AY 2022/23 Semester 1

IFS4205 Information Security Capstone Project Group 3

System Design Report

Crowd Sensing Medical Application

Oscar Lai Chun Ho	A0204697W
Alicia Tay	A0204813N
Daryl Hung Dejian	A0217127L
Isaac Lai Zile	A0216952B
Lim Jie Rui	A0217086A

1. Overview

This report provides a detailed and extensive outline of each subsystem in our web service application. The report will be an extension to the system design and architecture diagram described in the Tool Assessment Report. In each subsystem, the components, implementation, interaction with the database and security concerns will be explained.

The subsystems in our web service are *Create Account, Login account, Anonymised Records Flow, Viewing Crowd Sensor count and Interaction between a Doctor and Patient.*

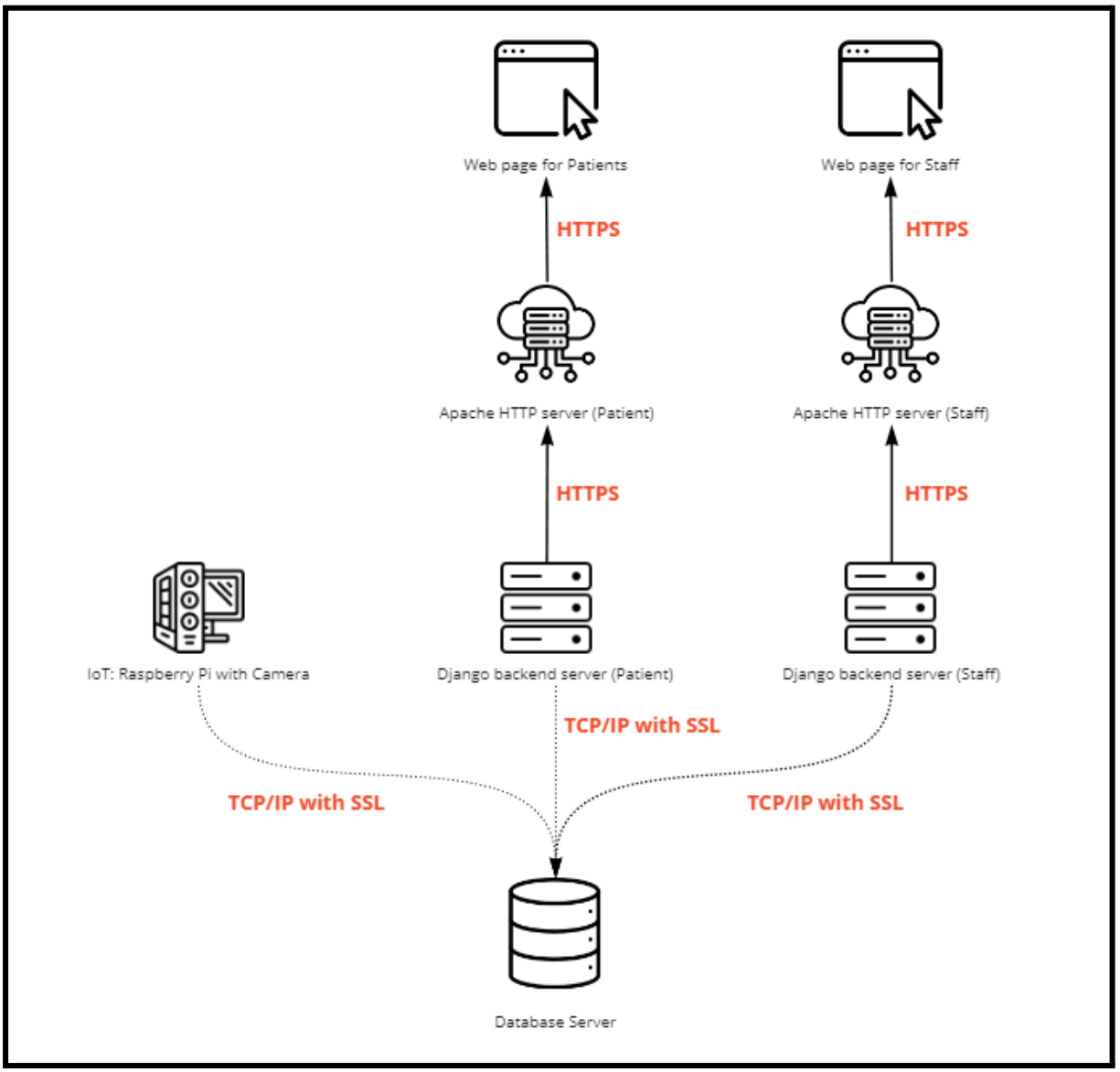

Figure 1: Architecture

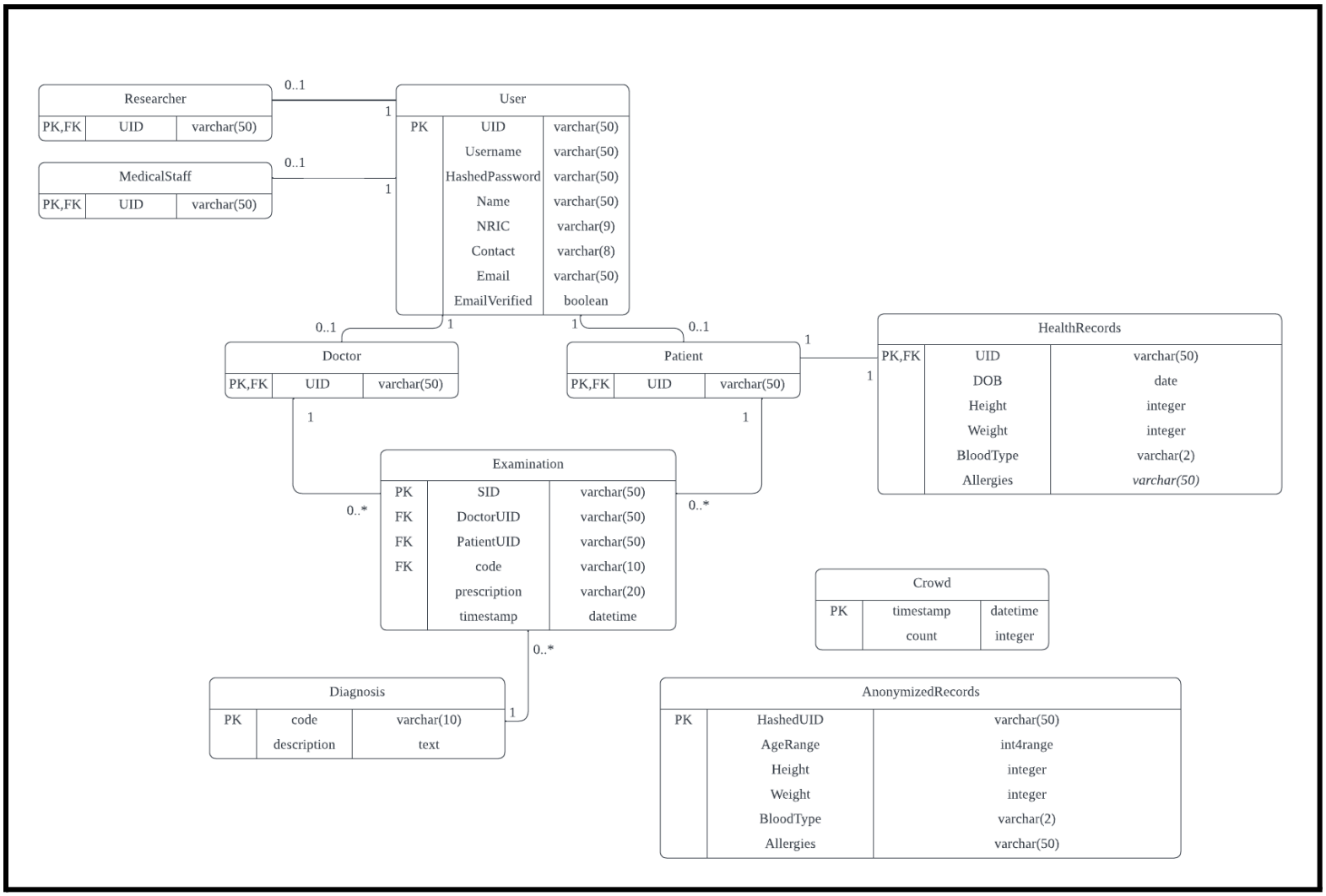

Figure 2: ER Diagram of Database

2. Roles

This section highlights all the roles, privileges involved in our web application. Our application will have a main landing page that allows users to specify their role: Patients, Doctors, Researchers or Medical Staff. After specifying their role, they will be redirected to a portal based on their role.

Patients

Patients are the primary target group of our application and hence, patients can access several functions. The patient's portal enables patients to manage their health records. They can create an account, login, generate a queue number and view their health records. Real-time data from the crowd-sensing IoT devices will be displayed on the patient's portal so that patients can generate a queue number during less crowded timings.

Doctors

Doctors are the secondary target group of our application. Doctors are able to add data to the examinations table where they add records such as prescribed medication and diagnosis after examining a patient. In order to ensure that doctors are not allowed to view any random patients' data, the doctor can only access their own patient's data and to do so, they have to enter the session identifier generated by the user in order to access the patient's data. More details can be seen in Section 7.

Researchers

Researchers portal will allow researchers to view anonymised general health records.

Medical Staff

Medical Staff refer to counter staff that check in the patients to the medical facility. They will be only allowed to view real-time data from the crowd-sensing IoT device. This is to ensure efficient crowd control of the medical facility.

3. Create Account

General Description

Only patients are able to create an account. On the web app's Main Page, the patient chooses the role of "Patient" to get to the Patient Frontend page. He selects "Create Account", which brings him to a Form Page, which he fills in with his account information (including his email for account verification) and health records. An Acknowledgement Page is then rendered (Figure 3).

This will trigger two main logic components in the backend:

1. Input validation (Figure 3, in turquoise) on the form fields is done. A UID is generated at the backend and records containing this UID and the validated form fields are inserted into the User, Patient and HealthRecords tables.
2. Email verification (Figure 3, in turquoise)
    1. A POST request is sent to the app server containing the patient's credentials.
    2. When the app server receives the request, the verification link generator will create a link and email it to the patient.
    3. The patient clicks on the link in the email which will send the credentials back to the app server.
    4. The app server renders the Verified Page which can lead the patient back to the Patient Frontend Page for logging in with his new credentials. At the same time, the app server updates the *EmailVerified* field in the record that was inserted into the User table at logic component 2.

Implementation (Diagram)

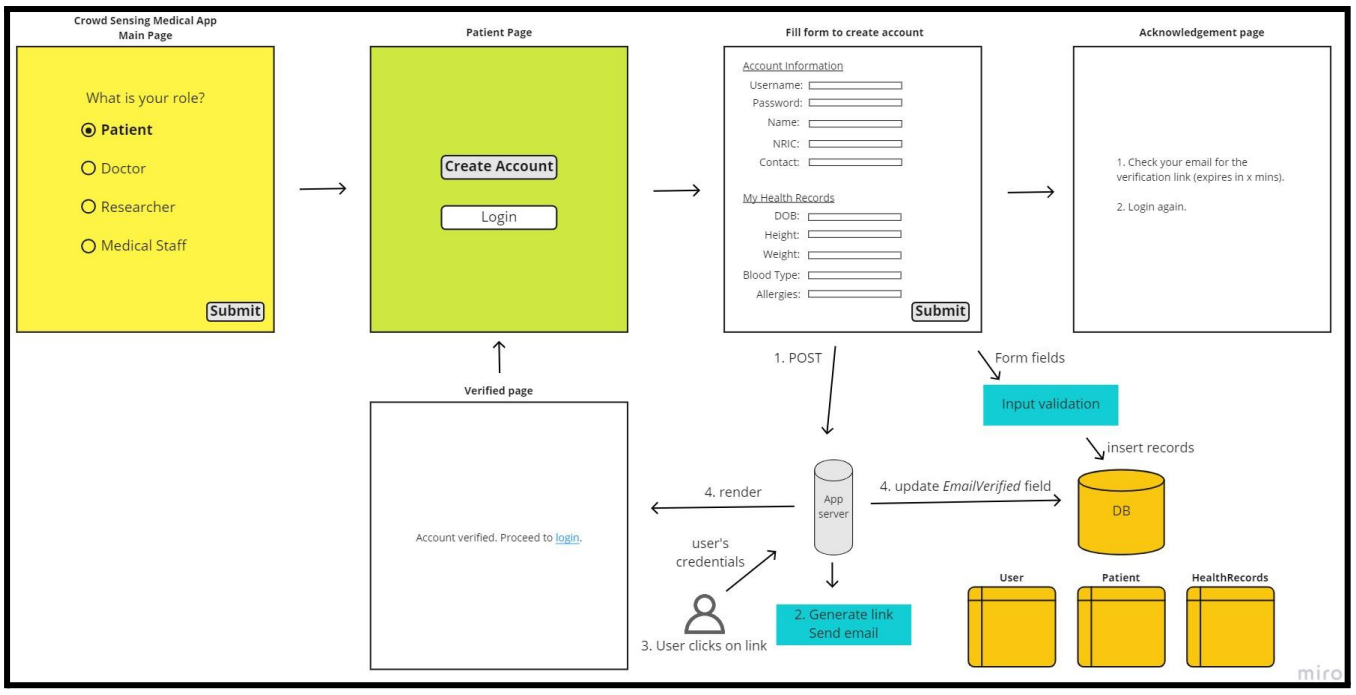

Figure 3: Flow of patient creating an account

Interaction with Database

User Table

A record will be inserted into the User table with the fields UID, Username, HashedPassword, Name, NRIC, Contact and Email. UID is generated while the rest of the fields are from the respective form fields sent in from the web app.

Patient Table

This UID will also be inserted as a record into the Patient table.

HealthRecords Table

A record will also be inserted into the HealthRecords table with the fields UID, DOB, Height, Weight, BloodType, Allergies. The UID is the same while the rest of the fields are from the remaining respective form fields.

Security Concerns

- During the creation of an account, there is insertion of records into the database. This opens up possibilities of stored XSS attacks. Input validation for all fields of the Form Page and checks to safeguard against disabling of JavaScript will be carried out to prevent this and SQL injection on the Form Page.
- The Form Page will also make POST requests that contain sensitive information such as the patient's password to the app backend server. As HTTPS is used, the likelihood of man-in-the-middle attacks will be reduced.
- An attacker may spam account creations using random email addresses. This can be detected by checking for multiple form submissions from the same IP address in a short period of time and blocking the IP address. A scheduled job can also be run to clean up records in the User table that still have the *EmailVerified* field set to false.

4. Login

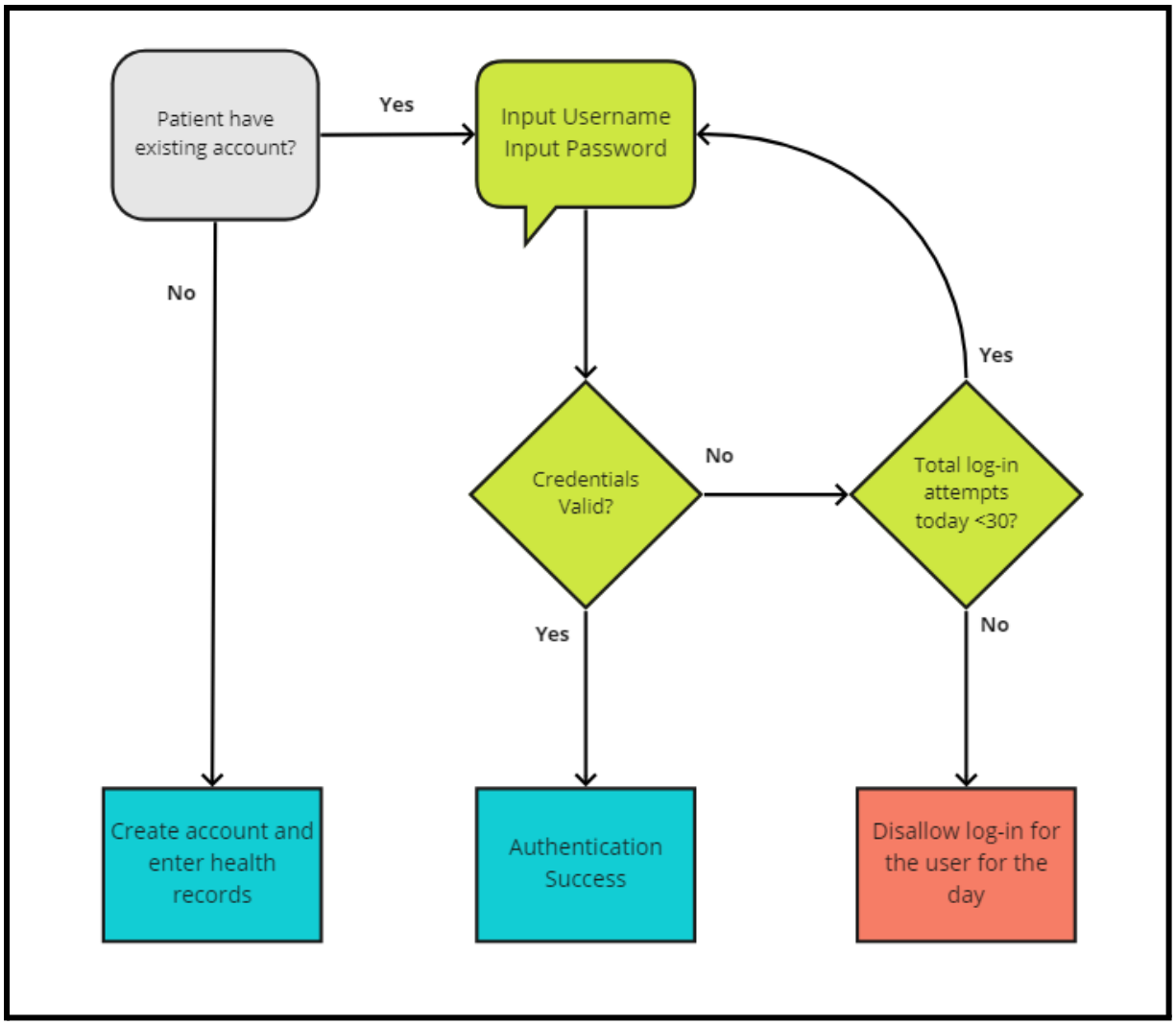

Figure 4: User Authentication Flow (Regular sign in)

Login portal

At the login page, users will be required to fill in their username and password, recorded in the form of a HTML form. Input parameters of the form will be sent to the Django backend using REST API. REST APIs use HTTPS protocol, encrypted using Transport Layer Security (TLS) ensuring secure communication between the web application and the Django backend, protecting against man-in-the-middle (MITM) attacks.

Back-end

Once the backend receives the user's credentials, it will then try to authenticate the user. Django uses the Password-Based Key Derivation Function 2 (PBKDF2) algorithm with a SHA256 hash to hash the user's password. PBKDF2_SHA256 is a password stretching mechanism recommended by NIST, requiring massive amounts of computing time to break, deterring brute force attacks. Django will construct a SQL query using query

parameterisation to verify the user's credentials stored in our database server. Using Django's querysets, any unsafe user-provided inputs are escaped by the underlying database driver, preventing SQL injections.

Interacting with Database

A connection will be made to our database server using database credentials delegated to Django. The TCP/IP connection to the database server will be secured using SSL to prevent MITM attacks. If the user's entered credentials match the credentials from our database, the user is logged in.

Patient Table
Credentials input by the user will be checked against Username and HashedPassword in the Patient table.

Using OAuth for Authentication

OAuth is an open standard for access delegation. It is commonly used as a way for Internet users to grant websites or applications access to their information on third-party websites, without sharing the credential information with these third-party websites.

Besides authenticating through our database, users can authenticate using Google sign-in. Google's OAuth 2.0 server serves to simplify integrating Google Sign-In with web applications, allowing users to sign-in via Google. Users do not need to manage an independent account on our platform and it removes the need to store and manage users' log-in credentials, a common security concern.

OAuth 2.0 works as follows: First, our application requests authorisation to sign into Google through OAuth. OAuth authenticates us using our client id and client secret, which are provided when setting up OAuth. OAuth will verify with Google that our requested scopes are permitted and Google will grant access. If successful, OAuth then redirects back to us with an Access Token, which authorises us to be signed in via Google.

Security Concerns

- Brute force attacks on user credentials, although mitigated through strong password requirements and throttling via Django, are still a potential security concern.
- Django templates protect against the majority of XSS attacks, however we need to understand its limitations to protect against certain edge cases.
- Django has in-built protection against most types of Cross Site Request Forgery (CSRF) attacks, provided it is correctly configured.

5. Anonymised Records Flow

General Description

When a user creates an account, they are prompted to add their health records such as weight, height, allergies, blood type and date of birth. When a record is added to the HealthRecord table, a process in the SQL server will automatically anonymize the health records and insert it into the AnonymizedRecords which can be viewed only by Researchers.

Implementation (Diagram)

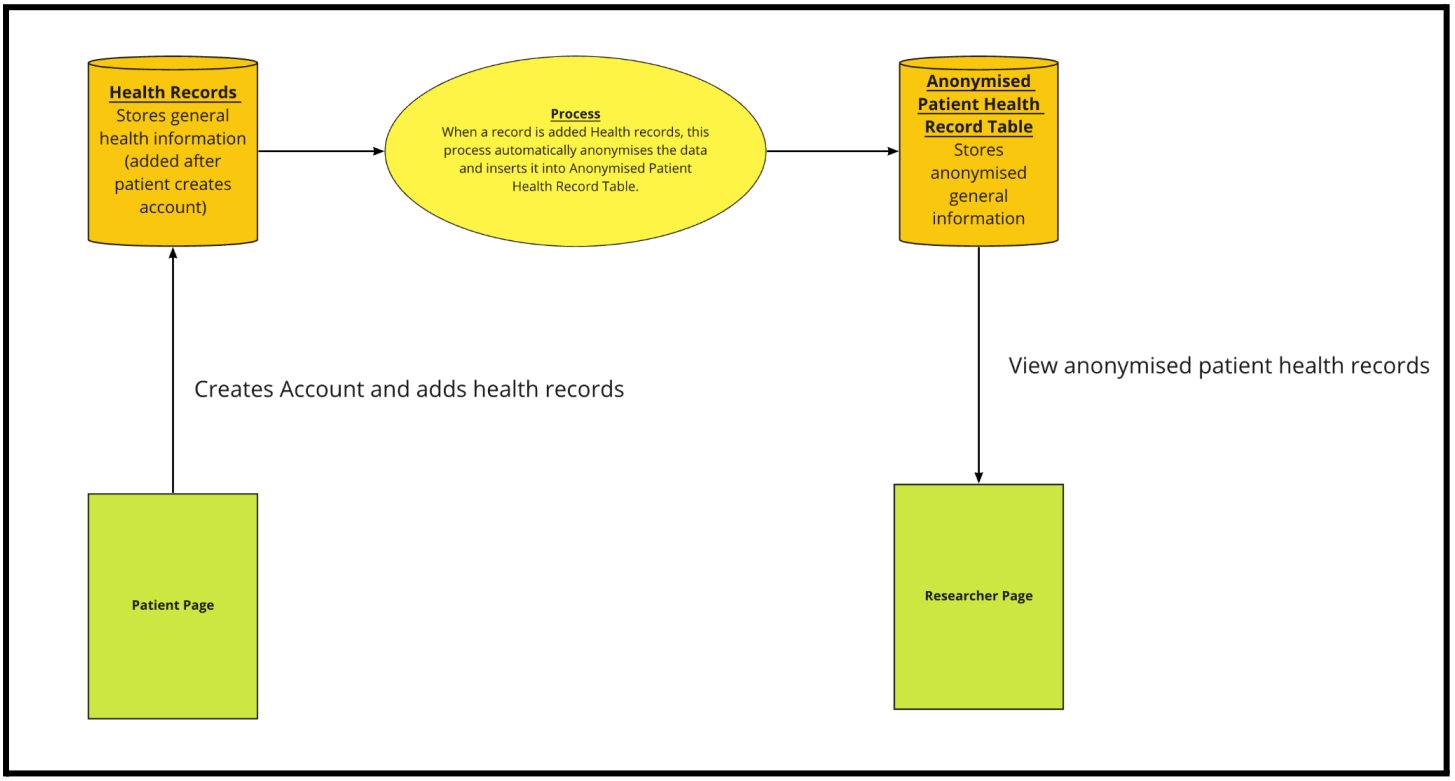

Figure 5: Anonymising Health Records Sub System Design

The table below shows how each column data will be anonymised.

Health Record Column	Process	AnonymisedRecords Column	Example
Unique Identifier	Use MD5 to hash the unique identifier	Hashed Unique Identifier	A89AUSDF9X → b13c7e985d881b2672 b649e3756eb437
Date of Birth	Get the year of birth and create a range of +/- 2 years, choose a random year within the range and create a new range of +/- 2	Range of year of birth	20/05/1998 → 1997 - 2001

	years using the chosen random year as the midpoint of the new range		
Height	Get the height and create a range of +/- 3cm, choose a random height within the range and create a new range of +/- 3cm using the chosen random height as the midpoint of the new range	Range of height	180cm → 179cm - 185cm
Weight	Get the weight and create a range of +/- 3kg, choose a random weight within the range and create a new range of +/- 3kg using the chosen random weight as the midpoint of the new range	Range of weight	54kg → 53kg - 59kg
Blood Type	*Not anonymised*	Blood Type	
Allergies	*Not anonymised*	Allergies	

Security Concerns

- Anonymised process between HealthRecord and AnonymisedRecord tables could be exposed in the SQL server. Processes that transmit data between the tables should be encrypted to ensure no MITM attack.
- SQL injection should not be possible if we parameterize all queries.
- Any database security practices or tools must be highly scalable to address distant and near-future requirements.

6. Viewing Crowd Sensor Count

General Description

This subsystem will utilise OpenCV IOT devices to detect the crowd count in a specific room and store it in a database table. This subsystem will also allow patients, doctors and medical staff to view the crowd sensor count through the web interface. The crowd count will be displayed in 2 different ways to the user. It will be displayed as a number count percentage and a traffic light display system.

1. Green - Clinic is relatively empty. (0% - 35% full)
2. Orange - Clinic is relatively crowded. (35%-70% full)
3. Red - Clinic is crowded. (70% - 100% full)

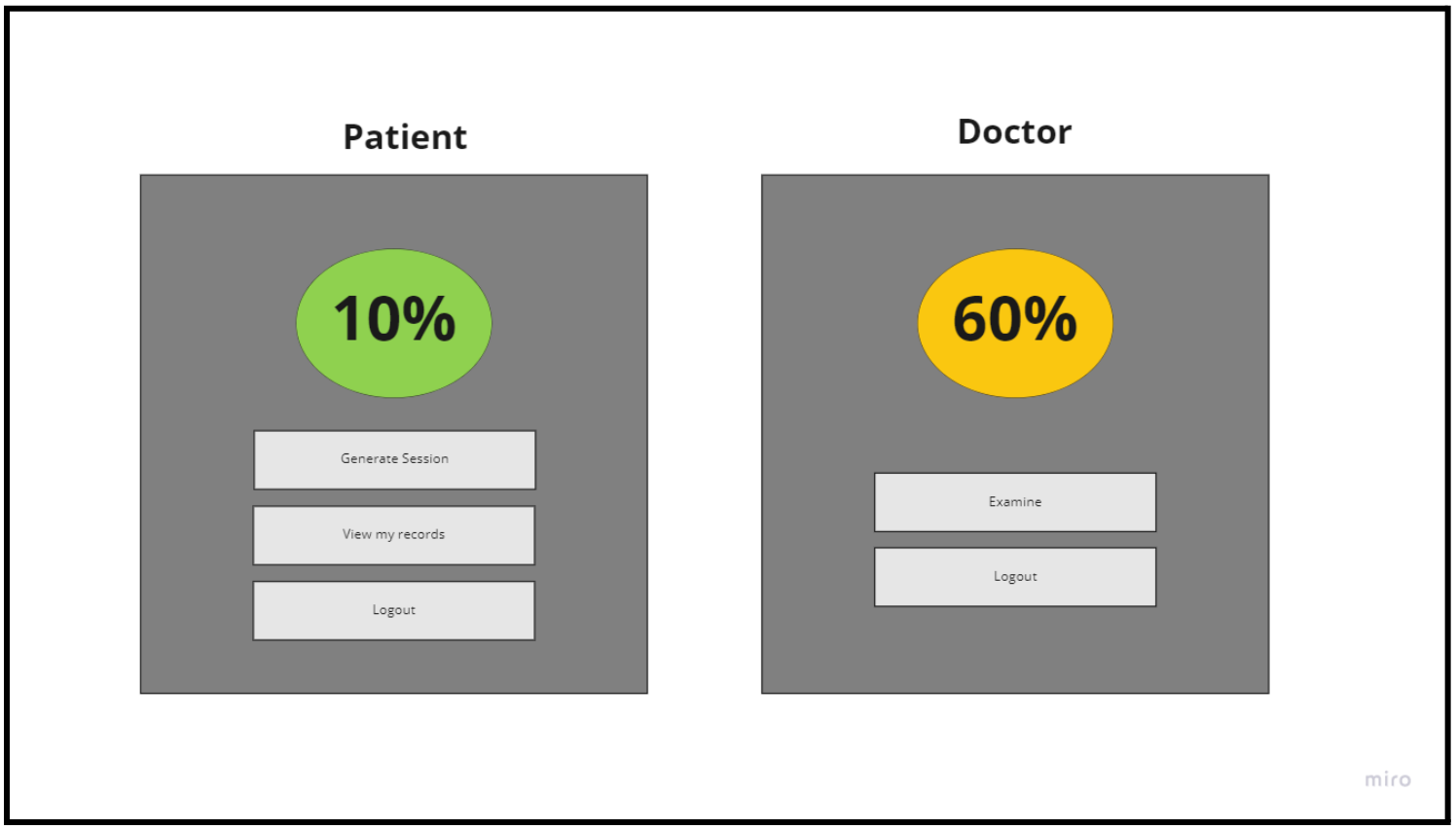

FIgure 6: Traffic light display system

Process:

Detection and storing of crowd count:

1. An OpenCV IOT device detects the number of people that are present in the room. The detection will occur in a 10 minute interval. After obtaining the count, it will be added to the Crowd table.

Viewing of crowd count:

1. In the patient, doctor and medical staff web interface, they will be able to view live updates on the crowd count in the clinic.

Implementation

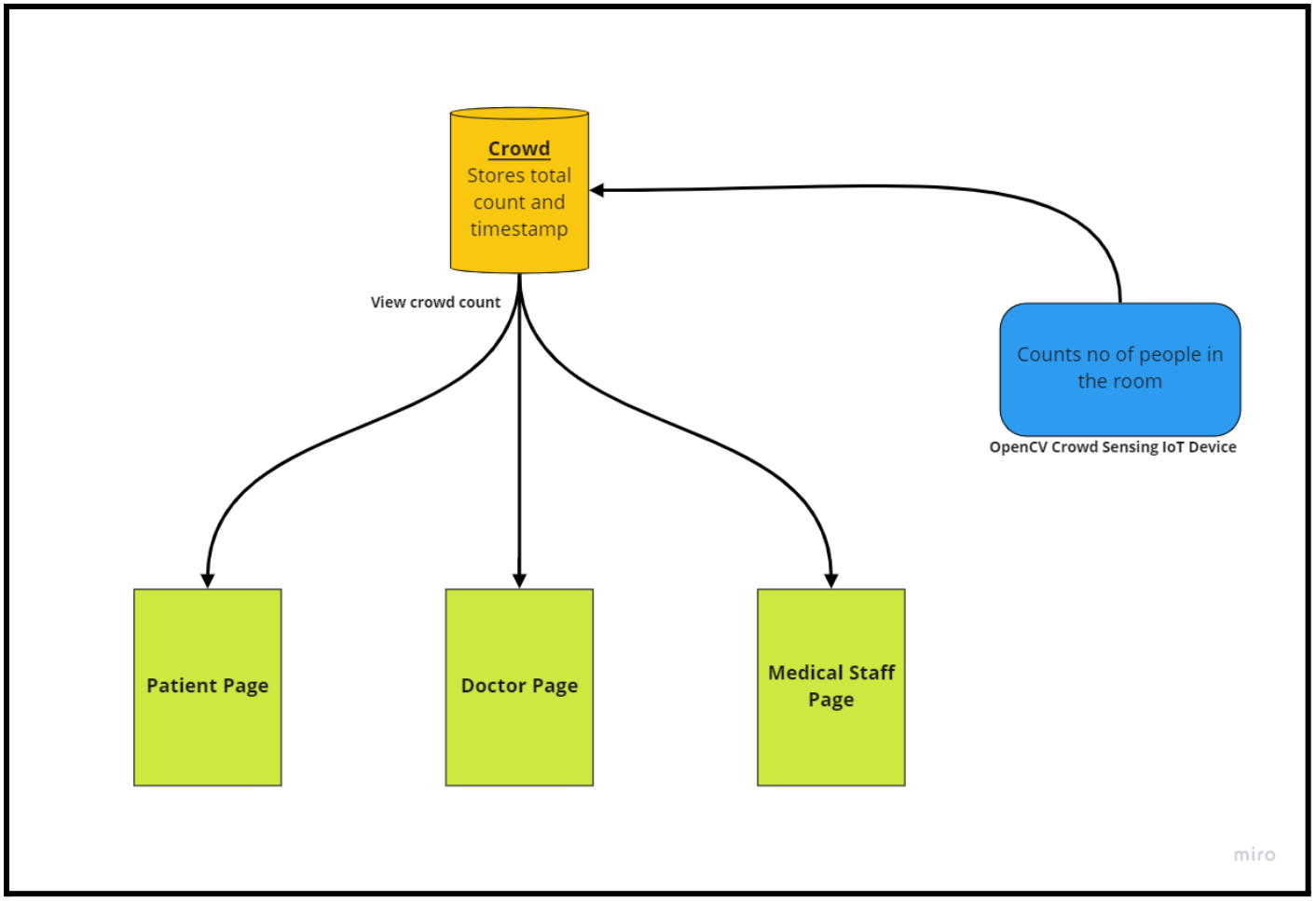

Figure 7: Crowd sensor Sub System Design

Interaction with Database

Crowd Table

As the IOT device will detect the number of people that are in the room every 10 minutes, a new record will be inserted into the crowd data table in 10 minute intervals. This record will store the timestamp in the (DD-MM-YYYY HH:MI) format followed by the crowd count as an integer. The primary key for the table will be the timestamp.

The patients, doctor and medical staff will be able to view the crowd count which will be taken from the crowd data table. The data obtained from their own device will be used to query the timestamp table. The latest record for the particular date obtained will be shown.

In the frontend, the crowd count obtained from the table will be compared with the max capacity of the clinic. A percentage will be calculated to show the crowd level and shown to the users as green, orange or red indicators.

Devices required

- Raspberry pi with camera

Security Concerns

- SQL injection should not be possible if we parameterize all queries.
- The attacker may be able to intercept any traffic between the camera and raspberry pi and raspberry and database server. The attacker may modify the count to display incorrect information. However, by using Secure Sockets Layer (SSL) and Transport Layer Security Protocol (TLS) and a strong encryption scheme such as SHA-256, the attacker should not be able to intercept the traffic and modify the count.
- The attacker should not be able to do any XSS attack through the web interface as the web interface only allows the user to view the crowd count. No input is required hence it reduces the possibility of XSS attacks.

7. Doctor and Patient

General Description

This subsystem provides doctors with the functionality to view the current patient's health record and all previous visits. This subsystem also allows doctors to record the details of the current visit like the diagnosis and prescriptions (if any).

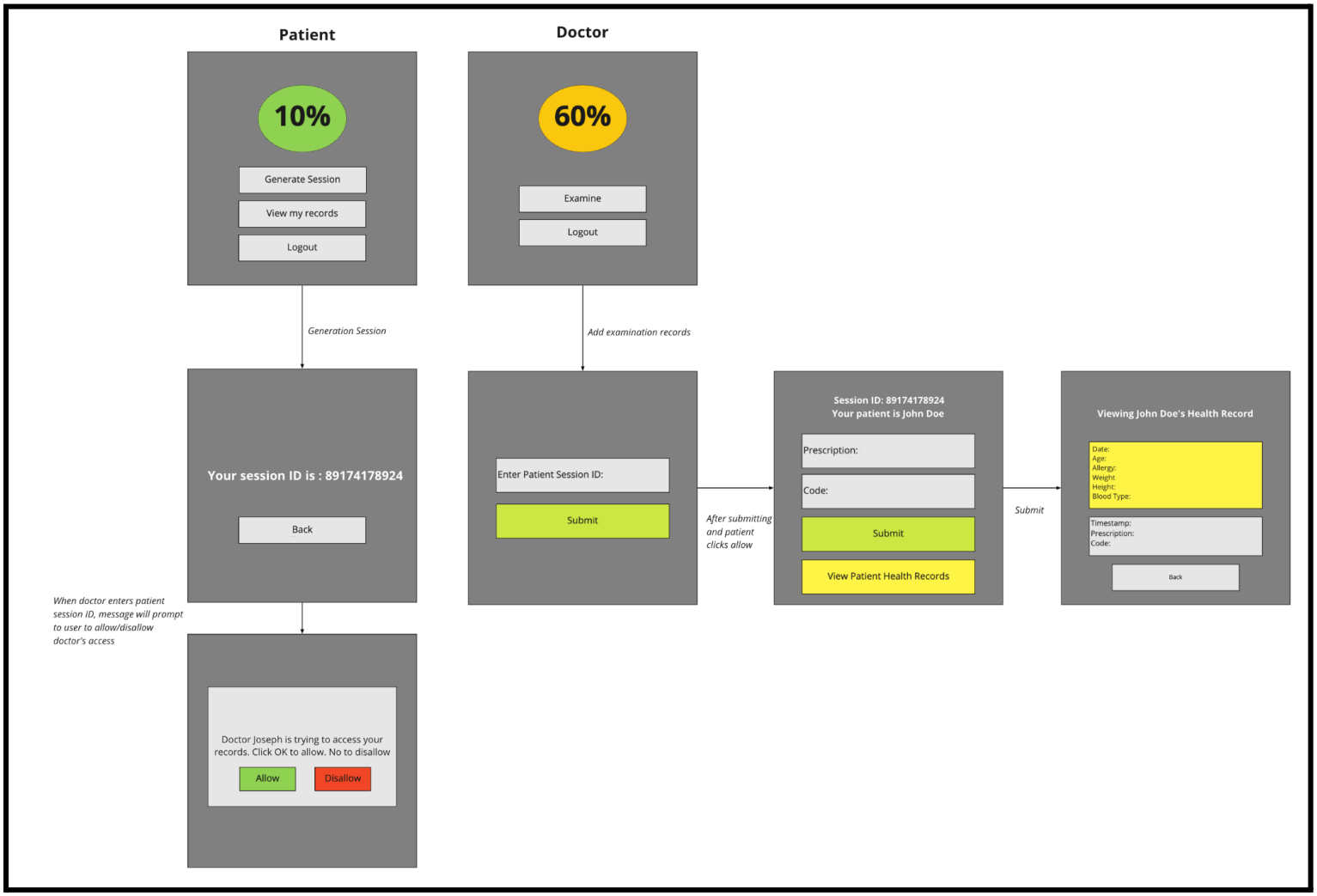

Figure 8: User Interface for Patient and Doctor Page

Process

1. When the patient enters the examination room, the patient creates a new session ID and shows it to the doctor.
2. The doctor enters the session ID which will send a confirmation dialog to the patient stating that the doctor is trying to access his/her records.
3. After the patient authorises the access, the doctor can now examine the patient and fill in the details accordingly.

4. When the examination is over, the doctor submits the form which will insert a new row into the database.

Interaction with Database

Examination table

A new record will be inserted every time a doctor finishes an examination of a patient. This record will store the patient UID, doctor UID, the diagnosis made, any prescriptions as well as the timestamp and session ID which are automatically generated.

The doctor also can view any past records in the examination table containing the same patient UID.

HealthRecords

The doctor can view the patient general health records which include details like their allergies, blood type, etc.

Diagnosis

Each examination will have a diagnosis code that describes the patient's condition.

Security Concerns

- Users can choose to view the details of their past examinations. This opens the possibility for reflected XSS where the doctor entered some malicious script into the prescription field for example. This can be prevented by ensuring that all data submitted after an examination is sanitised..
- SQL injection should not be possible if we parameterize all queries.
- There is a possibility that a malicious doctor attempts to brute force session IDs to gain unauthorised access to patient records. Since access to a patient's records requires the patient's approval through a confirmation dialog, this should not be a problem.
- Each patient is able to generate one unique session ID per day to prevent spam.