INFS7901 Final Report

Roland Thompson - 43534744

Project Aim

In a global pandemic, it is crucial for health organizations to track human movement to detect high-risk contacts and possible contraction clusters. To help solve this problem, we developed a relational database which stores information of patients, their movement, and people that have been with contact with them. By utilizing this database with a web application, users can see the situation of the pandemic, know the position of confirmed cases and avoid those areas if necessary.

Queries

Join Query

The below join query will return the location, name and information on how many people were there from both the event and location tables. First, two right outer joins are performed on both event and location tables against the area table. Then, the union of this returns the location, name and visitor information for all 'area' records.

```
SELECT Latitude, Longitude, Name, TotalVisitors as 'Visitors' FROM Area A
RIGHT JOIN `Event` E ON A.AreaID = E.AreaID
UNION ALL
SELECT Latitude, Longitude, Name, AvgVisitors FROM Area A
RIGHT JOIN Location L ON A.AreaID = L.AreaID
```

Latitude	Longitude	Name	Visitors
-27.48483139	153.03616652	1st Test: Australia v Pakistan (d1)	13561
-37.81866339	144.98332940	AFL: Richmond d Carlton	21000
-33.85416325	151.20916583	Ludovico Einaudi | 20 & 21 Jan	3577
-33.84166330	151.05799977	NRL: Rabbitohs d Sharks	6235
-27.49199803	153.00766664	University of Queensland	9000
-33.85597991	151.20666584	International Towers Sydney 1	4650
-35.10210000	139.14240000	Monarto Zoo	1600
-33.86982985	151.20433252	Hilton Hotel Sydney	200
-42.00000000	147.00000000	MONA	950

This information can then be used with a GPS software (such as Google maps) to visualize each location with a confirmed case.

Aggregation Query

The below query can be used to quickly find how many confirmed cases each state has:

```
SELECT State, Count(*)
FROM Patient
GROUP BY State
```

State	Count(*)
New South Wales	2
Northern Territory	1
Queensland	3
South Australia	1
Tasmania	1
Victoria	1
Western Australia	1

We can expand on this to include the HighRiskContact table to get a better understanding of how much the virus is spreading for each state:

```
SELECT State, SUM(count) FROM
(
    SELECT State, Count(*) as count
    FROM Patient
    GROUP BY State
    UNION
    SELECT State, Count(*)
    FROM HighRiskContact
    GROUP BY State
) s
GROUP BY State
```

State	SUM(count)
New South Wales	9
Northern Territory	1
Queensland	7
South Australia	3
Tasmania	1
Victoria	4
Western Australia	1

This information can be used in the summary page of the website to quickly show the current situation of the pandemic for each state.

Jurisdiction	Total confirmed cases
Australia	7,290
ACT	108
NSW	3,116
NT	29
QLD	1,064
SA	440
TAS	228
VIC	1,703
WA	602

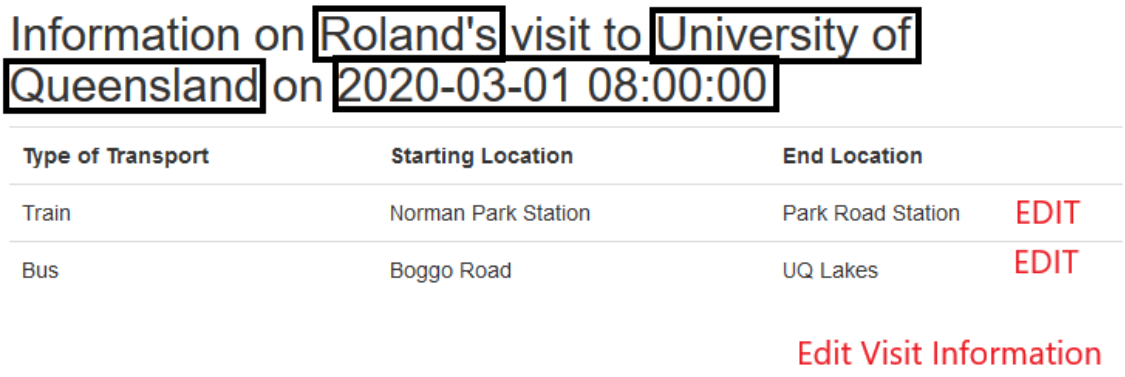
Update Operation

The below query can update information on a patient's visit to a location, or the transport they used:

```
UPDATE `Visited` SET
`PatientID` = A,
`AreaID` = B,
`StartTime` = C,
`EndTime` = D WHERE VisitID = visit_id
```

```
UPDATE `Transport` SET
`TransportType` = E,
`StartLocation` = F,
`EndLocation` = G WHERE TransportID = transport_id
```

This function can be used to edit the information on visit sessions for all patients.



SQL Assertions

The 'Patient' table is in total participation with the 'contracted' relationship with the 'Virus' table. That is, the patient has to contract at least one virus in the database. In the front-end code base, we assure this by setting the min_entries flag in the virus FieldList to 1:
viruses = FieldList(FormField(ContractedVirusForm), min_entries=1)
We can also implement an assertion in SQL to make sure that the patient table is in total participation.

```
CREATE ASSERTION patient_has_virus
CHECK
(NOT EXISTS (
  SELECT PatientID
  FROM Patient
  WHERE PatientID NOT IN (SELECT PatientID FROM Contracted)))
```

Since MySQL does not support assertions, we can instead use triggers:

```
CREATE TRIGGER patient_has_virus
AFTER INSERT ON Patient
FOR EACH ROW
BEGIN
    DECLARE msg varchar(128)
    IF NEW.PatientID NOT IN (SELECT PatientID FROM Contracted) THEN
            set msg = concat("Error: PatientID ", cast(NEW.PatientID as
char)," has no virus contracted!");
        signal sqlstate '45000' set message_text = msg;
    END IF;
END
```

This check can be used when patient information is created/edited to see if the patient has contracted a virus:

Add Patient

First Name

Last Name

Gender

Male

Nationality

State

Postcode

Date of Birth

dd / mm / yyyy

Phone Number

Viruses

Virus Name	Date of Contraction
SARS-CoV-2	dd / mm / yyyy

Add Virus

Add Patient

If "Viruses" field is empty... show the following message

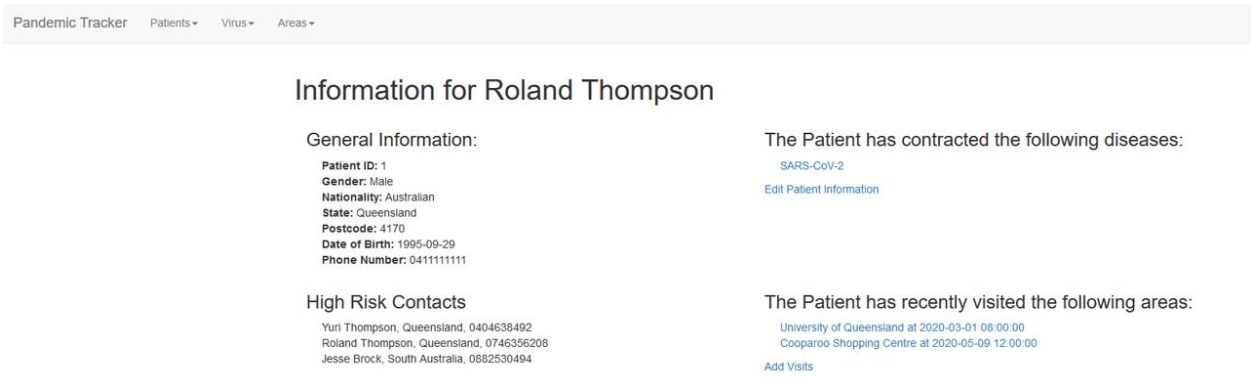
ERROR! Please select at least one virus

GUI

Select Query - In the patient information page

```
# Retrieve patient information
SELECT * FROM Patient WHERE PatientID = {patient_id}


# Retrieve people who have been in contact with the patient
SELECT * FROM HighRiskContact WHERE PatientID = {patient_id}


# Retrieve the viruses that the patient have contracted
SELECT V.VirusID, V.Name FROM Contracted C, Virus V WHERE C.VirusID =
V.VirusID
                                                    AND C.PatientID =
{patient_ID}


# Retrieve the areas that the patient have visited
SELECT AreaID, StartTime FROM Visited WHERE PatientID = {patient_id}


@app.route('/patient/<patient_id>')
def patient(patient_id):
    query = f"SELECT * FROM Patient WHERE PatientID = {patient_id}"
    cursor.execute(query)
    patient_info = cursor.fetchall()

    query = f"SELECT * FROM HighRiskContact WHERE PatientID = {patient_id}"
    cursor.execute(query)
    high_risk_contacts = cursor.fetchall()

    query = f"SELECT V.VirusID, V.Name FROM Contracted C, Virus V WHERE
C.VirusID = V.VirusID AND C.PatientID = {patient_id}"
    cursor.execute(query)
    contracted_viruses = cursor.fetchall()

    query = f"SELECT AreaID, StartTime FROM Visited WHERE PatientID =
{patient_id}"
    cursor.execute(query)
    visited_areas = cursor.fetchall()

    location_names = []
```

```
    for area_id in [area[0] for area in visited_areas]:
        query = f"(SELECT Name FROM Location WHERE AreaID = {area_id}) \
                UNION \
                (SELECT Name FROM Event WHERE AreaID = {area_id})"
        cursor.execute(query)
        location_names.append(cursor.fetchall()[0][0])

    for i, name in enumerate(location_names):
        visited_areas[i] = visited_areas[i] + (name,)
    visited_areas = list(set(visited_areas))

    return render_template('patient.html', title=f'Patient Number:
{patient_id}', patient=patient_info[0], contacts=high_risk_contacts,
viruses=contracted_viruses, visits=visited_areas)
```

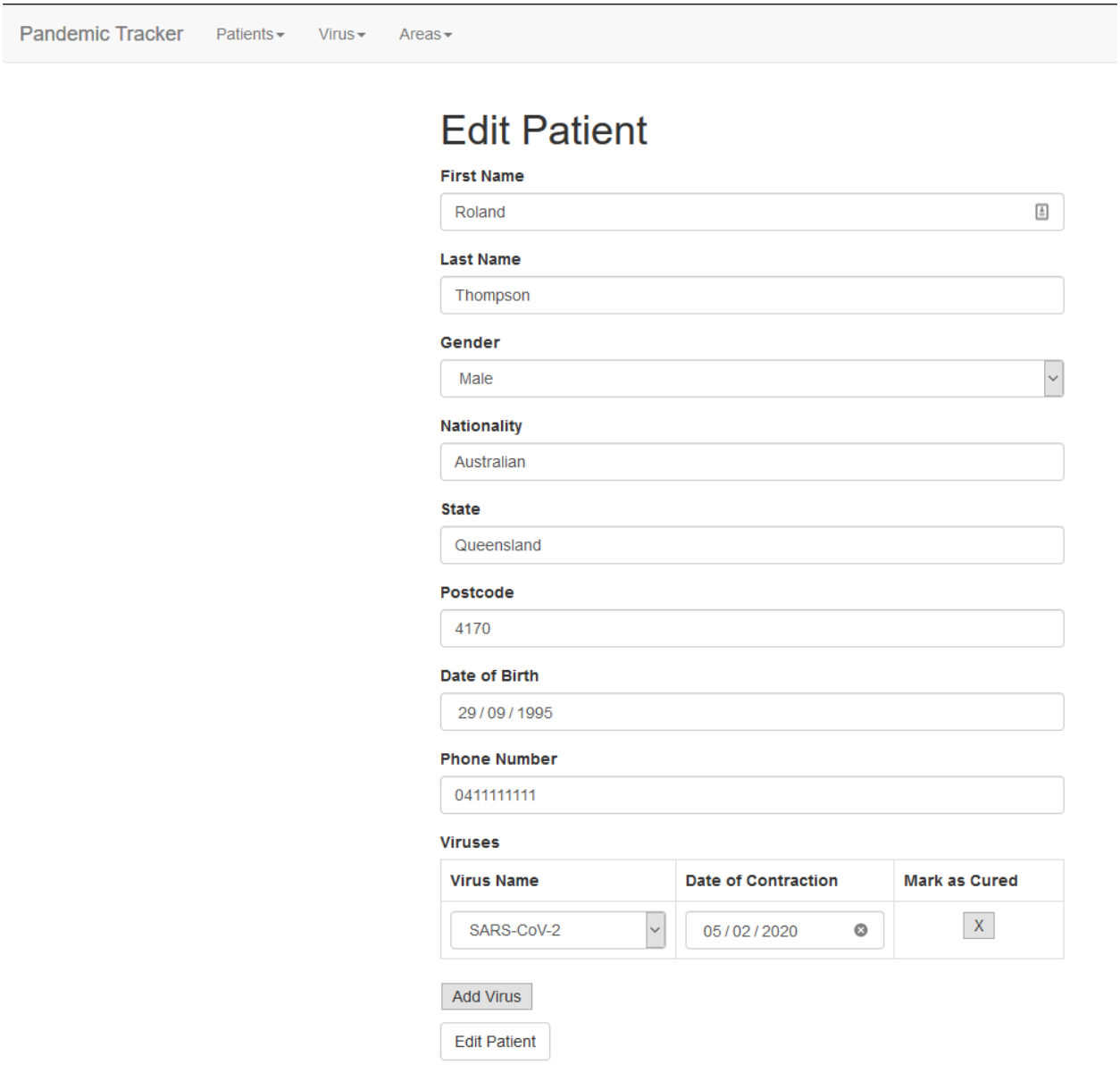


Update Query - Editing patient information

```
# Updating general patient information
UPDATE `Patient` SET `FirstName` = '{form.first_name.data}', \
        `LastName` = '{form.last_name.data}', \
        `Gender` = '{form.gender.data}', \
        `Nationality` = '{form.nationality.data}', \
        `State` = '{form.state.data}', \
        `Postcode` = '{form.postcode.data}', \
        `DOB` = '{form.date_of_birth.data}', \
        `PhoneNumber` = '{form.phone_number.data}' WHERE PatientID =
{patient_id}


# Updating virus information, if the contracted date have changed
UPDATE `Contracted` SET `ContractDate` = '{contract_date}' \
        WHERE PatientID = {patient_id} AND VirusID = {virus_id}


@app.route('/patient/<patient_id>/edit', methods=['GET', 'POST'])
def edit_patient(patient_id):
    form = PatientEdit()


    query = f"SELECT * FROM Patient WHERE PatientID = {patient_id}"
    cursor.execute(query)
    records = cursor.fetchall()


    query = f"SELECT V.VirusID, C.ContractDate FROM Contracted C, Virus V
WHERE C.VirusID = V.VirusID AND C.PatientID = {patient_id}"
    cursor.execute(query)
    contracted_viruses = cursor.fetchall()


    if not form.is_submitted():
        patient_info = Patient(records[0])
        form.process(obj=patient_info)
        if contracted_viruses:
            form.viruses.pop_entry()
        for virus_id, contract_date in [entry for entry in
contracted_viruses]:
            virus_form = ContractedVirusForm()
            virus_form.virus = virus_id
```

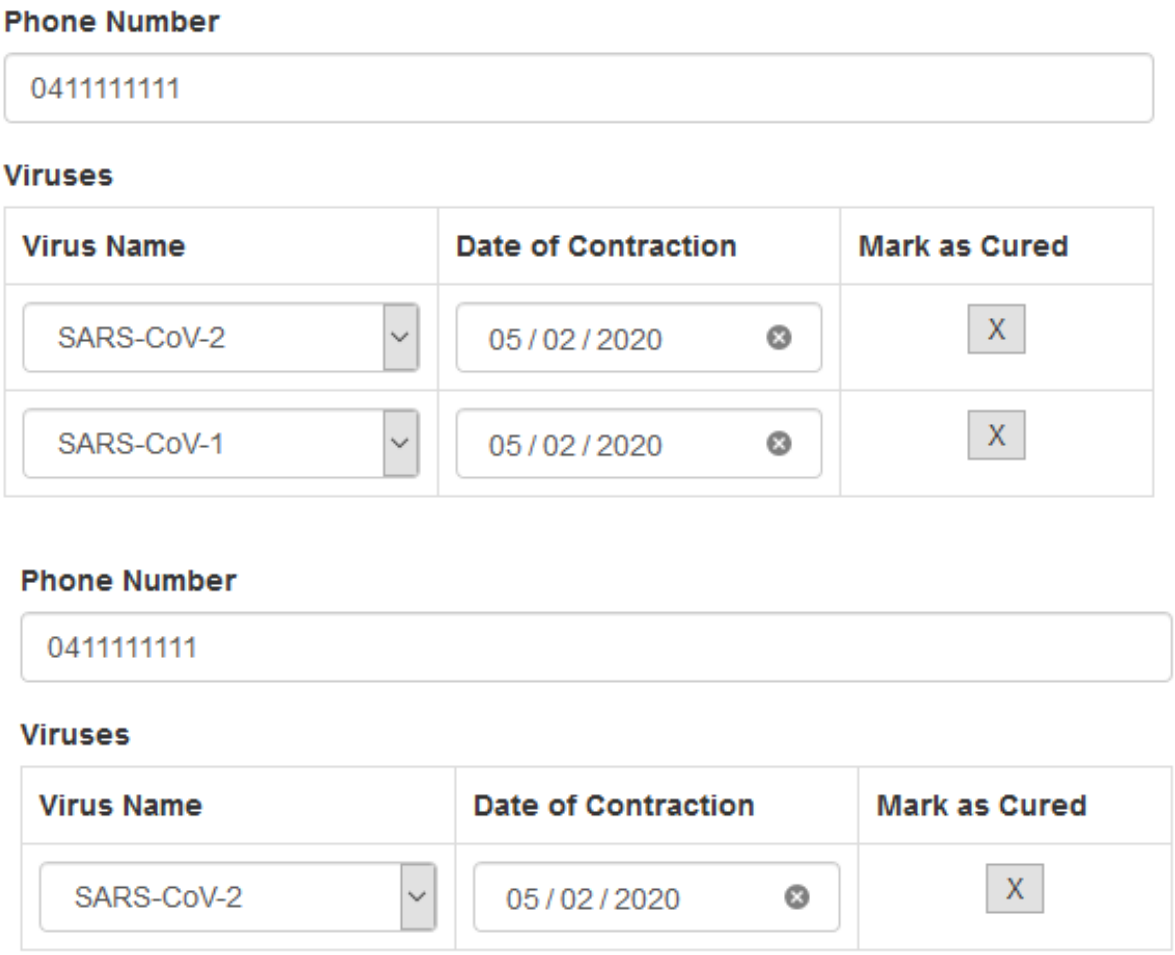
```
            virus_form.contract_date = contract_date
            form.viruses.append_entry(virus_form)
    else:
        form = PatientEdit()


    if form.validate_on_submit():
        flash(f'Patient {form.first_name.data} successfully edited.')
        query = f"Update `Patient` SET `FirstName` =
'{form.first_name.data}', \
                `LastName` = '{form.last_name.data}', \
                `Gender` = '{form.gender.data}', \
                `Nationality` = '{form.nationality.data}', \
                `State` = '{form.state.data}', \
                `Postcode` = '{form.postcode.data}', \
                `DOB` = '{form.date_of_birth.data}', \
                `PhoneNumber` = '{form.phone_number.data}' WHERE PatientID =
{patient_id}"
        cursor.execute(query)
        db.commit()


        for entry in form.viruses.data:
            virus_id = int(entry.get('virus'))
            contract_date = entry.get('contract_date')
            if virus_id not in [entry[0] for entry in contracted_viruses]:
                query = f"INSERT INTO `Contracted` (`PatientID`, `VirusID`,
`ContractDate`) VALUES \
                        (%s, %s, %s)"
                data_list = [patient_id, virus_id, contract_date]
                cursor.execute(query, data_list)
                db.commit()
            else:
                if contract_date != [entry[1] for entry in
contracted_viruses if entry[0] == virus_id][0]:
                    query = f"UPDATE `Contracted` SET `ContractDate` =
'{contract_date}' \
                            WHERE PatientID = {patient_id} AND VirusID =
{virus_id}"
                    cursor.execute(query)
                    db.commit()
```

```
        # Delete entries that has been marked as cured
        for entry in contracted_viruses:
            if str(entry[0]) not in [form_data.get('virus') for form_data in
form.viruses.data]:
                query = f"DELETE FROM `Contracted` WHERE PatientID =
'{patient_id}' AND VirusID = '{entry[0]}'"
                cursor.execute(query)
                db.commit()
        return redirect(url_for('patient', patient_id=patient_id))
    return render_template('edit_patient.html', title='Edit Patient
Information', form=form)
```



Delete query - Deleting entries if a patient is marked as cured

```
DELETE FROM `Contracted` WHERE PatientID = '{patient_id}' AND VirusID =
'{entry[0]}'
```



Insert query - Adding a location where the patient have visited

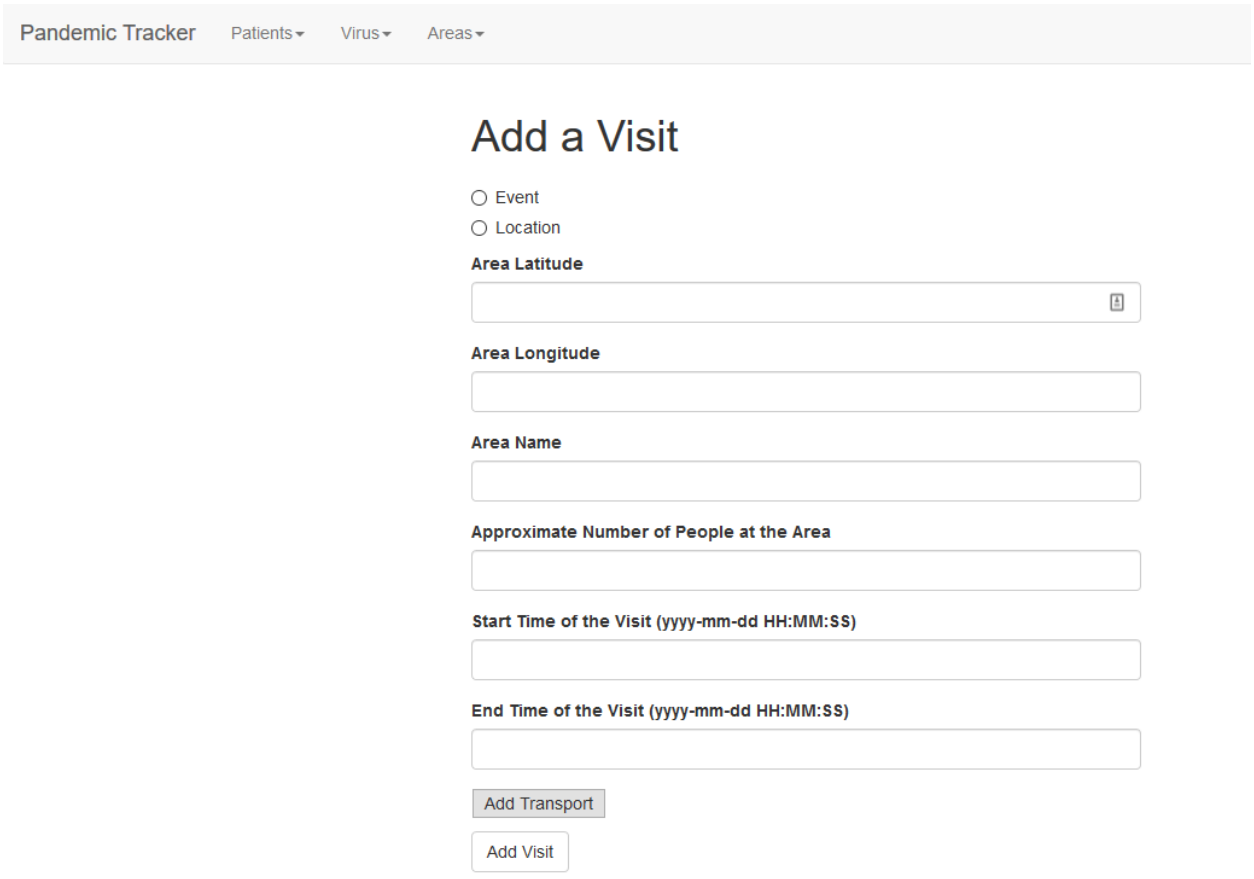
```
# Add information to Area table
INSERT INTO `Area` (`Latitude`, `Longitude`) VALUES (%s, %s)


# Add information to Location/Event table, depending on user selection
INSERT INTO `Location` (`AreaID`, `Name`, `AvgVisitors`) VALUES (%s, %s, %s)
INSERT INTO `Event` (`AreaID`, `Name`, `TotalVisitors`) VALUES (%s, %s, %s)


# Insert into transports table
INSERT INTO `Transport` (`TransportType`, `StartLocation`, `EndLocation`)
VALUES (%s, %s, %s)


# Insert into Visited table
INSERT INTO `Visited` (`PatientID`, `TransportID`, `AreaID`, `StartTime`,
`EndTime`) VALUES (%s, %s, %s, %s, %s)


@app.route('/patient/<patient_id>/add_visit', methods=['GET', 'POST'])
def add_visit(patient_id):
    form = VisitedAreaAdd()
    query = f"SELECT FirstName FROM Patient WHERE PatientID = {patient_id}"
    cursor.execute(query)
    patient_name = cursor.fetchall()[0][0]


    if form.validate_on_submit():
        flash(f"{patient_name}'s visit to {form.area_name.data} successfully
recorded.")


        # Find if existing location/event exists
        if form.area_type.data == 'location':
            query = f"SELECT AreaID FROM Location WHERE Name =
'{form.area_name.data}'"
            cursor.execute(query)
            location_match = cursor.fetchall()


            # If the location does not exist
            if not location_match:
```

```
                query = f"INSERT INTO `Area` (`Latitude`, `Longitude`)
VALUES \
                        (%s, %s)"
                data_list = [form.area_lat.data, form.area_lon.data]
                cursor.execute(query, data_list)
                db.commit()
                area_id = cursor.lastrowid


                query = f"INSERT INTO `Location` (`AreaID`, `Name`,
`AvgVisitors`) VALUES \
                        (%s, %s, %s)"
                data_list = [area_id, form.area_name.data,
form.area_visitors.data]
                cursor.execute(query, data_list)
                db.commit()
            else:
                area_id = location_match[0][0]
        else:
            query = f"SELECT AreaID FROM Event WHERE Name =
'{form.area_name.data}'"
            cursor.execute(query)
            event_match = cursor.fetchall()


            # If the event does not exist
            if not event_match:
                query = f"INSERT INTO `Area` (`Latitude`, `Longitude`)
VALUES \
                        (%s, %s)"
                data_list = [form.area_lat.data, form.area_lon.data]
                cursor.execute(query, data_list)
                db.commit()
                area_id = cursor.lastrowid


                query = f"INSERT INTO `Event` (`AreaID`, `Name`,
`TotalVisitors`) VALUES \
                        (%s, %s, %s)"
                data_list = [area_id, form.area_name.data,
form.area_visitors.data]
                cursor.execute(query, data_list)
                db.commit()
```

```
            else:
                area_id = event_match[0][0]



        # Parse transport data
        transports = form.transports.data
        if len(transports) == 1 and transports[0].get('start_location') ==
'None' and transports[0].get('end_location') == 'None':
            transport_id = None
            # Add data to visited table
            query = f"INSERT INTO `Visited` (`PatientID`, `TransportID`,
`AreaID`, `StartTime`, `EndTime`) VALUES \
                    (%s, %s, %s, %s, %s)"
            data_list = [patient_id, transport_id, area_id,
form.visit_start_time.data, form.visit_end_time.data]
            cursor.execute(query, data_list)
            db.commit()
        else:
            for transport in transports:
                query = f'''SELECT TransportID FROM Transport \
                        WHERE TransportType =
"{transport.get('transport_type')}" \
                        AND StartLocation =
"{transport.get('start_location')}" \
                        AND EndLocation =
"{transport.get('end_location')}"'''
                cursor.execute(query)
                transport_match = cursor.fetchall()
                if transport_match:
                    transport_id = transport_match[0][0]
                else:
                    query = f"INSERT INTO `Transport` (`TransportType`,
`StartLocation`, `EndLocation`) VALUES \
                            (%s, %s, %s)"
                    data_list = [transport.get('transport_type'),
transport.get('start_location'), transport.get('end_location')]
                    cursor.execute(query, data_list)
                    db.commit()
                    transport_id = cursor.lastrowid
```

```
                # Add data to visited table
                query = f"INSERT INTO `Visited` (`PatientID`, `TransportID`,
`AreaID`, `StartTime`, `EndTime`) VALUES \
                        (%s, %s, %s, %s, %s)"
                data_list = [patient_id, transport_id, area_id,
form.visit_start_time.data, form.visit_end_time.data]
                cursor.execute(query, data_list)
                db.commit()
```

```
        return redirect(url_for('patient', patient_id=patient_id))
    return render_template('add_visit.html', title='Add Visit', form=form)
```



Reflection of the Project

I think I spent about 30 hours on this project. 5 hours on the project proposal, another 5 hours on the formal specifications and the rest for the final deliverable. A lot of time was spent on setting up the database on phpMyAdmin, filling it with fake data and importing it to be used with python. The finishing touches to the website (Javascript/styling) took a lot of time as well, especially with the dynamic form fields for virus/transport.

I enjoyed building the API for this project, but that is because I had some experience with it. I still learned a lot when it comes to Python/MySQL integration. It was interesting to learn how to:

- Export a database on phpMyAdmin to a sql file
- Create a MySQL database from the sql file
- Running the MySQL server and making it communicate with Python Flask.

I didn't enjoy making mock data for the database. It was mundane and time consuming.

There were a few resources that were very useful in helping me figure out what to do for this project. For SQL/database theory, looking over the lecture notes helped the most. By looking at that, it was very straight forward. For the web development, I went through Miguel Grinberg's Flask mega tutorial (https://blog.miguelgrinberg.com/index). It used SQLalchemy, so I had to replace that with MySQL. However, every other component (routing, styling, app configs) were very useful in creating my own project.

I think the project requires too much web development when it is not within the scope of the course. I understand that it makes the project more interesting, but a lot of students don't have past experience in web development and I think it will be very hard on them. I think making the project less orientated on web development and more on databases (phpMyAdmin) might make it fair for the students.