CSE 545: Software Security

Secure Hospital System

Design Document – Group 9

Yash Bhokare (L)	Kiran Shanthappa (DL)	Anthony Mac
Sai Manoja Gadde	Shashank Baradi	Karan Naik
Jayanth Goritha	Bhaskara Atluri	Rajesh Badugula

Arizona State University

Table of Contents

1. Introduction

With advances in healthcare informatics, one can provide better means to process patient records and therefore speed up the treatment, which in turn reduces the overall cost. Many tools exist for facilitating patient record processing: from assisting data entry to manipulating records, from generating output in required form to transferring it to other physicians for further examination, or to save it digitally for future use. The project adopts waterfall implementation methodology. This design document incorporates a software engineering approach during development, and incorporates many aspects from database design, web deployment, and security. While developing our design, we follow formal patient privacy requirements for user account management and secure transactions. Also adopts ML based chat bot and malicious login prevention, to provide a secure hospital management system.

Electronic medical records (EMRs) contain an individual patient's extremely sensitive confidential information including identifying information and personal medical records. However, these EMRs are created, maintained, and stored across isolated hospital and insurance providers creating "data silos" that cause major patient linkage and data sharing problems in healthcare. This work extends Hyperledger Fabric as a solution to overcome the problems of traditional records management in hospital/medical system such as accessibility, security, and transaction/approval records.

1.1 Scope

The aim of the course project is to develop a skeleton secure hospital system (SHS) with limited functional, performance, and security requirements for secure hospital management, user account management and secure transactions. Multiple users should be able to securely use this system from any place and at any time with the availability of Internet access and web browser.

Hosting the secure hospital system on AWS with minimal resource to highlight the functionality and security design learnt throughout the course. The free tier AWS i.e., micro, or small EC2 instance will be used for hosting the application. This system will not be capable of diverting DDoS attacks.

2. Overview

SHS is a greenfield development requiring requirement analysis with design and development in a new clean environment. Various security measures such as communication over secure channel, data encryption, session management, secure login with ML based malicious login prevention and, blockchain are considered at software design to reduce vulnerabilities in the system.

3. System Design

SHS can be divided into a 3-tier architecture, involving web, app, and data tier. All tiers will be hosted on AWS. The apache server is used for hosting the web and app tier components. The data tier has two components for storing the hospital data. Postgre SQL for storing the hospital data and Hyperledger blockchain to capture the approved transaction details.

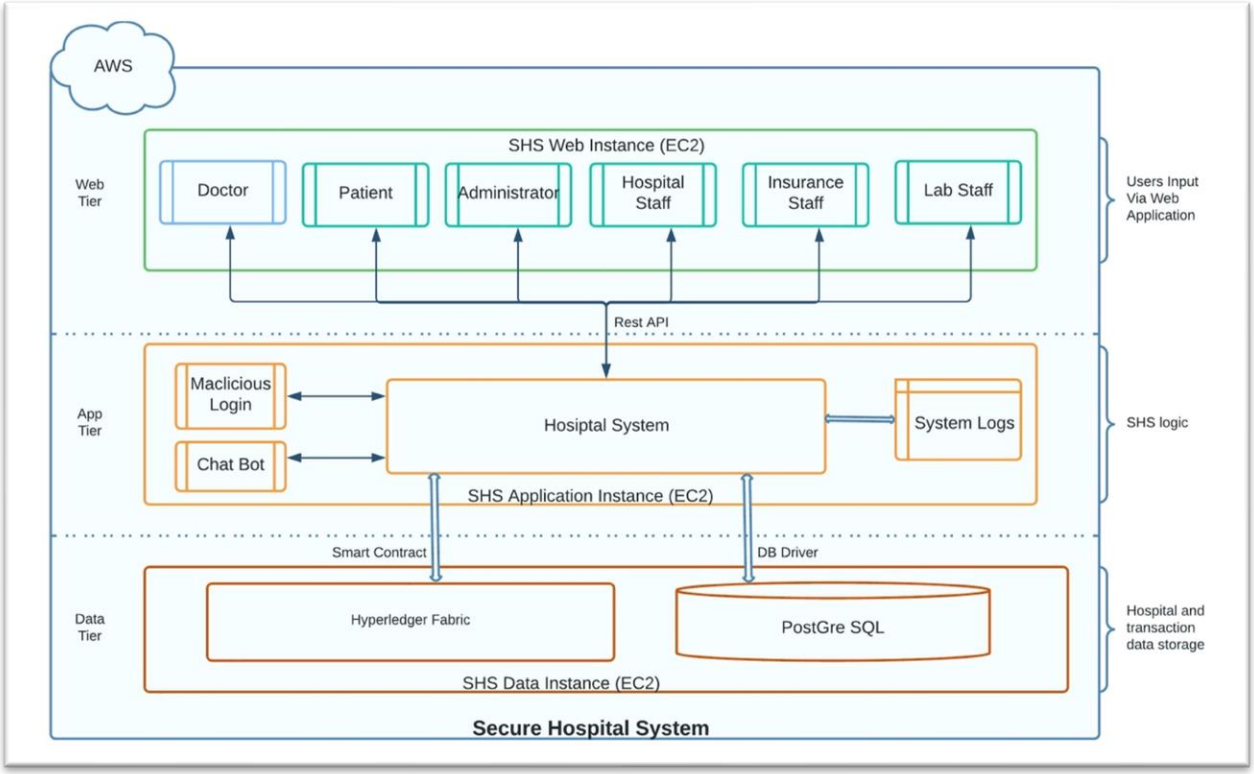


Fig.1 Secure Hospital System Architecture

The Web application are hosted as website. Web pages are designed using angular, as it is highly preferable as a robust frontend tool supplying components that assist people to write easy-to-use, readable, and maintainable code. Angular has built-in support to help prevent two common HTTP vulnerabilities, cross-site request forgery (CSRF or XSRF) and cross-site script inclusion (XSSI). Both must be mitigated primarily on the server-side, but Angular provides helpers to make integration on the client-side easier.

A security group acts as a virtual firewall for SHS EC2 instances to control incoming and outgoing traffic. Inbound rules control the incoming traffic to your instance, and outbound rules control the outgoing traffic from your instance. When component launched as an instance, you can specify same security groups. All components access across tier will be through secure protocol such as HTTPS. The secure hospital system will use a certificate for the web application for accessing webpages. A key pair, consisting of a public key and a private key, is a set of security credentials that you use to prove your identity when connecting to an Amazon EC2 instance. Amazon EC2 stores the public key on your instance, and you store the private key.

Blockchain technology is based on the principle of providing an individual control over their data and information, and hence potentially appropriate as part of a solution to patient ownership of their health data. Blockchain can create a decentralized identification that allows health care providers and patients to interact with one another directly without the need for an intermediary. The immutable audit trail which allows all changes of personal information to be tracked and traced. The increased security of patient's medical information. Not only is the data stored in multiple nodes across the distributed ledger; but it can also be encrypted with the only key being in the hands of the patient. Due to limited functionality the feature of key with patient will be considered as future work. Blockchain applications may, therefore, provide solutions to the current problems of interoperability of medical records, incomplete patient data at the point of service, integrate insurance claim, and lack of access to personal records while ensuring security.

3.1 Data Management

The data in SHS can be divided into sensitive and regular. The sensitive data represents the personal data of the patients such as SSN, Insurance number, and phone number or user password. The patient personal data are prevented from direct view by unprivileged users by data masking. The user password is encrypted and stored to prevent any hackers from easily accessing password. All other data are considered as regular data and stored as raw information in the database.

The various transaction and user data are captured through the data in the Database. The relation between tables can be seen in the ER diagram below. Here Postgre SQL will be hosted in AWS EC2 instance. With the access from the application for data processing.

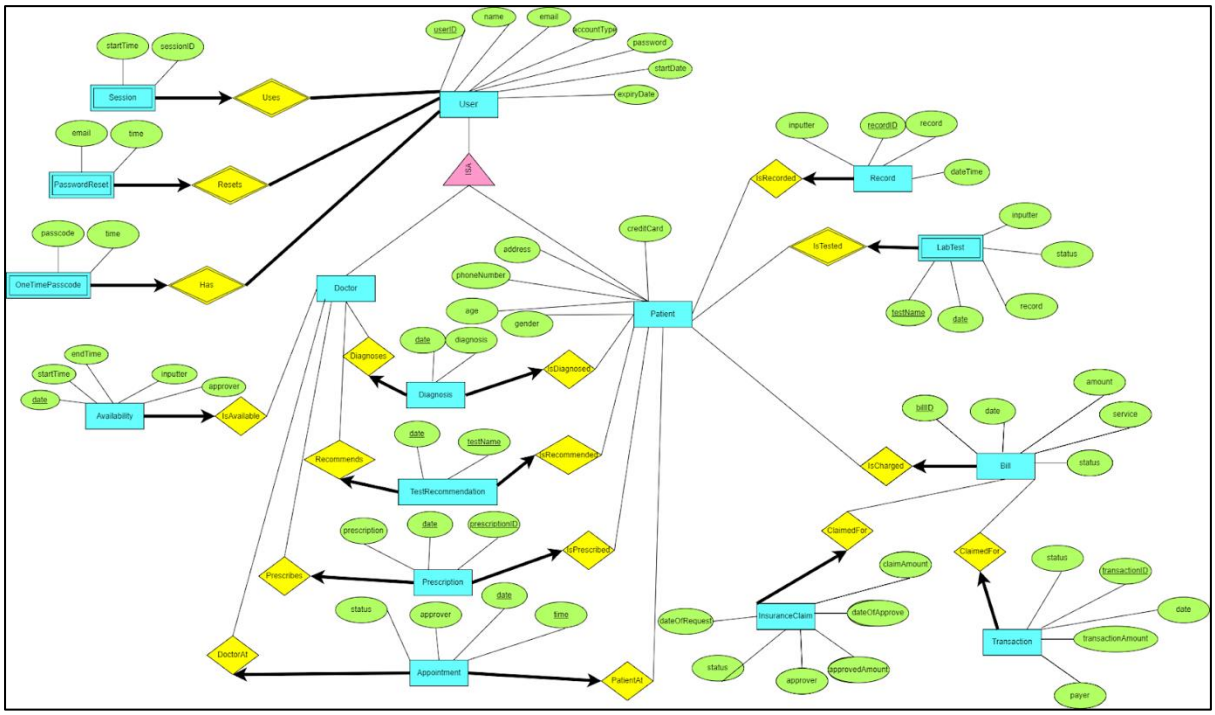


Fig.2 SHS – ER Diagram

3.2 Hyperledger Network

The network consists of infrastructure from organisation R1 and organization R2. The initial request to input request is considered as the A1 and approval request will be considered as A2. P1 and P2 represents the peer of organisation R1 and R2, respectively. CA1 and CA2 represents the certificate of organisation R1 and R2, respectively. L1 and L2 represents the ledger of organisation R1 and R2, respectively. O4 is the ordering service with CA4 as its certificate. All the instances hosted on a micro instance of AWS EC2.

A smart contract S5 has been installed onto P1. Client application A1 in organization R1 can use S5 to access the ledger via peer node P1. A1, P1 and O4 are all joined to channel C1 i.e., they can all make use of the communication facilities provided by that channel. Specifically, R2 has added peer node P2, which hosts a copy of ledger L1, and chaincode S5. R2 approves the same chaincode definition as R1. P2 has also joined channel C1, as has application A2. A2 and P2 are identified using certificates from CA2. All of this means that both applications A1 and A2 can invoke S5 on C1 either using peer node P1 or P2.

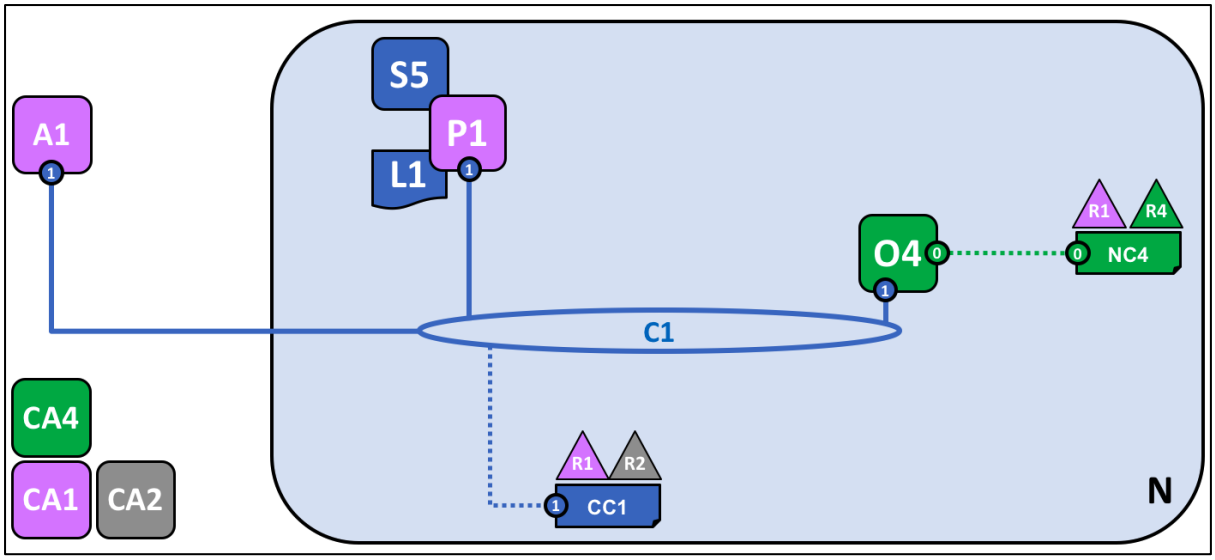


Fig.3 Blockchain Network for SHS

4. Detailed Design

The use flow of secure hospital transaction requests and response can be understood with below data flow diagram and class diagram.

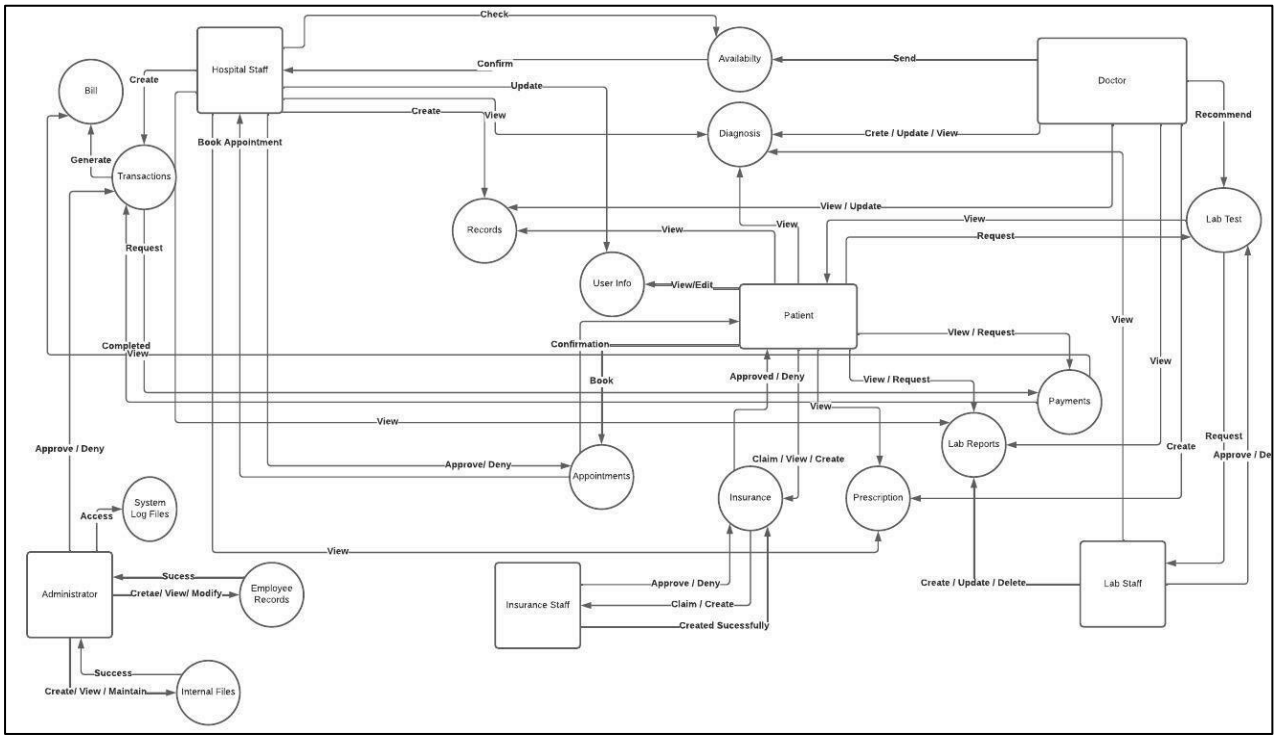


Fig.4 SHS – Data Flow Diagram

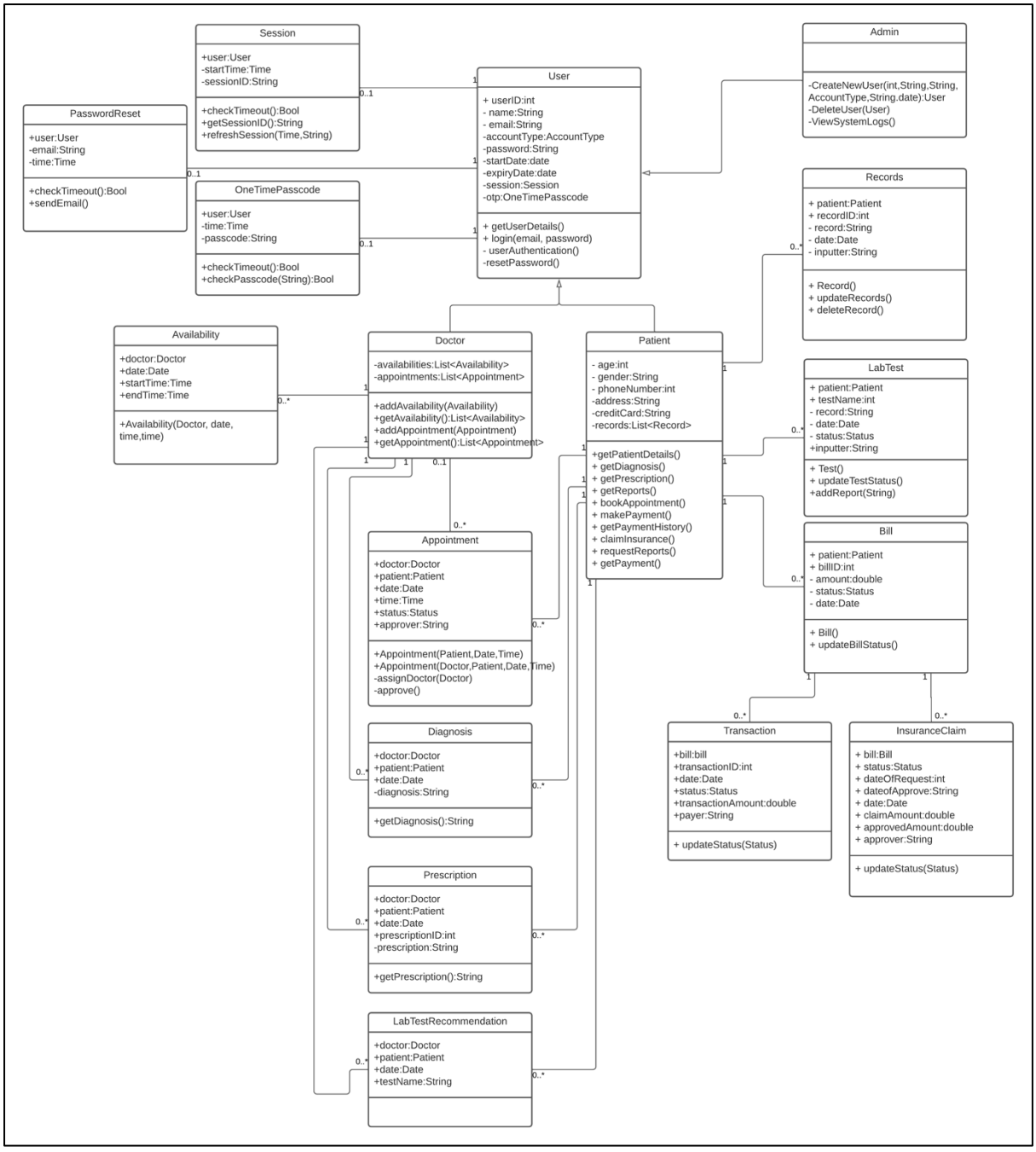


Fig.5 SHS – Class Diagram

4.1 Hospital Staff Sequence Diagram

Hospital staff are at the centre of the transaction management and engage in smooth functionality of the hospital digital system. They can approve appointment requests based on doctor availability, create patient records, update personal information of patient on request by patient, view patient diagnosis, view prescription, view lab test reports, create transaction requests, complete transactions upon approval and, create receipts and bills.

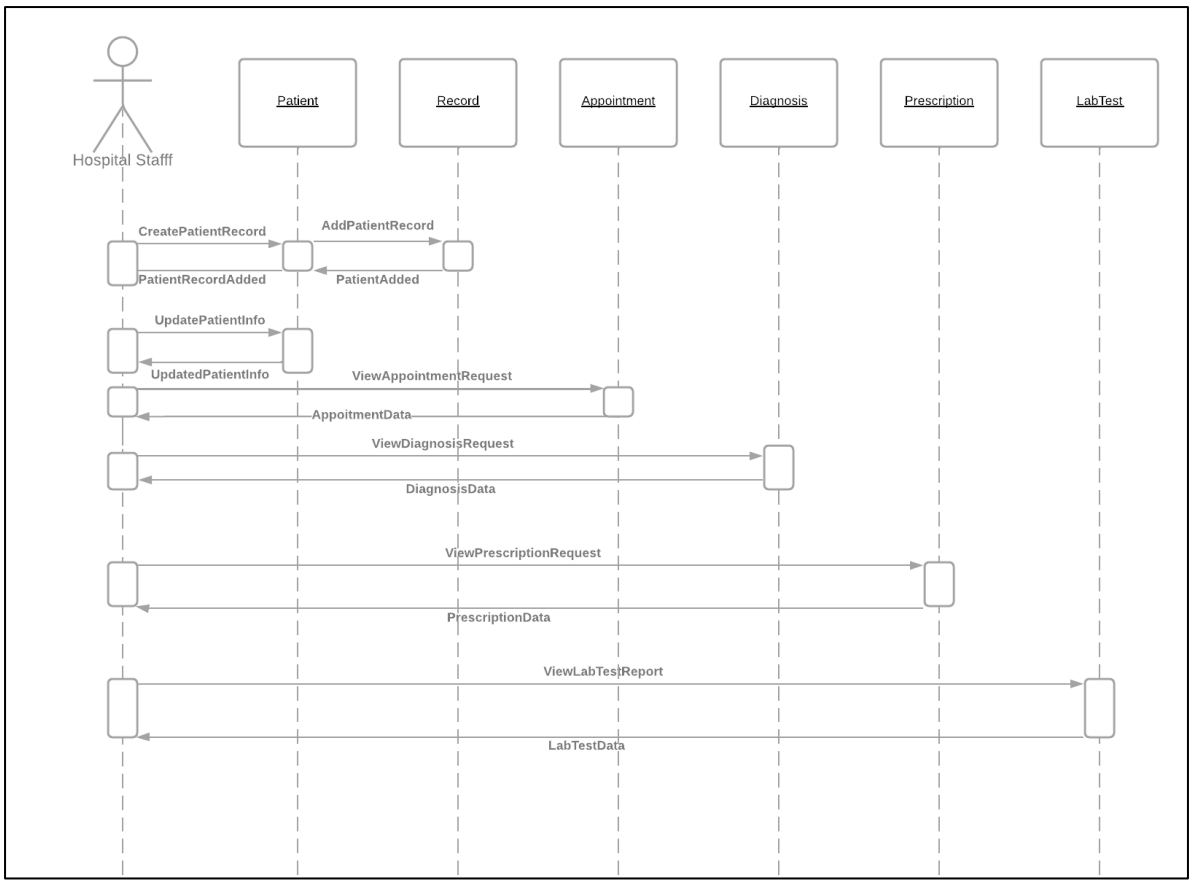


Fig.6 Hospital Staff Sequence diagram

4.1.1 Appointment Approval

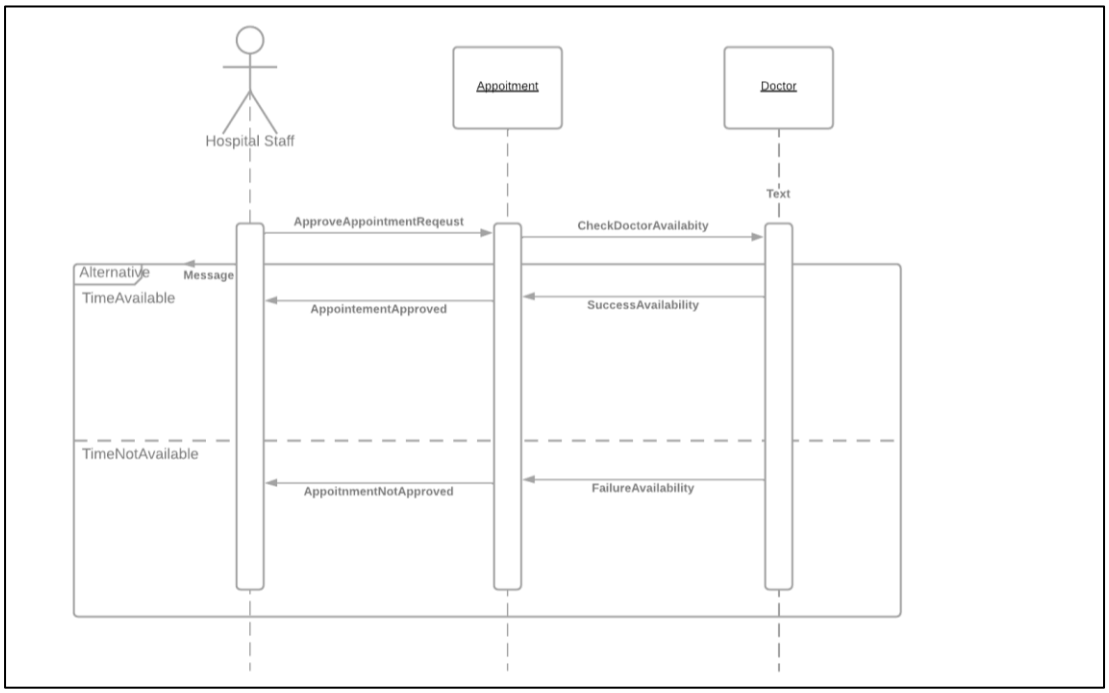


Fig.7 Appointment approval by hospital staff – Sequence diagram

4.1.2 Transaction Approval

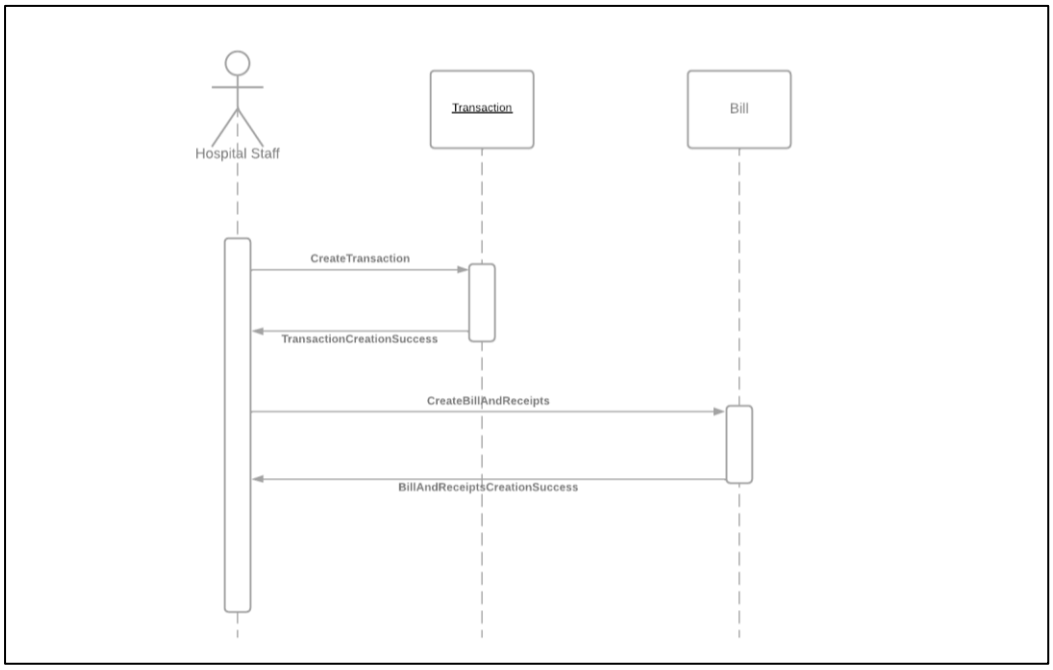


Fig.8 Hospital staff transaction approval – Sequence diagram

4.2 Patient Sequence Diagram

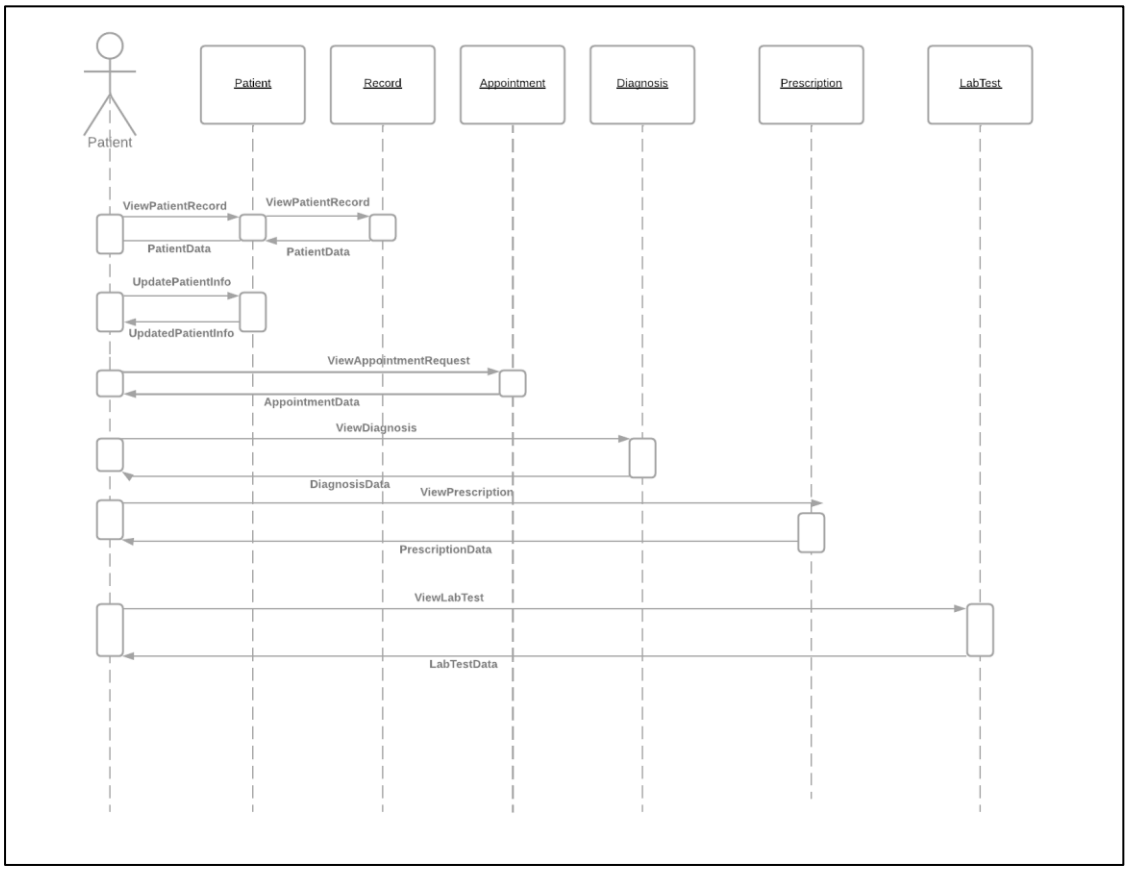


Fig.9 Patient Sequence diagram

4.2.1 Appointment Request and Approval

Patient initiates appointment request. Hospital staff can approve or deny the request based on availability.

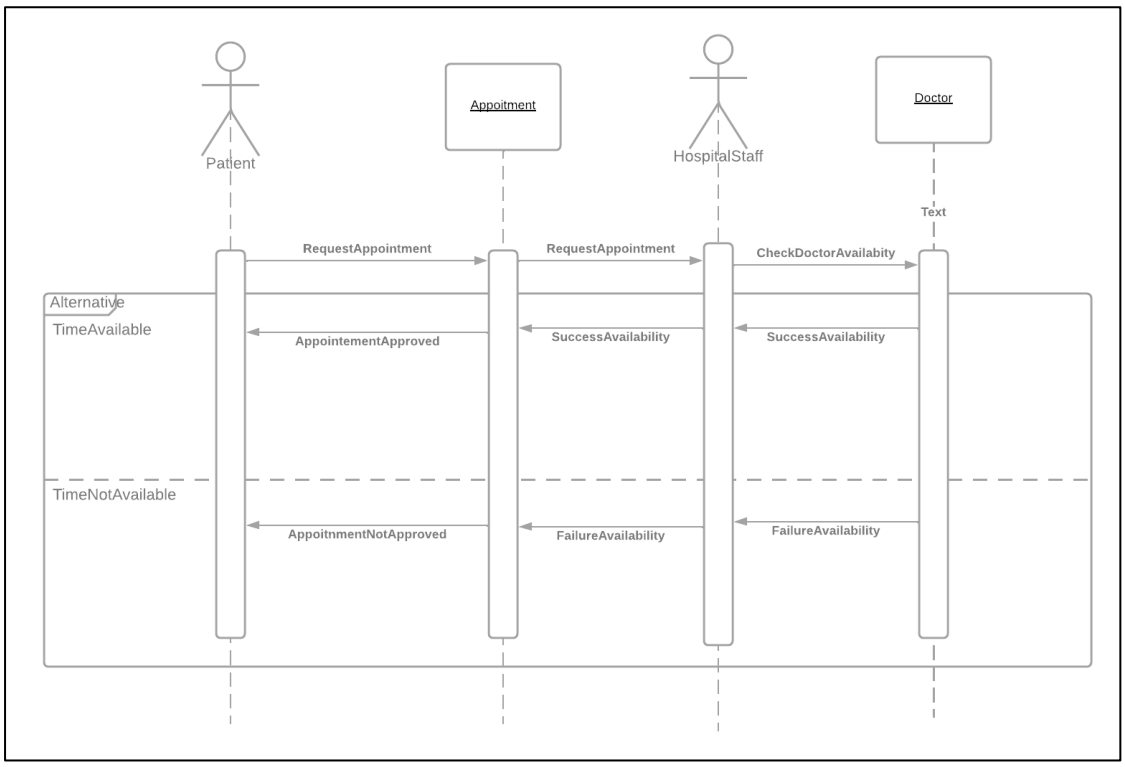


Fig.10 Appointment request and approval – Sequence diagram

4.2.2 Insurance Claim and Bill payment

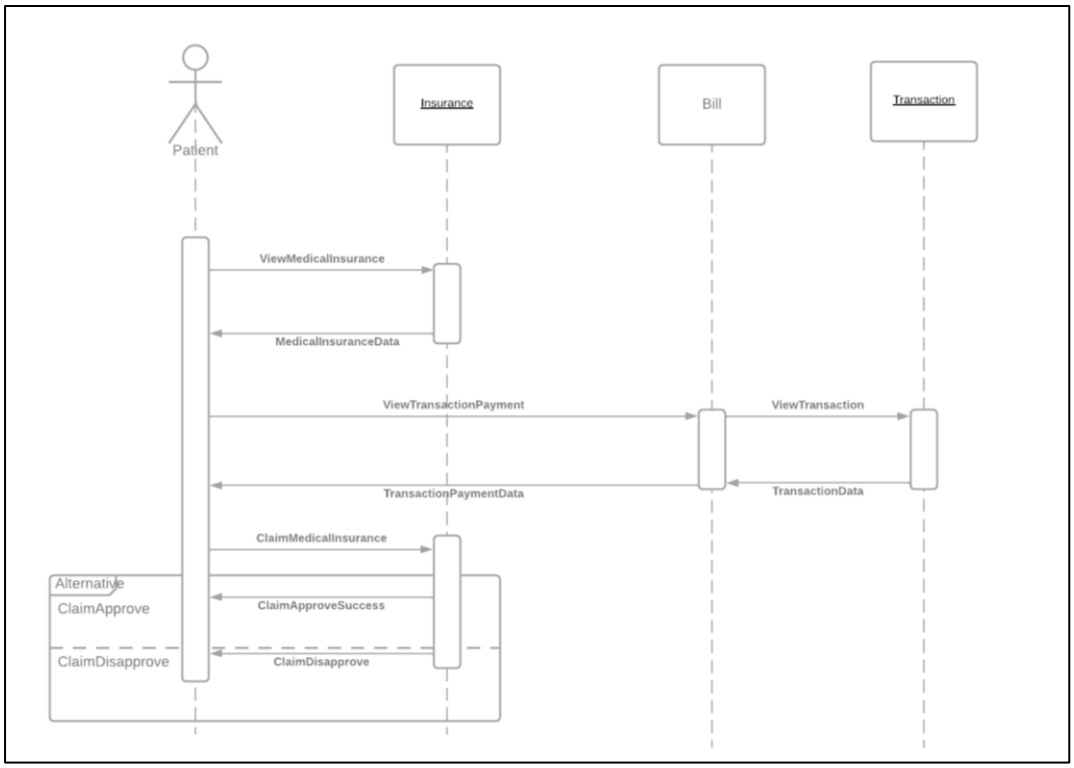


Fig.11 Insurance claim request and Bill payment – Sequence diagram

4.3 Doctor Sequence Diagram

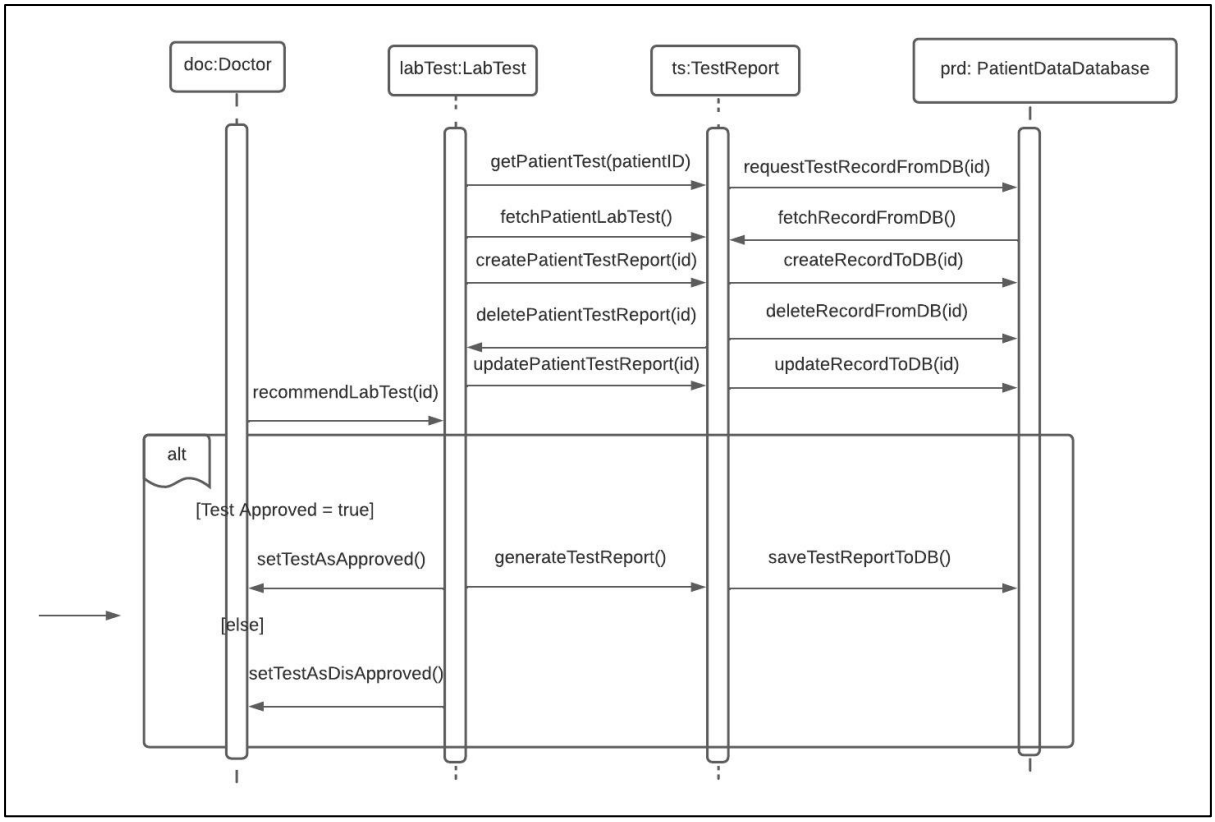


Fig.12 Lab test request processing – Sequence diagram

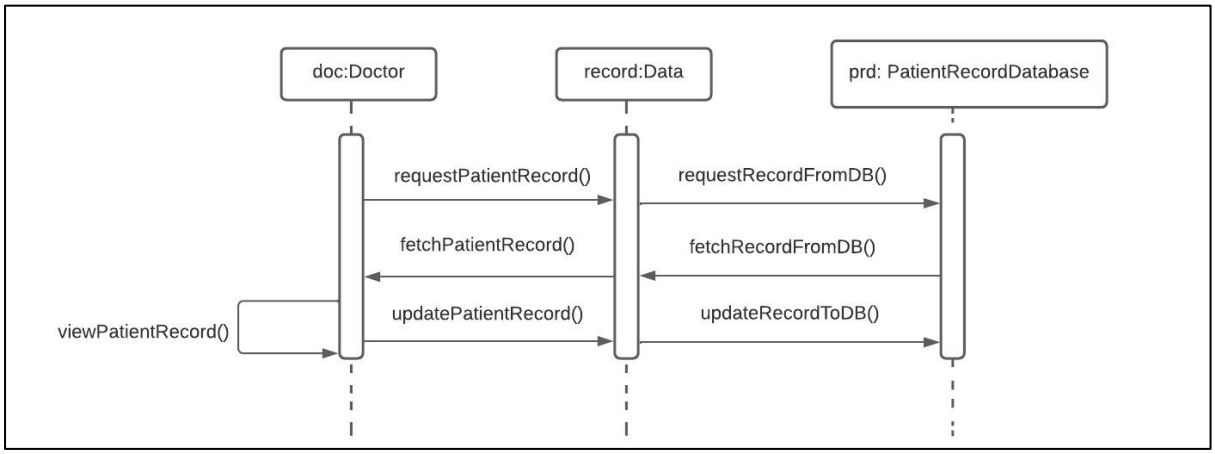


Fig.13 View patient history by doctor – Sequence diagram

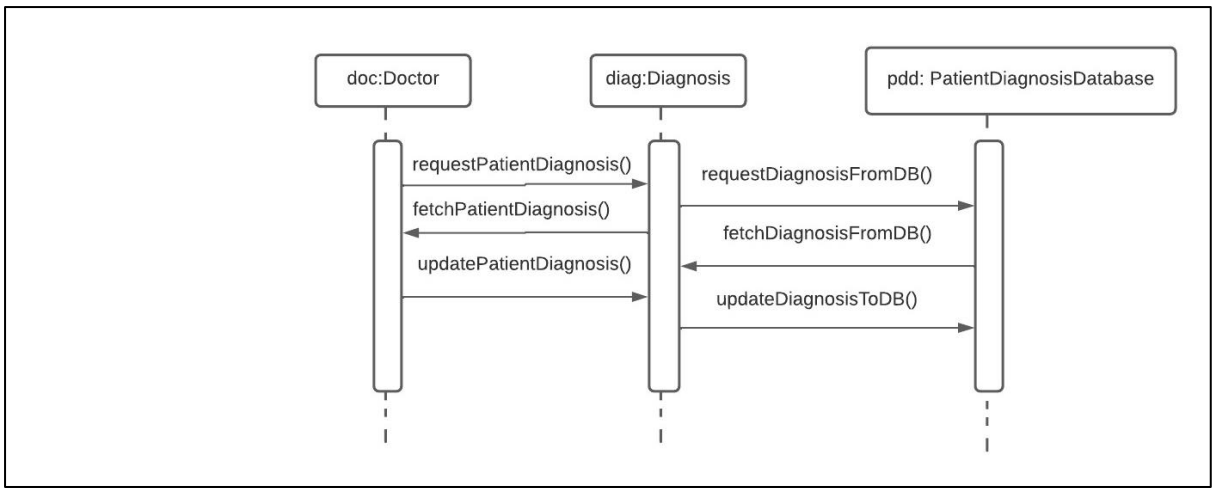


Fig.14 Patient diagnosis – Sequence diagram

4.4 Insurance Staff Sequence Diagram

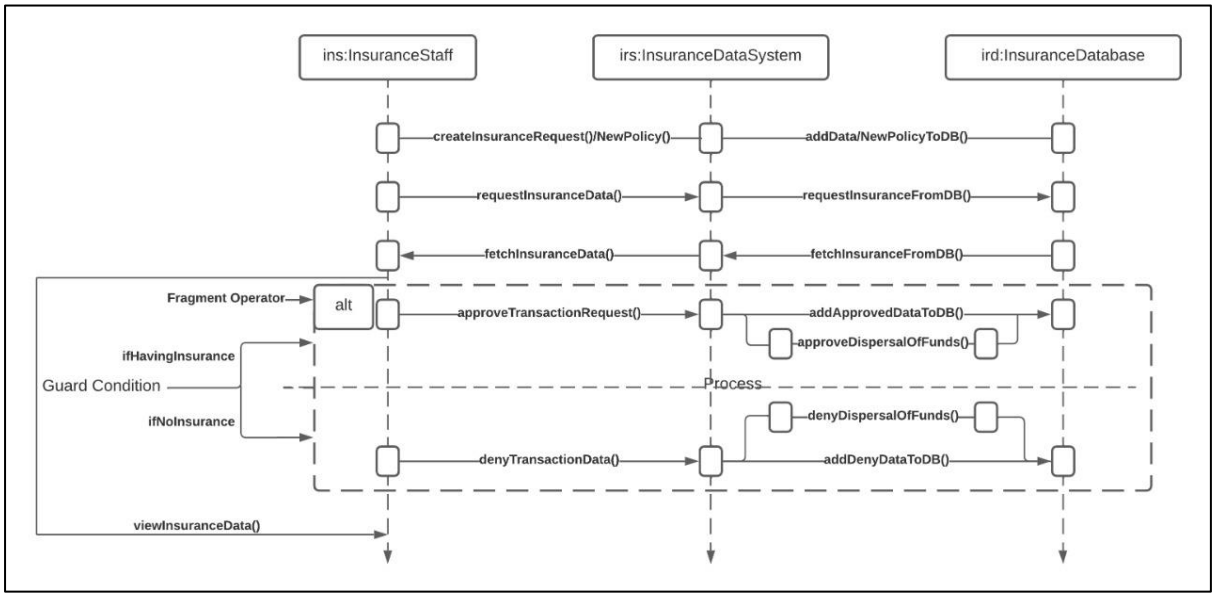


Fig.15 Insurance claim process – Sequence diagram

4.5 Administrator Sequence Diagram

The administrator can create, view, modify, and delete employee records. He maintains all internal files. Authorize (approve or deny) transaction requests. He is provided system user to access system log files (System log files are only accessible by administrators).

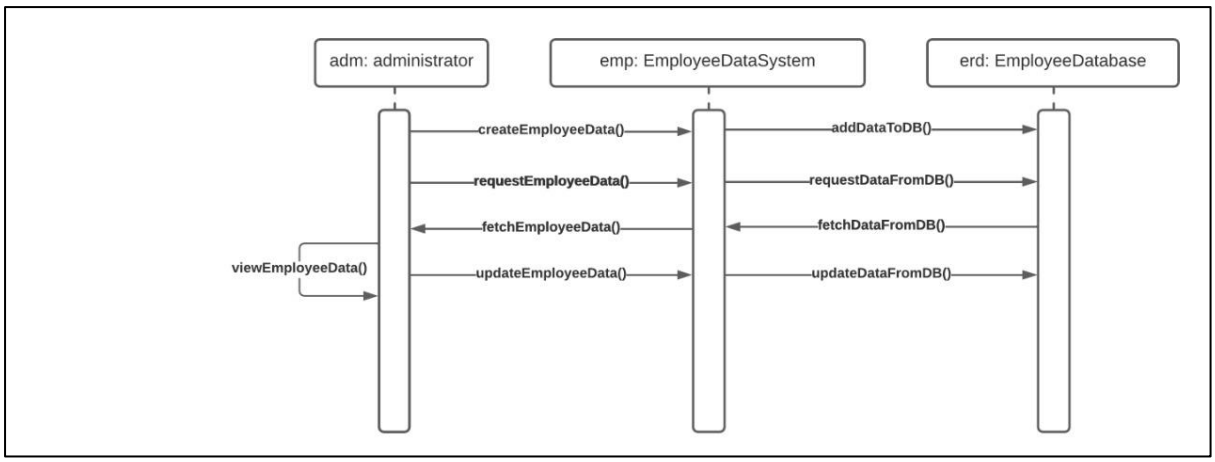


Fig.16 Administrator access to employee data – Sequence diagram

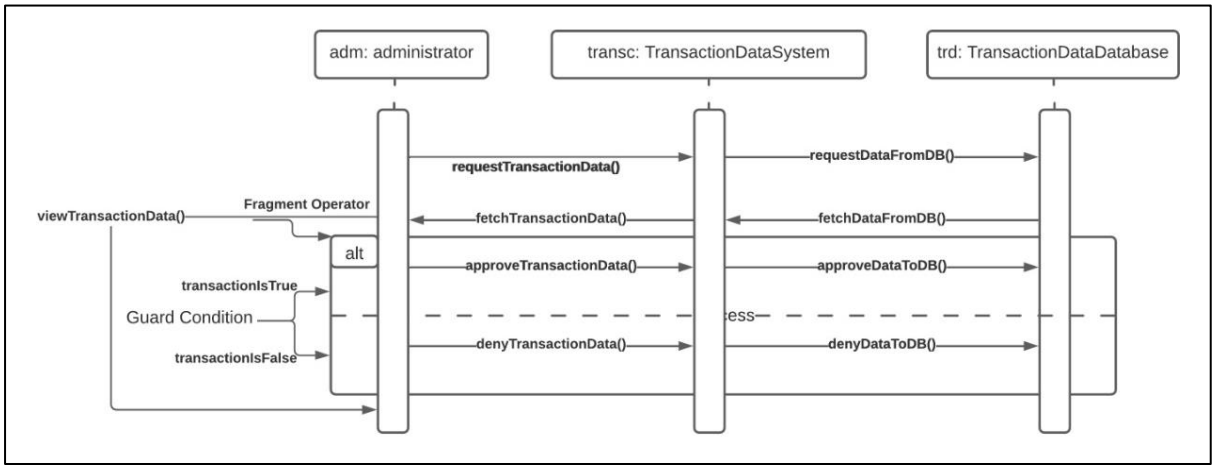


Fig.17 Administrator access to transaction data – Sequence diagram

4.6 Help & Support Centre

Web page to provide the emergency contact details with location. FAQ to include:

1. Instruction for appointment booking.
2. Insurance Claim
3. Facilities available at the Hospital

4.7 Chatbot

We are using an Amazon Lex service with advanced natural language models to design, build, test, and deploy conversational interfaces in our application.

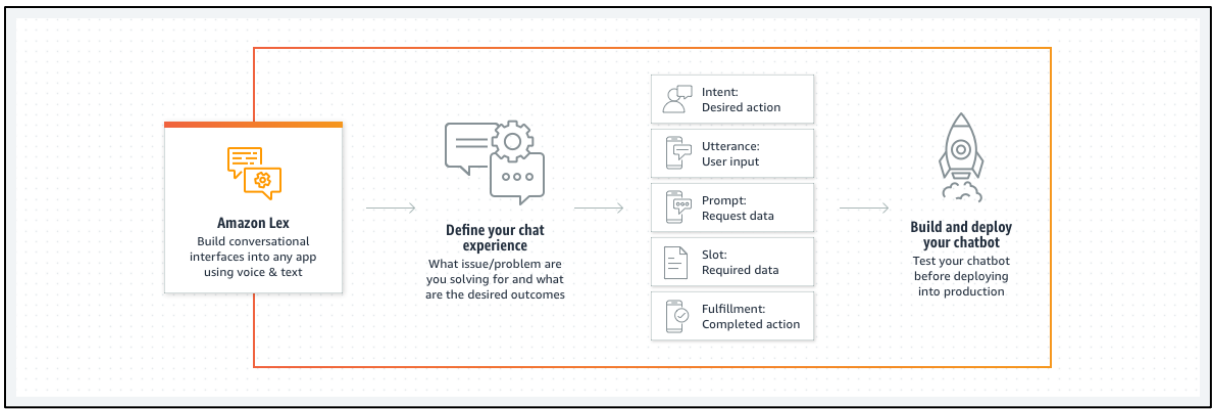


Fig.18 Chat bot – data flow diagram

Amazon Lex follows the basic chatbot architecture adhered to by all the big cloud based Conversational AI providers. We will need to create intents, entities, script (dialog wording), and then the dialog flow.

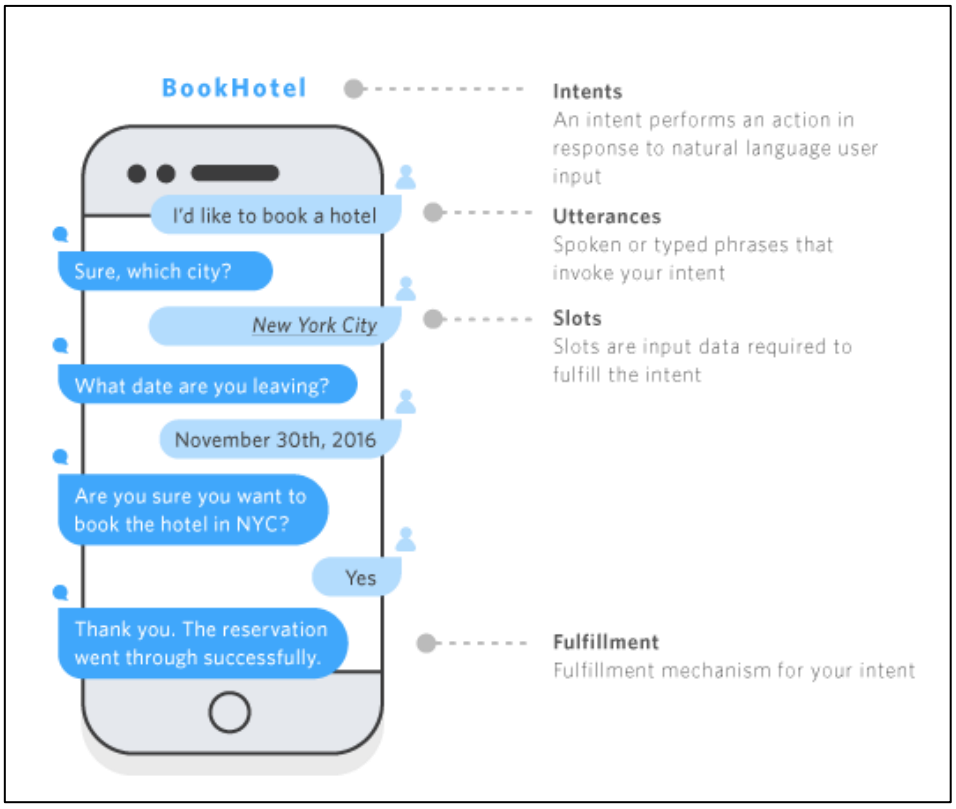


Fig.19 Sample of chat bot

5. User Interface

Few wireframes design for the secure hospital functionality is provided below as a reference:

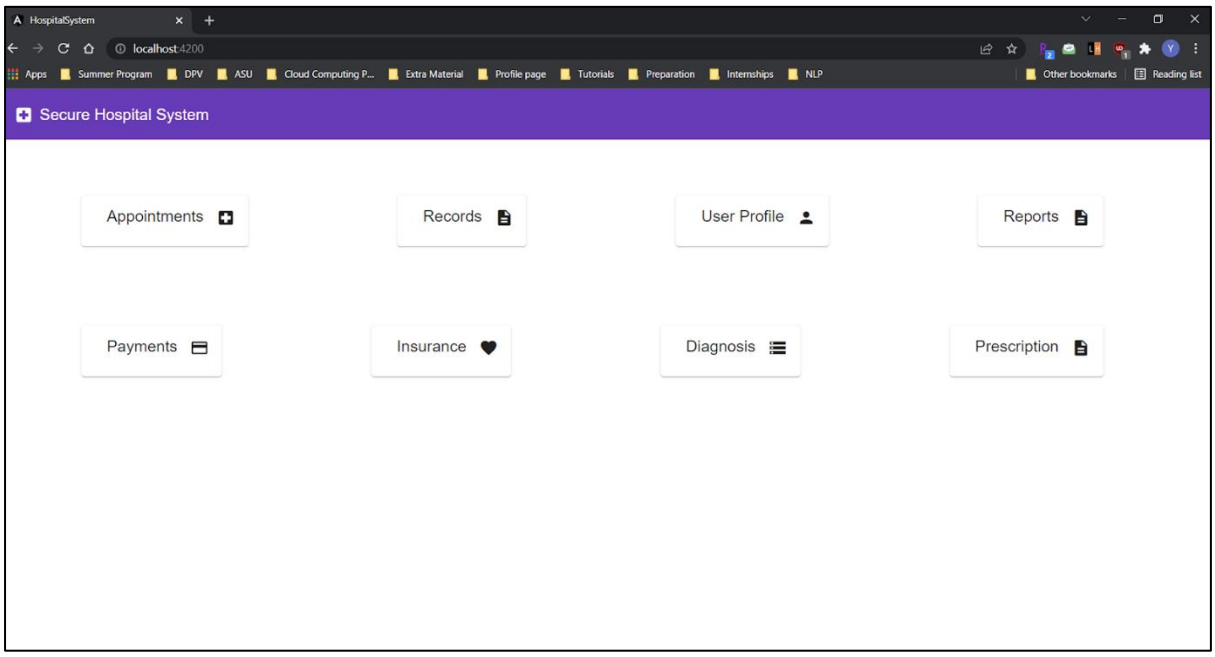


Fig.20 Wireframe Design - Dashboard

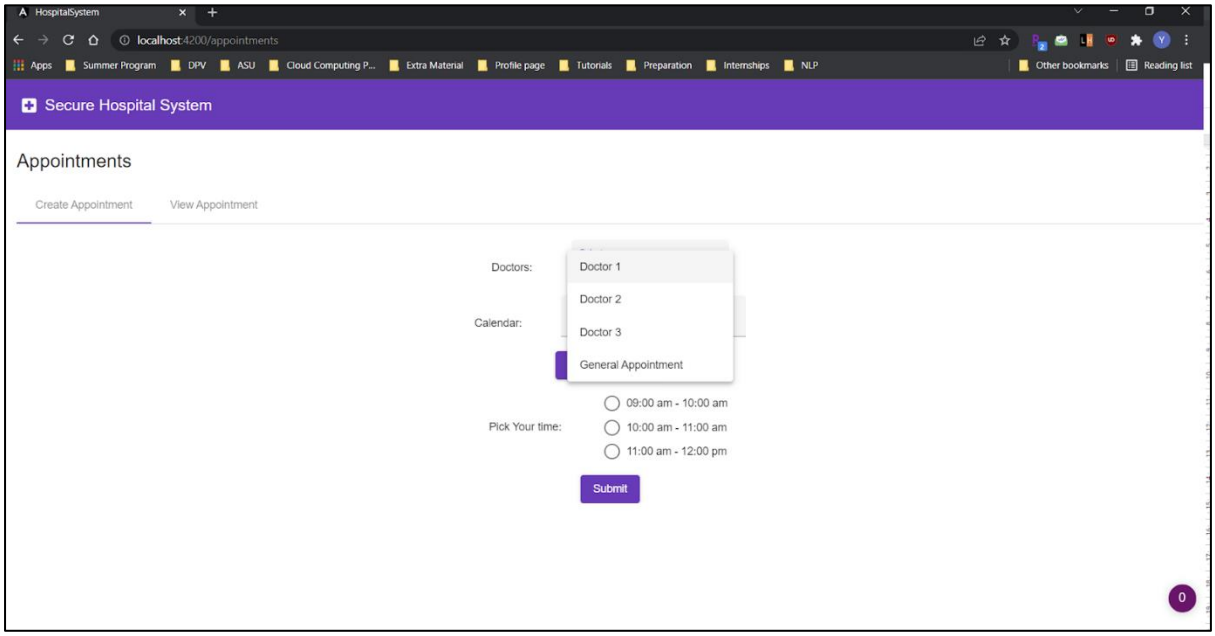


Fig.21 Wireframe Design – Appointment request

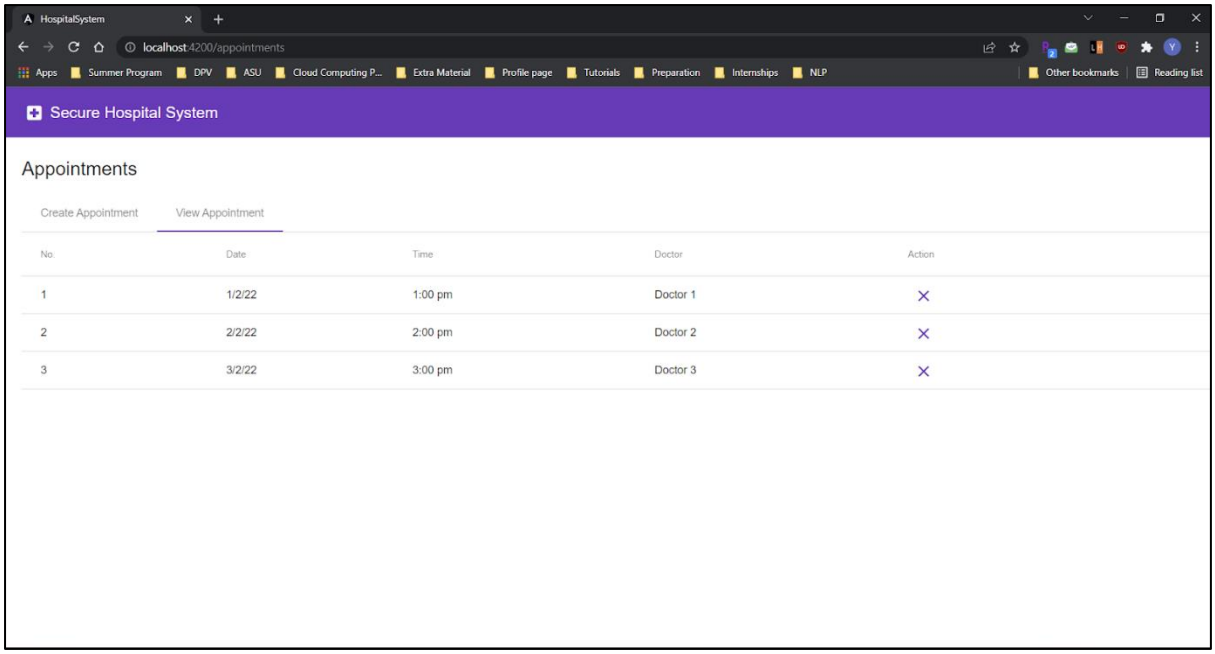


Fig.22 Wireframe Design – Appointment Approve/View

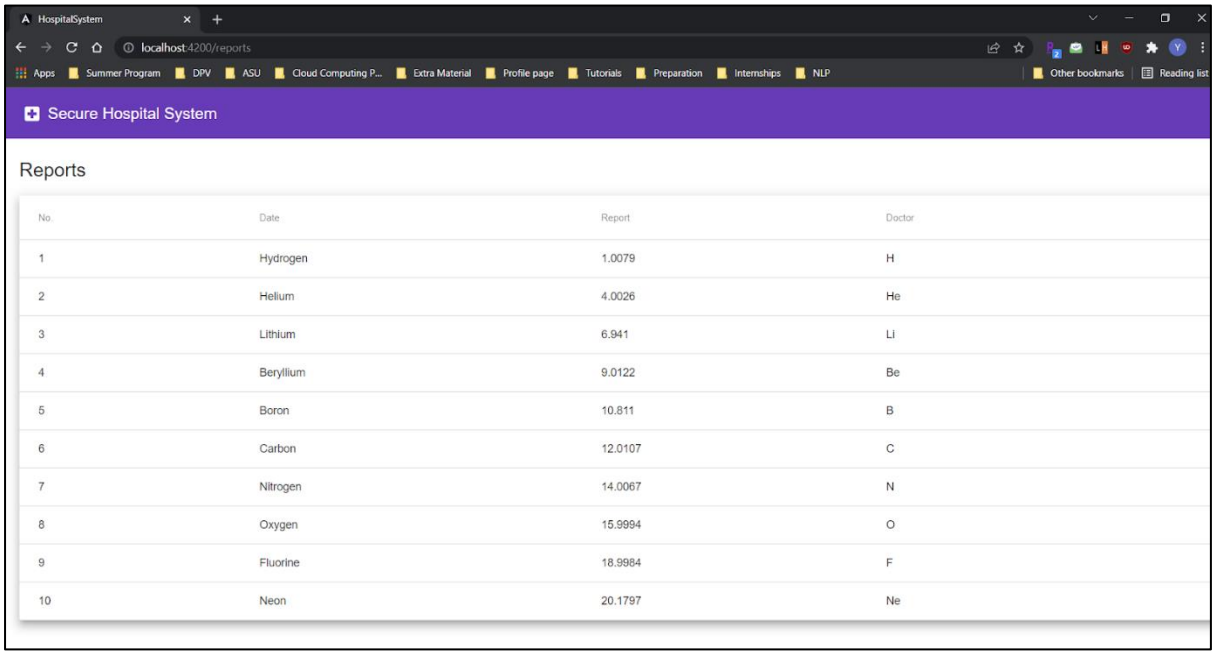


Fig.23 Wireframe Design – Reports view

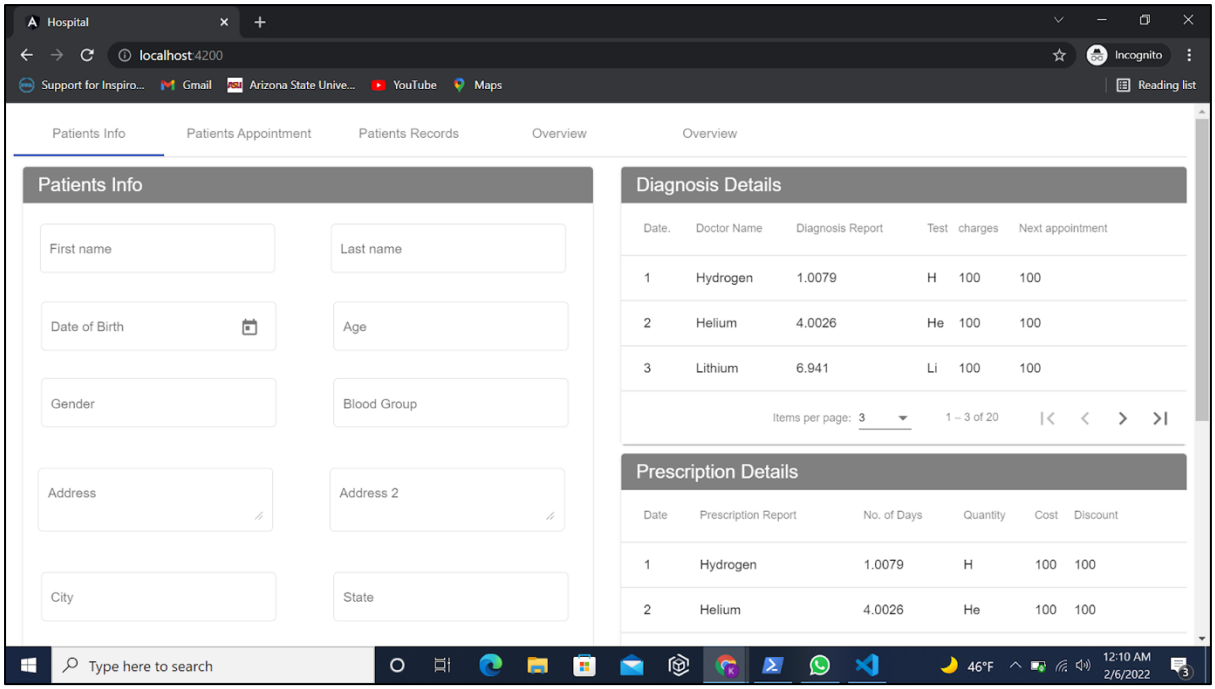


Fig.24 Wireframe Design – Patient Info dashboard

6. Non-Functional Requirements

6.1 Security Design

6.1.1 Secure Login with token and session management

[Public key certificates, OTP with Virtual Keyword for two functionality]

Secure login for any user is done in the two following ways. First method is the JWT authentication token. As depicted in the figure below, a new user signs up for an account and the database checks for an existing account for validation. Once, the account is created and the user signs in with his/her credentials, we generate an JWT token (string with a secret) and return it. This token must always be sent with a request from the front-end to access the user data. The access to other data/pages is restricted for that particular token.

The second method is two-factor authentication, where we are going to implement Two Factor Authentication functionality with a Soft Token and Spring Security. We are going to be adding the new functionality into an existing, simple login flow and use the Google Authenticator app to generate the tokens. To use Google Authenticator in our app we need to generate a secret key and provide secret key to the user via QR-code and verify token entered by the user using this secret key. And so, users provide an extra “verification token” during authentication – a one-time password verification code based on Time-based One-time Password TOTP algorithm. We will use a simple server-side library to generate/verify one-time password by adding the certain dependencies to the app start-up xml.

All sign-in history will be logged into session table. The session expiry is checked for each transaction request from the users.

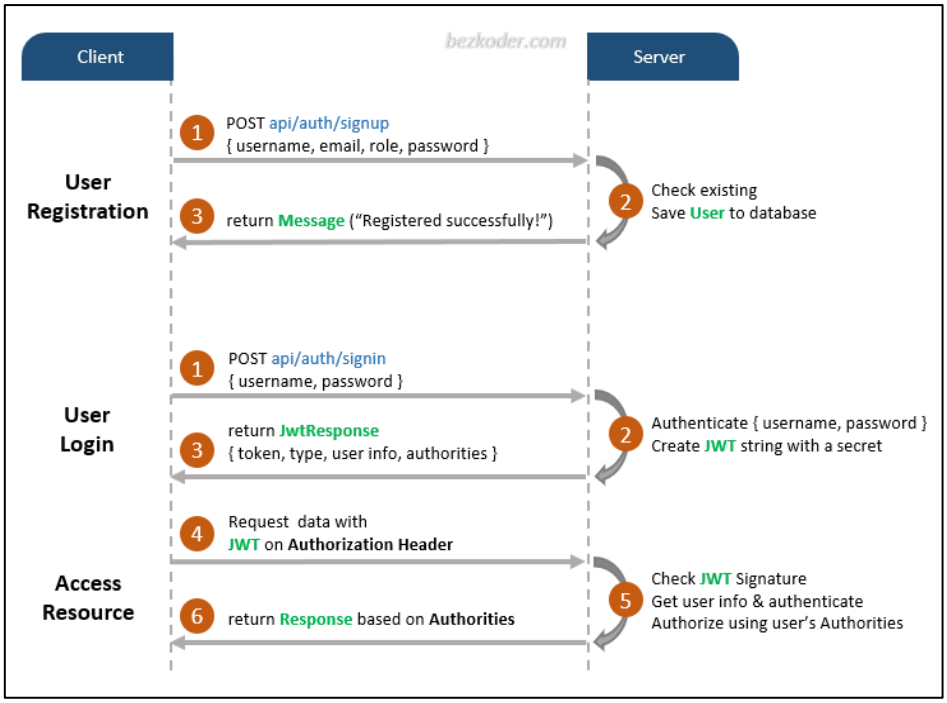


Fig.25 Secure Login interaction b/w client and server

6.1.2 Malicious Login Control

While many solutions are available for malicious login controls that are attempted by hackers, like Rainbow table attacks, brute force attacks, Dictionary attacks etc., adding ML to the solution takes a different approach. We employ a semi-supervised approach to group together malicious Ips. For each IP, we use the labels we must find features that are helpful in separating the good IPs from the bad ones. We tune the features and the parameters of our clustering algorithm by looking at how effectively the clusters separate the know good labels from the known bad labels. We select about ten features and then use the DBSCAN clustering algorithm (refer to figure below) to find the clusters of IPS.

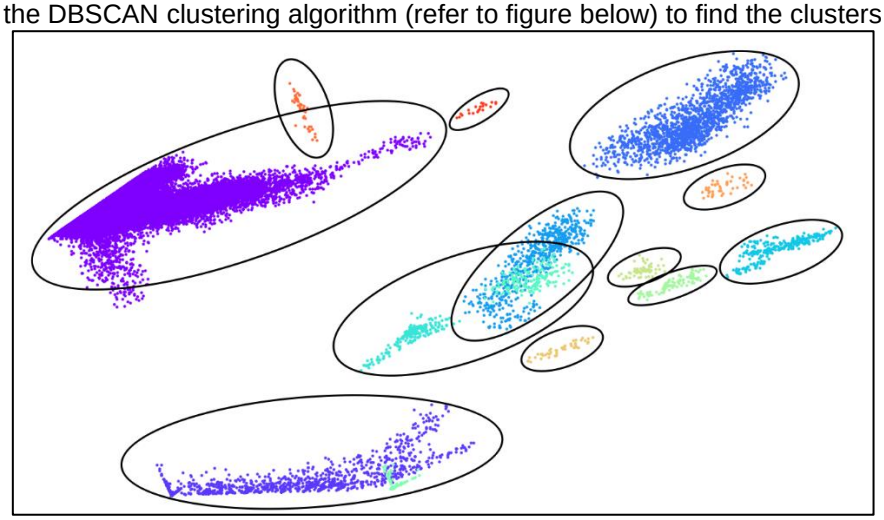


Fig.25 Malicious login pattern recognition

6.1.3 Blockchain

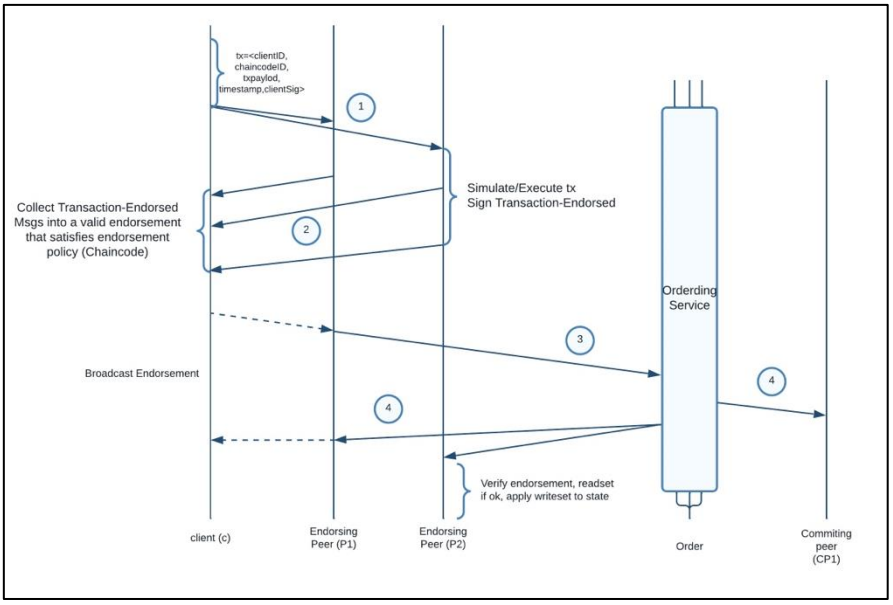


Fig.26 Smart Contract flow in Hyperledger

6.1.4 Data Masking/Hashing

To oversee hashing and encryption at rest, we will be using PostgreSQL's pgcrypto module. Effectively, this module provides hashing and encryption functions for use in queries. Data in transit is not encrypted by this module, so communications should be over SSL.

Hashing should be used over encryption when the original value is not necessary and only confirmation that the value to be hashed is equivalent to the original value is necessary - in this case, the hashing algorithm must be slow, to increase the brute forcing work required to crack the hash. The most common case of this is passwords. A second use case is verification - in the case of verification, the hash only serves to verify that the value has not changed from the original value. In this case, the hashing algorithm need not be slow. However, now, we do not have any verification use cases in the database for this project, only in the blockchain.

Encryption, on the other hand, will be used when the original value is necessary. While pgcrypto offers both asymmetric and symmetric encryption functions, we will be using symmetric encryption because all the encrypted data stored in the database must be accessed by the backend, and only the backend - there is no two-party communication involved. Thus, asymmetric encryption does not offer benefits. Meanwhile, symmetric encryption is faster, which is why we will be using it. Patient medical records, diagnoses, prescriptions, and lab test reports will be encrypted, as well as any financial data we store like credit card numbers.

Finally, for data masking, we will be managing it in the backend. Due to the limitations of PostgreSQL's dynamic data masking module, PostgreSQL Anonymizer, it can only mask data on a per-user basis. Since the backend must access all database data, unless a corresponding database user is made for each individual application user, no data can be masked in the database unless the data's complete, original value is not necessary. Thus, we plan to send unmasked data to the backend and have the backend return masked data to the frontend.

6.2 Backup and Archiving

A simple script to export the database as an incremental backup. Incremental backup only captures the change hence require less time for backup.

- Once a week full backup should be taken.

- Production Backup are stored in AWS S3. (Considering limited functionality backups are stored in local machine)

6.3 Performance

Following performance requirement must be catered:

1. Response time of any event should not exceed the standard response time for hospital applications (3 to 5 seconds) – Use of light weight interaction through rest api.
2. System should withstand user load of approximately fifty users per second and operate in 24/7 environment – Setup of multiple EC2 instance with session/thread management

7. Test Cases

7.1 Testing Strategy

The strategy and test cases adopted for evaluating the application are mentioned below. These cover the key security aspects and make sure that each requirement is met.

Classification of Tests:

(a) Blackbox Testing: This type of testing is performed irrespective of the internal working of the application. This includes usability testing, functional testing, security testing, regression testing and performance testing.

- ***Usability Testing:*** In usability testing, application flow and system navigation is evaluated for user-friendliness check.
  - Web page content should be correct without any spelling or grammatical errors
  - All fonts should be the same
  - All the text should be properly aligned
  - All the error messages should be correct without any spelling or grammatical errors and the error message should match with the field label
  - All the buttons should be in a standard format and size
  - Check if the dropdown data is not truncated due to the field size
  - Check whether the data is hardcoded or managed via administrator
- ***Functional Testing:*** In functional testing, the features and operational behaviour of a product are evaluated to ensure they bind to the specifications
  - All the mandatory fields should be validated
  - Each role has its duties fulfilled as per the requirements
  - Payments are made successfully
  - Mails are received when and wherever required
  - User should be able to view and download files
  - The chatbot should conduct the tasks with precision
- ***Security Testing:*** This is the key testing type as it identifies the risks, threats and vulnerabilities and prevents any attacks. It makes sure that the system cannot be exploited by intruder attacks.
  - Security certificate must be used for the application self-signed or authorized by a Certification Authority.
  - Verify password rules are implemented on all authentication pages like Registration, forgot password, change password.
  - Verify the essential information like password, credit card numbers etc should display in encrypted format.

- Verify if the user is logged out from the system or user session was expired, the user should not be able to navigate the site
- Verify the error messages should not display any valuable information.
- Verify the “View Source code” option is disabled and should not be visible to the user.
- Verify the cookies should not store passwords.
- Verify the user account gets locked out if the user is entering the wrong password several times.
- All the records, tests and transactions are approved.
- Sign-in history to verify the time and date of user login
- One time password to be sent to the user for login
- Verify for SQL injection attacks and DOS attacks

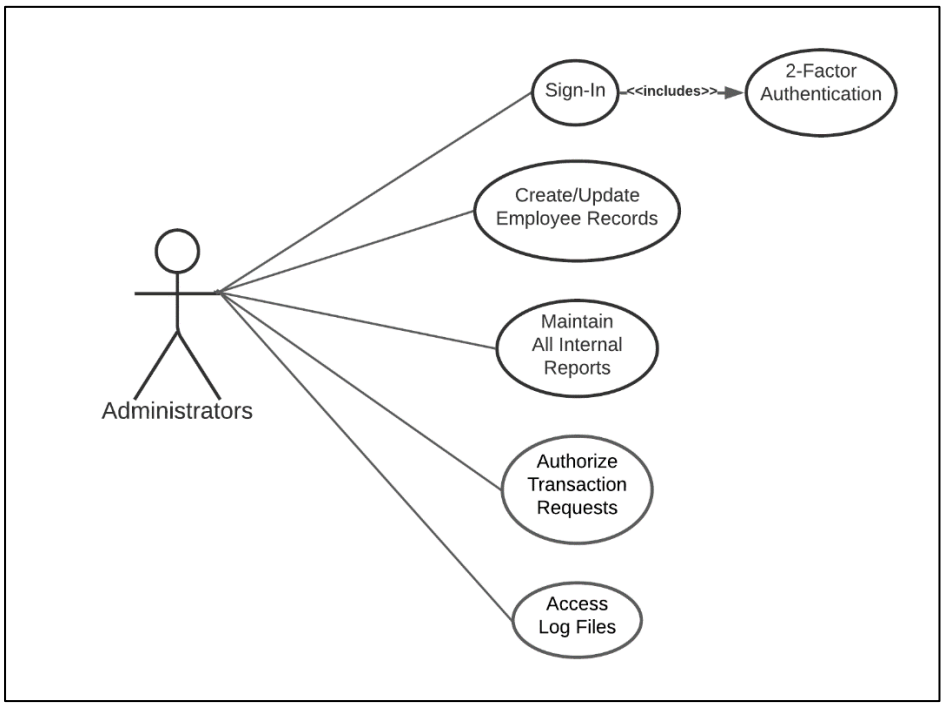
- ***Regression Testing:*** It is conducted to make sure that any code change or enhancements are not impacting the linked functionalities.
- ***Performance Testing:*** In performance testing, compliance of the application at system and component level is verified.
  - The system must be accessible 24/7
  - Response time of the events should not be more than 3 to 5 seconds
  - System should withstand a user load of fifty users per second

(b) Whitebox Testing: This type of testing is performed with a detailed investigation of the internal logic and code structure and identifies which unit of the code is working inappropriately. This includes unit testing, integration testing, database testing, compatibility testing, statement coverage and memory leak testing

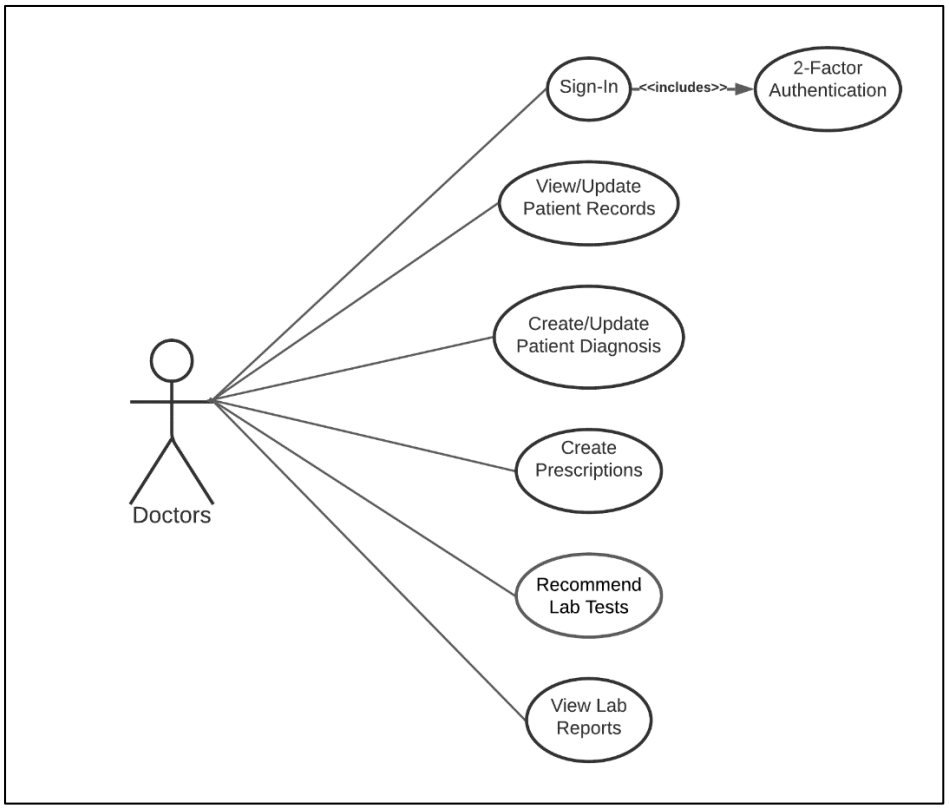
- ***Unit Testing:*** Once the code is ready for a specific unit or functionality, the developer performs unit testing to verify the functionality is up to mark. This testing is performed before integrating the developed code into the system to eliminate the bugs after the development phase.
- ***Integration Testing:*** Once the unit testing is done and code is integrated into the system, any two individual functionalities are evaluated to verify the satisfied requirement and that they do not interrupt each other. Integration testing is done to see that every functionality is independently in a perfect working condition
- ***Database Testing:*** In database testing, the records inserted through the web application are evaluated to make sure that the records in the database and the once displayed in application are matching
  - Verify whether the column allows a null or not.
  - Verify the Primary and foreign key of each table.
  - Verify the Stored Procedure:
  - Verify the encrypted data in the database.
  - Test whether the stored procedure returns the values.
- ***Compatibility Testing:*** It is conducted to evaluate that the software is compatible with the browser, operating system, and hardware it should operate
  - The application should work as expected in the specified browser
  - HTML version being used should be allowed by the specified browser
  - Images and texts should be displayed as expected in the browser
- ***Memory Leak Testing:*** This testing is performed to identify any memory leaks and optimize the memory usage.
- ***Statement Coverage:*** Finally, it is important to make sure that each statement mentioned in the requirement document for a particular functionality is covered, each line of source code is executed and evaluated.

7.2 Use Case

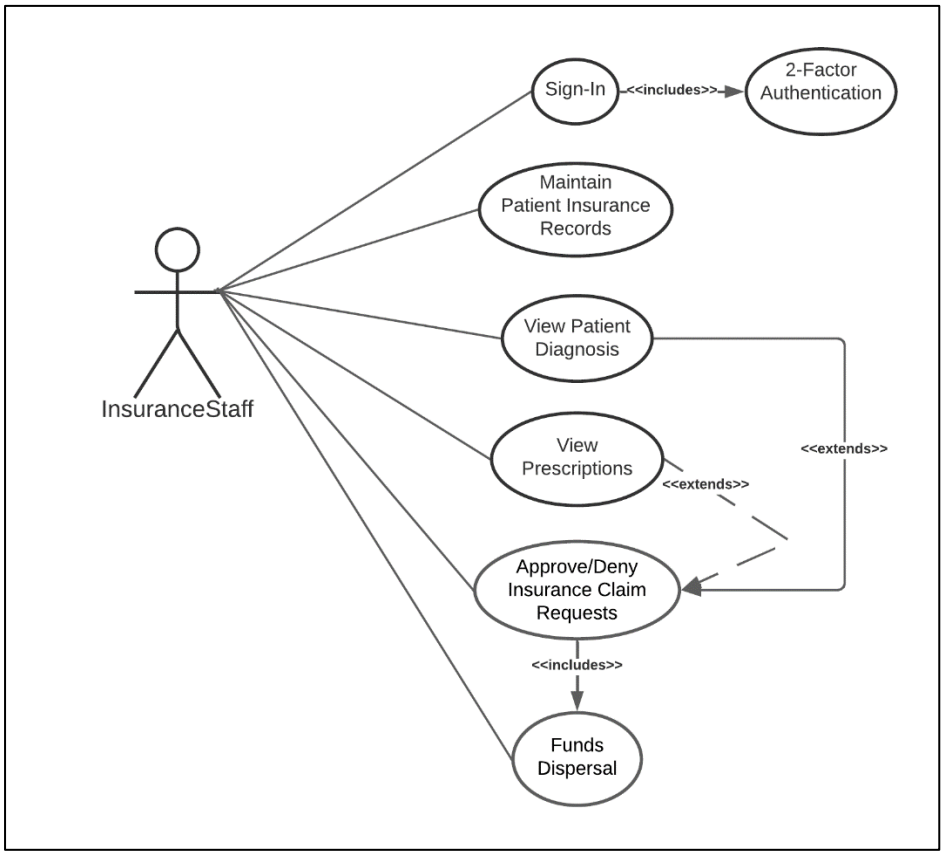
7.2.1 Administrator Use Case



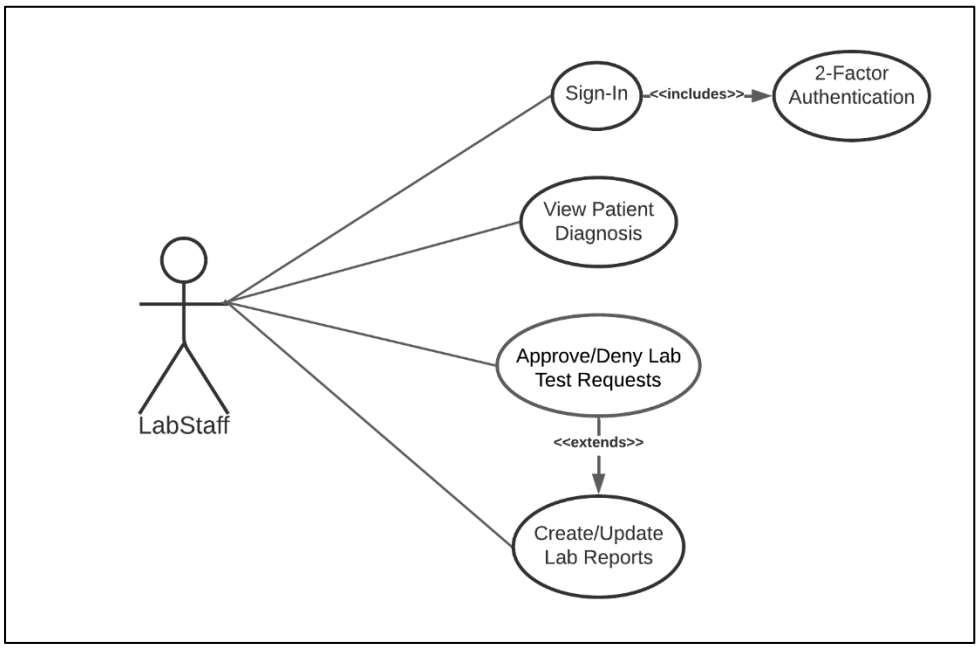
7.2.2 Doctor Use Case



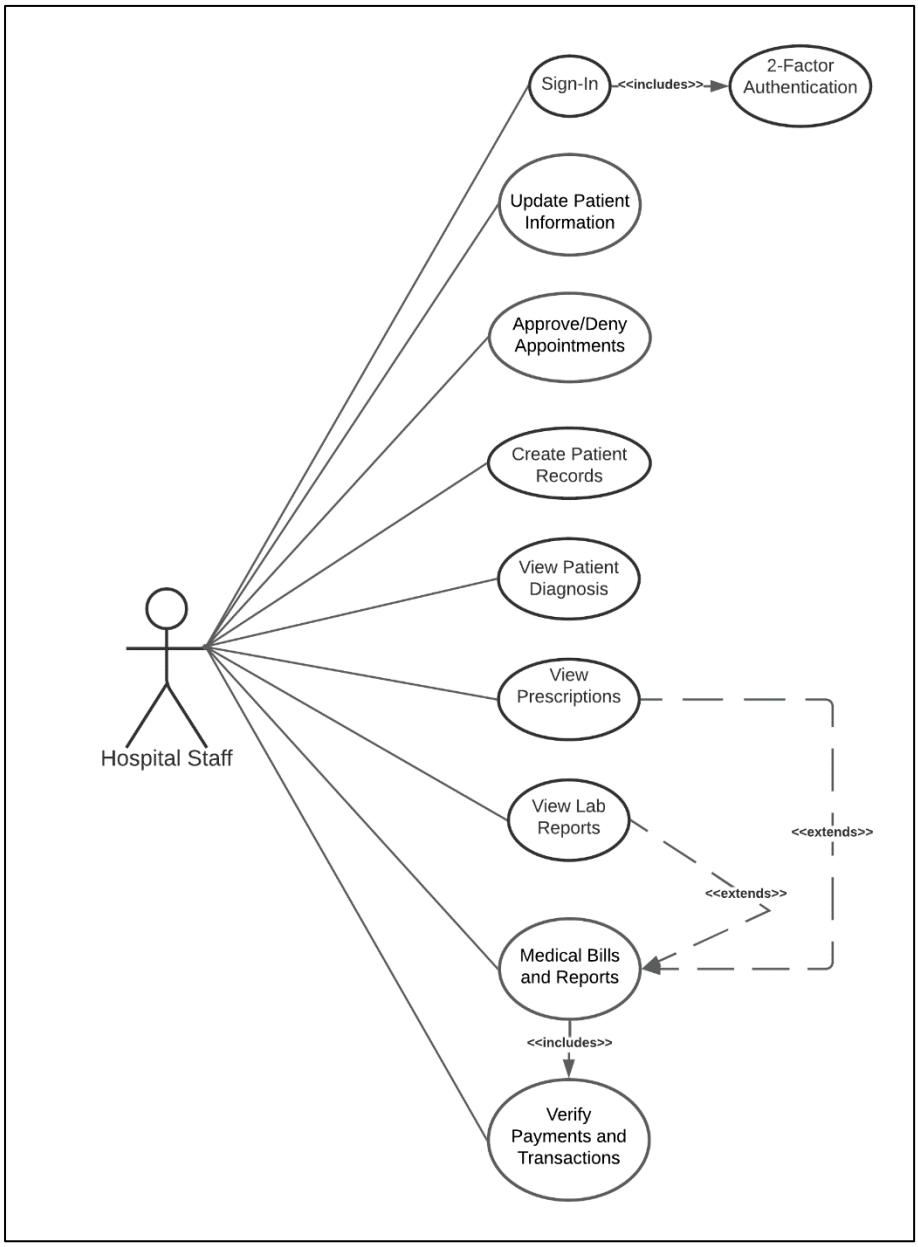
7.2.3 Insurance Staff Use Case



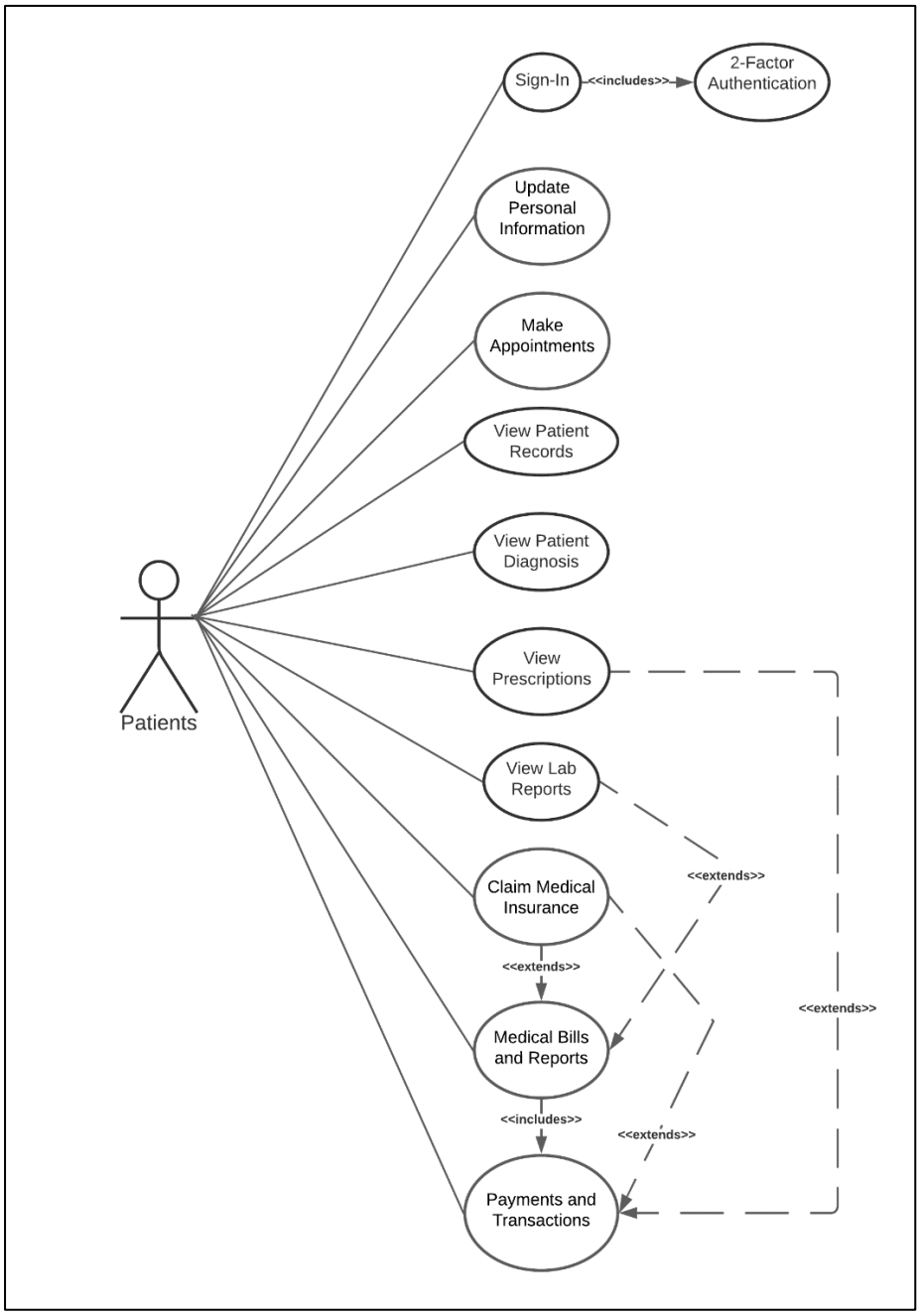
7.2.4 Lab Staff Use Case



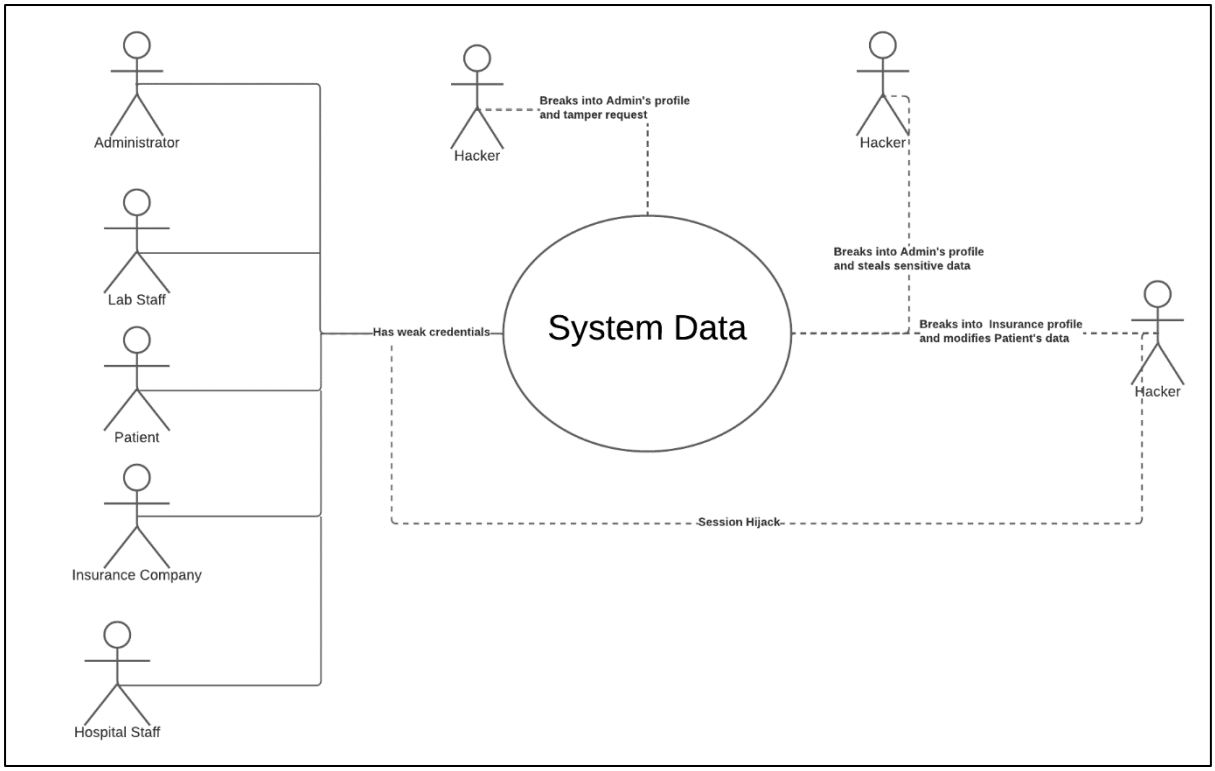
7.2.5 Hospital Staff Use Case



7.2.6 Patient Use Case



7.3 Misuse Case



8. Packaging/Installation

All the components of data tier (Postgre DB and Hyperledger fabric) will be setup directly in the AWS EC2 instance. The application and web component will be deployed as an archive into web application server in the respective tiers.

9. References

[References with hyperlink]

1. https://hyperledger-fabric.readthedocs.io/en/release-2.2/network/network.html
2. https://www.bezkoder.com/spring-boot-security-postgresql-jwt-authentication/
3. https://medium.com/uber-security-privacy/uber-machine-learning-account-security-3aaadef11e45