January 2018

Remote Screening And Self-Monitoring For Vision Loss Diseases Based On Smartphone Applications

Hussein M. H. Khairallah
Wayne State University, hkairalla@yahoo.com

REMOTE SCREENING AND SELF-MONITORING FOR VISION LOSS DISEASES BASED ON SMARTPHONE APPLICATIONS

by

HUSSEIN M. H. KHAIRALLAH

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2018

MAJOR: COMPUTER ENGINEERING

Approved By:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my wife

My wife has shown me that a smile and positive attitude indeed goes a long way.

To my kids

Mahmoud, Mustafa, Jana, Haya and Al-Waleed for their love and patience

ACKNOWLEDGEMENTS

In the name of Allah, first, I would like to start by acknowledging my closest friend and best supporter, my wife. Thank you for all the love and encouragement. Second, I would like to take this opportunity to express my profound gratitude and thanks to both of my supervisors, Dr. Nabil Sarhan and Dr. Lubna Alazzawi. They have been very supportive of my work and have always provided me with the encouragement to develop my career with independence. Most profound gratitude is also given to my committee members, Dr. Mumtaz Usmen, Dr. Harpreet Singh, and Dr. Pai-Yen Chen, who monitored my work and made the effort to read and provide valuable comments on this dissertation. Thank you very much for your guidance. Finally, I am genuinely thankful for my colleagues at Wayne State University for their constant support and contribution.

TABLE OF CONTENTS

DEDICATION.....**ii**

ACKNOWLEDGEMENTS.....**iii**

LIST OF FIGURES.....**xi**

LIST OF TABLES.....**xii**

CHAPTER 1: INTRODUCTION AND MOTIVATION.....**1**

1.1 Introduction.....1

1.2 Motivation and Problem Statement 1

1.3 Research Objective and Specific Aims 2

1.4 Literature Review.....4

1.5 Original Contributions.....9

1.6 Organization of the Dissertation.....10

CHAPTER 2: ARCHITECTURAL DESIGN SOLUTIONS FOR RHMS.....**12**

2.1 Introduction.....12

2.2 Related Work.....13

2.3 Architecture of RHMS.....14

2.3.1 Body Area Network (BAN).....15

2.3.2 Personal Area Network Coordinators (PANC).....16

2.3.3 Back- Medical End Systems (BMEsys) 17

2.4 Multi-Patient Network Coordinators 18

2.4.1 Wireless Sensor Network (WSN) 19

2.4.2 Personal Digital Assistant (PDA) 20

2.4.3 Smart Phone.....21

2.5 Performance Evaluation of Multi-Patients Network Coordinators.....23

2.6 Proposed System Components.....24

2.6.1 Sensors.....24

2.6.2 Smartphone Applications.....25

2.6.2.1 Mobile Browser or Web Apps.....25

2.6.2.2 Hybrid Apps.....26

2.6.2.3 Native Apps.....26

2.6.2.4 Email.....27

2.6.2.5 Short Message Service (SMS) Text Messaging.....27

2.6.2.6 Recording Pictures, Audio, and Video.....27

2.6.3 Medical End System.....28

2.7 Conclusions.....29

CHAPTER 3: UTILIZATION OF MEDICAL APPS AND SMARTPHONE SENSORS IN RHMS.....30

3.1 Introduction.....30

3.2 Smartphone Applications in Healthcare.....30

3.2.1 Patient Care and Monitoring.....30

3.2.2 Health Apps for the Layperson.....32

3.3 Smartphone Sensors.....33

3.3.1 Accelerometers.....33

3.3.2 Gyros or Gyroscope Sensors.....34

3.3.3 Proximity Sensors.....35

3.3.4 Light Sensors.....35

3.3.5 Magnetometer Sensors……..…...….36

3.3.6 Barometer Sensors.....37

3.3.7 Camera Sensors.....37

3.4 Smartphones Sensors in Health Applications……..39

3.4.1 Microphone Sensors in Healthcare……...40

3.4.2 Accelerometer Sensors in Healthcare……...40

3.4.3 Camera Sensors in Healthcare……..41

3.5 Using Smartphone Apps and Its Camera Sensors in Healthcare……41

3.5.1 My Cancer Diary (MCD)……..41

3.5.2 Blood Culture (BC)……...42

3.5.3 Point-of-Care HIV Check.....43

3.6 Conclusions….....44

CHAPTER 4: REMOTE SCREENING AND SELF-MONITORING FOR VISION LOSS DISEASES……....45

4.1 Introduction……45

4.2 Related Work……..……48

4.3 Diabetic Retinopathy……...51

4.3.1 Diabetic Eye Problems……..52

4.3.2 Diabetic Retinopathy Symptoms and Risk Factors……..……...53

4.3.3Diabetic Retinopathy Treatment.….....53

4.4 Cataracts…..…….……...…...54

4.4.1 Cataract Causes and Risk Factors……55

4.4.2 Cataracts Symptoms.....…...56

4.4.3 Cataracts Treatment……..56

4.5 Macular Degeneration……..……...56

4.5.1 Types of Macular Degeneration or AMD……...…….57

4.5.2 Symptoms and Risk Factors for AMD……....58

4.5.3 Treatment Options for AMD……...59

4.6 Glaucoma……60

4.6.1 Causes and Symptoms……..60

4.6.1.1 Open Angle Glaucoma…….….60

4.6.1.2 Closed Angle Glaucoma……....61

4.6.2 Risk Factors for Glaucoma……...61

4.6.3 Glaucoma Treatment……61

4.7 Smartphone Ophthalmic Adapters for Eye Health Imaging Diagnostics……..….62

4.7.1 Smartphone Apps in Ophthalmology….....63

4.7.2 Evaluating Patients' Eyes Using the Smartphone Camera and Smartphone Adapters……64

4.7.2.1 Portable Eye Examination kit (Peek)……....65

4.7.2.2 EyeGo Adapter……66

4.7.2.3 D-Eye Adapter.....66

4.8 Performance Comparison between the Three Adapters, EyeGo Peek, and D-Eye.....68

4.9 Conclusions……..68

CHAPTER 5: HEALTHCARE MONITORING SYSTEM FOR EYE DISEASES USING A SMARTPHONE CAMERA AND D-ADAPTER……..70

5.1 Introduction……..70

5.2 The Ultimate Aim of the Dissertation……..71

5.3 Healthcare Monitoring Mobile App…….71

5.3.1 APP Client……71

5.3.2 Collect Server…….....…….…….71

5.3.3 Devices…….……..……...……...……...72

5.3.3.1 Smartphone and D-Eye Adapter……...……...……72

5.3.3.2 Healthcare Monitoring Mobile APP- Flow Chart...…...…...……73

5.3.3.3 Healthcare Monitoring Mobile APP – Algorithm (1)"User Login"……75

5.3.3.4 Healthcare Monitoring Mobile APP– Algorithm (2)"Make A new Request……...76

5.3.3.5 Healthcare Monitoring Mobile APP – Algorithm (3) "Solve the Request"…….…...78

5.3.3.6 Healthcare Monitoring Mobile App: "Skype or Viber Communication"……79

5.3.3.7 Conclusions..…….……....80

CHAPTER 6: CONTRIBUTIONS AND FUTURE WORK..... 81

6.1 Contributions……......81

6.2 Future work…….....82

APPENDICES……...……...84

BIBLIOGRAPHY ……...……..158

ABSTRACT ……...……170

AUTOBIOGRAPHICAL STATEMENT……...….172

LIST OF FIGURES

Figure 1: Architecture of the RHMS 15

Figure 2: Body Area Network (BAN) 16

Figure 3: Personal Area Network Coordinator (PANC) 17

Figure 4: Back-Medical End Systems (BMEsys) 18

Figure 5: Multi-Patient PAN Coordinators.....18

Figure 6: Wireless Sensor Network (WSN) 20

Figure 7: Basic PDA Layouts 21

Figure 8: Smartphone Sensors 22

Figure 9: Mobile Phone Sensing Architecture.....23

Figure 10: The Wireless Monitoring Sensors Integrated Into Garments.....25

Figure 11: The Three Sorts of Smartphone Applications 27

Figure 12: QuitPal App for Smoking Cessation 28

Figure 13: Proposed System Components 29

Figure 14: Blood Pressure Smartphone App 32

Figure 15: ITriage Apps in Smartphone 33

Figure 16: The Accelerometer Sensors in Smartphone 34

Figure 17: The Gyroscope Sensors in Smartphone.....34

Figure 18: Proximity Sensors in Smartphone 35

Figure 19: The Light Sensor in Smartphone.....36

Figure 20: The Magnetometer Sensor.....36

Figure 21: The Barometer Sensor 37

Figure 22: The Smartphone Module: Sensor and the Lens.....38

Figure 23: CMOS (Complementary Metal-Oxide-Semiconductor) Sensor 38

Figure 24: The Lenses of Smartphone Camera.....39

Figure 25: My Cancer Diary app (A) Patient Barcode Sample (B) Screen Shot of Main Menu...42

Figure 26: Blood Culture app (A) Barcode Reading Menu; (B) Barcode Reading Phase; (C) Reading Results and Entering Data such as: the Sample Site and Volume; (D) Saving Data. ID and Patient Name Were Used 43

Figure 27: Point-of-Care HIV Check App (A) Screenshot of Main Menu; (B) Screenshot of Barcode Reading Phase.....43

Figure 28: A Normal Eye and Eye with Diabetic Retinopathy 52

Figure 29: Non-proliferative (NPDR) and Proliferative Diabetic Retinopathy (PDR) 53

Figure 30: Treatment Options: Laser Surgery, Vitrectomy Surgery and Medication Injections...54

Figure 31: Eye with Clear Lens and Eye with Cataracts 55

Figure 32: Normal macular degeneration and anatomy of a normal eye and loss of central vision……..57

Figure 33: The Two Types: Wet and Dry of Macular Degeneration.....58

Figure 34: The Symptoms of Macular Degeneration 59

Figure 35: Healthy Eye and the Eye with Glaucoma.....60

Figure 36: Open Angle Glaucoma and Acute Glaucoma 61

Figure 37: Medication: Glaucoma Treatment.....62

Figure 38: Shows the Retina Adapter and The Examination Test.....63

Figure 39: Shows the eye care professionals examine patient's with different eyes smartphone adapters……64

Figure 40: shows (peek retina adapter) and the examination test…….65

Figure 41: Shows the EyeGo Adapter and Examination.....66

Figure 42: The D-Eye Adapter and the Image Received From the Test.....67

Figure 43: D-Eye Adapter with the Smartphone: "Examination Test"…..…….. 72

Figure 44 Healthcare Monitoring App: "Flow chart"….……...……73

Figure 45: Healthcare Monitoring App: "Registration Process for the Doctor"…...…….....74

Figure 46: Healthcare Monitoring App: "Registration Process for the Patient".....74

Figure 47: Healthcare Monitoring App - Algorithm (1). "User login"……75

Figure 48: Healthcare Monitoring App: Login Process "Doctor Login and Patient Login"….....75

Figure 49: Healthcare Monitoring App - Algorithm (2). "Make a New Request"....….…….. 76

Figure 50: (a) Making a Request Process: "Choose a Doctor, Upload a Picture, then Make a Request" ……...……..…76

Figure 51: (a) Making a Request Process: "Choose a Doctor, Upload a Picture, then Make a Request" ……...……..…77

Figure 52: Healthcare Monitoring App: "Upload the Request"…. ……..…..…….77

Figure 53: Healthcare Monitoring App Algorithm (3) "Solve the Request".....…...78

Figure 54: Healthcare Monitoring App Algorithm (3) "Solve the Request".....…...78

Figure 55: (a) Healthcare Monitoring Mobile App: "Skype and Viber Communication"…...…79

Figure 56: (b) Healthcare Monitoring Mobile App: "Skype and Viber Communication"…….....79

LIST OF TABLES

Table 1: Comparison the Three Types of the Coordinators: WSN, PDA and Smartphone….23

Table 2: Comparison the Three Adapters, EyeGo, Peek, and D-Eye……...…….……...68

CHAPTER 1: INTRODUCTION AND MOTIVATION

1.1 Introduction

Healthcare monitoring systems have rapidly evolved over the past two decades, displaying the potential to change the way healthcare is administered to patients. The goal of Remote Healthcare Monitoring System (RHMS) is to give access to restorative administration organizations to anyone at any point, overcoming the necessities of location, time, and character. Most commonly, patients have health issues such as weight, hypertension, erratic heartbeat, or diabetes. In these cases, people are urged to see healthcare professionals for routine and remedial check-ups. In the mobile healthcare system, the patient's health information is gathered by body sensors, and this data is transmitted to RHMS so specialists can then provide treatment.

Sensors play an essential part in RHMS. They measure the physical parameters and give continuous information to health organizations and doctors. The presence of smartphones and other portable devices has allowed us to utilize RHMS for an assortment of structures. Wireless Sensor Network (WSN) advances are considered one of the critical research factors in healthcare applications for enhancing the standard of living of patients. This work represents three tiers operating in the RHMS: Body Area Network (BAN), Personal Digital Assistant Coordinator (PDAC) and Back-Medical End System (BMEsys). This dissertation focuses on several patients' PDAC, which include Wireless Sensor Network, Personal Digital Assistant, and smartphone. It also provides a meaningful utilization comparison between Wireless Sensor Network, PDA, and smartphone in RHMS architecture design.

1.2 Motivation and Problem Statement

Smartphones and mobile devices are present in peoples' daily lives, allowing a continuous and fast flow of information. Nowadays, almost everyone has a smartphone, and new apps are continually being developed. Another fact of our current society is that we are always

trying to improve our healthcare systems; we know that early diagnoses are critical when it comes to treating a disease. Because of these two facts, the development of a Health-Monitoring App has proven to be a crucial idea.

This dissertation, then, aims to develop a Health-Monitoring Mobile App that will be capable of processing, evaluating, interacting with, and storing health data that will continuously be measured by Personal Health Monitors (PHM). This new app is meant to help the patients who have low vision and are suffering from diseases which may cause vision loss, such as diabetic retinopathy, cataracts, age-related macular degeneration, and glaucoma. This proposed app exchanges the information directly between patients and doctor, which allows for higher security and privacy.

1.3 Research Objective and Specific Aims

RHMS has become an essential aspect of our everyday lives. It represents a remote observation of patients' wellbeing and can provide therapeutic services. Sensors play an essential part in RHMS. They measure the physical parameters and give continuous information to health organizations and doctors. Smartphones have allowed us to use RHMS for a variety of tasks to be completed in optimum time. For example, in an emergency, the correspondence may experience freezes, delays, and data mishaps. To avoid these situations; the proposed arrangements for RHMS structures are meant to provide support for people in need of immediate care: those who suffer heart attacks, high blood pressure, and other urgent medical problems. One of the primary considerations of the RHMS is choosing a suitable directing tradition to ensure that data is transmitted as efficiently as possible. In addition to the fast transference of data, the RHMS allows medical staff immediate access to the patient's record and other information, enabling care at the patient's location as soon as possible. WSN is considered one

of the critical research factors in healthcare applications for enhancing the standard of living for patients.

The first objective is to represent the three tiers operating in the RHMS: Body Area Network (BAN), PAN Coordinator and Back-Medical End System (BMEsys). The BAN consists of wearable sensors: patients can use a variety of sensors which collects the information from the human body, including heart rates, blood pressure, and temperature. The data is collected and transmitted to the PAN coordinator for processing. PAN coordinators, such as smart devices, then forward the data to remote servers using long-range communication. Clinical back-end servers receive and process all the patients' data and, depending on the situation; appropriate action is taken by doctors or nurses.

The second objective is to explain in detail the three operating types in the RHMS: Body Area Network (BAN), PAN Coordinator and Back-Medical End System (BMEsys). This system focuses on several patients' PAN coordinators, including Wireless Sensor Network (WSN), and Personal Digital Assistant (PDA), and smartphones. WSN can be utilized at altering story screen area and occasional estimations. PDA can be used as a part of patients' own home or group setting with persistent estimations. Mobile phones can be utilized anywhere with the full range of parameters.

Once the operating systems have been explained, the next, third objective is to provide a meaningful utilization comparison between the three different devices: Wireless Sensor Network (WSN), Personal Area Network Coordinator (PANC) and smartphones in RHMS architecture design. The fourth aim is to evaluate the approaches of the healthcare monitoring system architecture and investigate the use of advanced technologies recording the patient's vital signs and enabling their diagnostic medical team in real-time. The fifth objective is to provide another

meaningful utilization comparison between the three different smartphone adapters: Portable Eye Examination kit (Peek), EyeGo, and D-Eye adapter.

Once all previous goals have been met, the sixth and final aim of this dissertation is to develop a Health-Monitoring App that will be applicable in the field of Ophthalmology. The idea for the development of a Health-Monitoring App consists of the following parts: a smartphone application, a data collection center, and professionals in ophthalmology. The idea is as follows: the patient should be registered in a system—for instance, the Retina Michigan Center or Glaucoma Michigan Center. After the registration, the user is instructed on how to take photos of his/her eye correctly. Then, using the smartphone application, the patient takes the photo of his/her eye and sends it to the data collection server. Opthalmology specialists gain access to this data and prescribe a treatment according to their analysis. Finally, the mobile app system is used, including via the Skype or Viber links that facilitate the exchange of information between patient and doctor.

1.4 Literature Review

This section provides a complete and thorough review of existing literature linked to electronic health. E-health is a system of communication used to deliver medical assistance over the internet. The advancement of information has made it simple for medical professionals to keep records of patients and make decisions directly after consultation with the patient over the internet. In this way, the system provides patients with better medical service and support. As an added benefit, the medical information in the health system is easily modified or edited, and distribution of the data requires strong authentication for communication between patients, healthcare professionals, and providers.

Aramudhan et al. [1] proposed three secure communication rules of conduct for E-health systems: Secure and Auditable Agent-based Communication Protocol, Certificate-based Authentication and Attribute-based Policy Assigned Authorization and Token-based Cross Verification. These rules of conduct ensure secure communication between the user and the healthcare system.

Rifat Shahriyar et al. [2] introduced the "Intelligent Mobile Health Monitoring System (IMHMS)," which involves the usage of the portable wireless personal area network to gather data from patients. This system tracks patients' health status and provides feedback directly to their mobile device. Through this system, IMHMS collects and stores the patient's data using the bio-sensor and sends a summary to the patient's mobile device. After the patient receives the data, he/she is to forward it to the medical server to be analyzed for feedback. Patients have instant access to their feedback, which can be provided by the healthcare expert anywhere and anytime. Despite the advantages of this system, mobile healthcare requires further study to be designed, developed, and evaluated to help patients participate in their health.

To address some of these concerns, Elkhodr et al. [3] provide a reliable discussion approach that enhances the strengths of the Transport Layer Security (TLS). This will result in necessary improvements in security-related matters compared to the classic identity-based access control approach. In this way, trust negotiation can be combined with the secure protocol of TLS on a mobile application. Trust negotiations only occur after the TLS session to ensure that it is refined through a protected contact channel to secure the communication between the client and healthcare expert. Transport Layer Security's (TLS) primary purpose is to provide access and control decisions established on attributes rather than identity. This offers a solution for a specific environment where status is not enough. In this way, several security requirements for

access to patients' EHR have been designed. Alternative security concerns address the liability of mobile devices' theft or loss. The approach provided in this study addresses specifically this problem. The approach plans to enhance the strengths of existing approaches to secure the communication of data through unsecured systems like the web. Also, the approach addresses the privacy and security matters for access to patient's data through mobile devices. This guarantees that the patients' data is only exposed to an authorized healthcare expert through the use of certified devices at legitimate locations.

Similarly, Johannes Barnickel et al. [4] addressed the privacy and security construction and operation of the health net mobile electronic health monitoring and data collection system. The health net project represents the joint research of multiple research groups of RWTH Aachen University established on a sensor system embedded in clothing. The sensor system gathers basic guidelines and wirelessly communicates with the patients' mobile device, which is used to direct, store and relocate data in a secure manner. The patients' data can then be relocated to other parties such as private parties, medical experts, and emergency care services with the permission of the patient. Except for emergency physicians, who can access medical data without consent, the patient decides who accesses his/her data in all other cases. The system is constructed to provide mechanical emergency contact when essential guidelines replicate reddened diagrams. In this way, Cryptography routines are used to avoid security threats on wireless data communication. This shows that their construction is beneficial on currently available technology and that large parts of operations are achievable with available consumer equipment.

In another study, Rahman et al. [5] deal with the application of lightweight asymmetric cryptography algorithm, arranged for widespread healthcare applications. In this study, a general healthcare monitoring system was designed for continuous health monitoring of patients who

have long-term diseases. In the discussed application, the protected framework is designed to acquire the patients' medical data regularly. Medical data of the patient is transferred to the healthcare using mobile communication. The proposed healthcare monitoring system is developed to conserve data privately, ensuring data integrity and certification in a healthy environment. The arranged model functions in a computing environment consisting of a healthcare server and ad-hoc system of portable devices like the mobile phones. Based on restrictions, the most common tool is a mobile phone which is used by most patients.

Fahed Al-Nayadi et al. [6] introduced a policy for sharing medical information among healthcare information systems in a peer-to-peer (P2P) environment. This policy provides safety and confidentiality to healthcare professionals by establishing a high level of confidence concerning a patient's information. Privacy in the e-health system has always been a problem. To address this problem; this system assigns a certificate to each user, and every document has a username and attribute associated with it. This policy provides effective coordination and ensures the safe sharing of information between to healthcare professionals. Every system includes the patient's database, a policy the reputation of other healthcare professionals and associated credential information. A patient's database stores his or her data such as diagnosis, treatments, and medications. As the holder of such information, the health care provider is obligated to protect a patient's data from any unauthorized person. P2P might be helpful, but it has to be safe and secure first. That is why a level of trust and security must be established for it to be beneficial.

Along the same lines, Praveen Baskar et al. [7] provided application (app) developers valuable information about security issues. They argue the safest way to collect and share information is by using security prototypes. Each prototype can detect possible threats and

counter them. All the information collected by the prototypes will keep app developers informed about potential security issues concerning the app.

Since safety is a chief concern in e-health, Aramudhan et al. [8] propose a communication protocol called Secure and Auditable Agent-based Communication Protocol (SAACP). SAACP provides a safe and comfortable means of communication between a user and mobile agent and helps to reduce the delay in the connection between the two in the case of an emergency. An example of the usage of SAACP is in the case of accidents. A user who has been involved in an accident can send messages about his or her crash to the health care system through his or her phone. Based on the information received via messages, the healthcare system will know of the damage inflicted by accident and will also send an ambulance and direct it to the nearest hospital.

Leister et al. [9] introduced a set of scenarios that are used to access security models techniques and prototypes. In Adaptive Security for Smart Internet of Things (ASSET)—based on the Internet of Things—the scenarios can predict interactions between users and patient monitoring systems and help to evaluate adaptive security techniques in e-health. Also, the scenarios develop a framework for the assessment of security solutions. In a scenario-based assessment, scenarios are made to capture two types of requirements: security requirements and quality of service requirements. The scenes in security requirements capture security information for the data processed and share it within the patient's monitoring system. The scenarios in the variety of service requirements capture the needs concerning the quality of service and explain the interaction between the needs and the security requirements.

Hans Löhr et al. [10] developed an ideal model of e-health cloud that grasps the common forms of healthcare telematics infrastructures. Based on the model presented, the three nuclear

security and privacy issues are data storage processing, management of e-health infrastructures, and usability aspects of end-users. This study focused on a commonly forgotten issue, which is the security of client platforms. This study also shows how the privacy domain could be used to protect e-health systems from the current network security solutions to a complete infrastructure. To identify the different issues; the study introduced a model of the e-health cloud and enhanced it, and then determined the principal parties that concern the e-health cloud.

In the final study discussed here, Eman Abu-Khousa et al. [11] introduced an e-health cloud for a healthy environment. The authors address security issues and, to build a better e-health cloud, attempt to work with the four principal factors that contribute to making the e-health cloud: cloud-based solutions and HIT applications and systems; platform solutions; e-health implementation model; and security solutions for the e-health cloud. This ideal model is presented while trying to work with current security solutions such as focusing on network security, access control policies, and weak platform security. In doing so, the study tries to find solutions to both the technical and non-technical issues that could hinder the success of the e-health cloud.

1.5 Original Contributions

The original contributions of this dissertation are as follows:

I provide a meaningful utilization comparison between Wireless Sensor Network, PDA and smartphone in Remote Healthcare Monitoring System (HRMS) architecture design. To accomplish this, I evaluate the approaches of the healthcare monitoring system architecture and investigate the use of advanced technologies enabling the transmittal of patients' vital signs to a diagnostic medical team in real-time. In this comparison, the Smartphone gives the best performance as it can be used anywhere, providing early hospital admission and continuous

measurement of patient status. As such, the smartphone is turning out to be of significant advantage compared to Wireless Sensor Network and Personal Digital Assistant. In addition to comparing WSN, PDA, and smartphones, I also provide a comparison between the following smartphone adapters: D-Eye, EyeGo and Portable Eye Examination Kit (PEEK).

Most importantly, in this dissertation, I developed an app (Remote Healthcare-Monitoring Mobile App) to help patients who have low vision and who are suffering from diseases which may cause vision loss, such as Diabetic Retinopathy, Cataracts, etc. This app is capable of processing, evaluating, interacting with and storing health data which is continuously measured by (Personal Health Monitors). This new App can transmit information directly to the Smartphone users (patients and doctors) in a way that allows for higher security and privacy. The idea of the App consists of the following: A Smartphone Application, a Data Collection Center, and Professionals in Ophthalmology. Finally, I completed the development of the Mobile app including the Skype and Viber links that enhance the ability to exchange information between the patient and the doctor.

1.6 Organization of the Dissertation

This dissertation is organized into separate sections as follows: chapter 2 discussed the three tiers operating in the remote healthcare monitoring system: Body Area Network (BAN), PAN Coordinator and Back- Medical End System (BMEsys). This section also focuses on several patient PAN coordinators, including Wireless Sensor Network (WSN) that can be used at the fixed tale-monitor location and periodic measurements; Personal Digital Assistant (PDA), used in patients' own home or community setting with continuous measurements; and smartphones that can be utilized anywhere with broad range parameters. This chapter is meant to provide a meaningful utilization comparison between Wireless Sensor Network, Personal Digital

Assistant and smartphone in Remote Healthcare Monitoring System (RHMS) architecture design. Building on the work of the first two chapters, Chapter 3 further discusses and goes into detail about the uses of smartphones apps and smartphone sensors in the healthcare system. Each section in this chapter presents a specific area in which smartphones apps or sensors are used in healthcare services.

Chapter 4 primarily focuses on the causes and symptoms of vision loss of an individual who previously had normal vision. This chapter discusses the two types of vision loss, complete and partial, non-proliferative diabetic retinopathy (NPDR) and proliferative diabetic retinopathy (PDR). Also, an overview is provided for the two types of glaucoma: closed angle and open angle, before moving on to the symptoms and treatments of two other kinds of vision loss diseases, including cataracts and dry and wet macular degeneration. After all these conditions have been covered, the chapter continues with an examination of smartphone adapters for eye-health monitoring, along with applications used in ophthalmology such as the Portable Eye Examination Kit (PEEK), the Eye Go, and D-Eye Adapter.

Chapter 5 discussed the development of a new app (Remote Healthcare-Monitoring Mobile App) to help patients who have low vision and who are suffering from the diseases mentioned above. After discussing the proposed RHM app, the chapter explains how to examine the patient's eye by using the smartphone camera and D-Eye Adapter. D-Eye Adapter is a portable eye and retinal system that attaches to a Smartphone, creating a retinal camera for health screening and evaluation and screening of the eye. Its primary purpose is to examine a patient's retina, optic nerve, and any part of the eye to determine any disorders such as Glaucoma, Macular Degeneration, and any other eye diseases. The last chapter concludes this dissertation and provides the future directions.

CHAPTER 2: ARCHITECTURAL DESIGN SOLUTIONS FOR REMOTE HEALTHCARE MONITORING SYSTEMS

2.1 Introduction

The primary goal of remote health monitoring is to give restorative administrations organizations to anybody at any point, overcoming the necessities of location, time and character. Most commonly, patients have health issues like weight, hypertension, erratic heartbeat, or diabetes. In these cases, people are urged to visit healthcare professionals for routine remedial check-ups regularly. In the mobile healthcare system, patients' information is gathered by the body sensors, and the collected data is transferred to remote healthcare monitoring systems. Using this information, the doctors can optimize treatment, Shetty et al. [12]. The advances in cellular telephones have an enormous impact on our standard life. Among the advantages of smartphone technology are timely processing and consistency, provided to remedial staff via remote access to patients' health data, Varshney et al. [13]. The goal of remote health monitoring systems is to monitor a medical parameter of patients from a location. Because of the technology involved, the implementation of mobile healthcare is not complicated. Wireless communication is used on a substantial basis in the healthcare environment to transfer the data. In an emergency case like a heart attack, the person's health information (such as blood pressure) needs to be updated before the dispatch of the ambulance and the arrival of medical professionals, so a timely transfer of information is critical, Toninelli et al. [14].

This chapter provides an overview of three tiers operating in the remote healthcare monitoring system: Body Area Network (BAN), PAN Coordinator and Back- Medical End System (BMEsys). In addition, this chapter also focuses on several patient PAN coordinators, including Wireless Sensor Network, used at the fixed tale-monitor location and for periodic measurements; Personal Digital Assistant (PDA), which can also be used in patients' own home

or community setting with continuous measurements; and smartphones that can be utilized anywhere with broad range parameters. This chapter is meant to provide a meaningful utilization comparison between these three technologies about RHMS architecture design. It is organized as follows: Section 2 gives the related work description. The next section discusses the proposed network layer architecture of M-health monitoring systems and a brief description of Body Area Network, Personal Area Network and Back- Medical End Systems (BMEsys). Building on the previous discussion, Section 4 describes Multi-Patient Network Coordinators, which includes a description of Wireless Sensor network, Personal Digital Assistant, and smartphone application. The performance evaluation of Multi- patient network is presented in Section 5. Section 6 describes proposed system components. Conclusions are collected in Section 7.

2.2 Related Work

Remote Healthcare Monitoring System (RHMS) consists of the remote observation of patients' well-being and the delivery of therapeutic services. The sensor in RHMs measures the physical parameters and provides information to health organizations. Wireless Sensor Network (WSN) advances are considered a critical research factor in healthcare applications for enhancing the standard of living. Conejar et al. [15] developed a marvelous workforce in human healthcare areas using smartphones, which helps the patient reduce medical expenses while receiving the best care.

Aslantas et al. [16] introduced a state-of-the-art affordable, handy, portable, reliable, pocket-based PC for the wireless monitoring of health and alarm system. Within this system, Human's Electrocardiogram, Body Temperature, and Blood Pressure information are taken into account and sent to a Personal Digital Assistant using IEEE 802.15.1 Bluetooth. The proposed system aims to lower the intervention time for the patient in emergency cases. Accordingly, the

proposed low-cost system can increase the quality of life of the patients. In another study, She et al. [17] proposed a system for medical sensors which are portable depending on the patient's requirement. This feature makes the system flexible for all medical applications such as home monitoring and hospital health care. Aminian et al. [18] provided a system which was capable of monitoring multiple patients simultaneously to check on their physiological parameters, which the Wireless Sensor Network system can transmit from wireless nodes to the coordinator node.

2.3 Architecture of Remote Healthcare Monitoring Systems

This section discusses the technology used to collect and transmit vital physiological data from the patient to the remote monitoring station. The architecture of Remote Healthcare Monitoring Systems (RHMS) aimed to develop a set of modules, which can facilitate diagnosis for doctors through telemonitoring of patients; it also provides the possibility of a continuing investigation of the patient for emergencies looked over by doctors. Medical and environmental sensors monitor both the health and the surrounding environment of the patient. The sensor information is then sent to the server, allowing the doctors the chance to monitor all patient information, Hamida et al. [19] and Alazzawi and Khairallah [20]. The architecture of the RHMS is composed of three tiers: The Body Area Network (BAN), PAN Coordinator and Back-Medical End System (BMEsys), shown in Figure 1.

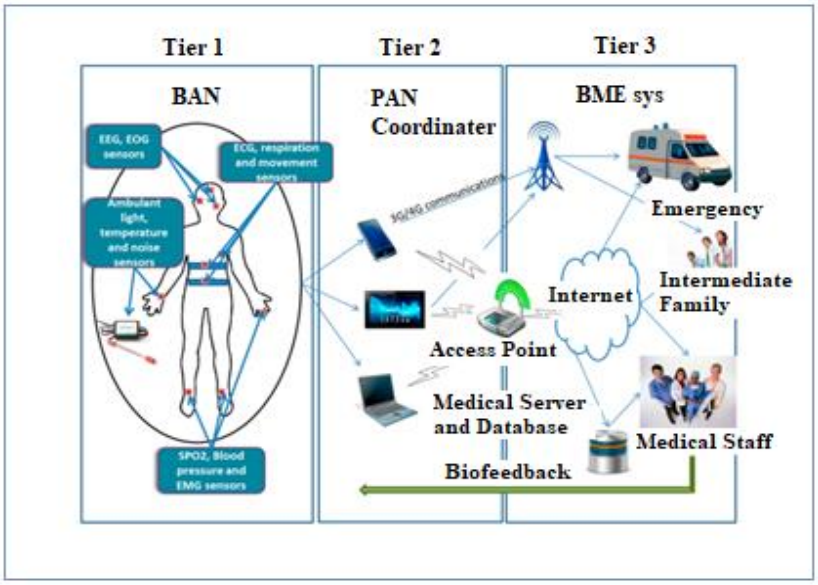

Figure 1: Architecture of the RHMS [19].

The Body Area Network (BAN) consists of wearable sensors. Patients can wear a variety of sensors that collect information from the human body, such as like heart rates, blood pressure, and temperature. The data is collected and transmitted to the PAN coordinator for processing. PAN coordinators such as smart devices then forward health data to remote servers using one-Range communications. Clinical back-end servers receive and process all the patients' data. Depending on the situation, appropriate action will be taken by doctors, Hamida et al. [19], Kamarudin et al. [21], and Shahriyar et al. [22]. Below is the RHM system architecture in details.

2.3.1 Body Area Network (BAN)

Body Area Network (BAN), is a stand-alone structure in which the sensors might be embedded into patient's body. The central idea driving wireless body range system is to evacuate all wireless joining sensors on the patient and to create remote system access between sensors (Figure 2). Different types of sensors include thermal and temperature sensors like Calorimeter, Thermocouple, Thermistor, Proximity & Presences sensors. The different types of data are collected and stored in smart devices, then sent to hospital server via WAN network, enabling

the required suggestions to be given. Each patient's data is stored in the hospital database, Shetty et al. [12] and Ali et al. [23]. Mobile healthcare has built up BAN along with versatile nonexclusive administrations for patients and doctors.

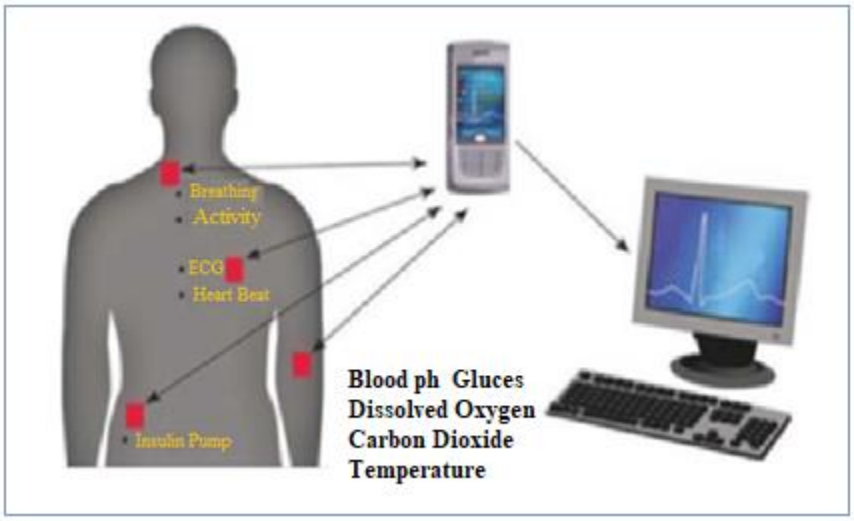

Figure 2: Body Area Network (BAN) [12].

2.3.2 Personal Area Network Coordinators (PANC)

The Personal Area Network Coordinator (PANC) typically involves mobile devices such as a cell phone, a handheld computing device or PDA equipped with an easy to use and intuitive graphical client interface. The PANC can perform the role of investigating the physiological information and decide the client's wellbeing status by taking into account information received from MSS. Figure 3 shows the PAN Coordinator architecture designed for home monitoring application. The different kinds of environmental sensors deployed are an integral part of this personal area network. In this Figure, the PAN subsystem is connected to Wide area networks and mobile devices through gateway subsystems, and the mobile devices carried by the user or sensor nodes with the WAN interface. The network congestion in the gateway is highly dependent on local processing capabilities of PAN subsystems), Ali et al. [23] and Vouyioukas et al. [24].

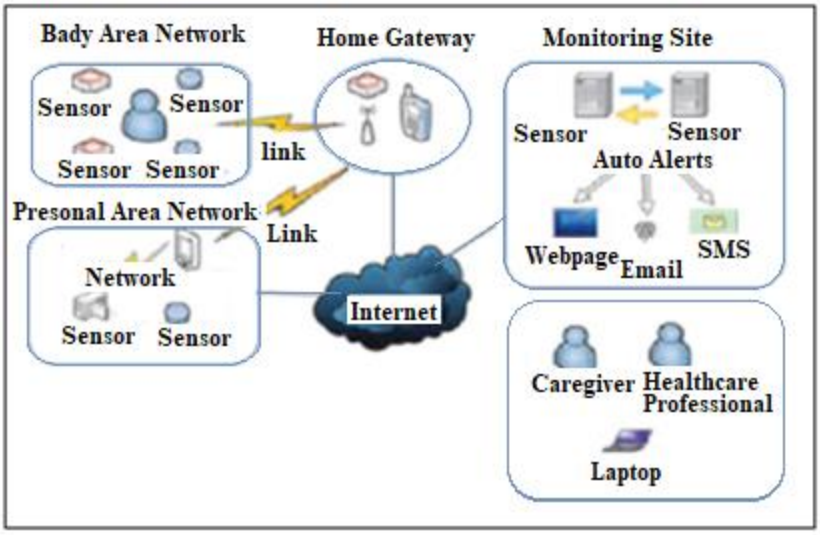

Figure 3: Personal Area Network Coordinator (PANC) [24].

2.3.3 Back- Medical End Systems (BMEsys)

The Back-Medical End System (BMEsys), shown in Figure 4 below, is the backbone of this entire architecture. It receives data from all the PAN Coordinators. Whenever patient data is received, the results are stored in the central database. The BMEsys keeps patient-specific records. It can infer any trend of diseases for the patient, his/her family, and even locality. The Backend terminal has a graphical interface used by the doctors and any emergency contacts associated with patients like ambulance, police, hospital, Khairallah, and Alazzawi [20] and Godara et al. [25]. Ambulances are informed when the patient's conditions cross the maximum limit set on the sensors. Doctors access the data stored in a back-end system to monitor the patient's health. Hospitals fetch the information from back-end data for the full pledged information regarding the patient.

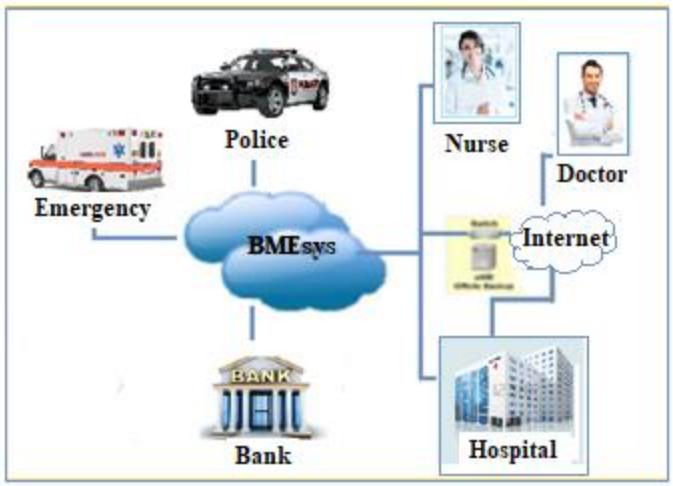

Figure 4: Back-Medical End Systems (BMEsys) [20].

2.4 Multi-Patient Network Coordinators

Multi patients' networks contain more than one patient in the center region. The patients' network coordinators are responsible for gathering and bundling the entry signals from other sensors, and then sending them to the therapeutic focus, Abidoye et al. [26], and Jilt et al. [27]. Fig. 5 shows multi patients PAN coordinators scenario.

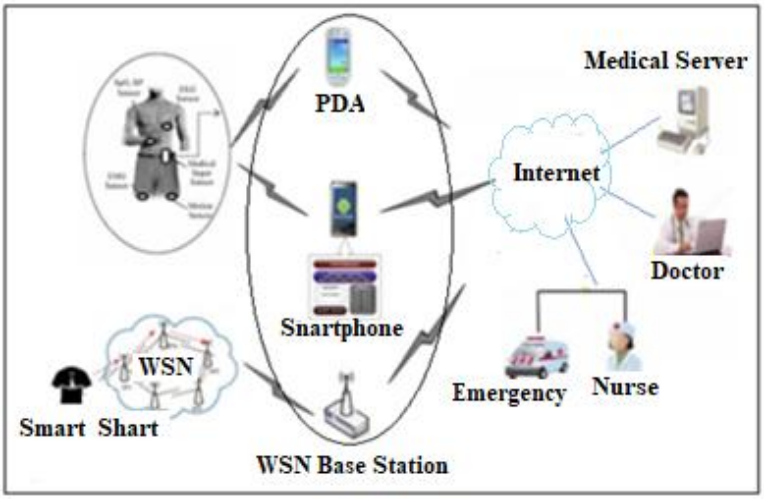

Figure 5: Multi-Patient (PAN) Coordinators [26].

The central region is distinguished by a unique ID which is utilized to recognize every patient in the system. The accompanying sub-areas depict the distinctive sorts of patient organizers

innovations and their usefulness. Wireless Sensor Network (WSN) can be used at altered story screen area and occasional opinions. Personal Digital Assistant PDA can be used as a part of patients' own home or group setting with persistent estimations, and advanced mobile phones can be used anyplace with broad range parameters.

2.4.1 Wireless Sensor Network (WSN)

WSN is a group of sensors which convert a variation in physical quantity into electrical signals with a communication infrastructure for monitoring and recording patients' conditions. Commonly monitored parameters are temperature, blood pressure, and heartbeat rate. Sensor systems take into account physically small sensors trading, for the most part, measured data. Sensor arrangements are made out of wearable and embedded sensors. The sensor networks do not need to be designed and pre-decided. This permits irregular sending in or out of calamity help operations. Then again, this likewise implies sensor system conventions and calculations must have self-sorting out abilities. Another extraordinary component of sensor systems is the helpful exertion of sensor networks. Sensor networks are fitted with an onboard processor. Rather than sending the raw information to the hubs in charge of the combination, sensor networks utilize their handling capacities to locally complete necessary calculations and transmit only the required and halfway prepared information.

Noida et al. [27] and Custodio et al. [28] developed the health sensor devices and the smart shirts which collect and process the physiological parameters, assuming a base station receives all data. The Management subsystem connects to the Information Technology foundation that handles the data connection with every patient. It comprises a management server, which processes and stores all the information related to the patients; and a Graphical User Interface, which permits the healing center staff to screen the status of the patients. The

Base Station architecture serves as coordinator. It is placed in between the Location and Healthcare-Monitoring and the Management Subsystem responsible for carrying information back and forth between the two. Figure 6 shows the architecture of wireless sensor networks. In this figure, the Base station acts as a PAN coordinator.

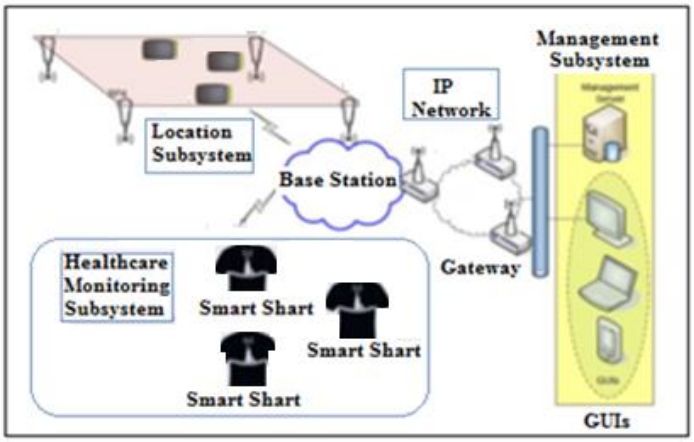

Figure 6: Wireless Sensor Network (WSN) [28].

2.4.2 Personal Digital Assistant (PDA)

The Personal Digital Assistant (PDA) is a small hand-held electronic device that provides computing, information storage and acts as a PAN coordinator that links patients and back-end medical systems. The ECG signals and temperature of individual patients are transferred to a PDA that displays the data collected from sensors and stores it. In an emergency, data is transferred to the central server via wireless communication, Alazzawi and Khairallah [20] and Johnson et al. [29]. Figure 7 shows the PDA layout.

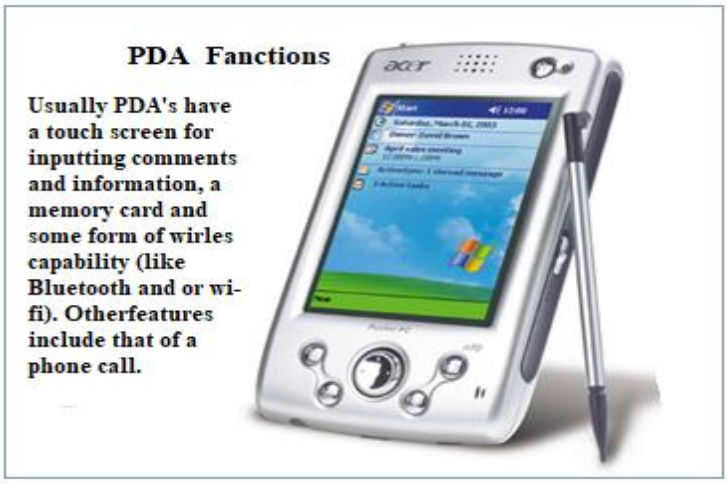

Figure 7: PDA functions [20].

- Application screen: Display the application installed in PDA.
- Battery life: shows how much power in the PDA to run the application.
- Home icon: If we tap the icon, it shows the application installed.
- Menu icon: Tap to view options for PDA or current application.
- Address book: Shows the contact data of people stored in the assistant.
- On-screen keyboard: individual letters and numbers can be tapped to enter the data.

2.4.3 Smartphones

Cell phones are quickly turning into the primary personal computer and specialized gadget in individuals' lives. Today's cell phone not only serves as the key figure in correspondence, but it may also establish its network and become the PAN coordinator. It accompanies a vibrant arrangement of embedded sensors, for instance, an accelerometer, digital compass, GPS, amplifier, and camera; these sensors are empowering another class of uses to rise over a wide assortment of spaces, including, in this case, health care.

A significant portion of the more up to date sensors are added to bolster the client interface (e.g., the accelerometer) or augment location-based services (e.g., the digital compass). Additionally,

they speak to a critical chance to assemble information about individuals and their surroundings. For instance, accelerometer information is well equipped to portray the physical developments of the client in possession of the telephone. The different patterns within the accelerometer can be used to recognize varied activities automatically (e.g., running, strolling, and standing), Nicholas D. et al. [30] and Al-Taee et al. [31]. Figure 8 shows these sensors. Likewise, the smartphone incorporates more traditional gadgets that can be utilized to sense, for example, front and back cameras, a microphone, GPS and Wi-Fi, and Bluetooth radios.

Figure 8: Smartphone Sensors [30].

The mobile phone is sensing architecture shown in Figure 9. This architecture first comprises two possible uses: Inform and Share, which represents a grouping of components with a universal meaning. In Inform, only the user is made aware in the case of personal sensing applications, but a group of users is reported with a community sensing application, hiding the identity of individual users. The Share option is optimized through visualization of data. Web applications that connect with phone sensing applications sometimes allow the sharing of the widely collected data. Social media sites can leverage this data, for example, to enable users to compare fitness data.

In addition to the Inform and Share options, other possible uses include Learn and Share. Sharing of data allows for user engagement, as might be the goal of a social media company, and provides for better user experience. Through Learn, the sensor information collected from user phones can be analyzed by applying machine learning and data mining techniques. During Sense, only smartphones collect data available through the sensors. These sensors include for example the microphone, camera, accelerometers, and GPS. This phase represents purely raw data collection and limited by current smartphone technology, Nicholas D. et al. [30].

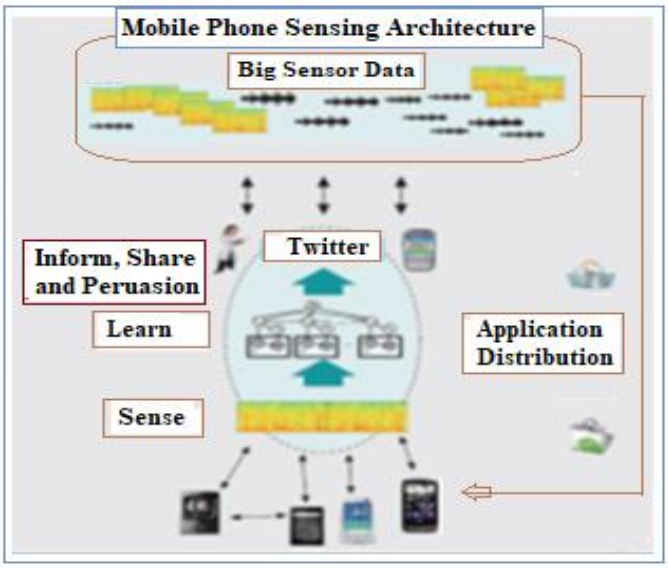

Figure 9: Mobile Phone Sensing Architecture [30].

2.5 Performance Evaluation of Multi-Patients Network Coordinators

TABLE 1: the comparison between the three different coordinators

Attributes	WSN	PDA	Smartphone
Use	Fixed tale-monitor location	Patients home or community setting	Anywhere carried by the patient
Portability	Low/Medium	Medium	High
Purpose	Prevention of long-term reaching hospital	Hospital at home, Short-term	Early hospital admission

Parameters (examples)	BP, Weight, resp., glucose	EVG, SpO2, temp., BP	ECG, EMG, EEG, Body temperature and heart rate
Diseases (Examples)	CHF, COPD, asthma, diabetes	Acute exacerbations of chronic diseases, infections post OP	Unstable angina, pacemaker patients
Measurements	Periodic	Continuous	Continuous
Tele-monitor location	Fixed	Next to	Mobile
Staff assistance	Home: feasible, Residential: not feasible	Yes	Yes
User Friendliness	Medium	Medium	High
Alarm necessary?	No	Yes	Yes
Communications Medium	POTS, ADSL, ISDN	POTS, ADSL, ISDN	GSM, GPRS, Satellite
Data Analysis	Remote most of the time	On exceeding threshold (automated)	On exceeding threshold, on request
Power	Mains	Mains	Battery

2.6 Proposed System Components

Following the assessment and evaluation of multi-patient network coordinators, the framework for the system proposed in this dissertation consists of three segments: sensors, smartphone, and medical end system components.

2.6.1 Sensors

Sensors are in charge of the data gathering process and guarantees that a physical phenomenon, for example, muscle action or blood flow, is initially converted; Prashanth Sensors in the BAN can measure pulse, oxygen level, and glucose level, Shyamkumar et al. [32]. Figure 10 shows the sensor network used in the proposed health care system.

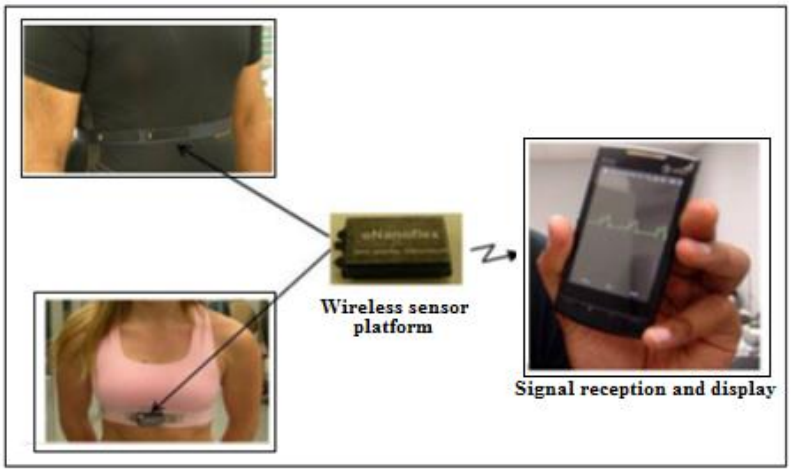

Figure 10: The Wireless Monitoring Sensors Integrated into Garments [32].

This system incorporates a progression of sensors coordinated into a sports bra for women and vest for men. Using a lightweight and wireless module that snaps onto these articles of clothing, the sensors communicate with the system software to gather information and send it over a remote system. Electrical signs and other physiological information accumulated by the sensors are sent to the remote module information, then stream to cell phones and hand-held gadgets, which extends the utilization of the system in healthcare.

2.6.2 Smartphone Applications

Concerning scientific classification of the term apps VS applications and app classification plans, there are three sorts of smartphone applications: native applications, Web applications, and hybrid applications that consolidate highlights found in both native and Web applications.

2.6.2.1 Mobile Browser or Web Apps

Mobile browser or web apps are websites that convey information by utilizing the smartphone's browser. The determination of content is controlled by the logic contained in a system facilitated on a remote server (server-side). Likewise, with desktop-based Web

intercessions, Mobile Health mediations utilizing mobile Web applications can fuse refined levels of intelligence, customization, and engagement tracking.

2.6.2.2 Hybrid Apps

Hybrid apps fuse the features and usefulness found in native applications with the flexibility and productivity connected with utilizing versatile browser applications. They can show program content utilizing a browser that is embedded inside of the native application itself instead of just utilizing the smartphone's browser. Thus, hybrid applications can offer more firmly incorporated environment (envelope) than versatile mobile browser applications for RHM systems.

2.6.2.3 Native Apps

Native apps utilize the advanced components and usefulness made accessible through the cell phone’s operating system (e.g., iOS for iPhones and also Android). For instance, they can utilize GPS-derived area location, system calendar, system alerts, and different notification. Some native applications can work adequately without constant or live Internet access. Since local applications use information accessible through the cellular telephone's operating system, they must adhere to different outline and survey necessities of the organization administering the operating system and can be downloaded using application stores facilitated by the smartphone maker, Alazzawi and Khairallah [20]. Figure 11 shows the three different applications: web, hybrid, and native.

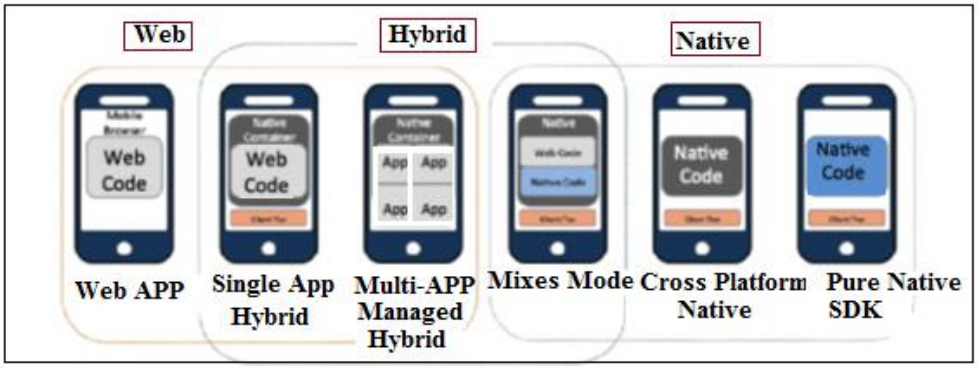

Figure 11: The Three Sorts of the Smartphone Applications [20].

2.6.2.4 Email

Email fits inside of the more broad classification of cellular telephone text notice apparatuses. Like text messaging and IVR calls, email proactively pushes substance to RHM systems members. Naturally, the arrival of email is far less notable because of not having an audible or visible signal. In these circumstances, email might oblige members to search on their drive.

2.6.2.5 Short Message Service (SMS) Text Messaging

SMS messaging includes the conveyance of brief instant messages shared between/among cell phones. Text messaging can reach every single cell phone independently of the service provider and is the most widely recognized non-voice utilization of cell phones. Receiving text messages can acquire charges from the client's cellular plans, even though this expense differs by plan, and unlimited messaging is turning into a more ordinary packaged alternative.

2.6.2.6 Recording Pictures, Audio, and Video

In mobile health applications, users can take photos of the meals they eat using the smartphone camera to determine meal portion size. They can also play tailored music to increase and motivate continued exercise or take images of the carbon monoxide meter in the area to

confirm they are following their no-smoking program, Danaher et al. [33]. Figure 12 below showed a mobile application for quitting smoking.

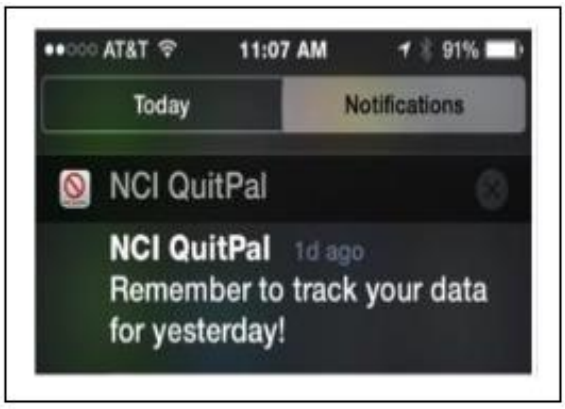

Figure 12: QuitPal App for Smoking Cessation [33].

2.6.3 Medical End System

The body sensor networks gathered the medical user's health information and transferred it to the smartphone via Bluetooth or another application. This, in turn, is transferred to the remote healthcare system center. Using this provided date, medical professionals assist users and aid in saving their lives. Figure 13 depicts the body sensor network. This system has three majors parts, described as follows: Wearable sensors, BSN manager and back-end equipment (smartphone Server). Wearable body sensors continuously transmit the data about the patient's health to the smartphone client, which is then collected and sent to the smartphone server using the cellular network. Finally, the smartphone server provides back support to the staff, Kumar et al. [34].

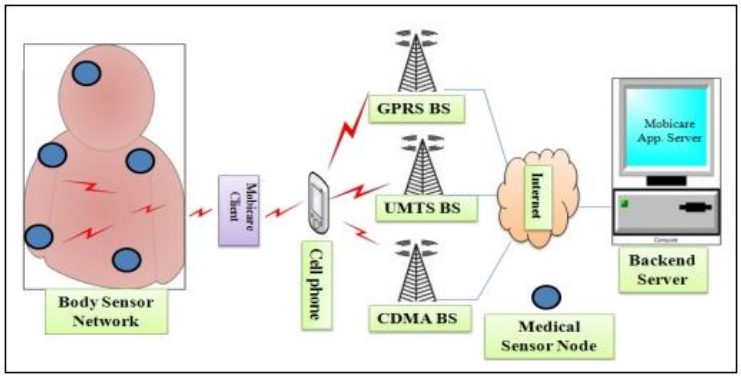

Figure 13: Proposed System Components [34].

2.7 Conclusions

In this chapter, I depicted the development of the Remote Healthcare Monitoring Systems (RHMS), summarizing structured tiers and the different types of sensors through which various patients can be assessed and given the required remedial care on time in the midst of emergencies. Furthermore, I discussed the means through which the data from the patient can be transmitted to healthcare facilities and how suitably the patient can be watched using such mechanisms. Smartphones give the best performance, as they can be used anywhere. They can allow for early hospital admission and continuous measurements of the units. Smartphones are turning out to be of significant advantage compared to Personal Digital Assistant (PDA) and Wireless Sensor Network (WSN). Once the advancement is refined, therapeutic costs for updating interminable remedial conditions will be diminished. We are completing the remote health monitoring system to help not only individual patients but also improve the overall well-being and ensure the flourishing of humanity.

CHAPTER 3: UTILIZATION OF MEDICAL APPS AND SMARTPHONE SENSORS IN REMOTE HEALTHCARE MONITORING SYSTEMS

3.1 Introduction

During the last decade, a significant number of people worldwide have started using smartphones. This new trend is expected to continue growing shortly. One of the most vital areas of usage is in the realm of health services. Some smartphone applications (apps) and smartphone sensors record medical data and work with other apps to try to fix health issues, Paschou et al. [35]. This chapter will discuss in detail the use of smartphone apps and smartphone sensors in the healthcare system; the discussion is divided into sections based on specific type of usage. The first section looks at the use of healthcare apps in patient care and monitoring for the layperson. Section 2 discusses smartphone sensors and the two kinds of sensors in healthcare apps: smartphone-based sensors and add-on sensors. Section 3 considers a few specific sensors used in healthcare apps, sensors such as microphone sensors, accelerometer sensors, and camera sensors. Section 4 discusses the usage of apps and camera sensors in healthcare services. This section describes the role of the camera sensors in apps like My Cancer Diary, Blood Culture, and Point-of-Care HIV Check. Finally, conclusions can be found in Section 5.

3.2 Smartphone Applications in Healthcare

This section discusses the uses of smartphone applications in healthcare systems: the health apps in patient care and monitoring and Health Apps for the Layperson.

3.2.1 Patient Care and Monitoring

Patient care and monitoring are one of many features in smartphones. One example in which patient care and monitoring can be used is monitoring patients with Alzheimer's disease. The Android app iWander performs this task precisely. IWander tracks patients at all times by using the smartphone's GPS. The app inputs the patient's age, level of dementia, and home

location into the software. The app will request that the patient confirm his or her status if he or she is away from home, in certain situations such as unusual tardiness or experiencing lousy weather. Failure of confirmation of status will trigger an alarm that notifies the patient's family, doctor and emergency personnel. It is also possible to detect depression in Alzheimer patients by monitoring their behavior, using Bluetooth, communication patterns, and level of activity from the GPS to evaluate mood. Smartphones can also be used to rehabilitate patients when linked through Bluetooth to a single-lead Electrocardiograph (ECG) device. This feature in smartphones dramatically benefits those who are unable to rehabilitate in hospitals. Instead, it allows patients to rehabilitate at home while being monitored through their smartphones along the way, Paschou et al. [35] and Ozdalga et al. [36].

Another successful example is Diageo, an app that has been proven beneficial to patients with type 1 diabetes. Diageo makes insulin dosing recommendations based on information collected, including self-measured plasma glucose, carbohydrate counts, and planned physical activity. The results of a six-months multicenter study, conducted in France on 180 adult patients with type 1 diabetes with glycated hemoglobin above 8%, show that patients who are using Diageo along with telephone conversations had lower glycated hemoglobin levels than patients who made visits to a clinic.

Blood pressure is usually one of the earliest signs of a health issue. Nowadays, an increasing number of people measure their blood pressure at home, and numbers of apps have been developed to help patients in this effort. Writhing's company has developed a blood pressure cuff that tracks and monitors the entire process. This is accomplished by saving the patient's recorded BP data to his or her phone and sending the results to an online database that can be accessed by any computer connected to the internet. That allows the patient to stay

entirely up to date on his or her blood pressure, as well as to send the measurements to his or her doctor, Paschou et al. [35], Ozdalga et al. [36], and Cha et al. [37]. The Smartphone blood pressure app is shown below in Figure 14.

Figure 14: Blood Pressure Smartphone App [37].

3.2.2 Health Apps for the Layperson

In addition to patient care and monitoring apps, another smartphone feature that is of great use to people includes commonly used health apps. Fitness and weight loss apps are some of the most popular and widely used. These apps help people keep track of the number of calories they consume and determine how much they should burn to achieve their weight loss goal. Two examples of these kinds of apps are Lose It and Calorie Counter. The way these apps can help people with their weight loss goal is by calculation a person's total daily caloric expenditure based on the type and quantity of food consumed, Paschou et al. [35].

The second example is the iTriage application, shown in Figure 15 below; this app provides patients with relevant information such as the locations of nearby emergency rooms, doctors, and other necessary information. It also provides patients with the times of emergency room waits and allows them to register at participating locations. A similar app was developed to improve diagnosis and treatment times of stroke; this app allows patients to make appointments

with participating doctors connected in the application, Paschou et al. [35] and Christina Hernandez [38].

Figure 15: iTriage Apps in Smartphone [38].

3.3 Smartphone Sensors

A significant number of healthcare apps use sensors for data entry. The two kinds of sensors that healthcare apps use are the Smartphone-based sensors and Add-on sensors. Accelerometers, Gyroscopes, Proximity Sensors, Light Sensors, Magnetometers, Barometers Sensors, and Camera Sensors are some of the smartphone-based sensors. Newer smartphones use biometric sensors that include heart rate monitors and fingerprint sensors use external devices such as smartphone cases, Stankevich et al. [39], Nick T. [40], and Rama et al. [41].

3.3.1 Accelerometers

Accelerometers work by sensing the changes in orientation of the smartphone and adjusting it to suit the viewing angle of the user. For instance, a person who wants to widen the screen to view a video will change the orientation of the phone from straight to horizontal. The same mechanism also works for smartphone cameras. Another example is racing games on smartphones; the user can control the car by pointing the phone in specific directions. This feature is possible because the accelerometer sensor senses the change in orientation by 3D (X,

Y, and Z axis) measurements of acceleration of the device concerning free fall, Stankevich et al. [39], Nick T. [40], and Rama et al. [41]. The accelerometer sensor is shown below in Figure 16.

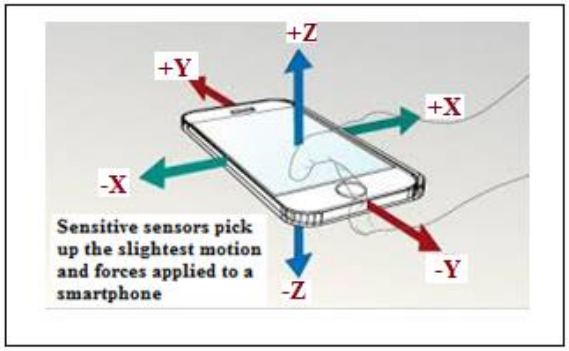

Figure 16: The Accelerometer Sensor in Smartphone [41].

3.3.2 Gyros or Gyroscope Sensors

The gyroscope sensor's role is to control and maintain the position and orientation based on the principle of angular momentum. Gyroscopes and accelerometers together sense the motions of the six axes which are right, left, up, down, forward, and backward. Gyroscope also detects the roll, pitch, and yaw motions which are the angular moments seen from the three axes (X, Y, and Z). Gyroscope sensors help in navigation purposes and with detecting the gesture recognition system in smartphones by using Micro Electrical and Mechanical system (MEMS) technology, Stankevich et al. [39], Nick T. [40], and Rama et al. [41]. Fig. 17 shows the Gyroscope sensor.

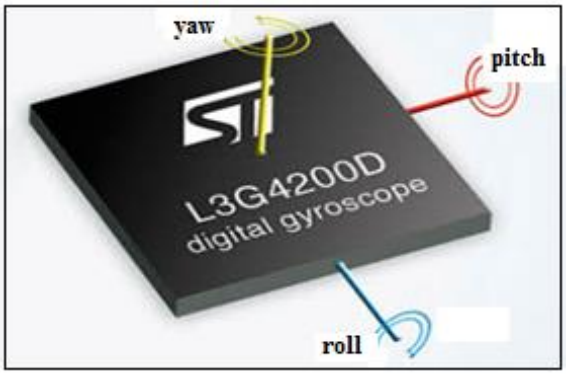

Figure 17: The Gyroscope Sensor in Smartphone [41].

3.3.3 Proximity Sensors

The proximity sensor detects how close a user's body is to the phone. It also detects the position of the ear of the user and turns off the light of the screen when near save the battery of the phone. The proximity sensor also checks the strength of the signal of the phone; it identifies interfering sources if there are any, and amplifies by using the Beam Forming Technique, Stankevich et al. [39], Nick T. [40], and Rama et al. [41]. The proximity sensor is shown in Figure 18.

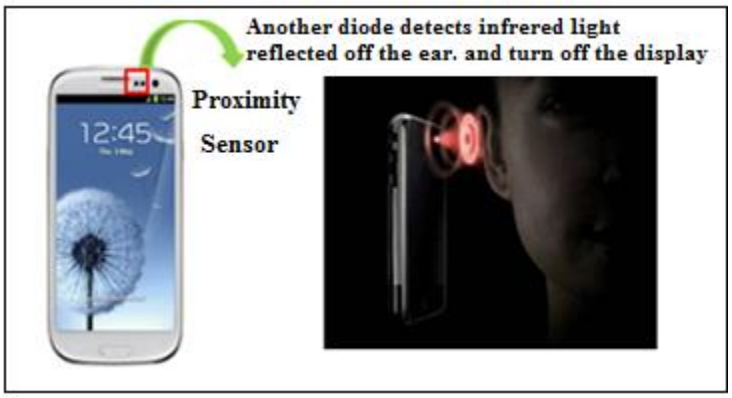

Figure 18: The Proximity Sensor in Smartphone [41].

3.3.4 Light Sensor in Smartphone

Light sensors make the most effective use of the light and are mainly used for brightness on the phone to preserve phone battery by adjusting brightness in ordinary light conditions. Photodiodes, which are very reactive to different colors of light, allow light sensors to alternate the gain and output change of the light on the screen, Stankevich et al. [39], Nick T. [40], and Rama et al. [41]. The light sensor is shown below in Figure 19.

Figure 19: The Light Sensor in Smartphone [41].

3.3.5 Magnetometer Sensors

A smartphone always knows the direction. It can point the user north, south, east, and west at all times, depending, of course, on the physical orientation of the user. This task can be done through the use of a digital compass. The digital compass, based on magnetometer sensors, gives the smartphone information about the user's physical orientation about the earth's magnetic field, Stankevich et al. [39], Nick T. [40], and Rama et al. [41]. The Magnetometer sensors showed in Fig. 20.

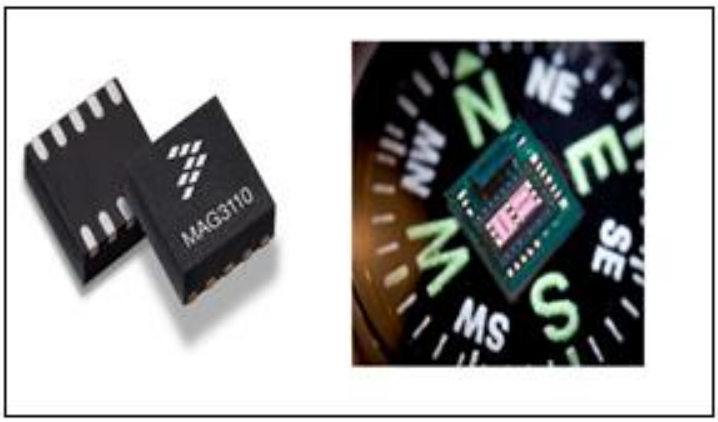

Figure 20: The Magnetometer Sensor [41]

3.3.6 Barometer Sensors

The more technologically advanced phones have a built-in barometer, which is a sensor that can measure atmospheric pressure. With those measurements, the barometer can find out how high above sea level the phone is. This feature provides the phone with better GPS accuracy, Stankevich et al. [39], and Rama et al. [41]. The barometer is shown below in Figure 21.

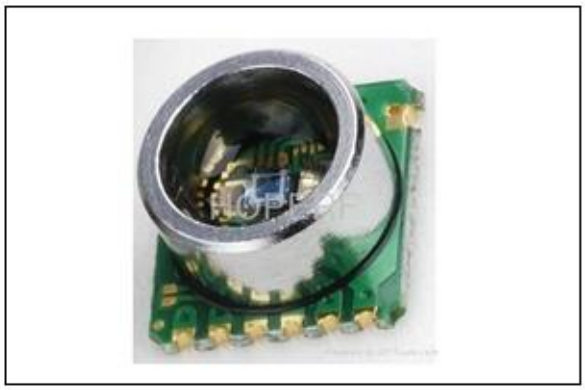

Figure 21: The Barometer Sensor [41].

3.3.7 Camera Sensors

Cameras are one of the critical features of smartphones. The sensor and the lens are the two essential parts of the camera module. The sensor and lens are packaged together into a single unit and are a part of the smartphone's system. The sensor is part of the camera that takes the picture. Its inner workings and functions are complicated; they involve photodetectors, transistors, and power management. The sensor provides the smartphone's camera with all the data that it needs, Tim Schiesser [42] and Terry Relph [43]. Figure 22 shows the smartphone module.

Figure 22: The Smartphone module: Sensor and the Lens [42].

Almost all smartphone camera sensors use the Complementary Metal-Oxide-Semiconductor system (CMOS), which is a form of active pixel. CMOS uses photodetectors that are similar to each pixel in the picture to get information about the photons. This information, in turn, gets amplified and changes into a digital signal relating to the brightness of the photons. In a colored picture, an RGBG Bayer filter is layered over a selection of photodetectors then an inserted software algorithm makes the colored picture. The amount of megapixels in a camera all depends on how many photodetectors there are in the sensor's array, Tim Schiesser [42]. The CMOS image sensor integrated circuit architecture is shown in Figure 23.

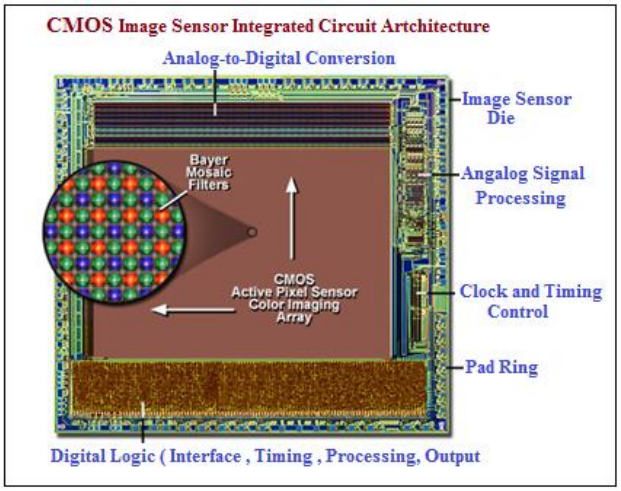

Figure 23: CMOS (Complementary Metal-Oxide-Semiconductor) Sensor [42].

The second essential part of the camera is the lens. The role of the lens is to focus the light on the sensor to make the picture clear. Taking a picture without a lens is possible. However, the resulting photo would be blurry because, without lenses, photons from everywhere would hit the sensor. Each lens has a specific role in focusing the light onto the sensor; one could adjust the light to fit the size of the sensor, while another could correct any issue or deliver the final focus point, Tim Schiesser [42] and Terry Relph [43]. Figure 24 shows the lenses of a smartphone camera. In these lenses, each element pictured has a particular, specialized function in focusing the light onto the sensor, whether that is generally sharpening the light to fit the size of the sensor, correcting issues, or providing the final focus point.

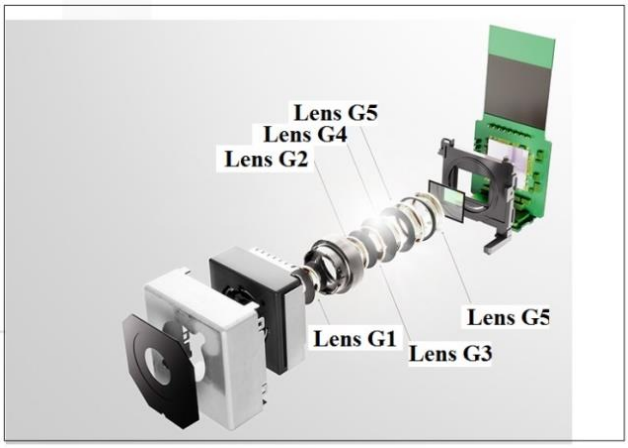

Figure 24: The Lenses of Smartphone Camera [42].

3.4 Smartphones Sensors in Health Applications

Currently, smartphones have incredibly influential as one of the most popular means of communication amongst people. Smartphones have many different ways of communication, such as GSM, Wi-Fi, Bluetooth, to name only a few. In addition to day-to-day communication, smartphones also have other beneficial features, including healthcare apps. These applications function because of embedded sensors in the phone. All the sensors in a smartphone are divided into two groups. The first, environment sensors, measure the different features of a smartphone,

for example, the microphone and camera. The second group is the position and orientation group which includes the accelerometer, digital compass, gyroscope, and GPS all the features that concern the orientation and location of a phone. Healthcare apps use a microphone, camera, and accelerometer sensors, Hong, et al. [44] and Ketabdar et al. [45].

3.4.1 Microphone Sensors in Healthcare

Microphone sensors are critical in healthcare apps; one of their primary uses is for communication. Healthcare apps can significantly benefit people in developing countries, particularly when it comes to communication and training for health care workers, disease, diagnostic and treatment support. The microphone can help patients with myotonic syndrome by merely monitoring them. The myotonic syndrome is a disorder characterized by a slow relaxation of muscles after contraction. One way treatment can be provided if patients call data collection services to discuss their health after eight weeks. The microphone classifies symptoms into four groups: muscle stiffness, weakness, pain, and tiredness, Raso et al. [47] and Bravo et al. [48].

3.4.2 Accelerometer Sensors in Healthcare

Accelerometer sensors for healthcare apps are usually used to monitor a person's level of physical activity. This feature is vital because it reduces the risk of chronic diseases. Specially designed accelerometer-based devices can measure the level of activity based on the exercise performed by a person. Patients do not have to visit a rehabilitation center; they have the option of rehabilitating at home thanks to project m-physio, which provides rehabilitation services through a smartphone. There are two stages to this system. The first one is the system learning a supervised rehabilitation and the second is an independent rehabilitation. The system can help a patient by informing him or her of whether they are performing exercises correctly; it categorizes

a patient's activity into four types: correct exercise, wrong exercise, exercise exceeds the maximum time, exercise does not exceed the minimum time, Statland et al. [46] and Raso et al. [47].

3.4.3 Camera Sensors in Healthcare

Cameras are the most popular features of a phone; cameras take pictures and record videos. The camera sensor can provide beneficial information about a patient and use it in a healthcare app. A prime example of its usage is in the field of teledermatology, where pictures of patients' skin are used to help a doctor make a diagnosis. There are many healthcare apps concerning the field of teledermatology. An example of one is ClickMedix, Ketabdar et al. [45] and Bravo et al. [48].

3.5 Using Smartphone Apps and Its Camera Sensors in Healthcare

Camera sensors are of the utmost importance when it comes to healthcare apps. In fact, many healthcare apps (such as My Cancer Diary, Blood Culture, and Point-of-Care HIV Check) use camera sensors to treat patients, apps

3.5.1 My Cancer Diary

My Cancer Diary (MCD) is an app for cancer patient self-management. Patient self-management apps generally provide patients with information about their disease or with appropriate assistance as needed. MCD uses the camera to acknowledge a patient's barcode for logging in. The app then provides a patient with three types of information. First, it offers general information about the disease; this may include anticancer drugs, severe symptoms that come along with the disease, and the most common questions asked by patients concerning the disease. The second service involves patient assistance like symptom management. The third type of information provided is about professional assistance, for example, cancer education,

Pandey et al. [49], Jang et al. [50] and Park et al. [51]. My Cancer Diary app is shown in Figure 25.

Figure 25: My Cancer Diary app (A) Patient Barcode Sample
(B) Screenshot of Main Menu [51].

3.5.2 Blood Culture

Blood Culture (BC) is an app that helps inform patients of the proper way and time to sample blood. A camera identifies patients' barcodes and makes timestamps for the blood. This app can be used by logging in using a hospital staff ID and scanning the patient and sample barcodes, Park et al. [51]. Figure 26 shows how the blood culture app identifies the patient's barcode using the smartphone camera.

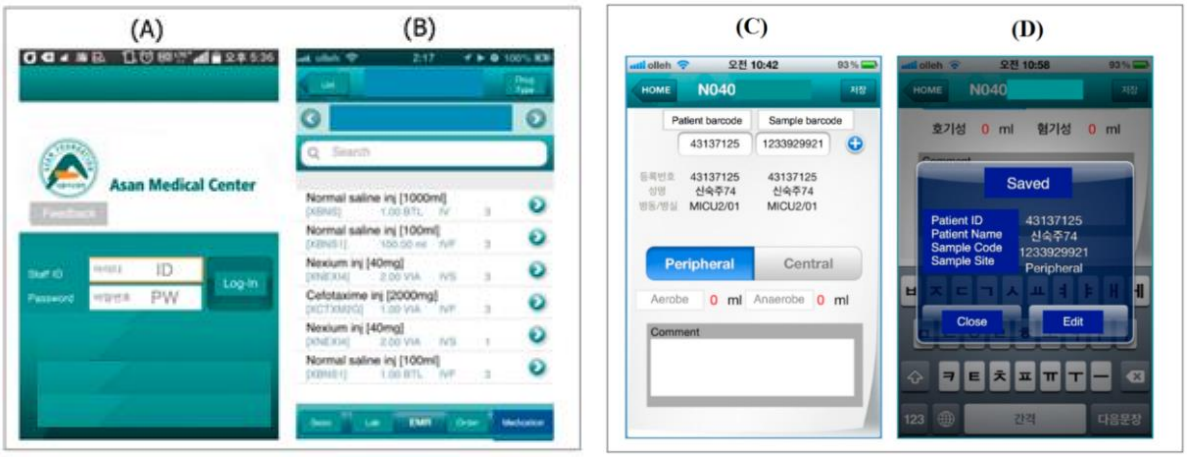

Figure 26: Blood Culture App (A) Barcode Reading Menu; (B) Barcode Reading Phase;(C) Reading Results and Entering Data such as Sample Site and Volume; (D) Saving Data, ID and Patient Name Were Used [51].

3.5.3 Point-of-Care HIV Check

Point-of-Care helps to improve patient safety. Point-of-Care checks patients for HIV infections in the hospital. Doctors call numerous patients who have not been previously evaluated for the possible infectious disease to come to the hospital and get evaluated. This app helps users send their samples to their local hospitals for a low cost. Similarly to the previous apps discussed, Point-of-Care is also able to identify a patient by using his or her barcode and can take a picture of the results by using the phone's camera, Park et al. [51]. Figure 27 shows the main menu of the app and the barcode reading phase.

Figure 27: Point-of-Care HIV Check App (A) Screenshot of Main Menu; (B) Screenshot of Barcode Reading Phase [51].

3.6 Conclusions

In Section 1 of this chapter, I have explained how healthcare apps have been used for patient care and monitoring and the layperson. Healthcare apps for patient care and monitoring were able to help and monitor patients with various diseases; one such example is patients with Alzheimer disease. Healthcare apps help such patients by tracking them during the time of wandering or confusion. Healthcare apps have also been beneficial to the layperson. A few of their many uses involve keeping users healthy and in physical shape, and they can also provide relevant medical information. Section 2 discussed smartphone sensors, including the categorization of the different types. The different kinds of smartphone sensors are smartphone-based sensors and add-on sensors. Each one of these categories includes multiple different sensors within them. This section discussed the smartphone-based sensors which include accelerometers, gyroscope, proximity, light, magnetometers, barometers, and camera sensors along with the add-on sensors, which include sensors such as smartphone cases and explained their purposes in healthcare apps. Section 3 focused on the two types of sensors used explicitly in healthcare apps. The first type is environment sensors, which include the microphone and camera sensors. Afterward, this section also explained the position and orientation group of sensors, which includes the accelerometer sensor. Uses for both types were explicitly considered about healthcare applications. Section 4 focused on the uses of camera sensors in healthcare apps and the many apps that use camera sensors to help patients, such as My Cancer Diary, Blood Culture, and Point-of-Care HIV Check. Camera sensors in these apps help the patient in many ways: they provide valuable medical information, educate patients about sampling blood, and help evaluate various conditions for patients.

CHAPTER 4 REMOTE SCREENING AND SELF-MONITORING FOR VISION LOSS DISEASES

4.1 Introduction

In the modern information and communication era, the technology of the smartphone application is one of the most advanced and rapidly developing areas. These applications and their technology in the field of medicine are advancing every day. This demonstrates the benefits of using smartphone healthcare apps for medical treatment in the field of Ophthalmology. Currently, smartphones and new apps have an enormous impact on our everyday life. Some of the most significant advancements include timely care and consistency in remote patient data access for remedial staff. Smartphones and mobile devices are present in people's daily lives, allowing a continuous and fast flux of information. Nowadays, almost everybody has a smartphone, and new apps are being developed continuously and improved. This, coupled with the fact that we are always trying to improve our health care systems, makes it imperative that we use new technologies to promote early diagnoses and more efficient treatment.

Smartphone cameras and applications can alter healthcare by providing doctors with the option to diagnose a patient beyond traditional treatment equipment, as well as being able to assist patients in receiving beneficial information and management of health, Melissa et al. [52]. In Ophthalmology, images captured with the camera sensor on the smartphone can provide as much useful information as a doctor consultation. We can take advantage of this technology to treat vision loss, a common disease that can occur suddenly or gradually. Vision loss differs from blindness, as blindness occurs at birth whereas vision loss occurs over time. The causes of vision loss could range naturally from conditions affecting the eye to conditions affecting the part of the brain that works the eye.

This problem is most familiar with age; in the case of the elderly, glaucoma, cataracts, and other eye diseases are the leading factors of vision loss, Melissa et al. [52]. In this case, the camera sensor works to provide accurate images of the eye which later can be analyzed by a medical professional. This idea is realized in the following process: First, the patient captures a photo of his/her eye and then sends it to the data collection server. Subsequently, medical experts gain access to the photos sent in by the patients and prescribe appropriate treatment to the patient. The smartphones apps system consists of the following parts: a smartphone application, a data collection center, and medical professionals. To utilize this service; the patient must register in the system, after which he or she instructed on how to take photos or video of the affected eye correctly; using the appropriate app, the patient can then take photos or videos of his/her eye and send them to the data collection server. Finally, the doctor gains access to this data, then prescribes a treatment according to medical analysis.

This chapter discusses how a smartphone and its technology can be used to diagnose loss of vision due to Diabetic Retinopathy, Glaucoma and other eye diseases. Most recent smartphones have been equipped with a featured camera with high megapixels and advanced sensors which can be used to record fundus photographs through a slit lamp or record videos from an operating microscope and display images from optical coherence tomography systems and other high-tech devices. The ophthalmologists can share these images and analyze them with colleagues utilizing media sharing applications, thus ensuring the optimal diagnostic and therapeutic results.

At present, three widely used adapters can improve the magnification and lighting of the camera, which enables smartphones to capture high-quality images of the eye. These are the Portable Eye Examination Kit (PEEK), EyeGo, and D-Eye. Peek Adapter consists of a

smartphone application and retina adapter which can be clipped onto the device and synchronized with the peek application for sharing and analyzing the images. This adapter can be used by anyone, anywhere to examine patients' eyes. EyeGo is an adapter intended to allow ophthalmologists to capture high-quality images of the eye using the ophthalmic lens. D-Eye Adapter is one of the extensively used adapters which yield excellent results. It consists of a portable eye and retinal system that fits onto a smartphone, creating a retinal camera for evaluation and screening of the eye. It uses LED lights as a light source and requires no extra power, making it an ideal solution for portable diagnostics. The medical field has widely accepted these smartphone adapters for diagnosing low vision and eye-related infections. This chapter is to provide a meaningful utilization comparison between these three smartphone adapters.

This chapter primarily focuses on the causes and symptoms of vision loss for an individual who previously had normal vision. It is organized into the following nine sections: After the introduction, Section 2 discusses how a smartphone and its technology can be used to diagnose loss of eye vision like Diabetic Retinopathy, Glaucoma, Cataracts, and Age-Related Macular Degeneration. It discusses vision loss and its two types: complete and partial. Section 3 includes Diabetic retinopathy and its two types: non-proliferative diabetic retinopathy (NPDR) and proliferative diabetic retinopathy (PDR). It also includes the symptoms of and treatments for the condition, such as laser surgery, medication injections, and vasectomy surgery. Section 4 includes cataracts and treatments for this condition such as eyeglasses, anti-glare sunglasses, and magnifying lenses. Section 5 provides information for macular degeneration and its two types: dry and wet (AMD). It also provides symptoms and treatments for it, including Anti-VEGF medication injections, photodynamic therapy, and laser surgery. Section 6 discusses glaucoma.

Specifically, it will include the two types of glaucoma, closed angle and open angle, and possible treatments of the condition. Section 7 and 8 discuss the smartphone adapters for eye-health monitoring along with the applications used in ophthalmology. They provide information about the three camera adapters, portable eye examination kit (PEEK), EyeGo, and D-Eye. Also, these sections provide a meaningful utilization comparison among these three smartphone adapters. Section 9 will wrap up the work of the previous eight sections with a conclusion.

4.2 Related Work

For over a decade, there have been advances in the field of smartphone technology. Smartphones nowadays are no longer considered simple phones; instead, they are seen as handheld computers with powerful capabilities, high-resolution screens, and, most importantly, operating systems that allow the development of applications. Current operating systems for smartphones include Android and IOS, Moradian et al. [53] and Chhablani et al. [64].

Smartphones can run multiple applications, including those used for medical specialties. This justifies a significant attraction for physicians, due to the simplicity of the natural and valuable communication and the patient made possible by the simple use of an application. A survey conducted in 2010 shows that 83% of members used their smartphone to accomplish professional tasks. Currently, staying on top of all the vast and latest medical information available out there is no easy task. However, if a medical application can gather and prepare data to present it comprehensively and straightforwardly, then it would undoubtedly prove a valuable and efficient use of a physician's time, Moradian et al. [53] and Lord et al. [55].

Ophthalmological applications have altered smartphones into medical devices and tools that are commonly used by ophthalmologists to evaluate their patients. Currently, eye caregivers have a great many applications at their disposal. Some applications are used for ocular surgery,

with the goal of enhancing surgical skills or better patient education. Smartphones have reduced the tools required for examination to a small portable adapter. Although they cannot replace office-based testing equipment, they prove helpful in providing fast and accurate data. Also, they can also provide videos, diagrams, and photographs that are convenient to showing treatment methods, Moradian et al. [53], Chhablani et al. [54], Zvornicanin et al. [57] and Bastawrous et al. [58].

The use of smartphones has allowed quality images to be captured in non-clinical settings. These images can be used to document visible images of the eye. The technology has also been successful in capturing quality images and videos during medical procedures. Smartphones can capture images and tomography scans directly from the display of a computer monitor. They can also record videos and send emails regarding patient information through secure encryption. Smartphones are effective in achieving goals in research by capturing high-resolution pictures to be evaluated by an expert. This does not only make treatment more comfortable, but it reduces cost and saves time, Bastawrous et al. [58] and Gillingham et al. [59].

Mohammad et al. [60], Chakrabarti et al. [61] described how cataract treatment becomes more expensive as the stages of the disease advance. The uses of smartphones in cataract treatment have reduced the cost of treatment by eliminating the need for expensive devices and saving time. Similarly, glaucoma treatment has dramatically improved because of the use of smartphones and their applications. Accessories added onto the smartphone such as the D-Eye adapter allow patients to receive the same results without the use of advanced machinery. Furthermore, smartphones have revolutionized the way physicians treat patients with Age-related Macular Degeneration disease. As with the other conditions discussed, new technologies reduce

the difficulty of treating macular degeneration by allowing patients to manually self-monitor their eye through the use of adapters.

Silva et al. [62] describe how smartphones have improved the cooperation and examination of patients diagnosed with diabetic retinopathy disease by 50% to 70% just due to how much time and money was saved in the process. This is causing regular follow up and examination between the patient and the physician, ultimately leading to enhanced, faster treatment. Russo et al. [63] developed a small optical device called D-Eye. D-Eye attaches magnetically to a smartphone, and, once attached, is used to examine and record photos and videos of the retina. The device uses a wireless connection or other possible means of internet connectivity to make it possible for users to obtain retinal photos in even the most remote locations.

Ting et al. [64] examine the use of Eyescan, a multi-purpose ophthalmic imaging device. Eyescan proves to be easy to use, and its methods of screening for diabetic retinopathy are accurate. Haddock et al. [65] describe in detail the Fundus photos of human and rabbit eyes using iPhone 4 and iPhone 5 smartphones. Vyas et al. [66] use a smartphone instead of a beam-splitter for recordings of various surgical procedures. Smartphones proved useful in recording ophthalmic surgery videos because they are very convenient, easy to use, and cost-efficient. All those traits would make smartphones the preferred tool of health professionals for surgical recordings.

Ludwig et al. [67] assessed the ability of ancillary services' staff to use EyeGo to take high-definition photos of eye findings. The ophthalmic imaging system's ability to capture high-quality photos makes it preferable to patients and healthcare professionals alike.

Building on these previous studies; the work was undertaken in this dissertation is meant to demonstrate how a smartphone can be used for medical treatment in the field of Ophthalmology. Smartphone applications and their high-quality camera sensors have potential in the field of medicine and Ophthalmology; they can be used to capture high-quality images that allow doctors to detect and treat patients even in the case of life-threating conditions, all from non-conventional settings.

4.3 Diabetic Retinopathy

Diabetic Retinopathy represents the most common diabetic eye disease. It occurs when the blood vessels change in the retina, swell and leak fluid. Also, new blood vessels appear on the surface of the retina, a sensitive tissue at the back of the eye that transfers images to the brain through the optic nerve. There are two types of retinas: the peripheral retina, which is responsible for vision through the corners of the eyes; and the macula retina, which is responsible for vision through the center of the eye such as reading a book or dialing a phone. Diabetic retinopathy often results in complete vision loss, and victims do not often notice its effect in its early stages, but only as it progresses. Once it has taken effect, there is no reversing the disease, Kambiz Negahban [68] and Madeline et al. [70]. Figure 28 shows a comparison of healthy vision and abnormal retina of diabetic retinopathy.

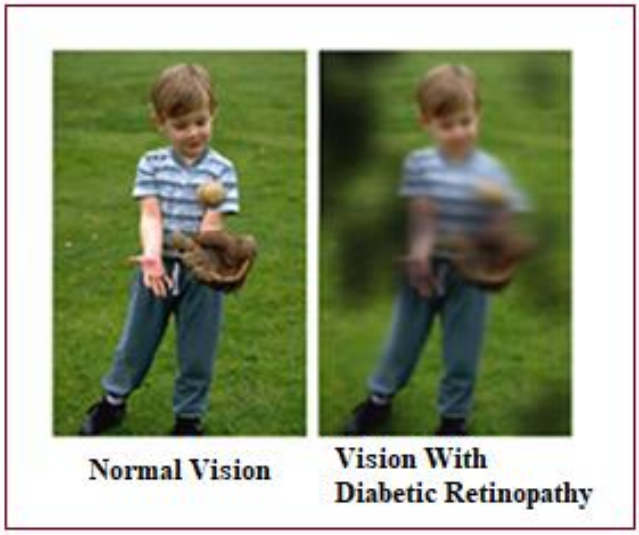

Figure 28: A normal eye and eye with diabetic retinopathy [68].

4.3.1 Diabetic Eye Problems

Diabetic patients could experience two types of Diabetic Retinopathy: Non-proliferative Diabetic Retinopathy (NPDR) and Proliferative Diabetic Retinopathy (PDR). (NPDR) can cause the following changes in the eye: Microaneurysms, Retinal Hemorrhages, hard Exudates, Macular Edema, and Macular Ischemia. The Microaneurysms are small parts in blood vessels of the retina that leak fluid. The Retinal Hemorrhage consists of drops of blood that leak into the retina. The Hard Exudates are cholesterol or fats that have leaked into the retina. The Macular edema is the swelling of the Macula due to fluids leaking from the blood vessels. Once it has reached this stage, the retina no longer functions appropriately. Macular edema causes vision loss in people with diabetes. The macular Ischemia occurs when the small blood vessels completely close off, causing blurry vision because the macula no longer receives blood to function correctly.

(PDR) can cause the following changes in the eye: the Vitreous hemorrhage is the when blood leaks into the vitreous, which is between the lens and the retina. The Traction Retinal Detachment occurs when the original position of the retina is pulled due to the shrinking of a

scarred tissue. Neovascular Glaucoma is the building up of pressure in the eye, which would obstruct the flow of fluid out of the eye because of a certain amount of closed retinal vessels. During this stage, pressure builds up, and damage is caused to the optic nerve, Jesse et al. [69], Madeline et al. [70], and Kiersten et al. [71]. The two types of diabetic retinopathy are shown in Figure 29.

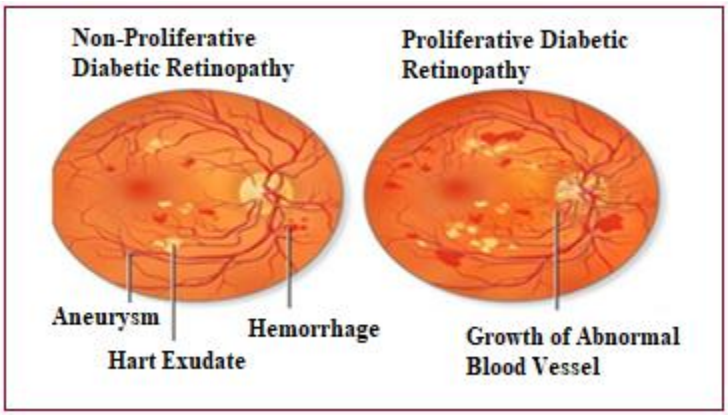

Figure 29: Non-Proliferative (NPDR) and Proliferative Diabetic Retinopathy (PDR) [69].

4.3.2 Diabetic Retinopathy Symptoms and Risk Factors

Diabetic Retinopathy symptoms include dots, dark strings floating in vision, and washed out colors. Diabetes can alter vision in an individual's eye even if he or she does not have retinopathy. Changes in blood sugar change the structure of an eye's lens, and the image on the retina will be out of focus. Blood sugar levels, lipid levels, blood pressure, and duration of diabetes influence the severity of diabetic retinopathy, Kierstan et al. [71].

4.3.3 Diabetic Retinopathy Treatment

The treatment for Diabetic Retinopathy is to control the blood sugar level as it will reduce the risk of vision loss and may slow the progress of vision loss. Treatments can vary depending on the stage of the disease and the location of the affected area in the eye. The treatment options are laser surgery, medication injections, and vitrectomy surgery. Laser surgery

is the process in which a bright light passes through the cornea, lens, and the vitreous without harm, Kierstan et al. [71]. The laser shrinks new abnormal blood vessels and reduces macular swelling. Medication injections help to slow the growth and progression of new abnormal blood vessels. Last is the vitrectomy surgery, which is the process in which a microscope is used to remove blood and scar tissues that support abnormal vessels in the eye. This surgery removes the vitreous hemorrhage allowing light rays to interact with the retina. Vitrectomy surgery often prevents future hemorrhages by removing the vessels that are causing the bleeding. Vitrectomy surgery also removes the damaged tissues, causing the retina to return to its normal position. Medication injections are another way to treat diabetic retinopathy. During this process, medication is injected into the center of the eye (vitreous) to reduce swelling and leakage in the retina, and this method of treatment may improve vision, Klein et al. [72], Marilyn et al. [73], and Wilkinson et al. [74]. The three types of treatment are shown in Figure 30.

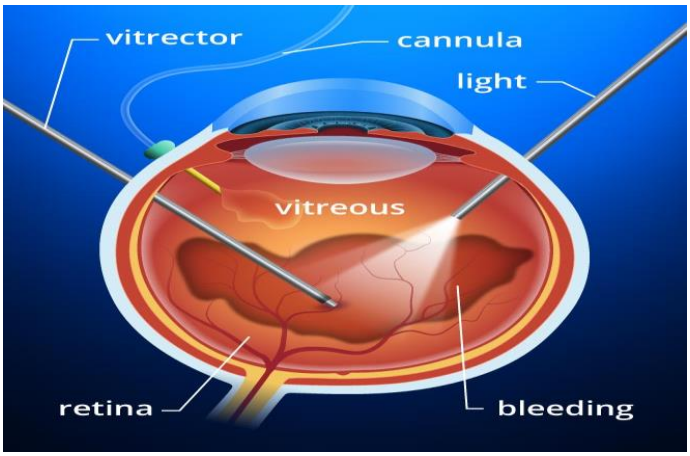

Figure 30: Treatment options: Laser surgery, Vitrectomy surgery and Medication injections [73].

4.4 Cataracts

Cataracts are the process in which the lens of the eye fogs up from time to time. The lens is the bright part of an eye that focuses on images for the retina. In a healthy eye, the transparent

lens receives an image that is sent to the retina, which in turn will send signals through the optic nerve to the brain, where the image is interpreted. The lens must be clear, to begin with; if the image is foggy, then the image will have a blurry effect to it. Cataracts can occur in either eye and do not transfer from one eye to the other. The leading cause of cataracts is age, and their different types are [1] secondary cataracts, which can form post-surgery due to eye problems such as glaucoma; [2] traumatic cataracts, which can be formed after an eye injury; [3] congenital cataracts, which are developed in infants through birth or during childhood; and, finally, [4] radiation cataracts, which can be developed after being exposed to radiation, NEI [75], Gretchen et al. [76] and Joseph et al. [77]. The eye with cataract is shown in Figure 31.

Figure 31: Eye with clear lens and eye has cataracts [77].

4.4.1 Cataracts Causes and Risk Factors

The primary objective of the lens located behind the pupil is to absorb light onto the retina where the image is created. This lens is contained water and proteins precisely arranged for light to pass through. As we age, these proteins may tend to come together to cause fogging in our vision. Risks of cataracts include certain diseases such as diabetes, as well as personal behavior such as alcohol use and smoking, Gretchen et al. [76] and Joseph et al. [77].

4.4.2 Cataracts Symptoms

Most common symptoms of cataracts are a blurry vision and washed out colors. As individuals progress through the stages of cataracts, less and less light reaches the retina, causing difficulty in vision. Cataracts also affect color vision: colors may begin to seem washed out, and it may become difficult to distinguish between specific colors. Being a victim of cataracts alters sensitivity to light. Bright colors may become too bright and dark colors may become too dark. Diplopia and double vision are symptoms of cataracts when using one eye as a source of vision, Joseph et al. [77].

4.4.3 Cataracts Treatments

The symptoms of cataracts can be improved with the following treatments: eyeglasses, anti-glare sunglasses, and magnifying lenses. If those treatments do not help improve vision, then surgery may be needed. Surgery for cataracts is done to each eye separately with a waiting time of four to eight weeks in between them. The most common surgery for cataracts is known as phacoemulsification. During this type of procedure, the surgeon makes incisions in the eye and breaks up the lens using ultrasound waves. The lens is then taken out of its position and replaced with an intraocular lens (IOL). Another type of cataracts surgery is known as other capsular cataracts surgery. This form of the procedure involves a large incision to remove the foggy lens in one piece rather than breaking it up. These surgeries often eliminate the need for eyewear or contact lenses, Joseph et al. [76].

4.5 Macular Degeneration

Macular Degeneration causes of vision loss among people aged 50 and older. This disease causes damage to the macula, which is a small area near the center of the retina. The retina's central portion is responsible for central vision within the eye. It controls our ability to

read a book, recognize faces and colors, and see objects in exquisite detail. The macula is the sensitive part of the retina, which is located at the back of the eye and provides sharp vision. Its primary function is to turn light into signals sent through the optic nerve to the brain where they turned into images. When the macula is damaged, the field of vision may appear blurry, distorted, or dark, Andrew A. Dahi [78] and Jager et al. [79]. Macular degeneration is shown below in Figure 32.

Figure 32: Normal macular degeneration and anatomy of a normal eye and loss of central vision [78].

4.5.1 Types of Macular degeneration or (AMD)

The main types of macular degeneration are dry and wet. Dry degeneration, the most common type, is the process in which cells within the macula gradually become thin or tend to break down, which is known as atrophy. Dry macular degeneration slowly reduces central vision and affects color perception. The condition can go unnoticed if it only occurs in one eye; this is because the healthy eye may compensate for vision loss. Wet macular degeneration, the less common form, can cause more damage to central vision compared to the dry type of (AMD), making it more severe. This type of (AMD) occurs when fluids leak from blood vessels under the macula, blurring central vision and causing rapid vision loss. Symptoms for this form of

(AMD) progress rapidly, Maureen et al. [80], and Lylas et al. [81]. The two types of macular degeneration are shown in Figure 33.

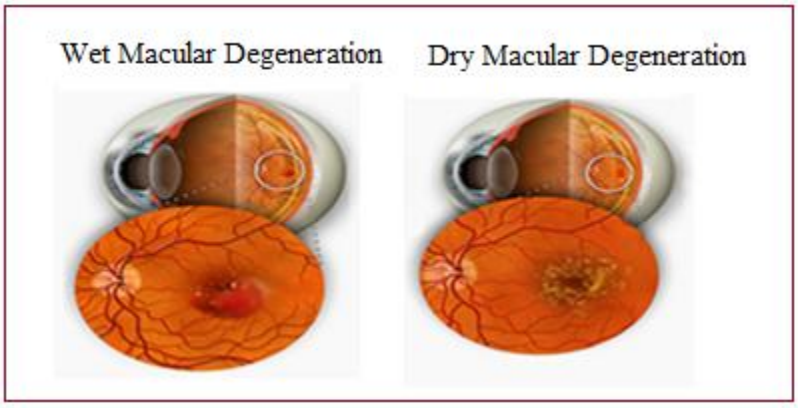

Figure 33: The two types: Wet and Dry of macular degeneration [81].

4.5.2 Symptoms and Risk Factors for (AMD)

As (AMD) progresses, loss of vision occurs in the form of blurred areas or blank spots in vision. However, the two types of (AMD) don't necessarily share symptoms. Dry (AMD) symptoms include blurred vision, a need for increased light for close up vision, and a faded and less vivid appearance of colors. Wet (AMD) symptoms include loss of central vision, dark spots that appear in vision, and the appearance of the same objects as different sizes between the two eyes. Age is the leading factor for (AMD) and is more likely to occur after the age of 60. Other factors include smoking, high blood pressure, obesity, and high cholesterol, Marilyn Haddrill [82]. Figure 34 shows the symptoms of macular degeneration.

Figure 34: The symptoms of macular degeneration [82].

4.5.3 Treatment Options for (AMD)

When the macula becomes too dry, there is no treatment once it has reached the advanced stage. However, there are methods to possibly delay and prevent intermediate (AMD) from progressing to the advanced stage in which vision loss occurs. Wet (AMD) can be treated with the following: anti-VEGF medication injections, photodynamic therapy, and laser surgery. None of the treatments listed above will cure the wet (AMD), but they do slow further progression of vision loss. Anti-VEGF medication injections involve a process in which medications are injected into the eye to block the development and leakage of new abnormal blood vessels. This method of treatment has been proven effective numerous times with patients that have vision loss. Laser surgery involves a high-energy laser light that diminishes the growth of abnormal vessels to prevent problems such as leakage and bleeding. Photodynamic therapy involves a combination of a light-activated drug called a photosensitizer and a low power cool laser. The photosensitizer is injected into the bloodstream and is absorbed by the abnormal blood vessels, followed by a cold laser which activates the drug causing the abnormal vessels to close off, Marilyn Haddrill et al. [82].

4.6 Glaucoma

This kind of disease damages the optic nerve within the eye. The condition becomes worse over time due to pressure build-up in the eye. Glaucoma is a condition that is most likely inherited. In patients suffering from this condition, intraocular pressure damages the optic nerve and causes permanent vision loss. It is a rapidly growing condition if not treated in its early stages and can take effect within a few years, Lylas et al. [81], and Vic Khemsara [83]. Figure 35 below shows a healthy eye and an eye with glaucoma.

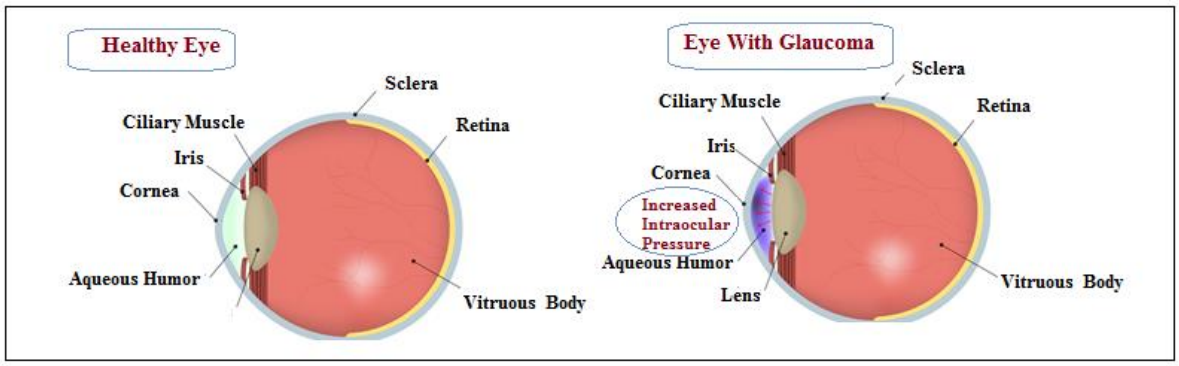

Figure 35: Healthy eye and eye with glaucoma [81].

4.6.1 Causes and Symptoms

Glaucoma occurs in a few different forms, all of which cause vision loss. Primary open angle glaucoma, the most common type, is processed in which fluids freely circulate within the eye and cause pressure build up over time. Another form, known as acute or angle closure glaucoma, develops suddenly and causes pressure to rise rapidly due to the normal flow of fluid being blocked off within the eye. Both forms of glaucoma cause blindness, but each has different symptoms, Andrew A. Dahl et al. [84], and Judy Torres et al. [85].

4.6.1.1 Open-Angle Glaucoma

Open-Angle Glaucoma is a form of glaucoma that occurs gradually and painlessly. During this condition, peripheral vision begins to fade away, so that objects are only visible

through central vision. If no treatment is provided for this condition, central vision begins to fade away as well, leading to complete loss of vision.

4.6.1.2 Closed Angle Glaucoma

Closed-angle glaucoma occurs suddenly. Symptoms of this condition include blurred vision, redness in the eye, severe headache, pain in the eye, extreme weakness, nausea and vomiting, and haziness in the cornea. Open-angle glaucoma and acute glaucoma are shown in Figure 36.

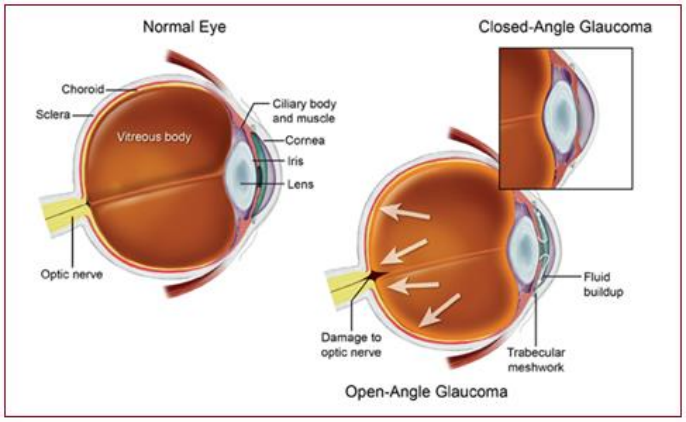

Figure 36: Open-angle glaucoma and acute glaucoma [85].

4.6.2 Risk Factors for Glaucoma

Glaucoma is a condition that can affect anyone. However, some people have a higher risk of developing the condition than others: the elderly, people who have had an eye injury, or those with family members afflicted by glaucoma. It is more likely to suffer from glaucoma at a later age, but open-angle glaucoma can also be developed suddenly after an eye injury or can be inherited genetically, Marilyn et al. [86].

4.6.3 Glaucoma Treatment

There are various methods to slow the progression of the condition, including medication, surgery, and laser treatment. Medicines help to decrease the buildup of pressure within the eye.

Surgery may become necessary if medication fails to slow the progression of pressure. Finally, laser treatments help to increase the flow of fluids. There is no exact limit of treatment use as some patients may need lifelong treatments, Marilyn et al. [86]. Figure 37 below shows that the patient could use eye drop permanently as suggested by the doctor.

Figure 37: Medication: glaucoma treatment [86].

4.7 Smartphone Ophthalmic Adapters for Eye Health Imaging Diagnostics

The use of smartphones in ophthalmology has dramatically increased; it is now used for a variety of purposes, from evaluating patients and sharing information to registration. New generation smartphones can run multiple advanced applications in medical specialties. Most recent smartphones feature the ability to capture high-quality photos through slit lamps, record videos from a microscope, and display images from optical coherence tomography systems and other high-tech devices. Nowadays, ophthalmologists can share images with their colleagues through several communication applications and make diagnostic decisions to help patients with deteriorating vision, Chhablani et al. [87] and Jill et al. [88]. Figure 38 shows one use of the smartphone in this field.

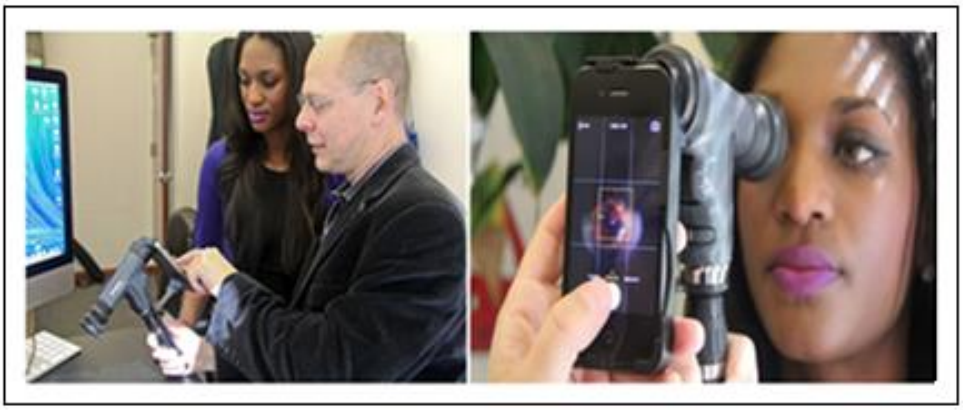

Figure 38: Shows the Retina Adapter and The Examination Test [88].

4.7.1 Smartphones Apps in Ophthalmology

In the field of ophthalmology, smartphone applications can be divided into three groups: those for patients, those for medical or nursing students, and for healthcare professionals. These applications can be used for patient education, assessment, and for physician education and reference, Moradian et al. [89]. Patient assessment: patient assessment uses several applications, such as near vision cards, color vision plates, penlights, Pupil gauges and rulers, and fluorescein lights to assess visual acuity for patients who cannot read, either because of young age or lack of literacy training. Patient education: Patient education includes visual or verbal materials designed to improve patients' understanding of medical conditions.

The most common applications for patient education include Eye handbook and Ikonion. These applications provide a database of information for common eye conditions, along with clinical features, treatment methods and references to websites for more information on the topic. Physician education: Physician education applications provide ophthalmologists with visual data on changes of the eye throughout different diseases or conditions. These applications also provide questionnaires for common diseases. Physician reference: Smartphone usage is mainly for personal communication, but physicians have been using them as a reference tool. Research

shows that 30-40% of physicians reported using their smartphones for clinical decision support. The use of smartphones during examination allows the physician to share and discuss photos or collect patient data.

4.7.2 Evaluating Patients' Eyes Using the Smartphone Camera and Smartphone Adapters

Healthcare professionals are increasingly using smartphone applications. These applications allow patients to monitor their health as well as to connect with healthcare providers. The uses of these devices for ophthalmic photography became ordinary eye care, where there is a growing market of applications and adapters. The new generation of smartphones features cameras with a resolution of 5-megapixels and higher, enabling users to capture clear images of the eye. There are several photo adapters available on the market for smartphones to capture both anterior and posterior eye segments. Nowadays, there are a lot of pocket-sized adapters such as portable eye examination kit (Peek), D-Eye, and EyeGo, which improve the lighting and magnification of smartphones, allowing the device to capture even higher quality images of the eye, Zvornicanin et al. [90], Lakshmi Nara et al. [91], and Roy et al. [92]. Figure 39 shows eye care professionals examining patients' eyes using the smartphone adapters.

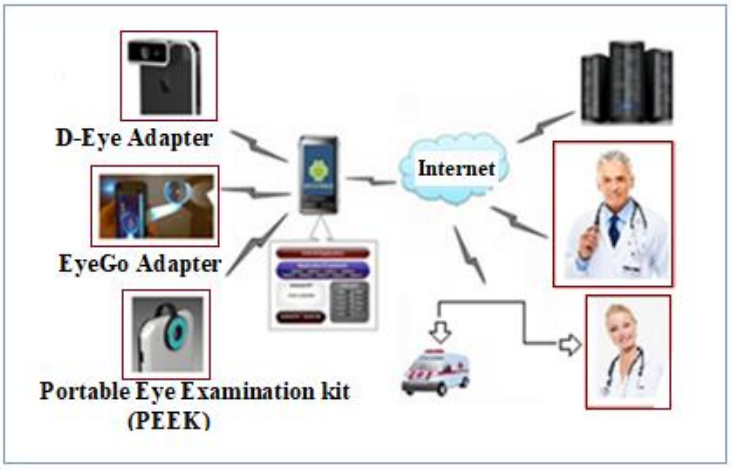

Figure 39: The eye care professionals, examine the patient's eyes by different smartphone adapters.

4.7.2.1 Portable Eye Examination Kit (PEEK)

The portable Eye Examination Kit (PEEK) consists of smartphone apps and retina adapters that clip onto to the device and sync with the PEEK app. This adapter can be used by anyone, anywhere in the world to examine eyes. It is essential as millions of people around the world are not able to get to clinics and hospitals to seek help. These two components allow eye care professionals to examine patients' eyes without ophthalmoscopes, bulky cameras, or other big, expensive equipment. PEEK comes equipped with a clip-on camera adapter that captures high-quality images of the back of the eye and the retina. It can help healthcare professionals treat people of all ages all around the world who have no access to eye care.

PEEK is used for necessary exams, as well as in treating glaucoma, cataracts, diabetic retinopathy, macular degeneration, nerve disease, and other eye conditions. Through the images that healthcare professionals receive from patients, they can determine two common conditions that often cause eye-related problems: diabetes and high blood sugar, Giardini et al. [93], Maryan Koberidze [94] and Giardini Mario [95]. Figure 40 shows where the retina adapter is placed onto the device and the examination test.

Figure 40: shows the retina adapter and the examination test [94].

4.7.2.2 EyeGo Adapter.

EyeGo is an adapter intended to allow ophthalmologists and healthcare specialists to capture high-quality images if the eye using ophthalmic lenses. This adapter consists of two adapters that are added to the smartphone camera and a three-dimensional posterior adapter to align the 20 diopter lens with the existing camera on the smartphone. The first adapter captures images of the surface of the eye, while the second adapter focuses light through the pupil on capturing images of the retina. During the process of examination, the right-hand steadies the smartphone and takes an image, and the left hand stabilizes the lens and lifts the upper eyelid, Robert et al. [96] and Ben Coxworth [97]. Fig. 41 shows the adapter location on the smartphone and the process of the examination.

Figure 41: The EyeGo adapter and the image received from the test [96]

4.7.2.3 D-Eye Adapter

D-Eye Adapter is an adapter consisting of a portable eye and retinal system that fits onto a smartphone creating a retinal camera for evaluation and screening of the eye. It uses LED lights as a light source and requires no extra power, making it an ideal solution for portable diagnostics. Its primary purpose is to examine a patient's retina, optic nerve, or other parts of the eye to determine disorders such as glaucoma, diabetic retinopathy, age-related macular degeneration, hemorrhages, and blood vessel abnormalities. Patients can capture images of their

eye which then can be sent to healthcare professionals for examination, David Nield [98] and Haddock et al. [99].

Figure 42: the D-Eye adapter test [98].

D-Eye adapter has many features and presents excellent advantages compared to EyeGo and Portable Eye Examination Kit (PEEK). D-Eye can record multiple images or videos in a field of view of up to 20 degrees without the use of an external power source, it prescreens anywhere and shares the collected data with other specialists immediately via the web; optional private and secure cloud-based storage system and iOS based applications are also available. Figure 42 above shows how eye care professionals examine a patient's eyes by using these smartphone adapters.

4.8. Performance Comparison between the Three Adapters, EyeGo, PEEK, and D-Eye

One way to analyze the performance of the smartphone adapters is to compare the three adapter types and their characteristics. This comparison is given in Table 2.

Attributes	EyeGo	PEEK	D-Eye
Components	*Two adapters and a three-dimensional posterior adapter to align the 20 diopter lens with the existing camera on the smartphone.*	*Retina adapters that clip onto the device and syncs to the app.*	*Portable eye and retinal system along with an LED light source that requires no power.*
Portability	*Medium*	*High*	*High*
Purpose	*Intended to capture images of the eye using ophthalmic lenses.*	*Captures images of patients' eyes without Ophthalmoscopes*	*Intended to capture images of the eye using ophthalmic lenses.*
Compatibility	*Smartphones.*	*Smartphones.*	*Smartphones.*
Examines	*Examines disorders within the eye.*	*Glaucoma, Cataracts, Diabetic retinopathy, Macular degeneration, and other eye conditions.*	*Examines retina, Optic nerve, and other disorders such as: Glaucoma, Cataracts, Diabetic retinopathy, Macular degeneration, and other eye conditions.*
User Friendliness	*Medium*	*High*	*High*
Quality of Lens	*High quality image*	*High quality image*	*High Megapixels image*

Table 2: Comparison between the three types of Smartphone adapters

4.9 Conclusions

Eyes are susceptible and prone to infections and diseases if proper care is not taken. Lowering stress, limiting exposure of eyes to direct light, decreasing usage of electronic devices, and getting proper sleep and nutritious food can all help reduce the pressures we put on our eyes. In this paper, I have discussed different types of eye diseases, their causes, symptoms, effect on vision and diagnosis. The four eye conditions considered here are diabetic retinopathy, cataracts, macular degeneration, and glaucoma. It is essential to treat these diseases at the initial stages, as

waiting for the condition to advance might result in permanent damage that cannot be treated. In this current technological era, smartphones with high-quality cameras can be used to diagnose eye diseases. The working principle behind this method is, the camera takes a high-quality magnifying picture of the eye and displays the images from optical coherence tomography systems. This allows the doctors to monitor and analyze the cause of the diseases closely and diagnose it accordingly based on the available data. It is essential for the smartphone to capture a high-quality magnifying image for accurate analysis of the diseases. Hence, pocket-sized adapters that improve the magnification and lighting are used to take a clear and sharper image of the eye. At present, there are three types of adaptors available: the Portable Eye Examination Kit (PEEK), EyeGo, and D-Eye adapter. The functioning and applications of these adapters are discussed in this paper. I have argued that the D-eye adapter shows the best performance when compared to D-Eye and PEEK. Currently, advanced research is being undertaken to develop smartphone apps and adapters that be more effective and extensively used in the medical field.

CHAPTER 5 HEALTHCARE MONITORING SYSTEM FOR EYE DISEASES USING A SMARTPHONE CAMERA AND D-EYE ADAPTER

5.1 Introduction

Smartphone cameras and applications can alter healthcare by providing doctors with the option to diagnose and treat patients beyond the confines of traditionally used equipment. Additionally, these technologies can help patients better manage their health and gain access to helpful information. The camera sensor on the smartphone can now provide the same benefits as a medical consultation. We can take advantage of this technology to treat vision loss, a common condition that can occur either suddenly or develop more gradually. Vision loss is different from blindness, as blindness occurs at birth while vision loss occurs over time. The causes of vision loss could range naturally from conditions affecting the eye to conditions affecting the part of the brain which works the eye. This problem is most familiar with age. Common causes of vision loss in elderly populations include diabetic retinopathy, cataracts, and other conditions. In this case, the camera sensor works to provide accurate images of the eye which later can be analyzed by a medical professional.

This chapter aims to demonstrate how a smartphone can be used for medical treatment in the field of ophthalmology, in which there is an excellent potential for the use of smartphone apps and their high-quality camera sensors. Smartphones can be used to capture high-quality images that allow the doctors to diagnose patients' conditions outside of traditional healthcare settings, help consumers control their health and access useful information whenever and wherever they need it.) A small optical device called D-Eye was developed. D-Eye attaches magnetically to a smartphone to examine and record photos and videos of the retina. D-Eye then uses wireless or other possible internet connectivity, allowing users to obtain retinal photos in even the most remote locations.

5.2 The Ultimate Aim of the Dissertation

The dissertation's goal is to develop a new app (Remote Healthcare-Monitoring Mobile App) to help patients who have low vision or are suffering from diseases that cause vision loss. This new app can exchange information directly between smartphone users—patients and doctors—allowing for increased security and privacy. The D-Eye adapter ophthalmic lens and this new app allow the patients, ophthalmologists, and other healthcare specialists to capture high-quality images of the eye and send these to the data collection server, where medical experts whom the prescribed treatment accesses them.

5.3 HealthCare-Monitoring Mobile App

5.3.1 APP Client

The software programming language used in the healthcare-monitoring app will be Objective-C language. Objective-C has become the backbone of iOS and is still the primary programming language used to create apps for the iPad and iPhone. It is one of the most common programming languages used in apps today, and it works similarly in programming software. In this work, I have supported iOS 10 iPhone system. The iPhone only runs the iOS system; the current system is the iOS 10. The app is fully functional on the following devices: iPhone 5/5s/6/6s, iPhone 6 Plus/6s Plus, iPhone 7 and iPhone8.

5.3.2 Collect Server

The collect server consists of the following: server system and database for the server system. I used LINUX as the server system, which can be installed for any purpose and is more secure than windows. In the web server, we used Nginx, which is known for its high performance, stability, rich feature set, simple configuration and low resource consumption. The web server with Nginx is straightforward and convenient. For the language, I used LUA, a

powerful and fast programming language that is easy to learn. For the database, I used REDIS, which is an open source database and memory.

5.3.3 Devices

5.3.3.1 Smartphone and D-Eye Adapter

As this dissertation has proven, smartphones can be used for medical treatment in the field of ophthalmology to diagnose loss of vision. Most recent smartphones have been equipped with a featured camera with high megapixels and advanced sensors that can be used to take fundus photographs through a slit lamp, record videos from an operating microscope, and display images from optical coherence tomography systems and other high-tech devices. Ophthalmologists can share and analyze these images with their colleagues using media sharing applications, thus ensuring the optimal diagnostic and therapeutic results for patients with low vision. D-Eye adapter is a portable eye and retinal system that attaches to a smartphone creating a retinal camera for health screening and evaluation and monitoring of the eye. Its primary purpose is to examine a patient's retina, optic nerve, and other parts of the eye to determine disorders such as glaucoma, macular degeneration, cataracts, and other eye diseases, Luis et al. [99]. Figure 43 shows the D-Eye adapter and the examination process.

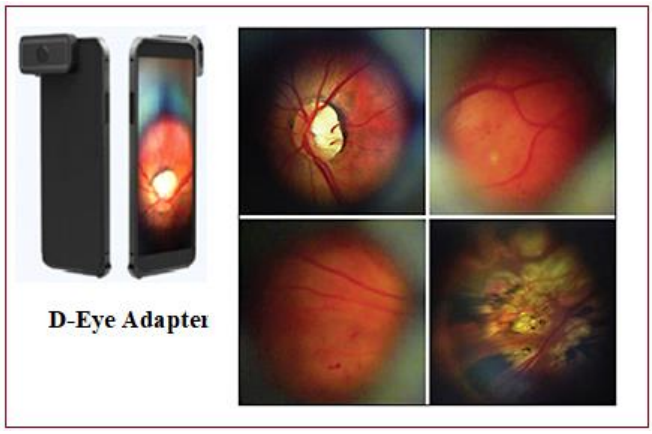

Figure 43: D-Eye Adapter with the Smartphone: Examination Test [99]

5.3.3.2 Healthcare Monitoring App – Flowchart

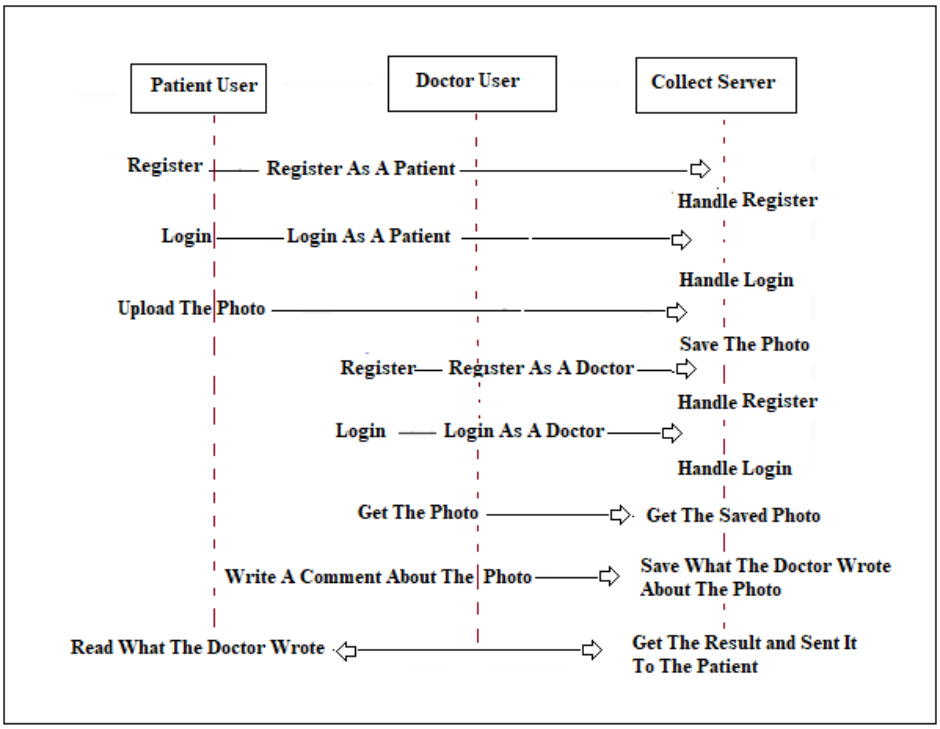

Figure 44: Healthcare Monitoring App – Flowchart

Patient and Collect Server:

The patient must first register for an account. The collect server will handle the request and create a patient-account in the system. The patient then logs in with the patient-account. The collect server will process the login request, check the password, load information from the database, etc. The patient will then be prompted to upload a photo. The collect server will save the picture in the database.

The registration process for the patient

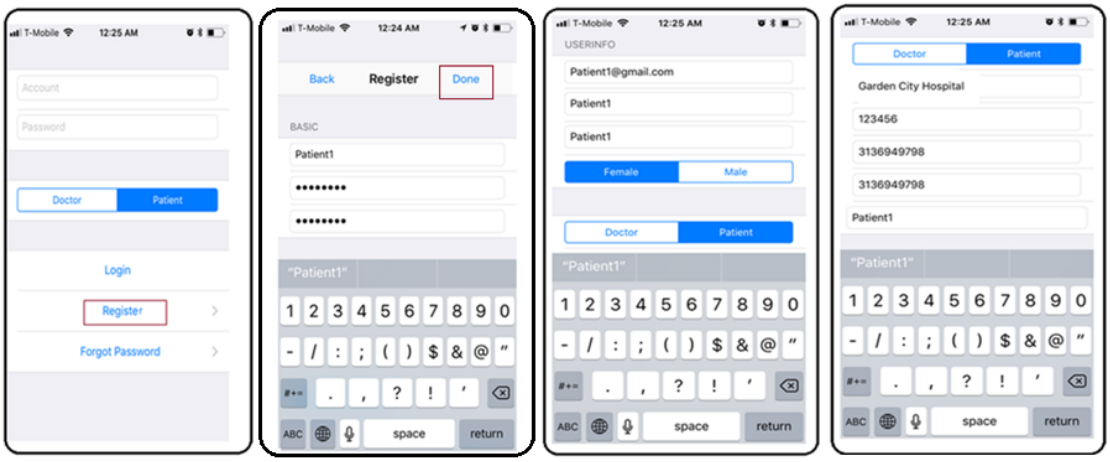

Figure 45: Healthcare Monitoring App: "Registration Process For the Patient."

Doctor and Collect Server:

On the other end, doctors must similarly create and log into a secure account. Once they have done so, they will receive photo patients from the collection server. The doctor will write something about the photo and update the photo. The patient reads what the doctor writes and gets the result. This is the registration process for the patient and the doctor. This is the set-up process for the App. First, you must register for an account as either a Patient or a Doctor. Steps: click the "Register button," fill out the required information, and click "Done" to finalize.

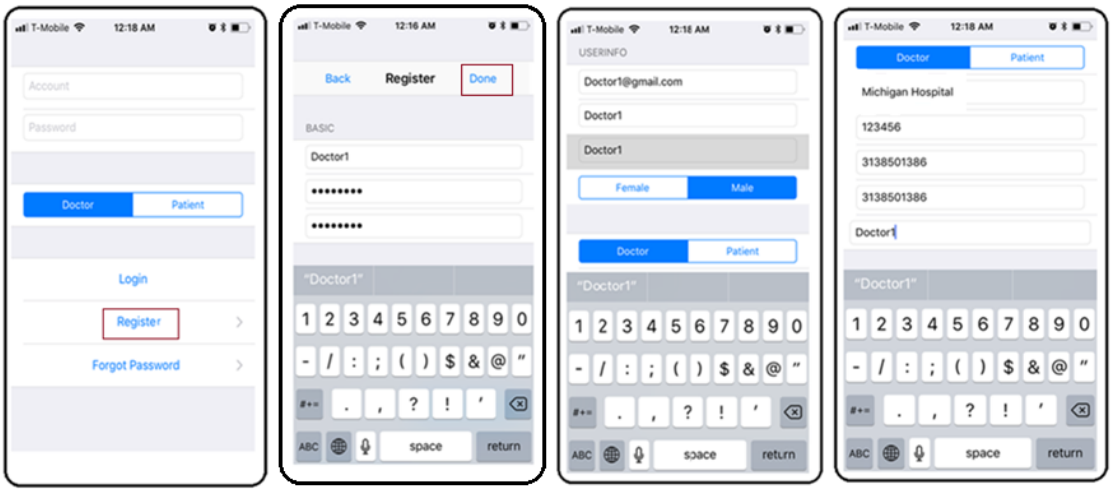

Figure 46: Healthcare Monitoring Mobile App: "Registration Process for the Doctor"

5.3.3.3 Healthcare Monitoring App Algorithm (1) "User Login"

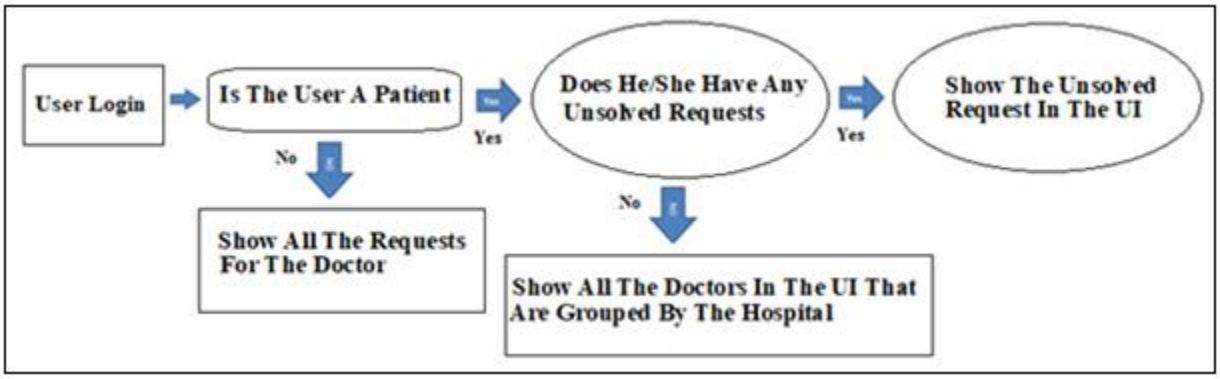

Figure 47: Healthcare Monitoring App - Algorithm (1) "User login."

The Login Process: " Doctor and Patient Login."

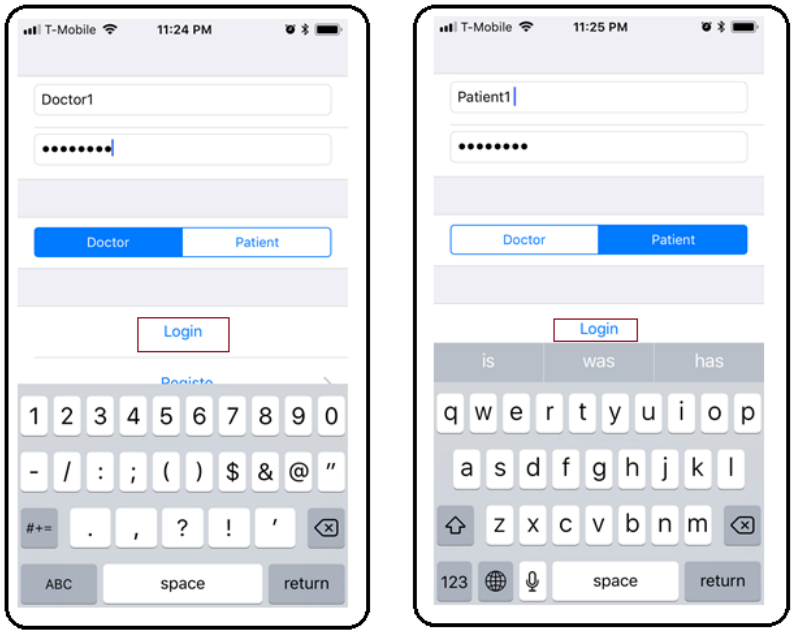

Figure 48: "Login Process For Doctor and Patient. "

User Login:

In this part, the user can log in into the system with his/her account and choose the type of account, for instance, "the patient or the doctor." If the user is the patient, then more information is needed; if the user is the doctor, then all the requests to the doctor will be shown in our UI list. If the patient has any unsolved request, this will also be shown in our UI list. If

there isn't any unsolved request, the patient would need to make a new request, which will be shown to all the doctors on the UI list who are grouped by the hospital.

5.3.3.4 Healthcare Monitoring App Algorithm (2) "Make a New Request"

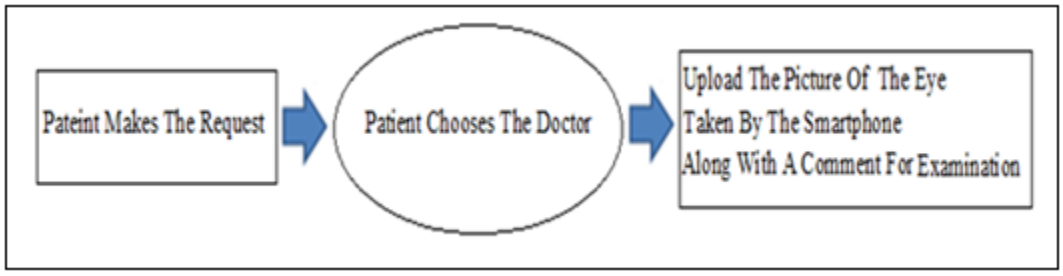

Figure 49: Healthcare Monitoring App - Algorithm (2) "Make a New Request."

After you have a login with a new account as either a Patient or a Doctor, you will be prompted to make a new request. To make a new request, first, choose a doctor by clicking on the "Doctor List," and then choose the "Docter Name" from the list. After that, you will need to upload an image of the eye taken by the smartphone camera with the D-Eye adapter.

Make Request Process: "Choose A Doctor, Upload A Picture, Then Make Request."

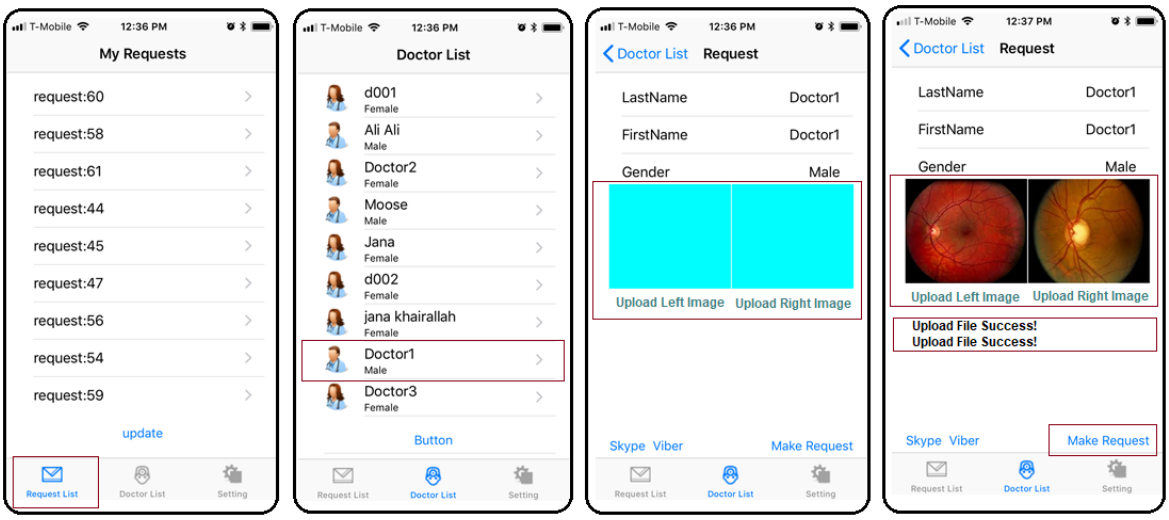

Figure 50: (a) Make Request Process: "Choose a Doctor, Upload a Picture, then Make Request."

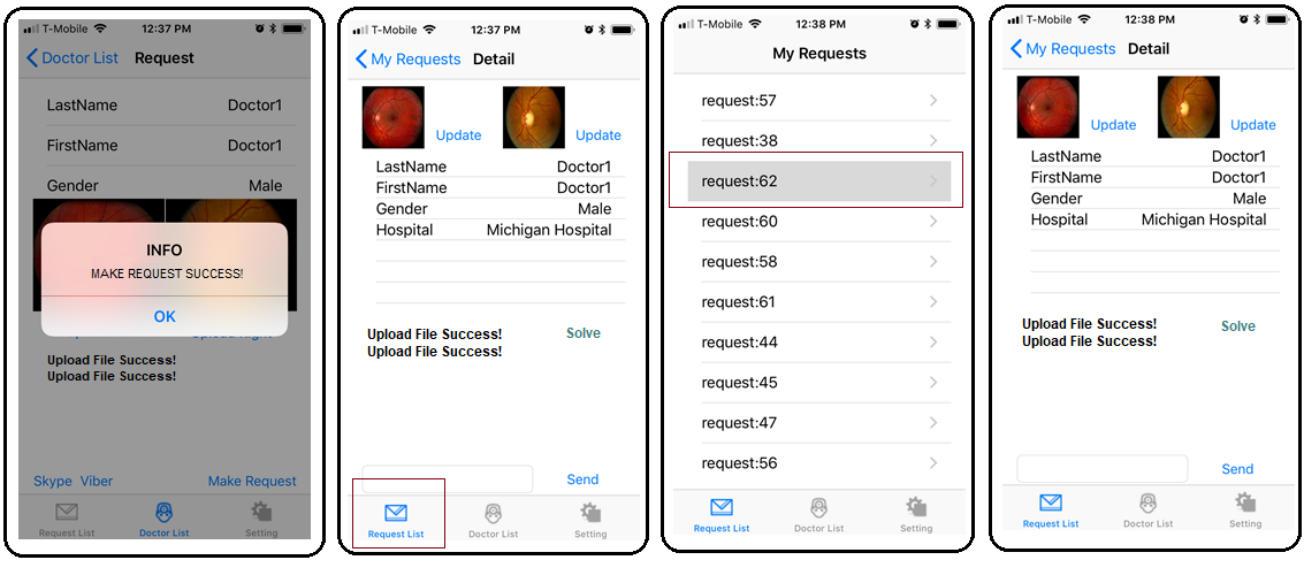

Figure 51: (b)Making a Request Process "Choose a Doctor, Upload a Picture, then Make Request."

Update Request: Choose A Request, Update picture, Then Update A Comment."

If the doctor has any comment regarding the picture not being clear, you will need to click the " Update " button , then you need to choose or take a more clear image; then, you will see that the picture has changed and a new message below it " Upload file success " that indicated that the image uploaded successfully.

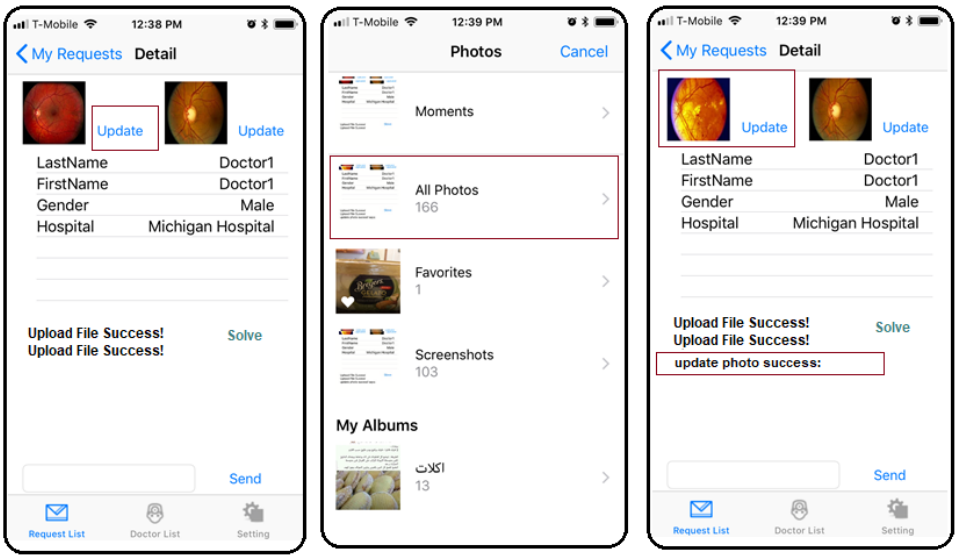

Figure 52: Update Request: " Update Picture" then Update a Comment."

5.3.3.5 Healthcare Monitoring App Algorithm (3) "Solve the Request."

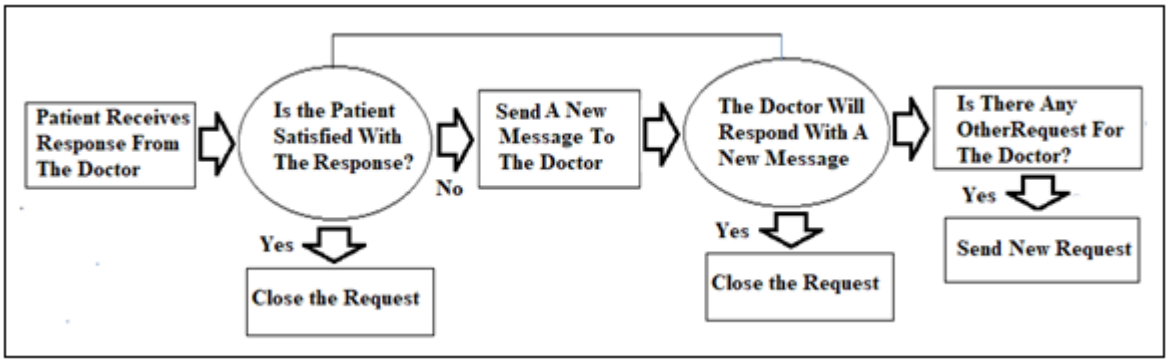

Figure 53: Healthcare Monitoring App Algorithm (3) "Solve the Request."

In this part, if there is a request sent in by a patient to the doctor, the doctor can view and analyze this request and respond to the patient with results and treatments. If the patient agrees with the results, then the request will automatically be closed. If the patient does not agree with the results, then he can send a new message to the doctor and keep communicating with him/her until the request is solved, then the request will be closed.

Solved Request: "Choose Solve Request, then Request Close."

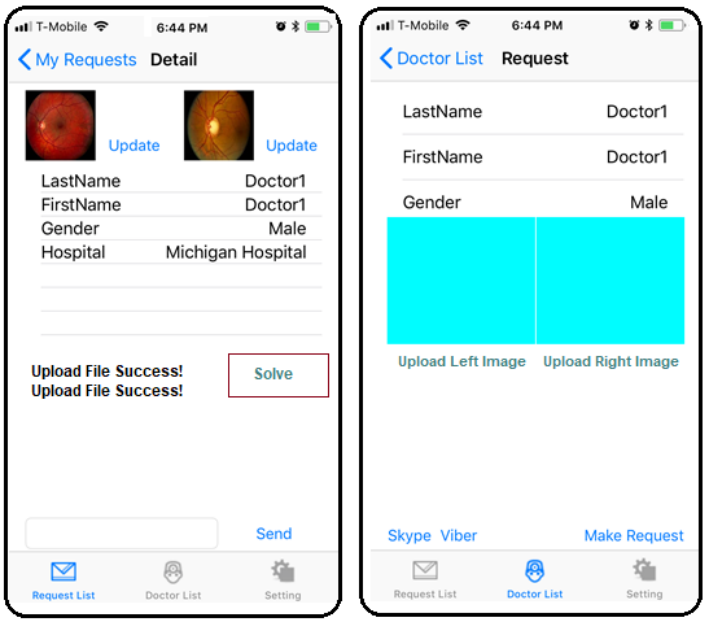

Figure 54: Solve Request Process "Choose to solve request will be resolved."

5.3.3.6 Healthcare Monitoring Mobile App: "Skype and Viber Communication."

In a case where you would need to speak with the doctor through a phone call. First, click on the "Doctor List," then choose the "Doctor Name" from the list. After that, you will see the two option of communication which is "Skype and Viber.

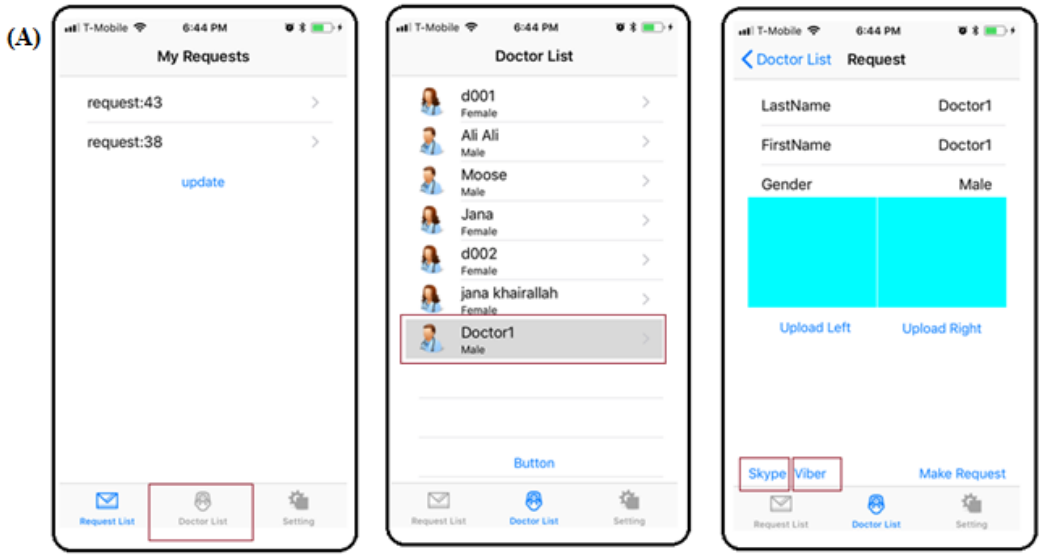

Figure 55: (a) Healthcare Monitoring Mobile App: "Skype and Viber Communication."

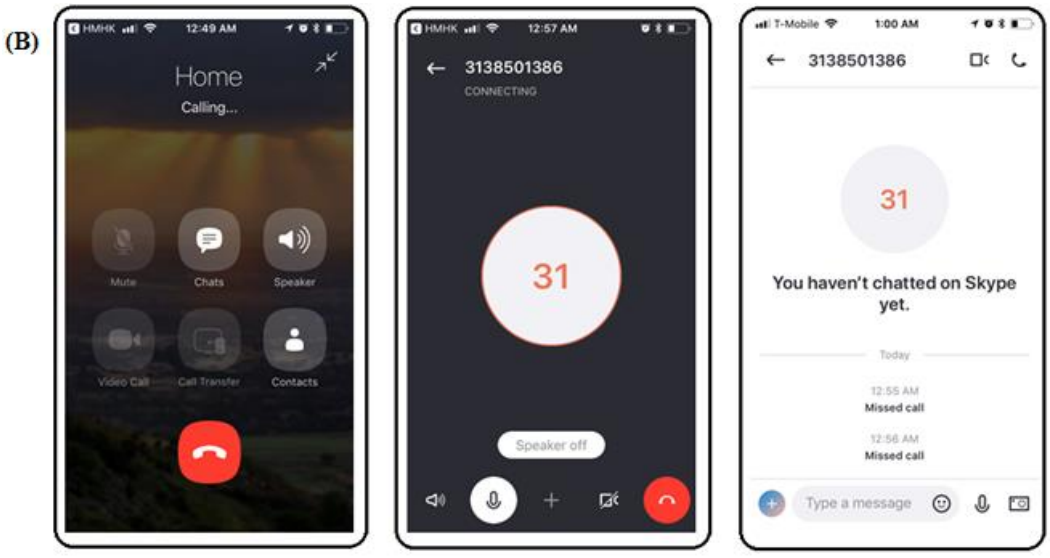

Figure 56: (b) Healthcare Monitoring Mobile App: "Viber and Skype Communication."

5.3.3.7 Conclusion

The use of Smartphone cameras has become vital for monitoring eye diseases as they can capture clear, high-quality images of the eye for examination by a physician. All this can be done through a smartphone adapter and an application which can be downloaded on IOS. The purpose of this app is to monitor a patient's eye for vision loss. In this chapter, I went over the details of how the smartphone application functions and all its features. I showed just how quickly a patient could make a request to a doctor concerning any eye disease. The app contains many features that better enhance the communication process between the patient and the doctor. For example, if a patient would like a thorough explanation of the results he received from the doctor concerning his condition, he can choose to call him via Skype or Viber. Last, but certainly not least, I discussed the D-Eye adapter which clips to the smartphone to capture clear, close-up images of the patient's eye. Its primary purpose is to examine a patient's retina, optic nerve, and any part of the eye to determine disorders such as glaucoma, macular degeneration, and any other eye diseases.

CHAPTER 6 CONTRIBUTIONS AND FUTURE WORK

This chapter will provide the main contributions of the dissertation, followed by a description of future avenues of research.

6.1 Contributions

This dissertation aims to demonstrate how a smartphone can be used for medical treatment in the field of ophthalmology, where smartphone apps and their high-quality camera sensors have many potential applications. They can be used to capture high-quality images that allow specialists to diagnose patients' conditions outside of traditional healthcare settings, as well as to help consumers control their health and wellness or access data whenever they need it. A small optical device called D-Eye was developed. D-Eye attaches magnetically to a smartphone to record and examine photos and videos of the retina. D-Eye also uses the wireless connection and any possible internet connectivity, which makes it possible for users to obtain retinal photos in even the most remote locations.

In this dissertation, I developed the following: first, I provided a meaningful utilization comparison between Wireless Sensor Network, PDA and smartphones in Remote Healthcare Monitoring System (RHMS) architecture design. I have evaluated different approaches of the healthcare monitoring system architecture and investigated the use of advanced technologies that enable the monitoring of patients' vital signs by a diagnostic medical team in real-time. In this comparison, the smartphone gives the best performance as it can be used anywhere. It allows for early hospital admission and continuous measurements of patient status. The smartphone is turning out to be of significant advantage compared to Wireless Sensor Network and Personal Digital Assistant. Second, I provided a meaningful utilization comparison between the smartphone adapters D-Eye, EyeGo and Portable Eye Examination Kit (PEEK). Third, I developed a new app (Remote Healthcare-Monitoring Mobile App) to help patients who have

low vision and who are suffering from the diseases which may cause a vision loss. This app is capable of processing, evaluating, interacting with and storing health data which is constantly measured by Personal Health Monitors (PHM). The app can facilitate the exchange of information between patients and doctors with a high degree of security and privacy. The idea for the app relies on the following components: a smartphone application, a data collection center, and professionals in Ophthalmology. The patient should be registered in the system—for example, Retina Michigan Center or Glaucoma Michigan Center. After registration, the patient is instructed on how to take photos of his/her eyes correctly and then uses the smartphone application to send these pictures to the data collection center. Specialists get access to this data and help in the treatment. Finally, I completed the development of the mobile app, including the Skype and Viber links, which can better enhance the ability to exchange information between the patient and the doctor.

6.2 Future Directions

As demonstrated in this dissertation, smartphones provide the highest performance rates for remote screening and self-monitoring healthcare apps. They allow for early hospital admission and continuous measurements of patients' conditions. In the future, individuals using healthcare monitoring apps on their phones will not only enjoy improved health and well-being but, once technology advances even further, therapeutic costs for updating interminable remedial conditions will be diminished. At a lower cost and with good results, remote health-monitoring apps will prevent future diseases and improve the condition of all patients.

Eyes are one of the most critical and vital organs of the human body. Eyes are susceptible and prone to infections and diseases if proper care is not taken. But even outside of ophthalmology, smartphone camera sensors can provide beneficial information about a multitude of medical

conditions. This technology can be used in the dermatology field, for example, if the patient is instructed to use the smartphone camera to take photos of his/her skin correctly, then send them to the doctor. Similarly to the system, I am proposing; the doctor receives these photos then prescribes a treatment according to analysis.

In the future, [1] I will develop a smartphone app which not only takes photos of the patient's eye but also captures an explicit video of the eye. Such an app could be used more effectively and extensively in the field of ophthalmology, but could also have broader application in other medical fields. [2] This app is currently only compatible with IOS mainly for testing purposes; it will be compatible with Android shortly. I will develop a secure system that includes encrypted files, automatic log off, a unique user ID, strong passwords, firewalls and secure Wi-Fi connections, all of which ensure the privacy and safety of patient information. [3] Additionally, payment methods will also be worked out. Patients receive the bill from the doctor through email and mailbox; I will develop an application which will allow payments to be settled through credit card, PayPal, etc.

APPENDIX A

HEALTHCARE-MONITORING MOBILE APP

APP Client

The software programming language used in the healthcare-monitoring app will be Objective-C language. Objective-C has become the backbone of iOS and is still the primary programming language used to create apps for the iPad and iPhone. It is one of the most common programming languages used in apps today, and it works similarly in programming software. In this work, I have supported iOS 10 iPhone system. The iPhone only runs the iOS system; the current system is the iOS 10. The app is fully functional on the following devices: iPhone 5/5s/6/6s, iPhone 6 Plus/6s Plus, iPhone 7 and iPhone8.

```
//
//  WebAPI.h
//  photo-demo
//

// import function from header file
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "ViewController.h"

// class of WebAPI
@interface WebAPI : NSObject

// this NSString stores the URL of the http request
@property (strong,atomic) NSString *urlStr;

// this NSString stores the API interface name
@property (strong,atomic) NSString *apiStr;

// this NSData handles the data to upload
@property (strong,atomic) NSData   *imageData;

/*
   @function :        initial the url and request api
   @param urlStr:     url of http head
   @param apiStr:     name of the http request
   @return :          void
 */
```

```
-(void)initWithUrlAndApi:(NSString*)urlStr api:(NSString*)apiStr;

/*
   @function:         generate a simple post request
   @param postData:    store body of the http request
   @param cb:          this NSString gives the notification name that handle the packet replyed by server
   @return:           void
 */
-(void)webSimplePost:(NSMutableDictionary*) postData cb:(NSString*)cb;

/*
   @function:         generate a post request to upload binary data.
   @param postData:    the binary data to upload
   @param cb:          this NSString gives the notification name that handle the packet replyed by server
   @return:           void
 */
-(void)webUploadData:(NSData*) postData cb:(NSString*)cb;

@end

//
//  WebAPI.m
//  photo-demo
//
//

#import "WebAPI.h"

// define marco for easy coding
#define StringWithData(data) [[NSString alloc] initWithData:(data) encoding:NSUTF8StringEncoding]
#define DataWithString(str) [(str) dataUsingEncoding:NSUTF8StringEncoding]

@implementation WebAPI

// discarded function
- (Boolean)uploadFileFromImageView:(id)imageView {
   return YES ;
}


-(void)initWithUrlAndApi:(NSString*)urlStr api:(NSString*)apiStr {
   // assignment urlStr to private var
   self.urlStr = urlStr;

   // assigment apiStr to private var
   self.apiStr = apiStr;
}

-(NSString*)getHttpURL {
   //return the http request url with format
```

```
    return [NSString stringWithFormat:@"http://%@/api/%@",self.urlStr,self.apiStr] ;
}


-(void)webSimplePost:(NSMutableDictionary*) postData cb:(NSString*)cb;
{
    NSLog(@"enter WebAPI:webSimplePost");

    // generate url object
    NSURL *url = [NSURL URLWithString:[self getHttpURL]];

    // check url object valid
    if (!url) {
        NSLog(@"WebAPI:webUploadImage get url failed:%@",url);
        return ;
    }

    // generate request object with url
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];

    // check request valid
    if (!request) {
        NSLog(@"WebAPI:webUploadImage get request failed:%@",request);
        return ;
    }

    NSError *error;

    // if postData is nill,just make a simple http post request without data
    if (!postData) {
        request.HTTPMethod = @"POST";
        //http-body
        request.HTTPBody = [@"" dataUsingEncoding:NSUTF8StringEncoding] ;

    }else
    {
        // seralize postData with json
        // generate a binary object
        NSData *jsonData = [NSJSONSerialization dataWithJSONObject:postData
                                                           options:NSJSONWritingPrettyPrinted // Pass 0 if you don't care
about the readability of the generated string
                                                             error:&error];
        // check the reuslt of seralize
        if (! jsonData) {
            NSLog(@"Got an error: %@", error);
            return ;
        }

        // generate NSString
        // convert binary data to string
```

```
        NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];

        // use xor to encode the net stream
        // make the http stream security
        NSString *jsonStringSend = StringWithData([self encrypt: DataWithString(jsonString)]);

        NSLog(@"json send:%@",jsonStringSend);
        NSLog(@"json send:%@",StringWithData([self encrypt: DataWithString(jsonStringSend)]));
        //http-method
        request.HTTPMethod = @"POST";
        //http-body
        request.HTTPBody = [jsonStringSend dataUsingEncoding:NSUTF8StringEncoding] ;
    }

    // generate session
    NSURLSession *session = [NSURLSession sharedSession];

    // use notification to handle the response from server
    NSURLSessionDataTask *sessionDataTask = [session dataTaskWithRequest:request
                                    completionHandler:^(NSData * _Nullable data, NSURLResponse *
_Nullable response, NSError * _Nullable error) {
                                        NSDictionary *dataDict = [NSDictionary
dictionaryWithObjectsAndKeys:data,@"1",response,@"2",error,@"3", nil];
                                        [[NSNotificationCenter defaultCenter]
                                         postNotificationName: cb
                                         object:nil
                                         userInfo:dataDict];
                                    }];

    // check the result of dispatch
    if (!sessionDataTask) {
        NSLog(@"WebAPI:webUploadImage get sessionDataTask failed:%@",sessionDataTask);
        return ;
    }

    // none bussiness with logic
    // no need to understand
    [sessionDataTask resume];

}


-(void)webUploadData:(NSData*) postData cb:(NSString*)cb
{
    NSLog(@"enter WebAPI:webUploadImage");

    // generate url object
    NSURL *url = [NSURL URLWithString:[self getHttpURL]];
```

```
    // check result
    if (!url) {
        NSLog(@"WebAPI:webUploadImage get url failed:%@",url);
        return ;
    }

    // generate request with url
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];

    // check result
    if (!request) {
        NSLog(@"WebAPI:webUploadImage get request failed:%@",request);
        return ;
    }

    //http-method
    request.HTTPMethod = @"POST";
    //http-body
    request.HTTPBody = postData;

    self.imageData = nil ;

    // use notification to handle the response from server
    NSURLSession *session = [NSURLSession sharedSession];
    NSURLSessionDataTask *sessionDataTask = [session dataTaskWithRequest:request
                                    completionHandler:^(NSData * _Nullable data, NSURLResponse *
_Nullable response, NSError * _Nullable error) {
                                        NSDictionary *dataDict = [NSDictionary
dictionaryWithObjectsAndKeys:data,@"1",response,@"2",error,@"3", nil];
                                        [[NSNotificationCenter defaultCenter]
                                         postNotificationName: cb
                                         object:nil
                                         userInfo:dataDict];
                                    }];

    // check the result of dispatch
    if (!sessionDataTask) {
        NSLog(@"WebAPI:webUploadImage get sessionDataTask failed:%@",sessionDataTask);
        return ;
    }

    // none bussiness with logic
    // no need to understand
    [sessionDataTask resume];

}


// function used to encrypt http stream with XOR
- (NSData *)encrypt:(NSData *)data {
```

```
    char *dataP = (char *)[data bytes];
    NSString* tmpstr = @"1#9#8#9#1#0#2#6";
    NSArray* strArr = [tmpstr componentsSeparatedByString: @"#"];
    for (int i = 0; i < data.length; i++) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsequenced"
        *dataP = *(++dataP) ^ [[strArr objectAtIndex:i% [strArr count]] integerValue];
#pragma clang diagnostic pop
    }
    return data;
}

@end

//
//  RegisterViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/23.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "support.h"

@interface RegisterViewController : UITableViewController

@end

//
//  RegisterViewController.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/23.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "RegisterViewController.h"
#import "RegisterCheck.h"

@interface RegisterViewController ()

// ---- user interface element to type in data during register --- start --

//Account
@property (weak, nonatomic) IBOutlet UITextField *strAccount;

//password
@property (weak, nonatomic) IBOutlet UITextField *strPassWord;
```

```
//confirm password
@property (weak, nonatomic) IBOutlet UITextField *strConfirmPassword;

//e-mail
@property (weak, nonatomic) IBOutlet UITextField *strEMail;

//first name
@property (weak, nonatomic) IBOutlet UITextField *strFirstName;

//last name
@property (weak, nonatomic) IBOutlet UITextField *strLastName;

//gender
@property (weak, nonatomic) IBOutlet UISegmentedControl *strGender;

//type
@property (weak, nonatomic) IBOutlet UISegmentedControl *strType;

//hospital information
@property (weak, nonatomic) IBOutlet UITextField *strHospital;

//med number
@property (weak, nonatomic) IBOutlet UITextField *strMedNumber;

//no use
@property (nonatomic,strong) NSDictionary* dictData;

//skype account informantion
@property (weak, nonatomic) IBOutlet UITextField *strSkype;

//viber account informantion
@property (weak, nonatomic) IBOutlet UITextField *strViber;

// specialty of doctor
@property (weak, nonatomic) IBOutlet UITextField *strSpecialty;

@end
// ---- user interface element to type in data during register --- end --

@implementation RegisterViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   // Do any additional setup after loading the view.

   // gneerate bundle object to read bundle file
   NSBundle *bundle = [NSBundle mainBundle];

   // get data in ServerInfo.plist
```

```
    NSString *plistPath = [bundle pathForResource:@"ServerInfo"
                                           ofType:@"plist"];

    // init the private var "dictData" with bundle information
    self.dictData = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

    // registe the notifycation to handle http response
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onRegisterHttpResonpse:)
                                                 name:@"RegisterHttpResonpse"
                                               object:nil];

    NSLog(@"%@",[Singleton sharedManager].singletonData);
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

// discarded function
- (IBAction)doBack:(id)sender {
   /*[self performSegueWithIdentifier:@"ShowReg" sender:self];*/

    [self dismissViewControllerAnimated:YES completion:^{
        NSLog(@"Modal View done");
    }];
}

- (IBAction)doRegister:(id)sender {
    NSLog(@"do Register enter");

    //do check
    NSString* error;

    // check the password
    BOOL ret = [RegisterCheck CheckPassword:self.strPassWord.text
confirm:self.strConfirmPassword.text error:&error];
    if (!ret) {
        UIAlertView *alertView = [[UIAlertView alloc]
                                  initWithTitle:@"Alert"
                                  message:error
                                  delegate:self
                                  cancelButtonTitle:@"No"
                                  otherButtonTitles:@"Yes", nil];
        [alertView show];
        return ;
    }

    // check the e-mail
```

```
    ret = [RegisterCheck CheckEmail:self.strEMail.text];
    if (!ret) {
        UIAlertView *alertView = [[UIAlertView alloc]
                                  initWithTitle:@"Alert"
                                  message:@"e-mail address invalid"
                                  delegate:self
                                  cancelButtonTitle:@"No"
                                  otherButtonTitles:@"Yes", nil];
        [alertView show];
        return ;
    }



    // generate the array used for storing register information
    NSArray *arrTextFields = [[NSArray alloc] initWithObjects:StringWithData([self encrypt:
DataWithString(@"su{Ijbowhu")]),
                              StringWithData([self encrypt: DataWithString(@"su{XhrsUism")]),
                              StringWithData([self encrypt: DataWithString(@"su{KfofktlYizrwmte")]),
                              StringWithData([self encrypt: DataWithString(@"su{MD`in")]),
                              StringWithData([self encrypt: DataWithString(@"su{N`ssvH`dm")]),
                              StringWithData([self encrypt: DataWithString(@"su{DhrtLgll")]),
                              StringWithData([self encrypt: DataWithString(@"su{@frpkr`e")]),
                              StringWithData([self encrypt: DataWithString(@"su{EleNwkclz")]),
                              StringWithData([self encrypt: DataWithString(@"su{[bxpg")]),
                              StringWithData([self encrypt: DataWithString(@"su{^`cep")]),
                              StringWithData([self encrypt: DataWithString(@"su{[ydckgm}q")]),
                              nil];

    // generate the array used for storing register information
    NSArray *arrUISegmentedControlFields = [[NSArray alloc] initWithObjects: StringWithData([self
encrypt: DataWithString(@"su{Olodgt")]),
                              StringWithData([self encrypt: DataWithString(@"su{\\pqe")]),
                              nil];


    // generate a mutable dictionary
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc] init];

    //get param from all the text field
    NSEnumerator *en = [arrTextFields objectEnumerator];
    id obj;
    while (obj = [en nextObject]) {
        NSLog(@"UITextField:%@",obj);
        SuppressPerformSelectorLeakWarning([muDict setObject:[[self
performSelector:NSSelectorFromString(obj)] text]
                                            forKey:obj]);
    }
```

```
    // generate an enmuerator to loop the dictionary
    en = [arrUISegmentedControlFields objectEnumerator];

    while (obj = [en nextObject]) {
        id UISegmentedControl ;
        SuppressPerformSelectorLeakWarning(UISegmentedControl = [self
performSelector:NSSelectorFromString(obj)]);
        // generate Title with the string in the dictionary
        NSString *strTitle = [UISegmentedControl titleForSegmentAtIndex:[UISegmentedControl
selectedSegmentIndex]];

        // save user input data to dictionary
        [muDict setObject:strTitle
                forKey:obj];
    }

    // generate WebAPI obj
    WebAPI * webapi = [[WebAPI alloc] init];

    // generate strings used for making request -- start --
    NSString *hostKey = StringWithData([self encrypt: DataWithString(@"hnz|")]);
    NSString *host = StringWithData([self encrypt: DataWithString([self.dictData
objectForKey:hostKey])]) ;

    NSString *apiKey = StringWithData([self encrypt: DataWithString(@"WDKWHQI]TDNAZUEP")]);
    NSString *api  = StringWithData([self encrypt: DataWithString([self.dictData
objectForKey:apiKey])]) ;
    // generate strings used for making request -- end --

    // init webapi object with IP and request name
    [webapi initWithUrlAndApi:[Singleton sharedManager].server_ip api:api];

    // Do make the request
    [webapi webSimplePost:muDict cb:@"RegisterHttpResonpse"] ;
}


// function to handle the resonpse from server
-(void)onRegisterHttpResonpse:(NSNotification*)notification {

    // generate dictionary to save the response data
    NSDictionary * response = [notification userInfo];

    // get binary data from response
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];

    // convert binary data to NSString mode
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
```

```
    retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);

    // check the result of Register
    if ([retString isEqualToString:StringWithData([self encrypt:
DataWithString(@"RDNWZTCGUR")])]){

        // if registe success
        // update the user interface
        NSLog(@"Register Success!") ;
        dispatch_async(dispatch_get_main_queue(), ^{
            [self dismissViewControllerAnimated:YES completion:^{
                NSLog(@"Modal View done") ;
                NSDictionary *dataDict = [NSDictionary dictionaryWithObject:self.strAccount.text
                                                                     forKey:@"username"];
                [[NSNotificationCenter defaultCenter]
                 postNotificationName:@"RegisterCompletionNotification"
                 object:nil
                 userInfo:dataDict];
            }];
        });
    } else if ([retString isEqualToString:@"REG_FAILED_ALREADY_REGISTED"]) {

        // if register failed
        // show alert
        UIAlertView *alertView = [[UIAlertView alloc]
                                  initWithTitle:@"Alert"
                                  message:@"account already exist"
                                  delegate:self
                                  cancelButtonTitle:@"No"
                                  otherButtonTitles:@"Yes", nil];
        [alertView show];
        return ;
    }
}


// function to encrypt http stream
- (NSData *)encrypt:(NSData *)data {
    char *dataP = (char *)[data bytes];
    NSString* tmpstr = @"1#9#8#9#1#0#2#6";
    NSArray* strArr = [tmpstr componentsSeparatedByString: @"#"];
    for (int i = 0; i < data.length; i++) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsequenced"
        *dataP = *(++dataP) ^ [[strArr objectAtIndex:i% [strArr count]] integerValue];
#pragma clang diagnostic pop
    }
    return data;
```

```
}

/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end

//
//  ViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/23.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UITableViewController


@end

//
//  ViewController.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/23.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "ViewController.h"
#import "support.h"

#include <netdb.h>
#include <arpa/inet.h>



@interface ViewController ()

// text field to type in account
```

```
@property (weak, nonatomic) IBOutlet UITextField *textAccount;

// text field to type in password
@property (weak, nonatomic) IBOutlet UITextField *textPassword;

// Segment Control to select account type
@property (weak, nonatomic) IBOutlet UISegmentedControl *segmentType;


@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.


    // register notification to handle response of login request
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onLoginHttpResponse:)
                                                 name:@"LoginResponseNotification"
                                               object:nil];

    // register notification to handle resonse of get_user_data request
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onUserDataHttpResponse:)
                                                 name:@"UserDataNotification"
                                               object:nil];

    // get cookies from disk
    NSData *cookieData = [[NSUserDefaults standardUserDefaults]
objectForKey:@"ApplicationCookie"];

    // if cookies exits
    // set cookies into http stream
    if ([cookieData length] > 0) {
        NSArray *cookies = [NSKeyedUnarchiver unarchiveObjectWithData:cookieData];
        for (NSHTTPCookie *cookie in cookies) {
            [[NSHTTPCookieStorage sharedHTTPCookieStorage] setCookie:cookie];
        }
    }

    // generate singleton object
    Singleton* sl = [Singleton sharedManager];

    // test data
    sl.singletonData = @"test";
```

```
    // domain of the server
    sl.server_ip = [self getIPWithHostName:@"s2.legitobj.club"];

}


// get ip with domain
- (NSString *) getIPWithHostName:(const NSString *)hostName
{
    // generate c string
    const char *hostN= [hostName UTF8String];

    // generate struct to get result
    struct hostent* phot;

    // call gethostbyname to get ip address -- start --
    @try {
        phot = gethostbyname(hostN);

    }
    @catch (NSException *exception) {
        return nil;
    }
    // call gethostbyname to get ip address -- start --


    // generate struct to get ip
    struct in_addr ip_addr;

    // copy ip from stuct to ip_addr;
    memcpy(&ip_addr, phot->h_addr_list[0], 4);
    char ip[20] = {0};
    inet_ntop(AF_INET, &ip_addr, ip, sizeof(ip));

    // convert c-string to NSString
    NSString* strIPAddress = [NSString stringWithUTF8String:ip];

    //  return ip address
    return strIPAddress;
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)doLogin:(id)sender {
    NSLog(@"do login enter") ;
```

```
    // generate mutable dictionary
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];

    // get account from user interface
    NSString *strAccount = self.textAccount.text ;

    // get password from user interface
    NSString *strPassword = self.textPassword.text ;

    // get account type from user interface
    NSInteger index = [self.segmentType selectedSegmentIndex];

    // conver from index to string
    NSString *strType = [self.segmentType titleForSegmentAtIndex:index];

    // save data into dictionary
    [muDict setObject:strAccount forKey:@"Account"];
    [muDict setObject:strPassword forKey:@"Password"];
    [muDict setObject:strType forKey:@"Type"];

    // generate webapi object
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "login" as request name
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"login"];

    // do the http request
    [wb webSimplePost:muDict cb:@"LoginResponseNotification"];
}


- (IBAction)doRegister:(id)sender {
    //[self performSegueWithIdentifier:@"ShowReg" sender:self];
    // get Main storyboard
    UIStoryboard* mainStoryboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];

    // get ViewController of registerView
    UIViewController *registerViewController = [mainStoryboard
instantiateViewControllerWithIdentifier:@"registerViewController"];

    // set the style of viewcontroller
    registerViewController.modalTransitionStyle =
    UIModalTransitionStyleCoverVertical;

    // set the animated
    [self presentViewController:registerViewController animated:YES completion:^{
        NSLog(@"Present Modal View");
    }];
}
```

```
-(void)onLoginHttpResponse:(NSNotification *)notification {
    // generate dictionary to save the data of response
    NSDictionary * response = [notification userInfo];

    // get response body in binary
    NSData *retData = [response objectForKey:@"1"];

    // get response head
    NSURLResponse *retHttp = [response objectForKey:@"2"];

    // conver response body to NSString
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
   // retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);

    // if the result of request is "LOGIN_SUCCESS"
    if ([retString isEqualToString:@"LOGIN_SUCCESS"]) {
        //generate http response objection
        NSHTTPURLResponse *HTTPResponse = (NSHTTPURLResponse *)retHttp;
        //get headerfields
        NSDictionary *fields = [HTTPResponse allHeaderFields]; NSURLConnection
        // NSDictionary *fields = [operation.response allHeaderFields]; //afnetworking
        NSLog(@"fields = %@",[fields description]);

        //NSArray *cookies = [[NSHTTPCookieStorage sharedHTTPCookieStorage] cookies];

        // get cookies from http response -- start --
        NSData *cookieData = [NSKeyedArchiver archivedDataWithRootObject:[[NSHTTPCookieStorage
sharedHTTPCookieStorage] cookies]];
        [[NSUserDefaults standardUserDefaults] setObject:cookieData forKey:@"ApplicationCookie"];
        [[NSUserDefaults standardUserDefaults] synchronize];
        // get cookies from http response -- end --

        // save account to singleton object
        [Singleton sharedManager].userAccount = self.textAccount.text;

        // generate webapi objection
        WebAPI *wb = [[WebAPI alloc]init];

        // init request with the name of "get_user_data"
        [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"get_user_data"];

        // generate nsmutable dictionary
        NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
```

```
        // save account in the http-body
        [muDict setObject:self.textAccount.text forKey:@"account"];

        // make request
        [wb webSimplePost:muDict cb:@"UserDataNotification"];
    } else {

        // login failed
        // show alert
        dispatch_async(dispatch_get_main_queue(), ^{
            UIAlertView *alertView = [[UIAlertView alloc]
                                      initWithTitle:@"Alert"
                                      message:retString
                                      delegate:self
                                      cancelButtonTitle:@"No"
                                      otherButtonTitles:@"Yes", nil];
            [alertView show];
        });
    }
}

-(void)onUserDataHttpResponse:(NSNotification *)notification {
    // generate dictionary to save the data of response
    NSDictionary * response = [notification userInfo];

    // get response body in binary
    NSData *retData = [response objectForKey:@"1"];

    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    // conver response body to NSString
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    // convert response data from binary to dictionary -- start --
    NSLog(@"Response String:%@",retString);
    NSError *jsonError;
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                         options:NSJSONReadingMutableContainers
                                                           error:&jsonError];
    // convert response data from binary to dictionary -- end --

    // save json to singleton object
    [[Singleton sharedManager] Update:json];


    // show user interface
    dispatch_async(dispatch_get_main_queue(), ^{
        [self performSegueWithIdentifier:@"ShowMainUI" sender:self];
```

```
    });

}


// discarded function
- (void)onRegisterSuccess:(NSNotification*)notification {
    /*
    dispatch_async(dispatch_get_main_queue(), ^{
        NSDictionary *theData = [notification userInfo];
        NSString *username = [theData objectForKey:@"username"];
        self.textAccount.text = username ;
    });
    */
}

// this function used to encrypt function
- (NSData *)encrypt:(NSData *)data {
    char *dataP = (char *)[data bytes];
    NSString* tmpstr = @"1#9#8#9#1#0#2#6";
    NSArray* strArr = [tmpstr componentsSeparatedByString: @"#"];
    for (int i = 0; i < data.length; i++) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsequenced"
        *dataP = *(++dataP) ^ [[strArr objectAtIndex:i% [strArr count]] integerValue];
#pragma clang diagnostic pop
    }
    return data;
}

@end

//
//  DoctorListViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/24.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "support.h"
#import "RequestViewController.h"

@interface DoctorListViewController : UIViewController
<UITableViewDataSource,UITableViewDelegate>
@property (strong,nonatomic) NSDictionary *doctorDict;
@property (strong,nonatomic) NSArray *doctorList ;
@end
```

```
//
//  DoctorListViewController.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/24.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "DoctorListViewController.h"

@interface DoctorListViewController ()
@property (weak, nonatomic) IBOutlet UITableView *tableView;


@end

@implementation DoctorListViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    //set delegate of table view
    self.tableView.delegate = self;
    //set data source of table view
    self.tableView.dataSource = self;
    //set the string as title
    self.title = @"Doctor List" ;
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)test:(id)sender {
}



// function to handle the sections of the tableview
#pragma mark -
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // set the section number as the account of doctor
    return [[Singleton sharedManager].userDoctorList count];
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // generate nsstring as cell identifier
    static NSString *CellIdentifier = @"DoctorCell";

    // generate a local var
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
forIndexPath:indexPath];

    // set row as current row
    NSInteger row = [indexPath row];

    // get dictionary data from singleton , use row as index
    NSDictionary *dict = [[Singleton sharedManager].userDoctorList objectAtIndex:row];

    // set the value of "account" in the dictionary as text label
    cell.textLabel.text =[dict objectForKey:@"account"];

    // set the value of "gender" in the dictionary as detail label
    cell.detailTextLabel.text = [dict objectForKey:@"gender"];

    // generate local var
    NSString *imagePath = [dict objectForKey:@"gender"];

    // generate the path of image
    imagePath = [imagePath stringByAppendingString:@".ico"];

    // set the image view
    cell.imageView.image = [UIImage imageNamed:imagePath];

    // set the access type
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;

    return cell;
}

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{

    if([segue.identifier isEqualToString:@"ShowMakeRequest"])
    {
        // if the identifier of Segue is "ShowMakeRequest"

        // generate a object to hold the view controller
        RequestViewController *requestViewController = segue.destinationViewController;

        // get selected index
        NSInteger selectedIndex = [[self.tableView indexPathForSelectedRow] row];
```

```
        // get the data in dictionary at selected index
        NSDictionary *dict = [[Singleton sharedManager].userDoctorList objectAtIndex:selectedIndex];

        // set doctoraccount to the value of "account" in the dictionary
        requestViewController.doctorAccout =[dict objectForKey:@"account"];

        // refresh the view contronller
        [requestViewController reloadAllData];
    }
}

/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end

//
//  RequestViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/26.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "support.h"

@interface RequestViewController : UIViewController
<UITableViewDataSource,UITableViewDelegate,UIImagePickerControllerDelegate,UINavigationContr
ollerDelegate,UITextFieldDelegate>

@property (nonatomic, strong) NSString *doctorAccout;

-(void)reloadAllData;


@end

//
//  RequestViewController.m
//  UI_TEST_2
//
```

```
//  Created by Hussein Khairallah on 2016/11/26.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "RequestViewController.h"

@interface RequestViewController ()
@property (strong, nonatomic) NSArray *listDocInfo ;
@property (strong, nonatomic) NSDictionary *dictDoctorInfo ;
@property (weak, nonatomic) IBOutlet UITableView *tableView;
@property (weak, nonatomic) IBOutlet UIImageView *imageView1;
@property (weak, nonatomic) IBOutlet UIImageView *imageView2;
@property (weak, nonatomic) IBOutlet UITextView *textView;

@property int lr ;
@property NSString* luid ;
@property NSString* ruid ;
@property UIAlertView * alert;
@end

@implementation RequestViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.


    // set the notification to handle response of upload photo
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onUploadHttpResponse:)
                                                 name:@"UploadPhotoNotification"
                                               object:nil];

    // set the notification to handle response of request
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onMakeRequestHttpResponse:)
                                                 name:@"MakeRequestNotification"
                                               object:nil];



    // set all the value to show in the dictionary
    self.listDocInfo = [NSArray arrayWithObjects:@"LastName",
                        @"FirstName",
                        @"Gender",
                        @"Hospital",
                        @"Skype",
                        @"Viber",
                        @"Specialty",
                        nil];
```

```
    // set the delegate of tableview
    self.tableView.delegate = self;

    // set  the data source of tableview
    self.tableView.dataSource = self ;


}

- (IBAction)uploadPhoto:(id)sender {
    // reset lr to 0
    self.lr = 0;

    // generate object to hold the upload button
    UIButton *button = (UIButton *)sender ;

    // if the title is "Upload Left"  set lr to 1
    // else set lr to 2
    if ([button.titleLabel.text isEqualToString:@"Upload Left"]){
        self.lr = 1 ;
    } else {
        self.lr = 2 ;
    }

    // log
    NSLog(@"%@", button.titleLabel.text);

    // generate UIImagePickerController object
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];

    //set self to the delegate of picker
    picker.delegate = self;

    //set the allowEditing to NO
    //the picker can not be edited
    picker.allowsEditing = NO;

    // set the sourceType
    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

    // set the present view controller
    [self presentViewController:picker animated:YES completion:nil];
}


#pragma mark - delegate function
```

```
-(void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary<NSString *,id> *)info{

    // get the chosen image from stack
    UIImage* chosenImage = info[UIImagePickerControllerOriginalImage];

    // if self.lr == 1
    // set the chosen image to imageview1
    // else
    // set the chosen image to imageview2
    if (self.lr == 1) {
        self.imageView1.image = chosenImage;
    }else {
        self.imageView2.image = chosenImage;
    }

    //set the animate option
    [picker dismissViewControllerAnimated:YES completion:nil];

    //convert chosenImage to binary data
    NSData* imageData = UIImageJPEGRepresentation(chosenImage,1.0);
    //[self.alert show];

    //generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    //init the webapi with "upload_file"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"upload_file"];

    //make the request
    [wb webUploadData:imageData cb:@"UploadPhotoNotification"];
}

// system call back function
// no need to understand
-(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker{
    [picker dismissViewControllerAnimated:YES completion:nil];
}

// system call back function
// no need to understand
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

-(void)reloadAllData
{
    //log
    NSLog(@"RequestViewController Try to reload all data");
```

```
    //log
    NSLog(@"account:%@",self.doctorAccount);

    //reload the tableview
    [self.tableView reloadData];

}

- (void)onDoctorInfoHttpResponse:(NSNotification*)notification {

}

- (void)onUploadHttpResponse:(NSNotification*)notification {

    //----------get response from stack ------ start----
    NSDictionary * response = [notification userInfo];
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    //----------get response from stack ------ end----
    NSLog(@"Response String:%@",retString);
    NSError *jsonError;

    // unseralize the response from json to dictionary
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                         options:NSJSONReadingMutableContainers
                                                           error:&jsonError];

    // get the result from json by the key "result"
    if ([json objectForKey:@"result"]) {
        if (self.lr == 1) {
            self.luid = [json objectForKey:@"uid"];
        }else {
            self.ruid = [json objectForKey:@"uid"];
        }
    }

    // update the user interface  in a queue
    dispatch_async(dispatch_get_main_queue(), ^{
        self.textView.text = [ self.textView.text stringByAppendingString:@"Upload File Success!\n"];

    });
   }
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // get the count of elements in the tableview
    return [self.listDocInfo count];
}


- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // generate the identifier of cell
    static NSString *CellIdentifier = @"DoctorInfoCell";

    // generate a object hold the cell
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
forIndexPath:indexPath];

    // generate the row
    NSInteger row = [indexPath row];

    // get the key from listDocInfo by "row"
    NSString *key = [self.listDocInfo objectAtIndex:row];

    // set the label of cell
    cell.textLabel.text = key ;

    // generate key with "str"
    key = [@"str" stringByAppendingString:key];

    // get dictionary from singleton
    NSDictionary *dict = [[Singleton sharedManager].doctorListDetail objectForKey:self.doctorAccout];

    //set the detailLabel
    cell.detailTextLabel.text = [dict objectForKey:key];
    return cell;
}

// system call back function
// no need to understand
- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event{
    [self.textView resignFirstResponder];
   }


- (IBAction)doMakeRequest:(id)sender {
    // get doctor account from singleton
    NSString *doctorAccount = [[[Singleton sharedManager].doctorListDetail
```

```
objectForKey:self.doctorAccout] objectForKey:@"strAccount"];

    // get comment from user interface textView
    NSString *comment = self.textView.text ;

    // get id for left image
    NSString *uidForLeft = self.luid;

    // get id for right image
    NSString *uidForRight = self.ruid ;

    // generate mutable dictionary
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];

    // set information to mutable dictionary
    [muDict setObject:doctorAccount forKey:@"doctor"];

    // set information to mutable dictionary
    [muDict setObject:comment forKey:@"comment"];

    // set information to mutable dictionary
    [muDict setObject:uidForLeft forKey:@"uidl"];

    // set information to mutable dictionary
    [muDict setObject:uidForRight forKey:@"uidr"];

    // log
    NSLog(@"make_request:%@",muDict);

    // generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "make_request"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"make_request"];

    // make request
    [wb webSimplePost:muDict cb:@"MakeRequestNotification"];
}

- (void)onMakeRequestHttpResponse:(NSNotification*)notification {
    // ---------- get response from http ---- start-----
    NSDictionary * response = [notification userInfo];
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);
```

```
    // ---------- get response from http ---- end-----

    NSError *jsonError;

    // unseralize data from json to NSDictionary
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                         options:NSJSONReadingMutableContainers
                                                           error:&jsonError];

    // call the function  to update
    [[Singleton sharedManager]UpdateWithHttp ];

    dispatch_async(dispatch_get_main_queue(), ^{
        UIAlertView *alertView = [[UIAlertView alloc]
                                  initWithTitle:@"INFO"
                                  message:@"MAKE REQUEST SUCCESS!"
                                  delegate:self
                                  cancelButtonTitle:@"OK"
                                  otherButtonTitles:nil, nil];
        [alertView show];
    });

}

- (IBAction)callSkype:(id)sender {
    NSString* acc = [[[Singleton sharedManager].doctorListDetail objectForKey:self.doctorAccount]
objectForKey:@"strSkype"];
    NSLog(@"%@",acc);
    NSString* url=@"skype:";
    url = [url stringByAppendingString:acc];
    url = [url stringByAppendingString:@"?call"];

    BOOL installed = [[UIApplication sharedApplication] canOpenURL:[NSURL
URLWithString:@"skype:"]];
    if(installed){
        //skype:your_skype_account?call hkairalla@yahoo.com
        [[UIApplication sharedApplication] openURL:[NSURL URLWithString:url]];
    } else {
        [[UIApplication sharedApplication] openURL:[NSURL
URLWithString:@"https://itunes.apple.com/in/app/skype/id304878510?mt=8"]];
    }
}

- (IBAction)callViber:(id)sender {
    NSString* acc = [[[Singleton sharedManager].doctorListDetail objectForKey:self.doctorAccount]
objectForKey:@"strViber"];
    NSLog(@"%@",acc);
    // viber://contact?number= 3132647101
    NSString* url = [NSString stringWithFormat:@"viber://contact?number=%@", acc];
```

```
    BOOL installed = [[UIApplication sharedApplication] canOpenURL:[NSURL
URLWithString:@"viber://"]];
    if(installed){
        [[UIApplication sharedApplication] openURL:[NSURL URLWithString:url]];
    } else {
        [[UIApplication sharedApplication] openURL:[NSURL
URLWithString:@"https://itunes.apple.com/us/app/viber-messenger-text-call/id382617920?mt=8"]];
    }
}


/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end

//
//  DetaileViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/28.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "support.h"

@interface DetaileViewController : UIViewController<UITableViewDataSource,UITableViewDelegate>


@property (strong,nonatomic) NSString *requestID;

-(void)loadRequestDetail;
@end

//
//  DetaileViewController.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/28.
//  Copyright © year of 2016 com.turing. All rights reserved.
//
```

```
#import "DetaileViewController.h"
#import "XWScanImage.h"

@interface DetaileViewController ()

// image on the left
@property (weak, nonatomic) IBOutlet UIImageView *imageView1;

// image on the right
@property (weak, nonatomic) IBOutlet UIImageView *imageView2;

// user intface to type in comment
@property (weak, nonatomic) IBOutlet UITextView *testView;

// user interface to show the comment
@property (weak, nonatomic) IBOutlet UITextField *textField;

// table view show the information about doctor
@property (weak, nonatomic) IBOutlet UITableView *tableView;

// button to update left image
@property (weak, nonatomic) IBOutlet UIButton *UpdateLeft;

// button to update right image
@property (weak, nonatomic) IBOutlet UIButton *UpdateRight;

// button to slove the request
@property (weak, nonatomic) IBOutlet UIButton *SloveRequest;

// flag for update image
@property int lr ;

// dictionary for request
@property (strong,nonatomic) NSDictionary *requestDetail ;

// dictonary for doctor
@property (strong,nonatomic) NSDictionary *detailDoctorDict;

// list of doctor account
@property (strong,nonatomic) NSArray      *detailDoctorList;
@end

@implementation DetaileViewController

- (void)viewDidLoad {
   [super viewDidLoad];
   // Do any additional setup after loading the view.
```

```
    // set self to delegate of tableview
    self.tableView.delegate = self;

    // set self to datasource tableview
    self.tableView.dataSource = self ;


    // notification for detail information about doctor
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onDetailDoctorHttpResponse:)
                                                 name:@"DetailDoctorInfoNotification"
                                               object:nil];

    // notification for detail information about image clicked
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onDetailImageHttpResponse:)
                                                 name:@"DetailImageNotification"
                                               object:nil];

    // notification for update the comment
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(doUpdateComment:)
                                                 name:@"UpdateCommentNotification"
                                               object:nil];

    // notification for update the photo
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(doUpdatePhoto:)
                                                 name:@"UpdatePhotoNotification"
                                               object:nil];




    // ---------- setup imageView1 and imageView2 ------------ start ------------
    // setup the imageview for the event of tap
    // touch the imageview the photo will show details
    UITapGestureRecognizer *tapGestureRecognizer1 = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(scanBigImageClick1:)];
    [_imageView1 addGestureRecognizer:tapGestureRecognizer1];
    [_imageView1 setUserInteractionEnabled:YES];

    UITapGestureRecognizer *tapGestureRecognizer2 = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(scanBigImageClick2:)];
    [_imageView2 addGestureRecognizer:tapGestureRecognizer2];
    [_imageView2 setUserInteractionEnabled:YES];

    self.testView.layoutManager.allowsNonContiguousLayout = NO;
    // ---------- setup imageView1 and imageView2 ------------ end ------------
```

```
}

// system function
// no need to understand
-(void)scanBigImageClick1:(UITapGestureRecognizer *)tap{
    UIImageView *clickedImageView = (UIImageView *)tap.view;
    [XWScanImage scanBigImageWithImageView:clickedImageView];
}

// system function
// no need to understand
-(void)scanBigImageClick2:(UITapGestureRecognizer *)tap{
    UIImageView *clickedImageView = (UIImageView *)tap.view;
    [XWScanImage scanBigImageWithImageView:clickedImageView];

}


// system function
// no need to understand
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)doSendComment:(id)sender {
   // NSString strComment = [NSString stringWithFormat:[@"%@ says:\n%@" ];
                    //  [Singleton sharedManager].userAccount,self.textField.text];

    // get comment string from user interface
    NSString* strComment = [[NSString alloc] initWithFormat:@"%@ says:\n%@",[Singleton
sharedManager].userAccount,self.textField.text];

    // generate webapi
    WebAPI *wb = [[WebAPI alloc] init]  ;

    // init webapi with "update_comment"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"update_comment"];

    // generate mutable dictionary for http request --------start----------
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
    [muDict setObject:self.requestID forKey:@"requestID"];
    //[muDict setObject:self.textField.text forKey:@"addComment"];
    [muDict setObject:strComment forKey:@"addComment"];
    // generate mutable dictionary for http request --------end----------

    // make http request
    [wb webSimplePost:muDict cb:@"UpdateCommentNotification"];
}
```

```
- (IBAction)doUpdatePicture:(id)sender {

}


- (IBAction)doSloveQuest:(id)sender {
    NSLog(@"Slove the request:%@",self.requestDetail);

    // generate webapi
    WebAPI *wb = [[WebAPI alloc] init]  ;

    // init webapi with "slove_request"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"slove_request"];

    // generate mutable dictionary for http request --------start-----------
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
    [muDict setObject:[self.requestDetail objectForKey:@"patient"] forKey:@"patient"];
    [muDict setObject:[self.requestDetail objectForKey:@"doctor"] forKey:@"doctor"];
    [muDict setObject:self.requestID forKey:@"requestID"];
    // generate mutable dictionary for http request --------end-----------

    // make http request
    [wb webSimplePost:muDict cb:@"UpdateCommentNotification"];
}


// -------------------- update the picture ----------- start ----------
- (IBAction)doUpdatePictureLeft:(id)sender {
    NSLog(@"Update Picture Left");

    // update left image
    self.lr = 1 ;
    // setup picker for choseing image
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.delegate = self;
    picker.allowsEditing = NO;
    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

    [self presentViewController:picker animated:YES completion:nil];

}

- (IBAction)doUpdatePictureRight:(id)sender {
    NSLog(@"Update Picture Left");

    // update right image
    self.lr = 2 ;
    // setup picker for choseing image
```

```
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.delegate = self;
    picker.allowsEditing = NO;
    picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

    [self presentViewController:picker animated:YES completion:nil];
}
// -------------------- update the picture ----------- end ----------


#pragma mark
-(void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary<NSString *,id> *)info{
    // --------- chose image view ------ start ---
    UIImage* chosenImage = info[UIImagePickerControllerOriginalImage];
    if (self.lr == 1) {
        // chose left image
        self.imageView1.image = chosenImage;
    }else {
        // chose right image
        self.imageView2.image = chosenImage;
    }
    // --------- chose image view ------ end ---

    // setup picker
    [picker dismissViewControllerAnimated:YES completion:nil];

    // convert image to binary data
    NSData* imageData = UIImageJPEGRepresentation(chosenImage,1.0);
  //  [self.alert show];

    // generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "upload_file"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"upload_file"];

    // make request
    [wb webUploadData:imageData cb:@"UpdatePhotoNotification"];
}



// update user interface comment
- (void) doUpdateComment:(NSNotification*)notification {
    dispatch_async(dispatch_get_main_queue(), ^{
        [[Singleton sharedManager] Update:nil];

        // generate comment
        NSString* strComment = [[NSString alloc] initWithFormat:@"%@ says:\n%@",[Singleton
```

```
sharedManager].userAccount,self.textField.text];

        // append comment string
        self.testView.text = [self.testView.text stringByAppendingString:strComment];

        // append comment string
        self.testView.text = [self.testView.text stringByAppendingString:@"\n"];

        // append.comment string
        self.textField.text = @"";

        // scroll to the end
        [self.testView scrollRangeToVisible:NSMakeRange(self.testView.text.length, 1)];

    });

}

- (void) doUpdatePhoto:(NSNotification*)notification {
    dispatch_async(dispatch_get_main_queue(), ^{
        //update comment
        NSString* strComment = [[NSString alloc] initWithFormat:@"%@ says:\n%@",@"update photo
sucess!",self.textField.text];

        // append comment string
        self.testView.text = [self.testView.text stringByAppendingString:strComment];

        // append comment string
        self.testView.text = [self.testView.text stringByAppendingString:@"\n"];

        // append comment string
        self.textField.text = @"";

        // scroll to the end
        [self.testView scrollRangeToVisible:NSMakeRange(self.testView.text.length, 1)];

        //update request
        // get resopnse from http stack ---------- start ----------
        NSDictionary * response = [notification userInfo];
        NSData *retData = [response objectForKey:@"1"];
        //NSURLResponse *retHttp = [response objectForKey:@"2"];
        NSString  *retString = [[NSString alloc] initWithData:[retData
subdataWithRange:NSMakeRange(0, [retData length]-1)] encoding:NSUTF8StringEncoding];
        // retString = StringWithData([self encrypt: DataWithString(retString)]);

        NSLog(@"Response String:%@",retString);
        NSError *jsonError;
        // get resopnse from http stack ---------- end ----------
```

```
        // unseralize the response from json to NSDictionary
        NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                options:NSJSONReadingMutableContainers
                                                  error:&jsonError];

        // if http result is sucess
        if ([json objectForKey:@"result"]) {

            // generate webapi
            WebAPI *wb = [[WebAPI alloc]init];

            // init the webapi with "request_update_photo"
            [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"request_update_photo"];

            // generate mutable dictionary for request --- start ---
            NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
            [muDict setObject:[NSString stringWithFormat:@"%d",self.lr] forKey:@"lr"];
            [muDict setObject:self.requestID forKey:@"request_id"];
            [muDict setObject:[json objectForKey:@"uid"] forKey:@"photo_uid"];
            // generate mutable dictionary for request --- end ---

            // make http request
            [wb webSimplePost:muDict cb:@""];
        }

    });
}

-(void)loadRequestDetail {
    // get request detail from singleton
    self.requestDetail = [[Singleton sharedManager].requestListDetail objectForKey:self.requestID];

    // update user interface in the queue
    dispatch_async(dispatch_get_main_queue(), ^{

        // update textview
        self.testView.text = [self.requestDetail objectForKey:@"comment"];

        // scroll to the end
        [self.testView scrollRangeToVisible:NSMakeRange(self.testView.text.length, 1)];

    });


    // generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "download_data"
```

```
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"download_data"];

    // generate mutable dictionary for http request --- start ---
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
    NSString *key = [@"upload:data:" stringByAppendingString:[self.requestDetail
objectForKey:@"uidl"]];
    [muDict setObject:key forKey:@"key"];
    // generate mutable dictionary for http request --- end ---

    // make http request
    [wb webSimplePost:muDict cb:@"DetailImageNotification"];

}


- (void)onDetailHttpResponse:(NSNotification*)notification {
    // ---------- get response from http stack ----- start ----
    NSDictionary * response = [notification userInfo];
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);
    NSError *jsonError;
    // ---------- get response from http stack ----- end ----

    // unseralize http response from json to nsdictionary
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                         options:NSJSONReadingMutableContainers
                                           error:&jsonError];

    // set json to requestdetail
    self.requestDetail = json;

    // update comment
    dispatch_async(dispatch_get_main_queue(), ^{
        self.testView.text = [self.requestDetail objectForKey:@"comment"];
    });

    // generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "doctor_info"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"doctor_info"];

    // generate nsmutabledictionary for http request ------- start --------
    NSMutableDictionary *mudict = [[NSMutableDictionary alloc]init];
```

```
    [mudict setObject:[self.requestDetail objectForKey:@"doctor"] forKey:@"account"];
    // generate nsmutabledictionary for http request ------- end --------

    // make request
    [wb webSimplePost:mudict cb:@"DetailDoctorInfoNotification"];

}


- (void)onDetailDoctorHttpResponse:(NSNotification*)notification {
    // ---------- get response from http stack ----- start ----
    NSDictionary * response = [notification userInfo];
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);
    NSError *jsonError;
    // ---------- get response from http stack ----- end ----

    // unseralize http response from json to nsdictionary
    NSDictionary *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                         options:NSJSONReadingMutableContainers
                                                           error:&jsonError];

    // set local var
    self.detailDoctorDict = json;
    self.detailDoctorList = [json allKeys];
    [self.tableView reloadData];

    // generate webapi
    WebAPI *wb = [[WebAPI alloc]init];

    // init webapi with "download_data"
    [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"download_data"];

    // generate nsmutabledictionary for http request ------- start --------
    NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
    NSString *key = [@"upload:data:" stringByAppendingString:[self.requestDetail
objectForKey:@"uidl"]];
    [muDict setObject:key forKey:@"key"];
    // generate nsmutabledictionary for http request ------- end --------

    // make http request
    [wb webSimplePost:muDict cb:@"DetailImageNotification"];
}
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{

    // count in the tableview
    return  [[NSArray arrayWithObjects:@"LastName",@"FirstName",@"Gender",@"Hospital", nil]
count];
}


- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // get the indetifier of cell
    static NSString *CellIdentifier = @"DetailDoctorCell";

    // generate cell
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
forIndexPath:indexPath];


    // generate arry for information
    NSArray * arr = [NSArray arrayWithObjects:@"LastName",
                     @"FirstName",
                     @"Gender",
                     @"Hospital",
                     nil];

    // ------------- setup detail in the cell --------- start -----------
    NSInteger row = [indexPath row];
    NSString* key = [arr objectAtIndex:row];
    NSString* doctor = [self.requestDetail objectForKey:@"doctor"];
    NSDictionary* dict = [[Singleton sharedManager].doctorListDetail objectForKey:doctor];
    cell.textLabel.text = key ;

    cell.detailTextLabel.text = [dict objectForKey: [@"str" stringByAppendingString:key]];

    // ------------- setup detail in the cell --------- end -----------
    return cell;
}

// system function
// no need to understand
- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event{
    [self.textField resignFirstResponder];
}

- (void)onDetailImageHttpResponse:(NSNotification*)notification {
    // ---------- get response from http stack ----- start ----
    NSDictionary * response = [notification userInfo];
```

```
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    // ---------- get response from http stack ----- end ----

    // convert image to binary data
    NSData *myImageData = [retData subdataWithRange:NSMakeRange(0, [retData length]-1)];
    if (!self.imageView1.image) {
        // update left image interface in the queue
        dispatch_async(dispatch_get_main_queue(), ^{
            self.imageView1.image = [[UIImage alloc] initWithData:myImageData];
        });

        // generate webapi
        WebAPI *wb = [[WebAPI alloc]init];

        // init webapi with "download_data"
        [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"download_data"];

        // generate mutable dictionary for http request ----- start ---------
        NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
        NSString *key = [@"upload:data:" stringByAppendingString:[self.requestDetail
objectForKey:@"uidr"]];
        [muDict setObject:key forKey:@"key"];
        // generate mutable dictionary for http request ----- end ---------

        // make http request
        [wb webSimplePost:muDict cb:@"DetailImageNotification"];

    } else if (!self.imageView2.image) {

        //update right image
        dispatch_async(dispatch_get_main_queue(), ^{
            self.imageView2.image = [[UIImage alloc] initWithData:myImageData];
        });

        return ;
    }
}



/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/
```

```
@end

//
//  MyReqsViewController.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/26.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "support.h"
#import "DetaileViewController.h"

@interface MyReqsViewController :
UIViewController<UITableViewDataSource,UITableViewDelegate>

@end

//
//  MyReqsViewController.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/26.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "MyReqsViewController.h"

@interface MyReqsViewController ()
@property (weak, nonatomic) IBOutlet UITableView *tableView;
@property (strong,nonatomic) NSArray *listData ;
@property (strong,nonatomic) NSDictionary *dictData;
@property (weak, nonatomic) IBOutlet UIButton *update;

@end

@implementation MyReqsViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // notification for myreqeust
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(onMyRequestHttpResponse:)
                                                 name:@"MyRequestsNotification"
                                               object:nil];
```

```
    self.tableView.delegate = self;
    self.tableView.dataSource = self ;
}

// system function
// no need to understand
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


- (IBAction)doUpdate:(id)sender {
    // update singleton
    [[Singleton sharedManager]UpdateWithHttp];

    // update table view
    [self.tableView reloadData];
}


- (void)onMyRequestHttpResponse:(NSNotification*)notification {
    // ---------- get response from http stack ----- start ----

    NSDictionary * response = [notification userInfo];
    NSData *retData = [response objectForKey:@"1"];
    //NSURLResponse *retHttp = [response objectForKey:@"2"];
    NSString  *retString = [[NSString alloc] initWithData:[retData subdataWithRange:NSMakeRange(0,
[retData length]-1)] encoding:NSUTF8StringEncoding];
    // retString = StringWithData([self encrypt: DataWithString(retString)]);

    NSLog(@"Response String:%@",retString);
    NSError *jsonError;
    // ---------- get response from http stack ----- end ----

    // unseralize json to nsarray
    NSArray *json = [NSJSONSerialization JSONObjectWithData:[retString
dataUsingEncoding:NSUTF8StringEncoding]
                                                    options:NSJSONReadingMutableContainers
                                                      error:&jsonError];
    self.listData = json;

    // reload table view
    [self.tableView reloadData];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // set count int the tableview
    return [[Singleton sharedManager].userRequestList count];
```

```
}


- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // get cell name
    static NSString *CellIdentifier = @"RequestCell";

    // get cell by name
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
forIndexPath:indexPath];

    // setup the detail
    NSInteger row = [indexPath row];
    cell.textLabel.text = [[Singleton sharedManager].userRequestList objectAtIndex:row];


    return cell;
}

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // get the view controller
    DetaileViewController *detaileViewController = segue.destinationViewController;

    // get the index
    NSInteger selectedIndex = [[self.tableView indexPathForSelectedRow] row];

    // set the requestID by the data in the singleton
    detaileViewController.requestID =[[Singleton sharedManager].userRequestList
objectAtIndex:selectedIndex];

    // load details
    [detaileViewController loadRequestDetail];
}


/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end
```

```
//
//  XWScanImage.h
//  XWScanImageDemo
//


#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>

@interface XWScanImage : NSObject

+(void)scanBigImageWithImageView:(UIImageView *)currentImageview;
@end

//
//  XWScanImage.m
//  XWScanImageDemo
//


#import "XWScanImage.h"

@implementation XWScanImage


static CGRect oldframe;

+(void)scanBigImageWithImageView:(UIImageView *)currentImageview{

    UIImage *image = currentImageview.image;

    UIWindow *window = [UIApplication sharedApplication].keyWindow;

    UIView *backgroundView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, [UIScreen
mainScreen].bounds.size.width, [UIScreen mainScreen].bounds.size.height)];

    oldframe = [currentImageview convertRect:currentImageview.bounds toView:window];
    [backgroundView setBackgroundColor:[UIColor colorWithRed:107/255.0 green:107/255.0
blue:99/255.0 alpha:0.6]];

    [backgroundView setAlpha:0];

    UIImageView *imageView = [[UIImageView alloc] initWithFrame:oldframe];
    [imageView setImage:image];
    [imageView setTag:0];
    [backgroundView addSubview:imageView];

    [window addSubview:backgroundView];
```

```
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(hideImageView:)];
    [backgroundView addGestureRecognizer:tapGestureRecognizer];


    [UIView animateWithDuration:0.4 animations:^{
        CGFloat y,width,height;
        y = ([UIScreen mainScreen].bounds.size.height - image.size.height * [UIScreen
mainScreen].bounds.size.width / image.size.width) * 0.5;

        width = [UIScreen mainScreen].bounds.size.width;

        height = image.size.height * [UIScreen mainScreen].bounds.size.width / image.size.width;
        [imageView setFrame:CGRectMake(0, y, width, height)];

        [backgroundView setAlpha:1];
    } completion:^(BOOL finished) {

    }];

}


+(void)hideImageView:(UITapGestureRecognizer *)tap{
    UIView *backgroundView = tap.view;

    UIImageView *imageView = [tap.view viewWithTag:0];

    [UIView animateWithDuration:0.4 animations:^{
        [imageView setFrame:oldframe];
        [backgroundView setAlpha:0];
    } completion:^(BOOL finished) {

        [backgroundView removeFromSuperview];
    }];
}

@end

//
//  UserDataModel.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/30.
//  Copyright © year of 2016 com.turing. All rights reserved.
//
```

```
#import <Foundation/Foundation.h>

#define StringWithData(data) [[NSString alloc] initWithData:(data) encoding:NSUTF8StringEncoding]
#define DataWithString(str) [(str) dataUsingEncoding:NSUTF8StringEncoding]

@interface Singleton : NSObject

+ (Singleton*)sharedManager;
@property (nonatomic ,strong) NSString* singletonData;

@property (nonatomic, strong) NSString* userAccount;

@property (nonatomic, strong) NSString* server_ip;

//use for MyReqsViewController
@property (nonatomic, strong) NSArray* userRequestList;

//use for DoctorListViewController
@property (nonatomic, strong) NSArray* userDoctorList;
@property (nonatomic, strong) NSDictionary *userDoctorListDict;

//use for requestViewController
@property (nonatomic,strong) NSMutableDictionary *doctorListDetail;

//use for detaileViewController
@property (nonatomic, strong) NSMutableDictionary *requestListDetail ;

-(void)Update:(NSDictionary*) dict;

-(void)UpdateWithHttp;
@end

//
//  UserDataModel.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/11/30.
//  Copyright © year of 2016 com.turing. All rights reserved.
//


// singleton code
// no need to understand

#import "UserDataModel.h"
#import "WebAPI.h"
```

```
@implementation Singleton

@synthesize singletonData = _singletonData;
@synthesize userAccount = _userAccount;

@synthesize userRequestList = _userRequestList;

@synthesize userDoctorList = _userDoctorList;
@synthesize userDoctorListDict = _userDoctorListDict;

@synthesize doctorListDetail = _doctorListDetail;
@synthesize server_ip = _server_ip;


static Singleton *sharedManager = nil;

+ (Singleton*)sharedManager
{
   static dispatch_once_t once;
   dispatch_once(&once, ^{
      sharedManager = [[self alloc] init];
   });
   return sharedManager;
}

-(void)UpdateWithHttp{
   WebAPI *wb = [[WebAPI alloc]init];
   [wb initWithUrlAndApi:[Singleton sharedManager].server_ip api:@"get_user_data"];
   NSMutableDictionary *muDict = [[NSMutableDictionary alloc]init];
   [muDict setObject:_userAccount forKey:@"account"];
   [wb webSimplePost:muDict cb:@"UserDataNotification"];
}

-(void)Update:(NSDictionary *)dict
{
   if ([dict objectForKey:@"requests"]) {
      self.userRequestList = [dict objectForKey:@"requests"];
   }

   if ([dict objectForKey:@"doctors"]) {
      self.userDoctorList =[dict objectForKey:@"doctors"] ;
   }

   [self UpdateDoctorDetail];
   [self UpdateRequestDetail];
}

-(void)UpdateDoctorDetail
{
   if (!self.doctorListDetail) {
```

```
        self.doctorListDetail = [[NSMutableDictionary alloc]init];
    }

    NSEnumerator *enumerator = [self.userDoctorList objectEnumerator];

    id key;
    while ((key = [enumerator nextObject])) {
        NSLog(@"UpdateDoctorDetail, key:%@",key);

        if (![self.doctorListDetail objectForKey:[key objectForKey:@"account"] ]) {
            NSLog(@"enter WebAPI:webSimplePost");
            NSURL *url = [NSURL URLWithString:[NSString
stringWithFormat:@"http://%@/api/%@",[Singleton sharedManager].server_ip,@"doctor_info"]];
            if (!url) {
                NSLog(@"WebAPI:webUploadImage get url failed:%@",url);
                return ;
            }
            NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
            if (!request) {
                NSLog(@"WebAPI:webUploadImage get request failed:%@",request);
                return ;
            }

            NSError *error;
            NSMutableDictionary *postData = [[NSMutableDictionary alloc]init];
            [postData setObject:[key objectForKey:@"account"]  forKey:@"account"];

            NSData *jsonData = [NSJSONSerialization dataWithJSONObject:postData
                                                               options:NSJSONWritingPrettyPrinted // Pass 0 if you don't care
about the readability of the generated string
                                                                 error:&error];
            if (! jsonData) {
                NSLog(@"Got an error: %@", error);
                continue ;
            }

            NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
            NSString *jsonStringSend = StringWithData([self encrypt: DataWithString(jsonString)]);

            NSLog(@"json send:%@",jsonStringSend);
            NSLog(@"json send:%@",StringWithData([self encrypt: DataWithString(jsonStringSend)]));
            //http-method
            request.HTTPMethod = @"POST";
            //http-body
            request.HTTPBody = [jsonStringSend dataUsingEncoding:NSUTF8StringEncoding] ;



            NSURLSession *session = [NSURLSession sharedSession];
```

```
        NSURLSessionDataTask *sessionDataTask = [session dataTaskWithRequest:request
                                                 completionHandler:^(NSData * _Nullable data, NSURLResponse
* _Nullable response, NSError * _Nullable error) {
                                                     NSString  *retString = [[NSString alloc] initWithData:[data
subdataWithRange:NSMakeRange(0, [data length]-1)] encoding:NSUTF8StringEncoding];

                                                     NSError *jsonError;
                                                     NSDictionary *json = [NSJSONSerialization
JSONObjectWithData:[retString dataUsingEncoding:NSUTF8StringEncoding]

options:NSJSONReadingMutableContainers
                                                                                 error:&jsonError];
                                                     [self.doctorListDetail setObject:json forKey:[json
objectForKey:@"strAccount"]];
                                                 }];
        if (!sessionDataTask) {
            NSLog(@"WebAPI:webUploadImage get sessionDataTask failed:%@",sessionDataTask);
            return ;
        }

        [sessionDataTask resume];
    }

  }
}

-(void)UpdateRequestDetail
{
  if (!self.requestListDetail) {
    self.requestListDetail = [[NSMutableDictionary alloc]init];
  }

  NSEnumerator *enumerator = [self.userRequestList objectEnumerator];

  id key;
  while ((key = [enumerator nextObject])) {
    NSLog(@"UpdateRequestDetail, key:%@",key);


    if (![self.doctorListDetail objectForKey:key]) {
        NSLog(@"enter WebAPI:webSimplePost");
        NSURL *url = [NSURL URLWithString:[NSString
stringWithFormat:@"http://%@/api/%@",[Singleton sharedManager].server_ip,@"request_info"]];
        if (!url) {
            NSLog(@"WebAPI:webUploadImage get url failed:%@",url);
            return ;
        }
        NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
        if (!request) {
            NSLog(@"WebAPI:webUploadImage get request failed:%@",request);
```

```
            return ;
        }

        NSError *error;
        NSMutableDictionary *postData = [[NSMutableDictionary alloc]init];
        [postData setObject:key forKey:@"request"];

        NSData *jsonData = [NSJSONSerialization dataWithJSONObject:postData
                                                           options:NSJSONWritingPrettyPrinted // Pass 0 if you don't care
about the readability of the generated string
                                                             error:&error];

        if (! jsonData) {
            NSLog(@"Got an error: %@", error);
            continue ;
        }

        NSString *jsonString = [[NSString alloc] initWithData:jsonData
encoding:NSUTF8StringEncoding];
        NSString *jsonStringSend = StringWithData([self encrypt: DataWithString(jsonString)]);

        NSLog(@"json send:%@",jsonStringSend);
        NSLog(@"json send:%@",StringWithData([self encrypt: DataWithString(jsonStringSend)]));
        //http-method
        request.HTTPMethod = @"POST";
        //http-body
        request.HTTPBody = [jsonStringSend dataUsingEncoding:NSUTF8StringEncoding] ;



        NSURLSession *session = [NSURLSession sharedSession];
        NSURLSessionDataTask *sessionDataTask = [session dataTaskWithRequest:request
                                                           completionHandler:^(NSData * _Nullable data, NSURLResponse
* _Nullable response, NSError * _Nullable error) {
                                                               NSString  *retString = [[NSString alloc] initWithData:[data
subdataWithRange:NSMakeRange(0, [data length]-1)] encoding:NSUTF8StringEncoding];

                                                               NSError *jsonError;
                                                               NSDictionary *json = [NSJSONSerialization
JSONObjectWithData:[retString dataUsingEncoding:NSUTF8StringEncoding]

options:NSJSONReadingMutableContainers
                                                                                                    error:&jsonError];
                                                               [self.requestListDetail setObject:json forKey:key];
                                                           }];
        if (!sessionDataTask) {
            NSLog(@"WebAPI:webUploadImage get sessionDataTask failed:%@",sessionDataTask);
            return ;
        }
```

```
            [sessionDataTask resume];
        }

    }

}

- (NSData *)encrypt:(NSData *)data {
    char *dataP = (char *)[data bytes];
    NSString* tmpstr = @"1#9#8#9#1#0#2#6";
    NSArray* strArr = [tmpstr componentsSeparatedByString: @"#"];
    for (int i = 0; i < data.length; i++) {
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsequenced"
        *dataP = *(++dataP) ^ [[strArr objectAtIndex:i% [strArr count]] integerValue];
#pragma clang diagnostic pop
    }
    return data;
}

@end

//
//  RegisterCheck.h
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/12/2.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface RegisterCheck : NSObject

+(BOOL)CheckPassword:(NSString*)passwd confirm:(NSString*)confirm error:(NSString**) error;

+(BOOL)CheckEmail:(NSString*)email ;
@end

//
//  RegisterCheck.m
//  UI_TEST_2
//
//  Created by Hussein Khairallah on 2016/12/2.
//  Copyright © year of 2016 com.turing. All rights reserved.
//

#import "RegisterCheck.h"

@implementation RegisterCheck
```

```
+(BOOL)CheckPassword:(NSString*)passwd confirm:(NSString*)confirm error:(NSString**) error
{
   if (![passwd isEqualToString:confirm]) {
      *error = @"password different from confirm";
      return NO;
   }

   // lenght of password must bigger than 8
   if ([passwd length] < 8) {
      *error = @"password must more thean 8 letters";
      return NO;
   }

   // must have lowercase in password
   if ([passwd isEqualToString:[passwd uppercaseString]]) {
      *error = @"must combine with uppercase and lowercase letter";
      return NO;
   }

   // must have uppercase in password
   if ([passwd isEqualToString:[passwd lowercaseString]]) {
      *error = @"must combine with uppercase and lowercase letter";
      return NO;
   }

   return YES;
}

// check the email
+(BOOL)CheckEmail:(NSString*)email
{
   NSString *emailRegex = @"[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}";
   NSPredicate *emailTest = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
emailRegex];
   return [emailTest evaluateWithObject:email];
}

@end
```

APPENDIX B

HEALTHCARE-MONITORING MOBILE APP

Collect Server:

The collect server consists of the following: server system and database for the server system. I used LINUX as the server system, which can be installed for any purpose and is more secure than windows. In the web server, we used Nginx, which is known for its high performance, stability, rich feature set, simple configuration and low resource consumption. The web server with Nginx is straightforward and convenient. For the language, I used LUA, a powerful and fast programming language that is easy to learn. For the database, I used REDIS, which is an open source database and memory.

Access_check.lua

```
local api = {}
-- import redis api pagkage
local redis = require "comm.redis"

-- this function check the cookies in the http body which client send to server
-- what is cookies ?
-- 1. Cookies are usually small text files
-- 2. Cookies are created when you use your browser to visit a website that uses cookies to keep track of your
movements within the site,

function api.check_cookies (cookies)
        -- Use regular expressions to resolve accounts and cookies
        _,_,acc,cookies=string.find (ngx.var.http_cookie,"name=%s*(.+),value=%s*(.+)")
        -- log information to file
        ngx.log(ngx.INFO,"!!!!!!!!!cookies-name!!!!!!!!!!!!!!!!!",acc)
        -- log information to file
        ngx.log(ngx.INFO,"!!!!!!!!!cookies-value!!!!!!!!!!!!!!!!!",cookies)

        -- create the key in redis database,which stores the account information
        -- for example: acc:test001
        local redis_acc_key = string.format ("acc:%s",acc)
        -- create the key in redis database,which stores the cookies of the account
        -- for example: test001:cookies
        local redis_cookies_key = string.format("%s:cookies",redis_acc_key)

        -- create a new connect to redis database
        local red = redis:new() ;
        -- get the cookies from redis database
```

```
	local ok,error = red:get(redis_cookies_key)
	-- if the cookies get from client http body equals the cookies stores in the database ,then return true
	-- if the cookies get from client http body dose not equal the cookies stores in the database, then
return false
	if ok == cookies then
		return true
	else
		return false
	end
end

return api
```

add.lua

```
-- This file is an API test
-- There is no need to understand
local args = ngx.req.get_uri_args()
ngx.log(ngx.ERR, "!!!!!!!!!!!!!!!args:!!!!!!!!!!!!!!!!!!", args.a, args.b)
ngx.log(ngx.INFO, "######################args:##################", args.a, args.b)
ngx.say(args.a+args.b)
```

doctor_info.lua

```
--[[
	This Lua Script handles the request "*/api/doctor_info".

	You can use this  http request to get the details information of the docotr.
	Including the gender,email,firstname,lastname,ect.

	http-head
	---------
	http-body
	{
		account = "test001"
	}
--]]

-- import the api in the packet in dng_help.lua
local debug = require "dbg_helper"
-- import the api in the packet in comm/redis.lua
local redis = require "comm.redis"
-- import the api in the packet in access_check
local check = require "access_check"
-- import the api in the packet in comm/encrypt
local encrypt = require "comm.encrypt"
-- import the api in the packet in cjson.lua
local json = require("cjson")

-- this is the key use for XOR encode/decode
local mykey = {1,9,8,9,1,0,2}
```

```
-- read the http body form nginx and save it in the stack
ngx.req.read_body()

-- read the http body and save it in to local var
local data = ngx.req.get_body_data()

-- log data to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- use encrypt function to decode the emcrypted data,get the data sent from client, which is serialized in json.
local data = encrypt.encrypt(data,mykey)

-- use json.decode to unserialize the data to get doctor info
local doctor_info = json.decode(data);

-- save the account into local var "acc"
local acc = doctor_info.account

-- create a table with the strings
local keys = {"strType","strGender","strEmail","strLastName","strHospital","strFirstName","strLastName","strSpecialty","strViber","strSkype"}

-- create a connection to redis database
local red = redis:new()

-- create a local var to save result
local result = {}

-- save the account to the strAccount of result
result.strAccount = acc;

-- loop the keys table
for _,_key in pairs (keys) do
        -- use every key in the table to make the key in the redis
        -- for example
        -- acc:test001:strType
        -- acc:test001:strGender
        -- acc:test001:strEmail
        local key = string.format ("acc:%s:%s",acc,_key)

        -- get the data from redis with the key
        local value,err = red:get (key)

        -- save the value get from redis database to result table.
        result[_key]=value
end

-- send result to client
```

```
ngx.say (json.encode(result))
doctor_list.lua
--[[
        This script handles the */api/doctor_list request.
        This request get all the doctor registed in the system.

--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local check = require "access_check"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --

-- this is the key used to decode http-body
local mykey = {1,9,8,9,1,0,2}

-- get the data from nginx and save it in the stack
ngx.req.read_body()

-- create a new connection to redis database
local red = redis:new()

-- check then cookies from client ,make sure the cookies is valid
local ret_check_cookies = check.check_cookies(ngx.var.http_cookie)

-- log the information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!check_cookies!!!!!!!!!!!!!!!!:",ret_check_cookies)

-- get all the doctor account stores int the all:doctor:acc from database
local ok,err = red:smembers ("all:doctor:acc")

-- if the type of "ok" is not a table,return client Doctor list is empty
if type(ok) ~= "table" then
        return ngx.say("DOCTOR_LIST_EMPTY")
end

-- if the err is NOT nil means something gose wrong, during get data from database
if err then
        return ngx.say("DOCTOR_LIST_ERROR")
end

-- create a local var "dl" to store all the doctor information
local dl = {}

-- loop the table "ok",the data in "ok" is the string of doctor account
for _,acc in pairs (ok) do
        --create a local var "d", stores the single doctor infomation
```

```
        local d = {}
        -- save acc to d.account
        d.account = acc

        -- create the key of doctor account
        -- for example : acc:test002
        local redis_acc_key = string.format("acc:%s",acc)

        -- get the gender of single doctor from redis database
        -- for example: acc:test002:strGender
        ok,err = red:get (string.format("%s:strGender",redis_acc_key))

        -- make sure ok is NOT NIL
        assert (ok)

        -- make sure the err is NIL
        assert (not err)

        -- save the gender infomation to d.gender
        d.gender = ok

        -- log the information to file
        ngx.log(ngx.INFO, "!!!!!!!!!!!!!!d!!!!!!!!!!!!!!!:",d.account)
        ngx.log(ngx.INFO, "!!!!!!!!!!!!!!d!!!!!!!!!!!!!!!:",d.gender)

        -- add single doctor information to dl table.
        table.insert (dl,d)
end

-- send the result back to client
ngx.say (json.encode(dl))

download_data.lua
--[[
        This script handles the */api/download_data
        This request will download the data saved in the redis database

        http-head
        */api/download_data
        ----------
        http-body
        {
                key = "upload:data:96"
        }
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local check = require "access_check"
```

```
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --


-- the Key used to decode the data from client
local mykey = {1,9,8,9,1,0,2,6}

-- get the data form nginx,save it in the stack
ngx.req.read_body()

-- get the data from stack then save in local var "data"
local data = ngx.req.get_body_data()

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- decode the data with the key,save the decoded data to local var "data"
local data = encrypt.encrypt(data,mykey)

-- use json.decode unseralize the data,save the data to request_info
local request_info = json.decode(data);


-- create a new connection to redis database
local red = redis:new()

-- get the data form redis database
local download_data = red:get(request_info.key)

-- return the data to client
ngx.say(download_data)

get_user_data.lua
--[[
        This scripts handles */api/get_user_data
        This request will return the user information
        includes
        1. all the requests reference to the account.
        2. all the doctor information.


        http-head
        */api/get_user/data
        -------
        http-body
        {
                account = "test001"
        }
--]]
```

```
-- import function from packet -- start
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end

-- this is the key used for decode from client
local mykey = {1,9,8,9,1,0,2}

-- get data from nginx,save in the stack
ngx.req.read_body()

-- get the data from stack,save in the local var data
local data = ngx.req.get_body_data()

-- log infomation to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- use mykey decode the data from data
local data = encrypt.encrypt(data,mykey)

-- use json.decode  unseralize the data
local request_info = json.decode(data);

-- create the local var "result" to save the result.
local result = {}

-- Save the "GET_USER_DATA_SUCCESS" to result.ret
result.ret = "GET_USER_DATA_SUCCESS"

-- create an connection to redis database
local red = redis:new()

-- get all the request id from the key (acc:test001:requests)
-- which is all the requests created by this account as a patient,or sent to this account as a doctor
result.requests,_ = red:smembers (string.format("acc:%s:requests",request_info.account))

-- get all the doctor accounts from database
local ok,err = red:smembers ("all:doctor:acc")

-- create a local var "dl", to store the docotr information
local dl = {}

-- loop the table "ok"
for _,acc in pairs (ok) do
        -- create the local var "d"
        local d = {}
```

```
        -- save the account to d.account
        d.account = acc

        -- create an local var redis_acc_key to save the key of the single doctor in database (acc:test01)
        local redis_acc_key = string.format("acc:%s",acc)

        -- get the gender of the single doctor
        ok,err = red:get (string.format("%s:strGender",redis_acc_key))

        -- make sure ok is not nil
        assert (ok)
        -- make sure err is nil
        assert (not err)

        -- save "ok" to d.gender which means the gender of doctor
        d.gender = ok

        -- log information to file
        ngx.log(ngx.INFO, "!!!!!!!!!!!!!!d!!!!!!!!!!!!!!!:",d.account)
        ngx.log(ngx.INFO, "!!!!!!!!!!!!!!d!!!!!!!!!!!!!!!:",d.gender)

        -- insert single doctor information to doctor information list
        table.insert (dl,d)
end

-- save the result of doctor list to result.doctors
result.doctors = dl

-- return the result to client
ngx.say(json.encode(result))
```

login.lua

```
--[[
        This script handles the login request
--]]

-- import functino form packet -- start
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
local mykey = {1,9,8,9,1,0,2,6}
-- import functino form packet -- end

-- get data from nginx ,save in stack
ngx.req.read_body()

-- get data from stack ,save in local var "data"
local data = ngx.req.get_body_data()
```

```
-- log informaton to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- decode data with "mykey"
data = encrypt.encrypt(data,mykey)

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- unseralize data with json.decode
local arg = json.decode(data);

-- create local var to save redis keys -- start

local redis_acc_key = string.format("acc:%s",arg.Account)
local redis_passwd_key = string.format("%s:strPassWord",redis_acc_key);
local redis_type_key = string.format("%s:strType",redis_acc_key);

-- create local var to save redis keys -- end


-- create a connection to redis server
local red = redis:new() ;


-- check the redis_acc_key
-- if account already exist in the database  then return error information
local ok,error = red:get(redis_acc_key)
if tonumber(ok) ~= 1 then
        return ngx.say("LOGIN_FAILED_ACCOUNT_ERROR");
end

-- if "error" is not nil means something wrong during get data from redis
if error then
        return ngx.say("LOGIN_FAILED_UNKNOWERROR1")
end


---------------------- CHECK PASSWORD VALID -- start --
-- Get password stores in database
local ok,error = red:get(redis_passwd_key)

-- log infromation to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",ok)

-- log infromation to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",arg.Password)
-- if password from http-body is not equal the password stores in the database, then return error information
if tostring(ok) ~= tostring(arg.Password) then
        return ngx.say("LOGIN_FAILED_PASSWD_ERROR");
```

```
end

-- if "error" is not nil means something wrong during get data from redis
if error then
        return ngx.say("LOGIN_FAILED_UNKNOWERROR2")
end
---------- CHECK PASSWORD VALID -- end --

---------- CHECK ACCOUNT TYPE -- start --
-- Get type stores in the database
local ok,error = red:get(redis_type_key)

-- if type from http-body is not equal the password stores in the database, then return error information
if tostring(ok) ~= tostring(arg.Type) then
        return ngx.say("LOGIN_FAILED_TYPE_ERROR");
end

-- if "error" is not nil means something wrong during get data from redis
if error then
        return ngx.say("LOGIN_FAILED_UNKNOWERROR3")
end
---------- CHECK ACCOUNT TYPE -- end --


---------- HADNLE THE COOKIES -- start --
--import function from resty.md5
local resty_md5 = require "resty.md5"

-- create new md5 local var
local md5 = resty_md5:new()
if not md5 then
        ngx.say("FAILED_TO_CREATE_MD5_OBJECT")
        return
end

-- add redis_acc_key to md5 var
local ok = md5:update(redis_acc_key)
if not ok then
        ngx.say("FAILED_TO_ADD_DATA")
        return
end

-- add timetag to md5 var
ok = md5:update(tostring(os.time()))
if not ok then
        ngx.say("FAILED_TO_ADD_DATA")
        return
end

-- add salt to md5 var
```

```
ok = md5:update("huangxudong")
if not ok then
        ngx.say("FAILED_TO_ADD_DATA")
        return
end

-- get the final MD5 result
local digest = md5:final()

-- import function from resty.string
local str = require "resty.string"

-- get HEX data of the MD5 data
local md_digest = str.to_hex(digest)

-- generate cookies
local cookie_value = ngx.var.cookie_Foo
ngx.header['Set-Cookie'] = string.format("name=%s,value=%s",arg.Account,md_digest)

-- return result to client
ngx.say("LOGIN_SUCESS")

-- create the redis key to store cookies
local redis_cookies_key = string.format("%s:cookies",redis_acc_key)

-- save cookies value into redis database
local ok,err = red:set(redis_cookies_key,md_digest)

-- log information to file
ngx.log(ngx.INFO,"set cookies result",ok)
ngx.log(ngx.INFO,"set cookies result",err)
------------------- HADNLE THE COOKIES -- end --



Make_request.lua
--[[
        This script handles the */api/make_request
        This request can make a request to selected doctor
--]]

-- import function from packet -- start
local debug = require "dbg_helper"
local redis = require "comm.redis"
local check = require "access_check"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end

-- this is the key use to decode data from client
```

```
local mykey = {1,9,8,9,1,0,2,6}

-- get data from nginx, save in the stack
ngx.req.read_body()

-- get data from stack, save in local var
local data = ngx.req.get_body_data()

-- decode the data with mykey
local data = encrypt.encrypt(data,mykey)

-- log information to file
ngx.log(ngx.INFO, "\n----------[json]: read data from http request----------\n",data)

-- unseralize the data
local request_info = json.decode(data);

-- init picture history table
request_info.history_left = {}
table.insert (request_info.history_left,{uid = request_info.uidl,timetag = os.time()})
request_info.history_right = {}
table.insert (request_info.history_right,{uid = request_info.uidr,timetag =os.time()})

-- save false to request_info.is_solved means the request is unsloved
request_info.is_solved = false
-- save the time-tag to request_info,request_timetag
request_info.request_timetag = os.time();


-- save request info
-- create a connection to redis database
local red = redis:new()

-- get uid from redis database
local uid = red:incr ("request:uid")

-- generate the requset_key
local request_key = "request:"..tostring(uid)

-- create the local var "doctor_acc" save the data from request_info.doctor whch get from redis
local doctor_acc = request_info.doctor

-- get the patient account though prase the cookies
local _,_,patient_acc,_ =  string.find (ngx.var.http_cookie,"name=%s*(.+),value=%s*(.+)")

-- save patient_acc to request_info.patient
request_info.patient = patient_acc

-- seralize the request_info to data
data = json.encode(request_info)
```

```
-- log information to file
ngx.log(ngx.INFO, "\n------------[json]: save data to redis-----------------\n",data)

-- save request to database
local ok,err = red:set (request_key,data)


-- save data to doctor and patient
-- log information to file
ngx.log(ngx.INFO,"\n------------[string]: doctor acc in the request -----------------\n",doctor_acc)
-- log information to file
ngx.log(ngx.INFO,"\n------------[string]: patient acc in the request -----------------\n",patient_acc)

-- generate the keys to save requests
local doctor_request_key = string.format ("acc:%s:requests",doctor_acc);
local patient_request_key = string.format ("acc:%s:requests",patient_acc);

-- reference the request with doctor and patient
local ok,error = red:sadd (doctor_request_key,request_key);
local ok,error = red:sadd (patient_request_key,request_key);

-- return result to client
ngx.say("MAKE_REQUEST_SUCESS")

my_requests.lua
--[[
        This script handles */api/my_request
--]]

-- import function from packet -- start
local debug = require "dbg_helper"
local redis = require "comm.redis"
local check = require "access_check"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end

-- this is the keys used to decode data from client
local mykey = {1,9,8,9,1,0,2,6}

-- get data from nginx ,save to stack
ngx.req.read_body()

-- get data from stack, save to local var "data"
local data = ngx.req.get_body_data()
-- get account thouth prase cookies
local _,_,acc,_ =  string.find (ngx.var.http_cookie,"name=%s*(.+),value=%s*(.+)")

-- generate the key to get all requests
```

```
local key = string.format("acc:%s:requests",acc)

-- create a connection to redis database
local red = redis:new()

-- get all the requests id to "data"
local data = red:smembers(key)

-- return the result client
ngx.say(json.encode(data))

register.lua
--[[
        This Script handles */api/register
        This request will register an acoount in the system
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --

-- This is the key used to decode data from client
local mykey = {1,9,8,9,1,0,2}

-- get data from nginx, save to stack
ngx.req.read_body()

-- get data from stack, save to data
local data = ngx.req.get_body_data()

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- decode data with mykey
local data = encrypt.encrypt(data,mykey)

-- unseralize the data
local register_data = json.decode(data);

-- generate the redis key of the account to register
local redis_acc_key = string.format("acc:%s",register_data.strAccount)

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!redis-key!!!!!!!!!!!!!!!!:",redis_acc_key)

-- generate a table "t" with strings
local                                                t                                                =
```

```
{"strSkype","strViber","strSpecialty","strMedNumber","strType","strConfirmPassword","strEMail","strFirstNa
me","strGender","strPassWord","strHospital","strLastName"}

-- generate the table "redis_keys" to store the keys
local redis_keys = {}

-- loop table "t"
for _,v in pairs (t) do
        -- generate the key (test001:strSkype)
        local key = string.format("%s:%s",redis_acc_key,v)
        -- make sure redis_keys[key] is nil
        assert (not redis_keys[key])
        -- save the redis key to the table "redis_keys"
        redis_keys[key] = register_data[v] ;
end

-- create a connection to redis database
local red = redis:new() ;

-- get the data of redis_acc_key
local ok,error = red:get(redis_acc_key)

if not ok and not error then
        -- if enter this path means this account is not registed .
        -- save 1 to redis_acc_key,which means redis_acc_key will be registe
        red:set(redis_acc_key,1)

        -- loop the redis_keys
        for k,v in pairs (redis_keys) do
                -- get the data from redis database
                local ok,error = red:get(k)
                -- check unkonw error -- start
                if error then
                        return ngx.say(encrypt.encrypt("REG_UNKNOW_ERROR2",mykey))
                end

                if ok then
                        return ngx.say(encrypt.encrypt("REG_UNKNOW_ERROR3",mykey))
                end
                -- check unkonw error -- end

                -- save the register information to the redis database
                red:set(k,v)
        end
        --add doctor information to redis set
        if register_data.strType == "Doctor" then
                local ok,error = red:sadd ("all:doctor:acc",register_data.strAccount)
                if not ok or err then
                        return ngx.say(encrypt.encrypt("REG_UNKNOW_ERROR3",mykey))
                end
```

```
        end
        return ngx.say(encrypt.encrypt("REG_SUCESS",mykey))
elseif tonumber(ok) == 1 and not error then
        -- if "ok" equals 1 means the account was already registed
        return ngx.say(encrypt.encrypt("REG_FAILED_ALREADY_REGISTED",mykey))
else
        -- if enter this path means unkonw error happened
        return ngx.say(encrypt.encrypt("REG_UNKNOW_ERROR",mykey))
end
```

request_add_comment.lua

```
-- NO USE !!!!!!!!!!!!!!
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
local mykey = {1,9,8,9,1,0,2}

ngx.req.read_body()
local data = ngx.req.get_body_data()
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!!:",data)
local data = encrypt.encrypt(data,mykey)
local request_info = json.decode(data);
debug.printArgs(request_info)
```

request_info.lua

```
--[[
        This script handles  */api/request_info
        This request get the information of single request
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --

-- this is the key use to decode data from client
local mykey = {1,9,8,9,1,0,2}

-- get data from nginx,save to  stack
ngx.req.read_body()

-- get data from stack,save to local var "data"
local data = ngx.req.get_body_data()

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!!:",data)
```

```
-- decode data with the key
local data = encrypt.encrypt(data,mykey)

-- unseralize the data
local request_info = json.decode(data);

-- create a connection to redis database
local red = redis:new()

-- get requst data from redis database
local request_info_json = red:get(request_info.request)

-- log information to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!request_info_json!!!!!!!!!!!!!!!!:",request_info_json)

-- seralize data to local var "request_info_table"
local request_info_table = json.decode(request_info_json)
--debug.printArgs(request_info_table)

-- save "REQUEST_INFO_SUCESS" to request_info_table.request
request_info_table.result="REQUEST_INFO_SUCCESS"

-- send result back to client
ngx.say(json.encode(request_info_table))


request_update_photo.lua
--[[
        This script handles */api/request_update_photo
        This request is used for update the photo in the request
--]]

-- import function from packes -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packes -- end --

-- this is the key used to decode data from client
local mykey = {1,9,8,9,1,0,2}

-- get data from nginx , save to stack
ngx.req.read_body()
-- get data from stack, save to local var "data"
local data = ngx.req.get_body_data()

-- decode data with mykey
local data = encrypt.encrypt(data,mykey)
```

```
-- log infromation to file
ngx.log(ngx.INFO, "\n------------------[update photo]----------------\n",data)

--  unseralize the data
local request_info = json.decode(data);

-- create a connection to databse
local red = redis:new()

-- get detal of request from redis
local req_detail = red:get(request_info.request_id)

-- log infromation to file
ngx.log(ngx.INFO, "\n------------------[update photo]:detail in redis----------------\n",req_detail)

-- unseralize data
req_detail = json.decode(req_detail);


-- generate the photo history information
if tonumber(request_info.lr) == 1 then
        req_detail.uidl = request_info.photo_uid
        table.insert(req_detail.history_left,{uid=request_info.photo_uid,timetag=os.time()})
else
        req_detail.uidr = request_info.photo_uid
        table.insert(req_detail.history_right,{uid=request_info.photo_uid,timetag=os.time()})
end

-- seralized data to send back to client
req_detail = json.encode(req_detail);

--log information to file
ngx.log(ngx.INFO, "\n------------------[update photo]:detail changed----------------\n",req_detail)

--save modifed data to database
red:set (request_info.request_id,req_detail)

-- sned result to client
ngx.say("UPDATE_PHOTO_SUCCESS")
```

slove_request.lua

```
--[[
        This script handles */api/slove_request
        This request will close the request , doctor and patient will not able to see the request.
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
```

```
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --

-- this is the key used to decode data from client
local mykey = {1,9,8,9,1,0,2}

-- log information to file
ngx.log(ngx.INFO, "\n----------------------slove request start-------------------\n")

-- get data from nginx , save to stack
ngx.req.read_body()

-- get data from stack, save to local var data
local data = ngx.req.get_body_data()

-- decode data with mykey
local data = encrypt.encrypt(data,mykey)

-- log infroamtion to file
ngx.log(ngx.INFO, "\n----------------------[slove_request]-------------------\n",data)

-- unsearlize data
local request_info = json.decode(data);

-- create a connection
local red = redis:new()

-- generate the local var to save data in request_info -- start
local doctor_acc = request_info.doctor
local patient_acc = request_info.patient
local request_key = request_info.requestID

local doctor_sloved_request_key = string.format ("acc:%s:sloved_requests",doctor_acc);
local patient_sloved_request_key = string.format ("acc:%s:sloved_requests",patient_acc);

local doctor_request_key = string.format ("acc:%s:requests",doctor_acc);
local patient_request_key = string.format ("acc:%s:requests",patient_acc);
-- generate the local var to save data in request_info -- end

ngx.log(ngx.INFO, "\n----------------------[slove_request]:keys-------------------\n",doctor_sloved_request_key)
ngx.log(ngx.INFO, "\n----------------------[slove_request]:keys-------------------\n",patient_sloved_request_key)
ngx.log(ngx.INFO, "\n----------------------[slove_request]:keys-------------------\n",doctor_request_key)
ngx.log(ngx.INFO, "\n----------------------[slove_request]:keys-------------------\n",patient_request_key)

--remove from acc:*:requests
ngx.log(ngx.INFO,                                    "\n----------------------[slove_request]:result--------------------
\n"..doctor_request_key.."\n"..request_key)
local ok,error = red:srem (doctor_request_key,request_key)
ngx.log(ngx.INFO, "\n----------------------[slove_request]:result-------------------\n",ok,error)
```

```
local ok,error = red:srem (patient_request_key,request_key)
ngx.log(ngx.INFO, "\n----------[slove_request]:result----------\n",ok,error)

--add to acc:*:sloved_requests
local ok,error = red:sadd (doctor_sloved_request_key,request_key);
ngx.log(ngx.INFO, "\n----------[slove_request]:result----------\n",ok,error)
local ok,error = red:sadd (patient_sloved_request_key,request_key);
ngx.log(ngx.INFO, "\n----------[slove_request]:result----------\n",ok,error)


--send result to client
ngx.say("SLOVE_REQUEST_SUCCESS")


update_comment.lua
--[[
        This script handles the */api/update_comment
        This request will update the comment in single comment
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local redis = require "comm.redis"
local encrypt = require "comm.encrypt"
local json = require("cjson")
-- import function from packet -- end --

-- this is the key to decode data from client
local mykey = {1,9,8,9,1,0,2}

-- get data from nginx, save to stack
ngx.req.read_body()

-- get data from stack, save to local var "data"
local data = ngx.req.get_body_data()

-- log infroamtion to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!data!!!!!!!!!!!!!!!!:",data)

-- decode data with mykey
local data = encrypt.encrypt(data,mykey)

-- unseralize data
local request_info = json.decode(data);
-- connection to redis database
local red = redis:new()

-- get detail information from redis database
local request_info_json = red:get(request_info.requestID)
```

```
-- log infroamtion to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!request_info_json!!!!!!!!!!!!!!!!:",request_info_json)

-- unseralize data from redis
local request_info_table = json.decode(request_info_json)

-- update the comment string
request_info_table.comment = request_info_table.comment .. request_info.addComment .. "\n"

-- seralize the data
request_info_json = json.encode(request_info_table)

-- log infroamtion to file
ngx.log(ngx.INFO, "!!!!!!!!!!!!!!request_info_json_2!!!!!!!!!!!!!!!!:",request_info_json)

-- save modified data to redis database
red:set (request_info.requestID,request_info_json)

-- send data back to client
ngx.say("UPDATE_SUCESS")

upload_file.lua
--[[
        This script handles */api/upload_data
        This request update the data to server
--]]

-- import function from packet -- start --
local debug = require "dbg_helper"
local json = require "cjson"
local redis = require "comm.redis"
-- import function from packet -- end --


-- create a connection to redis server
local red = redis:new() ;

-- log information to file
ngx.log(ngx.INFO, "Enter upload_file")

-- this function can read data from a tmp file in the disk
local function getFile(file_name)
        -- open file
        local f = assert(io.open(file_name, 'r'))
        -- read data
        local string = f:read("*all")
        -- close file
        f:close()
        -- return the data read from disk
        return string
```

```
end

-- get args http head from nginx,save to stack
local arg = ngx.req.get_uri_args()
debug.printArgs(arg)

-- get headers from stack
local headers = ngx.req.get_headers()


-- log information to file
ngx.log(ngx.INFO, "start ReadBody")

-- get data from nginx
ngx.req.read_body()

-- log information to file
ngx.log(ngx.INFO, "finish ReadBody")

-- get data from stack,save to local var "data"
local data = ngx.req.get_body_data()

-- if data exist
if nil == data then
        -- get file name
        local file_name = ngx.req.get_body_file()
        -- log information to file
        ngx.log(ngx.INFO,">> temp file: ", file_name)
        -- if file_name exist
        if file_name then
                -- read data from disk
                data = getFile(file_name)
        end
end

-- generate uid
local data_index,err = red:incr("upload:data:uid")

-- save data to redis database
local ok,err = red:set ("upload:data:"..tostring(data_index),data)

-- generate result table,then update reuslt
local result = {}
result.result="UPLOAD_SUCESS"
result.uid = tostring(data_index)

-- send result back to client
ngx.say(json.encode(result))
```

BIBLIOGRAPHY

[1] Aramudhan, M., and K. Mohan. "New secure communication protocols for mobile e-health system." In *International Conference on Networked Digital Technologies*, pp. 639-647. Springer, Berlin, Heidelberg, 2010.

[2] Shahriyar, Rifat, Md Faizul Bari, Gourab Kundu, Sheikh Iqbal Ahamed, and Md Mostofa Akbar. "Intelligent mobile health monitoring system (IMHMS)." In *International Conference on Electronic Healthcare*, pp. 5-12. Springer, Berlin, Heidelberg, 2009.

[3] Elkhodr, Mahmoud, Seyed Shahrestani, and Hon Cheung. "Enhancing the security of mobile health monitoring systems through trust negotiations." In *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, pp. 754-757. IEEE, 2011.

[4] Barnickel, Johannes, Hakan Karahan, and Ulrike Meyer. "Security and privacy for mobile electronic health monitoring and recording systems." In *World of Wireless Mobile and Multimedia Networks (Wow Mom), 2010 IEEE International Symposium on a*, pp. 1-6. IEEE, 2010.

[5] Rahman, Mahmudur, Bogdan Carbunar, and Madhusudan Banik. "Fit and vulnerable: Attacks and defenses for a health monitoring device." *arXiv preprint arXiv: 1304.5672* (2013).

[6] Al-Nayadi, Fahed, and Jemal H. Abawajy. "An authorization policy management framework for dynamic medical data sharing." In *Intelligent Pervasive Computing, 2007. IPC. The 2007 International Conference on*, pp. 313-318. IEEE, 2007.

[7] Baskar, Praveen, J. Gitanjali, A. Vamsi Krishna, J. Indumathi, and G. V. Uma. "Annotation of Security and Privacy Requirements for E-Health Applications by the Design and Development of Security Ontology."

[8] Mohan, and M. Aramudhan "Secure and auditable agent-based communication protocol for e-health system framework." *Oriental Journal of Computer Science and Technology* 3, no. 2 (2010).

[9] Leister, Wolfgang, Mohamed Hamdi, Habtamu Abie, and Stefan Poslad. "An evaluation scenario for adaptive security in eHealth." In *Proceedings of Fourth International Conference on Performance, Safety and Robustness in Complex Systems and Applications, Nice, France*, vol. 2327. 2014.

[10] Löhr, Hans, Ahmad-Reza Sadeghi, and Marcel Winandy. "Securing the e-health cloud." In *Proceedings of the 1st ACM International Health Informatics Symposium*, pp. 220-229. ACM, 2010.

[11] AbuKhousa, Eman, Nader Mohamed, and Jameela Al-Jaroodi. "e-Health cloud: opportunities and challenges." *Future Internet*4, no. 3 (2012): 621-645.

[12] Shetty, R., and G. S. Thyagaraju. "Design and Development of Mobile Phone Based Healthcare System for Emergency Situation." *Int. J. Comput. Trends Technol. (IJCTT)* 12, no. 3 (2014): 119-122.

[13] Varshney, Upkar. "Pervasive healthcare and wireless health monitoring." *Mobile Networks and Applications* 12, no. 2-3 (2007): 113-127.

[14] Toninelli, Alessandra, Rebecca Montanari, and Antonio Corradi. "Enabling secure service discovery in mobile healthcare enterprise networks." *IEEE Wireless Communications* 16, no. 3 (2009).

[15] Conejar, Regin Joy, and Haeng-Kon Kim. "A Design of Mobile Convergence Architecture for U-healthcare." *International Journal of Software Engineering and Its Applications* 9, no. 1 (2015): 253-260.

[16] Aslantas, Veysel, Rifat Kurban, and Tuba Caglikantar. "A low-cost wireless healthcare monitoring system and communication to a clinical alarm station." In *5th International Conference on Electrical and Electronics Engineering*. 2007.

[17] She, Huimin, Zhonghai Lu, Axel Jantsch, Li-Rong Zheng, and Dian Zhou. "A network-based system architecture for remote medical applications." In *Network Research Workshop*, vol. 27. 2007.

[18] Aminian, M., and H. Reza Naji. "A hospital healthcare monitoring system using wireless sensor networks." *J. Health Med. Inform* 4, no. 02 (2013): 121.

[19] Hamida, Sana Tmar-Ben, Elyes Ben Hamida, and Beena Ahmed. "A new mHealth communication framework for use in wearable WBANs and mobile technologies." *Sensors* 15, no. 2 (2015): 3379-3408.

[20] Khairallah, Hussein, and Lubna Alazzawi. "Architectural Design Solutions for RHMS."

[21] Kamarudin, Nazhatul Hafizah, Y. Mohd Yussoff, and H. A. B. I. B. A. H. Hashim. "Two-tier e-health monitoring system." *WSEAS Applied Computational Science (ACACOS), Kuala Lumpur, Malaysia* (2014).

[22] Shahriyar, Rifat, Md Faizul Bari, Gourab Kundu, Sheikh Iqbal Ahamed, and Md Mostofa Akbar. "Intelligent mobile health monitoring system (IMHMS)." In *International Conference on Electronic Healthcare*, pp. 5-12. Springer, Berlin, Heidelberg, 2009.

[23] Ali, Khaled A., and Hussein T. Mouftah. "Wireless personal area networks architecture and protocols for multimedia applications." *Ad Hoc Networks* 9, no. 4 (2011): 675-686.

[24] Vouyioukas, Demosthenes, and Alexandros Karagiannis. "Pervasive homecare monitoring technologies and applications." In *Telemedicine Techniques and Applications*. InTech, 2011.

[25] Godara, Balwant, and Konstantina S. Nikita. "Wireless mobile communication and healthcare." In *Third International Conference, MobiHealth*, pp. 21-23. 2012.

[26] Abidoye, Ademola P., Nureni A. Azeez, Ademola Olusola Adesina, and Kehinde K. Agbele. "Using wearable sensors for RHMS." (2011).

[27] Jllt, Noida. "Wearable Sensors for RHMS."

[28] Custodio, Víctor, Francisco J. Herrera, Gregorio López, and José Ignacio Moreno. "A review on architectures and communications technologies for wearable health-monitoring systems." *Sensors* 12, no. 10 (2012): 13907-13946.

[29] Johnson, Cathy. "Nurses and the use of personal digital assistants (PDAs) at the point of care." (2008).

[30] Lane, Nicholas D., Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T. Campbell. "A survey of mobile phone sensing." *IEEE Communications magazine* 48, no. 9 (2010).

[31] Al-Taee, Majid A., Nadine A. Jaradat, and Dima M. Abu Ali. "Mobile phone-based health data acquisition system using Bluetooth technology." In *Applied Electrical Engineering and Computing Technologies (AEECT), 2011 IEEE Jordan Conference on*, pp. 1-6. IEEE, 2011.

[32] Shyamkumar, Prashanth, Pratyush Rai, Sechang Oh, Mouli Ramasamy, Robert E. Harbaugh, and Vijay Varadan. "Wearable wireless cardiovascular monitoring using textile-based nanosensor and nanomaterial systems." *Electronics* 3, no. 3 (2014): 504-520.

[33] Danaher, Brian G., Håvar Brendryen, John R. Seeley, Milagra S. Tyler, and Tim Woolley. "From black box to toolbox: outlining device functionality, engagement

activities, and the pervasive information architecture of mHealth interventions." *Internet interventions* 2, no. 1 (2015): 91-101.

[34] Kumar, Pardeep, and Hoon-Jae Lee. "Security issues in healthcare applications using wireless medical sensor networks: A survey." *Sensors* 12, no. 1 (2011): 55-91.

[35] Paschou, Mersini, Nikolaos Nodarakis, Athanasios Tsakalidis, and Evangelos Sakkopoulos. "Mobile healthcare systems: Generating dynamic smartphone apps to serve multiple medical specializations." *Future* 10, no. 2 (2013).

[36] Ozdalga, Errol, Ark Ozdalga, and Neera Ahuja. "The smartphone in medicine: a review of current and potential use among physicians and students." *Journal of medical Internet research* 14, no. 5 (2012).

[37] Cha, Bonnie. "House Call: Five Smartphone Accessories That Help Monitor Your Health." *All Things Digital* (2014).

[38] Christina Hernandez Sherwood, "Health app guides patients from symptoms to treatment," 2011. [Online]. Available: https://www.zdnet.com/article/health-app-guides-patients-from-symptoms-to-treatment/.

[39] Stankevich, Evgeny, Ilya Paramonov, and Ivan Timofeev. "Mobile phone sensors in health applications." In *Proc. 12th Conf. of Open Innovations Association FRUCT and Seminar on e-Tourism*, pp. 136-141. 2012.

[40] Nick, T. "Did you know how many different kinds of sensors go inside a smartphone?" (2014).

[41] Rama Krishna Raju, "How Many Different Kinds of Sensors is available Inside a Smartphone?" 2015. [Online]. Available: https://www.quora.com/how-many-different-sensors-are-available-inside-a-smartphone.

[42] Tim Schiesser, “Know your Smartphone: A guide to Camera Hardware,” 2014. [Online]. Available: https://www.techspot.com/guides/850-smartphone-camera-hardware/.

[43] Terry Relph, “Smartphone camera performance: What does the sensor's megapixel count really tell you,” 2015. [Online]. Available: www.zdnet.com/.../smartphone-camera-performance-what-does-the-sensors-megapixe.

[44] Lu, Hong, Wei Pan, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. "Sound Sense: scalable sound sensing for people-centric applications on mobile phones." In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pp. 165-178. ACM, 2009.

[45] Ketabdar, Hamed, Kamer Ali Yüksel, and Mehran Roshandel. "MagiTact: interaction with mobile devices based on compass (magnetic) sensor." In *Proceedings of the 15th international conference on Intelligent user interfaces*, pp. 413-414. ACM, 2010.

[46] Statland, Jeffrey M., Yunxia Wang, Rachel Richesson, Brian Bundy, Laura Herbelin, Joe Gomes, Jaya Trivedi et al. "An interactive voice response diary for patients with non-dystrophic myotonia." *Muscle & nerve* 44, no. 1 (2011): 30-35.

[47] Raso, Iván, Ramón Hervás, and José Bravo. "M-Physio: personalized accelerometer-based physical rehabilitation platform." In *Proceedings of the Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pp. 416-421. 2010.

[48] Bravo, José, F. J. Navarro, Jesus Fontecha, and Ramon Hervas. "A mobile proposal for frailty monitoring by rehabilitation and physical daily activity." In *Consumer Electronics-Berlin (ICCE-Berlin), 2011 IEEE International Conference on*, pp. 176-180. IEEE, 2011.

[49] Pandey, Ambarish, Sayeedul Hasan, Divyanshu Dubey, and Sasmit Sarangi. "Smartphone apps as a source of cancer information: changing trends in health information-seeking behavior." *Journal of Cancer Education* 28, no. 1 (2013): 138-142.

[50] Jang, Dasom, Soo-Yong Shin, Dong-Woo Seo, Segyeong Joo, and Soo-Jin Huh. "A smartphone-based system for the automated management of point-of-care test results in hospitals." *Telemedicine and e-Health* 21, no. 4 (2015): 301-305.

[51] Park, Yu Rang, Yura Lee, Guna Lee, Jae Ho Lee, and Soo-Yong Shin. "Smartphone applications with sensors used in a tertiary hospital—current status and future challenges." *Sensors* 15, no. 5 (2015): 9854-9869.

[52] Melissa Conrad Stöppler, "Vision Loss Symptoms & Signs," 2012. [Online] Available: https://www.medicinenet.com/vision_loss/symptoms.htm.

[53] Moradian, Siamak, and Sare Safi. "Application of mobile phones in ophthalmology." *Journal of ophthalmic & vision research* 10, no. 2 (2015): 200.

[54] Chhablani, Jay, Simon Kaja, and Vinay A. Shah. "Smartphones in ophthalmology." *Indian journal of ophthalmology* 60, no. 2 (2012): 127.

[55] Lord, Ron K., Vinay A. Shah, Ashley N. San Filippo, and Rohit Krishna. "Novel uses of smartphones in ophthalmology." *Ophthalmology* 117, no. 6 (2010): 1274-1274.

[56] Davis, Elizabeth A., John A. Hovanesian, James A. Katz, Manus C. Kraff, and William B. Trattler."Professional life and the smartphone." *Cataract Refract Surg Today* (2010): 21-2.

[57] Zvornicanin, Edita, Jasmin Zvornicanin, and Bahrudin Hadziefendic. "The use of smart phones in ophthalmology." *Acta Informatica Medica* 22, no. 3 (2014): 206.

[58] Bastawrous, A., R. C. Cheeseman, and A. Kumar. "iPhones for eye surgeons." (2012): 343.

[59] Gillingham W, Holt A, Gillies J. Hand-held, “Computers in healthcare: what software programs are available?” 2002. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/12386664.

[60] Mohammadpour, Mehrdad, Zahra Heidari, Masoud Mirghorbani, and Hassan Hashemi. "Smartphones, tele-ophthalmology, and VISION 2020." *International journal of ophthalmology* 10, no. 12 (2017): 1909.

[61] Chakrabarti, Rahul, and Chandra Shan Perera. "Smartphone medical imaging: applications and future considerations." *Journal of Mobile Technology in Medicine* 3, no.1 (2014): 1-1.

[62] Silva, Paolo S., Jerry D. Cavallerano, Lloyd M. Aiello, and Lloyd Paul Aiello. "Telemedicine and diabetic retinopathy: moving beyond retinal screening." *Archives of ophthalmology*129, no. 2 (2011): 236-242.

[63] Russo, Andrea, Francesco Morescalchi, Ciro Costagliola, Luisa Delcassi, and Francesco Semeraro. "A novel device to exploit the smartphone camera for fundus photography." *Journal of ophthalmology* 2015 (2015).

[64] Ting, Daniel SW, Mei Ling Tay- Kearney, and Yogesan Kanagasingam. "Light and portable novel device for diabetic retinopathy screening." *Clinical & experimental ophthalmology*40, no. 1 (2012).

[65] Haddock, Luis J., David Y. Kim, and Shizuo Mukai. "Simple, inexpensive technique for high-quality smartphone fundus photography in human and animal eyes." *Journal of ophthalmology* 2013 (2013).

[66] Vyas, Hitesh J. "Smartphone Ophthalmic Surgery recording for e-records and education." *Journal of Education Technology in Health Sciences* 3, no. 1 (2016): 13-15.

[67] Ludwig, Cassie A., Somasheila I. Murthy, Rajeev R. Pappuru, Alexandre Jais, David J. Myung, and Robert T. Chang. "A novel smartphone ophthalmic imaging adapter: user feasibility studies in Hyderabad, India." *Indian journal of ophthalmology*64, no. 3 (2016): 191.

[68] Kambiz Negahban, "Diabetic & Hypertensive Retinopathy," 2009. [Online]. Available: http://www.eastoneye.com/diabetic-hypertensive-retinopathy.aspx.

[69] Jesse Vislisel and Thomas Oetting, "Diabetic retinopathy: from one medical student to another," 2010. [Online]. Available: http://www.newhealthguide.org/Pdr.html.

[70] Madeline R. Vann, MPH, "Understanding Diabetic Retinopathy," 2015. [Online]. Available:https://www.everydayhealth.com/hs/diabetes-eye-health/understanding-diabetic-retinopathy/.

[71] Kierstan Boyd, "What Is Diabetic Retinopathy?" 2017. [Online]. Available: https://www.aao.org/eye-health/diseases/what-is-diabetic-retinopathy.

[72] Klein, Ronald, Barbara EK Klein, Scot E. Moss, Matthew D. Davis, and David L. DeMets. "The Wisconsin Epidemiologic Study of Diabetic Retinopathy: III. Prevalence and risk of diabetic retinopathy when age at diagnosis is 30 or more years." *Archives of ophthalmology* 102, no. 4 (1984): 527-532.

[73] Marilyn Haddrill, "Treatment of Diabetic Retinopathy, and Macular Edema," 2017. [Online]. Available: http://www.allaboutvision.com/conditions/diabetic-treatment.htm.

[74] Wilkinson, C. P., Frederick L. Ferris, Ronald E. Klein, Paul P. Lee, Carl David Agardh, Matthew Davis, Diana Dills, Anselm Kampik, R. Pararajasegaram, and Juan T.

Verdaguer. "Proposed international clinical diabetic retinopathy and diabetic macular edema disease severity scales." *Ophthalmology* 110, no. 9 (2003): 1677-1682.

[75] National Eye Institute (NEI), "Facts about Cataract", 2009. [Online]. Available: https://nei.nih.gov/health/cataract/cataract_facts.

[76] Gretchyn Bailey, Vance Thompson, "Cataracts: Common Types, Causes, and Symptoms," 2016. [Online] Available: http://www.ncascade.com/services/cataract-surgery/.

[77] Joseph Bennington-Castro, Robert Jasmer, "What Are Cataracts?" 2016. [Online]. Available: http://www.specstore.org/eye-health/cataracts?next=&id=3.

[78] Andrew A. Dahl, "Macular degeneration facts", 2012. [Online]. Available: https://www.medicinenet.com/macular_degeneration/article.htm#macular_degeneration facts.

[79] Jager, Rama D., William F. Mieler, and Joan W. Miller. "Age-related macular degeneration." *New England Journal of Medicine* 358, no. 24 (2008): 2606-2617.

[80] Maureen Salamon, "Macular Degeneration: Symptoms, Diagnosis and Treatments," 2015. [Online]. Available: https://www.livescience.com/34781-macular-degeneration-eye-disease.html.

[81] Lylas G. Mogk, "What is Age-Related Macular Degeneration," 2015. [Online]. Available: http://healthlifemedia.com/healthy/what-is-age-related-macular-degeneration/.

[82] Marilyn Haddrill "What Is Age-Related Macular Degeneration?" 2018. [Online]. Available: http://www.allaboutvision.com/conditions/amd.htm.

[83] Vic Khemsara, "Glaucoma: The Silent Disease," 2016. [Online]. Available: https://www.forsythwoman.com/summit-eye-care-glaucoma-the-silent-disease/.

[84] Andrew A. Dahl, “What are glaucoma symptoms and signs?” 2012. [Online]. Available: https://www.medicinenet.com/glaucoma/article.htm#glaucoma_facts.

[85] Judy Torres, “Learn the Signs & Symptoms of Acute Angle Closure Glaucoma,” 2013. [Online]. Available: http://www.jems.com/articles/print/volume-38/issue-10/features/learn-signs-symptoms-acute-angle-closure.html?c=1.

[86] Marilyn Haddrill, Charles Slonim, “Glaucoma Treatment: Eye Drops and Other Medications,” 2016. [Online]. Available: http://www.allaboutvision.com/conditions/glaucoma-3-treatment.htm.

[87] Chhablani, Jay, Simon Kaja, and Vinay A. Shah. "Smartphones in ophthalmology." *Indian journal of ophthalmology* 60, no. 2 (2012): 127.

[88] Jill Goetz, “UA Engineering Vision Researcher Turns Smartphones into Eye-Screening Tools,” 2014. [Online]. Available: https://uanews.arizona.edu/story/ua-engineering-turns-smartphones-into-eye-screening-tools.

[89] Moradian, Siamak, and Sare Safi. "Application of mobile phones in ophthalmology." *Journal of ophthalmic & vision research* 10, no. 2 (2015): 200.

[90] Zvornicanin, Edita, Jasmin Zvornicanin, and Bahrudin Hadziefendic. "The use of smart phones in ophthalmology." *Acta Informatica Medica* 22, no. 3 (2014): 206.

[91] Lakshminarayanan, Vasudevan, John Zelek, and Annette McBride. " Smartphone Science" in Eye Care and Medicine." *Optics and Photonics News* 26, no. 1 (2015): 44-51.

[92] Roy, Somak, Liron Pantanowitz, Milon Amin, Raja R. Seethala, Ahmed Ishtiaque, Samuel A. Yousem, Anil V. Parwani, Ioan Cucoranu, and Douglas J. Hartman. "Smartphone adapters for digital photomicrography." *Journal of pathology informatics* 5 (2014).

[93] Giardini, Mario E., Iain AT Livingstone, Stewart Jordan, Nigel M. Bolster, Tunde Peto, Matthew Burton, and Andrew Bastawrous. "A smartphone based ophthalmoscope." In *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, pp. 2177-2180. IEEE, 2014.

[94] Maryna Koberidze, "Eye Exams On-the-Go with PEEK," 2017. [Online]. Available: https://medium.com/innovate4health/eye-exams-on-the-go-with-peek-7354dbc7a6a

[95] Giardini, Mario Ettore. "The Portable Eye Examination Kit: Mobile phones can screen for eye disease in low-resource settings." *IEEE pulse* 6, no. 6 (2015): 15-17.

[96] Robert Chang and David Myung, "Advances in Smartphone Photography of the Eye," 2014. [Online]. Available: http://crstodayeurope.com/wp-content/themes/crste/assets/downloads/0914CRSTEuro_BF3.pdf.

[97] Ben Coxworth, "EyeGo adapters let you perform eye exams with a smartphone," 2014. [Online]. Available: https://newatlas.com/eyego-smartphone-eye-exam/31166/.

[98] David Nield, "A new smartphone attachment can perform eye exams from anywhere in the world," 2015. [Online]. Available: https://www.sciencealert.com/a-new-smartphone-attachment-that-could-save-you-a-trip-to-the-eye-doctor.

[99] Haddock, Luis J, "Smartphone Technology for Fundus Photography," June 1, 2015. [Online]. Available: https://www.retinalphysician.com/issues/2015/june-2015/smartphone-technology-for-fundus-photography.

ABSTRACT

REMOTE SCREENING AND SELF-MONITORING FOR VISION LOSS DISEASES BASED ON SMARTPHONE APPLICATIONS

by

HUSSEIN M. H. KHAIRALLAH

August 2018

Advisor: Dr. Nabil Sarhan

Co-Advisor: Dr. Lubna Alazzawi

Major: Computer Engineering

Degree: Doctor of Philosophy

Remote Healthcare Monitoring Systems (RHMS) enable remote observing of patients' well-being and provide therapeutic services. Sensors play an essential part in RHMS. They measure the physical parameters and give continuous information to health organizations and doctors. Smartphones have allowed us to use RHMS for a variety of tasks to be completed in optimum time. In this dissertation, I provided a meaningful utilization comparison between three tiers operating in the Remote Healthcare Monitoring System (HRMS) architecture design: the Body Area Network (BAN), the PAN Coordinator and the Back- Medical End System (BMEsys). This system focuses on several patients' PAN coordinators, including Wireless Sensor Network (WSN), and Personal Digital Assistant (PDA), and smartphones.

This dissertation demonstrates that smartphones can be used for medical treatment in the field of ophthalmology and discusses how different technologies could be used to diagnose loss of vision. Most recent smartphones have been equipped with a featured camera with high megapixels, and advanced sensors can be used to take fundus photographs by a slit lamp, record videos from an operating microscope, and display images from optical coherence tomography

systems and other high-tech devices. Ophthalmologists can share and analyze these images with their colleagues using media sharing applications, thereby ensuring the optimal diagnostic and therapeutic results for patients with low vision. At present, three widely used pocket-sized adapters can improve the magnification and lighting of the camera, which enables the smartphones to capture high-quality images of the eye. These are the Portable Eye Examination Kit (PEEK), EyeGo, and D-Eye. I provided a meaningful utilization comparison between these three adapters. Finally, I have developed an RHMS app to help patients suffering from loss of vision. The app can facilitate the exchange of information between patients and doctors with a high degree of security and privacy. I completed the development of the mobile app including the Skype and Viber links, which can help in exchanging the information between the patient and the doctor.

AUTOBIOGRAPHICAL STATEMENT

HUSSEIN M. H. KHAIRALLAH

EDUCATION

Hussein Khairallah received a Bachelor of Science in Computer of Science from Warsaw University of Technology, Warsaw-Poland and a Master of Science in Computer of Science from University of Technology, Warsaw- Poland. At this time, Khairallah is working toward the Doctor of Philosophy degree in Department of Electrical and Computer Engineering at Wayne State University, Michigan, USA.

PUBLICATIONS

1- Hussein Khairallah, Lubna Alazzawi. "Intelligent E-Health Monitoring Systems Architecture," IEEE SEM Humanitarian Technology Conference, July 17, 2015.

2- Hussein M. H. Khairallah, Lubna Alazzawi. "Architecture Solutions Design for RHMS," International Journal of Engineering Sciences & Emerging Technologies Volume 8, Issue 4, pp: 177-190, Jan. 2016.

3- Hussein Khairallah, Lubna Alazzawi. "Smartphone Ophthalmic Adapters for Eye Health Imaging Diagnostics," IEEE SEM Humanitarian Technology Conference, July 9, 2016.

4- Hussein Khairallah, Lubna Alazzawi, Nabil Sarhan. "Remote Screening and Self-Monitoring for Vision Loss Diseases," Book Chapter, "Histopathological Image Analysis in Medical Decision Making," IGI Global "Book Chapter" Submission System, Editorial Discovery, Mar.14, 2018.