CSE 579 Individual Project Report - Group 12.
Secure Health System

Pavan Kumar Raja

MS computer science, School of Computing and Augmented Intelligence, ASU

praja3@asu.edu | ASU ID: 1222316641

***Abstract*— The project aims at building a web application for handling a hospital system and incorporating security-related topics taught in the course to make the system robust against any cyber-attacks by securing the critical operation and protecting private information.**

I. Overview

In the traditional healthcare system, there are a lot of pitfalls in information passing, record keeping, and passing on the records. All the hassle involved in making this process effective can be handled by an automated system using technology. The Secure Healthcare System project is developed to do exactly this. In this project, we are creating a web application that makes all the individual actors involved in a healthcare system have ease of access to all the information and ease of execution to support their operations (for employees).

Our Secure HealthCare System provides a user-friendly interface for Patients, Hospital staff, Doctors, Lab staff, Insurance staff, and System Admin. The system is built using technologies such as React.JS and bootstrap for user interface, Python and Flash Api package for backend API framework, Manogodb and Redis server for data storage services, Hyperledger blockchain framework for registering valid transactions in the blockchain, and Machine learning techniques for help chatbot[1]. Fig 1. shows the general architecture of our system.

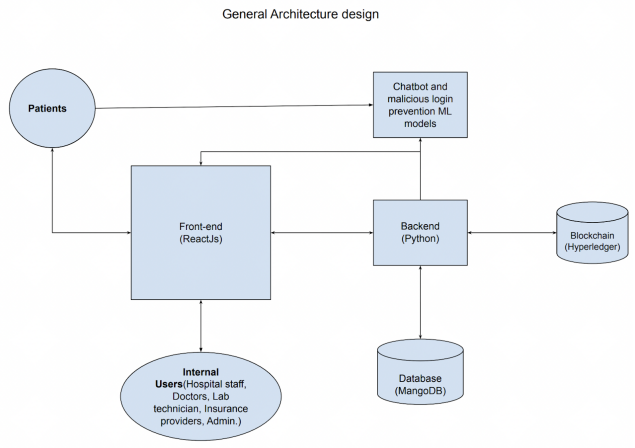


Fig 1. General Architecture our SHS web application.

Our web application provides different user functionalities for each kind of user,

Patients: Create accounts, Update their personal information, Schedule an appointment with doctors, View their medical records, View their lab test reports, View their transactions and can make payments, View insurance policy and Request an insurance claim.

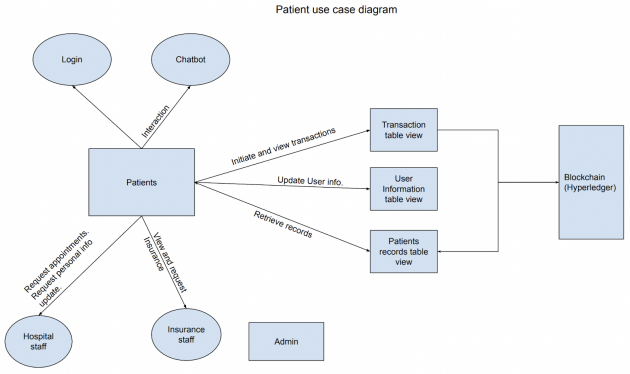


Fig 2. patient use case flows.

Doctors: View and update the patients' records, Recommend a lab test for patients, View lab reports of their recommendations, Create and Update the diagnosis of the patients, and Create and View the prescription recommendations.

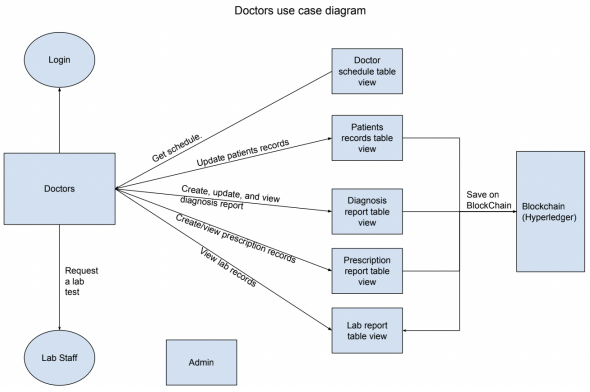


Fig 3. Doctors use case flows[1].

Hospital staff: Create the patients' records, Approve or deny appointment requests from patients, Create a transaction, and Complete a transaction.

[1] Drawings generated using https://draw.io/

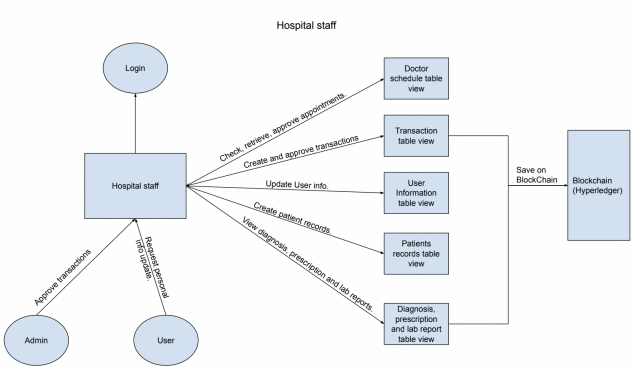


Fig 4. Hospital staff use case flows.

Lab Staff: Can approve/deny doctor requests for lab tests, Can create a lab test repost, and Can update a lab test.

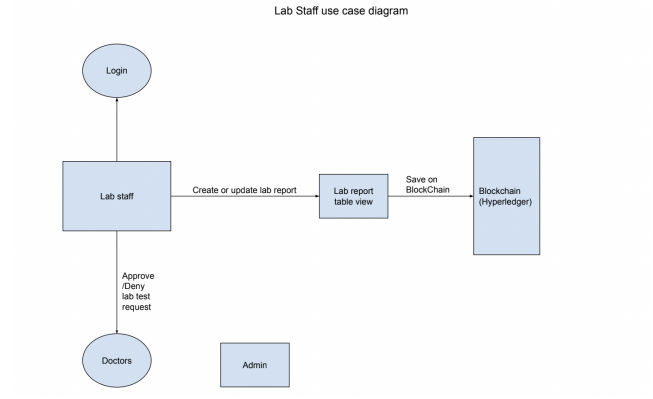
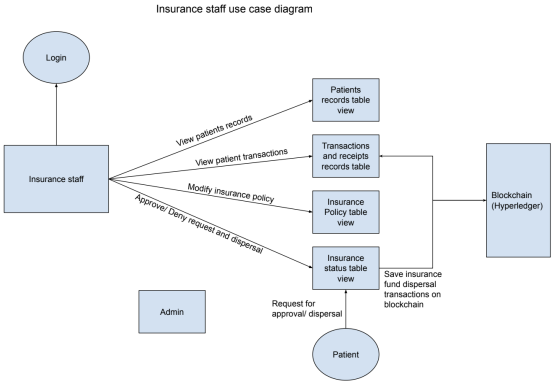


Fig 5. Lab staff use case flows.

Insurance Staff: View receipts, payments, and transactions. View insurance records of patients, Add/Update insurance policy, View claims requests and Can approve/deny insurance requests.

Fig 6. Insurance staff use case flows



Admin: Can create, view, and employee records, Authorize the transactions. Authorize new employee accounts created by employees. Can view system logs to monitor the health of the application.

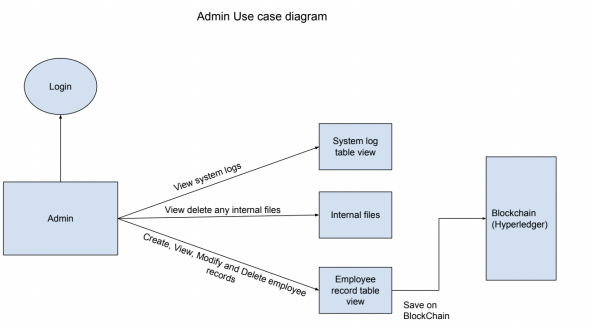


Fig 7. Admin use case flow[2].

The project is developed to showcase our understanding and skills to implement the security topics discussed and taught in the class. Therefore, we have implemented many such security features.

a. Jwt Authentication in every request.
b. Amazon Virtual Private Cloud for backend, database, and Redis server: We are using MongoDB for storing our database, along with Redis to manage in-memory data used for applications like OTP verification. Such data are usually vulnerable to data breaches and hacks. To mitigate this security risk we are hosting them on an Amazon Virtual Private Cloud, which implements harsher restrictions, only the admin authenticated by Amazon will have access to these data.
c. Field-level encryption: Provides privacy and also protects us from injection attacks on the database.
d. End to End encryption of payload for network communication and eavesdropping protection.
e. We have end-to-end encryption logic cooked into our entire communication system (i.e front-end and back-end). This means right from the launch of the website all the request made to our backend has encryption enabled on the payload level, which ensures application secret integrity and eavesdropping on the payload is pointless.
f. Field Level encryption in the blockchain[2]: To safeguard our data from data break on the 3rd party blockchain provider.
g. Authentication and user-type based routing: Every link is user-type protected, which means only the operations of the user's type are visible and accessible by that user.
h. Login through OTP: All login should be confirmed by otp, which expires after 10 mins.

II. Individual Contribution

I am the lead of this project undertaken by group no.12. My technical contributions included:

1. Detailed design of the project: The design of the project in terms of development flow, technologies,

[2] Drawings generated using https://draw.io/

project design, and use case flows are designed by me. Technologies used were based on the preference and experience of team members handling the vertical.

2. Design of all the sequence flows: Sequence flow is one of the most important documents that should be followed when developing parallelly, I was solely responsible to determine the sequence of all the use cases required for us to implement.

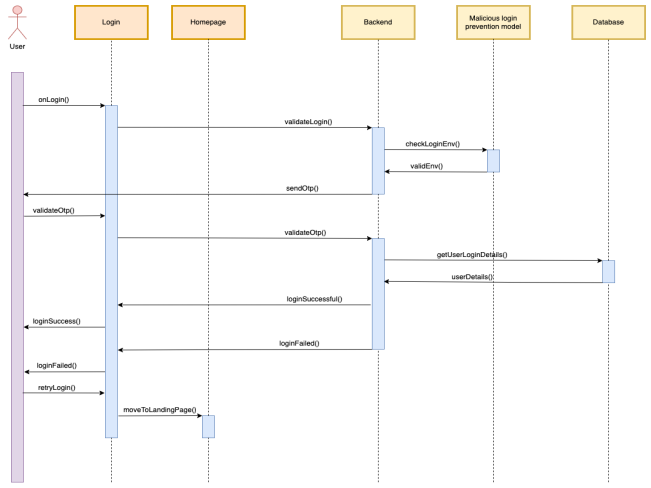


Fig 8. An example of login sequence flow.

3. I led the development of UX/UI design[4]: Deciding on the technologies used, UI element, Type of implementation, and Coming up with skeleton and base components for our UI. I have developed around 40% of the front-end Ux/UI elements.
4. All the Ui components are based on bootstrap and designed using functional components in ReactJS[4].
5. I implemented secure features for the front-end which include: Implementing session context for the software for multi-login is supported across the browser, Implementing user-based routing to protect our intermediate links, and an HTTP protocol set up to communicate with the backend with retries (3) upon failure.
6. I implemented end-to-end end-point encryption to secure our communication in the HTTP network.
    a. Design details:
    b. Generated Asymmetric keys at the backend (RSA 256 SHA keys), private keys are not exposed to any request and are stored in a secret manager.
    c. When the website is launched, there is a default request to our backend where we fetch the public key.
    d. The frontend in its application memory generates a symmetric key, using a random salt, random Initialization vector, a random validation data, and a random password (all 16 bytes), the random salt and password are used to generate a key using the pbkdf2 hashing algorithm which is run for 10000 iterations.
    e. The key along with IV and validation data is used to encrypt our payload and route of the request.
    f. We used AES-GCM 256 block encryption technology, which generates a key and encrypted data.
    g. The secret, which consists of a symmetric key and validation data, is hashed with the public key that we fetched from the backend.
    h. The request payload is a json with {payload, route and secret}
    i. Once the backend gets this payload, it'll get the symmetric keys by using its private key and decrypting the secret field, after this secret is used to first validate the encrypted blobs using the validation data part of the secret, this will detect any manipulation in the encrypted blobs, and then it'll use the keys in the secret to decrypt the actual payload and route and they are used as required.
    j. For the response, our backend uses the same secret it had gotten in the request and encrypts the response.
    k. This resulted in secure end-end encrypted communication[5].

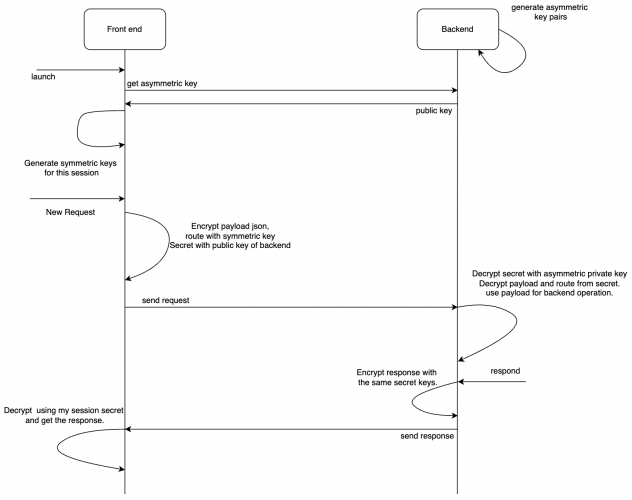


Fig 9. Sequence diagram of our end-point encryption logic.

7. I have developed around 20% of the backend API, mostly these APIs are related to lab staff, hospital staff, admin, and patient use cases. These APIs were leftover/ overlooked or had to be redesigned according to correct use cases.

8. I was solely responsible for integrating all the backend APIs which were around 70 with the frontend flow using the HTTP module that I had created in earlier phases.
9. I was also responsible for coming up with ML[3] design and integrating the chatbot model with our backend and creating APIs for it.

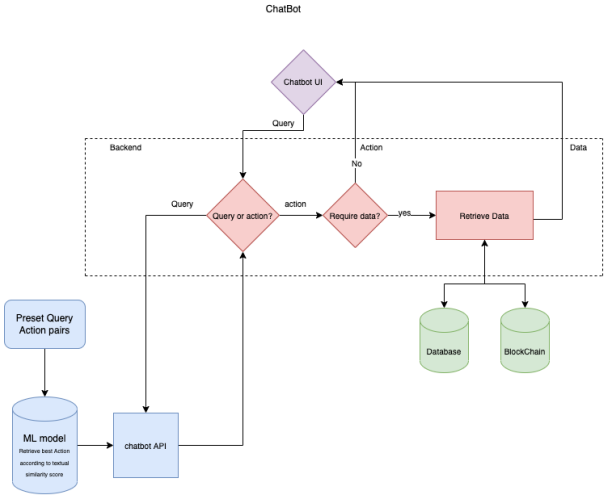


Fig 10. Chatbot ML design and flow.

As a team leader, I had additional responsibilities which included a. Deciding on the design of this project with input from the team members. b. Dividing the work according to their preference. c. Setting up an Agile workflow for monitoring the project progress. d. Driving weekly team meetings and discussions. e. Providing resources and helping team members. f. Reviewing weekly reports and progress of the project. g. Reviewing security aspects of all the vertices. h. Reviewing vulnerability reports and driving the vulnerability discussion.

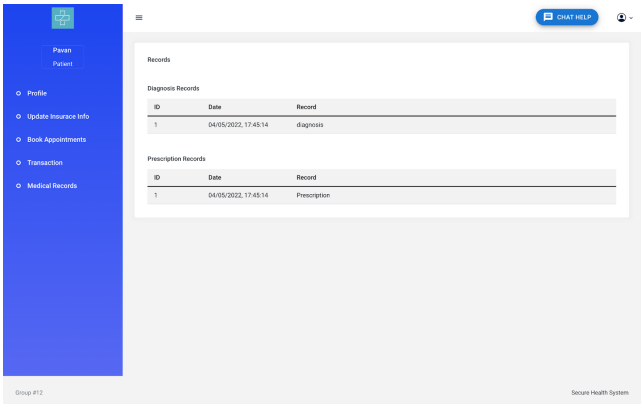


Fig 11. An example[6] snapshot from our Web application[3].

III. Lesson Learned[4]

1. I learned how to manage a team in Agile workflow, set the phase of the development, and monitor it using Gantt charts.
2. I learned how to handle unforeseen circumstances in the project as a leader.
3. I learned how to design a web application using the inputs from the team.
4. I learned how to infer requirements and come up with the sequence flow of an entire application.
5. I learned the concept of securing web communication through end-point encryption.
6. I became more proficient in frontend development using react functional components.
7. I learned how to protect our backend links using authorization tokens.
8. I learned the importance of HTTP headers in web application development.
9. I learned how to develop backend APIs and integrate them [7].
10. I learned how to deploy ML models and integrate them into web applications.
11. I learned the intricacies of blockchain technology in securing the data, albeit small size ones.
12. I also learned how to evaluate a web application for its security vulnerabilities.

IV. References

[1] CSE 545 Course Project Requirements Document.

[2] IBM hyperledger framework, and examples https://www.hyperledger.org/use/fabric

[3] Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://arxiv.org/abs/1908.10084

[4] React tutorials and libraries. https://reactjs.org/, https://npmjs.com

[5] OWASP security standard reference. https://owasp.org/

[6] Project Report, Group #12 CSE 545 spring 2022.

[7] Backend Flask documentation. https://flask.palletsprojects.com

[3] Drawings generated using https://draw.io/

[4] Github project link https://github.com/Pavan-pk/CSE-545-SHS