Design Details

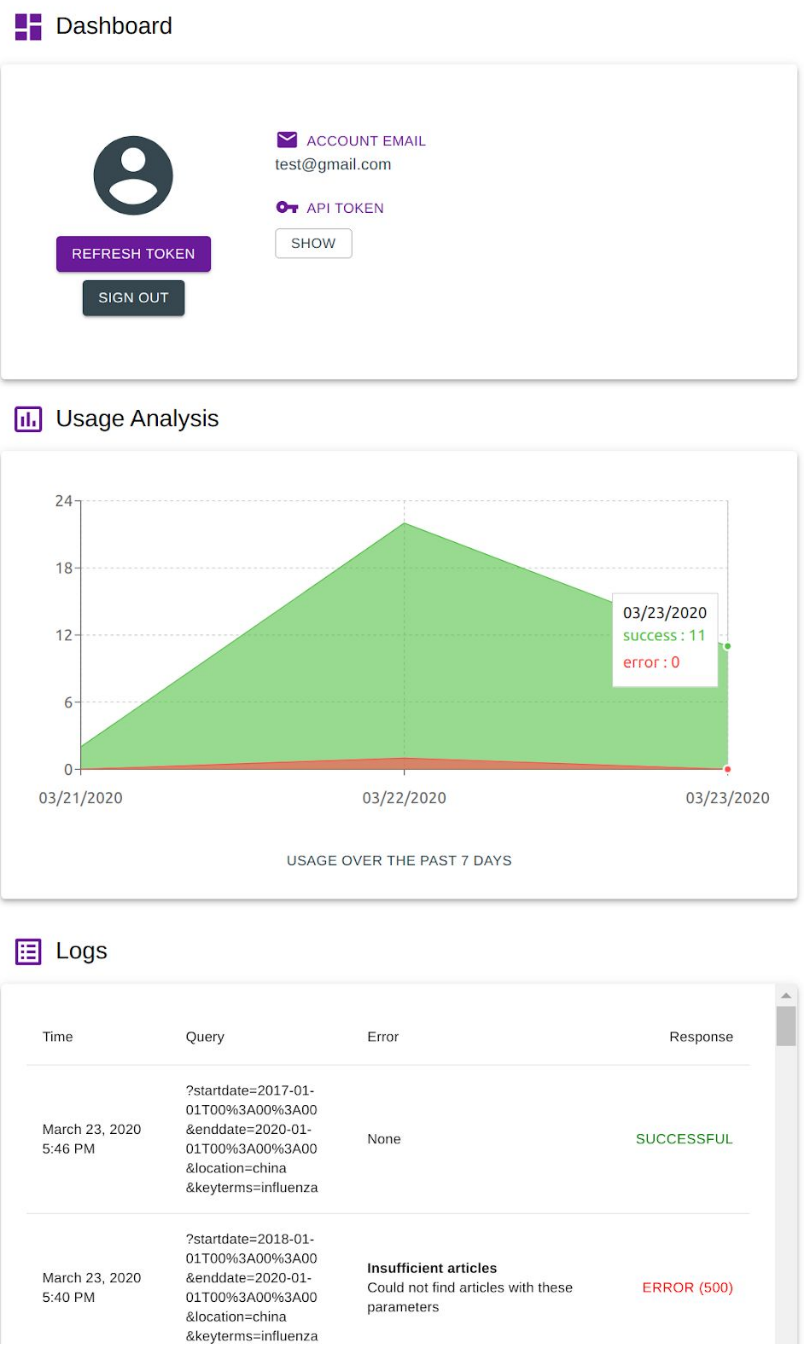
Design decisions were made on three considerations: the developer audience, resources and constraints and the business drivers. The goal for the API was to collate and retrieve disease reports which fit under the same search criteria. This was done so that searching for articles was easier and the user is able to filter out content which is irrelevant to them.

Beams API contains one GET endpoint which returns disease articles based on the date of publication, location and optional key terms. Example requests are discussed below. For developer information visit: http://sympt-swagger.herokuapp.com/docs/

API Usage

In order to use the Beams API, a developer must register online at https://symptdev.netlify.com/
Once registered, they can sign in and will be issued an API authorisation token which they can use to make requests by passing it in the header.

On the dashboard, a developer can expect to see their information as well as a log of their requests. Shown below is the dashboard of a user with email test@gmail.com.



ACCOUNT EMAIL
test@gmail.com

Time	Query	Error	Response
March 23, 2020 5:46 PM	?startdate=2017-01-01T00%3A00%3A00&enddate=2020-01-01T00%3A00%3A00&location=china&keyterms=influenza	None	SUCCESSFUL
March 23, 2020 5:40 PM	?startdate=2018-01-01T00%3A00%3A00&enddate=2020-01-01T00%3A00%3A00&location=china&keyterms=influenza	**Insufficient articles** Could not find articles with these parameters	ERROR (500)

API Module - Choice of Implementation

Beams utilised the following technologies to develop the API module:

- NodeJS
- ExpressJS
- Typescript
- Firestore
- Puppeteer

The backend was constructed in Express with NodeJs. Express is useful in creating server applications and is considerably simple, flexible and easy to use. Given the short time frame of the project, Beams decided to utilise NodeJS + Express to aid in creating the base of the backend in a shorter time frame. This allowed for more time to be dedicated to other opportunities such as improving scraping technologies and future web-app features.

Moreover, Beams developed in Typescript, a superset of the Javascript language which enforces strict data types. We utilised ts-node to compile our TS code into ES2020 compatible JS on-the-fly, thus allowing us to take advantage of the latest JS features and produce more compact, readable code. Strong typing also provided further rigidity in our code which helped us detect issues before run time, and, together with strict ESlint integration, allowed us to keep our codebase consistent with potential to scale.

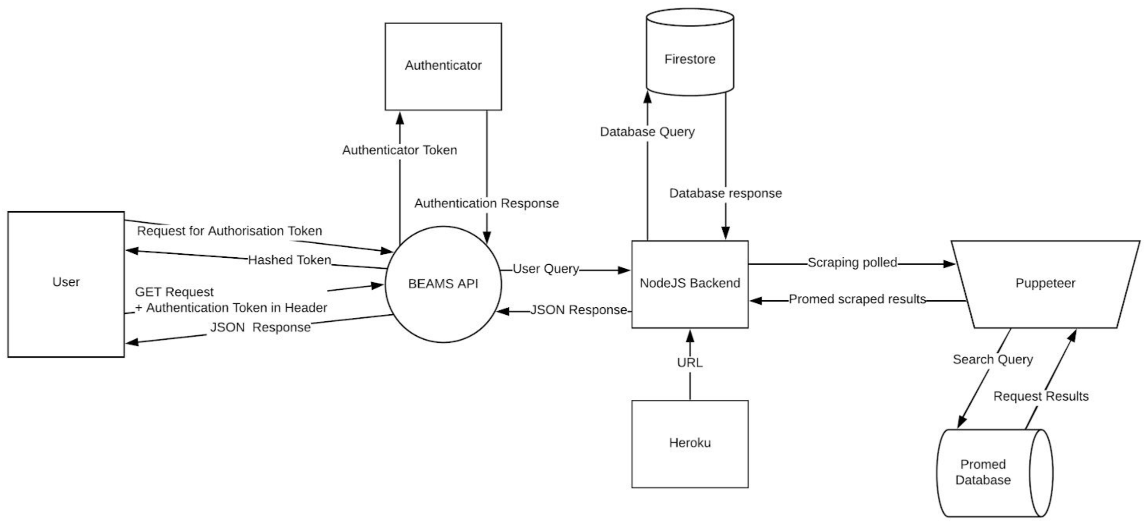
Firebase's Firestore database tool was used in conjunction with our web scraper. Results from the web scraper were formatted and stored in the Firestore database for fast retrieval by user API requests. Firestore's online interface opened opportunities to test data structures online, have multiple tenants interacting with the same data, and provided real-time updates to data changes both in-app and through our servers, bringing us opportunities to experiment and produce results rapidly.

Beams used Puppeteer, a Google Developer Tool, which instantiates a Headless Chrome component that allows developers to interact with the browser DOM without the GUI of Chrome. This allowed us to generate server-side rendered content to interact with the ProMed site, including entering terms into ProMed's search system and scraping relevant articles and post results. We can further process and parse this raw data, along with other data received from other sources, into JSON format and structure it as an accessible API endpoint.

Additionally, Beams integrated CircleCI into our GitHub repository to ensure our commits and pull requests abide by preset linting standards, and pass unit tests which will be developed overtime. CD methods are implemented through the heroku hosting platform.

In conjunction with CI, Beams used commit hooks with husky to ensure code is neat and passed tests before it enters the CI pipeline. We utilised Airbnb’s ESLint specification (https://github.com/airbnb/javascript) since it provides a rigorous guideline to follow and because it allowed us to adapt to industry-level code consistency and style. This ensured mistakes were caught before they were pushed into the master branch and saved time when there were several branches being tested and built through our CI pipelines.

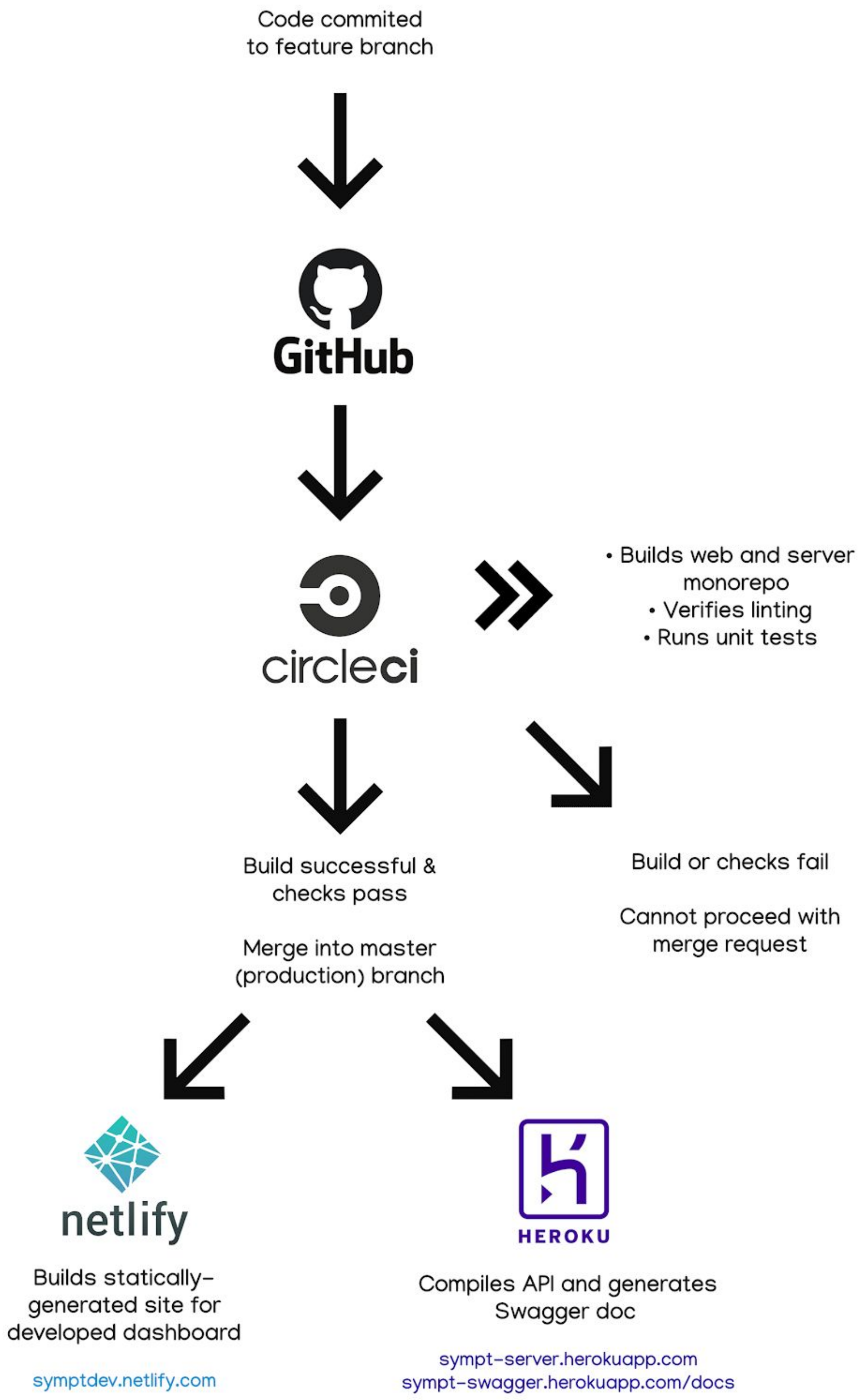
System Design Diagram



Deployment & DevOps

Using Heroku, the API is deployed at http://sympt-server.herokuapp.com/. The developer dashboard is hosted separately as a statically-generated site on Netlify. By utilising GitHub branch checks with CircleCI, commits are only merged into the master (i.e. production) branch if tests are passed to assure errors do not slip into the hosted site/server. These 3 platforms together create a strong DevOps ecosystem that automates vital processes, therefore speeding up development and code confidence significantly.

DevOps Cycle


Code commited to feature branch
GitHub
circleci
• Builds web and server monorepo
• Verifies linting
• Runs unit tests
Build successful & checks pass
Merge into master (production) branch
Build or checks fail
Cannot proceed with merge request
netlify
Builds statically-generated site for developed dashboard
symptdev.netlify.com
HEROKU
Compiles API and generates Swagger doc
sympt-server.herokuapp.com
sympt-swagger.herokuapp.com/docs

API Request Design

The API serves responses to users using the standard URL query method. The API collates and formats data scraped from https://promedmail.org/ using a web-scraper.

API Request Format:

```
http://sympt-server.herokuapp.com/articles/?startdate=2018-01-01T08:45:00&enddate=2018-01-01T08:45:00&location=somelocation&keyterms=firstterm[,secondterm,thirdterm…]&count=requests-per-page&page=page-no
```

An example request for searching for 5 articles on coronavirus and SARS during January, 2020 in China with an existing authorisation token passed in the header:

```
http://sympt-server.herokuapp.com/articles/?startdate=2020-01-01T00:00:00&enddate=2020-02-01T00:00:00&location=china&keyterms=coronavirus,sars&count=5&page=0
```

The API does not enforce strict URL parameters so that usage is flexible. However, the following conditions must be met:

- The start date and end date meet the format `yyyy-mm-dd` where:
  - yyyy is the year
  - mm is the month
  - dd is the day
- Start date is before end date

Any deviation from the above conditions will result in a malformed request response from the API. We provide dynamic error checking that can inform the API users of bad endpoint format, missing queries or unauthenticated access in the future.

The parameters are discussed below:

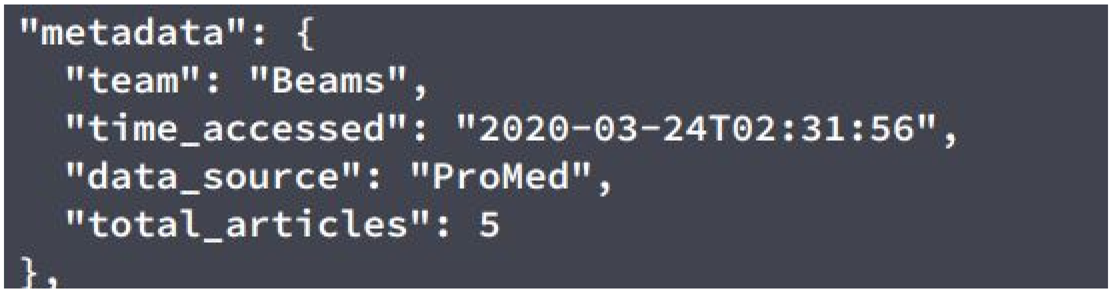
- The start date and end date also accept the format `yyyy-mm-ddThh:mm:ss` where:
  - `hh` is the hour
  - `mm` is the minutes
  - `ss` is the seconds
- Location is a string and is case insensitive

- Keyterms are a list of comma separated terms which must come from the disease_list.json file otherwise results returned are too vague
- Count is a number between 0 and 10. If set to 0 or not set the API will return all articles matching other search criteria
- Page is a positive number or 0. Page 0 displays the most recent articles with each page after going further back into the database. If the page is not set it becomes 0. Each page contains 10 articles.

API Response Design

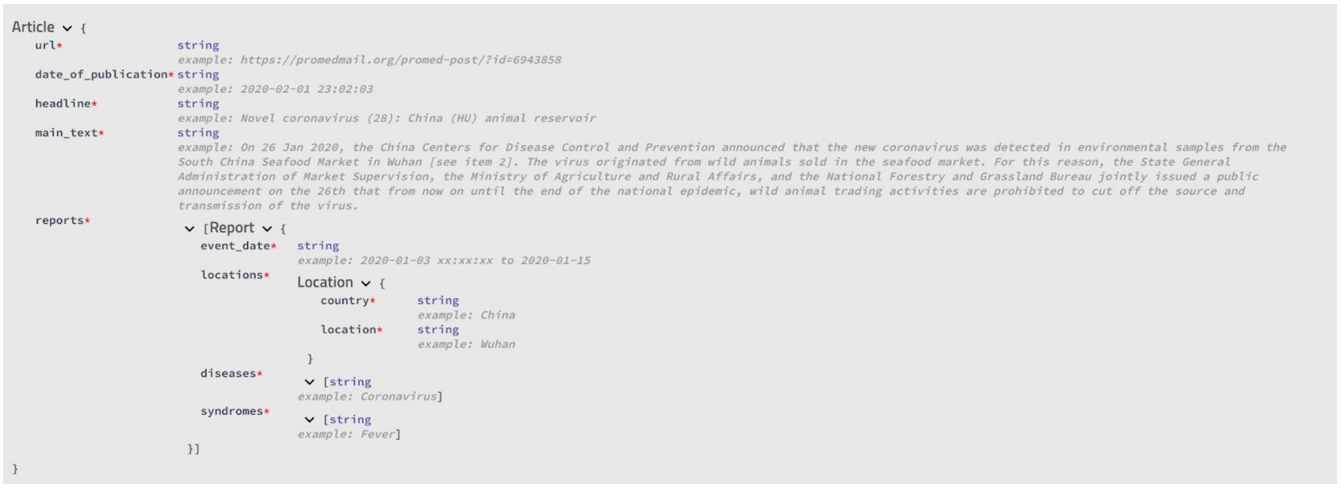
The API returns a JSON object with two items: metadata and articles.

- Metadata is an object, shown below.



```
"metadata": {
  "team": "Beams",
  "time_accessed": "2020-03-24T02:31:56",
  "data_source": "ProMed",
  "total_articles": 5
},
```

- Articles is an array of Article objects. Shown below.



```
Article ⌄ {
   url*                 string
                        example: https://promedmail.org/promed-post/?id=6943858
   date_of_publication* string
                        example: 2020-02-01 23:02:03
   headline*            string
                        example: Novel coronavirus (28): China (HU) animal reservoir
   main_text*           string
                        example: On 26 Jan 2020, the China Centers for Disease Control and Prevention announced that the new coronavirus was detected in environmental samples from the
                        South China Seafood Market in Wuhan [see item 2]. The virus originated from wild animals sold in the seafood market. For this reason, the State General
                        Administration of Market Supervision, the Ministry of Agriculture and Rural Affairs, and the National Forestry and Grassland Bureau jointly issued a public
                        announcement on the 26th that from now on until the end of the national epidemic, wild animal trading activities are prohibited to cut off the source and
                        transmission of the virus.
   reports*
                         ⌄ [Report ⌄ {
                            event_date*   string
                                          example: 2020-01-03 xx:xx:xx to 2020-01-15
                            locations*    Location ⌄ {
                                             country*     string
                                                          example: China
                                             location*    string
                                                          example: Wuhan
                                           }
                            diseases*
                                            ⌄ [string
                                          example: Coronavirus]
                            syndromes*
                                            ⌄ [string
                                          example: Fever]
                          }]
}
```

Challenges faced during development

These will be discussed below:

- Retrieval from ProMed and text processing for the user
- Limited resources
- Security

Usability Considerations

Scraping of the ProMed website was initially difficult as the website itself does not change URL when a new article is clicked, choosing to instead change a component on the page. This meant that accessing articles and their html source was difficult. To combat this, Beams used Puppeteer (as described above) and was able to successfully scrape the ProMed website based on the key terms and diseases which were given to us and upload them to our database.

In terms of processing the text received from ProMed, Beams found the articles to be lengthy and contain references to other articles and random chunks of text which did not provide value to the user. As such, we have processed the text and retrieved what we deem is the most important part of the article. This ensures the user is given nicely formatted text. Processing was done through regex and stripped the article of any bare bones html as well as random hyperlinks that were placed throughout and at the end of the article.

In the cases of many search results, the parameters count and page have been introduced. This ensures that users can access articles periodically and specify how many articles they wish to receive. This is done so that one request does not throttle the network or take too long to return a response.

Count: This parameter is optional and must be between 0 and 10. When specified, it will limit the amount of returned articles to its value. If count is larger than 10, a 403: bad request error will be returned.

Page: This parameter is optional. Pages start at 0 and contain 10 articles per page. If the requested page contains no articles, the user will receive a 500 error notifying them that the current page exceeds the amount of articles remaining.

The introduction of these parameters makes our API flexible and allows the developer to easily dictate which articles and how many they want.

Manageability Considerations

In order to use the API, a user must register on the API website (symptdev.netlify.com). This ensures that the API is being used by registered users, but also allows for users to manage and monitor the requests they are making. A developer dashboard is provided so that the user can see when and what request they made along with other information such as the response code and what errors were returned. Users can retrieve their API access tokens from here, and are also given the option to refresh their tokens in case their current one has expired or been leaked.

Security Considerations

As stated before, a user must register to use the API. This is done to monitor API usage and prevent DDoS attacks. Once registered, the user will be given an authentication token that they are required to pass in the header through with each request. This allows us to identify each user and maintain a record of the types of queries our users are making. In case our server or firebase service is attacked, we are able to disable firestore's read/write functionality to prevent data loss, and, after identifying the attacker, disable their account from access to the server.

Furthermore, we are able to utilise firebase's built-in authentication functionality to provide industry-level account management and security practices throughout the platform. Firebase can validate user IDs and generate JWTs that can be passed between client and server; the server can then validate these JWTs, verify who the request has come from, and perform actions securely.

The Beams team also distributes vital API keys in a private setting without external access. Keys are stored in environment variables and are ignored by git, so they are not unwillingly uploaded. Important actions are performed only through client-server interactions, so even if the client is compromised, the attacker will not have access to admin-level tasks.