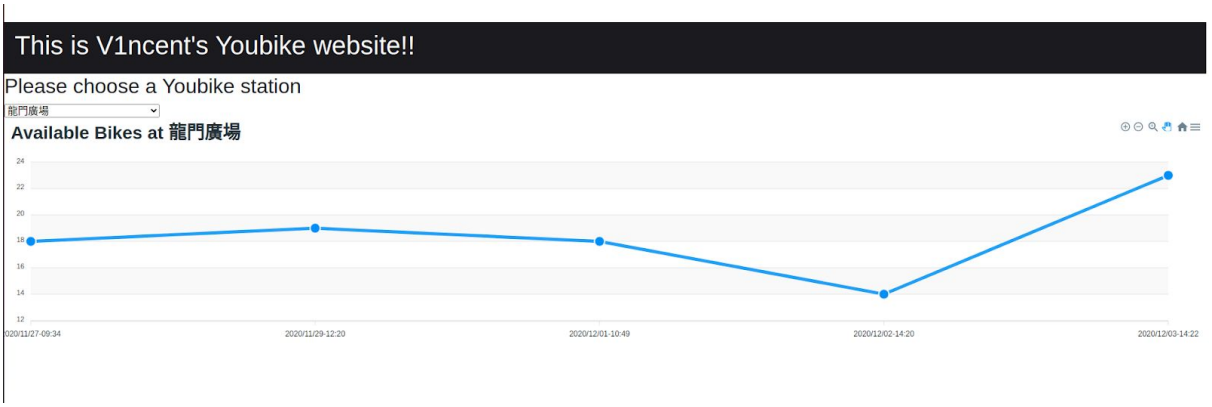
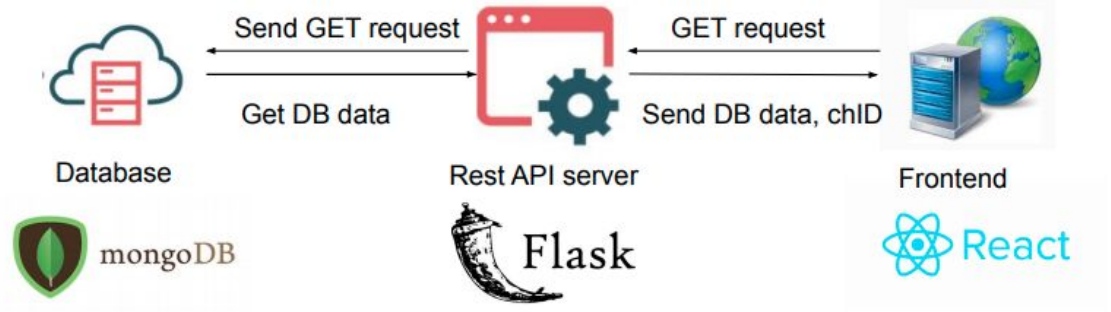
Cloud_Computing HW5

r09922102 資工所碩一 韓秉勳



1. Environment
   Docker engine on ubuntu20.04,
   RAM - 16G,
   CPU - Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
   Base image: ubuntu:latest, node:latest, mongo, mongo-express
2. Github link:
   https://github.com/nba556677go/cloud_computing2020/tree/main/hw5
3. installation & run guide: Check github README.md
   IMPORTANT: api URL in frontend/src/containers/myDBselect.js is set to localhost currently. It means you can only access frontend on your localhost. If you have static IP available, please replace localhost with your static IP instead!!!
4. design layout



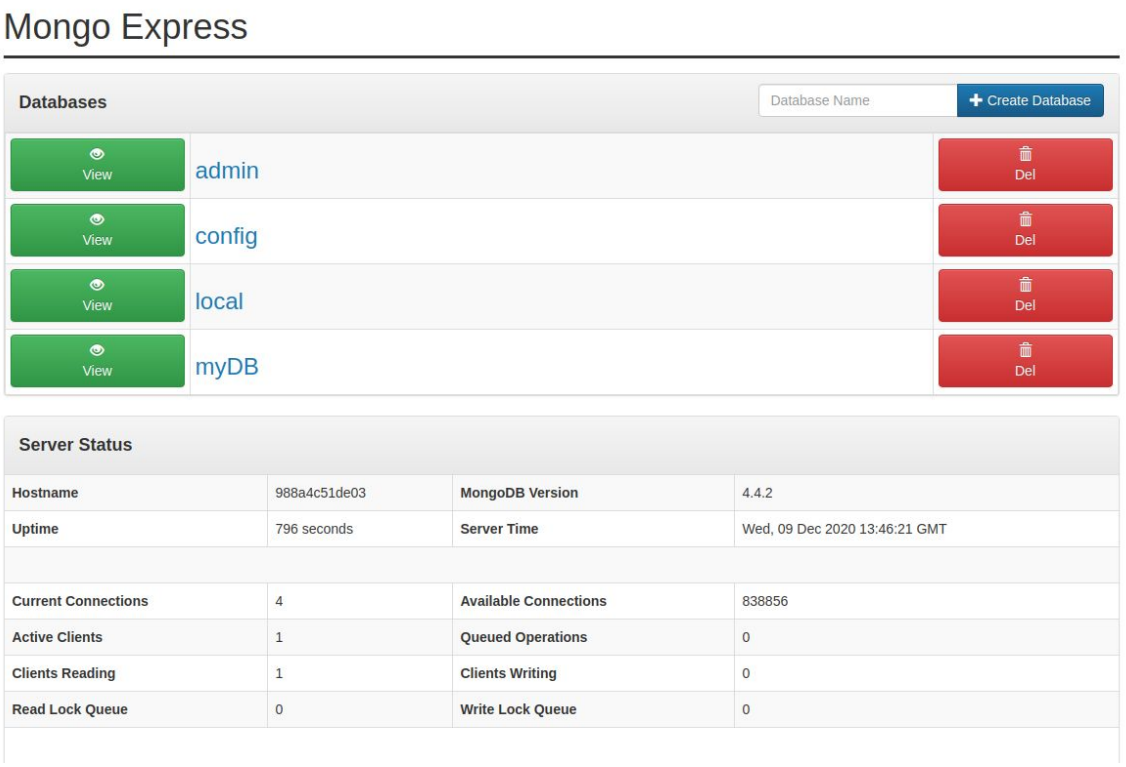
5. Design explain
   I used Youbike open data to get bike station info in Tapei city, and stored them in MongoDB. User can select a station, and this server will respond with a chart of available bikes at the station from 11/27 to 12/03
   a. mongoDB, mongexpress & data
      - data

    - I used Youbike open data as my input data, we can get json file from this url: https://tcgbusfs.blob.core.windows.net/blobyoubike/YouBikeTP.json
    - I stored 5 files during this 6 day period in host, and then I inserted the files into mongo db.
- MongoDB
    - Insert data
        - Insertion will be done right after the entire cluster is up. In init.sh, insert.py will do the insertion job for us. I've created a Mongdb in HW2, so the insertion process is quite the same. However, since we need to store and load all data in json format, I used dictionary to convert the required data. Code will be something like this:

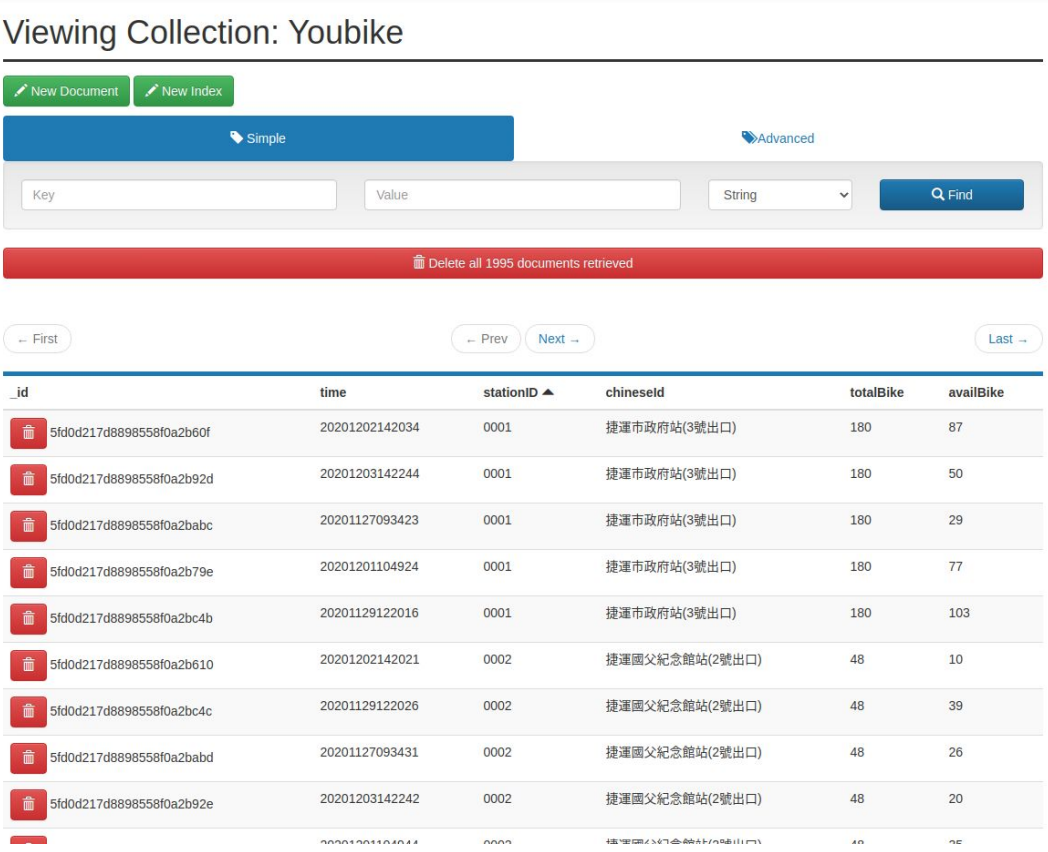
```
db = client["myDB"]
collection_Youbike = db['Youbike']
#argv1: folder
files  = os.listdir(sys.argv[1])

for i in files:
    with open(os.path.join('.', sys.argv[1], i)) as f:
        file_data = json.load(f)
        for k,v in file_data["retVal"].items():
            data = {
                "time" : v["mday"] ,
                "stationID" : v["sno"],
                "chineseId" : v["sna"],
                "totalBike" : v["tot"],
                "availBike" : v["sbi"]
            }
            #j_data = json.dumps(data)
            collection_Youbike.insert_one(data)
```

        - I've created 5 fields in this collection: time, stationID, chineseId, totalBike, and availBike. there are much more information that we can utilize in this dataset, but I'll leave that to final project
    - See data in mongo-express
        - I've also used mongo-express docker image to visualize our DB. user can type in ***localhost:8001*** to enter the site, it should look like this:



our youbike collection is under myDB, so we can click "view" in myDB, then we can see our youbike collection. click "view" once again, then we can see our collection as below:



b. Flask & API server

- Our docker image webserver represents our backend API server. I implemented two APIs:
    - getallChID
        - get all chinese station name in our mongoDB. This will be our selection list.Url looks like this:

    - getdata
        - Get all the available bikes of a certain station. This requires sending station chinese name as argument, so the entire url looks like this: *http://localhost:5000/getdata?id=龍山國小*

```python
@app.route('/getdata')
def querydata():
    station_id = request.args.get('id')
    #print(station_id)
    #print(type(station_id))
    mongo = MongoAPI(IP="172.18.0.3", DBname="myDB", collection="Youbike")
    data = mongo.queryDBdata(station_id)
    print(data)
    return data



@app.route('/test/<iid>')
def test(iid):
    print(iid)

    return {'Hello': 'World'}

@app.route('/getallChID')
def getallChID():
    mongo = MongoAPI(IP="172.18.0.3", DBname="myDB", collection="Youbike")
    data = mongo.getallChID()
    print(data)
    return data


if __name__ == "__main__":
    app.run(host='0.0.0.0', port = 5000, debug=True)
```

- To better managing APIs, I implemented class MongoAPI to maintain all the DBquery operations. For example, queryDBdata(ID) is called by the *http://localhost:5000/getdata?id=龍山國小* url, and it generates json format response. Code is like this:

```
def queryDBdata(self, ID):
    try:
        #db = self.client[DBname]
        #self.doc = db[collection]
        cursor = self.doc.find({"chineseId": ID})
        #list_cursor =  list(cursor)
    except Exception as error:
        print(error)
        sys.exit()
    #response schema
    response = {"stationID" : "",
                "chineseID" : "",
                "time": [],
                "totalBike" :"",
                "availBike" : []
                }

    for entry in cursor:
        response["stationID"] = entry["stationID"]
        response["chineseID"] = entry["chineseId"]
        #response["time"].append(entry["time"][:4]+'/'+entry["time"][4:6]+'/'+entry["time"][6:8]+
        response["totalBike"] = entry["totalBike"]
        #response["availBike"].append(entry["availBike"])
        response["availBike"].append({"time" : int(entry["time"]),
                                      "availBike" : entry["availBike"]})
    response["availBike"] = sorted(response["availBike"], key = lambda k : k["time"])
    response["time"] = [str(i["time"])[:4]+'/'+str(i["time"])[4:6]+'/'+str(i["time"])[6:8]+"-"+st
                        for i in response["availBike"]]

    #print(response)
    #encode chinese
    #encoded_data = codecs.encode(list_cursor)
    #json_data = json.dumps(response)
   # json_data = dumps(list_cursor)
    #print(len(self.data))
    return response
```

When I get mongo find() results, I sort them by time since data are not stored in chronicle order.

We can see the query result by typing in the query url, like the image below:

c. React & frontend - http://localhost:3000/

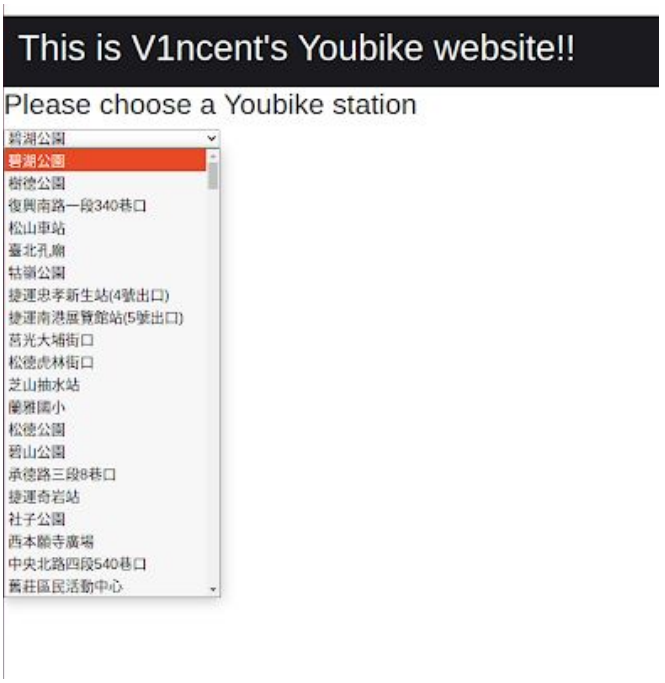
- I tried to build a frontend with React.js for the first time, and it is much more complicated than I thought. There are three major components in my website: Navbar, DBSelect, and Chart. Seems simple but it took me days to complete.
- Navbar.js - for displaying Navigation bar. I haven't done anything fancy aside from showing the title

```
import React from 'react'
import '../styles/NavBar.css'

export default () => {
    return (
        <div className="navbar">
            <h1>This is V1ncent's Youbike website!!</h1>
        </div>
    )
}
```

- MyDBselect.js

- implemented in MyDBselect.js. First, we need to let the select option display all station names before doing anything first.



- So we need to render our site after reading our station names. So in our React.component constructor, we setState right after call our getallchID API to render page:

```
class MyDBSelect extends React.Component {
  constructor(props) {
    super(props)
    this.state = { station: 'stationA', show: false, stationData: null, refresh:
    //set all stations
    this.stations = null

    //make sure total station list loaded first before rendering the first time
    axios({
      method: 'get',
      url: `http://localhost:5000/getallChID`
    })
    .then(response => this.stations = response.data.chineseID )
    .then(() => {  this.setState({ refresh: true }) })//nees to call setstate to
  }
```

- Next, when we hit our selection list, we need to enter another state that outputs our chart of available bikes. So I set a show flag in this.state. We enable our chart display only when hitting the select button. This will then create

the chart we are aiming for.

TADA! There's the graph we wanted.
The handleselect and render page part is listed below:

```
// fetch data from our data base
handleSelect = (e) => {
  axios({
    method: 'get',
    url: `http://localhost:5000/getdata?id=${e.target.value}`
  })
  .then(response => this.stationData = response.data )
  .then(() => { console.log(this.stationData) })
  .then(() => this.setState({ station: e.target.value, show: true }) )


}



render() {
  let display = null
  let selectStations = null

  if (this.state.show) {
    display = (
      <div>

        <MyChart info={this.stationData}/>

      </div>
    )
  }

  if (this.stations) {
    selectStations = (
      <select onChange={this.handleSelect}>
        {this.stations.map(s => (
          <option key={s} value={s}>{s}</option>
        ))}
      </select>
    )
  }

  return (
    <div>
      <h2>Please choose a Youbike station</h2>
      {selectStations}
        <div>{display}</div>

    </div>
  )
}
```

- MyChart.js
  The chart is implemented by importing apex-charts, a powerful extension for charts. I implemented the basic version, but it still provided some extra functions, Like showing data label and zooming in:

6. Discussions
   a. Future Work:
      I still have some functions to improve, including:
      - Security: My API URL is not private. Actual production API shouldn't show anything of actual IP address, which can be a security breach. I will try to implement Nginx to route the traffic to the frontend anonymously.
      - Realtime features ：User should get real time data in order to get actual bikes available now. This should be important and not hard to implement(by using multithread Timer in python)

      The above features will be implemented in the final project.
7. References
   a. Creating Web APIs with Python and Flask
      https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask
   b. Dockerizing a React App

      https://mherman.org/blog/dockerizing-a-react-app/

   c. Apex-charts https://apexcharts.com/docs/react-charts/