1. Background Survey

Pass

2. Requirements

1）Customers

- **Log in/out**
  - Using Django’s built-in authentication system:
  - Users can register, log in, and log out.
  - $Users may can find back their password.
- **Search vehicles**: Search vehicle information based on location, type, available time, etc.
- **Rent vehicles**: Users can rent a vehicle at any location in the city as long as there is an available working vehicle at that location.
- **Return vehicles**: Users can return a vehicle to any location, and their account will be charged based on the rental duration and vehicle type.
- **Report vehicles**: Allows users to report rented defective vehicles.
- **Order management**: Includes the following points
  - Create order (add): When renting a vehicle, an order is created.
  - $Modify order (modify): Users can modify order details before the rental period ends.
  - $Search order (search): Users can view past orders.
  - Close order: Once a vehicle is returned and the bill is paid, the order is closed.
- **$Prebook**
- **$Profile management**: Includes the following points
  - Create profile: Users create an account during registration.
  - Modify profile: Users can modify personal information.
  - Cancel profile (or cancel account): Users can delete their accounts.
  - Search profile: Users can view and retrieve their own information.
- **Comment/feedback**: Users can provide feedback or leave comments on the vehicles or services (potentially part of "Order management").

2）Operators

- **Vehicle info management**: Includes the following points

    - Add vehicles: Operators can add vehicles at different city locations as needed.
    - Modify vehicles: Modify both static (e.g., type, name, ID, creation time) and dynamic (e.g., status: repairing, charging) information.
        - **Charge**: Operators can charge vehicles when battery levels are low.
        - **Repair**: Operators can repair defective vehicles.
        - **Move**: Operators can relocate vehicles to different locations within the city.
    - Search vehicles: Search vehicles based on conditions like location and status.
        - **Track**: Operators can track the real-time location of all vehicles in the city.
    - Delist vehicles: Operators can remove vehicles from the system.
- **¥Order management**: Operators can search and modify orders but cannot create or delete them.

3）Managers

- **Generate reports**: Generate reports showing all vehicle activities over a defined time period using appropriate data visualization techniques.
- **$Profile management**: It’s unclear if operators manage their accounts or are managed by administrators.

3. Design

3.1 Total Design

The system will be built with a separation of the front end and back end:

- **UI**: The front end will use Bootstrap (HTML, CSS, JavaScript) for a responsive design.
- **Backend**: The back end will use Django (Python) with Django REST Framework to create APIs.
- **Database**: Options include MySQL, PostgreSQL, or SQLite as the database management system.
- **APIs**: The system may need to integrate with third-party APIs like payment APIs and Google Maps API.
- **Testdata for the table:**

3.2 Database Design

1. **User Table**
    - user_id
    - name
    - email
    - password
    - payment_info
    - registration_date
    - type_of_person:
    - is_deleted:
    - phone_number

2. **Vehicle Table**
    - vehicle_id (Primary key)
    - type (e.g., Scooter, Bike)
    - location_id
    - battery_level
    - status (e.g., Available, In use, Defective,not in use)
    - last_service_date
    - service by which operator

3. **Order Table**
    - order_id (Primary key)
    - customer_id
    - vehicle_id
    - start_time
    - return_time
    - end_time
    - total_price
    - status (e.g., Open, Closed)
    - Fault Report
    - Remarks
    - rented_location
    - returned_location

4. **Payment Table**
    - payment_id (Primary key)
    - order_id

    - payment_method
    - amount
    - payment_date

5. **City table**:
    - city_id
    - postcode
    - cityname
    - service_location

6. **Reports (View)**: Dynamic views can be generated to show vehicle usage, financial data, etc., over a specific time frame.
7. **Charging Station Table:** Records the location of each charging station and the number of available charging points.

#3.3 Maps API Integration(Choose one)

Two method is recommended:
1.Google Maps API for the following functionalities:

(I have checked for the feasibility of using Google Maps API)

- Display the distribution of vehicles across the city, allowing users to choose the nearest available vehicle.
- Real-time vehicle tracking for operators to adjust and optimize vehicle dispatch.
- Provide users with the closest rental locations and best routes.

2.Or we screen shot the map and use the map to show the location of the vehicle.

3.4 $ Permission System

Django's built-in permission system will be used to manage API access for different user roles (customers, operators, managers):

- **Customers**: Can only access their own account information and order history.
- **Operators**: Can access vehicle management interfaces and view orders but cannot create orders.
- **Managers**: Have full access to account management, order data, and financial reports.

3.5 Security & Deployment(That's for document writing maybe?)

- **User data protection**: All user passwords and payment information must be securely encrypted.
- **Cloud deployment**: Consider deploying the Django application on a cloud platform like AWS or Heroku, and configure the database and domain settings appropriately.