C19 Trace

By Eoin Wilkie

A dissertation submitted to the
School of Science and Computing
Galway-Mayo Institute of Technology
In Partial Fulfillment of the Requirements
for the Degree in Software Development
May 2021

Supervised by Dr. Brian Ginley

Contents

1 Introduction

1.1 About

With the outbreak of the COVID-19 pandemic in 2019, many schools, workplaces and other public building have been closed as society has tried to slow the spread of the disease. By tracking attendances and adding a COVID-19 status to each, this project aims to aid in the return to normality at GMIT campuses.

GitHub
https://github.com/c19trace

1.1.1 System Design

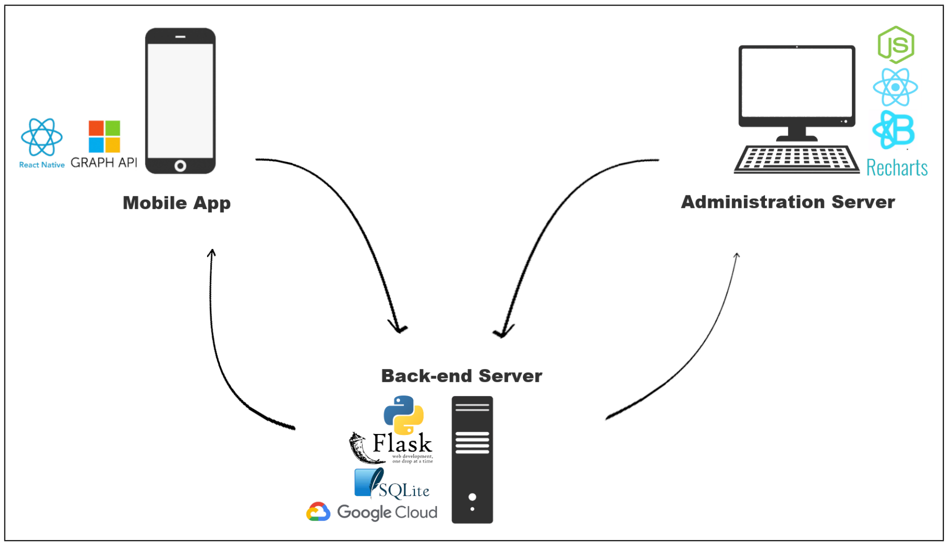


Figure 1: System architecture.

The application is divided into three parts.

A mobile application provides students with the means to register their attendance on campus and to check if any of these attendances have exposed them to COVID-19.

A back-end server maintains a database of students and student attendances, monitoring and updating the COVID-19 exposure status of both.

An administration server provides a view into the back-end server's database, as well as providing COVID-19 related statistics.

1.2 Administration Server

The administration server as a React application hosted on a Debian virtual machine on Google Cloud. It monitors the spread of COVID-19 at GMIT campuses by providing a view of the back-end server's database.

The application uses React router to handle navigation, which provides a navbar at the top of the screen.

1.2.1 Components

The application has a dashboard component and a database component.

Dashboard component

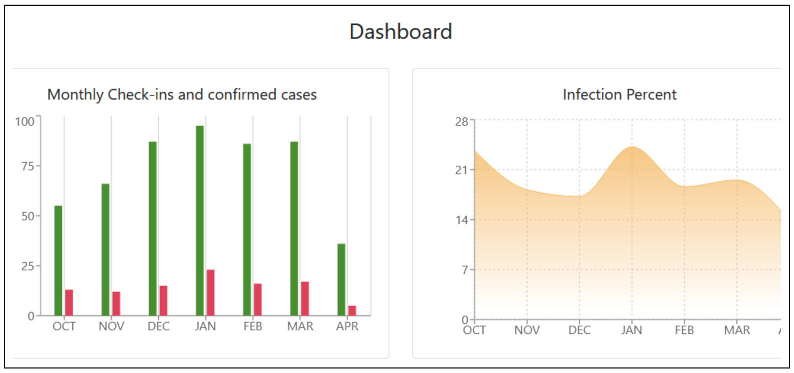


Figure 2: Dashboard component.

The dashboard component uses recharts to help monitor the analyse the spread of COVID-19 in GMIT.

Checks ins are separated by month, with non-exposed check ins coloured green and possible exposed check ins coloured red. The second graph shows the percentage of attendances which have been potentially exposed to COVID-19.

Database component
The Database component provides a view of the students in the database.

From this component the user can change the current COVID-19 status of a student or can check for more information, which will show each of attendance by the selected student, along with the date and the exposure status of that that token.

1.3 Mobile Application

The mobile app is a React Native. The application serves two purposes, firstly it provides students with a convenient means to register their attendance on GMIT campus. Secondly, the application will update students on the spread of COVID-19 through the Exposure Check view.

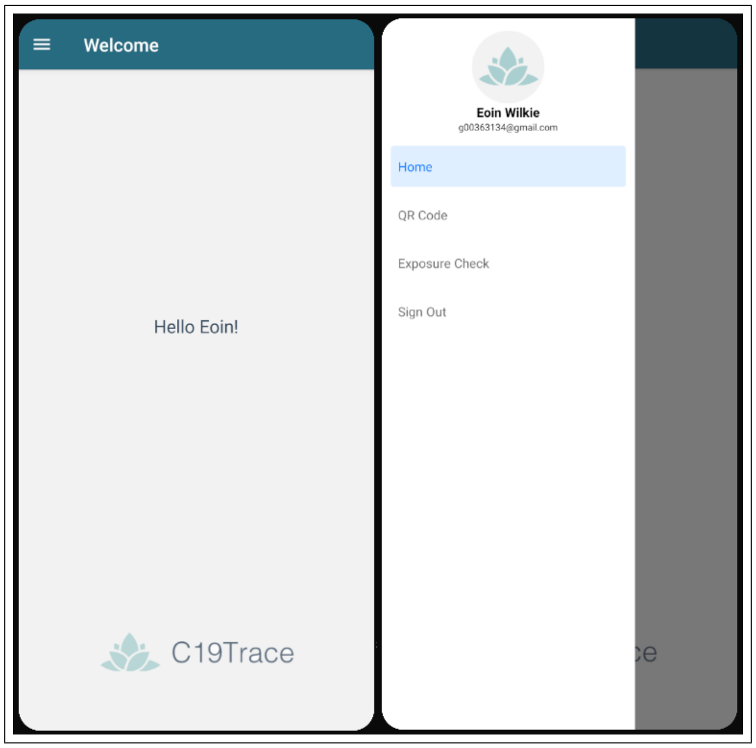


Figure 3: The mobile app's home screen and navigation drawer.

User authentication is handled by Microsoft's Azure AD, which allows the application to access Microsoft Graph, providing user details such as name and e-mail. This feature works for Microsoft accounts but not for GMIT accounts, to allow logins from GMIT accounts would require GMIT administration approval.

The application uses a navigation drawer to navigate between views. Drawer navigation is a common pattern which provides the user with a means of navigation between screens. The drawer can be accessed by swiping from the left side of the screen.

1.3.1 Views

The application contains five views, several of which are related to authentication. All views are styled with the application logo positioned at the bottom.

Home screen
The intention of this view is to greet the student after successful authentication. The student's name is displayed in the welcome message, which is derived from Microsoft Graph.

Sign in screen
This view prompts the student to sign in by displaying a sign in button. The Microsoft authentication process will begin once pressed.

Loading Screen
A simple screen which informs the student that their login is in progress.

QR Code screen
The QR Code screen fulfils one of the attendance registration functions of the mobile application.

This is done by touching a QR code which is displayed on screen. Once pressed a randomly generated token will be sent to the back-end server, along with the students ID.

Exposure check screen
The Exposure check screen informs the student of their exposure status.

This is done by requesting the list of the exposed tokens stored on the back-end server and comparing these against tokens stored on the device. If a match is found the student is informed of this exposure.

The student is given visual feedback of their status with the displaying of either a green checkmark, indicating that no exposure has been detected, or a red cross, indicating that an exposure has been detected.

1.4 Back-end Server

The back-end server is a Python3 application built with the Flask, a WSGI web framework. It is hosted on a Debian virtual machine on Google Cloud. The server can be run on Windows or Bash with the ./start_server.sh script. Activating the pip package manager is required before starting the server on Bash.

The back-end server is intended to resemble an internal GMIT database. It has a table containing a list of students and a second table which monitors student's attendances during this COVID-19 period.

Both student and attendance tokens have a COVID-19 status, a student with a positive COVID-19 status will create a positive attendance token and a student with a negative COVID-19 status will create a negative attendance token. Positive attendance tokens which are recorded will alter the COVID-19 status of all students in attendance from the registering students programme and year.

Test data is created by running the ./testdata.sh script. This script runs both the setup_student_testdata.py and python setup_token_testdata.py python files. The test data is randomly generated.

1.4.1 Table: Students

This table stores a list of students registered at GMIT, it stores the student ID, name, programme, and year of a student. A COVID-19 status is assigned to each student. This server provides several endpoints which enable the administration server to monitor students COVID-19 status.

Endpoints

- GET: /get-students
  Returns a list of all of the students in the student table.
- POST: /get-student-status
  Returns the COVID-19 status of a given student from the student table.
- POST: /update-student-status
  Updates the COVID-19 status of a given from the student table.

1.4.2 Table: Exposure tokens

This table stores a list of tokens which have registered attendance at the GMIT campus, it registers the student ID along with the date. A status is assigned to this token by checking the COVID-19 status of this student ID from the student database. This server provides several endpoints which enable the administration server to monitor attendances.

End points

- POST: /delete-token
  Deletes a token.
- POST: /update-status
  Updates the status of a token.
- GET: /get-tokens
  Returns all of tokens stored in the database.
- GET: /get-student-tokens
  Returns all of tokens generated by a given student.
- GET: /get-exposure-list
  Returns a list of all tokens that may have been exposed to COVID-19.
- GET: /get-monthly-checkins
  Returns a list of monthly check ins with a count of exposed and non-exposed tokens.
- POST: /submit-token
  Submits an encrypted token to the database, this token will be decrypted before being added to the database.

2 Methodology

The project was undertaking as part of a group for the first half of development and as a solo effort for the second half. Each of these periods had a different approach due to different circumstances.

2.1 Project planning

The initial planning phase was a brief period wherein the project was largely mapped out. The goals of this period were to outline what we wished to accomplish with this project and to understand how we would approach it by deciding which technologies we intended to work with.

After group formation we brainstormed ideas for our project. This involved discussing the technologies which we wished to work with, my group partner had Raspberry Pi which we were eager to include in our project and this shaped much of our brainstorming.

With our supervisor's input, we decided what our project would be. We created a project roadmap and decided that our goal was to have each component of our project working independently from the other components by the end of the first development stage and to work on project integration during the second development stage.

The foundations for successful project management and group work were set during this planning phase. We created a GitHub organisation with repositories for our project components and we a project on Overleaf which would allow us to begin documentation using Latex.

A weekly meeting scheduled with our project supervisor was established using Microsoft Teams and we had created a Discord server to maintain regular contact within the group. Discord offers an instant messaging service, along with servers which we had sectioned into categories for each of our components. This serves as record keeping for the discussions, we had on each of our development areas without being overwhelmed. Important messages could be pinned, and Discord can easily be searched for links, images, or words.

2.2 Research

Research took place alongside development and difficulties were empirically solved. Each area of the project required extensive research as the technologies used were new to our group.

With the benefit of hindsight, much of the problems encountered were not difficult to resolve as there is a wealth of information available. Often, the trouble is discovering what the actual issue is, this involves testing a variety of solutions. An effective solution might in fact be overlooked because of incorrect suppositions as to what the problem is.

2.3 Testing

The back-end database
The back-end server requires test data, which is created by running the ./test-data.sh script.

This script creates 100 students, 10 percent of which have a positive exposure status and 500 attendance tokens, 20 percent of which have a positive exposure status.

The mobile app
The mobile app was created and tested using the Android Studio integrated development environment.

2.4 Development stage 1: Grouped

The first stage of the project lasted from October to December and was undertook as a group. The goal of this period was to have each component of our project working independently from the other components by the end of this development stage.

Our approach to this development stage was to work independently on different parts of our project while maintaining regular communication on Discord We updated each other on our progress and collaborated on research. Most of my efforts during this period were focused on the React Native and the mobile application.

2.5 Development stage 2: Solo

Unfortunately, the project became a solo project after the first development stage. Initially we had decided that stage two of development would be concerned with integration and documentation. However, with the change of circumstances, the project had to be re-scoped as my teammate owned hardware which was to be used as part of the project. This piece of hardware was to serve as a terminal which integrated the components of our project and was a critical part of stage two development.

The effects of this departure are difficult to quantify. During the second development stage I had to have a micro-level understanding of the entire project, whereas previously we had a shared responsibility, and it was sufficient to have a macro-level understanding of the overall project with micro-level understanding of the parts which I had focused on.

The structure created during the first development stage still proved to be effective, although, the use of Discord became redundant as it is simpler to keep notes on text files as there was no group members to share these with.

Stage two of development did go according to the plan laid out at the start of stage one, but a readjustment period was required.

2.6 Contrasting the development stages

The first stage of development felt like an exploration into a variety technologies. We knew what we wanted to create but there was room for investigation and to refine our vision based on our discoveries.

The given timeframe was generous and there was a lot of time to work through the issues encountered.

In contrast to this, time was a constant concern in the second stage of development. Each week, progress was evaluated and goals for the upcoming week were defined. The approach resembled something of a one-person scrum based on a weekly sprint cycle.

3 Technical Review

3.1 System Architecture

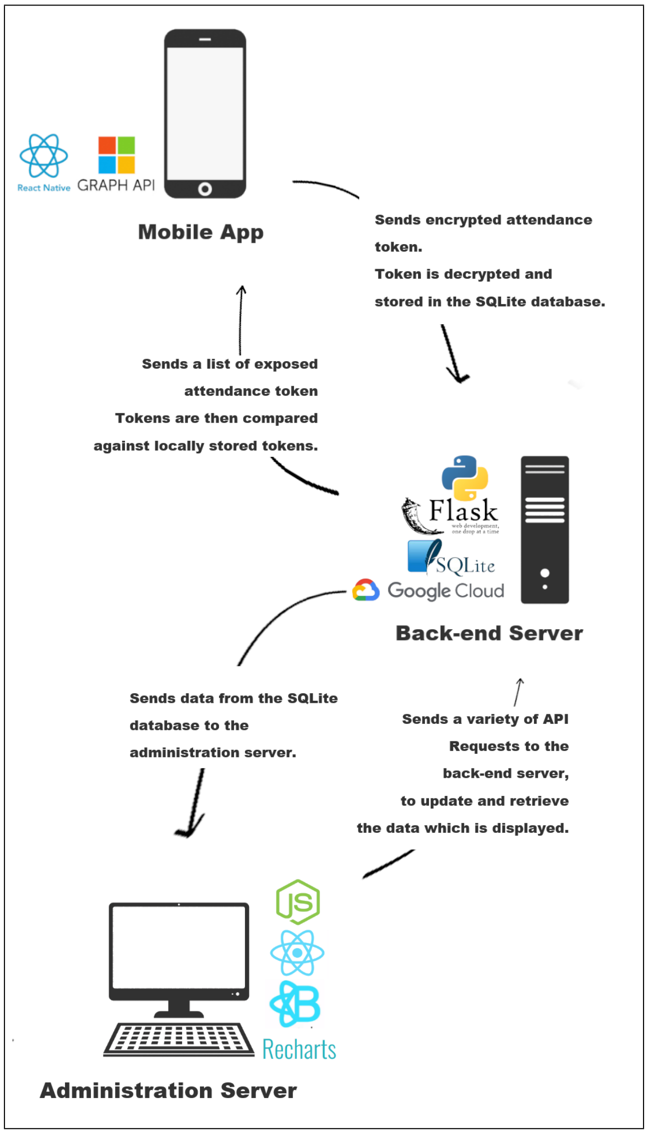

React Native
GRAPH API
Mobile App
Sends encrypted attendance token.
Token is decrypted and stored in the SQLite database.
Sends a list of exposed attendance token
Tokens are then compared against locally stored tokens.
Flask
SQLite
Google Cloud
Back-end Server
Sends data from the SQLite database to the administration server.
Sends a variety of API Requests to the back-end server, to update and retrieve the data which is displayed.
Recharts
Administration Server

Figure 4: System architecture

3.2 Administration server

The administration server is a React app built with the Node package manager(NPM)[6] . The application is run on a node server, and it is hosted on a Debian virtual machine on Google Cloud.

The administration server provides a means to monitor the spread of COVID-19 at GMIT campuses and provides visual feedback for administration purposes. The application has a view of the back-end database, showing the registered students and it can be expanded to view the tokens generated by each student.

3.2.1 Technology Selection

The decision to build the administration server with react was based on its popularity. Its advantages are outlined in Stackoverflow's 2020 developer survey [18]. JavaScript is the most commonly used programming language and React is the second most commonly used web framework, after JQuery[3].

Working with React[8] provided a great opportunity to gain experience with what is perhaps the most useful web framework today. The fact that could be paired with a React Native mobile application provided additional enticement to work with React, as this offered an opportunity to not simply learn the technology but to contrast similar technologies.

3.2.2 Navigation

The application uses React Router[9] to handle navigation, which provides a nav-bar at the top of the screen. React Router is a collection of navigational components that provide declarative routing for a React application. There is a route defined for each component.

3.2.3 Components

Components[11] are React elements which describe what should appear on the screen. The administration server provides two components, a dashboard, and a database.

Dashboard
The dashboard component provides statistical feedback based on GMIT's monthly attendances in the form of two charts.

A bar chart provides a view of monthly attendances, indicating the total attendances alongside the number of potentially infected attendances. An area chart displays the percentage of attendances which have been exposed each month.

The data is retrieved from the back-end server through the /get-monthly-checkins endpoint.

Database
The database component provides a view into the back-end database.

On loading the component, a list of students will be displayed. Two buttons are associated with each student. The first button will alter the COVID-19 status of the student and the second will display more information, displaying the attendance tokens associated with the selected student.

This component uses React-Bootstrap for styling.

3.2.4 Packages

React-Bootstrap
React-Bootstrap[10] replaces the Bootstrap JavaScript. Each component has been built from scratch as a true React component, without unneeded dependencies like jQuery. React-Bootstrap is used for component styling. The tables, buttons and containers use React-Bootstrap.

Previous design iterations utilized CSS for component styling, but the result was a dated aesthetic. React-Bootstrap offers modern feel to the components with a variety of colour and style options. The buttons use the ‘outline‘ design to achieve a minimalist feel.

React-Bootstrap provided a much cleaner outcome with much less effort than CSS.

Recharts

Recharts[16] is a composable charting library built on React components. It is used to data obtained from the back-end database as.

This application uses a bar chart to display the monthly attendances, comparing total attendances with exposed attendances and it uses an area chart to display the monthly percentage of attendances which have been exposed.

3.3 Mobile App

The mobile app is a React Native[12] app build with the Node package executor (NPX), and that uses the Microsoft Graph API for user authentication. The purpose of this app is to provide GMIT students with a means to register their college attendance and to notify them of potently exposures to help control the spread of COVID-19 within GMIT.

The user must sign in with a Microsoft account, although the user sign in could be modified to allow for only GMIT accounts. To do so would require a GMIT server administrator to implement.

3.3.1 Technology Selection

For mobile app development, React Native was an appealing as a cross platform framework choice. It provided an opportunity to work with JavaScript. Additionally, seeing as there is both a web app and a mobile app component this project, React and React Native provided an opportunity to explore and contrast these development tools.

By default, React Native uses JavaScript but this application uses Typescript due to the necessity of creating interfaces and storing states.

3.3.2 Navigation

Navigation is handled by react-navigation[15] package.

A navigation drawer to navigating between views, this is imported from @react-navigation/drawer. These views are Home QR Code and Exposure Check. Included in this navigation draw is a option to Sign Out.

3.3.3 Authentication

To support authentication with Azure AD, an Azure AD[1] native application had to be registered using the Azure Active Directory admin centre. This provided the application with an application ID which is used for authentication.

Authenticating the login with Azure allows the application to make calls to Microsoft Graph[5], giving access to data user data associated with the Microsoft account.

If correctly configured by GMIT administrators, Microsoft Graph should provide the application with the required data to link the application with GMIT internal databases, such as name, email.

It's possible but unknown if other relevant details such as year and programme can be accessed from Microsoft Graph. Student ID can, however, be inferred from the student email.

3.3.4 Views

In React Native, a view[14] is the most fundamental component. Two views in the mobile application are of particular interest, the QR code display view and the exposure check view.

QR Code view
The QR Code screen is a view which displays a pressable QR Code to the student. The QR code is a rendered component which creates the QR Code as an image using the react-native-qrcode-svg package. This view has a context called QRCodeScreenState which consists of a string variable called token which is generated when the state is loaded.

Token generation has following steps. 16 random base64 bit are generated and saved to the internal database, a message is constructed by concatenating the token to an e-mail address.

Encryption then takes place using the tweetnacl package. The output of this encryption is saved to QRCodeScreenState and is then passed to the QRCodeImage component when loaded, and then displayed to the user.

The QR Code is a TouchableOpacity, a wrapper which makes views respond to touch, allowing the encrypted data is sent to the exposure server when pressed.

It's worth noting that the QR Code screen was significantly altered due to the project re-scope. Initially this view would present a QR code which the terminal would scan. The terminal would read the QR code and send send the attendance to the back-end server. Making the QR Code touchable was a solution the the loss of the terminal.

Exposure check view
The ExposureCheck screen is a view which updates the students on their exposure status. Like the QRCode screen, the exposure check screen has a context named ExposureScreenState.

The state consists of three variables, loading, exposureFound and status. The states initial will be loading: true, exposureFound: false and status indicates that no exposure has been detected.

The view will display a graphic indicating that the application is loading while loading is true. When loading is false, the status is displayed alongside an accompanying image.

When the component mounts, a get request is sent to the exposure server which will retrieve a list of tokens which have been tagged as exposed. These tokens are then checked against the internal database for matching tokens. And if a match is detected the state will update the exposureFound and status values. Loading is set to false once the search has finished, which allows the status to render.

The view displays both an image and the status text. The image is selected from the Images component based on the value of exposureFound. The result is either a green checkmark when no exposure is detected, or a red X when n exposure is detected.

3.3.5 Packages

tweetnacl
TweetNaCl.js is a port of TweetNaCl[19] / NaCl to JavaScript. The hope was to use tweetnacl-util in conjunction with tweetnacl but the required functions did not work with React Native. Fortunately, the creator of tweetnacl-util has created @stablelib packages to address this limitation.

The @stablelib/utf8 package provides UTF-8 encoding and decoding, and the @stablelib/base64 package provides constant-time Base64 encoding and decoding.

Encrypting data sent from the React Native application is an important aspect of the project, student privacy must be protected. The Networking and Cryptography library is a popular and effective cryptography library which has been ported to many languages and as such, is available for use in the other areas of this project.

react-native-randombytes
Randombytes generates completely random bytes.

The react-native-simple-crypto library[13] was selected for usage during the project planning phase but like the tweetnacl-util package, it proved problematic to get working with React Native. This would have handled several of the required functions such as base64 encoding and decoding.

The react-native-simple-crypto package does fork the react-native-randombytes package, and because it was only the randombytes which was required, it made sense to use this package.

react-native-qrcode-svg
Qrcode-svg is a QR Code generator for React Native based on react-native-svg and javascript-qrcode.

The QR Code is a remnant of the original project design, initially the project would have a terminal which would read this QR Code, and this is how data would be transferred to the exposure notification server. While it was a vital element of the initial design, it now serves as a button which can be pressed. It does add to the aesthetics of the project but functionally, it does nothing.

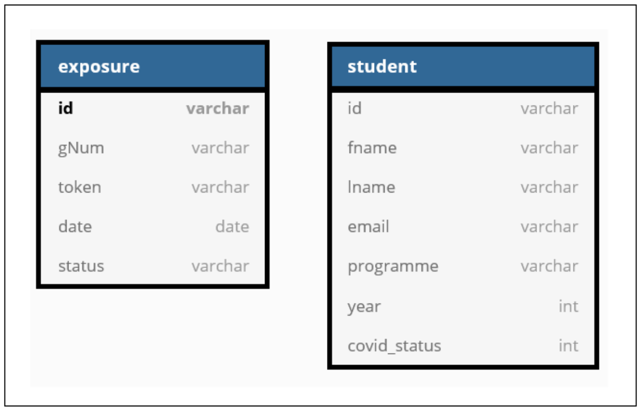
react-native-sqlite-storage
SQLite3[17] Native Plugin for React Native. It is an easy to use, lightweight SQL database.

SQLite3 is used for storing attendance tokens generated by the student in order to compare these against the exposure list. The student will be notified that they have been exposed to COVID-19 if a match is found.

3.4 Back-end Server

The back-end server is a Python3 application built with Flask[2], a WSGI web framework. It is hosted on a Debian virtual machine on Google Cloud.

The purpose of the back-end server is to allow API requests to the database tables. The database is created using the sqlite3 module. The database tables have the following schema.



exposure	
id	**varchar**
gNum	varchar
token	varchar
date	date
status	varchar

student	
id	varchar
fname	varchar
lname	varchar
email	varchar
programme	varchar
year	int
covid_status	int

Figure 5: Schema for both the exposure and the student tables.

3.4.1 Technology Selection

Flask, being a lightweight framework was appropriate for the back-end server as it is relatively small and Flask offers everything which was required, the ability to handle API requests and database access.

There was a desire to fit Python into our project as it is a versatile and flexible language, and the back-end servers provided a good opportunity to do so.

3.4.2 Initial design

Initially the back-end server was separated into two parts, the exposure notification server, and the student server. This required the use of the Requests for communication between the databases, Requests is a simple HTTP library for Python. When a student's attendance token was sent to the exposure server, the exposure server would request that students covid_status to apply this status to the token.

Later in development process these servers were merged, removing the need for the Requests library. This allowed for more fluid interaction between the database tables, as queries could be made without sending API requests and the tables could freely interact with one another. This allows for the status of multiple tokens to be updated when an exposed status is registered.

3.4.3 Libraries & modules

Flask-Cors
A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.

Enabling CORS was required for API requests.

PyNaCl
PyNaCl[7] is a Python binding to libsodium, which is a fork of the Networking and Cryptography library. Libsodium is a easy-to-use software library for encryption, decryption, signatures, password hashing and more.

Encrypting data sent from the React Native application is an important aspect of the project, student privacy must be protected. The Networking and Cryptography library is a popular and effective cryptography library which has been ported to many languages and as such, is available for use in the other areas of this project.

sqlite3
SQLite is a C library that provides a lightweight disk-based database that does not require a separate server process and implements most of the SQL standard. The sqlite3 module provides a SQL interface compliant with PEP 249, Python Database API Specification v2.0. SQLite does have a flaw, in that it does not support updating with join queries, but such tasks can be achieved by using sub-queries.

4 System Design

4.1 System Architecture

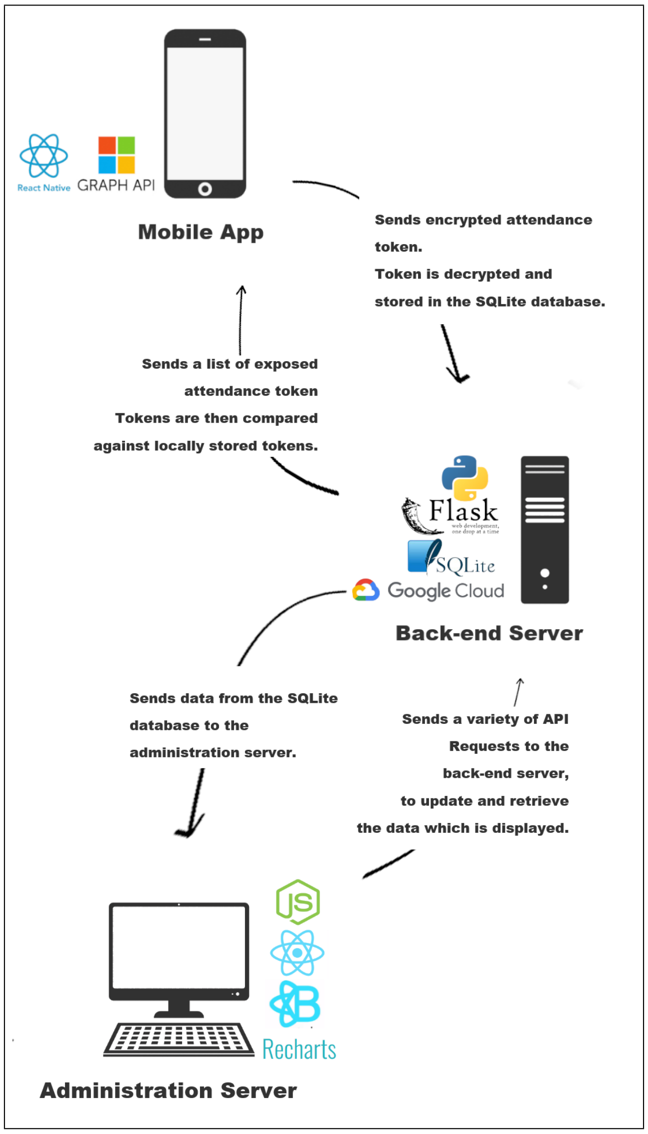

React Native
GRAPH API
Mobile App
Sends encrypted attendance token.
Token is decrypted and stored in the SQLite database.
Sends a list of exposed attendance token
Tokens are then compared against locally stored tokens.
Flask
SQLite
Google Cloud
Back-end Server
Sends data from the SQLite database to the administration server.
Sends a variety of API Requests to the back-end server, to update and retrieve the data which is displayed.
JS
Recharts
Administration Server

Figure 6: System architecture

4.2 Administration Server

The administration server provides a means to monitor the spread of COVID-19 at GMIT campuses. The application receives data from the back-end server's database through several different endpoints.

The data is then displayed on both the dashboard and the database components.

4.2.1 Dashboard

The dashboard retrieves the required data from the /get-monthly-checkins endpoint. The back-end server returns the month in numeric form, total attendances recorded that month and the infected attendance count in json format.

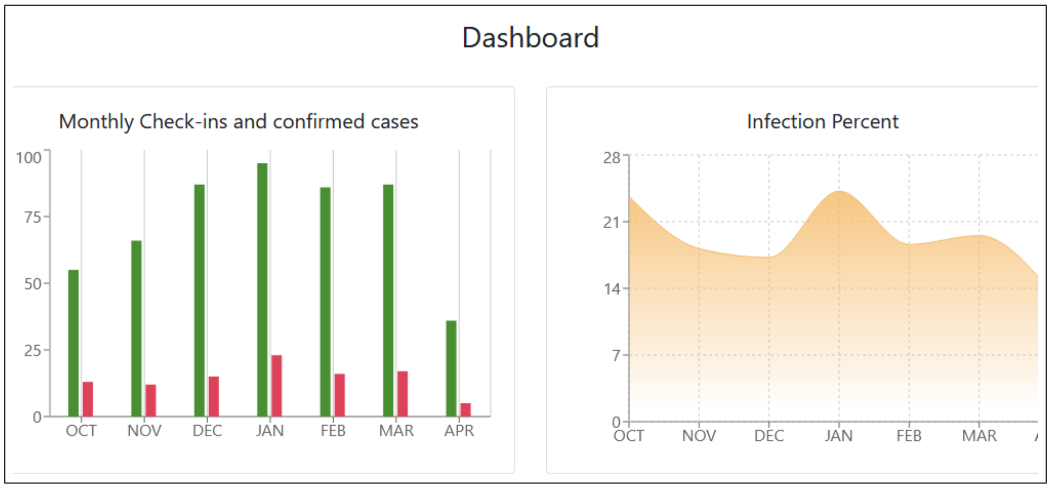


Figure 7: Dashboard component of the administration server.

The incoming data is then ordered so that it can be presented as a graph beginning in October and ending in May. The infection percent graph calculates the percentage of total attendances which has been exposed using this data.

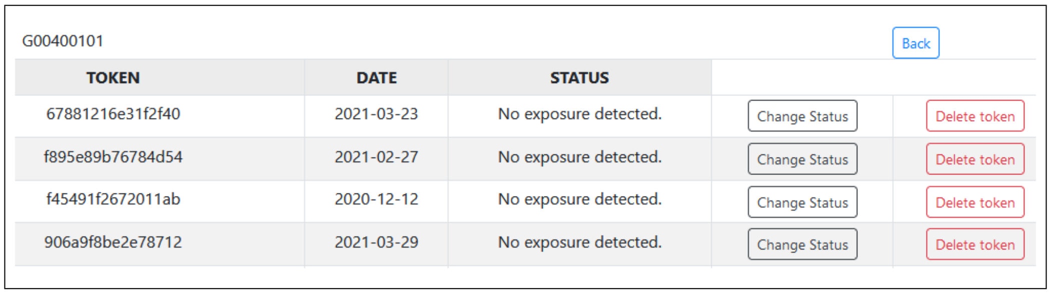
4.2.2 Database

The Database component renders a different page based on the components state. The componentDidMount method sends a request to the /get-students endpoint, this returns the entire student table.

Student table
The page will display an error if this request results in an error, it will inform the user that the page is loading while the request is being processed and it will display a list of all the students contained in the back-end server if the request has processed successfully. The back-end server returns this list of students in json format, and this is displayed as a table when the Database component is successfully rendered.

Each row of the table has a button labelled ‘Change Status‘ which will alter the COVID-19 status of the associated student. This is done by sending a POST request to the /update-student-status endpoint. Included in this POST request is the students ID and the desired status. The back-end server will read this the incoming data and update the status of the student.

Attendance token table
A second button labelled ‘info‘ will provide the user with a list of all attendance tokens and their COVID-19 status which have registered by the selected student. This is achieved with the help of local storage. When this button is clicked the student’s, ID is stored to local storage, and an additional boolean labelled studentSearch is stored to local storage.


G00400101

TOKEN	DATE	STATUS		
67881216e31f2f40	2021-03-23	No exposure detected.	Change Status	Delete token
f895e89b76784d54	2021-02-27	No exposure detected.	Change Status	Delete token
f45491f2672011ab	2020-12-12	No exposure detected.	Change Status	Delete token
906a9f8be2e78712	2021-03-29	No exposure detected.	Change Status	Delete token

Figure 8: Attendance token table from the administration server.

If this boolean is set to true the componentDidMount behaves differently, it will set studentSearch within the component state to true and render an attendance token table rather than a student table. The displayed tokens are obtained from the get-student-tokens endpoint. The students ID is sent to the back-end server and all of the tokens submitted by this student are returned in json format.

The token table also has two buttons. The first will change the status of the token by send the token id and desired status to the /update-token-status endpoint, this works similar to /update-student-status. The second button will delete a token by sending the token ID to the /delete-token endpoint.

4.3 Mobile App

The mobile app generates attendance tokens, which are sent to back-end server for storage. It also retrieves a list of exposed tokens which are used to inform the student if they have been may have been exposure whilst on GMIT campus.

4.3.1 Generating tokens

A token is generated by navigating to the QR Code page. A pressable QR code will be displayed on the application, when this is pressed the student's ID and token are send to the back-end server.

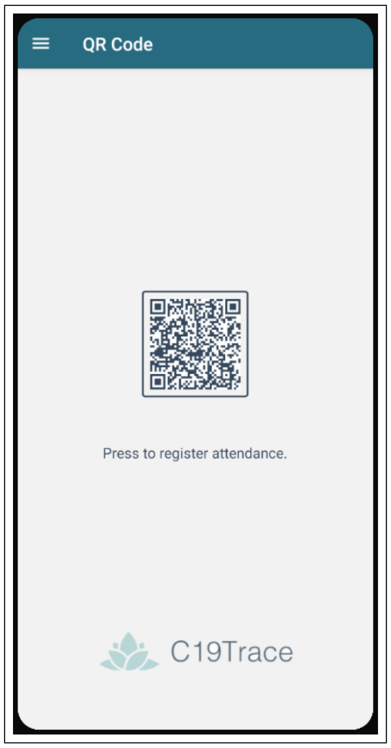


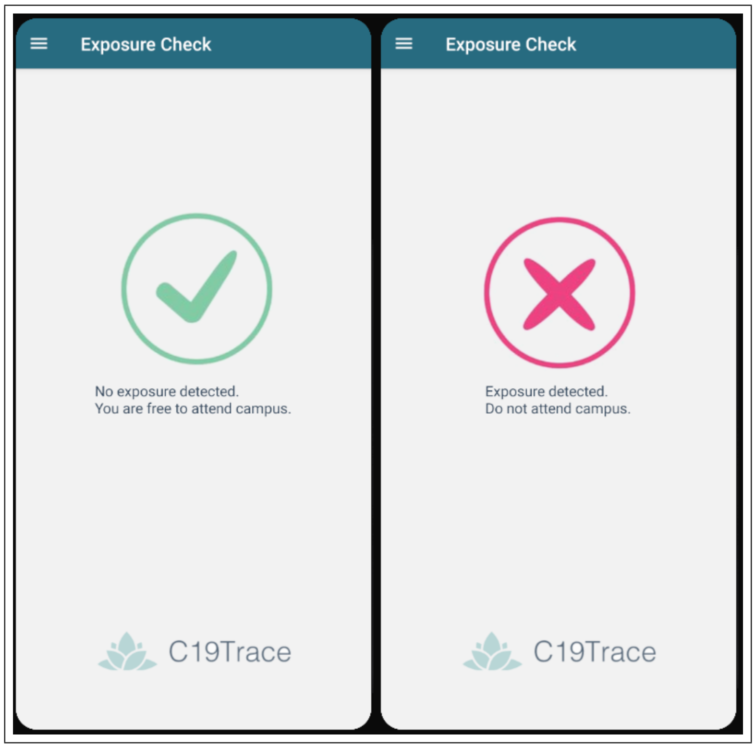
Figure 9: QRCode view of the mobile app.

On loading the QR Code view the token is generated and assigned to the view's state by calling the generateQRCode function. This function has several steps.

First it generates a token using nacl and randombytes and then saves this token to the internal database, the generated token is 16 bytes. Next, the token is encrypted using nacl's secretbox function. And lastly, the encrypted data is saved to the view's state in json format.

The pressable QR Code calls the storeToken function when pressed. This function will post the encrypted token saved to the view's state to the /submit-token endpoint.

4.3.2 Retrieving exposure list



No exposure detected.
You are free to attend campus.

Exposure detected.
Do not attend campus.

Figure 10: Exposure check view of the mobile app.

On loading the Exposure Check view a GET request is sent to the /get-exposure-list endpoint. A list of exposed tokens is returned, and these tokens are then compared against tokens stored in the local database. If there is a match, the status is set to exposed. Otherwise, the status is set to not exposed. Afterwards, the Exposure Check view will display the state status with an accompanying image.

4.4 Back-end Server

The back-end server is intended to resemble an internal GMIT database. It has a table containing a list of students and a second table which monitors student's attendances during this COVID-19 period.

The administration server and mobile application will both send to the back end server, and receive data from it. The back-end server contains a variety of endpoints and methods for this purpose.

4.4.1 Methods

decryptMsg
When an attendance token is sent to the back-end server, the received message is decrypted with nacl's secretbox using a hard-coded key. This involves splitting the message at the ':' character to separate the nonce from the message and then using the nonce and key to decrypt the message. The decrypted message will later be added to the token table with the students ID.

add_token
The add_token method accepts a student ID and a token. The first step in the add_token method is obtaining the registering students COVID-19 status and applying this to the token. The token is then added to the token table.

Next, other students and tokens must be updated if the token has an exposed status. Any students from the same year and programme who are in attendance on that day will have both their student status and token status from that day set to exposed. This is achieved with the use of a sub-query.

4.4.2 API calls

/delete-token
Accepts a token ID and removes the associated token from the token table.

/update-status
Accepts a token ID and a new status and updates the status from the associated token in the token table.

/get-tokens
Returns a list of all tokens stored in the token table.

/get-student-tokens
Accepts a student ID and returns all of tokens associated with that student.

/get-exposure-list
Returns a list of all tokens that may have been exposed to COVID-19.

/get-monthly-checkins
Returns a list of check ins for each month, expressed in numeric format, with a count of total attendance tokens as well as exposed attendance tokens.

/submit-token
Accepts a encrypted token, decrypts the token and adds it to the token table. This end point will utilize the decryptMsg and add_token methods.

5 System Evaluation

5.1 Evaluation of project goals

The goal was to develop a system which could successfully be implemented within GMIT, in that regard the project has not been a success, but it is close reaching these goals. In order to successfully meet this goal, the mobile app would need to be given the required permissions by GMIT administrators. The outlined goals have largely been accomplished, albeit with some limitations.

5.2 System Limitations And opportunities

5.2.1 Administration Server

R0 calculation
Attempts to add R0 [4] calculation unsuccessful.

R0 describes how many cases of a disease an infected person will go on to cause. The hope was to display R0 calculations on the administration server's dashboard.

There were difficulties in understanding how this is calculated.

5.2.2 Mobile App

MS Graph
The user e-mail is not derived from Microsoft Graph.

This information is obtainable, as proven by obtain and display a variety of user information from Microsoft Graph, including the e-mail.

The issue faced is that so far, this data has proven to be obtainable only within a view. This means that with how the React Native application is constructed, the e-mail address is not available for use when the attendance registration takes place. A re-planning of how views are constructed may be required for this to work correctly.

NaCl
The encryption key is hard-coded and should be coming from the back-end server. Can be resolved with a get request which will retrieve this key from the exposure server.

This was decided to be a low priority area for development.

Mobile app database
Data in the SQL database is not being deleted after a specified period. This could be resolved by deleting data which has been saved greater than two weeks prior, when searching for exposures.

The token is also being saved at the incorrect time. Rather than being saved when the user presses the QR Code, it is saved when it is generated. This can potentially be resolved by adding a encrypted state to the QR Code, which takes a string argument. The token can be created and saved to the token state, and afterwards the encryption can occur based on this token. However, a complete re-structuring of the QR Code view may be necessary.

5.2.3 Back-end Server

With the exposure and student databases being merged, there could be more intelligent interaction with the databases. For example, the mobile app could send the students ID and retrieve only tokens associated with that ID.

NaCl
Data being sent to the administration server is not being encrypted.

Google Cloud
The server is not running automatically on Google Cloud when turned on, it must be activated from the console.

Getting a start-up script to work has proven problematic.

Changes to token status
While students who are in attendance from the same class and year are updated when an exposed token is added to the database, tokens from previous dates which have their status changed will not update other students in a similar fashion.

6 Conclusion

The objective of this project was to create a system capable of monitoring the spread of COVID-19 at GMIT campus. To do so required means for student attendance registration alongside the ability to store and access this attendance data.

The most convenient way for attendances to be submitted is through the student's mobile phone and a mobile application was created for this purpose.

The initial goal was to tie this registration to the GMIT campus with a terminal, but due to unforeseen events this terminal could not be developed. The student registers their attendance by pressing a button rather than scanning their phone at a terminal. These attendance tokens are encrypted before being sent with the student ID to the back-end server.

The mobile application design requires that a student can sign in with their student account, this was another problematic feature to implement as this requires GMIT administrators to grant the application login permissions. Because GMIT authentication was not achievable, the mobile application allows for Microsoft authentication.

Attendance tokens are saved to a database once received by the back-end server. The back-end server provides several endpoints, enabling access to data of both students and attendances.

The administration server provides tables and charts based on data retrieved from the back-end server.

6.1 Future Development

There are a few opportunities for future development, the most obvious of which is general attendance tracking at GMIT.

Data integrity would be greatly increased with the addition of a terminal.

6.2 Reflection

This is the first project of this scale which I have had the opportunity to undertake during my studies at GMIT. It has been a great opportunity to learn first-hand about project management and to experience new technologies in a different learning environment than which I have been accustomed to.

Having my teammate depart the project midway through disrupted the project goals and it had to be re-scoped as the original project specification became impossible. This was an interesting period because it is defeating. The value of being able to relay on others to approach the goal with urgency is highlighted, as is the value of communication and idea sharing. While re-scoping the project was a challenge, it was this communication and idea sharing which I missed as its always interesting to see how others approach problems.

As for my approach to the project, in hindsight I find it ineffective. Due to time pressures, I felt as if I had to constantly have some new addition to the project and I did not spend the time I should have intricately planning the course of the project when I had returned for the second stage of development. I believe I would have been able to keep a greater overview and understanding of what I needed to accomplish if I had.

References

[1] *Azure-AD*. `https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis`. Accessed: 08-05-2021.

[2] *Flask*. `https://flask.palletsprojects.com/en/1.1.x/`. Accessed: 08-05-2021.

[3] *JQuery*. `https://jquery.com/`. Accessed: 08-05-2021.

[4] *Macmillan dictionary, R0*. `https://www.macmillandictionary.com/dictionary/british/r0`. Accessed: 08-05-2021.

[5] *Microsoft Graph*. `https://docs.microsoft.com/en-us/graph/overview`. Accessed: 08-05-2021.

[6] *NPM: Node package manager*. `https://www.npmjs.com/`. Accessed: 08-05-2021.

[7] *PyNaCl*. `https://pypi.org/project/PyNaCl/`. Accessed: 08-05-2021.

[8] *React*. `https://reactjs.org/`. Accessed: 08-05-2021.

[9] *React Router*. `https://reactrouter.com/`. Accessed: 08-05-2021.

[10] *React-Bootstrap*. `https://react-bootstrap.github.io/`. Accessed: 08-05-2021.

[11] *React-Components*. `https://www.w3schools.com/react/react_components.asp`. Accessed: 08-05-2021.

[12] *React-Native*. `https://reactnative.dev/`. Accessed: 08-05-2021.

[13] *React-Native-Simple-Crypto*. `https://www.npmjs.com/package/react-native-simple-crypto`. Accessed: 08-05-2021.

[14] *React-Native-View*. `https://reactnative.dev/docs/view`. Accessed: 08-05-2021.

[15] *React-Navigation*. `https://reactnavigation.org/`. Accessed: 08-05-2021.

[16] *Recharts*. `https://recharts.org/`. Accessed: 08-05-2021.

[17] *SQLite*. `https://www.sqlite.org/index.html`. Accessed: 08-05-2021.

[18] *StackOverflow survey*. `https://insights.stackoverflow.com/survey/2020`. Accessed: 08-05-2021.

[19] *Tweetnacl*. `https://tweetnacl.cr.yp.to/`. Accessed: 08-05-2021.