

ENHANCING GUIDELINES IN CLOUD FOR PERSONAL HEALTH RECORDS

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in
Information Technology

By

**Praveen K.S (Reg No. 39120083)
Ravi Shankar M.D(Reg No. 39120093)**



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

MAY- 2



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Praveen K.S** (Reg.No - 39120083) and **Ravi Shankar M.D** (Reg.No - 39120093) who carried out the Project Phase-2 entitled "**ENHANCING GUIDELINE IN CLOUD FOR PERSONAL HEALTH RECORDS**" under my supervision from January 2023 to May 2023.

R.M. Gomathi
25/4/23

Internal Guide

Dr. R.M. GOMATHI M.Tech., Ph.D.

R. Subhashini
14/5/23

Head of the Department

Dr. R. SUBHASHINI M.E., Ph.D.

Submitted for Viva voce Examination held on 11/05/23

[Signature]
11/5/23

Internal Examiner

[Signature]
11/5/23

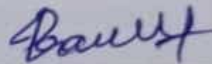
External Examiner

DECLARATION

I, Praveen K.S (Reg.No- 39120083), hereby declare that the Project Phase-2 Report entitled "ENHANCING GUIDELINE IN CLOUD FOR PERSONAL HEALTH RECORDS" done by me under the guidance of Dr. R.M. GOMATHI, M.Tech.,Ph.D is submitted in partial fulfillment of the requirements for the award Bachelor of Technology degree in Information Technology.

DATE: 11/05/2023

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. R. SUBHASHINI M.E., Ph.D.**, Head of the Department, Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.R.M GOMATHI M.Tech,Ph.D**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ASTARCT

The sharing of Personal Health Records (PHRs) among healthcare providers is essential for efficient and effective healthcare delivery. However, the security and privacy of sensitive health data are major concerns when sharing these records. To address these concerns, researchers have proposed various approaches to secure the storage and sharing of PHRs in the cloud. In this context, this paper proposes a policy update scheme for personal health records sharing that combines attribute-based encryption and proxy re-encryption. Attribute-Based Encryption (ABE) is an effective cryptographic technique for securing PHRs by allowing access control policies to be embedded in the encryption process. The proposed scheme enables a patient to share their PHRs with authorized users based on attribute-based policies. The patient can also update these policies dynamically, without having to re-encrypt the PHRs or re-distribute the decryption keys. The scheme achieves this by employing a proxy re-encryption technique that allows the delegated authority to re-encrypt the PHRs to maintain confidentiality while granting access to authorized users. The scheme is evaluated using simulation experiments, and the results demonstrate that it achieves efficient and secure policy updates with low computational overhead. The main advantage is its design, which reduces the computational and storage overhead associated with traditional policy update schemes. In conclusion, the proposed scheme provides an efficient and secure approach to policy updates for outsourced personal health records sharing in the cloud.

TABLE OF CONTENTS

Chapter No	TITLE		Page No.
	ABSTRACT		v
	LIST OF FIGURES		viii
	LIST OF ABBREVIATIONS		ix
1	INTRODUCTION		1
2	LITERATURE SURVEY		
	2.1	Inferences from Literature Survey	12
	2.2	Open problems in Existing System	14
3	REQUIREMENTS ANALYSIS		
	3.1	Feasibility Studies/Risk Analysis of the Project	15
	3.2	Software Requirements Specification Document	15
	3.3	System Use case	16
4	DESCRIPTION OF PROPOSED SYSTEM		
	4.1	Selected Methodology or process model	19
	4.2	Architecture of Proposed System	20
	4.3	Description of Software for Implementation and Testing plan of the Proposed Model/System	21
	4.4	Project Management Plan	21
5	IMPLEMENTATION DETAILS		
	5.1	Development and Deployment Setup	23
	5.2	Module	24
	5.3	Algorithms	25
6	RESULTS AND DISCUSSION		29
7	CONCLUSION		
	7.1	Conclusion	30
	7.2	Future work	31
	7.3	Research Issues	31
	7.4	Implementation Issues	31
	REFERENCES		33

	APPENDIX	
	A. SOURCE CODE	35
	B. SCREENSHOTS	50
	C. RESEARCH PAPER	61

LIST OF FIGURES

FIGURE NO	FIGURE NAME	Page No.
3.3.1	Patient	16
3.3.2	Doctor	17
3.3.3	Management	17
3.3.4	Authority	18
3.3.5	Proxy Server	18
4.1	System Architecture	20

LIST OF ABBREVIATION

ACRONYM	DESCRIPTION
PHR	Personal Health Records
ABE	Attribute-Based Encryption
CSRF	Cross-Site Request Forgery
IoT	Internet of Things
RABE	Revocable Attribute-Based Encryption
ABPRE	Attribute-Based Proxy Re-Encryption
CP-ABPRE	Ciphertext-Policy Attribute-Based Proxy Re-Encryption
UML	Unified Modeling Language
PRE	Proxy-Re-Encryption
DES	Data Encryption Standard
CA	Certification Authority
AES	Advanced Encryption Standard

CHAPTER 1

INTRODUCTION

Cloud storage and other outsourced shared data environments necessitate continual access to external servers in order to offer unrestricted access to shared data and services. Due to cost-saving benefits and efficient resource management, many businesses and organizations opt to store critical data on external servers, such as cloud storage. To address privacy and security concerns, data handlers often encrypt data before uploading it to cloud servers. Encryption is widely considered the most effective method for preventing unauthorized access to sensitive data. Yet encryption alone is not enough to establish strict security controls. Another commonly required security barrier is an access control system. Additionally, PHR handlers need the freedom and security to design their own access policies for the offloaded data. Nowadays, commercial cloud systems usually offer both symmetric and public key cryptography, in addition to basic authentication capabilities. However, these traditional encryption techniques are not well-suited for data outsourcing scenarios due to the large key management overhead and high maintenance costs associated with symmetric encryption. Public key cryptography solutions can also result in multiple copies of ciphertext. Now days, many companies and individuals prefer to store their valuable data in outsourced servers such as cloud storage due to cost saving and efficient resource management provided by cloud providers. Regarding to the privacy and security issue, data owners usually encrypt their data before outsourcing it to the cloud server. Encrypting data is the most suitable way to protect the sensitive data from unauthorized access. Nevertheless, encryption alone is not adequate to support rigorous security control. Access control mechanism is another security perimeter generally required. CSRF (Cross-Site Request Forgery) is use to security is achieved. It is type of security allows an attacker to perform actions on behalf of a victim user on a website the paper proposes a scheme that uses ciphertext fragmentation to handle different encrypted domains. In other words, it breaks down the ciphertext into smaller fragments and encrypts each fragment with a different key. This enables access control over individual fragments of the ciphertext, allowing users to access only the parts of the data that they are authorized to see. the proposed system employs a

multi-layered access control model to enable flexible and fine-grained access control while protecting patient privacy. The access control model consists of three layers: attribute-based encryption (ABE) layer, access policy updating (APU) layer, and an access control layer. The ABE layer utilizes a type of encryption that uses attributes or properties as a basis for access control. Each user and data item is assigned a set of attributes, and access is granted based on matching these attributes with a predefined access policy. This enables fine-grained access control, as users can only access data items for which they have the required attributes. The APU layer is responsible for updating the access policy as required. This allows for dynamic changes in access control, such as revoking access for a particular user or group of users, or adding new access criteria.

Finally, the access control layer implements the access control policy using ciphertext fragmentation. As mentioned before, this method breaks down the ciphertext into smaller fragments and encrypts each fragment with a different key. This enables fine-grained access control, as users can only access the parts of the data that they are authorized to see. The access control layer also includes a CSRF prevention mechanism to ensure that unauthorized access is prevented.

CHAPTER 2

LITERATURE SURVEY

2.1 Inferences from Literature Survey

Cloud computing has become a popular paradigm for storing and accessing data due to its numerous advantages, including scalability, accessibility, and cost-effectiveness. One popular technique for ensuring data privacy is attribute-based encryption (ABE), which allows data to be encrypted and decrypted based on attributes rather than individual user credentials. Access control and encryption are both important concepts in information security, but they address different aspects of security. Access control is concerned with regulating who is allowed to access what resources and what actions they are allowed to perform once they have access. Encryption, on the other hand, is the process of converting plain text or data into ciphertext, which can only be read by someone who has the key to decrypt it. For instance, in [1], the author introduces attribute-based searchable methods based on cryptography, which is considered a flexible access policy. The method allows for efficient search of encrypted data based on user-defined attributes. To reduce the cost of user decryption, Green et al [2] proposed an attribute-based encryption (ABE) scheme to the cloud server. It provided as the mechanism for optimized resources management. IN the Internet of Things (IoT), many tools and technologies have rendered a new vision, enabling near-field communication devices, RFID tags, readers, etc., which collect and share information across different platforms such as AI [7]. In [3], a toolkit is provided to enable two trapdoor public key cryptosystems to handle different encrypted domains in the cloud server. The method ensures secure and efficient communication among different IoT devices.

As the cloud is a computing paradigm that stores data in centralized cloud and allows users to access it anytime using simple client devices, it has become a popular choice for storing medical data. In [5], ABE with time encodings is used for the data handlers to effectively control access to the authority of dynamically changing users. The method ensures that only authorized users can access the data and that the data remains secure at all times.

In conclusion, attribute-based encryption is a powerful technique for ensuring data privacy in cloud environments. However, policy updates can hinder the

scalability and efficiency of the technique.[8]a new access control model that uses a combination of authorizing workflow and task-role-based access control to regulate access to resources that to ensure that users have access to only the resources they need to complete their assigned tasks.[9] presents a cloud-based personal health record system that enables patients to store, manage, and share their health records with healthcare providers.[10] explores the benefits and challenges of cloud-based personal health record systems and discusses the design and implementation of a prototype system that enables patients to manage their health records.[11] the authors propose an approach to data clustering in IoT that uses encryption and redundancy to improve privacy and availability. the authors propose a dynamic access control model that uses a combination of authorizing workflow and task-role-based access control to regulate access to resources. In the second paper, the authors describe a cloud-based personal health record management system that uses encryption to secure patient data. In the third paper, the authors propose a cloud-based personal health record system to empower patients, but access control and encryption are not mentioned as part of the system. In the fourth paper, the authors propose an approach to enhance the privacy and availability of data clustering in IoT, but access control and encryption are not directly related to their proposal [12] proposes a new scheme called AKSER for secure and efficient attribute-based keyword search in cloud computing. The scheme uses a revocable attribute-based encryption (RABE) technique to encrypt the search query and the corresponding document keywords, and an index-based search algorithm for efficient search. The revocation mechanism allows revoking access to the search query and the corresponding documents in case of unauthorized access or change in access control policies. [13] proposes a privacy-preserving data publishing and subscription mechanism in cloud platforms. The scheme uses attribute-based encryption (ABE) to encrypt data and keywords, and a publish-subscribe model for data dissemination. The proposed scheme allows for flexible access control policies and efficient keyword-based searches while preserving the privacy of the data owners. [14] proposes an attribute-based proxy re-encryption (ABPRE) scheme with delegating capabilities. The scheme allows a delegator to delegate decryption capabilities for specific attributes to a proxy, who can then perform the decryption on behalf of the delegator. The proposed scheme allows for fine-grained access control and efficient

data sharing in a cloud environment. [15] proposes Flexible ciphertext-policy attribute-based proxy re-encryption" proposes a flexible ciphertext-policy attribute-based proxy re-encryption (CP-ABPRE) scheme with outsourced re-encryption key generation. The scheme allows for fine-grained access control and efficient data sharing while delegating the expensive re-encryption key generation process to a third-party service provider. The proposed scheme is efficient and flexible and can be used in cloud environments.

From the above-mentioned literature works, it is clear that there has been effective research. A Simplified Policy Update Scheme for Outsourced Personal Health Sharing have been proposed. It is evident that the above-mentioned systems have their own pros and cons. While some of the recent works involve hybrid technologies and provide better accuracies, they are still far from what is needed. With higher accuracy, comes the need for low computational costs, high processing speed, and most of all, convenience of use.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

Existing commercial cloud platforms usually provide symmetric or public key encryption as optional feature to support data confidentiality for their tenants. However, such traditional encryption schemes are not suitable for data outsourcing environment because of high key management overhead of symmetric encryption and high maintenance cost for handling multiple copies of cipher text for public key encryption solution.

Disadvantages:

- High complexity.
- Low Computation.
- Requires skilled persons

CHAPTER-3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES

From the above mentioned literature works, it is clear that there has been effective research on A Simplified Policy Update Scheme For Outsourced Personal Health Sharing have been proposed .It is evident that the above mentioned systems have their own pros and cons .While some of the recent works involve hybrid technologies and provide better accuracies, they are still far from what is needed .With higher accuracy, comes the need for low computational costs, high processing speed, and most of all, convenience of use.

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

Operating System	: Windows 7/8/10
Server side Script	: HTML, CSS & JS
IDE	: Pycharm
Libraries Used	: Numpy, IO, OS, SCIKIT-LEARN
Technology	: Python 3.6+
Frame Work	: Flask.

3.3 USE CASE DIAGRAM

- A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis.
- Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.
- The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

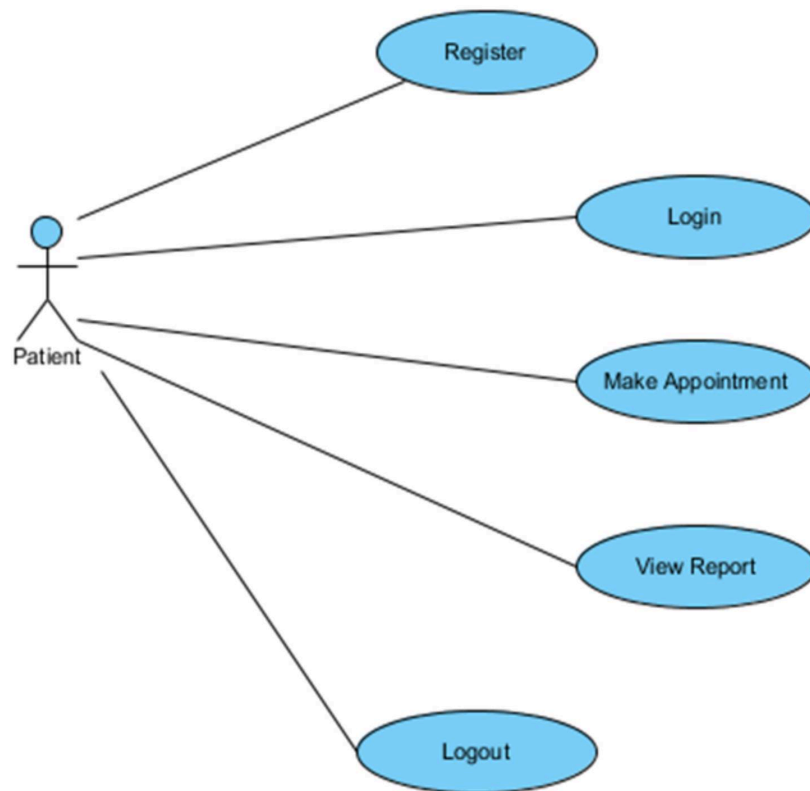


Fig:3.3 Patient

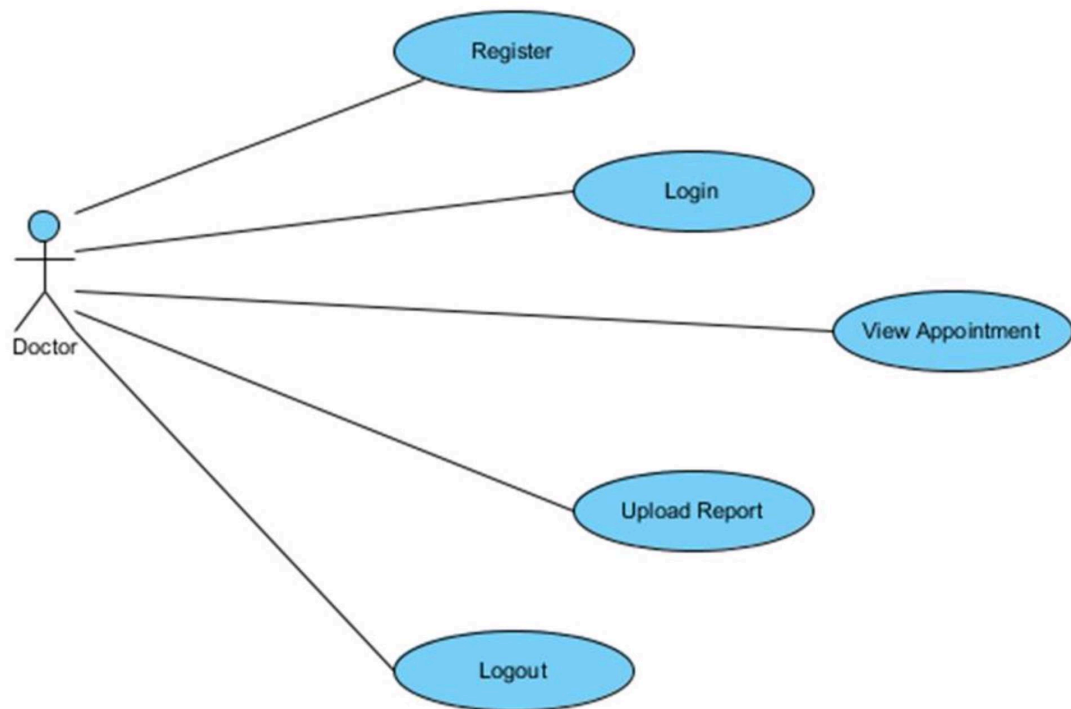


Fig 3.3.2 Doctor

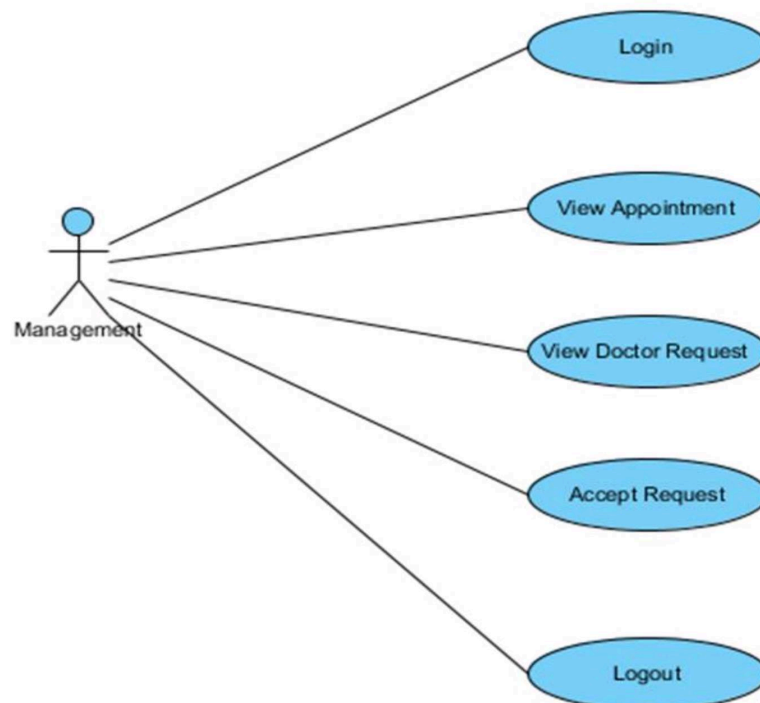


Fig:3.3.3 Management

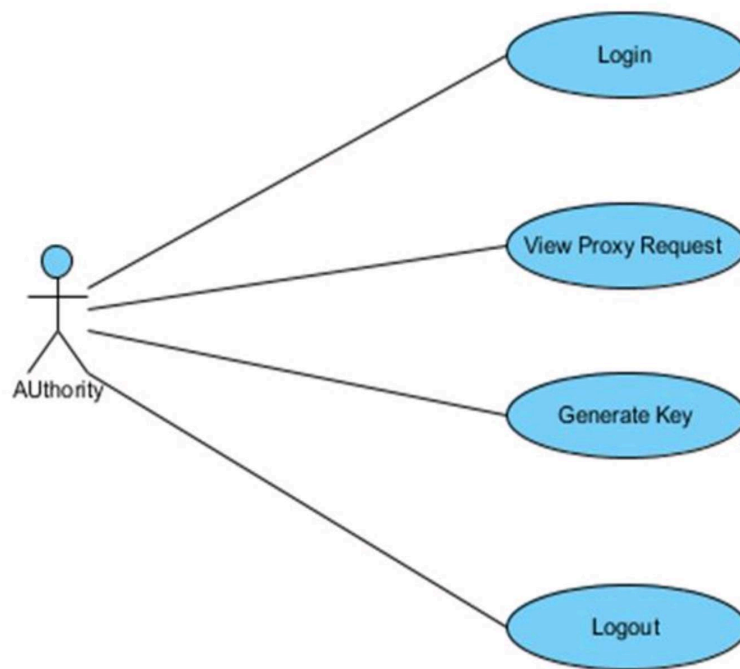


Fig 3.3.4 Authority

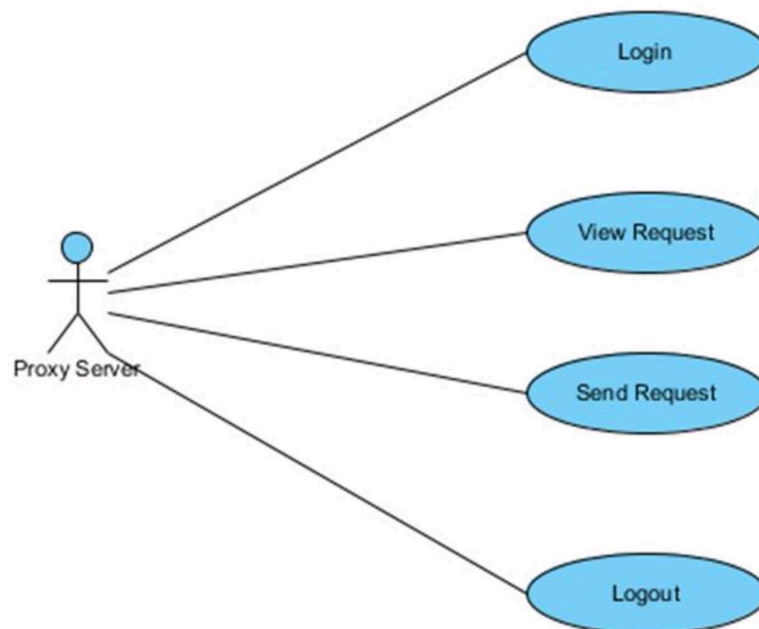


Fig:3.3.5 Proxy Server

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

This proposed scheme is based on the Cipher Text Policy Attribute-Based Encryption (CP-ABE) and Proxy Re-Encryption (PRE). In addition, introduce a policy versioning technique to support the full traceability of policy changes. Finally, the conducted the performance evaluation to demonstrate.

Advantages:

- Accuracy is good.
- Low complexity.
- High Computation.
- No need of skilled person

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

4.1.1 CLOUD

Cloud includes three basic services:

- Infrastructure as a Service (IaaS),
- Platform as a Service (PaaS), and
- Software as a Service (SaaS).

Software-as-a-service (SaaS) involves the licensure of a software application to customers. Licenses are typically provided through a pay-as-you-go model or on-demand. This type of system can be found in Microsoft Office Infrastructure-as-a-service (IaaS) involves a method for delivering everything from operating systems to servers and storage through IP-based connectivity as part of an on-demand service. Clients can avoid the need to purchase software or servers, and instead procure these resources in an outsourced, on-demand service. Popular examples of the IaaS system Azure.

Platform-as-a-service (PaaS) is considered the most complex of the three layers of cloud-based computing. PaaS shares some similarities with SaaS, the primary difference being that instead of delivering software online, it is actually a platform for creating software that is delivered via the Internet.

This model includes platforms like Salesforce.com and Heroku.

4.1.2. PURPOSE

The purpose of data encryption is to protect digital data confidentiality as it is stored on computer systems and transmitted using the internet or other computer networks. The outdated Data Encryption Standard (DES) has been replaced by modern encryption algorithms that play a critical role in the security of IT systems and communications. These algorithms provide confidentiality and drive key security initiatives including authentication, integrity, and non-repudiation. Authentication allows for the verification of a message's origin, and integrity provides proof that a message's contents have not changed since it was sent. Additionally, non-repudiation ensures that a message sender cannot deny sending the message.

4.2 ARCHITECTURE OF PROPOSED SYSTEM

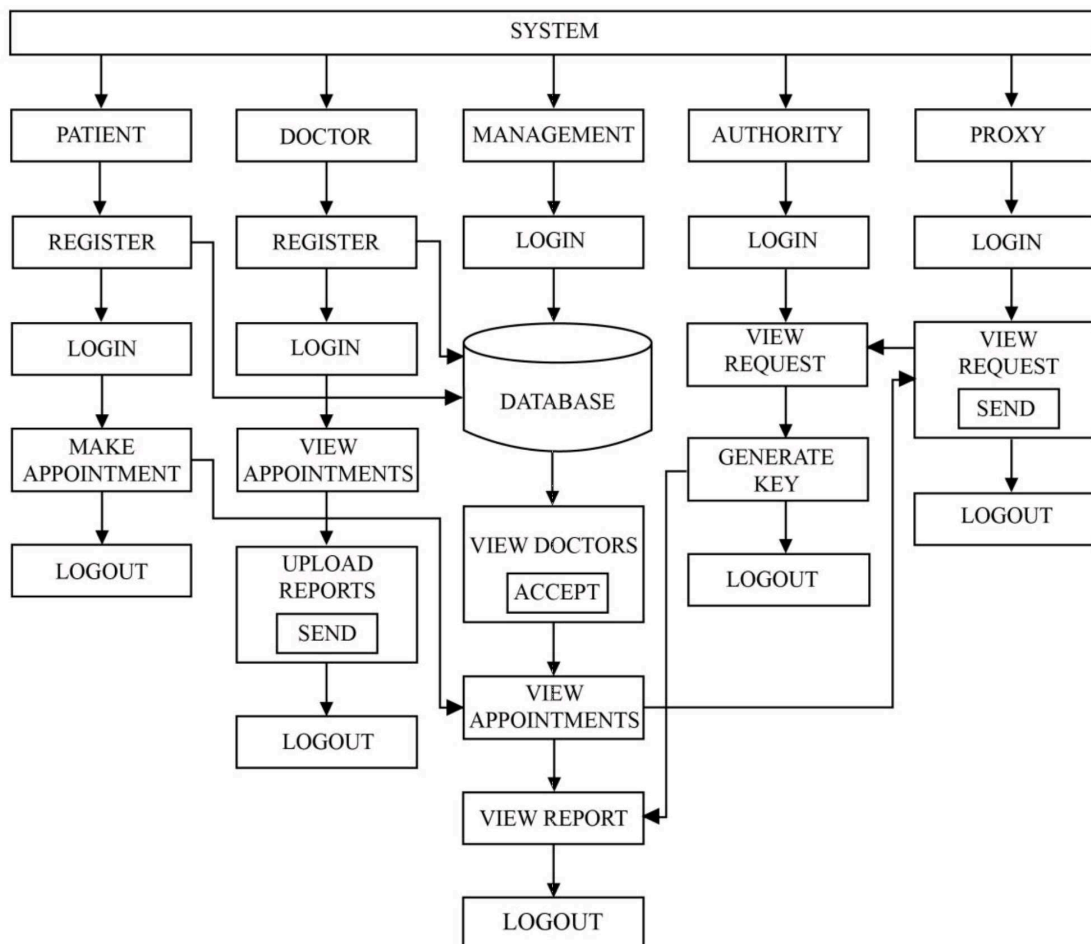


Fig:4.2 System Architecture

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN PROPOSED MODEL/SYSTEM

4.3.1 Implementation:

PHR owners upload data files such as treatment records, patient profile, etc. in encrypted format to the cloud server and users such as doctors can access the shared file if they have the capability (i.e. decryption key) satisfying the access control policy. PHR owners and users are issued a set of attributes in the form of user decryption key from attribute authorities. Our model supports multiple authorities who can issue the attributes to users. For example, a patient may be issued keys issued from different authorities such as hospitals, insurance company. In the data outsourcing environment such as cloud computing, a proxy which is a semi-trusted server is exploited to be responsible for performing re-encryption task when any policy is updated. Hence, major heavy cryptographic operations and secure computation are offloaded to the proxy. The proxy is installed with an X.509 certificate issued by a trusted Certification Authority (CA). The certificate is used for the authentication with other system entities. Thus, the proxy only communicates to the entities having the valid certificate as predetermined in its configuration system.

4.4 PROJECT MANAGEMENT PLAN

- Upload data files (PHR Owners)-Using login in encrypted credentials format. Access the shared files –Using the access control Policy.
- Setting of Attributes in the form of decryption key from attribute authorities.
- Using proxy to validate the certificate as configuration system.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DESIGN AND DEVELOPMENT

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc. Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties –

- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding.
- It should serve specific purpose effectively such as storing, recording, and retrieving the information.

5.1.1 Objectives for Input Design

The objectives of input design are

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

5.1.2 Output Design

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

The objectives of input design are

- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end user's requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

5.2 MODULES

5.2.1 PATIENT

Login: Patient has to login with valid details which are used in his / her Registration

Register: Each and every patient has to Register.

Appointment: Patient will raise an appointment with symptoms

View Report: Patient will view reports after his / her medical checkup. Logout, Finally, logout from the system.

5.2.2 DOCTOR

Login: Doctor has to login with valid details which are used in his / her Registration

Register: Each and every patient has to Register and management has to accept request.

View Appointment: View all the appointments

Upload Report: Uploads report.

Send File: Send's file to the proxy

Logout: Finally logout from the system.

5.2.3 HOSPITAL MANAGEMENT

Login: Management will login with default details

Appointment: Views all the appointments requests from the Patients

View Doctor Request: View all the doctor requests who are register

Send Information: Patient request will be passed to doctor

Logout: Finally, logout from the system.

5.2.4 AUTHORITY

Login: Authority will login with default details

View Request: View all requests from proxy.

Generate Key: Generate Key to pass it to the authorized patient

Logout: Finally, logout from the system.

5.2.5 PROXY SERVER

Login: Authority will login with default details

View Request: View all requests from Doctors.

Send Request: Pass requests to the authority

Logout: Finally, logout from the system.

5.4 ALGORITHMS

The algorithm selection was based on criteria such as efficiency, security, and compatibility with the specific use case. The algorithm used in the proposed scheme is referred to as a "key update algorithm (AES.CP-ABE)" which is used to update the encryption key used to protect personal health records. The proposed scheme is designed to be flexible and efficient, while also providing strong security guarantees for personal health records. Firstly, the Advanced Encryption Standard (AES) was chosen for data encryption and decryption. AES is a widely used symmetric key algorithm that provides a high level of security and performance. It is considered one of the strongest encryption algorithms available today and has been extensively analysed and tested for security.

Secondly, the authors used the Ciphertext-Policy Attribute-Based Encryption (CP-ABE) algorithm for access control. CP-ABE is a type of attribute-based encryption that allows for fine-grained access control based on attributes associated with users and data. It is a suitable choice for secure PHR sharing as it allows for flexible policy updates and revocation of access rights. Finally, the authors utilized the SHA-256 hash function for data integrity verification. SHA-256 is a widely used cryptographic hash function that generates a fixed-length output for any input data. It is a suitable choice for ensuring data integrity as any change in the data will result in a different hash value, making it easy to detect any tampering.

In summary, the selected algorithms are well-established and widely used in the field of cryptography, providing a high level of security and efficiency for the proposed PHR sharing scheme. The paper proposes a scheme that uses ciphertext fragmentation to handle different encrypted domains. In other words, it breaks down the ciphertext into smaller fragments and encrypts each fragment with a different key. This enables access control over individual fragments of the ciphertext, allowing users to access only the parts of the data that they are authorized to see. To achieve this, the paper uses a variant of attribute-based encryption called ciphertext-policy attribute-based encryption (CP-ABE). In CP-ABE, the ciphertext is encrypted under a set of attributes rather than a specific key. A user can decrypt the ciphertext if they have a secret key that matches the attributes associated with the ciphertext's proposed scheme extends CP-ABE to handle multiple encrypted domains by using a technique called attribute-based proxy re-encryption (ABPRE). In ABPRE, a trusted authority generates a re-encryption key that allows a ciphertext encrypted under one set of attributes to be transformed into a new ciphertext encrypted under a different set of attributes. The paper's approach is to use ABPRE to fragment the ciphertext into smaller encrypted fragments, each encrypted under a different set of attributes. Each fragment is associated with a different set of attributes, and users can only access fragments for which they have the matching attributes. By breaking down the ciphertext in this way, the proposed scheme enables fine-grained access control over different parts of the data, even when they are encrypted under different sets of attributes.

CP-ABE:

The paper uses the CP-ABE (Ciphertext-Policy Attribute-Based Encryption) algorithm for securing the personal health records. CP-ABE is an encryption scheme that allows data owners to encrypt their data using a set of attributes, and data users to decrypt the encrypted data if they have the required set of attributes. The access policy is defined as a Boolean function over attributes. The ciphertext is associated with an access policy and can be decrypted if the attributes of the user satisfy the access policy. In the paper, the CP-ABE algorithm to encrypt the personal health records, where the access policy is defined as a set of attributes such as patient ID, doctor ID, and time period. The ciphertext is associated with an access policy that defines which user can access the encrypted data. A trusted authority is responsible for generating and distributing secret keys to users based on their attributes. A combination of CP-ABE and proxy encryption is used to achieve secure and efficient policy updates in outsourced personal health record sharing. The authors justify their algorithm selection by showing that this approach can provide fine-grained access control and policy updates while minimizing the computational and communication overheads. They also validate their algorithm through simulation experiments, where they compare the performance of their proposed scheme with other state-of-the-art approaches. The results show that the proposed scheme outperforms the other approaches in terms of communication overhead and policy update time.

Suppose there are two users, User A and User B, who want to share their personal health records with their doctors. The doctors are categorized into two roles: Role 1 and Role 2. User A wants to share their data with doctors in Role 1, and User B wants to share their data with doctors in Role 2. First, a CP-ABE algorithm is used to encrypt the personal health records of User A and User B. The access policies for the two users are set as follows:

User A: Attribute 1 = "Role 1"

User B: Attribute 2 = "Role 2"

Then, a proxy encryption algorithm is used to generate a proxy key for each doctor. The proxy key includes the attributes of the doctor's role, and it allows the doctor to decrypt the records of the corresponding user.

For example, a doctor in Role 1 is given a proxy key that includes Attribute 1 = "Role 1". This allows them to decrypt the records of User A, but not User B, since User B's records are encrypted with Attribute 2 = "Role 2". Similarly, a doctor in Role 2 is given a proxy key that includes Attribute 2 = "Role 2". This allows them to decrypt the records of User B, but not User A. By combining CP-ABE and proxy encryption, the personal health records are protected and can only be accessed by authorized doctors with the correct attributes in their proxy keys. This ensures that the records are secure and accessible only to those who have permission to access them.

CHAPTER 6

RESULTS & DISCUSSION

6.1 RESULTS:

The proposes a update scheme for efficient and secure sharing of outsourced personal health records (PHRs). The proposed scheme uses a combination of cipher policy attribute-based encryption (CP-ABE), advanced encryption standard (AES), and proxy re-encryption techniques to achieve efficient and secure policy updates.

The results of the proposed scheme is effective in terms of the computational overhead, communication overhead, and storage overhead. The experiments show that the proposed scheme has a low computational cost, which makes it suitable for resource-constrained environments. Furthermore, the communication overhead of the proposed scheme is also low, which means that it is suitable for use in low-bandwidth environments. Finally, the storage overhead of the proposed scheme is also low, which means that it can be used in systems with limited storage capacity.

The evaluates the security of the proposed scheme by analysing its resilience to various attacks, including data leakage, policy violation, and replay attacks. The results show that the proposed scheme is resistant to these attacks and provides a high level of security for the outsourced PHRs.

In conclusion, the proposed policy update scheme for outsourced PHR sharing provides an efficient and secure way to update policies in a distributed environment. The use of CP-ABE, AES, and proxy re-encryption techniques helps to achieve both efficiency and security in the proposed scheme. The experimental results and security analysis show that the proposed scheme is effective and provides a high level of security for the outsourced PHRs.

CHAPTER 7

CONCLUSION & FUTURE WORK

7.1 CONCLUSION

In conclusion, the proposed scheme proposes an enhanced policy update scheme for outsourced personal health records sharing, which leverages the combination of CP-ABE and proxy re-encryption techniques to achieve efficient and secure policy updates. The proposed scheme allows for policy updates to be performed without requiring the re-encryption of all data, thereby reducing computational overhead and ensuring the privacy of sensitive data. One of the main strengths of the proposed scheme is its ability to handle dynamic policy updates in a scalable and efficient manner. The scheme achieves this by using a policy tree structure that enables the efficient management of policy updates for large-scale healthcare systems. Furthermore, the proposed scheme provides a flexible and fine-grained access control mechanism that allows authorized users to access only the necessary data based on their attributes and role. The uniqueness of the proposed scheme lies in its design, which ensures that the scheme is efficient in terms of both computation and communication overhead. The scheme achieves this by using a combination of CP-ABE and proxy re-encryption techniques, which allows for policy updates to be performed without requiring the re-encryption of all data. The proposed scheme also leverages the use of symmetric key encryption for data encryption and decryption, which further reduces the computational overhead. Overall, the proposed scheme provides a practical and secure solution for policy updates in healthcare systems, which can be extended to other domains that require efficient and secure policy updates.

7.2 FUTURE WORK

The paper proposes a novel scheme for updating access policies for outsourced personal health records sharing in a lightweight manner. The authors have presented a comprehensive evaluation of their proposed scheme and have shown its superiority over the existing schemes in terms of efficiency, security, and flexibility. There is a potential for future work in this area. One possible direction could be to integrate the proposed scheme with emerging technologies such as blockchain, to enhance the security and privacy of personal health records sharing. Additionally, future research could also focus on improving the scalability of the proposed scheme to handle large-scale personal health record sharing scenarios.

7.3 RESEARCH ISSUES

- It is difficult to store such a large amount of information without overloading traditional computer systems.
- It is difficult to protect great volumes of digital data when it is being stored.
- The resources required to constantly manage and maintain digital data accurately can be expensive.

7.4 IMPLEMENTATION ISSUES

Security: Security is a major concern when deploying applications or services in the cloud. Cloud environments are vulnerable to cyber-attacks, and therefore it's important to implement appropriate security measures to protect against data breaches, hacking, and other security threats.

Performance: Cloud environments can sometimes suffer from performance issues due to factors such as network latency, limited bandwidth, and shared resources. To mitigate these issues, it's important to optimize application performance by using techniques such as caching, load balancing, and content delivery networks (CDNs).

Scalability: One of the main benefits of cloud computing is its ability to scale up or down based on demand. However, implementing scalability can be challenging, especially for applications with complex architectures. To ensure scalability, it's important to design applications with scalability in mind from the outset and use technologies such as auto-scaling and horizontal scaling.

Data management: Managing data in the cloud can be complex, especially when dealing with large datasets or multiple data sources. It's important to implement appropriate data management practices, such as data backup and recovery, data encryption, and data governance.

Vendor lock-in: Deploying applications or services in the cloud can sometimes result in vendor lock-in, where it becomes difficult to switch to a different cloud provider or move back to an on-premises environment. To avoid vendor lock-in, it's important to choose a cloud provider that supports open standards and provides data portability.

Cost management: Cloud computing can be expensive, especially if resources are not managed effectively. To manage costs, it's important to use cost optimization tools provided by cloud providers, monitor resource usage, and use reserved instances or spot instances where appropriate.

Overall, implementing applications or services in the cloud can be challenging, but by addressing these implementation issues, organizations can realize the benefits of cloud computing, such as increased flexibility, scalability, and cost savings.

REFERENCE

- [1] Z. Ying, W. Jang, S. Cao, X. Liu and J. Cui, "A Lightweight Cloud Sharing PHR System with Access Policy Updating," in *IEEE Access*, vol. 6, pp. 64611-64621, 2018, doi: 10.1109/ACCESS.2018.2877981.
- [2] K. Liang and W. Susilo, "Searchable Attribute-Based Mechanism With Efficient Data Sharing for Secure Cloud Storage," in *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 9, pp. 1981-1992, Sept. 2015, doi: 10.1109/TIFS.2015.2442215.
- [3] J. Wei, X. Chen, X. Huang, X. Hu and W. Susilo, "RS-HABE: Revocable-Storage and Hierarchical Attribute-Based Access Scheme for Secure Sharing of e-Health Records in Public Cloud," in *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2301-2315, 1 Sept.-Oct. 2021, doi: 10.1109/TDSC.2019.2947920.
- [4] C. -J. Wang, X. -L. Xu, D. -Y. Shi and W. -L. Lin, "An Efficient Cloud-Based Personal Health Records System Using Attribute-Based Encryption and Anonymous Multi-receiver Identity-Based Encryption," 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014, pp. 74-81, doi: 10.1109/3PGCIC.2014.38.
- [5] Xiong, J., Zhang, Y., Li, X. et al. RSE-PoW: a Role Symmetric Encryption PoW Scheme with Authorized Deduplication for Multimedia Data. *Mobile Netw Appl* 23, 650–663 (2018). <https://doi.org/10.1007/s11036-017-0975-x>
- [6] R. Wang, J. Yan, D. Wu, H. Wang and Q. Yang, "Knowledge-Centric Edge Computing Based on Virtualized D2D Communication Systems," in *IEEE Communications Magazine*, vol. 56, no. 5, pp. 32-38, May 2018, doi: 10.1109/MCOM.2018.1700876.
- [7] C. Li and W. Ma, "Comments on "An Efficient Privacy-Preserving Outsourced Calculation Toolkit With Multiple Keys"," in *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2668-2669, Oct. 2018, doi: 10.1109/TIFS.2018.2825143.

- [8] J. Xiong et al., "Enhancing Privacy and Availability for Data Clustering in Intelligent Electrical Service of IoT," in *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1530-1540, April 2019, doi: 10.1109/JIOT.2018.2842773
- [9] B. H. Venkatesh, A. P. Sai, M. R. Reddy and S. Fathimabi, "Cloud based Personal Health Record Management System and Medical Recommender System," 2022 7th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2022, pp. 1744-1749, doi: 10.1109/ICCES54183.2022.9835988.
- [10] Y. -T. Song, S. Hong and J. Pak, "Empowering patients using cloud based personal health record system," 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Takamatsu, Japan, 2015, pp. 1-6, doi: 10.1109/SNPD.2015.7176216.
- [11] M. Uddin, S. Islam and A. Al-Nemrat, "A Dynamic Access Control Model Using Authorising Workflow and Task-Role-Based Access Control," in *IEEE Access*, vol. 7, pp. 166676-166689, 2019, doi: 10.1109/ACCESS.2019.2947377.
- [12] J. Cui, H. Zhou, H. Zhong and Y. Xu, "AKSER: Attribute-based keyword search with efficient revocation in cloud computing", *Inf. Sci.*, vol. 423, pp. 343-352, Jan. 2018.
- [13].Privacy-preserving attribute-keyword based data publish-subscribe service on cloud platforms *Information Sciences*, Volume 387, 2017, pp. 116-131
- [14].X. Liang, Z. Cao, H. Lin and J. Shao, "Attribute based proxy re-encryption with delegating capabilities", *Proc. 4th Int. Symp. Inf. Comput. Commun. Secur. (ASIACCS)*, pp. 276-286, 2009.
- [15].Y. Kawai, "Outsourcing the re-encryption key generation: Flexible ciphertext-policy attribute-based proxy re-encryption", *Proc. Int. Conf. Inf. Secur. Pract. Exper. (ISPEC)*, pp. 301-315, 2015.

APPENDIX

A. SOURCE CODE

```
from flask import *

import mysql.connector

import pandas as pd

import random

from flask_mail import *

db=mysql.connector.connect(user='root',port=3306,database='phr')

cur=db.cursor()

import os

app=Flask(__name__)

app.secret_key='!@#$H%S$BV#AS><)SH&BSGV*(_Sjnkxcb9+_ )84JSUHB&*%$^+=

,

@app.route('/')

def index():

    return render_template('index.html')

@app.route('/patient',methods=['POST','GET'])

def Patientlog():

    if request.method=='POST':

        name=request.form['Name']

        password=request.form['Password']

        cur.execute("select * from patient_reg where Name=%s and

Password=%s",(name,password))

        content=cur.fetchall()

        print(content)
```

```

db.commit()

if content==[]:

    msg="Credentials Does't exist"

    return render_template('patientlog.html',msg=msg)

else:

    print(content)

    age = content[0][-3]

    return render_template('patienthome.html',name=name,age=age)

return render_template('patientlog.html')

@app.route('/patientreg',methods=['POST','GET'])

def Patientreg():

    if request.method=='POST':

        name=request.form['Name']

        age=request.form['Age']

        email=request.form['Email']

        password1=request.form['Password']

        password2=request.form['Con_Password']

        if password1 == password2:

            sql="select * from patient_reg where Name='%s' and
Email='%s'"%(name,email)

            cur.execute(sql)

            data=cur.fetchall()

            db.commit()

            print('----',data)

            if data==[]:

```

```
sql="insertintopatient_reg(Name,Age,Email>Password)values(%s,%s,%s,%s)"
```

```
val=(name,age,email,password1)
```

```
cur.execute(sql,val)
```

```
db.commit()
```

```
return render_template('patientlog.html')
```

```
else:
```

```
warning='Details already Exist'
```

```
return render_template('patientreg.html',msg=warning)
```

```
error='password not matched'
```

```
flash(error)
```

```
return render_template('patientreg.html')
```

```
@app.route('/proceed')
```

```
def proceed():
```

```
return render_template('proceed.html')
```

```
@app.route('/patientreq',methods=['POST','GET'])
```

```
def patientreq():
```

```
if request.method=='POST':
```

```
Name=request.form['Name']
```

```
doc=request.form['Doc']
```

```
Age=request.form['Age']
```

```
Symptoms=request.form['symptoms']
```

```
AppointmentDate=request.form['AppointmentDate']
```

```
Time=request.form['Time']
```

```
sql="insert into patientreq (Name,Type,Age,symptoms,AppointmentDate,Time)
values
('%s','%s','%s','%s','%s','%s')"%(Name,doc,Age,Symptoms,AppointmentDate,Time)
```



```

cur.execute(sql)

db.commit()

msg="Your appointment request Sent to Management"

return render_template('patienthome.html',msg=msg)

return render_template('patienthome.html')

@app.route('/hospitalmanagement',methods=['POST','GET'])
def hospital_management():

    if request.method=='POST':

        name=request.form['Username']

        password=request.form['passcode']

        print(name,password)

        if name=="Hospital" and password=='management':

            return render_template('managementhome.html')

        return render_template('management.html')

@app.route('/viewappointments')
def view_appointments():

    sql="select Id,Name,Type,Age,AppointmentDate,Time from patientreq where
status='pending'"

    data=pd.read_sql_query(sql,db)

    db.commit()

    return
render_template('viewappointments.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/accept_request/<x>/<y>/<z>')
def acceptreq(x=0,y="",z=""):

    print(x,y)

    print(z)

```

```

    sql="select slno,Name,Department,Email from docreg where Department='%s'
   "%(z)

    data=pd.read_sql_query(sql,db)

    db.commit()

    print(data)

    if data.empty:

        flash('Doctot is not available')

        return redirect(url_for('view_appointments'))

    else:

        sql = "update patientreq set status='accepted' where status='pending' and
        Id='%s' and Name='%s'" % (x, y)

        cur.execute(sql)

        db.commit()

    return
    render_template('acptreq.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/Connect/<x>/<y>/<z>/<v>')

def mergereq(x=0,y="",z="",v=""):

    print(x)

    print(y)

    print(z)

    sql="select Id,name,Type,Age from patientreq where status='accepted' and
    Type='%s'"%(z)

    cur.execute(sql)

    da=cur.fetchall()

    db.commit()

    dat=[j for i in da for j in i]

    print(dat)

```

```

print(dat[0],dat[1],dat[3],dat[2])

sql="insert into
connectdata(Patientid,Patientname,patientAge,Type)values('%s','%s','%s','%s')"%(da
t[0],dat[1],dat[3],dat[2])

cur.execute(sql)

db.commit()

return redirect(url_for('view_appointments'))

@app.route('/doctorreg',methods=['POST','GET'])
def doctorreg():
    if request.method=='POST':
        dept=request.form['Department']
        name=request.form['Name']
        age=request.form['Age']
        number=request.form['Number']
        email=request.form['email']
        password=request.form['password']
        conpassword = request.form['conpassword']
        if password == conpassword:
            print("True")
            sql="select * from docreg"
            cur.execute(sql)
            data=cur.fetchall()
            db.commit()
            for i in data:
                if email in i[5]:
                    msg="Email already Exist's"

```

```

        return render_template('doctorreg.html',msg=msg)

    else:

        sql="insert into docreg(Name,Department,Age,Number,Email>Password)
values('%s','%s','%s','%s','%s','%s')"%(name,dept,age,number,email,password)

        cur.execute(sql)

        db.commit()

        msg="Your Request Sent to Management"

        return render_template('doctorreg.html',msg=msg)

    else:

        msg = "Password doesn't Match"

        return render_template('doctorreg.html', msg=msg)

return render_template('doctorreg.html')

@app.route('/Doc_requests')

def Docrequests():

    sql="select Name,Department,Age,Number,Email from docreg where
status='pending'"

    data=pd.read_sql_query(sql,db)

    db.commit()

    return
render_template('Doc.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/acpt_doc/<x>/<y>')

def acceptdoc(x="",y=""):

    sql="update docreg set status='accepted' where status='pending' and Name='%s'
and Email='%s'"%(x,y)

    cur.execute(sql)

    db.commit()

"""

```

Have to Complete Email Code

```
"""

sender_address = 'praveensankar3005@gmail.com'

sender_pass = 'bdlthdljmeijhofd'

content = "Your Request Is Accepted by the Management . You Can Login Now"

receiver_address = y

message = MIMEMultipart()

message['From'] = sender_address

message['To'] = receiver_address

message['Subject'] = "A updated, more straightforward guideline for exchanging
cloud based PHR"

message.attach(MIMEText(content, 'plain'))

ss = smtplib.SMTP('smtp.gmail.com', 587)

ss.starttls()

ss.login(sender_address, sender_pass)

text = message.as_string()

ss.sendmail(sender_address, receiver_address, text)

ss.quit()

return redirect(url_for('Docrequests'))

@app.route('/Docs')

def Docs():

    sql="select Name,Department,Age,number,Email from docreg where
status='accepted'"

    data=pd.read_sql_query(sql,db)

    db.commit()

    return
render_template("docs.html",cols=data.columns.values,rows=data.values.tolist())
```

```

@app.route('/doctor_log',methods=['POST','GET'])

def doctorlog():

    if request.method=='POST':

        Docname=request.form['Docname']

        passcode=request.form['Docpasscode']

        sql="select * from docreg where status='accepted' and name='%s' and
        Password='%s'"%(Docname,passcode)

        cur.execute(sql)

        data=cur.fetchall()

        db.commit()

        print(data)

        if data !=[]:

            email = data[0][-3]

            session['doc'] = email

            i=[i for i in data]

            session['dept']=i[0][2]

            print(session['dept'])

            # if Docname in i[0][1] and passcode in i[0][-2]:

            msg="Doctor Login Success"

            return render_template("docrequest.html",msg=msg)

        else:

            msg="Details doesn't exist"

            return render_template("doctorlog.html",msg=msg)

    return render_template('doctorlog.html')

```

```

@app.route('/view_patient')

def viewpatient():

    sql="select * from connectdata where Type='%s'"%(session['dept'])

    cur.execute(sql)

    data=cur.fetchall()

    db.commit()

    print(data)

    if data == []:

        msg="You dont have any appointments "

        return render_template("viewpatient.html",msg=msg)

    Name=data[0][2]

    Age=data[0][3]

    Type=data[0][4]

    return render_template('viewpatient.html',name=Name,age=Age,type=Type)

@app.route("/patient_access/<a>/<b>")

def patientaccess(a="",b=0):

    sql="select Email from patient_reg where Name='%s' and Age='%s'"%(a,b)

    cur.execute(sql)

    data=cur.fetchall()

    db.commit()

    print(data)

    if data !=[]:

        Email=data[0][0]

        session['email']=Email

        return render_template("uploadfile.html", email=Email)

```

```

"""Patient mail access code"""

msg="Your Appointment is accepted "

# return render_template("patientaccess.html", msg=msg)

return render_template("uploadfile.html")


@app.route('/upload_file',methods=['POST','GET'])
def uploadfile(email=""):

    print(email)

    if request.method=='POST':

        filedata=request.files['filedata']

        n=filedata.filename

        data=filedata.read()

        print(data)

        path=os.path.join("uploadfiles/", n)

        filedata.save(path)

        status="accepted"

        sql="insert into reports(FileName,FileData,patientEmail,Status)
values(%s,AES_ENCRYPT(%s,'sec_key'),%s,%s)"

        val=(n,data,session['email'],status)

        cur.execute(sql,val)

        db.commit()

        msg="file Uploaded successfully"

        return render_template('uploadfile.html',msg=msg)

    return render_template('uploadfile.html')


@app.route('/view_files')

```



```

def viewfiles():

    sql="select FileName,Filedata,PatientEmail from reports where PatientEmail='%s'
and status='accepted'""%(session['email'])

    data = pd.read_sql_query(sql, db)

    db.commit()

    return
render_template('files.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/performs')

def performs():

    sql="update reports set status='updated' where status='accepted' and
PatientEmail='%s'""%(session['email'])

    cur.execute(sql)

    db.commit()

    return redirect(url_for('viewfiles'))

    # return render_template('performs.html')

@app.route('/authority',methods=['POST','GET'])

def authority():

    if request.method=='POST':

        name = request.form['Username']

        password = request.form['passcode']

        if name=='Authority' and password =='auth':

            return render_template('authhome.html')

        return render_template('authority.html')

@app.route('/vr')

def vr():

    sql="select Id,FileName,PatientEmail from reports where status='updated'"

    data=pd.read_sql_query(sql,db)

```

```

db.commit()

return
render_template('vr.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/proxy_server',methods=['POST','GET'])

def proxyserver():

    if request.method=='POST':

        name = request.form['Username']

        password = request.form['passcode']

        if name=="proxy" and password == "server":

            return render_template('proxylog.html')

        return render_template('proxy.html')

@app.route('/Generate_Key/<c>/')

def generatekey(c=0):

    x = random.randrange(000000, 999999)

    print(x)

    print(c)

    sql = "update reports set Key1='%s',status='done' where Id = '%s' and
status='updated' " % (x, c)

    cur.execute(sql)

    db.commit()

    return redirect(url_for('vr'))

@app.route('/all_requests')

def allrequests():

    sql="select Id,FileName,PatientEmail,Key1 from reports where status='done' and
PatientEmail='%s'"%(session['email'])

    data=pd.read_sql_query(sql,db)

    db.commit()

```

```

    return
render_template('all.html',cols=data.columns.values,rows=data.values.tolist())

@app.route('/sentmail/<e>/<k>')

def sentmail(e="",k=0):

    sender_address = 'praveensankar3005@gmail.com'

    sender_pass = 'bdlthdljmeijhofd'

    content=str(k)

    print(content)

    receiver_address = e

    message = MIMEMultipart()

    message['From'] = sender_address

    message['To'] = receiver_address

    message['Subject'] = "A Lightweight Policy Update Scheme for Outsourced
Personal Health Records Sharing project started"

    message.attach(MIMEText(content, 'plain'))

    ss = smtplib.SMTP('smtp.gmail.com', 587)

    ss.starttls()

    ss.login(sender_address, sender_pass)

    text = message.as_string()

    ss.sendmail(sender_address, receiver_address, text)

    ss.quit()

    sql = "update reports set status='complete' where status='done' and
PatientEmail='%s'" % (session['email'])

    cur.execute(sql)

    db.commit()

    return redirect(url_for("allrequests"))

```

```

@app.route('/view_report',methods=['POST','GET'])

def viewreport():

    try:

        sql="select * from reports where status='done' and
PatientEmail='%s'"%(session['email'])

        data=pd.read_sql_query(sql,db)

        db.commit()

        if request.method=='POST':

            keyvalue=request.form['keycvalue']

            return render_template('views.html',data=data)

        return
render_template('report.html',cols=data.columns.values,rows=data.values.tolist())

    except:

        msg="Your reports are not available"

        return render_template('patienthome.html',msg=msg)

@app.route('/logout')

def logout():

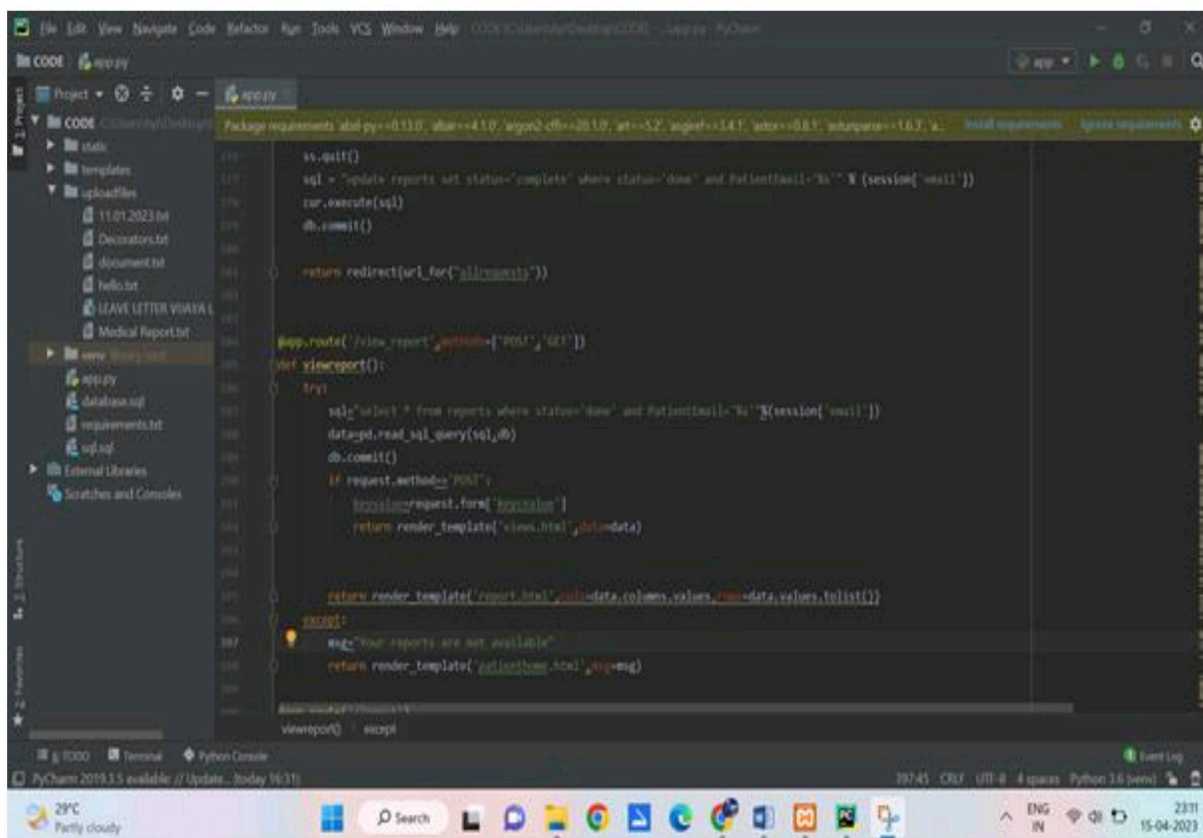
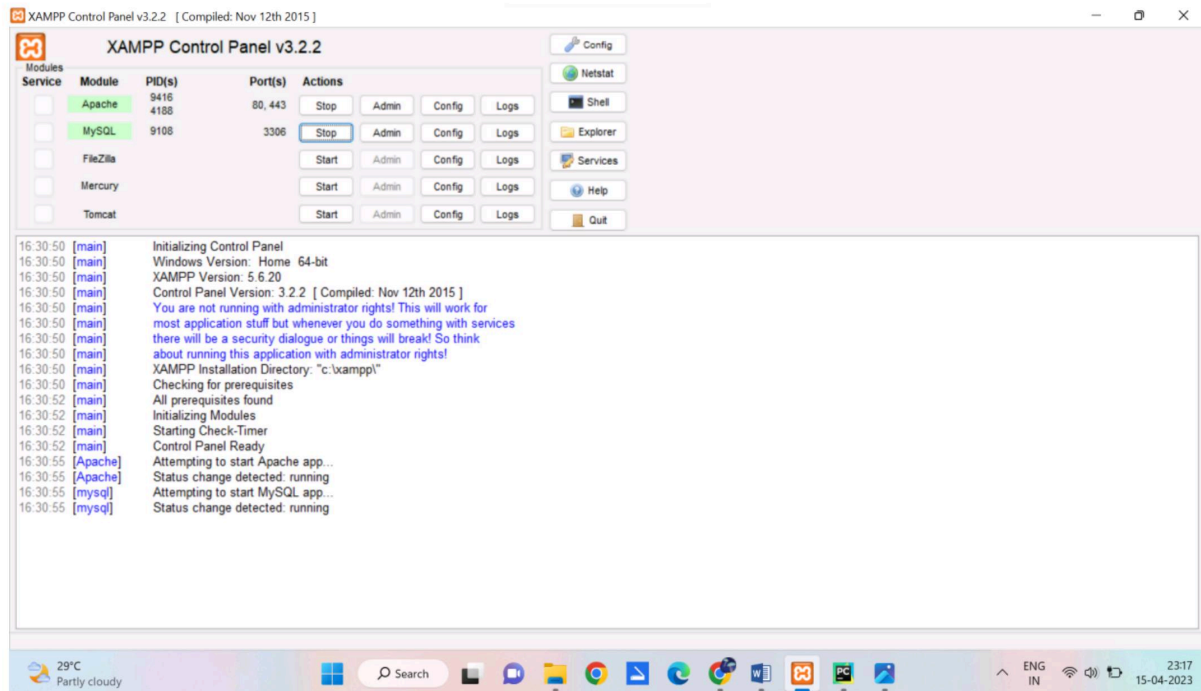
    return redirect(url_for('/'))

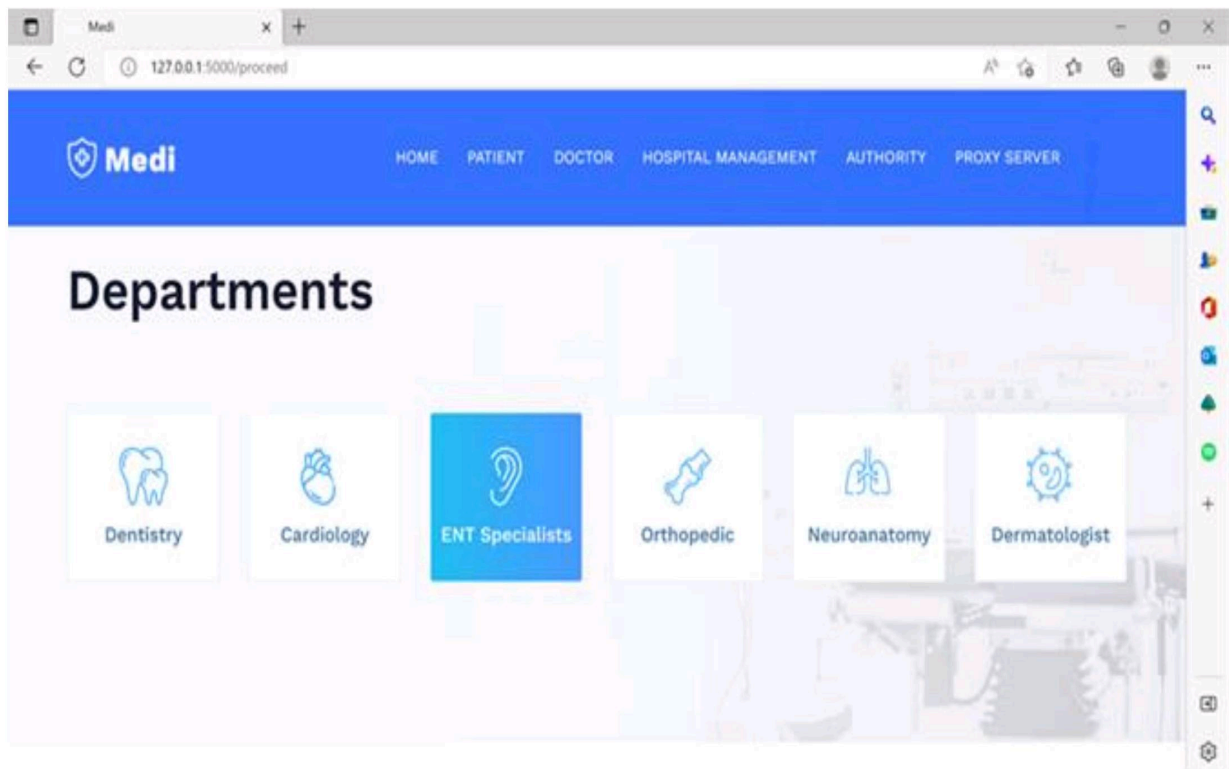
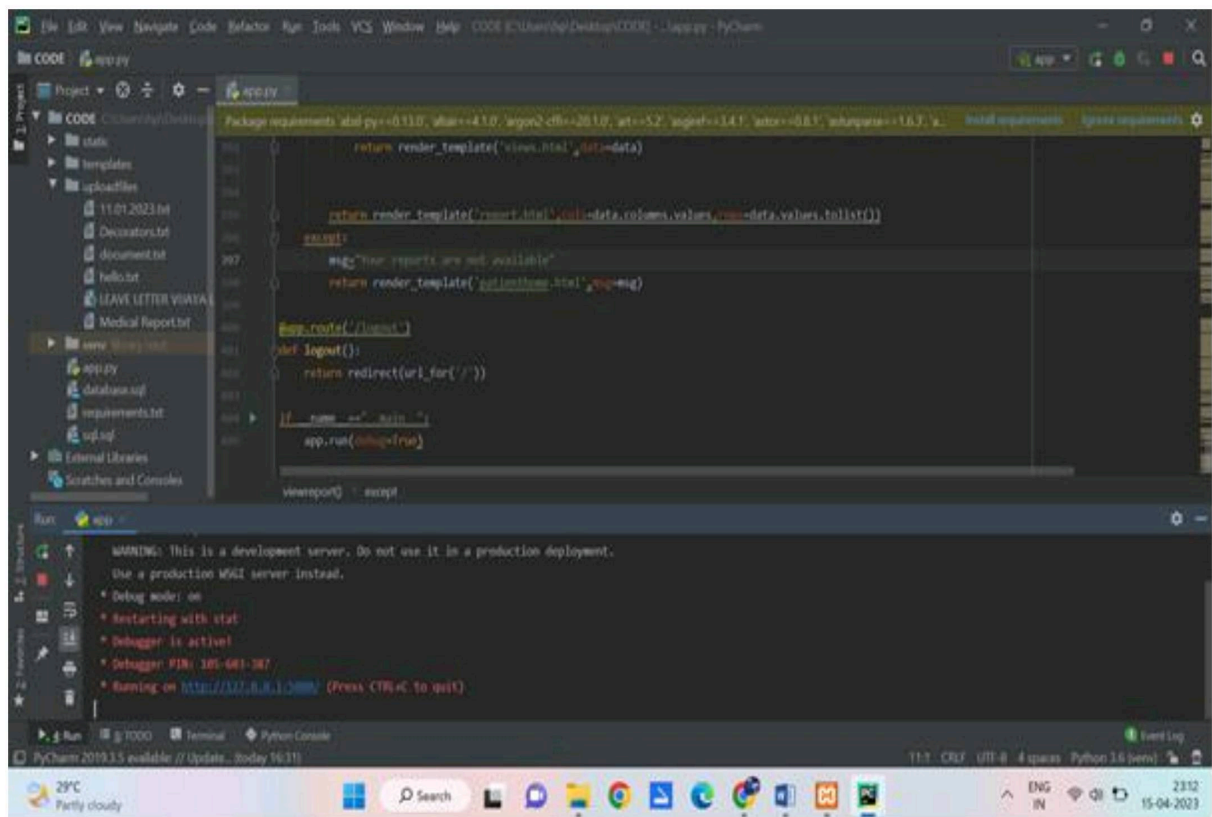
if __name__=="__main__":

    app.run(debug=True)

```

B. SCREEN SHOTS





The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/patient". The browser's title bar shows "Medi". The page has a blue header with the "Medi" logo on the left and a "REGISTRATION" link on the right. The main content area features a background image of a person in a white lab coat. Overlaid on this is a white "PATIENT LOGIN FORM". The form contains two input fields: "YOUR NAME" and "PASSWORD", followed by a blue button labeled "Click to Login".

Patient Login Form

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/hospitalmanagement". The browser's title bar shows "Medi". The page has a blue header with the "Medi" logo on the left. The main content area features a background image of a person in a white lab coat. Overlaid on this is a white "Hospital Management Login" form. The form contains two input fields: "Profile" and "Password", followed by a blue button labeled "Login".

Hospital Management Login

Medi x +

127.0.0.1:5000/doctor_log

Home Register

Doctor Login Form

Your name

Password

Login

Medi x +

127.0.0.1:5000/patient

Home View Report Logout

Appointment

Arjun

Pulmonary

20 cough

11/23/2022 12:00

Make An Appointment

Plagiarism Checker | 100% Free x Medi x +

127.0.0.1:5000/Doc_requests

Medi Appointments Doctor Requests Doctors Logout

Doctor Requests

Name	Department	Age	Number	Email	Action
Praveen	Pulmonary	56	9876543210	hosiptal@gmail.com	Accept

26°C Mostly cloudy

Search

ENG IN 03:35 01-02-2023

SQLyog Enterprise - MySQL GUI - [New Connection - root@localhost]

File Edit Favorites DB Table Objects Tools Powertools Window Help

information_schema
Tables
Views
Stored Procs
Functions
Triggers
Events
mysql
performance_schema
phpmyadmin
phr

connectdata
docreg
patient_req
Columns
ID [int(200), NOTNULL]
Name [varchar(100), NULL]
Age [varchar(100), NULL]
Email [varchar(100), NULL]
Password [varchar(100), NULL]
Indexes
patientreq
Columns
Indexes
Id, PRIMARY
reports
Views
Stored Procs
Functions
Triggers

Query QueryBuilder SchemaDesigner

Autocomplete: [Tab]->Next Item. [Ctrl+Space]->List Matching Items. [Ctrl+Enter]->List All Items.

```
1 information_schemaphmyadminphmyadminphrconnectdataconnectdatapatient_reqtestpatientreq
```

1 Result 2 Messages 3 Table Data 4 Objects 5 History

Show All or Limit 0 50 Refresh

Id	Patientid	PatientName	PatientAge	Type
1	2	praveen	20	Pulmonary
*	(NULL)	(NULL)	(NULL)	(NULL)

Rebuilding tag file completed 0 ms 1 row(s) Connections: 1 Registered To: TaMaMBolo

29°C Partly cloudy

Search

ENG IN 23:15 15-04-2023

SQLyog Enterprise - MySQL GUI - [New Connection - root@localhost]

File Edit Favorites DB Table Objects Tools Powertools Window Help

phr

information_schema
Tables
Views
Stored Procs
Functions
Triggers
Events
mysql
performance_schema
phpmyadmin
phr
Tables
connectdata
docreg
patient_reg
Columns
ID [int(200), NOTNULL]
Name [varchar(100), NULL]
Age [varchar(100), NULL]
Email [varchar(100), NULL]
Password [varchar(100), NULL]
Indexes
patientreq
Columns
Indexes
Id, PRIMARY
reports
Views
Stored Procs
Functions
Triggers

Query QueryBuilder SchemaDesigner

Autocomplete: [Tab]->Next Item. [Ctrl+Space]->List Matching Items. [Ctrl+Enter]->List All Items.

```
1 information_schemaphmyadminphmyadminphrconnectdataconnectdatapatient_regtestpatientreq
```

1 Result 2 Messages 3 Table Data 4 Objects 5 History

Show All or Limit 0 50 Refresh

idno	Name	Department	Age	Number	Email	Password	status
1	prithivi	Dermatologist	45	9876543210	prithiviring@gmail.com	987654321	accepted
2	charu	Pulmonary	40	9876543210	charulatha@gmail.com	987654321	accepted
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	pending

Rebuilding tag file completed 0 ms 2 row(s) Connections: 1 Registered To: TaMaBoLo

29°C Partly cloudy Search ENG IN 23:15 15-04-2023

SQLyog Enterprise - MySQL GUI - [New Connection - root@localhost]

File Edit Favorites DB Table Objects Tools Powertools Window Help

phr

information_schema
Tables
Views
Stored Procs
Functions
Triggers
Events
mysql
performance_schema
phpmyadmin
phr
Tables
connectdata
docreg
patient_reg
Columns
ID [int(200), NOTNULL]
Name [varchar(100), NULL]
Age [varchar(100), NULL]
Email [varchar(100), NULL]
Password [varchar(100), NULL]
Indexes
patientreq
Columns
Indexes
Id, PRIMARY
reports
Views
Stored Procs
Functions
Triggers

Query QueryBuilder SchemaDesigner

Autocomplete: [Tab]->Next Item. [Ctrl+Space]->List Matching Items. [Ctrl+Enter]->List All Items.

```
1 information_schemaphmyadminphmyadminphrconnectdataconnectdatapatient_regtestpatientreq
```

1 Result 2 Messages 3 Table Data 4 Objects 5 History

Show All or Limit 0 50 Refresh

ID	Name	Age	Email	Password
1	sakthi	23	sakthiganesh070@gmail.com	987654321
2	praveen	20	praveensankar3005@gmail.com	987654321
3	raj ganesh	30	rajganesh@gmail.com	987654321
4	praveens	25	praveen@gmail.com	987654321
*	(NULL)	(NULL)	(NULL)	(NULL)

Rebuilding tag file completed 0 ms 4 row(s) Connections: 1 Registered To: TaMaBoLo

29°C Partly cloudy Search ENG IN 23:15 15-04-2023

