SRS of IoT-AI Powered Healthcare Network System iOS Application

iOS App Team

1. HARSHIT KUMAR
2. CHETAN
3. PRIYANSHU AHLAWAT
4. ATUL TYAGI
5. YASHIKA RATHI

Project Coordinator

Dr. BRIJESH KUMAR GUPTA
Mr. SURAJ KUMAR BHATNAGAR

*Purpose

The purpose of this document is to define the functional and non-functional requirements for the **IoT-AI Powered Healthcare Network System iOS Application**. This application serves as the primary user interface for patients, caregivers, and medical professionals to interact with a centralized healthcare network. By leveraging real-time data from Internet of Things (IoT) devices and applying Artificial Intelligence (AI) for analysis, the application aims to facilitate a shift from reactive to **proactive, data-driven healthcare**. This document will guide the development, testing, and deployment phases, ensuring the final product meets all specified requirements for performance, security, and usability.

---

*Product Scope

The scope of this project is to develop a native iOS application that seamlessly integrates with the larger IoT-AI healthcare network. The application will be a comprehensive personal health management tool, enabling users to:

- **Connect and sync** with various Bluetooth-enabled medical IoT devices (e.g., smartwatches, blood pressure monitors, glucometers).
- **Visualize** real-time and historical health data through intuitive dashboards and graphs.
- **Receive AI-powered insights** and predictive alerts based on their health data.
- **Engage in secure, HIPAA-compliant communication** with their healthcare providers.

The application will not include the development of the physical IoT devices or the AI models themselves, which are part of the back-end system. Its core function is to act as a secure, user-friendly front-end that facilitates the flow of data and insights.

---

*Definitions, Acronyms, and Abbreviations

- **IoT (Internet of Things):** A network of interconnected devices that collect and exchange data, such as wearable health sensors and medical monitoring devices.
- **API (Application Programming Interface):** A set of protocols and tools that allow the app to communicate with external systems or devices.
- **HIPAA (Health Insurance Portability and Accountability Act):** A U.S. regulation that ensures the security and privacy of health data.
- **BLE (Bluetooth Low Energy):** A wireless communication technology used to connect the iOS app with IoT-enabled healthcare devices.
- **DFD (Data Flow Diagram):** A graphical representation of how data moves through the system.
- **UI (User Interface):** The visual part of the app with which users interact.
- **UX (User Experience):** The overall experience of the user while using the app.

- **SDK (Software Development Kit):** A collection of software tools for building applications on iOS.
- **FHIR (Fast Healthcare Interoperability Resources):** A standard for exchanging healthcare information electronically.
- **Cloud Storage:** Remote servers where patient health data is stored securely.

---

*References

➔ Apple Developer Documentation – https://developer.apple.com/documentation

➔ IEEE 830-1998 – Recommended Practice for Software Requirements Specifications.

➔ HIPAA Privacy and Security Rules – https://www.hhs.gov/hipaa

➔ FHIR Standard by HL7 – https://www.hl7.org/fhir/

➔ Bluetooth Low Energy (BLE) Specification – Bluetooth SIG.

➔ Research Papers on IoT in Healthcare:

➔ "IoT in Healthcare: Applications, Benefits, and Challenges" – IEEE Access.

➔ "Wearable IoT Sensors for Health Monitoring" – Elsevier Journal.

➔ Apple Human Interface Guidelines (HIG) for iOS – https://developer.apple.com/design/human-interface-guidelines

*Overview

This document specifies the requirements for the **IoT-powered Healthcare iOS Application**. It provides a complete description of the system, including its objectives, scope, functionalities, interfaces, constraints, and design considerations.

The document is organized as follows:

- **Section 1:** Introduction (purpose, scope, definitions, references).
- **Section 2:** Overall description of the application, including product perspective, user types, operating environment, and constraints.
- **Section 3:** System features, each described in detail with inputs, outputs, and conditions.

- **Section 4:** External interface requirements for hardware, software, and communication.
- **Section 5:** Non-functional requirements such as performance, security, scalability, and usability.
- **Section 6:** System models including UML diagrams (use case, data flow, sequence).
- **Section 7:** Other requirements such as compliance standards and data management.

---

*Main Protocols Used

1. Device-to-App Protocols

These protocols enable the iOS application to directly communicate with nearby IoT health devices.

- **Bluetooth Low Energy (BLE)**: This is the most common protocol for connecting wearable devices (like smartwatches and continuous glucose monitors) to an iPhone. BLE is ideal for this purpose because it's **energy-efficient**, crucial for battery-powered health sensors, and designed for short-range data transfer.
- **Wi-Fi**: For devices that require higher bandwidth or a more stable connection, such as smart scales or some home health hubs, Wi-Fi is used. It's suitable for transmitting larger data sets but consumes more power than BLE.

2. App-to-Cloud Protocols

These protocols handle the secure transmission of data from the iOS application to the central cloud server where the AI analysis takes place.

- **HTTPS (Hypertext Transfer Protocol Secure)**: This is the standard for secure communication over the internet. By using an **SSL/TLS encryption** layer, HTTPS ensures that all patient data transmitted from the app to the cloud server is encrypted, protecting it from eavesdropping and man-in-the-middle attacks. It is a fundamental protocol for maintaining HIPAA or GDPR compliance.
- **MQTT (Message Queuing Telemetry Transport)**: A lightweight messaging protocol, MQTT is highly efficient for sending small data packets from the app to the cloud. It's often used in resource-constrained environments and is well-suited for the rapid, real-time transmission of sensor data, such as heart rate or blood oxygen levels. It operates on a **publish-subscribe model**, which allows for a high volume of data to be sent efficiently.

3. Data Integration and Interoperability Protocols

These protocols are used to ensure that the healthcare data can be understood and exchanged between different healthcare information systems, such as Electronic Health Records (EHRs).

- **Health Level Seven (HL7)**: A set of international standards for the transfer of clinical and administrative data between software applications used in healthcare settings. While not a direct

communication protocol for the app, it's essential for the system's back-end to ensure the data collected by the IoT devices can be properly integrated into a patient's medical record.

- **Fast Healthcare Interoperability Resources (FHIR)**: A modern standard that uses web technologies to make healthcare data more accessible and interoperable. FHIR is a more flexible and developer-friendly alternative to HL7, allowing for seamless data exchange between the iOS application's back-end and hospital information system.

*Overall Description

1.Product Perspective

The system is an iOS-based mobile application that integrates with IoT-enabled healthcare devices (e.g., wearables, sensors, Apple Watch) to continuously collect, process, and transmit patient health data. The app connects to a backend server that stores and analyzes the data, ensuring accessibility for patients, doctors, and administrators. The product leverages Apple HealthKit and Bluetooth Low Energy (BLE) for seamless device integration and uses secure APIs for communication with external services.

2.Product Functions

Key functions of the application include:

1. Remote Health Monitoring: Continuous tracking of vital signs (e.g., heart rate, blood pressure, oxygen levels) via IoT devices.

2. Data Collection & Visualization: Aggregating health data in real-time and displaying it through interactive charts and dashboards.

3. Alerts & Notifications: Sending push notifications for abnormal readings, medication reminders, and doctor alerts.

4. Consultation Booking: Enabling patients to schedule virtual or in-person consultations with doctors directly through the app.

5. Reports & Analytics: Generating medical history reports for patients and decision-support insights for doctors.

6. Secure Data Sharing: Allowing patients to share selected data with healthcare providers or caregivers.

---

*User Classes & Characteristics

- **Patients:** Primary users who monitor health data, receive alerts, and book consultations. Typically non-technical, requiring a simple and intuitive interface**.**
- **Doctors:** Healthcare professionals who review patient data, provide consultations, and prescribe treatments. Require advanced data visualization and analytics tools.
- **Administrators:** Manage system operations, user accounts, security policies, and compliance with healthcare standards. Require access to monitoring and audit features**.**
- **Caregivers:** Secondary users (family or caretakers) who can monitor patient health data and receive emergency alerts with limited permissions.

---

*Operating Environment

- **Mobile OS:** iOS 15.0 and above.
- **Devices Supported:** iPhone 11 and newer models, Apple Watch (Series 5 and above).
- **IoT Devices:** Bluetooth-enabled medical sensors (BP monitors, glucose monitors, pulse oximeters, ECG devices).
- **Backend Environment:** Cloud-based server infrastructure (AWS/Azure/Google Cloud) supporting RESTful APIs and database management.
- **Network Requirements:** Stable internet connectivity (Wi-Fi/4G/5G).

---

*Tools Used

For developing an iOS mobile application for an affordable and highly available IoT-AI powered healthcare network system, a multifaceted toolkit is required, spanning from front-end mobile development to back-end infrastructure and data processing.[1] The selection of tools should prioritize security, scalability, and cost-effectiveness to meet the project's core requirements.

Core Development and Design

The foundation of the project lies in the iOS application itself. Development will primarily revolve around Apple's native ecosystem to ensure optimal performance and user experience.

- ➔ **Programming Languages: Swift** is the modern, powerful, and intuitive language of choice for iOS development.
    - Its safety features and concise syntax make it ideal for building robust healthcare

applications.
    - While **Objective-C** is the legacy language, it might be necessary for maintaining or integrating with older libraries.

➔ **Integrated Development Environment (IDE): Xcode** is the essential IDE for creating iOS applications.
    - It provides a complete suite of tools for coding, debugging, and performance testing.
    - Its built-in interface builder and simulators are indispensable for a smooth development workflow.

➔ **UI/UX Design:** To create an intuitive and user-friendly interface, which is critical for a healthcare application, designers will likely use tools like **Figma**, **Sketch**, or **Adobe XD**.
    - These platforms allow for the creation of wireframes, mockups, and interactive prototypes, facilitating a collaborative design process.

IoT Integration and Management

Connecting and managing IoT devices is a cornerstone of this project. The chosen tools must be able to handle data from various medical sensors and devices reliably.

- **IoT Platforms:** Cloud-based IoT platforms are crucial for managing device connectivity, data ingestion, and security. Leading options that are also HIPAA-compliant include **AWS IoT Core**, **Google Cloud IoT Core**, and **Microsoft Azure IoT Hub**. These platforms provide SDKs that can be integrated into the iOS application for seamless communication with IoT devices.
- **Communication Protocols:** The application will need to support various communication protocols to interact with a wide range of medical IoT devices. This includes **Bluetooth Low Energy (BLE)** for short-range communication with wearable sensors, as well as **MQTT** and **CoAP** for efficient messaging between devices and the cloud, especially in low-power or constrained network environments.

Artificial Intelligence and Machine Learning

The "AI-powered" aspect of the project necessitates tools for developing and deploying machine learning models to analyze healthcare data and provide valuable insights.

- **Machine Learning Frameworks:** For on-device intelligence, which can enhance privacy and reduce latency, **Core ML** is Apple's native framework for integrating trained machine learning models into iOS apps.For more complex, cloud-based AI processing, popular frameworks like **TensorFlow** and **PyTorch** are the standards. These are often used in conjunction with cloud services.
- **AI Cloud Services:** Cloud platforms offer powerful and scalable AI and machine learning services. **Amazon SageMaker**, **Google AI Platform**, and **Azure Machine Learning** provide comprehensive environments for building, training, and deploying models that can handle large datasets typical in healthcare.

Backend and Database Infrastructure

A secure, scalable, and highly available backend is critical to support the mobile application and the entire healthcare network. Affordability is also a key consideration in selecting these services.

- **Backend Development:** For building the server-side logic and APIs, popular and well-supported frameworks are recommended. **Node.js** (with frameworks like Express.js) is known for its performance and is suitable for real-time applications.
  - ➔ **Python** (with frameworks like Django or Flask) is another excellent choice, especially given its strong ecosystem for data science and AI.
- **Database Solutions:** The choice of database will depend on the nature of the data. For structured data, a relational database like **PostgreSQL** or **MySQL** is a solid and affordable option.
  - ➔ For the vast and often unstructured data generated by IoT devices, a NoSQL database such as **MongoDB** or **Firebase Realtime Database** is more appropriate.
  - ➔ Firebase, in particular, offers a suite of tools that can accelerate development and is known for its real-time data synchronization capabilities.
- **Cloud Hosting and High Availability:** To ensure the system is "highly available," deploying the backend on a major cloud provider is essential. **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)**, and **Microsoft Azure** all offer a range of services designed for high availability and scalability, including load balancing, auto-scaling, and multi-region deployments. They also provide HIPAA-compliant services, which is a legal requirement for handling protected health information (PHI) in the United States. These platforms offer pay-as-you-go pricing models, which can contribute to the affordability of the solution.

By carefully selecting and integrating these tools, a development team can build a robust, scalable, and affordable iOS-based healthcare network system that effectively leverages the power of IoT and AI.

---

* Design & Implementation Constraints

Technology Constraints:

- Must adhere to Apple's iOS Human Interface Guidelines (HIG).
- Implementation using Swift and SwiftUI.
- Integration with Apple HealthKit for health data management.
- Regulatory & Compliance Constraints:
- Data privacy and security must comply with HIPAA (U.S.) and GDPR (EU).
- Encrypted storage and secure transmission of patient data.
- Resource Constraints: Limited to iOS ecosystem; Android support not in scope.

Assumptions & Dependencies

- The app assumes continuous internet availability for real-time data synchronization.
- IoT devices used must be compatible with iOS and BLE standards.
- Functionality depends on third-party APIs (e.g., telemedicine platforms, payment gateways, cloud services).
- Apple App Store approval is required for deployment.
- Assumes users (patients/doctors) have the necessary IoT devices and iOS hardware for compatibility.
- The system is an iOS-based mobile application that integrates with IoT-enabled healthcare devices

(e.g., wearables, sensors, Apple Watch) to continuously collect, process, and transmit patient health data.
- The app connects to a backend server that stores and analyzes the data, ensuring accessibility for patients, doctors, and administrators. The product leverages Apple HealthKit and Bluetooth Low Energy (BLE) for seamless device integration and uses secure APIs for communication with external services.

* Product Functions

Key functions of the application include:
- Remote Health Monitoring: Continuous tracking of vital signs (e.g., heart rate, blood pressure, oxygen levels) via IoT devices.
- Data Collection & Visualization: Aggregating health data in real-time and displaying it through interactive charts and dashboards.
- Alerts & Notifications: Sending push notifications for abnormal readings, medication reminders, and doctor alerts.
- Consultation Booking: Enabling patients to schedule virtual or in-person consultations with doctors directly through the app.
- Reports & Analytics: Generating medical history reports for patients and decision-support insights for doctors.
- Secure Data Sharing: Allowing patients to share selected data with healthcare providers or caregivers.

---

* Design & Implementation Constraints

➔ Technology Constraints:

- ◆ Must adhere to Apple's iOS Human Interface Guidelines (HIG).
- ◆ Implementation using Swift and SwiftUI.
- ◆ Integration with Apple HealthKit for health data management.

➔ Regulatory & Compliance Constraints:

- ◆ Data privacy and security must comply with HIPAA (U.S.) and GDPR (EU).
- ◆ Encrypted storage and secure transmission of patient data.

➔ Resource Constraints: Limited to iOS ecosystem; Android support not in scope.

---

* Assumptions & Dependencies

- The app assumes continuous internet availability for real-time data synchronization.
- IoT devices used must be compatible with iOS and BLE standards.

- Functionality depends on third-party APIs (e.g., telemedicine platforms, payment gateways, cloud services).
- Apple App Store approval is required for deployment.
- Assumes users (patients/doctors) have the necessary IoT devices and iOS hardware for compatibility.

---

* System Features

Patient Registration & Login

- **Description:** Enables patients to register and log into the system using Apple ID, email, or mobile number. Provides authentication and account management.
- **Input:** User credentials (Apple ID/email/password/OTP).
- **Output:** Successful login session or error messages.
- **Preconditions:** User must have a valid Apple ID or registered account.
- **Postconditions:** Patient is authenticated and redirected to the dashboard.
- **Priority: High** – required for access control and system security.

IoT Device Integration

- **Description:** Allows connection of IoT-enabled healthcare devices (e.g., smartwatch, glucose monitor, BP monitor) via Bluetooth Low Energy (BLE) or Apple HealthKit.
- **Input:** Device pairing request and health data streams.
- **Output:** Confirmation of device connection and synchronized health data.
- **Preconditions:** Device must be compatible with iOS and BLE/HealthKit.
- **Postconditions:** Health data is automatically collected and available in the app.
- **Priority: High** – core functionality of IoT healthcare system.

Real-Time Health Monitoring

- **Description:** Collects patient vitals (heart rate, blood pressure, glucose levels, oxygen saturation, ECG, etc.) from connected IoT devices and displays them on the dashboard.
- **Input:** Continuous health data from IoT devices.
- **Output:** Real-time charts, notifications, and alerts for abnormal readings.
- **Preconditions:** IoT device must be paired and active.
- **Postconditions:** Patient vitals are stored securely in the backend database.

- **Priority: High** – essential for healthcare monitoring.

Doctor Consultation & Prescription

- **Description:** Enables patients to book consultations (virtual/in-person) with doctors and receive digital prescriptions.
- **Input:** Patient consultation request, symptoms, preferred schedule.
- **Output:** Appointment confirmation, doctor's notes, and e-prescription.
- **Preconditions:** Patient must be registered; doctor must have an active profile.
- **Postconditions:** Consultation details stored in patient records; notifications sent.
- **Priority: Medium-High** – enhances healthcare delivery.

Notifications & Alerts

- **Description:** Provides push notifications for critical health alerts, emergency warnings, medication reminders, and appointment reminders.
- **Input:** Trigger events (abnormal health reading, scheduled medication, upcoming consultation).
- **Output:** Push notifications to patient, caregiver, or doctor.
- **Preconditions:** User must have app installed with notifications enabled.
- **Postconditions:** Notification is delivered, and user action (acknowledge, dismiss, respond) is logged.
- **Priority: High** – ensures timely intervention.

Medical Data Storage

- **Description:** Stores patient health data securely in cloud servers with backup and recovery support.
- **Input:** Health data from IoT devices, consultation records, prescriptions.
- **Output:** Securely stored and retrievable medical history.
- **Preconditions:** Cloud service and database connectivity must be available.
- **Postconditions:** Data is encrypted, securely stored, and accessible to authorized users only.
- **Priority: High** – required for compliance and continuous care.

Reports & Analytics

- **Description:** Generates health trend reports, analytics dashboards, and progress insights for

patients and doctors.

- **Input:** Stored patient health data over time.
- **Output:** Graphical charts, PDF reports, insights (e.g., weekly glucose trends).
- **Preconditions:** Sufficient health data must be available in the system.
- **Postconditions:** Reports generated, downloadable, and shareable with doctors/caregivers.
- **Priority: Medium** – supports better medical decisions.

*External Interface Requirements

User Interfaces

➔ The application will follow **Apple's Human Interface Guidelines (HIG)** to ensure a smooth iOS-native experience.

➔ **Onboarding Flow:** Simple account creation, device pairing tutorial, and permission setup (Bluetooth, notifications, HealthKit).

➔ **Dashboards:**

- ◆ Patient Dashboard: Vital stats, alerts, consultation booking, medical history.
- ◆ Doctor Dashboard: Patient list, vitals monitoring, consultation management.

➔ **Alerts & Notifications:** Push alerts for abnormal vitals, reminders, and emergency situations.

➔ **Accessibility:** Support for VoiceOver, Dynamic Type (font scaling), and color contrast compliance.

Hardware Interfaces

- **IoT Devices:** Medical sensors (BP monitors, glucose monitors, pulse oximeters, ECG devices) that connect via BLE.
- **Apple Watch:** Continuous health monitoring (heart rate, SpO2, ECG).
- **iPhone Sensors:** Optional integration with motion sensors (for activity tracking).

Software Interfaces

- **APIs:** RESTful APIs for backend communication (patient data, appointments, reports).
- **Apple HealthKit:** For secure storage and retrieval of health data.
- **Firebase/AWS:** For authentication, push notifications, and cloud storage.
- **Payment Gateway API (Optional):** For online consultation payments.

Communication Interfaces

- **Wi-Fi & Cellular (4G/5G):** Required for real-time data sync and consultations.
- **Bluetooth (BLE):** For IoT device connectivity.
- **Push Notifications (APNs):** For delivering alerts, reminders, and emergency messages.

---

* Functional Requirements

Patient Registration & Authentication

- **Description:** Patients can register/login using Apple ID, email, or phone.
- **Inputs:** User credentials, Apple ID.
- **Outputs:** User account creation, access granted.
- **Precondition:** Internet connection required.
- **Postcondition:** Patient logged in with personalized dashboard.

IoT Device Integration

- **Description:** Connect IoT devices (e.g., smartwatch, BP monitor, glucose tracker) via Bluetooth or Wi-Fi.
- **Inputs:** Device pairing request.
- **Outputs:** Device linked to patient profile.
- **Precondition:** Device compatibility with iOS.
- **Postcondition:** Continuous health data collection.

Real-Time Health Monitoring

- **Description:** Collect and display health parameters (heart rate, BP, glucose, oxygen level).
- **Inputs:** Sensor readings from IoT devices.
- **Outputs:** Real-time graphs, alerts, and logs.
- **Precondition:** IoT device must be active.
- **Postcondition:** Health data stored in cloud & accessible in app.

Alerts & Notifications

- **Description:** Generate alerts for abnormal readings, medication reminders, and emergency warnings.
- **Inputs:** Threshold breach in health data, scheduled reminders.
- **Outputs:** Push notifications, sound/vibration alerts.
- **Precondition:** Notifications enabled.
- **Postcondition:** Patient notified; alert recorded in history.

Doctor Consultation & Appointment Scheduling

- **Description:** Patients can book appointments with healthcare providers.
- **Inputs:** Date, time, doctor selection.
- **Outputs:** Appointment confirmation, calendar update.
- **Precondition:** Patient registered; doctor available.
- **Postcondition:** Appointment saved in both patient & provider schedules.

Prescription & Medical Records Management

- **Description:** Doctors can upload prescriptions and patients can view/store them.
- **Inputs:** Prescription details, test results.
- **Outputs:** Secure storage of records, patient access.
- **Precondition:** Doctor’s approval required.
- **Postcondition:** Prescription added to patient profile.

Reports & Analytics

- **Description:** Generate weekly/monthly health reports with trends.
- **Inputs:** Stored health data.
- **Outputs:** Graphs, PDF/CSV reports.
- **Precondition:** Sufficient data available.
- **Postcondition:** Report generated and downloadable.

Admin Dashboard

- **Description:** Admin can manage users, doctors, devices, and system logs.

- **Inputs:** Admin credentials.
- **Outputs:** User/device status updates, analytics.
- **Precondition:** Admin authentication.
- **Postcondition:** Data updated in backend.

---

*Non-Functional Requirements

Performance Requirements

- App load time should not exceed **3 seconds** on supported iOS devices.
- Real-time updates for IoT data should have a **latency < 2 seconds**.

Security & Privacy

- End-to-end encryption (AES-256) for data transmission and storage.
- Multi-factor authentication for doctors and admins.
- HIPAA and GDPR compliance for handling health records.

Reliability & Availability

- System should achieve **99.9% uptime**.
- Automatic failover and fault recovery for backend services.

Usability

- Simple, intuitive UI for non-technical users (patients, elderly).
- Accessibility compliance with iOS VoiceOver, font scaling, and colorblind-friendly themes.

Scalability

- The system should scale to support **thousands of patients and doctors simultaneously**.
- Cloud-based architecture should handle peak consultation loads (e.g., during health crises).

---

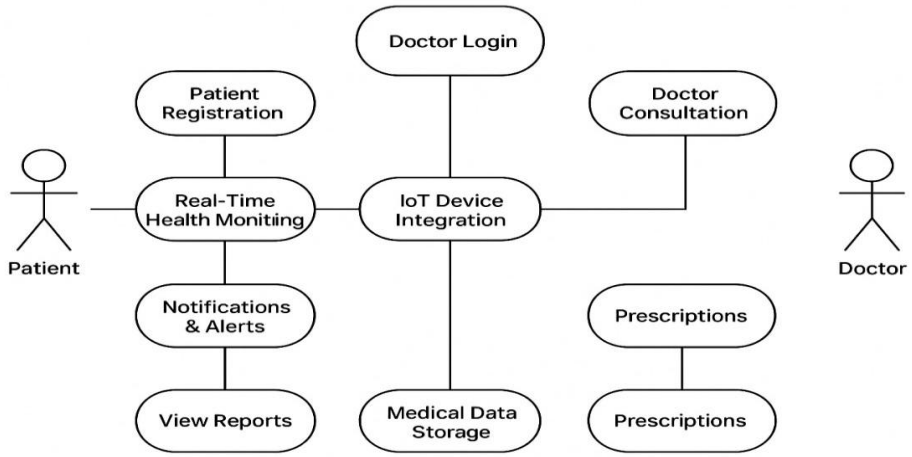
*System Models

Use Case Diagrams

Actors

- **Patient**: The primary user of the application who monitors their health, connects devices, and communicates with healthcare providers.
- **Healthcare Provider**: A doctor, nurse, or medical staff who accesses patient data, provides consultations, and manages treatment plans.
- **System Admin**: A user who manages the backend, user accounts, and system maintenance.
- **IoT Device**: An external actor (non-human) that generates and transmits health data to the application.

Use Case Diagram



Use Cases

Patient Use Cases

- **Register/Login**: A patient can create a new account or log in to an existing one.
- **Connect IoT Device**: A patient can pair their personal health devices (e.g., smartwatches, blood pressure monitors) with the app.
- **View Health Data**: The patient can view real-time and historical health data on a dashboard. This includes sub-use cases like:
  - View Vital Signs
  - View Activity Metrics
  - View Medication Adherence
- **Receive Alerts**: The patient receives push notifications and in-app alerts based on AI analysis of their data (e.g., low blood sugar alert).
- **Communicate with Provider**: The patient can securely message or initiate a video call with their healthcare provider.
- **Update Profile**: The patient can manage their personal information and health preferences.

Healthcare Provider Use Cases

- ➔ **Login**: The provider logs in to their secure portal.
- ➔ **View Patient Data**: The provider can access a patient's comprehensive health data dashboard, including data from all connected IoT devices. This includes sub-use cases like:
  - ◆ View Patient Vital Signs
  - ◆ View Health Trends
- ➔ **Send Messages/Consult**: The provider can send secure messages or conduct a virtual consultation with a patient.
- ➔ **Generate Report**: The provider can generate a summary report of a patient's health data for a specific period.
- ➔ **Set Alerts**: The provider can set custom alert thresholds for a patient's vital signs.

System Admin Use Cases

- ➔ **Manage User Accounts**: The admin can create, modify, or deactivate patient and provider accounts.
- ➔ **Monitor System Status**: The admin can view the overall health and performance of the network.
- ➔ **Update System**: The admin can deploy new features or bug fixes to the application and backend.

IoT Device Use Cases

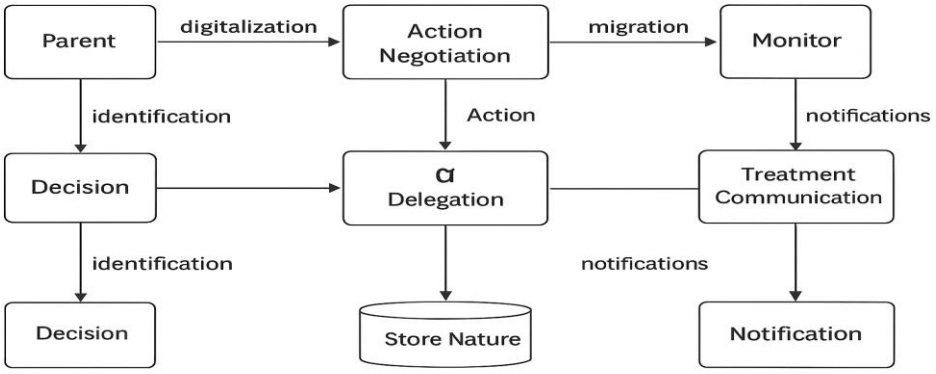
- ● **Transmit Data**: The IoT device automatically sends raw health data to the iOS application. This is a crucial, non-user-initiated use case that underpins the entire system.
- ● **Sync Data**: The device syncs any stored data with the application when a connection is established.

Data Flow Diagrams (DFD)

A data flow diagram (DFD) for the IoT-AI Powered Healthcare Network System provides a high-level view of how data moves through the system. It illustrates the processes that transform data, the external entities that interact with the system, and the data stores where information is held.

Here is a DFD illustrating the key processes and data flows.

Data Flow Diagram



IOT 21 Powereed Nobis; Reformer Nature & System

Explanation of the Data Flow

The diagram above details the journey of health data from its point of origin (an IoT device) to its final destination (a user's view and a healthcare provider's analysis).

1. Data Collection & Transmission

- **Data Flow**: Health Data
- **From**: **IoT Device** (External Entity)
- **To**: **iOS Application** (Process)
- **Description**: An IoT device (e.g., a smartwatch, a smart scale) continuously or periodically collects raw health data (e.g., heart rate, blood pressure, weight) from the patient. This data is transmitted to the iOS application, typically via Bluetooth Low Energy (BLE).

2. Data Processing & Forwarding

- **Data Flow**: Aggregated Health Data
- **From**: **iOS Application** (Process)
- **To**: **Backend Server** (Process)
- **Description**: The iOS application receives raw data from one or more IoT devices. It may perform a minor level of initial processing or aggregation before sending the data securely to the Backend Server. This data is often timestamped and tagged with the user's ID.

3. AI Analysis & Storage

- **Data Flow**: Processed Data & Insights
- **From**: **Backend Server** (Process)
- **To**: **Insights & Alerts Data Store** (Data Store)
- **Description**: The Backend Server receives the aggregated data. It stores the raw data for archival purposes and feeds it into the **AI Analytics Engine** (a subprocess within the server). The AI analyzes the data to detect anomalies, identify trends, and generate insights. These insights

and any triggered alerts are then stored in a separate data store, ready to be sent back to the app.

4. Real-time Feedback & Alerts

- **Data Flow**: Alerts & Recommendations
- **From**: **Backend Server** (Process)
- **To**: **iOS Application** (Process)
- **Description**: The Backend Server sends back the AI-generated alerts and recommendations to the iOS application. This happens in real time for critical alerts or as part of a scheduled data sync for general insights.

5. User Interaction & Display

- **Data Flow**: User-Viewable Data
- **From**: **iOS Application** (Process)
- **To**: **Patient** (External Entity)
- **Description**: The iOS application presents the patient with a user-friendly view of their health data, trends, and any alerts received from the server. The patient can also use the app to initiate communication with a healthcare provider.

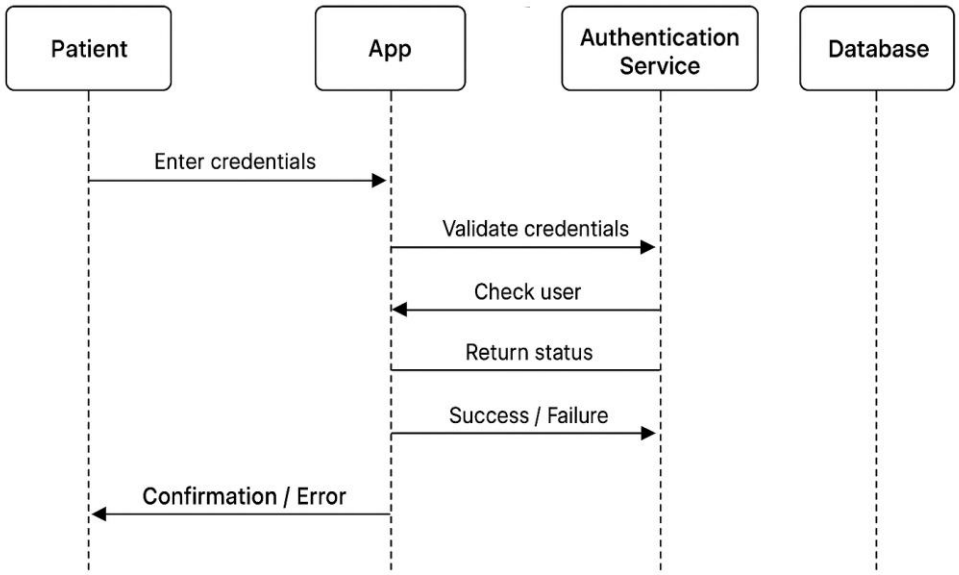
6. Provider Access & Management

- **Data Flow**: Patient Health Records
-
- **From**: **Backend Server** (Process)
- **To**: **Healthcare Provider** (External Entity)
- **Description**: Healthcare Providers access the same backend data via a secure portal (which could be a separate application or a dedicated section of the iOS app). They can view a patient's complete health history, analyze trends, and make informed decisions.

Sequence Diagrams

A sequence diagram for the **IoT-AI Powered Healthcare Network System iOS Application** illustrates the dynamic interactions between the key system components over time. It provides a step-by-step visual walkthrough of how data flows from the IoT device, through the mobile app, to the backend server, and back to the user.

Sequence Diagram



Here is a sequence diagram illustrating the process of a patient's health data being collected and analyzed, and an alert being generated.

Breakdown of the Data Flow Sequence

1. **Start Data Collection**: The sequence begins when a user wears or uses a connected **IoT Device**. The device, which is always monitoring, detects a change in a health metric (e.g., an increase in heart rate or a glucose reading).
2. **Transmit Data to App**: The IoT Device securely transmits the raw `Health Data` to the **iOS Application** via a protocol like Bluetooth Low Energy (BLE).
3. **Receive and Process Data**: The iOS Application receives the raw data. It validates and aggregates this data with the user's information. It then sends an `Aggregated Data` message to the **Backend Server**.
4. **Forward Data to Backend**: The iOS Application sends the `Aggregated Data` to the Backend Server using a secure protocol like HTTPS.
5. **Analyze Data**: The Backend Server receives the data and stores it in the `Database`. It then triggers the **AI Analytics Engine** to process the new data point.
6. **Generate Insight/Alert**: The AI Analytics Engine analyzes the data against predefined rules and historical trends. In this scenario, it detects a value that exceeds a set threshold and generates an `Alert` message.
7. **Store Alert**: The Backend Server stores the newly generated `Alert` in the `Alert Data Store` for future reference and persistence.
8. **Send Alert to App**: The Backend Server sends the `Alert` back to the iOS Application, notifying it of the critical event.
9. **Receive and Display Alert**: The iOS Application receives the `Alert` message. It then

displays a push notification or an in-app message to the **Patient**, warning them of the health issue. This completes the loop, providing real-time feedback to the user.

This sequence diagram is crucial for understanding the **order and timing** of events and ensures that developers and other team members have a clear, shared understanding of the system's dynamic behavior. It highlights the critical path for a data-to-alert process, which is a core function of the application.

Example A: Patient Registration & Login

Actors: Patient, App, Authentication Service, Database
Flow:

1. Patient → App: Enter credentials (Apple ID/email/password)
2. App → Authentication Service: Validate credentials
3. Authentication Service → Database: Check user
4. Database → Authentication Service: Return status
5. Authentication Service → App: Success/Failure
6. App → Patient: Confirmation / Error

Example B: IoT Device Data Sync

Actors: IoT Device, App, Health Monitoring Module, Cloud DB
Flow:

1. IoT Device → App: Send health readings (BP, HR, glucose)
2. App → Health Monitoring Module: Validate/format data
3. Health Monitoring Module → Cloud DB: Store readings
4. Cloud DB → Health Monitoring Module: Acknowledge
5. Health Monitoring Module → App: Display updated dashboard

Example C: Doctor Consultation

Actors: Patient, App, Doctor, Consultation Module, Notification Service
Flow:

1. Patient → App: Request consultation
2. App → Consultation Module: Send request
3. Consultation Module → Doctor: Notify availability
4. Doctor → Consultation Module: Accept appointment
5. Consultation Module → App: Confirm appointment

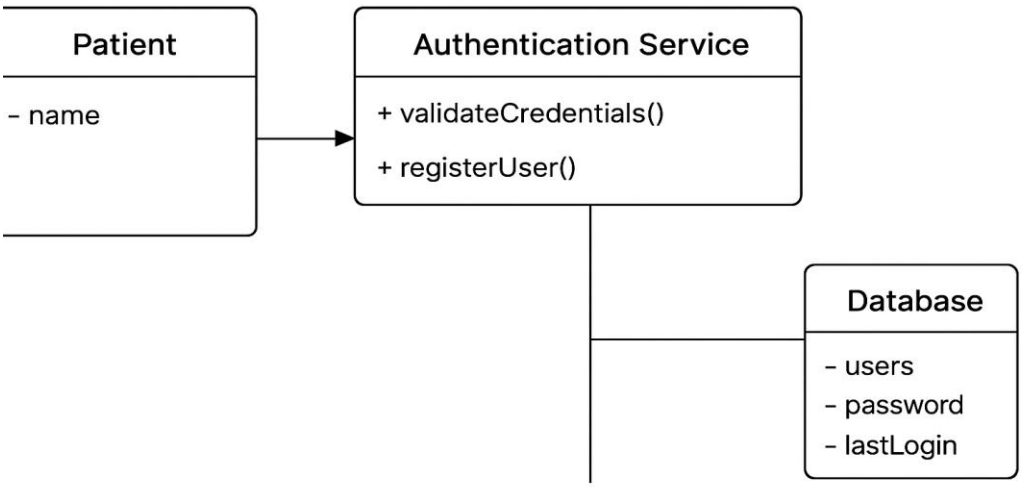
6. App → Patient: Show confirmation
7. Consultation Module → Notification Service: Send reminders

---

Class Diagrams

A class diagram is a fundamental part of a software design document, providing a static, structural view of the system. It models the classes, their attributes, methods, and the relationships between them. For the **IoT-AI Powered Healthcare Network System iOS Application**, the class diagram focuses on the core objects and how they interact to manage patient data and provide insights.

🏛 Class Diagram



📝 Key Classes and Their Relationships

Here is a breakdown of the core classes and their relationships within the system:

1. Core Classes

- **User**: An abstract parent class representing a system user.
  - Attributes: `userID`, `name`, `email`, `passwordHash`.
  - Methods: `login()`, `logout()`, `updateProfile()`.
- **Patient** (`User` subclass): Represents an individual using the app for personal health management.
  - Attributes: `dateOfBirth`, `gender`, `healthConditions`.
  - Methods: `connectDevice()`, `viewDashboard()`, `sendMessage()`.
- **HealthcareProvider** (`User` subclass): Represents a medical professional.
  - Attributes: `specialty`, `licenseNumber`.
  - Methods: `viewPatientData()`, `sendConsultation()`, `setAlertThresholds()`.
- **IoTDevice**: Represents a physical health device.
  - Attributes: `deviceID`, `deviceName`, `deviceType`, `batteryStatus`.
  - Methods: `transmitData()`, `syncData()`.

- **HealthData**: Represents a single data point collected from an IoT device.
  - Attributes: `dataID`, `dataType`, `value`, `timestamp`.
- **Alert**: Represents an AI-generated notification.
  - Attributes: `alertID`, `message`, `timestamp`, `severityLevel`.
- **Communication**: Represents a secure message or call log between a patient and a provider.
  - Attributes: `communicationID`, `senderID`, `receiverID`, `timestamp`, `messageContent`.

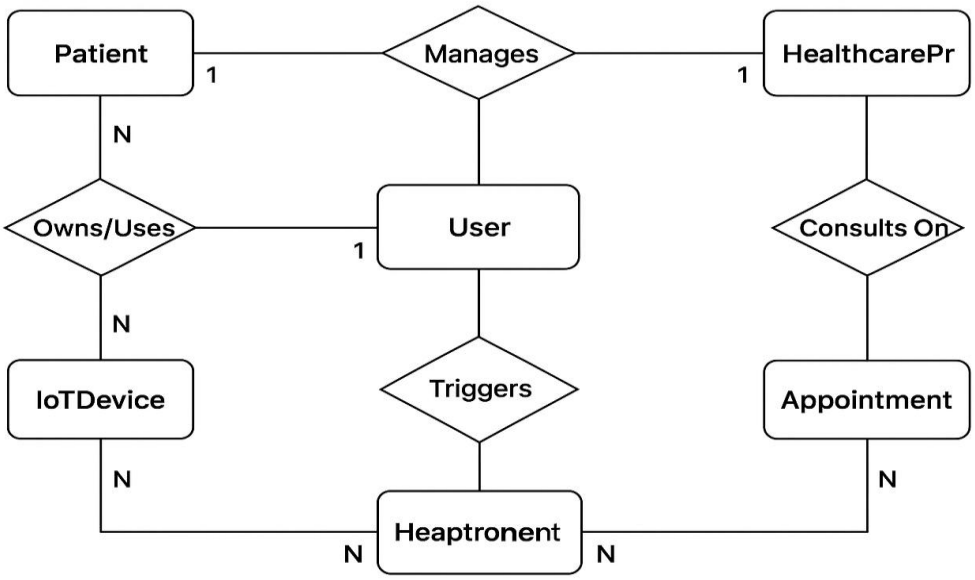
2. Relationships

- **Generalization (Inheritance)**: This is an "is-a" relationship, shown with a solid line and an empty triangle.
  - `Patient` and `HealthcareProvider` **are a** type of `User`.
- **Association**: This represents a simple relationship where one class is connected to another.
  - `Patient` **has a** one-to-many relationship with `IoTDevice` (a patient can have many devices, but each device is linked to one patient).
  - `Patient` and `HealthcareProvider` have a many-to-many relationship with `Communication` (a patient can communicate with many providers, and a provider can communicate with many patients).
- **Aggregation**: A "has-a" relationship, indicated by an empty diamond. The contained objects can exist independently.
  - `Patient` **has a** collection of `HealthData` objects, but a `HealthData` object could exist on its own even if the patient is deleted from a local app (it would still be on the cloud server).
- **Dependency**: A "uses" relationship, shown as a dashed line. One class relies on another for its functionality.
  - The `IoTDevice` class **depends on** the `HealthData` class to transmit data.
  - The `HealthcareProvider` class **depends on** the `Alert` class to receive notifications.

Entity-Relationship

An entity-relationship (ER) diagram for the **IoT-AI Powered Healthcare Network System** visually represents the data model, showing the entities (the "things" in the system) and the relationships between them. This is crucial for designing the database and understanding how different pieces of information are connected.

Entity-Relationship Diagram



Key Entities and Relationships

1. Entities

- **Patient**: Represents a user of the application.
  - Attributes: `patientID` (Primary Key), `firstName`, `lastName`, `dateOfBirth`, `contactInfo`, `healthConditions`.
- **HealthcareProvider**: Represents a medical professional.
  - Attributes: `providerID` (Primary Key), `firstName`, `lastName`, `specialty`, `contactInfo`.
- **IoTDevice**: Represents a physical health monitoring device.
  - Attributes: `deviceID` (Primary Key), `deviceName`, `deviceType`, `serialNumber`, `batteryStatus`.
- **HealthData**: Represents a single data record collected from an IoT device. This entity is the core of the system's data flow.
  - Attributes: `dataID` (Primary Key), `timestamp`, `value`, `unit`, `dataType` (e.g., heart rate, blood pressure, etc.).
- **Alert**: Represents a notification generated by the AI system.
  - Attributes: `alertID` (Primary Key), `alertType`, `message`, `timestamp`, `severity`.
- **Appointment**: Represents a scheduled consultation or interaction between a patient and a provider.
  - Attributes: `appointmentID` (Primary Key), `date`, `time`, `status`, `notes`.

2. Relationships

The lines connecting the entities show how they are related, with cardinalities indicating the number of instances involved in the relationship.

- **Patient & HealthcareProvider**:
  - Relationship: **Manages**

- ➢ Cardinality: One **HealthcareProvider** can manage many **Patients**. A **Patient** is managed by one or more **HealthcareProviders**.

❖ **Patient & IoTDevice**:
- ➢ Relationship: **Owns/Uses**
- ➢ Cardinality: A **Patient** can own or use many **IoTDevices**. An **IoTDevice** is used by one **Patient**.

❖ **IoTDevice & HealthData**:
- ➢ Relationship: **Generates**
- ➢ Cardinality: An **IoTDevice** generates many **HealthData** records. A single **HealthData** record is generated by one **IoTDevice**.

❖ **HealthData & Alert**:
- ➢ Relationship: **Triggers**
- ➢ Cardinality: Many **HealthData** records can trigger one **Alert** (e.g., a series of high blood pressure readings). One **Alert** is triggered by many **HealthData** records (or a single one, but the relationship is many-to-many to account for trends).

❖ **Patient & Alert**:
- ➢ Relationship: **Receives**
- ➢ Cardinality: A **Patient** can receive many **Alerts**. An **Alert** is received by one **Patient**.

❖ **Patient & Appointment**:
- ➢ Relationship: **Schedules**
- ➢ Cardinality: A **Patient** can schedule many **Appointments**. An **Appointment** is scheduled by one **Patient**.

❖ **HealthcareProvider & Appointment**:
- ➢ Relationship: **Consults On**
- ➢ Cardinality: A **HealthcareProvider** can have many **Appointments**. An **Appointment** involves one **HealthcareProvider**.

---

*Other Requirements

- **Compliance:** Must adhere to **HIPAA (US)**, **GDPR (EU)**, and **Apple App Store policies**.
- **Data Backup & Recovery:** Automated daily backups with disaster recovery plan.
- **Integration with Apple Ecosystem:**
  - ➔ HealthKit for health data.
  - ➔ Apple Watch integration for continuous monitoring.
  - ➔ Siri shortcuts for quick actions (e.g., “Show my health report”).