IS4205 Application Demo

Group 3

Group members:
Hiong Kai Han (A0199814M)
Michelle Toh Hui Ping (A0190377X)
Nguyen Thanh Duc (A0184534B)
Yang Kai Ze (A0183622H)

Introduction

SG COVID-SAFE
Your Health, Our Priority

SG Covid-Safe, is a prototype health record web application.

Primary Purpose: Tracking of Covid-19 related information such as vaccination status as well as Covid-19 test results.

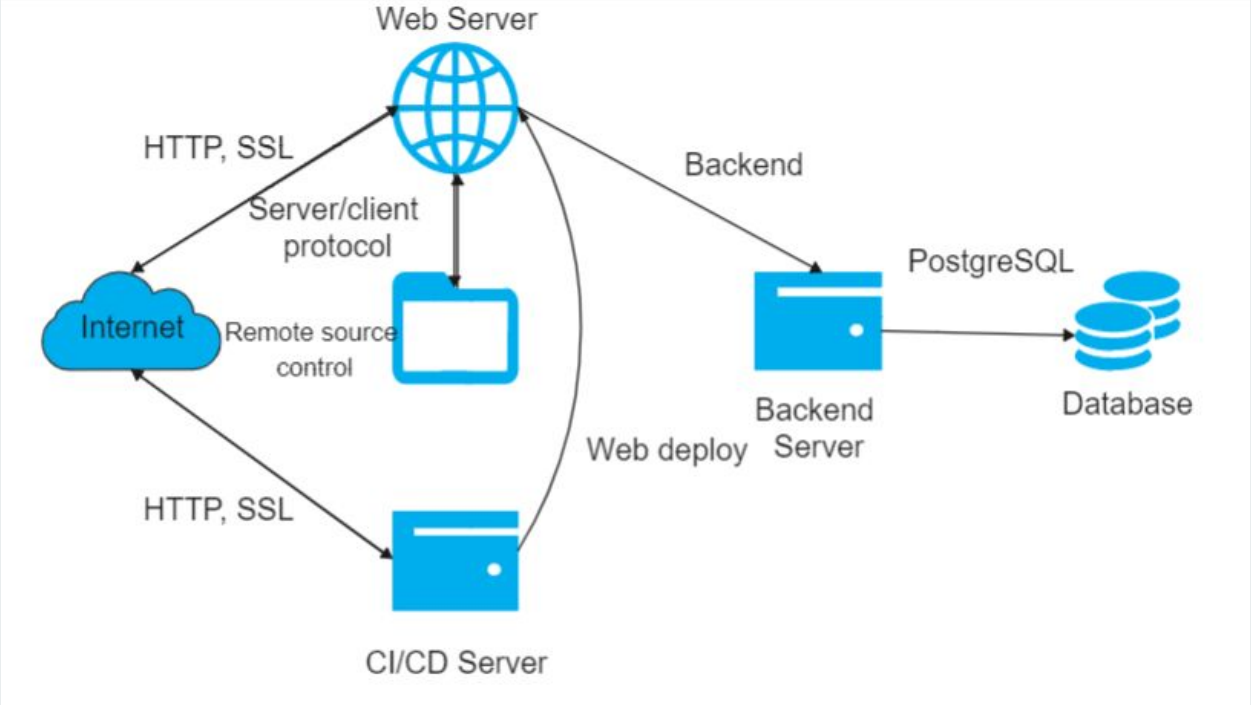
Secondary Function:

- Query information from a public database
- New Bulletin
- Health Declaration

Users

1) Admin Users
2) Covid Personnel Users
3) Public Users

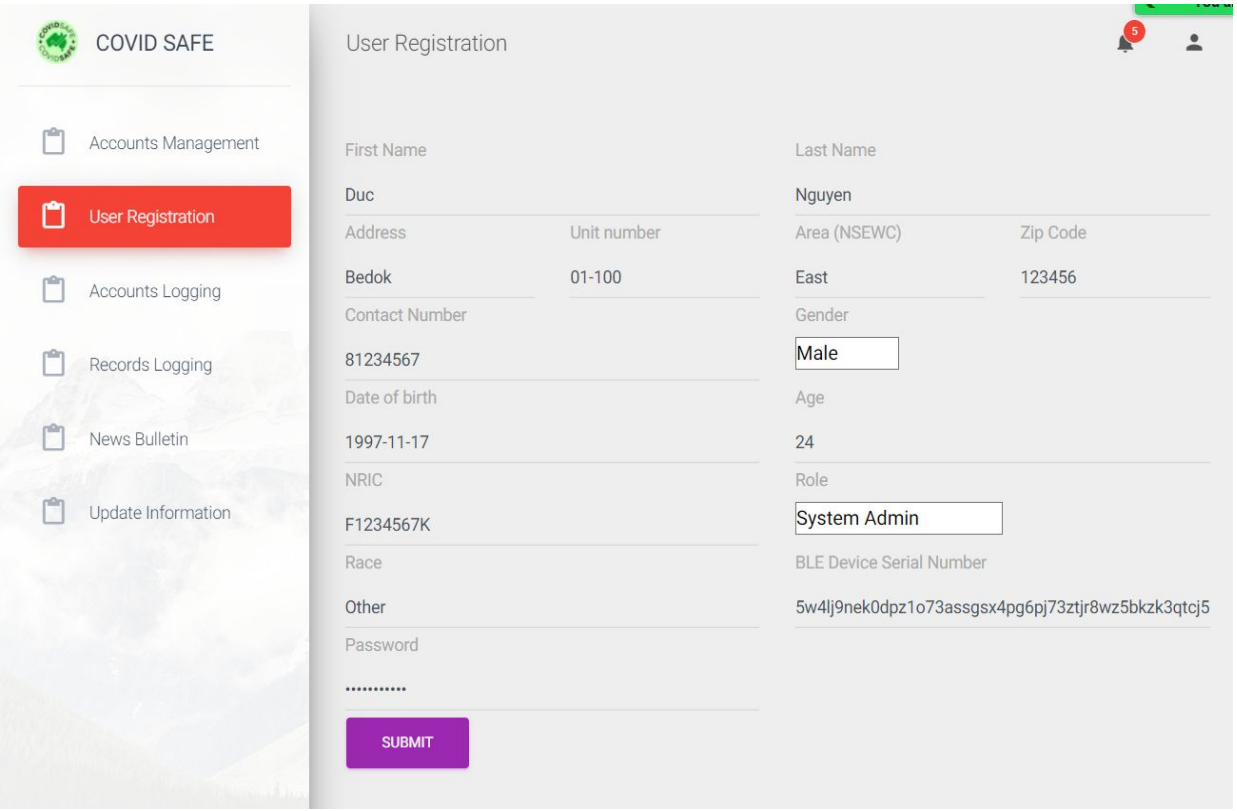
Application Infrastructure



Main Features

a. Registration
b. Login authentication
c. MFA
d. Password Reset
e. Upload Vaccination
f. Upload Test Result
g. Query database
h. Bulletin

Registration (ADMIN)



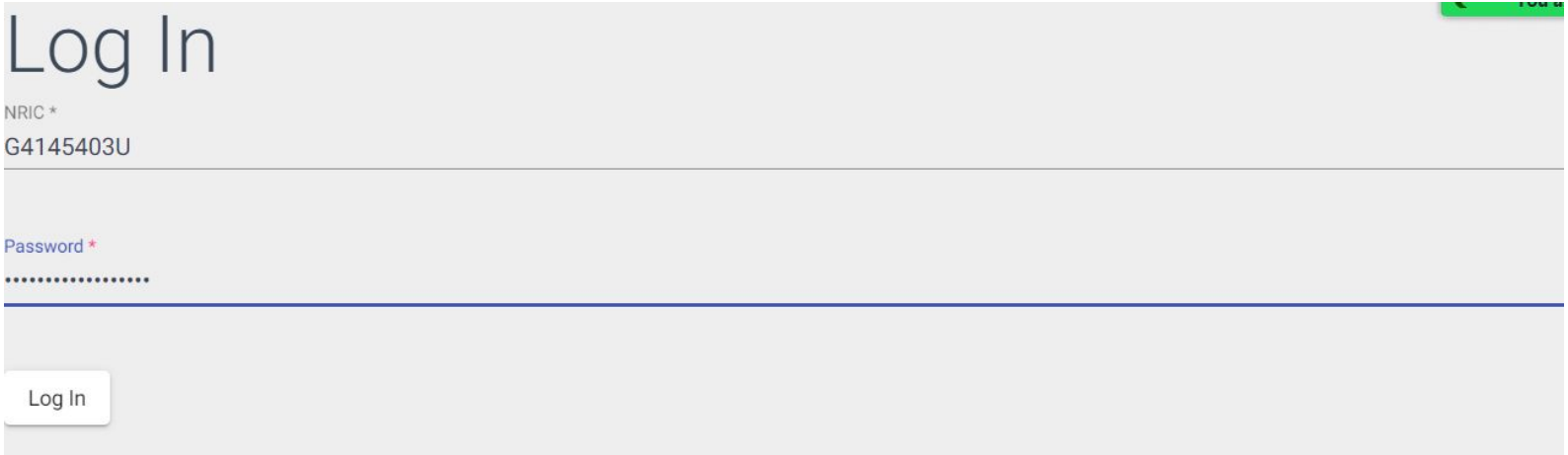
Update in Credential Database

```
credentials_encrypted=# SELECT nric,
credentials_encrypted-# hashed_password,
credentials_encrypted-# pgp_sym_decrypt(ble_serial_number::bytea,'mysecretkey') as ble_serial_number,
credentials_encrypted-# account_status,
credentials_encrypted-# pgp_sym_decrypt(account_role::bytea,'mysecretkey') as account_role
credentials_encrypted-# FROM login_credentials
credentials_encrypted-# WHERE nric = 'F1234567K';
   nric    |                      hashed_password                       |                        ble_seri
al_number                        | account_status | account_role
-----------+------------------------------------------------------------+--------------------------------
----------------------------------+----------------+--------------
 F1234567K | $2a$10$pVrKc3ui1VpfZtJW3bVdtudeT6fEddKUBjDlb1kQom9pwvY7t.Z7W | 5w4lj9nek0dpzlo73assgsx4pg6pj73
ztjr8wz5bkzk3qtcj5miexhqajka7re4c | 1              | 1
(1 row)
```

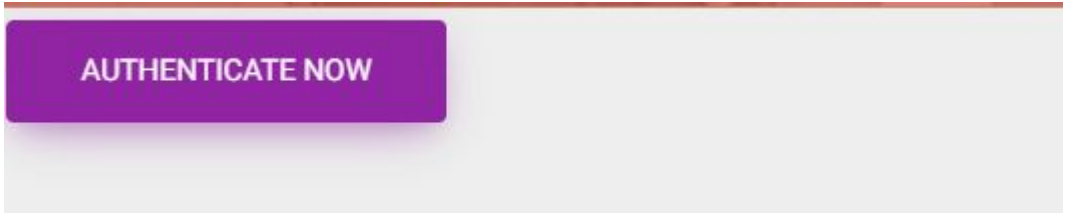
Update in Health Record Database

```
healthrecord_encrypted=# select nric,
healthrecord_encrypted-# pgp_sym_decrypt(first_name::bytea,'mysecretkey') as first_name,
healthrecord_encrypted-# pgp_sym_decrypt(last_name::bytea,'mysecretkey') as last_name,
healthrecord_encrypted-# pgp_sym_decrypt(date_of_birth::bytea,'mysecretkey') as date_of_birth,
healthrecord_encrypted-# pgp_sym_decrypt(age::bytea,'mysecretkey') as age,
healthrecord_encrypted-# pgp_sym_decrypt(gender::bytea,'mysecretkey') as gender,
healthrecord_encrypted-# pgp_sym_decrypt(race::bytea,'mysecretkey') as race,
healthrecord_encrypted-# pgp_sym_decrypt(contact_number::bytea,'mysecretkey') as contact_number
healthrecord_encrypted-# from user_particulars where nric = 'F1234567K';
   nric     | first_name | last_name | date_of_birth | age | gender | race  | contact_number
-----------+------------+-----------+---------------+-----+--------+-------+----------------
 F1234567K | Duc        | Nguyen    | 1997-11-17    | 24  | 1      | Other | 81234567
(1 row)
```

Login Authentication



Multi-Factor Authentication



+ Dongle

Reset Password (Admin)

Populate user information

NRIC

F1234567K

RETRIEVE

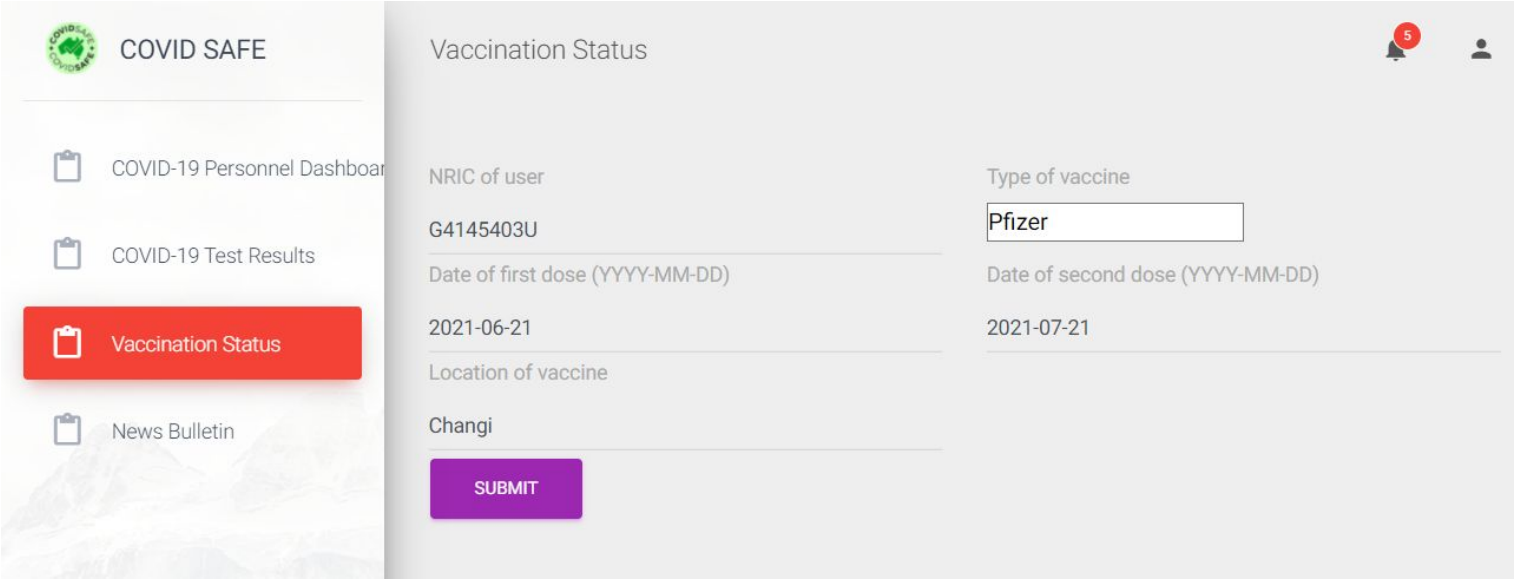
Change Password

New password

••••••••••••

UPDATE

Upload Vaccination Results (COVID 19 Personnel)

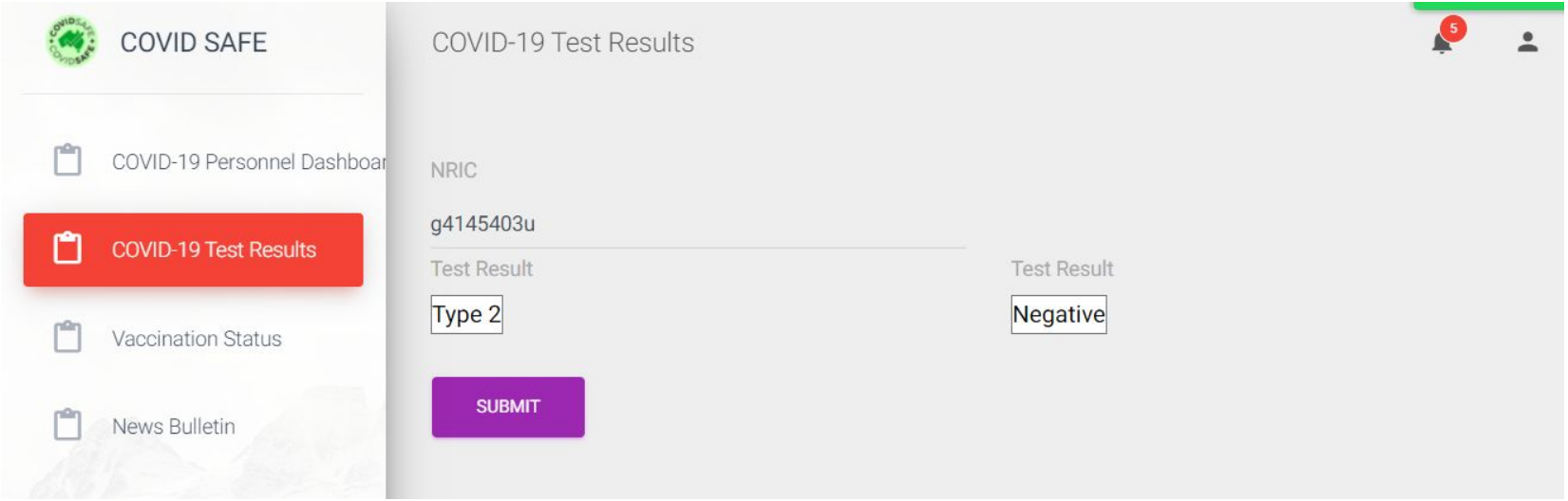


Upload Vaccination Results (COVID 19 Personnel)

```
healthrecord_encrypted=# select nric,
healthrecord_encrypted-# pgp_sym_decrypt(vaccination_status::bytea,'mysecretkey') as vaccination_status,
healthrecord_encrypted-# pgp_sym_decrypt(vaccine_type::bytea,'mysecretkey') as vaccine_type,
healthrecord_encrypted-# pgp_sym_decrypt(vaccination_centre_location::bytea,'mysecretkey') as vaccination_centre_location,

healthrecord_encrypted-# pgp_sym_decrypt(first_dose_date::bytea,'mysecretkey') as first_dose_date,
healthrecord_encrypted-# pgp_sym_decrypt(second_dose_date::bytea,'mysecretkey') as second_dose_date
healthrecord_encrypted-# from vaccination_results where nric = 'F1234567K';
   nric    | vaccination_status | vaccine_type | vaccination_centre_location | first_dose_date | second_dose_date
-----------+--------------------+--------------+-----------------------------+-----------------+------------------
 F1234567K | 0                  | pfizer       | Changi                      | 2021-09-28      | 2021-10-28
(1 row)
```

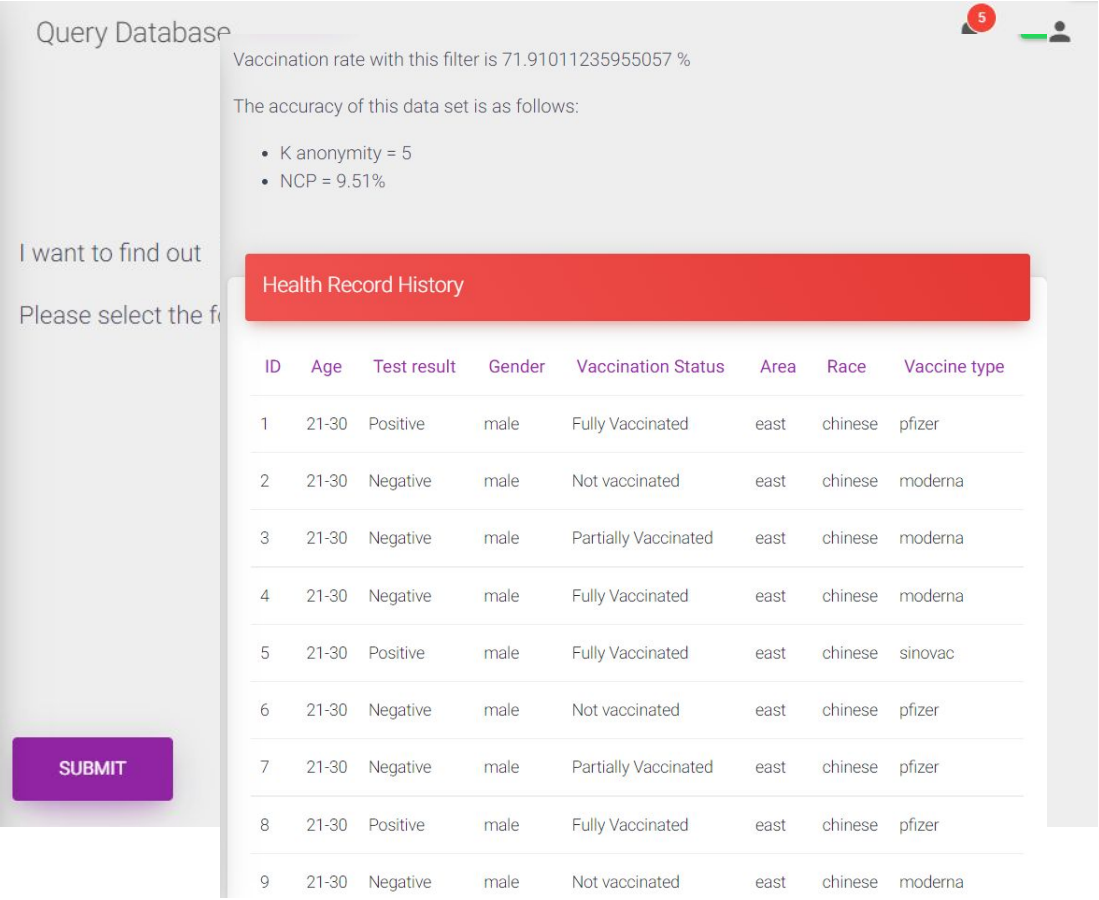
Upload Test Results (COVID 19 Personnel)



Upload Test Results (COVID 19 Personnel)

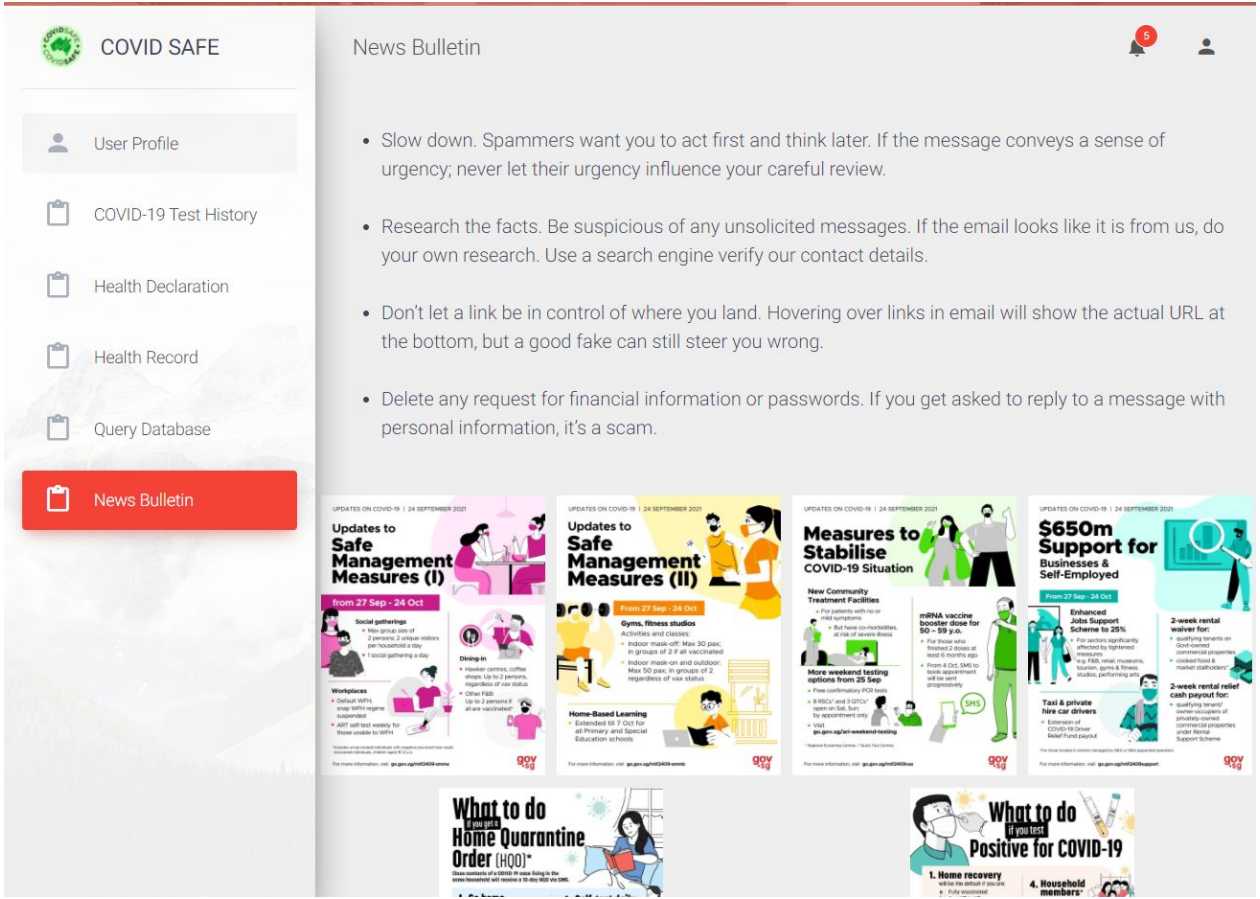
```
healthrecord_encrypted=# select nric,
healthrecord_encrypted-# pgp_sym_decrypt(covid19_test_type::bytea,'mysecretkey') as covid19_test_type,
healthrecord_encrypted-# pgp_sym_decrypt(test_result::bytea,'mysecretkey') as test_result
healthrecord_encrypted-# from covid19_test_results where nric = 'g4145403u';
   nric     | covid19_test_type | test_result
-----------+-------------------+-------------
 g4145403u | 2                 | 0
 g4145403u | 1                 | 0
(2 rows)
```

Query Database (Public User)



ID	Age	Test result	Gender	Vaccination Status	Area	Race	Vaccine type
1	21-30	Positive	male	Fully Vaccinated	east	chinese	pfizer
2	21-30	Negative	male	Not vaccinated	east	chinese	moderna
3	21-30	Negative	male	Partially Vaccinated	east	chinese	moderna
4	21-30	Negative	male	Fully Vaccinated	east	chinese	moderna
5	21-30	Positive	male	Fully Vaccinated	east	chinese	sinovac
6	21-30	Negative	male	Not vaccinated	east	chinese	pfizer
7	21-30	Negative	male	Partially Vaccinated	east	chinese	pfizer
8	21-30	Positive	male	Fully Vaccinated	east	chinese	pfizer
9	21-30	Negative	male	Not vaccinated	east	chinese	moderna

News Bulletin



Security Features

Describe the main security features that you implemented. (15 mins)

a. JSON web token - Postman - duc - 2 mins
b. BLE device as 2nd factor authentication & Encrypted serial number - Python, arduino -KH -3 mins
c. Single Session Usage - Show from application - KH - 1min
   https://github.com/ifsgrp3/Database-1/blob/master/encrypted_database_init.sql
d. Encrypted Database - PGAdmin (Mention claim + show adding, select-decrpyt, select-nondecrypt) - MICH -2 mins
e. Logging - 3 mins
   i. Simplified Logs for each DB (credentials, health record)
      - Show from application (duc shares screen)
   ii. Detailed logs
   - Show from database
f. Anonymisation for the database (Pycharm - show anonymisation function) -KZ - 3mins

Security Features

a. JSON web token
b. BLE device as Multi-Factor Authentication
c. Single Session Usage , Limited password attempts, Password hashing
d. Database Encryption
e. Logging
f. Anonymisation for the database

Security Features

JSON Web Token

COVID Safe Backend / Credentials / **Account logs**

Save

GET {{url}}/auth/acc/logs

Send

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJucmljIjoiRzQ...	
Key	Value	Description

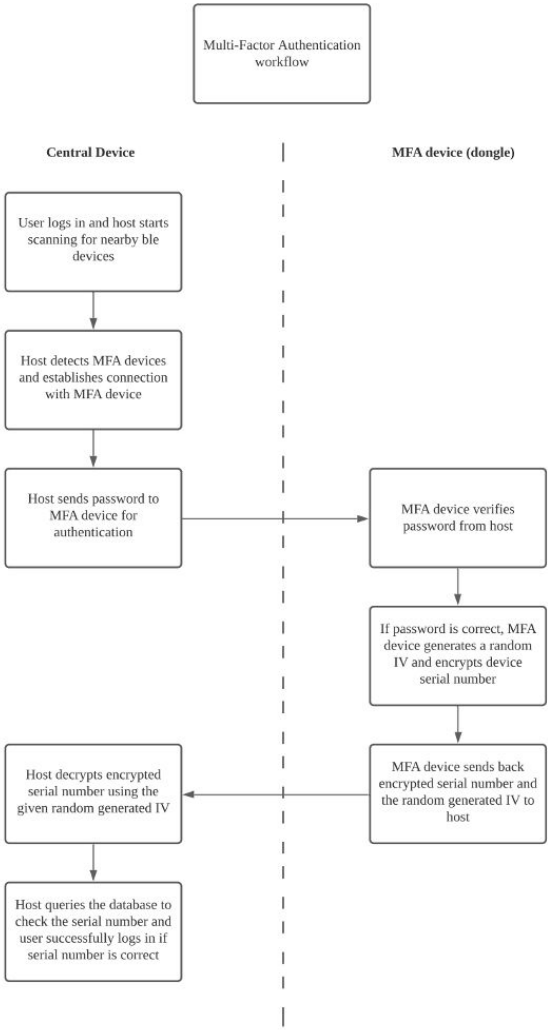
Body | Cookies | Headers (9) | Test Results

Status: 200 OK Time: 4 ms Size: 320 B Save Response

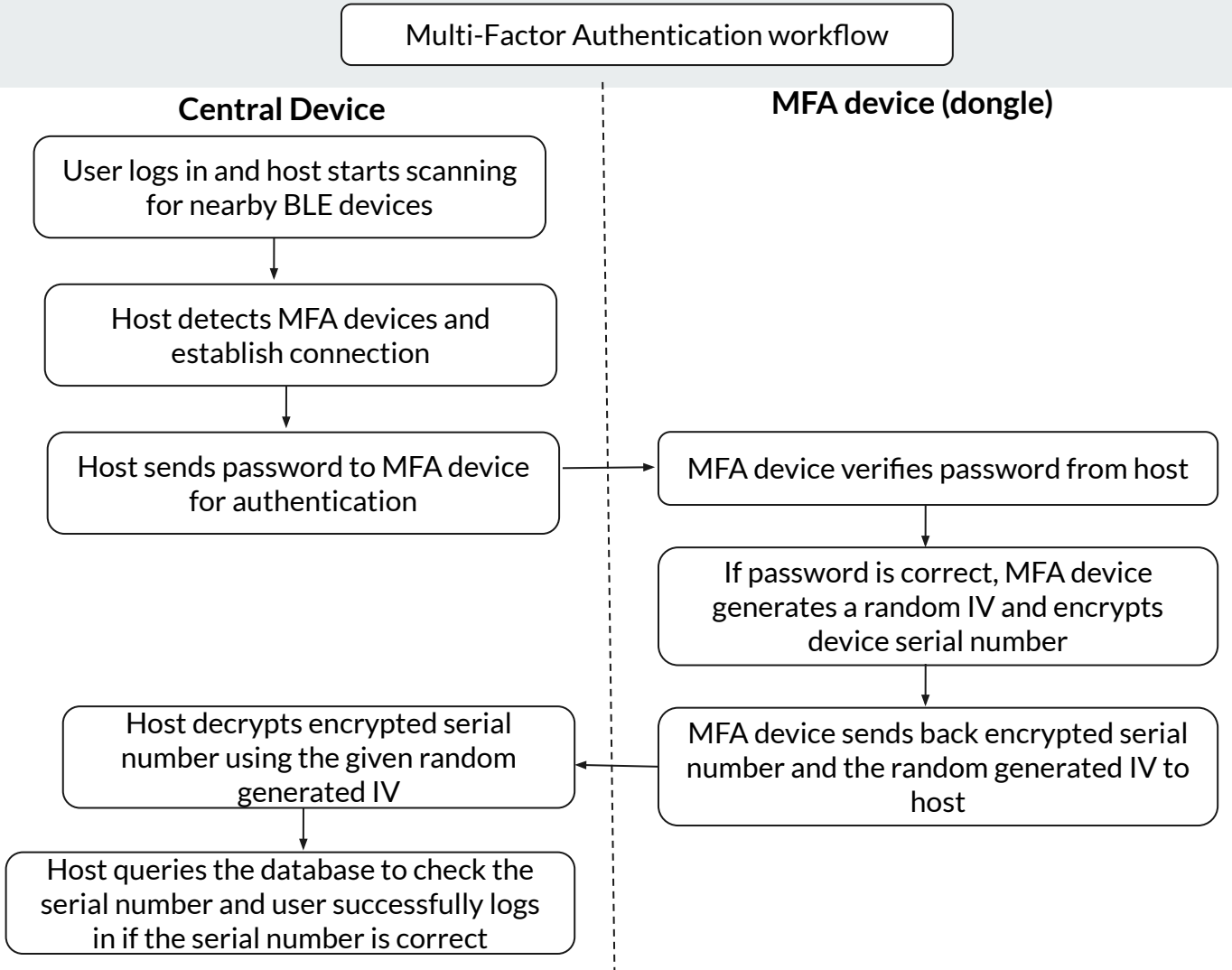
Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "status": 404
3 }
```

```
11        },
12        {
13            "age": "41-50",
14            "test_result": "Positive",
15            "gender": "male",
```


Multi-Factor Authentication workflow
Central Device
MFA device (dongle)
User logs in and host starts scanning for nearby ble devices
Host detects MFA devices and establishes connection with MFA device
Host sends password to MFA device for authentication
MFA device verifies password from host
If password is correct, MFA device generates a random IV and encrypts device serial number
MFA device sends back encrypted serial number and the random generated IV to host
Host decrypts encrypted serial number using the given random generated IV
Host queries the database to check the serial number and user successfully logs in if serial number is correct

Multi-Factor Authentication



```
MFA device found: ble_device_9
Connected to ble_device_9
Receiving transmission...
Encrypted serial number: E60810167353561792032DA28F5932FA6791E6A31C7502CA8EFEB4C9567A5CEE2EB5199CC60721B91781432EA1F2321153C224B79464
2C2C01A969FFEC5833D2
Serial number: 5w4lj9nek0dpz1o73assgsx4pg6pj73ztjr8wz5bkzk3qtcj5miexhqajka7re4c
```

```
MFA device found: ble_device_9
Connected to ble_device_9
Receiving transmission...
Encrypted serial number: 4742AE83C730E205D3AC7237C85A37320B85558DFF57B1601B2C093F67C647528816DA8217C2254A917398E4764F17ACC29B07D2EB12
4AA0DE28B25506340E1D
Serial number: 5w4lj9nek0dpz1o73assgsx4pg6pj73ztjr8wz5bkzk3qtcj5miexhqajka7re4c
```

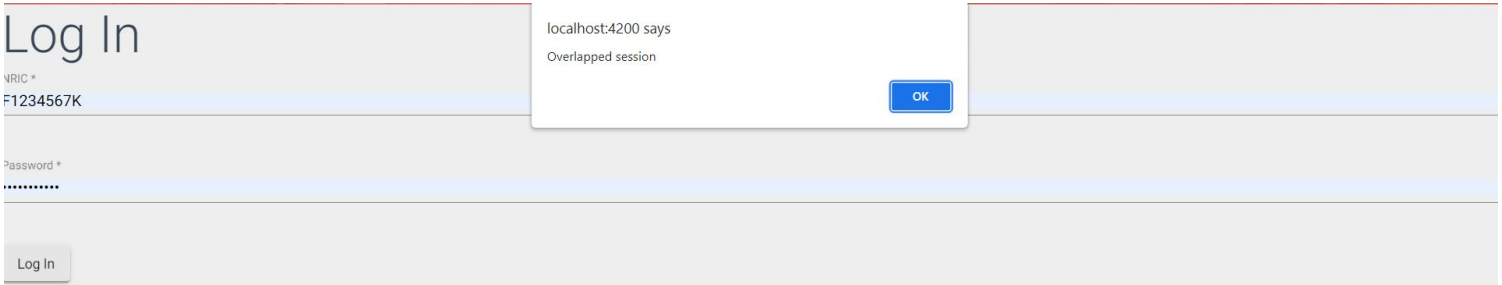
```
ble_device_9 device restarting...
.............................
Password success
Generating random iv
Encrypting serial number
Sending encrypted data
........
.......................................
```

```
ble_device_9 device restarting...
.........................................
..
```

Autoscroll Show timestamp Newline 9600 baud Clear output

Single Session Usage

```
credentials_encrypted=# select * from online_users;
   nric
-----------
 G4145403U
(1 row)
```



Limited Password Attempts

```
else {
  await db.query(
    "UPDATE login_credentials SET password_attempts = (password_attempts::INTEGER + 1)::VARCHAR WHERE nric = $1" ,
    [nric]
  );
  if( newData[0].password_attempts > 10) {
    return await deactivate({ nric: nric });
  }
    return {
        error: 'Invalid username or password'
    };
}
```

Password Hashing and random generated salt

```
const salt = bcrypt.genSaltSync(10);
let hashed_password = bcrypt.hashSync(this.form.value.password, salt);
```

```
// return { token }
const compareRes = await bcrypt.compare(password, data
    if (compareRes) {
      const user = await db.query(
        "SELECT * from online_users WHERE nric = $1" ,
        [nric]
      );
      const userData = helper.emptyOrRows(user);
      console.log(userData)
      if (userData.length > 0) {
        return {
          error: 'Overlapped session'
        };
      } else {
        await db.query(
          "CALL add_online_user($1)" ,
          [nric]
        );
        await db.query(
          "UPDATE login_credentials SET password_attem
          [nric]
        );
        const token = jwt.sign(
          data[0], secret, { expiresIn: '7d' }
        );
        return { token };
      }
```


```
QZAGkZcwX,795032299fdbb2f4791c884fd78b11645f472e0c46d6abf74f219da5b9ae12bf,6a5aae41a0994917935bbd5221c1fe64
ImipXgmvIEt,eda5c2f784b7d6e1b1eff12d2418ab9d7e002be9abc5118ddf0f66625400ea51,82250b4002d94fe292fbf7a5ef3cb533
M6lUFPilyFc8Xc4sJlg,4040049d0b44214c995b0a806ec9f3be676beadc90f8706a845c58b0eb639806,d5ab449b0c15478f8b56b25230
xzXgFYzDJyQ,871b7a2acd5ee0923b8941134e2303536810d168eae4a0eabaa92bf186cffbe2,fbb9aca0c778415281a0ef527dc780d5
aMQuOz36tYWtxZ,1b765a28647cf77680896484ff5a8ac6e3acdee7fa82e7e5594d77adb8fb4d8b,524f75b22dad4800ace34b3777f6876
1dlmbMBw6DF6FPmQN,67687f0dfe98ad99adfa9d3fdd7cd565f975de3ad3793bd428798e9bda8a3b89,585d9d20373b447d9f6f70336058
kvsWLzSRT,86cad2d455286b0585cc7cc9775d7b2e6afb0efa1b513e8ce3b07290a7f56c30,30141356d52843358eb9b45455e0debf
q4C1EnbNYCM2wjWLw,5bf0f3a5ae243c9dbd7d497488ad4e13b63b2280c41699b54f60cc23cb39a90a,27f41b1f4f4c44f4bd6de628aba8
lmtbFIH5e,70bd416151398c5d79c96d1396cd2cb5c81cdd67c3ccac0eae3e0774c3441003,80016b6adc9a478ab59222fa1db2657f
VFgJC6qw6RvqCVbR0V,5d4bcc6c3e82689c486cd39c28a81524a741c57d087b6840daae4c10f97c2be1,0ab06d7b56b040fd826235c9b60
iWWY8KsOm4L1sC,6a6cd9f7bb227425cb2640bed453dcd4ffb4ee3809403757f87f01bea47cbf39,69291342076649d4b983433021fdf6f
kC3JIWpjJ3tQ,873b382f1793f86f0f69e0003583925968b7369486015b38ce9fefbad2680d75,9921a70755274a32ac4bb3fbc6a48d5c
19sZySxX9sw6zRQ,3b4c057c165a414d3fd0cda7dff9afe215de04c39c0df856f03323b9e8f7bce2,de9ec761e4424096ac48538b5c64f5
```

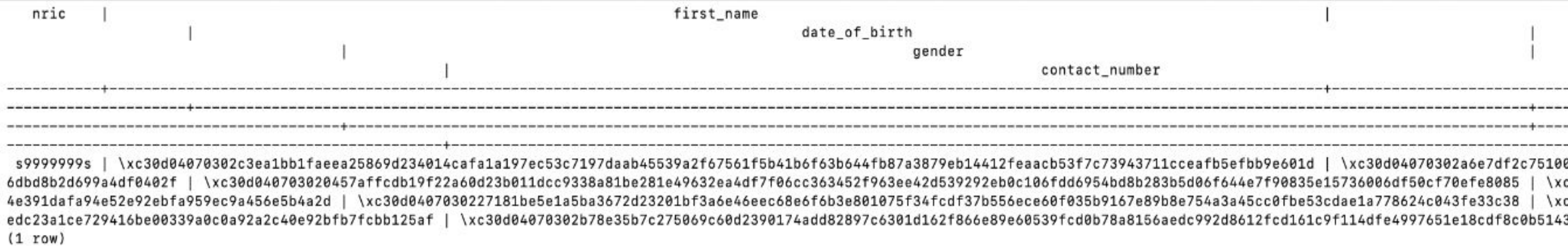
Encrypted Database

1. Add user particulars by calling the procedure add_user_particulars

```
CALL add_user_particulars ('s9999999s', 'toh', 'cher', '10/21/2021', '3', '0', 'chinese', '98999999');
```

2. Select the user particulars for the individual with nric='s9999999s'

```
select * from user_particulars where nric='s9999999s';
```



```
    nric    |                                                                    first_name                                                                      |
                      |                                                                          date_of_birth                                                                            |
                                         |                                                                    gender                                                                      |
                                                     |                                                                        contact_number
------------+----------------------------------------------------------------------------------------------------------------------------------------------------+----------------------------
----------------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------+----
-----------------------------------------+------------------------------------------------------------------------------------------------------------------------------------------------+----
-----------------------------------------------------+-----------------------------------------------------------------------------------------------------------------------------------------
 s9999999s | \xc30d04070302c3ea1bb1faeea25869d234014cafa1a197ec53c7197daab45539a2f67561f5b41b6f63b644fb87a3879eb14412feaacb53f7c73943711cceafb5efbb9e601d | \xc30d04070302a6e7df2c7510
6dbd8b2d699a4df0402f | \xc30d040703020457affcdb19f22a60d23b011dcc9338a81be281e49632ea4df7f06cc363452f963ee42d539292eb0c106fdd6954bd8b283b5d06f644e7f90835e15736006df50cf70efe8085 | \xc
4e391dafa94e52e92ebfa959ec9a456e5b4a2d | \xc30d0407030227181be5e1a5ba3672d23201bf3a6e46eec68e6f6b3e801075f34fcdf37b556ece60f035b9167e89b8e754a3a45cc0fbe53cdae1a778624c043fe33c38 | \xc
edc23a1ce729416be00339a0c0a92a2c40e92bfb7fcbb125af | \xc30d04070302b78e35b7c275069c60d2390174add82897c6301d162f866e89e60539fcd0b78a8156aedc992d8612fcd161c9f114dfe4997651e18cdf8c0b5143
(1 row)
```

Return result: all columns are encrypted except the nric

3. Obtain the decrypted user particulars

```
select nric, pgp_sym_decrypt(first_name::bytea, 'mysecretkey') as first_name,
pgp_sym_decrypt(last_name::bytea, 'mysecretkey') as last_name,
pgp_sym_decrypt(date_of_birth::bytea, 'mysecretkey') as date_of_birth,
pgp_sym_decrypt(age::bytea, 'mysecretkey') as age, pgp_sym_decrypt(gender::bytea, 'mysecretkey')
as gender, pgp_sym_decrypt(race::bytea, 'mysecretkey') as race,
pgp_sym_decrypt(contact_number::bytea, 'mysecretkey') as contact_number from user_particulars
where nric='s9999999s';
```

nric	first_name	last_name	date_of_birth	age	gender	race	contact_number
s9999999s	toh	cher	10/21/2021	3	0	chinese	98999999

(1 row)

Simplified Logging

Accounts Logging

Account Logging

Log ID	Action	User NRIC	Date and Time
1	CREATE	F0058169X	2021-10-27T05:57:24.036Z
2	CREATE	G0272727M	2021-10-27T05:57:24.051Z
3	CREATE	S2557244J	2021-10-27T05:57:24.056Z
4	CREATE	F2137800M	2021-10-27T05:57:24.061Z
5	CREATE	F3609454X	2021-10-27T05:57:24.066Z
6	CREATE	G4144820U	2021-10-27T05:57:24.070Z
7	CREATE	F1941406W	2021-10-27T05:57:24.075Z
8	CREATE	G4367973U	2021-10-27T05:57:24.079Z
9	CREATE	G4114743T	2021-10-27T05:57:24.084Z
10	CREATE	G4145403U	2021-10-27T05:57:24.089Z

COVID SAFE

- Accounts Management
- User Registration
- Accounts Logging
- Records Logging
- News Bulletin
- Update Information

Records Logging

Record Logging

Record ID	Action	Table Affected	Date and Time
1	CREATE	user_particulars	2021-10-27T05:55:14.060Z
2	CREATE	user_particulars	2021-10-27T05:55:14.072Z
3	CREATE	user_particulars	2021-10-27T05:55:14.079Z
4	CREATE	user_particulars	2021-10-27T05:55:14.086Z
5	CREATE	user_particulars	2021-10-27T05:55:14.093Z
6	CREATE	user_particulars	2021-10-27T05:55:14.100Z
7	CREATE	user_particulars	2021-10-27T05:55:14.107Z
8	CREATE	user_particulars	2021-10-27T05:55:14.114Z
9	CREATE	user_particulars	2021-10-27T05:55:14.121Z
10	CREATE	user_particulars	2021-10-27T05:55:14.128Z

Detailed Logging

- All the logs for the postgres docker is stored in the logs database
- Logs are stored in a table with two columns; log_time and activity

```
logs=# select * from logs_data limit 10;
        log_time          |                                                                    activ
ity
---------------------------+-----------------------------------------------------------------------
-----------------------------------------------------------------
 2021-10-27 03:10:29.758 UTC |  [1] LOG:  starting PostgreSQL 13.4 (Debian 13.4-1.pgdg100+1) on x86_64-pc-l
inux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
 2021-10-27 03:10:29.758 UTC |  [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
 2021-10-27 03:10:29.759 UTC |  [1] LOG:  listening on IPv6 address "::", port 5432
 2021-10-27 03:10:29.766 UTC |  [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
 2021-10-27 03:10:29.774 UTC |  [26] LOG:  database system was shut down at 2021-10-27 02:50:04 UTC
 2021-10-27 03:10:29.781 UTC |  [1] LOG:  database system is ready to accept connections
 2021-10-27 03:11:28.682 UTC |  [42] FATAL:  role "postgres" does not exist
 2021-10-27 03:15:41.524 UTC |  [1] LOG:  received fast shutdown request
 2021-10-27 03:15:41.529 UTC |  [1] LOG:  aborting any active transactions
 2021-10-27 03:15:41.533 UTC |  [1] LOG:  background worker "logical replication launcher" (PID 32) exited
with exit code 1
(10 rows)
```

Anonymisation for the database

Pycharm Live Demo

Anonymisation for the database

root@group3-1-i: ~

```
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@daily /usr/bin/python3 /home/sadm/IFS/K-anonymity-auto-Transformation/data/database_reader.py
@daily /usr/bin/python3 /home/sadm/IFS/K-anonymity-auto-Transformation/anonymizer.py
@daily /usr/bin/python3 /home/sadm/IFS/K-anonymity-auto-Transformation/data/database_uploader.py
@monthly /usr/bin/python3 /home/sadm/IFS/Database-1/delete_older_health_declaration.py
@daily  PGPASSWORD="[illegible]" /usr/lib/postgresql/13/bin/pg_dumpall -h group3-1-i.comp.nus.edu.sg -p 5435  -U po
stgres > /home/sadm/IFS/all.sql
@daily /usr/bin/python3 /home/sadm/IFS/db_docker/postgres_logging/convert_sql.py
@daily /usr/bin/python3 /home/sadm/IFS/db_docker/postgres_logging/add_logs.py
                                                                                    30,1          Bot
```

Security Claims in the application

Security Claims in the application

8.0. Security Claims

- 8.1. SQL Injection
- 8.2. Cross Site Scripting
- 8.3. Access Control/ Role Authentication
- 8.4. Transport Layer Security
- 8.5. Single Session Usage
- 8.6. Cross Site Request Forgery
- 8.7. Multi Factor Authentication
- 8.8. K-Anonymity
- 8.9. User Credentials Authentication
- 8.10. Social engineering
- 8.11. Bluetooth or wifi sniffing attacks
- 8.12. Non-Repudiation
- 8.13. Security of database

5 Main Security Claims

SQL Injection

Access Control/ Role Authentication

Multi Factor Authentication

Non-Repudiation

Security of database

SQL Injection

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. Several mechanisms are put in place to prevent SQL injection throughout the web application.

The following mechanisms are listed below.

- Defensive programming
  - Input Validation
  - Parameterized queries
  - Stored Procedures

Attacker Claims:
Attackers are not able to access the database by any other means apart from the UI.
Attackers do not have access to stored procedures and functions.

Access Control/ Role Authentication

Attackers are unable to have any unauthorized access to the database server and backend server.
Access to any systems or services that are limited by a “need to have” basis. All servers except the web server are configured not to be exposed to the public. The database servers and backend servers are configured to only accept connections from within a docker network.

It is also not possible for users to perform actions outside their stipulation permission specifications.
Users are unable to access the URL of the application without proper authentication. To verify the user role, a JSON web token is used as a verification method to ensure specific role based actions can only be performed by a user with valid tokens.

Attackers claim: Attackers do not have access to our docker network.
A valid JSON token is not known by attackers.

Multi Factor Authentication

It is not possible to log in without both username and password as well as their MFA device.

Even if an attacker manages to steal the username and password of the user, they will not be able to log in to the user's account without the MFA device.
Furthermore, as we are using bluetooth low energy, the MFA devices have a short range, which limits attackers as they have to get very close to the MFA device to be able to log in.
The MFA devices also encrypt all information with random generated IVs, making it hard for attackers to obtain the serial numbers and replicate a MFA device.

Attacker claim: Attackers are not able to log in without both the password and the dongle.
Attackers are not able to see the code and secret key that has been uploaded on the devices.

Non-Repudiation

Non-repudiation is the assurance that **someone would not be able to deny the validity of something**. It proves the origin and integrity of data.
This log file will be generated at the end of every day and within this log file, it contains information of the username of the individual who has done a certain action such as inserting or updating values. These log files are persistent which means that even if the docker containers restarts, it will still be present and not be deleted. This allows us to track back and ensures non-repudiation.

Attacker claim: Attackers should have no ability to log into our database to update or delete the logs table.

Security of database

In the event that an attacker is able to get hold of the database login credentials, he/she would not be able to obtain any sensitive data from the database.

To ensure the security of the database, we encrypted the values in the database.
Even with the database, he/she would not be able to log into the web application as he does not have the password of individuals. This prevents the leak of information and thus, ensures confidentiality.

Attacker's claim: In this case, we assume that the attacker would not be able to see the secret key used for symmetric encryption for the database as he does not have access to the open source code. The attacker only has access to the encrypted database.

Live UI Demo

END OF PRESENTATION

Any questions?

User accounts

```
Admin:
Username: f2057642k
Password: QZAGkZcwX
Hashed password = $2a$10$tzsFDAsq9632rd94rJkMf.PYasIXDkXQc7Ux0eKURGaeEFQJOJd4K
Salt = $2a$10$tzsFDAsq9632rd94rJkMf.
BLE:5w4lj9nek0dpz1o73assgsx4pg6pj73ztjr8wz5bkzk3qtcj5miexhqajka7re4c

Covid Personnel:
Username: g1271758q
Password: ImipXgmvIEt // Hashed password=$2a$10$ulkRGnz6fX.0QiZm7SltFetp9O3icUJPPUgapLbJ7ZJU0jCfnKuey
Salt=$2a$10$ulkRGnz6fX.0QiZm7SltFe

BLE: kpnz5r392si6cm3497ohj74spxsx13gjvagz09n9ynrvdu8pnr51k3zf1bha32po

Public User:

Username:s3616980z

Password:VFgJC6qw6RvqCVbR0V  Hashed password=$2a$10$ZAP4y9MIr4QsLLqy1BstEOBOv2VZRHEM3bKVyFR7ZKWehlRb3i2FK
Salt=$2a$10$ZAP4y9MIr4QsLLqy1BstEO

BLE:m6cf0lxfncuy4gsckde7doudhzxfk7z1qe0bvcimurtb5x48sstzc5vr0n3g5mmk
```