
# SENG3011 21T1

## Final Report

**Group 200OK**

| | |
|---|---|
| Abhyudit Gupta | z5196145 |
| Jiahui Luo (Lacey) | z5158415 |
| Haoran Xu (Matthew) | z5134675 |
| Yueru Duan (Ellen) | z5210986 |
| Ayaan Adil | z5213315 |

# Table of Contents

# Application Programming Interface (API)

## Introduction

The API built by our team scrapes news about outbreaks in a specific location, from the Centre for Disease Control's website and presents this information in the form of concise and informative disease reports.

To get these disease reports a user has to make calls to an API endpoint. In these calls, the user can mention keywords, which can be used to filter the disease reports.

These disease reports clearly indicate the date of publication, the main text, the diseases, corresponding symptoms and locations mentioned in an article related to an outbreak.

Additionally, the user can also view the logs of all calls made to the API.

## Purpose

The purpose/reasons for building this API was/were -

1) To learn about outbreak and epidemic detection using public data sources
2) To understand the detailed functioning of a REST API
3) To use it in SENG3011 D2 and in the web app mentioned later

## Intended use

The intended use of this API includes but is not limited to -

1) To easily enable a client to find and access all the disease reports related to a given search term including disease and location for a given period of time.
2) To access logs of previous calls to the API.

## Intended Audience

The intended audience of the API includes but is not limited to -

1) Team 200OK
2) Other SENG3011 teams
3) People looking for information on disease outbreaks around the world

All of these groups would be referred to as “average users” of the API in further sections of the report.

## Assumptions and Dependencies

1) The user has access to an internet connection and a browser to make calls to an API.
2) The user is situated in a location where websites hosted on GCP are not restricted.

## Requirements Classification

The following outline the requirements of the API and Destinated. Each requirement is prioritised according to the MoSCoW convention.
[Priority 1] Must Have: These outline the essential requirements of the system.
[Priority 2] Should Have: These are requirements that extend the basic functionality of the system.
[Priority 3] Could Have: Requirements that improve the user experience or customer satisfaction.
[Priority 4] Won’t Have: Requirements that are extensive to the core system and not delivered for our time frame.

## Requirements

Requirement 1: The client is able to find and access all the disease reports related to given search terms including a comma-separated list of keywords (optional), location(optional) for a given period of time (start time, end time) in JSON format. [Priority 1]

Requirement 2: The API is to be documented on Swagger. There should be a description of all the different calls a user can make to the API along with sample responses. The user should be able to try out the API on swagger. [Priority 1]

Requirement 3: The API should follow REST design principles. [Priority 1]

Requirement 4: The client is able to access logs of all calls made to the API in JSON format. Details included in each log are query parameters of the API call, data source, end point name, number of reports found, report's database key, API access time, response status code and execution time. [Priority 2]

Requirement 5: The API also has a log file in the backend that tracks resource utilisation for monitoring API performance. [Priority 3]

## User Cases of the API

| | |
|---|---|
| User Case 1 | Search for disease outbreak reports |
| Story | As a user, I want to view disease outbreak reports so that I can gather information on outbreaks for research. |
| Relevant Requirements | Requirement 1,2,3 |
| Actors | Average user |
| Basic Flow | **When** the user provides start date and valid end date in YYYY-MM-DDThh:mm:ss format (xx can be used as placeholders but year must be provided), a location (optional) and a comma separated list of key terms that the user wants to search for (optional);<br>**Then** disease reports are filtered according to the period (start date + end date) entered by the user and then the key word and locations, and disease reports satisfying the searching criteria are returned in JSON format. |
| Alternative Flow | **When** the user provides either empty start date and/or end date, or invalid end date (end date before start date);<br>**Then** a response with status code of 400 is returned, informing the user that valid start date and end date are required and need to be in YYYY-MM-DDThh:mm:ss format. |

| | |
|---|---|
| User Case 2 | View API logs |
| Story | As a user, I want to view logs of the API so that I can view how other users have used the API and hence tailor the calls that I make |

| | |
|---|---|
| Relevant Requirements | Requirement 2,3,4 |
| Actors | Average user |
| Basic Flow | **When** the user provides the number of logs they want to view (optional; default 10 logs); **Then** the API returns the provided number of logs (default 10 logs). Details included in each log are query parameters of the API call, data source, end point name, number of reports found, report's database key, API access time, response status code and execution time. |
| Alternative Flow | **When** the user provides an invalid input for the number of logs they want to view, **Then** a response with a status code of 400 is returned, informing the user that they need to enter a valid number. |

# Destinated - Web App

## Introduction

Destinated is the name of the web app built by our team 200OK. Destinated provides it's users with all the information they need to travel safely and makes their experience at the airport fluid. Destinated provides the user with information on any travel restrictions or local disease outbreaks in the place that they are travelling to. It also recommends vaccines based on the user's travel destination and allows them to store proof of their vaccination records which can be shown at airports, ports or railway stations during travel.

## Purpose

The purpose/reasons for building this web-app was/were -

1) To use it in SENG3011 Deliverable 3 and 4
2) To learn how to make a web-app using react-js
3) To learn about cloud services like Google Cloud Platform
4) To understand engineering challenges in the COVID-19 era

## Intended use

The intended use of this web-app includes -

1) A client can view disease outbreaks in the location they are travelling to.
2) A client can check vaccines they need to get before travelling to a specified location.
3) A client can check travel restrictions associated with the specified destination
4) A client can upload their vaccination records with visual proof to the app, eliminating the need to carry vaccination certificates while they are travelling.
5) An authorised official can view vaccination records of passengers on a particular flight.

## Intended Audience

The intended use of this web-app includes but is not limited to -

1) Travellers around the world
2) Airport/port/train-station officials

These 2 groups will be referred to as “average user” and “admin user” respectively in further sections of the report.

## Assumptions and Dependencies

- Since several APIs have been used, one of the main assumptions is they are all updated and work well, providing correct information.
- Since we haven't developed verification mechanisms, we need to assume that the users provide valid vaccination certification.

## Requirements

1) An average user can register and login to use our app. [Priority 1]
2) An admin user can register and login to use our app. [Priority 1]
3) An average user can view the recommended vaccinations and CDC advice for almost all countries around the globe. [Priority 1]
4) An average user can view disease outbreaks in a specified location. [Priority 1]
5) An average user can check COVID travel restrictions for a specific location. [Priority 1]
6) An average user can upload their vaccination records with visual proof to the web-app and view previously uploaded vaccination history. [Priority 1]
7) An average user can search and register for flights to one destination to another, on a specific date or month of year. [Priority 1]
8) An admin user can check the recommended vaccines and vaccination records of all registered average users on a specific flight. [Priority 1]

## User Cases

| | |
|---|---|
| User Case 1 | Register and login to website |
| Story | As an average user, I want to be able to register and login to the app so that my data is safe. |
| Relevant Requirements | 1 |
| Actors | An average user |
| Basic Flow | Given an average user is on the home page;<br>When a average user clicks the register button;<br>Then the average user is directed to the |

| | |
|---|---|
| | registration page where they can enter valid details and create their account, after which they will be directed to the home page where they can login with the same details |

| | |
|---|---|
| Use Case 2 | View vaccination recommendations/requirements of a country |
| Story | As an average user, I want to see what vaccinations I will need before I can travel to a country so that I can get these vaccinations before departure. |
| Relevant Requirements | 1,3 |
| Actors | An average user |
| Basic Flow | **Given** the average user has logged in and is on the homepage;<br>**When** the average user clicks on 'Vaccines' link in the Navbar;<br>**Then** the average user is directed to a dedicated page where they can enter the country they want to travel to;<br>**When** the average user enters a country name, selects it from the dropdown list and clicks the 'Search' button;<br>**Then** the average user is directed to a new page where all vaccination recommendations/requirements along with CDC recommendations and brands of each vaccine are shown in a table. |

| | |
|---|---|
| User Case 3 | View disease outbreak reports of a country |
| Story | As an average user, I want to see disease outbreak reports of a country so that ... |
| Relevant Requirements | 1,4 |
| Actors | An average user |
| Basic Flow | Given an average user is on the page where they can view all vaccination recommendations/requirements of a country; |

| | |
|---|---|
| | When a average user clicks the ‘Outbreak News’ button;<br>Then the average user is directed to a new page where they can view recent disease outbreaks reports of the previously selected country scraped from the CDC website. |

| | |
|---|---|
| User Case 4 | View entry restrictions of a country |
| Story | As an average user, I want to know if I am allowed to enter a country so that ... |
| Relevant Requirements | 1,5 |
| Actors | An average user |
| Basic Flow | Given a average user is on the page where they can view all vaccination recommendations/requirements of a country;<br>When a average user clicks the ‘Restriction’ button;<br>Then the average user is directed to a new page where they can view travel restrictions of the previously selected country. |

| | |
|---|---|
| User Case 5 | View vaccination history |
| Story | As an average user, I want to view my vaccination history so that I know what vaccinations I have already taken. |
| Relevant Requirements | 1,6 |
| Actors | An average user |
| Basic Flow | **Given** the average user has logged in;<br>**When** the average user clicks the ‘Vaccination History’ link in the Navbar;<br>**Then** the average user is directed to a new page where they can see the disease name, vaccine name, vaccination date and they can also view the proof document they have uploaded. |

| User Case 6 | Log new vaccination |
|---|---|
| Story | As an average user, I want to log a new vaccination I have taken so that I can prove that I have taken this vaccination later on. |
| Relevant Requirements | 1,6 |
| Actors | An average user |
| Basic Flow | **Given** an average user has logged in and clicked the ‘Vaccination History’ button in the Navbar;<br>**When** the average user enters the disease name, vaccine name, date when the vaccination was taken (the date must be before today), uploads a proof document in pdf/png/jpg/jpeg format and clicks the ‘Submit Record’ button;<br>**Then** a modal window pops up, informing the average user that a new vaccination has been logged successfully. |
| Alternative Flow | **Given** an average user has logged in and clicked the ‘Vaccination History’ button in the Navbar;<br>**When** the average user either leaves at least one the fields empty, enter an invalid date (either today or dates after today) or upload a file in unsupported formats and clicks the ‘Submit Record’ button;<br>**Then** a modal window pops up, informing the average user that some fields were missing or invalid inputs were given. |

| User Case 7 | Search for flights |
|---|---|
| Story | As an average user, I want to see all flights from a given origin to a given destination on a specific date so that ... |
| Relevant Requirements | 1,7 |
| Actors | An average user |
| Basic Flow | **Given** an average user has logged in;<br>**When** the average user clicks the ‘Register |

| | |
|---|---|
| | Flights' link in the Navbar;<br>**Then** the average user is directed to a new dedicated page for flight registration;<br>**When** the average user enters origin, destination, selects origin and destination airports from dropdown lists, selects the date of departure (the date must not be before today) and clicks the 'Search' button;<br>**Then** available directed flights and transit flights from the origin airport to the destination airport on the specific date of departure are displayed. |
| Alternative Flow | **Given** an average user is on the new dedicated page for flight registration;<br>**When** the average user either leaves at least one fields empty, enters invalid origin/destination airport (origin/destination airports that are not from the dropdown list), or enters an invalid date of departure (dates before today) and clicks the 'Search' button;<br>**Then** a modal window pops up, informing the average user that some fields were missing or invalid inputs were given. |

| User Case 8 | Flight Registration |
|---|---|
| Story | As an average user, I want to register for a flight so that ... |
| Relevant Requirements | 1,7 |
| Actors | An average user |
| Basic Flow | **Given** an average user has logged in and is viewing all direct/transit flights;<br>**When** the average user clicks the 'Register' button next to a flight;<br>**Then** a modal window pops up, informing the average user that registration was successful. |

| User Case 9 | View registered flights |
|---|---|
| Story | As an admin user, I want to view all flights that average users have registered for. |

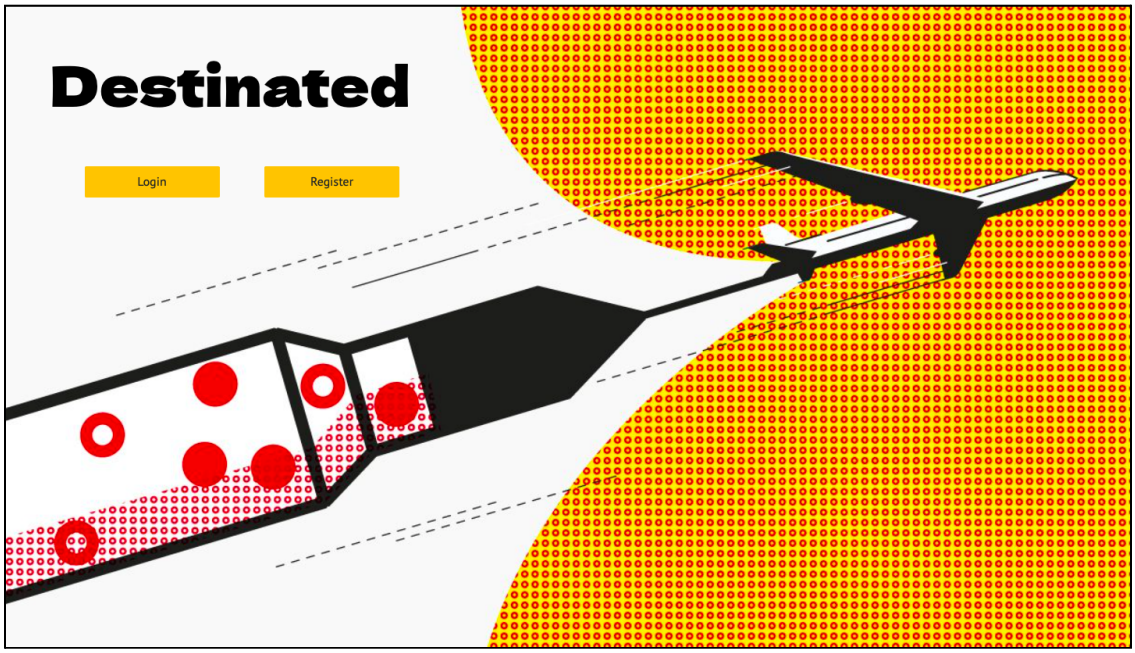
| | |
|---|---|
| Relevant Requirements | 1,8 |
| Actors | An admin user |
| Basic Flow | **Given** an admin user has logged in;<br>**When** the admin user clicks the 'Flight Details' button in the Navbar;<br>**Then** the admin user is directed to a new dedicated page;<br>**When** the admin user selects a date;<br>**Then** all registered flights that depart on the selected date are displayed automatically. |

| | |
|---|---|
| User Case 10 | View passenger details |
| Story | As an admin user, I want to view passenger details of a registered flight so that ... |
| Relevant Requirements | 1,8 |
| Actors | An admin user |
| Basic Flow | **Given** an admin user has logged in and is viewing registered flights on a specific date;<br>**When** the admin user double clicks a registered flight;<br>**Then** the admin user is directed to a new dedicated page where passengers' name and their vaccination history are displayed. |

## Screenshots of Final Web-Application, Destinated

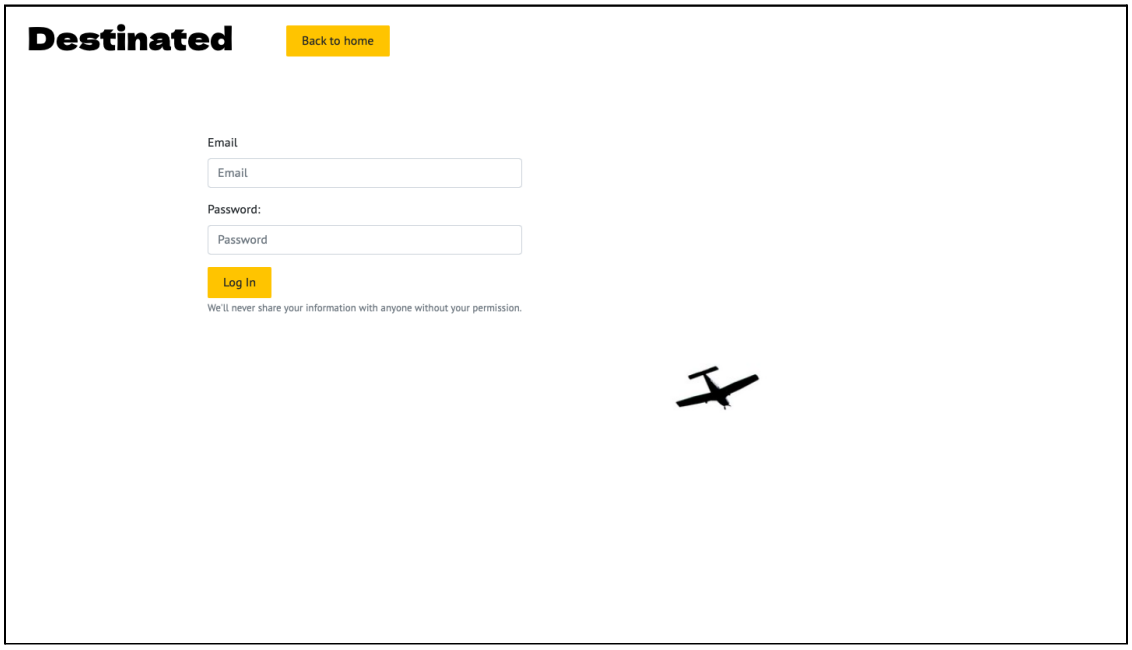
This section shows the final design of our web application.

## Landing Page

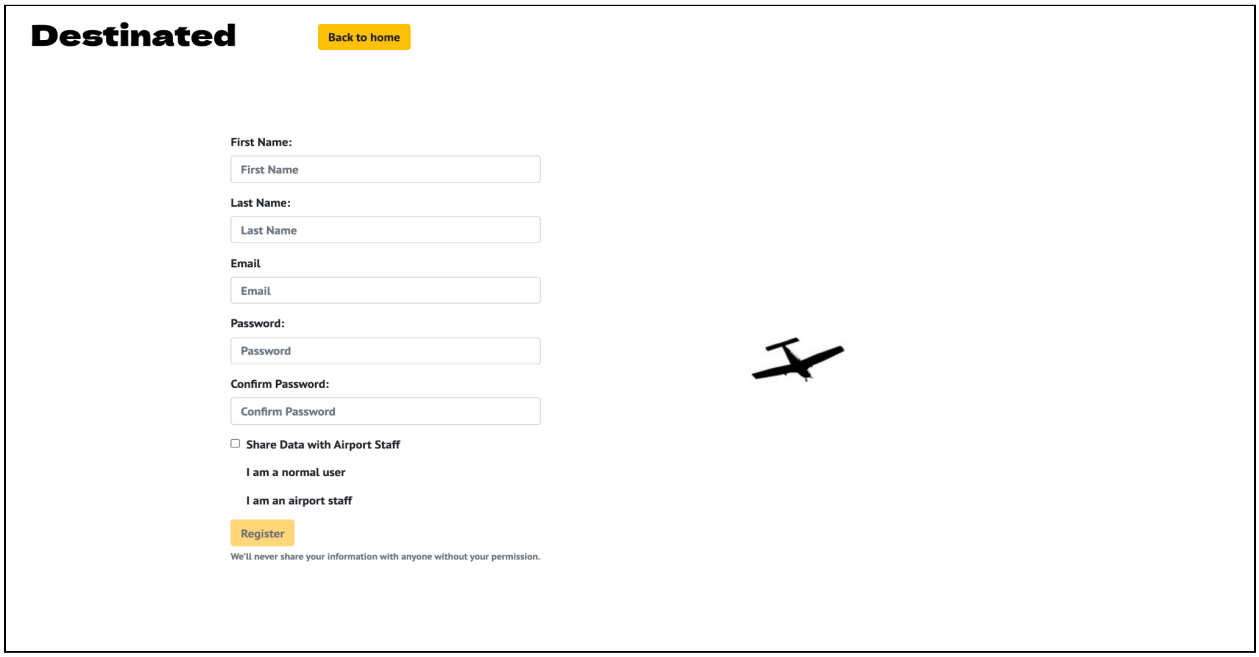



The above is the homepage, this is where both average user and admin user first land. Our web app is hosted on google cloud hosting services. (link to part on hosting)

## Login



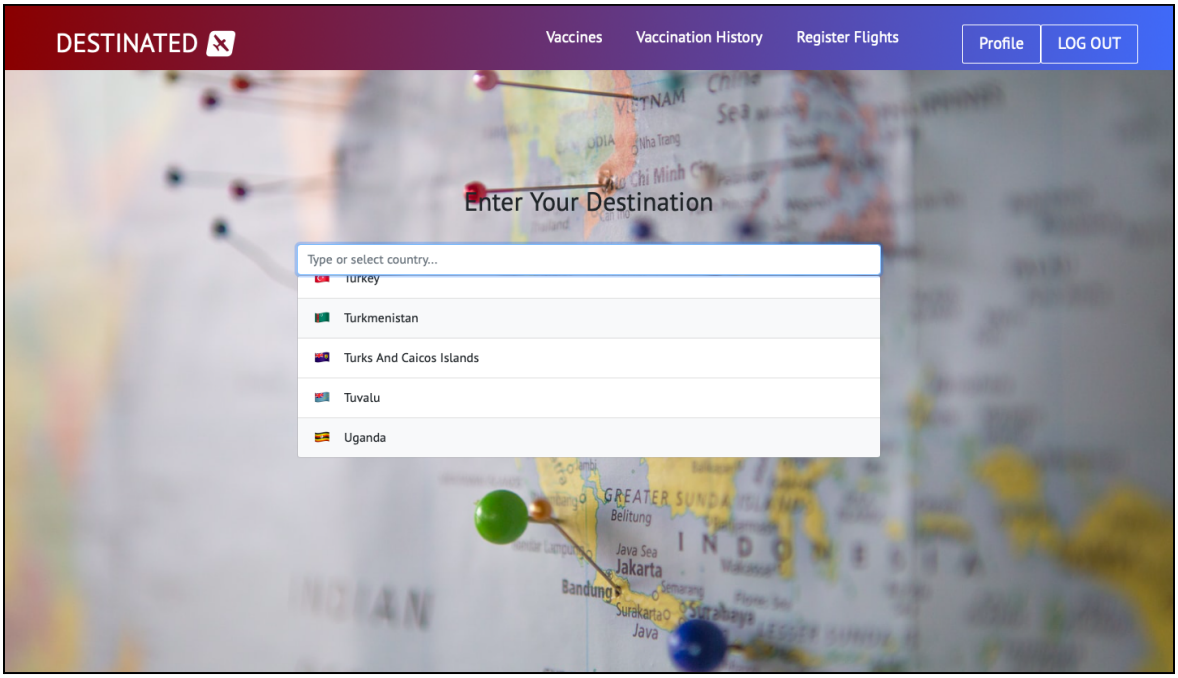
## Registration



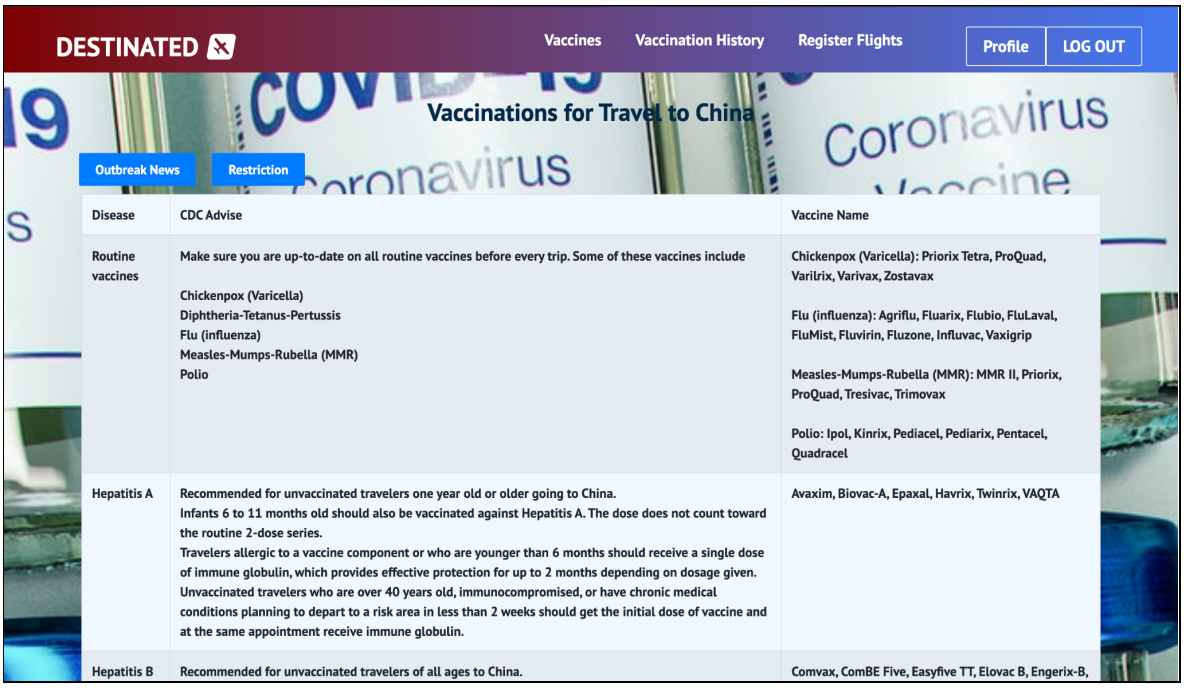

The above 2 images show the login and registration page of our web app from where the average/admin user can register or login. For authentication we have used Google's authentication services. It is necessary for an average user to authenticate to use our app, as it aims to store vaccination records for an average user which is specific to each user and is also sensitive data. (link to part on google auth). In addition, it is also critical for an admin user to authenticate to use our app since admin users will access average users' personal information.

# Average User

## Destination Selection Page

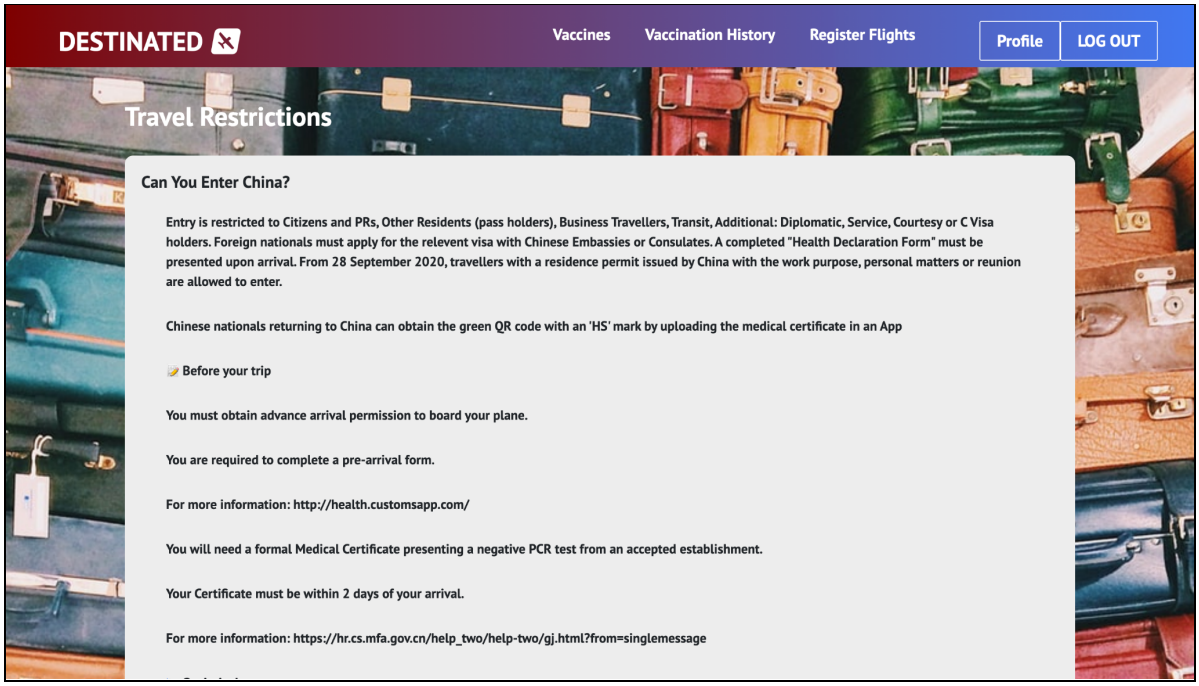



## Vaccines Page

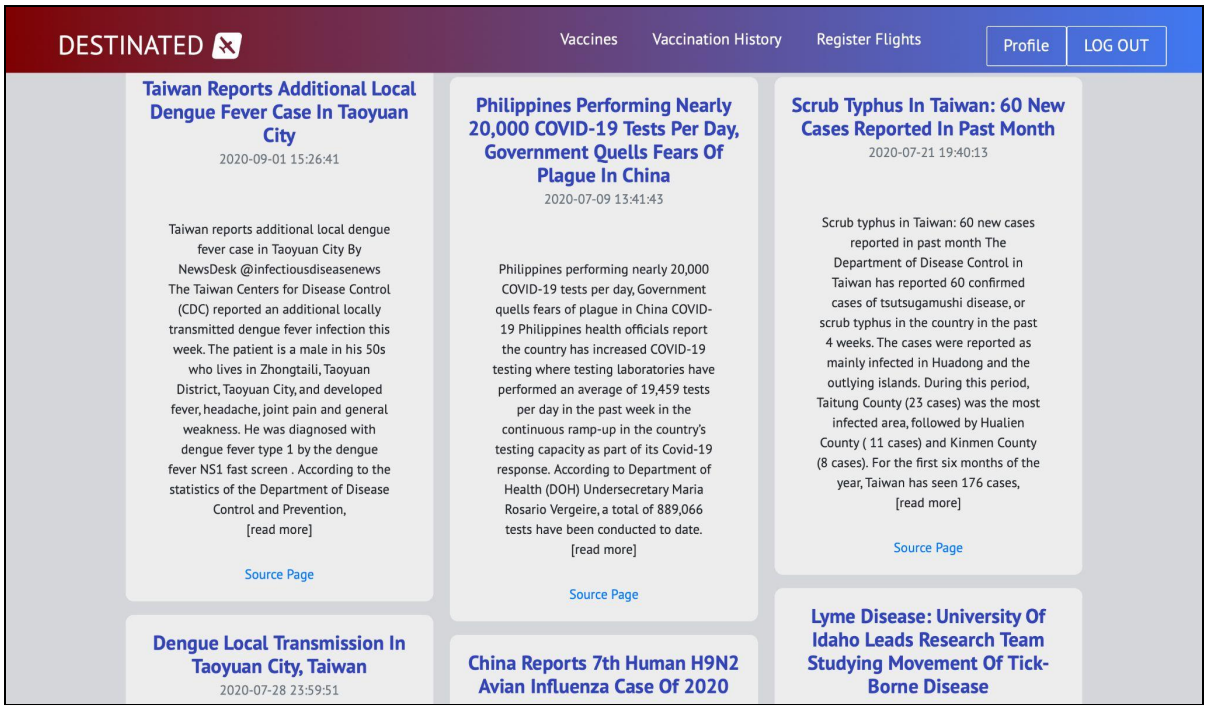



The above images show where an average user can select the country they are travelling to and then our app displays the corresponding CDC advice for the same location. For this we are scraping from the CDC.

## Restrictions

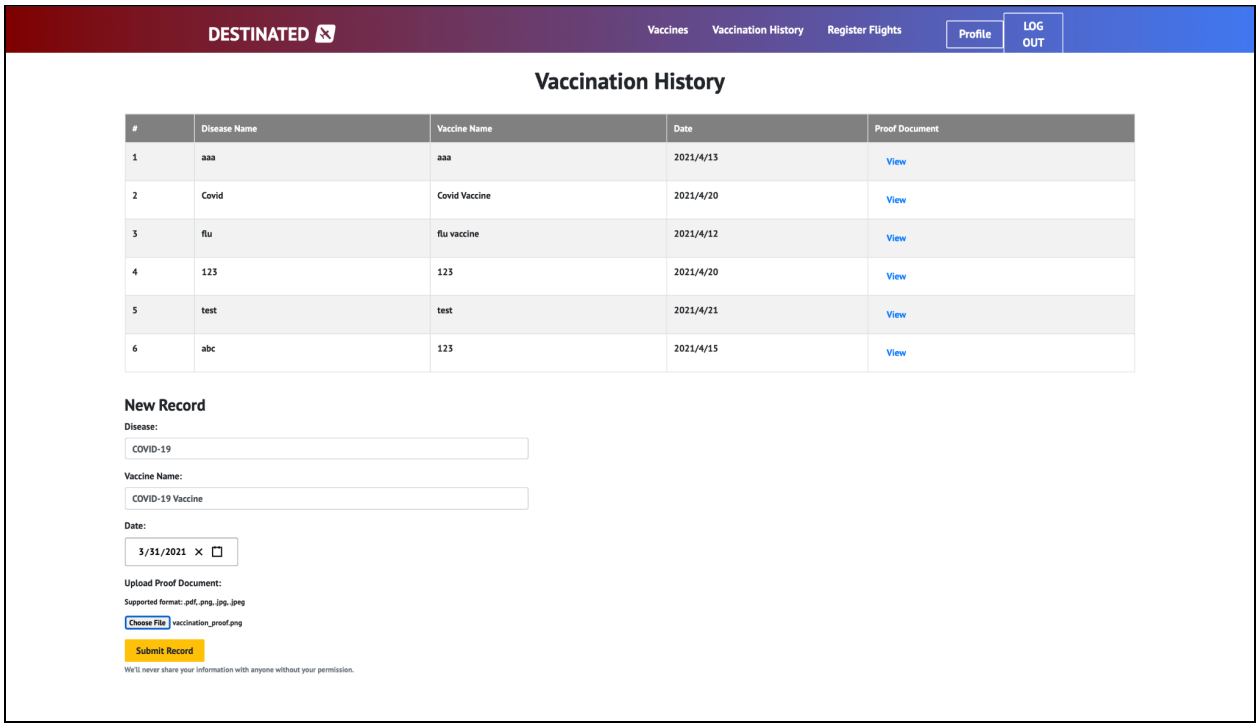



## Outbreak News



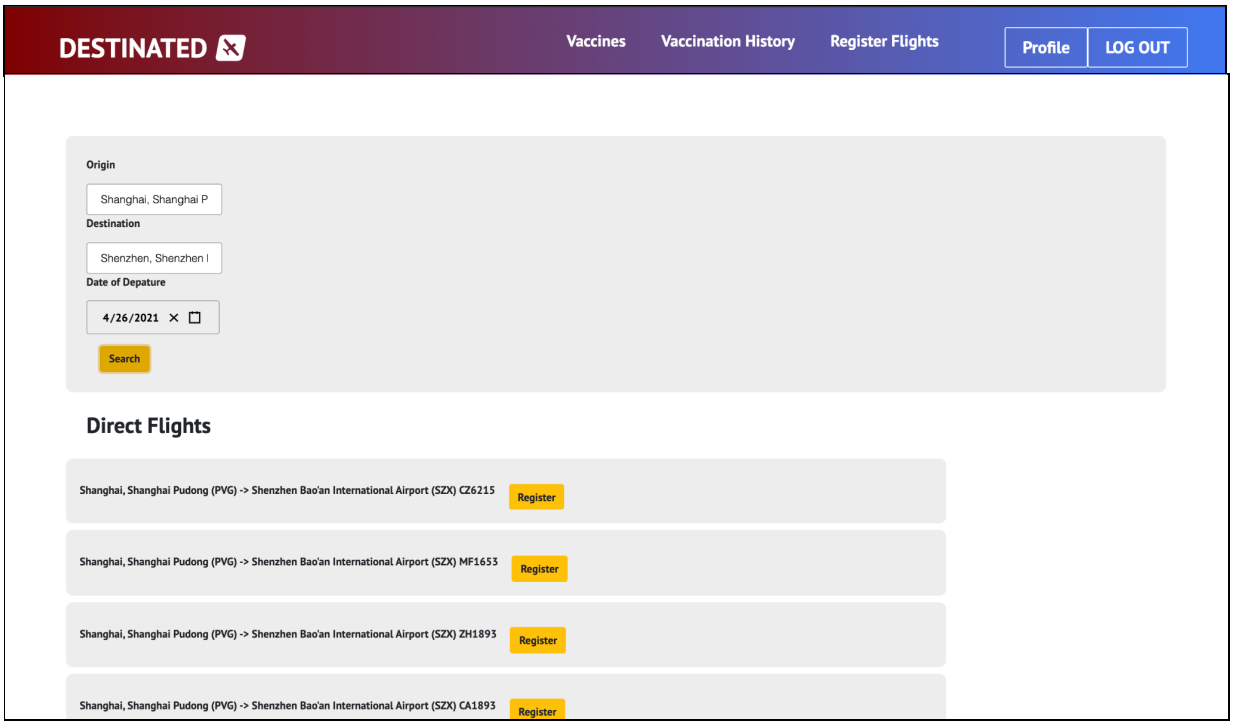

The above 2 images show the travel restriction page and outbreak page where an average user can view travel restrictions and recent disease outbreak news of the previously selected country. Here we have selected China for demonstration purposes.

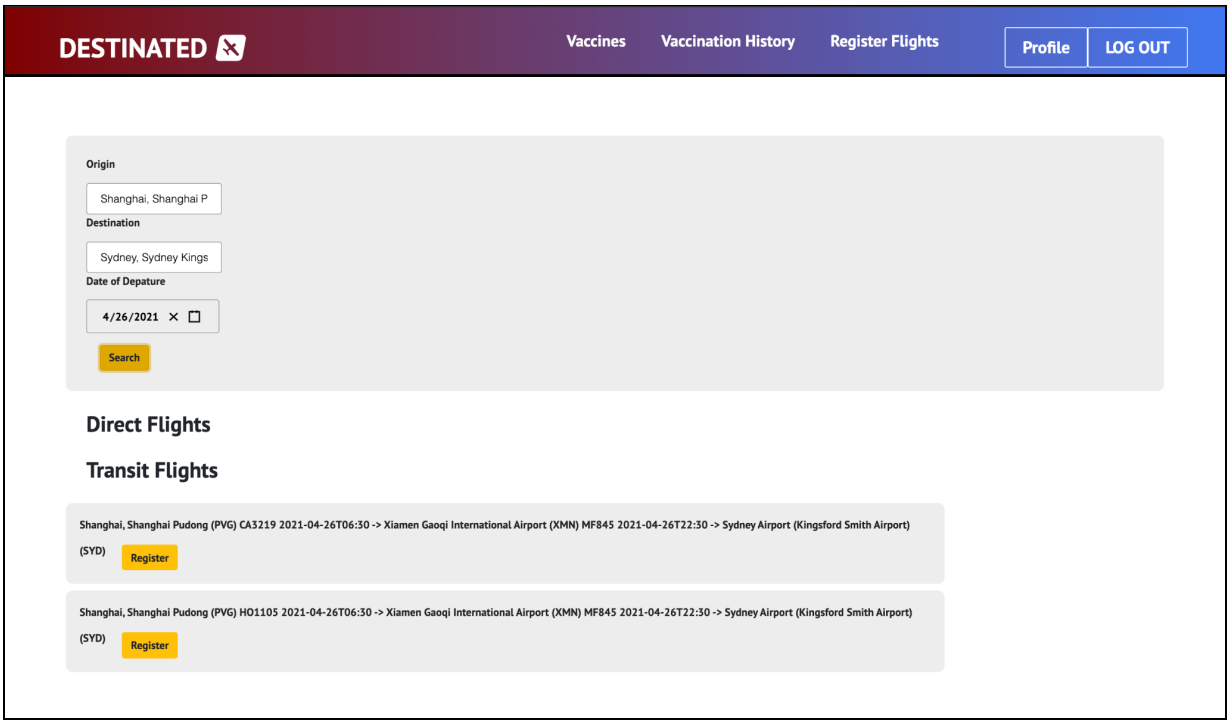
## Vaccine History




The above image shows the page where an average user can view their vaccination history and create new vaccination records by providing disease name, vaccine name, date of vaccination and uploading a vaccination proof document in pdf/png/jpeg/jpg format.

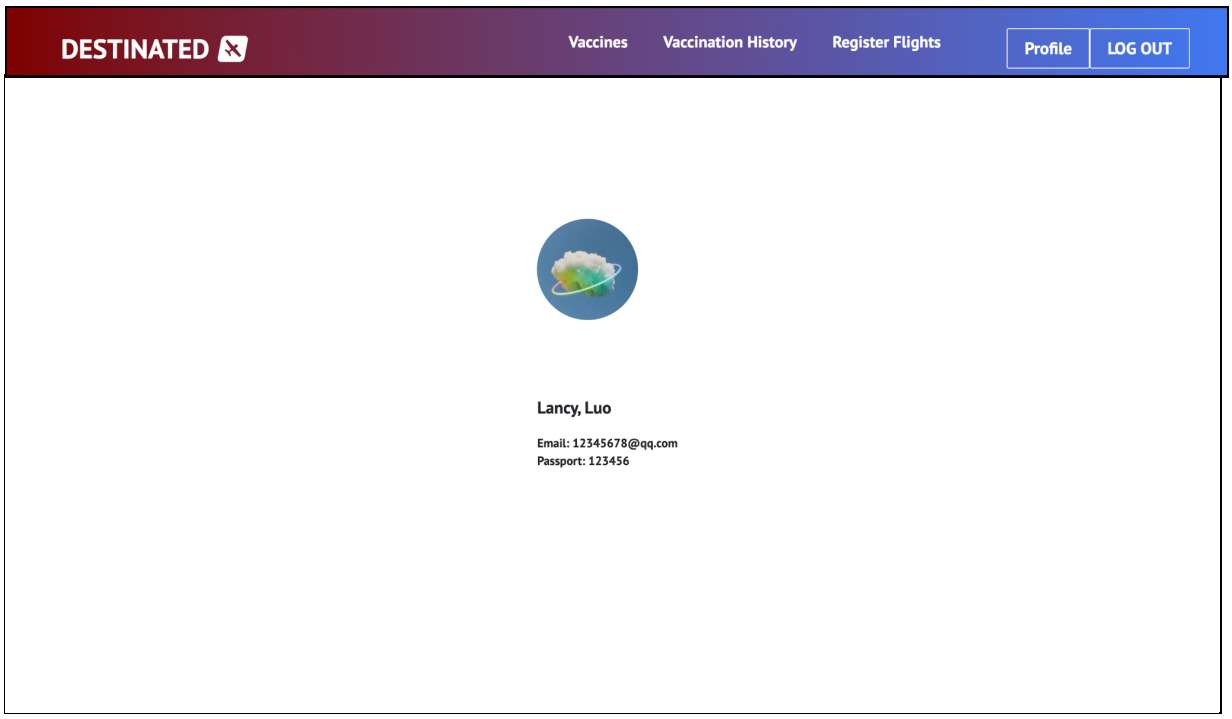
# Flight Registration



The above image shows the flight registration page where an average user can search for scheduled flights from the origin airport to the destination airport on a specific date of departure. Both direct flights and transit flights are displayed if there are any.

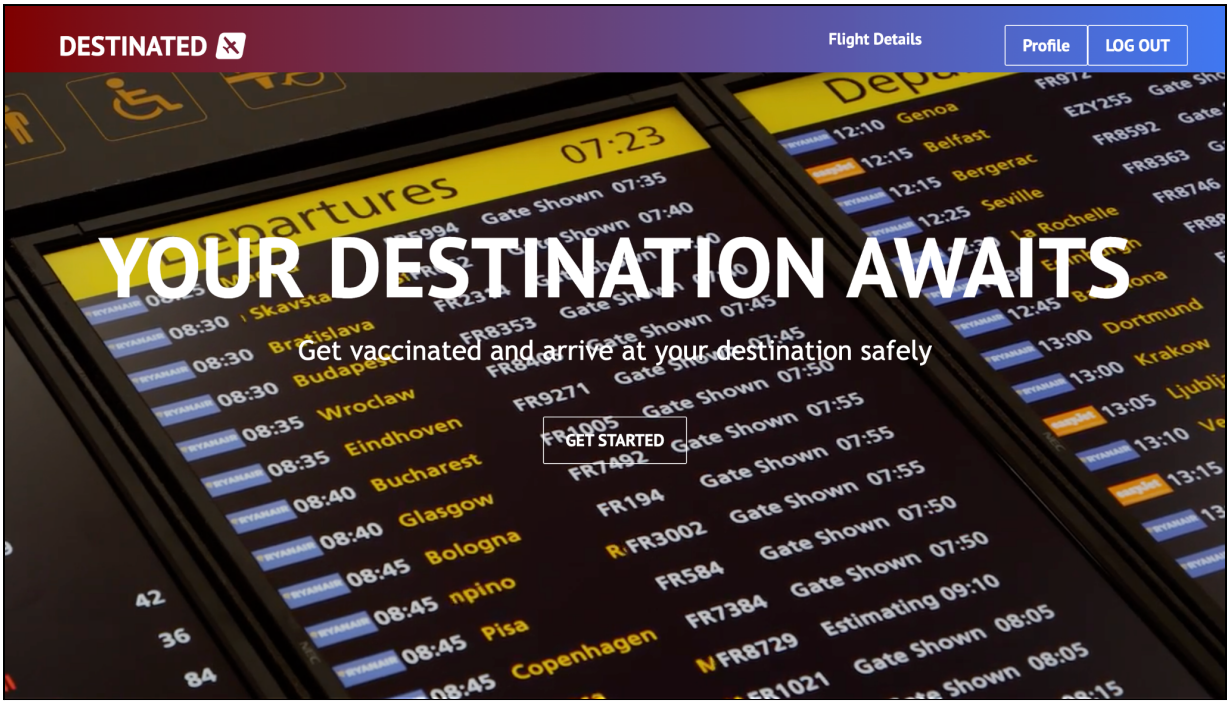
## Profile Page




This above image shows the average user's profile page. On this page, the average user's full name, email address and passport number are displayed.

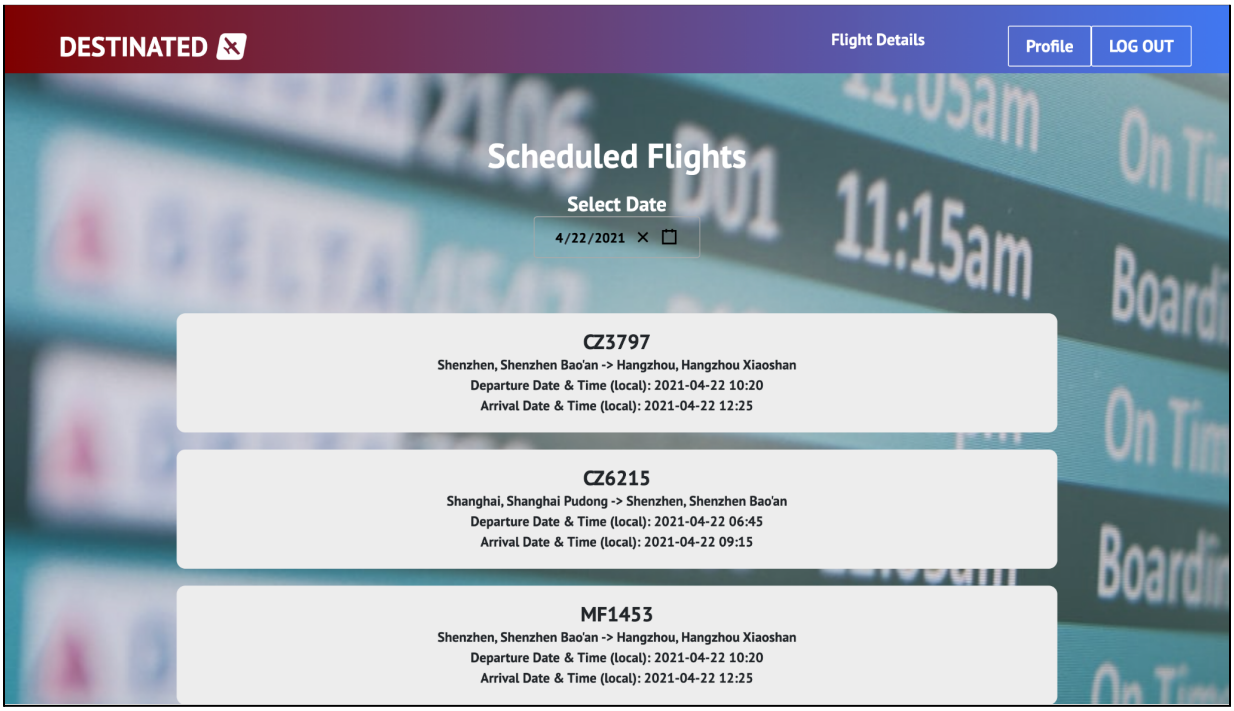
## Admin User

### Home Page



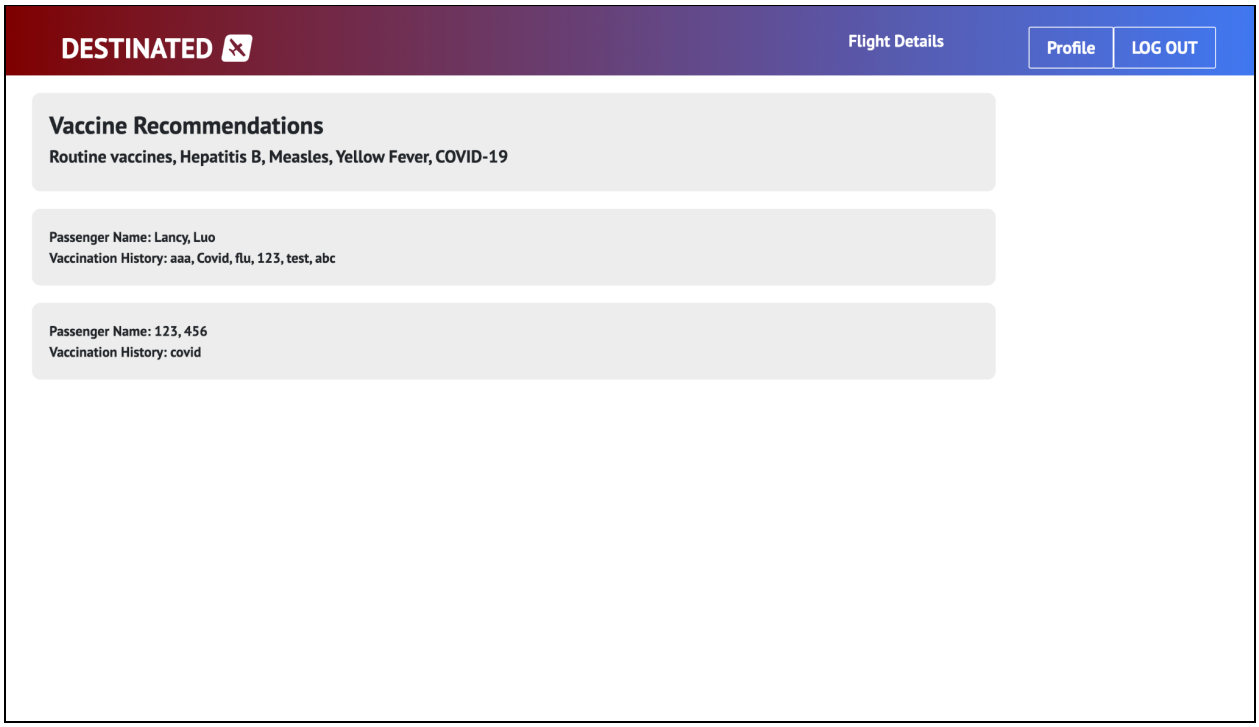

The above image shows the home page for admin users.

### Flight Details Page



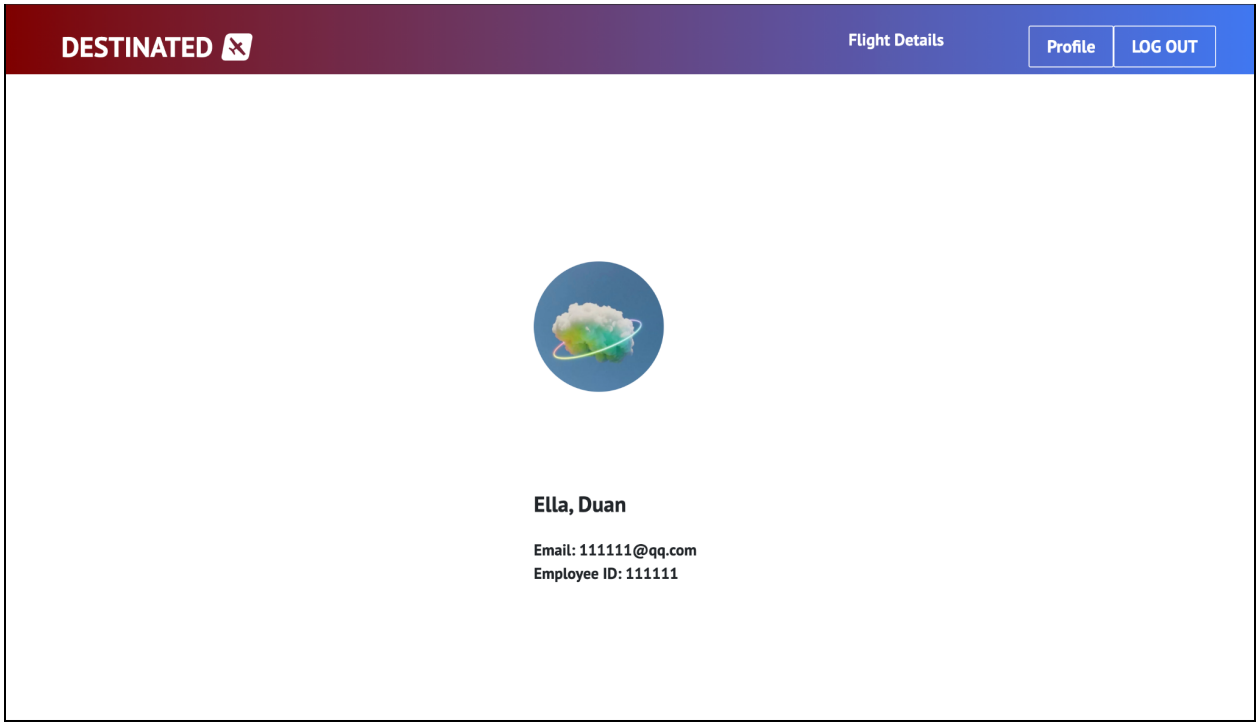
The above image shows the flight details page where an admin user can view lists of scheduled flights that average users have registered for on a specific date.

## Passenger Details Page




The admin user is directed to the passenger details page shown in the image above when he/she double clicks a specific scheduled flight. On this page, the admin user can view vaccine recommendations of the destination country as well as passengers' name and their vaccination history.

Profile Page




The above image shows the profile page where the admin's full name, email address and employee ID are displayed.

# System Design and Implementation

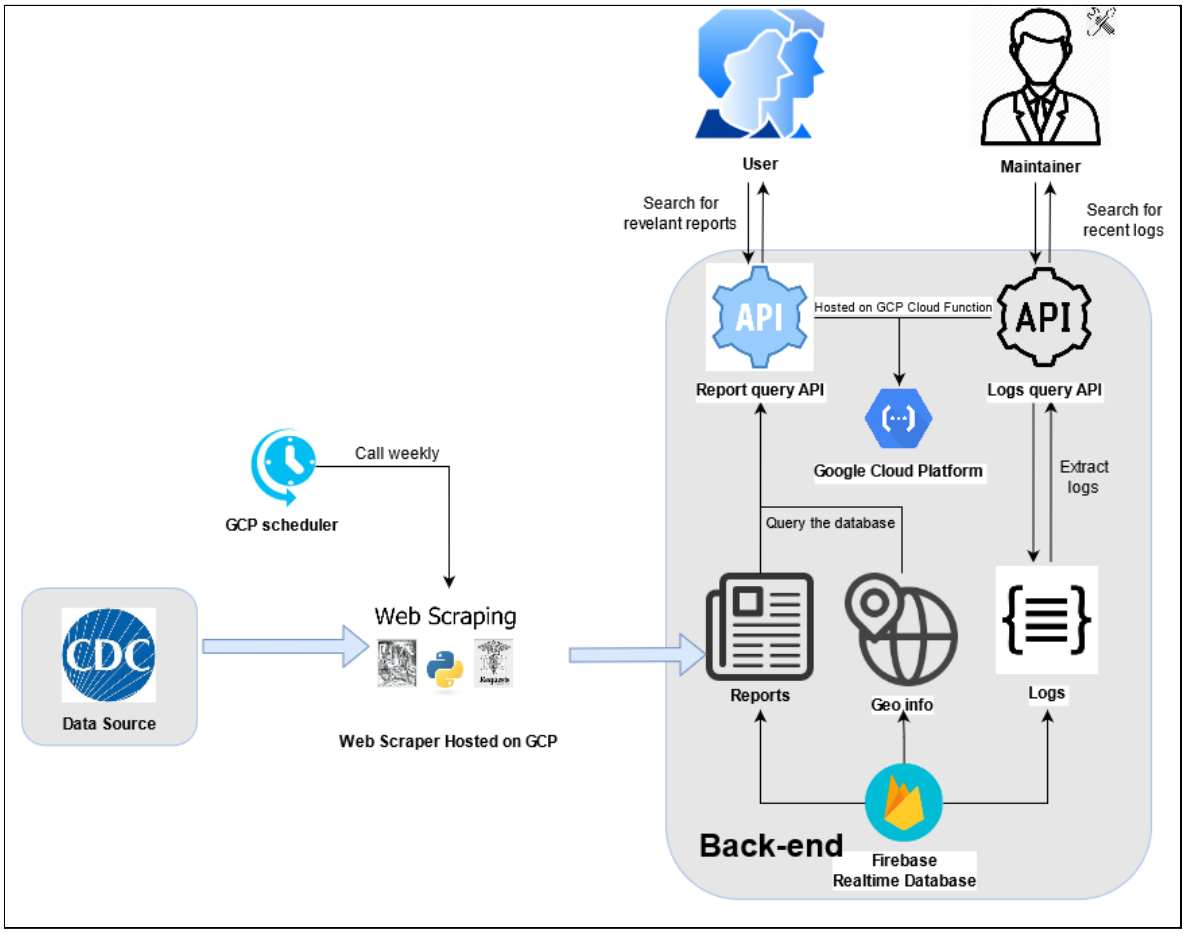
## Final API architecture description

Our API is written in Python and hosted as a cloud function on Google Cloud Platform. The API when called fetches relevant disease reports from the Google Realtime Database where the reports are stored. The disease reports are scraped by a web scraper that is hosted on a cloud function as well and it scrapes the CDC's website on a weekly basis using a GCP's scheduler.

The scraper makes use of libraries such as BeautifulSoup4, Spacy and APIs such as Geocoder to scrape the disease reports.

The API also has an endpoint that returns detailed logs of the previous calls made to the API.

The image below shows the overall architecture of our API.



## Libraries

- **Pyrebase** - It is a powerful python library that allows python programs to interact with Firebase Realtime Database with simple commands. We have made use of functions from this library extensively to make updates to our database.

- **Requests** - It is a widely used python library that allows to make HTTP requests in python. We have used this library in our scraper to access the CDC website.

- **BeautifulSoup4** - This is the primary library that we have used for scraping data. It is especially great for scraping irregular pages like the ones on CDC. It's CSS selectors are very useful for extracting data.

- **spaCy** - We have used this library for doing NLP in our scraped data. It helped us to extract vital information from the main text of the outbreak news.

## API Usage

- **Geocoder:** this API is used for querying list of hierarchy of a geo-location to enable nested location search for disease reports. (https://geocoder.readthedocs.io/)

## Deployment

The deployment of our API fully depends on the Google Cloud Platform (GCP).

### Google Cloud Platform

#### Cloud Functions

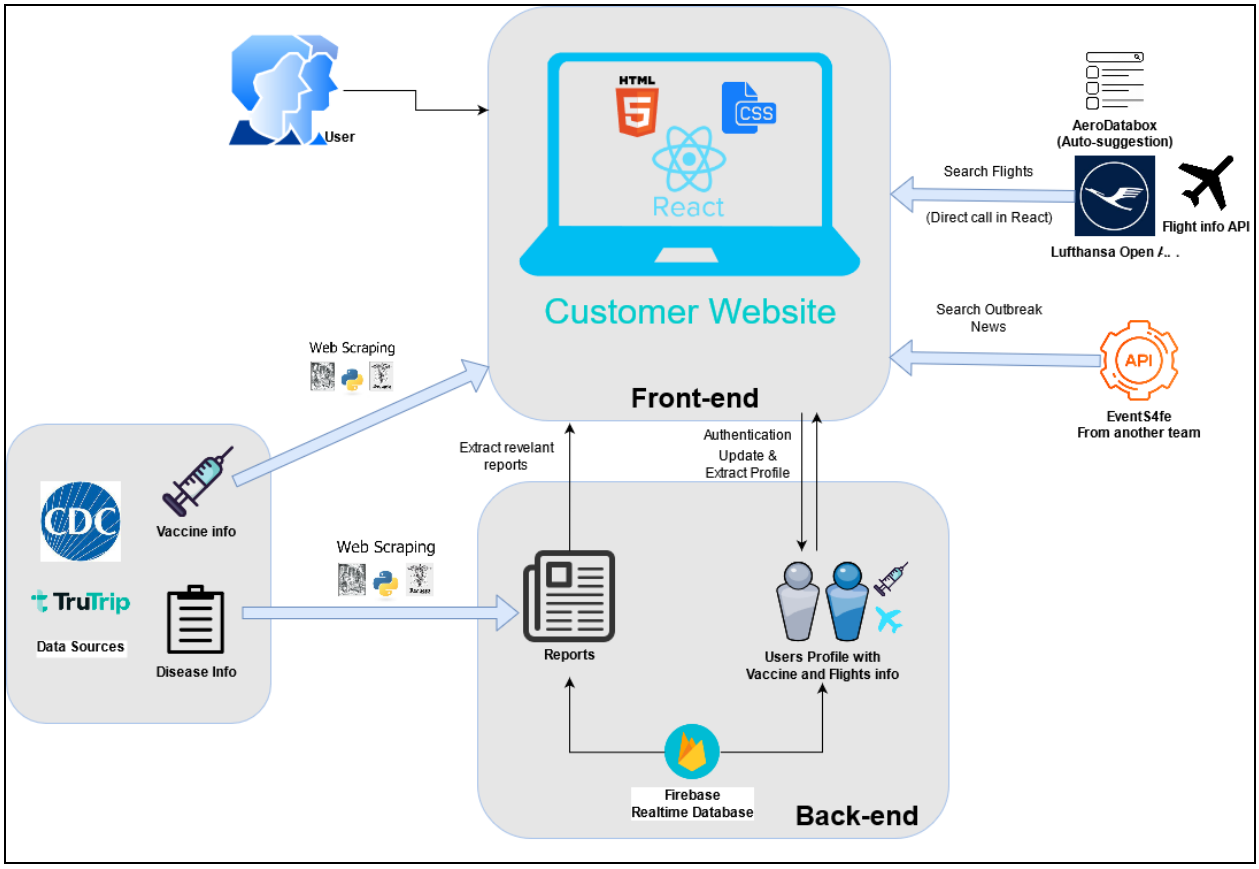
Google Cloud Functions is a serverless framework, which can automatically run backend code in response to events triggered by Firebase features and HTTPS requests. It's easy to integrate with other tools on the Firebase platform and it has zero maintenance cost as the Firebase automatically scales up computing resources to match the usage patterns of the users. Our scraper has been deployed in a Cloud Function and it will run every week to fetch new outbreak reports from the CDC website (see details in Google Scheduler section). Also, We have deployed our disease reports API using Cloud Functions, to allow users to use simple HTTPS requests to perform the query. In addition, for the performance measure, we have deployed a separate API to retrieve the log data and this API is also hosted in a Cloud Function.

#### Cloud Scheduler

Cloud Scheduler is a fully managed enterprise-grade cron job scheduler. Allowing to schedule virtually any job, including batch, big data jobs, cloud infrastructure operations etc. It acts as a single pane of glass, allowing us to manage all the automation tasks from one place. We use Cloud Scheduler to automatically call our scraper once every week, to maintain and update the

scraped data. Also, it can be used to automatically measure the performance by integrating with our Log API and generating reports which can be very useful in real-world app maintenance.

## Final Web App Architecture




Our web-app is called Destinated, The above image shows the overall architecture of our platform. It is hosted using Google's Cloud Services, and is made using ReactJS, HTML and CSS in combination with libraries such as React Bootstrap (details mentioned below). In addition, Realtime Database is used to store both average and admin user data (passport number, username, password etc), both sets of users are authenticated using firebase authentication. Destinated gets information directly from 2 scrapers. And the scrapers primarily use BeautifulSoup to extract information on vaccinations and restrictions from the CDC and TruTrip respectively.

Destinated also gives the average user local outbreak news of a particular destination, for this it makes use of EventS4fe API from the SENG3011 team GiveUsABr (https://events4fe.herokuapp.com/swagger/).

The web-app allows average users to store their vaccination records. These records are stored in the JSON database in base 64 format. We have used React-file-base64 library to convert images to this format.

For it’s final functionality, the app allows average users to register for flights.The flights data comes from several APIs (see API usage section)

Once a user is registered for a flight, an admin user can check the flights on a given day and see the vaccination records for any passenger on a particular flight by retrieving data from the database.

## Frontend

### React

It is a JS library that makes creating interactive UIs easy. It is also very easy to host react projects on GCP. Since we had to build our demo and web app in a short time, we chose React. This helped us to focus on the main functionalities of our rather than UI and deployment. As stated above it also made hosting very simple.

## Backend

### Firebase Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to the app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter etc.
Firebase Authentication provides two ways to secure our user:

- FirebaseUI Auth, which is a drop-in authentication solution that can provide a complete sign-in system to our app, handles the UI flows for signing in users with email addresses and passwords, phone numbers, and with popular federated identity providers, including Google Sign-In and Facebook Login.
- Firebase SDK Authentication provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails. Also, It can authenticate users by integrating with federated identity providers such as Google, Facebook, Twitter, and GitHub accounts.

We chose Firebase SDK Authentication for JS to implement the authentication part for our web app in the final decision, because it can be easily tested and our web app relies on the authentication to be easy to use.

### Firebase Realtime Database

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. Applicable for building cross-platform apps with it’s iOS, Android, and JavaScript SDKs, all of the clients share one Database instance and

automatically receive updates. We have been using Realtime Database all along in our project, mainly because it's easy to use and maintain, especially applicable for testing purposes. For disease report API we used three main entries including reports, logs and geo hierarchy list(to improve the query performance). Our web application is also related to Realtime Database as it's a user based app that needs mass of user data with reliability. Realtime Database is easily connected with Firebase Authentication to set Security Rules to highly secure users credential information.

Compare with another database provide by Google, the Firestore, they both offer:

- Client-first SDKs, with no servers to deploy and maintain
- Realtime updates
- Free tier, then pay for what you use

But after we considering the following checklist:

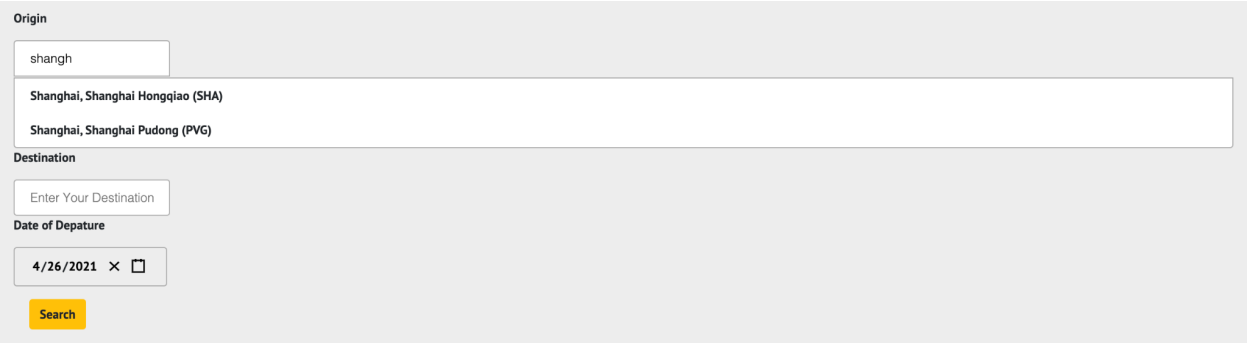
| Tasks | Realtime Database | Firestore |
|---|---|---|
| **Role of the database** | Basic querying with synchronized data. | Advanced querying, sorting and transactions |
| **Operations on data** | Few GB's data that can change frequently | Handle a large amount (TB's)of data. Read much more often than it changes. |
| **Data model** | A simple JSON tree | Documents organized into collection |
| **Availability** | High uptime at 99.95% at least | Extremely high uptime, 99.999% garenteened |
| **Offline queries on local data** | Concentrate on data synchronization, so need users to be frequently online. | Sophisticated querying capabilities on local data when the user is offline |
| **Number of database instances** | Can add multiple databases to a single Firebase project | Single database per project |

We considered the Realtime Database is the best option for all deliverables. Since we only need to perform basic querying and have a few data for disease reports and geo/vaccine info. Concentrating on data synchronization also serves the user's profile storing. Additionally, Realtime databases also let us add several databases in one project in different deliverables.

## API Usage

- EventS4fe API from team GiveUsABr (https://events4fe.herokuapp.com/swagger/): we have used this API to get recent outbreak reports of a location.

- AeroDataBox API (https://www.aerodatabox.com/):




The AeroDataBox API is used to get a list of airports with airport name or city name containing the search term case-insensitively as shown in the image above.

- Lufthansa Open API (https://developer.lufthansa.com/docs): we have used Lufthansa Open API to get all scheduled flights from the origin airport to the destination airport on a specific date.

- Airport info API (https://rapidapi.com/Active-api/api/airport-info): this API is used to get the airport name given an airport's IATA code.

## Deployment

In this project, we have successfully used two platforms provided by Google: GCP and Firebase, as a set of solutions to deploy our API and web app. There are several solutions we take for each are listed below:

### Google Cloud Platform

#### Cloud Functions

For our web app, we have developed two more scrapers to get all necessary information needed. One scraper is to get vaccine recommendations of a country from the CDC website (https://wwwnc.cdc.gov/travel/destinations/list/) while the other scraper is to get travel restrictions of a country from trutrip.co (https://trutrip.co/covidentrycheck/?lang=en). Both scrapers have been deployed in separate Cloud Functions and are called directly in the frontend.

### Firebase Hosting

Firebase Hosting provides fast and secure hosting for web apps, static and dynamic content, and microservices. With Firebase CLI, it's easy to deploy files from local directories to it's hosting servers. All content is served over an SSL connection from the closest edge server on Google's global CDN. With only a few commands we can deliver or update our React app to the server quickly. And also, it's easy to rollback if something goes wrong. As a result, this is a very convenient way of testing for everyone in the team.

In general, GCP and Firebase is charged based on the usage and starts for free, so it's definitely the economic solution for students to use.

# Key Achievements

The aforementioned Software design decision taken by our team allowed us to host our entire website on Google Cloud Platform so that it can be used by a normal internet user. Our entire deployment and Database Management was done on GCP, Firebase and Realtime Database, making our websites client information safe and secure from loss of users vaccination history, along with fast processing of the website and least latency in loading webpages and information from the database. This in turn allowed us to be more innovative in how we used our users vaccination history and showed it to the airport staff for further examination of whether they are fit to arrive at the destination or not. ReactJS and GCP allowed our ideas to become a reality, thanks to the variety of libraries and functions that were interactive for the user and were smoothly integrated with React, examples of such have been provided above. The fluidity of React allowed us to create helper functions to transfer data from one framework to another and also allowed us to easily deploy our web pages using Firebase. Finally, to make our website more user friendly, we made it such that a mobile user could also use Destinated with all the website's features intact. This allows a user to log onto Destinated anytime anywhere through his smartphone and stable internet connection, hence broadening our audience on the website.

# Project Management Information

## Organisation and Responsibility

| Name | zid | Major responsibility | Strengths |
|---|---|---|---|
| Abhyudit Gupta | z5196145 | Front-end, Project plan, Reports | Python, React, JS, HTML,CSS, Web-scraping |
| Jiahui Luo (Lacey) | z5158415 | Front-end, Project plan, API integration | JavaScript, React, Python,Web-scraping |
| Haoran Xu (Matthew) | z5134675 | Back-end, API design | Python, Firebase |
| Yueru Duan (Ellen) | z5210986 | Back-end, API integration, Reports | Python, Firebase, React, Web-scraping |
| Ayaan Adil | z5213315 | Back-end, API design, Project Plan, Reports | Python,React Web-scraping, Flask |

Table 1: Team members' responsibilities and strengths

Our team management has been the same since our first deliverable report of Management Information, where we planned to basically divide the team into two parts, front-end and back-end. Three of the members were mainly focusing on building the API and API integration, and two of the members were responsible for the implementation of the User-Interface of the web-application, as can be seen from the table above. During the Phase 1 of the project, the front-end team worked on the report and Swagger documentation, whereas the back-end team worked on the API implementation and API testing. During the second half of the project, Phase 2, the front-end members worked on the UI of the web-application, and our back-end team members worked on the API implementation and integration, database management, along with web scraping and website hosting. Before every deliverable, we collaborated as a team and checked on each other's work, finalising it together on group meetings and making it such that everyone agrees with the submission. For the final report, all 5 of us worked on it together to have the best possible outcome.

## Roles of Team Members in each Deliverable

| Abhyudit & Lacey | Ayaan, Ellen, Mattew |
|---|---|

| | |
|---|---|
| **Deliverable 1:**<br>● Management information report<br>○ Team responsibilities and work management<br>○ Decision making<br>○ Management tools<br>● Design Details Report<br>○ Justifying language<br>○ Justifying deployment host<br>○ Software Architecture | **Deliverable 1:**<br>● API design<br>● Implementation & Justification of language, deployment and development environment.<br>● Web-scraper testing<br>● Parameter passing<br>● Exploring cloud functions<br>● Design Details report |

Table 2: Team members' roles in Deliverable 1

| Abhyudit & Ayaan | Lacey, Ellen | Matthew |
|---|---|---|
| **Deliverable 2:**<br>● Web scraping disease reports, combine scarper<br>● Testing API<br>● Design Details Report<br>● Management information report | **Deliverable 2:**<br>● Web scraping disease reports<br>● Scraper helper function, combine final scraper code.<br>● Host web scraper on GCP<br>● API (Swagger) documentation | **Deliverable 2:**<br>● API implementation<br>● Database setup |

Table 3: Team members' roles in Deliverable 2

| Abhyudit & Ayaan | Lacey, Ellen | Matthew |
|---|---|---|
| **Deliverable 3:**<br>● Web App design<br>● Demo User Interface<br>● Demo presentation<br>● API integration | **Deliverable 3:**<br>● Low-fi prototype<br>● Demo User Interface Finalising<br>● Web scraping | **Deliverable 3:**<br>● Finalising Design<br>● Database Setup |

Table 4: Team members' roles in Deliverable 3

| Abhyudit & Ayaan | Lacey, Ellen | Matthew |
|---|---|---|
| **Deliverable 4:**<br>● Final Website<br>○ Web scraping travel restrictions page<br>○ Flight Register API<br>○ UI of Web-App<br>● Testing APIs<br>● Final Report | **Deliverable 4:**<br>● Final website<br>○ Vaccination pages implementation<br>○ Outbreaks news API<br>○ Flight Register API<br>○ UI of Web-App<br>● Initiating Final Report | **Deliverable 4:**<br>● Final website:<br>○ Database setup<br>○ Cloud functions, Website hosting<br>● Changes to final report according to previous deliverables feedback. |

Table 5: Team members' roles in Deliverable 4

## Communication

Since the members are located in different countries currently, most of the communication is conducted online.

- Weekly meetings
    - We had at least three meetings every week using MS Teams. One of them discussed the progress and follow-up plans of the whole team with the mentor. Feedback from the mentor allowed us to hold our other team meetings, which talked about decision making, every member's progress, problems faced as well as assigning the future work if needed. The team members working on the same component had meetings within a smaller group.
    - The most preferable meeting time is decided by doing polls on messenger group chat.
- Instant communication
    - We also have a Facebook messenger group for instant communication. If anyone encounters a problem, it is more efficient to ask in the group chat than waiting for the weekly meeting. If the issue is not solved within the group, we contact our mentor, Richard Liu, who replied and solved our doubts almost instantly. Richard often agreed on taking extra meetings with us whenever necessary on short notices.

# Management Tools

- Task Management
  - The Github Issue Board was used for task control.
  - During each phase, the issues were created firstly.
  - The requirement and test criteria were written in the issue and the issue is assigned to one or more of the team members before actually starting to code.
  - When there are bugs, new issues are created for that.
  - All issues are clearly labelled. (example: help needed, bug, database, API, frontend, duplicate, enhancement, UI, etc)
  - The issue is closed when it has been implemented and tested.

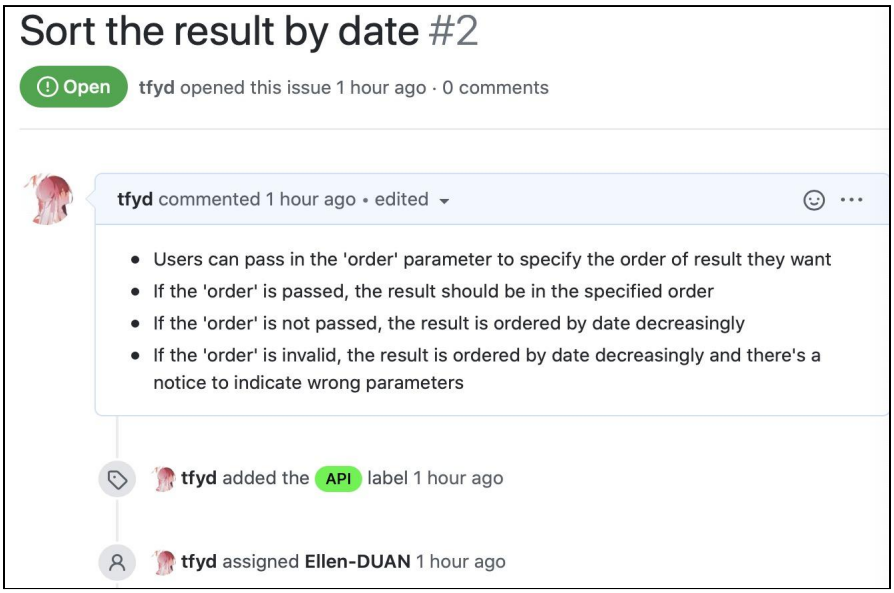



Figure (a): Github Issue board

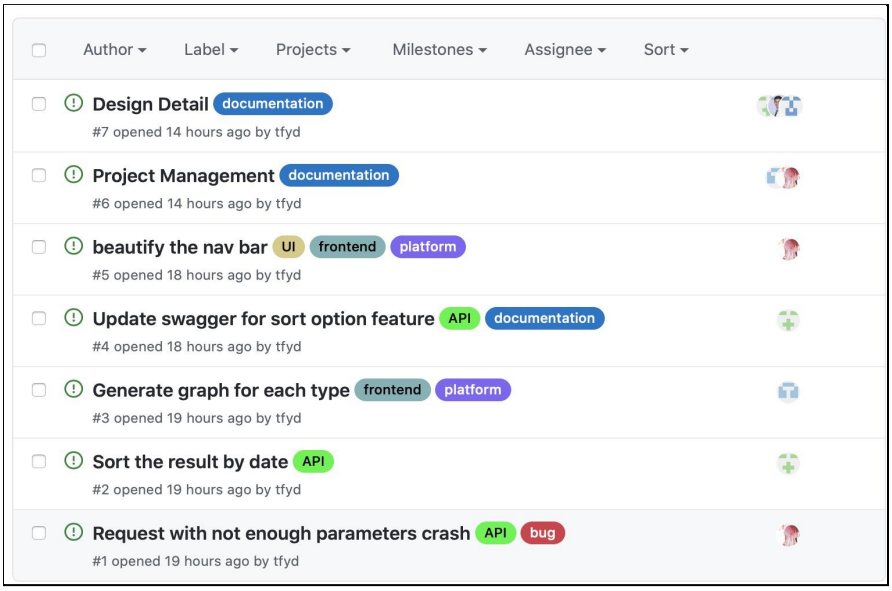



Figure (b): Github Issue board

- Code Management
    - We use Github as the tool for code management and version control. Each new feature starts from a new branch, and Github Pull Request is used to ensure the code in the main branch is the latest runnable version.
- Documentation
    - For documentation, we have been using google docs to collaborate with each other for the reports and other documents, such as API design. When there are conflicting ideas on the document, those are commented respectively on google docs, which are later resolved in group meetings.
    - The final documents get uploaded on GitHub.

## Decision Making

During the project, decision making was done similarly throughout the project. After having a meeting to list all the alternatives and evaluate the advantages and disadvantages of every possible option, voting was done to understand everyone's interest and make the best possible decision. If members have conflicting decisions, we went with the majority and further consulted our mentor for what's best.

Group environments can be challenging for many reasons and when we encountered any problems during the project, we as a team resolved the issue and tried our personal best to solve it.

## Our experience and how we went

Collectively, we felt that the SENG3011 experience was ultimately a big learning curve in terms of both learning to deal with different group dynamics and new web frameworks. However, excluding the challenges faced, it was a positive experience and as a group we are proud of the amount of software engineering knowledge we have acquired over a short period of time and the final product we produced, which ultimately was a culmination of our efforts and our newfound learnings.

## Learning Experience

Throughout the project, there were a number of technical and soft skills that we all learnt making this course valuable in more ways than one regardless of our final product.

- How to work as a group to achieve a common goal - Dealing with new group dynamics was something we all had to overcome and learn in order for us to achieve the common goal of producing a final product to a high standard. As a result of this we were able to learn and develop the following skills.
    - Organising and Planning Skills - As we had ensured that all deliverables were completed on time after editing, finalising and proofreading.

    - Decision making skills - Decision making skills were utilised as we collectively came up with an innovative website in which all team members had the opportunity to make decisions and further implement them.
    - Problem solving skills - Problem solving skills were used while we encountered issues relating to the integration of the API, how to physically implement the ideas we had and how to flesh out ideas in order to make our website more important and worthwhile.
    - Being responsive to feedback - Needed to rebuild or edit any code or documentation that our mentor thought was not executed to the best of our ability or was incorrect.
    - Ability to build rapport and trust in a short period of time with not only our teammates but also our mentor, was also a vital learning curve.
- How to effectively deal with team members who have different levels of experience, skill sets and understandings- Overcoming the gfap in the different level of understanding between each team member was a challenge but we also were able to harness the 4 following skills in order to deal with this
    - Communication skills- learnt how to deal with the limited interactions with our teammates and mentor through online meetings.
    - Conflict resolution skills were utilised as our team was a mix of five very different individuals with varying skill sets and ideas. Hence, our conflict resolution skills were developed to overcome conflicting ideas and opinions to ensure we were all working towards the same common goal.
    - Goal setting skills were developed due to the rapid pace of this curse and the constant need to put out work in accordance to the deliverables.
    - Creative thinking skills were also utilised when we had to decide a unique business idea for our website and how to efficiently use our API along with other API's and data sources to implement helpful features on the web application.
- How to build and develop a API and web application from start to finish- Being thrown into a foreign environment from the beginning was a positive experience as we were able to experience and implement each stage thoroughly.
    - Design Phase and Documentation - Process of choosing the best tools and software technology among numerous others to have the best plausible design solution. Documenting our design and team organization along with our thoughts in a formal manner.
    - Backend - creating the functionality of all our features(GCP functions and Web scrapers in python, noSQL Database, API creation, set up and integration)
    - Frontend - developing a high level UI (Dynamic and interactive web pages in ReactJS, consistent navigation bar in ReactJS and many other components within the ReactJS web application in Javascript)
    - Integration of Backend and Frontend - developing both front-end and back-end such that they work seamlessly together when hosted on Google Cloud Platform.

# Challenges Faced

## Communication Problems

a) Due to the limitations placed on us because of the COVID-19 pandemic, not having physical meetings and mentoring sessions disadvantaged the group greatly. The lack of in person communication limited us from understanding each other personally and efficiently working in a team, and hence it became difficult to hold people accountable if they did not meet the deadlines set. Hence, group connection was low and thus we were hardly on the same page causing unnecessary confusion.
b) As the team members live in different timezones, we faced communication problems. Deciding a meeting time which was comfortable for all 5 of us was tricky at times.
c) We also faced a major miscommunication problem during D2 when we only scraped reports of U.S based outbreaks although we had to scrape reports for all diseases in the disease list.
d) Due to division of labour, we had to wait for the person responsible for that section to respond in order to clear our doubts or solve a bug.
e) We faced few communication problems in D3 and D4 as well when we were stuck and couldn't progress further as we waited for a team member's work or reply to an important message.

## Time Management

a) Our lack of time management led to some last minute issues, but this did not stop us from working hard and delivering our best.
b) WE faced time management issues in Deliverable 2 as well as Deliverable 3. Learning from our mistakes we started early and worked as efficiently as we could to finish Deliverable 4 on time without having last minute panics.

## Web Scraping

a) HTML pages on the CDC website have inconsistent formats. Due to this we had to make customised scrapers for each individual disease, which was a very tedious task.
b) Having no previous experience in Web scraping, it was a big learning curve for us, and took us some time to get familiar with different types of web scraping frameworks.

## New Experience with Cloud Services, Libraries and API integration

a) Further, when it comes to the programming side of the workshop, the development of a web stack was foreign territory for some of the group members. Thus, our final challenge was having to balance learning new frameworks and implementation of these frameworks into our project.

b) Since all of us had never worked with cloud services, we had to learn how GCP Cloud functions and Firebase Realtime Database work, this took up valuable time we could have spent on adding functionalities.
c) We also had to work with new libraries like BeautifulSoup, Spacy and geocoder which took up time we could have spent on the API.
d) Using the geocoder to get the location information was also a challenge. Initially, the problem was deciding the query logic between the child method and the hierarchy method, and we realized the hierarchy method was always faster since the child method is nested and involves searching down all the trees. The second problem was that even if we used the hierarchy method for each API request, it would still be very slow as it would involve repeated calls to the geocoder API (with each api call taking 300ms on average, to complete a query will be using approx 30 seconds to complete, which is definitely not what we wanted) . Our solution was to save the geo information into our database and go through the db first prior to the API call to reduce the api calls (after improvement, we saved 28 sec on average for each query, so it's an impressive change).
e) Different APIs work differently, integrating multiple APIs into one web application got confusing and at times one or the other API didn't work as desired. Few days before our deadline we had to change our flight registration API, from skyscanner to lufthansa. At times our AeroDataBox API worked slow or didn't work as it was a free API and could only handle 100 requests per hour. All these issues took us extra hours which we could have spent on adding new features or improving our user friendly website interface.
f) None of our team members had experience with web hosting, getting familiar to the environment and reading through online helper guides took us time, but in the end we successfully hosted our website on GCP.

Despite these setbacks, the team was diligent enough to complete deliverables with a high quality and finally due to the nature of the workshop we were able to learn plenty of new skills such as the development of a WebStack, web-scraping and creation of API along with API integration and web hosting.

## Shortcomings

1) The team was unable to get reports for all diseases in the disease list because of the lack of outbreak information on the CDC website.
2) The team wanted to implement a CI/CD pipeline which would have allowed us to constantly integrate new code and focus on the requirements of the application rather than deployment. But due to the paucity of time we were unable to add this feature.
3) The team had also planned to program the scraper such that it scrapes important information about deaths, hospitalisation etc. But again due to lack of time management we were unable to add this feature.
4) We only added the basic functionality of a GET request in the API and were unable to add other kinds of requests ex POST, DELETE.

5) We were not able to dedicate much time to the testing of the API as we kept a greater focus on the scraper.
6) We were not able to deliver the best User Interface in our D3 demo presentation, due to poor time management and lack of experience with React.
7) Having multiple functionalities and integration og 4 API's for the D4 web-application, we were not able to dedicate more time to the UI design and make it more interactive or customisable for the user.

## Skills we wish we had before the workshop

1) React
    a) None of our team members had deep understanding or good experience with reactJS.
    b) During D3 and D4 we had to learn React from scratch and go through online helper guides and posts to debug our code.
    c) Even if any 1 or 2 of our team members had experience with react, we could have successfully given a higher level of end product with more features and more user friendly and reactive interface.
2) Web scraping
    a) None of our team members had previously used a web scraping framework and scraped a website.
    b) During D2, learning how to use scrapy and beautifulSoup, along with understanding the use of other scraping frameworks, in order to choose the best framework for our CDC datasource took each team member some time.
    c) Had anyone had previous knowledge about web scraping frameworks, we would have not spent time on deciding which is the best framework for our datasource and could have finished our D2 deliverable one day prior to the due date.
3) Web hosting
    a) None of our team members had previously hosted a web app online on a cloud function. This was a new learning curve for us which took us some few hours to understand but we managed to ace it in time.

## Working on the workshop again

- Considering the group had never worked together before, the project went well. If we get the chance to do it again, we think we would ensure from day one that before each deliverable began we have a strong written down plan as to how all the targets would get achieved. After the responsibilities and plan have been finalised, we would straight go to finishing our parts as early as possible.
- Despite few hiccups in deliverable 2, when it came to organisation, our team was extremely efficient and was able to meet the deadlines. We split up responsibilities in each deliverable between the five of us and made sure that we gave each other constant updates and resolved any issues we were facing either through group calls or

messages. We would follow the same organisation and splitting of responsibilities to get the best results on time.

- Just like we did during this workshop, every individual will be given a task to undertake in each deliverable which plays to their strengths, and if a given task is too complicated for one person, it will be shared between two people to even out the load. We will give each other deadlines for specific tasks in order to stay on task.
- Additionally, we believe we would ask further detailed questions to our mentor which will enable us to maximise our marks and end with the highest order of final product.
- We would stick by the same tools we used during this workshop as we are more familiar with them now and we can now use those tools at their full potential and finish up with twice better product than we did during this workshop.
- We would still use python for some of the back-end coding and use the same scarpers that we used during this project. Instead of ReactJS we could explore NextJS as it it is blazing fast, it’s easy to deploy, gives API routes and it lets us customize our webpack configuration, along with many more features.
- To improve our time management, we would change our working and get straight to working on our assigned parts, as soon as the plan gets final before every deliverable. Unlike this project, we won't relax for a day or 2 before actually starting coding, to be on track and submit a few days before each deliverable, this would give us extra time to get feedback from our mentor and do the desired changes easily, without panicking.

# Conclusion

In conclusion, we have built an outbreak news API by the name 200OK API, and a web application in this project. Our API scraps information from the CDC website every week and provides the desired and relevant reports, which include disease name, date, location of outbreaks, all the relevant news of disease outbreak(as per the parameters provided), syndromes of disease, to the clients. Our travel and vaccination website, Destinated, integrates the vaccination requirement, travel restriction, destination news, personal history, flight search and flight registry functionalities. This platform helps users who want to travel to another country to obtain almost all kinds of required information regarding outbreaks at a one stop website. However, due to the time limit, there are further improvements needed, including adding algorithms to verify vaccination history, showing vaccination certification to airport staff, distinguishing recommended and required vaccines, substituting the API with more stable ones and adding many more funcalities to the website. The limit for Destinated is beyond what we could deliver in this short span of time.

All in all, despite the challenges that arise with such a foreign environment, as a group we are proud of our final solution and the amount of software engineering knowledge we have acquired over a short period of time.