Part A

520199901 lab26_3

Part B

database.py

```
def show_all_trips():
    conn = database_connect()
    if conn is None:
        return None    # Returns None if the connection could not be established
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")
    try:
        trips_list = dictfetchall(cur, "SELECT * FROM trips ORDER BY tripid")
        cur.close()    # close cursor
        conn.close()
        return trips_list
    except Exception as e:
        # If there are any errors, return None and print the error message.
        print(f"Unexpected error retrieving trips: {e}")
        cur.close()                                # Close the cursor
        conn.close()                               # Close the connection to the db
        raise

def filter_trips_by_date(date):
    conn = database_connect()
    if conn is None:
        return None    # If the connection cannot be established, return None
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")
    try:
        trips_list = dictfetchall(cur, "SELECT * FROM trips WHERE traveldate = %s ORDER BY tripid", (date,))
        cur.close()    # Close the cursor
        conn.close()    # Close the connection to the db
        return trips_list
    except Exception as e:
        # If any error occurs, return None and print the error message
```

```
            print(f"Unexpected error retrieving trips: {e}")
            cur.close()                             # Close the cursor
            conn.close()                             # Close the connection to the db
            raise

def get_trip(trip_id):
    conn = database_connect()
    if conn is None:
        return None    # If the connection cannot be established, return None
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")
    try:
        trips_list = dictfetchall(cur, "SELECT * FROM trips WHERE tripid = %s", (trip_id,))
        cur.close()    # Close the cursor
        conn.close()    # Close the connection to the db
        return trips_list
    except Exception as e:
        # If any error occurs, return None and print the error message
        print(f"Unexpected error retrieving trips: {e}")
        cur.close()                             # Close the cursor
        conn.close()                             # Close the connection to the db
        raise

def add_trip(trip_data):
    conn = database_connect()
    if conn is None:
        return {'success': False, 'errors': ['Unable to establish database connection']}
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")

    try:
        sql = """
        INSERT INTO trips(cardid, traveldate, entrystationid, exitstationid, tripstarttime)
        VALUES (%s, %s, %s, %s, %s)
        """
        cur.execute(sql,  (trip_data['cardid'],  trip_data['traveldate'],  trip_data['entrystationid'],
trip_data['exitstationid'], trip_data['tripstarttime']))
        conn.commit()
        return {'success': True}
    except Exception as e:
        conn.rollback()
        print(f"Unexpected error adding trip: {e}")
        return {'success': False, 'errors': [str(e)]}
    finally:
```

```
            cur.close()
            conn.close()    # Close connection

def update_trip(trip_id, trip_data):
    conn = database_connect()
    if conn is None:
        return {'success': False, 'errors': ['Unable to establish database connection']}
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")

    try:
        sql = """
        UPDATE trips
        SET cardid=%s, traveldate=%s, entrystationid=%s, exitstationid=%s, tripstarttime=%s
        WHERE tripid=%s
        """
        cur.execute(sql,  (trip_data['cardid'],  trip_data['traveldate'],  trip_data['entrystationid'],
trip_data['exitstationid'], trip_data['tripstarttime'], trip_id))

        if cur.rowcount == 0:
            conn.rollback()
            return {'success': False, 'errors': ['No corresponding trip ID was found.']}

        conn.commit()
        return {'success': True}
    except Exception as e:
        conn.rollback()
        print(f"Unexpected error updating trip: {e}")
        return {'success': False, 'errors': [str(e)]}
    finally:
        cur.close()
        conn.close()

def delete_trip(trip_id):
    conn = database_connect()
    if conn is None:
        return {'success': False, 'errors': ['Unable to establish database connection']}

    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")

    try:
        sql = """DELETE FROM trips WHERE tripid=%s"""
        cur.execute(sql, (trip_id,))
```

```
            if cur.rowcount == 0:
                conn.rollback()    # Rollback if no rows were deleted, implying trip_id was not found
                return {'success': False, 'errors': ['The corresponding trip ID was not found and the delete operation was not executed.']}

            conn.commit()    # Commit the changes if the row was deleted
            return {'success': True}
        except Exception as e:
            conn.rollback()
            print(f"Unexpected error deleting trip: {e}")
            return {'success': False, 'errors': [str(e)]}
        finally:
            cur.close()
            conn.close()

def generate_report():
    conn = database_connect()
    if conn is None:
        return {'success': False, 'errors': ['Unable to establish database connection']}

    cur = conn.cursor()
    try:
        cur.execute("SET search_path TO opaltravel;")

        sql = "SELECT entrystationid, COUNT(*) as trip_count FROM trips GROUP BY entrystationid"
        cur.execute(sql)
        rows = cur.fetchall()

        # Convert rows to a list of dictionaries to make them easier to work with
        columns = [column[0].decode('utf-8') if isinstance(column[0], bytes) else column[0] for column in cur.description]
        report = [dict(zip(columns, row)) for row in rows]

        # Decode byte strings to normal strings
        for record in report:
            for key, value in record.items():
                if isinstance(value, bytes):
                    record[key] = value.decode('utf-8')

        return {'success': True, 'data': report}
    except Exception as e:
        print(f"Unexpected error generating report: {e}")
```

```
            return {'success': False, 'errors': [str(e)]}
        finally:
            cur.close()
            conn.close()



def check_trip_data(trip_data):
    conn = database_connect()
    if conn is None:
        return {'success': False, 'errors': ['Unable to establish database connection']}
    cur = conn.cursor()
    cur.execute("SET search_path TO opaltravel;")

    errors = []

    # Converting a traveldate to a date object in string format
    try:
        travel_date = datetime.strptime(trip_data['traveldate'], '%Y-%m-%d').date()
    except ValueError:
        errors.append("Travel date format is invalid. Use YYYY-MM-DD.")

    # Check if cardid exists in OpalCards
    card_check_sql = "SELECT cardid FROM OpalCards WHERE cardid = %s"
    card_exists = dictfetchone(cur, card_check_sql, (trip_data['cardid'],))
    if not card_exists:
        errors.append("Card ID does not exist in OpalCards.")

    # Check if traveldate is today or before
    if travel_date > datetime.now().date():
        errors.append("Travel date cannot be in the future.")

    # Check if entrystationid exists in Stations
    entry_station_check_sql = "SELECT stationid FROM Stations WHERE stationid = %s"
    entry_station_exists = dictfetchone(cur, entry_station_check_sql,
(trip_data['entrystationid'],))
    if not entry_station_exists:
        errors.append("Entry Station ID does not exist in Stations.")

    # Check if the exitstationid exists in Stations
    exit_station_check_sql = "SELECT stationid FROM Stations WHERE stationid = %s"
    exit_station_exists = dictfetchone(cur, exit_station_check_sql, (trip_data['exitstationid'],))
    if not exit_station_exists:
        errors.append("Exit Station ID does not exist in Stations.")
```

```
    # Validation Time Format
    try:
        # This will throw a ValueError if the time is not in HH:MM:SS format or if the time value
is incorrect.
        datetime.strptime(trip_data['tripstarttime'], '%H:%M:%S').time()
    except ValueError as e:
        errors.append(f"Trip start time format is invalid: {e}")

    # Check for errors before attempting a query
    if errors:
        return errors    # If there is an error, return the error list directly

    # Check for duplicate travel entries
    trip_check_sql = """SELECT tripid FROM Trips WHERE cardid=%s AND traveldate=%s AND
                                entrystationid=%s AND exitstationid=%s AND tripstarttime=%s"""
    trip_exists = dictfetchone(cur, trip_check_sql, (trip_data['cardid'], travel_date,
                                                     trip_data['entrystationid'],
trip_data['exitstationid'],
                                                     trip_data['tripstarttime']))
    if trip_exists:
        errors.append("An identical trip already exists.")

    return errors
```

route.py

```
@app.route('/trips')
def show_trips():
    try:
        # Call the function directly to get the list of trips
        trips_list = database.show_all_trips()
        if trips_list is None:
            raise Exception("Could not retrieve trips")
    except Exception as e:
        flash('An error occurred: ' + str(e))
        trips_list = []

    page['title'] = 'All Trips'
    return render_template('list_trips.html', page=page, session=session, trips=trips_list)

@app.route('/filter_trips', methods=['GET', 'POST'])
def filter_trips():
    page['title'] = 'Filter Trips by Date'
```

```
    if request.method == 'POST':
        date = request.form['date']
        # Check if the date field is empty
        if not date:
            flash('Please select a date to filter.', 'error')
            return render_template('filter_trips.html', page=page, session=session)

        # Check if the selected date is not in the future
        selected_date = datetime.strptime(date, "%Y-%m-%d").date()
        if selected_date > datetime.now().date():
            flash('Cannot select a future date. Please select today\'s or a past date.', 'error')
            return render_template('filter_trips.html', page=page, session=session)

        try:
            trips_list = database.filter_trips_by_date(date)
            if not trips_list:
                flash('No trips found for the selected date.', 'info')
                return render_template('filter_trips.html', page=page, session=session)
        except Exception as e:
            flash(f'An error occurred while retrieving trips: {e}', 'error')
            return render_template('filter_trips.html', page=page, session=session)

        page['title'] = 'Filtered Trips'
        return render_template('list_trips.html', page=page, session=session, trips=trips_list)

    # GET request, show the form
    return render_template('filter_trips.html', page=page, session=session)

@app.route('/add_trip', methods=['GET', 'POST'])
def add_trip():
    page = {'title': 'Add Trip'}    # Define page title here
    if request.method == 'POST':
        trip_data = {
            'cardid': request.form['cardid'],
            'traveldate': request.form['traveldate'],
            'entrystationid': request.form['entrystationid'],
            'exitstationid': request.form['exitstationid'],
            'tripstarttime': request.form['tripstarttime']
        }
        errors = database.check_trip_data(trip_data)

        if errors:
            for error in errors:
```

```
                flash(error)
            return render_template('add_trip.html', trip_data=trip_data, page=page)  # Include page in the context
        else:
            result = database.add_trip(trip_data)
            if result['success']:
                flash('Trip added successfully!', 'success')
                return redirect(url_for('show_trips'))
            else:
                flash('Failed to add trip.', 'error')
                return render_template('add_trip.html', trip_data=trip_data, page=page)  # Include page in the context

    # If it's a GET request, just render the template with the page title
    return render_template('add_trip.html', page=page)  # Include page in the context

@app.route('/trips/delete', methods=['POST'])
def delete_trip_route():
    trip_id = request.form['trip_id']
    result = database.delete_trip(trip_id)
    if result['success']:
        flash('Trip deleted successfully!', 'success')
    else:
        for error in result['errors']:
            flash(error, 'error')
    return redirect(url_for('show_trips'))

@app.route('/report')
def view_report():
    report = database.generate_report()
    if report['success']:
        print("Report generated successfully.")
        page = {'title': 'Trip Report'}
        return render_template('report.html', page=page, report_data=report['data'])
    else:
        print("Failed to generate report.")
        print("Errors:", report['errors'])
        flash('An error occurred while generating the report: ' + ', '.join(report['errors']))
        return redirect(url_for('index'))

@app.route('/update_trip/<int:trip_id>', methods=['GET', 'POST'])
def update_trip_page(trip_id):
    # Define the page title context for the update page
    page = {'title': 'Update Trip'}
```

```
    # Get the existing trip data from the database before POST request handling
    original_trip_data_result = database.get_trip(trip_id)
    if not original_trip_data_result:
        flash('Trip ID not found.', 'error')
        return redirect(url_for('show_trips'))

    # Extract the first dictionary from the result as original data
    original_trip_data = original_trip_data_result[0]

    if request.method == 'POST':
        # Collect form data from POST request
        trip_data = {
            'cardid': request.form['cardid'],
            'traveldate': request.form['traveldate'],
            'entrystationid': request.form['entrystationid'],
            'exitstationid': request.form['exitstationid'],
            'tripstarttime': request.form['tripstarttime']
        }

        # Validate the data (implementation of check_trip_data is not shown)
        errors = database.check_trip_data(trip_data)

        if errors:
            # If there are errors, flash them and re-render the update page with ORIGINAL data
            for error in errors:
                flash(error)
            return render_template('update_trip.html', trip_data=original_trip_data, trip_id=trip_id, page=page)

        # Attempt to update the trip in the database
        result = database.update_trip(trip_id, trip_data)
        if result['success']:
            flash('Trip updated successfully!', 'success')
            return redirect(url_for('show_trips'))
        else:
            flash('Failed to update trip: ' + result['error'], 'error')
            # If update fails, render the page with the data that was attempted to be saved
            return render_template('update_trip.html', trip_data=trip_data, trip_id=trip_id, page=page)
    else:
        return render_template('update_trip.html', trip_data=original_trip_data, trip_id=trip_id, page=page)
```

/templates/welcome.html

```
{% include 'top.html' %}
<body>
    <div style="text-align: center; margin-top: 40vh;">
        <h1 class="page-title" style="font-size: 5vw;">Welcome, {{ session.name }}</h1>
        <!-- Existing button to show all trips -->
        <a href="{{ url_for('show_trips') }}" class="btn btn-primary" style="font-size: 2vw; margin-top: 20px;">Show All Trips</a>
    </div>
</body>
```

/templates/list_trips.html

```
{% include 'top.html' %}
<body>
    <div class="container">
        <h1>{{ page.title }}</h1>

        <!-- Button to filter trips -->
        <a href="{{ url_for('filter_trips') }}" class="btn btn-secondary" style="margin: 5px;">Filter Trips</a>
        {% if session['isadmin'] %}
            <!-- Button to add a new trip -->
            <a href="{{ url_for('add_trip') }}" class="btn btn-secondary" style="margin: 5px;">Add New Trip</a>
        {% endif %}
        <!-- Button to view report -->
        <a href="{{ url_for('view_report') }}" class="btn btn-secondary" style="margin: 5px;">View Report</a>

        {% if trips %}
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Trip ID</th>
                        <th>Card ID</th>
                        <th>Travel Date</th>
                        <th>Entry Station ID</th>
                        <th>Exit Station ID</th>
                        <th>Trip Start Time</th>
                    </tr>
                </thead>
```

```
                    <tbody>
                        {% for trip in trips %}
                            <tr>
                                <td>{{ trip.tripid }}</td>
                                <td>{{ trip.cardid }}</td>
                                <td>{{ trip.traveldate }}</td>
                                <td>{{ trip.entrystationid }}</td>
                                <td>{{ trip.exitstationid }}</td>
                                <td>{{ trip.tripstarttime }}</td>
                                <!-- Adding the Update button to a new table data cell -->
                                {% if session['isadmin'] %}
                                <td>
                                    <a href="{{ url_for('update_trip_page', trip_id=trip.tripid) }}" class="btn btn-primary">Update</a>
                                <!-- Delete button -->
                                <form action="{{ url_for('delete_trip_route') }}" method="post" style="display: inline;" onsubmit="return confirm('Are you sure you want to delete the trip with ID {{ trip.tripid }}?');">
                                    <input type="hidden" name="trip_id" value="{{ trip.tripid }}">
                                    <button type="submit" class="btn btn-danger">Delete</button>
                                </form>
                                </td>
                            {% endif %}
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            {% else %}
                <p>No trips to display.</p>
            {% endif %}
        </div>
</body>
```

/templates/report.html

```
{% include 'top.html' %}
<body>
    <div class="container">
        <h1>{{ page.title }}</h1>
        <div class="report-container">
            {% if report_data %}
                <table class="table">
```

```
                        <thead>
                            <tr>
                                <th>Entry Station ID</th>
                                <th>Number of Trips</th>
                            </tr>
                        </thead>
                        <tbody>
                            {% for record in report_data %}
                            <tr>
                                <td>{{ record.entrystationid }}</td>
                                <td>{{ record.trip_count }}</td>
                            </tr>
                            {% endfor %}
                        </tbody>
                    </table>
                {% else %}
                    <p>No data available to display for the report.</p>
                {% endif %}
            </div>
            <!-- Add a return button -->
            <a href="{{ url_for('show_trips') }}" class="btn btn-secondary">Return to Trips List</a>
        </div>
</body>
```

/templates/filter_trips.html

```
{% include 'top.html' %}
<!-- This is the new filter_trips.html page -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ page.title }}</title>
    <!-- Include any required CSS or JavaScript -->
</head>
<body>
    <div class="container">
        <h1>{{ page.title }}</h1>
        <!-- Create a form to submit the date -->
        <form action="{{ url_for('filter_trips') }}" method="post">
            <div class="form-group">
                <label for="date">Select Date:</label>
                <input type="date" class="form-control" id="date" name="date">
            </div>
```

```
                <button type="submit" class="btn btn-primary">Filter Trips</button>
                <!-- Add a return button -->
                <a href="{{ url_for('show_trips') }}" class="btn btn-secondary">Return to Trips List</a>
            </form>
        </div>
</body>
</html>
```

/templates/add_trip.html

```
{% include 'top.html' %}
{% block content %}

<!-- Form Start -->
<form action="{{ url_for('add_trip') }}" method="post">
    <div class="form-group">
        <label for="cardid">Card ID:</label>
        <input type="text" class="form-control" id="cardid" name="cardid"
                required value="{{ request.form.cardid or '' }}">
    </div>
    <div class="form-group">
        <label for="traveldate">Travel Date:</label>
        <input type="date" class="form-control" id="traveldate" name="traveldate"
                required value="{{ request.form.traveldate or '' }}">
    </div>
    <div class="form-group">
        <label for="entrystationid">Entry Station ID:</label>
        <input type="text" class="form-control" id="entrystationid" name="entrystationid"
                required value="{{ request.form.entrystationid or '' }}">
    </div>
    <div class="form-group">
        <label for="exitstationid">Exit Station ID:</label>
        <input type="text" class="form-control" id="exitstationid" name="exitstationid"
                required value="{{ request.form.exitstationid or '' }}">
    </div>
    <div class="form-group">
        <label for="tripstarttime">Trip Start Time (HH:MM:SS):</label>
        <input type="text" class="form-control" id="tripstarttime" name="tripstarttime"
                placeholder="HH:MM:SS" required pattern="\d{2}:\d{2}:\d{2}"
                value="{{ request.form.tripstarttime or '' }}">
    </div>

    <button type="submit" class="btn btn-primary">Add Trip</button>
```

```
    <!-- Add a return button -->
    <a href="{{ url_for('show_trips') }}" class="btn btn-secondary">Return to Trips List</a>
</form>
<!-- End of form -->

{% endblock %}
```

/templates/update_trip.html

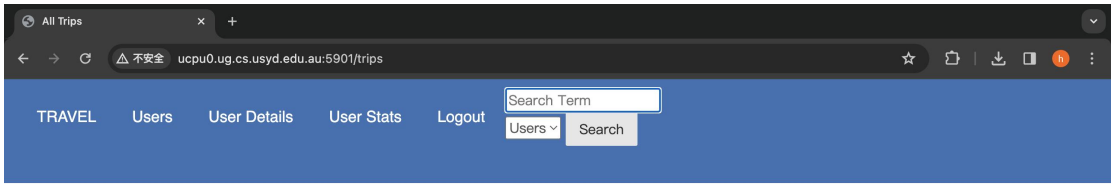
```
{% include 'top.html' %}
{% block content %}
<!-- Display trip details -->
{% if trip_data %}
<div class="trip-details">
    <h2>Trip Details:</h2>
    <p>Card ID: {{ trip_data.cardid or 'N/A' }}</p>
    <p>Travel Date: {{ trip_data.traveldate or 'N/A' }}</p>
    <p>Entry Station ID: {{ trip_data.entrystationid or 'N/A' }}</p>
    <p>Exit Station ID: {{ trip_data.exitstationid or 'N/A' }}</p>
    <p>Trip Start Time: {{ trip_data.tripstarttime or 'N/A' }}</p>
</div>
{% endif %}

<!-- Forms for updating itineraries -->
<form action="{{ url_for('update_trip_page', trip_id=trip_id) }}" method="post">
    <input type="hidden" name="trip_id" value="{{ trip_id }}">
    <div class="form-group">
        <label for="cardid">Card ID:</label>
        <input type="text" class="form-control" id="cardid" name="cardid"
                    required value="{{ trip_data.cardid or '' }}">
    </div>
    <div class="form-group">
        <label for="traveldate">Travel Date:</label>
        <input type="date" class="form-control" id="traveldate" name="traveldate"
                    required value="{{ trip_data.traveldate or '' }}">
    </div>
    <div class="form-group">
        <label for="entrystationid">Entry Station ID:</label>
        <input type="text" class="form-control" id="entrystationid" name="entrystationid"
                    required value="{{ trip_data.entrystationid or '' }}">
    </div>
    <div class="form-group">
        <label for="exitstationid">Exit Station ID:</label>
        <input type="text" class="form-control" id="exitstationid" name="exitstationid"
```

```
                required value="{{ trip_data.exitstationid or '' }}">
    </div>
    <div class="form-group">
        <label for="tripstarttime">Trip Start Time (HH:MM:SS):</label>
        <input type="text" class="form-control" id="tripstarttime" name="tripstarttime"
                placeholder="HH:MM:SS" required pattern="\d{2}:\d{2}:\d{2}"
                value="{{ trip_data.tripstarttime or '' }}">
    </div>
    <button type="submit" class="btn btn-primary">Update Trip</button>
    <a href="{{ url_for('show_trips') }}" class="btn btn-secondary">Return to Trips List</a>
</form>
{% endblock %}
```

PartC

In Part C, I created a new table, but I used another table when doing bounds checking on the data. For example, the cardid in the trips table that I am responsible for refers to the cardid in the OpalCards table, so when I do data checking, I will first make a judgment on the cardid to see if it is in the OpalCards table. The same goes for entrystationid and exitstationid. I will check whether the required stationid exists in the Stations table where they are located.
Among Flask's extra features:

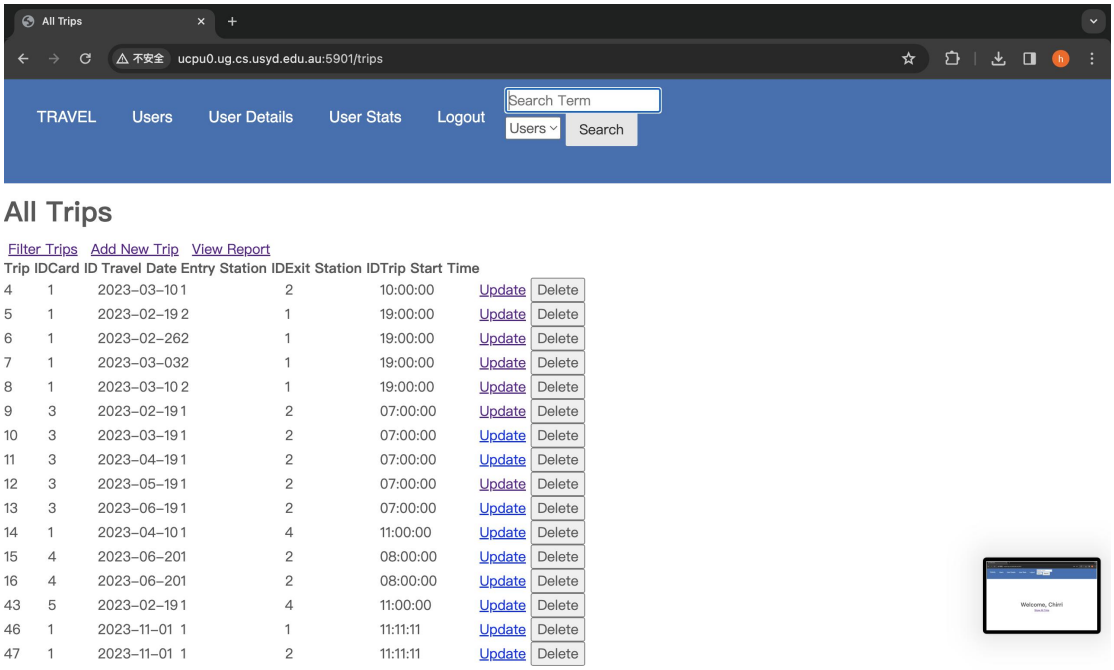
1. Add different functions according to whether the user is an administrator or not



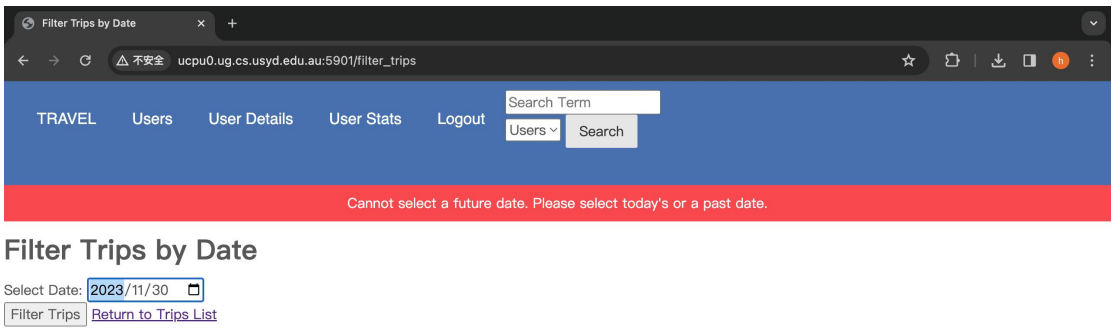
All Trips

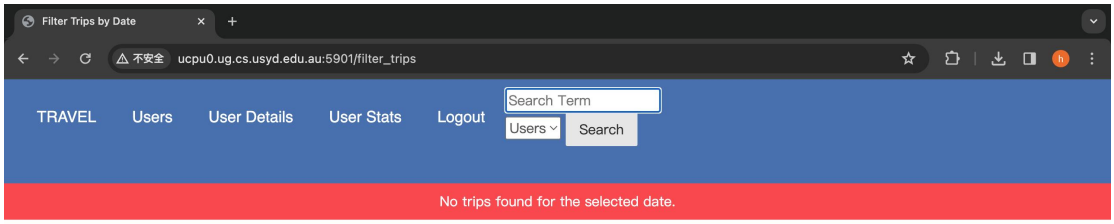
Filter Trips View Report

Trip ID	Card ID	Travel Date	Entry Station ID	Exit Station ID	Trip Start Time
4	1	2023-03-10	1	2	10:00:00
5	1	2023-02-19	2	1	19:00:00
6	1	2023-02-26	2	1	19:00:00
7	1	2023-03-03	2	1	19:00:00
8	1	2023-03-10	2	1	19:00:00
9	3	2023-02-19	1	2	07:00:00
10	3	2023-03-19	1	2	07:00:00
11	3	2023-04-19	1	2	07:00:00
12	3	2023-05-19	1	2	07:00:00
13	3	2023-06-19	1	2	07:00:00
14	1	2023-04-10	1	4	11:00:00
15	4	2023-06-20	1	2	08:00:00
16	4	2023-06-20	1	2	08:00:00
43	5	2023-02-19	1	4	11:00:00
46	1	2023-11-01	1	1	11:11:11
47	1	2023-11-01	1	2	11:11:11



2. The increased range for determining time must be today and before today. For no selection, selecting a date with no data and a date after selection have different flash error messages. After success, there will be a flash message showing success.





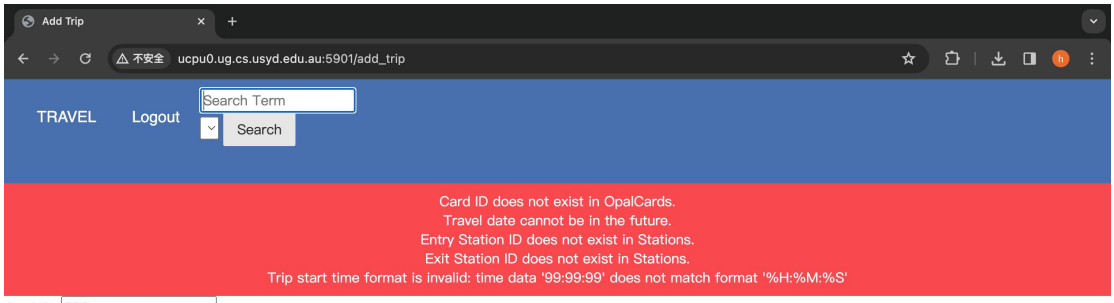
Filter Trips by Date

Select Date: 年 /月/日

Filter Trips Return to Trips List

The date is not shown in English because it will be changed according to the system language.

3. All columns in the table have corresponding error messages. After success, a flash message will display successfully.



Card ID: 999

Travel Date: 2023/11/30

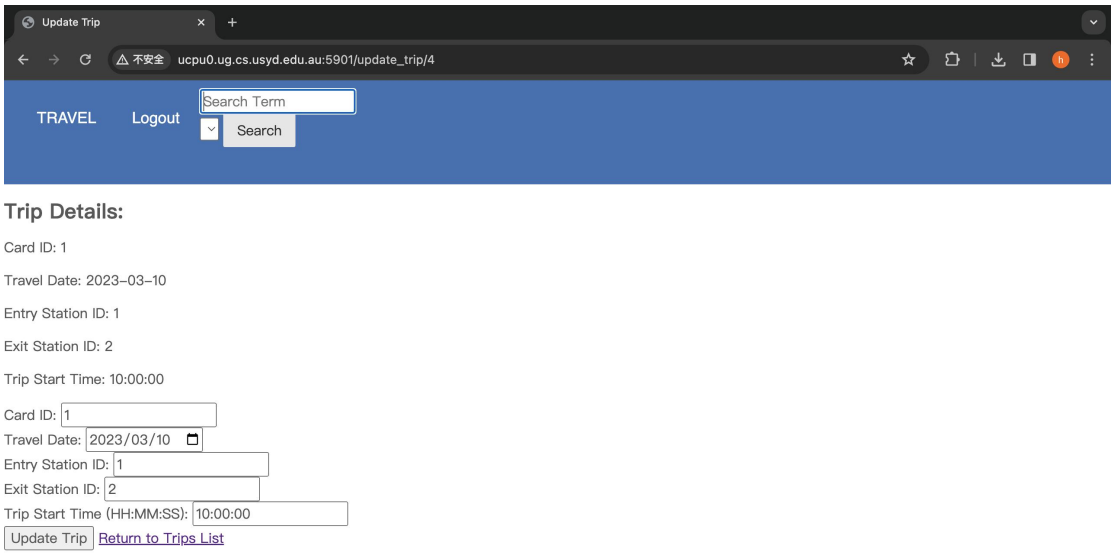
Entry Station ID: 999

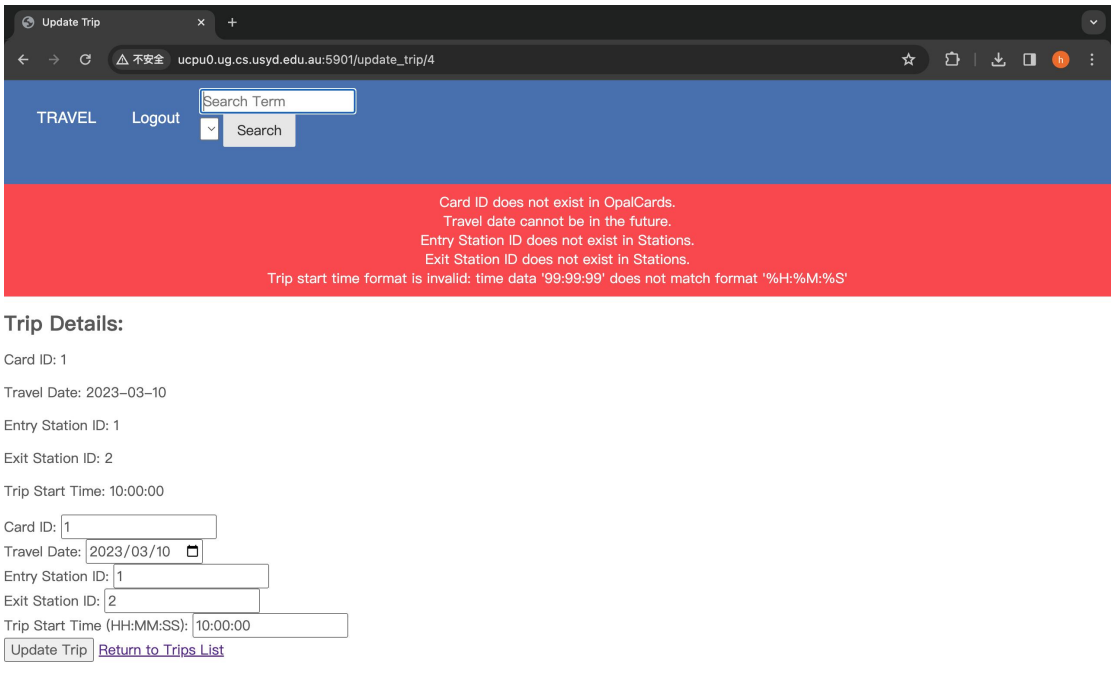
Exit Station ID: 999

Trip Start Time (HH:MM:SS): 99:99:99

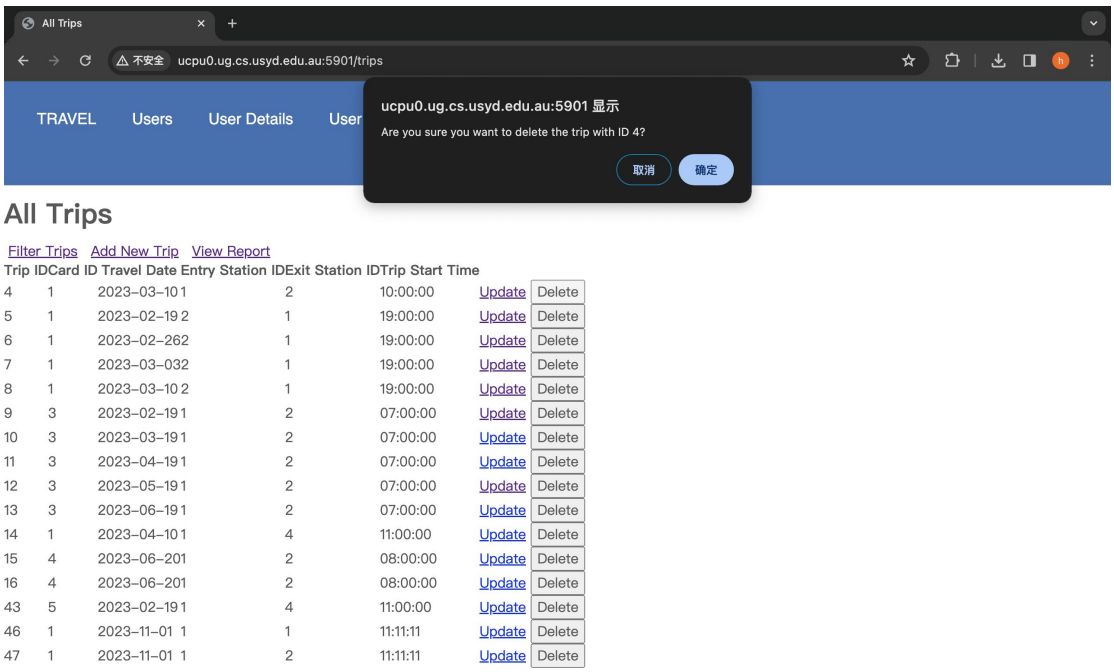
Add Trip Return to Trips List

4. During update, for the sake of user-friendliness, when the user clicks update of the corresponding data, the data will be displayed above and the data will be automatically filled in. It is convenient for users to not need to re-enter the remaining data if they only need to change part of the data. And the data is stored above, which makes it convenient for users to observe the data without having to operate back and forth. At the same time, there will also be an error report for each data, and after an error, the data will be restored to its original appearance, making it less likely for users to be confused. After success, there will be a flash message showing success.





5. When deleting, for the sake of user-friendliness, you will be prompted to confirm the deletion, and the trip ID that you clicked to delete will be displayed. After success, a flash message will show success.



Part D

Strengths

Modularity:

The code uses template inheritance and modularization (for example, include 'top.html'), which helps reduce duplication and improve maintainability.

Routing and database logic are clearly separated and follow the principles of MVC (Model-View-Controller) architecture, making management and expansion more convenient.

Readability and maintainability:

The HTML template uses the Jinja2 template language, which improves the readability of the code through template tags and filters.

Flask's route decorator makes the mapping between URLs and Python functions intuitive and clear.

The UI/UX of a system can be enhanced by adding additional front-end technologies (such as JavaScript frameworks) without the need to refactor the back-end.

safety:

When the form is submitted, a CSRF protection token is used to prevent cross-site request forgery attacks.

The deletion operation requires user confirmation to reduce the possibility of misoperation.

Using SQLAlchemy as an ORM makes it easy to perform database queries and operations without

writing raw SQL, reducing the risk of SQL injection attacks.

user experience:
Responsive design, such as font-size: 5vw in the welcome page, helps maintain consistency across different devices.
The form includes some basic validation (such as regular expression validation for time format).
The application includes features such as dynamic filtering, report generation and CRUD operations, providing a comprehensive user interface for managing trips.

Permission control:
The system displays different functions based on the user's administrator status, which is reflected in list_trips.html. Administrators can add, update, and delete trips.

limitation

user interface:
The interface aesthetics and user experience may be relatively basic and may require more modern front-end technology and design to improve.

Code duplication:
Certain HTML elements, such as the back button, are repeated across multiple templates and could probably be optimized further by creating reusable components.

Front-end and back-end coupling:
The current design has a high degree of coupling between the front-end view and the back-end logic. Once the business logic changes, the front-end and back-end codes may need to be modified at the same time.

Scalability:
For large-scale applications, a single database file may not be sufficient and a more robust database solution may need to be considered.