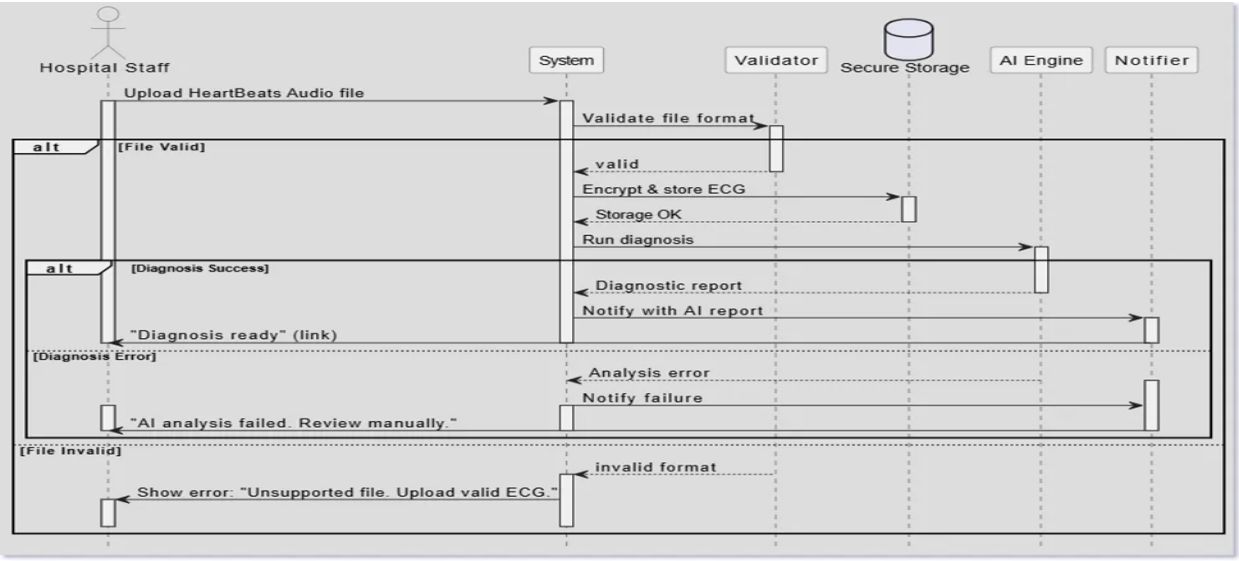


Figure 17: Sequence Diagram for Audio based Diagnosis

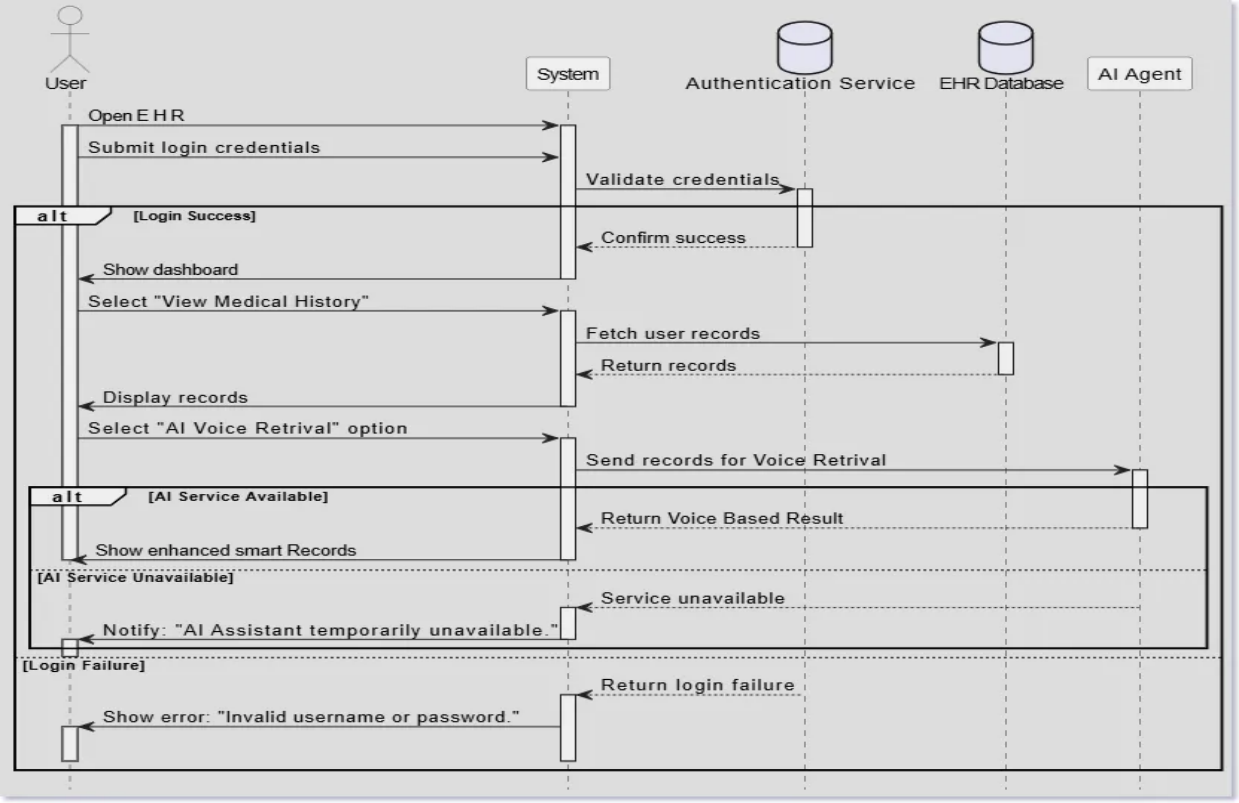


Figure 18: Sequence Diagram for EHR Access

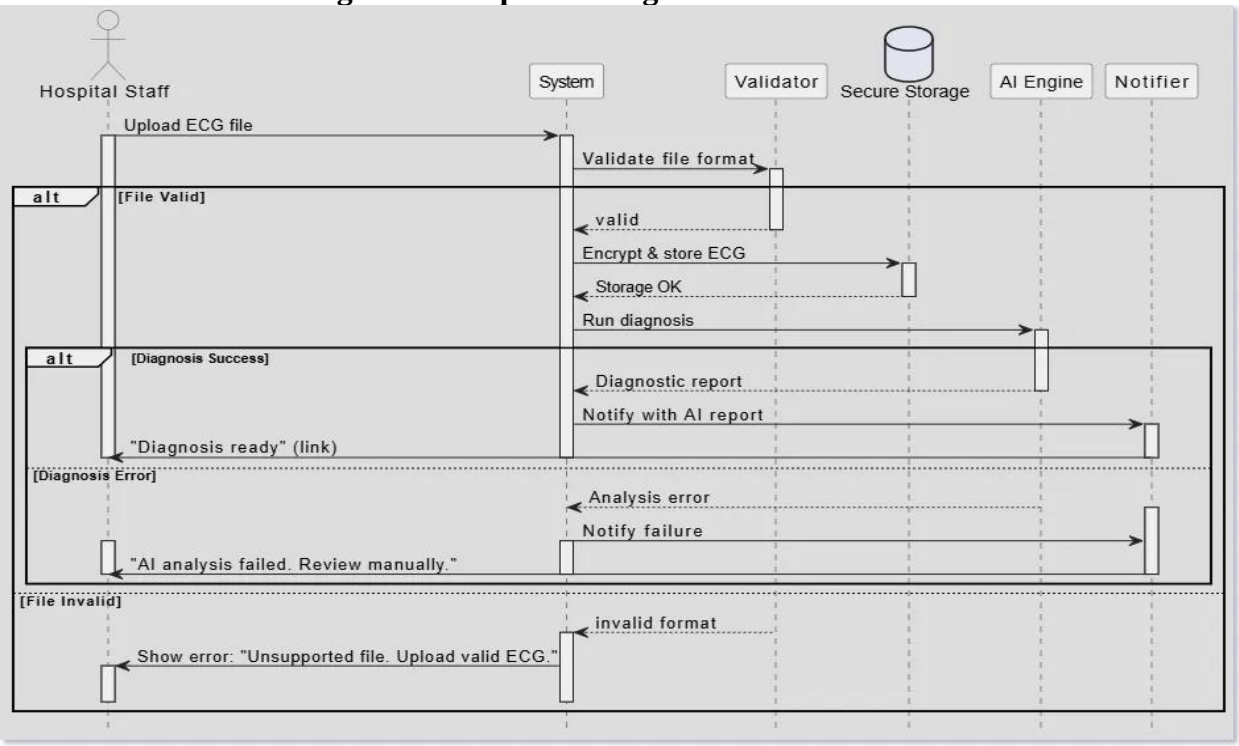


Figure 19: Sequence Diagram for ECG based Diagnosis

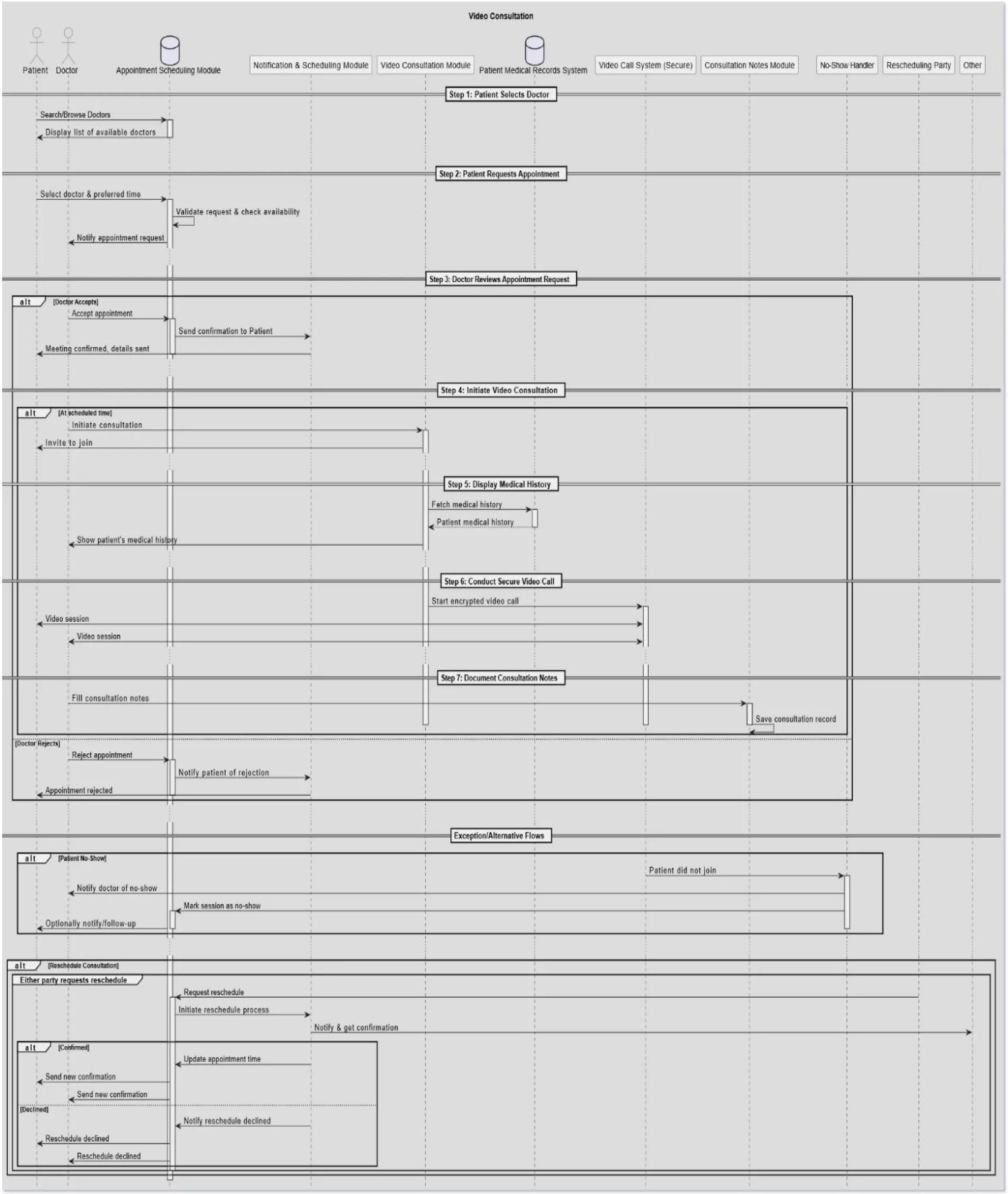


Figure 20: Sequence Diagram for Teleconsultation Process

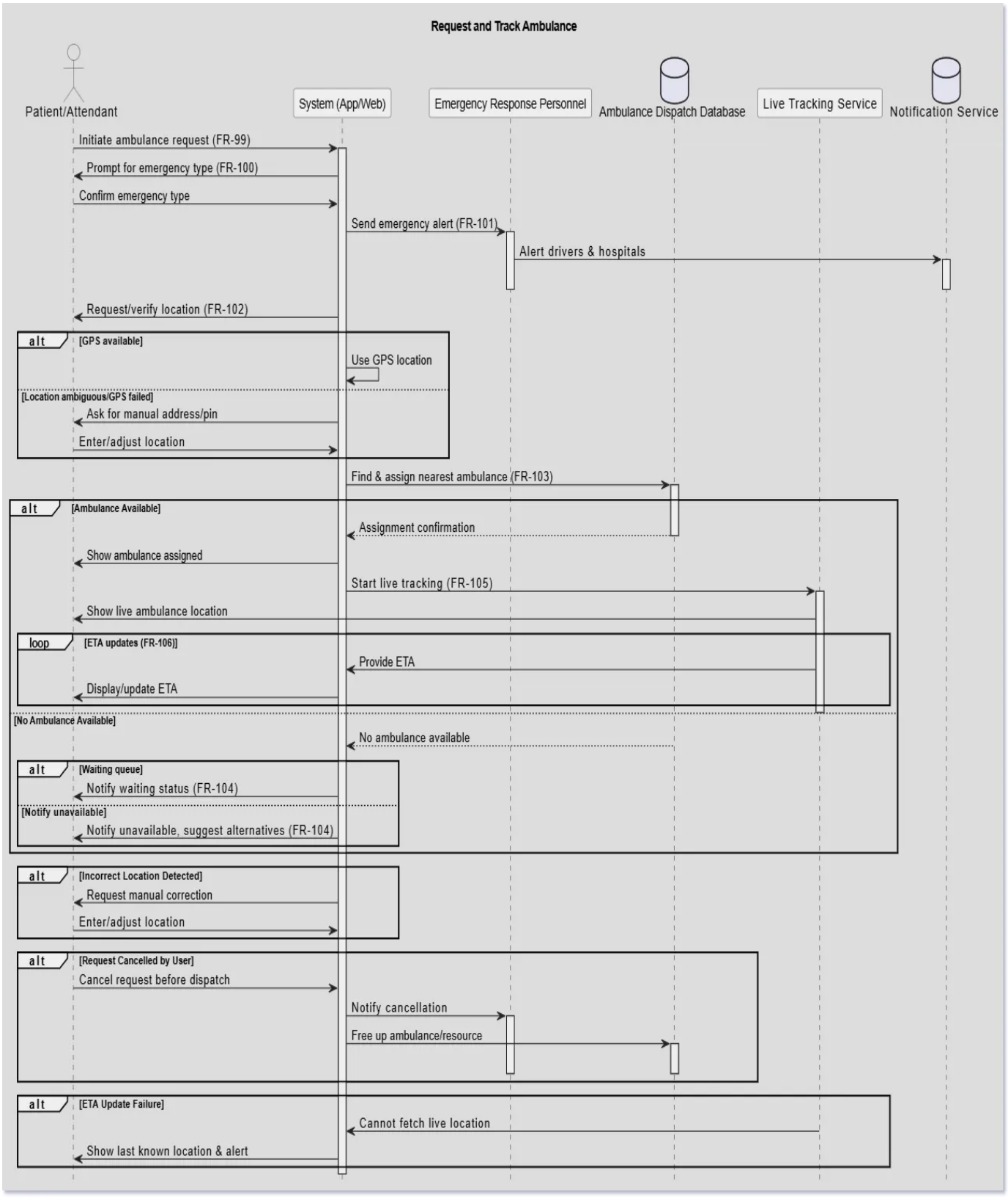


Figure 21: Sequence Diagram for Booking and Tracking Ambulance

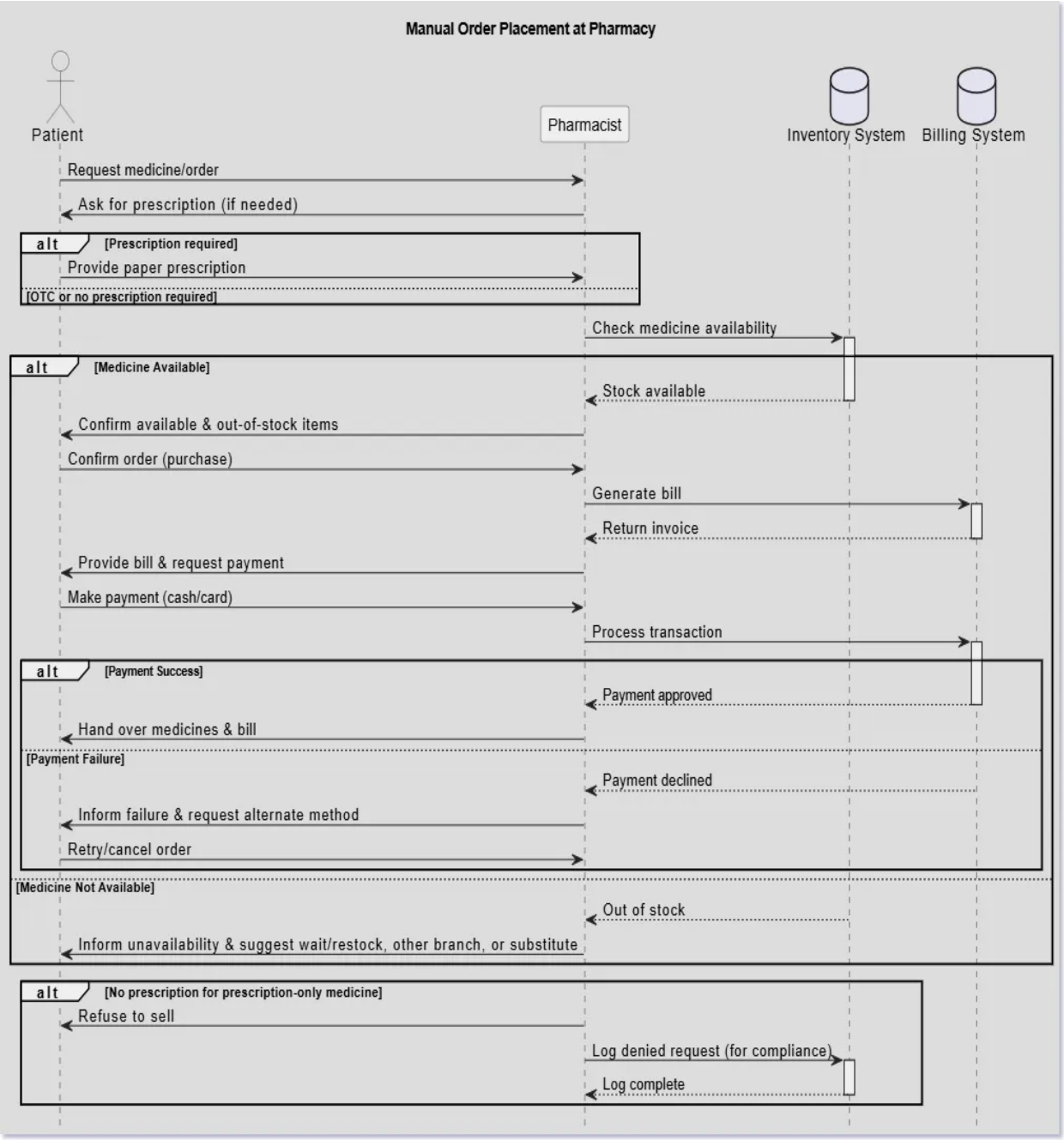


Figure 22: Sequence Diagram for Manual Pharmacy Order

4.4. State-Transition Diagrams:

This section shows the State Transition diagrams of CardioLink.

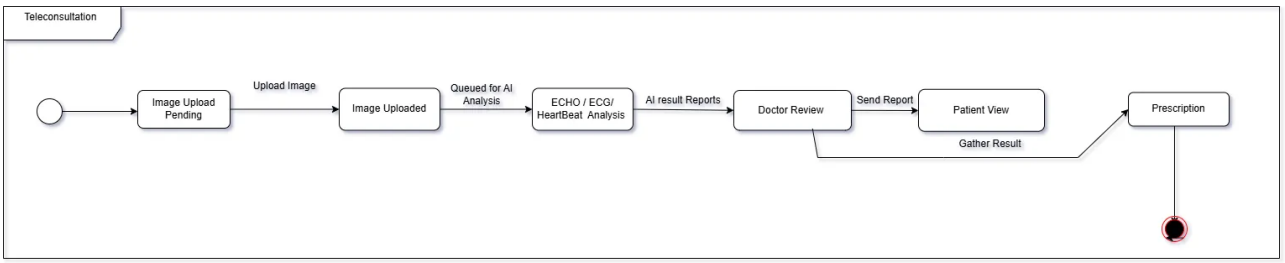


Figure 23: State Transition Diagram of AI Diagnosis

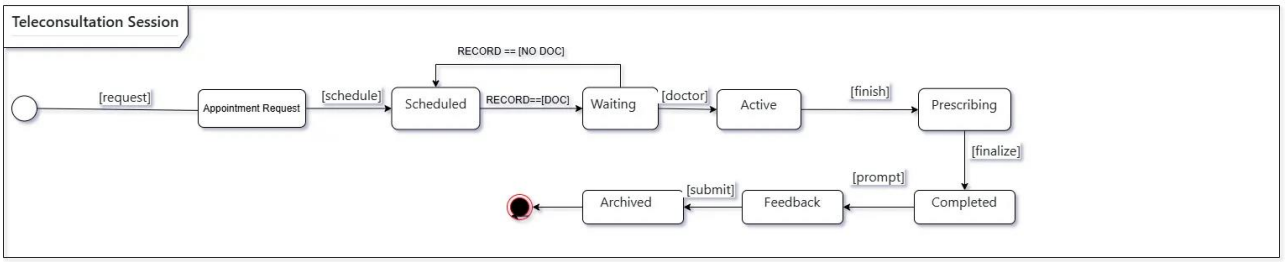


Figure 24: State Transition Diagram of Teleconsultation Process

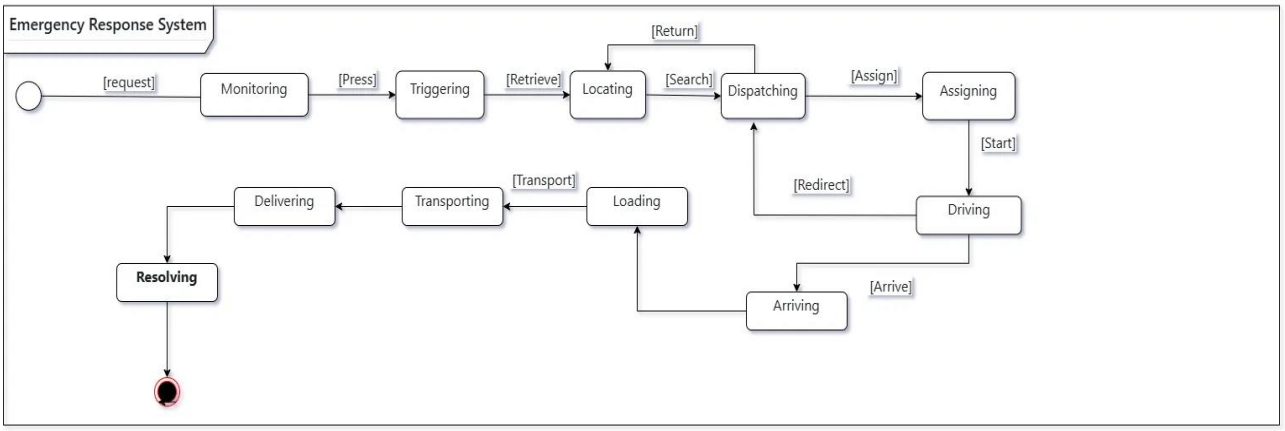


Figure 25: State Transition Diagram of Emergency Response System

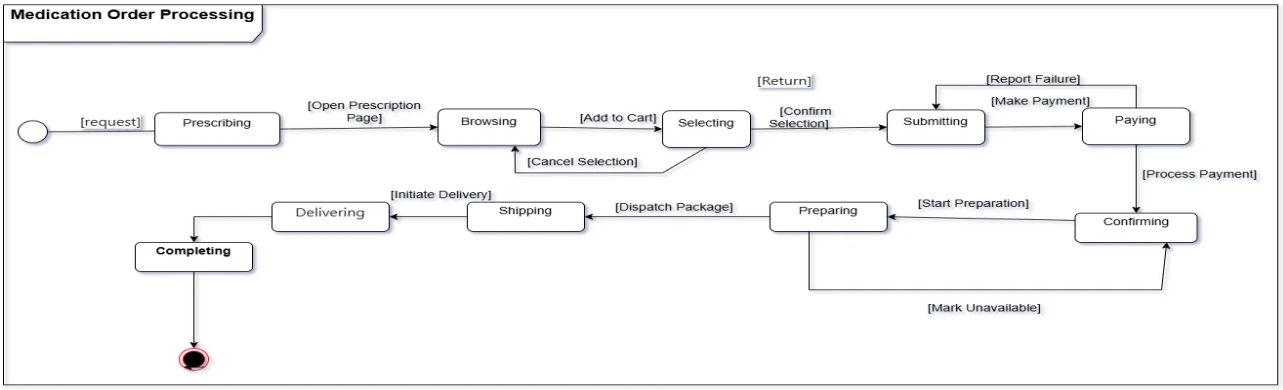


Figure 26: State Transition Diagram of Medication Order Processing

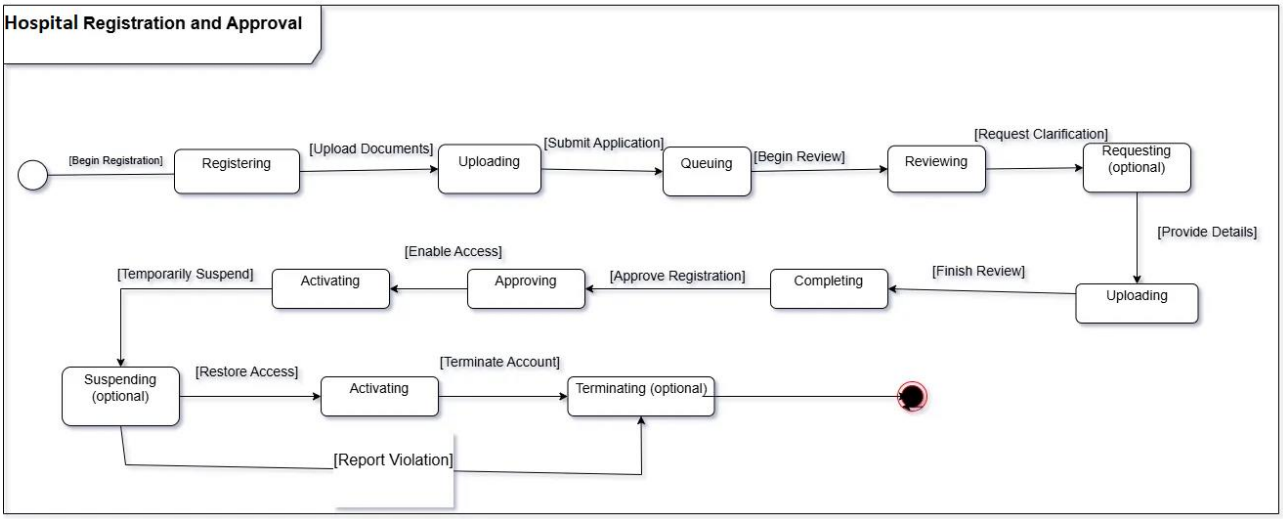


Figure 27: State Transition Diagram of Hospital Registration and Approval

5. Data Design

5.1 Schemas

5.1.1 Admin Schema

{

_id: ObjectId,

email: String (required, unique),

password: String (required),

name: String (required),

lastLogin: Date (default: Date.now),

isVerified: Boolean (default: false),

role: String (enum: "admin", required)

}

5.1.2 Patient Schema

```
{
 _id: ObjectId,
 firstName: String (required, trim: true),
 lastName: String (required, trim: true),
 dateOfBirth: Date (required),
 gender: String (enum: ['Male', 'Female', 'Other'], required),
 email: String (required, unique, trim: true, lowercase: true),
 password: String (required, minlength: 6),
 phoneNumber: String (required, trim: true),
 address: {
  street: String,
  city: String,
  state: String,
  zipCode: String,
  country: String
 },
 allergies: {
  medicinal: Array,
  environmental: Array,
  food: Array
 },
 insurance: {
  provider: String,
  policyNumber: String,
  groupNumber: String,
  expiryDate: Date
 }
}
```

5.1.3 Doctor Schema

```
{
 _id: ObjectId,
 email: String (required, unique),
 password: String (required),
```

```
  name: String (required),
  specialty: String,
  licenseNumber: String (required),
  consultationFee: Number,
  rating: Number (default: 0),
  isApproved: Boolean (default: false)
}
```

5.1.4 Ambulance Schema

```
{
  _id: ObjectId,
  vehicleNumber: String (required, unique),
  driverName: String (required),
  driverPhone: String (required),
  currentLocation: {
    lat: Number,
    lng: Number
  },
  status: String (enum: ["available", "busy", "maintenance"], default: "available"),
  equipmentLevel: String (enum: ["basic", "advanced", "critical"])
}
```

5.1.5 AmbulanceRequest Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  ambulanceId: ObjectId (ref: 'Ambulance'),
  emergencyLocation: {
    lat: Number,
    lng: Number,
    address: String
  } (required),
  emergencyType: String (enum: ["cardiac", "respiratory", "trauma", "other"]),
  urgencyLevel: String (enum: ["low", "medium", "high", "critical"], default: "medium"),
  status: String (enum: ["pending", "assigned", "en-route", "arrived", "completed", "cancelled"],
default: "pending"),
  requestTime: Date (default: Date.now)
}
```

5.1.6 Appointment Schema

```
{
  _id: ObjectId,
```

```
  patientId: ObjectId (ref: 'Patient', required),
  doctorId: ObjectId (ref: 'Doctor', required),
  appointmentDate: Date (required),
  type: String (enum: ["initial", "follow-up", "emergency", "second-opinion"], required),
  status: String (enum: ["scheduled", "confirmed", "in-progress", "completed", "cancelled", "no-
show"], default: "scheduled"),
  consultationFee: Number,
  reason: String (required)
}
```

5.1.7 DiagnosticReport Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  reportType: String (enum: ["ecg", "echo", "heartbeat", "heart-disease-prediction"], required),
  uploadedFile: String (required),
  analysisResults: Object (required),
  confidence: Number (0-100),
  reportStatus: String (enum: ["pending", "analyzed", "validated", "reviewed"], default:
"pending"),
  generatedAt: Date (default: Date.now)
}
```

5.1.8 DoctorRating Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  doctorId: ObjectId (ref: 'Doctor', required),
  appointmentId: ObjectId (ref: 'Appointment', required),
  rating: Number (required, min: 1, max: 5),
  review: String
}
```

5.1.9 HealthTracker Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  date: Date (required),
  physicalActivity: {
    steps: Number,
    exercises: Array,
    totalCaloriesBurned: Number
  },
```

```
  dietaryIntake: {
    meals: Array,
    totalCalories: Number,
    totalSaturatedFats: Number,
    totalCholesterol: Number
  },
  vitalSigns: {
    bloodPressure: Object,
    heartRate: Number,
    weight: Number
  },
  medicationCompliance: {
    prescribed: Array,
    taken: Array,
    missed: Array,
    complianceRate: Number
  }
}
```

5.1.10 VitalSign Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  date: Date (required),
  temperature: {
    value: Number,
    unit: String (enum: ['C', 'F'], default: 'C')
  },
  bloodPressure: {
    systolic: Number,
    diastolic: Number,
    unit: String (default: 'mmHg')
  },
  heartRate: {
    value: Number,
    unit: String (default: 'bpm')
  },
  weight: {
    value: Number,
    unit: String (enum: ['kg', 'lbs'], default: 'kg')
  },
```

```
  recordedBy: String (required)
}
```

5.1.11 LabResult Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  testName: String (required),
  testType: String (enum: ['Blood Test', 'Urine Test', 'Stool Test', 'Culture', 'Biopsy', 'Other'],
required),
  date: Date (required, default: Date.now),
  facility: String (required),
  results: [{
    parameter: String (required),
    value: String (required),
    unit: String,
    referenceRange: String,
    status: String (enum: ['Normal', 'High', 'Low', 'Critical'], default: 'Normal')
  }],
  status: String (enum: ['Pending', 'Completed', 'Cancelled'], default: 'Pending'),
  reportUrl: String (required)
}
```

5.1.12 Imaging Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  type: String (enum: ['X-Ray', 'MRI', 'CT Scan', 'Ultrasound', 'Mammogram', 'Other'], required),
  date: Date (required, default: Date.now),
  facility: String (required),
  findings: String (required),
  imageUrl: String (required),
  status: String (enum: ['Pending', 'Completed', 'Cancelled'], default: 'Pending')
}
```

5.1.13 Medication Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  name: String (required),
  dosage: String (required),
  frequency: String (required),
  startDate: Date (required),
```

```
  prescribedBy: String (required),
  status: String (enum: ['Active', 'Completed', 'Discontinued'], default: 'Active')
}
```

5.1.14 Product Schema

```
{
  _id: ObjectId,
  name: String (required),
  category: String (enum: ["medication", "supplement", "medical-device", "cardiovascular-
health"], required),
  description: String (required),
  price: Number (required),
  stockQuantity: Number (default: 0),
  prescriptionRequired: Boolean (default: false),
  isActive: Boolean (default: true)
}
```

5.1.15 Order Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  orderNumber: String (required, unique),
  items: [{
    productId: ObjectId (ref: 'Product'),
    name: String,
    quantity: Number,
    price: Number,
    prescriptionRequired: Boolean
  }],
  totalAmount: Number (required),
  orderStatus: String (enum: ["pending", "confirmed", "processing", "shipped", "delivered",
"cancelled"], default: "pending"),
  paymentStatus: String (enum: ["pending", "paid", "failed", "refunded"], default: "pending"),
  shippingAddress: {
    street: String,
    city: String,
    state: String,
    zipCode: String,
    country: String
  }
}
```

5.1.16 Pharmacist Schema

{
_id: ObjectId,
email: String (required, unique),
name: String (required),
licenseNumber: String (required),
pharmacyName: String,
pharmacyAddress: String
}

5.1.17 Radiologist Schema

{
_id: ObjectId,
email: String (required, unique),
name: String (required),
imagingExpertise: String,
licenseNumber: String (required),
hospitalAffiliation: String
}

5.1.18 HospitalAdmin Schema

{
_id: ObjectId,
email: String (required, unique),
name: String (required),
hospitalName: String,
hospitalAddress: String,
licenseNumber: String
}

5.1.19 MedicationAlert Schema

{
_id: ObjectId,
patientId: ObjectId (ref: 'Patient', required),
medicationName: String (required),
scheduledTime: Date (required),
frequency: String (enum: ["once", "daily", "twice-daily", "thrice-daily", "weekly"], required),
isActive: Boolean (default: true)
}

5.1.20 Visit Schema

{
_id: ObjectId,
patientId: ObjectId (ref: 'Patient', required),
date: Date (required),

```
  type: String (enum: ['routine', 'followup', 'emergency', 'specialist'], required),
  provider: String (required),
  reason: String (required),
  status: String (enum: ['scheduled', 'completed', 'cancelled', 'no-show'], default: 'scheduled')
}
```

5.1.21 Procedure Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  date: Date (required),
  procedureName: String (required),
  hospital: String (required),
  physician: String (required),
  indication: String (required),
  status: String (enum: ['Scheduled', 'Completed', 'Cancelled'], default: 'Scheduled')
}
```

5.1.22 MonthlyHealthReport Schema

```
{
  _id: ObjectId,
  patientId: ObjectId (ref: 'Patient', required),
  reportMonth: Date (required),
  activitySummary: {
    totalSteps: Number,
    totalExercises: Number,
    averageStepsPerDay: Number,
    mostActiveDay: Date
  },
  dietarySummary: {
    averageCalories: Number,
    averageSaturatedFats: Number,
    averageCholesterol: Number,
    healthiestDay: Date
  },
  vitalSignsSummary: {
    averageBloodPressure: Object,
    averageHeartRate: Number,
    weightChange: Number
  },
  recommendations: Array
}
```

5.2 Data Dictionary

5.2.1 Admin

System administrators who manage overall operations and user approvals.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each admin
email	String	Admin email address
password	String	Encrypted password for authentication
name	String	Full name of the admin
lastLogin	Date	Timestamp of last login
isVerified	Boolean	Email verification status
role	String	User role identifier

Table 1: Data Dictionary for Admin

5.2.2 Ambulance

Emergency response vehicles and services.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each ambulance
vehicleNumber	String	License plate or identification number
driverName	String	Name of the ambulance driver
driverPhone	String	Contact number of the driver (required)
currentLocation	Object	GPS coordinates of current position
status	String	Current availability status
equipmentLevel	String	Type of medical equipment available

Table 2: Data Dictionary for Ambulance

5.2.3 AmbulanceRequest

Emergency ambulance service requests.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each request
patientId	ObjectId	Reference to requesting patient
ambulanceId	ObjectId	Reference to assigned ambulance
emergencyLocation	Object	GPS coordinates of emergency location
emergencyType	String	Type of medical emergency
urgencyLevel	String	Priority level of the request

status	String	Current status of the request
requestTime	Date	Time when request was made

Table 3: Data Dictionary for AmbulanceRequest

5.2.4 Appointment

Teleconsultation appointments between patients and doctors.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each appointment
patientId	ObjectId	Reference to patient
doctorId	ObjectId	Reference to doctor
appointmentDate	Date	Scheduled date and time
type	String	Type of consultation
status	String	Current appointment status
consultationFee	Number	Fee for the consultation
reason	String	Reason for consultation

Table 4: Data Dictionary for Appointments

5.2.5 DiagnosticReport

AI-generated diagnostic reports from medical analyses.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each report
patientId	ObjectId	Reference to patient
reportType	String	Type of diagnostic analysis
uploadedFile	String	Reference to uploaded file
analysisResults	Object	AI analysis output
confidence	Number	Confidence score of AI analysis
reportStatus	String	Current status of the report
generatedAt	Date	Time when report was generated

Table 5: Data Dictionary for DiagnosticReport

5.2.6 Doctor

Medical professionals providing teleconsultations.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each doctor
email	String	Doctor email address
password	String	Encrypted password for authentication
name	String	Full name of the doctor

specialty	String	Medical specialization
licenseNumber	String	Medical license number
consultationFee	Number	Fee per consultation
rating	Number	Average patient rating
isApproved	Boolean	Admin approval status

Table 6: Data Dictionary for Doctor

5.2.7 Patient

Patients using the cardiovascular health system.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each patient
firstName	String	Patient's first name
lastName	String	Patient's last name
dateOfBirth	Date	Patient's date of birth
gender	String	Patient's gender
email	String	Email address
password	String	Encrypted password
phoneNumber	String	Contact phone number
address	Object	Patient's address
allergies	Object	Patient allergies
insurance	Object	Insurance information

Table 7: Data Dictionary for Patient

5.2.8 DoctorRating

Patient ratings and reviews for doctors.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each rating
patientId	ObjectId	Reference to patient who gave rating
doctorId	ObjectId	Reference to rated doctor
appointmentId	ObjectId	Reference to consultation appointment
rating	Number	Numerical rating
review	String	Written review comments

Table 8: Data Dictionary for DoctorRating

5.2.9 HealthTracker

Daily health monitoring data.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each tracking entry
patientId	ObjectId	Reference to patient

date	Date	Date of tracking entry
physicalActivity	Object	Physical activity data
dietaryIntake	Object	Food consumption data
vitalSigns	Object	Health metrics
medicationCompliance	Object	Medication adherence

Table 9: Data Dictionary for HealthTracker

5.2.10 VitalSign

Patient vital signs and measurements.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each vital sign record
patientId	ObjectId	Reference to patient
date	Date	Date of measurement (required)
temperature	Object	Body temperature
bloodPressure	Object	Blood pressure reading
heartRate	Object	Heart rate measurement
weight	Object	Body weight
recordedBy	String	Person who recorded the vitals

Table 10: Data Dictionary for VitalSigns

5.2.11 LabResult

Laboratory test results and reports.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each lab result
patientId	ObjectId	Reference to patient
testName	String	Name of the laboratory test
testType	String	Category of test
date	Date	Date of test
facility	String	Testing facility
results	Array	Test parameters and values
status	String	Current status of test
reportUrl	String	URL to lab report

Table 11: Data Dictionary for LabResult

5.2.12 Imaging

Medical imaging records and reports.

Attribute	Type	Description

_id	ObjectId	Unique identifier for each imaging record
patientId	ObjectId	Reference to patient
type	String	Type of imaging study
date	Date	Date of imaging study
facility	String	Facility where imaging was performed
findings	String	Imaging findings and interpretation
imageUrl	String	URL to imaging files
status	String	Current status of imaging

Table 12: Data Dictionary for Imaging

5.2.13 Medication

Patient medication records and prescriptions.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each medication record
patientId	ObjectId	Reference to patient
name	String	Name of the medication
dosage	String	Medication dosage
frequency	String	Frequency of administration
startDate	Date	Start date of medication
prescribedBy	String	Prescribing physician
status	String	Current medication status

Table 13: Data Dictionary for Medication

5.2.14 Product

Medications and health products available.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each product
name	String	Product name
category	String	Product category
description	String	Product description
price	Number	Product price
stockQuantity	Number	Available stock
prescriptionRequired	Boolean	Whether prescription is needed
isActive	Boolean	Whether product is available for sale

Table 14: Data Dictionary for Product

5.2.15 Order

Pharmacy orders placed by patients.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each order
patientId	ObjectId	Reference to patient
orderNumber	String	Unique order number
items	Array	Ordered items
totalAmount	Number	Total order amount
orderStatus	String	Current order status
paymentStatus	String	Payment status
shippingAddress	Object	Delivery address

Table 15: Data Dictionary for Order

5.2.16 Pharmacist

Pharmacists managing medication orders.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each pharmacist
email	String	Pharmacist email address
name	String	Full name of the pharmacist
licenseNumber	String	Pharmacy license number
pharmacyName	String	Name of associated pharmacy
pharmacyAddress	String	Pharmacy address

Table 16: Data Dictionary for Pharmacist

5.2.17 Radiologist

Radiologists interpreting medical imaging.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each radiologist
email	String	Radiologist email address
name	String	Full name of the radiologist
imagingExpertise	String	Areas of imaging expertise
licenseNumber	String	Medical license number
hospitalAffiliation	String	Associated hospital or clinic

Table 17: Data Dictionary for Radiologist

5.2.18 HospitalAdmin

Hospital administrators managing operations.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each hospital admin
email	String	Hospital admin email address

name	String	Full name of the hospital admin
hospitalName	String	Name of associated hospital
hospitalAddress	String	Hospital address
licenseNumber	String	Hospital license number

Table 18: Data Dictionary for HospitalAdmin

5.2.19 MedicationAlert

Medication reminders for patients.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each alert
patientId	ObjectId	Reference to patient
medicationName	String	Name of medication
scheduledTime	Date	Time for medication
frequency	String	Alert frequency
isActive	Boolean	Whether alert is active

Table 19: Data Dictionary for MedicationAlert

5.2.20 Visit

Patient visits and consultations.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each visit
patientId	ObjectId	Reference to patient
date	Date	Date of visit
type	String	Type of visit
provider	String	Healthcare provider name
reason	String	Reason for visit
status	String	Visit status

Table 20: Data Dictionary for Visit

5.2.21 Procedure

Medical procedures performed on patients.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each procedure
patientId	ObjectId	Reference to patient
date	Date	Date of procedure
procedureName	String	Name of the procedure
hospital	String	Hospital where procedure was performed

physician	String	Performing physician
indication	String	Medical indication for procedure
status	String	Procedure status

Table 21: Data Dictionary for Procedure

5.2.22 MonthlyHealthReport

Generated monthly health summaries.

Attribute	Type	Description
_id	ObjectId	Unique identifier for each report
patientId	ObjectId	Reference to patient
reportMonth	Date	Month and year of report
activitySummary	Object	Physical activity summary
dietarySummary	Object	Dietary intake summary
vitalSignsSummary	Object	Health metrics summary
recommendations	Array	Health recommendations

Table 22: Data Dictionary for MonthlyHealthReport

6. Implementation

6.1 Algorithms

Following are the algorithms used in implementing the Cardiovascular Health Management System:

6.1.1 User Authentication Algorithm

MODULE: User_Authentication
FUNCTION: authenticate_user()

```
INPUT: user_credentials, account_type
OUTPUT: authentication_status, user_session
BEGIN
    username = user_credentials.username
    password = user_credentials.password
    user_data = FETCH_USER_DATA(username)
    IF user_data IS NULL THEN
```

```
    RETURN "User not found"
  ENDIF
  IF VERIFY_PASSWORD(password, user_data.hashed_password) THEN
    otp = GENERATE_OTP()
    SEND_OTP(user_data.phone_number, otp)
    session_token = CREATE_SESSION_TOKEN(user_data.user_id)
    RETURN "OTP_SENT", session_token
  ELSE
    RETURN "Invalid credentials"
  ENDIF
END
```

6.1.2 Medical Records Upload Algorithm

MODULE: Medical_Records_Management
FUNCTION: upload_medical_record()

```
INPUT: file_data, patient_id, record_type
OUTPUT: upload_status, record_id
BEGIN
  IF VALIDATE_FILE_FORMAT(file_data) == FALSE THEN
    RETURN "Invalid file format"
  ENDIF
  encrypted_file = ENCRYPT_FILE(file_data)
  record_id = GENERATE_RECORD_ID()
  metadata = {
    record_id: record_id,
    patient_id: patient_id,
    record_type: record_type,
    upload_date: CURRENT_TIMESTAMP(),
    file_path: GENERATE_FILE_PATH(record_id)
```

```
    }
    STORE_FILE_SECURELY(encrypted_file, metadata.file_path)
    SAVE_METADATA(metadata)
    RETURN "Upload successful", record_id
END
```

6.1.3 AI ECG Analysis Algorithm

MODULE: ECG_AI_Interpreter
FUNCTION: analyze_ecg()

```
INPUT: ecg_file, patient_id
OUTPUT: diagnosis_result, confidence_score
BEGIN
    ecg_data = LOAD_ECG_FILE(ecg_file)
    processed_data = PREPROCESS_ECG(ecg_data)
    features = EXTRACT_ECG_FEATURES(processed_data)
    prediction = AI_ECG_MODEL.PREDICT(features)
    confidence = AI_ECG_MODEL.GET_CONFIDENCE()
    IF prediction >= ARRHYTHMIA_THRESHOLD THEN
        diagnosis = "Arrhythmia Detected"
    ELSE IF prediction >= MI_THRESHOLD THEN
        diagnosis = "MI Risk Detected"
    ELSE
        diagnosis = "Normal ECG"
    ENDIF
    STORE_DIAGNOSIS(patient_id, diagnosis, confidence)
    RETURN diagnosis, confidence
END
```

6.1.4 ECHO Video Analysis Algorithm

MODULE: AI_EchoVision
FUNCTION: analyze_echo_video()

```
INPUT: echo_video_file, patient_id
OUTPUT: heart_function_report
BEGIN
    video_frames = EXTRACT_FRAMES(echo_video_file)
    processed_frames = PREPROCESS_FRAMES(video_frames)
    ejection_fraction = AI_ECHO_MODEL.CALCULATE_EF(processed_frames)
    end_systolic_volume = AI_ECHO_MODEL.CALCULATE_ESV(processed_frames)
    end_diastolic_volume = AI_ECHO_MODEL.CALCULATE_EDV(processed_frames)
    report = {
        ejection_fraction: ejection_fraction,
        esv: end_systolic_volume,
        edv: end_diastolic_volume,
        analysis_date: CURRENT_TIMESTAMP()
    }
    STORE_ECHO_REPORT(patient_id, report)
    RETURN report
END
```

6.1.5 Heartbeat Classification Algorithm

MODULE: AI_Heartbeat_Classifier
FUNCTION: classify_heartbeat()

```
INPUT: audio_file, patient_id
OUTPUT: classification_result, murmur_location
BEGIN
    audio_data = LOAD_AUDIO_FILE(audio_file)
    features = EXTRACT_AUDIO_FEATURES(audio_data)
    classification = AI_AUDIO_MODEL.CLASSIFY(features)
    IF classification == "ABNORMAL" THEN
        murmur_detected = TRUE
        location = AI_AUDIO_MODEL.DETECT_MURMUR_LOCATION(features)
```

```
    ELSE
        murmur_detected = FALSE
        location = "None"
    ENDIF
    result = {
        classification: classification,
        murmur_present: murmur_detected,
        murmur_location: location
    }
    STORE_CLASSIFICATION(patient_id, result)
    RETURN result
END
```

6.1.6 Teleconsultation Booking Algorithm

MODULE: Teleconsultation_Management
FUNCTION: book_appointment()

```
INPUT: patient_id, doctor_id, preferred_time
OUTPUT: booking_status, appointment_id
BEGIN
    available_slots = GET_DOCTOR_AVAILABILITY(doctor_id, preferred_time)
    IF available_slots IS EMPTY THEN
        RETURN "No slots available"
    ENDIF
    selected_slot = FIND_CLOSEST_SLOT(available_slots, preferred_time)
    appointment_id = GENERATE_APPOINTMENT_ID()
    appointment = {
        appointment_id: appointment_id,
        patient_id: patient_id,
        doctor_id: doctor_id,
        scheduled_time: selected_slot,
```

```
    status: "BOOKED"
  }
  SAVE_APPOINTMENT(appointment)
  SEND_CONFIRMATION(patient_id, doctor_id, appointment)
  RETURN "Booking successful", appointment_id
END
```

6.1.7 Pharmacy Order Processing Algorithm

MODULE: PharmaLink
FUNCTION: process_medication_order()

```
INPUT: prescription_id, patient_address
OUTPUT: order_status, delivery_estimate
BEGIN
  prescription = GET_PRESCRIPTION(prescription_id)
  FOR EACH medication IN prescription.medications DO
    stock_available = CHECK_STOCK(medication.name, medication.quantity)
    IF stock_available == FALSE THEN
      RETURN "Medication unavailable"
    ENDIF
  ENDFOR
  order_id = GENERATE_ORDER_ID()
  total_cost = CALCULATE_ORDER_COST(prescription.medications)
  order = {
    order_id: order_id,
    prescription_id: prescription_id,
    patient_address: patient_address,
    total_cost: total_cost,
    status: "PROCESSING"
  }
  SAVE_ORDER(order)
```

```
    DISPATCH_ORDER(order_id)
    delivery_time = ESTIMATE_DELIVERY_TIME(patient_address)
    RETURN "Order processed", delivery_time
END
```

6.1.8 HeartWise AI Assistant Algorithm

MODULE: HeartWise_AI
FUNCTION: provide_health_guidance()

```
INPUT: user_query, patient_health_profile
OUTPUT: ai_response, educational_content
BEGIN
    query_intent = ANALYZE_QUERY_INTENT(user_query)
    health_context = GET_HEALTH_CONTEXT(patient_health_profile)
    IF query_intent == "SYMPTOM_INQUIRY" THEN
        response = GENERATE_SYMPTOM_GUIDANCE(user_query, health_context)
    ELSE IF query_intent == "PREVENTION_ADVICE" THEN
        response = GENERATE_PREVENTION_ADVICE(health_context)
    ELSE IF query_intent == "LIFESTYLE_GUIDANCE" THEN
        response = GENERATE_LIFESTYLE_RECOMMENDATIONS(health_context)
    ELSE
        response = GENERATE_GENERAL_EDUCATION(user_query)
    ENDIF
    educational_links = GET_RELEVANT_RESOURCES(query_intent)
    RETURN response, educational_links
END
```

6.1.9 Health Tracking Algorithm

MODULE: CardioHealth_Tracker
FUNCTION: track_daily_health()

```
INPUT: patient_id, activity_data, dietary_data, vital_signs
```

```
OUTPUT: tracking_summary, health_alerts
BEGIN
    daily_summary = {
        steps: activity_data.steps,
        calories_intake: SUM(dietary_data.meals.calories),
        blood_pressure: vital_signs.bp_reading,
        heart_rate: vital_signs.hr_reading
    }
    // Check for alerts
    alerts = []
    IF daily_summary.blood_pressure.systolic > 140 THEN
        ADD_ALERT(alerts, "High blood pressure detected")
    ENDIF

    IF daily_summary.heart_rate > 100 THEN
        ADD_ALERT(alerts, "Elevated heart rate")
    ENDIF
    STORE_DAILY_TRACKING(patient_id, daily_summary)
    IF alerts IS NOT EMPTY THEN
        SEND_HEALTH_ALERTS(patient_id, alerts)
    ENDIF
    RETURN daily_summary, alerts
END
```

6.1.10 Heart Disease Prediction Algorithm

MODULE: Heart_Disease_Prediction
FUNCTION: predict_heart_disease_risk()

```
INPUT: patient_health_data
OUTPUT: risk_probability, risk_level, recommendations
BEGIN
```

```
    features = [
        patient_health_data.age,
        patient_health_data.cholesterol,
        patient_health_data.blood_pressure,
        CALCULATE_BMI(patient_health_data.weight, patient_health_data.height),
        ENCODE_SMOKING_STATUS(patient_health_data.smoking)
    ]
    normalized_features = NORMALIZE_FEATURES(features)
    risk_probability = ML_MODEL.PREDICT(normalized_features)
    IF risk_probability >= 0.7 THEN
        risk_level = "High Risk"
    ELSE IF risk_probability >= 0.4 THEN
        risk_level = "Moderate Risk"
    ELSE
        risk_level = "Low Risk"
    ENDIF

    recommendations = GENERATE_RECOMMENDATIONS(risk_level, patient_health_data)
    RETURN risk_probability, risk_level, recommendations
END
```

6.1.11 Ambulance Assignment Algorithm

MODULE: Ambulance_Connect
FUNCTION: request_ambulance()

```
INPUT: patient_location, emergency_type
OUTPUT: ambulance_id, estimated_arrival
BEGIN
    available_ambulances = GET_AVAILABLE_AMBULANCES()
    min_distance = INFINITY
    selected_ambulance = NULL
```

```
    FOR EACH ambulance IN available_ambulances DO
        distance = CALCULATE_DISTANCE(patient_location, ambulance.location)
        IF distance < min_distance THEN
            min_distance = distance
            selected_ambulance = ambulance
        ENDIF
    ENDFOR
    IF selected_ambulance IS NOT NULL THEN
        ASSIGN_AMBULANCE(selected_ambulance.id, patient_location)
        estimated_time = CALCULATE_ETA(selected_ambulance.location, patient_location)
        RETURN selected_ambulance.id, estimated_time
    ELSE
        RETURN "No ambulance available"
    ENDIF
END
```

6.1.12 Doctor Schedule Management Algorithm

MODULE: Physician_Gateway
FUNCTION: manage_doctor_schedule()

```
INPUT: doctor_id, availability_data, schedule_action
OUTPUT: schedule_status, updated_slots

BEGIN
    current_schedule = GET_DOCTOR_SCHEDULE(doctor_id)
    IF schedule_action == "ADD_AVAILABILITY" THEN
        new_slots = CREATE_TIME_SLOTS(availability_data.date, availability_data.time_range)
        updated_schedule = MERGE_SCHEDULES(current_schedule, new_slots)
    ELSE IF schedule_action == "BLOCK_TIME" THEN
        updated_schedule = BLOCK_TIME_SLOTS(current_schedule,
availability_data.blocked_slots)
```

```
    ELSE IF schedule_action == "CANCEL_APPOINTMENT" THEN
        updated_schedule = CANCEL_APPOINTMENT_SLOT(current_schedule,
availability_data.appointment_id)
        NOTIFY_PATIENT(availability_data.appointment_id, "Appointment cancelled")
    ENDIF
    SAVE_DOCTOR_SCHEDULE(doctor_id, updated_schedule)
    available_slots = GET_AVAILABLE_SLOTS(updated_schedule)
    RETURN "Schedule updated", available_slots
END
```

6.1.13 Admin System Management Algorithm

MODULE: Admin_Control_Panel
FUNCTION: manage_system_operations()

```
INPUT: admin_action, target_entity, action_data
OUTPUT: operation_status, system_response
BEGIN
    IF admin_action == "APPROVE_DOCTOR_REGISTRATION" THEN
        doctor_id = action_data.doctor_id
        ACTIVATE_DOCTOR_ACCOUNT(doctor_id)
        SEND_APPROVAL_NOTIFICATION(doctor_id)
        result = "Doctor approved successfully"
    ELSE IF admin_action == "APPROVE_AMBULANCE_SERVICE" THEN
        service_id = action_data.service_id
        ACTIVATE_AMBULANCE_SERVICE(service_id)
        result = "Ambulance service approved"
    ELSE IF admin_action == "SEND_SYSTEM_NOTIFICATION" THEN
        BROADCAST_NOTIFICATION(action_data.message, action_data.target_users)
        result = "Notification sent"
    ELSE IF admin_action == "GENERATE_SYSTEM_REPORT" THEN
        report = GENERATE_ADMIN_REPORT(action_data.report_type, action_data.date_range)
```

```
        result = report
    ENDIF
    LOG_ADMIN_ACTION(admin_action, target_entity, CURRENT_TIMESTAMP())
    RETURN "Operation completed", result
END
```

6.2. External APIs/SDKs

The third-party APIs/SDKs used in the project implementation in the following table

No.	API Name & Version	Description	Purpose	Endpoints/Functions
1.	**Stripe Payment API v2020-08-27**	Online payment processing	Process payments for teleconsultations and pharmacy orders	stripe.paymentMethods.create stripe.paymentIntents.create
2.	**Google Maps API v3**	Location and mapping services	Track ambulance location and calculate routes for emergency services	maps.googleapis.com/maps/api/geocode maps.googleapis.com/maps/api/directions
3.	**TensorFlow Serving API v2.8**	Machine learning model serving	Deploy and serve AI models for ECG analysis and heart disease prediction	/v1/models/ecg_model:predict /v1/models/heart_disease_model:predict
4.	**Cloudinary API v1.1**	Media management platform	Process and store medical images and videos with automatic optimization	https://api.cloudinary.com/v1_1/{cloud-name}/upload
5.	**Socket.IO v4.0**	Real-time bidirectional communication	Enable real-time updates for ambulance tracking and emergency alerts	io.connect() socket.emit()

Table 23: External APIs/SDKs

6.3. User Interface

6.3.1 Landing Page

It features **role-based access portals** (Hospital, System Admin, Pharmacist) with dedicated login/sign-up options, ensuring tailored workflows for each user type. The clean design highlights core functionalities, emphasizing integration and efficiency in cardiac healthcare management.

Figure 28: Landing Page

6.3.2 Authentication

These sign-in pages provide secure, role-based access for hospital personnel (doctors, lab technologists, radiologists, administrators) and system-wide users (pharmacists, system admins). Each portal offers tailored workflows, ensuring efficient management of hospital operations, medications, and platform configurations.

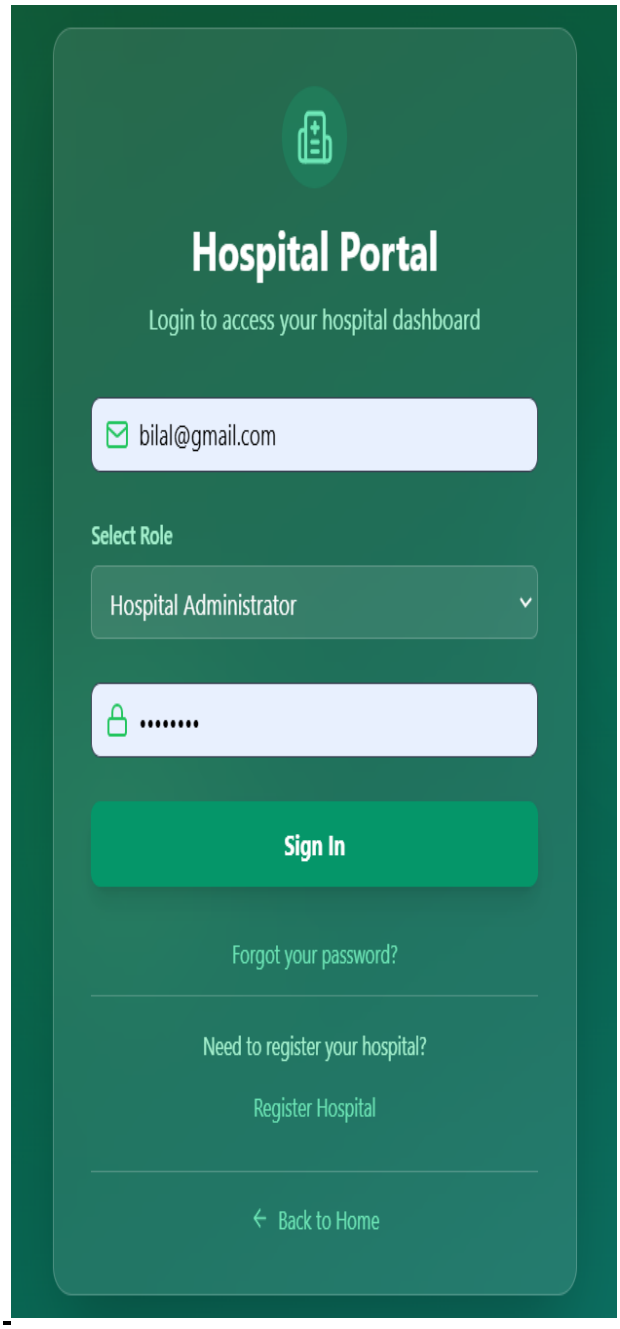


Figure 29: Hospital Sign in Page

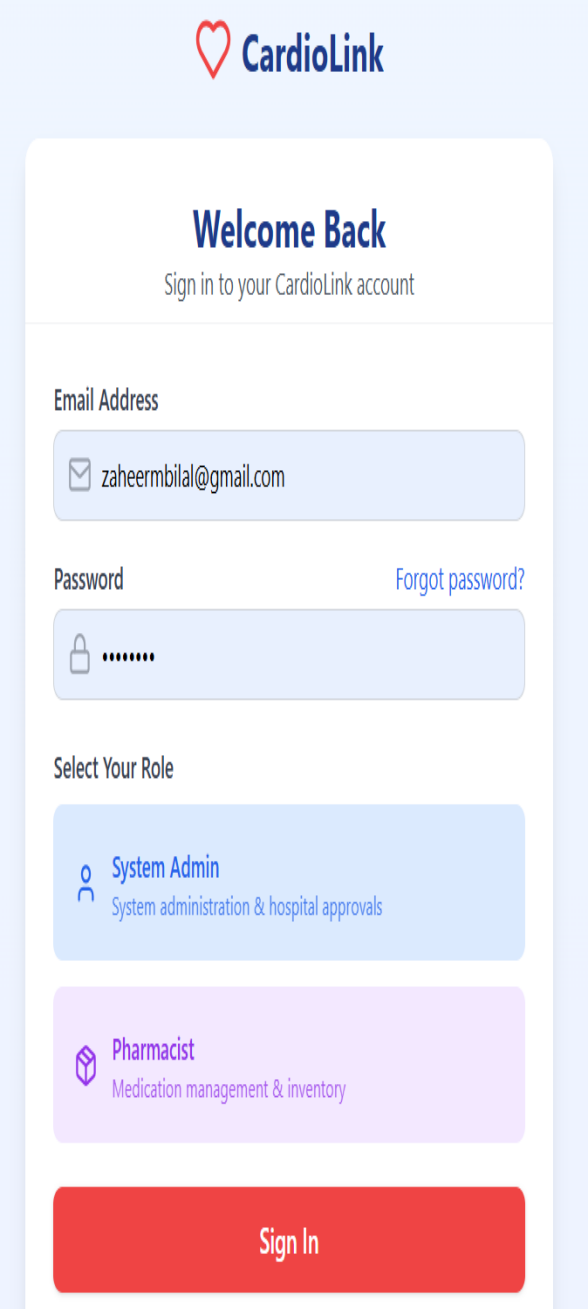


Figure 30: System Admin & pharmacist Sign in Page

6.3.3 System Admin Dashboard

This dashboard provides the system administrator with comprehensive control, including approving/rejecting hospital registration requests, monitoring real-time analytics, and managing user activity. It offers an overview of system performance, user distribution, and pending tasks for efficient platform administration.

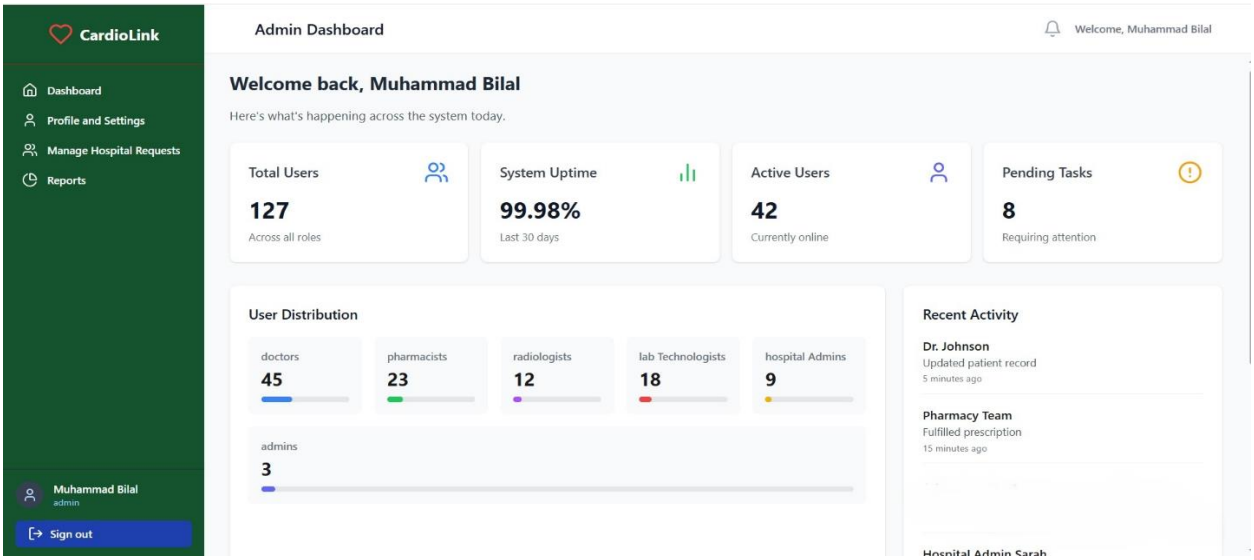


Figure 31: System Admin Dashboard

6.3.4. Hospital Admin Dashboard

This dashboard empowers hospital administrators to manage staff (doctors, lab technologists, radiologists, etc.), oversee hospital operations, and approve/update personnel records. It provides centralized control for seamless coordination of hospital workflows and user management.

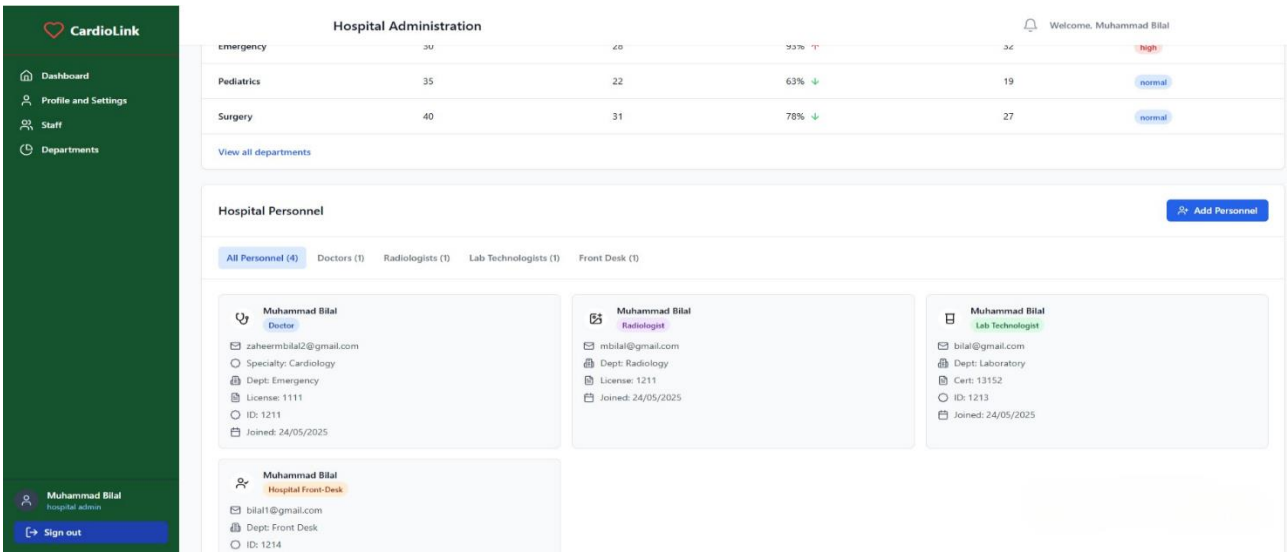


Figure 32: Hospital Admin Dashboard

6.3.5 Lab-Technologist Dashboard

This dashboard enables lab technologists to upload, manage, and track diagnostic reports for their hospital. It provides a streamlined interface for efficient handling of lab data and patient test results.

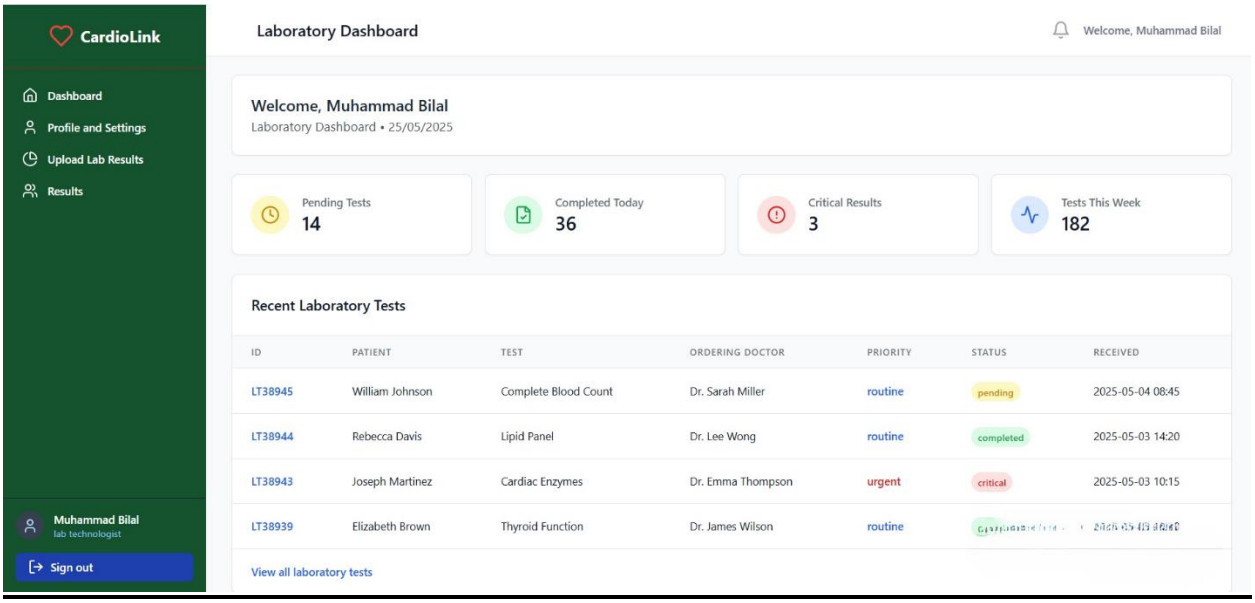


Figure 33: Lab-Technologist Dashboard

6.3.6 Front desk officers

This dashboard allows front desk officers to manage patient hospitalizations, including associated procedures, imaging, and lab results for their facility. It provides centralized tracking and coordination of all patient admission documentation and diagnostic records.

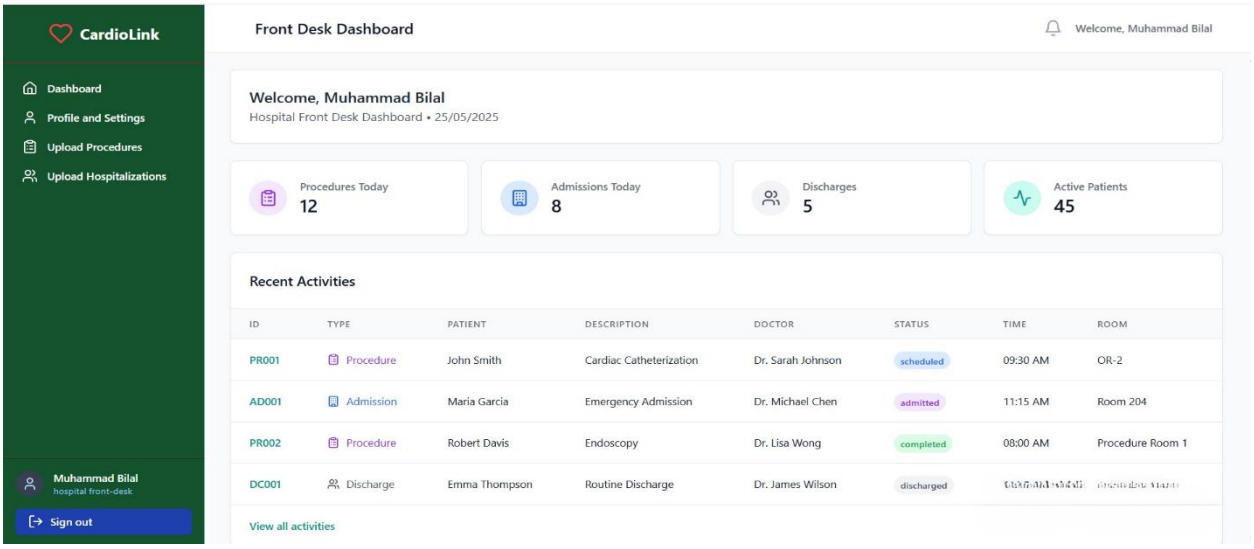


Figure 34: Front desk officer Dashboard

6.3.7. Hospital Registration

This multi-stage form allows hospitals to submit registration requests by providing details and uploading required documents. Upon successful submission, the request is sent to the system administrator for review and approval.

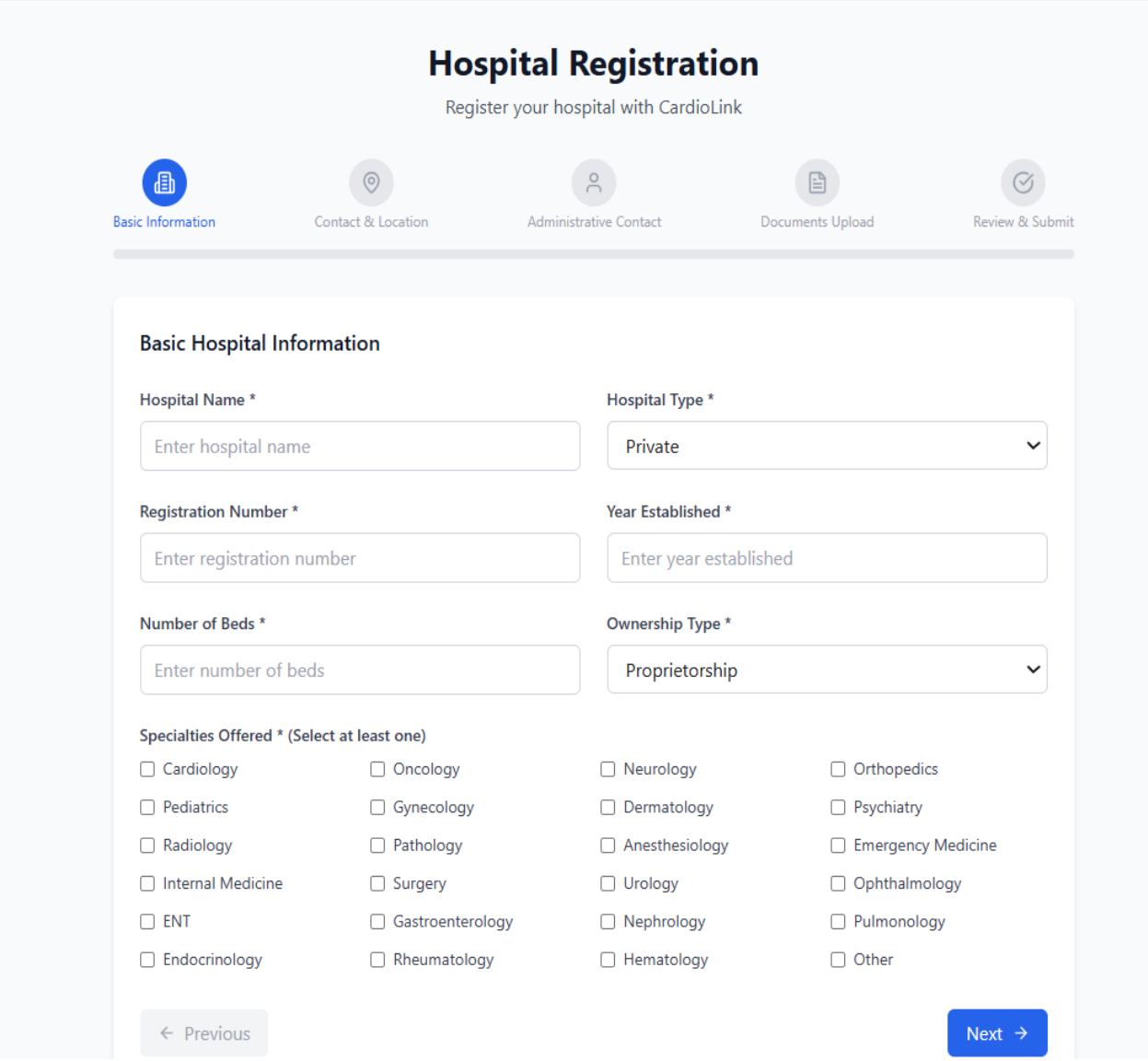


Figure 35: Hospital Registration

6.3.8. Electronic Health Record for Doctors

This interface provides doctors with comprehensive access to patient medical records, including hospitalizations, procedures, imaging, visits, vital signs, allergies, and special directives. It enables detailed review of specific records for informed clinical decision-making.

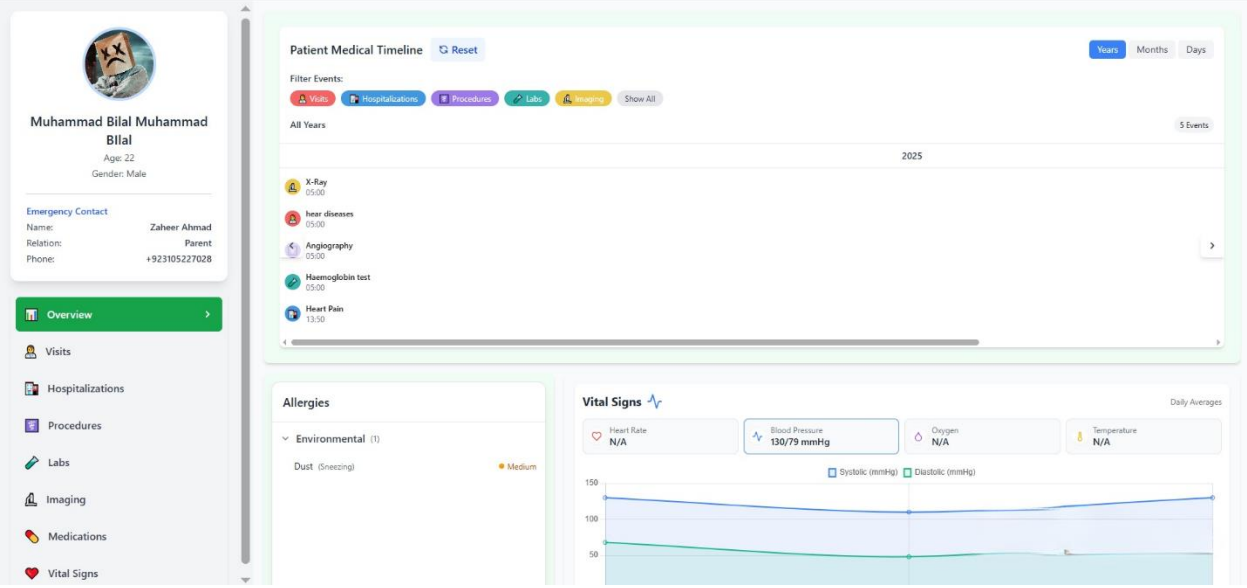


Figure 36: EHR for Doctor

6.3.9. ECG Analysis Page

This interface allows radiologists to upload ECG files, which the system automatically analyzes to provide AI-powered diagnostic insights and interpretations

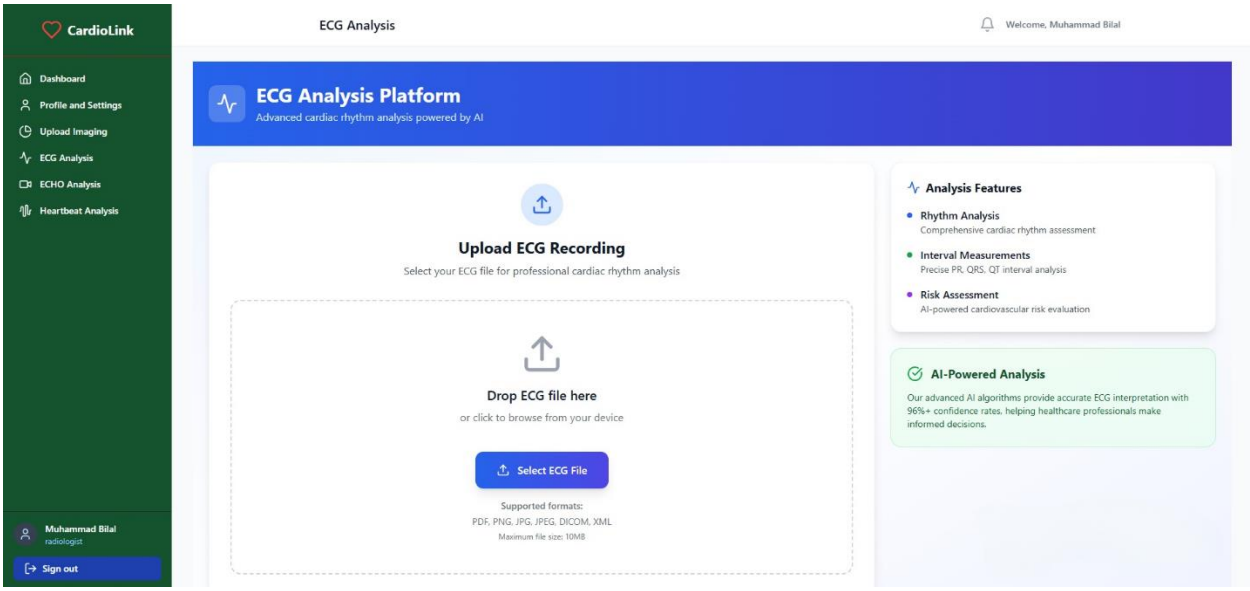


Figure 37: ECG Analysis Page

6.3.10. ECHO Analysis Page

Radiologists can submit echocardiogram files here, where the system evaluates them and delivers AI-generated diagnostic reports for cardiac assessment.

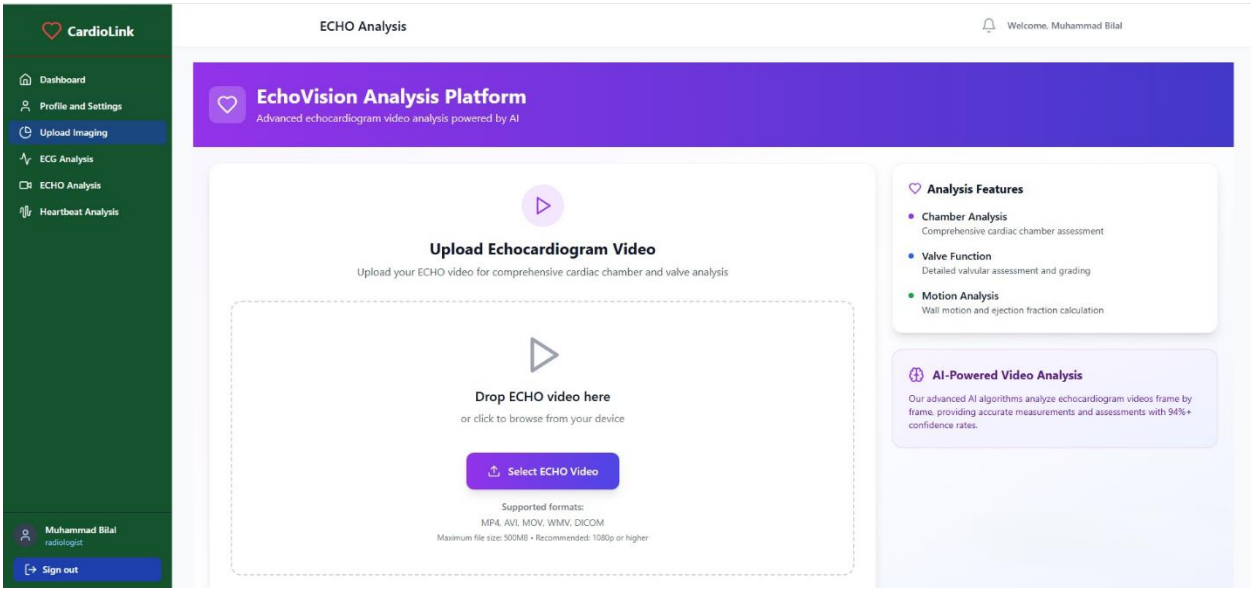


Figure 38: ECHO Analysis Page

6.3.10. HeartBeat Analysis Page

This page enables radiologists to upload heartbeat rhythm data for automated AI analysis, producing real-time diagnostic feedback and potential abnormalities detection.

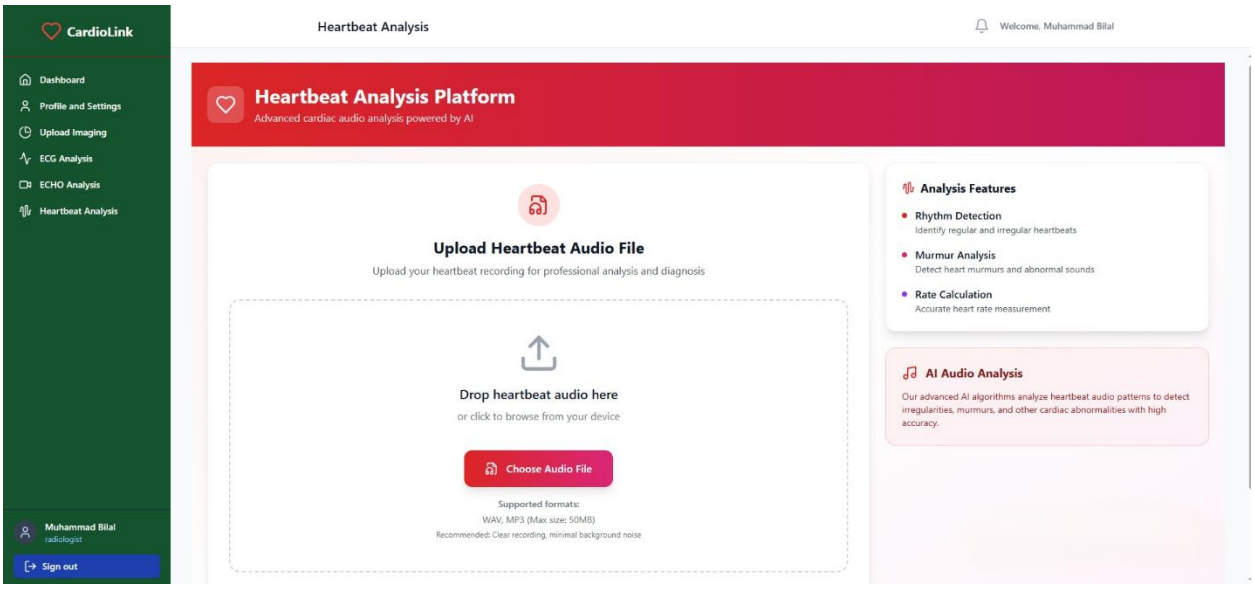


Figure 39: HeartBeat Analysis Page

6.4. Deployment

This section provides details about the current deployment stage of our application and future plans.

Current Deployment

Our application is currently in the development phase and is running on localhost. This setup allows us to efficiently develop and test features in a local environment before moving to staging and production phases. The local development environment provides a controlled space where we can build functionality, debug issues, and ensure the stability of the application before deployment.

Future Deployment Plans

In the future, we plan to deploy our application using Azure Kubernetes Service (AKS) with Kubernetes. This transition will enable us to leverage the scalability, reliability, and orchestration capabilities of Kubernetes, ensuring that our application can handle increased traffic and provide a seamless user experience. AKS will allow us to automate the deployment, scaling, and management of our containerized applications, providing a robust infrastructure for our production environment.

Version Control

Throughout the development process, we have utilized Git for version control. Git has facilitated efficient collaboration among our development team, enabling us to manage code changes, track progress, and maintain a history of all modifications. By using Git, we ensure that our codebase remains organized, and we can easily revert to previous versions if necessary.

7. Testing and Evaluation

7.1 Unit Testing

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Validate email format in registration	Email: doctor@cardiolink.com	Accepts valid email format	Pass
2	Validate password strength	Password: SecurePass@123	Accepts strong password meeting criteria	Pass
3	Validate required name field	Name: Dr. John Smith	Accepts valid name	Pass

4	Validate role selection	Role: doctor	Accepts valid role from enum	Pass
5	Reject invalid email format	Email: invalid-email	Shows email format error	Pass
6	Reject weak password	Password: 123	Shows password strength error	Pass
7	Reject empty required fields	Name: (empty)	Shows required field error	Pass
8	Reject invalid role	Role: invalid-role	Shows invalid role error	Pass
9	Generate verification token	Valid registration data	Creates 6-digit verification token	Pass
10	Hash password before storage	Password: plaintext	Stores bcrypt hashed password	Pass

Table 24: Unit Test Table for User Registration and Signup

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Validate correct login credentials	Email: doctor@cardiolink.com, Password: correct, Role: doctor	Returns success with user data	Pass
2	Reject incorrect password	Email: doctor@cardiolink.com, Password: wrong	Returns authentication error	Pass
3	Reject non-existent user	Email: nonexistent@test.com	Returns user not found error	Pass
4	Reject unverified user login	Verified: false	Returns verification required error	Pass
5	Generate JWT token on success	Valid credentials	Creates valid JWT token	Pass
6	Set authentication cookie	Valid login	Sets httpOnly authentication cookie	Pass
7	Hospital admin login validation	Email: admin@hospital.com, Password: correct	Validates against hospital admin credentials	Pass
8	Role-based login validation	Email: doctor@test.com, Role: pharmacist	Rejects role mismatch	Pass
9	Update last login timestamp	Successful login	Updates user lastLogin field	Pass
10	Logout functionality	Valid session	Clears authentication token and cookie	Pass

Table 25: Unit Test Table for User Authentication and Login

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Verify valid OTP code	OTP: 123456	Accepts valid verification code	Pass
2	Reject invalid OTP code	OTP: 999999	Shows invalid verification code error	Pass
3	Reject expired OTP code	OTP: expired token	Shows verification code expired error	Pass
4	Set user as verified	Valid OTP	Updates user isVerified to true	Pass
5	Send verification email	Valid user email	Sends email with 6-digit code	Pass
6	Generate new verification code	Request resend	Creates new 6-digit token	Pass
7	Set token expiration	Token creation	Sets 24-hour expiration	Pass

Table 26: Unit Test Table for Email Verification

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Upload valid PDF file	File: medical_record.pdf	File uploaded successfully	Pass
2	Upload valid image file	File: xray_image.jpg	Image uploaded successfully	Pass
3	Reject invalid file type	File: malicious.exe	Shows invalid file type error	Pass
4	Validate file size limit	File: large_file.pdf (>10MB)	Shows file size limit error	Pass
5	Generate unique filename	File: document.pdf	Creates unique filename with timestamp	Pass
6	Store file metadata	Uploaded file	Saves original name, size, type in database	Pass
7	Create upload directory	File upload	Creates uploads directory if not exists	Pass
8	Multiple file upload	Files: [doc1.pdf, doc2.jpg]	Uploads multiple files successfully	Pass

Table 27: Unit Test Table for File Upload Functionality

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Create visit record	Visit data with required fields	Visit created successfully	Pass
2	Create lab result record	Lab result with test data	Lab result created successfully	Pass
3	Create imaging record	Imaging data with file	Imaging record created successfully	Pass
4	Create procedure record	Procedure data	Procedure record created successfully	Pass
5	Create hospitalization record	Hospitalization data	Hospitalization created successfully	Pass
6	Validate required patient ID	PatientId: missing	Shows patient ID required error	Pass
7	Validate user hospital association	User without hospitalId	Shows hospital association required error	Pass
8	Auto-populate upload metadata	Valid record data	Sets uploadedBy and hospitalId from user	Pass
9	Parse JSON arrays from form data	FormData with arrays	Correctly parses associatedLabResults array	Pass
10	Validate date formats	Date: invalid format	Shows invalid date format error	Pass

Table 28: Unit Test Table for Medical Record Creation

7.2. Functional Testing Tables

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Login as System Admin	Email: admin@cardiolink.com, Password: admin123, Role: admin	Admin dashboard loads with hospital management features	Admin dashboard with hospital approval interface	Pass
2	Login as Doctor	Email: doctor@hospital.com, Password: doc123, Role: doctor	Doctor dashboard with EHR	Doctor dashboard with EHR,	Pass

			and patient management	appointments, prescriptions	
3	Login as Hospital Admin	Email: hadmin@hospital.com, Password: hadmin123	Hospital admin dashboard with staff management	Hospital admin dashboard with personnel management	Pass
4	Login as Pharmacist	Email: pharmacist@cardiolink.com, Password: pharm123, Role: pharmacist	Pharmacist dashboard with medication management	Pharmacist dashboard with prescriptions and medications	Pass
5	Login as Radiologist	Email: radiologist@hospital.com, Password: rad123, Role: radiologist	Radiologist dashboard with imaging features	Radiologist dashboard with upload imaging and reports	Pass
6	Login as Lab Technologist	Email: lab@hospital.com, Password: lab123, Role: lab-technologist	Lab dashboard with lab results features	Lab dashboard with upload lab results	Pass
7	Login as Front Desk	Email: frontdesk@hospital.com, Password: desk123, Role: hospital-front-desk	Front desk dashboard with procedure uploads	Front desk dashboard with procedures and hospitalizations	Pass
8	Invalid role login attempt	Email: valid@test.com, Role: invalid-role	Shows invalid role error	Error message displayed	Pass
9	Unverified user login	Email: unverified@test.com, isVerified: false	Redirects to email verification	Email verification page shown	Pass
10	Hospital admin without approval	Hospital status: Pending	Shows hospital not approved error	Error message displayed	Pass

Table 29: Functional Test Table for Role-Based Login

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Submit hospital registration	Complete hospital data with documents	Application submitted successfully	Registration confirmation displayed	Pass
2	Upload hospital documents	Files: registration.pdf, license.pdf	Documents uploaded and stored	Files saved with metadata	Pass
3	Admin view pending applications	Status filter: Pending	Shows list of pending hospitals	Pending hospitals displayed in grid	Pass
4	Admin approve hospital	Hospital ID: valid, Action: Approve	Hospital status changed to Approved	Status updated, admin user created	Pass
5	Admin reject hospital	Hospital ID: valid, Action: Reject, Reason: incomplete docs	Hospital status changed to Rejected	Status updated with rejection reason	Pass
6	Hospital admin login after approval	Approved hospital admin credentials	Successful login to hospital dashboard	Hospital admin dashboard accessible	Pass
7	Search hospitals by name	Search: "City General Hospital"	Filters hospitals by name	Matching hospitals displayed	Pass
8	Filter hospitals by status	Filter: Approved	Shows only approved hospitals	Approved hospitals listed	Pass
9	View hospital details	Hospital ID: valid	Displays complete hospital information	Full hospital details with documents	Pass
10	Validate required documents	Missing required document	Shows document required error	Error displayed, submission blocked	Pass

Table 30: Functional Test Table for Hospital Registration and Approval

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result

1	Upload patient visit record	Visit data with documents	Visit record created successfully	Visit saved with auto-populated metadata	Pass
2	Upload lab results	Lab data with report PDF	Lab result created with file link	Lab result saved with reportUrl	Pass
3	Upload imaging study	Imaging data with DICOM/image files	Imaging record created	Imaging saved with imageUrl	Pass
4	Upload procedure record	Procedure data with documents	Procedure record created	Procedure saved with file attachments	Pass
5	Upload hospitalization record	Hospitalization data with discharge report	Hospitalization created	Hospitalization saved with discharge document	Pass
6	Associate lab results with visit	Visit with existing lab result IDs	Lab results linked to visit	AssociatedLabResults array populated	Pass
7	Associate imaging with procedure	Procedure with imaging reference	Imaging linked to procedure	Imaging association created	Pass
8	View patient medical timeline	Patient ID: valid	Timeline shows all medical events	Chronological display of visits, labs, imaging	Pass
9	Download medical documents	Document URL: valid	File downloads successfully	PDF/image file downloaded	Pass
10	Search patients for EHR	Search term: patient name	Finds matching patients	Patient list filtered by search	Pass

Table 31: Functional Test Table for EHR Upload and Management

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Add doctor to hospital	Doctor data with specialty	Doctor account created	Doctor added with hospital association	Pass
2	Add radiologist	Radiologist data with license	Radiologist account created	Radiologist added to hospital	Pass

3	Add lab technologist	Lab tech data with certification	Lab technologist account created	Lab tech added to hospital	Pass
4	Add front desk staff	Front desk data	Front desk account created	Front desk staff added	Pass
5	View hospital personnel	Filter: All roles	Shows all hospital staff	Personnel list with role filters	Pass
6	Filter personnel by role	Filter: doctors	Shows only doctors	Doctor personnel displayed	Pass
7	Personnel login after creation	Added personnel credentials	Successful login to role dashboard	Personnel can access appropriate dashboard	Pass
8	Validate personnel unique email	Email: existing email	Shows email already exists error	Error displayed, account not created	Pass
9	Auto-verify hospital personnel	Personnel creation	Created as verified user	isVerified set to true	Pass
10	Hospital association validation	Personnel without hospital ID	Shows hospital association required	Error displayed	Pass

Table 32: Functional Test Table for Hospital Personnel Management

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Access EHR with valid role	Role: doctor, Valid patient	EHR data displayed	Patient data accessible	Pass
2	Block unauthorized role access	Role: pharmacist accessing EHR	Access denied	Error message displayed	Pass
3	Secure file URL access	Direct file URL access	Requires authentication	Authentication check performed	Pass
4	Patient context security	Patient ID in session	Secure patient data access	Patient context validated	Pass
5	File upload role validation	Role: radiologist uploading lab results	Role permission check	Access denied for mismatched role	Pass
6	Hospital data isolation	User from Hospital A	Cannot access Hospital B data	Data properly isolated by hospitalId	Pass

7	JWT token validation	Invalid/expired token	Access denied	Token validation failed	Pass
8	Protected route access	Unauthenticated user	Redirects to login	Login page displayed	Pass
9	Role-based navigation	User role: doctor	Shows doctor-specific navigation	Doctor menu items displayed	Pass
10	Secure logout	Logout action	Clears authentication data	Token cleared, redirected to login	Pass

Table 33: Functional Test Table for Patient Data Security

7.3. Business Rules Testing Tables

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Valid User Role?	Yes	Yes	No	No	Yes	Yes	No	No
Appropriate Resource Access?	Yes	No	Yes	No	Yes	No	Yes	No
Hospital Association Required?	N/A	Yes	N/A	Yes	No	No	No	No
Grant Access	Yes	Yes	No	No	No	No	No	No
Show 'Access Denied'	No	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 34: Role-Based Access Control

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
User Has Hospital Association?	Yes	No	Yes	Yes	No	Yes
User Role Matches Upload Type?	Yes	Yes	No	Yes	Yes	No
Patient Exists?	Yes	Yes	Yes	No	Yes	Yes
Allow Upload	Yes	No	No	No	No	No
Show Permission Error	No	Yes	Yes	Yes	Yes	Yes

Table 35: Medical Record Upload Permissions

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
All Required Documents?	Yes	No	Yes	Yes	No

Valid Registration Number?	Yes	Yes	No	Yes	Yes
Unique Hospital Name?	Yes	Yes	Yes	No	Yes
Valid Admin Contact?	Yes	Yes	Yes	Yes	No
Accept Registration	Yes	No	No	No	No
Show Validation Error	No	Yes	Yes	Yes	Yes

Table 36: Hospital Registration Validation

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Valid File Type?	Yes	No	Yes	Yes	No
Within Size Limit?	Yes	Yes	No	Yes	Yes
Authenticated User?	Yes	Yes	Yes	No	Yes
Accept Upload	Yes	No	No	No	No
Show File Error	No	Yes	Yes	Yes	Yes

Table 37: File Upload Security

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Hospital Status Approved?	Yes	No	Yes	No
Valid Admin Credentials?	Yes	Yes	No	No
Allow Admin Login	Yes	No	No	No
Show Authorization Error	No	Yes	Yes	Yes

Table 38: Hospital Admin Authorization

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Patient Exists?	Yes	No	Yes	No
Valid Record Association?	Yes	Yes	No	No
Same Hospital?	Yes	Yes	Yes	Yes
Create Association	Yes	No	No	No
Show Association Error	No	Yes	Yes	Yes

Table 39: Patient Data Association

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
User Role Admin?	Yes	No	Yes	No
Hospital Application Valid?	Yes	Yes	No	No
Allow Approval/Rejection	Yes	No	No	No
Show Permission Error	No	Yes	Yes	Yes

Table 40: System Admin Hospital Management

7.4. Integration Testing Tables

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	User registration initiation	Email: newuser@test.com, Password: Secure@123, Role: doctor	Registration form accepted	Form submitted successfully	Pass
2	Email verification sent	Registration success	Verification email dispatched	Email sent with 6-digit code	Pass
3	Email verification completion	OTP: received code	User marked as verified	User isVerified set to true	Pass
4	First login after verification	Verified user credentials	Successful login and dashboard access	Role-appropriate dashboard loaded	Pass

Table 41: Integration Test Table for Complete User Registration Flow

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Hospital submits registration	Complete hospital application with documents	Application submitted to admin queue	Hospital created with Pending status	Pass
2	Admin views application	Admin dashboard access	Pending application visible	Hospital listed in pending applications	Pass

3	Admin reviews documents	Hospital detail view	All documents accessible	Documents display and download correctly	Pass
4	Admin approves hospital	Approval action	Hospital status updated, admin user created	Status: Approved, HospitalAdmin user created	Pass
5	Hospital admin first login	Approved hospital admin credentials	Successful login to hospital dashboard	Hospital admin dashboard accessible	Pass

Table 42: Integration Test Table for Hospital Registration to Approval Workflow

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Doctor uploads visit record	Visit data with patient ID	Visit created in database	Visit saved with auto-populated metadata	Pass
2	Lab tech uploads lab results	Lab results for same patient	Lab result created	Lab result linked to patient	Pass
3	Doctor associates lab with visit	Existing visit and lab result	Association created	Visit.associatedLabResults updated	Pass
4	View patient timeline	Patient with multiple records	Chronological timeline displayed	Timeline shows visit, lab results chronologically	Pass
5	Download associated documents	Visit with lab result documents	Files download successfully	Both visit and lab documents accessible	Pass

Table 43: Integration Test Table for Medical Data Upload and Retrieval

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Doctor accesses patient EHR	Valid doctor credentials	EHR data displayed	Complete patient records shown	Pass
2	Radiologist uploads imaging	Patient with existing visit	Imaging record created	Imaging linked to patient and hospital	Pass

3	Lab tech accesses only lab features	Lab technologist role	Limited dashboard access	Only lab-related navigation shown	Pass
4	Cross-hospital data isolation	Doctor from Hospital A	Cannot access Hospital B data	Data properly filtered by hospitalId	Pass
5	Unauthorized role access attempt	Pharmacist accessing EHR upload	Access denied	Permission error displayed	Pass

Table 44: Integration Test Table for Role-Based Data Access

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Upload visit document	PDF file with visit record	File stored and linked	File saved in uploads directory	Pass
2	Upload multiple imaging files	DICOM and JPEG files	All files processed	Multiple files stored with correct paths	Pass
3	Associate uploaded files	Files with medical record	File URLs stored in database	Document URLs saved in record	Pass
4	Retrieve and display files	Medical record with files	Files accessible via URLs	Files download/display correctly	Pass
5	File security validation	Direct URL access attempt	Authentication required	File access requires valid session	Pass

Table 45: Integration Test Table for File Upload and Storage

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Hospital admin adds doctor	Complete doctor information	Doctor account created	Doctor user created with hospital association	Pass
2	Added doctor first login	Doctor credentials	Successful login	Doctor dashboard accessible	Pass
3	Doctor uploads patient visit	Visit data	Visit created with doctor metadata	Visit.uploadedBy set to doctor ID	Pass

4	View hospital personnel	Hospital admin dashboard	All personnel listed	Personnel displayed with role filters	Pass
5	Personnel role-based access	Different personnel roles	Appropriate dashboard access	Each role sees correct navigation and features	Pass

Table 46: Integration Test Table for Hospital Personnel Management

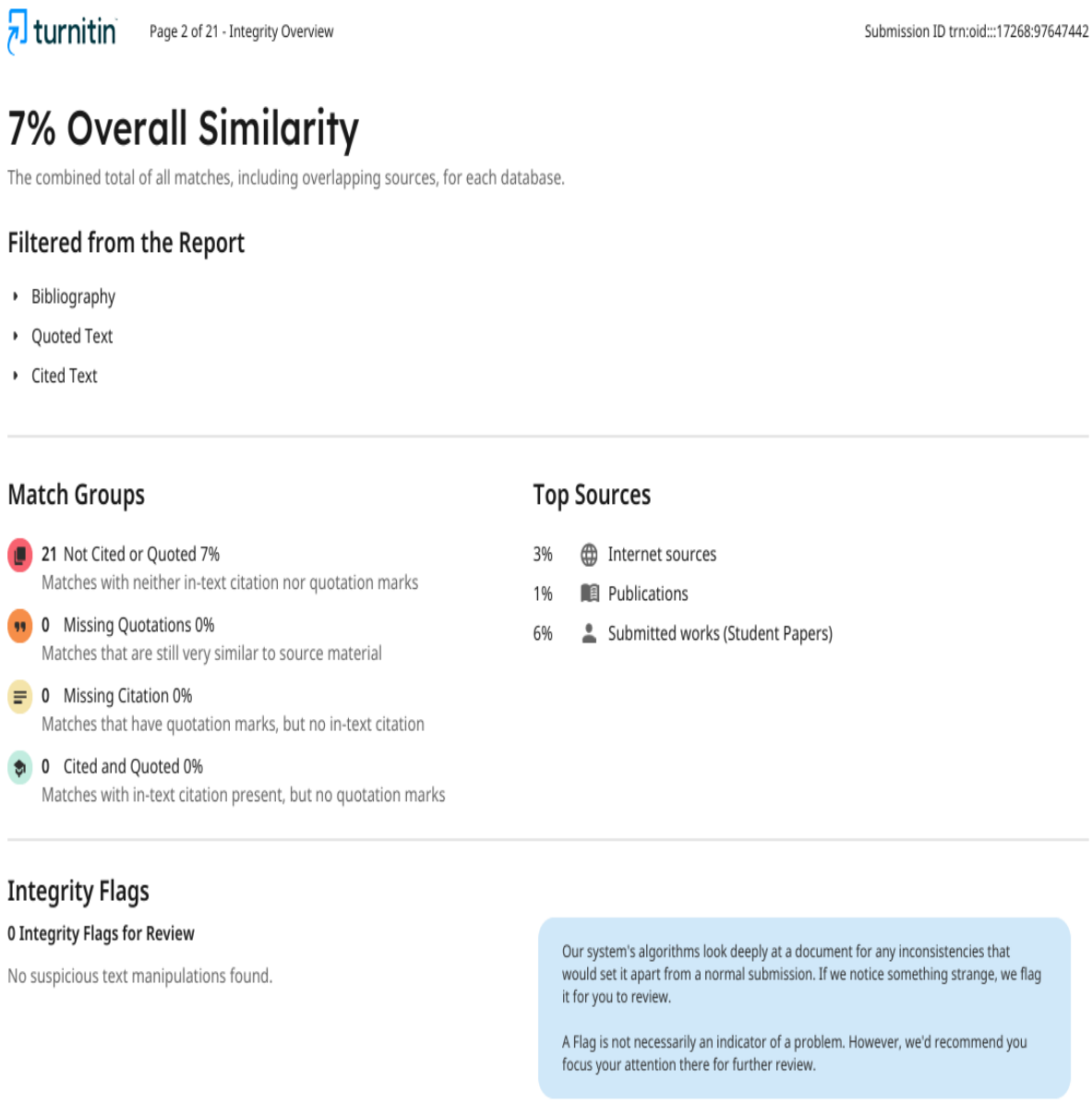
No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Create patient record	Basic patient information	Patient created in system	Patient saved with unique ID	Pass
2	Upload visit for patient	Visit data with patient ID	Visit linked to patient	Visit.patientId references patient	Pass
3	Upload lab results	Lab data for same patient	Lab results linked	LabResult.patientId references patient	Pass
4	Upload imaging study	Imaging data for patient	Imaging linked	Imaging.patientId references patient	Pass
5	View comprehensive timeline	Patient with all data types	Complete medical timeline	Chronological display of all medical events	Pass
6	Filter timeline by type	Timeline with type filter	Filtered view displayed	Timeline shows only selected event types	Pass

Table 47: Integration Test Table for Patient Data Timeline

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	User login with valid credentials	Email, password, role	JWT token generated	Token created and cookie set	Pass
2	Access protected route	Valid JWT token	Route access granted	Protected content displayed	Pass
3	Token expiration handling	Expired token	Access denied, redirect to login	Login page displayed	Pass
4	Role-based route protection	Wrong role for route	Access denied	Permission error or redirect	Pass
5	Logout token invalidation	Active session logout	Token cleared	Authentication state cleared	Pass

Table 48: Integration Test Table for Authentication and Authorization Chain

8. Plagarism Report



turnitin Page 2 of 21 - Integrity Overview Submission ID trn:oid:::17268:97647442

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

Match Groups

21 Not Cited or Quoted 7%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

3% Internet sources
1% Publications
6% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.