

實名制快篩地圖

Github: https://github.com/al5923647/realtime_antigen_test

0. 主要的使用方法在 demo 中介紹

1. Motivation

前不久根據網路上現有的快篩地圖去買快篩時發現每種版本的距離功能都採用兩點間直線距離計算，後來使用 google map 計算發現實際上的距離竟相差了兩倍之多。另外有些藥局沒提供快篩購買時段，因此很難判斷可購買的時間，導致去到現場卻撲空的窘境。

因此我想用這學期學到的資料庫知識配合一些 javascript 前端套件來寫一版可以看得到實際距離和特定藥局內快篩數量變化的快篩地圖。

2. Application description

實作台灣家用快篩試劑實名制地圖。預設釘選最近的 400 個販售點。

前端使用 leaflet, axios, bootstrap-table, bootstrap, chart.js 等套件實作，網站託管在 AWS 和 gh-page 上

(<https://twcovidtestkit.s3.amazonaws.com/index.html> or https://al5923647.github.io/realtime_antigen_test/)。

獲取資料的方式是使用 python 的 Flask 建立 API 與前端溝通。另外計算實際距離的部分本來是要用網路上的 ORS API，但由於網路傳輸速度和可用次數限制，後來是在主機上建個 ORS docker container 讓 Flask 直接和 ORS container 用 docker network 互相傳輸資料。

3. Data sources and how you collect and import the data (manually or automatically)

即時更新 CSV 檔來源:

https://data.nhi.gov.tw/resource/Nhi_Fst/Fstdata.csv

自動更新的部分和上次作業 bouns 差不多，但 CSV 檔是 30 秒更新一次，為了增加 query 效率，每次會更新三個 table: states(用來存所有紀錄)、new_states(用來存 7 天內的資料，用於建圖表)、store_latest(保留每間商店出現過最新的一筆紀錄，用於產生地圖上的 marker)。另外每過一天，我會計算目前所有商店內的快篩總和放進 date_ava 裡，這就能快速 query 到每天快篩供需變化，否則從 new_states 內 query 要花數分鐘。

自動更新的功能我是寫一個 python 檔，包進另一個 container 跑，但後來想用 AWS lambda，所以決定把它另外放在 AWS 上跑。

Execution result: succeeded ([logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
null
```

Summary

Code SHA-256	Request ID
pLgfvUznYG5XXUTXB65OVCFNOYGN7IW148ANNKM=	72829ff5-2e78-4809-8ac9-6be8afc6dd4b
Init duration	Duration
819.56 ms	7153.78 ms
Billed duration	Resources configured
7154 ms	2048 MB
Max memory used	
107 MB	

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
[INFO] 2022-06-22T14:37:39.213Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b SELECT * FROM information_schema.tables WHERE table_name='new_states'
[INFO] 2022-06-22T14:37:39.219Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b query completes, cost 0.803223896026611328 seconds
[INFO] 2022-06-22T14:37:39.687Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b Insert states into new_states starts
[INFO] 2022-06-22T14:37:39.687Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b 
[INFO] 2022-06-22T14:37:39.687Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b INSERT INTO new_states (選舉機構代碼, 地區項目, 候選號碼至目前補給存貨數量, 來源資料時間, 備註) VALUES ('
[INFO] 2022-06-22T14:37:40.678Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b Insert states into new_states completes, cost 0.990988001612854 seconds
[INFO] 2022-06-22T14:37:40.678Z    72829ff5-2e78-4809-8ac9-6be8afc6dd4b update complete @2022-06-22 14:37:40.678557, cost 6.953515529632568 seconds.
END RequestId: 72829ff5-2e78-4809-8ac9-6be8afc6dd4b
REPORT RequestId: 72829ff5-2e78-4809-8ac9-6be8afc6dd4b Duration: 7153.78 ms Billed Duration: 7154 ms Memory Size: 2048 MB Max Memory Used: 107 MB Init Duration: 819.56 ms
```

```

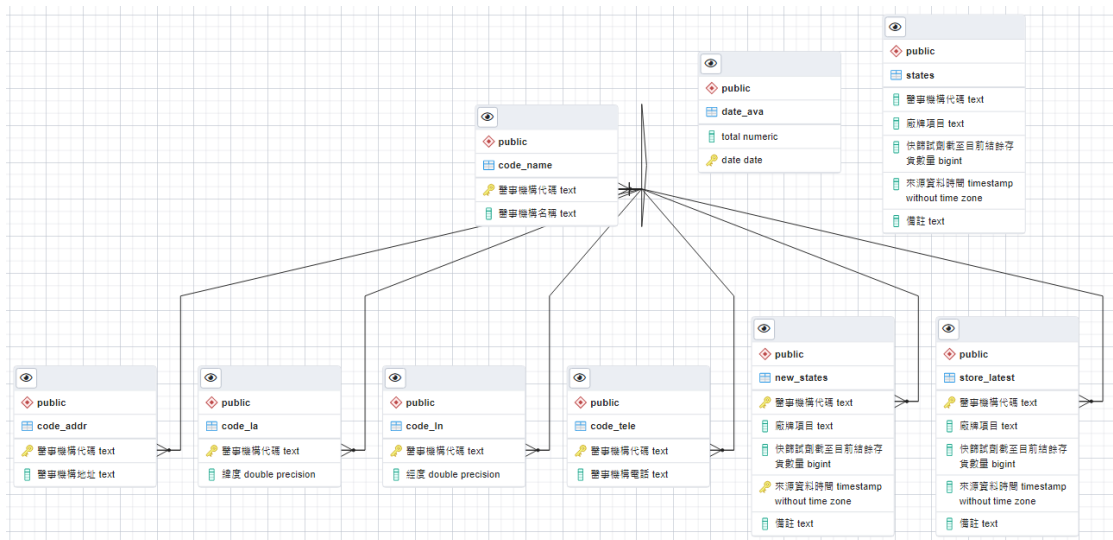
table_indices = {
    'code_addr' : [CODE],
    'code_col' : [CODE],
    'code_la' : [CODE],
    'code_ln' : [CODE],
    'code_name' : [CODE],
    'code_tele' : [CODE],
    'new_states' : [TIME, CODE],
    'store_latest' : [CODE],
    'date_ava' : ['date']
}

fks = {
    'code_addr' : [('code_name', CODE)],
    'code_col' : [('code_name', CODE)],
    'code_la' : [('code_name', CODE)],
    'code_ln' : [('code_name', CODE)],
    'code_name' : [('code_name', CODE)],
    'code_tele' : [('code_name', CODE)],
    'new_states' : [('code_name', CODE)],
    'store_latest' : [('code_name', CODE)]
}

pks = {
    'code_addr' : [CODE],
    'code_col' : [CODE],
    'code_la' : [CODE],
    'code_ln' : [CODE],
    'code_name' : [CODE],
    'code_tele' : [CODE],
    'new_states' : [TIME, CODE],
    'store_latest' : [CODE],
    'date_ava' : ['date']
}

```

Schema



states 很大而且要求要快速 insert，所以不用 primary key 或 index 以節省計算時間。

建立 table 的方式

```
def create_table(engine, table_name, src):
    if isinstance(src, pd.DataFrame):
        src.iloc[:,1,:].to_sql(table_name, engine, if_exists='append', index=False)
    elif isinstance(src, str):
        run_queries(engine, [src], f"create table {table_name} by SQL statement")
    else:
        return
    task_q = list()
    if table_name in pks.keys():
        task_q.append(f"ALTER TABLE {table_name} ADD PRIMARY KEY({', '.join(pks[table_name])})")
    if table_name in fks.keys():
        for ref_table, col in fks[table_name]:
            task_q.append(f"ALTER TABLE {table_name} ADD CONSTRAINT {table_name}_ref FOREIGN KEY ({col}) REFERENCES {ref_table} ({col})")
    if table_name in table_indices.keys():
        task_q.append(f"CREATE INDEX {table_name}_index ON {table_name} ({', '.join(table_indices[table_name])})")
    run_queries(engine, task_q, f"setup table {table_name}")
```

The application's functions and the related SQL queries used for the function.

重要檔案說明

index.html: 網頁的 html 檔

app.js: 網頁前端 JS

docker_compose/backend/app.py: flask python 檔

docker_compose/update/update.py: 更新資料用的 python 檔

query

```
def gen_all_latest_query(codes, memo=False):
    set_str = ','.join(f'c' for c in codes)
    return f"""
    SELECT {codes}, {STOCK}, {TIME} {', ' + MEMO if memo else ''}
    FROM store_latest
    WHERE {CODE} IN ({set_str})
    """

def gen_latest_query(code, memo=False):
    return f"""
    SELECT {CODE}, {STOCK}, {TIME} {', ' + MEMO if memo else ''}
    FROM store_latest
    WHERE {CODE} = '{code}'
    """

def gen_nearby_query(la, ln, hdist, limit=10):
    """
    hdist is in kilometer
    """
    return f"""
    SELECT {CODE}, {NAME}, {ADDR}, {LA}, {LN}, ( 6371 * acos( cos( radians({la}) ) * cos( radians( {LA} ) ) + cos( radians( {LN} ) - radians({ln}) ) + sin( radians({la}) ) * sin(radians({LA})) ) ) AS distance
    FROM code_la
    natural join code_ln
    natural join code_addr
    natural join code_name
    WHERE ( 6371 * acos( cos( radians({la}) ) * cos( radians( {LA} ) ) + cos( radians( {LN} ) - radians({ln}) ) + sin( radians({la}) ) * sin(radians({LA})) ) ) <= {hdist}
    ORDER BY distance
    LIMIT {limit};"""

def gen_interval_query(code, interval):
    other_constraints = []
    if 'HOUR' in interval.upper():
        other_constraints.append(f"EXTRACT(SECOND FROM {TIME}) < 30")
    elif 'DAY' in interval.upper():
        other_constraints.append(f"EXTRACT(SECOND FROM {TIME}) < 30")
        other_constraints.append(f"EXTRACT(MINUTE FROM {TIME}) = 0")
        other_constraints.append(f"EXTRACT(HOUR FROM {TIME}) = 0")
    other_constraints_s = " and ".join(other_constraints)
    if other_constraints:
        other_constraints_s = "and " + other_constraints_s
    return f"""
    set timezone='Asia/Taipei';
    SELECT {STOCK}, {TIME}
    FROM new_states
    WHERE {CODE} = '{code}'
    and
    ({TIME} BETWEEN NOW() - interval '{interval}' AND NOW())
    {other_constraints_s}
    ORDER BY {TIME}
    """
```

```
@app.route('/date_summary', methods=['GET'])
def date_summary():
    q = "SELECT * FROM date_ava"
    ret_response = Response(json.dumps(run_query(q, conn), default=str))
    ret_response.headers['Access-Control-Allow-Origin'] = "*"
    return ret_response
```

● update

update store info

```
def insert_non_exists(table_name, df, conn):
    def double2single_quote(s):
        if s.endswith("'") and s.startswith("'"):
            s = s[1:-1] + "''"
        return s
    vals_s = ['(' + ', '.join([double2single_quote(json.dumps(row[field], ensure_ascii=False)) for field in table_cols[table_name]]) + ')'] for index, row in df.iterrows()]
    q = f"""
    INSERT INTO {table_name} ({', '.join(table_cols[table_name])}) VALUES {', '.join(vals_s)} ON CONFLICT ({CODE}) DO NOTHING;
    """
    #print(q)
    #logger.info(q)
    with conn.cursor() as cur:
        cur.execute(q)
    conn.commit()
```

update store_latest, states, new_states

```

def update(engine, table_names=["states", "new_states"], latest_table="store_latest"):
    start_time = time.time()
    logger.info(f"update starts.")
    tables = fetch_data()
    task_q = list()
    for table_name in ['code_name', 'code_addr', 'code_la', 'code_ln', 'code_tele']:
        q = gen_insert_non_exists_q(engine, table_name, tables[table_name])
        task_q.append(q)
    run_queries(engine, task_q, "insert non exists")

    df = tables['new_states']
    df.to_sql('temp_table', engine, if_exists='replace', index=False)
    if not table_exists(engine, latest_table):
        create_table(engine, latest_table, df)
    update_latest_q = f"""
    INSERT INTO {latest_table}
    (select * from temp_table)
    ON CONFLICT ({CODE}) DO UPDATE
    SET {STOCK} = excluded.{STOCK},
        {TIME} = excluded.{TIME},
        {BRAND} = excluded.{BRAND},
        {MEMO} = excluded.{MEMO};"""
    run_queries(engine, [update_latest_q], f"update latest table({latest_table})")

    df.to_sql("states", engine, if_exists='append', index=False)
    q = gen_insert_non_exists_q(engine, "new_states", df, conflict_cond=[CODE, TIME])
    run_queries(engine, [q], f"insert states into new_states")

    if cross_day(engine):
        daily_update(engine)
    logger.info(f"update complete @ {datetime.datetime.now()}, cost {time.time()-start_time} seconds.")

```

update date_ava

```

def cross_day(engine, latest_update_table='date_ava', time_col='date'):
    if not table_exists(engine, "date_ava"):
        create_q = f"""
        CREATE TABLE date_ava AS (
            WITH d as (SELECT {TIME}, {STOCK}
                        FROM new_states
                        WHERE EXTRACT(hour FROM {TIME}) = 0
                        AND EXTRACT(minute FROM {TIME}) = 0
                        AND EXTRACT(second FROM {TIME}) < 30)
            SELECT sum({STOCK}) as total, min({TIME})::date as date
            FROM d
            GROUP BY EXTRACT(day FROM {TIME})
        )
        """
        create_table(engine, "date_ava", create_q)
    q = f"SELECT max({time_col})::date < NOW()::date FROM {latest_update_table}"
    res = run_queries(engine, [q], "check cross day")[0][0]
    return True if res or res is None else False

def daily_update(engine):
    task_q = list()
    task_q.append(f"""INSERT INTO date_ava ({', '.join(table_cols['date_ava'])})
    SELECT SUM({STOCK}) as total, MAX({TIME})::date as date
    FROM store_latest""")
    print(task_q)
    run_queries(engine, task_q, "update date_ava")

```

remove outdated rows from new_states

```

def remove_obsolete(engine, tables=["new_states"], interval='7 days'):
    task_q = list()
    for table_name in tables:
        q = f"DELETE FROM {table_name} WHERE {TIME} < (NOW() - interval '{interval}');"
        task_q.append(q)
        # reserve only one record per minute
        q = f"DELETE FROM {table_name} WHERE {TIME} < (NOW() - interval '1 day') AND EXTRACT(SECOND FROM {TIME}) >= 30;"
        task_q.append(q)
    run_queries(engine, task_q, "remove")


```

5. 心得

透過這次作業學到了一些 Javascript 套件的使用方法，配合之前在 SA 學到的一些系統知識從無到有把一個應用搭出來，做完簡直成就感爆棚。

但 Learner Lab 的限制真的多。想用 command line 上傳 code 到 S3，結果發現 Learner Lab 不給 Key、試著用 EC2 跑 Docker 只能跑數個小時就會被自動殺掉、API Gateway 也不給用，最後想到的也就只有把前端的網頁放在 S3 和把更新的部分從 docker-compose 拆出來用 Lambda 跑才能用到除了 RDS 以外的 function。

Trigger configuration

 **API Gateway**
api application-services aws serverless

✗

User: arn:aws:sts::757042450848:assumed-role/voclabs/user1867898=a15923647@gmail.com is not authorized to perform: apigateway:GET on resource: arn:aws:apigateway:us-east-1::restapis because no identity-based policy allows the apigateway:GET action (Service: AmazonApiGateway; Status Code: 403; Error Code: AccessDeniedException; Request ID: e7d356ed-bc2f-440c-9d30-89763e7025f1; Proxy: null)

✗

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

API

Create a new API or attach an existing one.

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

API Gateway 不給用

```
swag      http validation is selected
swag      Certificate exists; parameters unchanged; starting nginx
swag      [cont-init.d] 50-config: exited 0.
swag      [cont-init.d] 60-renew: executing...
swag      The cert does not expire within the next day. Letting the cron script handle the renewal attempts overnight (2:08am).
swag      [cont-init.d] 60-renew: exited 0.
swag      [cont-init.d] 70-templates: executing...
swag      [cont-init.d] 70-templates: exited 0.
swag      [cont-init.d] 90-custom-folders: executing...
swag      [cont-init.d] 90-custom-folders: exited 0.
swag      [cont-init.d] 99-custom-files: executing...
swag      [custom-init] no custom files found exiting...
swag      [cont-init.d] 99-custom-files: exited 0.
swag      [cont-init.d] done.
swag      [services.d] starting services
swag      [services.d] done.
swag      Server ready
realtime_antigen_backend | 172.18.0.3 - - [23/Jun/2022 17:06:50] "GET / HTTP/1.1" 200 -
realtime_antigen_backend | 172.18.0.3 - - [23/Jun/2022 17:06:50] "GET / HTTP/1.1" 200 -
realtime_antigen_backend | 172.18.0.3 - - [23/Jun/2022 17:06:55] "GET / HTTP/1.1" 200 -
realtime_antigen_backend | 172.18.0.3 - - [23/Jun/2022 17:07:02] "GET / HTTP/1.1" 200 -
osrm_backend exited with code 0

[1]+  Stopped                  sudo docker compose up
[ec2-user@ip-172-31-90-115 docker_compose]$ bg
[1]+ sudo docker compose up &
[ec2-user@ip-172-31-90-115 docker_compose]$
```

使用 EC2 跑所有服務