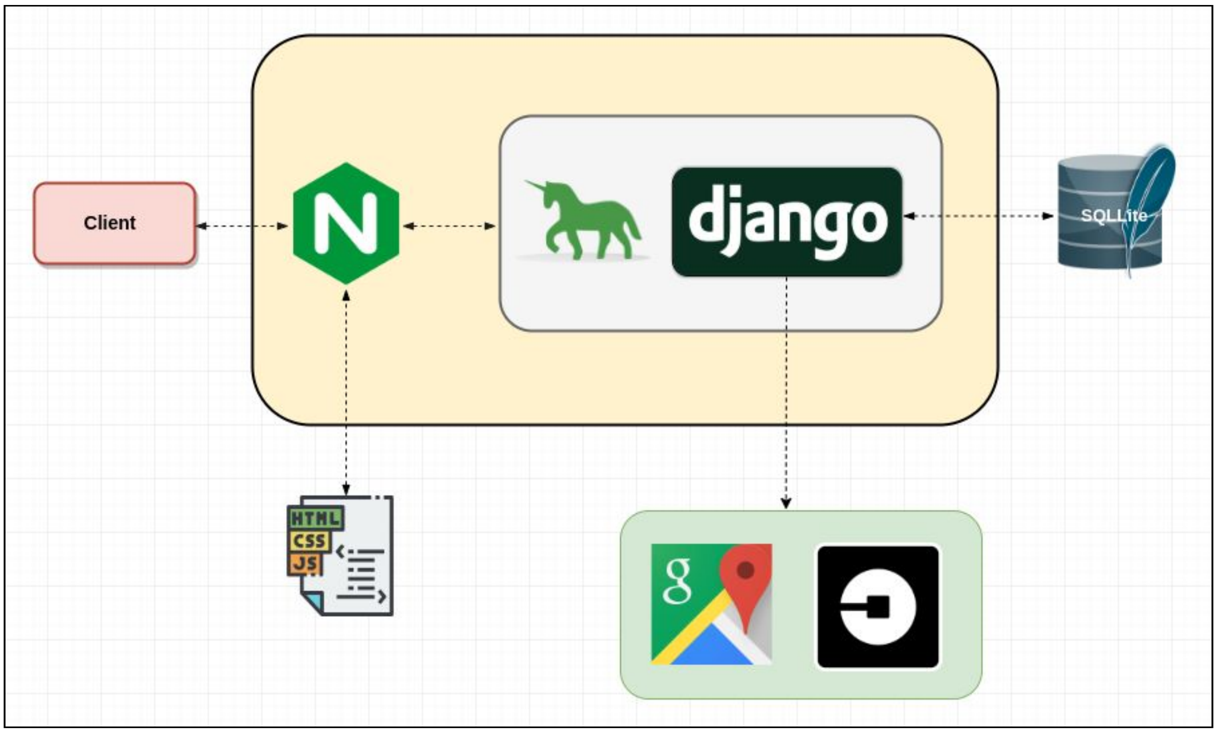
Schedule Uber Rides

Problem Statement:
Design a web application to schedule uber rides and notify a user to book an uber by sending an email at an appropriate time so that he can reach his destination on time.

Architecture Diagram:



Approach

Storage Mechanism

Database: SQLite Database
Table: *scheduled_trip*
Columns:

Name	Datatype	Purpose
id	Integer	Primary Key
source	String	Consists of latitude and longitude
destination	String	Consists of latitude and longitude
time_to_reach_dest	Datetime	Time to reach the destination
email	String	An email address to reach the user

create_time	Datetime	Time at which ride requested
ride_time	Integer	The ideal time to reach the destination without any traffic (seconds)
frequency	Integer	Indicates how frequently the service will check the maps API for this particular record. Here are the different frequency levels: 0 - This indicates a new ride request and it has to be checked at least once 1 - This is an indication to check the maps API only if pending time i.e. time remaining to start the ride is less the *pending_time / 2*. 2 - This indicates that the ride time is continuously increasing and so we'll have to check the maps API on each execution.
pending_time	Integer	Time remaining to start the ride (seconds)
traffic_time	Integer	Time to complete the ride with traffic (seconds)
state	Integer	Status of the ride 0 - The ride record is inserted in the DB but the service has to still find out the optimal time to notify the user 1 - This indicates that the service has notified the user

Service Logic

When a user requests a ride (adds *source*, *destination* and the *time* to reach the destination), this record is added in the database with the *ride_time* value as the ideal time to reach the destination from the source at the moment without any traffic considerations. The *traffic_time* is also updated with the same value as the *ride_time*.

The default values of the *pending_time* and *frequency* are 0 which indicates that this record has not yet been picked up by the service.

The scheduler service (an API) runs every 10minutes to check if it needs to send a notification for a request. We'll fetch only those records from the database for which:

Condition 1:

- The *time to reach the destination* minus current time is less than *ride_time* (the ideal time to reach the destination) plus 60 mins (Max variation possible to reach the destination) plus possible time to get the uber assuming it to be 20 mins
- and the value in the *frequency* column for this record is *zero* (a new request which has not been checked before).

*time_to_reach_dest - current_time < ride_time + 60 * 60 + 20 * 60 and frequency = 0*

Condition 2:

- *frequency = 2* -- The time to reach the source from the destination is continuously increasing. These are called *hot records* and we'll check them every 10mins. But let's say after some time, the ride time for these records becomes constant, then they'll be evaluated as per the 3rd condition.

Condition 3:

- *frequency = 1* -- The time to reach the destination is more or less constant and the time to reach the destination minus the current time is less than the *pending_time* that is stored in the previous check.

  time_to_reach_dest - current_time < pending_time / 2 and frequency = 1

Uber API is fired only if *pending_time* is less than or equal to the time that we assumed earlier (20mins to get the uber cab) and the error_margin is 5mins.

pending_time <= UberTime (20mins) + ErrorMargin (5 Mins)

Let's take an example to understand the above scenario:

A user has to reach the destination at 10 pm and it takes an hour to reach there from the source. Ideally, the cab should be booked at 9 pm. There is a maximum variance of 60 mins which means that if the user books a cab at 8:00 pm, he'll reach his destination on time no matter the circumstances. So, in the worst-case scenario, the cab starts at 8 pm (60 mins ride time + 60 mins variance time). I'm assuming a buffer time of 20 mins here. The service will fetch this particular record at 7:40 pm (8 pm - 20mins = 7:40 pm) from the database. Please note that the frequency of this record would be 0 as this is the first time we're evaluating its ride time.

Now, we'll call the Google Maps API at 7:40 pm to check the time it takes to reach the destination at this moment. Let's say the time that it takes to reach the destination now as per the google maps API is 80 mins. This time (80 mins) will then be updated in the *traffic_time* column in the database and the *frequency* of this record would be modified to 2. The *pending_time* column will be updated to 60 mins. As mentioned above, the Google maps API says it takes 80 mins to reach the destination if we book the cab at 7:40 pm. We have a buffer time of 60mins here! (7:40 to 10 pm is 140mins and it takes only 80 mins to reach as per the API and that's how the pending time comes to 60mins; 140 - 80 = 60).

As the frequency of this record is 2 (hot record), the service will again fetch this record in the next 10 mins iteration at 7:50 pm and repeat the process of calculating the time it takes to reach the destination. If the ride time comes near to 80 mins again, the service will update the frequency of this record to 1. This record is no longer a hot record and it will not be picked up in the next 10mins cycle. This record will be picked up from the database using the following logic:

Time to reach the destination - current time < pending_time / 2 + traffic_time
10:00 - 8:00 = 120mins
Checking if 120 < 80 + (60 / 2) => false
Since the above condition is false, this record will not be picked from the database at 8:00 pm.

Now at 8:10 cycle,
10:00 - 8:10 = 110 mins
Checking if 110 < 80 + (60 / 2) => false
Since the above condition is false, this record will again not be picked from the database at 8:10 pm

Same happens at 8:20 mins

Let's check at 8:30 pm
10:00 - 8:30 = 90 min
Checking if 90 < 80 + (60 / 2) => true
This record will now be picked up from the database and we'll fire the Google Maps API to check the time to reach the destination at this moment.

If we get the *ride_time* again as 80 mins, this calculation boils down as:
8:30 + 80 mins = 9:50 (this comes under the assumption margin of 20 mins so the service will book the cab now).

But if the *ride_time* is lesser than 80 mins and it is taking 60 mins now,
pending_time now will be 30 mins (8:30 + 60 = 9:30 and it is a difference of 30 mins from the actual destination time). This 30 mins time is greater than the assumed margin time of 20 mins so the *traffic_time* will be updated to 60 mins and the *pending_time* will be updated to 30 mins.

This record will then be evaluated by following the same logic in the subsequent execution cycles.

Final Steps: Booking an Uber

If the record qualifies to book the uber at the particular time, the service will fire the Uber API for 3 times to get the nearest 3 cabs and it will book the uber that takes the least amount of time to reach the destination.

Now that we have the nearest cab, the app will notify the user to book the cab by email. We are currently sending the email synchronously but it can be improved to send it asynchronously

Dependencies
googlemaps pypi library for getting the ideal *ride_time* and also the *ride_time with traffic* to reach from the source to the destination.

APIs

/api/trip/

Purpose: Save the request ride in the database
Method: POST
Request body:

```
{
        source: '41.43206, -81.38992',
        destination: '42.8863855, -78.8781627',
        email: 'happymishra66@gmail.com',
        time_to_reach_dest: '2019-12-12 05:07:00'
}
```

/api/trip

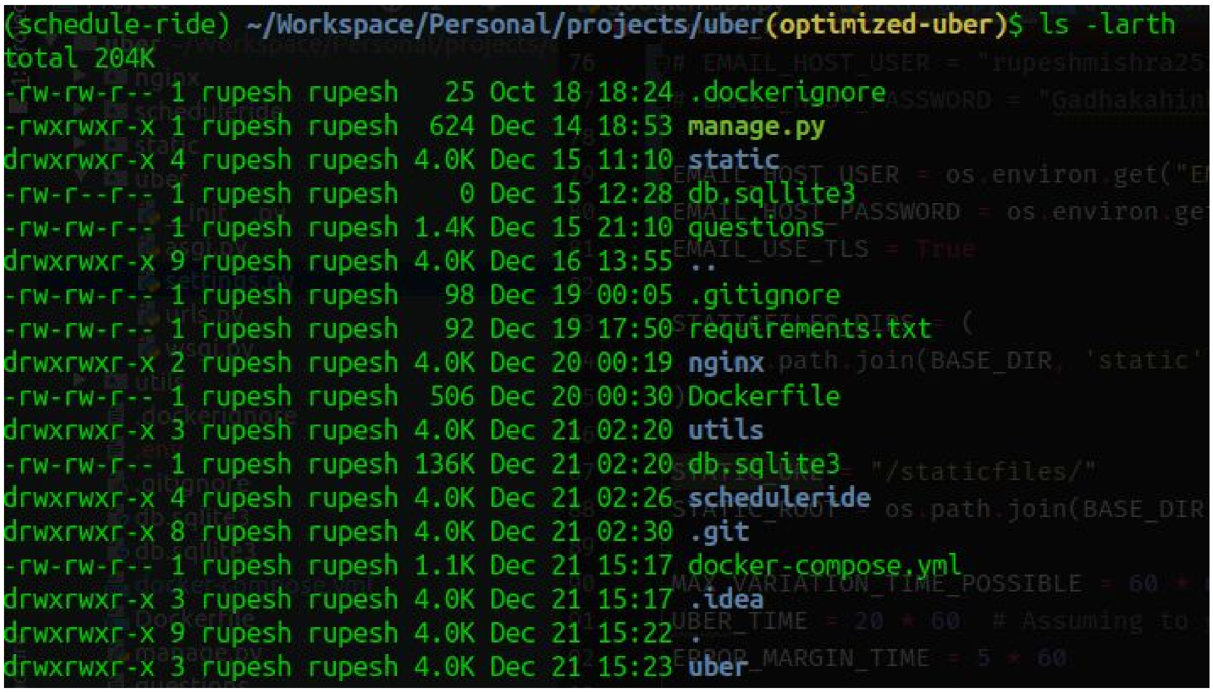
Purpose: This is fired every 10 mins to check the status of the ride. If the Google maps or the Uber API is fired, the API will return the response with the appropriate details
Method: GET
Output:

```
[
      2019-12-12 05:07:00: Maps API for fired for happymishra66@gmail.com
      2019-12-12 05:07:00: Uber apis fired for happymishra66@gmail.com
]
```

Steps to run the web app

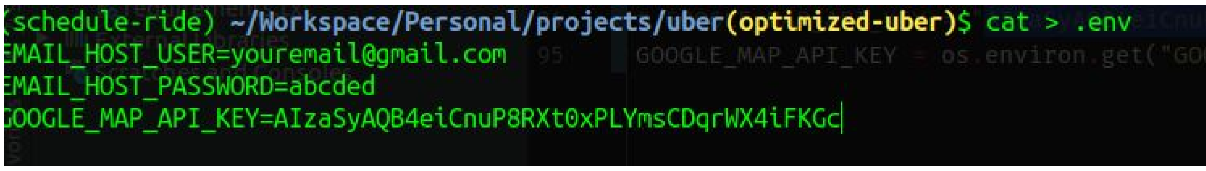
Prerequisite: *Docker* and *docker-compose* installed on the machine

1. cd */uber*
2. File structure inside uber dir



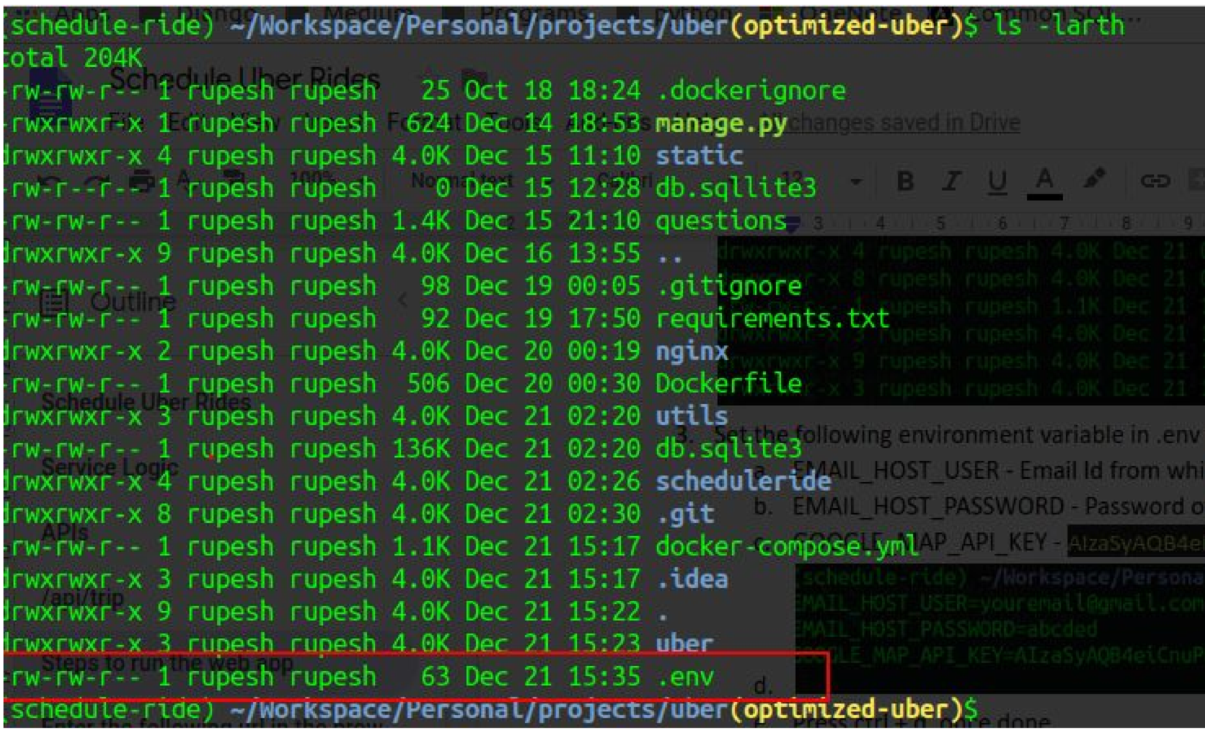
```
(schedule-ride) ~/Workspace/Personal/projects/uber(optimized-uber)$ ls -larth
total 204K
-rw-rw-r-- 1 rupesh rupesh   25 Oct 18 18:24 .dockerignore
-rwxrwxr-x 1 rupesh rupesh  624 Dec 14 18:53 manage.py
drwxrwxr-x 4 rupesh rupesh 4.0K Dec 15 11:10 static
-rw-r--r-- 1 rupesh rupesh    0 Dec 15 12:28 db.sqllite3
-rw-rw-r-- 1 rupesh rupesh 1.4K Dec 15 21:10 questions
drwxrwxr-x 9 rupesh rupesh 4.0K Dec 16 13:55 ..
-rw-rw-r-- 1 rupesh rupesh   98 Dec 19 00:05 .gitignore
-rw-rw-r-- 1 rupesh rupesh   92 Dec 19 17:50 requirements.txt
drwxrwxr-x 2 rupesh rupesh 4.0K Dec 20 00:19 nginx
-rw-rw-r-- 1 rupesh rupesh  506 Dec 20 00:30 Dockerfile
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 02:20 utils
-rw-rw-r-- 1 rupesh rupesh 136K Dec 21 02:20 db.sqlite3
drwxrwxr-x 4 rupesh rupesh 4.0K Dec 21 02:26 scheduleride
drwxrwxr-x 8 rupesh rupesh 4.0K Dec 21 02:30 .git
-rw-rw-r-- 1 rupesh rupesh 1.1K Dec 21 15:17 docker-compose.yml
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 15:17 .idea
drwxrwxr-x 9 rupesh rupesh 4.0K Dec 21 15:22 .
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 15:23 uber
```

3. Set the following environment variable in the .env file:
   a. *EMAIL_HOST_USER* - Email Id from which email will be sent
   b. *EMAIL_HOST_PASSWORD* - Password of the above email id
   c. *GOOGLE_MAP_API_KEY* - AIzaSyAQB4eiCnuP8RXt0xPLYmsCDqrWX4iFKGc



```
(schedule-ride) ~/Workspace/Personal/projects/uber(optimized-uber)$ cat > .env
EMAIL_HOST_USER=youremail@gmail.com
EMAIL_HOST_PASSWORD=abcded
GOOGLE_MAP_API_KEY=AIzaSyAQB4eiCnuP8RXt0xPLYmsCDqrWX4iFKGc
```

   d. Press Ctrl + d, once done with adding the data in the .env file to save it
   e. File structure after creating the .env file



```
schedule-ride) ~/Workspace/Personal/projects/uber(optimized-uber)$ ls -larth
otal 204K
rw-rw-r-- 1 rupesh rupesh   25 Oct 18 18:24 .dockerignore
rwxrwxr-x 1 rupesh rupesh  624 Dec 14 18:53 manage.py
drwxrwxr-x 4 rupesh rupesh 4.0K Dec 15 11:10 static
rw-r--r-- 1 rupesh rupesh    0 Dec 15 12:28 db.sqllite3
rw-rw-r-- 1 rupesh rupesh 1.4K Dec 15 21:10 questions
drwxrwxr-x 9 rupesh rupesh 4.0K Dec 16 13:55 ..
rw-rw-r-- 1 rupesh rupesh   98 Dec 19 00:05 .gitignore
rw-rw-r-- 1 rupesh rupesh   92 Dec 19 17:50 requirements.txt
drwxrwxr-x 2 rupesh rupesh 4.0K Dec 20 00:19 nginx
rw-rw-r-- 1 rupesh rupesh  506 Dec 20 00:30 Dockerfile
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 02:20 utils
rw-rw-r-- 1 rupesh rupesh 136K Dec 21 02:20 db.sqlite3
drwxrwxr-x 4 rupesh rupesh 4.0K Dec 21 02:26 scheduleride
drwxrwxr-x 8 rupesh rupesh 4.0K Dec 21 02:30 .git
rw-rw-r-- 1 rupesh rupesh 1.1K Dec 21 15:17 docker-compose.yml
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 15:17 .idea
drwxrwxr-x 9 rupesh rupesh 4.0K Dec 21 15:22 .
drwxrwxr-x 3 rupesh rupesh 4.0K Dec 21 15:23 uber
rw-rw-r-- 1 rupesh rupesh   63 Dec 21 15:35 .env
schedule-ride) ~/Workspace/Personal/projects/uber(optimized-uber)$
```

4. Once that is done, use the below command to run the application
   `sudo docker-compose -f docker-compose.yml up -d --build`
5. Enter the following URL in the browser: http://localhost:1337/

Query

```
SELECT id, source, destination, time_to_reach_dest, email,
       ride_time, frequency, pending_time, traffic_time
FROM scheduled_trip
WHERE ((CAST(strftime('%s', time_to_reach_dest) as integer) -
        CAST(strftime('%s', DATETIME(CURRENT_TIMESTAMP,'LOCALTIME'))
           as integer) < ride_time + 60*60 + 20*60
           AND frequency = 0)
      || (frequency = 2)
      || (frequency = 1 and (CAST(strftime('%s',time_to_reach_dest)
as integer) - CAST(strftime('%s',
DATETIME(CURRENT_TIMESTAMP, 'LOCALTIME')) as integer))
< (pending_time / 2) + ride_time)) AND state != 1
```