🎓 Django Assignment — Vehicle Management API (Drive Exercise)

📌 Problem Statement

You are required to build a **Vehicle Management REST API** using **Django + SQLite**.
This API will allow users to manage a company's vehicle fleet.

Your API must support:

- Adding a vehicle (POST)
- Listing all vehicles (GET)
- Updating a vehicle (PUT)
- Deleting a vehicle (DELETE)
- Storing all data in SQLite database using Django Models
- Testing through Postman

🎯 Objectives

By completing this assignment, you will learn:

- How to create Django models
- How to run migrations
- How to handle GET/POST/PUT/DELETE in Django
- How to parse JSON request bodies
- How to return JSON responses
- How to test APIs with Postman

📂 Tasks / Requirements

1. Create the Vehicle Model

Inside `api/models.py`, create a model with fields:

Field	Type	Description
number_plate	CharField	Vehicle registration number
vehicle_type	CharField	Car, Bike, Truck
manufacturer	CharField	Vehicle brand
year	IntegerField	Manufacturing year

Run migrations:

```
python manage.py makemigrations
python manage.py migrate
```

---

2. Build a CRUD API using Raw Django Views

Create a single endpoint:

```
/vehicles/
```

Requirements:

✔ GET → Return all vehicles

JSON array of all records.

✔ POST → Insert a new vehicle

Body example:

```
{
  "number_plate": "TS09AB1234",
  "vehicle_type": "Car",
  "manufacturer": "Toyota",
  "year": 2020
}
```

✔ PUT → Update an existing vehicle

Body example:

```
{
  "id": 1,
  "number_plate": "TS09AB9999",
  "vehicle_type": "Car",
  "manufacturer": "Toyota",
  "year": 2022
}
```

✔ DELETE → Delete a vehicle

Body example:

```
{"id": 1}
```

Use:

```
from django.views.decorators.csrf import csrf_exempt
```

---

3. Add URL Mapping

`api/urls.py` → map `/vehicles/` to your view.

`myproject/urls.py` → include the app URLs.

---

4. Test All Endpoints in Postman

You must test:

- GET
- POST
- PUT
- DELETE

with correct request bodies.

Include screenshots in your submission.

🧪 Test Cases

Students must validate that the API behaves correctly using the following test cases:

Test Case 1 — Add a vehicle

Method: POST
Body:

```
{
  "number_plate": "TS09AB1234",
  "vehicle_type": "Car",
  "manufacturer": "Toyota",
  "year": 2020
}
```

Expected:

`{"message": "POST EXECUTED"}`

Test Case 2 — Get all vehicles

Method: GET
Expected:
A JSON array containing all inserted vehicles.

Example:

```
[
  {
    "id": 1,
    "number_plate": "TS09AB1234",
    "vehicle_type": "Car",
    "manufacturer": "Toyota",
    "year": 2020
  }
]
```

Test Case 3 — Update a vehicle

Method: PUT
Body:

```
{
  "id": 1,
  "number_plate": "TS09AB5555",
  "vehicle_type": "Car",
  "manufacturer": "Toyota",
  "year": 2021
}
```

Expected:

```
{"message": "PUT EXECUTED"}
```

Test Case 4 — Delete a vehicle

Method: DELETE
Body:

```
{"id": 1}
```

Expected:

```
{"message": "DELETE EXECUTED"}
```

Test Case 5 — Validate DB after DELETE

Method: GET
Expected:
The deleted vehicle must no longer appear in the list.

📝 Expected Output Summary

Operation	Expected Output
POST	“POST EXECUTED”
GET	List of all vehicles
PUT	“PUT EXECUTED”
DELETE	“DELETE EXECUTED”