Store Monitoring System

API Requirements

1. {store\_id}/trigger\_report/
2. get\_report/
3. Background task which precomputes the report for each store\_id every hour and also updates it in the combined\_report.csv file

Note: combined\_report.csv is a file which contains report for all the stores combined in one CSV file.

Report Output Format

store\_id, uptime\_last\_hour(in minutes), uptime\_last\_day(in hours), update\_last\_week(in hours), downtime\_last\_hour(in minutes), downtime\_last\_day(in hours), downtime\_last\_week(in hours)

Data provided

1. Store Status (store\_id, timestamp\_utc, status)
2. Store Timings (store\_id, dayOfWeek(0=Monday, 6=Sunday), start\_time\_local, end\_time\_local)
3. Store Zone (store\_id, timezone\_str)

Note: Data is loaded in the database sqlite where store\_id is a **foreign key** in Store Status and Store Timings and **primary key** in Store Zone

Tech Stack Used

1. Django Framework
2. SQLite

Packages Used

1. Djangoreactframework
2. Celery (background task scheduler for precomputation)

**Logic for computing Business Hours OverLap**

The logic for computing uptime is the same for past hour , day and week

1. Uptime Last Hour
2. Uptime Last Day
3. Uptime Last Week

Step 1) Fetch the Store status of the last one week as shown below

last\_one\_week\_logs = store.status\_logs.filter(timestamp\_\_gte=utc\_time - datetime.timedelta(days=7)).order\_by('timestamp')

Note: last\_one\_week\_logs contains a list of objects which have a timestamp greater than or equal to past one week timestamp.  
Note: utc\_time is the current timestamp or the maximum timestamp from the status logs.

Step 2) Fetch the Store timings for the particular store to check if the store was open for a particular Log in last\_one\_week\_logs.

Note: This is done so that we do not have to look up in the database again and again.

store\_hours = StoreTimings.objects.filter(store\_id = store.store\_id)

Step 3) Traverse the last\_one\_week\_logs array

For each log (store\_id, timestamp\_utc, status) we need to first check if the store was open on timestamp\_utc.

for log in last\_one\_week\_logs:

        # checkig if log is in store business hours

        store\_timezone = pytz\_timezone(timezone)

        timestamp\_utc = log.timestamp

        local\_time = timestamp\_utc.astimezone(store\_timezone)

        log\_in\_store\_business\_hours = store\_hours.filter(

            day=log.timestamp.weekday(),

            start\_time\_\_lte=local\_time,

            end\_time\_\_gte=local\_time

            ).exists()

Note: Store Timings contains time in local\_time but the timestamp\_utc from last\_one\_week\_logs is in UTC time , hence timestamp\_utc is converted to localtime

log\_in\_store\_business\_hours (True or False) checks if the status timestamp lies in between the store timings or not.

Step 4) If the timestamp does not lie in business hours on that day of the store , then we will simply go to the next log in last\_one\_week\_logs.

if not log\_in\_store\_business\_hours:

            continue

Step 5) If the timestamp lies between the business hours on that day of the store , then we must check if the status was active or not active.

status\_interval\_time = 0;

        if index != len - 1:

            status\_interval\_time = min(60 , ((log[log\_index + 1].timestamp - log[index].timestamp).seconds)/60)

        else:

            status\_interval\_time = min(60 , store\_time.end\_time - local\_time.time())

        if log.status == StoreStatus.ACTIVE:

            last\_one\_week\_data["uptime"] += status\_interval\_time

        else:

            last\_one\_week\_data["downtime"] -= status\_interval\_time

        index += 1

Note: We increment the uptime or decrement the downtime by the status\_interval\_time

Status\_interval\_time is calculated by the difference between two status or 60 mins whichever is lesser

Step 6) Likewise we can calculate uptime for past hour and past day by checking the difference between status time and current time

if log.status == StoreStatus.ACTIVE:

            if current\_time - timestamp\_utc <= timedelta(hours = 1):

                last\_one\_week\_data["uptime\_hour"] += status\_interval\_time

                last\_one\_week\_data["uptime\_day"] += status\_interval\_time

                last\_one\_week\_data["uptime\_week"] += status\_interval\_time

            if current\_time - timestamp\_utc <= timedelta(days = 1):

                last\_one\_week\_data["uptime\_day"] += status\_interval\_time

                last\_one\_week\_data["uptime\_week"] += status\_interval\_time

            if current\_time - timestamp\_utc <= timedelta(days = 7):

                last\_one\_week\_data["uptime\_week"] += status\_interval\_time

        else:

            if current\_time - timestamp\_utc <= timedelta(hours = 1):

                last\_one\_week\_data["uptime\_hour"] -= status\_interval\_time

                last\_one\_week\_data["uptime\_day"] -= status\_interval\_time

                last\_one\_week\_data["uptime\_week"] -= status\_interval\_time

            if current\_time - timestamp\_utc <= timedelta(days = 1):

                last\_one\_week\_data["uptime\_day"] -= status\_interval\_time

                last\_one\_week\_data["uptime\_week"] -= status\_interval\_time

            if current\_time - timestamp\_utc <= timedelta(days = 7):

                last\_one\_week\_data["uptime\_week"] -= status\_interval\_time

Background Task (for precomputing as the store status data gets updated)

A celery task which runs every hour to check if there is any new data.

def hourly\_precomputation():

    # Your task code here

    utc\_time = datetime.now(timezone.utc)

    last\_one\_hour\_logs = store.status\_logs.filter(timestamp\_\_gte=utc\_time - datetime.timedelta(hours = 1)).order\_by('timestamp')

    csv\_data = []

    for log in last\_one\_hour\_logs:

        store\_id = log.store\_id

        store = Store.objects.filter(store\_id = store\_id).first()

        report = StoreReport.objects.create(store=store, status=ReportStatus.PENDING , report\_id=str(uuid.uuid4()))

        data = generate\_report\_data(store)

        csv\_data.append(data)

        generate\_combined\_csv\_file(report, csv\_data)

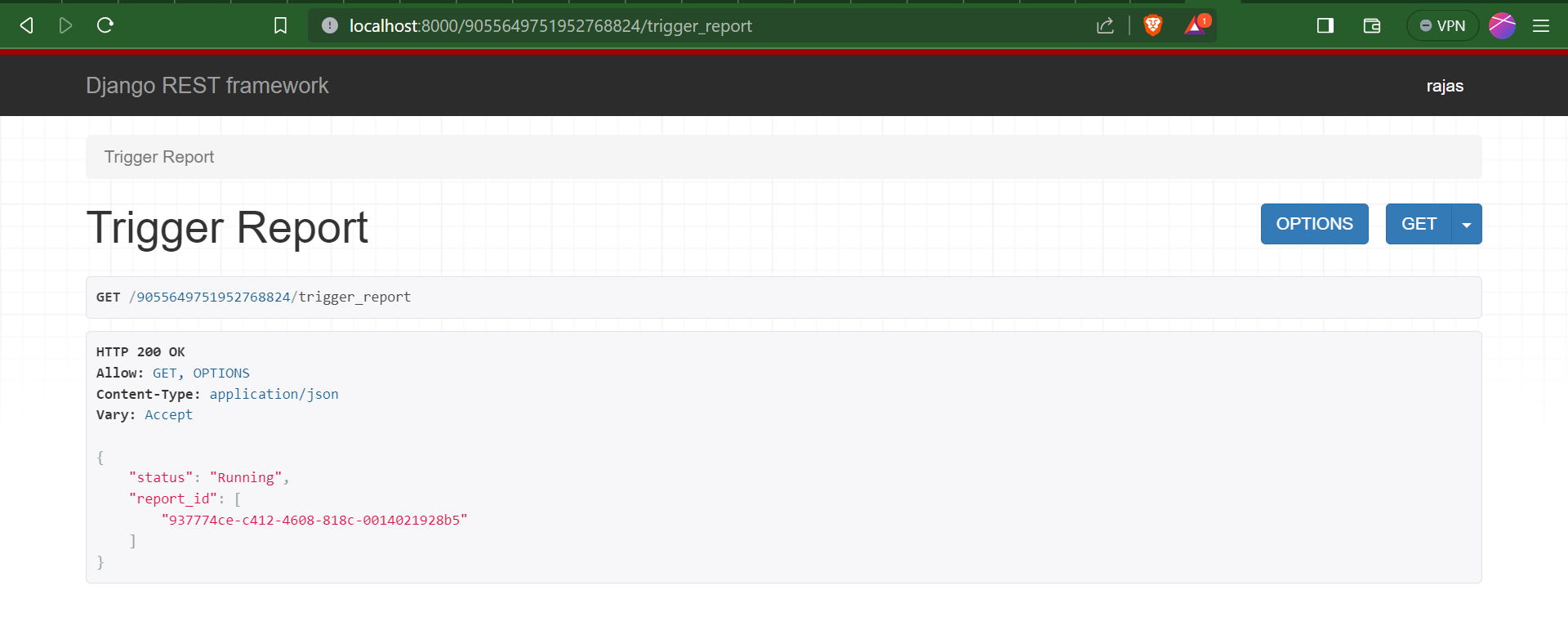
It stores uptime and downtime for each store and stores it in the database for each store\_id.  
Then generate\_combined\_csv\_file can go and update data in corresponding rows of the csv file.

Note: One Precomputation function precomputes and stores combined report of all the stores in a csv file beforehand.

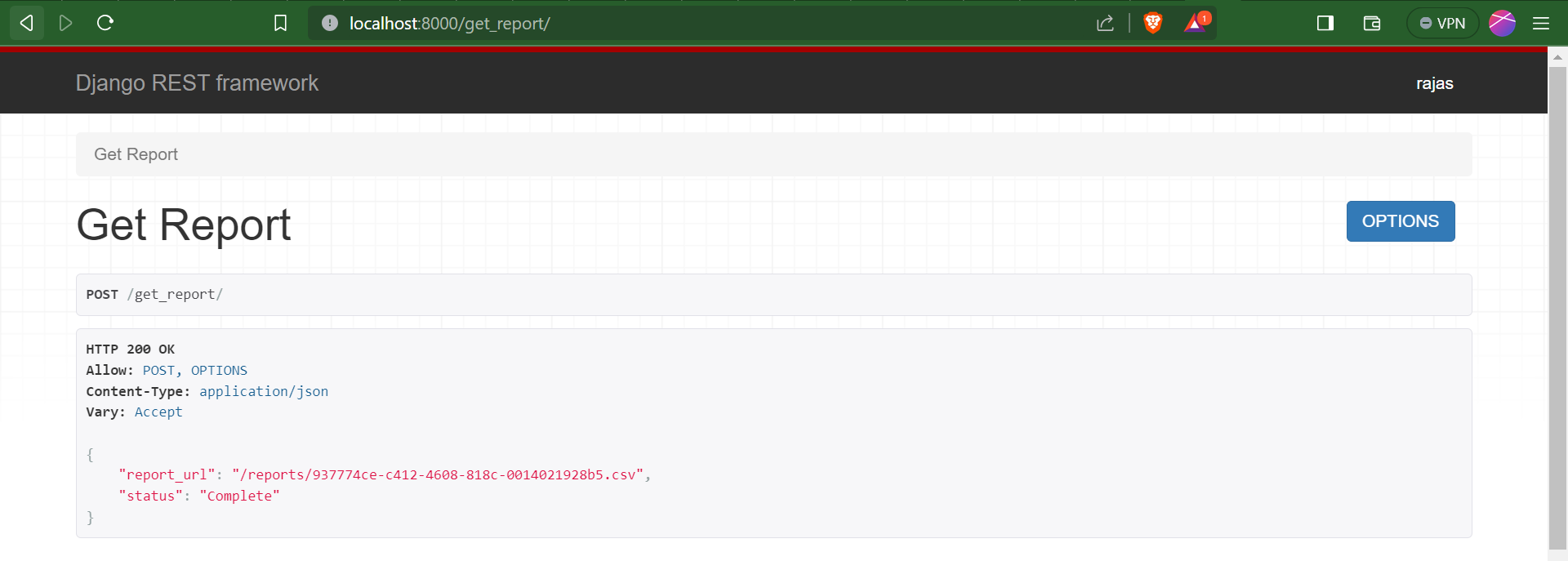
Celery task just updates selected data for selected stores every hour.

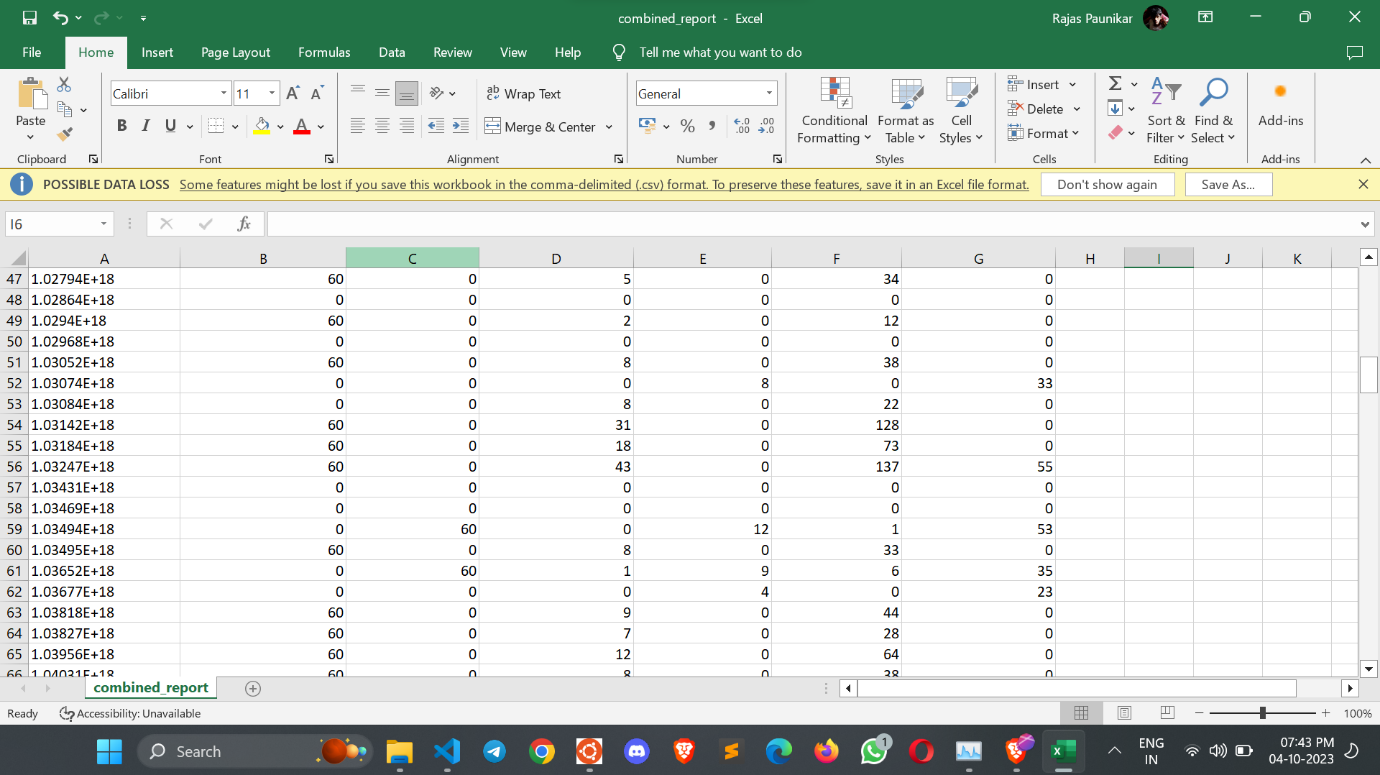
Sample Outputs

1. Trigger\_report/



1. Get\_report/



1. Combined Report Output