8-2015

Dynamic Profile Based Access Control in Health Care Systems

Koushik Chilukuri
St. Cloud State University

Dynamic Profile Based Access Control in Health Care Systems

by

Koushik Chilukuri

A Thesis

Submitted to the Graduate Faculty

of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science in

Information Assurance

July, 2015

Dennis Guster, Chairperson
Jim Chen
Balsy Kasi

Abstract

The growing concerns for patient privacy, maintaining recordkeeping integrity and ensuring confidentiality have all significantly increased in health care. There is more attention than ever before given to health care systems that store very sensitive personal information for millions of individuals. As it is, information security professionals in the health care industry must carefully balance the fine line that exists between providing medical staff the critical access to health records they need to care for patients while at the same time protecting against malicious acts or unintentional misuse originating from people having inappropriate access to data. The following proposed conceptual model would provide the health care industry a solution to this problem by allowing medical professionals access to only the pertinent data needed to perform a given task without compromising patient care. Additionally, the privacy and confidentiality of patient records are greatly enhanced by this model, which in turn increases regulatory compliance and industry innovation. This proposed concept model is also a perfect blend of role-based access control and process based access control mechanisms. Numerous hours of research and testing of this proposed concept model have revealed significant promise of success by clearly limiting access of information to only authorized individuals.

The enormous depth of knowledge that it takes for an IT professional to fully understand the intricacies of healthcare systems is often overlooked. However, in order to truly secure these types of systems, developers in particular need to achieve greater sophistication with the software code that operates within these systems especially when it comes to access controls. At the same time, funding for the healthcare industry is often a wavering challenge so this proposed conceptual model also seeks to leverage existing role models without the expensive overhead of a costly and extravagant third-party solution. It goes without saying that patients being admitted into a hospital are often in serious health situations and that presents a unique information security challenge because in no way should technology interfere in the welfare of an individual. Consequently, implementing access controls must not contradict with the necessary treatment from medical professionals. This proposed concept model will enable the necessary staff to see all data, but only when provided with a reason and this reason will be forwarded to the patient, making it hard to unnecessary information. Furthermore, the proposed conceptual model is smart enough to know what information is relevant and what is not.

This model can also be used as a proof of concept to implement a full scale system and gain more privacy controls over the (EHR^2)

Acknowledgements

I would like to express my deep gratitude to my advisor Dr. Dennis Guster for guidance and direction of my thesis, my committee members Dr. Jim Chen and Dr. Balsy Kasi for their suggestions and proposed changes. They taught me how to think technically. Without these people, completing this project would not have been possible.

I would also like to thank Dr. Susantha Herath for being patient with me and helping me overcome the roadblocks in my study.

I want to thank all my family and friends for trusting and supporting me all the time. Especially, Brahma Reddy for patiently testing my project. Your encouragement was very helpful in allowing me to finish my project.

This would not have been possible without all of you. Thank you for being patient with me all the time.

Table of Contents

Chapter Page

List of Figures

Chapter I

INTRODUCTION

Introduction

"Health care information about a patient is usually scattered among several clinical systems" (Meland, Røstad, Tondel, & Nytro, 2007, p. 2015). As previously stated, healthcare systems like never before hold vast amounts of digital data and that has further heightened public concern for privacy. The significant growth in software solutions in health care domains helps to address some of these privacy issues. There have been multiple models already implemented. However, they each have their own advantages and disadvantages

The following proposed model will be a combination of Role-based Access Control and Profile Based Access Control. It is designed in a way so that anyone can access the Electronic Health Record (EHR) of the patient, although, they will have to provide a valid reason, which will be logged and shared with the patient. It also addresses both emergency and non-emergency situations without compromising privacy. This proposed concept model can additionally serve as a nationwide tool by intelligently recognizing patterns of diseases".

This thesis paper will present all the steps involved with illustrated figures and tables. Additionally, the code is nicely segmented into various sections and commented to facilitate understanding the design which could be developed further at a later date. This code can also be imported into appropriate java development tools and can operate as a good starting point for any continuation.

Problem Statement

Healthcare utilizes large amounts of information, which is either created or transferred from one health organization to another for the purpose of administering healthcare to an individual or patient. The fact that patients are not bound to one single hospital for the treatment of their health conditions and the digital data flows from one healthcare organization to another healthcare organization, leads to a plethora of various privacy concerns. To address these concerns, the government of The United States created stringent legislation called the Health Insurance Portability and Accountability Act (HIPAA) in 1996 (United States Department of Health and Human Services [HHS], n.d.).

Over the last couple of years, there has been a significant growth in digital and software solutions to health care industry after the (EHR) superseded the traditional paper reports. This vast digital data that is being transferred from one healthcare organization to another increased the need for access control to ensure the privacy of the patients by securing their EHR.

Access control in a healthcare organization is quite a challenge due to the fact that almost all the patients usually do not show up until there is an emergency of some kind. Further, training of hospital staff on using the software is often a time intensive procedure. In this scenario, the current access control models are not really effective because of the fact that all the doctors treating a patient need to see the full medical history of that patient. This nullifies the concept of access control because the nurses and other administrative staff in the hospital could also have access to the patient's EHR.

The Profile Based Access Control model, currently being proposed, is a perfect theoretical fusion of Role-based Access Control and Process Based Access Control mechanisms.

This is a concept model that involves massive data processing which came into reality through Big Data concepts which greatly increases in the processing power of general computing.

Nature and Significance of the Problem

Health is a major consideration for every human being living on the planet. Health coverage is often very expensive and a lot of companies that invested in the healthcare domain are in an enormous race to provide better and more efficient health care to individuals. The information era led to the digitization of all data and every patient's record is predominately stored in an EHR. This EHR can be shared with the hospital staff to ascertain the patient's condition. If the patient changes hospital, his EHR will be shared with the new hospital. The sharing process is beyond the scope of the project. All the staff in that hospital could potentially have access to that EHR. This leads to many privacy concerns. Also, with scientific advancements in health monitoring devices and social media, privacy is becoming more and more of a concern every day. Implementing access control in such a scenario is often a difficult task because the enforced access control should not restrict accessing the necessary information during emergencies and should restrict any unauthorized access. If this proposed concept is successfully implemented, the data in current healthcare systems would be more private and secure than it is today. We can also apply the same logic to other similar scenarios.

Objective of the Study

The objective of the study is to provide a prototype that is more feasible and secure than the current methodologies used in accessing EHRs in health care domains.

Research Questions

1. How to implement the access control in health care systems?

2. What are the challenges in hardware and software for implementing the proposed methodology?
3. How will the proposed method enrich the access control better than current models?

Limitations of the Research

1. Someone has to manually check the patterns and place them in appropriate pools.
2. A lot of data analysis has to be done to find the common patterns among the diseases to even start this method.
3. All hospitals need a computer literate technician to manage the whole software.

Definition of Terms

Big Data concepts: The latest techniques used in storing and processing of huge amounts of data to meet the demands of the consumers are called big data concepts. There are many tools that support big data for storage, retrieval, processing and backup.

Casual Type Data Analysis: A type of data analysis where the behavior of variables is analyzed when one of them is changed. This is the gold standard of data analysis.

Data analysis: It is a process to inspect for commonality, cleaning of data, modeling of data and supporting decision making. We will use it for finding commonality.

EHR: Electronic Health Record is the digitalized information of a patient's paper chart. The goal of EHR is to make information available instantly and securely to authorized users.

Hardware: The physical equipment needed to implement the process.

HIPAA: Health Insurance Portability and Accountability Act 1996, is the legislation passed by the government of United States to protect the confidentiality and security of information in healthcare systems (HHS, n.d.).

Patient Control Health Record: The type of access control, in which the patient will have total control over his record and each patient can decide to whom his/her record can be shared with. These types of records are mostly confined to the individual clinics.

Pattern: A pattern is a recording of similarities between diseases.

Process Based Access Control: Based on the audit logs provided and looking at the medical guidelines, rules are created and the access control is enforced on these rules.

Prototype: A preliminary model which when adequately deployed and tested, gives a working product.

Role-based Access Control: A type of access control where each user is given a role and each role will be given access to perform specific tasks or functionalities. It is the most common type of access control.

Software: The tools required to use the hardware and to build the proposed methodology is referred to as software.

VEPR: Virtual-Electronic Patient Record is the digital form of an electronic patient record that holds all of patient's information in a file.

Summary

In this chapter we have covered why we need to solve the problem of increasing privacy concerns and also what are the major challenges involved in a healthcare system. This chapter also addressed the limitations encountered after the execution of this methodology. In the next chapter we will look at the background information of the problem and go through the literature review.

Chapter II

BACKGROUND AND REVIEW OF LITERATURE

Introduction

This chapter will cover the background related to the problem, literature reviewed to find the problem and then implement the proposed methodology.

Background Related to the Problem

According Røstad and Edsberg (2006, cited in Røstad & Nytroa, 2008), in healthcare organizations where role-based access control systems are being used, a careful study of audit logs showed that there has been an extensive use of exception mechanisms in situations where there is no need for an exception. The use of these exceptions to bypass access control in non-emergency situations and not providing informative reasons makes it impossible to audit the log for any misuse. The reason why the situation is so challenging is obvious because the existing algorithms are based on a human entity to make decisions for access control accordingly. There are multiple algorithms available in the market like Discretionary Access Control, Mandatory Access Control, Role-based Access Control and Process Based Access Control. However, because of the extensive abuse of the access control, the current mechanisms cannot strictly implement access control. However, simply bypassing the human entity from these methodologies will fix the problem to a significant extent and under no circumstances should the proposed concept model be a reason for the loss of healthcare or result in the death of a human being. This is what makes it so challenging of a task.

My first thought was to incorporate artificial intelligence or machine learning into the existing systems so as to bypass the human entity. But with the available resources at hand, I

thought about implementing a conceptual model that replicates the behavior of AI or a machine learning tool. I researched more into the depths of the current models and systems to make sure the proposed solution won't end up with similar disadvantages. There are many types of access control mechanisms that are currently being adopted around the world. We will be looking at the some of the popular mechanisms like Discretionary Access Control, Mandatory Access Control, Role-based Access Control, and Process Based Access Control in the next session. The major difference between the proposed system and the former systems is that the dependency on a human entity is reduced. We start where the system will be partially dependent on human entities and, once the proposed model is fully implemented and the patterns are stored, the model achieves near complete independence from human dependency.

Literature Related to the Problem

"Studies performed on audit data, in workshops, by observation and interviews helped determine the requirements" (Røstad, 2009). The currently implemented models for access control are sought out for and reviewed in science publications to understand the concepts in depth.

1. *Discretionary Access Control (DAC) and Mandatory Access Control (MAC)*: In DAC based systems, whoever creates the EHR will have complete ownership of the record and will have the capabilities to share it with a new user. On the other hand, in MAC based systems, all the users and their capabilities to access records are determined by one central authority. Either cases fail in situations where a technical or health emergencies arise.

2. *Role-based Access Control mechanism (RBAC):* The (RBAC) mechanism is a form of identity based access control system where the permissions are assigned to roles but not individuals (Zhao, 2008). Each individual is assigned a role depending on the nature of his/her work. Thereby, only the authorized personnel will be able to do certain functions. This method is the most widely implemented in the current market for its nature of inexpensive maintenance and easy to adapt features. The problem with this method is that, all the rules are static and do not adapt to the changing environment.

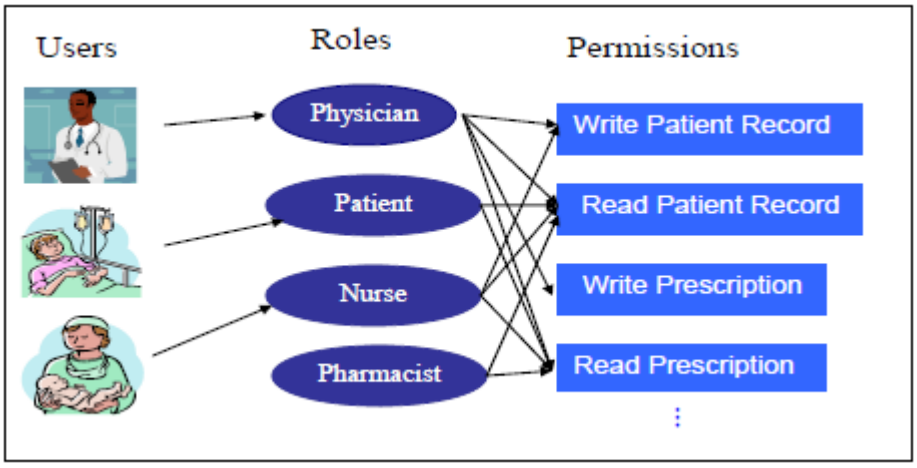

Figure 1. Illustration of Role-based Access Control System.

3. *Process Based Access Control mechanism:* A defined logic from the understanding of various patterns lying in the audit log, combined with the medical guidelines from three different viewpoints: user, system, and ideal version will help to create a (PBAC) mechanism. The problem with this mechanism is that all the audit logs have to be read manually and new rules have to be enforced, which makes it not only

expensive, but also very difficult to adapt to changes in a busy environment like a health care information system.

The proposed model is a mix of the Role-based and Process Based Access Control Systems with minor modifications to suit the requirements. This is in contrast to the Patient Controlled Health Records (PCHR), where "the patient is in control and determines who gets to access to his health information" (Røstad, 2008, p. 235.). The proposed conceptual model will fully automate the process with the ability to perform manual overrides.

Literature Related To Methodology

"The Biostatistics and Medical Informatics Department of Porto's Faculty of Medicine recently implemented a centralized Electronic Patient Record in order to integrate heterogeneous patient information within a university hospital" (Ferreira, Correia, Antunes, Palhares, Farinha, Altamiro, 2004, ¶ 2). In the case study, the VEPR system was studied for 10 months after its deployment and it has been observed that an average of 2,000 reports are collected every day from around 800 patients. How would this tool tackle the access control and manage the workflow of the entire system? The following suggestions are made.

Access Control Policy: Have all the chief doctors, information security specialists, and some IT staff participate in a meeting to discuss who can have access to which functionalities. Then the resulting policy could then be implemented. Mechanisms to be implemented:

- All doctors should have the functionality to access information of any patient in case of emergency or a system failure.
- All activities must be constantly monitored.
- Administrative information of the users of the systems should be up-to-date.

The role-based access control is the most compatible type of access control for the requirements observed. The requirements in the case study encouraged my proposed system to incorporate The Role-Based Access Control System.

Another interesting approach for providing access to healthcare environments would be a behavior-based access control system as proposed in by Yarmand, Sartipi, and Down, (2008) This model is fully dynamic and offers customizable access control. The model will receive security parameters to be considered and it will automatically scan through logs to determine which user needs which access dynamically and grant it accordingly. The draw back with this mechanism is that if a user repeatedly abuses the system, then the system will automatically create false rules for the user. This could result in giving a user unnecessary access to records.

The 4-Eyes-Principle is a form of multiple authorization process. In this method, both user and patient should login at the same time. The user will be accessing the information of the patient while the patient monitors the whole scenario. This method will give the patient an understanding of trust within access control. However, this method is not very applicable in real-life scenarios where emergency situations often arise in health care environments.

Degree of Adaptability

Røstad (2009) stated that it is very clear that there has been a tremendous demand for access control in healthcare systems. As a result, any enrichment or improvement made in the current system has a high potential in the market within the healthcare industry. Besides potential in the market, the access control management for the EHR in healthcare based industries is highly adaptable with significant potential for future development.

Summary

This chapter will share how I got the idea of addressing the problem of access control in healthcare systems and how it can be mitigated. It also lists all the references I went through to achieve this proposed system. The methodology that was used to create this proposed concept model will be shared in the following chapter.

Chapter III

METHODOLOGY

Introduction

This chapter will briefly cover how the proposed method is implemented. It will also cover various subsections, tasks and functions used in the proposed system.

Design of the Study

This study employs a qualitative approach for information gathering and the primary rationale is to explore possible solutions for one of the major challenges, access control, in the healthcare systems. The fact that the resources are limited in finding the required information compels to adopt this approach. It also enables us to focus on access control for healthcare but not the access control for the for other IT functions. Also, the information collected is purely based on the published papers that demonstrate the challenges involved in health care domains.

Data Collection

The data for the proposed methodology was created to demonstrate the purpose of the proposed conceptual model and how it could solve one of the challenges in implementing access control for healthcare domain. More detailed information on the type of data used and their declaration types are mentioned in the following figures.

Field	Type	Collation	Null	Key	Default	Pointed to
ID	varchar(12)	latin1_swedish_ci	NO	PK	(NULL)	User_details.userid
PASSWORD	varchar(255)	latin1_swedish_ci	YES		(NULL)	
TYPE	varchar(255)	latin1_swedish_ci	YES		(NULL)	
STATUS	int(11)	(NULL)	YES		(NULL)	

Figure 2. Login All Table Storing the Login Credentials for All Users in the System.

Field	Type	Collation	Null	Key	Default
USERID	varchar(255)	latin1_swedish_ci	NO	PK	(NULL)
FIRST_NAME	varchar(255)	latin1_swedish_ci	YES		(NULL)
LAST_NAME	varchar(255)	latin1_swedish_ci	YES		(NULL)
DOB	varchar(255)	latin1_swedish_ci	YES		(NULL)
EMAIL	varchar(255)	latin1_swedish_ci	YES		(NULL)
MOBILE	varchar(255)	latin1_swedish_ci	YES		(NULL)
GENDER	varchar(255)	latin1_swedish_ci	YES		(NULL)
USERTYPE	varchar(255)	latin1_swedish_ci	YES		(NULL)
STATUSMASTER	int(11)	(NULL)	YES		(NULL)
CREATIONBY	varchar(255)	latin1_swedish_ci	YES		(NULL)
PERMISSIONGIVEN	varchar(255)	latin1_swedish_ci	YES		(NULL)

Figure 3. User Details Table Storing All the Details Required for All Users in the System.

Field	Type	Collation	Null	Key	Default	Pointed to
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User details.userid
DOC_ASSIGNED	varchar(255)	latin1_swedish_ci	YES		(NULL)	
CREATEDDATE	datetime	(NULL)	YES		(NULL)	
STATUS	int(11)	(NULL)	YES		(NULL)	

Figure 4. Doc Assigned Table Storing the List of All the Doctors Assigned.

Field	Type	Collation	Null	Key	Default	Pointed to
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
ADMIN_ASSIGEND	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
CREATEDDATE	Datetime	(NULL)	YES		(NULL)	
STATUS	int(11)	(NULL)	YES		(NULL)	

Figure 5. Admin Assigned Table Storing the List of All the Admins Assigned.

Field	Type	Collation	Null	Key	Default	Pointed to
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.user_id
NURSE_ASSIGNED	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.user_id
CREATEDDATE	Datetime	(NULL)	YES		(NULL)	
STATUS	int(11)	(NULL)	YES		(NULL)	

Figure 6. Nurse Assigned Table Storing the List of All the Nurses Assigned.

Field	Type	Collation	Null	Key	Default	Features	Pointed to
NID	int(11)	(NULL)	NO	PK	(NULL)	auto_increment	
NOTIFICATION	varchar(255)	latin1_swedish_ci	YES		(NULL)		
FROMID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)		User_details.userid
TOID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)		User_details.userid
STATUS	int(11)	(NULL)	YES		(NULL)		
PID	varchar(255)	latin1_swedish_ci	YES		(NULL)		Patient_details

Figure 7. Notifications Table Storing the Information Required for Pop-Up Notifications.

Field	Type	Collation	Null	Key	Default	Features	Pointed to
ID	int(11)	(NULL)	NO	PK	(NULL)	auto_increment	
MESSAGE	varchar(255)	latin1_swedish_ci	YES		(NULL)		
CREATEDBY	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)		User_details.userid

Figure 8. Override Table Storing the Log of Overrides.

Field	Type	Collation	Null	Key	Default	Pointed to
USERID	varchar(255)	latin1_swedish_ci	NO	PRI	(NULL)	User_details.userid
ADDRESS	varchar(255)	latin1_swedish_ci	YES		(NULL)	
INSURANCE	varchar(255)	latin1_swedish_ci	YES		(NULL)	
POLOCIES	varchar(255)	latin1_swedish_ci	YES		(NULL)	
WARD	varchar(255)	latin1_swedish_ci	YES		(NULL)	
ROOM	varchar(255)	latin1_swedish_ci	YES		(NULL)	
SSN	INT(10)	latin1_swedish_ci	YES		(NULL)	
CREATEDDATE	Datetime	(NULL)	YES		(NULL)	

Figure 9. Patient Details Table Storing All Necessary Patient Information.

Field	Type	Collation	Null	Key	Default	Pointed to
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
MEDICINE	varchar(255)	latin1_swedish_ci	YES		(NULL)	
PRICE	decimal(10,2)	(NULL)	YES		(NULL)	
DATE	Datetime	(NULL)	YES		(NULL)	
PATERNID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	Patterns.patternid
SPECILIZATIONID	varchar(255)	latin1_swedish_ci	YES		(NULL)	

Figure 10. Patient Med Table Storing the Information of Medicines Used for Patient.

Field	Type	Collation	Null	Key	Default	Pointed to
PATTERNID	varchar(255)	latin1_swedish_ci	YES	PK	(NULL)	
PATTERN	varchar(255)	latin1_swedish_ci	YES		(NULL)	
CODE	varchar(255)	latin1_swedish_ci	YES		(NULL)	
COUNT	int(255)	(NULL)	YES		0	
MEDICINAMES	varchar(255)	latin1_swedish_ci	YES		(NULL)	
DESCRIPTION	varchar(255)	latin1_swedish_ci	YES		(NULL)	
FLAG	varchar(255)	latin1_swedish_ci	YES		(NULL)	
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
CURRENTLY_ADMITED	varchar(255)	latin1_swedish_ci	YES		(NULL)	

Figure 11. Patterns Table Storing All the Information Necessary for Recording Patterns.

Field	Type	Collation	Null	Key	Default	Pointed to
DOC_ID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
ADMIN_ID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
PATIENT_ID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
NURSE_ID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid

Figure 12. Sequence Table Storing the List of Patients and Staff Grouped Together.

Field	Type	Collation	Null	Key	Default	Foreign Key
USERID	varchar(255)	latin1_swedish_ci	YES	FK	(NULL)	User_details.userid
PRIVATE	longtext	latin1_swedish_ci	YES		(NULL)	
PUBLIC	longtext	latin1_swedish_ci	YES		(NULL)	
CREATEDBY	varchar(255)	latin1_swedish_ci	YES		(NULL)	
STATUS	int(11)	(NULL)	YES		(NULL)	
DATE	datetime	(NULL)	YES		(NULL)	

Figure 13. Notes Table Holding the Information Necessary for Note Taking Capabilities.

Proposed system. The proposed access control model is based on custom entities called profiles, tags, functionalities, share matrix, pattern, and pools.

Profiles are categorized into the four following major categories: (a) patient profile, (b) doctor profile, (c) nurse profile, and (d) admin profile.

- Patient profile: Each patient will be created with a patient profile. If this is the first time the patient is visiting the medical facility, then any employee in the hospital, with an Admin profile will create a profile for the patient. The patient will have the owner rights for his EHR. Upon his diagnosis, a patient tag, ward tag, a hospital tag and a disease tag will be assigned to the patient.
- Doctor profile: Each doctor will have a predefined a doctor tag, a ward tag and associated with a group of nurses who work with similar ward tags. Also, the tags of patients, whom the doctor is currently treating are associated with the profile. A display of (exception count[5]) will be displayed and shall be reviewed later by the admin.
- Nurse profile: Each nurse will be given a nurse tag, predefined ward tags and associated with a group of doctors who work with similar ward tags. Also, the tags of patients, whom the nurse is currently treating, are associated with the profile. A display of exception count will be displayed and shall be reviewed later by the admin.
- Admin profile: Each admin will be given an admin tag associated with a list of all the available tags in the healthcare organization which includes wards, doctors, nurses, other hospitals.

Tags are categorized into the following seven different sets:

- Patient tags: Each patient is given a tag that acts as a unique identifier.
- Doctor tags: Each doctor is given a unique tag name based on their real name.
- Nurse tags: Each nurse is given a unique tag name based on their real name.
- Admin tags: Each admin is given a unique tag name based on their real name.

- Ward tags: Each ward is given a unique tag name based on their functionality.
- Hospital tags: Each hospital in the system is given a unique tag name to identify them.
- Disease tags*:* Each disease is given a specific tag name and recorded in every hospital in the system.

Functionalities for each profile differ as their job junction differs. The available functions contained in each profile are mentioned below.

- Patient profile: Each patient can grant or revoke access rights to the doctor, nurse and/or admin. All these access rights (read, write, and share) can also be given based on the tag category instead of certain specific people. For example, I being a cancer patient can grant read access to all the doctors working under the tag "cancer ward". Also, the patient can specifically mention if the person gaining access to your EHR can view your record completely or only certain categories as mentioned in the (Share matrix[5]).
- Doctor profile: Each doctor will have the ability to choose a patient under a common ward tag, assign nurse(s) to patients, and create doctor records, which can be shared with the nurses associated with that patient or other nurses in the same ward. She/he can also view nurse records created by all the nurses who attended the patient. The doctor can also override to gain ownership of the patient's EHR, in case of an emergency.
- Nurse profile: Each nurse will be able to choose or pick certain patients from a ward with certain ward tag. They can also create nurse records which are shared

automatically with the doctors who are treating that particular patient. A nurse will also have the ability of assigning a doctor to a patient. In case of an emergency, a nurse can see all the data seen by a doctor. However, they will not have the ability to gain ownership of the (EPR[6]).

- Admin profile: Each admin will be able to create and modify tags for any patient. However, they will not be able to view the details of patient's EHR without an override function. Admins will also be able to view admission records, billing and insurance information, physical location, doctors and nurses assigned to that particular patient.

Share matrix is a custom matrix that defines the level of sharing by patients to their respective doctors/nurses. The rows of the matrix represent share category, while the columns represent the share type. Share categories are of six types:

- Insurance record [I]
- Medications record [M]
- Diseases record [D]
- Bills record [B]

Share types are of two types:

- Vague [V]
- Detailed [T]

	V	T
M	Generic categories of medication used	Detailed names of medication used
D	Names of diseases had so far	Names, state and all the doctor reports generated for all the diseases
B	Bills paid per hospital/disease and name(s) of current insurance provider(s)	Detailed transaction reports and all the insurance details.
I	Insurance name	Insurance name and list of policies registered.

Figure 14. Share Matrix.

Any combination of access can be given by a patient to a doctor(s)/ nurse(s).

Pattern is a recording that is observed by the software when a doctor or nurse views certain tags of the patient profiles. Certain patterns are locked and require admin override. Certain patterns are already recorded in the (green pool[7]). Any pattern that is not in the green pool will cause an increase in the exception count for doctor(s)/ nurse(s) and raises an alert. Any pattern that requires information other than the tags specified by the admin generates alerts. All alerts should contain an explanation as of why that information is accessed. This information will be used for auditing purposes. Over time, when similar alerts are generated, these patterns are pushed into the green pool after a certain threshold value is reached. This way, the software adapts to human ways of handling information and thereby less overhead will be generated in the audit log.

For example if a patient arrives suffering from a fever and has a history of cold and cough, it is quite common to check for information on cold and cough too. However, if the

admin only attached a fever tag to the patient, the doctor will simply make an override to view information on a cold and cough. This creates an alert that the doctor viewed something that he was not supposed to. Therefore, a request for that reason will be prompted later that day and the doctor/nurse will submit the reason. A number of such cases will push this pattern to green pool. At that point, the doctor/nurse does not have to provide the reason anymore for the same pattern, as it is already adapted.

These patterns can be further shared among all the healthcare organizations for quicker adaptability.

Pools are used along with patterns to adapt to human ways. There are three types of pools:

- Red pool: Any pattern that is new to the software comes to red pool. Information about why it is an exceptional case should be provided by the doctor/nurse. Certain patterns are locked in the red pool because those patterns (which are violations) can be pushed into green pool if too many people keep repeating the same mistake.
- Research pool: Any pattern that rarely occurs, but is scientifically acceptable, based on the reasons provided for exception comes into this pool. These reasons can be viewed either manually, to push these patterns into research pools. Or a certain threshold value can be given for the pattern to be pushed into this pool.
- Green pool: Any pattern that reaches certain amount of hits will be automatically sent to green pool. These patterns do not raise any alerts causing smooth work and less overhead for auditing and thereby giving better access control.

Tools and Techniques

The following tools were used in the process of implementing the proposed system:

MyEclipse 9: My Eclipse was used to build the front end development for the whole project. It acts as a framework and user interaction tool. Another benefit to this tool is that it bears in mind that many users will be in non-technical roles thus great care was taken to keep the user interface simple and clear to understand

SQLyog: SQLyog was used for an easy manipulation of sql queries. The user interaction provided by the tool helps us manipulate queries on the fly. It will also allow us to input manual values into the tables that we have already created using SQLyog.

We can use any front end and back end development tools as long as it solves the problem. A further enhancement for Profile Based Access Control would be to implement the same using one of the big data databases to process the massive amounts of unstructured data. Since, the current healthcare domain will mostly use structured data, implementing the health care in table based databases will suffice for now.

Summary

A qualitative approach was used to implement the methodology. The data used here is made up data to demonstrate that the whole process is feasible. The methodology involves different type of roles and each role will have certain set of functions, tasks and authorization to view a patient's EHR. The share matrix assists administrators in deciding which role has access to which level of detail for a patient's EHR. The tools used to complete the prototype are MyEclipse for Graphical User Interface (GUI) and SQLyog for the manipulating the sql queries. Any other tools and language that serves a similar purpose can also be used.

Chapter IV

DATA PRESENTATION AND ANALYSIS

Introduction

This chapter will demonstrate the experimental data used in the prototype can be implemented in the proposed system.

Data Presentation

The data for this experiment is categorized into the following categories:

- Admins Assigned: All the admins assigned to a specific patient is called admins assigned.
- Admins Unassigned: All the admins who are not assigned to a specific patient come under the list of unassigned admins.
- Doctors Assigned: All the doctors who are assigned to a specific patient are the doctors assigned for that patient.
- Doctors Unassigned: All the doctors who are not assigned to any given patient are placed on the list of unassigned doctors.
- Nurses Assigned: All the nurses that are assigned to a specific patient are called nurses assigned.
- Nurses Unassigned: All the nurses not treating a given patient are called the list of unassigned nurses.
- Patients: List of patients admitted.

For example, let us take a scenario where Nitish Singh is a patient. He is quite sick with the flu and goes to the hospital in need of urgent care. An admin (say Koushik Chilukuri,

AD000004) will seek his details and creates a profile for him. Let the patient ID be PI000001. After diagnosing the patient, he will be assigned a doctor by Koushik Chilukuri. Say, Naresh Amirineni (DR000003) is the doctor assigned to Nitish Singh and all other doctors will come under doctors unassigned. Now, Naresh Amirineni will start treating Nitish Singh and he will assign nurses (say Jagruth Ainapudi, NR000001) to keep an eye on Nitish Singh. The assigned list of nurses will be nurses assigned and all others will come under nurses unassigned. Now, all the list of assigned staff for Nitish (Naresh, Koushik and Jagruth) will be able to see the detailed information and all the staff that is not assigned will either only be able to see a summary of information or no information at all, depending upon their role.

Data Analysis

(Casual type)[11] data analysis was made to check the results of the implemented prototype. Additional exploratory research of this proposed concept model would be useful in studying the relationships among pattern tags. After going through and checking each and every profiles view of same data, it is clarified that the flow of information is controlled by the proposed access control. In our above stated example,

Nitish's data is categorized into two types—admin related data and doctor related data. Naresh and Jagruth, being the staff treating Nitish, will see only disease related data but not admin related data. On the other hand, Koushik will only see admin related data with no access to disease related data. This Other admins will only see a summarized version of the admin related data and other doctors will only see a summarized version of the disease related data.

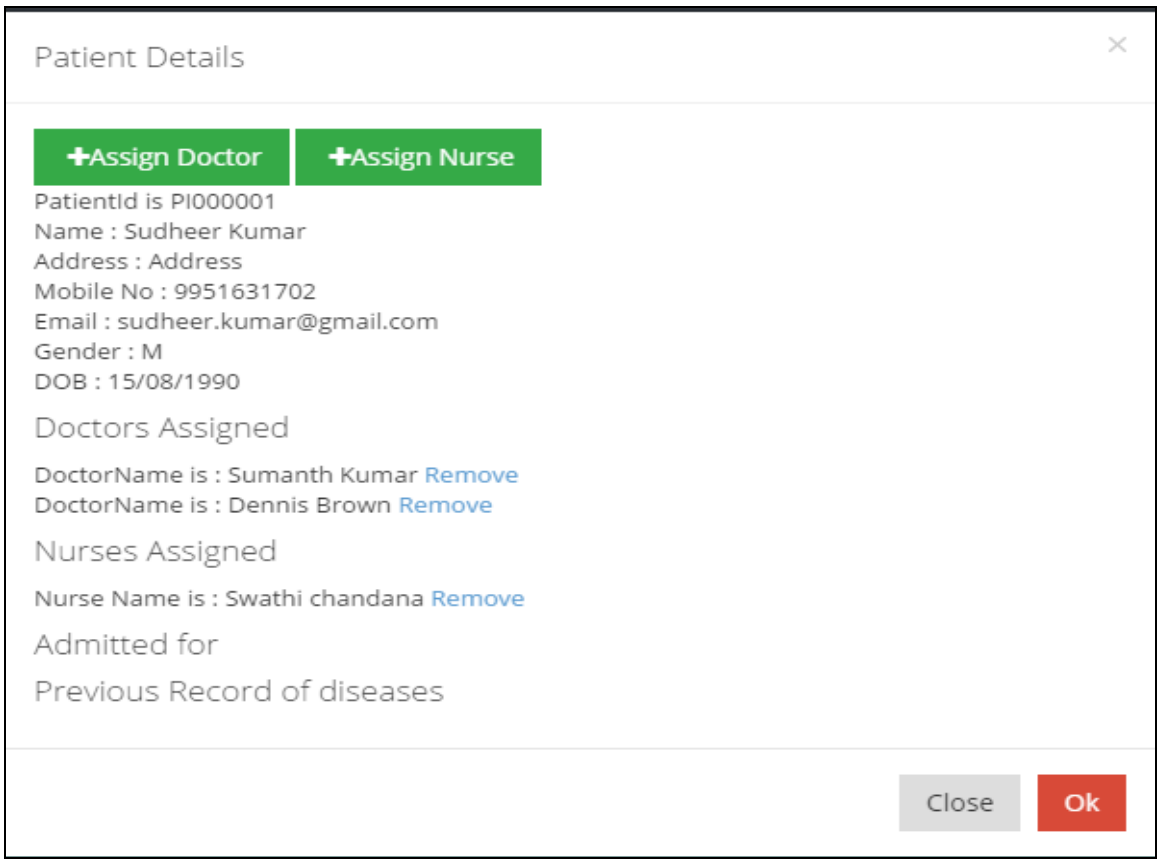

Figure 15. Screenshot Showing the Patient Details When Seen as a Doctor Who is Treating.

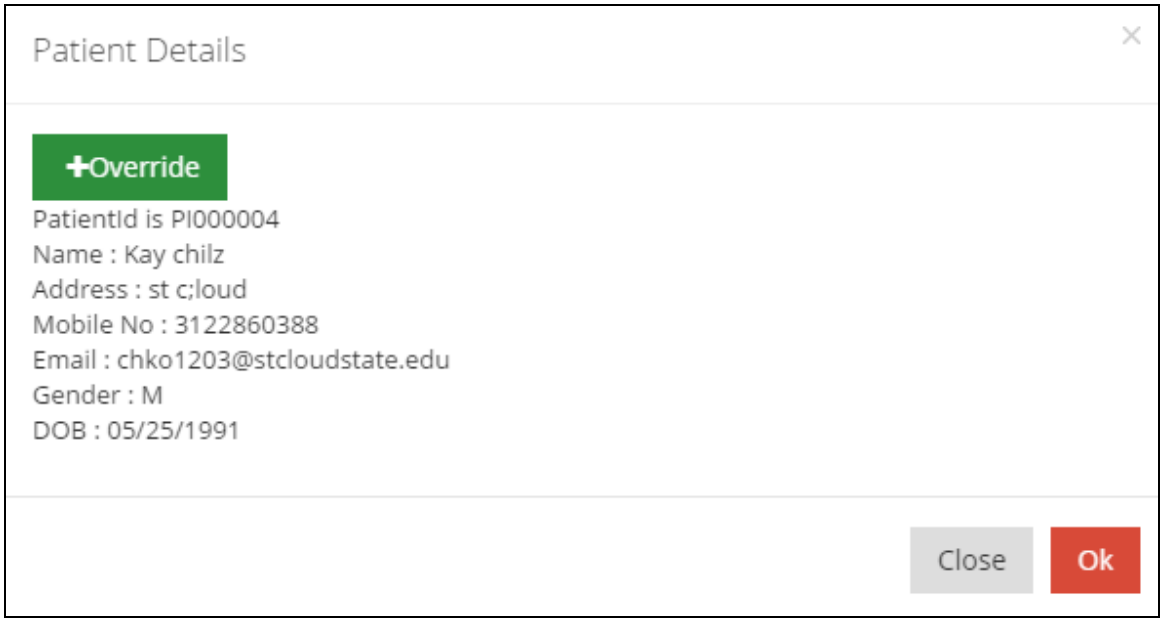

Figure 16. Patient Information When Seen as a Doctor Who is Not Assigned to the Patient.

Figure 17. Patient Information When Seen as an Admin Who is Assigned to the Patient.

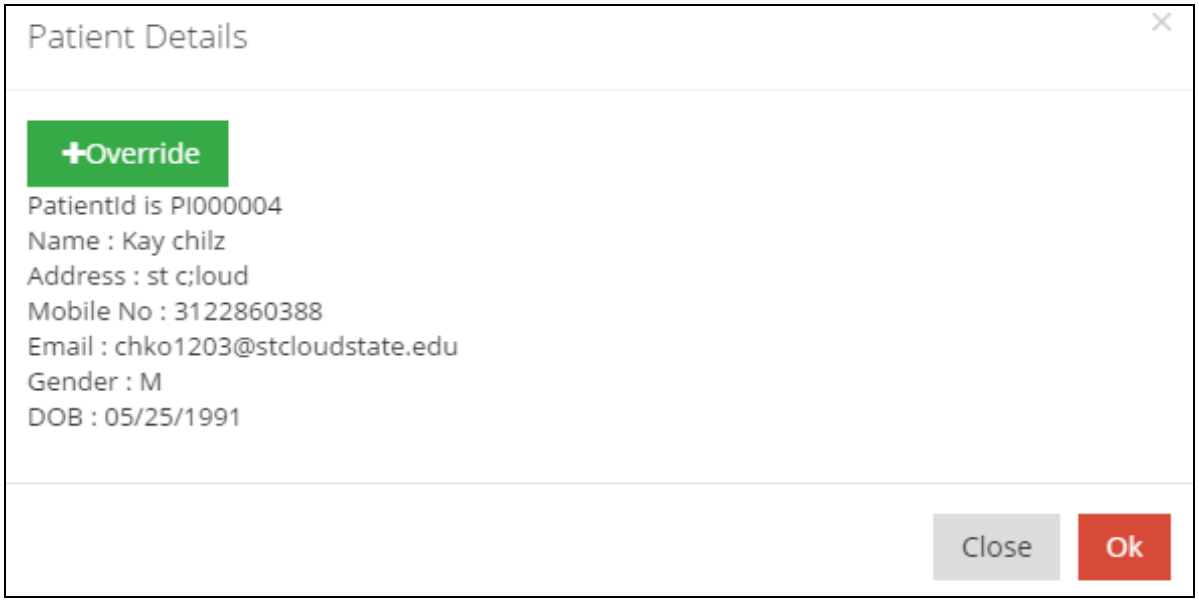

Figure 18. Patient Information When Seen as an Admin Who is Not Assigned To the Patient.

These screenshots taken after the implementation of the proposed model clearly shows the level of implementation of access control.

Summary

Overall, this chapter has covered how the data was created for the prototype and what parameters are considered during the creation of this sample data and how we have analyzed the data to validate the results after the implementation of the prototype. The next chapter will depict the results, conclusions and recommendations.

Chapter V

RESULTS, CONCLUSION, AND RECOMMENDATIONS

Introduction

This chapter will briefly summarize the methodology used in the proposed system, show the results achieved from the implementation of the process and address the study questions discussed in above chapters.

Results

The proposed system will contain profiles, tags, and patterns. There are four types of profiles: Doctor, Nurse, Patient, and Admin. Each profile will have access to certain types of data. For example, the doctors and nurses will see disease related data and the admins will see admin related data. The patients can see their HER and then go through the log report on who accessed which data. Data elements are each assigned a tag whether it is a disease tag, doctor tag, nurse tag, admin tag, or patient tag. The same tags are also used to record patterns among the data elements to identify suspicious activities within the system after a certain period of learning by the machine. These patterns are then automatically classified into red, green, and yellow pools to determine, whether to raise an alarm or to stay calm or to submit for further research. Further, all the staff treating a patient will be classified as staff treating and will have full access to the patient's information and all the staff not treating the patient will be classified as non-treating staff and will not be given full access to information. Instead, a summarized version of information is seen. This way the information is neatly decentralized and shows only relevant information to the appropriate person. In cases of emergency, an override can be gained by the non-treating staff when they enter the reason. However, that will send an email to the patient

immediately that someone has gained access to their information through an override, thereby making the patient aware of the situation.

The study questions for the project are answered in this section:

Q1). How to implement the access control in health care systems?

A1). By following the Dynamic Profile Based Access Control, we will be one step closer to fully implementing access control in health care systems and other similar environments. The figures 15, 16, 17 and 18 show clearly that the same information when accessed by different profiles results in different information views. Therefore, verifying that successful access control has been implemented.

Q2). What are the challenges in hardware and software for implementing the proposed methodology?

A2). There are no hardware challenges considering the massive cloud computing architecture that is available and also all the big data databases make it possible to configure the hardware to fully implement the system with terra bytes of real data. For the software, one major challenge is to do the analysis to set the predefined patterns among the data elements. The current analysis and machine learning tools seem to offer a promising solution to overcome this challenge.

Q3). How will the proposed method enrich access control to be better than current models?

A3). The proposed system will decrease the dependency on human entity, thereby increasing the automation of the whole process. Even though it is not 100%

automated, it will learn the patterns so as not to raise false alarms, making the problem of abusing the override option obsolete.

Conclusion and Future Work

The Dynamic Profile Based Access Control proposed will have a little more control on audit systems when compared with the Role-based Access Control or the Process based Access Control in due time because of its adaptable nature. Considering the fact that these patterns can be tampered with and yet all patients need not be computer literate to deal with problem, future work is still needed to provide more enriched and efficient models to provide better access control. Also, more work needs be done in defining patterns among diseases in the current world and shared in the appropriate pools, for this model to be fully efficient. Artificial intelligence is emerging as a promising concept that could be used to further advance this proposed concept offering even more opportunities for efficiencies. As popular as A1 is among industry trends, it is still considered an expensive solution with a debatable amount of adaptability in the market.

References

Ferreira, A., Correia, R., Antunes, L., Palhares, E., Farinha, P., & Altamiro, P. (2004). *Modelling access control for a complex health care organization*. Retrieved from http://www.dcc.fc.up.pt/~lfa/rbac.pdf

Meland, H., Røstad, I., Tondel, I., &Nytro, O. (2007). Building sustainable health systems. *Proceedings of the 12th World Congress on Health Medical Informatics*, 2015-2016, Brisbane.

Røstad, L. (2008). *An initial model and a discussion of access control in patient controlled health records*. Paper presented at the International Workshop on Privacy and Assurance, Barcelona, Spain.

Røstad L. (2009). *Access control in health care information systems* (Doctoral thesis, Norwegian University of Science and Technology). Retrieved from http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A134515&dswid=-3369

Røstad, L., & Nytroa, O. (2008). *Personalized access control for personally controlled health records*. Paper presented at The Second Computer Security Arthitecture Workshop, Fairfax, VA.

United States Department of Health and Human Services. (n.d.). Summary of the HIPPA privacy rule. Retrieved from http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/index.html

Yarmand, M. H., Sartipi, K., & Down, D. G. (2008). *Behavior-based access control for distributed healthcare environment*. Paper presented at Computer-Based Medical Systems, Jyvaskyla, Finland.

Zhao L. (2008). *A role-based access control security model for workflow management system in an e-healthcare enterprise*. (Unpublished master's thesis). The Florida Agricultural and Mechanical University College of Arts And Sciences, Tallahassee, FL.

Appendix

Password Generation Process

```
package org.hospital.utils;

/**
 * @author Koushik
 * @version 1.1.0
 * @since 19-march-2015
 *
 */
public class PasswordGenerator {
   /**
    * Minimum length for a decent password
    */
   public static final int MIN_LENGTH = 8;


   /**
    * The random number generator.
    */
   protected static java.util.Random r = new java.util.Random();


   /* Set of characters that are valid. Must be printable, memorable,
    * and "won't break HTML" (i.e., not '<', '>', '&', '=', ...).
    * or break shell commands (i.e., not '<', '>', '$', '!', ...).
    * I, L and O are good to leave out, as are numeric zero and one.
    */
   protected static char[] goodChar = {
```

```
    // Comment out next two lines to make upper-case-only, then
    // use String toUpper() on the user's input before validating.
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'm', 'n',
    'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'M', 'N',
    'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    '2', '3', '4', '5', '6', '7', '8', '9',
    '+', '-', '@',
};

/* Generate a Password object with a random password. */
public static String getNext() {
    return getNext(MIN_LENGTH);
}

/* Generate a Password object with a random password. */
public static String getNext(int length) {
    if (length < 1) {
        throw new IllegalArgumentException("Ridiculous password length " + length);
    }
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < length; i++) {
        sb.append(goodChar[r.nextInt(goodChar.length)]);
    }
    return sb.toString();
}
```

}

Mail Transportation process

```
package org.hospital.utils;

import java.io.File;
import java.io.UnsupportedEncodingException;
import java.util.Properties;
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.Authenticator;
import javax.mail.BodyPart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
```

```
/**
 * SMTP mail process
 * @author Koushik
 *
 */
public class SendMailUsingAuthenticationwithattachment {

	private static final String SMTP_HOST_NAME = "198.38.82.160";
	private static final String SMTP_AUTH_USER = "support@kayhospitals.in";
	private static final String SMTP_AUTH_PWD = "Mellow12@";
	private static final String emailFromAddress = "support@kayhospitals.in";
	private static  String sub = "";
	private static String MsgTxt="";

	public static void main(String args[])
	{

		SendMailUsingAuthenticationwithattachment  mail = new
SendMailUsingAuthenticationwithattachment();
		String tos[]={"kaychilz88@gmail.com"};
		String filenames[]=null;
		String sub="hi";
		String body="hello";
				try
				{
```

```
                    System.out.println("Before sending mail");

                        mail.postMail(tos,null,null, sub, body, filenames);

                    System.out.println("after sending mail");
                }
                catch (Exception e) {
                    e.printStackTrace();

                }

    }

    public void postMail(String to[],String cc[],String bcc[], String subject,String bodymsg, String[]
filenames) throws MessagingException, UnsupportedEncodingException {
        boolean debug = false;
        java.security.Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
        sub=subject;
        MsgTxt=bodymsg;

        //Set the host smtp address
        Properties props = new Properties();
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.starttls.enable","false");
        props.put("mail.smtp.host", SMTP_HOST_NAME);
```

```
props.put("mail.smtp.auth", "true");
props.setProperty("mail.smtp.port", "" + 2525);
Authenticator auth = new SMTPAuthenticator();
Session session = Session.getDefaultInstance(props, auth);
session.setDebug(debug);

// create a message
Message msg = new MimeMessage(session);
// set the from Address
InternetAddress addressFrom = new InternetAddress(emailFromAddress,"Team
Hospital");
msg.setFrom(addressFrom);

// set the To Address
if(to!=null)
{
	InternetAddress[] addressTo = new InternetAddress[to.length];
	for (int i = 0; i < to.length; i++) {
		addressTo[i] = new InternetAddress(to[i]);
	}
	msg.setRecipients(Message.RecipientType.TO, addressTo);
}
// set the CC Address
if(cc!=null)
{
```

```
        InternetAddress[] addressCC = new InternetAddress[cc.length];
        for (int i = 0; i < cc.length; i++) {
                addressCC[i] = new InternetAddress(cc[i]);
        }
        msg.setRecipients(Message.RecipientType.CC, addressCC);
}
// set the BCC Address
if(bcc!=null)
{
        InternetAddress[] addressBCC = new InternetAddress[bcc.length];
        for (int i = 0; i < bcc.length; i++) {
                addressBCC[i] = new InternetAddress(bcc[i]);
        }
        msg.setRecipients(Message.RecipientType.BCC, addressBCC);
}
// Setting the Subject and Content Type
msg.setSubject(sub);
if(filenames==null)
        msg.setContent(getMultipart());
else
{
        File[] attachments=new File[filenames.length];
        for(int i=0;i<filenames.length;i++)
                attachments[i]=new File(filenames[i]);
        msg.setContent(createAttachment(attachments));
```

```
            }
            Transport.send(msg);
}



/**
 * SimpleAuthenticator is used to do simple authentication when the SMTP
 * server requires it.
 */
private static Multipart getMultipart() throws MessagingException
{

            Multipart multi = new MimeMultipart("mixed");
            multi.addBodyPart(createContent());
            return multi;

}

private static BodyPart createHtmlBody() throws MessagingException {
            BodyPart html = new MimeBodyPart();
            html.setContent(MsgTxt, "text/html;charset=gbk");
            return html;
}
```

```
private static BodyPart createContent() throws MessagingException {
        BodyPart content = new MimeBodyPart();
        Multipart relate = new MimeMultipart("related");
        relate.addBodyPart(createHtmlBody());



        content.setContent(relate);
        return content;
}
private static Multipart createAttachment(File[] files) throws MessagingException
{

        Multipart multi = new MimeMultipart("mixed");

        for(int i=0;i<files.length;i++)
        {
                BodyPart attach = new MimeBodyPart();
                DataSource ds = new FileDataSource(files[i]);
                attach.setDataHandler(new DataHandler(ds));
                attach.setFileName(ds.getName());
                multi.addBodyPart(attach);

        }
        multi.addBodyPart(createContent());
        return multi;
```

```
        }
        private class SMTPAuthenticator extends javax.mail.Authenticator {

                public PasswordAuthentication getPasswordAuthentication() {
                        String username = SMTP_AUTH_USER;
                        String password = SMTP_AUTH_PWD;
                        return new PasswordAuthentication(username, password);
                }
        }
}
```

Authentication of USER:

```
package org.hospital.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.MappingDispatchAction;
import org.hospital.dao.GenericQueries;
import org.hospital.utils.CommonConstants;
```

```
import org.hospitals.forms.UserForm;

/**

 *

 * Authentication class defines the security of an application. It secures from CSRF (Cross Site Request Forgery) from hackers.

 * @author Koushik

 *

 */

public class Authentication extends MappingDispatchAction{


        /**

         * Defines the login page code and loads the login Page

         * @param mapping

         * @param form

         * @param request

         * @param response

         * @return

         * @throws Exception

         */

        public ActionForward loginForm(ActionMapping mapping,

                        ActionForm form,

                        HttpServletRequest request,

                        HttpServletResponse response)throws Exception{


                @SuppressWarnings("unused")

                HttpSession session=request.getSession(true);
```

```
            System.out.println("we are in login Form");
            request.setAttribute(CommonConstants.TITLE, "Login Page");
            request.setAttribute(CommonConstants.MAIN_PAGE, "/pages/leftpane/login.jsp");
            return mapping.findForward(CommonConstants.GLOBAL_CONSTANT);
    }

    /**
     * authenticate the user and navigate to the appropriate modules
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return
     * @throws Exception
     */
    public ActionForward login(ActionMapping mapping,
   ActionForm form,
   HttpServletRequest request,
   HttpServletResponse response)
  throws Exception {
     HttpSession session = request.getSession(true);
            UserForm user=null;
            String errorText=new String();
            String loginId=request.getParameter("username"); //requesting the username
            String password= request.getParameter("password"); // Requesting the password
            String page = null;
```

```
                if((user=GenericQueries.authenticate(loginId, password))!=null){
                        request.setAttribute(CommonConstants.ROLETYPE, user.getUserType());
                        session.setAttribute("user", user);

                                request.setAttribute(CommonConstants.ROLETYPE, user.getUserType());
                                request.setAttribute(CommonConstants.TITLE,"Welcome To Hospital
Management");

        request.setAttribute(CommonConstants.MAIN_PAGE,"/pages/main/admin_home.jsp");
                                page = "login";

                        }else if(((user=GenericQueries.authenticate(loginId, password))==null)) {

                                errorText="UserIdPassword";
                                request.setAttribute("errorText", errorText); //represents the error message
                                page = "relogin";
                        }
                        return mapping.findForward(page);
                }

        /**
         * Logout the user
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return
```

```
     * @throws Exception
     */
    public ActionForward logout(ActionMapping mapping,
   ActionForm form,
   HttpServletRequest request,
   HttpServletResponse response)
  throws Exception {
            HttpSession session = request.getSession(false);

            session.removeAttribute("user"); // remove the user session
            session.removeAttribute("txnDone"); // clearing the transactions
            request.setAttribute("title","Welcome");
            request.setAttribute("errortext", "logout");

                return mapping.findForward("logout");

            }
}
```

Action Generation for Appropriate Roles :

```
package org.hospital.action;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.MappingDispatchAction;
import org.hospital.dao.GenericQueries;
import org.hospital.service.GenericService;
import org.hospital.utils.AdaranSecurity;
import org.hospital.utils.CommonConstants;
import org.hospitals.forms.NotesForm;
import org.hospitals.forms.NotificationForm;
import org.hospitals.forms.UserForm;
/**
 *
 * This class defines generations of
 *  {Super Admin}
 *  {Admin}
 *  {Doctor}
 *  {Nurse}
 * . Here all the operations performed will be captured due to the security purpose
 * @author Koushik
 *
 */
```

```
public class CreationAction extends MappingDispatchAction implements CommonConstants{

	/**
	 * Navigation to the user's home page once after the user click on home.htm
	 * @param mapping
	 * @param form
	 * @param request
	 * @param response
	 * @return
	 * @throws Exception
	 */
	public ActionForward home(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		if(AdaranSecurity.authenticated(request)){
			HttpSession session = request.getSession(false);

			UserForm user = (UserForm)session.getAttribute("user");
			saveToken(request);
			request.setAttribute(ROLETYPE, user.getUserType());
			request.setAttribute(TITLE, "My Portal");
			request.setAttribute(MAIN_PAGE,"/pages/main/admin_home.jsp");
		}
		return mapping.findForward(GLOBAL_CONSTANT);
	}
```

```
/**
 * Defines the admin registration page.
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward addAdmin(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response)
                    throws Exception {
          if(AdaranSecurity.authenticated(request)){
                    HttpSession session = request.getSession(false);


                    UserForm user = (UserForm)session.getAttribute("user");
                    saveToken(request);
                    request.setAttribute(ROLETYPE,user.getUserType());
                    request.setAttribute(TITLE,"Admin Creation Form");
                    request.setAttribute(MAIN_PAGE,"/pages/main/create_admin.jsp");
          }
          return mapping.findForward(GLOBAL_CONSTANT);
}
```

```
/**
 * Defines the doctor creation page
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward addDoctor(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);

                UserForm user = (UserForm)session.getAttribute("user");
                this.saveToken(request);
                request.setAttribute(ROLETYPE,user.getUserType());
                request.setAttribute(TITLE,"Doctor Creation Form");
                request.setAttribute(MAIN_PAGE,"/pages/main/create_doctor.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}
```

```
/**
 * Defines the Nurse Creation pages
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward addNurse(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);

                UserForm user = (UserForm)session.getAttribute("user");
                this.saveToken(request);
                request.setAttribute(ROLETYPE,user.getUserType());
                request.setAttribute(TITLE,"Nursce Creation Form");
                request.setAttribute(MAIN_PAGE,"/pages/main/create_nurse.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}
```

```
/**
 * Defines the patient creation
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward addPatient(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response)
                    throws Exception {
          if(AdaranSecurity.authenticated(request)){
                    HttpSession session = request.getSession(false);

                    UserForm user = (UserForm)session.getAttribute("user");
                    GenericService service = new GenericQueries();

                    List<UserForm> userForms = service.getDetailsofDoctorsOrNurses("D");
                    List<UserForm> nurses = service.getDetailsofDoctorsOrNurses("N");

                    request.setAttribute("nurses", nurses);
                    request.setAttribute("userForms", userForms);
```

```
                this.saveToken(request);
                request.setAttribute(ROLETYPE,user.getUserType());
                request.setAttribute(TITLE,"Patient Creation Form");
                request.setAttribute(MAIN_PAGE,"/pages/main/create_patient.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}



/**
 * submit the pages according to the roles
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward submitAdminForm(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                UserForm userForm = (UserForm)form;
```

```
                String message = null;

                if(isTokenValid(request)){
                        //typeSubmition = userForm.getType();
                        GenericService service = new GenericQueries();
                        message = service.insertUserDetails(userForm,user);
                }
                request.setAttribute("message", message);
                request.removeAttribute("userForm");
                resetToken(request);

                // navigating to the profiles according to their appropriate roles
                if(userForm.getUserType().equals("admin")){
                        return addAdmin(mapping, form, request, response);
                }else if(userForm.getUserType().equals("patient")){
                        return addPatient(mapping, form, request, response);
                }else if(userForm.getUserType().equals("nurse")){
                        return addNurse(mapping, form, request, response);
                }else if(userForm.getUserType().equals("doctor")){
                        return addDoctor(mapping, form, request, response);
                }
        }
        return null;
    }
/**
```

```
 * Updating the users, For all the modules according to the role basis
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward profile(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");



                request.setAttribute(TITLE, "Profile");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/profile.jsp");


        }
        return mapping.findForward(GLOBAL_CONSTANT);
}


/**
```

```
        * updating the user profiles with respective to the modules
        * @param mapping
        * @param form
        * @param request
        * @param response
        * @return
        * @throws Exception
        */
public ActionForward updateProdile(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {
                if(AdaranSecurity.authenticated(request)){
                        HttpSession session = request.getSession(false);
                        UserForm user = (UserForm)session.getAttribute("user");
                        UserForm userForm = (UserForm)form;
                        String message = null;
                        GenericService service = new GenericQueries();
                        message = service.updateProfileDetails(userForm,user);
                        user = service.getDetailsofProfiles(user);

                        session.setAttribute("user", user);
                        request.setAttribute(TITLE, "Profile");
                        request.setAttribute(ROLETYPE, user.getUserType());
                        request.setAttribute(MAIN_PAGE, "/pages/main/profile.jsp");
```

```
		}
		return mapping.findForward(GLOBAL_CONSTANT);
}

/**
 * Generating the patient details
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward patientDetail(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		if(AdaranSecurity.authenticated(request)){
			HttpSession session = request.getSession(false);
			UserForm user = (UserForm)session.getAttribute("user");
			GenericService service = new GenericQueries();
			UserForm userForm = service.getDetailsofProfiles(user);

			request.setAttribute("userForm", userForm);
			request.setAttribute(TITLE, "Profile");
			request.setAttribute(ROLETYPE, user.getUserType());
```

```
                request.setAttribute(MAIN_PAGE, "/pages/main/patientProfiles.jsp");


        }
        return mapping.findForward(GLOBAL_CONSTANT);
}


/**
 * List of patient profiles
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward patientDetailList(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                GenericService service = new GenericQueries();
                String type = "P";
                List<UserForm> userForms = service.getDetailsofProfiles(type,user);
```

```
                List<UserForm> antForms = service.getDetailsofProfilesANT(type,user);

                request.setAttribute("antForms", antForms);
                request.setAttribute("userForms", userForms);

                request.setAttribute(TITLE, "Profile");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/patientsList.jsp");

        }
        return mapping.findForward(GLOBAL_CONSTANT);
}


/**
 * Patient Details list with doctors prospective
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward patientDetailListforDoc(ActionMapping mapping, ActionForm form,
```

```
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {
                if(AdaranSecurity.authenticated(request)){
                        HttpSession session = request.getSession(false);
                        UserForm user = (UserForm)session.getAttribute("user");
                        GenericService service = new GenericQueries();
                        String type = "P";
                        List<UserForm> userForms =
service.getDetailsofProfilesRelatedToDoc(type,user);

                        List<UserForm> antForms =
service.getDetailsofProfilesReletedtoDocANT(type,user);

                        request.setAttribute("antForms", antForms);
                        request.setAttribute("userForms", userForms);

                        request.setAttribute(TITLE, "Profile");
                        request.setAttribute(ROLETYPE, user.getUserType());
                        request.setAttribute(MAIN_PAGE, "/pages/main/docpatientsList.jsp");

                }
                return mapping.findForward(GLOBAL_CONSTANT);
        }

        /**
         * Overriding the patient details in terms of doctor prospective
```

```
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward overrideDetailListforDoc(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                String patientId = request.getParameter("pid");

                request.setAttribute("patientId", patientId);
                request.setAttribute(TITLE, "Override Message");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/override.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}

/**
 * overriding the admin details in terms of admin prospective
```

```
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward overrideDetailListforAdmin(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                String patientId = request.getParameter("pid");

                request.setAttribute("patientId", patientId);
                request.setAttribute(TITLE, "Override Message");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/adminoverride.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}


/**
```

```
	 * Storing the override details according to the doctro prospective
	 * @param mapping
	 * @param form
	 * @param request
	 * @param response
	 * @return
	 * @throws Exception
	 */
	public ActionForward overrideDetailListforDocsubmition(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		if(AdaranSecurity.authenticated(request)){
			HttpSession session = request.getSession(false);
			UserForm user = (UserForm)session.getAttribute("user");
			GenericService service = new GenericQueries();
			String patientId = request.getParameter("patientId");
			String message = request.getParameter("message");
			service.overrideDetailsofPatientToDoc(patientId,user,message);



			request.setAttribute("patientId", patientId);
		}
		return patientDetailListforDoc(mapping, form, request, response);
	}
```

```
	/**
	 * storing the admin details according to the admin prospective
	 * @param mapping
	 * @param form
	 * @param request
	 * @param response
	 * @return
	 * @throws Exception
	 */
	public ActionForward overrideDetailListforAdminsubmition(ActionMapping mapping,
ActionForm form,
				HttpServletRequest request, HttpServletResponse response)
				throws Exception {
		if(AdaranSecurity.authenticated(request)){
				HttpSession session = request.getSession(false);
				UserForm user = (UserForm)session.getAttribute("user");
				GenericService service = new GenericQueries();
				String patientId = request.getParameter("patientId");
				String message = request.getParameter("message");
				service.overrideDetailsofPatientToDoc(patientId,user,message);



				request.setAttribute("patientId", patientId);
```

```
        }
        return patientDetailList(mapping, form, request, response);
}

//removedoc
/**
 * removing the doctores in patiens list
 */
public ActionForward removedoc(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                String docId = request.getParameter("docId");
                String patientId = request.getParameter("patientId");
                GenericService service = new GenericQueries();
                service.deleteDocFromDOCASSIGN(docId,patientId);
        }
        return patientDetailListforDoc(mapping, form, request, response);
}

//removenurse
/**
 * removing the nurse from patient's assiging
```

```
 */
public ActionForward removenurse(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                String docId = request.getParameter("docId");
                String patientId = request.getParameter("patientId");
                GenericService service = new GenericQueries();
                service.deleteDocFromNURASSIGN(docId,patientId);
        }
        return patientDetailListforDoc(mapping, form, request, response);
}



/**
 * generating the patients details in Nurse assigned In terms of AT & ANT
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
```

```
	public ActionForward patientDetailListforNurse(ActionMapping mapping, ActionForm form,
				HttpServletRequest request, HttpServletResponse response)
				throws Exception {
			if(AdaranSecurity.authenticated(request)){
				HttpSession session = request.getSession(false);
				UserForm user = (UserForm)session.getAttribute("user");
				GenericService service = new GenericQueries();
				String type = "P";
				List<UserForm> userForms =
service.getDetailsofProfilesforNurseRelated(type,user);



				List<UserForm> antForms =
service.getDetailsofProfilesANTNurseRelated(type,user);

				request.setAttribute("antForms", antForms);
				request.setAttribute("userForms", userForms);

				request.setAttribute(TITLE, "Profile");
				request.setAttribute(ROLETYPE, user.getUserType());
				request.setAttribute(MAIN_PAGE, "/pages/main/nursepatientsList.jsp");

			}
			return mapping.findForward(GLOBAL_CONSTANT);
	}
```

```
/**
 * Assigning nurse page generation
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward assignNurse(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response)
                    throws Exception {
          if(AdaranSecurity.authenticated(request)){
                    HttpSession session = request.getSession(false);
                    UserForm user = (UserForm)session.getAttribute("user");
                    GenericService service = new GenericQueries();
                    List<UserForm> nurses = service.getDetailsofDoctorsOrNurses("N");

                    request.setAttribute("nurses", nurses);

                    saveToken(request);
                    request.setAttribute("patientId", request.getParameter("docid"));
                    request.setAttribute(TITLE, "Assign Nurses");
                    request.setAttribute(ROLETYPE, user.getUserType());
```

```
                request.setAttribute(MAIN_PAGE, "/pages/main/assignNurse.jsp");


            }
            return mapping.findForward(GLOBAL_CONSTANT);
        }
/**
         * storing the nurse assigned pages
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return
         * @throws Exception
         */
        public ActionForward assignNurseSubmit(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {
                if(AdaranSecurity.authenticated(request)){
                        HttpSession session = request.getSession(false);


                        UserForm user = (UserForm)session.getAttribute("user");
                        if(isTokenValid(request)){
                        GenericService service = new GenericQueries();
                        String patientId = request.getParameter("patientId");
                        String nurseId = request.getParameter("nurseId");
```

```
                String message = service.assignPatientToNurse(user,patientId,nurseId);


                request.setAttribute("message", message);

                }

                resetToken(request);


        }

        return assignNurse(mapping, form, request, response);

}




/**
 * Assigning the doctors page generation
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward assignDoctor(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
```

```
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                GenericService service = new GenericQueries();
                UserForm userForm = service.getDetailsofProfiles(user);
                List<UserForm> userForms = service.getDetailsofDoctorsOrNurses("D");

                request.setAttribute("userForms", userForms);



                saveToken(request);
                request.setAttribute("patientId", request.getParameter("docid"));
                request.setAttribute("userForm", userForm);
                request.setAttribute(TITLE, "Profile");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/assignDoctor.jsp");

        }
        return mapping.findForward(GLOBAL_CONSTANT);
}

/**
 *
 * @param mapping
 * @param form
 * @param request
```

```
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward assignDoctorSubmit(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);

                UserForm user = (UserForm)session.getAttribute("user");
                if(isTokenValid(request)){
                GenericService service = new GenericQueries();
                String patientId = request.getParameter("patientId");
                String docId = request.getParameter("docId");
                String message = service.assignPatientToDoctor(user,patientId,docId);

                request.setAttribute("message", message);
                }
                resetToken(request);

        }
        return assignDoctor(mapping, form, request, response);
}
```

```
/**
 *
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward patientNotes(ActionMapping mapping, ActionForm form,
                    HttpServletRequest request, HttpServletResponse response)
                    throws Exception {
          if(AdaranSecurity.authenticated(request)){
                    HttpSession session = request.getSession(false);
                    UserForm user = (UserForm)session.getAttribute("user");
                    String patientId = request.getParameter("patientId");
                    GenericService service = new GenericQueries();

                    service.insertNotestDetails(patientId,user);

                    NotesForm notest = service.getNotesFromDatabase(user,patientId);
                    List<NotesForm> publicN= service.getPatientNotespublic(user,patientId);
                    /*List<NotesForm> publicN= service.getPatientNotespublic(user);
                    List<NotesForm> privateN= service.getPatientNotesprivate(user);
                    request.setAttribute("publicN", publicN);
```

```
                request.setAttribute("privateN", privateN);*/
                request.setAttribute("publicN", publicN);
                request.setAttribute("notest", notest);
                saveToken(request);
                request.setAttribute("patientId", patientId);
                request.setAttribute(TITLE, "Adding Notes");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/addnotes.jsp");

            }
            return mapping.findForward(GLOBAL_CONSTANT);
}



/**
 *
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward submitPatinetNotes(ActionMapping mapping, ActionForm form,
```

```
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
            if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);

                UserForm user = (UserForm)session.getAttribute("user");
                if(isTokenValid(request)){
                GenericService service = new GenericQueries();
                String patientId = request.getParameter("patientId");
                String privateN = request.getParameter("private");
                String publicN = request.getParameter("public");
                String message =
service.assignNotestTOPatient(user,patientId,publicN,privateN);

                request.setAttribute("message", message);
                }
                resetToken(request);

            }
            return patientNotes(mapping, form, request, response);
        }


        /**
         *
```

```
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
public ActionForward partialaddnotespatientNotes(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
        if(AdaranSecurity.authenticated(request)){
                HttpSession session = request.getSession(false);
                UserForm user = (UserForm)session.getAttribute("user");
                String patientId = request.getParameter("patientId");
                GenericService service = new GenericQueries();

                service.insertNotestDetails(patientId,user);

                NotesForm notest = service.getNotesFromDatabase(user,patientId);
                request.setAttribute("notest", notest);
                saveToken(request);
                request.setAttribute("patientId", patientId);
                request.setAttribute(TITLE, "Adding Notes");
                request.setAttribute(ROLETYPE, user.getUserType());
                request.setAttribute(MAIN_PAGE, "/pages/main/partialaddnotes.jsp");
```

```
                }
                return mapping.findForward(GLOBAL_CONSTANT);
        }


/**
 * partial notes storing
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
 */
        public ActionForward submitpartialaddnotesPatinetNotes(ActionMapping mapping, ActionForm
form,
                        HttpServletRequest request, HttpServletResponse response)
                        throws Exception {
                if(AdaranSecurity.authenticated(request)){
                        HttpSession session = request.getSession(false);


                        UserForm user = (UserForm)session.getAttribute("user");
                        if(isTokenValid(request)){
                        GenericService service = new GenericQueries();
```

```
                        String patientId = request.getParameter("patientId");

                        String privateN = request.getParameter("private");

                        String publicN = "";

                        String message =
service.assignNotestTOPatient(user,patientId,publicN,privateN);


                        request.setAttribute("message", message);

                        }

                        resetToken(request);


                }

                return patientNotes(mapping, form, request, response);

        }


        /**
         * Notification List
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return
         * @throws Exception
         */
        public ActionForward notificationsList(ActionMapping mapping, ActionForm form,

                        HttpServletRequest request, HttpServletResponse response)
```

```
            throws Exception {
        if(AdaranSecurity.authenticated(request)){
            HttpSession session = request.getSession(false);

            UserForm user = (UserForm)session.getAttribute("user");

            GenericService service = new GenericQueries();
            List<NotificationForm> notificationForms = service.getNotificationsList(user);

            request.setAttribute("notificationForms", notificationForms);
            request.setAttribute(TITLE, "Notifications");
            request.setAttribute(ROLETYPE, user.getUserType());
            request.setAttribute(MAIN_PAGE, "/pages/main/notifications.jsp");
        }
        return mapping.findForward(GLOBAL_CONSTANT);
}

/**
 * Notifications check list
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 * @throws Exception
```

```
 */
	public ActionForward notificationsChecked(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		if(AdaranSecurity.authenticated(request)){
			HttpSession session = request.getSession(false);

			UserForm user = (UserForm)session.getAttribute("user");

			GenericService service = new GenericQueries();
			String nId = request.getParameter("id");
			service.updatenotificationStatus(nId,user);


		}
		return notificationsList(mapping, form, request, response);
	}

}
```

Sequence Generation algorithms

```
package org.hospital.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

/**
 *
 * The Class GENERATEDIDS performs the sequence.
* SUPER ADMIN  - SA - 000000 {FIRST TWO CHARACTERS FROM THE USERNAME}
 * ADMIN        - SA - 000000 {FIRST TWO CHARACTERS FROM THE USERNAME}
 * DOCTOR       - DR - 000000 {FIRST TWO CHARACTERS FROM THE USERNAME}
 * NURSE        - NR - 000000 {FIRST TWO CHARACTERS FROM THE USERNAME}
 * PATIENT      - PT - 000000 {FIRST TWO CHARACTERS FROM THE USERNAME}
 *
 * @author Koushik
 *
 */
public class GenerateIds {

        /**
         * Generating sequence for Nurse
```

```
* @param con
* @return
* @throws Exception
*/
public synchronized static String getNurserId(Connection con) throws Exception {
	String id = "";
	if (con != null) {
		try {
			String sql = "SELECT NURSE_ID FROM sequence";
			PreparedStatement st = con.prepareStatement(sql);
			ResultSet res = st.executeQuery();
			String tempId;
			if (res.next()) {
				tempId = res.getString("NURSE_ID");
			} else {
				throw new Exception();
			}
			System.out.println("The Temp Id is :: " +tempId);

			res.close();
			st.close();
			int num = Integer.parseInt(tempId.substring(2));
			if (num == 999999) {
				id = "NR0000001";
			} else {
```

```
                    num++;
                    String temp1 = num + "";
                    if (temp1.length() < 6) {
                        int s = temp1.length();
                        for (int i = 0; i < (6 - s); i++) {
                            temp1 = "0" + temp1;
                        }
                    }
                    id = tempId.substring(0, 2) + temp1;
                }
                sql = "UPDATE sequence SET NURSE_ID='" + id + "'";
                Statement st1 = con.createStatement();
                st1.executeUpdate(sql);
                st1.close();
            } catch (Exception e) {
                e.printStackTrace();
                throw e;
            }
        }
        return id;
}


public synchronized static String getPatientId(Connection con) throws Exception {
        String id = "";
```

```
if (con != null) {
        try {
                String sql = "SELECT PATIENT_ID FROM sequence";
                PreparedStatement st = con.prepareStatement(sql);
                ResultSet res = st.executeQuery();
                String tempId;
                if (res.next()) {
                        tempId = res.getString("PATIENT_ID");
                } else {
                        throw new Exception();
                }
                System.out.println("The Temp Id is :: " +tempId);

                res.close();
                st.close();
                int num = Integer.parseInt(tempId.substring(2));
                if (num == 999999) {
                        id = "PT0000001";
                } else {
                        num++;
                        String temp1 = num + "";
                        if (temp1.length() < 6) {
                                int s = temp1.length();
                                for (int i = 0; i < (6 - s); i++) {
                                        temp1 = "0" + temp1;
```

```
                        }
                    }
                    id = tempId.substring(0, 2) + temp1;
                }
                sql = "UPDATE sequence SET PATIENT_ID='" + id + "'";
                Statement st1 = con.createStatement();
                st1.executeUpdate(sql);
                st1.close();
            } catch (Exception e) {
                e.printStackTrace();
                throw e;
            }
        }
        return id;
}



public synchronized static String getDoctorId(Connection con) throws Exception {
        String id = "";
        if (con != null) {
            try {
                String sql = "SELECT DOC_ID FROM sequence";
                PreparedStatement st = con.prepareStatement(sql);
                ResultSet res = st.executeQuery();
```

```
String tempId;
if (res.next()) {
        tempId = res.getString("DOC_ID");
} else {
        throw new Exception();
}
System.out.println("The Temp Id is :: " +tempId);


res.close();
st.close();
int num = Integer.parseInt(tempId.substring(2));
if (num == 999999) {
        id = "DR0000001";
} else {
        num++;
        String temp1 = num + "";
        if (temp1.length() < 6) {
                int s = temp1.length();
                for (int i = 0; i < (6 - s); i++) {
                        temp1 = "0" + temp1;
                }
        }
        id = tempId.substring(0, 2) + temp1;
}
sql = "UPDATE sequence SET DOC_ID='" + id + "'";
```

```
                Statement st1 = con.createStatement();
                st1.executeUpdate(sql);
                st1.close();
            } catch (Exception e) {
                e.printStackTrace();
                throw e;
            }
        }
        return id;
}


public synchronized static String getAdminId(Connection con) throws Exception {
        String id = "";
        if (con != null) {
            try {
                String sql = "SELECT ADMIN_ID FROM sequence";
                PreparedStatement st = con.prepareStatement(sql);
                ResultSet res = st.executeQuery();
                String tempId;
                if (res.next()) {
                    tempId = res.getString("ADMIN_ID");
                } else {
                    throw new Exception();
                }
```

```
            System.out.println("The Temp Id is :: " +tempId);


            res.close();
            st.close();
            int num = Integer.parseInt(tempId.substring(2));
            if (num == 999999) {
                  id = "AD0000001";
            } else {
                  num++;
                  String temp1 = num + "";
                  if (temp1.length() < 6) {
                        int s = temp1.length();
                        for (int i = 0; i < (6 - s); i++) {
                              temp1 = "0" + temp1;
                        }
                  }
                  id = tempId.substring(0, 2) + temp1;
            }
            sql = "UPDATE sequence SET ADMIN_ID='" + id + "'";
            Statement st1 = con.createStatement();
            st1.executeUpdate(sql);
            st1.close();
      } catch (Exception e) {
            e.printStackTrace();
            throw e;
```

```
                }
            }
            return id;
        }
}
```

Queries or DAO layer

```
package org.hospital.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.hospital.service.GenericService;
import org.hospital.utils.CommonConstants;
import org.hospital.utils.ConnectionPool;
import org.hospital.utils.PasswordGenerator;
import org.hospital.utils.QueryConstants;
import org.hospital.utils.SendMailUsingAuthenticationwithattachment;
import org.hospitals.forms.NotesForm;
import org.hospitals.forms.NotificationForm;
```

```
import org.hospitals.forms.UserForm;



/**
 * All the persistance operations will performed here.
 *
 * {Creation, retrieving, editing, Updating, deleting }
 * For recording purpose all the deleted data will be changed the status id to delete
 * Statusmasterids as enums
 * 1 - Active
 * 2 - Delete
 * For logged in DAO
 * @author Koushik
 *
 */
public class GenericQueries implements CommonConstants,GenericService{

	private PreparedStatement psmt = null;
	private ResultSet rsmt = null;
	int STATEMENTCOUNT = 0;
	/**
	 * Checks the USer weather he used correct USERID or NOT
	 * @param id
	 * @param password
	 * @return
```

```
     * @throws Exception
     */
    public static UserForm authenticate(String id, String password)
                    throws Exception {
                    Connection con = ConnectionPool.getConnection();
                        if (con != null) {
                            String sql1= null;
                            try {
                                sql1 =
QueryConstants.AUTHENTICATE_USER_TYPE_FOM_HOSPITAL_LOGIN;
                                PreparedStatement st1 = con.prepareStatement(sql1);
                                st1.setString(1, id);
                                st1.setString(2, password);
                                ResultSet res = st1.executeQuery();
                                UserForm user = null;
                                if (res.next()) {
                                    String userType = res.getString("TYPE");

    user=(UserForm)getRecordDetails(id,con,res,st1);
                                        user.setUserType(userType);
                                        user.setLoggedIn(true);
                            }
                        res.close();
                        st1.close();
                        return user;
                    } catch (SQLException e) {
```

```
                    e.printStackTrace();
                } finally {
                    con.close();
                }
            }
            return null;
        }

        /**
         * get the details
         */
        private static UserForm getRecordDetails(String id,Connection con, ResultSet rsmt,
PreparedStatement psmt) throws Exception{
            String sql = null;
            UserForm userForm = null;
            try{
                sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERID = ?";
                psmt = con.prepareStatement(sql);
                psmt.setString(1, id);
                rsmt = psmt.executeQuery();
                if(rsmt.next()){
                    userForm = new UserForm();
                    userForm.setDob(rsmt.getString("DOB"));
                    userForm.setEmail(rsmt.getString("EMAIL"));
```

```
                    userForm.setFirstName(rsmt.getString("FIRST_NAME"));
                    userForm.setLastName(rsmt.getString("LAST_NAME"));
                    userForm.setMobileNo(rsmt.getString("MOBILE"));
                    userForm.setUserId(rsmt.getString("USERID"));
                    userForm.setUserType(rsmt.getString("USERTYPE"));
                    userForm.setGender(rsmt.getString("GENDER"));
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }

        return userForm;
}

/**
 * Inserting the users and admins
 */
public String insertUserDetails(UserForm userForm, UserForm user) throws Exception {
        Connection connection = null;
        String message = null;
        String sql = null;
        String userId = null;
        String userType = null;
```

```
STATEMENTCOUNT++;
String password = null;
try{
    connection = ConnectionPool.getConnection();
    if(userForm.getUserType().equals("admin")){
        userId = GenerateIds.getAdminId(connection);
        userType = "A";
    }else if(userForm.getUserType().equals("patient")){
        userId = GenerateIds.getPatientId(connection);
        userType = "P";
    }else if(userForm.getUserType().equals("nurse")){
        userId = GenerateIds.getNurserId(connection);
        userType = "N";
    }else if(userForm.getUserType().equals("doctor")){
        userId = GenerateIds.getDoctorId(connection);
        userType = "D";
    }
    password = PasswordGenerator.getNext();
    sql = "INSERT INTO
`login_all`(`ID`,`PASSWORD`,`TYPE`,`STATUS`)VALUES (?,?,?,?)";
    psmt = connection.prepareStatement(sql);
    psmt.setString(STATEMENTCOUNT++, userId);
    psmt.setString(STATEMENTCOUNT++, password);
    psmt.setString(STATEMENTCOUNT++, userType);
    psmt.setInt(STATEMENTCOUNT++, ACTIVE);
```

```
                    psmt.executeUpdate();


                    if(userType.equalsIgnoreCase("P")){
                        STATEMENTCOUNT = 1;
                        sql = "INSERT INTO
`patient_details`(`USERID`,`ADDRESS`,`INSURANCE`,`POLOCIES`,`WARD`,`ROOM`,`SSN`,`CRE
ATEDDATE`) VALUES (?,?,?,?,?,?,?,?)";
                        psmt = connection.prepareStatement(sql);
                        psmt.setString(STATEMENTCOUNT++, userId);
                        psmt.setString(STATEMENTCOUNT++, userForm.getAddress());
                        psmt.setString(STATEMENTCOUNT++,
userForm.getInsuranceName());
                        psmt.setString(STATEMENTCOUNT++, userForm.getPolicies());
                        psmt.setString(STATEMENTCOUNT++, userForm.getWard());
                        psmt.setString(STATEMENTCOUNT++, userForm.getRoomNo());
                        psmt.setString(STATEMENTCOUNT++, userForm.getSSN());
                        psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));
                        psmt.executeUpdate();


                        if(userForm.getDocId() != null){
                            STATEMENTCOUNT = 1;
                            sql = "INSERT INTO
`doc_assigned`(`USERID`,`DOC_ASSIGEND`,`CREATEDDATE`,`STATUS`)VALUES (?,?,?,?)";
                            psmt = connection.prepareStatement(sql);
                            psmt.setString(STATEMENTCOUNT++, userId);
                            psmt.setString(STATEMENTCOUNT++, userForm.getDocId());
```

```
                                        psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));

                                        psmt.setInt(STATEMENTCOUNT++, ACTIVE);

                                        psmt.executeUpdate();

                                }


                                if(userForm.getNurseId() != null){

                                        STATEMENTCOUNT = 1;

                                        sql = "INSERT INTO
`nurse_assigned`(`USERID`,`NURSER_ASSIGNED`,`CREATEDDATE`,`STATUS`)VALUES
(?,?,?,?)";

                                        psmt = connection.prepareStatement(sql);

                                        psmt.setString(STATEMENTCOUNT++, userId);

                                        psmt.setString(STATEMENTCOUNT++,
userForm.getNurseId());

                                        psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));

                                        psmt.setInt(STATEMENTCOUNT++, ACTIVE);

                                        psmt.executeUpdate();

                                }

                        }


                        sql = "INSERT INTO
`user_details`(`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`U
SERTYPE`,`STATUSMASTER`,`CREATIONBY`)VALUES (?,?,?,?,?,?,?,?,?,?)";

                        STATEMENTCOUNT = 1;

                        psmt = connection.prepareStatement(sql);

                        psmt.setString(STATEMENTCOUNT++, userId);
```

```
                psmt.setString(STATEMENTCOUNT++, userForm.getFirstName());
                psmt.setString(STATEMENTCOUNT++, userForm.getLastName());
                psmt.setString(STATEMENTCOUNT++, userForm.getDob());
                psmt.setString(STATEMENTCOUNT++, userForm.getEmail());
                psmt.setString(STATEMENTCOUNT++, userForm.getMobileNo());
                psmt.setString(STATEMENTCOUNT++, userForm.getGender());
                psmt.setString(STATEMENTCOUNT++, userType);
                psmt.setInt(STATEMENTCOUNT++, ACTIVE);
                psmt.setString(STATEMENTCOUNT++, user.getUserId());
                int n = psmt.executeUpdate();
                if(n !=0){
                        message = "Successfully registered! User name is "+userId+" and
password is "+password+".      " +
                                        "<button type=\"button\" class=\"btn btn-success\"
onclick=\"javascript:window.location='home.htm'\">Cancel</button>";
                }else{
                        message = "some internal problem occeur, Please try again later";
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeConnection(psmt, connection);
        }
```

```
		return message;
	}



	/**
	 * Updateing the user profile details
	 */
	public String updateProfileDetails(UserForm userForm, UserForm user)
				throws Exception {
		Connection connection = null;
		String message = null;
		String sql = null;
		STATEMENTCOUNT =1;
		try{
			connection = ConnectionPool.getConnection();
			sql = "UPDATE `user_details` SET `EMAIL` = ?,`MOBILE` = ? where
`USERID` = ?";
			psmt = connection.prepareStatement(sql);


			psmt.setString(STATEMENTCOUNT++, userForm.getEmail());
			psmt.setString(STATEMENTCOUNT++, userForm.getMobileNo());
			psmt.setString(STATEMENTCOUNT++, user.getUserId());
			int n = psmt.executeUpdate();
			if(n !=0){
```

```
                    message = "Profile Update Successfully! ";
                }else{
                    message = "some internal problem occeur, Please try again later";
                }
            }catch(SQLException e){
                throw e;
            }catch(Exception e){
                throw e;
            }finally{
                ConnectionPool.closeConnection(psmt, connection);
            }



            return message;
    }

    /**
     * get details of users
     */
    public UserForm getDetailsofProfiles(UserForm user) throws Exception {
            Connection connection = null;
            String sql = null;
            STATEMENTCOUNT = 1;
            UserForm userForm = null;
            try{
```

```
                connection = ConnectionPool.getConnection();

                sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERID = ?";

                psmt = connection.prepareStatement(sql);

                psmt.setString(STATEMENTCOUNT++, user.getUserId());

                rsmt = psmt.executeQuery();

                if(rsmt.next()){

                        userForm = new UserForm();

                        userForm.setDob(rsmt.getString("DOB"));

                        userForm.setEmail(rsmt.getString("EMAIL"));

                        userForm.setFirstName(rsmt.getString("FIRST_NAME"));

                        userForm.setLastName(rsmt.getString("LAST_NAME"));

                        userForm.setMobileNo(rsmt.getString("MOBILE"));

                        userForm.setUserId(rsmt.getString("USERID"));

                        userForm.setUserType(rsmt.getString("USERTYPE"));

                        userForm.setGender(rsmt.getString("GENDER"));

                }

        }catch(SQLException e){

                throw e;

        }catch(Exception e){

                throw e;

        }finally{

                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);

        }
```

```
		return userForm;
	}

	/**
	 * profile details with entity List generation
	 */
	public List<UserForm> getDetailsofProfiles(String type,UserForm user) throws Exception {
		Connection connection = null;
		String sql = null;
		STATEMENTCOUNT = 1;
		UserForm userForm = null;
		List<UserForm> userForms = null;
		try{
			userForms = new ArrayList<UserForm>();
			connection = ConnectionPool.getConnection();
			sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERTYPE = ? AND CREATIONBY = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(STATEMENTCOUNT++, type);
			psmt.setString(STATEMENTCOUNT++, user.getUserId());
			rsmt = psmt.executeQuery();
			while(rsmt.next()){
				userForm = new UserForm();
				userForm.setDob(rsmt.getString("DOB"));
```

```
                        userForm.setEmail(rsmt.getString("EMAIL"));
                        userForm.setFirstName(rsmt.getString("FIRST_NAME"));
                        userForm.setLastName(rsmt.getString("LAST_NAME"));
                        userForm.setMobileNo(rsmt.getString("MOBILE"));
                        userForm.setUserId(rsmt.getString("USERID"));
                        userForm.setUserType(rsmt.getString("USERTYPE"));
                        userForm.setGender(rsmt.getString("GENDER"));
                        userForms.add(userForm);
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
        }
        return userForms;
}

/**
 * Doctors and Nurses List generation
 */
public List<UserForm> getDetailsofDoctorsOrNurses(String type) throws Exception {
        Connection connection = null;
        String sql = null;
```

```
		STATEMENTCOUNT = 1;
		UserForm userForm = null;
		List<UserForm> userForms = null;
		try{
			userForms = new ArrayList<UserForm>();
			connection = ConnectionPool.getConnection();
			sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERTYPE = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(STATEMENTCOUNT++, type);
			rsmt = psmt.executeQuery();
			while(rsmt.next()){
				userForm = new UserForm();
				userForm.setDob(rsmt.getString("DOB"));
				userForm.setEmail(rsmt.getString("EMAIL"));
				userForm.setFirstName(rsmt.getString("FIRST_NAME"));
				userForm.setLastName(rsmt.getString("LAST_NAME"));
				userForm.setMobileNo(rsmt.getString("MOBILE"));
				userForm.setUserId(rsmt.getString("USERID"));
				userForm.setUserType(rsmt.getString("USERTYPE"));
				userForm.setGender(rsmt.getString("GENDER"));
				userForms.add(userForm);
			}
		}catch(SQLException e){
			throw e;
```

```
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
		}
		return userForms;
	}

	/**
	 * ANT profile details of Patients
	 */
	public List<UserForm> getDetailsofProfilesANT(String type, UserForm user)throws Exception {
		Connection connection = null;
		String sql = null;
		STATEMENTCOUNT = 1;
		UserForm userForm = null;
		List<UserForm> userForms = null;
		try{
			userForms = new ArrayList<UserForm>();
			connection = ConnectionPool.getConnection();
			sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERTYPE = ? AND CREATIONBY != ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(STATEMENTCOUNT++, type);
			psmt.setString(STATEMENTCOUNT++, user.getUserId());
```

```
                rsmt = psmt.executeQuery();
                while(rsmt.next()){
                        userForm = new UserForm();
                        userForm.setDob(rsmt.getString("DOB"));
                        userForm.setEmail(rsmt.getString("EMAIL"));
                        userForm.setFirstName(rsmt.getString("FIRST_NAME"));
                        userForm.setLastName(rsmt.getString("LAST_NAME"));
                        userForm.setMobileNo(rsmt.getString("MOBILE"));
                        userForm.setUserId(rsmt.getString("USERID"));
                        userForm.setUserType(rsmt.getString("USERTYPE"));
                        userForm.setGender(rsmt.getString("GENDER"));
                        userForms.add(userForm);
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
        }
        return userForms;
}

/**
 * Generating the doc Name
```

```
	* @param connection
	* @param psmt
	* @param rsmt
	* @param Id
	* @return
	* @throws Exception
	*/
	public static String getDocName(Connection connection, PreparedStatement psmt,ResultSet rsmt,String Id) throws Exception {
		String sql = null;
		String name = "";
		try{

			sql = "SELECT `FIRST_NAME`,`LAST_NAME` FROM user_details WHERE USERID = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, Id);
			rsmt = psmt.executeQuery();
			if(rsmt.next()){
				name = rsmt.getString("FIRST_NAME")+" "+rsmt.getString("LAST_NAME");
			}
		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
```

```
		}

		return name;

	}



	/**

	 * assigning patient to nurses

	 */

	public String assignPatientToNurse(UserForm user, String patientId,String nurseId) throws
Exception {

		Connection connection = null;

		String message = null;

		String sql = null;

		STATEMENTCOUNT = 1;



		try{

			connection = ConnectionPool.getConnection();



			sql = "INSERT INTO
`nurse_assigned`(`USERID`,`NURSER_ASSIGNED`,`CREATEDDATE`,`STATUS`)VALUES
(?,?,?,?)";

			psmt = connection.prepareStatement(sql);

			psmt.setString(STATEMENTCOUNT++, patientId);

			psmt.setString(STATEMENTCOUNT++, nurseId );

			psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));
```

```
                psmt.setInt(STATEMENTCOUNT++, ACTIVE);
                psmt.executeUpdate();
                sql = "INSERT INTO `notification`
(`NOTIFICATION`,`FROMID`,`TOID`,`STATUS`,`PID`)VALUES (?,?,?,?,?)";
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, "Patient Profile");
                psmt.setString(2, user.getUserId());
                psmt.setString(3, nurseId);
                psmt.setString(4, "1");
                psmt.setString(5, patientId);
                int n = psmt.executeUpdate();
                if(n !=0){
                        message = "Successfully Assigned.";
                }else{
                        message = "some internal problem occeur, Please try again later";
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeConnection(psmt, connection);
        }
```

```
		return message;
	}

	/**
	 * assign patient to doctors
	 */
	public String assignPatientToDoctor(UserForm user, String patientId,String docId) throws
Exception {
		Connection connection = null;
		String message = null;
		String sql = null;
		STATEMENTCOUNT = 1;

		try{
			connection = ConnectionPool.getConnection();

			sql = "INSERT INTO
`doc_assigned`(`USERID`,`DOC_ASSIGEND`,`CREATEDDATE`,`STATUS`)VALUES (?,?,?,?)";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, patientId);
			psmt.setString(2, docId );
			psmt.setTimestamp(3, new Timestamp(System.currentTimeMillis()));
			psmt.setInt(4, ACTIVE);
			psmt.executeUpdate();
			sql = "INSERT INTO `notification`
(`NOTIFICATION`,`FROMID`,`TOID`,`STATUS`,`PID`)VALUES (?,?,?,?,?)";
```

```
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, "Patient Profile");
                psmt.setString(2, user.getUserId());
                psmt.setString(3, docId);
                psmt.setString(4, "1");
                psmt.setString(5, patientId);
                int n = psmt.executeUpdate();
                if(n !=0){
                        message = "Successfully Assigned.";
                }else{
                        message = "some internal problem occeur, Please try again later";
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeConnection(psmt, connection);
        }



        return message;
}
```

```
	/**
	 * Adding the notes to patients from Nurses
	 */
	public String assignNotestTOPatient(UserForm user, String patientId,String publicN, String privateN) throws Exception {
		Connection connection = null;
		String message = null;
		String sql = null;
		STATEMENTCOUNT = 1;

		try{
			connection = ConnectionPool.getConnection();

			sql = "UPDATE notes SET PRIVATE = ?, PUBLIC= ? WHERE USERID = ? AND CREATEDBY = ?";
			psmt = connection.prepareStatement(sql);

			psmt.setString(STATEMENTCOUNT++, privateN );
			psmt.setString(STATEMENTCOUNT++, publicN );
			psmt.setString(STATEMENTCOUNT++, patientId);
			psmt.setString(STATEMENTCOUNT++, user.getUserId() );

			int n = psmt.executeUpdate();
			if(n !=0){
				message = "Notes Added Successfully.";
			}else{
```

```
                        message = "some internal problem occeur, Please try again later";
                }
            }catch(SQLException e){
                throw e;
            }catch(Exception e){
                throw e;
            }finally{
                ConnectionPool.closeConnection(psmt, connection);
            }



            return message;
        }

        //SELECT `USERID`,`PRIVATE`,`PUBLIC`,`CREATEDBY`,`STATUS`,`DATE` FROM
`notes` WHERE CREATEDBY = ?
        /**
         * generating public and private notes for users
         */
        public List<NotesForm> getPatientNotespublic(UserForm user, String patientId)throws
Exception {
            Connection connection = null;
            String sql = null;
            NotesForm notesForm = null;
            List<NotesForm> notesForms = null;
            try{
```

```
                notesForms = new ArrayList<NotesForm>();
                connection = ConnectionPool.getConnection();
                sql = "SELECT `PRIVATE`,`PUBLIC` FROM `notes` WHERE USERID = ?";
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, patientId);
                rsmt = psmt.executeQuery();
                while(rsmt.next()){
                        notesForm = new NotesForm();
                        notesForm.setPrivateN(rsmt.getString("PRIVATE"));
                        notesForm.setPublicN(rsmt.getString("PUBLIC"));
                        notesForms.add(notesForm);
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
        }
        return notesForms;
}

/**
 * generating the patient private Notes
 */
```

```
public List<NotesForm> getPatientNotesprivate(UserForm user)throws Exception {
        Connection connection = null;
        String sql = null;
        STATEMENTCOUNT = 1;
        NotesForm notesForm = null;
        List<NotesForm> notesForms = null;
        try{
                notesForms = new ArrayList<NotesForm>();
                connection = ConnectionPool.getConnection();
                sql = "SELECT `PRIVATE`,`PUBLIC` FROM `notes` WHERE CREATEDBY
!= ?";
                psmt = connection.prepareStatement(sql);
                psmt.setString(STATEMENTCOUNT++, user.getUserId());
                rsmt = psmt.executeQuery();
                while(rsmt.next()){
                        notesForm = new NotesForm();
                        notesForm.setPrivateN(rsmt.getString("PRIVATE"));
                        notesForm.setPublicN(rsmt.getString("PUBLIC"));
                        notesForms.add(notesForm);
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
```

```
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
            }
            return notesForms;
        }



        /**
         * generating the patient profiles which was added by appropriate doctors
         */
        public List<UserForm> getDetailsofProfilesRelatedToDoc(String type,UserForm user) throws
Exception {
            Connection connection = null;
            String sql = null;
            STATEMENTCOUNT = 1;
            UserForm userForm = null;
            List<UserForm> userForms = null;
            List<String> strings = null;
            try{
                userForms = new ArrayList<UserForm>();
                strings = new ArrayList<String>();
                connection = ConnectionPool.getConnection();
                sql = "SELECT DISTINCT `USERID` FROM `doc_assigned` WHERE
DOC_ASSIGEND = ?";
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, user.getUserId());
                rsmt = psmt.executeQuery();
```

```
                while(rsmt.next()){
                        strings.add(rsmt.getString("USERID"));
                }
                for (String string : strings) {
                        userForm = new UserForm();
                        userForm.setUserId(string);
                        userForm = getDetailsofProfiles(userForm);
                        userForms.add(userForm);
                }
        }catch(SQLException e){
                System.out.println(e.getMessage());
        }catch(Exception e){
                System.out.println(e.getMessage());
        }finally{
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
        }
        return userForms;
    }



    /**
     * getDetailsofProfilesReletedtoDocANT
     */
    public List<UserForm> getDetailsofProfilesReletedtoDocANT(String type,UserForm user)
throws Exception {
```

```
		Connection connection = null;
		String sql = null;
		STATEMENTCOUNT = 1;
		UserForm userForm = null;
		List<UserForm> userForms = null;
		List<String> strings = null;
		try{
			userForms = new ArrayList<UserForm>();
			strings = new ArrayList<String>();
			connection = ConnectionPool.getConnection();
			sql = "SELECT DISTINCT `USERID` FROM `doc_assigned` WHERE
DOC_ASSIGEND != ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, user.getUserId());
			rsmt = psmt.executeQuery();
			while(rsmt.next()){
				strings.add(rsmt.getString("USERID"));
			}
			for (String string : strings) {
				userForm = new UserForm();
				userForm.setUserId(string);
				userForm = getDetailsofProfiles(userForm);
				userForms.add(userForm);
			}
		}catch(SQLException e){
```

```
                System.out.println(e.getMessage());
            }catch(Exception e){
                System.out.println(e.getMessage());
            }finally{
                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
            }
            return userForms;
        }

        /**
         * getDetailsofProfilesforNurseRelated
         */
        public List<UserForm> getDetailsofProfilesforNurseRelated(String type,UserForm user) throws
Exception {
            Connection connection = null;
            String sql = null;
            STATEMENTCOUNT = 1;
            UserForm userForm = null;
            List<UserForm> userForms = null;
            try{
                userForms = new ArrayList<UserForm>();
                connection = ConnectionPool.getConnection();
                sql = "SELECT DISTINCT `USERID` FROM `nurse_assigned` WHERE
NURSER_ASSIGNED = ?";
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, user.getUserId());
```

```
                rsmt = psmt.executeQuery();
                while(rsmt.next()){


                        sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERID = ?";
                        psmt = connection.prepareStatement(sql);
                        psmt.setString(STATEMENTCOUNT++, rsmt.getString("USERID"));
                        rsmt = psmt.executeQuery();
                        if(rsmt.next()){
                                userForm = new UserForm();
                                userForm.setDob(rsmt.getString("DOB"));
                                userForm.setEmail(rsmt.getString("EMAIL"));
                                userForm.setFirstName(rsmt.getString("FIRST_NAME"));
                                userForm.setLastName(rsmt.getString("LAST_NAME"));
                                userForm.setMobileNo(rsmt.getString("MOBILE"));
                                userForm.setUserId(rsmt.getString("USERID"));
                                userForm.setUserType(rsmt.getString("USERTYPE"));
                                userForm.setGender(rsmt.getString("GENDER"));
                                userForms.add(userForm);
                        }
                }
        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
```

```
		}finally{
			ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
		}
		return userForms;
	}



	/**
	 * getDetailsofProfilesANTNurseRelated
	 */
	public List<UserForm> getDetailsofProfilesANTNurseRelated(String type,UserForm user)
throws Exception {
		Connection connection = null;
		String sql = null;
		STATEMENTCOUNT = 1;
		UserForm userForm = null;
		List<UserForm> userForms = null;
		try{
			userForms = new ArrayList<UserForm>();
			connection = ConnectionPool.getConnection();
			sql = "SELECT DISTINCT `USERID` FROM `nurse_assigned` WHERE
NURSER_ASSIGNED != ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, user.getUserId());
			rsmt = psmt.executeQuery();
			while(rsmt.next()){
```

```
                    sql = "SELECT
`USERID`,`FIRST_NAME`,`LAST_NAME`,`DOB`,`EMAIL`,`MOBILE`,`GENDER`,`USERTYPE`
FROM user_details WHERE USERID = ?";

                    psmt = connection.prepareStatement(sql);

                    psmt.setString(STATEMENTCOUNT++, rsmt.getString("USERID"));

                    rsmt = psmt.executeQuery();

                    if(rsmt.next()){

                        userForm = new UserForm();

                        userForm.setDob(rsmt.getString("DOB"));

                        userForm.setEmail(rsmt.getString("EMAIL"));

                        userForm.setFirstName(rsmt.getString("FIRST_NAME"));

                        userForm.setLastName(rsmt.getString("LAST_NAME"));

                        userForm.setMobileNo(rsmt.getString("MOBILE"));

                        userForm.setUserId(rsmt.getString("USERID"));

                        userForm.setUserType(rsmt.getString("USERTYPE"));

                        userForm.setGender(rsmt.getString("GENDER"));

                        userForms.add(userForm);

                    }

                }

            }catch(SQLException e){

                throw e;

            }catch(Exception e){

                throw e;

            }finally{

                ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
```

```
		}
		return userForms;
	}



	/**
	 * Insert Notes to the patient
	 */
	public void insertNotestDetails(String patientId, UserForm user)throws Exception {
		Connection connection = null;
		String sql = null;
		STATEMENTCOUNT++;

		try{
			connection = ConnectionPool.getConnection();
			sql = "SELECT USERID FROM notes WHERE USERID = ? AND
CREATEDBY = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, patientId);
			psmt.setString(2, user.getUserId() );
			rsmt = psmt.executeQuery();
			if(rsmt.next()){

			}else{
				sql = "INSERT INTO
`notes`(`USERID`,`CREATEDBY`,`STATUS`,`DATE`)VALUES (?,?,?,?)";
```

```
                        psmt = connection.prepareStatement(sql);
                        psmt.setString(STATEMENTCOUNT++, patientId);
                        psmt.setString(STATEMENTCOUNT++, user.getUserId() );
                        psmt.setInt(STATEMENTCOUNT++, ACTIVE);
                        psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));
                        psmt.executeUpdate();
                    }
                }catch(SQLException e){
                    throw e;
                }catch(Exception e){
                    throw e;
                }finally{
                    ConnectionPool.closeConnection(psmt, connection);
                }

        }



        /**
         * Generating the notes details
         */
        public NotesForm getNotesFromDatabase(UserForm user, String patientId)throws Exception {
            Connection connection = null;
            String sql = null;
```

```
		NotesForm form = null;
		try{
			connection = ConnectionPool.getConnection();
			sql = "SELECT PRIVATE,PUBLIC FROM notes WHERE USERID = ? AND
CREATEDBY = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, patientId);
			psmt.setString(2, user.getUserId() );
			rsmt = psmt.executeQuery();
			if(rsmt.next()){
				form =  new NotesForm();
				form.setPrivateN(rsmt.getString("PRIVATE"));
				form.setPublicN(rsmt.getString("PUBLIC"));
			}
		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeConnection(psmt, connection);
		}
		return form;

	}
```

```
	/**
	 * DOC ASSIGN deleteDocFromDOCASSIGN
	 */
	public void deleteDocFromDOCASSIGN(String docId,String patientId) throws Exception {
		Connection connection = null;
		String sql = null;
		try{
			connection = ConnectionPool.getConnection();
			sql = "DELETE FROM doc_assigned WHERE DOC_ASSIGEND = ? AND
USERID = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, docId);
			psmt.setString(2, patientId);
			psmt.executeUpdate();

		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeConnection(psmt, connection);
		}

	}
```

```
	/**
	 * deleteDocFromNURASSIGN
	 */
	public void deleteDocFromNURASSIGN(String docId, String patientId)throws Exception {
		Connection connection = null;
		String sql = null;
		try{
			connection = ConnectionPool.getConnection();
			sql = "DELETE FROM nurse_assigned WHERE `USERID` = ?AND `NURSER_ASSIGNED` = ?";
			psmt = connection.prepareStatement(sql);
			psmt.setString(1, patientId);
			psmt.setString(2, docId);
			psmt.executeUpdate();

		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeConnection(psmt, connection);
		}

	}
```

```
	/**
	 * sendMailAuthentication
	 * @param patternId
	 * @param user
	 */
	public void sendMailAuthentication(String patternId, UserForm user){
		String subject = "Privacy update";


		String message = "Doctor A has accessed the information for Disease
"+user.getFirstName()+" "+user.getLastName()+". Reason: ";
			message+="I'm sorry. no need for reason here";
			message+="The reason should be included during over ride situations. we shall
come to it later";
			Connection connection = null;
			String filenames[]=null;
			String sql = null;
			SendMailUsingAuthenticationwithattachment mail = new
SendMailUsingAuthenticationwithattachment();
			String code1 = "";
			String code2 = "";
			String userId = null;
			String mailStatus = null;
				try
				{
```

```
connection = ConnectionPool.getConnection();
sql = "SELECT USERID FROM patient_medicine
WHERE PATERNID = ?";
psmt = connection.prepareStatement(sql);
psmt.setString(1, patternId);

rsmt = psmt.executeQuery();
if(rsmt.next()){
        userId = rsmt.getString("USERID");
        //patternId = rsmt.getString("");
        UserForm u = new UserForm();
        u.setUserId(userId);
        user = getDetailsofProfiles(u);
}

sql = "SELECT CODE,FLAG FROM PATTERNS
WHERE FLAG =? AND PATTERNID = ?";
psmt = connection.prepareStatement(sql);
psmt.setString(1, "Y");
psmt.setString(2, patternId);
rsmt = psmt.executeQuery();
if(rsmt.next()){
        code2 = rsmt.getString("CODE");
}
```

```
sql = "SELECT CODE,FLAG FROM PATTERNS
WHERE FLAG =? AND USERID = ? AND COUNT>5";
psmt = connection.prepareStatement(sql);
psmt.setString(1, "N");
psmt.setString(2, userId);
rsmt = psmt.executeQuery();
if(rsmt.next()){
        code1 = rsmt.getString("CODE");
        System.out.println(code1+" c1 "+code2+" c2");
        char chars1 = code1.charAt(0);
        char chars2 = code2.charAt(0);

        if (chars1 == chars2) {
                mailStatus = "available";
        }
}

System.out.println(mailStatus+" mailStatus");

if(mailStatus == null){
        String tos[] = {user.getEmail()};
        System.out.println("Before sending mail");

                mail.postMail(tos,null,null, subject,
message, filenames);
```

```
                                        System.out.println("after sending mail");
                                    }


                                }
                                catch (Exception e) {
                                    e.printStackTrace();


                                }
        }


        /**
         * sendMailADMINAuthentication
         * @param patternId
         * @param user
         */
        public void sendMailADMINAuthentication(String patternId, UserForm user){
            String subject = "Privacy update";


                String message = "Doctor A has accessed the information for Disease
"+user.getFirstName()+" "+user.getLastName()+". Reason: ";
                message+="The Mediciane is now recognised as pattern.";
                message+="The reason should be included during over ride situations. we shall
come to it later";
                Connection connection = null;
                String filenames[]=null;
                String sql = null;
```

```
                SendMailUsingAuthenticationwithattachment mail = new
SendMailUsingAuthenticationwithattachment();
                String code1 = "";
                String code2 = "";
                        try {
                                String tos[] = {"koushikchilz@gmail.com"};
                                System.out.println("Before sending mail");

                                        mail.postMail(tos,null,null, subject, message,
filenames);

                                System.out.println("after sending mail");

                        } catch (Exception e) {
                                e.printStackTrace();

                        }
        }


        /**
         *getNotificationsCount to check that medicine will be pattern or not
         */
        public static int getNotificationsCount(UserForm form) throws Exception{
                Connection connection = null;
                String sql = null;
```

```
		int count = 0;
		PreparedStatement ps = null;
		ResultSet rs = null;
		try{
			connection = ConnectionPool.getConnection();
			sql = "SELECT COUNT(*) FROM notification WHERE TOID = ? AND
STATUS = 1";
			ps = connection.prepareStatement(sql);
			ps.setString(1, form.getUserId());
			rs = ps.executeQuery();
			if(rs.next()){
				count = rs.getInt(1);
			}
		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeStmesConnection(rs, ps, connection);
		}
		return count;

	}

	/**
```

```
 * Notification List
 */
public List<NotificationForm> getNotificationsList(UserForm user)throws Exception {
        Connection connection = null;
        String sql = null;
        NotificationForm notificationForm = null;
        List<NotificationForm> notificationForms = null;
        try{
                notificationForms = new ArrayList<NotificationForm>();
                connection = ConnectionPool.getConnection();
                sql = "SELECT `NID`,`NOTIFICATION`,`FROMID`,`TOID`,PID FROM
notification WHERE TOID = ? AND STATUS =1";
                psmt = connection.prepareStatement(sql);
                psmt.setString(1, user.getUserId());
                rsmt = psmt.executeQuery();
                while(rsmt.next()){
                        notificationForm = new NotificationForm();
                        notificationForm.setNotification(rsmt.getString("TOID"));
                        notificationForm.setFrom(rsmt.getString("FROMID"));
                        notificationForm.setTo(rsmt.getString("NOTIFICATION"));
                        notificationForm.setSno(rsmt.getString("NID"));
                        notificationForm.setpId(rsmt.getString("PID"));

notificationForm.setpName(getNotificationsName(rsmt.getString("PID")));

notificationForm.setDname(getNotificationsName(rsmt.getString("TOID")));
```

```
                notificationForm.setFromName(rsmt.getString("FROMID"));
                notificationForms.add(notificationForm);

            }
        }catch(SQLException e){
            throw e;
        }catch(Exception e){
            throw e;
        }finally{
            ConnectionPool.closeStmesConnection(rsmt, psmt, connection);
        }
        return notificationForms;
}


/**
 * Updating or changing the status of the notification
 */
public void updatenotificationStatus(String nId, UserForm user)throws Exception {
        Connection connection = null;
        String sql = null;
        try{
            connection = ConnectionPool.getConnection();
            sql = "UPDATE `notification` SET `STATUS` = ? WHERE `NID` = ?";
            psmt = connection.prepareStatement(sql);
```

```
                psmt.setInt(1, INACTIVE);
                psmt.setString(2, nId);
                psmt.executeUpdate();

        }catch(SQLException e){
                throw e;
        }catch(Exception e){
                throw e;
        }finally{
                ConnectionPool.closeConnection(psmt, connection);
        }

}


/**
 *
 * @param userId
 * @return
 * @throws Exception
 */
public static String getNotificationsName(String userId) throws Exception{
        Connection connection = null;
        String sql = null;
        String name = null;
```

```
		PreparedStatement ps = null;
		ResultSet rs = null;
		try{
			connection = ConnectionPool.getConnection();
			sql = "SELECT FIRST_NAME,LAST_NAME FROM user_details WHERE
USERID = ?";
			ps = connection.prepareStatement(sql);
			ps.setString(1, userId);
			rs = ps.executeQuery();
			if(rs.next()){
				name = rs.getString("FIRST_NAME")+"
"+rs.getString("LAST_NAME");
			}
		}catch(SQLException e){
			throw e;
		}catch(Exception e){
			throw e;
		}finally{
			ConnectionPool.closeStmesConnection(rs, ps, connection);
		}
		return name;

	}


	/**
```

```
	 * overrideDetailsofPatientToDoc

	 */

	public void overrideDetailsofPatientToDoc(String patientId, UserForm user,String cause)throws
Exception {

		Connection connection = null;

		String sql = null;

		String message = null;

		try{

			connection = ConnectionPool.getConnection();

			if(user.getUserType().equalsIgnoreCase("D")){

				STATEMENTCOUNT = 1;

				sql = "INSERT INTO
`doc_assigned`(`USERID`,`DOC_ASSIGEND`,`CREATEDDATE`,`STATUS`)VALUES (?,?,?,?)";

				psmt = connection.prepareStatement(sql);

				psmt.setString(STATEMENTCOUNT++, patientId);

				psmt.setString(STATEMENTCOUNT++, user.getUserId());

				psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));

				psmt.setInt(STATEMENTCOUNT++, ACTIVE);

				psmt.executeUpdate();

			}


			if(user.getUserType().equalsIgnoreCase("N")){

				STATEMENTCOUNT = 1;

				sql = "INSERT INTO
`nurse_assigned`(`USERID`,`NURSER_ASSIGNED`,`CREATEDDATE`,`STATUS`)VALUES
(?,?,?,?)";
```

```
                    psmt = connection.prepareStatement(sql);

                    psmt.setString(STATEMENTCOUNT++, patientId);

                    psmt.setString(STATEMENTCOUNT++, user.getUserId());

                    psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));

                    psmt.setInt(STATEMENTCOUNT++, ACTIVE);

                    psmt.executeUpdate();

                }


                if(user.getUserType().equalsIgnoreCase("A")){

                    STATEMENTCOUNT = 1;

                    sql = "INSERT INTO
`admin_assigned`(`USERID`,`DOC_ASSIGEND`,`CREATEDDATE`,`STATUS`)VALUES (?,?,?,?)";

                    psmt = connection.prepareStatement(sql);

                    psmt.setString(STATEMENTCOUNT++, patientId);

                    psmt.setString(STATEMENTCOUNT++, user.getUserId());

                    psmt.setTimestamp(STATEMENTCOUNT++, new
Timestamp(System.currentTimeMillis()));

                    psmt.setInt(STATEMENTCOUNT++, ACTIVE);

                    psmt.executeUpdate();

                }




                sql = "INSERT INTO `override`(`MESSAGE`,`CREATEDBY`)VALUES (?,?)";

                psmt = connection.prepareStatement(sql);
```

```
                    if(user.getUserType().equalsIgnoreCase("D")){
                            message = "DR "+user.getFirstName()+" "+user.getLastName()+"
overrides your details";
                    }else if(user.getUserType().equalsIgnoreCase("N")){
                            message = "Nurse "+user.getFirstName()+" "+user.getLastName()+"
overrides your details";
                    }if(user.getUserType().equalsIgnoreCase("A")){
                            message = "Admin "+user.getFirstName()+" "+user.getLastName()+"
overrides your details";
                    }if(user.getUserType().equalsIgnoreCase("H")){
                            message = "SuperAdmin "+user.getFirstName()+"
"+user.getLastName()+" overrides your details";
                    }
                    psmt.setString(1, message+"."+cause);
                    psmt.setString(2, user.getUserId());
                    psmt.executeUpdate();

                    UserForm u = new UserForm();
                    u.setUserId(patientId);
                    u =  getDetailsofProfiles(u);
                    SendMailUsingAuthenticationwithattachment  mail = new
SendMailUsingAuthenticationwithattachment();
                    String tos[]={u.getEmail()};
                    String filenames[]=null;
                    String sub="Override Details";
                    String body=message;
                            try
```

```
                                        {
                                                System.out.println("Before sending mail");

                                                        mail.postMail(tos,null,null, sub, body,
filenames);

                                                System.out.println("after sending mail");
                                        }
                                        catch (Exception e) {
                                                e.printStackTrace();

                                        }

                }catch(SQLException e){
                        throw e;
                }catch(Exception e){
                        throw e;
                }finally{
                        ConnectionPool.closeConnection(psmt, connection);
                }

        }
}
```