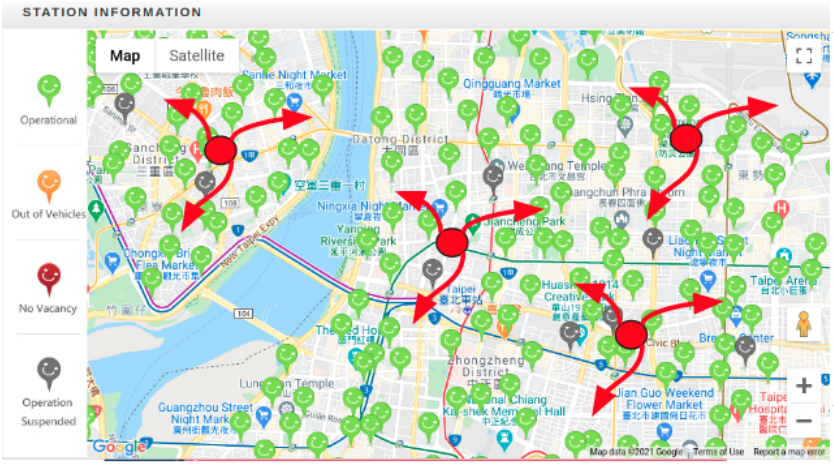
Final Project Report- Find YourBike

r09922102資工碩一 韓秉勳
r09942086電信碩一 湯正吉

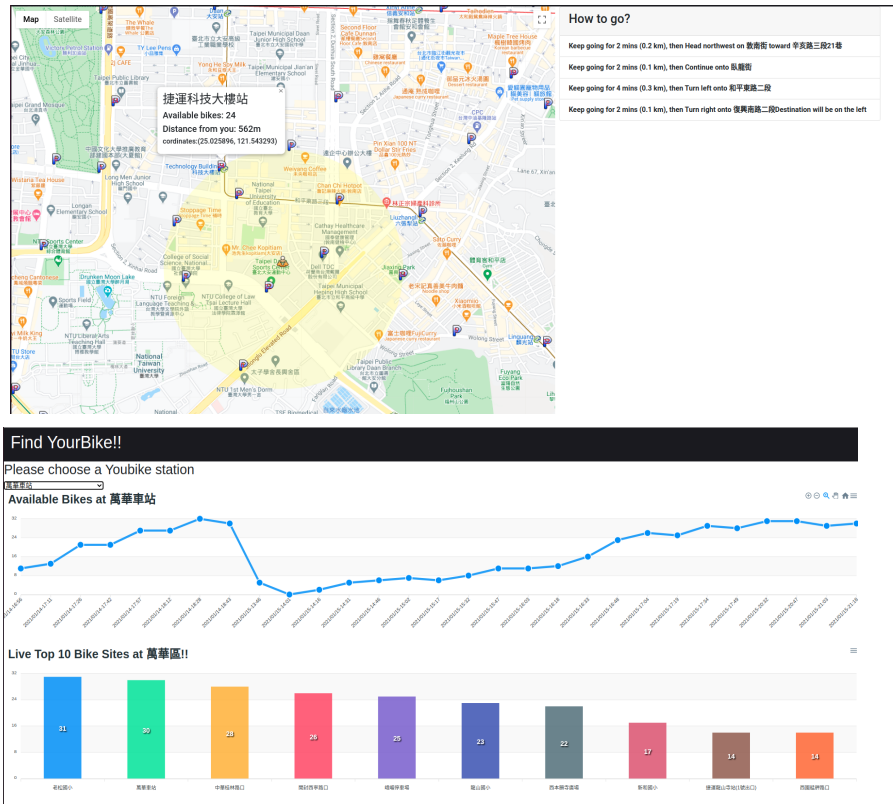
1. Project github URL - https://github.com/nba556677go/cloud_computing2020/tree/main/final
   Demo slides(Demo videos are in slides) - cloud security demo

2. Abstract
   a. Motivation
      The official Youbike website allows us to see vacancies in each station. However, it isn't user-centralized since it doesn't track users' location, while most people just want to figure out where the nearest bike is and where to go, the website doesn't support either. Therefore, we try to implement one by ourselves!



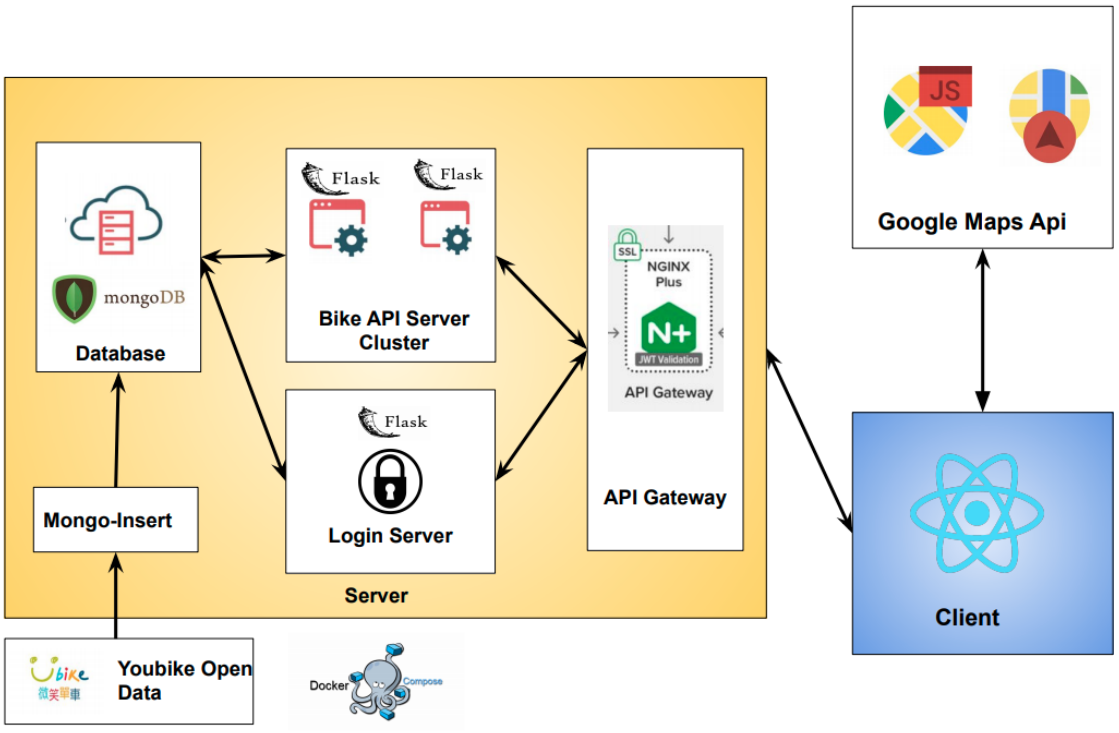
   b. Features
      We created a youbike query and navigation website based on docker, flask, mongodb, nginx, react, and google map api
      i. Creates membership of this website
      ii. Get Bike data from open government data
      iii. Shows the bike data geologically with Google map api
      iv. Displays nearby stations, provide user direction instructions
      v. Illustrates the availability of the bike in specified station

vi. Demonstrates the trend and popularity of stations



3. Approach

a. Architecture



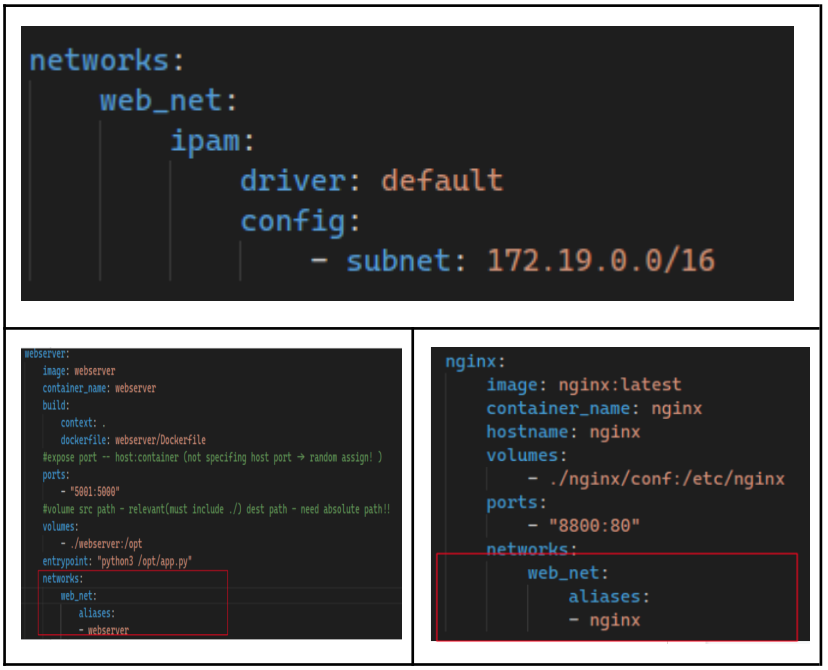
Our entire server works as our backend, while all backend services are constructed with Docker and orchestrated with Docker-compose. Dockerized environments allowed faster and portable deployment. As for client (frontend), we utilized React with node.js as our framework. client is not dockerized since client might not have docker supported. Google API is also connected to the frontend to visualize bike stations. We will explain each element in the following sections.

b. Backend

i. Docker & Docker-compose

Since I've played with docker before, I'll just introduce some tips when doing docker development that is not specified in class.

- Utilize docker-compose Embedded DNS server
  I managed all my services in the compose file, since it is not a cross-server service, compose is sufficient for us. We can define a docker network in compose network, and connect all services onto the network using aliases like this:



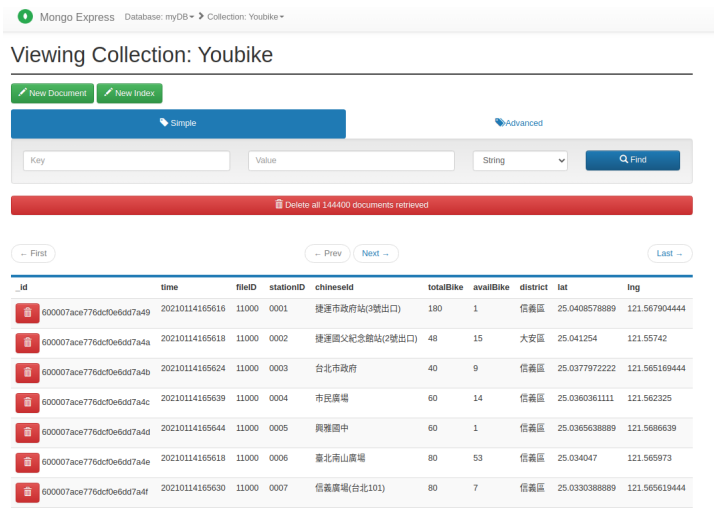
Then we can ping from webserver to nginx without specifying IP! Since docker dynamic assigns container IPs, it is recommended by docker tutorial to assign alias to containers, and embedded DNS will translate the correct IP for you

```
(base)vincent@vincent-ASUSPRO-D840MB-M840MB:~/.../cloud_computing2020/final ᛋmain*
 $ docker exec -it webserver ping nginx
PING nginx (172.19.0.3) 56(84) bytes of data.
64 bytes from nginx.final_web_net (172.19.0.3): icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from nginx.final_web_net (172.19.0.3): icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from nginx.final_web_net (172.19.0.3): icmp_seq=3 ttl=64 time=0.103 ms
64 bytes from nginx.final_web_net (172.19.0.3): icmp_seq=4 ttl=64 time=0.105 ms
```

- Use volumes
  Volumes not only persist data, but also update data concurrently when changing volume data at host. So we don't need to access containers every time when we are developing. Just volume the working directory to host and modify codes, then all the changes can persist. This is how I configured bike api servers and nginx.

ii. MongoDB & webUI

MongoDB is the official docker image, and mongo-express is also the official docker mongo Ui. So we can just deploy DB at ease. Only keep in mind that I set only one authentication user only, so I need to specify user and password. Also mongoDB data are persisted with volume. Bike data can be seen below:



_id	time	fileID	stationID	chineseId	totalBike	availBike	district	lat	lng
600007ace776dcf0e6dd7a49	20210114165616	11000	0001	捷運市政府站(3號出口)	180	1	信義區	25.0408578889	121.567904444
600007ace776dcf0e6dd7a4a	20210114165618	11000	0002	捷運國父紀念館站(2號出口)	48	15	大安區	25.041254	121.55742
600007ace776dcf0e6dd7a4b	20210114165624	11000	0003	台北市政府	40	9	信義區	25.0377972222	121.565169444
600007ace776dcf0e6dd7a4c	20210114165639	11000	0004	市民廣場	60	14	信義區	25.0360361111	121.562325
600007ace776dcf0e6dd7a4d	20210114165644	11000	0005	興雅國中	60	1	信義區	25.0365638889	121.5686639
600007ace776dcf0e6dd7a4e	20210114165618	11000	0006	臺北南山廣場	80	53	信義區	25.034047	121.565973
600007ace776dcf0e6dd7a4f	20210114165630	11000	0007	信義廣場(台北101)	80	7	信義區	25.0330388889	121.565619444

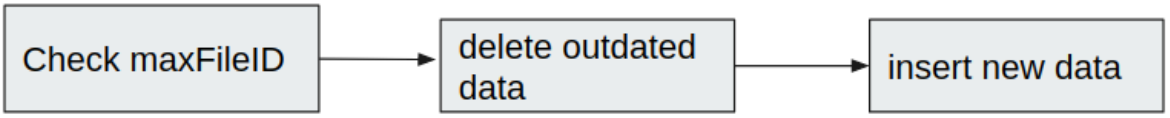
Stored data fields are as follows:

- time
- fileID - associate with time. same time means same fileID
- stationID
- chineseID
- totalBike

- availBike
- district
- lat - latitude
- lng = longtitude

iii. Mongo-insert

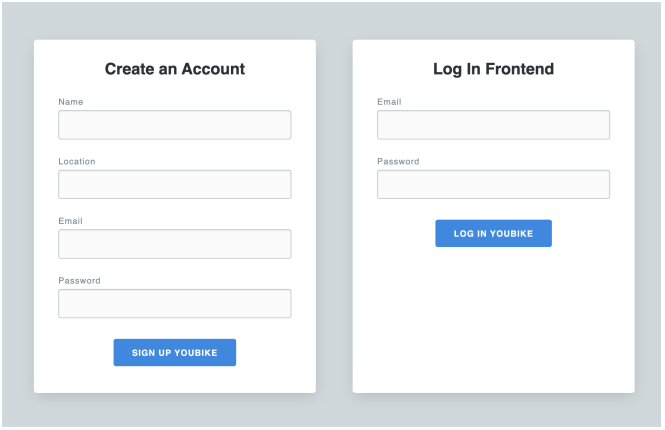
This container gets data from Youbike open data every minute, then checks if data has exceeded limitation (I set it to 360 timestamps) . If yes, delete outdated data and insert our new record. Procedure can be illustrated as below:



iv. Login Server

Here we use the Flask, Python and MongoDB to implement the server.

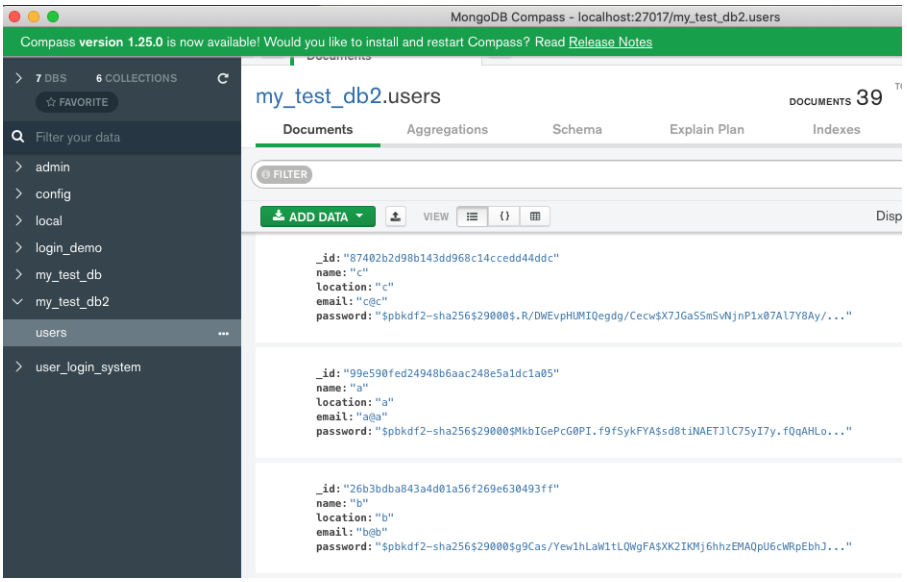
(Our log-in interface)



(some code in the log-in system, which is responsible for setting up a session)

```
def signout(self):
    session.clear() # clear the all information in session, so the l
    return redirect('/')

def login(self):
    user = db.users.find_one({
        "email" : request.form.get('email')
    })

    if user:
        return self.start_session(user)

    return jsonify({"error" : "Invalid login credentials"}), 401
```

(the database on local side)



v. Nginx API gateway

Nginx acts as the reverse proxy and API gateway for our entire service. Reverse proxy means that it can be a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. Therefore protecting backend service from DDOS attack and also deny malicious ip.

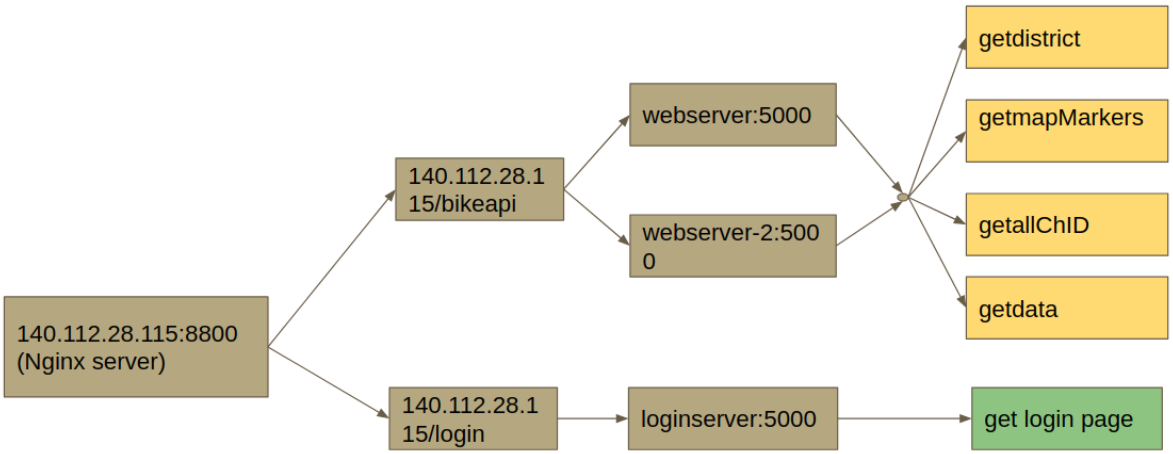
As for API gateway, as a reverse proxy, and also perform extra various cross-cutting tasks such as authentication, SSL termination, and load balancing. In our project, we use least connection as our balance method. Some configurations are shown below

```
upstream bikeapi{
 least_conn;
 server webserver:5000;
 server webserver-2:5000;
}
```
load balancing

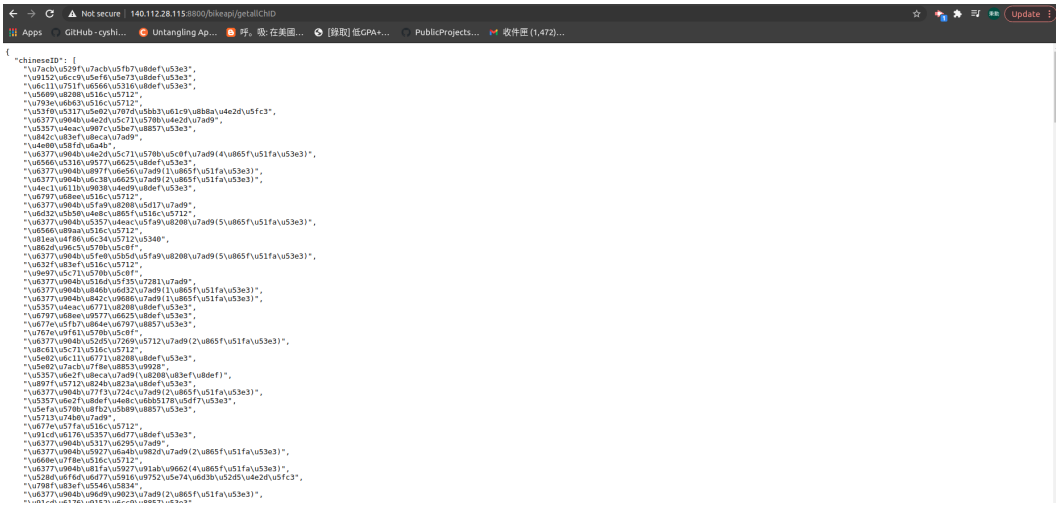
sites-enabled/bikeapi.conf

```
server {
 listen 80;
 error_log  /etc/nginx/logs/error.log;
 access_log /etc/nginx/logs/access.log;
 server_name   reverse.proxy;


 location / {
   root   /usr/share/nginx/html;
   index  index.html index.htm;
   limit_conn perclient 100;
 }
  location /bikeapi/ {
      proxy_pass http://bikeapi/;}}
```
limit total connections for DDOS

The image means that when we send a request to Nginx server, with a routing of **/bikeapi,** Nginx redirects traffic to one of the two bike API servers based on active connections on the two servers. With Nginx, the bike API server won't be visible to clients since the connection port is not published. It ensures our backend from being exposed to malicious attacks. Nginx routing can be shown as below:



This is the example of getting bikeapi/getChID response through nginx:



Also, we can volume logs to see if there are malicious ips. Image shown as follows.

vi. Bike API server cluster

Container Name : webserver, webserver-2

We scale the bike server up to 2 to allow more traffic. It is easy to scale since they share the same image, and Nginx will distribute traffic to each server.

We utilized python flask for our bike server, which is easy to implement. With pymongo module, we could easily access mongoDB. I defined another class called MongoAPI to maintain all data retrievals, so we can dynamically add on more functions. Here are current URLs that we use to get bike data:

- http://webserver:5000/getdistrict?district=信義區

    - get top-10 stations that have the most bikes now in 信義區.
- http://webserver:5000/getdata?id=龍山國小
    - get all availbike bike histories in station 龍山國小
- http://webserver:5000/getallChID
    - get all station chinese names. Used for bike selection list
- http://webserver:5000/getmapMarkers?latitude=25.021644799999997&&longtitude=121.5463424
    - calculate distance between current location(given latitude and longtitude) and every station. Retrieve geological data and mark top 10 nearest stations.

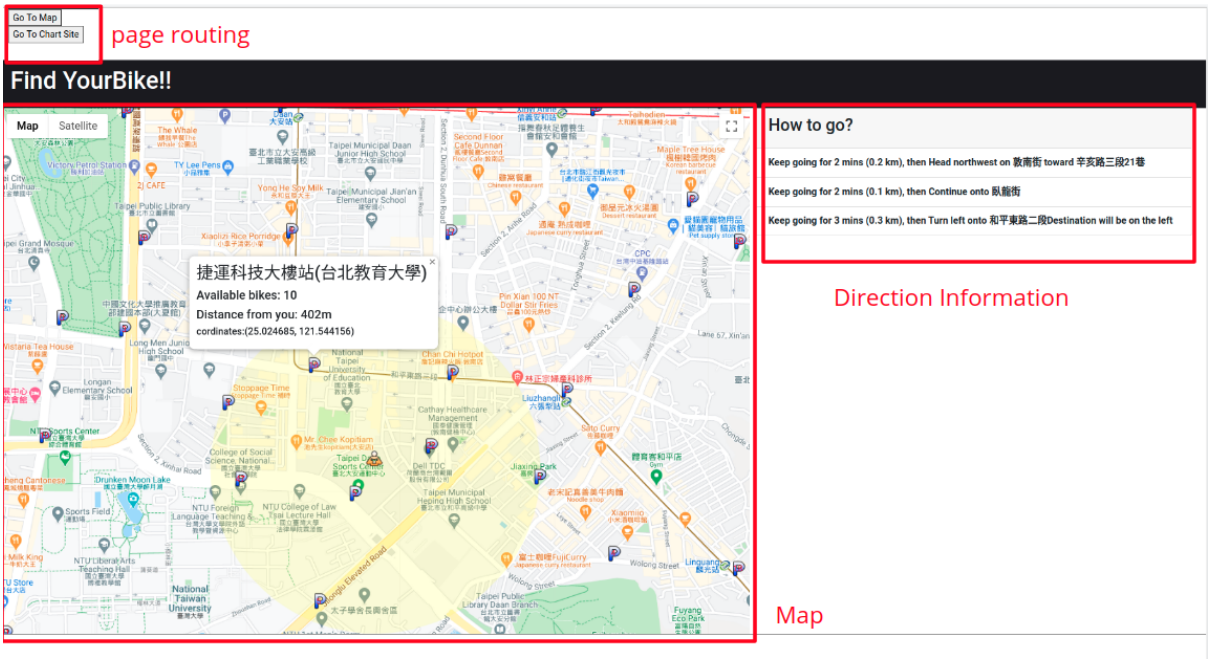
```python
@app.route('/getdata')
def getdata():
    station_id = request.args.get('id')
    #print(station_id)
    #print(type(station_id))
    mongo = MongoAPI(IP="mongo", DBname="myDB", collection="Youbike")
    data = mongo.queryDBdata(station_id)
    print(data)
    return data
#http://140.112.28.115:5000/getdistrict?district=信義區
@app.route('/getdistrict')
def getdistrict():
    district = request.args.get('district')
    mongo = MongoAPI(IP="mongo", DBname="myDB", collection="Youbike")
    data = mongo.queryDistrict(district)
    print(data)
    return data
```

c. Frontend

There are two pages in our frontend : map and chart. We will explain these two pages

i. Map page

We used Google Map API to display station distribution. There are three main components: page routing, Map, and Direction Information



- Page routing - switch from Map to Chart, and vise versa
- Map -

  I used google-maps-react npm module to implement the map. Here are the components:

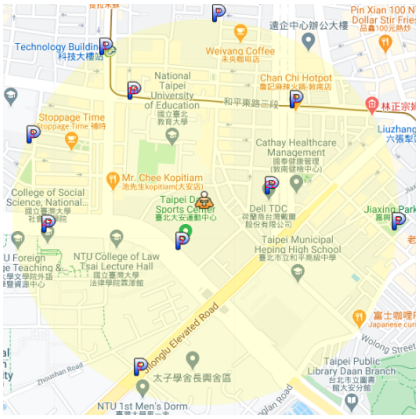
  a. Markers - get all bike station info and markers onto the map. We used the parking lot icon to demonstrate stations. The red number on the icon shows current available bikes at the station. show marker function is pasted below

```
showMarkers = () => {

  return this.state.stations.map((store, index) => {

    return <Marker icon={{ url: "./icons/parking_lot_maps.png", origin: {x: 0, y: 0}}}
      key={index} id={index} position={{
     lat: store.lat,
     lng: store.lng
   }}
   onClick={this.onMarkerClick} name={store.stationName} data={store} availBike={store.availBike} distance={store.distance} lat={store.lat} lng={store.lng}
   label={{text: String(store.availBike), color: "#F00", fontFamily: "Arial"}} labelOrigin={{0,0}}
    />
  })
}
```

  b. Nearby stations -

  The center yellow circle indicates our browser's current location, and the radius is set to the 10th nearest station near you. Since our backend has calculated and marked distance and near stations,

all the frontend has to do is to add marker information and set the radius. This helps users to track down the closest station possible.




```
<Circle
  radius={this.MaxDistance}
  center={{
    lat: this.state.liveLat,
    lng: this.state.liveLng
  }}

  strokeColor='transparent'
  strokeOpacity={0}
  strokeWeight={5}
  fillColor='#FFFF00'
  fillOpacity={0.2}
/>
```

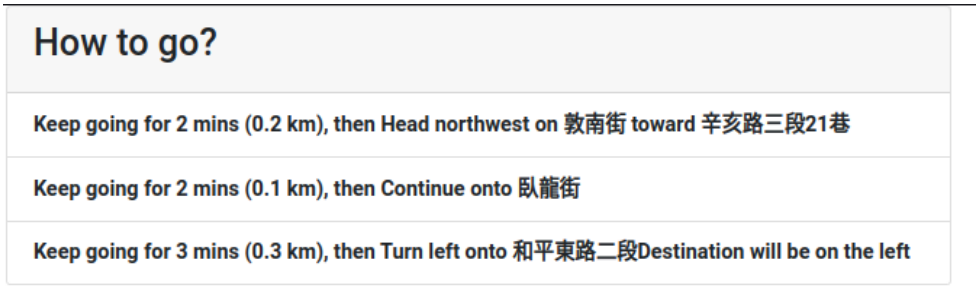
c. InfoWindow

When we click on a marker, station information will pop up, showing the name, available bikes, distance from you, and the coordinates. Users can check if the station is empty and decide where to go.

```
<InfoWindow
  marker={this.state.activeMarker}
  visible={this.state.showingInfoWindow}>
    <div>
      {/*{this.showInfo()}*/}
      <h3>{this.state.selectedPlace.name}</h3>

      <h5>Available bikes: {this.state.selectedPlace.availBike}</h5>
      <h5>Distance from you: {this.state.selectedPlace.distance}m</h5>
      <h6>cordinates:({this.state.selectedPlace.lat}, {this.state.selectedPlace.lng}) </h6>


    </div>
</InfoWindow>
```

d. Direction information

However, how to get to the station? We hereby add the navigation information. When a user clicks on a marker, our website will add direction instruction at the right of the page. This guides users to the destination they want without checking their google map app again!



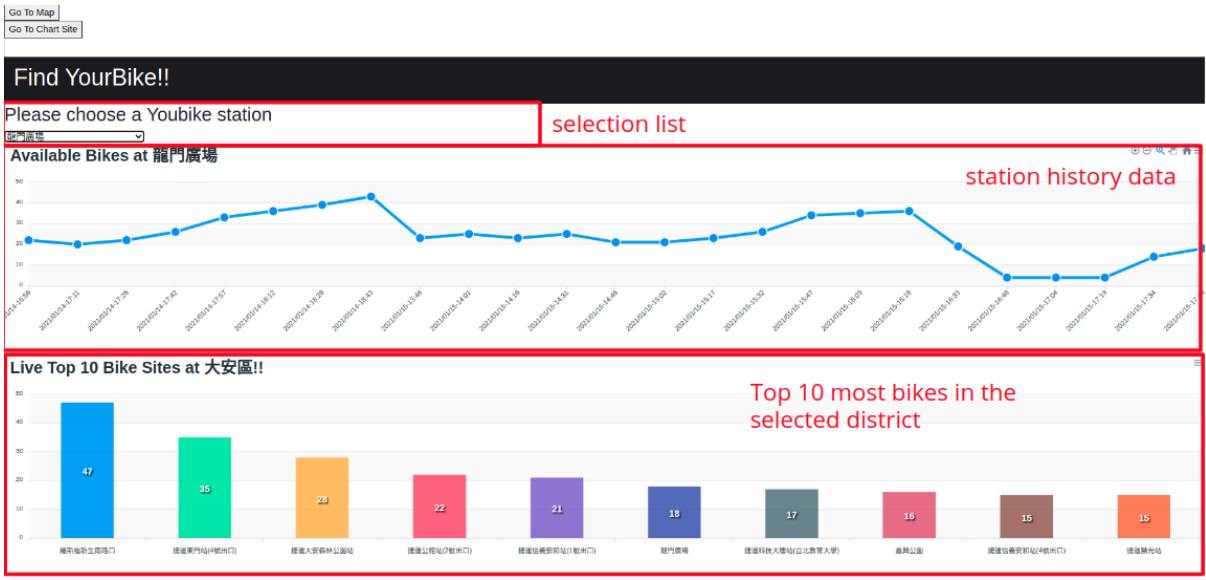
e. Live location tracking

Using js navigator.geolocation.getCurrentPosition allows us to get browser location. However, it requires https to retrieve this data. To do it simple, we used an insecure flag in chrome called **--unsafely-treat-insecure-origin-as-secure** to bypass https setup. Upgrade to https is also a future work.

ii. Chart page

Chart page is displayed for data analysis. We can find a routine in every station, since daily routine can be found. We can then make predictions about youbike station, and give recommendations to

users. This will be our future work. We used npm apex-chart as our chart display. (time stamp in image below is not continuous since I shutted down server for a while)

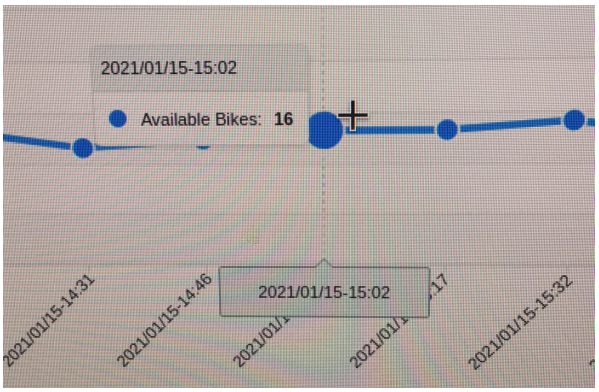


- selection list

  Allow users to click on the station the want to query

- station history data

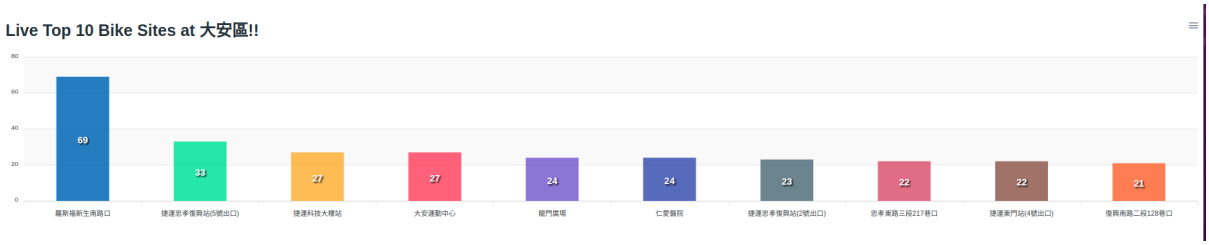
  shows available bikes every 15 minutes. hovering onto a data point can give us enlarged info.



- Top 10 most bikes in the selected district

  shows top 10 most bikes now in the selected district.

Usually bikes near MRT stations has the most bikes in average, but they can also dramatically be gone in minutes. Take 大安區 for example, 4 MRT bike stations occupies top 10 at 20 pm.



4. Results - Shown in Frontend section, also in demo slides [cloud security demo]

5. Discussion and Issues
    a. Login
    The most difficult is to merge the two different services (log-in service & bike service) into a complete one. we encountered several problems such as the server not responding, the routing problem and so on. When the server was not responding, it was because the js code somehow has previous memorization and can not be cleared up. To deal with this problem, we have to reopen the project every single time to remove the memorization. Finally, we redirected to the right page. Though the log-in is not the most important part of this web application, we do think the log-in system can be improved by adding OAuth such as Google Account/Facebook Account to let users easily use it. Moreover, because frontend is mainly using React with node.js, we believe it is better to use node.js all the time since merging these two services will be more seamless, while log-in has more interaction with frontend.

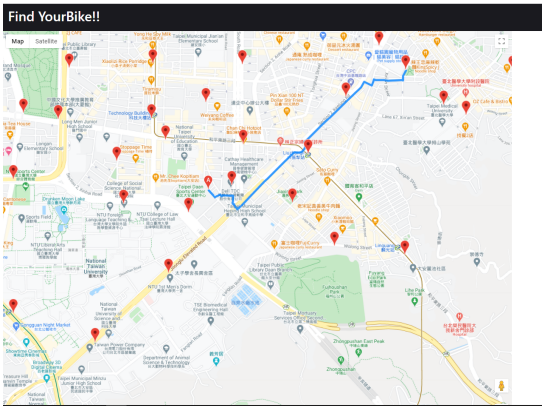
    b. React Map issues
        i. Map live location accuracy?
            - Since the live location is tracked with nearby cellular stations, it cannot be accurately measured. However, if we migrate to mobile apps, phone 4G network can have a more accurate tracking.

        ii. Will youbike2.0 stations be indistinguishable since they have great proximity?
            - It should be fine, since the latitude and longitude given in open data has an eighth place after decimal , and the

closest stations now have a difference in the third place after decimal, so it should be precise enough .

iii. Why navigation routes aren't shown on maps?

- This sounds simple, but is actually quite tricky since the module is open source. React npm modules for google maps tend to be complicated, and there are many variations. We tried multiple modules, but some of them can show routes without marker info, and some don't support navigation service rendering at all. The following picture is what we had successfully implemented with directions. However, the marker infowindow cannot be rendered, and it may due to multiple map instances rendering at the same time. Therefore in the end, we decided to demo the one without directions shown on maps, but all markers can work properly and also give direction information after clicks. This will be our future work.



c. Docker portability

i. Is docker really not platform dependent?

- If docker functions on every platform, then the answer is yes, while we build docker from docker hub, we can ensure our service can work on different platforms. However, docker is not fully functioning on mac, since MacOS blocks port publishing on host. There are some workarounds though, like this blog [Port forward not working on a macOS installation but works on others](#).

6. Conclusions

We created a youbike query and navigation website based on docker, flask, mongodb, nginx, react, and google api. Live locations are

detected and rendered on the site, so users can reach nearest stations by our navigation instruction. What we have are listed in below:

a. **portability**
    i. containerized docker environment, easy cloud migration
b. **scalability**
    i. **API server scaling** via docker compose
    ii. docker embedded **DNS server rerouting**
    iii. **Nginx load balancing**
c. **real-time data processing**
    i. mongoDB pipeline insertion & query on real time Youbike data
    ii. l**ive GPS location detection**
d. **security**
    i. API-based operation, **avoid DB injection**
    ii. **DDOS defense** - Nginx Reverse proxy & API gateway
    iii. **Login password SHA256 protection**
e. **Cloud-service**
    i. **Google Map js API**
    ii. **Google Map Directions API**

Future work includes upgrading to HTTPS, including Oauth login, and adding navigation routes on map.

7. Work distribution
    a. Login Service - (Frontend(page routing), MongoDB User Insertion, Login Server) - 湯正吉
    b. Bike Service and dockerization - (Frontend (Bike Data Chart + Google Map API), MongoDB & WebUI, Mongo-Insert, Nginx, Bike API server cluster) - 韓秉勳

References

A. Login with flask and python: https://www.youtube.com/watch?v=w1STSSumoVk
B. MongoDB with GUI: https://www.mongodb.com/try/download/compass
C. Login with React and node.js: https://www.youtube.com/watch?v=s1swJLYxLAA
D. React and node.js scr: https://github.com/keithweaver/MERN-boilerplate
E. flask and python src: https://github.com/LukePeters/user-login-system-assets
F. flask official website: https://flask.palletsprojects.com/en/1.1.x/
G. Youbike open datahttps://tcgbusfs.blob.core.windows.net/blobyoubike/YouBikeTP.json"
H. Creating Web APIs with Python and Flask https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask
I. Dockerizing a React App https://mherman.org/blog/dockerizing-a-react-app/
J. building-microservices-using-an-api-gateway with Nginx https://www.nginx.com/blog/building-microservices-using-an-api-gateway/

K. securing-api-ecosystem-nginx-controller-api-management-module/ https://www.nginx.com/blog/securing-api-ecosystem-nginx-controller-api-management-module/
L. Ngnix configuring-jwt-authentication https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-jwt-authentication/
M. Login with Flask and Python tutorial https://github.com/LukePeters/User-Login-System-Tutorial
N. MongoDB Documentation https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/