Department of Computer Science & Engineering
The University of Texas at Arlington

Architectural Design Specification
CSE 4316: Senior Design I
Fall 2020

Team 10
MedTech

Hemantha Govindu
Nikita Menon
Maxwell Pham
Nisarg Shah

Revision History

Revision	Date	Author(s)	Description
0.1	10.14.2020	GH	document creation
0.2	10.14.2020	HG,NM,MP,NS	Subsystems Draft
0.3	10.12.2015	AT, GH	release candidate 1
1.0	10.20.2015	AT, GH, CB	official release
1.1	10.31.2015	AL	added design review requests

CONTENTS

List of Figures

List of Tables

1 Introduction

MedTech is a tool that will allow medical professionals to document patient information and also decrease the time it takes to complete the process than industry standard software solutions. This software should enable its user to gain higher levels of productivity in short amount of time with greater use of automation, simple UI/UX design and lower latency delays.

This project is built specifically for Medical professionals like nurses and doctors to help them in their charting process. In addition, this system will also help them keep track of a patients medical history, possible allergies, medications and their hospital stay along with their other medical details.

This project is comprised of multiple systems that work together to help us run this system. It consists of a UI/UX component which will be build on node.js to provide a user friendly interface. It will also be combined with the Apache web server to host OpenEMR, an open source medical software. The program will also be using MySQL as its database for its easy to use nature and great compatibility to store medical information.

One of the most important requirements that we have is security. We need to make sure that user data stays safe and also that only people with correct and sufficient credentials can get the highly sensitive patient data. We will be enforcing the CORS policy for the authorization system. We also want to make the software user friendly and uncluttered with only necessary information displayed. We would have a two tab system where we will have one tab for all the patient assigned to the logged-in nurse and one for the patient that they want to view and inspect their full health details. Another system is to have a table of contents that the user can click on and point to a menu, such as on a patient file, to view vitals, simply go to the table of contents and click vitals, which in turn should take the user to the vitals page for viewing and charting.

2 System Overview

MedTech is a web application and we hope to follow a layered system architecture. This kind of architecture is most helpful when there is an interaction and communication across different platforms. The layers to our web application includes :

1) Open EMR layer
2) Client Layer
3) OpenEMR Database Layer
4) Log Controller Layer
5) Apache Web Service Layer

This kind of architecture helps improve flexibility and is easy to maintain. It helps enhance the client-server model and can dynamically process, modify, generate data and update user interface.

This architecture separates user interface from OpenEMR and OpenEMR from MySQL Database layer. Thus it helps to have a clear line of distinction between patient information database and the web application.

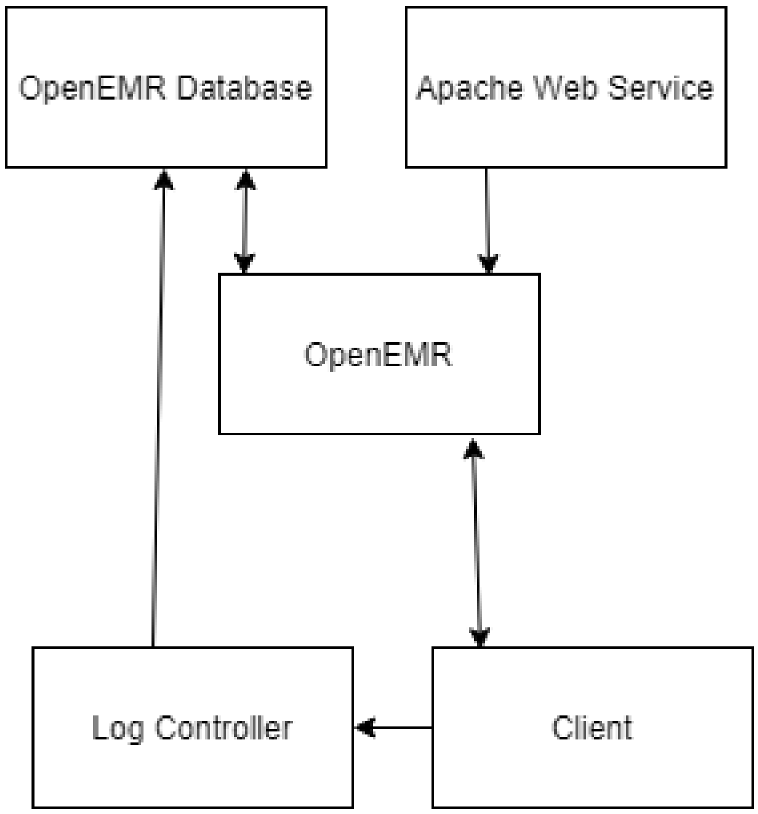

Figure 1: A simple architectural layer diagram

2.1 OpenEMR Layer

OpenEMR is part of the the Application layer in a web application holding the main functions for medical software.

2.2 Client Layer

The Client layer is part of the presentation Layer in a web application. It is the layer that is used to present data to the application layer that in an accurate, well-defined and uniform format. It receives the data from the application layer and translates it to a format that is readable to other computers and other users. We utilize components of the client like the REST API in JSON to exchange information.

2.3 OpenEMR Database Layer

OpenEMR Database Layer is part of the data access layer in a web application. It is the layer that provides simplified access to store data stored in a persistence storage of some kind. MedTech utilizes the OpenEMR Database.

2.4 Log Controller Layer

Log controller is part of the business logic controller of our web application. It is the layer that helps facilitate communication between the presentation layer and data access layer. So the log controller acts as an interface between Apache Web Service, OPEN EMR and MySQL.

2.5 Apache Web Service Layer

Apache Web Service Layer is also a part of the presentation layer of the web application. It is usually the layer that's responsible for communication protocols and the server that utilizes the TCP/IP in our application is Apache Web Server.

3 Subsystem Definitions & Data Flow

The core layer, OpenEMR, includes the core function of the system that allows for electronic medical records to be managed. In order to access this information from the outside, the REST API is needed, which communicates to the Rest Controller for OpenEMR to translate instructions routed into REST API into the Controller, then routed into OpenEMR Core which handles that information.

The OpenEMR Database keeps information for OpenEMR including user information and patient information, the 2 main information components we will be working with. They are stored in the Core Table. The Log Table is a separate table from OpenEMR which keeps track of all requests from the Client.

The Client is where the user manages information. This information is routed from OpenEMR to the user interface via REST API. Actions the client has done will also be posted to the log controller for record keeping.

The Log Controller's purpose is for record keeping for the client. This is done by receiving requests from the client's REST API commands, then posted to the Log Table.

The Apache Web Service hosts OpenEMR software and allows users to connect to the service. It will also facilitate permissions the client can and cannot have.

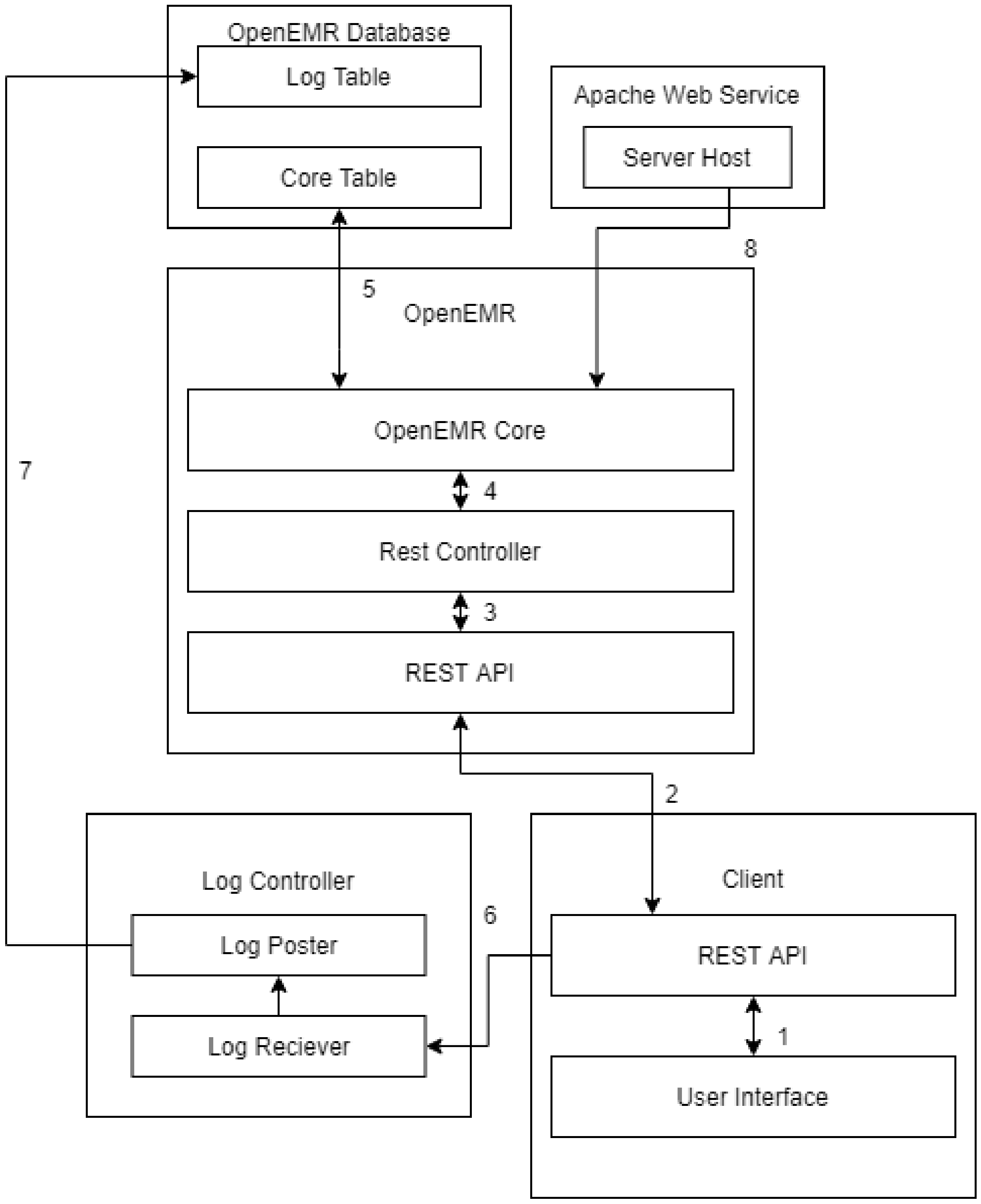

Figure 2: Subsystems Overview Diagram

4 OpenEMR Layer Subsystems

In this section, the layer is described in some detail in terms of its specific subsystems. Describe each of the layers and its subsystems in a separate chapter/major subsection of this document. The content of each subsystem description should be similar. Include in this section any special considerations and/or trade-offs considered for the approach you have chosen.

This layer includes subsystems: OpenEMR Core, Rest Controller, and REST API, and are responsible for making up the entire program of OpenEMR's functioning for medical practice. The data is exchanged to the front end using the REST API to handle HTTP requests. The Rest Controller handles translating the API requests to function with OpenEMR's Core.

4.1 OpenEMR Core

This subsystem, OpenEMR Core, handles functions for medical practice such as adding/editing/deletion of patients, patient data, facilities, procedures, lab reporting, visits, and many other functions. It backs up data from a MySQL server and is hosted from an Apache server. Its data can be translated into the Rest Controller for use of REST API application.

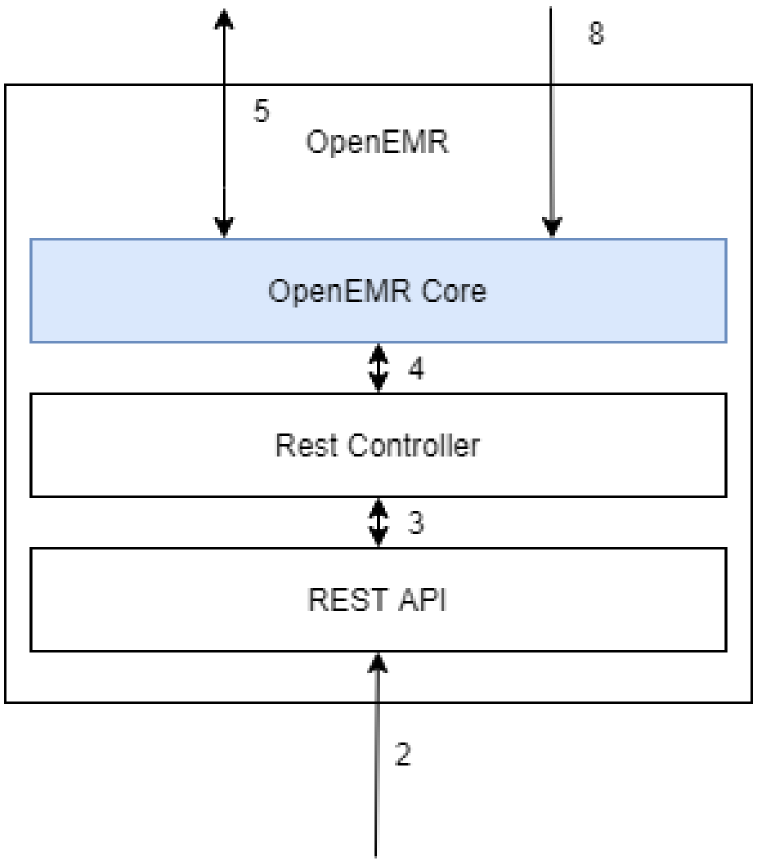

Figure 3: OpenEMR Core subsystem diagram

4.1.1 Assumptions

Inputs/Outputs from the front end should be going through the Rest Controller. OpenEMR should work as intended upon installation and its inner functioning should not be changed.

4.1.2 Responsibilities

Each of the responsibilities/features/functions/services of the subsystem as identified in the architectural summary must be expanded to more detailed responsibilities. These responsibilities form the basis for the identification of the finer-grained responsibilities of the layer's internal subsystems. Clearly describe what each subsystem does.

OpenEMR core is responsible for containing functions for medical practice. Patient core functions include add a patient, create visit, create encounter, chart vitals, and chart examinations. Documentation/Charting functions include procedures, lab results, and lab overview. These core functions will be responsible for handling the listed data.

4.1.3 Subsystem Interfaces

Table 2: OpenEMR Core Subsystem interfaces

ID	Description	Inputs	Outputs
#4	Recieve/Send information in to REST controller	Info recieve	Info Send
#5	Recieve/Send information from MySQL Database	Info Recieve	Info send/update
#8	Server host from Apache web to host OpenEMR	N/A	Host

4.2 Rest Controller

This subsystem, Rest Controller, handles exchange of data between OpenEMR core and the REST API. This handling is done by obtaining requests from the REST API, and then translate the request over to the controller, and then send the translated request over to OpenEMR Core to obtain or send data.

4.2.1 Assumptions

Inputs and Outputs to the REST API should be in REST format, which consists of HTTP commands such as POST or GET. The Rest Controller was provided in the OpenEMR installation and should work as intended.

4.2.2 Responsibilities

The Rest Controller is responsible for handling REST API requests, which can request information for the following: Facility, Practitioner, Patient, Immunization, Allergy, Procedure, Drug, Prescription, Insurance, Appointment, Document, and Message. The listed data can be used to exchange data between the REST API and OpenEMR core with the Rest Controller translating the requests from JSON into PHP format.

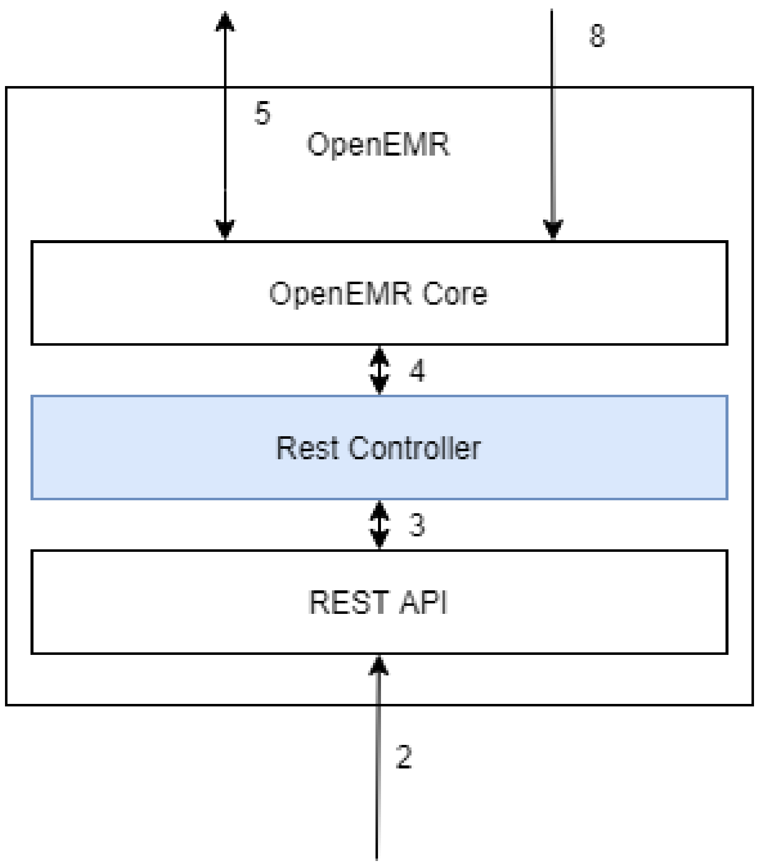

Figure 4: Rest Controller subsystem diagram

4.2.3 Subsystem Interfaces

Table 3: Rest Controller Subsystem interfaces

ID	Description	Inputs	Outputs
#4	Recieve/Send information in to REST controller	Info recieve	Info Send
#3	Recieve/Send information in REST format	REST format recieve	REST format send

4.3 REST API

This subsystem, REST API, handles HTTP requests to and from the client. Data is exchanged in JSON format, and this data is sent to the Rest Controller to translate it into PHP.

4.3.1 Assumptions

REST API should allow permissions for the server to exchange data with the client, otherwise none or all clients could exchange data with the server, OpenEMR, which results in exposed security risk.

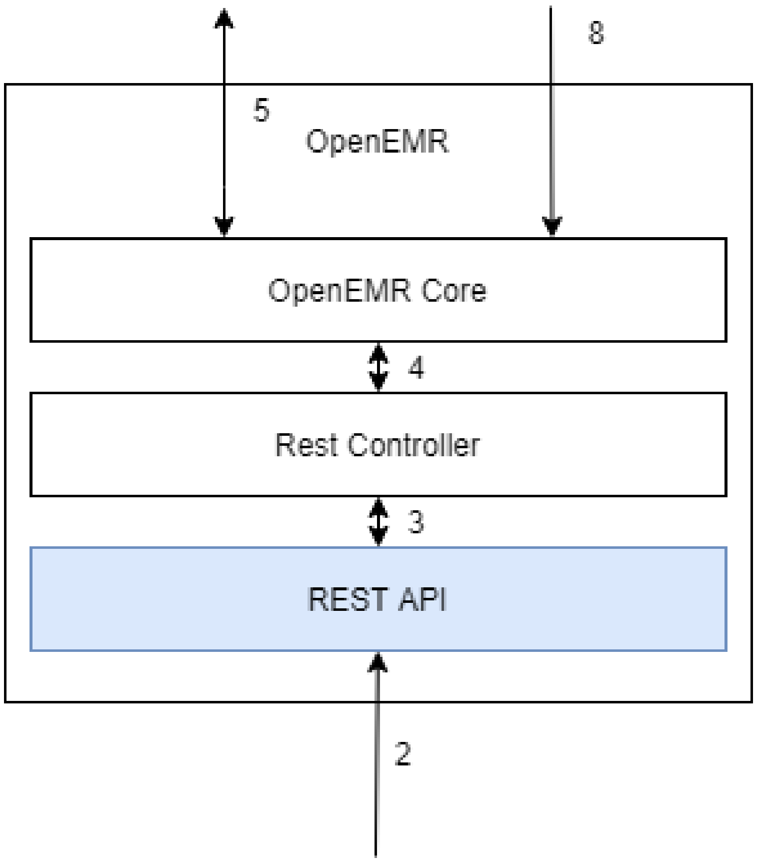

Figure 5: REST API subsystem diagram

The server should know which client to exchange data to, and the client should know which server to exchange data to.

4.3.2 Responsibilities

REST API is responsible for handling HTTP requests such as GET, POST, PUT, or DELETE. These requests come from the client along with a JSON message, which in turn the REST API sends over that request to the controller to interpret the message.

4.3.3 Subsystem Interfaces

Table 4: REST API Subsystem interfaces

ID	Description	Inputs	Outputs
#3	Recieve/Send information in REST format	REST format recieve	REST format send
#2	Recieve/Send information in REST format from/to client	Information Requests from client	Information send to client

5 OpenEMR Database Layer Subsystems

OpenEMR Database layer utilizes the MySQL database that is the data access layer for our web application.

5.1 Core Table

The Core Table is the main table provided by OpenEMR. It is usally the table the consists of the most number of tables related to it.In our application, it will be the table with patient info. Since we many of our features deal with patient care and their information for the same is vital.

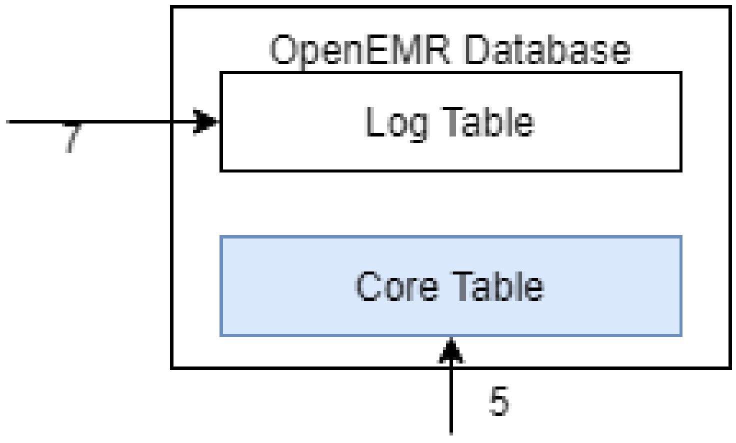

Figure 6: Core Table subsystem diagram

5.1.1 Assumptions

There should be no technical issues with the hardware or the connection to the MySQL server to access the core table.

5.1.2 Responsibilities

We would be sending regular database requests to the MySQL server that houses the OpenEMR database. The requests would then be forwarded to the presentation layer for the user to view their successful requests.

5.1.3 Subsystem Interfaces

The table below describes interfaces of the Core table. Inputs requires the section of the database the user wishes to access and the output would be request to the Open EMR database.

Table 5: Core Table Subsystem interfaces

s

ID	Description	Inputs	Outputs
#5	Receive/Send information to/from OpenEMR	OpenEMR Database requests	Database sends results to OpenEMR

5.2 Log Table

Log Table is the documenting tool used in this software. It is responsible for making an entry in the log table for any action the user is taking.

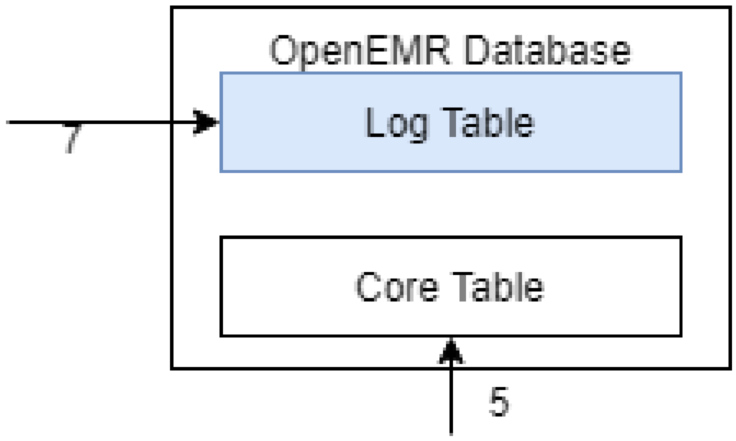

Figure 7: Log Table subsystem diagram

5.2.1 Assumptions

There should be no technical issues with the hardware or the connection to the server to record the logs. The activity should be posted from the log controller.

5.2.2 Responsibilities

It is responsible to maintain a log for every action that the user takes and document it correctly.

5.2.3 Subsystem Interfaces

Table 6: Log Table Subsystem interfaces

ID	Description	Inputs	Outputs
#7	Client activity is recorded into log table	Information from client activity is logged into table	N/A

6 Client Layer Subsystems

This subsystem includes REST API and it corresponding User interface. The User face will the customer-facing front end part of our system which will be responsible for providing our customers with adequate graphics so that they can interact with the software without any issues. The REST API will be used to connect to the Apache web server to request any information that our customer needs. The request gets passed from the user through the API keys to the server where the server looks for the requested information and then sends it back to the API key which then passes it to our user interface to display the necessary information.

6.1 User Interface

User Interface is the part of the software that deals with interacting with the users and providing them with information upon request. User Interface is also responsible for the general layout of our software from a customer standpoint of view and will be the primary interactive display for the user.

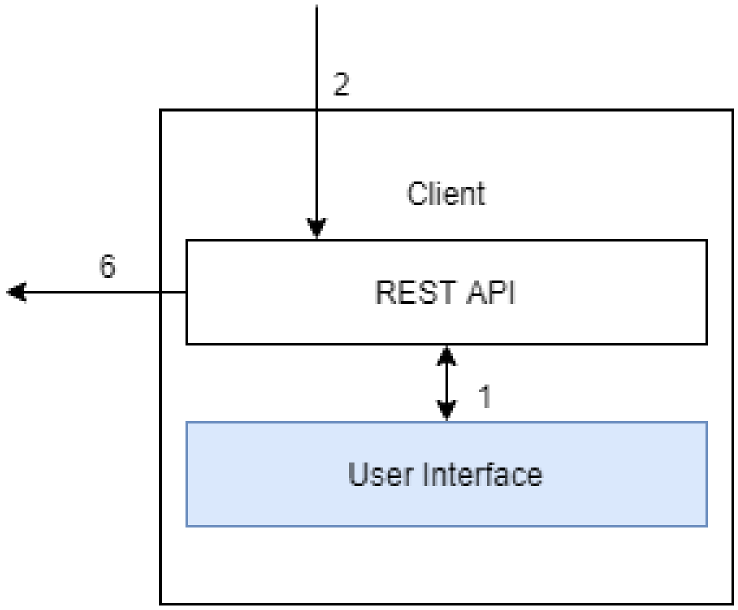

Figure 8: User Interface subsystem diagram

6.1.1 Assumptions

Our front end side is loaded on a compatible browser. All commands to fetch should be in the correct format with their authorization code to allow information exchange.

6.1.2 Responsibilities

The User Interface needs to be simple and uncluttered for maximum performance and should also display all the relevant information and should correctly parse the information to the REST API if any information is needed to be fetched from the database.

6.1.3 Subsystem Interfaces

The inputs for this subsystem will be the JSON responses that will be collected whenever the user is requesting some information which needs to sends a request to the server.

The outputs will be the encoded JSON responses that we get form the server which is then decoded and displayed to the user.

Table 7: UI Subsystem interfaces

ID	Description	Inputs	Outputs
#1	REST messages are decoded for reading in client, and encoded for sending to OpenEMR	JSON message from REST API	Encoded JSON message to send to REST API

6.2 REST API

REST API will be the primary gateway for interaction between our client side and the server side. This part is also significantly important for the overall health and quality of the software since it manages all the user request and has to correctly parse the JSON request to the server and then decode the JSON response it got back from the server. Apart from this, REST API keys request will also be important as this will be used to log all the request the customer makes in order to document every action the user takes for legal reasons.

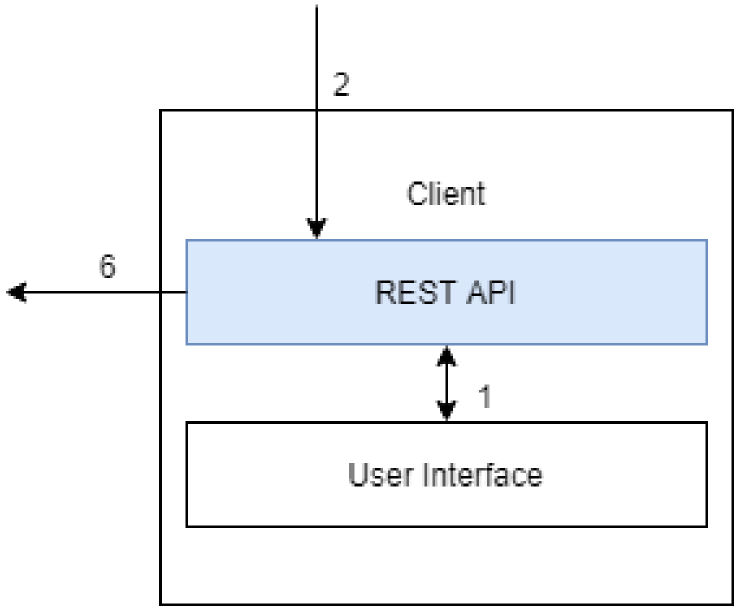

Figure 9: REST API subsystem diagram

6.2.1 Assumptions

The data is not corrupted in between the transfers.

6.2.2 Responsibilities

The REST API is responsible for parsing the requested information to the server and fetching the necessary information.

6.2.3 Subsystem Interfaces

The REST API wont have any display directly to the user but rather it will be the "middel man" between the client and the server. This will be done by taking in JSON responses and encoding it, then sends it to the server for its response. Then it receives an encoded message from the server with the appropriate information and then decodes it and then the client side displays that data.

Table 8: REST API Subsystem interfaces

ID	Description	Inputs	Outputs
#2	Receive/Send information via JSON through REST API calls	JSON message sent to Client's REST API for decoding	Encoded JSON message sent to OpenEMR's REST API
#6	Logs Client activity once client is authorized	JSON message sent to Client's REST API for decoding	Sends message over to log controller to record activity

7 Log Controller Layer Subsystems

This subsystem includes the log poster and it's corresponding log receiver.The Log Controller records actions that the client has done by recording the REST API requests.It is responsible for documenting all GET and POST requests that a user has made. This will then be formatted and sent over to the database for record keeping.

The main aim of having the log controller is to keep a record of any and all the confidential patient information the medical professional or administrator had access to while using the application.Along with the safety requirements, it also helps with the legal front in case of any alleged HIPPA violations.

7.1 Log Poster

The log poster is the part of the subsystem which receives a JSON encoded message from the log receiver whenever a user makes a GET or POST request in order to prepare for the post.The JSON encoded message is then sent to the database for record keeping as mentioned earlier.

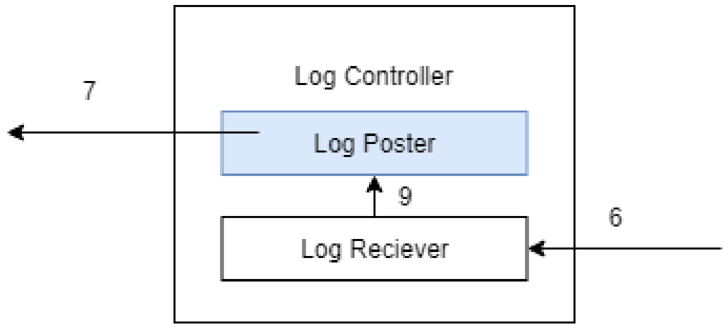

Figure 10: Log Poster subsystem diagram

7.1.1 Assumptions

There is no hardware failure during the process.

7.1.2 Responsibilities

The log poster is responsible for receiving the JSON message from the log receiver and then transferring it to the database where the message is added to the current records.

7.1.3 Log Poster Subsystem Interfaces

The inputs for this subsystem will be the JSON messages from the log receiver that will be collected whenever the user makes a request.The output will be a JSON message sent to the database.

Table 9: Log Poster Subsystem interfaces

ID	Description	Inputs	Outputs
#9	Receives JSON encoded message from Log Receiver to prepare for post	JSON message from Reciever	Send to poster
#7	JSON message is posted to database for record keeping	JSON message from poster	Send to database

7.2 Log Receiver

The log receiver is the part of the subsystem which receives a JSON encoded message whenever a user makes a GET or POST request in order to send the message to the log poster.The JSON encoded message is then sent to the log poster which then follows the aforementioned method to send it to the database for record keeping.

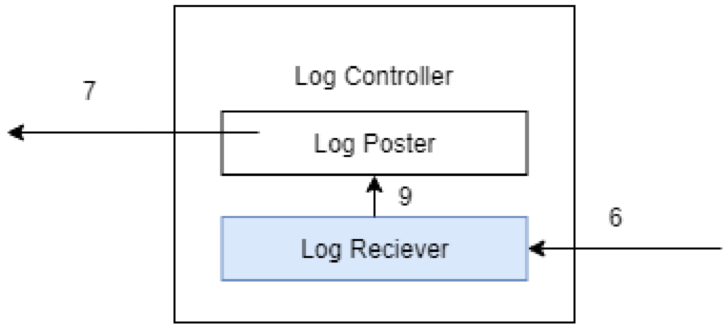

Figure 11: Log Receiver subsystem diagram

7.2.1 Assumptions

The data received from the REST API(client side) is not corrupted.

7.2.2 Responsibilities

The log receiver is responsible for receiving the JSON encoded message from the client side, sending the message to be posted and then relaying it to the log poster.

7.2.3 Subsystem Interfaces

Table 10: Log Receiver Subsystem interfaces

ID	Description	Inputs	Outputs
#6	Receives JSON encoded message from Client's activity	JSON message from Client	Send message for posting
#9	Receives JSON encoded message from Log Receiver to prepare for post	JSON message from Reciever	Send to poster

8 Apache Web Service Layer Subsystems

Apache web server allows our to host our software in an effective way. The web server will be hosting all out client side application and will be responsible to send requests from the client side to the database along with posting the requested information to the client side.

8.1 Server Host

The Apache HTTP Server, colloquially called Apache, is a free and open-source cross-platform web server software, released under the terms of Apache License 2.0. Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. We will be using this server to host our application.

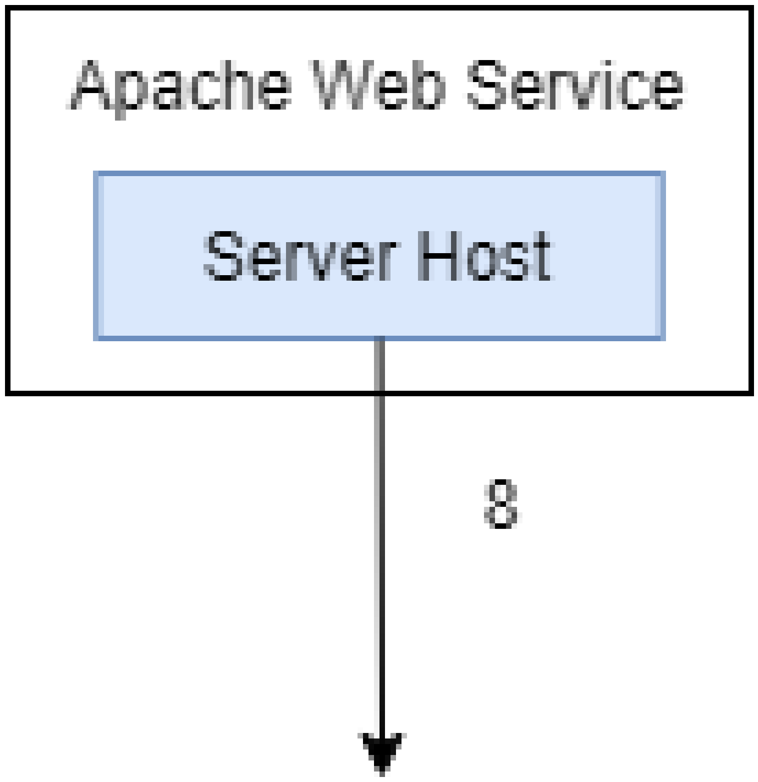

Figure 12: Server Host subsystem diagram

8.1.1 Assumptions

We are making an assumption that the server won't be creating any copies of data without out knowledge as data privacy is our primary concern here.

8.1.2 Responsibilities

The main responsibility for this sub system will be to host out software and to communicate with the OpenEMR API keys.

8.1.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here. Create a table with an entry for each labelled interface that connects to this subsystem. For each entry, describe any incoming and outgoing data elements will pass through this interface.

The Apache web server will be the primary use of hosting the OpenEMR software

Table 11: Server Host Subsystem interfaces

ID	Description	Inputs	Outputs
#8	Server allows hosting of OpenEMR software	N/A	Server Hosting

References