MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG352

Project Step 5

REPORT

Egemen Aksöz-- 2315083

Kaan Tandogan -- 2316784

PROJECT DESCRIPTION

"Hermes Car Pooling" is a carpooling website designed to facilitate shared rides for registered users, enabling them to act either as drivers or passengers. The platform allowsdrivers to offer rides in their vehicles and specify preferences such as no smoking or no eating, which passengers must agree to when joining the ride. Passengers can browse available trips, view details like the number of available seats and travel conditions, and choose a ride that suits their needs. The system maintains a record of all user details, trip information, and vehicle specifications to ensure a smooth matchmaking process.

Additionally, passengers can leave reviews for drivers, enhancing trust and safety within the community. "Hermes Car Pooling" will be implemented as a web application accessible through a browser, aiming to provide a user-friendly and efficient platform for managing carpooling activities. This system will be named "Hermes Car Pooling" to reflect its key features and capabilities.

DATA REQUIREMENTS

User: A user can be both driver and passenger but before doing that s(he) needs to register to the site using their name, surname, e-mail address, password and phone number. The system afterwards, records the user. However, it should be taken into consideration that a person can only sign up once with an e-mail.

Driver: A driver is also a user in our system, so the driver inherits the attributes of user. But for a user to be a driver in the Hermes Car Pooling s(he) needs to have a driver license. A driver in the system is the person who opens a trip and accepts passengers. S(he) also can have multiple preferences such as no smoking, no eating atthe car etc. If a user is adriver s(he) has to have at least one car assigned to itself.

Passenger: A passenger is a standard user with no extra data. Passenger is the user who takes trip. And therefore, a passenger inour system only has the data of a user.

Trip: When the driver creates a ride, s(he) can decide how many people he takes, but he can’t take more passengers than the car's limit. Each trip has a starting and endingplace, and a suggested price range is calculated from it according to the distance. The driver can decide on a payment that is within 10 percent of the suggested price. Afterwards the trip is assigned a unique trip-ID. Our database should hold the number of passengers, pricing, starting and ending address of trip and the id of trip.

Review: After a ride, passengers of “Hermes Car Pooling” may or may not give reviewto the driver. If the passenger wishes, s(he) can either only give a general rating or s(he) can only add comments. Afterwards, our system assigns an ID to the review.

Vehicle: A vehicle in the “Hermes Car Pooling "has to be related to a driver. All vehicles have a unique plate number by design. A vehicle can only belong to the one driver. The details such as model, year, color, plate number are held in our database.

TRANSACTION REQUIREMENTS

Data entry

Enter the details of a new passenger (such as Kaan (Name), Tandogan (Surname), vEr7?Sec6re_P@s5w0rD (password), +532 352 63 82 (phone number), my_made_up_email@gmail.com (e-mail Has to be unique))

Enter the details of a new driver (such as Egemen (Name), Aksoz (Surname), vEr7?Sec6re_P@s5w0rD_the_two (password), +532 352 63 82 (phone number), my_second_made_up_email@gmail.com (e-mail Has to be unique), 12345678A (License no), nosmoking (preference 1), no cussing (preference 2))

Enter the details of a new vehicle. (Such as, “34 vov 9876”, white, 1980, Rolls Royce, 6)

Enter the details of a trip, from / to details, desired payment (Such as , ‘Kalkanlı’ , ‘Girne’ ,100$)

Data update/deletion

- Insert/ Update/ Delete the details of passenger.
- Insert/ Update/ Delete the details of driver.
- Insert/ Delete the details of vehicles.

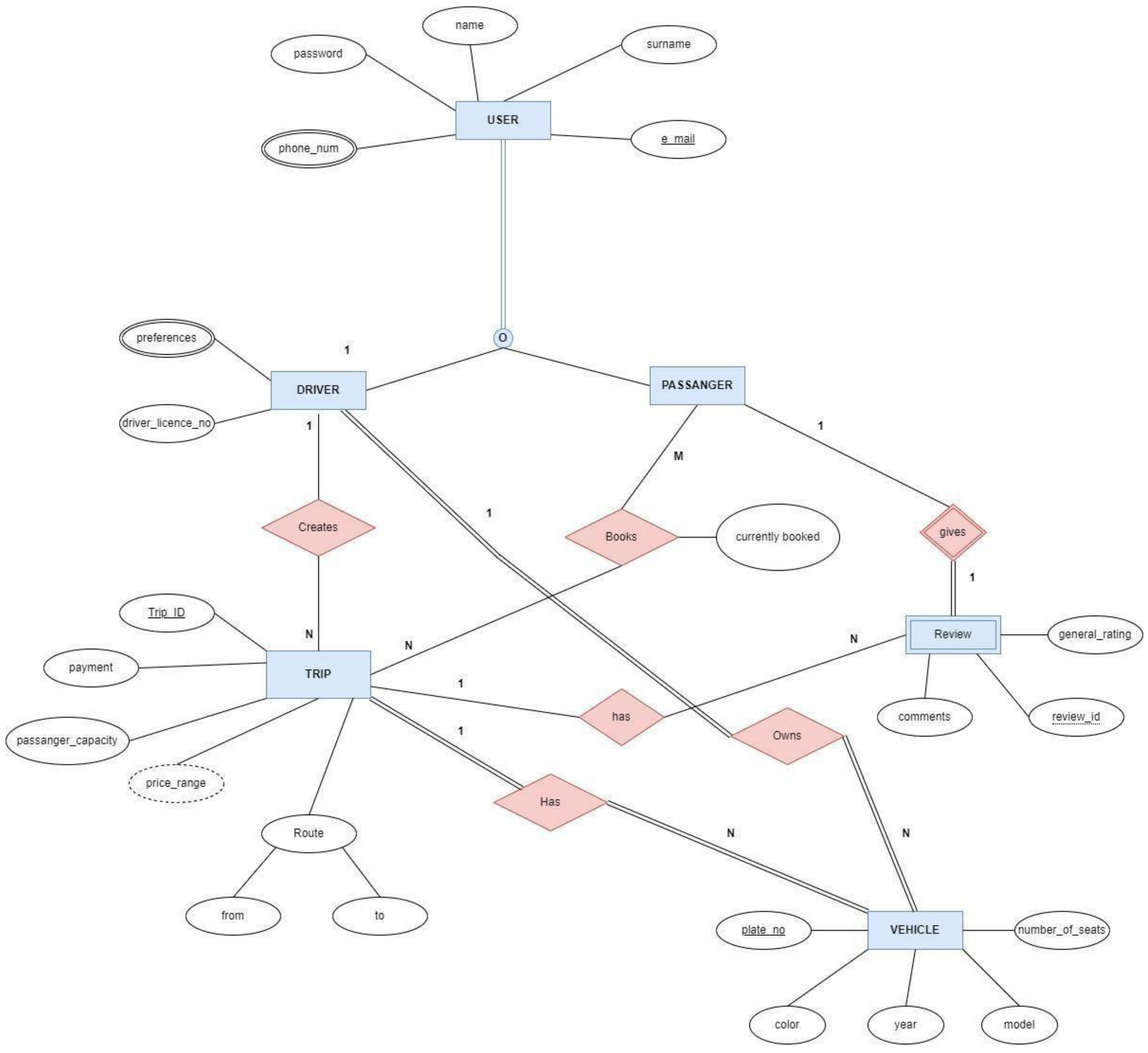
- Insert/ Update/ Delete the details of the trip.

Data Queries

- Identify the name, surname, phone number, password, and email of each passenger
- Identify how many trips each passenger has booked
- List the reviews given by each passenger
- Identify the preferences of each driver

- List the plate no, color, year, and model of each vehicle owned by a driver
- Identify the average rating level and the number of reviews for each driver
- Identify the trip id, start location, end location, and vehicle details for each trip
- List id, comments, and rating level of each review given to a specific driver
- Identify the pay amount and payment number of each payment
- Identify the total payment amount made by each passenger
- List all the trips based on given from/to details from the user.

ERR Diagram



Assumptions

- We assume that every passenger gives a review to the driver.
- We assumed every passenger in our system books at least one trip.

Table After Mapping

- **USER**: (e_mail)[**PK**], name, surname, password
- **PHONE NUMBER**: (phone_number, user_e_mail[FK:USER:e_mail]) [**PK**]
- **PASSANGER**: (passanger_e_mail[FK:USER:e_mail]) [**PK**]
- **DRIVER**: (driver_e_mail[FK:USER:e_mail]) [**PK**],driver_licence_no
- **PREFERENCES**: (driver_e_mail_for_preference[FK:DRIVER:driver_e_mail], preferences) [**PK**]
- **TRIP**: (trip_id) [**PK**], payment, passanger_capacity,
  from, to,
  driver_e_mail_for_trip[FK:DRIVER:driver_e_mail]
- **VEHICLE**: (plate_no)[**PK**], color, year, model, number_of_seats,
  driver_e_mail_for_vehicle[FK:DRIVER:driver_e_mail], trip_id_for_vehicle
  [FK:TRIP:trip_id]
- **REVIEW**: (review_id,
  passanger_e_mail_of_review[FK:PASSANGER:passanger_e_mail],
  trip_id_of_review[FK:TRIP:trip_id])[**PK**], comment, general_rating,
- **BOOKS**:
  (trip_id_of_booking[FK:TRIP:trip_id],passanger_e_mail_for_booking[FK:PASSANGER:passa
  nger_e_ mail]) [**PK**], currently_booked

Functional Dependencies

- FD1: {e-mail} -> {name, surname, password}
- FD2: {driver_ e_ mail} -> {driver _ license _
  no}
- FD3: {driver _ e _ mail} -> {name,
  surname, password}
- FD4: {driver _ e _ mail,} -> {preferences}
- FD5: {plate_ no} -> {color, year, model, number _of_ seats, driver _e_ mail_ for_ vehicle, trip _ id
  _ for _vehicle}
- FD6: {review _id, passenger _ e_ mail _ of_ review, trip_ id_ of_ review} -> {comment, general
  _rating}
- FD7: {trip _id _of _booking, passenger _e _mail _for _ booking} -> {currently _ booked}

Normalisations

1NF –

By ensuring that each record is unique (typically via a primary key), 1NF avoids duplicate rows ina table, which can lead to data inconsistencies. This means that our schema is in 2NF

2NF –

The attribute is already in 1NF and has no partial reliance, implying that all non- key attributes are entirely functionally dependent on the composite primary key as a whole.

3NF –

There are no transitive dependencies present. Every non-key attribute in each table is dependent solely on the primary key. Furthermore, there is no dependency among non-key attributes themselves. Therefore, this schema is in 3NF.

BCNF –

The tables provided meets the BCNF conditions for every determinant of our tables are a superkey.

Table Creation

-- Dropped the tables if they exist

```
DROP TABLE IF EXISTS bookings
CASCADE;

DROP TABLE IF EXISTS reviews
CASCADE;

 DROP TABLE IF EXISTS trips CASCADE;
DROP TABLE IF EXISTS vehicles
CASCADE;

DROP TABLE IF EXISTS preferences
CASCADE;

DROP TABLE IF EXISTS drivers CASCADE;
DROP TABLE IF EXISTS passengers
CASCADE;

DROP TABLE IF EXISTS phone_numbers
CASCADE; DROP TABLE IF EXISTS users
CASCADE;
```

```
-- We created the users table with respect to the attributes we determined
earlier.

CREATE TABLE users (

    email VARCHAR(255) PRIMARY
    KEY,name VARCHAR(255),

    surname
    VARCHAR(255),
    password
    VARCHAR(255)

);
-- We created the phone_numbers
tableCREATE TABLE phone_numbers
(

    phone_number
    VARCHAR(20),user_email
    VARCHAR(255),

    PRIMARY KEY (phone_number, user_email),
    FOREIGN KEY (user_email) REFERENCES
    users(email)

);
 -- We created the drivers table with respect to the attributes we determined
earlier.CREATE TABLE drivers (

    driver_email VARCHAR(255) PRIMARY
    KEY,driver_license_no VARCHAR(50),

    FOREIGN KEY (driver_email) REFERENCES users(email)
);
 -- We created the passengers table with respect to the attributes we determined
earlier.CREATE TABLE passengers (

    passenger_email VARCHAR(255) PRIMARY KEY,

    FOREIGN KEY (passenger_email) REFERENCES users(email)

);
```

```
-- We created the preferences table with respect to the attributes we determined earlier.
CREATE TABLE preferences (

    driver_email
    VARCHAR(255),
    preference
    VARCHAR(255),

    PRIMARY KEY (driver_email, preference),
    FOREIGN KEY (driver_email) REFERENCES drivers(driver_email)

);
```

```
-- We created the vehicles table with respect to the attributes we determined
earlier.CREATE TABLE vehicles (

    plate_no VARCHAR(20) PRIMARY
    KEY,color VARCHAR(50),

    year INT,

    model VARCHAR(50),

    number_of_seats INT,
    driver_email
    VARCHAR(255),

    FOREIGN KEY (driver_email) REFERENCES drivers(driver_email)

);


-- We created the trips table with respect to the attributes we determined
earlier.CREATE TABLE trips (

    trip_id SERIAL PRIMARY KEY,

    from_location
    VARCHAR(255),to_location
    VARCHAR(255),
    passenger_capacity INT,
    payment NUMERIC(10,2),
    driver_email
    VARCHAR(255),

    FOREIGN KEY (driver_email) REFERENCES drivers(driver_email)

);


-- We created the reviews table with respect to the attributes we determined
earlier.CREATE TABLE reviews (

    review_id SERIAL,
    passenger_email
    VARCHAR(255),trip_id INT,

    comment TEXT,
    general_rating
    INT,
    PRIMARY KEY (review_id, passenger_email, trip_id),
    FOREIGN KEY (passenger_email) REFERENCES
    passengers(passenger_email),FOREIGN KEY (trip_id) REFERENCES
    trips(trip_id)
);

-- We created the bookings table with respect to the attributes we determined earlier.
```

```
CREATE TABLE
bookings (trip_id INT,

    passenger_email VARCHAR(255),
    PRIMARY KEY (trip_id,
    passenger_email),

    FOREIGN KEY (trip_id) REFERENCES trips(trip_id),

    FOREIGN KEY (passenger_email) REFERENCES passengers(passenger_email)

);
```

Data Insertion

```
-- We inserted some made-up users here.

INSERT INTO users (email, name, surname, password) VALUES
('yyelizyesilada@metu.edu.tr', 'Yeliz', 'Yesilada', 'yel.23'),
('kalayci@gmail.com', 'Emre', 'Kalayci', 'emre1234'),
('fenerbahce01@gmail.com', 'Ali', 'Veli', 'fb1907'),

('tata11@hotmail.com', 'Tata', 'Tutu', 'tataPass'),
('semicoln@hotmail.com', 'Semic', 'Oln',
'semicolon'),('besiktas33@outlook.com', 'Kara',
'Kartal', 'bjk1903');


-- We inserted some made-up phone numbers for the users
INSERT INTO phone_numbers (phone_number, user_email)
VALUES('324324', 'yyelizyesilada@metu.edu.tr'),

('567890', 'kalayci@gmail.com'),
('123456', 'fenerbahce01@gmail.com');


-- We inserted some made-up driver details for users

INSERT INTO drivers (driver_email, driver_license_no)
VALUES('yyelizyesilada@metu.edu.tr', 'ee3132'),
('fenerbahce01@gmail.com', 'fb2020'),
('tata11@hotmail.com', 'tt3030');
```

```
-- We inserted some made-up passenger details for the same
usersINSERT INTO passengers (passenger_email) VALUES
('kalayci@gmail.com'),

('semicoln@hotmail.com'),
('besiktas33@outlook.com');
```

```
-- We inserted some made-up vehicle details for drivers
INSERT INTO vehicles (plate_no, color, year, model, number_of_seats, driver_email) VALUES
('cc665', 'Blue', 2020, 'Tofask', 4, 'fenerbahce01@gmail.com'),
('aa112', 'Red', 2019, 'Fiat', 4, 'tata11@hotmail.com');

-- We inserted some made-up trip details
INSERT INTO trips (from_location, to_location, passenger_capacity, payment, driver_email)
VALUES
('Kalkanli', 'Guzelyurt', 4, 100.00, 'tata11@hotmail.com'),
('Nicosia', 'Kyrenia', 3, 80.00, 'fenerbahce01@gmail.com');

-- Inserted booking for a trip by a passenger
INSERT INTO bookings (trip_id, passenger_email)
VALUES(1, 'semicoln@hotmail.com'),
(2, 'kalayci@gmail.com');

-- Inserted review for the trip
INSERT INTO reviews (passenger_email, trip_id, comment, general_rating)
VALUES('besiktas33@outlook.com', 1, 'Good ride, very kind and helpful !', 5),
('kalayci@gmail.com', 2, 'Smooth drive, comfortable car.', 4);


-- Inserted driver preferences
INSERT INTO preferences (driver_email, preference)
VALUES('yyelizyesilada@metu.edu.tr', 'No Smoking'),
('yyelizyesilada@metu.edu.tr', 'Pets Allowed'),
('fenerbahce01@gmail.com', 'No Smoking'),
('fenerbahce01@gmail.com', 'Music Allowed'),
('tata11@hotmail.com', 'No Smoking'),
('tata11@hotmail.com', 'No Food');
```

Data Deletion and Update

```
-- Update the details of trip
```

```
UPDATE trips
SET to_location = 'Larnaca', passenger_capacity
= 5WHERE trip_id = 1;

-- We thought updating password is real-life so we put this as
well.UPDATE users
SET password = 'newpass'
WHERE email = 'semicoln@hotmail.com';



-- Update details of
vehicleUPDATE vehicles
SET color = 'Green', model = 'Tesla Model
3'WHERE plate_no = 'aa112';



-- Delete a vehicle
DELETE FROM
vehicles
WHERE plate_no = 'cc665';

-- Delete a booking
DELETE FROM
bookings
WHERE trip_id = 1 AND passenger_email = 'semicoln@hotmail.com';

-- Delete phone numbers of the
userDELETE FROM
phone_numbers
WHERE user_email = 'yyelizyesilada@metu.edu.tr';
```

Data Manipulation Queries

1.) With this query we did "List all the trips based on given from/to details from the user." This query is created for analyzing the payment amount with respect to the number of passangers, from/to locations.

```
SELECT t.trip_id, t.from_location, t.to_location, t.payment, t.passenger_capacity
```

```
FROM trips t
WHERE t.from_location = 'Nicosia' AND t.to_location = 'Kyrenia'
ORDER BY t.from_location, t.to_location;
```

2.) With this query we did the “Identify the trip id, start location, end location, and vehicledetails for each trip.” This query might come handy when analyzing what kinds of vehicles travelto specific distances.

```
SELECT t.trip_id, t.from_location, t.to_location, v.plate_no, v.color, v.year,
v.modelFROM trips t
LEFT JOIN vehicles v ON t.driver_email =
v.driver_emailORDER BY t.trip_id;
```

3.) With this query, we obtained the “list of IDs, comments, and rating levels for each review given to a specific driver.” This query makes it possible to analyze a driver. This will be useful when we decide to solve the issues related to a low-ranking driver, as we can read the comments, we can understand what might be wrong with the driver.

```
SELECT r.review_id, r.comment,
r.general_ratingFROM reviews r
JOIN trips t ON r.trip_id = t.trip_id
WHERE t.driver_email =
'fenerbahce01@gmail.com'ORDER BY
r.review_id;
```

4.) With this query, we obtained the following information: 'the average rating level and the number of reviews for each driver.' This query allows us to assess customer satisfaction with each driver. The number of reviews is also useful to determine whether a driver is widely dislikedor if they are simply new and had a bad day.

```sql
SELECT d.driver_email, ROUND(AVG(r.general_rating), 2) AS avg_rating, COUNT(r.review_id) AS total_reviews

FROM drivers d
```

```
JOIN trips t ON d.driver_email =
t.driver_emailJOIN reviews r ON t.trip_id =
r.trip_id

GROUP BY d.driver_email;
```

5.) With this query, we obtained the following information: 'Identify the preferences of eachdriver.' This information allows customers to select a driver who meets their own preferences.

```
SELECT d.driver_email, array_agg(p.preference) AS preferences
FROM preferences p

JOIN drivers d ON d.driver_email =
p.driver_emailGROUP BY d.driver_email;
```

Discussions

Our platform consists of nine key tables, including bookings, reviews, trips, vehicles, preferences, drivers, passengers, phone_numbers, and users. The overall workload and performance of our system are directly influenced by the volume of users and the activity level within these tables.

When a large number of bookings, trips, and reviews are being processed simultaneously, the system could experience significant load on the bookings and trips tables. This could affect the speed and responsiveness of the platform, especially if there are numerous updates or queries happening at once.

Out of these tables, bookings, trips, and reviews are likely to experience the highest frequencyof updates and queries, as they represent the core activities on the platform. Conversely, tables like users, vehicles, and phone_numbers might experience fewer changes over time, primarily consisting of additions and occasional updates.

The size of each table will vary. For instance, the bookings, trips, and reviews tables are expectedto have a larger number of records, reflecting the ongoing activity and engagement on the platform. Meanwhile, tables like users, vehicles, and drivers might have relatively fewer records since they represent more static data.

Given that the platform caters to a specific region, the overall dataset might not be overwhelmingly large. This means that the system should be able to handle the typical workloadwithout significant performance issues. However, we still need to consider key optimization techniques to ensure efficient query processing.

For instance, the vehicles table has a foreign key linking it to the users table, so creating an index on the user_id column in the vehicles table could speed up JOIN operations and other related queries. Similarly, the bookings and trips tables reference multiple other tables through foreign keys, suggesting that indexing on the corresponding columns could be beneficial for performance during complex queries or when executing ORDER BY or GROUP BY operations.

These considerations help ensure that our system is both robust and responsive, even as the number of users and activities increases. Through effective indexing and careful design, we can maintain optimal performance and user experience.

Hermes Car Pooling

The “Hermes Car Pooling” is a carpooling application developed as a final part of the CNG352 course project. This application allows users to register, log in, and manage their carpooling activities such as adding trips, booking seats etc.

Functionalities

- **Initial Database Connection:** Our code should be able to connect to the database in our system. Implemented by Kaan Tandogan.
- **User Registration and Login**: Users can register with their email, name, surname, phone number, and password. They can log in using their credentials. Implemented by Kaan Tandogan.
- **Login Required**: An unlogged user shouldn’t be able to see anything other than login or registration pages. Implemented by Kaan Tandogan.
- **Logout**: Pops the session and redirects to login page. Implemented by Kaan Tandogan.
- **Edit Profile**: Users can edit their profile information, including name, surname, password, email, driver license number, and phone numbers. Implemented by Egemen Aksöz.
- **Trip Management**: Drivers can add trips with details such as “from” location, “to” location, car, and capacity. Users can book seats on available trips. Trip reservation by user is implemented by Egemen Aksöz. Trip addition by driver is implemented by Kaan Tandogan.
- **Vehicle Management**: Users can add and remove vehicles from their profile. Drivers are identified by having associated vehicles. Implemented by Kaan Tandogan.

- **Booking Management**: Users can book trips and the system sends an email to the driver with the booking details. Implemented by Egemen Aksöz. E-mail sending implemented Kaan Tandogan.
- **Trip Removal**: Users can remove trips they have added. Both drivers and passengers can remove trips. Implemented by Kaan Tandogan.
- **Email Notification**: An email should be sent to the driver after a user reserves a seat, providing trip and contact details. Implemented by Kaan Tandogan.

Technologies Used

- **Flask**: A python web framework.
- **PostgreSQL**: An open-source database system.
- **HTML/CSS**: Used for the frontend structure and design.
- **Python**: Core programming language for the backend of our website.
- **smtplib**: Python library for sending emails using the Simple Mail Transfer Protocol (SMTP).

Routes

1. **`/`**: Redirects to the login page.
2. **`/login`**: Handles user login.
3. **`/logout`**: Logs out the current user.
4. **`/register`**: Handles user registration.
5. **`/home`**: Displays the home page after login.
6. **`/edit_profile`**: Allows users to edit their profile information.
7. **`/add_trips`**: Allows drivers to add new trips.
8. **`/remove_trips`**: Allows users to remove trips they have added or booked.
9. **`/vehicle_operations`**: Allows users to add or delete their vehicles.
10. **`/add_vehicle`**: Allows users to add a new vehicle.
11. **`/get_trips`**: Displays trips available for booking.
12. **`/book_trip`**: Handles trip booking by users.

Usage

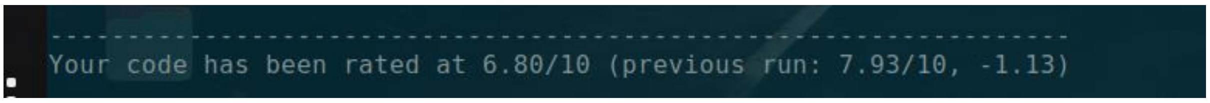
1. **Registration**: Visit **`/register`** to create a new account.

   a. Fill in the required fields: email, name, surname, phone number, and password.
   b. If registration is successful, you will be redirected to the login page.
2. **Login**: Visit **`/login`** to log in with your credentials.
   a. Enter your registered email and password.
   b. If login is successful, you will be redirected to the home page.
3. **Profile Management**: Edit your profile by visiting **`/edit_profile`**.
   a. Update your email, name, surname, password, and preferences if you are a driver.
   b. Save the changes to update your profile.
4. **Add Trips**: Drivers can add new trips by visiting **`/add_trips`**.
   a. Provide the trip details such as from location, to location, car, and capacity.
   b. Set the trip price range based on the distance between locations.
   c. Save the trip to make it available for booking.
5. **Remove Trips**: Users can remove trips they have added or booked by visiting **`/remove_trips`**.
   a. Select the trips you want to remove.
   b. Confirm the removal to delete the trips.
6. **Vehicle Management**: Add or remove vehicles by visiting **`/vehicle_operations`**.
   a. To add a vehicle, provide the vehicle details such as plate number, color, year, model, and number of seats.
   b. To remove a vehicle, select the vehicles you want to delete and confirm the deletion.
7. **Get Trips**: View and book available trips by visiting **`/get_trips`**.
   a. Provide the from location and to location to search for available trips.
   b. Book a trip by selecting it from the list of available trips.
8. **Book Trip**: Book a trip by submitting the form on the **`/get_trips`** page.
   a. Select the trip you want to book.
   b. Confirm the booking to reserve a seat on the trip.

Code Quality

Some of the tools I used are not compatible with Windows but all of them are compatible with Linux. So, I'm showing the Linux output here.

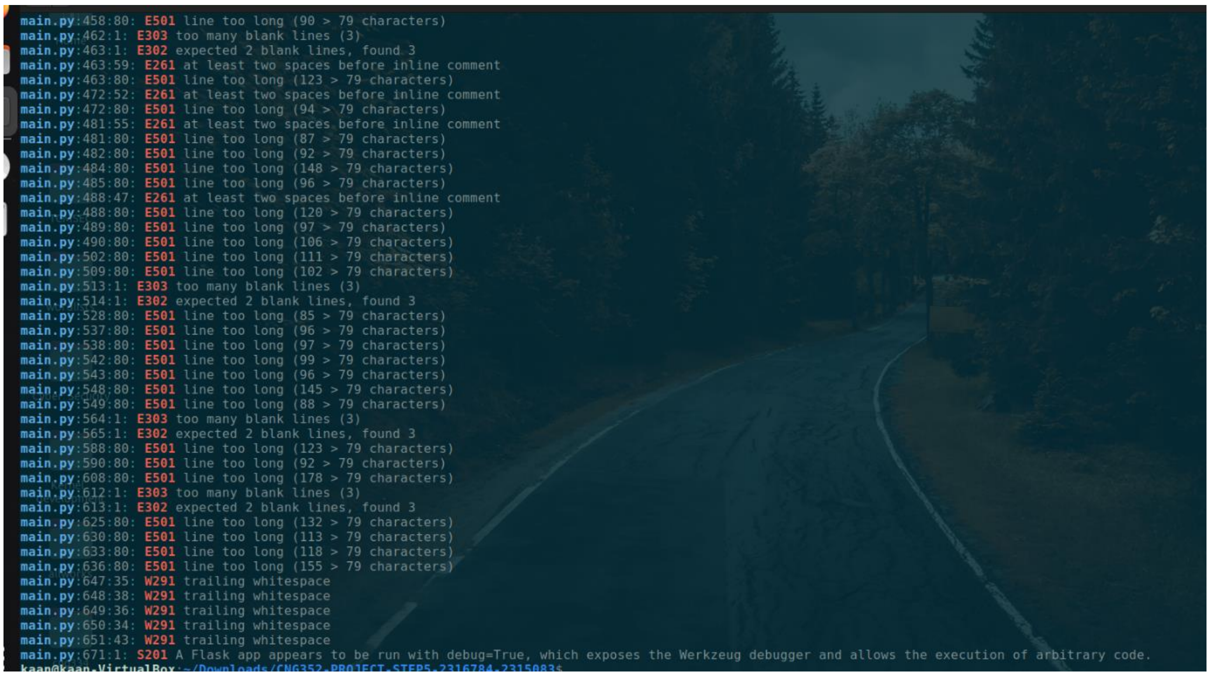
Pylint:



```
-------------------------------------------------------------------
Your code has been rated at 6.80/10 (previous run: 7.93/10, -1.13)
```

- **Description**: Pylint is used for code quality.
- **Output**: We got a score of 6.8/10. However, we received a lot of complaints about the length of the lines and blank spaces, which I don't think are significant problems. Please note that a tool called "black" modifies the code for the general standards. However, we opted out of using it because I whole-heartedly believe the code is more readable like this.

Flake8:



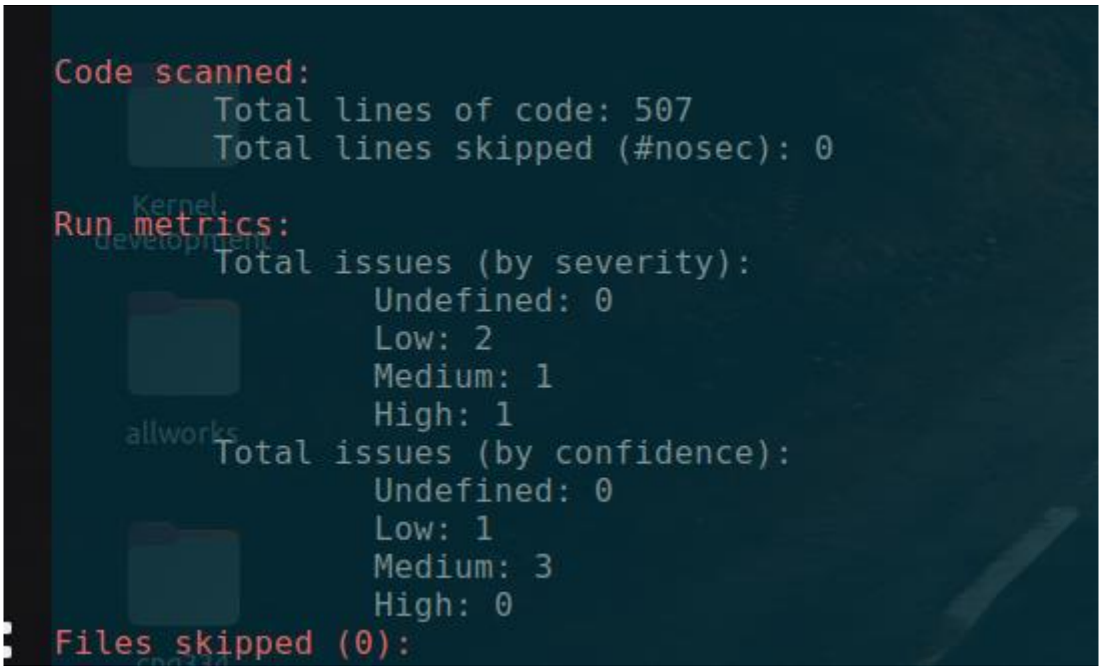
```
main.py:458:80: E501 line too long (90 > 79 characters)
main.py:462:1: E303 too many blank lines (3)
main.py:463:1: E302 expected 2 blank lines, found 3
main.py:463:59: E261 at least two spaces before inline comment
main.py:463:80: E501 line too long (123 > 79 characters)
main.py:472:52: E261 at least two spaces before inline comment
main.py:472:80: E501 line too long (94 > 79 characters)
main.py:481:55: E261 at least two spaces before inline comment
main.py:481:80: E501 line too long (87 > 79 characters)
main.py:482:80: E501 line too long (92 > 79 characters)
main.py:484:80: E501 line too long (148 > 79 characters)
main.py:485:80: E501 line too long (96 > 79 characters)
main.py:488:47: E261 at least two spaces before inline comment
main.py:488:80: E501 line too long (120 > 79 characters)
main.py:489:80: E501 line too long (97 > 79 characters)
main.py:490:80: E501 line too long (106 > 79 characters)
main.py:502:80: E501 line too long (111 > 79 characters)
main.py:509:80: E501 line too long (102 > 79 characters)
main.py:513:1: E303 too many blank lines (3)
main.py:514:1: E302 expected 2 blank lines, found 3
main.py:528:80: E501 line too long (85 > 79 characters)
main.py:537:80: E501 line too long (96 > 79 characters)
main.py:538:80: E501 line too long (97 > 79 characters)
main.py:542:80: E501 line too long (99 > 79 characters)
main.py:543:80: E501 line too long (96 > 79 characters)
main.py:548:80: E501 line too long (145 > 79 characters)
main.py:549:80: E501 line too long (88 > 79 characters)
main.py:564:1: E303 too many blank lines (3)
main.py:565:1: E302 expected 2 blank lines, found 3
main.py:588:80: E501 line too long (123 > 79 characters)
main.py:590:80: E501 line too long (92 > 79 characters)
main.py:608:80: E501 line too long (178 > 79 characters)
main.py:612:1: E303 too many blank lines (3)
main.py:613:1: E302 expected 2 blank lines, found 3
main.py:625:80: E501 line too long (132 > 79 characters)
main.py:630:80: E501 line too long (113 > 79 characters)
main.py:633:80: E501 line too long (118 > 79 characters)
main.py:636:80: E501 line too long (155 > 79 characters)
main.py:647:35: W291 trailing whitespace
main.py:648:38: W291 trailing whitespace
main.py:649:36: W291 trailing whitespace
main.py:650:34: W291 trailing whitespace
main.py:651:43: W291 trailing whitespace
main.py:671:1: S201 A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
kaan@kaan-VirtualBox:~/Downloads/CNG352-PROJECT-STEP5-2316784-2315083$
```

- **Description**: Flake8 checks the style guide enforcement and quality of the code.
- **Output**: We had a lot of warnings about lines being too long and blank lines. Please note that we have always used `app.run(debug=True)`, but it is not a good idea to have this when releasing the product. For now, it can stay.

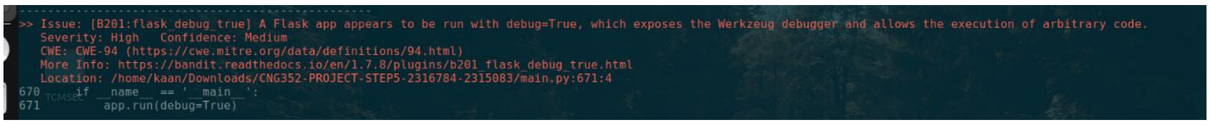
Code Security

Bandit:

- **Description**: Bandit is a security testing tool for Python.
- **Output**:



```
Code scanned:
        Total lines of code: 507
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 1
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 1
                Medium: 3
                High: 0
Files skipped (0):
```



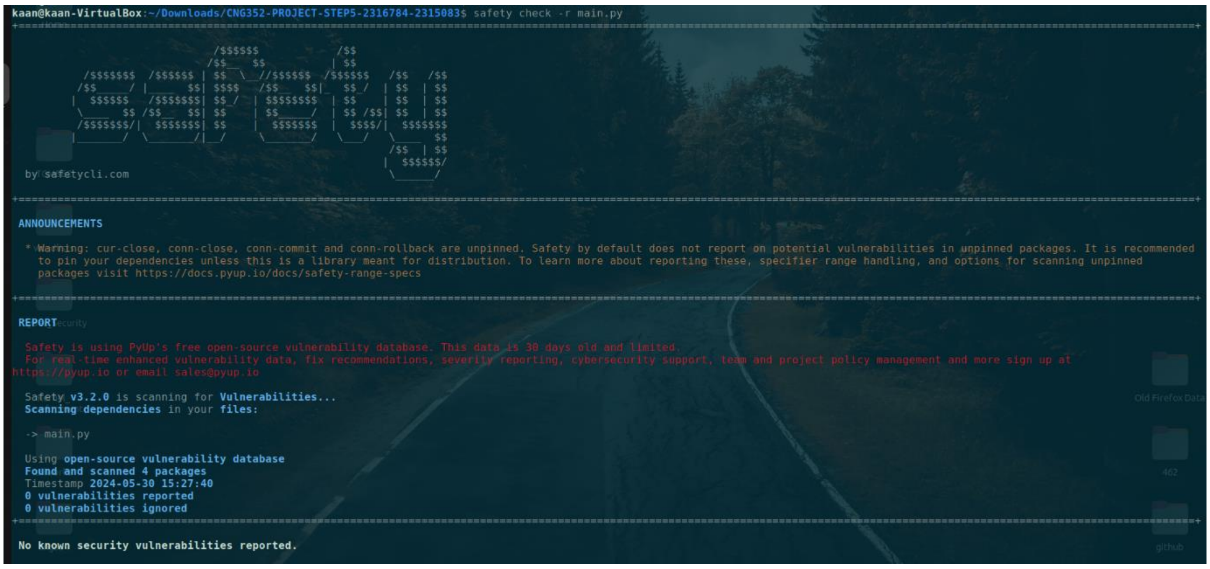
```
--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.7.8/plugins/b201_flask_debug_true.html
   Location: /home/kaan/Downloads/CNG352-PROJECT-STEP5-2316784-2315083/main.py:671:4
670	if __name__ == '__main__':
671	    app.run(debug=True)
```

- The high severity issue reported was that we were running the code with **`debug=True`**. As mentioned, this will be changed when releasing the code to the customer. For now, it is in development.
- The other two medium warnings were about storing passwords in the code. We won't do this in the release version. Currently, we are trying to demonstrate that the functionality works.

Safety:

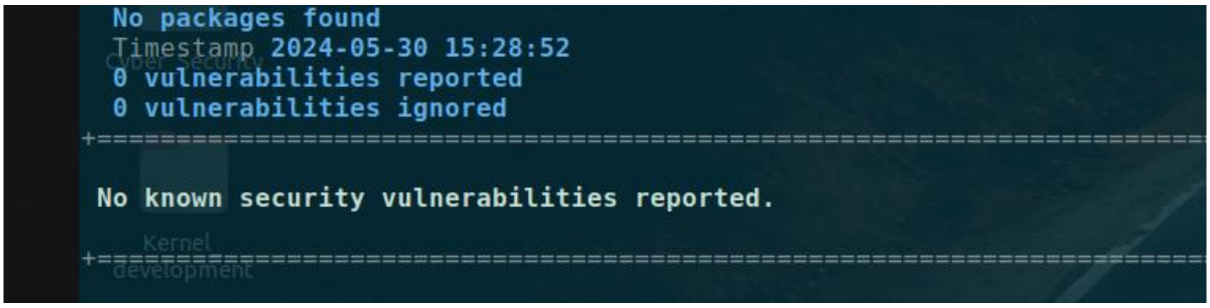
- **Description**: Safety checks your installed dependencies for known security vulnerabilities.

The safety check –r main.py didn’t find any vulnerabilities of the libraries imported in my code



```
kaan@kaan-VirtualBox:~/Downloads/CNG352-PROJECT-STEP5-2316784-2315083$ safety check -r main.py
+==========================================================================================================================================+

                               /$$$$$$            /$$
                              /$$__  $$          | $$
           /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$
          /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$
         |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$
          \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$
          /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$
         |_______/  \_______/|__/     \_______/   \___/   \____  $$
                                                          /$$  | $$
                                                         |  $$$$$$/
  by safetycli.com                                        \______/

+==========================================================================================================================================+

 ANNOUNCEMENTS

  * Warning: cur-close, conn-close, conn-commit and conn-rollback are unpinned. Safety by default does not report on potential vulnerabilities in unpinned packages. It is recommended
    to pin your dependencies unless this is a library meant for distribution. To learn more about reporting these, specifier range handling, and options for scanning unpinned
    packages visit https://docs.pyup.io/docs/safety-range-specs

+==========================================================================================================================================+

 REPORT

  Safety is using PyUp's free open-source vulnerability database. This data is 30 days old and limited.
  For real-time enhanced vulnerability data, fix recommendations, severity reporting, cybersecurity support, team and project policy management and more sign up at
https://pyup.io or email sales@pyup.io

  Safety v3.2.0 is scanning for Vulnerabilities...
  Scanning dependencies in your files:

  -> main.py

  Using open-source vulnerability database
  Found and scanned 4 packages
  Timestamp 2024-05-30 15:27:40
  0 vulnerabilities reported
  0 vulnerabilities ignored
+==========================================================================================================================================+

 No known security vulnerabilities reported.
```

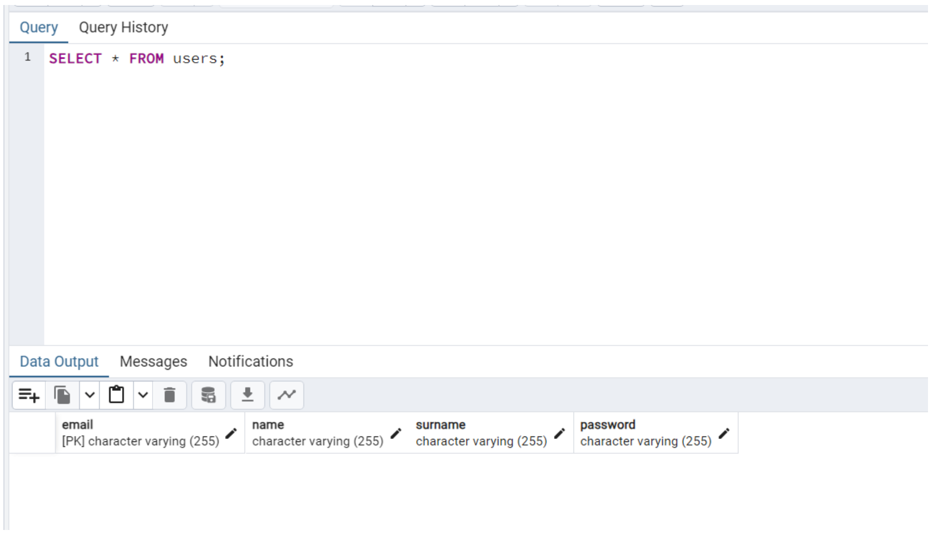
And also in the send_email.py



```
  No packages found
  Timestamp 2024-05-30 15:28:52
  0 vulnerabilities reported
  0 vulnerabilities ignored
+================================================================

 No known security vulnerabilities reported.

+================================================================
```

Code in Action

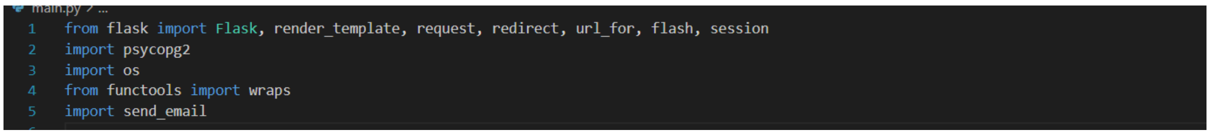
We emptied the database to start with. Just created and that's it



```sql
SELECT * FROM users;
```

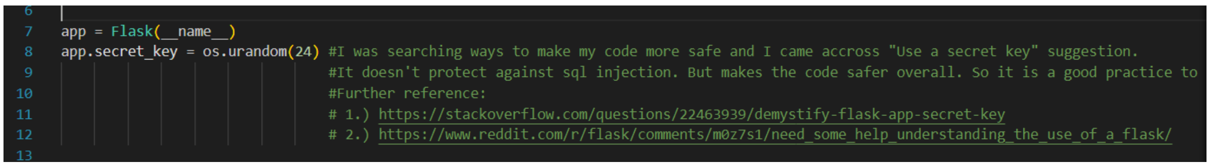
```
email                            name                      surname                   password
[PK] character varying (255)     character varying (255)   character varying (255)   character varying (255)
```

At first, we imported the necessary libraries



```python
from flask import Flask, render_template, request, redirect, url_for, flash, session
import psycopg2
import os
from functools import wraps
import send_email
```

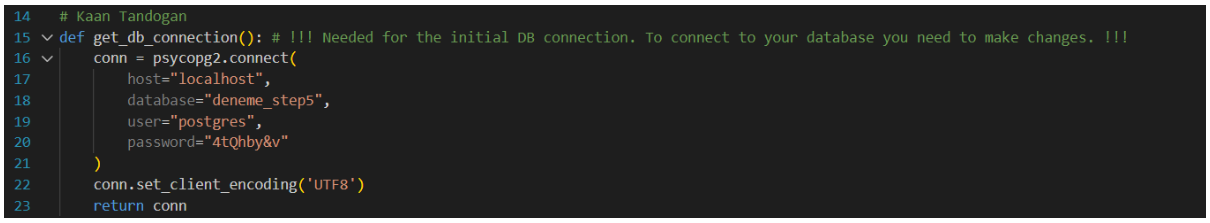
Afterwards, to make our code safer we included random. And c



```python

app = Flask(__name__)
app.secret_key = os.urandom(24) #I was searching ways to make my code more safe and I came accross "Use a secret key" suggestion.
                                #It doesn't protect against sql injection. But makes the code safer overall. So it is a good practice to
                                #Further reference:
                                # 1.) https://stackoverflow.com/questions/22463939/demystify-flask-app-secret-key
                                # 2.) https://www.reddit.com/r/flask/comments/m0z7s1/need_some_help_understanding_the_use_of_a_flask/

```

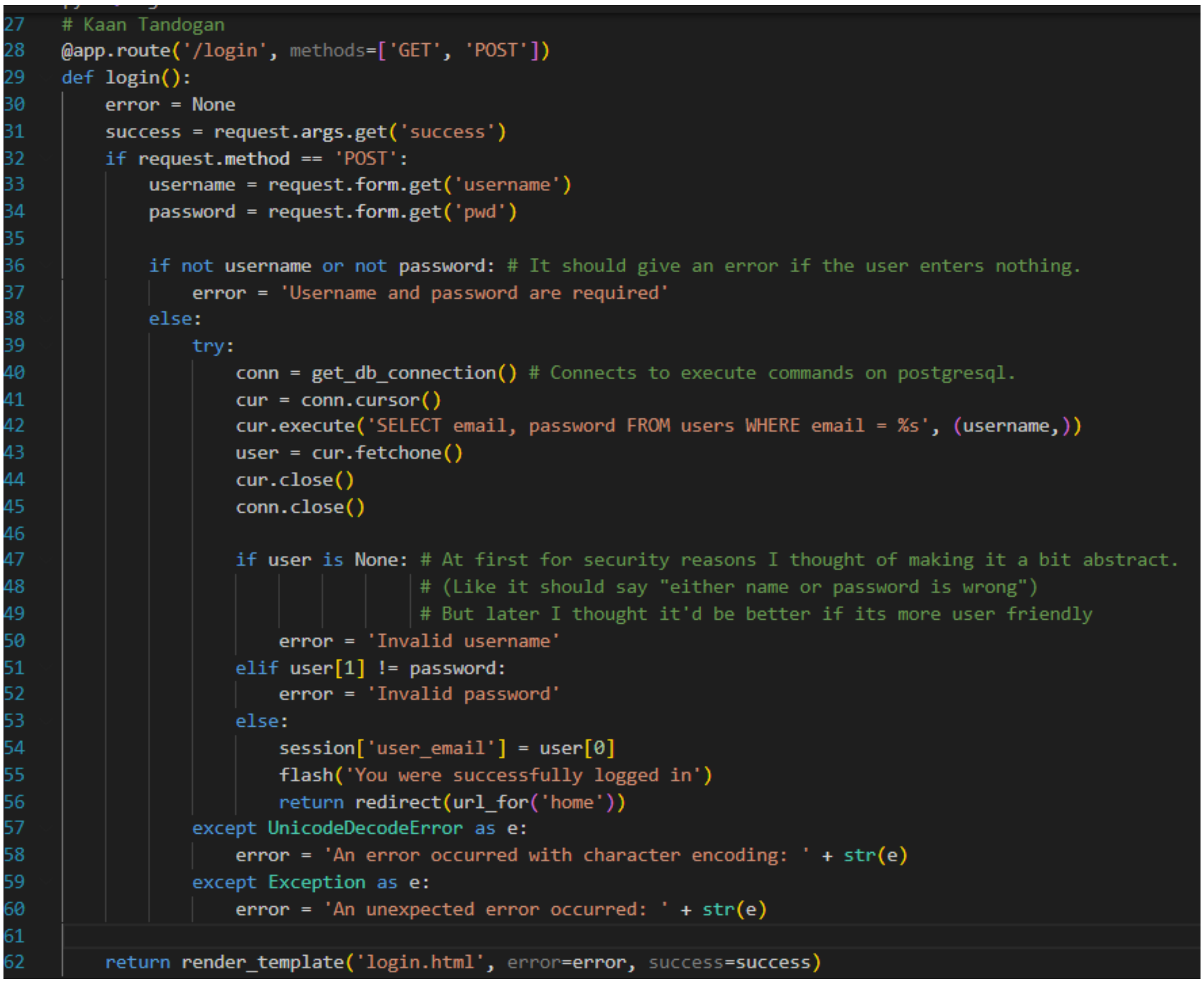
And we can get the database connection here when needed.

/login



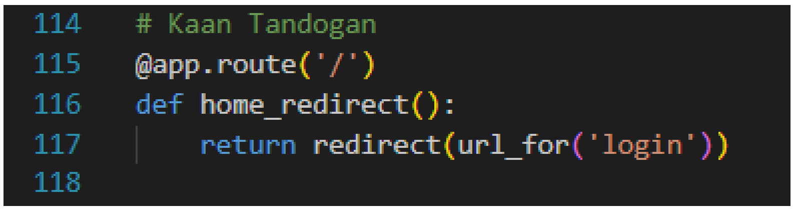
```python
# Kaan Tandogan
def get_db_connection(): # !!! Needed for the initial DB connection. To connect to your database you need to make changes. !!!
    conn = psycopg2.connect(
        host="localhost",
        database="deneme_step5",
        user="postgres",
        password="4tQhby&v"
    )
    conn.set_client_encoding('UTF8')
    return conn
```

Let’s start with **/login** part of our code



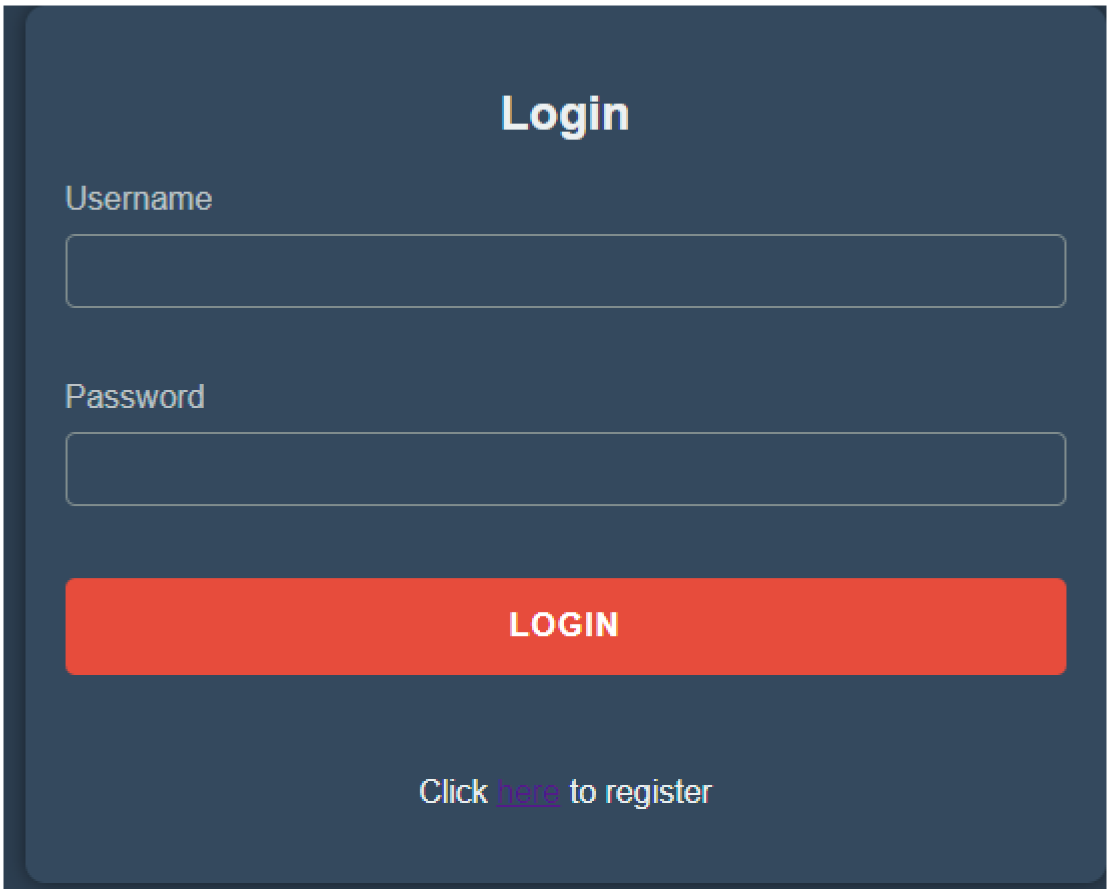
```python
# Kaan Tandogan
@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    success = request.args.get('success')
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('pwd')

        if not username or not password: # It should give an error if the user enters nothing.
            error = 'Username and password are required'
        else:
            try:
                conn = get_db_connection() # Connects to execute commands on postgresql.
                cur = conn.cursor()
                cur.execute('SELECT email, password FROM users WHERE email = %s', (username,))
                user = cur.fetchone()
                cur.close()
                conn.close()

                if user is None: # At first for security reasons I thought of making it a bit abstract.
                                 # (Like it should say "either name or password is wrong")
                                 # But later I thought it'd be better if its more user friendly
                    error = 'Invalid username'
                elif user[1] != password:
                    error = 'Invalid password'
                else:
                    session['user_email'] = user[0]
                    flash('You were successfully logged in')
                    return redirect(url_for('home'))
            except UnicodeDecodeError as e:
                error = 'An error occurred with character encoding: ' + str(e)
            except Exception as e:
                error = 'An unexpected error occurred: ' + str(e)

    return render_template('login.html', error=error, success=success)
```

Please note that / also directs us to the login page.



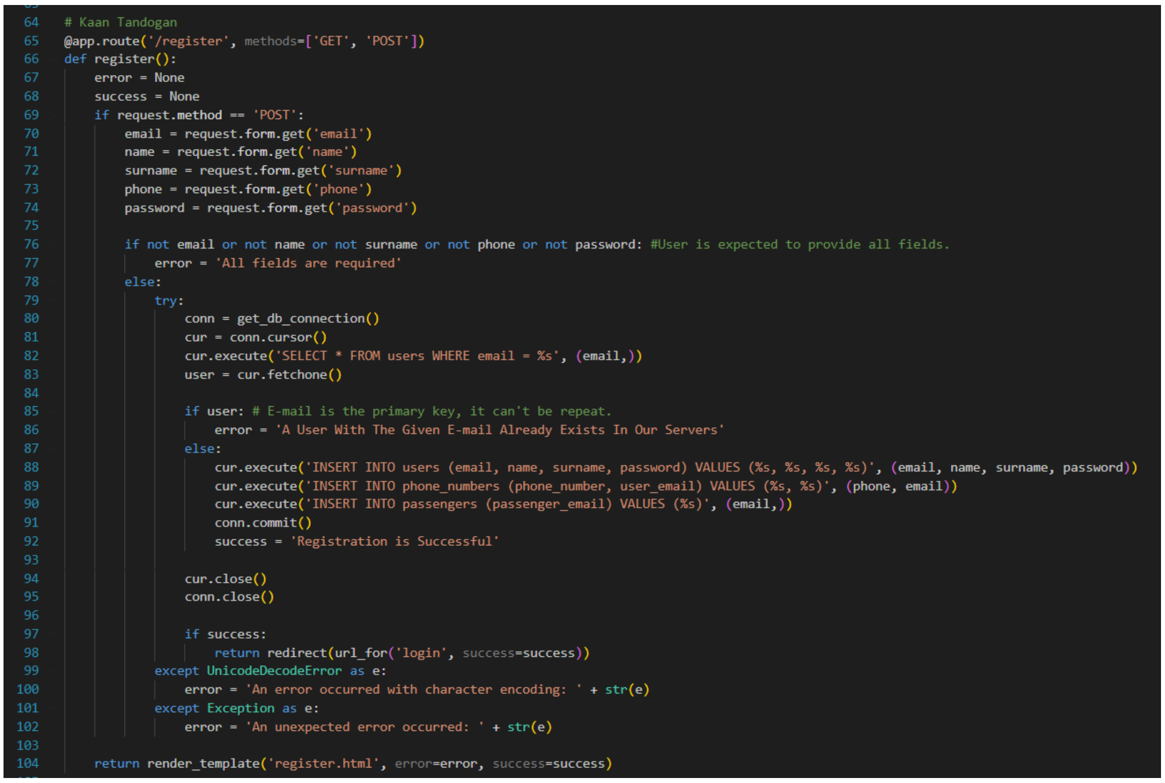
```python
# Kaan Tandogan
@app.route('/')
def home_redirect():
    return redirect(url_for('login'))

```

Login Page



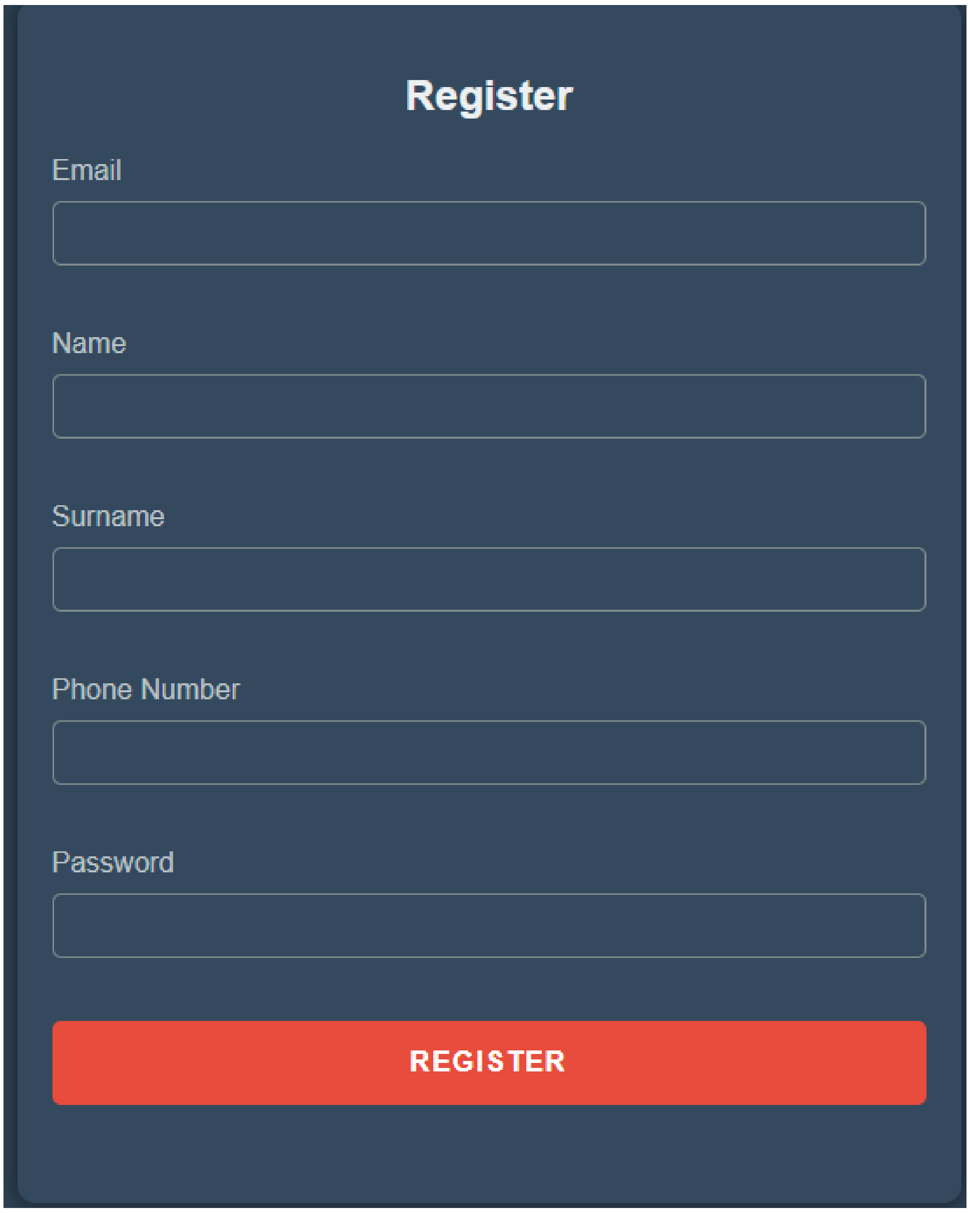
/register

There is the option “Register". If the user clicks “here” in the login page s/he will be directed to **/register**. Please notice that all of the users are initially considered passenger.

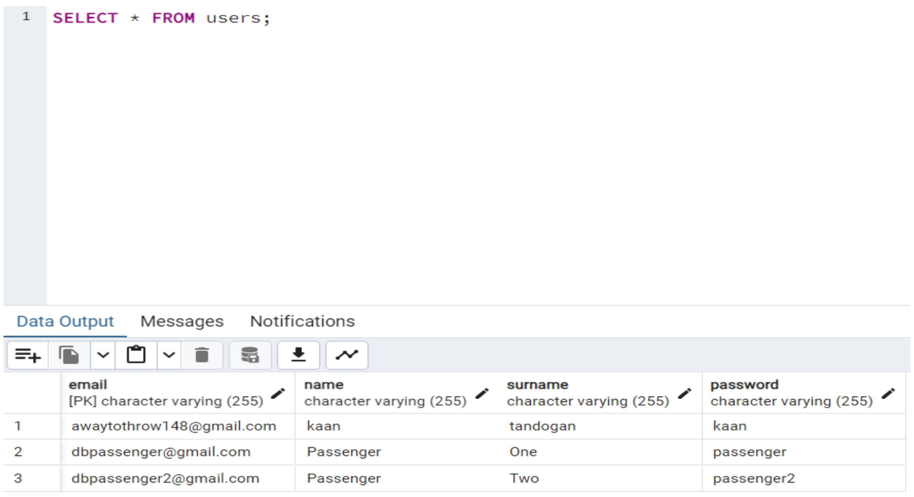


```python
# Kaan Tandogan
@app.route('/register', methods=['GET', 'POST'])
def register():
    error = None
    success = None
    if request.method == 'POST':
        email = request.form.get('email')
        name = request.form.get('name')
        surname = request.form.get('surname')
        phone = request.form.get('phone')
        password = request.form.get('password')

        if not email or not name or not surname or not phone or not password: #User is expected to provide all fields.
            error = 'All fields are required'
        else:
            try:
                conn = get_db_connection()
                cur = conn.cursor()
                cur.execute('SELECT * FROM users WHERE email = %s', (email,))
                user = cur.fetchone()

                if user: # E-mail is the primary key, it can't be repeat.
                    error = 'A User With The Given E-mail Already Exists In Our Servers'
                else:
                    cur.execute('INSERT INTO users (email, name, surname, password) VALUES (%s, %s, %s, %s)', (email, name, surname, password))
                    cur.execute('INSERT INTO phone_numbers (phone_number, user_email) VALUES (%s, %s)', (phone, email))
                    cur.execute('INSERT INTO passengers (passenger_email) VALUES (%s)', (email,))
                    conn.commit()
                    success = 'Registration is Successful'

                cur.close()
                conn.close()

                if success:
                    return redirect(url_for('login', success=success))
            except UnicodeDecodeError as e:
                error = 'An error occurred with character encoding: ' + str(e)
            except Exception as e:
                error = 'An unexpected error occurred: ' + str(e)

    return render_template('register.html', error=error, success=success)
```

Register Page



As told before, our database was initially empty. But to test the functionality we created three users. One of them will be a driver and two will be a passenger. I have successfully registered three users.

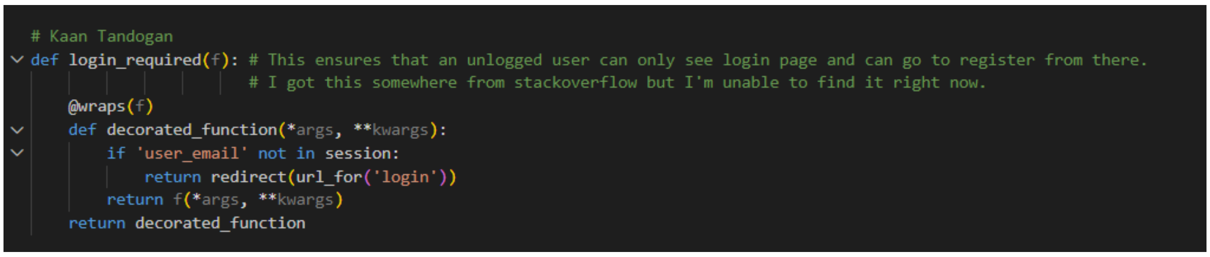


```sql
SELECT * FROM users;
```

	email [PK] character varying (255)	name character varying (255)	surname character varying (255)	password character varying (255)
1	awaytothrow148@gmail.com	kaan	tandogan	kaan
2	dbpassenger@gmail.com	Passenger	One	passenger
3	dbpassenger2@gmail.com	Passenger	Two	passenger2

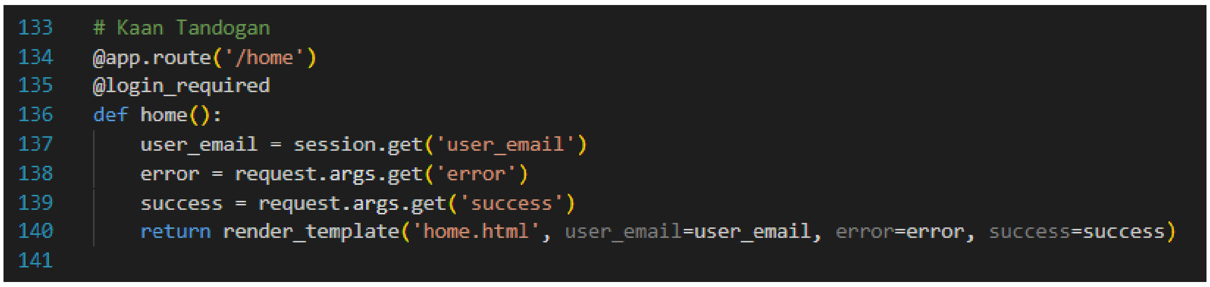
You can see that I added 3 users. All of them are initially passengers.

For a user to go to home she or he needs to be logged in to our system. We achieve this via the following code:



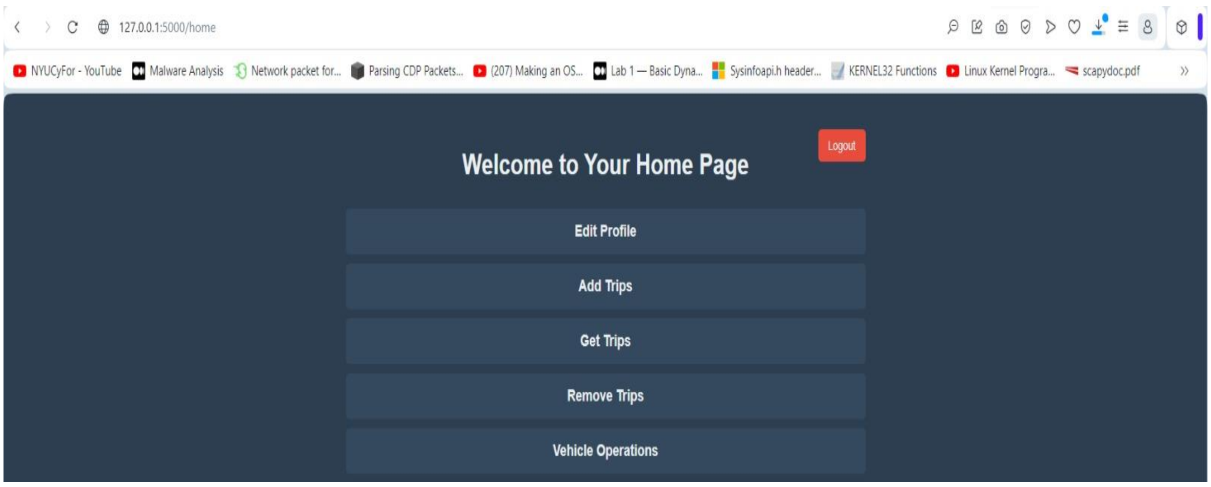
```python
# Kaan Tandogan
def login_required(f): # This ensures that an unlogged user can only see login page and can go to register from there.
                       # I got this somewhere from stackoverflow but I'm unable to find it right now.
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user_email' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

After logging in we are directed to the /home page



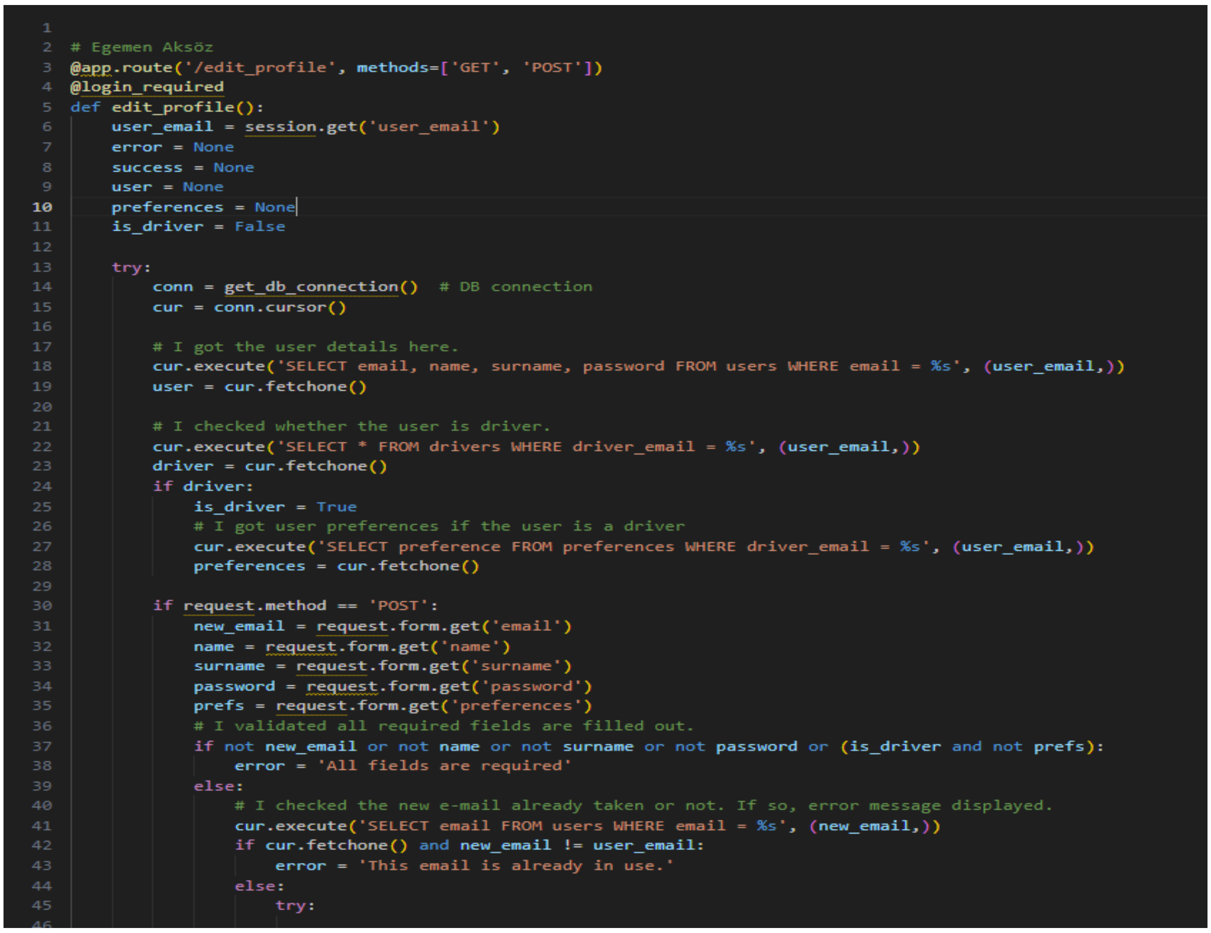
```python
# Kaan Tandogan
@app.route('/home')
@login_required
def home():
    user_email = session.get('user_email')
    error = request.args.get('error')
    success = request.args.get('success')
    return render_template('home.html', user_email=user_email, error=error, success=success)
```

Home Page

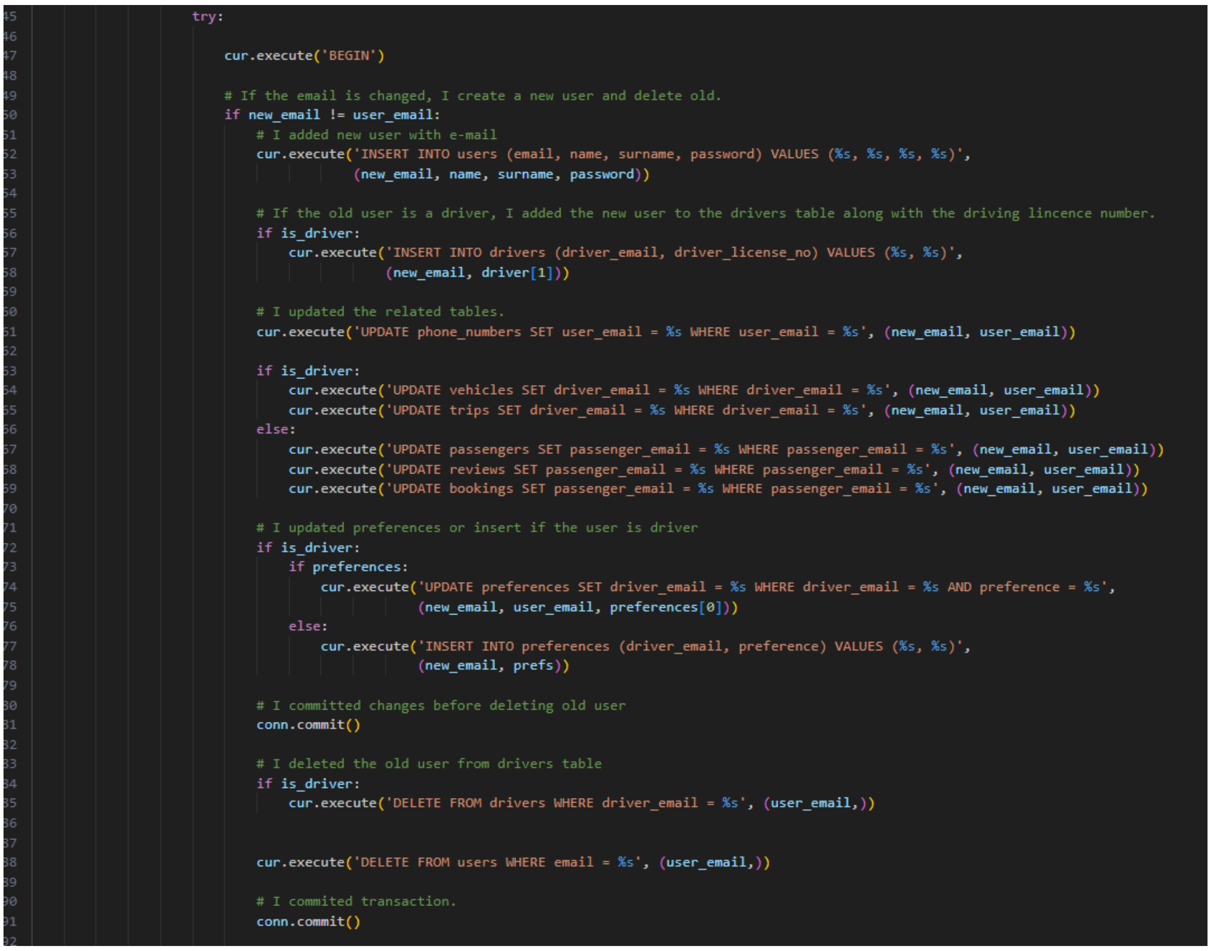

127.0.0.1:5000/home
NYUCyFor - YouTube
Malware Analysis
Network packet for...
Parsing CDP Packets...
(207) Making an OS...
Lab 1 — Basic Dyna...
Sysinfoapi.h header...
KERNEL32 Functions
Linux Kernel Progra...
scapydoc.pdf
Logout
Welcome to Your Home Page
Edit Profile
Add Trips
Get Trips
Remove Trips
Vehicle Operations

/edit_profile:

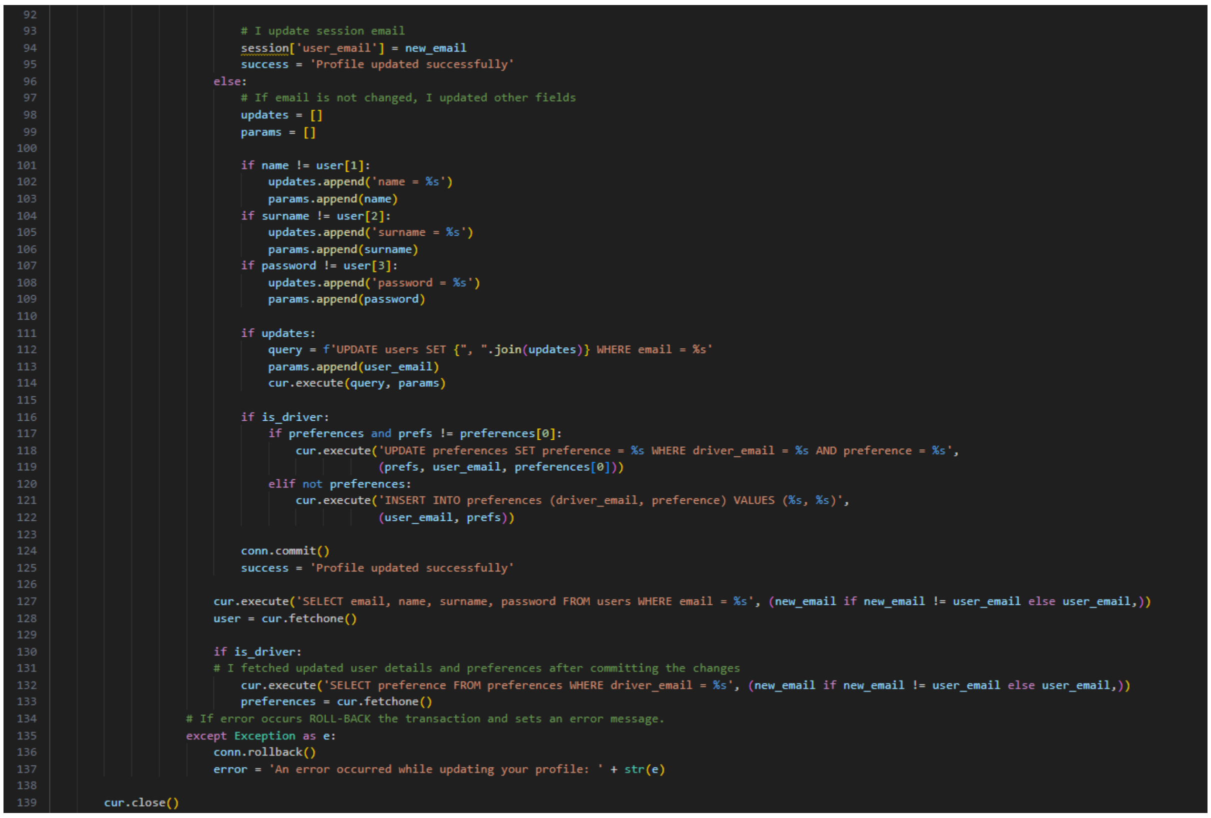
In this code, we create an endpoint to edit user profiles in our application. The endpoint retrieves user details, checks if the user is a driver, and updates the database with new information. It handles form submissions, updates related tables, ensures data consistency through transactions, and provides feedback to the user.



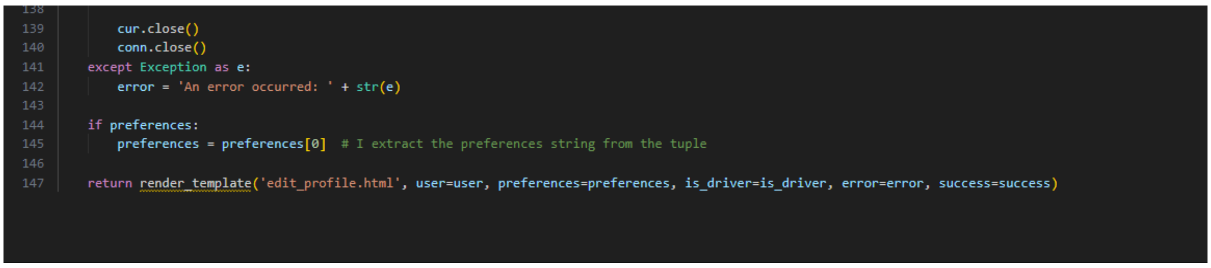
```python

# Egemen Aksöz
@app.route('/edit_profile', methods=['GET', 'POST'])
@login_required
def edit_profile():
    user_email = session.get('user_email')
    error = None
    success = None
    user = None
    preferences = None
    is_driver = False

    try:
        conn = get_db_connection()  # DB connection
        cur = conn.cursor()

        # I got the user details here.
        cur.execute('SELECT email, name, surname, password FROM users WHERE email = %s', (user_email,))
        user = cur.fetchone()

        # I checked whether the user is driver.
        cur.execute('SELECT * FROM drivers WHERE driver_email = %s', (user_email,))
        driver = cur.fetchone()
        if driver:
            is_driver = True
            # I got user preferences if the user is a driver
            cur.execute('SELECT preference FROM preferences WHERE driver_email = %s', (user_email,))
            preferences = cur.fetchone()

        if request.method == 'POST':
            new_email = request.form.get('email')
            name = request.form.get('name')
            surname = request.form.get('surname')
            password = request.form.get('password')
            prefs = request.form.get('preferences')
            # I validated all required fields are filled out.
            if not new_email or not name or not surname or not password or (is_driver and not prefs):
                error = 'All fields are required'
            else:
                # I checked the new e-mail already taken or not. If so, error message displayed.
                cur.execute('SELECT email FROM users WHERE email = %s', (new_email,))
                if cur.fetchone() and new_email != user_email:
                    error = 'This email is already in use.'
                else:
                    try:
```



```python
try:

    cur.execute('BEGIN')

    # If the email is changed, I create a new user and delete old.
    if new_email != user_email:
        # I added new user with e-mail
        cur.execute('INSERT INTO users (email, name, surname, password) VALUES (%s, %s, %s, %s)',
                    (new_email, name, surname, password))

        # If the old user is a driver, I added the new user to the drivers table along with the driving lincence number.
        if is_driver:
            cur.execute('INSERT INTO drivers (driver_email, driver_license_no) VALUES (%s, %s)',
                        (new_email, driver[1]))

        # I updated the related tables.
        cur.execute('UPDATE phone_numbers SET user_email = %s WHERE user_email = %s', (new_email, user_email))

        if is_driver:
            cur.execute('UPDATE vehicles SET driver_email = %s WHERE driver_email = %s', (new_email, user_email))
            cur.execute('UPDATE trips SET driver_email = %s WHERE driver_email = %s', (new_email, user_email))
        else:
            cur.execute('UPDATE passengers SET passenger_email = %s WHERE passenger_email = %s', (new_email, user_email))
            cur.execute('UPDATE reviews SET passenger_email = %s WHERE passenger_email = %s', (new_email, user_email))
            cur.execute('UPDATE bookings SET passenger_email = %s WHERE passenger_email = %s', (new_email, user_email))

        # I updated preferences or insert if the user is driver
        if is_driver:
            if preferences:
                cur.execute('UPDATE preferences SET driver_email = %s WHERE driver_email = %s AND preference = %s',
                            (new_email, user_email, preferences[0]))
            else:
                cur.execute('INSERT INTO preferences (driver_email, preference) VALUES (%s, %s)',
                            (new_email, prefs))

        # I committed changes before deleting old user
        conn.commit()

        # I deleted the old user from drivers table
        if is_driver:
            cur.execute('DELETE FROM drivers WHERE driver_email = %s', (user_email,))


        cur.execute('DELETE FROM users WHERE email = %s', (user_email,))

        # I commited transaction.
        conn.commit()
```



```
                            # I update session email
                            session['user_email'] = new_email
                            success = 'Profile updated successfully'
                        else:
                            # If email is not changed, I updated other fields
                            updates = []
                            params = []

                            if name != user[1]:
                                updates.append('name = %s')
                                params.append(name)
                            if surname != user[2]:
                                updates.append('surname = %s')
                                params.append(surname)
                            if password != user[3]:
                                updates.append('password = %s')
                                params.append(password)

                            if updates:
                                query = f'UPDATE users SET {", ".join(updates)} WHERE email = %s'
                                params.append(user_email)
                                cur.execute(query, params)

                            if is_driver:
                                if preferences and prefs != preferences[0]:
                                    cur.execute('UPDATE preferences SET preference = %s WHERE driver_email = %s AND preference = %s',
                                                (prefs, user_email, preferences[0]))
                                elif not preferences:
                                    cur.execute('INSERT INTO preferences (driver_email, preference) VALUES (%s, %s)',
                                                (user_email, prefs))

                            conn.commit()
                            success = 'Profile updated successfully'

                        cur.execute('SELECT email, name, surname, password FROM users WHERE email = %s', (new_email if new_email != user_email else user_email,))
                        user = cur.fetchone()

                        if is_driver:
                        # I fetched updated user details and preferences after committing the changes
                            cur.execute('SELECT preference FROM preferences WHERE driver_email = %s', (new_email if new_email != user_email else user_email,))
                            preferences = cur.fetchone()
                    # If error occurs ROLL-BACK the transaction and sets an error message.
                    except Exception as e:
                        conn.rollback()
                        error = 'An error occurred while updating your profile: ' + str(e)

        cur.close()
```



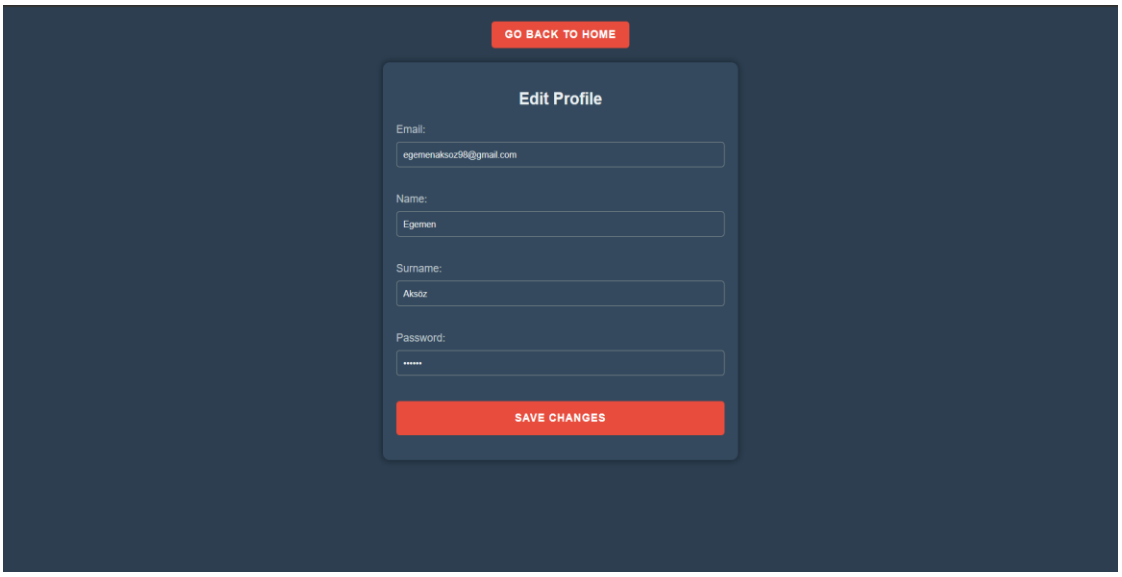
```
        cur.close()
        conn.close()
    except Exception as e:
        error = 'An error occurred: ' + str(e)

    if preferences:
        preferences = preferences[0]  # I extract the preferences string from the tuple

    return render_template('edit_profile.html', user=user, preferences=preferences, is_driver=is_driver, error=error, success=success)
```

Edit Profile Page

This page displays a user interface for editing a user's profile in a web application. It contains the following elements:

Important: If the user is not driver, preferences is not shown in the edit-profile page.

As shown in below; user did not add any vehicle yet, so web-site act as present to normal user.



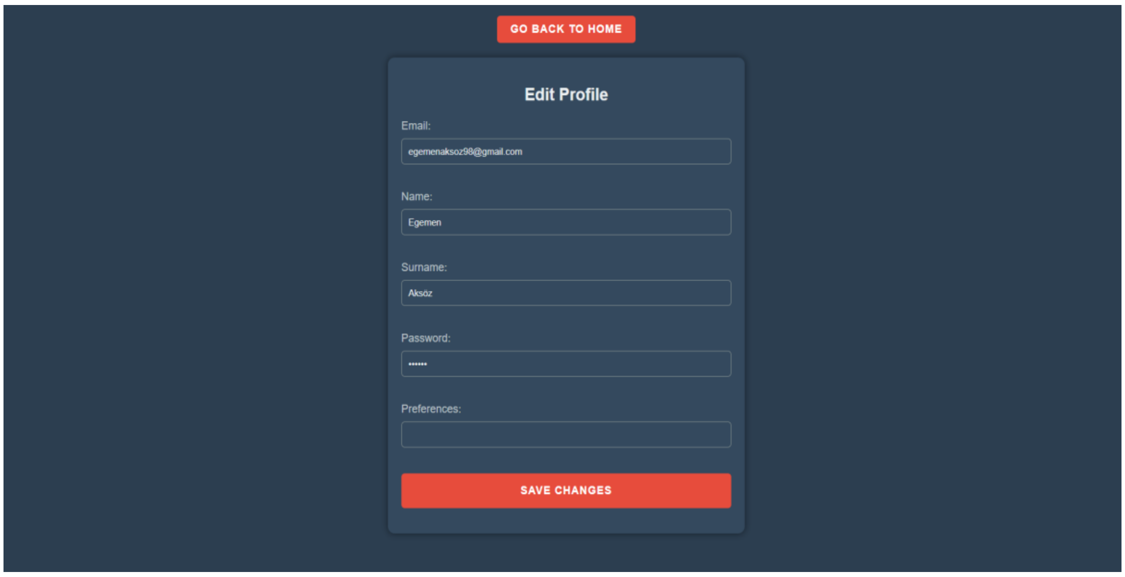
Email: egemenaksoz98@gmail.com

Name: Egemen

Surname: Aksöz

Password: ••••••

If user add a vehicle to our web application, user automatically classed as driver. After this time, they can add their preferences to their profile.



Email: egemenaksoz98@gmail.com

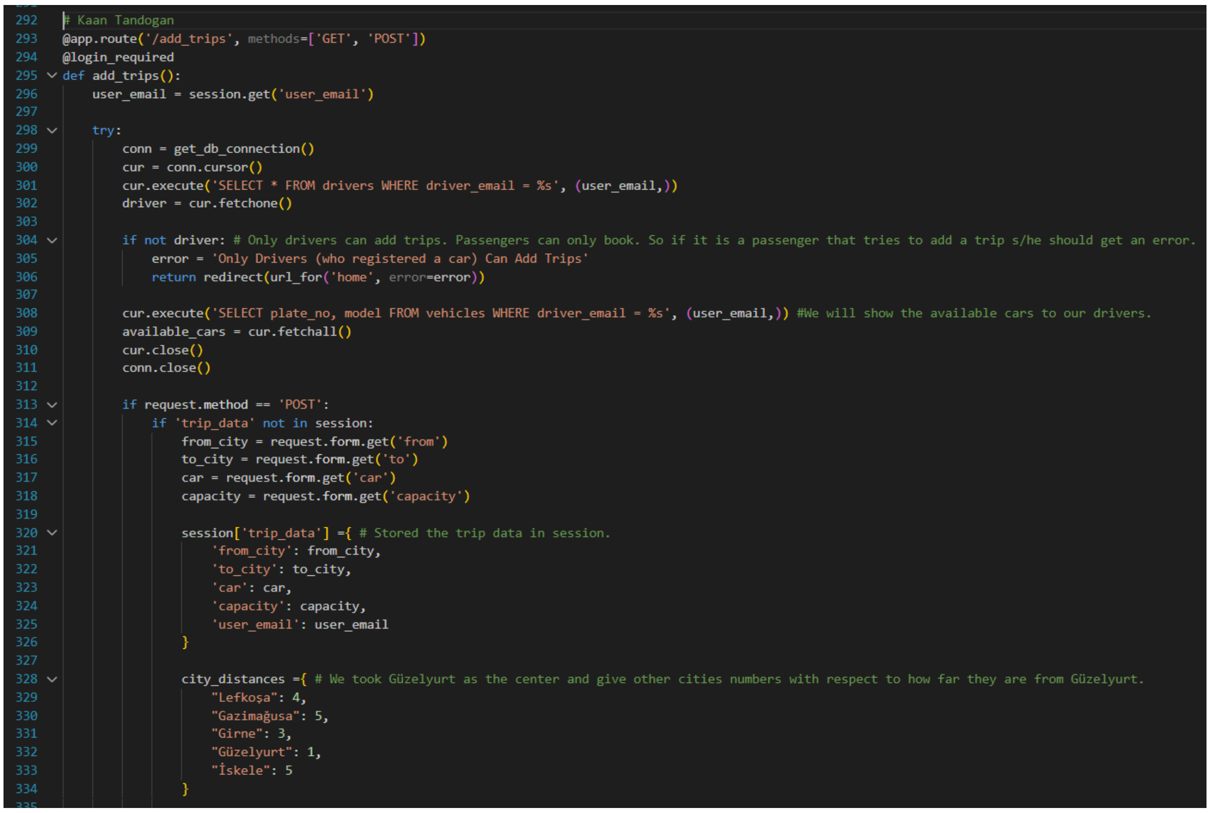
Name: Egemen

Surname: Aksöz

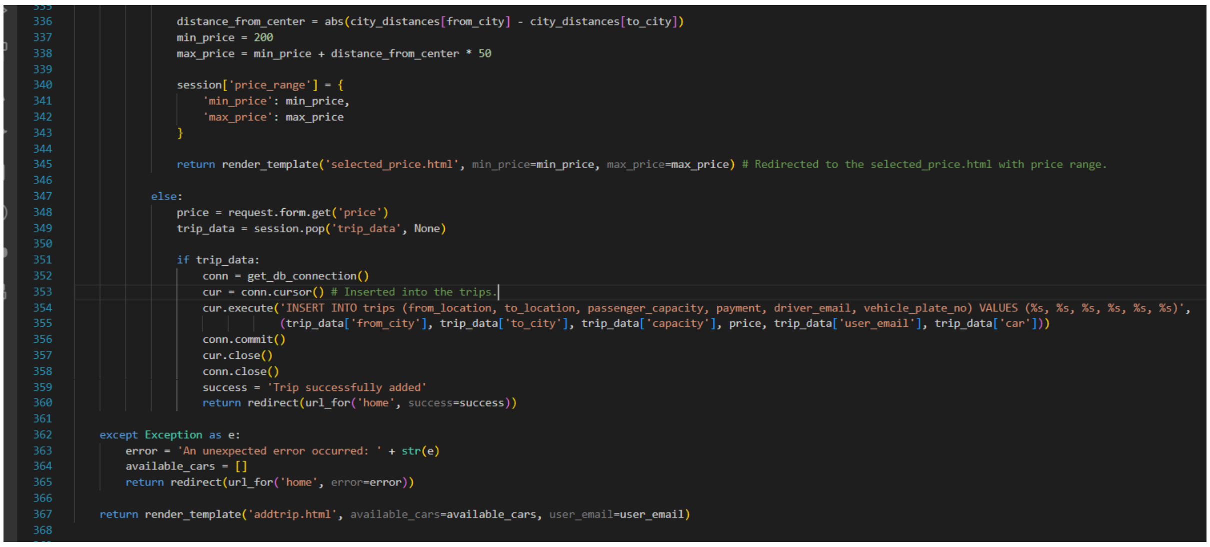
Password: ••••••

Preferences:

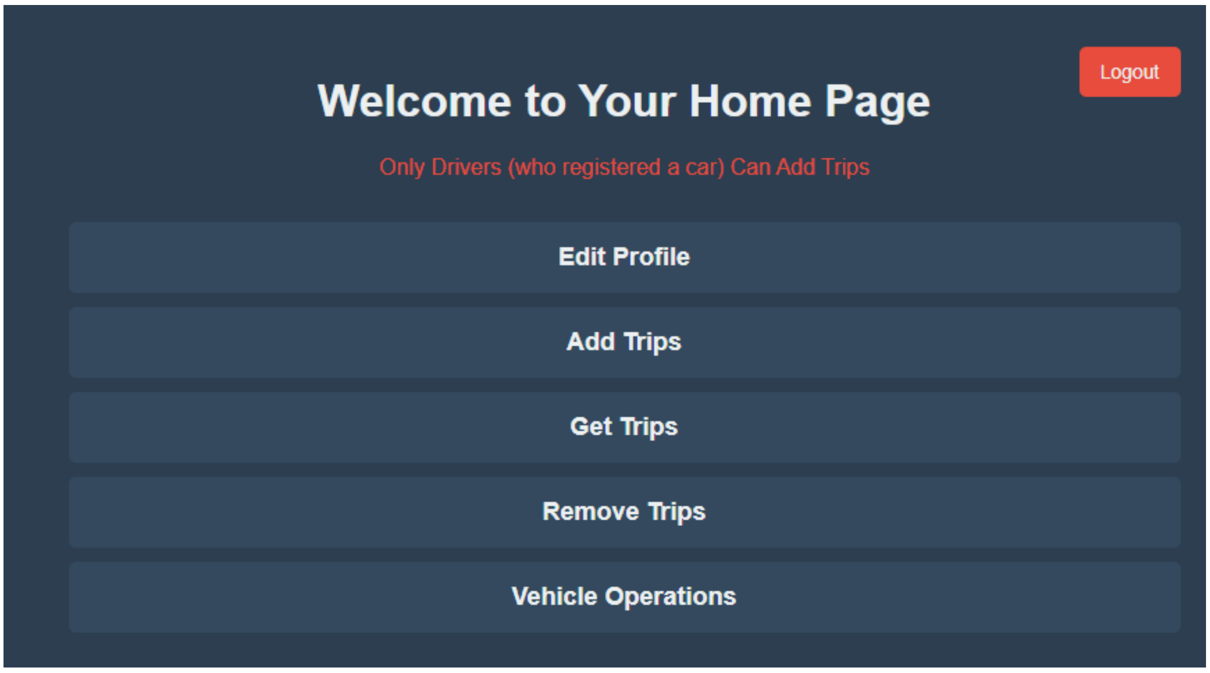
If a user who is not a driver, AKA passenger, tries to add a trip s/he gets an error



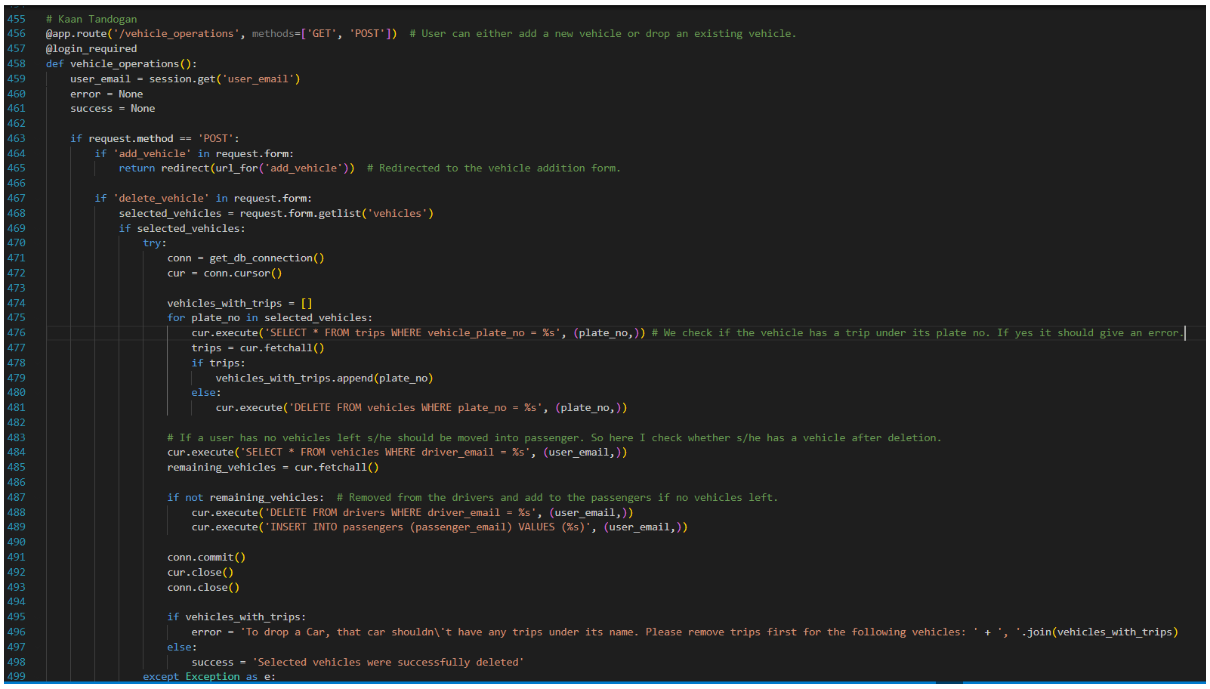
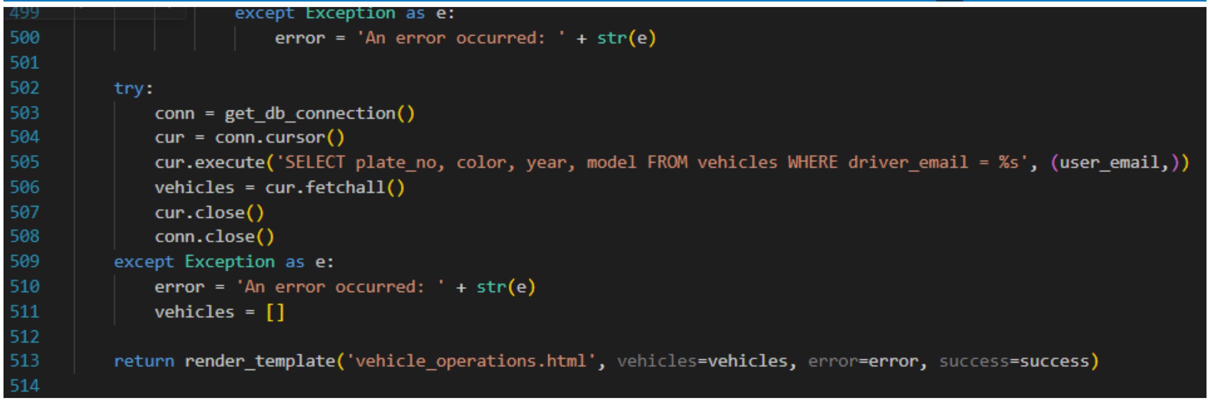
```python
# Kaan Tandogan
@app.route('/add_trips', methods=['GET', 'POST'])
@login_required
def add_trips():
    user_email = session.get('user_email')

    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT * FROM drivers WHERE driver_email = %s', (user_email,))
        driver = cur.fetchone()

        if not driver: # Only drivers can add trips. Passengers can only book. So if it is a passenger that tries to add a trip s/he should get an error.
            error = 'Only Drivers (who registered a car) Can Add Trips'
            return redirect(url_for('home', error=error))

        cur.execute('SELECT plate_no, model FROM vehicles WHERE driver_email = %s', (user_email,)) #We will show the available cars to our drivers.
        available_cars = cur.fetchall()
        cur.close()
        conn.close()

        if request.method == 'POST':
            if 'trip_data' not in session:
                from_city = request.form.get('from')
                to_city = request.form.get('to')
                car = request.form.get('car')
                capacity = request.form.get('capacity')

                session['trip_data'] ={ # Stored the trip data in session.
                    'from_city': from_city,
                    'to_city': to_city,
                    'car': car,
                    'capacity': capacity,
                    'user_email': user_email
                }

                city_distances ={ # We took Güzelyurt as the center and give other cities numbers with respect to how far they are from Güzelyurt.
                    "Lefkoşa": 4,
                    "Gazimağusa": 5,
                    "Girne": 3,
                    "Güzelyurt": 1,
                    "İskele": 5
                }
```



```python
                distance_from_center = abs(city_distances[from_city] - city_distances[to_city])
                min_price = 200
                max_price = min_price + distance_from_center * 50

                session['price_range'] = {
                    'min_price': min_price,
                    'max_price': max_price
                }

                return render_template('selected_price.html', min_price=min_price, max_price=max_price) # Redirected to the selected_price.html with price range.

            else:
                price = request.form.get('price')
                trip_data = session.pop('trip_data', None)

                if trip_data:
                    conn = get_db_connection()
                    cur = conn.cursor() # Inserted into the trips.
                    cur.execute('INSERT INTO trips (from_location, to_location, passenger_capacity, payment, driver_email, vehicle_plate_no) VALUES (%s, %s, %s, %s, %s, %s)',
                                (trip_data['from_city'], trip_data['to_city'], trip_data['capacity'], price, trip_data['user_email'], trip_data['car']))
                    conn.commit()
                    cur.close()
                    conn.close()
                    success = 'Trip successfully added'
                    return redirect(url_for('home', success=success))

    except Exception as e:
        error = 'An unexpected error occurred: ' + str(e)
        available_cars = []
        return redirect(url_for('home', error=error))

    return render_template('addtrip.html', available_cars=available_cars, user_email=user_email)
```

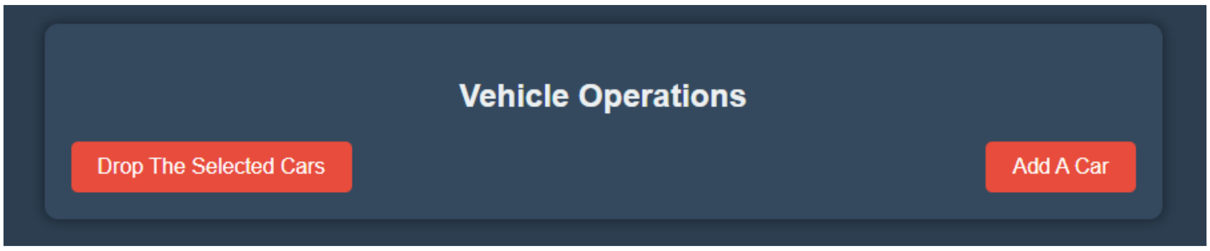

Logout
Welcome to Your Home Page
Only Drivers (who registered a car) Can Add Trips
Edit Profile
Add Trips
Get Trips
Remove Trips
Vehicle Operations

Now let’s try this again after adding a vehicle and therefore making our passenger a driver and then try again. If the “Vehicle Operations” is selected, we go to the **/vehicle_operations.**

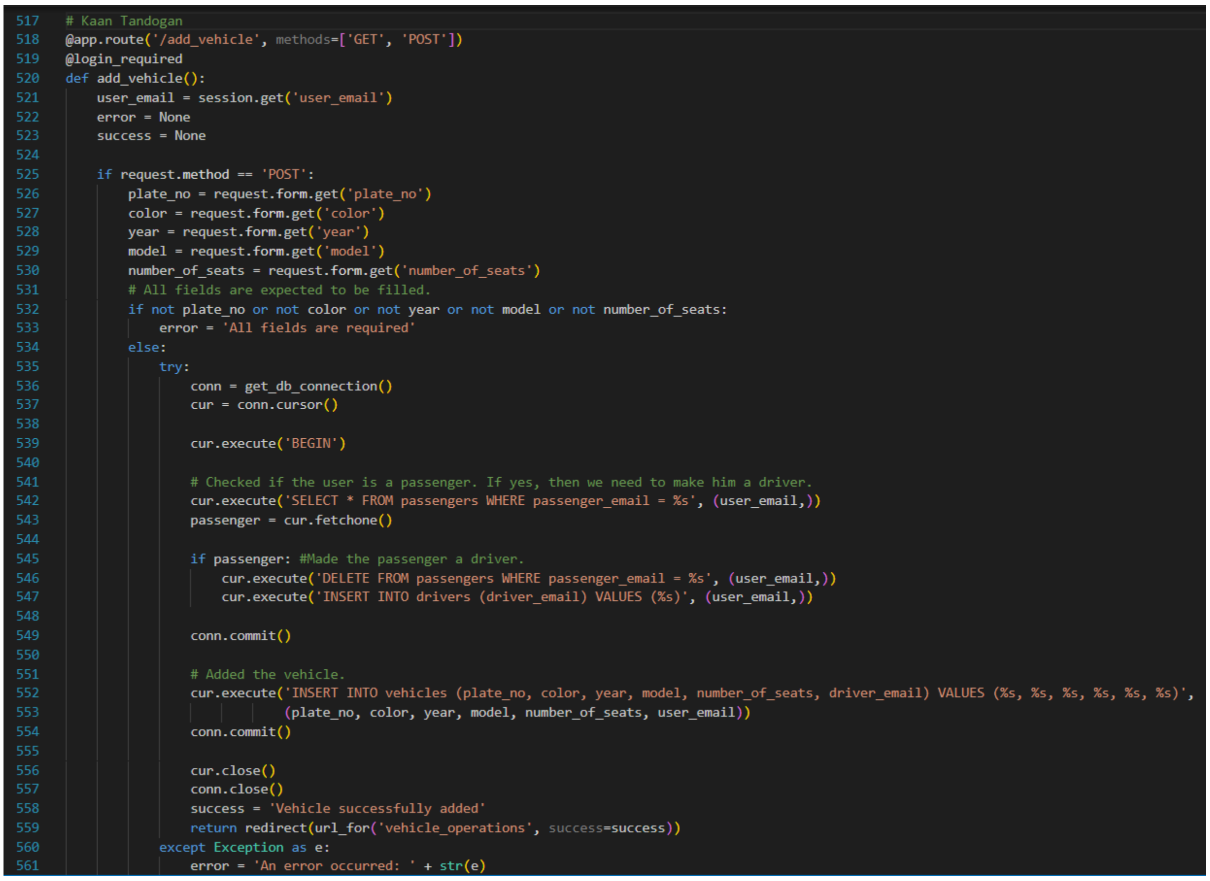



```python
# Kaan Tandogan
@app.route('/vehicle_operations', methods=['GET', 'POST'])  # User can either add a new vehicle or drop an existing vehicle.
@login_required
def vehicle_operations():
    user_email = session.get('user_email')
    error = None
    success = None

    if request.method == 'POST':
        if 'add_vehicle' in request.form:
            return redirect(url_for('add_vehicle'))  # Redirected to the vehicle addition form.

        if 'delete_vehicle' in request.form:
            selected_vehicles = request.form.getlist('vehicles')
            if selected_vehicles:
                try:
                    conn = get_db_connection()
                    cur = conn.cursor()

                    vehicles_with_trips = []
                    for plate_no in selected_vehicles:
                        cur.execute('SELECT * FROM trips WHERE vehicle_plate_no = %s', (plate_no,)) # We check if the vehicle has a trip under its plate no. If yes it should give an error.
                        trips = cur.fetchall()
                        if trips:
                            vehicles_with_trips.append(plate_no)
                        else:
                            cur.execute('DELETE FROM vehicles WHERE plate_no = %s', (plate_no,))

                    # If a user has no vehicles left s/he should be moved into passenger. So here I check whether s/he has a vehicle after deletion.
                    cur.execute('SELECT * FROM vehicles WHERE driver_email = %s', (user_email,))
                    remaining_vehicles = cur.fetchall()

                    if not remaining_vehicles:  # Removed from the drivers and add to the passengers if no vehicles left.
                        cur.execute('DELETE FROM drivers WHERE driver_email = %s', (user_email,))
                        cur.execute('INSERT INTO passengers (passenger_email) VALUES (%s)', (user_email,))

                    conn.commit()
                    cur.close()
                    conn.close()

                    if vehicles_with_trips:
                        error = 'To drop a Car, that car shouldn\'t have any trips under its name. Please remove trips first for the following vehicles: ' + ', '.join(vehicles_with_trips)
                    else:
                        success = 'Selected vehicles were successfully deleted'
                except Exception as e:
                    error = 'An error occurred: ' + str(e)

    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT plate_no, color, year, model FROM vehicles WHERE driver_email = %s', (user_email,))
        vehicles = cur.fetchall()
        cur.close()
        conn.close()
    except Exception as e:
        error = 'An error occurred: ' + str(e)
        vehicles = []

    return render_template('vehicle_operations.html', vehicles=vehicles, error=error, success=success)
```

Since we don’t have any car initially so it doesn’t show any vehicles. But lets add 2 cars.



If I click “Add A Car” option I got to the **/add_vehicle** part of our code. I wrote comments in my code to explain how it operates.



```python
# Kaan Tandogan
@app.route('/add_vehicle', methods=['GET', 'POST'])
@login_required
def add_vehicle():
    user_email = session.get('user_email')
    error = None
    success = None

    if request.method == 'POST':
        plate_no = request.form.get('plate_no')
        color = request.form.get('color')
        year = request.form.get('year')
        model = request.form.get('model')
        number_of_seats = request.form.get('number_of_seats')
        # All fields are expected to be filled.
        if not plate_no or not color or not year or not model or not number_of_seats:
            error = 'All fields are required'
        else:
            try:
                conn = get_db_connection()
                cur = conn.cursor()

                cur.execute('BEGIN')

                # Checked if the user is a passenger. If yes, then we need to make him a driver.
                cur.execute('SELECT * FROM passengers WHERE passenger_email = %s', (user_email,))
                passenger = cur.fetchone()

                if passenger: #Made the passenger a driver.
                    cur.execute('DELETE FROM passengers WHERE passenger_email = %s', (user_email,))
                    cur.execute('INSERT INTO drivers (driver_email) VALUES (%s)', (user_email,))

                conn.commit()

                # Added the vehicle.
                cur.execute('INSERT INTO vehicles (plate_no, color, year, model, number_of_seats, driver_email) VALUES (%s, %s, %s, %s, %s, %s)',
                            (plate_no, color, year, model, number_of_seats, user_email))
                conn.commit()

                cur.close()
                conn.close()
                success = 'Vehicle successfully added'
                return redirect(url_for('vehicle_operations', success=success))
            except Exception as e:
                error = 'An error occurred: ' + str(e)
```

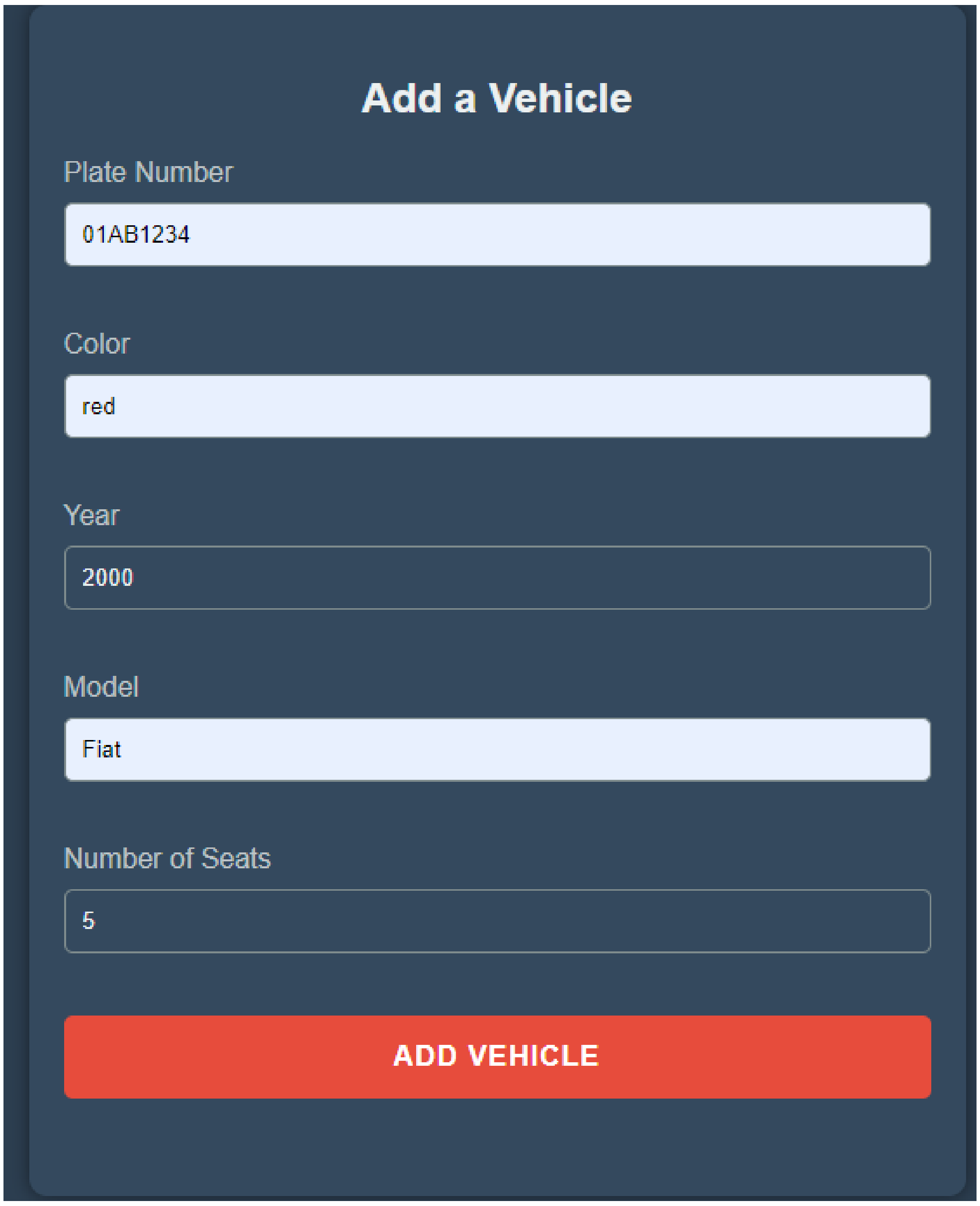

Add a Vehicle

Plate Number: 01AB1234

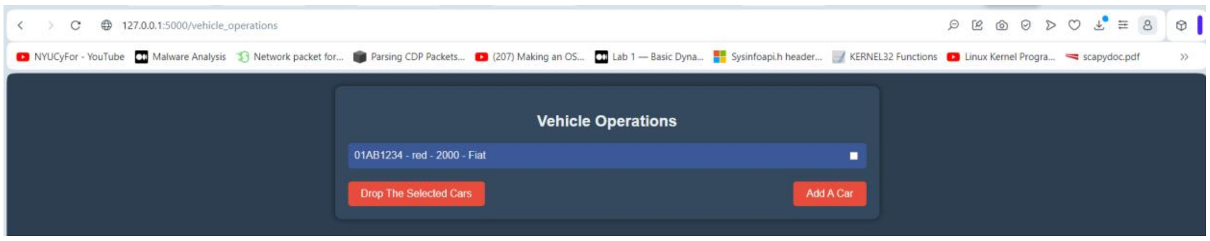
Color: red

Year: 2000

Model: Fiat

Number of Seats: 5

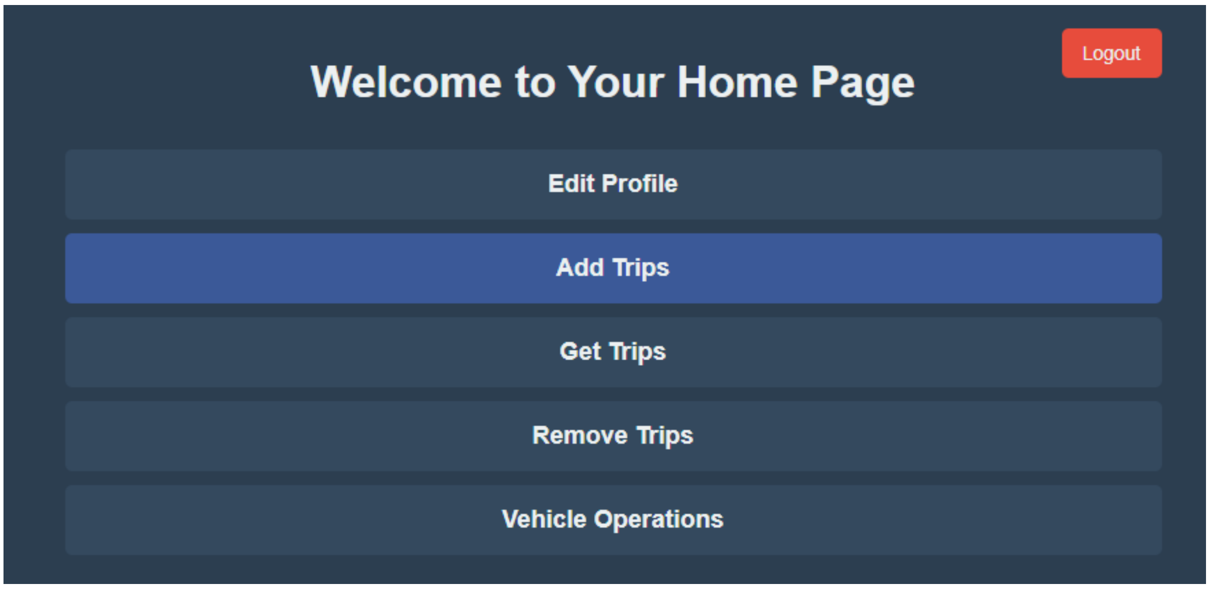
After we click on “Add Vehicle” we can now see the vehicle we added. I’ll add another one and then try to add trip.


127.0.0.1:5000/vehicle_operations

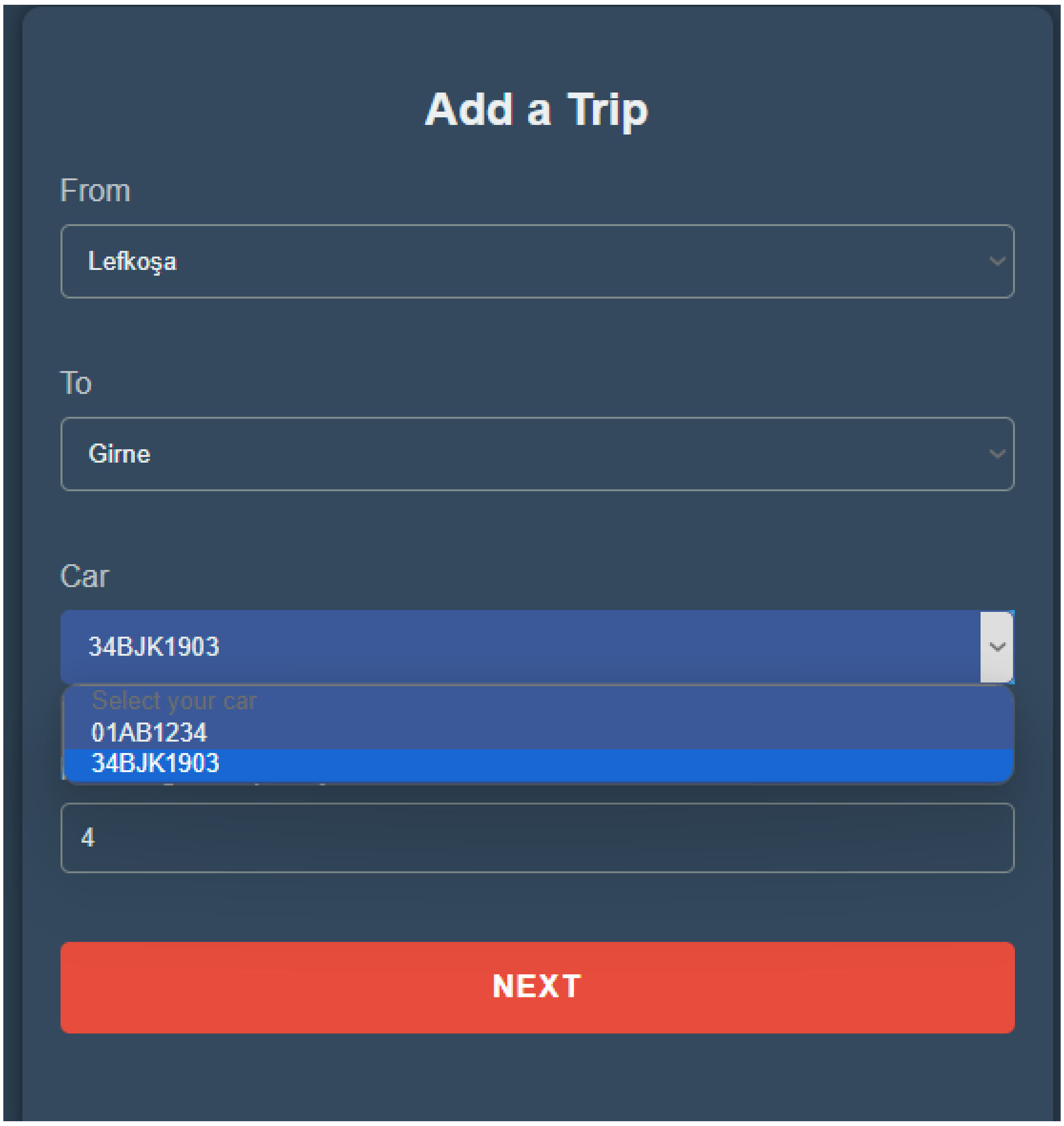
Vehicle Operations

01AB1234 - red - 2000 - Fiat

And now let’s try “Add Trips” option once again.



And now we are able to open the add_trips


Add a Trip

From

Lefkoşa

To

Girne

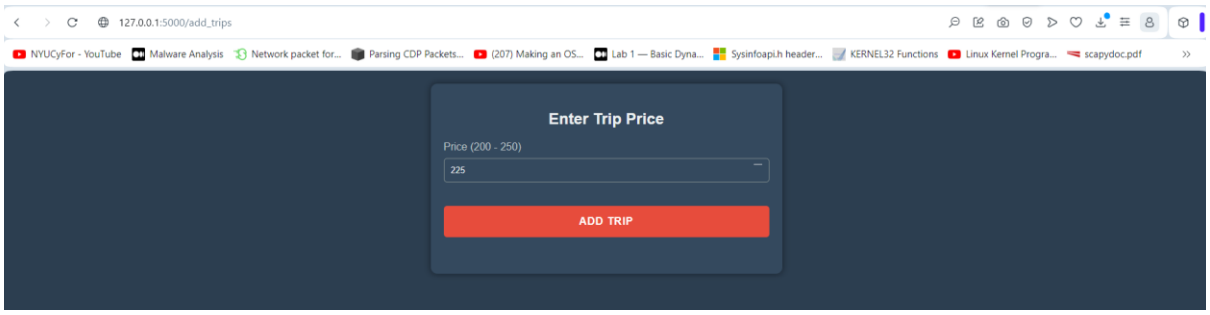
Car

34BJK1903

Select your car
01AB1234
34BJK1903

4

After selecting from, to, car, and passenger capacity we click next. Our code calculates the suggested price and forces the user to enter a valid input.


127.0.0.1:5000/add_trips

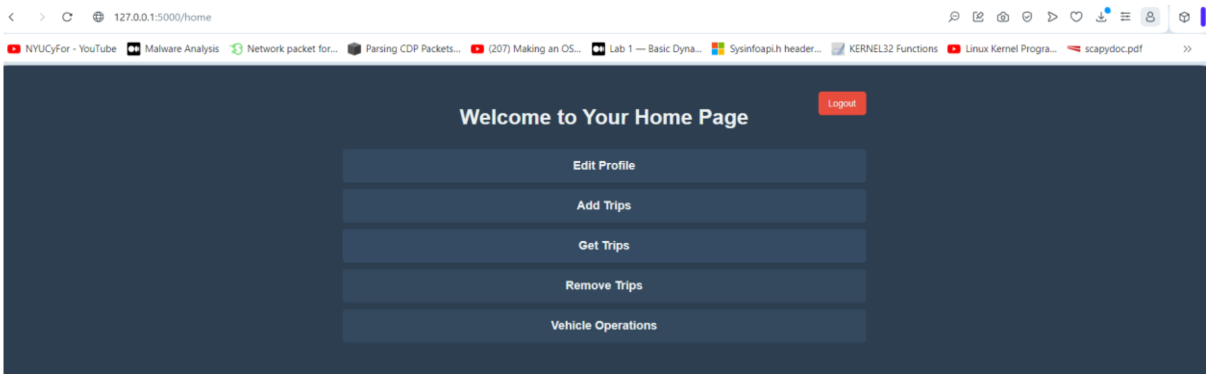
NYUCyFor - YouTube   Malware Analysis   Network packet for...   Parsing CDP Packets...   (207) Making an OS...   Lab 1 — Basic Dyna...   Sysinfoapi.h header...   KERNEL32 Functions   Linux Kernel Progra...   scapydoc.pdf

Enter Trip Price

Price (200 - 250)

225

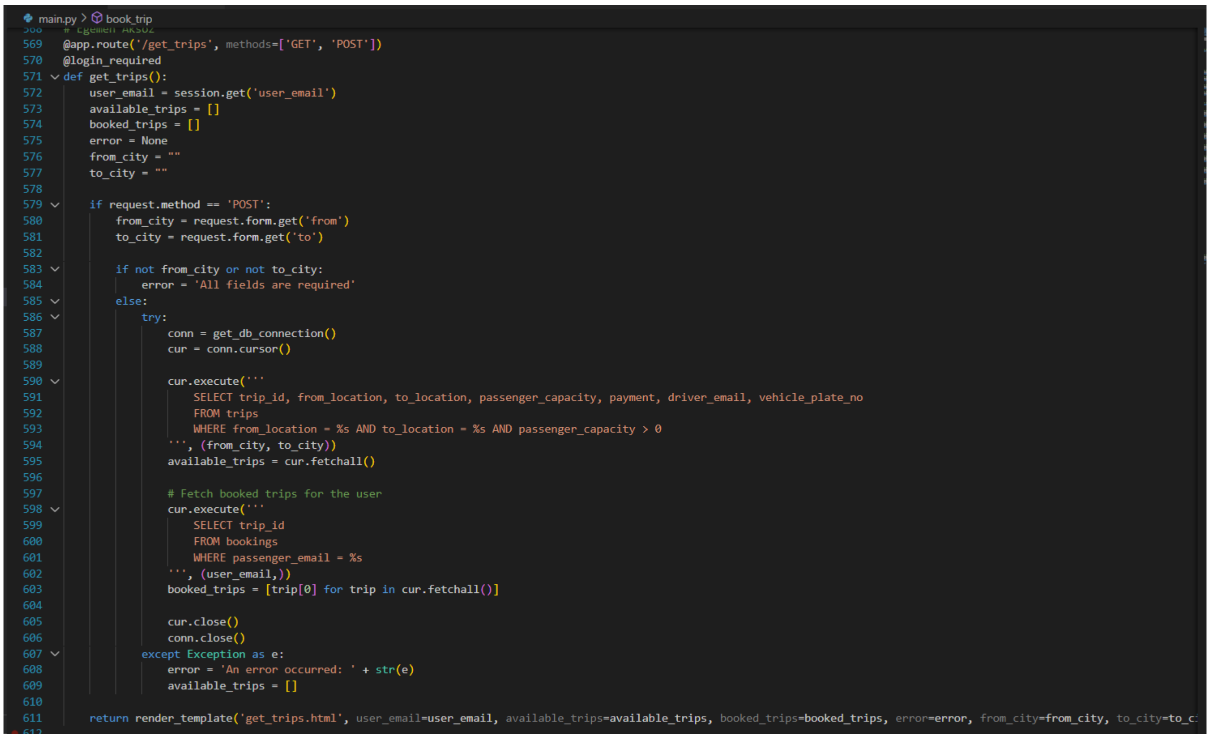
Afterwards, it adds the trip. Now let’s log out from driver and enter as a “Passenger One“



/get_trips

In this code, we create an endpoint to book a trip in our application. It verifies seat availability, updates the bookings and trips tables, retrieves passenger and driver details, commits the transaction, and sends a confirmation email to the driver. It also handles errors and provides user feedback.

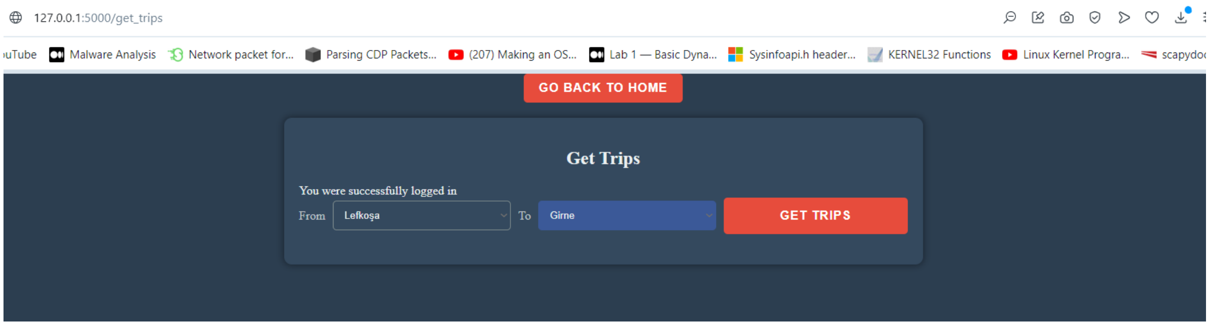
After that get the **/get_trips** part.



```python
@app.route('/get_trips', methods=['GET', 'POST'])
@login_required
def get_trips():
    user_email = session.get('user_email')
    available_trips = []
    booked_trips = []
    error = None
    from_city = ""
    to_city = ""

    if request.method == 'POST':
        from_city = request.form.get('from')
        to_city = request.form.get('to')

        if not from_city or not to_city:
            error = 'All fields are required'
        else:
            try:
                conn = get_db_connection()
                cur = conn.cursor()

                cur.execute('''
                    SELECT trip_id, from_location, to_location, passenger_capacity, payment, driver_email, vehicle_plate_no
                    FROM trips
                    WHERE from_location = %s AND to_location = %s AND passenger_capacity > 0
                ''', (from_city, to_city))
                available_trips = cur.fetchall()

                # Fetch booked trips for the user
                cur.execute('''
                    SELECT trip_id
                    FROM bookings
                    WHERE passenger_email = %s
                ''', (user_email,))
                booked_trips = [trip[0] for trip in cur.fetchall()]

                cur.close()
                conn.close()
            except Exception as e:
                error = 'An error occurred: ' + str(e)
                available_trips = []

    return render_template('get_trips.html', user_email=user_email, available_trips=available_trips, booked_trips=booked_trips, error=error, from_city=from_city, to_city=to_c
```

Here I got the trips as well.

Get Trip Page

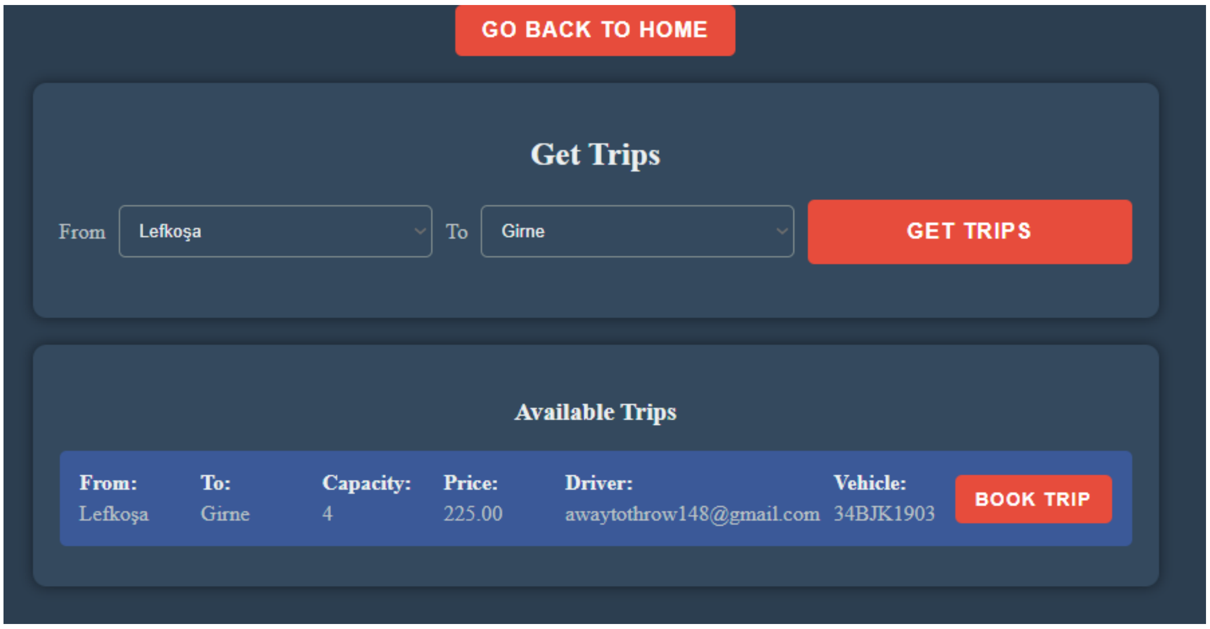


Get Trips

You were successfully logged in

From: Lefkoşa
To: Girne

After entering the from, to information we get the following:



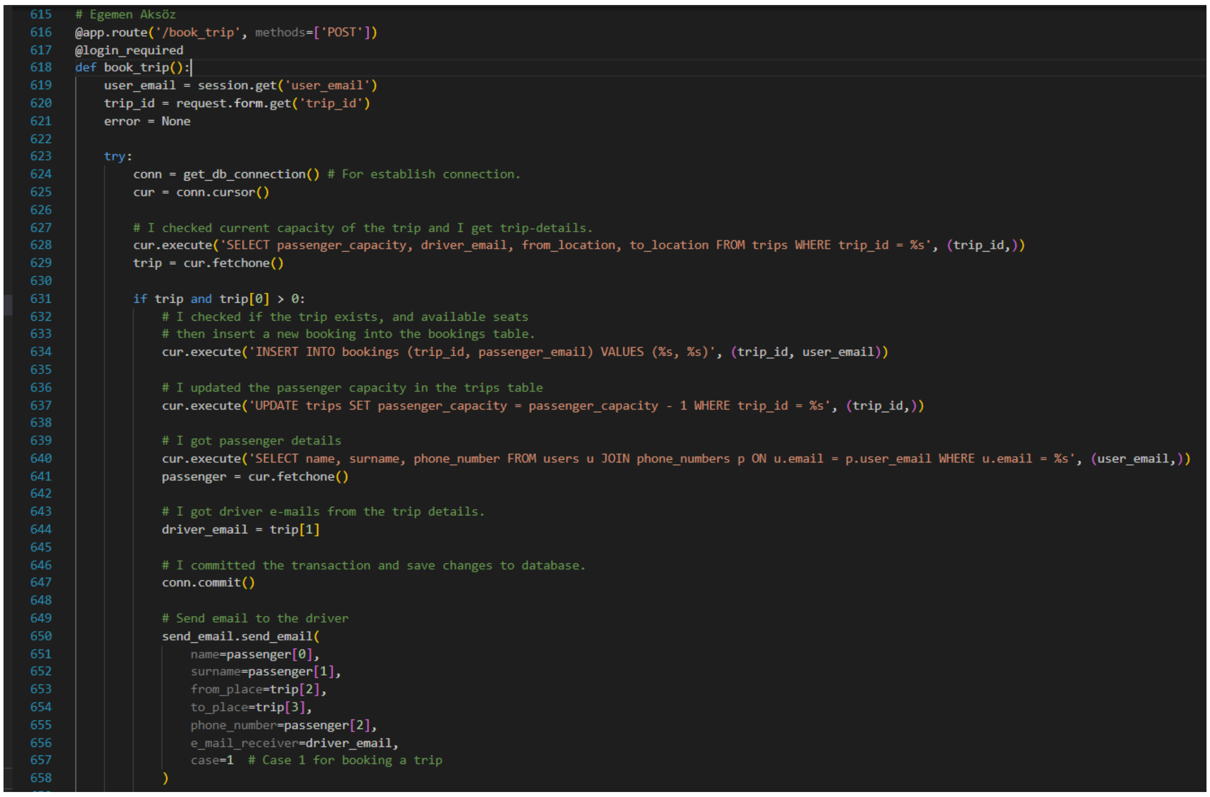
Get Trips

From: Lefkoşa
To: Girne

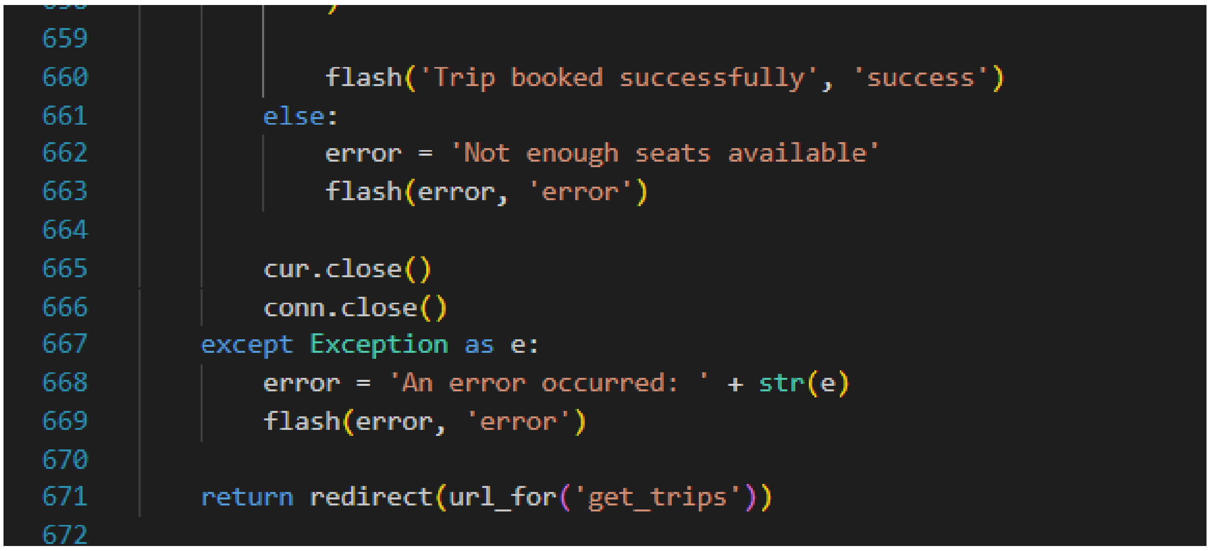
Available Trips

From:	To:	Capacity:	Price:	Driver:	Vehicle:
Lefkoşa	Girne	4	225.00	awaytothrow148@gmail.com	34BJK1903

And now the **/book_trip** part of the code does its job.



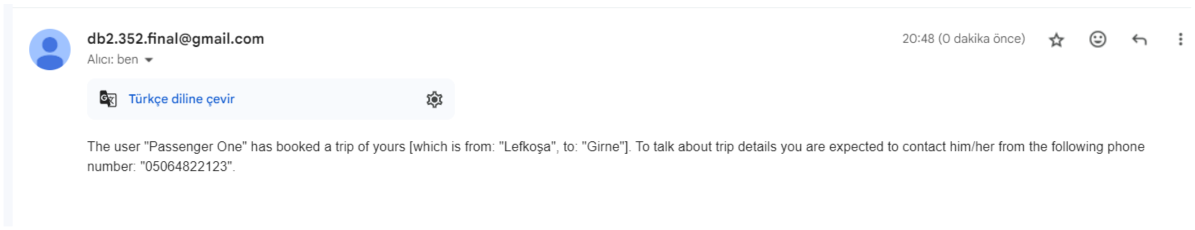
```python
# Egemen Aksöz
@app.route('/book_trip', methods=['POST'])
@login_required
def book_trip():
    user_email = session.get('user_email')
    trip_id = request.form.get('trip_id')
    error = None

    try:
        conn = get_db_connection() # For establish connection.
        cur = conn.cursor()

        # I checked current capacity of the trip and I get trip-details.
        cur.execute('SELECT passenger_capacity, driver_email, from_location, to_location FROM trips WHERE trip_id = %s', (trip_id,))
        trip = cur.fetchone()

        if trip and trip[0] > 0:
            # I checked if the trip exists, and available seats
            # then insert a new booking into the bookings table.
            cur.execute('INSERT INTO bookings (trip_id, passenger_email) VALUES (%s, %s)', (trip_id, user_email))

            # I updated the passenger capacity in the trips table
            cur.execute('UPDATE trips SET passenger_capacity = passenger_capacity - 1 WHERE trip_id = %s', (trip_id,))

            # I got passenger details
            cur.execute('SELECT name, surname, phone_number FROM users u JOIN phone_numbers p ON u.email = p.user_email WHERE u.email = %s', (user_email,))
            passenger = cur.fetchone()

            # I got driver e-mails from the trip details.
            driver_email = trip[1]

            # I committed the transaction and save changes to database.
            conn.commit()

            # Send email to the driver
            send_email.send_email(
                name=passenger[0],
                surname=passenger[1],
                from_place=trip[2],
                to_place=trip[3],
                phone_number=passenger[2],
                e_mail_receiver=driver_email,
                case=1  # Case 1 for booking a trip
            )
```



```python

            flash('Trip booked successfully', 'success')
        else:
            error = 'Not enough seats available'
            flash(error, 'error')

        cur.close()
        conn.close()
    except Exception as e:
        error = 'An error occurred: ' + str(e)
        flash(error, 'error')

    return redirect(url_for('get_trips'))

```

And books the trip.

The user “Kaan Tandogan“ receives an e-mail from the system.



db2.352.final@gmail.com
Alıcı: ben

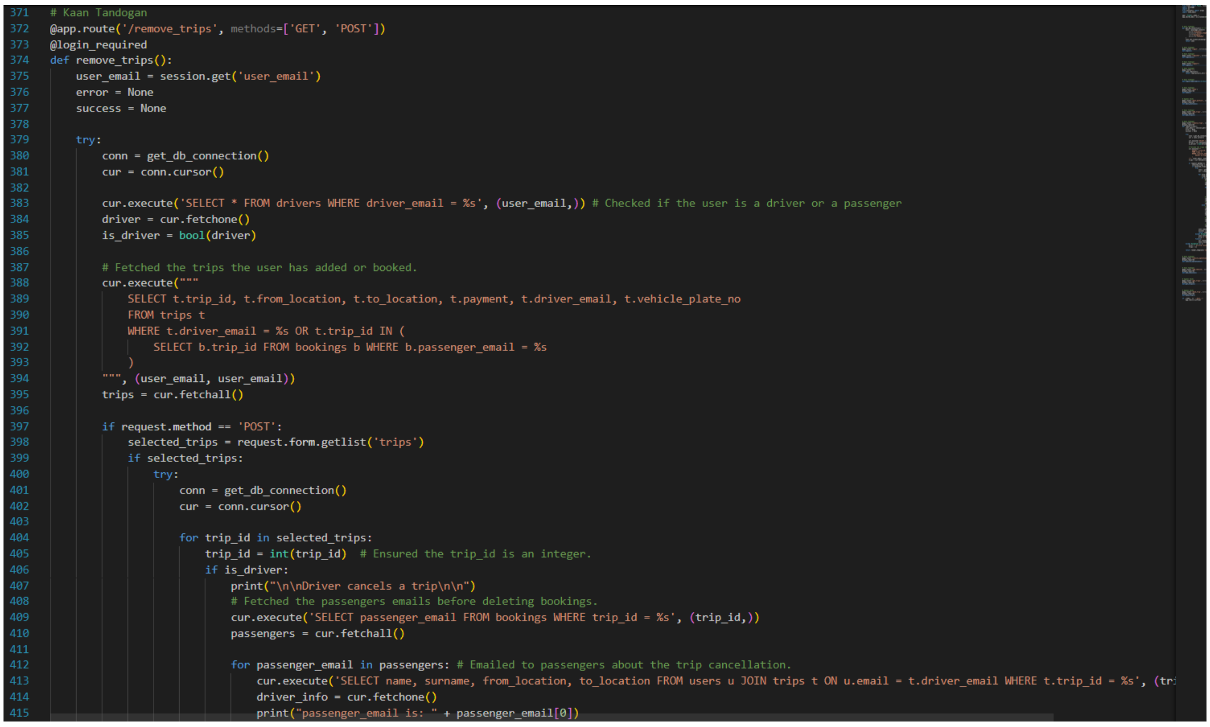
20:48 (0 dakika önce)

Türkçe diline çevir

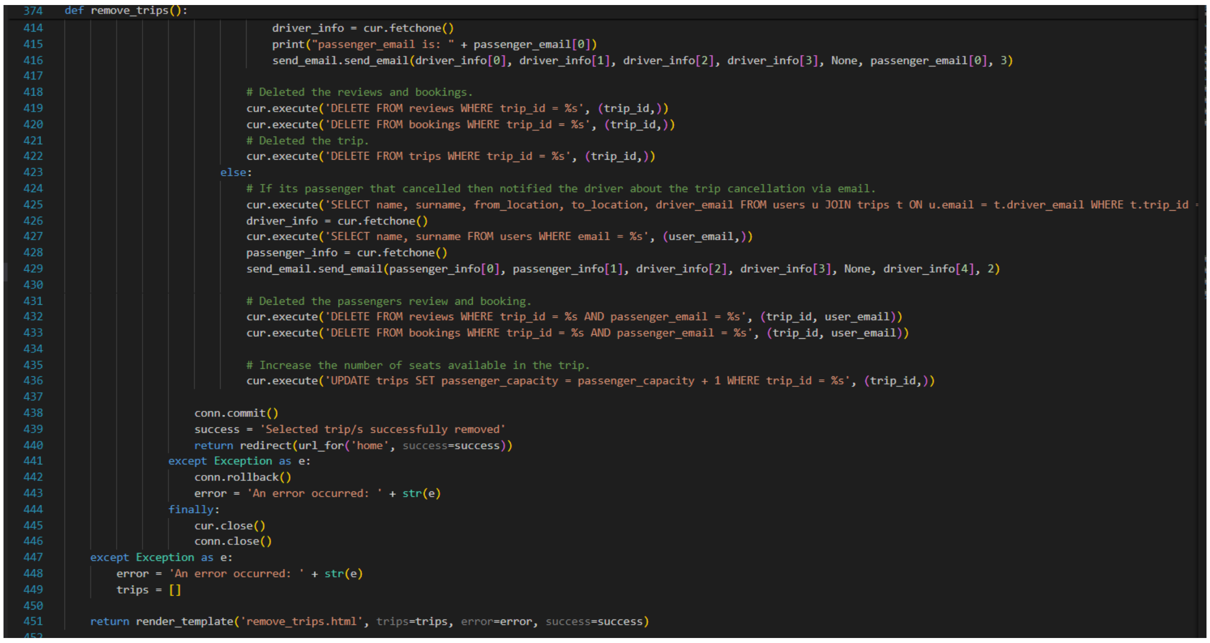
The user "Passenger One" has booked a trip of yours [which is from: "Lefkoşa", to: "Girne"]. To talk about trip details you are expected to contact him/her from the following phone number: "05064822123".

/remove_trips

We have also logged in as “passenger two” and booked on this trip as well. Now we logged in as “passenger one” and we will cancel the trip.



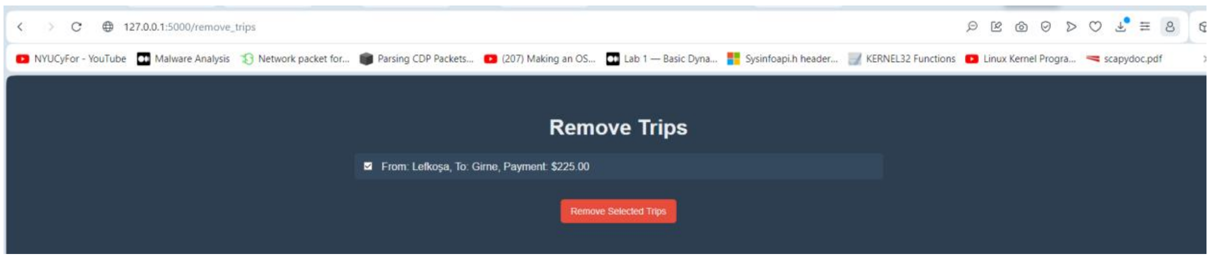
```python
# Kaan Tandogan
@app.route('/remove_trips', methods=['GET', 'POST'])
@login_required
def remove_trips():
    user_email = session.get('user_email')
    error = None
    success = None

    try:
        conn = get_db_connection()
        cur = conn.cursor()

        cur.execute('SELECT * FROM drivers WHERE driver_email = %s', (user_email,)) # Checked if the user is a driver or a passenger
        driver = cur.fetchone()
        is_driver = bool(driver)

        # Fetched the trips the user has added or booked.
        cur.execute("""
            SELECT t.trip_id, t.from_location, t.to_location, t.payment, t.driver_email, t.vehicle_plate_no
            FROM trips t
            WHERE t.driver_email = %s OR t.trip_id IN (
                SELECT b.trip_id FROM bookings b WHERE b.passenger_email = %s
            )
        """, (user_email, user_email))
        trips = cur.fetchall()

        if request.method == 'POST':
            selected_trips = request.form.getlist('trips')
            if selected_trips:
                try:
                    conn = get_db_connection()
                    cur = conn.cursor()

                    for trip_id in selected_trips:
                        trip_id = int(trip_id)  # Ensured the trip_id is an integer.
                        if is_driver:
                            print("\n\nDriver cancels a trip\n\n")
                            # Fetched the passengers emails before deleting bookings.
                            cur.execute('SELECT passenger_email FROM bookings WHERE trip_id = %s', (trip_id,))
                            passengers = cur.fetchall()

                            for passenger_email in passengers: # Emailed to passengers about the trip cancellation.
                                cur.execute('SELECT name, surname, from_location, to_location FROM users u JOIN trips t ON u.email = t.driver_email WHERE t.trip_id = %s', (tri
                                driver_info = cur.fetchone()
                                print("passenger_email is: " + passenger_email[0])
```



```python
def remove_trips():
                                driver_info = cur.fetchone()
                                print("passenger_email is: " + passenger_email[0])
                                send_email.send_email(driver_info[0], driver_info[1], driver_info[2], driver_info[3], None, passenger_email[0], 3)

                            # Deleted the reviews and bookings.
                            cur.execute('DELETE FROM reviews WHERE trip_id = %s', (trip_id,))
                            cur.execute('DELETE FROM bookings WHERE trip_id = %s', (trip_id,))
                            # Deleted the trip.
                            cur.execute('DELETE FROM trips WHERE trip_id = %s', (trip_id,))
                        else:
                            # If its passenger that cancelled then notified the driver about the trip cancellation via email.
                            cur.execute('SELECT name, surname, from_location, to_location, driver_email FROM users u JOIN trips t ON u.email = t.driver_email WHERE t.trip_id =
                            driver_info = cur.fetchone()
                            cur.execute('SELECT name, surname FROM users WHERE email = %s', (user_email,))
                            passenger_info = cur.fetchone()
                            send_email.send_email(passenger_info[0], passenger_info[1], driver_info[2], driver_info[3], None, driver_info[4], 2)

                            # Deleted the passengers review and booking.
                            cur.execute('DELETE FROM reviews WHERE trip_id = %s AND passenger_email = %s', (trip_id, user_email))
                            cur.execute('DELETE FROM bookings WHERE trip_id = %s AND passenger_email = %s', (trip_id, user_email))

                            # Increase the number of seats available in the trip.
                            cur.execute('UPDATE trips SET passenger_capacity = passenger_capacity + 1 WHERE trip_id = %s', (trip_id,))

                    conn.commit()
                    success = 'Selected trip/s successfully removed'
                    return redirect(url_for('home', success=success))
                except Exception as e:
                    conn.rollback()
                    error = 'An error occurred: ' + str(e)
                finally:
                    cur.close()
                    conn.close()
    except Exception as e:
        error = 'An error occurred: ' + str(e)
        trips = []

    return render_template('remove_trips.html', trips=trips, error=error, success=success)
```

After we checked the checkbox right to the trip we can drop it

Remove Trips Page

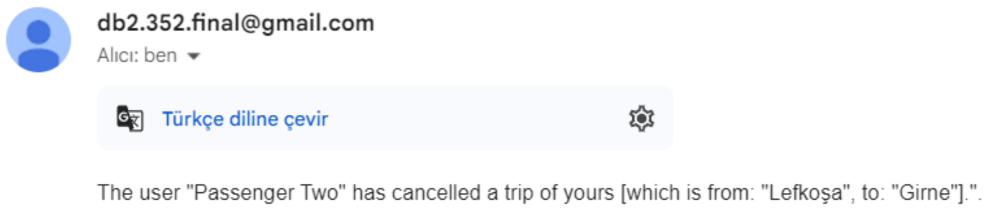


Remove Trips

☑ From: Lefkoşa, To: Girne, Payment: $225.00

Remove Selected Trips

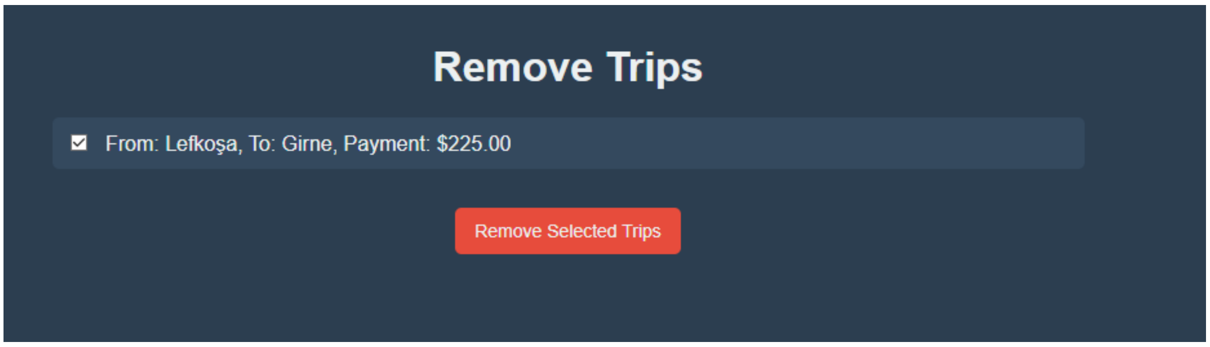
And as you can see the driver received an email



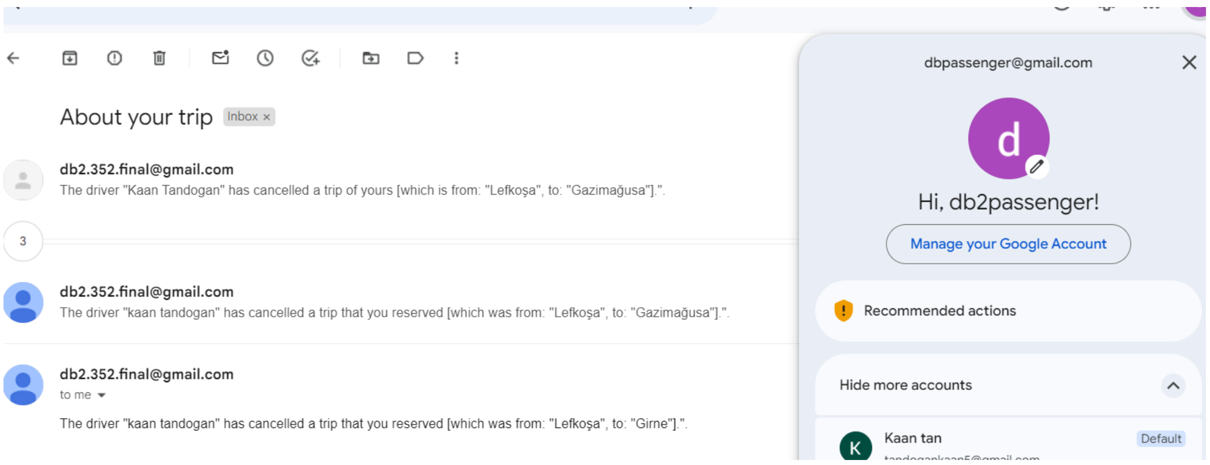
db2.352.final@gmail.com
Alıcı: ben

The user "Passenger Two" has cancelled a trip of yours [which is from: "Lefkoşa", to: "Girne"].".

And now let’s cancel the trip as a driver



The “Passenger One“ receives an email.



About your trip

db2.352.final@gmail.com
The driver "Kaan Tandogan" has cancelled a trip of yours [which is from: "Lefkoşa", to: "Gazimağusa"].".

3

db2.352.final@gmail.com
The driver "kaan tandogan" has cancelled a trip that you reserved [which was from: "Lefkoşa", to: "Gazimağusa"].".

db2.352.final@gmail.com
to me
The driver "kaan tandogan" has cancelled a trip that you reserved [which was from: "Lefkoşa", to: "Girne"].".