CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Implementation of mHealth Monitoring System using IoT Enabled Cloud Computing

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science in Computer Engineering

By

Beulah Preethi Vallur

December 2017

The graduate project of Beulah Preethi Vallur is approved:

Dr. Jeffrey Wiegley	Date:
Dr. John Valdovinos	Date:
Dr. Shahnam Mirzaei, Chair	Date:

California State University, Northridge

ACKNOWLEDGEMENT

I would like to thank my Project advisor Professor Dr Shannam Mirzaei, and the project committee members Professor Jeffrey Wiegley and Professor John Valdovinos and Dr Shahram Mirzai, for their constant support and guidance throughout my work. I am blessed to have such a talented crew of outstanding professors in my project committee.

A personal note of thanks to committee chair and project advisor Professor Shahnam Mirzaei for his patience and guidance in the right direction; his continued participation, input and constant feedback helped to successfully completion this project. He has been a sincere advisor and an inspiring strength throughout this work.

I would also like to thank Krishna Murthy Kattiyan Ramamoorthy for mentoring, especially, Dr Mirzaei and Krishna, have been extremely generous with their time and rendered me all possible help to see this work complete.

Finally, I would like to thank my family members who have always been there with their endless love, support and blessings.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

CBF Track: Implementation of mHealth Monitoring System using IoT Enabled

Cloud Computing

By

Beulah Preethi Vallur

Master of Science in Computer Engineering

This project presents a novel application of cloud computing enabled by internet of things (IoT) in monitoring parameters affecting cerebral blood flow (CBF) which is the movement of blood through the network of cerebral arteries and veins supplying the brain. The example design implemented in this proposal can be easily replaced with similar applications to generalize the concept offered in our work. This enables healthcare professionals to have pervasive access to the processed medical data (in this case cerebral blood flow) of patients during their hospitalization process.

Our scheme proposes a design in which the cerebral circulation data is captured using sensors connected to Raspberry Pi and then pushed to the cloud, stored in database, later processed, and analyzed. Results then will be retrieved and distributed to medical professionals via Android mobile application. This application is designed to keep track of cerebral circulation and process the data obtained through the sensors.

Anomalies such as oxygen imbalance, internal bleeding, swelling due to an increase of water, and disturbance in blood flow that can lead to serious health issues can be detected. We have used Amazon web services (AWS) cloud platform to perform cloud services.

Our approach is inspired by Amazon Simple Beer Service (SBS) [10]; a cloud-connected kegerator; that sends sensor data (beer flow and in our case cerebral circulation data flow) to AWS [5]. SBS publishes sensor data collected by an IoT enabled device (Raspberry Pi) to an AWS application program interface (API) gateway over Hypertext Transfer Protocol Secure (HTTPS). So, all the medical data registered with the application is stored securely in the AWS cloud using DynamoDB as the database and the raw data get processed inside cloud and activate real time system alarms based on the abnormal variations in vital parameters to medical professionals via android application. To the best of our knowledge this is the first scheme offered to replace the manual process of monitoring CBF using biomedical electronic devices.

CHAPTER 1

INTRODUCTION

There is a bulk amount of healthcare information generated every day. The data are essential and vital for decision making and hand over the best care for patients. Cloud computing is a cost-effective method that promote the real-time data collection, data storage and exchange between medical care organizations. Cloud infrastructure is characterized with a high throughput and a high-volume storage; These two important factors are helpful for adequate data analysis of large patients' population. Security and confidentiality are of the chief interests for consuming cloud-based healthcare services.

Healthcare organization should have electronic medical records to use the cloud infrastructure. To deal with the speedy advancements in information technology and the utilization of cloud based services, labors should be dedicated to move healthcare data form the traditional paper based to the electronic format. Then, regional legislation and policies should be passed to sanction and administrate the usage of healthcare data.

This project is an effective and stable implementation of a software package that is integrated with sensors, cloud and mobile computing devices utilizing Android operating system. Implementing a crucial challenge in patient healthcare by

leveraging the power of Internet of Things and cloud communication and processing in the Information Technology world by discovering the advantages of current trends in technology. It shows the steps towards the health informatics revolutionizing the patient health and healthcare systems around the globe. This application shows one of the novel implementation to improve the quality of health care data processing inside the cloud.

In a few years, cloud computing would become the center of the internet technology. Presently, there are many factors such as cost constraints stopping from the need to do high quality work with fewer and additional overpriced resources. Expectations for higher quality treatment from the healthcare services increase the need for point-of-care access to medical data and adoption of mobile devices; both for medical staff and patients; These requirements compel the need for internet technology (IT) systems to adapt [1].

Furthermore, the substantial evolution in digitization of medical records and the increasing rate of digital outputs from scanning and monitoring devices, such as magnetic resonance imaging (MRI), and computed axial tomography scan (CT-SCAN) maximize the possible benefit of cloud solutions. With usage of cloud we can use renewed software, do more with fewer data center usage, flexible costs and pay for what we use, tremendously reliable, enhanced collaboration and mobility, expenses can be quickly reduced, flexible capacity, very supportive for migrating

and withdrawing the applications. Which fulfills the need of providing more timely access to critical medical data. In this project, attention is given to possible implementation of cloud computing technology in the medical field. [1]

Especially Therapeutic concepts in neurosurgical severe care requires advanced and refined neuromonitoring applications, because the personalized approach to the treatment of head-injured patients relies on the assessment and interpretation of the vital parameters of brain tissue feasibility and function. Monitoring of Regional Cerebral Blood Flow (rCBF) has become a long-term trouble due to the life-threatening need of continuous bedside online monitoring. [15]

In Neurosurgical practice, monitoring of CBF plays a vigorous role, as the brain depends on nonstop blood supply due to its inability to store glucose or oxygen. During Pathological conditions of Patient, the operations of Brain needs to administer continuously. Under these conditions, rCBF is measured as an imperative upstream monitoring parameter, so in neuromonitoring strategies need supplementary refinement, in continuous monitoring of Intra Cranial Pressure(ICP) and Mean Artery Pressure(MAP) to discovery the Cerebral Perfusion Pressure(CPP) and to effectively identify the mal-perfusion in the brain-injured patient. In constant monitoring of rCBF could provide the opportunity to detect and

to correct insufficient rCBF before shortfalls in tissue oxygenation and metabolism are recognized.

There is a critical need for a huge IT platform where patient health can be tracked and monitored through the mobile devices around the world anytime anywhere [21]. Evolving health informatics get augmented results. Due to technology progressions over time, personal health monitoring devices and systems have gained popularity and are easily reachable to users. With Cloud available to every individual globally, a rapid cloud processing, controlling along with IoT wireless capabilities could be combined with protected data storage capacity and real time monitoring to develop a cloud application for the medical industry that will integrate patients, physicians and medical staff together. This application aims to strike this cause and track a critical patient's data, speedily and process efficiently inside the cloud with abiding the time and quality constraints in consideration and display the processed data via Line graphs in visualized representation, with timely alarms and notifications, and transmit the data securely to cloud and from there to mobile devices within less span of time.

The following is the contribution of our prototype application:

- Replacing traditional healthcare mechanisms with digital options by leveraging cloud computing technology.

- Exploiting Cloud Computing and IoT technologies to accelerate the accrual of real time medical knowledge and data to healthcare professionals and research community for practice improvement and remote care
- Presenting a new productive and cost-effective business model, and more efficient healthcare systems

The rest of this project is organized as follows: Chapter 2 describes the related work carried out by other medical researches and software developers. Chapter 3 presents the background to our implementation that includes an introduction to cerebral circulation and the parameters of interests. Furthermore, it covers the basics of cloud computing, its general architecture, and cloud computing web services and tools, the cloud software development process and the platform chosen for development, technical aspect of this system and its implementation and elaborate on future work. Chapter 4 illustrates the design flow and development of the cloud application. Chapter 5 discuss the user interface application development. Chapter 6 discusses the system functionality, a working prototype and use case scenarios for the application. Chapter 7 describes the application's architecture and design. We will conclude the project work in Chapter 8. Chapter 9 includes the Source code of this application.

CHAPTER 2
RELATED WORK

As the health industry needs are evolving hastily over time, fetching cloud based technology to the world of healthcare is very helpful and has many advantages. There are extensive range of systems that use remote monitoring systems for healthcare applications. There are systems used for management like Salesforce Health Cloud [22], which is a new patient relational management System with which doctors and healthcare providers effectively manage the health of patients across caregiver networks and associates data from numerous sources such as — electronic medical records, biomedical devices, even wearables—into a solitary location. Additional applications such as Cardiovascular healthcare services provider Boston Hear Diagnostics turns to UX design and UI development on a Microsoft stack to make their cloud apps addictive to patients by coalescing medical diagnostics and lab test data with personalized nutrition plans and lifestyle programs, occupied to transformation the way healthcare providers and patients inter communicate about heart health by providing online Web and mobile applications featuring a mobile application user interface design that patients and medical professionals find pleasurable to use and revisit regularly[23].

There are several cloud and smartphone applications in the market that tracks patients' blood- glucose levels and blood-pressure readings, or diet nutrition and

fitness apps that track calorie intake and workout routines. Constant Self-Monitoring Blood Glucose (SMBG) has been recognized to be a valuable tool in enlightening glycemic levels in type-2 diabetic patients [24].ECG-Remote Patient Monitoring using Cloud Computing[25], which provided end user with visualization of their Electro Cardiogram Waves(ECG).Healthcare Monitoring and alerting system using Cloud computing[26], which keeps track of patients basic vital parameters such as heartbeat, temperature and eye blink and transfer data to cloud and accessed by doctors via android mobile app.

Though the idea of building the medical application using IoT and Cloud computing is not totally new, there is a lack in building such a system for mission critical data monitoring. There is no application that caters to the need of keeping track of emergency patient's critical vitals parameters such as Cerebral Blood Flow at the same platform. However, cloud processing was not yet being fully utilized by medical applications. The application developed as a part of this project attempts to utilize this technology, however, is different as it keeps track of patient's critical vital data and uses cloud to process and alert medical experts via smartphones with visual line charts.

Intellectual Electronic Medical Record (EMR) application [27] developed for iPads and is presently used by doctors to effortlessly get patient's medical antiquity, medical problems, and all the updated info since the patient's last visit. This system,

named Wand, is developed as a web application using D3.js and Apache Cordova, and machine learning provided through an internal Web service API. It will aid physicians examine the patient's chart before checkup, apprise patient history during a consultation, fill-up electronic prescriptions, while rounding in the hospital or review the next day's appointments from anywhere. Wand supports the physician's workflow everyplace, and anytime. The concept of patient data gathering and utilization of analytics and visualization to help physicians better realize their patient needs is used to develop the project by research. However, the project is different from Wand as it is not an EMR application. However, it uses the patient data to track their medical routine. And is useful to medical staff who can now keep a live check on their patients and their emergency status.

There are wide range of instinctive advantages with Cloud Computing for Healthcare Organizations such as cloud makes Electronic Records far easier to archive and use patient records, medical image, access the data anytime if connected to an internet source, bulk storage capability to store massive range of data, apart from this cloud has strong security coverage with consideration of privacy and integrity. And presenting Streamlined Collaboration for doctors and medical staff, especially many physicians find cloud computing makes it easier to work together and offer care as a crew from remote location. Through numerous ways such as mobile devices, video conferencing, and applications built precisely for health care organizations, with cloud we can rapidly things up that permits better

communication at a remoteness. Patients get the instant assistant from expertise medical professionals whenever they need. Rural care and calamity response become more naturalistic. Cloud allows providers to save money by lessening in-house big data storage needs, more reachable from countless locations, and even if something happens on-site, the data is still well-maintained in cloud. Accessing high power-driven analytics.

Health care organizations can amalgamate these technologies and easily share industry data to generate even more composite systems. Cloud allows a lot of high-powered data solutions to superpower the investigation process with cutting-edge clinical research system such as innovative computing power of the cloud, using these massive data sets for growth becomes a certainty which helps in easily development of new drugs, and it especially presents interesting possibilities in DNA sequencing. Telemedicine Capabilities such as higher-tech devices, and mobile technology, providing health care from distant has become a realism such as giving instructions via video and audio conferences, tele-surgeries, and monitoring patients without having them come in [28].

Connecting healthcare industry [29], is becoming a certainty. Medical Centers are using IoT solutions to capture and evaluate data, make sensible decisions and finally saving money and millions of individuals lives around the world each day. But there are not the only ways how IoT can improve healthcare. Nowadays

management of drugs is one of the major expenditures in the healthcare industry. Internet of Things and smart devices could aid to manage this things in healthier way with boosted administration of drugs. Enlightening disease management by access to real time data, medical analytics and smart nursing. Diminish the costs by improving real time medical data analytics, drug management, and cut down unnecessary medical personnel appointments. Automated data and smart workflow measured by devices connected by IOT and decisions makes based on profound analytics that helps to reduced errors.

A cloud based asset administrating application named EZ Office Inventory [30] tracks assets speedily and precisely using an Android or iOS device. Since it is a cloud based service, all the information is reachable at any time with wide-ranging broadcasting through a web portal. Beyond catching basic asset attributes and location information, it has inordinate feature to send reminders about these assets. Being in the cloud, it takes no time for any customer to get this application up and running because of fast ramp-up time provided by the geographically disseminated cloud networks having high elasticity, accessibility, load balancing, and competent delivery chain.

This project leverages this idea that smartphones can be used to send notices and warnings to physicians and medical staff. If critical data accessing is integrated into cloud and the whole computing process will take place in cloud, along with the

superior system they can also be used for services like notifications. The mobile application implemented in this project is just fetches the processed emergency data and the cloud is the one that keeps track of vital facts for patient health assessment and regulation. Our Android is GUI that will be used for display.

In Neurosurgical practice, monitoring of CBF is a labor-intensive physical process as of now [15], [20]. There are several CBF measurement techniques accessible. However, our CBF monitoring application is an enhanced prototype solution and integration of Cloud, IoT devices (Raspberry Pi and Android Mobile Phone) and automation of the process. And delivers the utility to send notification to physicians and medical staff which would solve a lot of difficulties that patients in emergency care face while getting their medicine assistance when needed by nonstop bedside monitoring as well as helping the medical industry to serve the patients constantly.

CHAPTER 3

BACKGROUND

3.1 Cerebral Circulation and Related Concepts

Brain is a unique organ of the body that totally depends on the blood supply for its survival. Cerebral circulation is the movement of blood through the network of cerebral arteries and veins supplying the brain as shown in Figure 1.

Figure 1: Cerebral Blood Flow(CBF)

Changes in cerebral blood flow (CBF) can lead to several neurological disorders including stroke. Hence it is important to monitor the CBF for the diagnostic and therapeutic purposes. Human skull, can be considered as a bony box of fixed

volume that contains brain, blood and cerebrospinal fluid (CSF). Unlike other body parts, brain can resist for only a very short period lack of blood supply for about 3 to 8 minutes approximately [2]. Therefore, CBF must be maintained to guarantee a relentless delivery of oxygen and substrates and to eradicate the waste products of metabolism. Subsequently, there is significant interest in monitoring CBF in patients with miscellaneous brain diseases such as traumatic brain injury (TBI). CBF is reliant on on numerous factors that can be generally divided into two categories:

1. Those disturbing cerebral perfusion pressure
2. Those disturbing the radius of cerebral blood vessels

This association can be described by the Hagen-Poiseuille law [3] which describes the streamlined flow of a homogeneously viscous fluid through a cylindrical tube:

$$Q = \frac{\mathrm{dV}}{\mathrm{dt}} = \frac{\Delta \mathrm{P}\ \pi\ R^4}{8\ \eta\ L} \qquad \text{Eq. (II} - 1)$$

In our case, Q is the CBF, ΔP is the cerebral perfusion pressure (CPP) or pressure drop along the vessel, R is the radius of the blood vessel, η is the viscosity of the blood, and L is the length of the blood vessel. CBF choices from 20ml/100g/min to 70ml/100g/min for human brain [2]. CPP is reliant on on the pressure incline between the arteries and the veins where arteries carry oxygenated blood and veins carry deoxygenated blood. This is the alteration amid of mean arterial blood pressure (MAP) and mean cerebral venous pressure. The last is tough to measure and approaches to the more effortlessly measured intracranial pressure (ICP).

$$CPP = MAP - ICP \qquad \text{Eq. (II} - 2)$$

CPP can be impacted by whatsoever that deviates the MAP or ICP. The effect of decline in MAP and growth in ICP simultaneously can be disastrous with the jeopardy of brain ischemia. For illustration, blood loss roots reduction in MAP while an intracerebral hematoma will surge ICP which can outcome in losing consciousness.

In a normal brain, despite the potential for changes in MAP, CBF remnants persistent over a wide range of CPPs attained by a procedure called auto-regulation. CPP is associated to CBF and is changeable through its association with MAP and ICP [4]. Consequently, CPP has been the subject of noteworthy research efforts as a factor persuading outcomes in numerous brain pathologies, most remarkably TBI.

Many retrospective series and statistics bank studies have chosen the invasive ICP monitoring technique [6, 7]. Wilson et al. [8] demonstrate how dissimilar types of ICP sensors can be implanted into brain in a hostile technique. Kang et al. [9] report materials, device architectures, integration strategies, and in vivo demonstrations in rats of implantable, multi-functional silicon sensors for the brain, for which all the integral materials certainly resorb via hydrolysis and/or metabolic action, eradicating the need for extraction. In their experiments, insulated percutaneous wires connect to a visibly mounted, miniaturized wireless potentiated for data communication where constant monitoring of ICP and temperature demonstrates functionality crucial to the treatment of TBI.

This project presents a method based on cloud computing to seize medical data from the sensor using Raspberry Pi and push data to the cloud to perform analysis and finally broadcast the result to medical professionals via an Android application in real time.

There are numerous CBF measurement techniques available, such as stable xenon-enhanced computed tomography(sXe-CT), single-photon-emission computed tomography(SPECT), magnetic resonance imaging(MRI) and positron emission tomography(PET); However, these techniques are hindered by numerous clinical and hands-on drawbacks. While these methods can provide regional evidence about CBF, the information provided is a single snap shot in time. Intensive care of rCBF in neurosurgical intensive should ideally be achieved in a constant way at the bedside, providing quantifiable rCBF values.

Procedures for the incessant measurement of CBF have been explored and are now commercially accessible; Laser Doppler Flowmetry(LDF) and Thermal Diffusion Flowmetry(TDF) based measurement techniques provide continuously beside monitoring of Doppler's principle is the most common principle applied by the existing biometric sensors [12]. They have a benefit of producing high frequency signal however, the signal is effortlessly used and can be used for flow velocity assessment only. Yet another regularly used expertise is measurement of absorbance of oxygenated and deoxygenated hemoglobin.

The only disadvantage in this method would be ambiguous intracranial contribution to signal. For more accurate measurements, thermal diffusion principle is used. With thermal perfusion probe, which is sited intra-cerebrally via a burr hole in the vascular area of interest in the brain. The probe is coupled to a monitor that displays CBF statistics [20]. A Doppler sensor can be integrated to a wireless

communication protocol such as XBEE or NFC (Near Field Communication). These modules can be constructed to regulate the MAP and ICP algorithm.

3.2 Cloud Computing Basics

Cloud Computing is a model of retrieving a shared pool of computing resources such as storage, services, networks, and applications on demand, without worrying about how these resources are made available for the end user who will be going to use them. These resources can be vigorously configured depending on the need or the rate of consumption and released when not needed. This permits organizations to avoid large up-front infrastructure costs. Furthermore, it enables firms to focus on their core businesses. Figure 2 shows a typical cloud architecture. The cloud architecture typically consists of a frontend platform or client interface such as mobile devices or internet browsers, back end platforms such as servers, a cloud based delivery, and a wireless network such as internet.

Cloud client can be a computer hardware or software. Users can access the data stored in cloud using network enabled devices such as tablets or smart phones. These cloud clients depend on cloud computing for all or majority of their applications.

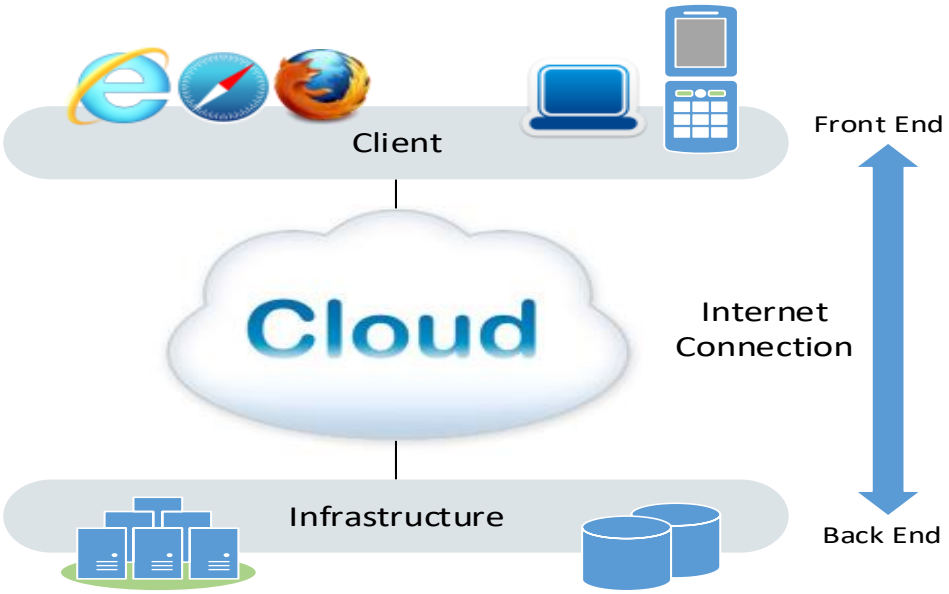

Figure 2: A typical cloud computing architecture

Cerebral circulation information is pushed to the cloud using Raspberry Pi microcontroller and is stored in a cloud database to be processed and analyzed. The result of the analysis is available to medical professionals via mobile application. Typical stages of high level data flow are shown in Figure 3. Initially we push the raw medical data to the cloud using Pi. Data will be analyzed and processed afterwards, and outcomes will be available to medical personnel and the patient. Several medical algorithms can be incorporated in the cloud to monitor the pattern of blood circulation in skull. These algorithms can be used to regulate the risk involved in the patient developing neurological disorders. If the risk is above a certain threshold, the doctor is immediately notified by the cloud service by sending text messages or invoking SMTP (Simple Mail Transfer Protocol) to generate a customized e-mail with vital medical information.

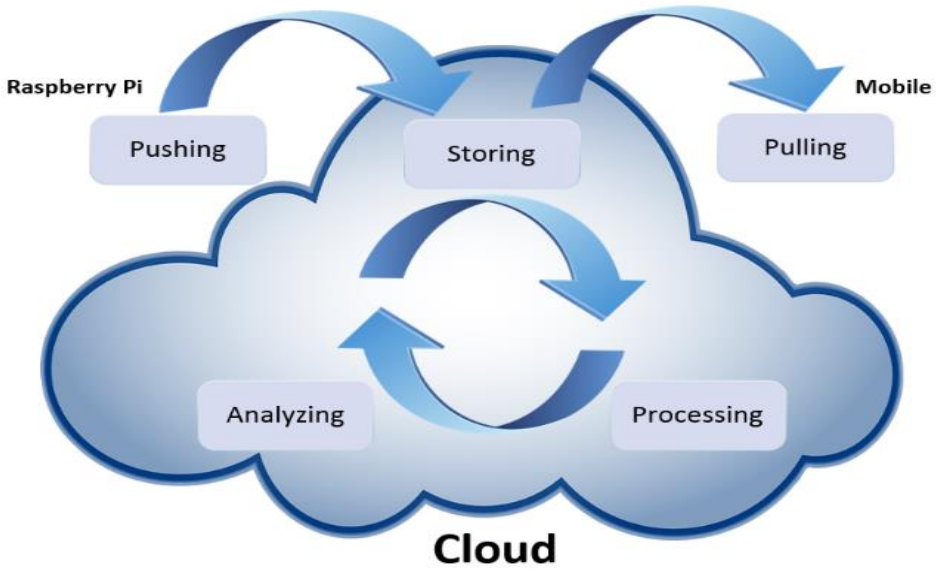

Figure 3: High level data flow push, process and pull with cloud

Information at initial stage is raw meaning it needs further processing to transform into information, knowledge and later wisdom. Figure 4 demonstrates Ackoff's DIKW[13] (Data/Information/Knowledge/Wisdom) pyramid. This figure describes the process required till raw data transforms to some meaningful and applicable wisdom.

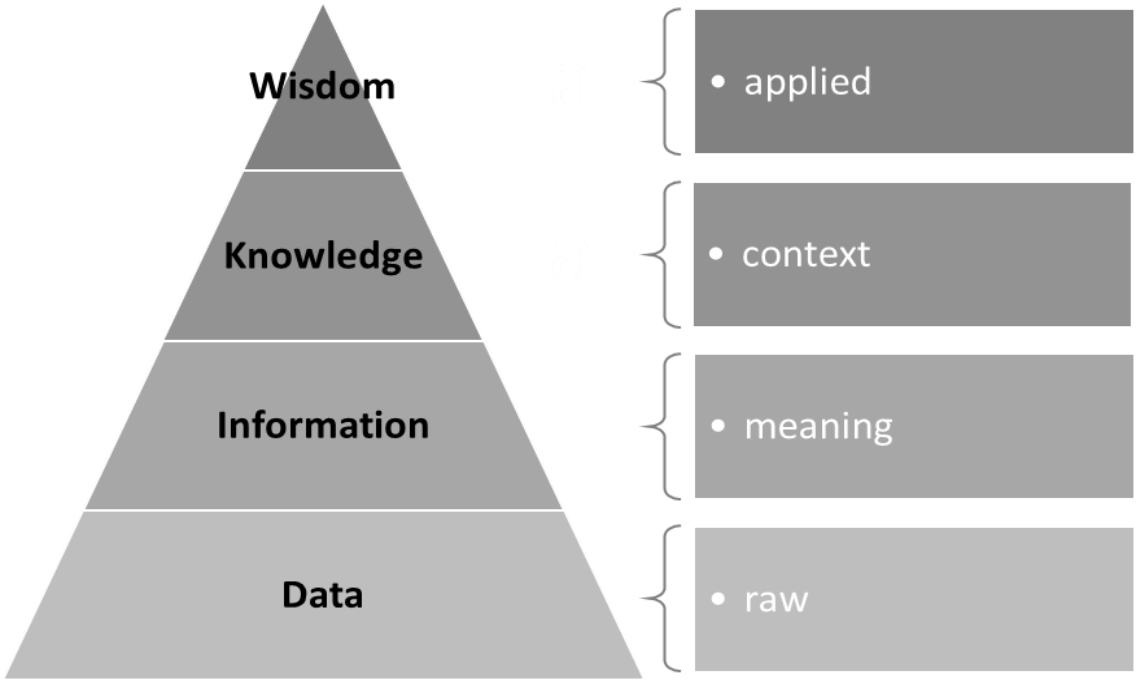

Figure 4: Ackoff's DIKW pyramid

In former schemes of IoT systems, processing data was performed after collecting data by the sensors and the result was sent to the cloud. Cutting-edge analytical algorithms were applied on raw data to extract meaning and context out of raw data. However due to progression in cloud computing technology in modern systems and high level of integration of processing units into sensors, these steps can vary in order, offering more system performance and flexibility. This will let the developers perform some of the data processing at later stages in the cloud after data is transmitted to the higher levels of network. Using this method, a higher level of knowledge can be transmitted to the hub for post-processing. Figure 5 compares these two schemes. While in central scheme data interpretation is done in the last level; e.g. cloud; distributed scheme tends to refine data before

transmitting it to the cloud. This would offer higher flexibility and simpler hardware within sensor nodes or hubs.

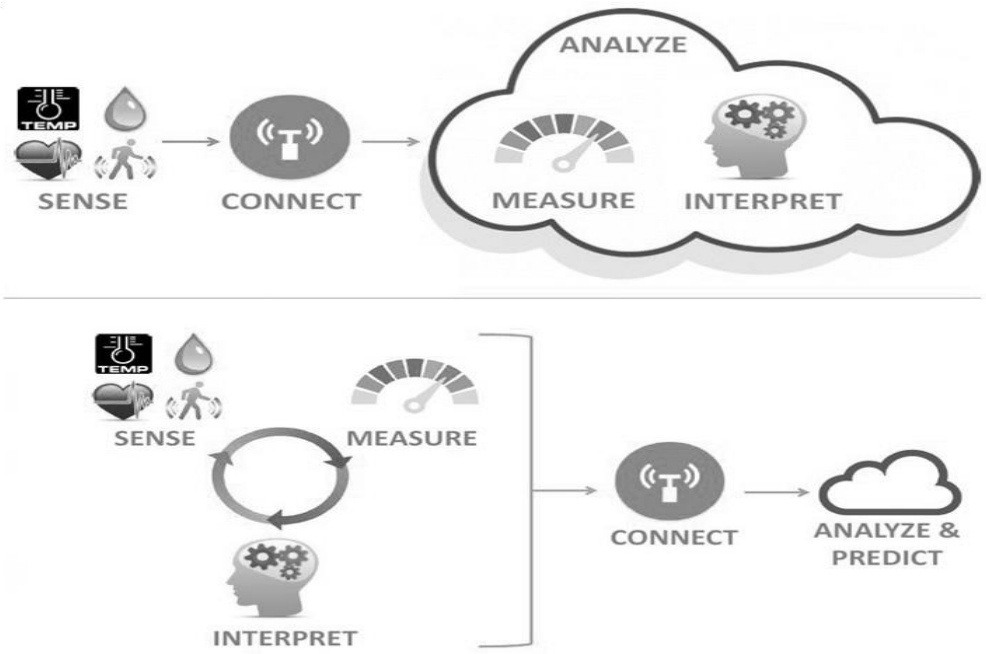

Figure 5: Central (top) vs. distributed (bottom) interpretation

3.3 Big Data Frameworks and Processing

In providing more timely access to critical medical data. In addition, a cloud computing system is a measured service that can analyses and control resource usage concerning the type of service provided e.g., storage and processing. In healthcare organizations, there are mammoth amounts of data composed and stored each day. This information is important and vivacious for decision making and for delivering the suitable treatment. Healthcare organizations need to use cloud computing systems where patients' data are stored, retrieved and shared with other

healthcare providers. Since the number of the healthcare data is continuously increasing, it will eventually affect the storage capacity of healthcare data servers

Medical institutions, scientific research labs and hospitals are changing to big data analytics to alleviate healthcare costs, predicting epidemics and enhancing the quality of human life by altering the models of treatment delivery. With the rise in the number of health records of millions of people to tens of billions, computing technology and infrastructure essential to condense a cost-efficient employment of data processing model. The novel model must deliver storage for billions and trillions of unstructured data sets, fault tolerance along with high accessibility of the system. Hadoop technology [14] is successful meeting these challenges by offering the capability to process thousands of terabytes of data. By the end of 2017 in collaboration with the American Biotechnology firm Illumina and Genomics England targets to map 100,000 human genomes to develop personalized medication for cancer patients [14].

In our development, continuous monitor, store and analyses of blood flow in veins and arteries of brain is achieved by collecting the data from sensors and pushing them to the cloud via Raspberry Pi using Wi-Fi. Rapid alerts can be sent to medical professionals was successfully achieved by using Hadoop ecosystem components, Apache Spark, Apache Hive and Storage Tools S3, DynamoDB, EMR and EC2. Further elaborates how these tools and web services are used in our implementation.

3.4 Web Services, Tools and CBF Cloud Architecture

It is critical to understand the design flow in this section before elaborating the tools used in our implementation. Our approach is inspired by Amazon SBS [10]; a cloud-connected kegerator; that sends sensor data (beer flow and in our case cerebral circulation data flow) to AWS [5]. SBS publishes sensor data collected by

an IoT enabled device (Raspberry Pi) to an AWS application program interface (API) gateway over HTTPS.

For our project, we propose the Raspberry Pi model B+ developed by Raspberry Pi Foundation [11] for interfacing with the cerebral data collection sensors. It leverages an ARM processor, provides an SD card for storage, and exposes several dozen GPIO pins to interface with sensors as well as Ethernet port. Figure 6 shows the high-level design flow incorporated into our proposed scheme using AWS and tools for cloud computing applications. The Python code running on the Pi polls the sensors and collects the result into a JSON (JavaScript Object Notation); a nominal readable format used primarily to transmit data between a server and web application. It is then transmitted to Amazon API Gateway by using EC2 (Elastic Cloud Compute) instance. EC2 is a web service that delivers secure, resizable compute capacity in the cloud. By connecting the created EC2 instance to a static IP address we are successful to permit communication with the internet. API gateway is used to generate, distribute, preserve, display, and secure APIs and fits effortlessly with IoT cases such as our experiment.

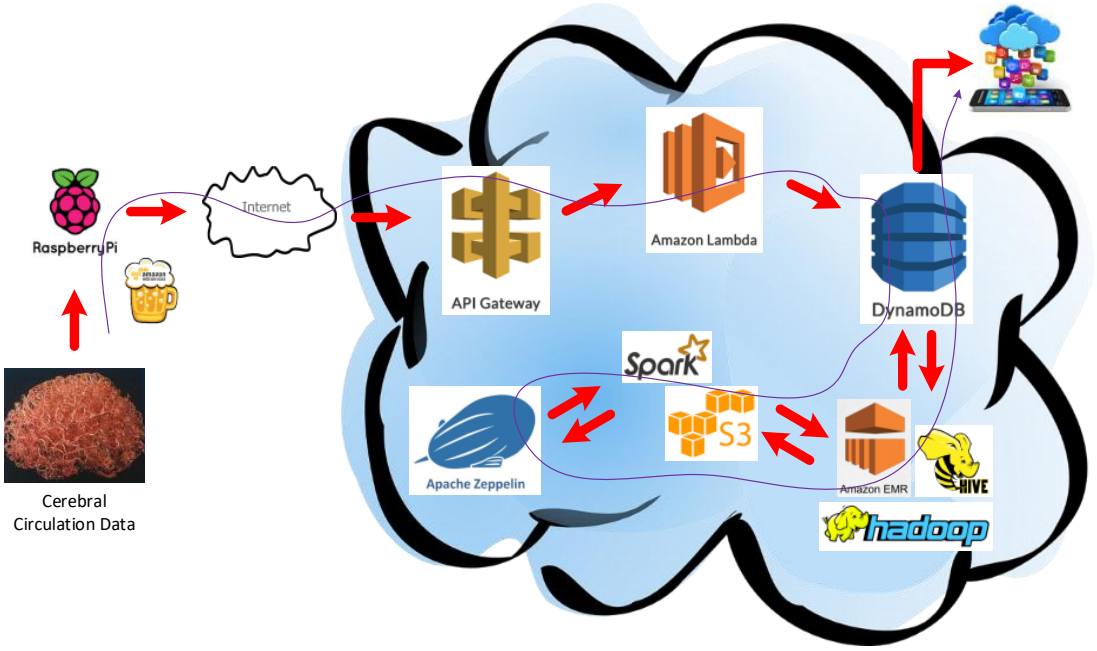

Figure 6: CBF monitoring IoT enabled cloud architecture

From API Gateway, the payload can yield two different paths: intermittent and speed. Since this is life-threatening information we select speed path. AWS Lambda function [5] helps the speed path and pushes and stores data into a table hosted by Amazon DynamoDB. DynamoDB is a low latency and high performance, non- SQL (NoSQL) based cloud database that provides a mechanism for storage and retrieval of data modeled in resources unlike the tabular relations used by relational SQL based databases.

AWS DynamoDB is a profligate and NoSQL database which delivers steadiness and single digit millisecond latency at any scale. It is a stretchy and a fully accomplished cloud database which provisions both document and key-value store models. As the information capacity grows and application demands performance rise, Amazon DynamoDB uses automatic partitioning and SSD technologies,

robotically spreads the information and circulation for the tables over sufficient servers; meeting the throughput necessities, sustaining consistency, and bringing low latency and high performance at any scale [17].

DynamoDB stores data as database tables, which are gatherings of individual element. Each element is a collection of information attributes. The elements are equivalent to rows in a spreadsheet, and the attributes are equivalent to columns. Each element is uniquely recognized by a primary key, which is composed of two attributes, called the hash and range. DynamoDB queries refer to the hash and range attributes of elements to access. Progressive Key-value information stores such as DynamoDB accomplish high scalability on lightly coupled clusters by using the primary key as the partitioning key to dispense facts across nodes. The query proficiencies are created on the default primary index and optional local secondary indexes of a DynamoDB table:

1. Primary Index: Two types of keys can be castoff for primary index querying: Simple Hash Keys and Composite Hash Key plus Range Keys. Simple Hash Key stretches DynamoDB the Scattered Hash Table concept. The key is hashed over the different partitions to enhance workload distribution. Composite Hash Key with Range Key permits to produce a primary key that is the composite of two attributes, a “hash attribute” and a “range attribute.” When querying against a composite key, the hash attribute needs to be uniquely matched, but a range procedure can be stated for the range attribute.
2. Local Secondary Index: Local Secondary Indexes permits to generate indexes on non-primary key attributes and rapidly recover records within a hash partition (i.e., elements that share the equivalent hash value in their primary key).

Global secondary indexes permit to proficiently query over the whole DynamoDB table, not just inside a partition as local secondary indexes, using any attributes (columns), even as the DynamoDB table horizontally scales. [18]

There's a significant dissimilarity between a "query" and a "scan" in DynamoDB. A query is a hunt by ID. All query-searches without the ID parameter and a hunt by other parameters outcomes in an exception. Searches grounded on parameters other than the ID are scan operations. An ID-based query is a lot quicker as each record is reliable to have non-null ID field(s) and the IDs are indexed. As there's no compulsory schema in DynamoDB a pursuit based on any property must inspect – or scan – each record and verify if it has that property [19]. A single Query or Scan request can recover a maximum of 1 MB of data; DynamoDB can optionally apply a filter expression to this information, narrowing the outcomes before they are returned to the user [17].

To prepare the information for further processing, information from DynamoDB tables should be exported to Simple Storage Service (S3) bucket. S3 [7] is used for object storage with a basic web service interface to store and regain any amount of data from anywhere on the internet. Detail of this process will be elaborated in Chapter 3. During this process, Amazon Elastic Map Reducer (EMR) [6] is used which manages the cluster platform by running big data frameworks such as Apache Hadoop and Apache Spark on AWS to process and analyze enormous amount of data. Amazon EMR maintains Apache Hadoop which is an open source, Java based programming framework that provisions processing and storage of bulky data sets in a distributed computing environment.

Using EMR, we can also track the distributed framework such as Apache Spark and intermingle with data storages such as S3 and DynamoDB. Apache Spark is an open-source, distributed processing platform used for big data workloads that ropes general batch processing proper for operations such as streaming analytics and graph databases used in our experiment. Finally, to perform mathematical calculations on data, we need to send the data to Apache Zeppelin notebook, a web

based notebook that empowers interactive analytics with using languages such as Scala, SQL, or IPython. The latter was used in our application development.

3.5 Android Platform

This is a custom-make easy to use Operating System, a Linux kernel mobile platform. The Android mobile operating system is reliant on upon the mobile device's processer capabilities for its performance. There are a few platforms available for the mobile environment: Android, iOS, Rim (Blackberry), Windows Mobile, and others. According to the Forbes latest survey report, the leading magazine, the Android platform globally somewhere looks like around 80 to 90 percentage [32].

3.6 Python

Python is a used high-level programming language for general-purpose software design. It has the characteristics of a dynamic type scheme and dynamic memory management and provisions numerous programming paradigms, counting object-oriented, imperative, functional programming, and procedural styles. It is also used as scripting language. It has a bulky and wide-ranging standard library. It stresses on code readability with the usage of whitespace indentation to delimit code blocks unlike of curly brackets or keywords like C#, C++, Java etc., and the syntax in python allows programmers to express acuities. It is Compatible with raspberry Pi's and delivers constructs intended to enable writing clear programs on both a small, medium and large-scale applications [30].

3.7 Android Studio

Android permits for Android app development within Android Studio. Android provisions add-ins that lets developers to build Android, within the IDE using code completion and IntelliSense. It also has extensions that support for the building,

deploying, and debugging of apps on a simulator or a device. Supporting different platforms such as Windows, macOS and Linux based operating systems

Gradle-based build support, android-specific refactoring and quick fixes, Lint tools to catch performance, usability, version compatibility, Pro-Guard integration and app-signing capabilities, template-based wizards to produce common Android designs and components ,rich layout editor that lets the users to drag-and-drop UI components, choice to preview layouts on multiple screen configurations, support for building Android Wear apps and built-in provision for Google Cloud Platform, enabling combination with Firebase Cloud Messaging and Google App Engine, Android Virtual Device (Emulator) to run and debug apps in the Android studio[31].

3.8 JSON

JavaScript object notation(JSON) is coolest way to recognize data exchange format which is autonomous of programming languages. Because it allows to load information rapidly. Asynchronous in back ground without much interruption. Made with key and value pairs enclosed in curly braces and Array of objects (as lists) stored in square brackets.

3.9 Table Schema

The application's backend is built in DynamoDB using the AWS Management Console. There are four tables in the database –Patient, Location, Physician and Medical Staff. Figure 7 displays the DynamoDB table schema.

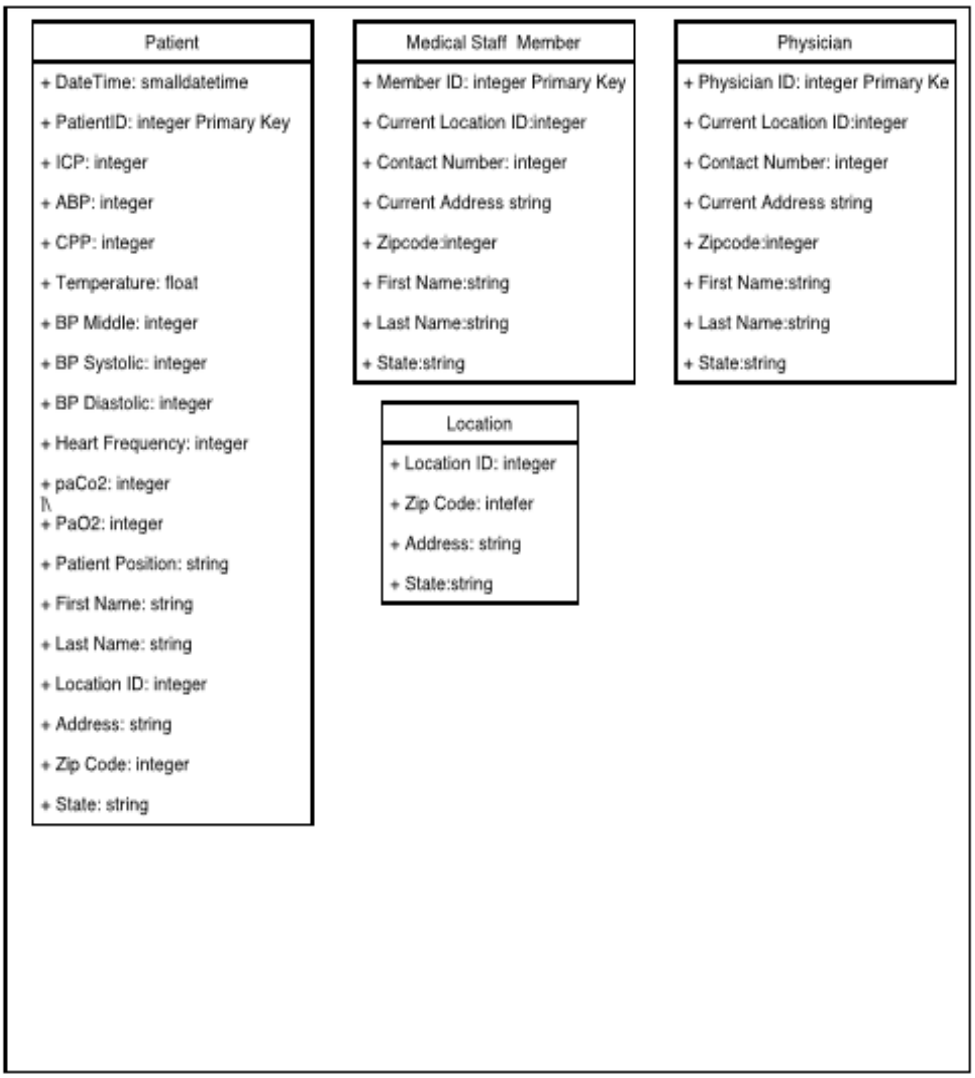

Figure 7: Table Schema of DynamoDB

Patient table embraces the profile of each registered Patient with in the application. It auto generates the PatientID which is unique for each patient and is the primary partition key for the table, and Last Name is the primary sort key for the table.

Physician table embraces the profile of each registered physician with in the application. It auto generates the PhysicianID which is unique for each physician

and is the primary partition key for the table, and Last Name is the primary sort key for the table.

Medical staff member table holds the profile of each registered staff member who can also have access to check the status of patient with in the application. It auto generates the Member ID which is unique for each physician and is the primary partition key for the table, and Last Name is the primary sort key for the table.

Location table holds the location addresses for hospitals and places of physicians with the info of Address, State and Zip code.

CHAPTER 4

DESIGN FLOW

Figure 8 shows the detail of our design flow with a summary of the tasks to be implemented at each step. Initially medical data (i.e. medical data such as MAP and ICP collected by the sensors connected to patient) will be pushed from Raspberry Pi via WiFi to the cloud by creating Amazon Elastic Compute Cloud (EC2) [5] instance that provides auto scalable computing capacity associated to our application. This connects our instance from Raspberry Pi and stores the data to the database table hosted by Amazon DynamoDB database.

To process the data in the cloud including mathematical calculations, we need to send the data to Apache Zeppelin notebook to perform interactive analytics. We can transfer the data from DynamoDB to Apache Zeppelin via Amazon Simple Storage Service (S3). At first, we will migrate data from DynamoDB database to S3 and store the input data into a S3 bucket. To be able to perform this process, Amazon DynamoDB integrates Amazon EMR. Amazon EMR allows us create tables that can store data natively in an EMR cluster or they can be mapped to information stored externally such as in DynamoDB and S3. Using EMR, we can transfer DynamoDB table data into S3 bucket or load data from S3 bucket to a DynamoDB table. This can be beneficial as multifaceted queries and data mining can be competently implemented on the native information.

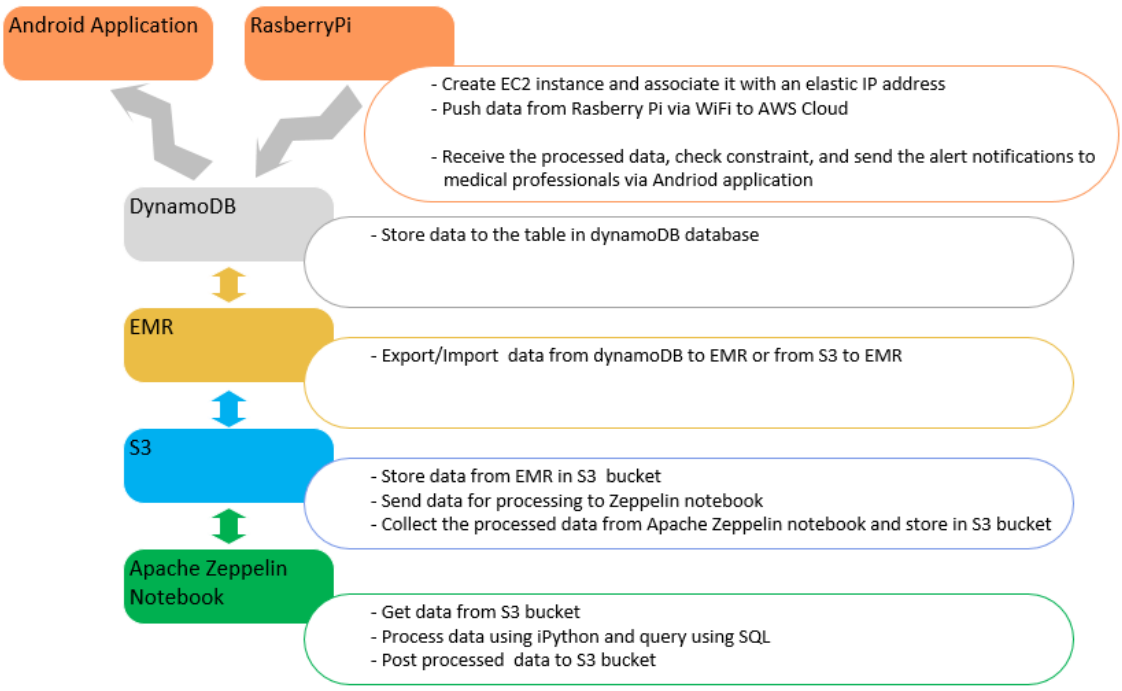

Figure 8: CBF monitoring cloud computing design flowchart

The process of exporting data from DynamoDB table to S3 bucket starts by generating two external hive tables as shown in Figure 9.

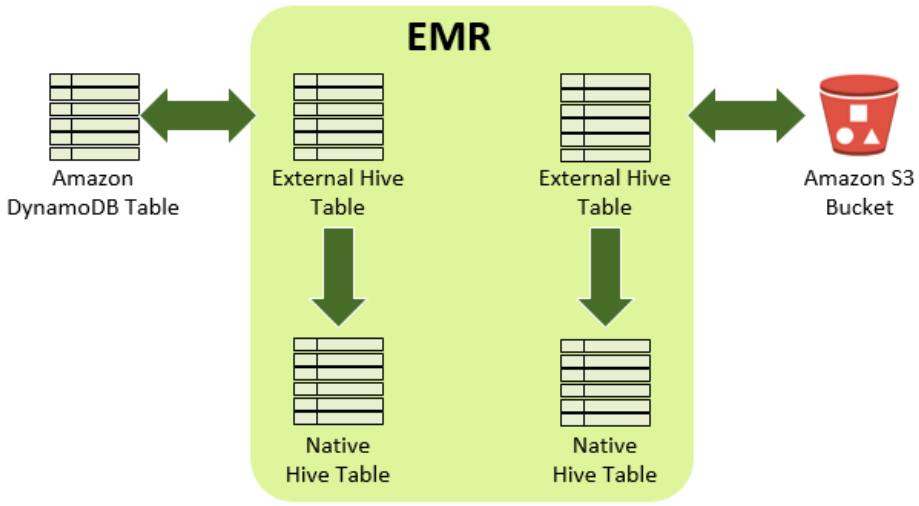

Figure 9: Export/Import process to/from DynamoDB/S3

One of the external hive tables maps to the DynamoDB table and the other will map to S3 bucket where information is stored. We use Apache Hive to process information stored in a DynamoDB table, S3 table, etc. Hive is an information warehouse application for Hadoop that permits to process and analyze information from numerous resources. Hive provides a SQL like language; also known as HiveQL; that permits working with facts stored locally in the Amazon EMR cluster or in an external information source such as DynamoDB or S3. Information is first delivered from DynamoDB to external hive table and then it will be sent to native hive table and from there to another external hive table and finally gets stored in S3 bucket. After copying the information to S3, we will generate another cluster known as Apache Spark.

With this cluster, we can use Apache Zeppelin to produce a notebook. We used IPython in Zeppelin notebook to transfer the information from the S3 bucket to notebook. Now data processing including mathematical calculations can be performed and the outcomes can be transferred back to DynamoDB through S3 bucket and hives using Zeppelin

notebook again. Using this scheme, information can be exported/imported to/from S3/DynamoDB very frequently. Furthermore, uploading periodic bulk of figures from another application can be done this way using scripts and schedules.

In our proposed scheme, constraints can be added to DynamoDB tables to verify if there are any changes in the patterns of CBF over a period or if a specific threshold limit has been crossed. In that case, proper alert messages can be sent to medical professionals via Android mobile application which is also developed by the authors. Graph charts of corresponding parameters such as CBF, MAP, and ICP can be displayed on mobile screens for better precision.

CHAPTER 5

ANDROID APPLICATION DEVELOPMENT

The user interface for the proposed design would be an application running on Android platform. The application was designed and developed using Android Studio integrated development environment (IDE). To integrate AWS with Android Studio, we have used AWS software development kit (SDK) for Android. We castoff ProGuard that has built-in provision for cloud platforms, permitting the user with Firebase Cloud Messaging (FCM). The processed medical data in the cloud will be drawn by the Android application by creating an API.

Developing a mobile based platform application would not only permit users to effortlessly access the medical information but also cuts down the expenditure involved in manufacturing the biometric device. The chief functionality of the mobile application is to launch a secure connection with the cloud and acquire the transmitted sensor values. The application takes benefit of the GSM (Global System for Mobile) communication module existing in the mobile device to send messages and make phone calls directly from the application itself. The implemented User Interface is shown in Figure 10.

Figure 10: Android emulator developed to display CPP data

As it can be seen in the application screenshot, the MAP and ICP values are captured by the sensors and pushed to the cloud using Raspberry Pi. CPP value can be calculated using our cloud computing algorithm and the results can be displayed on Android mobile device. Anomalies can be spotted by inspecting constraints deployed in the algorithm on the cloud. Once sensed, the alert notification can be sent to medical authorities by forwarding the values through Line charts. The developed code running on smart device can launch the connection with the cloud to read the information (outcome of the analysis) from the network and route it to the target device's.

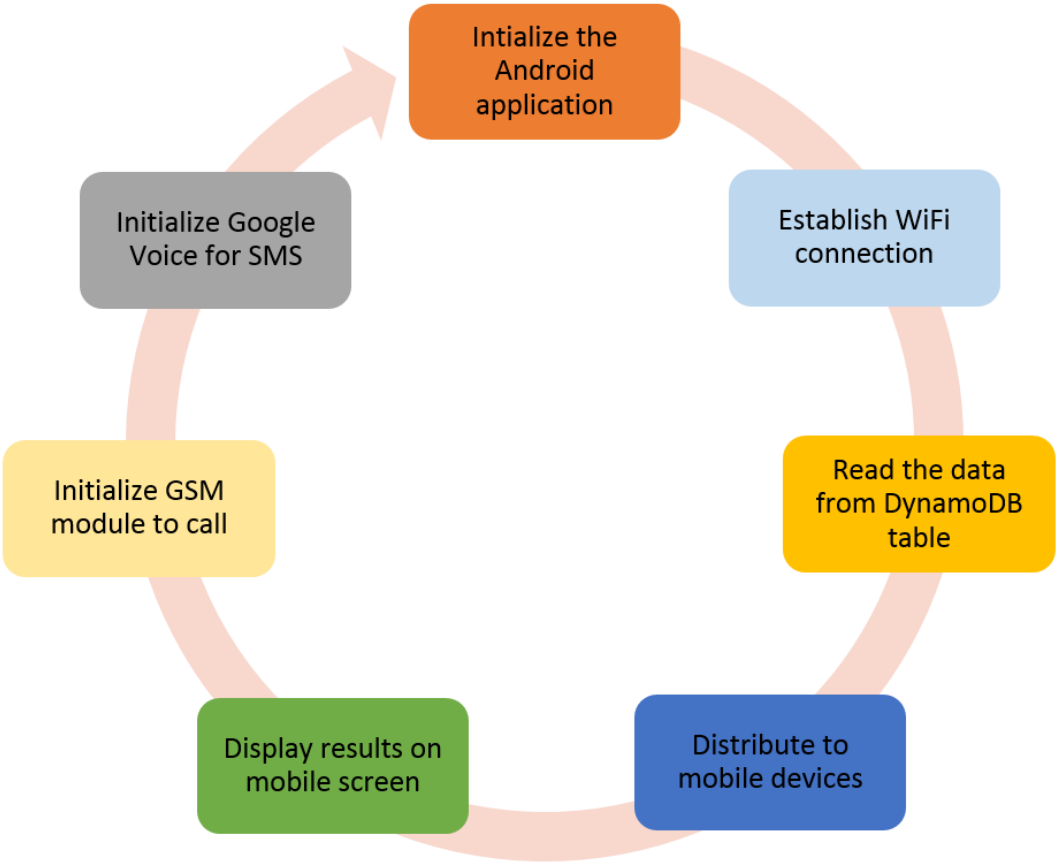

Figure 11: Android application flowchart

CHAPTER 6

SYSTEM ARCHITECTURE AND DESIGN

CBF Track is a cloud based emergency aided biomedical application that monitors and runs on cloud with an integration of IoT sensors and Android application as a user interface to notify the medical team.

6.1 System Architecture

CBF track cloud application is built on the three-tier software architecture which comprises of the User Presentation tier, the Business logic tier and the Database tier. Application's User Presentation tier is created for Android views, this is the user's interaction interface with the application where the user (in this scenario Doctors and medical staff) can view the info and receive messages or notifications from cloud in real time via pictorial line charts. The Business tier contains the request for processing and business logic of the application that is built on the Python classes, libraries like Boto3,JSON,uuid,Itertools,OS. The Database tier contains of the DynamoDB which is a NoSQL database on the Amazon Web Services cloud.

As the Figure 12 implies, in this system, client communicate with the Android presentation layer, views the information, and the business layer which processes the medical data, presents

the processed info to the user. The data is protected on the cloud using AWS DynamoDB.

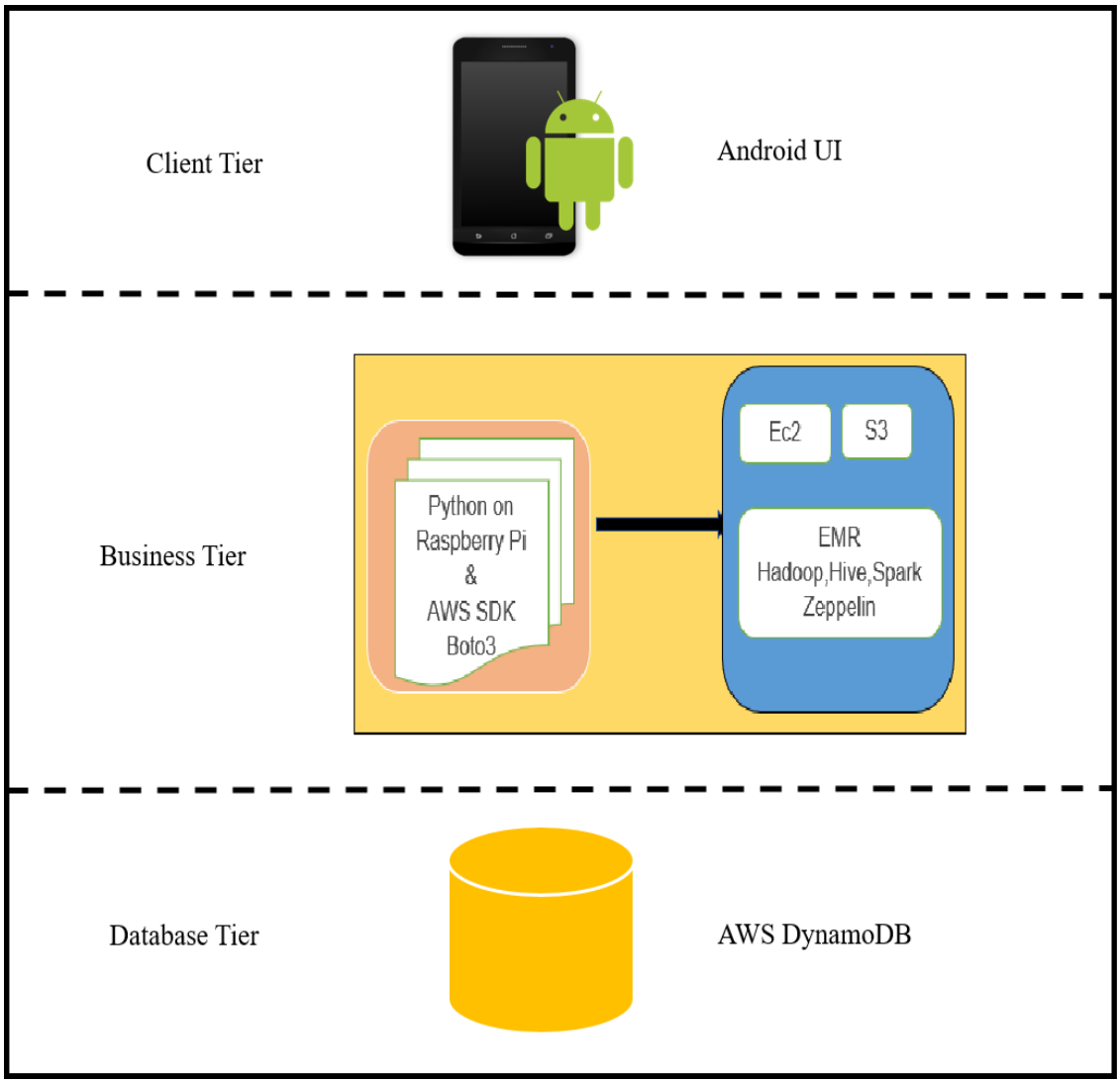

Figure 12: Three tier Software System Architecture

6.2 Architectural Design

The CBF monitoring application depends profoundly on user interface (through user interfaces). This characteristic puts it under the “interface system” category and is the reason why the application has been developed. using the Model-View-Presenter (MVP) architectural design pattern which is the mainstream architectural pattern labored in most visualize and present systems that involve flexible incorporation of human – computer interfaces. It is wise when designing interface systems, to emphasis on usability and modifiability.

MVP systems are encompassed of three separated components that handle independently the system’s input, processing, and output. Separating the system’s output from its core processing functions offers the benefits of allowing different representations of the system core to be easily supported as needed and the seclusion of functional requirements (core functions) and quality requirements (user displays/output) design and implementation therefore growing modifiability and usability.

Model is where the application’s data objects are stored. The model doesn’t know whatsoever about views and presenters and their logic. It is an interface defining the information to be presented or otherwise acted upon in the user interface. The view is a passive interface that shows data (the model) and routes user commands (events) to the presenter to act upon that information. View is what's presented to the users and how users communicate with the application. [33]. The presenter performances upon the model and the view. It fetches information from repositories (the model), and setups it for display in the view.

By using this architectural pattern, the CBF track app benefits from:

- Reusable and extendable code (modifiability).
- Separation of view logic from business logic.

- Exchangeability of user interfaces (usability).
- Easier to maintain code and automate the testing process and improve the separation of concerns in the presentation logic [33].

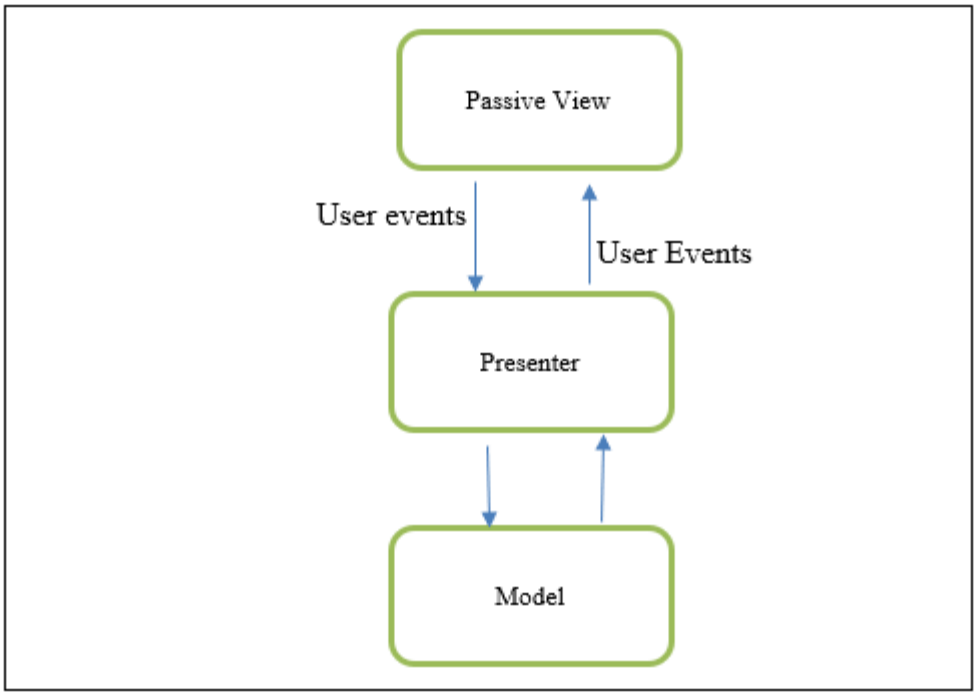

Figure 13: Model View Presenter Design Pattern

CHAPTER 7

SYSTEM USER, USE CASE AND ACTIVITY

Programming for this cloud based bases system is done using the Python Integrated Development Environment(IDE) with the AWS SDK boto3,AWS CLI, DynamoDB local version plugin installed, and python as the coding language framework. For processing inside cloud, AWS EC2,S3,EMR (Hadoop and Spark,Zepplin) instances are used. Android studio operation system and AWS Android SDK plugin is used to integrate with cloud.

7.1 System Users

The CBF Track application user can either be a Patient, a Physician or a Medical Staff Member. The functionality of this system depends on the type of user and the purpose of use of the application. An emblematic use case scenario of the system is discussed and represented in the Figure 14.

7.2 System User Cases

Patient: is the user of the application who adds his emergency medical data such as PatientID,ICP,ABP,CPP,Heart-Frequency,BP-Systolic,BP-Diastolic,BP-Middle,Temperature,PaCO2,PaO2,Patient-Position,Address,State,ZipCode,State, Location ID, Frist Name and Last Name.

Physician: is the user of the application who can add himself/herself as a registered user to track his/her patient's medical information.

Medical Staff Member: is the user of the application which if registered can tie up for monitoring the emergency patients, can also add himself/herself as a registered user to track his/her patient's medical data. Figure 14 describes the use case diagram of this application. this functionality as the bedside patient is kept in 24-hour observation and

nonstop track of regional Cerebral Blood Flow crucial parameters are monitored and processed in cloud and if there is any detection in anomalies then immediately set alarm and notify the medical team (in this scenario Physicians, Medical Staff) and display the line charts of the corresponding parameters over a time using Android App.

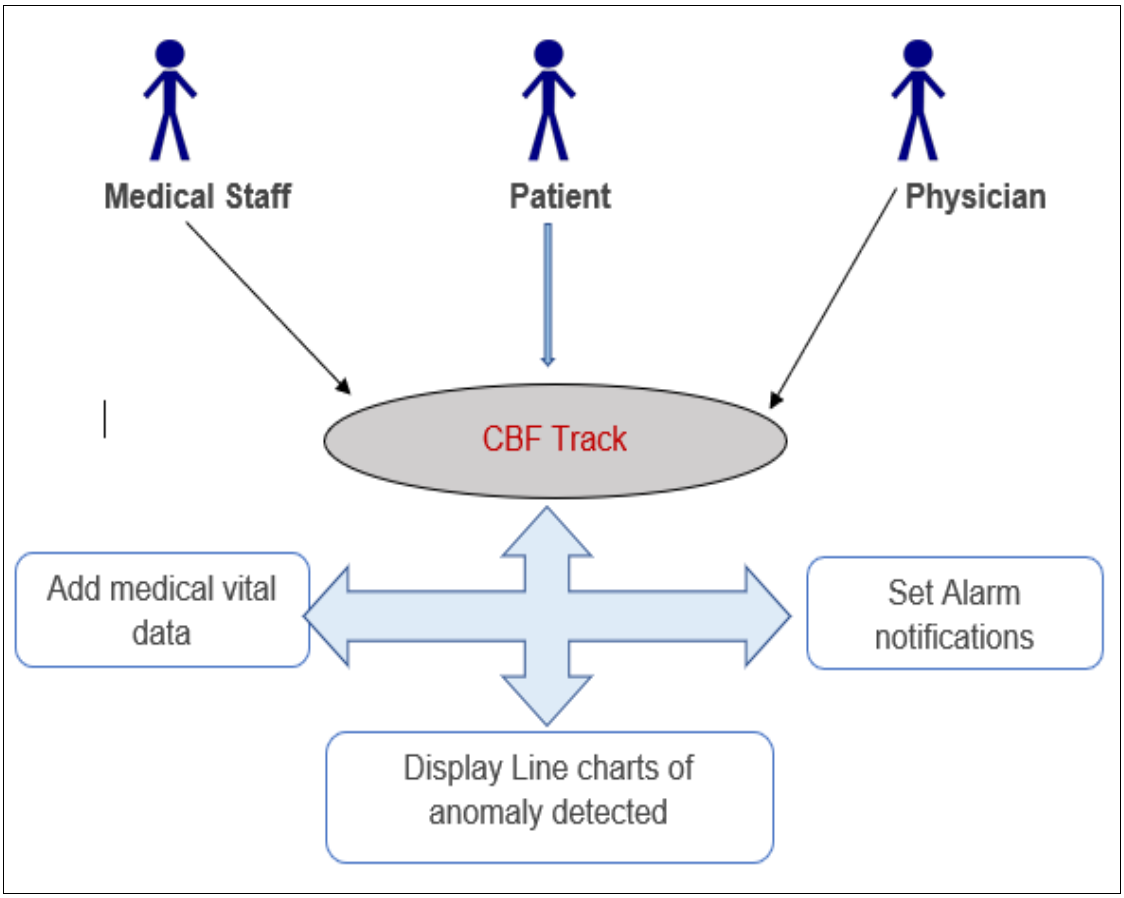

Figure 14: CBF-Track User Diagram

7.3 System Activity Diagram

The application navigates through several activity views and states when the user first begins the application until the end. There are three activity classes, that are IoT, Cloud and Android. The IoT activity class which gathers and pushes data via secure protocol (in this scenario HTTPS through AWS Gateway) into cloud. The cloud activity store,

process and send information to Android mobile devices. The android activity classes which extract the User Interface on the Android device in the Figure 15 below. These views are rendered because of an action activated when the application launches.

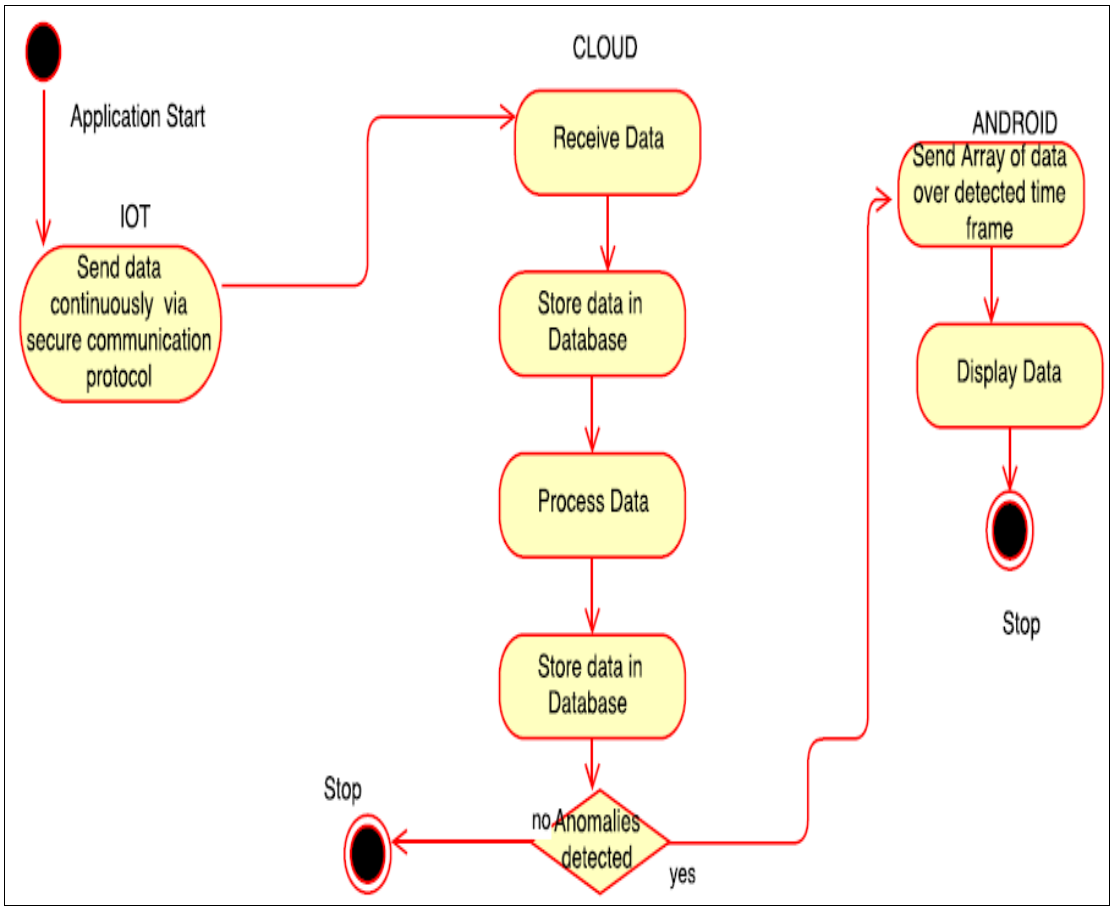

Figure 15: CBF Track Activity Diagram

CHAPTER 8

FUTURE WORK AND CONCLUSION

This Project presented a novel application for cloud computing enabled by IoT through innovative schemes to monitor CBF which is critical in monitoring patients with serious brain impairment. The proposed scheme in this project covers the cloud computing implementation as well as Android application development as UI while the front end of the design (interface with sensor using Raspberry Pi) remains as future work.

The cloud computing and application components were verified with test info. The cerebral circulation information from sensors has been stored in the cloud without any errors and the cloud computing algorithm running in the cloud was implemented. Furthermore, the developed User Interface found to send messages effectively and immediately to the medical professionals using smart devices when an anomaly is sensed. Throughout this implementation, several parameters of cerebral circulation such as MAP, ICP, and CPP have been watched and measured.

In this system, application and success of Information Technology in medicine has been demonstrated through the interaction of Internet of Things, Cloud processing and mobile devices. The application presented in this project allow to monitor emergency bedside patient's vital data in a timely manner and notify any uncharacteristic changes to medical squad. The application's cloud software and mobile platform being Python, AWS (EC2,S3,DynamoDB,EMR-Apache Hadoop,Hive,Spark,Zeppelin)Android studio for Android permits easy access Database using AWS Android SDK through DynamoDB and Android SDK libraries.

Three tier system architecture castoffs in this application permits separation of concern of the presentation, business logic and data layer. The system design provides an easy to use interface and easily supportable functionality for all the use cases of the application.

The application has been successfully tested in many android devices as well as on the cloud using Amazon Web Services. This application can be used as a prototype to design and develop a fully integrated medical application in future.

CHAPTER 9

SOURCE CODE

9.1 DynamoDB with Python

9.1.1Patient.py

```
from __future__ import print_function # Create Patient Table
import os
os.environ["TZ"] = "UTC"
import boto3
dynamodb = boto3.resource('dynamodb', region_name='us-west-2',
endpoint_url="https://dynamodb.us-west-2.amazonaws.com")#http://localhost:8000
table = dynamodb.create_table(
   TableName='Patient',
   KeySchema=[
      {
         'AttributeName': 'PatientID',
         'KeyType': 'HASH'  #Partition key
      },
      {
         'AttributeName': 'DateTime',
         'KeyType': 'RANGE'  #Sort key
      },
   ],
   AttributeDefinitions=[
      {
```

```
            'AttributeName': 'PatientID',
            'AttributeType': 'N'
        },
        {
            'AttributeName': 'DateTime',
            'AttributeType': 'S'
        },
    ],
    ProvisionedThroughput={
    'ReadCapacityUnits': 10,
    'WriteCapacityUnits': 10
    }
)
print("Table status:", table.table_status)
```

9.1.2 MedicalStaffMember.py

```
from __future__ import print_function # Create Medical staff member Table
import os
os.environ["TZ"] = "UTC"
import boto3
dynamodb = boto3.resource('dynamodb', region_name='us-west-2',
endpoint_url="https://dynamodb.us-west-2.amazonaws.com")#http://localhost:8000

    table = dynamodb.create_table(
        TableName='MedicalStaffMember',
        KeySchema=[
```

```
        {
            'AttributeName': 'MemberID',
            'KeyType': 'HASH'  #Partition key
        },
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'MemberID',
            'AttributeType': 'N'
        },
    ],
  ProvisionedThroughput={
        'ReadCapacityUnits': 10,
        'WriteCapacityUnits': 10
    }
)
print("Table status:", table.table_status)
```

9.1.3 Physician.py

```
from __future__ import print_function # Create Physician Table
import os
os.environ["TZ"] = "UTC"

                import boto3
                dynamodb = boto3.resource('dynamodb', region_name='us-west-2',
                endpoint_url="https://dynamodb.us-west-2.amazonaws.com")#http://localhost:8000
```

```
table = dynamodb.create_table(
    TableName='Physician',
    KeySchema=[
        {
            'AttributeName': 'PhysicianID',
            'KeyType': 'HASH'  #Partition key
        },
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'PhysicianID',
            'AttributeType': 'N'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 10,
        'WriteCapacityUnits': 10
    }
)
print("Table status:", table.table_status)
```

9.1.4 Location.py

```
from __future__ import print_function # Create Location Table
import os
os.environ["TZ"] = "UTC"
```

```
import boto3

dynamodb = boto3.resource('dynamodb', region_name='us-west-2',
endpoint_url="https://dynamodb.us-west-2.amazonaws.com")#http://localhost:8000
table = dynamodb.create_table(
  TableName='Location',
  KeySchema=[
    {
      'AttributeName': 'LocationID',
      'KeyType': 'HASH'  #Partition key

    },
  ],
  AttributeDefinitions=[
    {
      'AttributeName': 'LocationID',
      'AttributeType': 'N'
    },
 ],
   ProvisionedThroughput={
    'ReadCapacityUnits': 10,
    'WriteCapacityUnits': 10

        }
      )
      print("Table status:", table.table_status)
```

9.1.5 LoadPatientData.py

```
from __future__ import print_function # Load Data into Patient table
import os
os.environ["TZ"] = "UTC"
import boto3
import json
import decimal
import itertools
import uuid
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError
dynamodb = boto3.resource('dynamodb', region_name='us-west-2',
endpoint_url="https://dynamodb.us-west-2.amazonaws.com")#http://localhost:8000
table = dynamodb.Table('Patient')
with open("PatientVitalCBF111.json") as json_file:
    vitals = json.load(json_file, parse_float = decimal.Decimal)
for vital in vitals:
    response = table.scan(FilterExpression=Attr('PatientID').eq(int(vital['PatientID'])))
    items = response['Items']
   except ClientError as e:
    logger.error("Received error: %s", e, exc_info=True)
   # specific service error code   #if e.response['Error']['Code'] == 'The number of Conditionals
on the key is invalid':
  raise
  if(items):
```

```
    #try:
    print("Adding Vital Information of Registered Patient with ID:", int(vital['PatientID']))
    '''Getresponse = table.get_item(
       Key={
           'PatientID': int(vital['PatientID'])
       })
    ExistingPatientID = Getresponse['Item']['PatientID']'''
    print(int(vital['PatientID']))
    table.put_item(
        Item={
            'DateTime': vital['DateTime'],
            'PatientID': int(vital['PatientID']),
             'ICP': vital['ICP'],
             'MAP': vital['MAP'],
            'Temperature': vital['Temperature'],
            'BPmiddle': vital['BP middle'],
            'BPsystolic': vital['BP systolic'],
            'BPdiastolic': vital['BP diastolic'],
            'Heartfrequency': vital['Heart frequence'],
            'CPP': vital['CPP'],
            'paCO2': vital['paCO2'],
            'paO2': vital['paO2'],
'PatientPosition': vital['Patient position'],
'FirstName': vital['First Name'],
'LastName': vital['Last Name'],
```

```
            'LocationID': vital['Location ID'],
            'Address': vital['Address'],
            'ZipCode': vital['ZipCode'],
            'State': vital['State']

        },
        ConditionExpression="attribute_not_exists(ExistingPatientID)"
      )
  except ClientError as e:
  logger.error("Received error: %s", e, exc_info=True)
  #specific service error code   #if e.response['Error']['Code'] ==
'ConditionalCheckFailedException':
   raise
else:
 for vital2 in vitals:
   response = table.scan(FilterExpression=Attr('PatientID').eq(int(vital2['PatientID'])))
   items2 = response['Items']
   if(not items):
    #try:
    NewPatientID = (uuid.uuid4().int & (1<<8)-1)
    print("Adding Vital Information of New Patient with ID to the table:", NewPatientID)
    table.put_item(
              Item={
                 'DateTime': vital2['DateTime'],
                 'PatientID': NewPatientID,
```

```
        'ICP': vital2['ICP'],
        'MAP': vital['MAP'],
        'Temperature': vital2['Temperature'],
        'BPmiddle': vital2['BP middle'],
        'BPsystolic': vital2['BP systolic'],
        'BPdiastolic': vital2['BP diastolic'],
        'Heartfrequency': vital2['Heart frequence'],
        'CPP': vital2['CPP'],
        'paCO2': vital2['paCO2'],
        'paO2': vital2['paO2'],
        'PatientPosition': vital['Patient position'],
        'FirstName': vital['First Name'],
        'LastName': vital['Last Name'],
        'LocationID': vital['Location ID'],
        'Address': vital['Address'],
        'ZipCode': vital['ZipCode'],
        'State': vital['State']

    }

)
```

9.2 JavaScript Files Test and Verify the Loaded Data in Tables

```
        on Local Machine with Local Downloaded Version of DynamoDB
        Link to Run the Below Queries:   http://localhost:8000/shell/
var params = {
      TableName : "Patient",
      KeyConditionExpression: "#ID = :ID",
      ExpressionAttributeNames:{
        "#ID": "PatientID"
      },
      ExpressionAttributeValues: {
        ":ID": 1
      }
   };
   docClient.query(params, function(err, data) {
      if (err) {
        console.log("error");
      } else {
        console.log(data);
      }
   });
var params = {
        TableName : "Patient",
        KeyConditionExpression: "#ID = :ID",
        ExpressionAttributeNames:{
```

```
            "#ID": "PatientID"
        },
        ExpressionAttributeValues: {
            ":ID": 1
        }
    };
    docClient.query(params, function(err, data) {
        if (err) {
            console.log("error");
        } else {
            console.log(data);
        }
    });
 var params={};
 dynamodb.listTables(params,myFunc)
 var myFunc= function(err,data){console.log(data)}
 dynamodb.listTables(params,myFunc)
 var params = {
 };
 dynamodb.listTables(params, function(err, data) {
    if (err) ppJson(err); // an error occurred
    else ppJson(data); // successful response
});
var params={};
dynamodb.listTables(params,myFunc)
```

```
var myFunc= function(err,data){console.log(data)}
dynamodb.listTables(params,myFunc)
var params = {
};
dynamodb.listTables(params, function(err, data) {
   if (err) ppJson(err); // an error occurred
   else ppJson(data); // successful response
});
var params = {
      TableName : "Vitals",
      KeyConditionExpression: "#id = :ID",
      ExpressionAttributeNames:{
         "#id": "PatientID"
      },
      ExpressionAttributeValues: {
         ":ID": 250
      }
   };
   docClient.query(params, function(err, data) {
      if (err) {
         console.log("error");

} else {
   console.log(data);
}
});
```

9.3 Hive Tables

9.3.1 Hive Table to Store Data in S3

```
CREATE EXTERNAL TABLE PatientVitalInfo_s3
(
DateTime        smalldatetime,
PatientID       BIGINT,
ICP             BIGINT,
MAP             BIGINT,
Temperature     DOUBLE,
BPmiddle        BIGINT,
BPsystolic      BIGINT,
BPdiastolic     BIGINT,
Heartfrequency  BIGINT,
CPP             BIGINT,
paCO2           BIGINT,
paO2            BIGINT,
PatientPosition STRING,
FirstName       STRING,
LastName        STRING,
LocationID      BIGINT,
Address         STRING,
ZipCode         BIGINT,
        State           STRING
      )
```

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://patientvitaldata/input';
```

9.3.2 Hive Table to Store Data From DynamoDB Table

```
CREATE EXTERNAL TABLE PatientVitalInfo_dynamodb (
    PatientID       BIGINT,
    ICP             BIGINT,
    MAP             BIGINT,
    Temperature     DOUBLE,
    BPmiddle        BIGINT,
    BPsystolic      BIGINT,
    BPdiastolic     BIGINT,
   Heartfrequency   BIGINT,
   CPP              BIGINT,
    paCO2           BIGINT,
    paO2            BIGINT,
    PatientPosition  STRING,
    FirstName       STRING,
    LastName        STRING,
    LocationID      BIGINT,
    Address         STRING,
    ZipCode         BIGINT,
            State           STRING)
        STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
        TBLPROPERTIES ("dynamodb.table.name"="Patient","dynamodb.column.mapping" =
```

"PatientID:PatientID,ICP:ICP,Temperature:Temperature,BPmiddle:BPmiddle,BPsystolic:BPs
ystolic,BPdiastolic:BPdiastolic,Heartfrequency:Heartfrequency,

CPP:CPP,paCO2:paCO2,paO2:paO2,PatientPosition:PatientPosition,FirstName:FirstName,La
stName:LastName,LocationID:LocationID,Address:Address,ZipCode,ZipCode,State:State");

9.3.3 Queries to Load Data From Hive DynamoDB to Hive S3 Table

```
INSERT OVERWRITE TABLE PatientVitalInfo_s3
SELECT * FROM PatientVitalInfo_dynamodb;
```

9.3.4 Hive Table to Store Processed Data From Spark to S3(as Text File)

```
CREATE EXTERNAL TABLE IF NOT EXISTS PatientVitalsInfo_Spark_To_S3
PatientID                BIGINT,
ResultofICP              BIGINT,
ResultofMAP              BIGINT,
ResultofTemperature      DOUBLE,
ResultofBPmiddle         BIGINT,
ResultofBPsystolic       BIGINT,
ResultofBPdiastolic      BIGINT,
ResultofHeartfrequency   BIGINT,
ResultofCPP              BIGINT,
ResultofpaCO2            BIGINT,
ResultofpaO2             BIGINT,
    ResultofPatientPosition  STRING,
    FirstName                STRING,
    LastName                 STRING,
    LocationID               BIGINT,
```

```
Address                 STRING,
ZipCode                 BIGINT,
State                   STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 's3://patientvitaldata/output'
TBLPROPERTIES ("skip.header.line.count"="1");
```

9.3.5 Query to Verify Data Loaded in the Table or not for 5000 Records

```
Select * from PatientVitalsInfo_Spark_To_S3 limit 5000;
```

9.3.6 Hive Table to Store Processed Data from Spark to S3(as hive Format)

```
CREATE TABLE IF NOT EXISTS processedPatientvitals
(PatientID              BIGINT,
 ResultofICP            BIGINT,
 ResultofMAP            BIGINT,
 ResultofTemperature    DOUBLE,
 ResultofBPmiddle       BIGINT,
 ResultofBPsystolic     BIGINT,
 ResultofBPdiastolic    BIGINT,
 ResultofHeartfrequency BIGINT,
 ResultofCPP            BIGINT,
ResultofpaCO2           BIGINT,
ResultofpaO2            BIGINT,
ResultofPatientPosition STRING,
FirstName               STRING,
```

```
 LastName            STRING,
 LocationID          BIGINT,
 Address             STRING,
 ZipCode             BIGINT,
 State               STRING )
COMMENT 'Processed PatientVitalInfo'
STORED AS ORC
LOCATION 's3://patientvitaldata/output';
```

9.3.7 Queries to Load Data From Hive DynamoDB to Hive S3 Table

```
INSERT OVERWRITE TABLE processedPatientvitals SELECT * FROM
PatientVitalsInfo_Spark_To_S3;
```

9.3.8 Verify the Loaded Data From Hive DynamoDB to Hive S3 Table

```
Select * from processedPatientvitals;
```

9.3.9 Hive Table to Store Data From DynamoDB Table

```
CREATE EXTERNAL TABLE ddb_PatientProcessedVitals
  (PatientID             BIGINT,
   ResultofICP           BIGINT,
   ResultofTemperature   DOUBLE,
   ResultofBPmiddle      BIGINT,
     ResultofBPsystolic    BIGINT,
     ResultofBPdiastolic   BIGINT,
     ResultofHeartfrequency BIGINT,
     ResultofCPP           BIGINT,
```

```
    ResultofpaCO2          BIGINT,
    ResultofpaO2           BIGINT,
    ResultofPatientPosition  STRING,
    FirstName              STRING,
    LastName               STRING,
    LocationID             BIGINT,
    Address                STRING,
    ZipCode                BIGINT,
    State                  STRING)
 STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
 "dynamodb.table.name" = "processedPatientvitals",
"dynamodb.column.mapping"="PatientID:PatientID,ResultofICP:ResultofICP,ResultofTempe
rature:ResultofTemperature,ResultofBPmiddle:ResultofBPmiddle,
ResultofBPsystolic:ResultofBPsystolic,ResultofBPdiastolic:ResultofBPdiastolic,
ResultofHeartfrequency:ResultofHeartfrequency,ResultofCPP:ResultofCPP,ResultofpaCO2:R
esultofpaCO2,ResultofpaO2:ResultofpaO2,ResultofPatientPosition:ResultofPatientPosition,
FirstName:FirstName,LastName:LastName,LocationID:LocationID,Address:Address,ZipCod
e,ZipCode,State:State");
```

9.3.10 Queries to Load Data From Hive DynamoDB to Hive S3 Hive Table

```
INSERT OVERWRITE TABLE ddb_PatientsProcessedVitals
 SELECT
    PatientID,
    ResultofICP,
    ResultofTemperature,
```

```
ResultofBPmiddle,
ResultofBPsystolic,
ResultofBPdiastolic,
ResultofHeartfrequency,
ResultofCPP,
ResultofpaCO2,
ResultofpaO2,
ResultofPatientPosition,
FirstName,
LastName,
LocationID,
Address,
ZipCode,
State
FROM processedPatientvitals;
```

9.4 Ipython Queries to Process the Data in Zeppelin Notebook on AWS Cloud

```
%pyspark
#import local library
from pprint import pprint
#1 Read the file from the storage
VitalDataRDD = sc.textFile('s3://patientvitaldata/input/000000_0')
#2 RDD count action for 5000 records
    VitalDataCount = VitalDataRDD.count()
    print VitalDataCount
    print VitalDataRDD.take(5000)
```

```
#CSV usage
csv_vitalsRDD = VitalDataRDD.map(lambda x: x.split(","))
#Data Frame
   vital_row_data = csv_vitalsRDD.map(lambda p: Row(
   PatientID = int(p[0]),
   ResultofICP = int(p[1]),
   ResultofMAP = int(p[2]),
   ResultofTemperature = float(p[3]),
   ResultofBPmiddle = int(p[4]),
   ResultofBPsystolic = int(p[5]),
   ResultofBPdiastolic = int(p[6]),
  ResultofHeartfrequency = int(p[7]),
   ResultofCPP = int(p[8]),
   ResultofpaCO2 = int(p[9]),
  ResultofpaO2 = int(p[10]),
   ResultofPatientPosition = int(p[11]),
   FirstName = p[12],
   LastName = p[13],
   LocationID = int(p[14]),
   Address = p[15],
   ZipCode = int(p[16]),
      State = p[17]
     )
   )
```

```
Vitals_df = sqlContext.createDataFrame(vital_row_data)
Vitals_df.registerTempTable("PatientVitals")
#calculating a derived column on a data frame for all the patients vitals
def get_label_ICP(ResultofICP):
   if(ResultofICP >= 90):
      return "ICP is High"
   elif(ResultofICP <= 20):
      return "ICP is Low"
   else:
       return "ICP is normal"

def get_label_ICP(ResultofMAP):
   if(ResultofMAP >= 60):
      return "MAP  is High"
   elif(ResultofMAP <= 20):
      return "MAP is Low"
   else:
       return "MAP is normal"
def get_label_Temp(ResultofTemperature):
   if(ResultofTemperature >= 99):
      return "Temperature is High"
   elif(ResultofTemperature <= 90):
      return "Temperature is Low"
   else:
       return "Temperature is normal"
```

```
    def get_label_BPmiddle(ResultofBPmiddle):
        if(ResultofBPmiddle >= 125):
            return "BPmiddle is High"
        elif(ResultofBPmiddle <= 85):
            return "BPmiddle is Low"
        else:
            return "BPmiddle is normal"
    def get_label_BPsystolic(ResultofBPsystolic):
        if(ResultofBPsystolic >= 140):
            return "BPsystolic is High"
        elif(ResultofBPsystolic <= 100):
            return "BPsystolic is Low"
        else:
            return "BPsystolic is normal"
    def get_label_BPdiastolic(ResultofBPdiastolic):
        if(ResultofBPdiastolic >= 160):
            return "BPdiastolic is High"
        elif(ResultofBPdiastolic <= 120):
            return "BPdiastolic is Low"
        else:
            return "BPdiastolic is normal"

def get_label_HeartFrequency(ResultofHeartfrequency):
    if(ResultofHeartfrequency >= 67):
        return "Heart frequency is High"
    elif(ResultofHeartfrequency <= 38):
```

```
        return "Heart frequency is Low"
    else:
        return "Heart frequency is normal"
def get_label_CPP(ResultofCPP):
    if(ResultofCPP >= 70):
        return "CPP is High"
    elif(ResultofCPP <= 30):
        return "CPP is Low"
    else:
        return "CPP is normal"
def get_label_paCO2(ResultofpaCO2):
    if(ResultofpaCO2 >= 40):
        return "paCO2 is High"
    elif(ResultofpaCO2 <= 20):
        return "paCO2 is Low"
    else:
        return "paCO2 is normal"
def get_label_paO2(ResultofpaO2):
    if(ResultofpaO2 >= 60):
        return "paO2 is High"
    elif(ResultofpaO2 <= 40):
        return "paO2 is Low"
    else:
        return "paO2 is normal"
    def get_label_PatientPosition(ResultofPatientPosition):
```

```
    if(ResultofPatientPosition >= 50):
        return "Patient Position is critical"
    elif(ResultofPatientPosition <= 20):
        return "Patient Position is abnormal"
    else:
         return "Blood Pressure is normal"


#Data Frame
vital_labeled_data = csv_vitalsRDD.map(lambda p: Row(
    PatientID = int(p[0]),
    ResultofICP = get_label_ICP(int(p[1])),
    ResultofMAP = get_label_MAP(int(p[2])),
    ResultofTemperature = get_label_Temp(float(p[3])),
    ResultofBPmiddle = get_label_BPmiddle(int(p[4])),
    ResultofBPsystolic = get_label_BPsystolic(int(p[5])),
    ResultofBPdiastolic = get_label_BPdiastolic(int(p[6])),
    ResultofHeartfrequency = get_label_HeartFrequency(int(p[7])),
    ResultofCPP = get_label_CPP(int(p[8])),
    ResultofpaCO2 = get_label_paCO2(int(p[9]))
    ResultofpaO2 = get_label_paO2(int(p[10])),
    ResultofPatientPosition = get_label_PatientPosition(int(p[11])),
    FirstName = p[12],
      LastName = p[13],
      LocationID = int(p[14]),
      Address = p[15],
```

```
        ZipCode = int(p[16]),
        State = p[17]
            )
    )
    vitals_labeled_df = sqlContext.createDataFrame(vital_labeled_data)
    vitals_labeled_df.registerTempTable("VitalswithLabel")
    vitals_labeled_df.show()
    Vitals_RDD = vitals_labeled_df.rdd
    #2 RDD count action
    VitalDataRDDCount = Vitals_RDD.count()
    #print VitalDataRDDCount
    print Vitals_RDD.take(5000)

    #def toCSVLine(data):
    #    return ','.join(str(d) for d in data)
    #lines = Vitals_RDD.map(toCSVLine)
    #lines.saveAsTextFile('s3://patientvitaldata/output/sparktoS3.csv')

    df =
    sqlContext.createDataFrame(Vitals_RDD,['PatientID','ResultofICP','ResultofTemperature','Re
    sultofBPmiddle','ResultofBPsystolic','ResultofBPdiastolic',
    'ResultofHeartfrequency','ResultofCPP','MAP','ResultofpaCO2','ResultofpaO2','ResultofPatien
tPosition','FirstName','LastName','LocationID','Address','ZipCode','State'])
#df.show()
#writeCSV
```

```
df.coalesce(1).write.format('com.databricks.spark.csv').options(header='true').save('s3://patien
tvitaldata/output/outputfromsparktoS3.csv')


#Vitals_RDD.saveAsTextFile('s3://patientvitaldata/output/sparktoS3.txt)
#Vitals_RDD.saveAsTextFile('s3://patientvitaldataoutput/sparktoS3.q)
#vitals_labeled_df.select("ResultofCPP").groupBy("ResultofCPP").count
#vitals_labeled_df.select("ResultofICP").groupBy("ResultofICP").count()
#group by on a data frame
#query in sql
%sql
--select PatientID,Name,CPP,ICP,MAP from PatientVitals where PatientID = '90'
select PatientID,Name,CPP,ICP,MAP from PatientVitals
```

9.5 Android Application Files

9.5.1 Xml Files

9.5.1.1 Activity_Main.xml

```
 <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"

android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="app.num.linechart.MainActivity">
```

```
  <com.github.mikephil.charting.charts.LineChart
    android:id="@+id/chart"
    android:layout_width="match_parent"
    android:layout_height="300dp" />
</RelativeLayout>
```

9.5.1.2 AndrioidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

   package="app.num.linechart">

   <uses-permission android:name="android.permission.INTERNET" />

   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

   <application

      android:allowBackup="true"

      android:icon="@mipmap/ic_launcher"


      android:label="@string/app_name"

      android:supportsRtl="true"
```

```
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

9.5.1.3 Strings.xml

```
<resources>
    <string name="app_name">CPP(mmHg) , MAP and ICP</string>
</resources>
```

9.5.1.4 Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

9.5.1.5 Dimens.xml

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

9.5.1.6 Styles.xml

```
<resources>
    <!-- Base application theme. -->
        <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
            <!-- Customize your theme here. -->
```

```
        <item name="colorPrimary">@color/colorPrimary</item>

        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>

        <item name="colorAccent">@color/colorAccent</item>

    </style>

</resources>
```

9.5.2 Java Files

9.5.2.1 MainActivity.java

```
package app.num.linechart;

import android.graphics.Color;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import com.amazonaws.auth.CognitoCachingCredentialsProvider;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.dynamodbv2.*;

 import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.*;
```

```
import com.amazonaws.services.dynamodbv2.model.*;

import com.github.mikephil.charting.charts.BarChart;

import com.github.mikephil.charting.charts.LineChart;

import com.github.mikephil.charting.data.BarData;

import com.github.mikephil.charting.data.BarDataSet;

import com.github.mikephil.charting.data.BarEntry;

import com.github.mikephil.charting.data.Entry;

import com.github.mikephil.charting.data.LineData;

import com.github.mikephil.charting.data.LineDataSet;

import com.github.mikephil.charting.utils.ColorTemplate;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {


 @Override

 protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

LineChart lineChart = (LineChart) findViewById(R.id.chart);

ArrayList<Entry> entries = new ArrayList<>();

entries.add(new Entry(4f, 0));

entries.add(new Entry(8f, 1));

entries.add(new Entry(6f, 2));

entries.add(new Entry(2f, 3));

entries.add(new Entry(18f, 4));

entries.add(new Entry(9f, 5));

LineDataSet dataset = new LineDataSet(entries, "Time(sec)");

ArrayList<String> labels = new ArrayList<String>();


      labels.add("0");

      labels.add("500");
```

```
labels.add("1000");

labels.add("1500");

labels.add("2000");

labels.add("2500");

labels.add("3000");

LineData data = new LineData(labels, dataset);

dataset.setColors(ColorTemplate.COLORFUL_COLORS); //

dataset.setDrawCubic(false);

dataset.setDrawFilled(true);

lineChart.setData(data);

lineChart.animateY(5000);

// Initialize the Amazon Cognito credentials provider

CognitoCachingCredentialsProvider credentialsProvider = new

CognitoCachingCredentialsProvider(
```

```
            getApplicationContext(),
            "us-west-2:ef737445-b03e-4c64-af7d-a84235a843f5", // Identity pool ID
            Regions.US_WEST_2 ); // Region
    }

}
```

9.5.2.2 ApplicationTest.java

```
package app.num.linechart;

import android.app.Application;

import android.test.ApplicationTestCase;

public class ApplicationTest extends ApplicationTestCase<Application> {

    public ApplicationTest() {

        super(Application.class);

        }

    }
```

9.5.2.3 ExampleUnitTest.java

```
package app.num.linechart;

import org.junit.Test;

import static org.junit.Assert.*;

/**
 * To work on unit tests, switch the Test Artifact in the Build Variants view.
 */

public class ExampleUnitTest {

    @Test

    public void addition_isCorrect() throws Exception {

        assertEquals(4, 2 + 2);

    }

}
```

9.6 Configuration Files

9.6.1 Line Chart

```
// Top-level build file where you can add configuration options common to all sub-
```

```
    projects/modules.

    buildscript {

        repositories {

            jcenter()

        }
        dependencies
    {
            classpath 'com.android.tools.build:gradle:2.3.3'

            // NOTE: Do not place your application dependencies here; they belong

            // in the individual module build.gradle files
        }

    }

    allprojects {

        repositories {

            jcenter()

   }

}
```

```
task clean(type: Delete) {
    delete rootProject.buildDir
}
```

9.6.2 Application Dependencies

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 23
    buildToolsVersion '25.0.0'
    defaultConfig {
        applicationId "app.num.linechart"
        minSdkVersion 15

        targetSdkVersion 23
versionName "1.0"
```

```
    }

    buildTypes {

        release {

            minifyEnabled false

            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

        }
    }
}

repositories {


    maven { url "https://jitpack.io" }

}

dependencies {

    compile fileTree(dir: 'libs', include: ['*.jar'])

        testCompile 'junit:junit:4.12'

        compile 'com.android.support:appcompat-v7:23.1.1'
```

```
    compile 'com.github.PhilJayy:MPAndroidChart:v2.1.6'

    compile 'com.amazonaws:aws-android-sdk-core:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-s3:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-ddb:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-cognito:2.2.13'

    compile 'com.amazonaws:aws-android-sdk-rekognition:2.3.9'

    compile 'com.amazonaws:aws-android-sdk-ddb:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-cognito:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-apigateway-core:2.2.+'

    compile 'com.amazonaws:aws-android-sdk-s3:2.2.+'



    compile 'com.amazonaws:aws-android-sdk-ddb-mapper:2.2.+'

}
```

REFERENCES

[1] http://www.cloud-council.org, Impacts of Cloud Computing on Healthcare, Version 2.0, February 2017

[2] L. Hill, C Gwinnutt, “Cerebral Blood Flow and Intracranial Pressure”

[3] B. J. Kirby, “Micro and Nanoscale Fluid Mechanics: Transport in Microfluidic Devices”, Cambridge University Press

[4] MA. Kirkman, M. Smith, “Intracranial Pressure Monitoring, Cerebral Perfusion Pressure Estimation, and ICP/CPP-guided Therapy”, British Journal of Anaesthesia, January 2014

[5] https://aws.amazon.com

[6] Alali AS, et al., “Intracranial Pressure Monitoring Among Children With Severe Traumatic Brain Injury”, Journal of Neurosurgery: Pediatrics, 2015 Aug.

[7] Dawes AJ, et al., “Intracranial Pressure Monitoring And Inpatient Mortality In Severe Traumatic Brain Injury: A Prosperity Score Matched Analysis”, Journal of Trauma Acute Care Surgery, 2015 Mar.

[8] Marcus H.J. and Wilson M.H., “Insertion of an Intracranial-Pressure Monitor”, New England Journal of Medicine, 2015 Nov. 26, 373:e25

[9] Kang SK, et. al., “Bioresorbable Silicon Electronic Sensors for the Brain”, Nature. 2016 Feb.

[10] https://aws.amazon.com/blogs/startups/internet-of-beer-introducing-simple-beer-service

[11] https://www.raspberrypi.org/downloads

[12] Donnelly, et. al., "Regulation of the Cerebral Circulation: Bedside Assessment and Clinical Implications.” Journal of Critical Care, 2016 Jan.

[13] Rowley, Jennifer, "The wisdom hierarchy: representations of the DIKW hierarchy". Journal of Information and Communication Science 33, 2007.

[14] https://www.dezyre.com/article/5-healthcare-applications-of-hadoop-and-big-data/85

[15]http://hemedex.com/wp-content/uploads/2013/02/Publication-P.-Vajkoczy.pdf

[16]http://scholarworks.csun.edu/bitstream/handle/10211.3/171629/Malhotra-Nishika-thesis-2016.pdf;sequence=1

[17]Amazon Web Services Developer Guide © 2016

[18] Werner, V. (2013). Taking DynamoDB beyond Key-Value: Now with Faster, More Flexible, More Powerful Query Capabilities. All Things Beautiful http://www.allthingsdistributed.com/2013/12/dynamodb-global-secondary-indexes.html

[19] Names, A. (2015) Using Amazon DynamoDB with the AWS .NET API. Tips and tricks in C# .NET http://dotnetcodr.com/2015/01/22/using-amazon-dynamodb-with-the-aws-net-api-part-1-introduction/

[20]http://www.aetna.com/cpb/medical/data/700_799/0703.html

[21] mHealth: mobile technology poised to enable a new era in health care, Ernst & Young,© 2012 EYGM Limited.

[22]http://fortune.com/2015/09/02/salesforce-health-cloud-healthcare/

[23]http://searchcloudapplications.techtarget.com/news/4500278226/Boston-Heart-is-a-pacemaker-in-mobile-app-UI-design

[24] ran, J., Tran, R., and White, J. R. (2012). Smartphone-based glucose monitors and applications in the management of diabetes: an overview of 10 salient “apps” and a novel smartphone-connected blood glucose monitor. Clinical Diabetes, 30(4), 173-178.

[25] http://www.onlinejournal.in/IJIRV2I2/050.pdf

[26] http://www.ijritcc.org/download/ICAET15TR011452.pdf

[27] Truchikachorn, P., Liang, J. J., Devarakonda, M., and Mueller, K. Watson-Aided Non- Linear Problem-Oriented Clinical Visit Preparation on Tablet Computer.

[28] https://www.itgct.com/the-7-benefits-of-cloud-computing-for-healthcare/

[29]https://concisesoftware.com/4-benefits-of-internet-of-things-for-healthcare/

[30]https://en.wikipedia.org/wiki/Python_%28programming_language%29

[31] https://en.wikipedia.org/wiki/Android_Studio

[32]https://www.forbes.com/sites/johnkoetsier/2017/05/18/surprise-google-reveals-apples-ios-market-share-is-65-to-230-bigger-than-we-thought/#23fcb2e25890

[33]https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter