Cloud Doctor Project

Software Design Description

(In accordance with IEEE 1016-2009)

v1.0

BiGC^2

Halil Burak Noyan e2043537

Gökçen Nurlu e1881408

Can Carlak e1819184

Mehmet Cüneyit Kiriş e1819465

January 3, 2015

Change History

Date	Revision	Comment
03.01.2015	1.0	Created

Preface

This document contains the software design information for the “Cloud Doctor” project. The document is prepared according to the “IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE, 1016 – 2009”.

This Software Design Documentation provides a complete description of all the system design and views of the project. The first section of this document includes Project Identification, Stakeholders Identification and requirements, Composition of the developers’ team.

The following sections include document purpose and design viewpoints of the system.

Table of Contents

Table of Figures

1. Overview

This design report includes a complete description of the Cloud Doctor project. This document includes features, functionalities, specifications and explanations about the project which is a design project for the Computer Engineering Design course of the Department of Computer Engineering, Middle East Technical University.

1.1. Scope

The document holds the structural overview of all modules, interfaces, data and module designs in order to support design and development process. In the implementation of the process, this document will be a direction for developers.

1.2. Purpose

This document is prepared to describe and visualize the basic architecture of Cloud Doctor Project. The main aim of this document is to identify the software system which is designed to meet the requirements of the Software Requirements Specification document.

1.3. Intended Audience

The expected audience for this document is the development team of the software. The team can use this document for reviewing and implementing purposes.

2. Definitions

SDLC	System Development Cycle
ADL	Activities of Daily Living
MCU	Microcontroller Unit
BPM	Beats per Minute
RAD	Rapid Application Development
IEEE	The Institute of Electrical and Electronics Engineers
RAM	Random Access Memory
ER Diagram	Entity-Relationship Diagram
UML	Unified Modelling Language

3. Conceptual Model for Software Design Descriptions

The project involves wearable embedded devices and sensors, therefore a basic background in those topics would be helpful.

3.1. Software Design in Context

This project will be built in four main parts, and these parts will have their own design fashions. These parts are; embedded, intermediate, web server, mobile application, respectively. Their detailed design issues will be explained in later chapters.

In embedded device part, development will depend the environment and architecture of the device, and this low-level work will be done conveniently so that the code would be reusable, configurable and extendable with little effort. Additionally, any kind of framework, operating system or library can be used as long as it does not lead to over-engineering, does not have critical performance issues, but speeds up and eases the development.

In intermediate device part, development will be done in modular fashion so that the modules would be easily replaceable and extendable during development. It will also be cross platform and team will be able to switch the hardware with very small change on code. Hence, Python will be the team's current choice.

In web server part, the web application which serves as a user interface will be viewable through a simple web browser, therefore a MVC framework will be used to avoid reinventing the wheel. Such framework will also improve application's scalability in future and the development will be done accordingly. There will also be a web service component, which will accept data from intermediate and additionally, provide data for mobile application when requested, hence it will actually be implemented as set of functions that satisfy CRUD properties. It will modify and query the database in parallel the web application, so its development will be conducted in conformity with concurrency. Database preference will be PostgreSQL which is suitable for the purposes defined above.

Development in mobile side will be simple as possible, since it will be used only for receiving notifications and displaying patient data in a compact way. Native SDK's will be used (such as Android SDK, or iOS SDK) and their own best practices will be followed as much as possible.

3.2. Software Design Descriptions within the Live Cycle

3.2.1. Influences on SDD preparation

This document is prepared by considering the opinions of the stakeholders and the SRS document is an important reference to this document.

3.2.2. Influences on Software Life Cycle Products

The project consists of four parts, connected to their upper part, and each of them has their own development phase. The agile method is used for the software process model and the software product will reach to a final stage after a series of iterations. Our goal in first cycle is to deliver correct data through the system and generate notification accordingly which is the final part. To achieve this, first the sensors, then the other parts must work correctly and coherently. After that, as long as we don't break the interfaces between modules we will be able to accept feedbacks from stakeholders, concentrate on the modules separately and optimize the whole system in further cycles. The iterations and additional/changing requirements of the

stakeholders have influence on software life cycle products. After the first demo in the end of the semester, the model will be scaled to multiple-users with better software design and user interface. Additionally, discussion of physical design of the embedded module (wearable) will be started, which is also a critical issue in this project.

3.2.3. Design Verification and Design Role in Validation

Software design description is the primary reference for the verification and validation of whether the software product designed fulfills the specified requirements in Cloud Doctor SRS Document. The requirements for each specific intended use of the product are modeled in the design view parts of the document. The verification and validation of the design view models are carried out based on this document. SDD influences test plans and test cases in further stages. The testing process will be handled after the code development.

4. Design Description Information Content

4.1. Introduction

This SDD is written to provide architectural design identification of Cloud Doctor Health Monitoring System. This document defines stakeholders, design concerns and viewpoints which specifies different system properties. Additionally, the document consists of design views, overlays and design rationale.

4.2. SDD identification

Design specifications stated in this document will be used in architectural design, system implementation and development phases. After initial development phase, first prototype of system shall be ready to be demonstrated in 16 January 2015. Tentative project completion date is stated as 29 May 2015. Scope of the project is determined as within the boundaries of both hardware and software design, feasibility and health data reasoning research. All rights of Cloud Doctor Health Monitoring System belong to BiGC2 project group. BiGC2 project group is responsible for issuing and authorship.

In this design report UML is mainly used for demonstration the design viewpoints.

This is the initial SDD report of Cloud Doctor project.

4.3. Design Stakeholders and their concerns

BiGC2 team members and end users are main stakeholders of the project. Other stakeholders are instructor of Computer Engineering Design course, Prof. Dr. Atilla Özgit, project supervisor Dr. Onur Tolga Şehitoğlu, Course Assistant Emre Aksan.

Targeted end users are patients and relatives or observers of elders, babies, and patients. These stakeholders have reliability concerns about operation of the system. They demand to be notified in all cases of emergencies and significant conditions. Additionally, integrity and privacy of data are other major concerns.

BiGC2 team design concerns are focused on efficient reasoning of data and interoperability of subsystems. Data processing accuracy is a necessity for that sake.

Other stakeholders' concerns are for project team meeting the development deadlines and proper design documentation which are weekly reports, retrospective documents and this SDD document.

4.4. Design Views

This Project will be implemented as a full-stack application which consist of four adjacent layers. Those layers are Wearable Device, Intermediate Device, Web Service and Phone Application. Each one of layers have their viewpoints and all viewpoints correspond to a view.

In this document contextual, composition, interface, logical, interaction and state dynamics view will be explained in next sections. Detailed description and diagrams about these views will clarify them. Each view is given with its corresponding viewpoint.

4.5. Design Viewpoints

Context Viewpoint:

Roles of users and stakeholders are explained in this viewpoint. This viewpoint helps for verification and validation tests when specifying the context of product. Information will flow between its entities and system.

Composition Viewpoint:

This viewpoint describes interactions between high level modules of system.

Logical Viewpoint:

Logical viewpoint describes logical class structures of Wearable Device, Intermediate Device, Web Service and Phone Application layers individually.

Dependency Viewpoint:

This viewpoint explains the dependencies between four subsystems and inner dependencies of those subsystems.

Information Viewpoint:

Persistent data kept in database are explained in this section. Additionally, dynamic data belonging to the system is demonstrated.

Interface Viewpoint:

This viewpoint includes the details of external and internal interfaces. This viewpoint gives

the information of how each interface will be seen and used.

Interaction Viewpoint:

Interaction methods and structural design of interaction between layers are explained in Interaction Viewpoint.

State Dynamics Viewpoint:

State dynamic viewpoint describes behavior of the system when some particular action happened in the program flow.

4.6. Design Elements

All design elements in related viewpoints will be defined and explained inside their subsection in section “5. Design Viewpoints”.

4.7. Design Rationale

Important aspects such as maintainability, availability and robustness play crucial role to determine design choices. Since the system consists of distinct subsystems such as embedded device component or cloud services, interfaces between all of the subsystems must be well structured. Moreover, having a robust interface allows engineers to focus on individual subsystems. Efficiencies of each components can be improved in a modular manner without corruption of the whole system. Interfaces include communication and data protocols which we designed. In fact, protocols are one of the key decisions for interfaces. They will be analyzed and discussed in following sections using viewpoints.

4.8. Design Languages

Unified Modeling Language is preferred for the design viewpoint specification.

5. Design Viewpoints

5.1. Introduction

Several design viewpoints in terms of design concerns for use will be defined in following subsections. UML shall be used as a design language. The realization of these design viewpoints in terms of design language selections, relates design concerns with viewpoints, and establishes language (notation and method) neutral names for these viewpoints, will be illustrated.

5.2. Context Viewpoint

This context viewpoint is used for describing relationships, interactions and dependencies between the user and the system. The use case diagram is mostly responsible for showing relevant information between the actors and the services.

5.2.1. Design Concerns

The purpose of the context viewpoint is to be crystal clear in the field of services, operations and design scopes concerning the project. This part is obviously a key to development since it mostly investigates the relationship between actors and the services that is offered by the application, thus making it applicable to most design efforts.

5.2.2. Design Elements

Design Entities: Patient, Observer, Owner

Design Relationships:

Patient: A user provides data (via sensors) to the system.

Observer: A user can view after logged in:

- ✓ Real-time patient health data,
- ✓ Analysis of health situation of patient,
- ✓ Filtered information and graphics for viewing purposes by date, by sensor type etc.

Owner: A user can do every operation that an Observer can do.

Additionally, the user can manage after logged in.:

- ✓ Patient sensor configuration,
  - Activate/deactivate sensors
  - Frequency of getting information
  - Threshold of emergency cases.
- ✓ Add/confirm/delete observers,
- ✓ Notifications on/off

User can log in to dashboard panel and log out the system.

5.2.3. Example Languages

The UML use case diagram are for this section are drawn below, as Figure 1, Figure 2, and Figure 3.

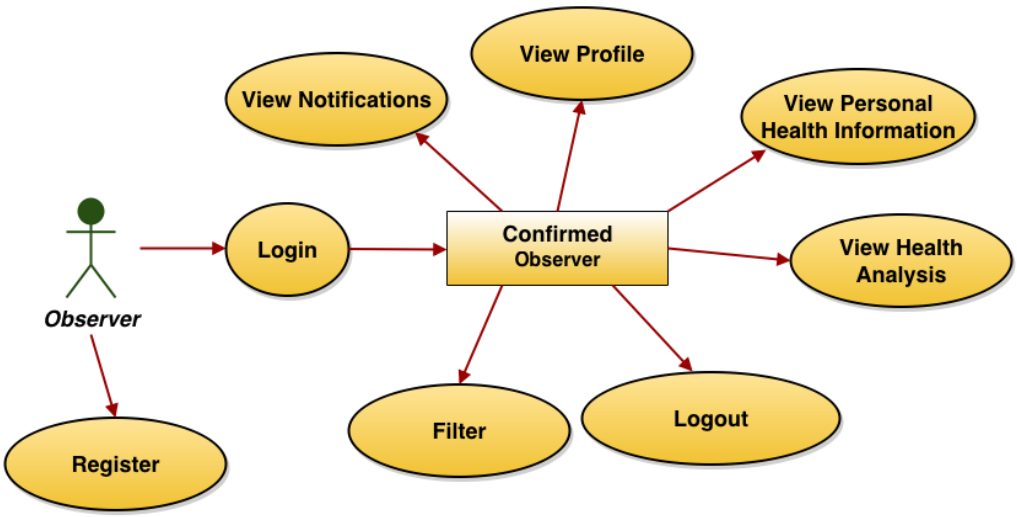

Figure 1 Observer Use Case

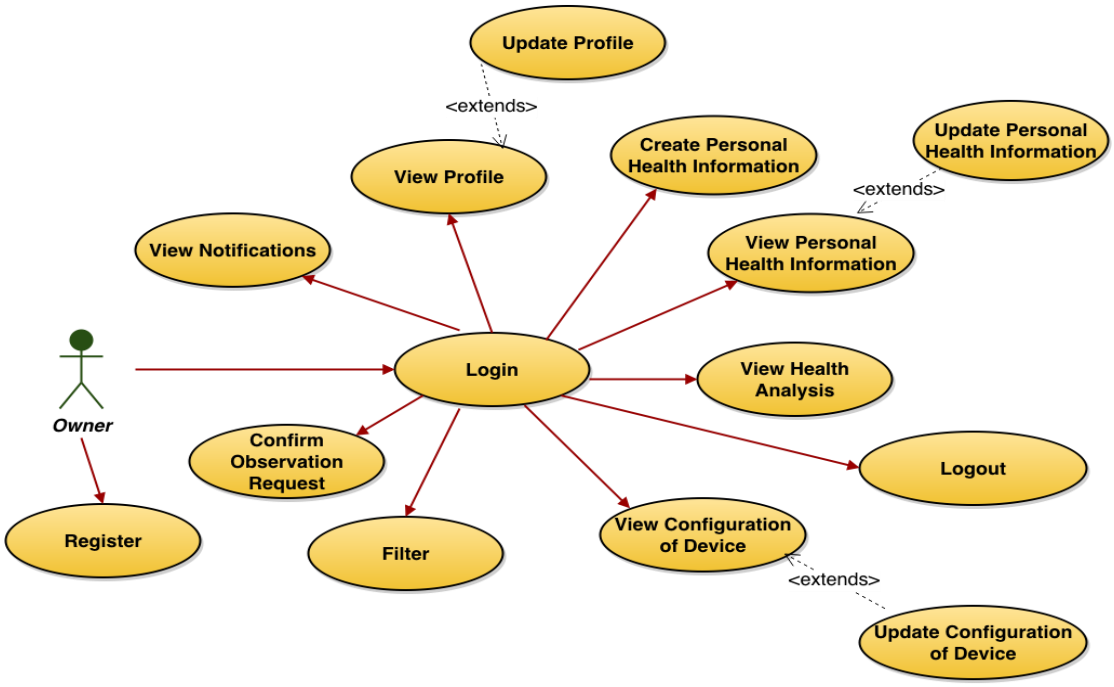

Figure 2 Owner User Case

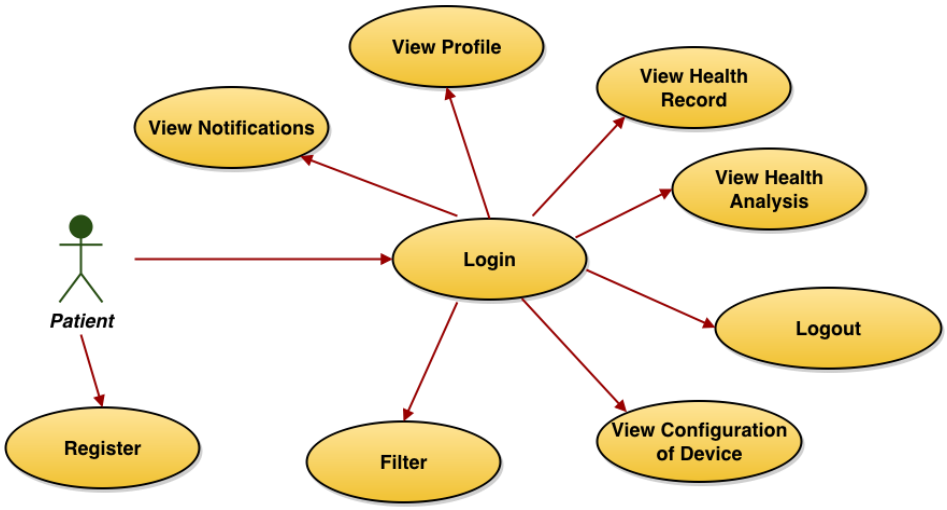

Figure 3 Patient Use Case

5.3. Composition Viewpoint

This section provides information about Cloud Doctor Project's components and their relations with each other.

5.3.1. Design Concerns

The aim of this viewpoint is providing information to stakeholders and programmers for planning and controlling the system. This kind of subsystem level illustration can be used for assembling components, cost estimation and schedules in terms of development effort.

System components as modules, packages, files and their interconnections are illustrated in Component diagram. Cloud Doctor is a higher level project that has four modules, and these are described based on component diagram and deployment diagram, drawn in Figure 4 and Figure 5 respectively.

5.3.2. Design Elements

In embedded device part, low level communication standart libraries will be used for internal and external communication of the hardware. Also, math and digital signaling libraries can be used.

In intermediate device, Python will be used, and Bluez connector package will be used for communication. The application will be multithreaded and communicate with web server through TCP, which is again provided by python's wrapper.

On web server, the project's database management system will be PostgreSQL. Python and C++ connectors will be used during the development of web application and web service component, if needed. Server will be able to execute python scripts, and web application will be built using Django Framework.

On mobile application, Apache's HTTP libraries will be used when a GET or POST request is needed to be sent inside the application. These request will be done towards the web application.

5.3.2.1. Function Attribute

Both the web application and web service component uses the same database. This database is used primarily to store users, sessions and patient data. Web application will provide a user interface to view the

patient data, and web service will provide requested data by querying it from the database, in a predefined form, to the mobile application.

5.3.3. Example Languages

The relevant UML component diagram for this section is as follows:

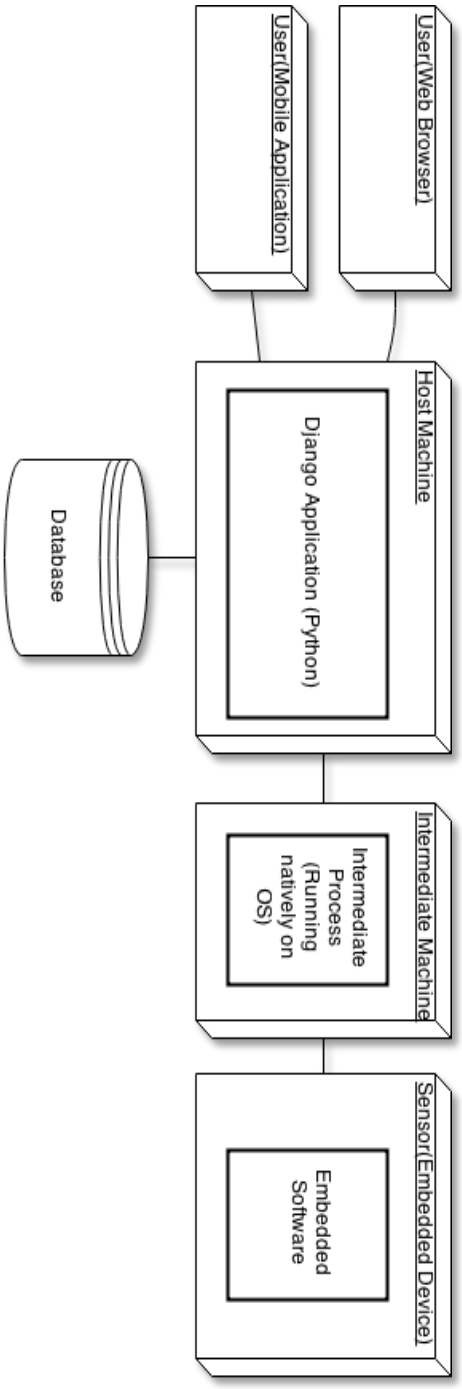

Figure 4 Deployment Diagram

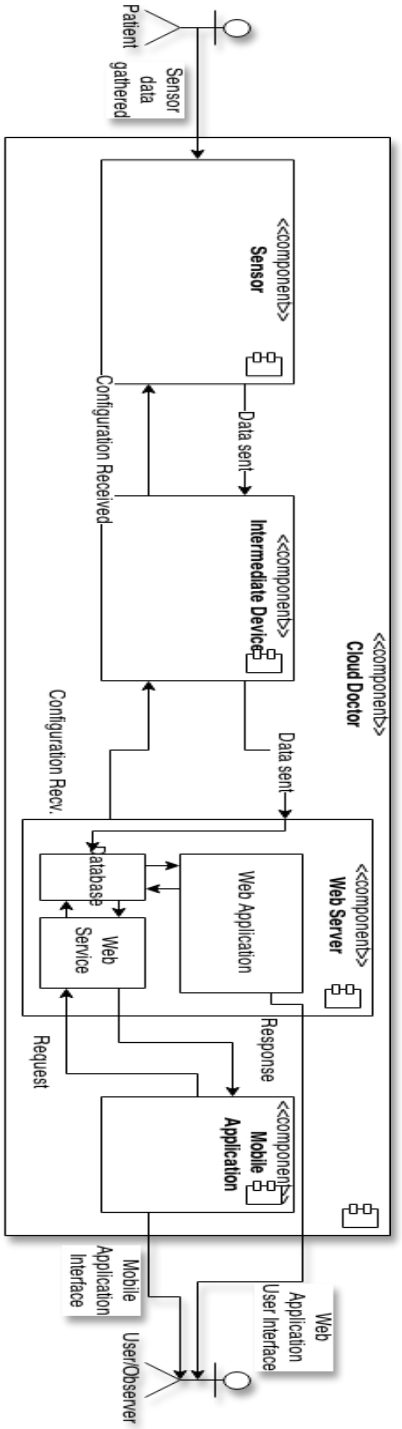

Figure 5 Component Diagram

5.4. Logical Viewpoint

5.4.1. Design Concerns

Logical viewpoint is mainly involved with the static structure in which the focus is compile time entities, associations and/or inheritance among them and the resulting reuse practices and pattern adaptations. Design view associated with the logical viewpoint is based on class diagram and entity - relationship diagram.

5.4.2. Design Elements

5.4.2.1. Intermediate Device

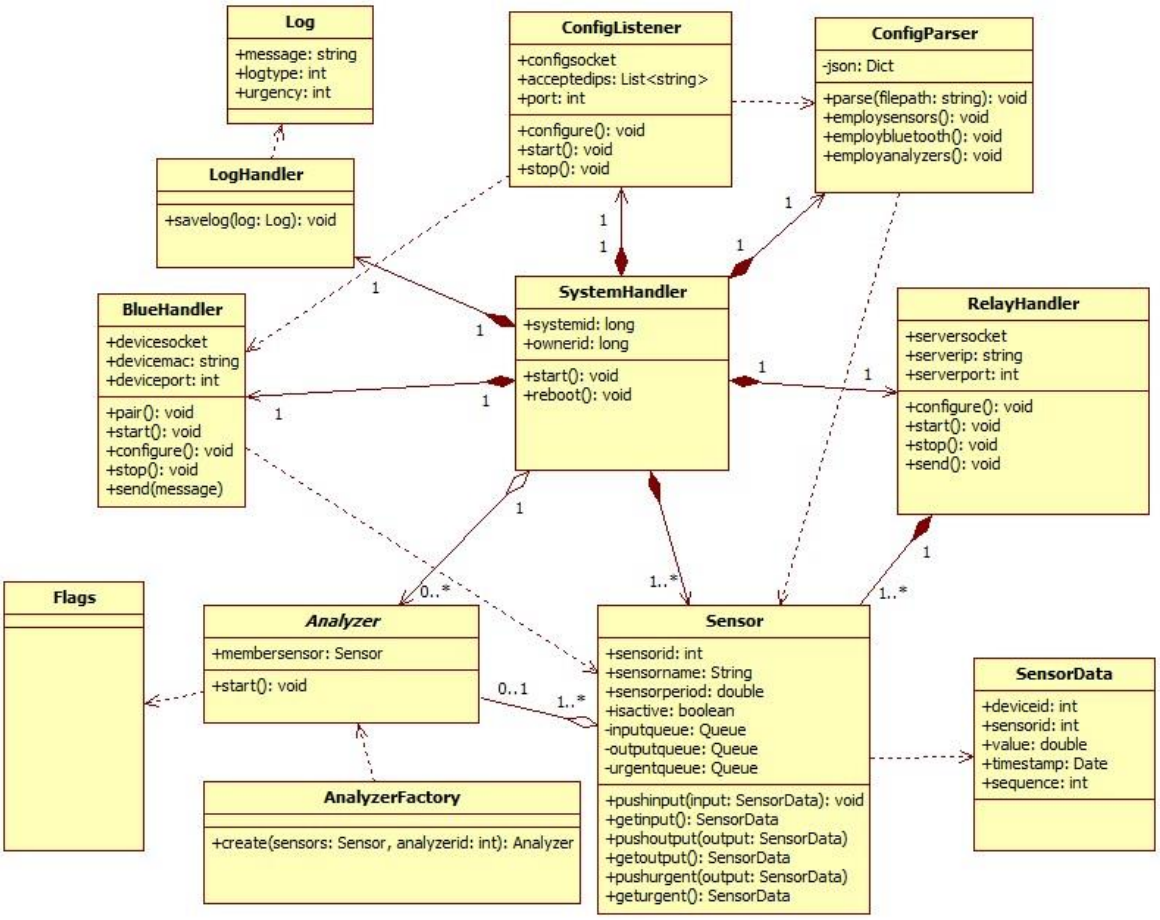

Figure 6 Intermediate Device UML Diagram

Object	***Description***
SystemHandler *(Singleton)*	SystemHandler is the major class which initiates tasks of the system and manages all other handlers. Employs singleton pattern. **Attributes:** systemid: Holds unique identifier of intermediate device. This identifier is required for authenticated transmission. ownerid: Holds unique identifier of owner of the intermediate device. This identifier is required for authenticated transmission. **Functions:** start(): Initiates all of the system tasks reboot(): Reboots the system in case of failure or configuration change

LogHandler *(Singleton)*	Used to create logs of failures, faults and events. **Functions:** createlog(log): creates log of incident. Takes a Log object as parameter. Log object holds attributes and message of a log.
BlueHandler *(Singleton)*	BlueHandler is a class which has a single instance that configures and handles listening part of bluetooth communication. **Attributes:** devicesocket: It keeps the bluetooth socket. devicemac : Holds unique mac address of paired wearable device. deviceport: Holds port number of paired wearable device. **Functions:** pair(): Performs pairing operation with the wearable device. start(): Starts bluetooth handling task which captures sensor data sent via bluetooth. configure(): Configures mac and port of bluetooth device to be connected. stop(): Deactivates bluetooth connection. send(message): Sends given message to bluetooth module of wearable device. Sensor properties can be changed via this command.
RelayHandler *(Singleton)*	RelayHandler handles data transmission with the web server. Employs singleton pattern. **Attributes:** serversocket: Holds the tcp socket. serverip: Holds the ip address of the web server. serverport: Holds the port number of the server. **Functions:** configure(): Configures the RelayHandler object. Transmission periods and server information can be configured. start(): Starts periodic transmission task. stop(): Stops transmission task. send(): Sends queued data to the web server.
Analyzer	An abstract class used for creating diagnosis specific analyzers. **Attributes:**

	membersensor: holds the sensor required for analysis. Analysis task is performed on specified sensor using flags for inter-thread communication. **Functions:** start(): Starts analysis task. Task ends up with outcome about urgency of measurement.
AnalyzerFactory (Factory)	A factory class used for creating analyzer objects. These objects are used and exits after analysis. **Functions:** create(): Creates new analyzer object
Flags (Singleton)	Flags are used for communications between analyzers if necessary.
Sensor	Sensor objects are used for identification of sensors and for input/output operations of these sensors. **Attributes:** sensorid: Holds the sensor identifier. sensorname: Holds sensor name which explains duty of sensor. sensorperiod: Data transmission period of sensor. isactive: States whether sensor is active or not. inputqueue: Holds sensor data obtained from wearable device. outputqueue: Holds already analysed sensor data tagged as regular by analyzer. urgentqueue: Holds already analysed sensor data tagged as urgent by analyzer. **Functions:** pushinput(input): Pushes item to input queue getinput(): Gets item from input queue pushoutput(): Pushes item to output queue. getoutput(): Gets item from output queue. pushurgent(): Pushes item to urgent queue. geturgent(): Gets item from urgent queue.
SensorData	Structure used in order to hold sensor data. **Attributes:** deviceid: Holds unique owner device id. sensorid: Holds unique sensor identifier

	value: Output of sensor measurement timestamp: A field of SensorData object that keeps measurement time of data. sequence: sequence number of sensor data.
ConfigParser *(Singleton)*	ConfigParser's duty is to obtain json configurations in order to modify device settings. JSON files are accepted from web server. **Attributes:** json: A dictionary holds parsed json dictionary. **Functions:** parse(filepath): Parses the json file in given file path. employsensors(): Extract sensor configurations from json and make configurations employbluetooth(): Extract bluetooth configurations from json and make configurations employanalyzers():Extract analyzer configurations from json and make configurations
ConfigListener *(Singleton)*	ConfigListener listens for TCP messages coming from Web Server and parses them to make proper configurations on both Intermediate Device and Wearable Device. **Attributes:** acceptedips: IP's which connection is accepted. port: ConfigListener's listening port number. configsocket: Configuration listener TCP socket **Functions:** configure(): Configures ConfigListener. Listening port start(): Starts ConfigListener. If a configuration message is received, it is decided whether this involves Wearable Device or not, if it involves, appropriate message is sent to Wearable Device. stop(): Stops ConfigListener module.

5.4.2.2. Cloud Service

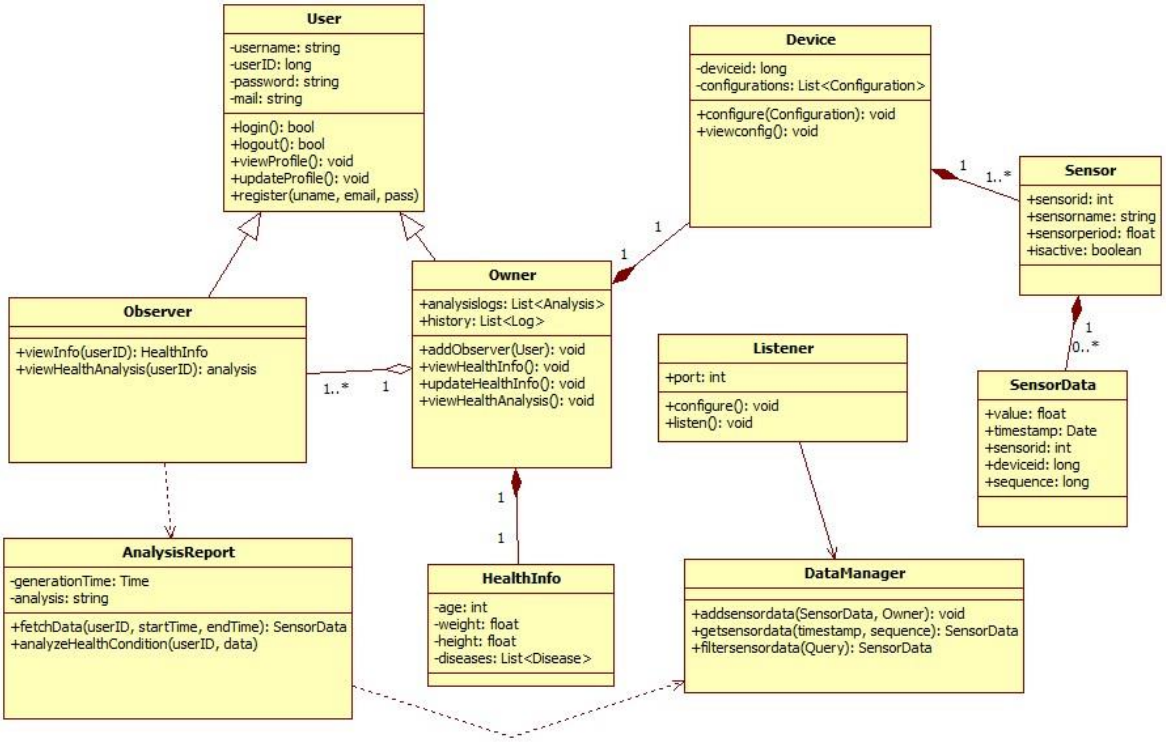

Figure 7 Cloud Service UML Diagram

Object	***Description***
User	Holds information of typical user and provides primitive operations **Attributes:** username: Holds unique username of user. Username is defined by user. password: Holds password of user. userid: Holds unique user identifier. Associate user with corresponding intermediate device. mail: **Functions:** login(): Used for login operation. logout(): Used for logout operation. viewprofile(): Used to view profile of user. updateprofile(): Updates profile of user. register(uname, email, pass): Used to register the user.

Observer	Observer is inherited from User class. Holds information of observing accounts and provides operations for observers. **Attributes:** observing: Holds the list of people who are being observed by the observer user. **Functions:** viewinfo(userID): Displays user information according to corresponding user id if user authorized the observer. viewhealthanalysis(userID): Authorized observers can display health analysis of patient.
Owner	Corresponds to the person who is being monitored and being observed. Inherited from User class. Indicates owner of wearable device. **Attributes:** analysislogs: List of previous analysis reports of owner. Authorized observers can see them. history: List of previous logs of owner activity. Authorized observers can see them. **Functions:** addobserver(): Add an observe to authorize access to owner information. viewhealthinfo(): Displays personal medical information of patient. updatehealthinfo(): Update personal medical information of patient. viewhealthanalysis(): Displays health analysis logs to the user.
Device	Holds device information of owner. Configurations can be done on intermediate or wearable device using instances of this class. **Attributes:** deviceid: Unique intermediate device id. configurations: Current device configurations are kept here. **Functions:** configure(Configuration): Configures the Device object and applies it to actual intermediate device via data transmission. Transmission periods, analysis settings and sensor configurations can be changed. viewconfig(): Displays current configuration of intermediate device.
Sensor	Holds sensor information and measurements belongs to the sensor. **Attributes:** sensorid: Unique identifier of the sensor.

	sensorname: Holds sensor name which explains duty of sensor. sensorperiod: Data transmission period of sensor. isactive: States whether sensor is active or not.
SensorData	Packages data of a single sensor measurement and tags of it. **Attributes:** sensorid: Holds unique sensor identifier value: Output of sensor measurement timestamp: A field of SensorData object that keeps measurement time of data. sequence: sequence number of sensor data.
HealthInfo	Personal medical information of owner of the device. **Attributes:** age: Age of owner. weight: Weight of owner. height: Height of owner. diseases: List of diseases owner has.
AnalysisReport	Provides an interface for management of analyzers. Responsible for initiating analyzer tasks. **Attributes:** generationTime: Generation time of the health report. analysis: This paragraph is created after analysis using fetched data and health information of patient. Paragraph states health inspection results and diagnosis of diseases if there is any. **Functions:** fetchData(userID, startTime, endTime): Fetches health data of user from database to be used for analysis process. analyzeHealthCondition(userID, data): Performs detailed analysis on data and generates analysis paragraph.
Listener	Listens communications originated from intermediate devices and associate them with related objects. **Attributes:** port: Number of listening port. **Functions:**

	configure(): Makes configurations on listening operation. listen(): performs listening operation
DataManager	Used to manage sensor data. Acts as an interface between server and database. **Functions:** addsensordata(SensorData, Owner) : Take sensor data and owner of sensor data as parameter. Checks and adds this data to database. getsensordata(timestamp,sequence): provides specified sensor data if it exists. filtersensordata(Query): Execute given query on database. Query is one of the pre-defined secure sql query objects.

5.5. Dependency Viewpoint

5.5.1. Design Concerns

Data flow across components are strictly dependent to communication between adjacent layers. If an intermediate layer fails, system faces with data losses. In this section, dependencies arose due to interconnections between subsystems and dependencies within those subsystems are defined.

5.5.2. Design Elements

Dependency diagram can be found below:

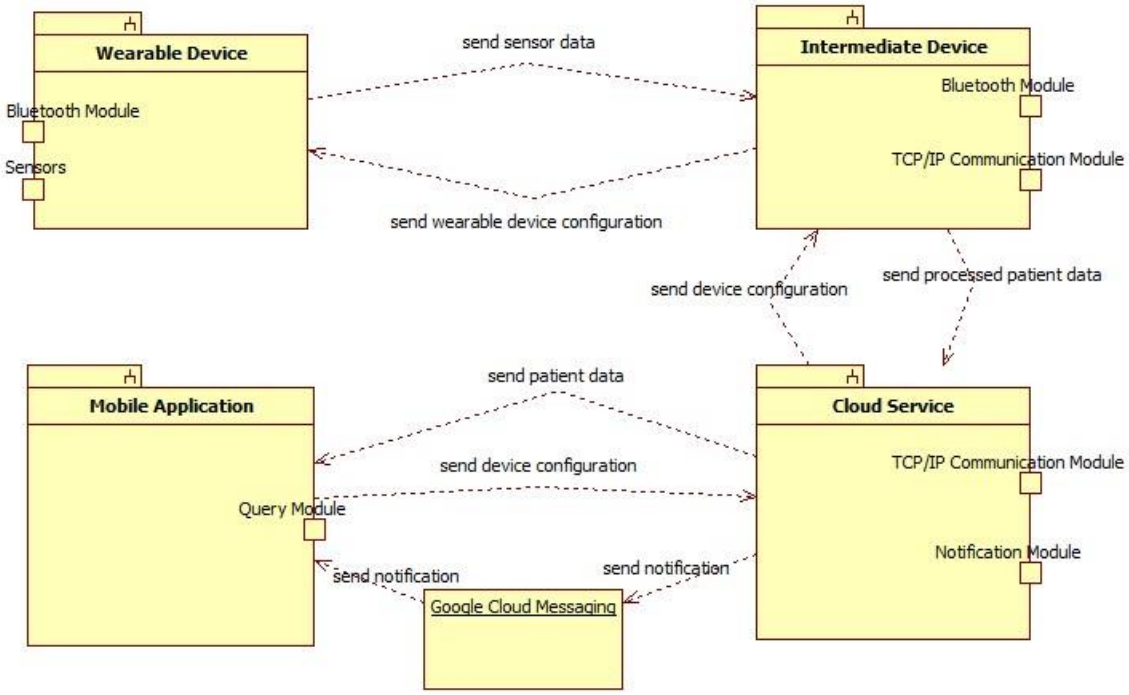

Figure 8 Dependency Diagram

5.5.2.1. Dependencies Attribute

- Wearable Device:

Wearable Device module relies on incoming sensor measurements to operate. Wearable device packs sensor measurements and Bluetooth Module transmits the packed data. Bluetooth Module also listens for wearable device configurations directed from Intermediate Device.

- Intermediate Device:

Intermediate Device operation depends on data transmission originated from Wearable Device. Bluetooth Module on Intermediate Device grabs packed sensor measurements. Analysis process on this device depends on incoming data. According to analysis results, those data is sent to Cloud Service as urgent data or regular data. Data reasoning is the successor process of sensor measurement.

- Cloud Service

Reasoned data is transferred to Cloud Service using TCP/IP Communication Module and further processing of data occurs in Cloud Service. Any failure in this module does not prevent offline warnings created on Intermediate Device but it breaks data flow chain, so it prevents generation of online notifications. Observer and patient health data access and detailed analysis depends on operation of listening TCP/IP Communication Module. Device configuration can be changed using web browser or mobile application and this subsystem is obliged to apply the configuration to lower levels of subsystems stack.

- Mobile Application

Mobile Application provides access to detailed patient information, real time health data, health records and analysis of patient. If any of the lower members of subsystems stack fails, user becomes unable to see up-to-date information. Notifications are sent to mobile application user via Google Cloud Messaging Service. Notification system almost completely depends on GCM from this view.

5.6. Information Viewpoint

5.6.1. Design Concerns

The ultimate purpose of any information system is to manipulate data in some form. This data may be stored persistently in a database management system, in ordinary files, or in some other storage medium such as flash memory, or it may be transiently manipulated in memory while a program executes.

The main purpose of using information viewpoint in this project is exactly data modelling. In the Cloud Doctor System, it must manipulate some data in a database. Storing, deleting and replacing some amount of data in database are kinds of manipulation. The Cloud Doctor System requires a database management system and data flow throughout database. Using information viewpoint make this requirements more clear.

5.6.2. Design Elements

5.6.2.1. User

“User” is model to store user data, consisting of fields:

UID(PK): unique id for each user

Username(QK): string field to store users' nickname

Password: encrypted field stored for authentication

UserType: string field to store user group: Owner or Observer

RealName: string field to store users' name and surname

Phone: string field to store users' phone number

E-mail: string field to store users' e-mail

Address: string field to store users' address

ObserverCount: integer field to store number of observer for Owner user, otherwise this field is NULL

5.6.2.2. RealTime Data

"RealTime Data" is model to store users' real time sensor data, consisting of fields:

DataID(PK): unique id for each data

UID(FK): foreign key for user ID

Type: text field to store data type: temperature, pulse, etc...

DataValue: text field to store data value

Timestamp: time field to store the data time

5.6.2.3. Analysis Report

"Analysis Report" is model to store users' health report which generated by analyzer, consisting of fields:

ReportID(PK): unique id for each report

UID(FK): foreign key for user ID

BeginTime: time field to store report begin time

EndTime: time field to store report end time

EmergencyLevel: string field to store report emergency status

Timestamp: time field to store report timestamp

5.6.2.4. Devices

"Devices" is model to store users' device information, consisting of fields:

DeviceID(PK): unique id for each device

UID(FK): foreign key for user ID

5.6.2.4.1. Sensors

"Sensors" is model to store devices configuration, consisting of fields:

DeviceID(FK): foreign key for device ID

Type: string field to store sensor type: pulse sensor

Status: string field to store sensors' status

SampleRate: integer field to store sensors' sampling rate

5.6.2.5. Notifications

"Notifications" is model to store users' notifications, consisting of fields:

NID(PK): unique id for each notifications

UID(FK): foreign key for user ID

EmergencyLevel: string field to store notifications' emergency status

Seen: time field to store notifications' seen time

Timestamp: time field to store generation of notification time

5.6.3. Example Languages

The relevant ER diagram for this section is drawn below:

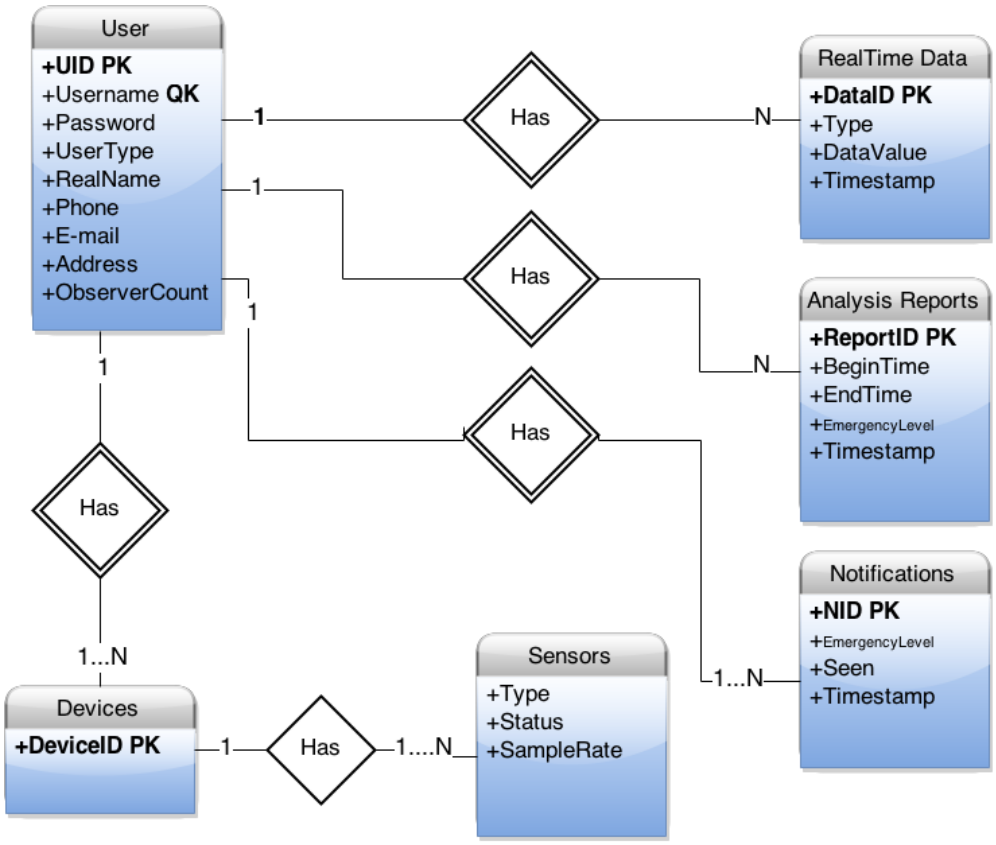

Figure 9 ER Diagram

5.7. Interface Viewpoint

This section explains all communication protocols, standards and connectivity between 4 major subsystems. Moreover interconnections and APIs for internal/external libraries, frameworks and software tools are explained.

5.7.1. Design Concerns

Communication interface between wearable and intermediate component must be fast and lightweight due to following reasons:

- Embedded system must not be late for real-time measurements so that emergent situations can be detected on right time.
- Power consumption should be at reasonable levels so that battery of the embedded device can last longer.

Remaining subsystems' communication interface mainly concern with modularity, security and efficiency. Same concerns are applicable to internal modules, service programs and daemons.

5.7.2. Design Elements

From embedded device to intermediate device, data packets are in the following form:

Packet: [Begin] [Timestamp] [Sensor Type] [Value] … … … [Sensor Type] [Value] [End]

1 Byte 4 Byte 1 Byte 16 Byte 1 Byte 16 Byte 1 Byte

As a communication media, Wearable device (Layer 1) communicates with intermediate computational device via Bluetooth standardization.

Communication and data formats between other system layers are JSON due to its simplicity, being a widely used standard and efficiency over XML like formats.

Local Area Network connects intermediate computational device (Layer 2) with either Ethernet or Wi-Fi standards. This provides intermediate device to access the Internet and eventually to cloud services (Layer 3). Packet transmission is achieved through using TCP/IP because its reliability is important for the system.

Cloud services push necessary notifications to the mobile and web platforms via the Internet. HTTP application protocol is used for web access.

Database access needs for related programs are handled via PostgreSQL connectivity drivers.

5.8. Interaction Viewpoint

Interaction viewpoint is provided through sequence diagrams to explain the main functionalities of modules of the project in a nutshell.

5.8.1. Design Concerns

The aim of this view is showing the flow of application running system. Cloud Doctor project has several workflows but two main sequences were shown here to illustrate all functionalities altogether.

5.8.2. Design Elements

The system starts with collecting data from patient since our project is based on collecting data about individual's health, storing them and trigger events if needed. Then, the data is sent to intermediate and stored them. These data will be sent to web server periodically for a permanent storing. It is also analyzed in intermediate module and necessary signals are sent if there is an emergency condition. After the server receives data, the data is pushed to database and analyzed in a more sophisticated way. Web application running on web server also prepares a user interface to serve that stored data through a web browser. Finally, mobile application can also provide an user interface to represent patients information and view received notifications.

5.8.3. Example Languages

UML Sequence Diagrams mentioned above are shown as FIGURE 10 and FIGURE 11 below.

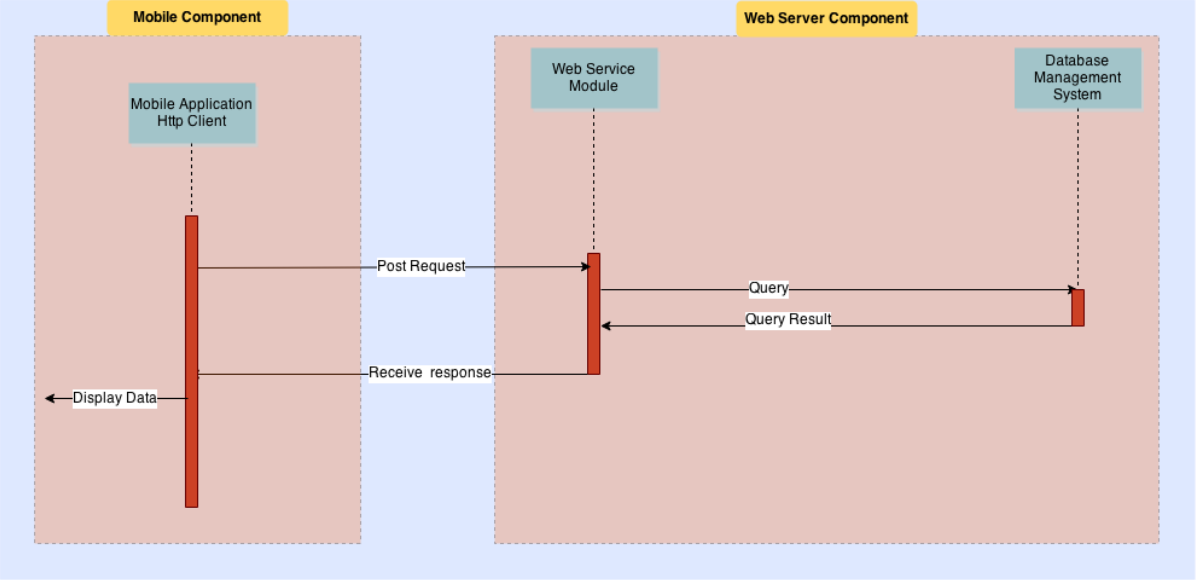

Figure 10 Mobile Application Sequence Diagram

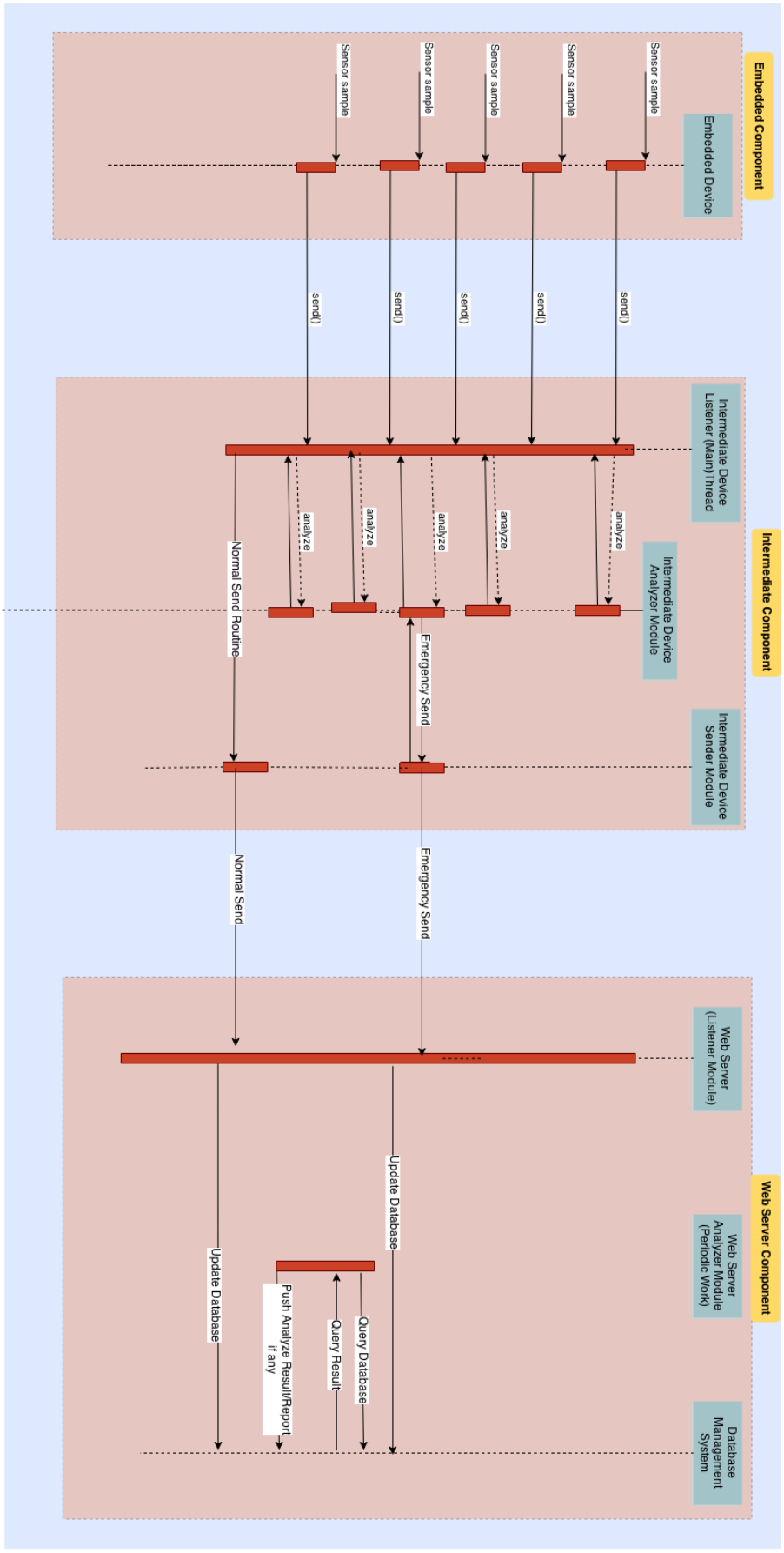

Figure 11 Overall Application Sequence Diagram

5.9. State Dynamics Viewpoint

State viewpoint deals with behavior of the system when some particular action happened in the program flow. It shows how the application reacts to that action.

5.9.1. Design Concerns

This viewpoint is basically about states and reaction of those states to events.

5.9.2. Design Elements

The viewpoint consists of states which gives information about program status before user interacts with the application for more action and the transition between some particular actions. Moreover, the state flow and the critical region are described in the diagram for further implementation.

5.9.3. Example Languages

UML state diagram of the project is below:

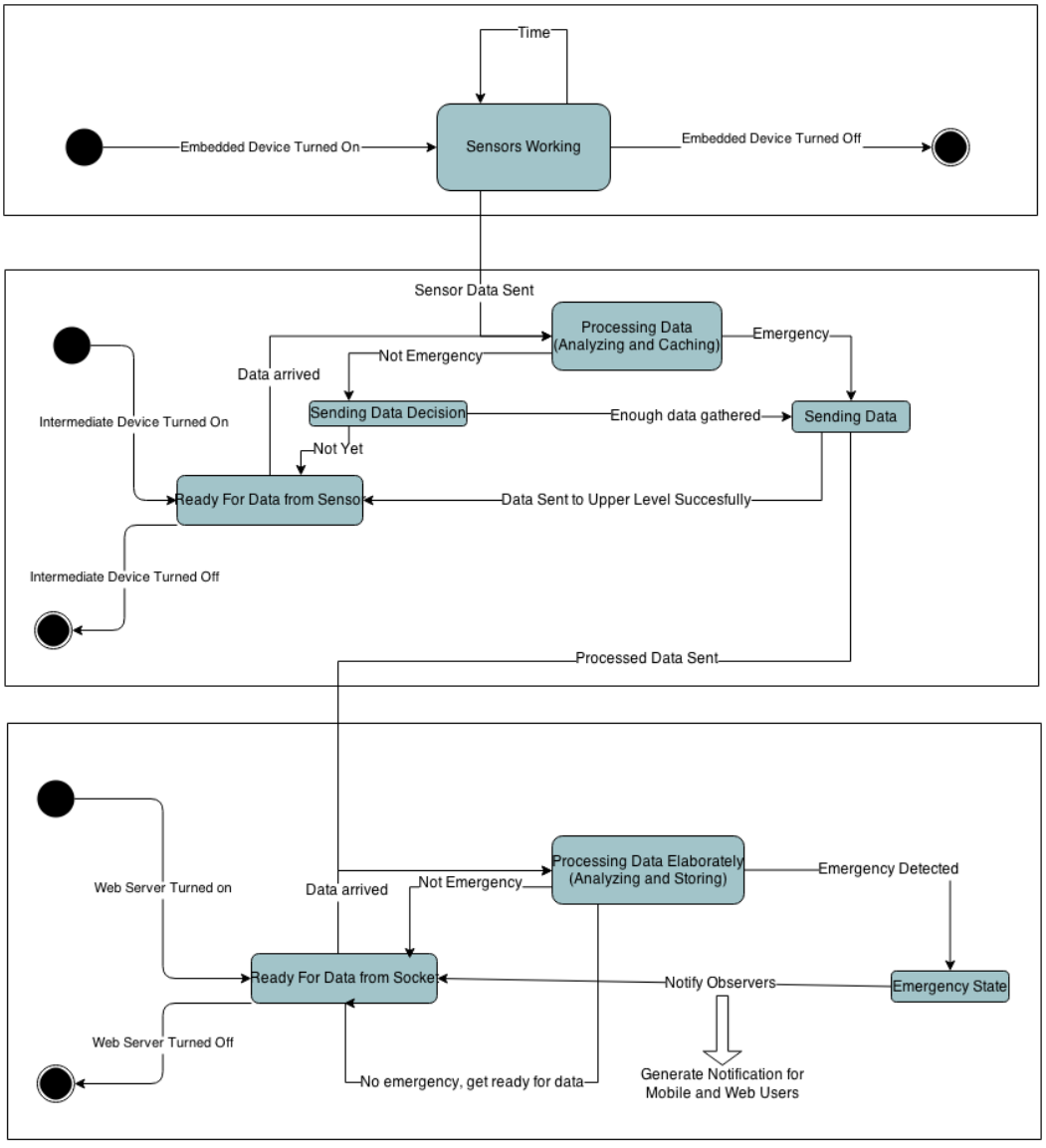

Figure 12 State Machine Diagram

6. Traceability Matrix

XREF\FIGURE	Figure 7	Figure 8	Figure 9	Figure 10	Figure 11	Figure 12
OWN1	X					
OWN2	X		X			
OWN3	X			X		
OWN4	X			X		
OWN5	X					
OWN6	X					X
OWN7	X			X		
OWN8	X					
OWN9	X			X		
OWN10	X				X	
OWN11		X				
OWN12	X	X			X	
OWN13	X				X	
OWN14	X					X
OBS1	X					
OBS2	X					
OBS3	X			X	X	
OBS4	X			X	X	
OBS5	X		X	X		
OBS6	X			X	X	
OBS7	X			X		
OBS8	X			X		
OBS9	X			X		

Index