SENG3011 Y_2 Final Report: Canary

Contents

Background

Requirements

Given the following specifications and problem space:

> Develop a platform that provides one or more of the following functions for an end user interested in epidemics detection:
>
> • Ability to integrate data from different sources and presented in a user-friendly way
> • Ability to browse news related to a disease outbreak over a period of time/geographically, identify news about outbreak of interest
> • Ability to examine social media related posts on disease outbreaks over a period of time, identify particular trends
> • Ability to provide advanced analysis facilities such as analysing the impact of an outbreak on residents of the region over a period of time or predicting potential outbreaks based on historical patterns.

we ideated that a useful solution would incorporate at least 3-4 of the above requirements in the features that are delivered to an end-user. Moreover, the delivered application should not only provide meaningful insights into epidemiology suitable for the average consumer, but also do so in a way that is both refreshing and provides value to both businesses/consumers in one of the following industries:

- Travel
- Food
- Retail/e-commerce

chosen for how impacted they were in the wake of the global pandemic. Narrowing down product scope to one of these industries not only allows the application to inform users interested in epidemics detection, but also widens the target market and hence potential user-base to consumers in that given industry. Instantly, it is evident where epidemics detection and analytics can be applied in each of these industries, with proposed ideas ranging from apps that restaurant owners could use to make their business outbreak resilient, or apps that online-shoppers could use to manage shipping/postal restrictions on overseas purchases. Ultimately, the travel industry was selected as it presented the most number of and interesting potential features/use-cases that could enrich a consumer's end-to-end travel and tourism experience in a post-pandemic world.

Problem Statement and motivation

What if you are planning your next overseas trip which has multiple locations spanning the length of an entire year? A multi-legged trip may prove to be a nightmare to plan, given differing health advice from different health authorities/regulatory bodies, travel restrictions that are subject to change and an increased chance of outbreaks due to the effects of population growth on the epidemiology and evolution of infectious diseases. Some of these considerations may be multi-tenanted and conflicting;

> "Local travel advice?"
> "Immigration health regulations?"
> "Vaccine history?"
> "Government health regulations?"
> "Travel restrictions?"

Below are two disjoint data sources both providing information that someone travelling to India may find useful:

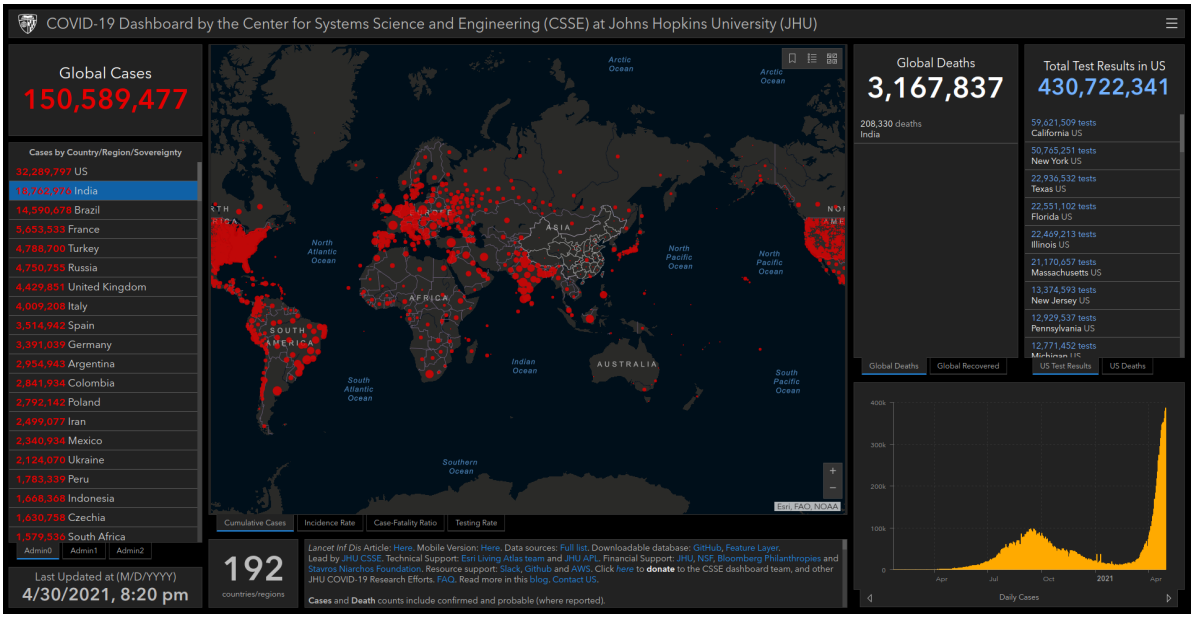


Figure 1.1 (above): COVID-19 statistics for India. A user would have to Google "covid19 dashboard", navigate to https://gisanddata.maps.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6, then select "India" as the country to view disease statistics.

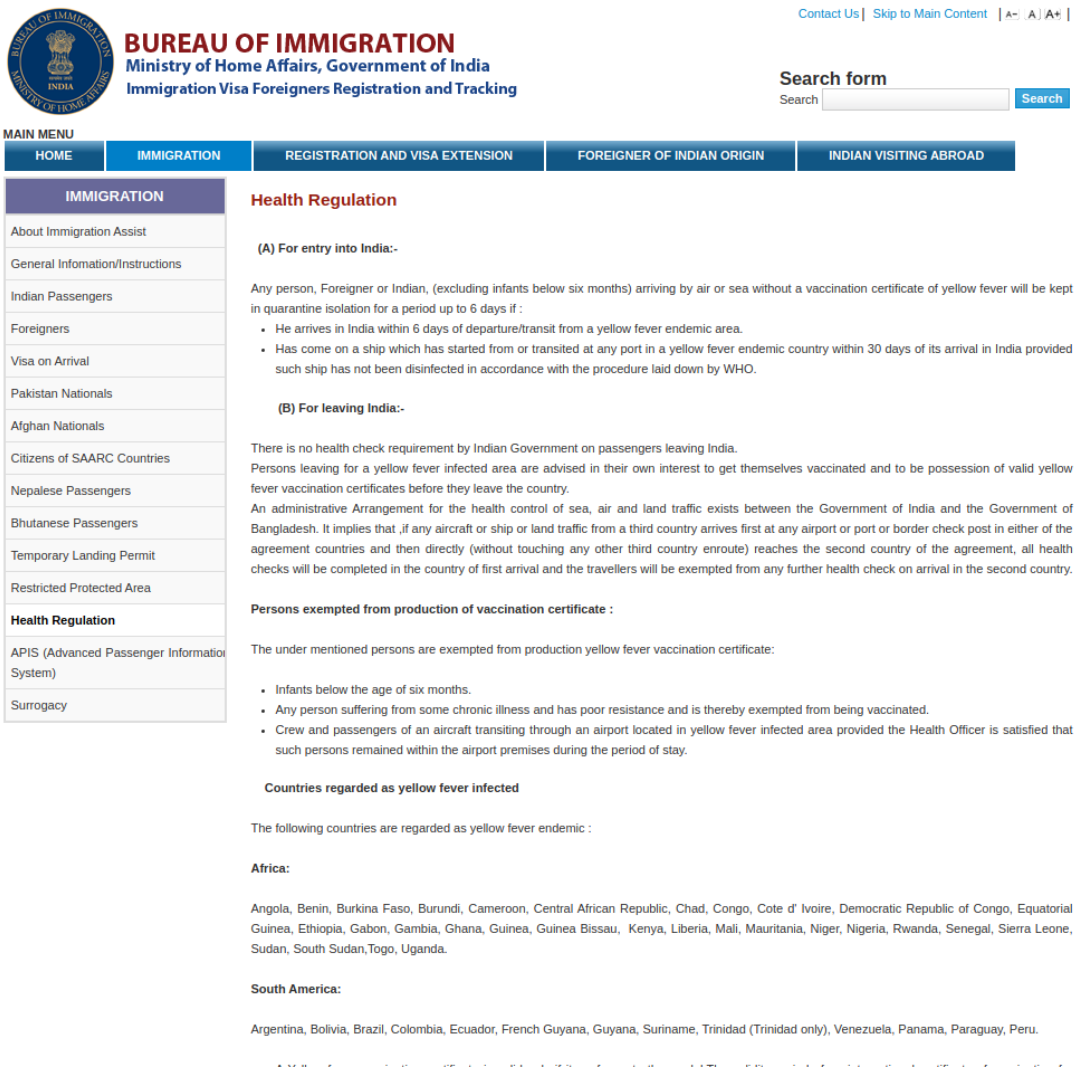


Figure 1.2 (above): Bureau of Immigration website for India, a user would have to Google "immigration india", then find "Health Regulation" from the confusingly named "Immigration" dropdown on the topbar.

Adding and multiplying the number of different data sources that must be researched and visited for each city/country in a trip spanning multiple locations and lengths of time, a user would have many hours of tedious health-safety planning added on to their trip planning. Clearly, there is a lot of information that must be researched, collected and collated meaningfully in a way for a consumer to make health-safety informed decisions whilst planning their next trip. This can prove to be difficult, inaccurate, time-consuming and confusing.

> *"How can I plan my next trip safely?"*

The above problem statement or product "log-line" is an encapsulation of the motivation behind Canary. As outlined above, for a traveller looking to book their next overseas trip, there are a large number of considerations to be made, considerations that are impacted by the current state of public health in their desired destinations.

Canary aims to take the pain out of this process with a simple, centralised platform that integrates real-time epidemiology data to provide a range of use-cases and features to enrich the end-to-end travel experience of a consumer with piece of mind regarding their health.

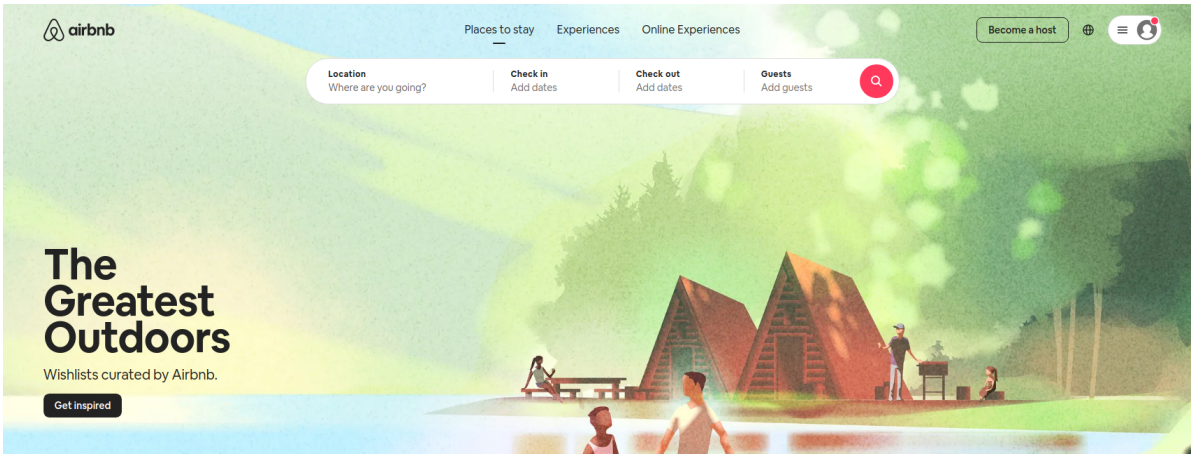
Features

Plan

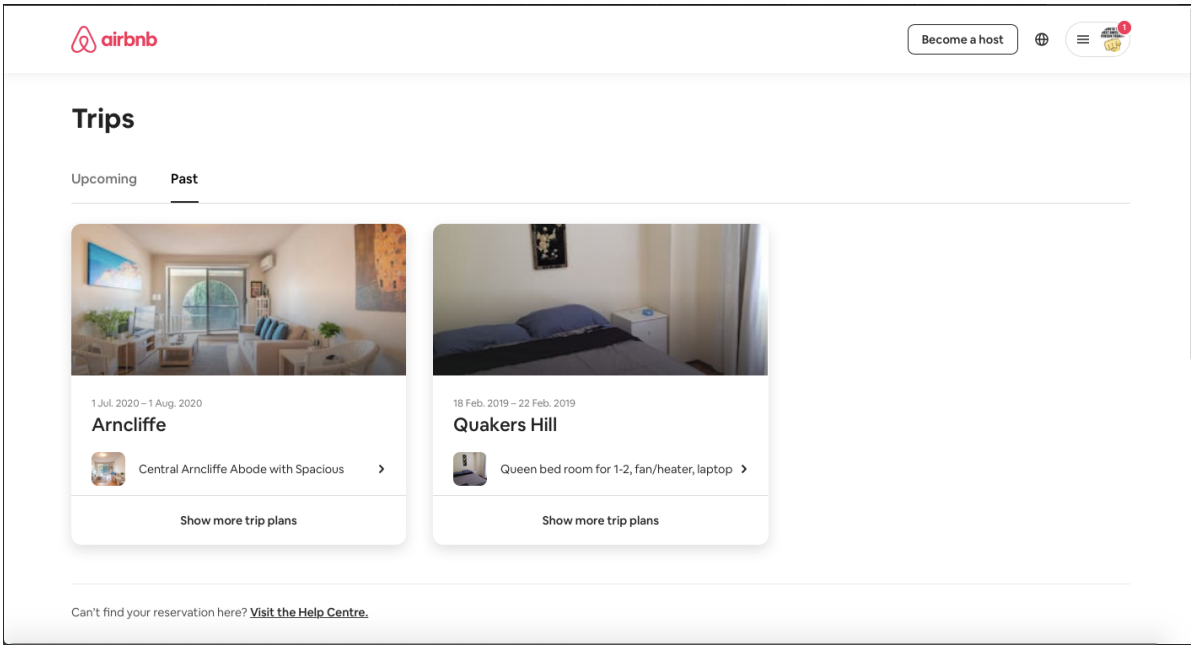
Share

Stay informed

A user planning their next overseas trip would likely start with a list of locations, and date ranges for those locations. Then, they would either book their flights or accomodation next. Below is the main page of Airbnb, where users can search for stays based off a location, and start/end dates for their stay at that location.



Similarly, Airbnb users can also save their trips:



Given that this is an unofficial 'itinerary standard', we wanted to give Canary users functionality and use-cases surrounding the following three main areas:

- Input a list of locations and start/end dates, then receive all the health travel advice for this proposed trip
- Save these plan results to a 'Trip' (Upcoming/In-progress/Past)
- Share 'Trips' to other users

- Providing real-time alerting for users with in-progress trips
- Provide generic disease/global and local health news reports
- Provide disease/epidemic/health news reports that are customisable by location, disease and data source

Trips

A user can generate a 'Trip' by beginning inputting their travel itinerary on the homepage. A trip consists of 'Stops' , with each 'Stop' consisting of a location, start date and end date.

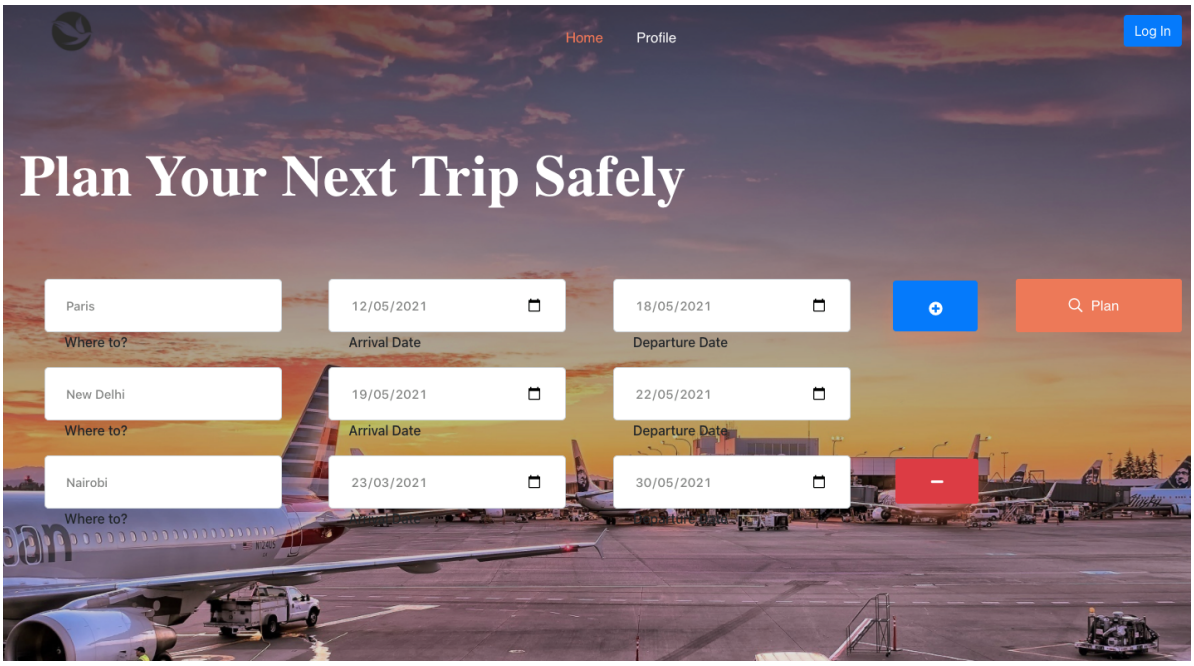
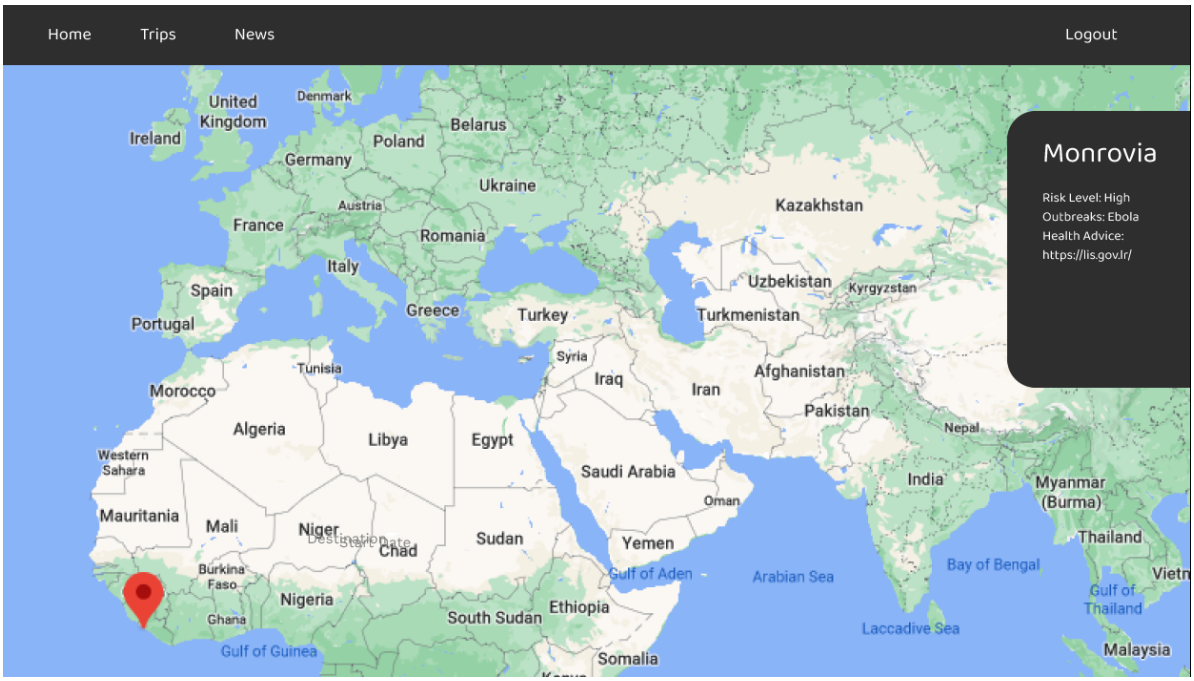


Figure 1.3 (above): User inputting a trip with Paris, New Delhi and Nairobi as stops along with date ranges for each location

After clicking 'Plan', a user is then taken to the 'Trips' page, where travel locations and corresponding health advice for each is displayable, via a slide-out modal and tooltips on a map. Below shows a mock up trips result for a trip to Monrovia.



Unfortunately not implemented in time for the front-end, below is a data excerpt for the intended travel-advice data returned from the back-end to be displayed after inputting a trip with Nairobi as a location:

```
    "country": "Kenya",
    "status": "Do not travel",
    "latest": "New restrictions are in place to prevent the spread of COVID-19.
Movement is prohibited in or out of the counties of Nairobi, Kajiado, Machakos,
Kiambu and Nakuru. These counties, declared a disease infected area, are subject
to an 8pm  4am curfew and additional restrictions on worship, sporting activities
and operations of bars and restaurants. The curfew in the rest of Kenya remains
10pm  4am. Adhere to restrictions and follow the advice of authorities. Monitor
the media for latest developments on COVID-19.",
    "summary": "\nCOVID-19 remains a risk in Kenya.Monitor the media for latest
developments on COVID-19 and follow the advice of localofficials.
SeeTravel.\nMalaria is widespread except in Nairobi and places above 2500m.
Consider taking anti-malarial medication. Other insect-borne diseases include
dengue, Rift Valley fever, filariasis and African sleeping sickness. Ensure your
accommodation is insect-proof. Use insect repellent.\nYellow Fever is widespread.
Get Vaccinated before you travel.\nHIV/AIDS infection rates are high. Take
precautions if you're taking part in high-risk activities.\nPolio outbreaks occur
in Kenya. Check your vaccinations are up to date at least 6 weeks before you
travel.\nFoodborne, waterborne and other infectious diseases include hepatitis,
meningococcal disease, measles and cholera. Drink only boiled or bottled water.
Avoid raw or undercooked food.\n",
    "info": "COVID-19\nCOVID-19 remains a risk in Kenya.Monitor the media for
latest developments on COVID-19 and follow the advice of localofficials.
SeeTravel.\nMore information:\n\nOffice of the Kenyan President\nKenyan Ministry
of Health\nNovel coronavirus - (Department of Health)\nCoronavirus (COVID-
19)\n\nInsect-borne diseases\nYellow fever is widespread in Kenya. Yellow fever
is a potentially fatal virus spread by mosquitoes. It's prevented by vaccination.
Get vaccinated before you travel.\nMalaria is widespread except in Nairobi and at
altitudes above 2500m.\nOther insect-borne diseases occur, such
as:\n\ndengue\nRift Valley fever\nfilariasis\nAfrican sleeping sickness\n\nTo
protect yourself from disease:\n\nmake sure your accommodation is insect-
proof\nuse insect repellent\nwear long, loose, light-coloured clothing\nconsider
taking medication to prevent malaria.\n\nVisit a doctor if you develop either a
fever, muscle pain, a rash or a bad headache.\nHIV/AIDS\nHIV/AIDS infection rates
are very high.\nTake precautions if taking part in activities that put you at
risk of infection.\nPolio\nPolio outbreaks happen in Kenya.\nCheck your
vaccination status for polio with a doctor or travel clinic. Do this at least 8
weeks before you travel.\nIf you aren't vaccinated, complete the full course of
vaccinations before you leave. If you've been vaccinated in the past, get a
booster dose if needed.\nMore information:\n\n\nAustralian Immunisation
Handbook\n\n\nOther health risks\nWaterborne, foodborne and other infectious
diseases are common. These include:\n\nhepatitis\nmeningococcal
disease\nmeasles\ncholera\ntuberculosis\n\nSerious outbreaks occur from time to
time.\nTo protect yourself from illness:\n\ndrink boiled water or bottled water
with sealed lids\navoid ice cubes\navoid raw and undercooked food, such as
salads\ndon't swim in fresh water\n\nGet medical advice if you have a fever or
diarrhoea.\n"
```


The travel advice consists of three main pieces of information:

1. `status`

   A 'rating' provided based on the health risk posed to a traveller visiting this country. This can either be `Do not travel`, `Reconsider your need to travel`, `Exercise a high degree of caution` or `Exercise normal safety precautions`.

2. `latest`

Latest new of the unfolding public health situation

3. `info`

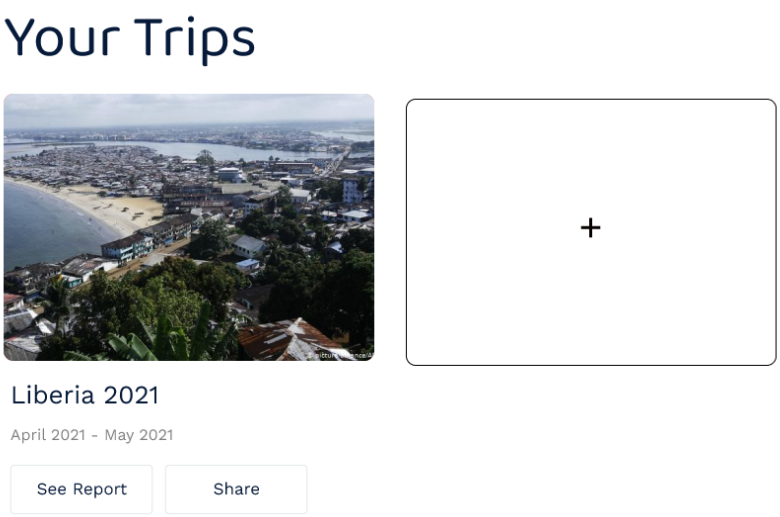
   Health advice detailing common infectious diseases prone to outbreaks in Kenya, and basic health advice for tourists.

4. `summary`

   A summarised display of `info`, expanded to `info` if the user selects the drop down arrow.

This provides data crucial to the knowledge and decision-making of a user planning a trip to Kenya.

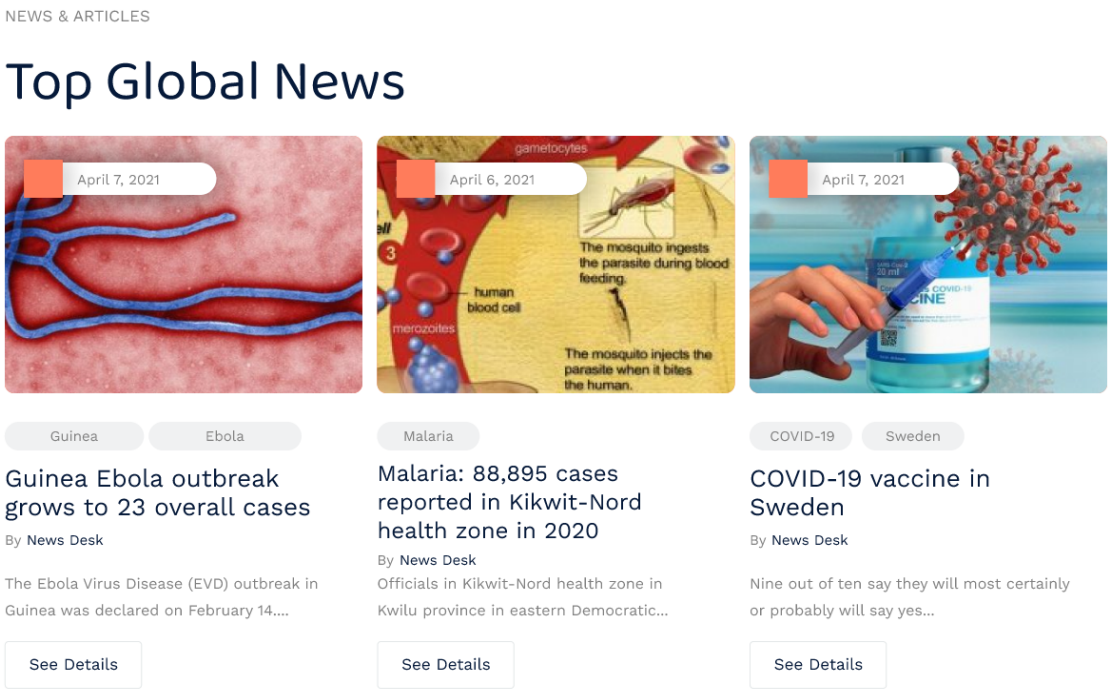
A user can also see their saved, upcoming and in-progress trips on their 'Trips' page:



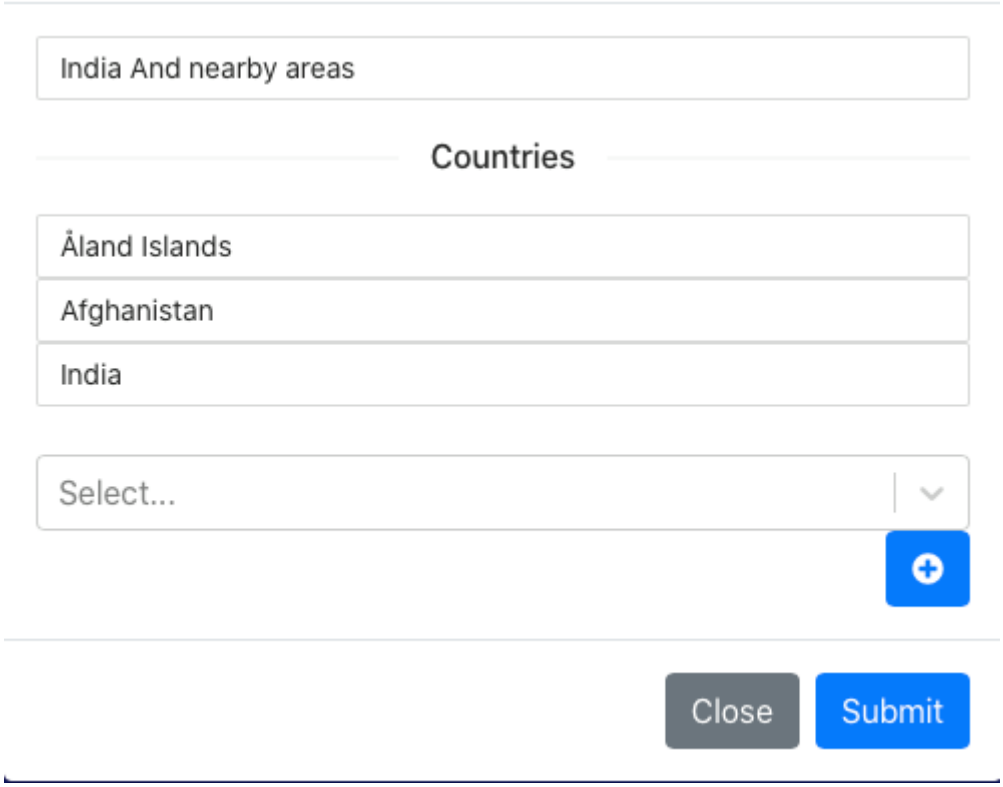
With the option to view the resulting travel advice page for a Trip, or share it with a shareable link to another person.

News

On the home-page, a user can scroll down below the 'Trips' input panel to see two news panels, one containing "Global" epidemiology news and another containing "Local" news articles:

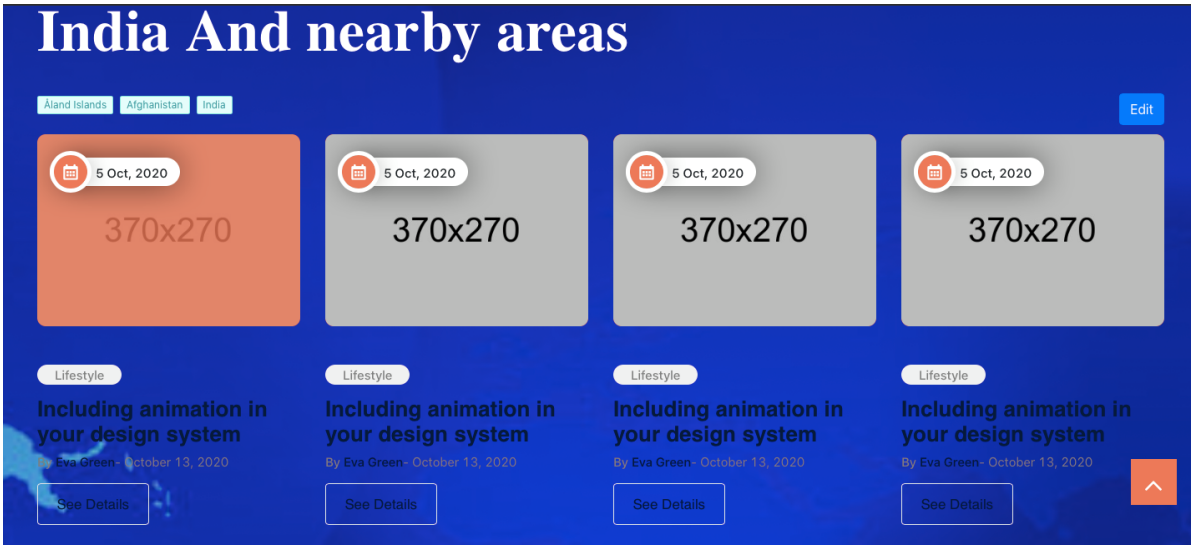


Alternatively, a user can customise their own news panels on the 'News' page, creating panels with the following modal:



Allowing users to name panels and select location tags which filter articles by those locations, with "Select" allowing a user to select from a range of data sources for their news articles.

Resulting in the following (sample text populated) news panel:



Alerting (Proof of Concept)

Another feature that was only developed to a Proof of Concept stage in the back-end was a feature in which a user receives real-time alerts based on users with in-progress/upcoming trips matching updated info for locations produced by a data source.

Alert info generated from the back-end:

```
"country": "Venezuela",
"status": "Do not to travel travel"
"advice": "COVID-19 is present in Venezuela. Measures are in place to limit the spread of the virus, including community-wide quarantines. Road travel between Venezuelan states is banned, with roadblocks in place and limited internal flights. Many international flight routes have been suspended. Commercial flights to and from Venezuela remain very limited. Australia doesn't have an embassy or consulate in Venezuela. Contact the Australian Embassy in Colombia for assistance.We continue to advise:      Do not to travel to Venezuela due to: the unstable political and economic situation, shortages of food, water, medicine and petrol, and high levels of violent crime, the health risks from the COVID-19 pandemic and the significant disruptions to global travel."
```

This would allow users to either receive email or SMS alerts tailored to their upcoming/in-progress trips.

Extensibility and Integration

Since Canary's 'Trips' feature consist of 'Stops' with the same input format as Airbnb 'Stays', an integration with Airbnb APIs could have Canary users search across Airbnb stays for a given 'Stop' on their trip. This could be integrated into the 'Trips' page, with an option to toggle an 'Airbnb stays' filter on the map, which would display tooltips for stays at the Stop locations which are available during the given date range. Clicking on a listing would take the user to the Airbnb listing on the Airbnb website.

Similarly, if a user is using Canary as this first point of deciding whether or not to book a trip, the input location and date-range for a given 'Stop' could also be used to integrate with flight booking APIs such as Skyscanner Flight Search API which would display available flights from booking providers such as trip.com or trivago.com. Alternatively given public APIs, the booking providers could be directly queried.

System Architecture

Translating features into functionality

Given the varied and rich feature set for Canary, how can the application be architected in a way that is scalable, easy to develop, performant, extensible, lightweight and fully managed? A micro-services based architecture could deliver against all these requirements, and by leveraging a Public Cloud Provider such as AWS, managed services can be harnessed for ease of development as well. Moreover, the use of server-less computing and AWS Lambda could also be leveraged to better integrate with micro-services, allowing each back-end function/feature to be mapped to a Lambda function.

The choice of GraphQL for the D2 API has made for a very extensible API, which was simply extended to include a query and attached Lambda resolver for each piece of back-end functionality. Below are the updated design decisions from D2, with new challenges, advantages and disadvantages.

Architecture

Technologies and justifying decisions + Implementation

Database

Choice

Given the ease of using a managed NoSQL database, I was inclined to stick with Amazon DynamoDB for the final application. Remaining AWS native allowed me to once again harness the ease and functionality of `boto3`, allowing new data sources to be scraped in new ways without changing the underlying database implementation. No schema allowed new tables containing simple json object store to be created for each different type of object. Once again, the data structure from the database, to the Lambda resolver and the GraphQL schema all remain equivalent due to the simple json object store nature of DynamoDB. The simplicity and ease of this allowed minimal ownership of objects, further simplifying not only the database implementation but also the back-end implementation, as the only state that is kept is a user's saved trips, and news preferences (if any).

Implementation details

The back-end data heirarchy consisted of four simple tables:
`who_articles`
`travel_advice`
`global_articles`

Which are populated by scrapers scraping from various data sources, and :

`Users`

which keeps state of a user.

`who_articles` objects are populated by scraping from www.whoint/countries/IND with country's Alpha-3 ISO 3166 code as the endpoint. They have the following data format:

```
article_preview: String
country: String
date_of_publication: String
main_text: String
thumbnail: String
title: String
who_article_id: String
```

and are uniquely identified by a 16 digit `uuid`.

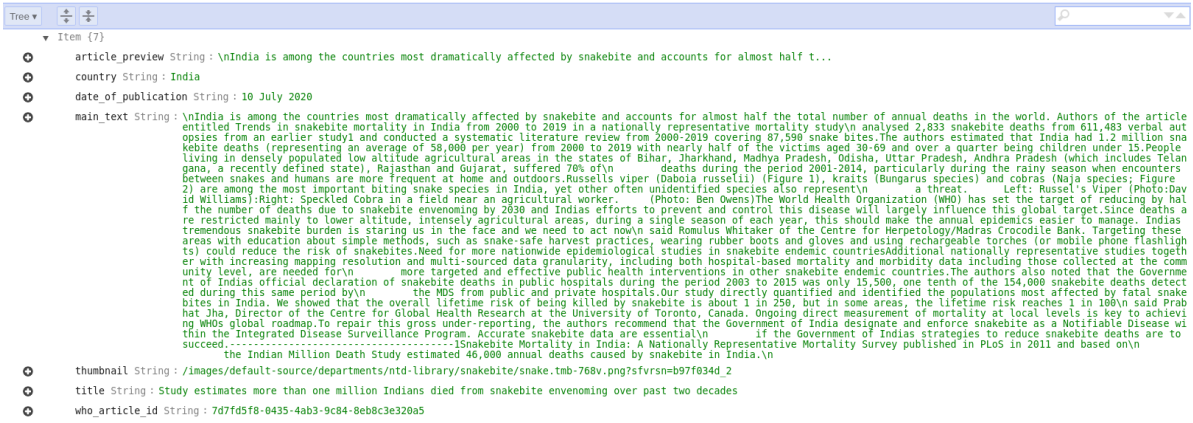


Figure 1.4 (above): An example `who_articles` object

`global_articles` objects have the same data format, but with `global_id` as it's unique identifier, and `country` as always `global`:

```
article_preview: String
country: String
date_of_publication: String
main_text: String
thumbnail: String
title: String
global_id: String
```

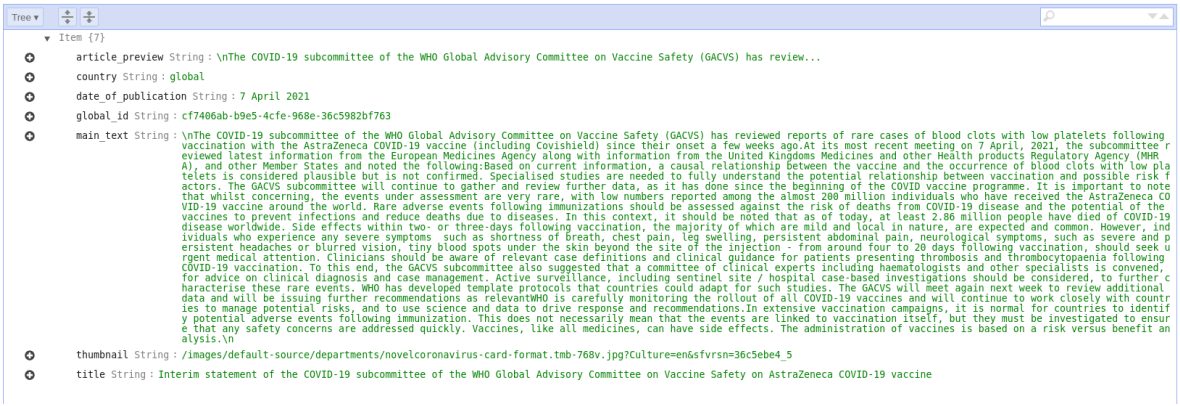


Figure 1.5 (above): An example `global_articles` object

`travel_advice` objects have the following data format:

```
country: String
info: String
latest: String
status: String
summary: String
travel_advice_id: String
```

Figure 1.6 (above): An example `travel_advice` object

A `Users` object book-keeps all relevant user related state. It has the following format:

```
user_id: String
user_name: String
user_email: String
user_mobile: String
trips:  [Trip]
news_streams: [Stream]
```

with a `Trip` child object having the following data format:

```
trip_name: String
trip_id: String
status: String
stops: [Stop]
```

and finally a leaf `Stop` object having the following data format:

```
city: String
start_date: String
end_date: String
```

a `Stream` child object:

```
stream_name: String
data_source: String
tags: [String]
```

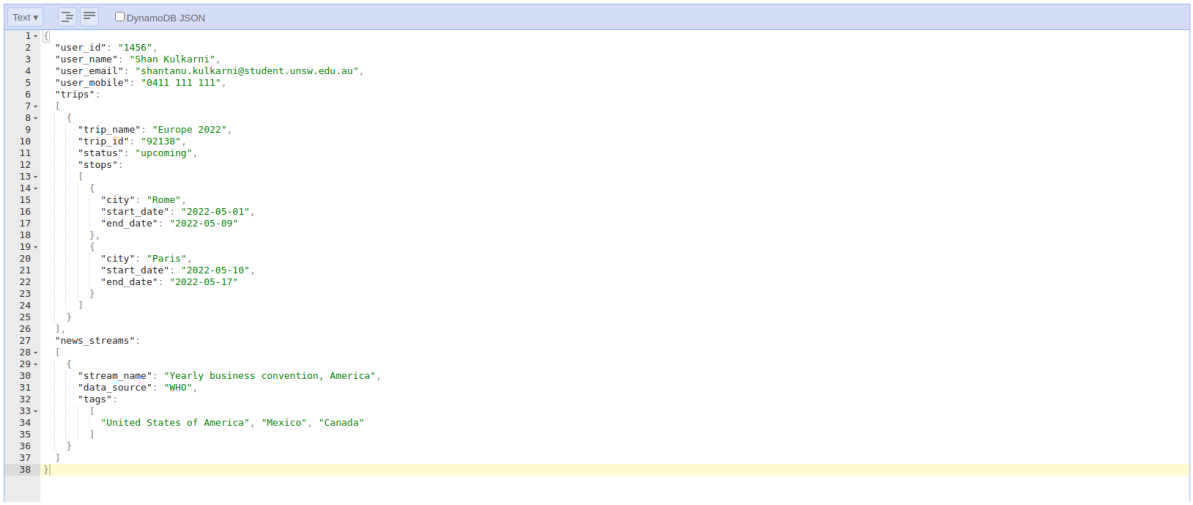


Figure 1.7 (above): An example object in the `Users` table.

Advantages

- Fully managed service - no setup/maintenance required
- No schema required
- No parsing of data into tables
- No knowledge of SQL required
- No complicated object/table relationships
- Allows for mostly stateless back-end

Disadvantages

- Inefficient querying

Challenges and findings

- After becoming familiarised with the `dynamodb()` resource for `boto3`, there are no further challenges.

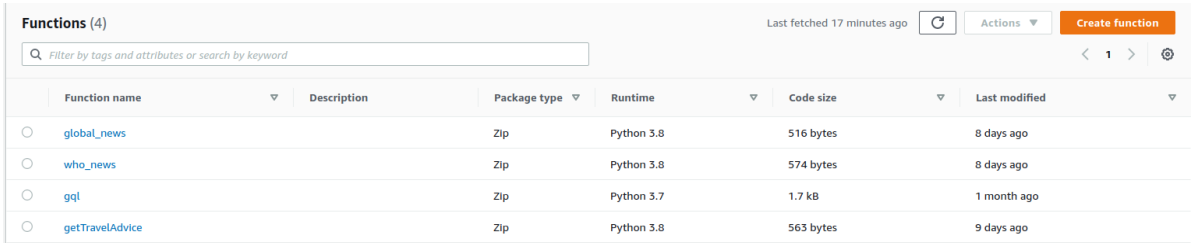
Compute + **Framework**

Choice

Once again the choice of both server-less and GraphQL has proven extremely useful. The ability to attach Lambda functions as resolvers to the AWS AppSync API made developing the core API functionality very easy. Queries to the API endpoint invoke several GraphQL queries, which each have a single Lambda function attached to it as a resolver (back-end).

Implementation details

There was three Lambda functions (ignoring gql from D2):



`global_news` which queries the `global_articles` DynamoDB table, `getTravelAdvice` which queries the `travel_advice` DynamoDB table and `who_news` which queries the `who_articles` table.

For each function; getting travel advice for trips, or getting articles, a new query in the AppSync GraphQL schema has been added.

```
type Articles {
    url: String!
    headline: String!
    main_text: String!
    reports: [Reports]!
    article_id: String!
    date_of_publication: String!
}

type Query {
    getArticles(period_of_interest: String, key_terms: String, location: String):
Response!
    getWhoArticles(countries: [String]!): WhoResponse!
    getGlobalArticles(param: String): WhoResponse!
    getTravelAdvice(countries: [String]!): TravelResponse!
}

type Reports {
    article_id: String!
    report_id: String!
    diseases: [String]!
    syndromes: [String]!
    locations: [String]!
    event_date: String!
}

type Response {
    body: [Articles]
    statusCode: Int!
    statusMessage: String!
}

type Schema {
    query: Query
    mutation: Mutation
}

type TravelAdvice {
    country: String!
    info: String!
    latest: String!
    summary: String!
    travel_advice_id: String!
}

type TravelResponse {
    body: [TravelAdvice]
    statusCode: Int!
    statusMessage: String!
}
```

```
type WhoReport {
    title: String!
    main_text: String!
    date_of_publication: String!
    article_preview: String!
    country: String!
    thumbnail: String!
}

type WhoResponse {
    body: [WhoReport]
    statusCode: Int!
    statusMessage: String!
}
```

Note the new types added for data objects such as `travel_advice`, matching the identical format as in the `travel_advice` DynamoDB table:

```
type TravelAdvice {
    country: String!
    info: String!
    latest: String!
    summary: String!
    travel_advice_id: String!
}
```

Each query `getWhoArticles`, `getGlobalArticles` and `getTravelAdvice` have the appropriate Lambda function attached as resolvers:

Query

Field	Resolver
getArticles(...): Response!	genReports
getWhoArticles(...): WhoResponse!	getWhoArticles
getGlobalArticles(...): WhoResponse!	genGlobalArticles
getTravelAdvice(...): TravelResponse!	genTravelAdvice

Also note the return type of the Lambda function mapping exactly to the `TravelResponse` type in the schema:

```
    return {
        'statusCode': 200,
        'body': travel_advice,
        'statusMessage': "OK"
    }
```

```
type TravelResponse {
    body: [TravelAdvice]
    statusCode: Int!
    statusMessage: String!
}
```

This ease of extensibility made adding new features extremely easy.

Below are sample queries for `getWhoArticles`, `getGlobalArticles` and `getTravelAdvice`:

```
{
  getWhoArticles(countries: ["India", "France"]) {
    body {
      article_preview
      country
      date_of_publication
      main_text
    }
  }
}
# A custom news stream with country tag "India" and "France"
```

```
{
  getTravelAdvice(countries: ["India", "Kenya"]) {
    statusCode
    statusMessage
    body {
      info
      latest
      summary
      travel_advice_id
    }
  }
}
# A trip consisting of a stop in an Indian city and a Kenyan city
```

```
{
  getGlobalArticles(param: "") {
    statusCode
    statusMessage
    body {
      article_preview
      country
      main_text
      date_of_publication
      thumbnail
    }
  }
}
```

```
# Getting global articles for home page
```

Note all article objects have a `thumbnail` attribute which contains a url of the news article's thumbnail image to be displayed as preview when viewed in the stream either on home page (global news) or News page (custom news).

Advantages

- Fully managed service - no setup/maintenance required
- No new endpoints - just one singular endpoints and more queries/types defined on that endpoint's schema
- No request forwarding to Lambda functions required, built-in when resolver is attached in AppSync
- Powerful querying

Disadvantages

- Lambda difficult to debug

Challenges and findings

The majority of challenges and findings surrounding AppSync as mentioned in D2 have since been surpassed, now that I have learnt how to operate, extend and manage the schema. That being said, Lambda proves to be difficult to use and annoying to debug. As detailed in the architecture diagram, Python scripts run offline to scrape from data sources into the relevant DynamoDB table. This is because using custom libraries such as `bs4` or `country_converter` (for converting country names into alpha-3 codes) cannot be imported in the Python3.8 Lambda environment without uploading a virtual environment. This is a long and tedious process, which involves creating a virtual environment in your local machine, writing all library dependencies (can be quite significant, especially for the above mentioned custom libraries given their `numpy` and `pandas` dependencies) to the `env/site-packages` . The entire working directory must then be zipped and uploaded on the Lambda GUI. To make things even more difficult, if your environment zip is greater than 10Mb in size (which is easy to do for almost any custom library) you *lose inline code editing on the Lambda GUI* which is extremely frustrating. This means for any change in the code, you must rezip the environment, upload the ~40Mb zip, and then re-test the function. For this reason, I decoupled the scrapers from Lambda, but in an ideal implementation they would run automated on an execution policy on some kind of dedicated compute (Lambda is perfect for this as it would take a simple execution policy but custom libraries must first be figured out).

Back-end

Choice

The programming language of choice remained as `python` given the timeframe for developing an MVP, and the simplicity of using it. The functionality of `boto3` is extremely powerful, and the performance tradeoff was not enough to warrant learning the AWs `sdk-for-go` in just 4 weeks. In an ideal implementation this would be Golang however.

Advantages

- Easy to develop in
- `boto3`

Disadvantages

- Comparatively slow to compiled languages

Challenges and findings

I really would have liked to have learnt `sdk-for-go` since whilst being strongly typed and compiled, Golang is still and enjoyable language to program in. Perhaps next time.

Front-end

Choice

JavaScript React was the front-end framework of choice, given Gaurung's knowledge and ability with it. He was also able to get his hands on a very useful ThemeForester front-end template which we were unfortunately unable to adapt in time for our application.

Advantages

- Known
- Cross-platform compatible
- CSS/HTML already given in template

Disadvantages

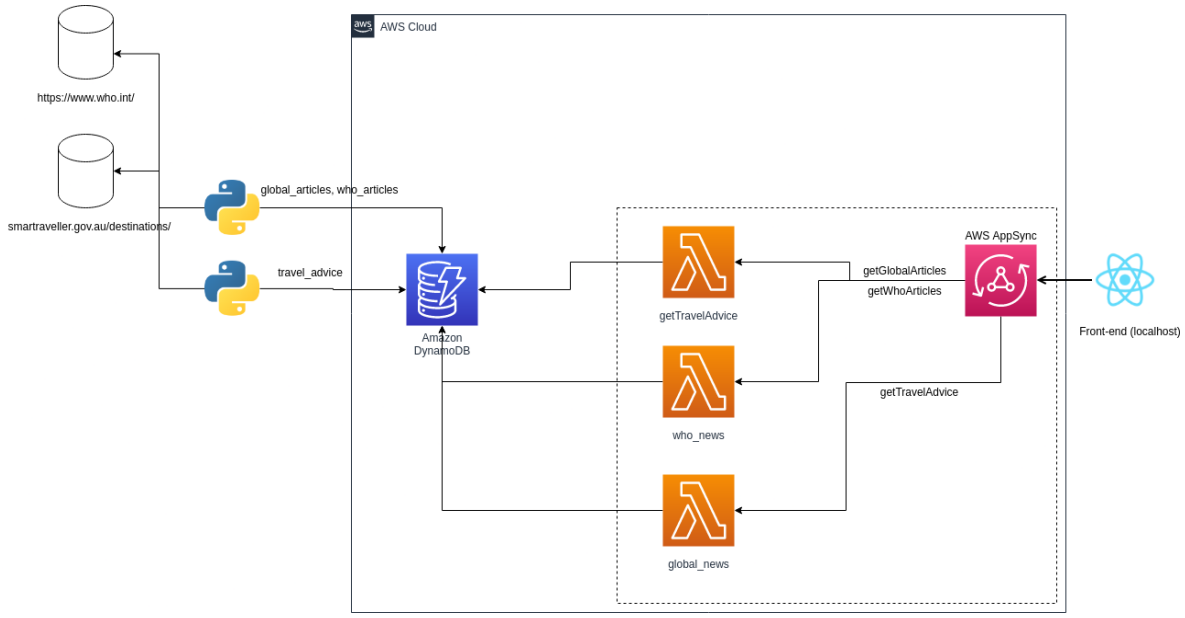
- Annoying template

Challenges and findings

The template's CSS components and React layer were tightly coupled, making modifying and extending difficult at times.

Final Architecture Diagram

Due to the front-end not being finalised, the ultimate deployment was ran on localhost. In the future, this would likely be a containerized application, running in either docker-hub or Kubernetes.



Summary

Given the incompetence of team members combined with time management which could have been better on mine and Gaurung's behalf, the application was not delivered as expected. That being said, I demoed a fully functional back-end for all discussed features in Canary, making it a useful travel-mate app for travellers wanting to plan their next overseas trip safely. The implementation benefits from a lightweight, loosely coupled micro-services and server-less framework, which allows for new features and easy extensilibity. The use of managed services like AppSync and DynamoDB also make for easier development. All in all, whilst falling short in the end, Canary is a workable MVP that with an extra few days could have been a fully functional web application.

Team organisation

Contribution and responsibilities

See email to LiC detailing this.

Reflections

Major Achievements

For Gaurung and I, learning GraphQL was a huge win. Going into this project, Gaurung had only used GraphQL endpoints in his work as a front-end developer, and I only knew the concept and overall idea of GraphQL. Going from that to actually implementing a GraphQL API from scratch was a huge achievement for myself and Gaurung, especially considering it serves as the major completed piece of functionality for Canary. Moreover, learning AppSync whilst a challenge was very rewarding, I would certainly use it again in future web-app projects.

What would be done differently

From an implementation standpoint, I would have maybe just copped the Golang learning curve earlier, as I really wanted to pick up `sdk-for-go` and also benefit from the added performance. Also, learning how to use AWS Amplify would have made connecting the front-end to the API easier. Given my extensive teaching duties and Thesis, as well as Gaurung's three jobs and coursework and Jordan's full-time job, we were all extremely strapped for time. That being said, Gaurung and I felt we perhaps could have managed our time better in between D3-4.

Note: I have disabled the API access keys and tokens since the repo is now public. As such, the functionality demonstrated in the final presentation cannot be replicated. This is for security reasons (when I made the report public the IAM policy was instantly Risk Quarantined since it could be used for malicious programmatic access to my AWS account).