Table of Contents

6.Deployment Pipeline Overview (Optional)

- **Build Stage**
- **Test Stage**
- **Staging Deployment**
- **Approval Gate**
- **Production Deployment**
- **Rollback Procedure**

7.Security Considerations (Optional)

- **Authentication**
- **Authorization**
- **Transport Security**
- **Configuration & Secrets**
- **CORS Policy**
- **Security Considerations**

1. Production Support & Testing Scenarios

1.1 Service Dependency Diagram

The system is deployed on Azure and consists of three primary components:

```
End Users (Buyers / Sellers / Admin)
       │
       ▼
React Frontend (Azure Web App, Node.js 22 LTS)
       │  REST API over HTTPS
       ▼
Flask Backend API (Azure Web App, Python 3.10, SQLAlchemy)
       │  Secure SSL connection
       ▼
Azure PostgreSQL Flexible Server (usedcars database)
```

Frontend: React application hosted on Azure Web App.

Backend: Flask REST API deployed on Azure Web App, implementing business logic such as user authentication, vehicle CRUD, and admin approval.

Database: Azure PostgreSQL Flexible Server storing users, vehicles, and admin data.

External APIs: None currently; image URLs are provided by users.

1.2 Monitoring

Backend logs: Captured in Azure App Service → Log Stream. Logs contain API request/response information and errors.

Frontend logs: Available through browser console (JavaScript or CORS errors) and Azure App Service access logs.

Database monitoring: Provided in Azure portal metrics, including active connections, query performance, and deadlocks.

Health checks:

Health checks

Current method (in use)
- Business probe: `GET /cars/approved` → returns **HTTP 200** and a JSON array (even if empty).
- DB liveness: lightweight `SELECT 1` from the application to verify connectivity.

**Optional endpoints **
- `GET /health` — *Liveness*: returns **200 OK** with `{"status":"ok"}`; no DB call.
- `GET /health/db` — *Readiness*: runs `SELECT 1`; returns **200** (`{"db":"connected"}`) or **503** if unreachable.
- **Security**: Never return secrets; if publicly reachable, restrict by IP/allow-list or require an internal header.

Operational notes

- If the optional endpoints are added later, configure **Azure App Service → Health check** to probe `/health` every **30s**.
- Trigger an alert on 2 consecutive liveness failures (~1 min) or readiness failing for > **3 min**.

1.3 Common Incidents & Recovery Steps

Although no blocking incidents are currently observed, the following are common issues maintainers should prepare for:

1.Database connection loss

Symptoms: API returns 500 errors, frontend list page does not load.

Resolution: Verify DATABASE_URL environment variable, check Azure PostgreSQL firewall rules and SSL requirement, restart backend app.

2.Backend crash or failed deployment

Symptoms: 502/503 returned on all endpoints.

Resolution: Restart backend Web App, roll back to the last working deployment, or fix dependency errors.

3.CORS errors

Symptoms: Browser console shows “CORS blocked” errors.

Resolution: Update Flask flask-cors configuration to allow the deployed frontend origin.

4.Invalid image URLs

Symptoms: Broken images in car listing cards.

Resolution: Require HTTPS URLs; optionally store images in Azure Blob Storage with fallback placeholders.

5.Missing admin account

Symptoms: Admin cannot log in to approve cars.

Resolution: Create admin user via POST /admin/users and verify login via POST /login (as confirmed in API test document).

1.4 Testing Scenarios & Results

1.4.1 Unit Test Cases

These are basic validation and utility tests at the function/model level.

ID	Function / Module	Input	Expected Result	Actual Result	Status
UT-1	VIN Validator	`VIN=INVALID123`	400 Bad Request with validation error	Application rejects invalid VIN	Passed
UT-2	Mileage Validation	`mileage=-100`	400 Bad Request with validation error	Application rejects input	Passed
UT-3	Price Validation	`price=-5000`	400 Bad Request with validation error	Application rejects input	Passed
UT-4	Password Hash/Verify	Plaintext password → hash → verify	Verify returns True	Function works correctly	Passed
UT-5	Username Uniqueness	Register duplicate username	409 Conflict or validation error	Application rejects duplicate	Passed

1.4.2 Integration / API Test Cases

1. **Post Users:**

Test Case 1- Create User 'alice-wang'

Endpoint: POST/users

Request JSON:

```
{
  "username": "alice_wang",
  "password": "securePass123"
}
```

Expected Result:

Status Code: 201 Created

Resopnse JSON:

```
{
  "data": {
          "id": 1,
          "username": "alice_wang"
  },
  "status": "success"
}
```

Actual Result: Passed

Test Case 2 – Create User ‘david_liu’

Endpoint: POST/users

Request JSON:

```
{
  "username": "david_liu",
  "password": "myPassword456"
}
```

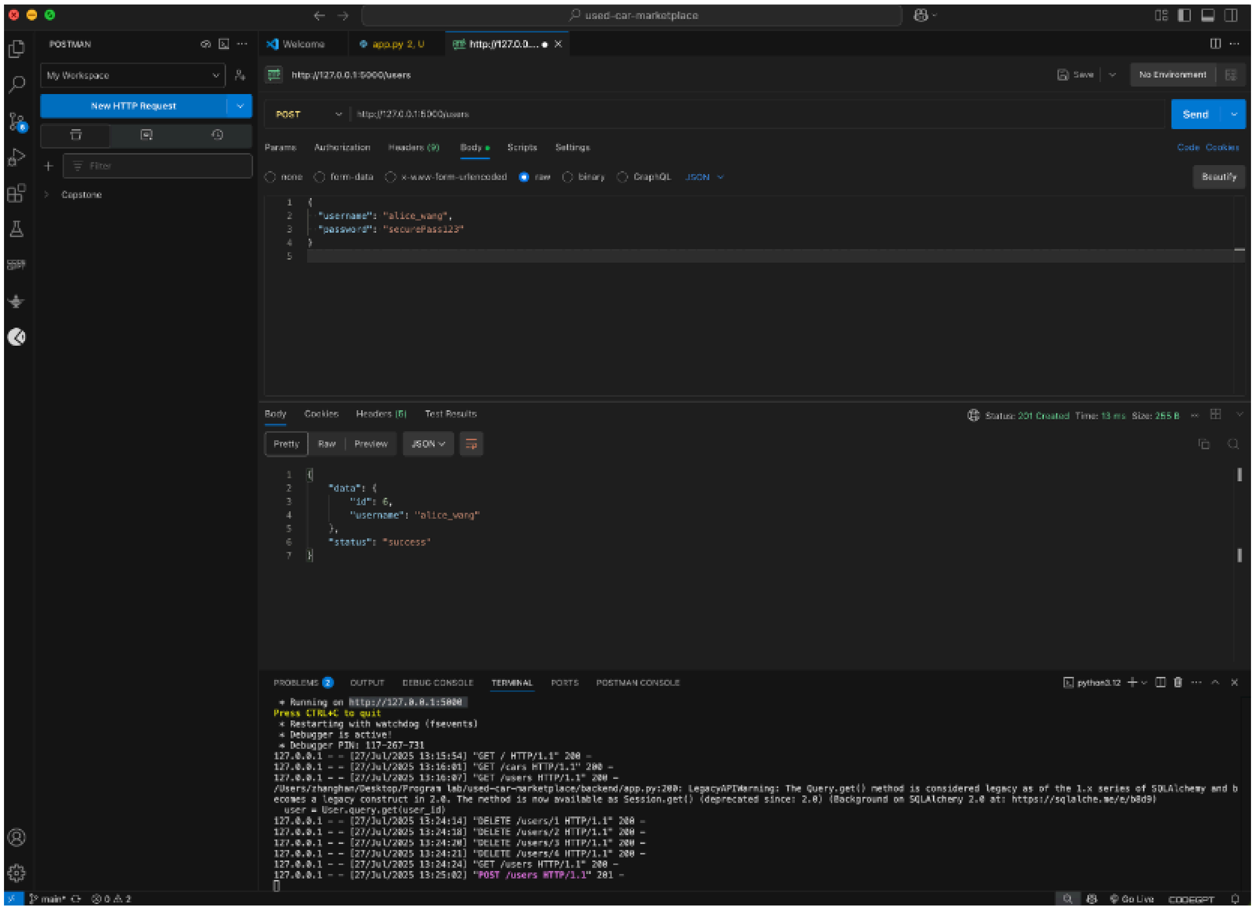
Expected Result:

Status Code: 201 Created

Resopnse JSON:

```
{
  "data": {
          "id": 2,
          "username": "david_liu"
  },
  "status": "success"
}
```

Actual Result: Passed



POST http://127.0.0.1:5000/users

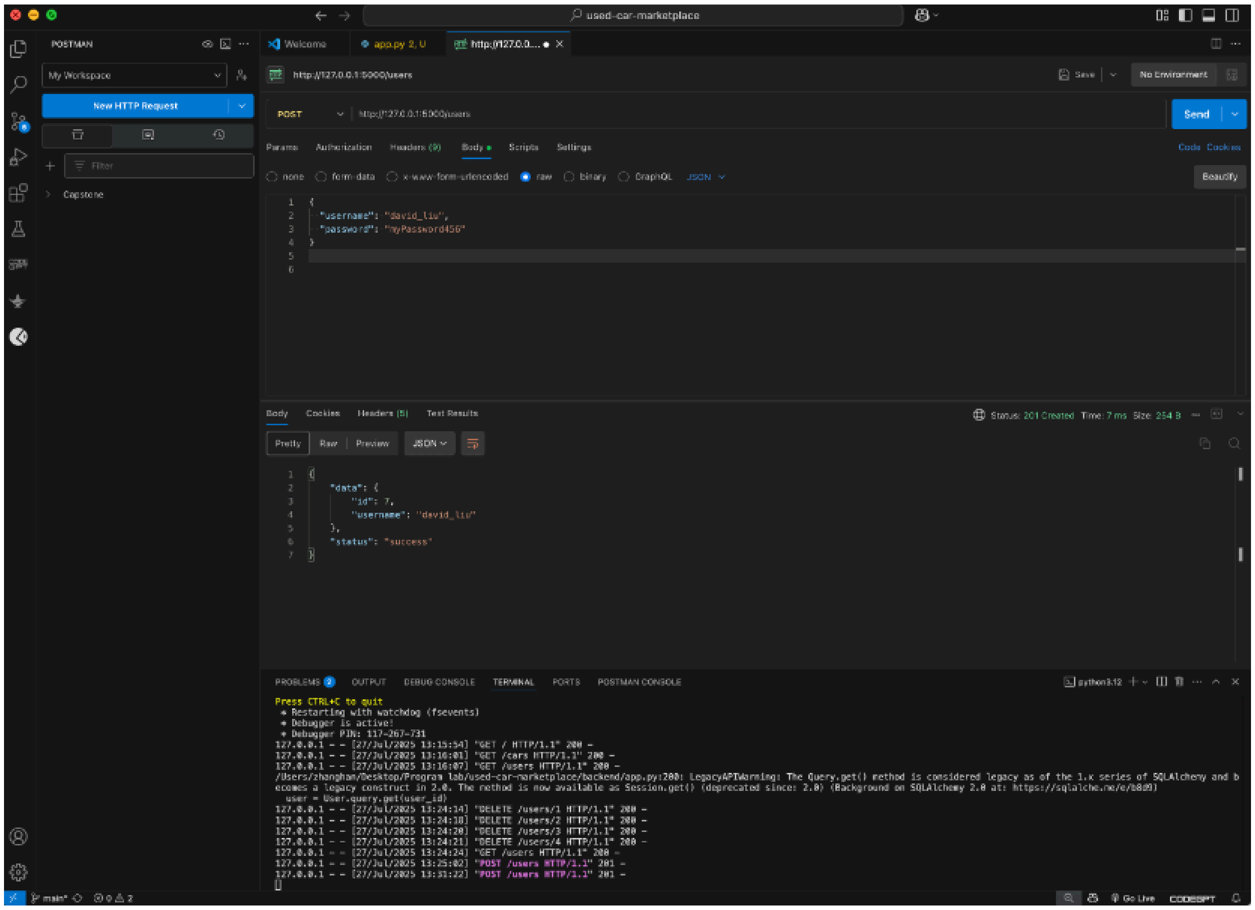
none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
  "username": "alice_wang",
  "password": "securePass123"
}
```

Body — Status: 201 Created Time: 13 ms Size: 255 B — JSON

```
{
    "data": {
        "id": 6,
        "username": "alice_wang"
    },
    "status": "success"
}
```

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 13:15:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:16:01] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:16:07] "GET /users HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:200: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 13:24:14] "DELETE /users/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:18] "DELETE /users/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:20] "DELETE /users/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:21] "DELETE /users/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:24] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:25:02] "POST /users HTTP/1.1" 201 -
```

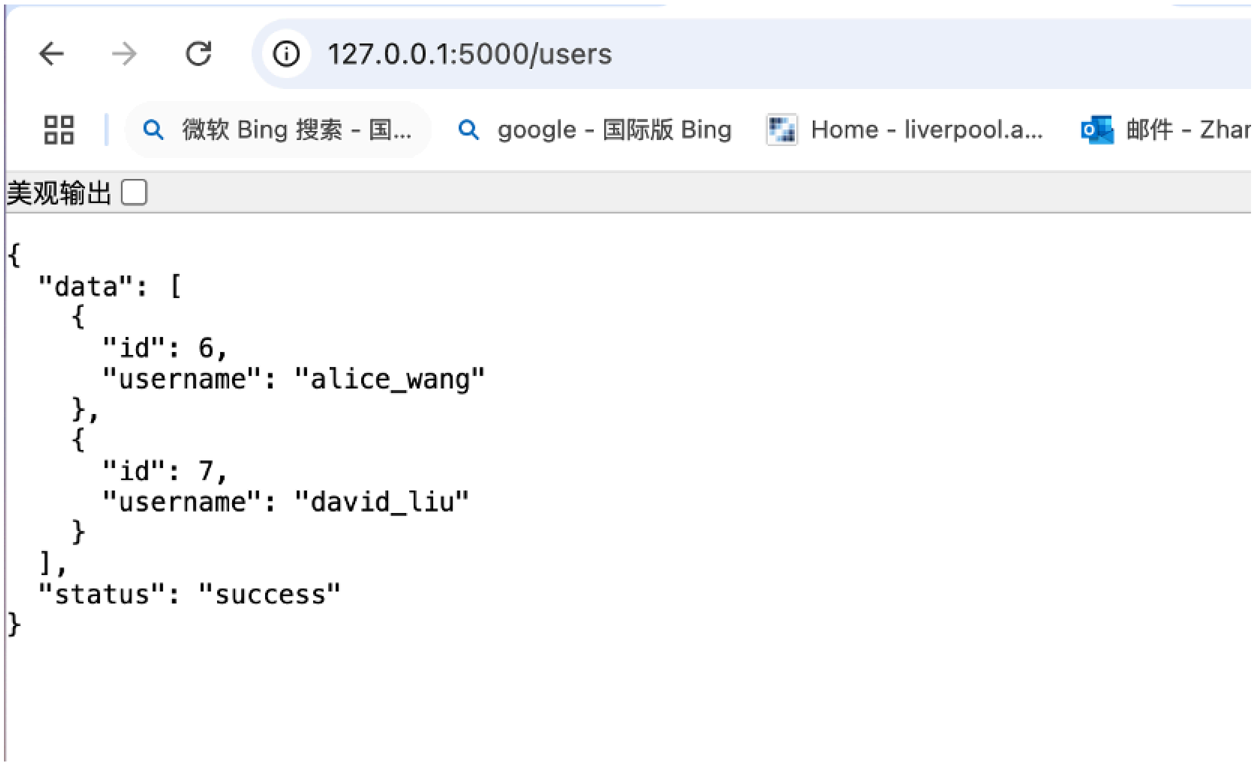


```
{
  "username": "david_liu",
  "password": "myPassword456"
}
```

Status: 201 Created Time: 7 ms Size: 254 B

```
{
    "data": {
        "id": 7,
        "username": "david_liu"
    },
    "status": "success"
}
```

```
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 13:15:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:16:01] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:16:07] "GET /users HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:200: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 13:24:14] "DELETE /users/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:18] "DELETE /users/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:20] "DELETE /users/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:21] "DELETE /users/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:24:24] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:25:02] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:31:22] "POST /users HTTP/1.1" 201 -
```



```
{
  "data": [
    {
      "id": 6,
      "username": "alice_wang"
    },
    {
      "id": 7,
      "username": "david_liu"
    }
  ],
  "status": "success"
}
```

2. **Post Cars:**

Test Case 1 – Add BMW X5 (by alice_wang)

Endpoint: POST/cars

Request JSON:

```
{
  "make": "BMW",
  "model": "X5",
  "year": 2021,
  "price": 42000.0,
  "mileage": 15000,
  "fuelType": "Gasoline",
  "transmission": "Automatic",
  "vin": "WBAXX1234BMWX5",
  "location": "New York, NY",
  "contactInfo": "alice.wang@example.com",
  "description": "Luxury SUV in excellent condition.",
  "imageUrl": "https://example.com/bmw-x5.jpg",
  "submittedBy": "alice_wang"
}
```

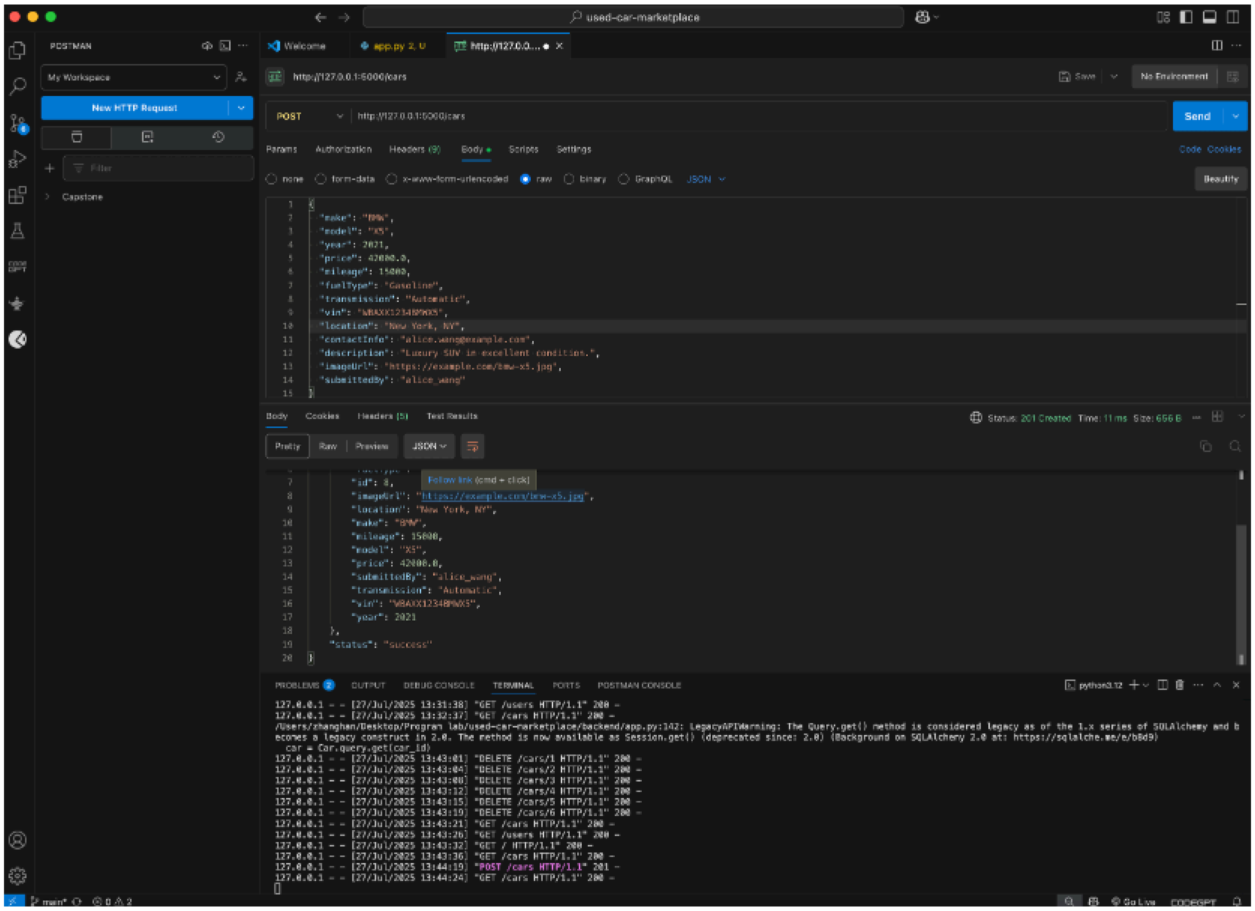
Expected Result:

Status Code: 201 Created

Response JSON:

```
{
        "data": {
        "approved": false,
        "contactInfo": "alice.wang@example.com",
        "description": "Luxury SUV in excellent condition.",
        "fuelType": "Gasoline",
        "id": 8,
        "imageUrl": "https://example.com/bmw-x5.jpg",
        "location": "New York, NY",
        "make": "BMW",
        "mileage": 15000,
        "model": "X5",
        "price": 42000.0,
        "submittedBy": "alice_wang",
        "transmission": "Automatic",
        "vin": "WBAXX1234BMWX5",
        "year": 2021
        },
        "status": "success"
}
```

Result: Passed


POST http://127.0.0.1:5000/cars

```
{
  "make": "BMW",
  "model": "X5",
  "year": 2021,
  "price": 42000.0,
  "mileage": 15000,
  "fuelType": "Gasoline",
  "transmission": "Automatic",
  "vin": "WBAXX12348MWX5",
  "location": "New York, NY",
  "contactInfo": "alice.wang@example.com",
  "description": "Luxury SUV in excellent condition.",
  "imageUrl": "https://example.com/bmw-x5.jpg",
  "submittedBy": "alice_wang"
}
```

Status: 201 Created Time: 11 ms Size: 656 B

```
        "id": 8,
        "imageUrl": "https://example.com/bmw-x5.jpg",
        "location": "New York, NY",
        "make": "BMW",
        "mileage": 15000,
        "model": "X5",
        "price": 42000.0,
        "submittedBy": "alice_wang",
        "transmission": "Automatic",
        "vin": "WBAXX12348MWX5",
        "year": 2021
    },
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:31:38] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:32:37] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:142: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 13:43:01] "DELETE /cars/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:04] "DELETE /cars/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:08] "DELETE /cars/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
```

Test Case 2 – Add Toyota Corolla (by david_liu)

Endpoint: POST/cars

Request JSON:

{

"make": "Toyota",

"model": "Corolla",

"year": 2018,

"price": 12500.0,

"mileage": 48000,

"fuelType": "Hybrid",

"transmission": "Automatic",

"vin": "JTDBR32EX81234567",

"location": "Chicago, IL",

```
  "contactInfo": "david.liu@example.com",
  "description": "Fuel-efficient and reliable daily driver.",
  "imageUrl": "https://example.com/toyota-corolla.jpg",
  "submittedBy": "david_liu"
}
```

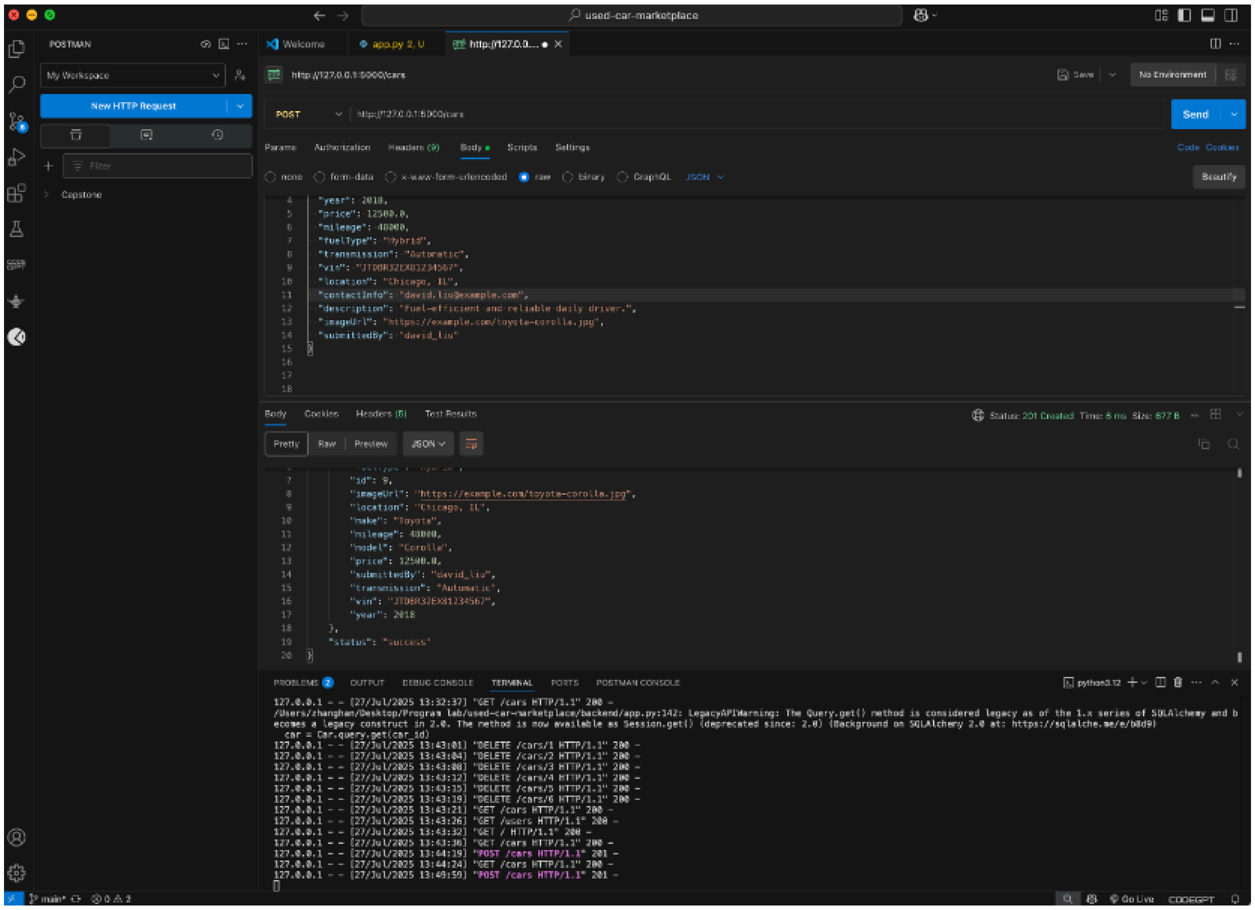
Expected Result:

Status Code: 201 Created

Response JSON:

```
{
        "data": {
        "approved": false,
        "contactInfo": "david.liu@example.com",
        "description": "Fuel-efficient and reliable daily driver.",
        "fuelType": "Hybrid",
        "id": 9,
        "imageUrl": "https://example.com/toyota-corolla.jpg",
        "location": "Chicago, IL",
        "make": "Toyota",
        "mileage": 48000,
        "model": "Corolla",
        "price": 12500.0,
        "submittedBy": "david_liu",
        "transmission": "Automatic",
        "vin": "JTDBR32EX81234567",
        "year": 2018
        },
        "status": "success"
}
```

Result: Passed



POST http://127.0.0.1:5000/cars

```
"year": 2018,
"price": 12500.0,
"mileage": 40000,
"fuelType": "Hybrid",
"transmission": "Automatic",
"vin": "JTDBR32EX01234567",
"location": "Chicago, IL",
"contactInfo": "david.liu@example.com",
"description": "Fuel-efficient and reliable daily driver.",
"imageUrl": "https://example.com/toyota-corolla.jpg",
"submittedBy": "david_liu"
}
```

Status: 201 Created Time: 8 ms Size: 877 B

```
        "id": 9,
        "imageUrl": "https://example.com/toyota-corolla.jpg",
        "location": "Chicago, IL",
        "make": "Toyota",
        "mileage": 40000,
        "model": "Corolla",
        "price": 12500.0,
        "submittedBy": "david_liu",
        "transmission": "Automatic",
        "vin": "JTDBR32EX01234567",
        "year": 2018
    },
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:32:37] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:142: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 13:43:01] "DELETE /cars/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:04] "DELETE /cars/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:08] "DELETE /cars/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
```

Test Case 3 – Add Honda Accord (by alice_wang)

Endpoint: POST/cars

Request JSON:

Expected Result:

Status Code: 201 Created

Response JSON:

{

"data": {

"approved": false,

"contactInfo": "alice.wang@example.com",

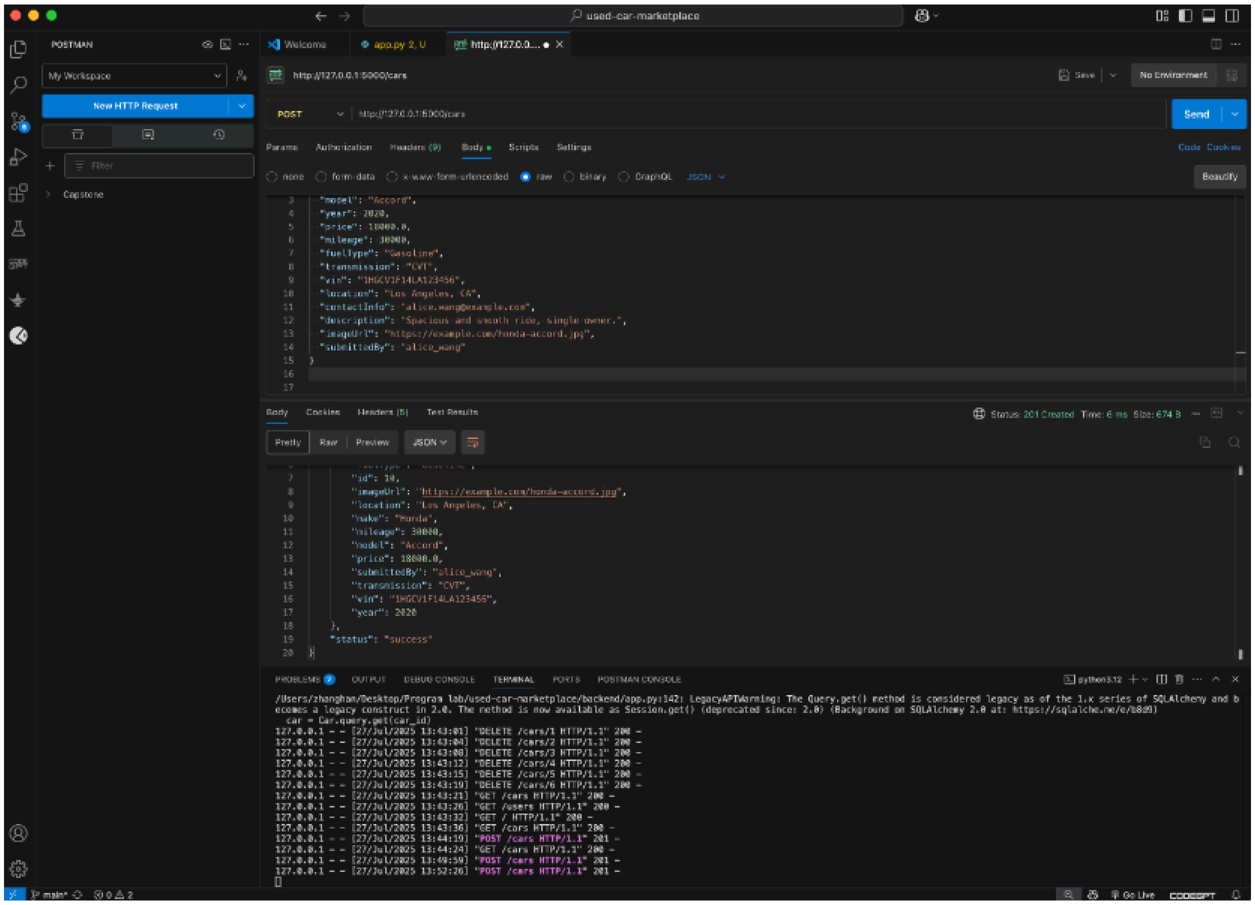
"description": "Spacious and smooth ride, single owner.",

"fuelType": "Gasoline",

"id": 10,

```
        "imageUrl": "https://example.com/honda-accord.jpg",
        "location": "Los Angeles, CA",
        "make": "Honda",
        "mileage": 30000,
        "model": "Accord",
        "price": 18000.0,
        "submittedBy": "alice_wang",
        "transmission": "CVT",
        "vin": "1HGCV1F14LA123456",
        "year": 2020
        },
        "status": "success"
}
```

Result: Passed



POST http://127.0.0.1:5000/cars

```
    "model": "Accord",
    "year": 2020,
    "price": 18000.0,
    "mileage": 30000,
    "fuelType": "Gasoline",
    "transmission": "CVT",
    "vin": "1HGCV1F14LA123456",
    "location": "Los Angeles, CA",
    "contactInfo": "alice.wang@example.com",
    "description": "Spacious and smooth ride, single owner.",
    "imageUrl": "https://example.com/honda-accord.jpg",
    "submittedBy": "alice_wang"
}
```

```
        "id": 10,
        "imageUrl": "https://example.com/honda-accord.jpg",
        "location": "Los Angeles, CA",
        "make": "Honda",
        "mileage": 30000,
        "model": "Accord",
        "price": 18000.0,
        "submittedBy": "alice_wang",
        "transmission": "CVT",
        "vin": "1HGCV1F14LA123456",
        "year": 2020
    },
    "status": "success"
}
```

```
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:142: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 13:43:01] "DELETE /cars/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:04] "DELETE /cars/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:08] "DELETE /cars/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:52:26] "POST /cars HTTP/1.1" 201 -
```

Test Case 4 – Add Tesla Model 3 (by david_liu)

Endpoint: POST/cars

Request JSON:

Expected Result:

Status Code: 201 Created

Response JSON:

{

"data": {

"approved": false,

"contactInfo": "david.liu@example.com",

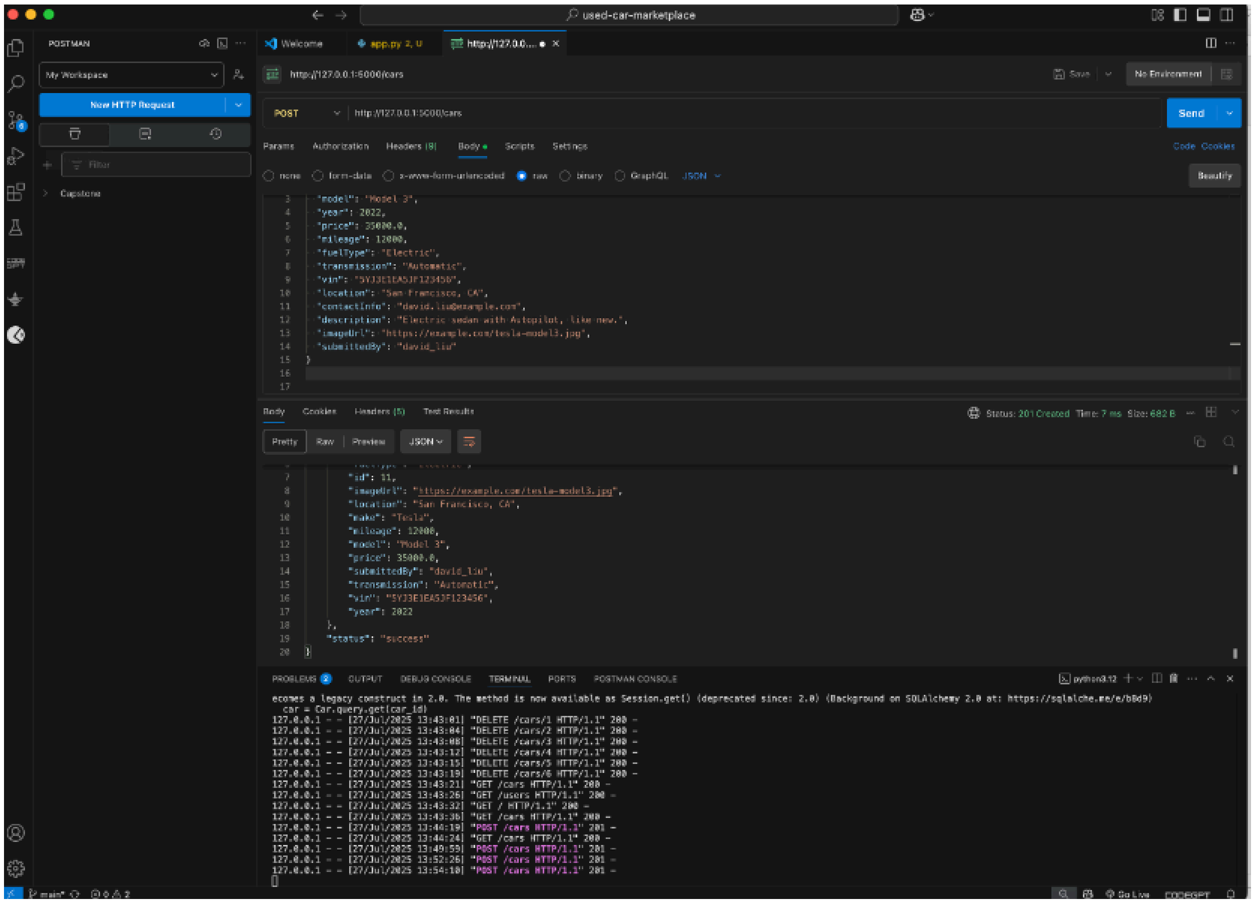
"description": "Electric sedan with Autopilot, like new.",

"fuelType": "Electric",

"id": 11,

```
        "imageUrl": "https://example.com/tesla-model3.jpg",
        "location": "San Francisco, CA",
        "make": "Tesla",
        "mileage": 12000,
        "model": "Model 3",
        "price": 35000.0,
        "submittedBy": "david_liu",
        "transmission": "Automatic",
        "vin": "5YJ3E1EA5JF123456",
        "year": 2022
        },
        "status": "success"
}
```

Result: Passed



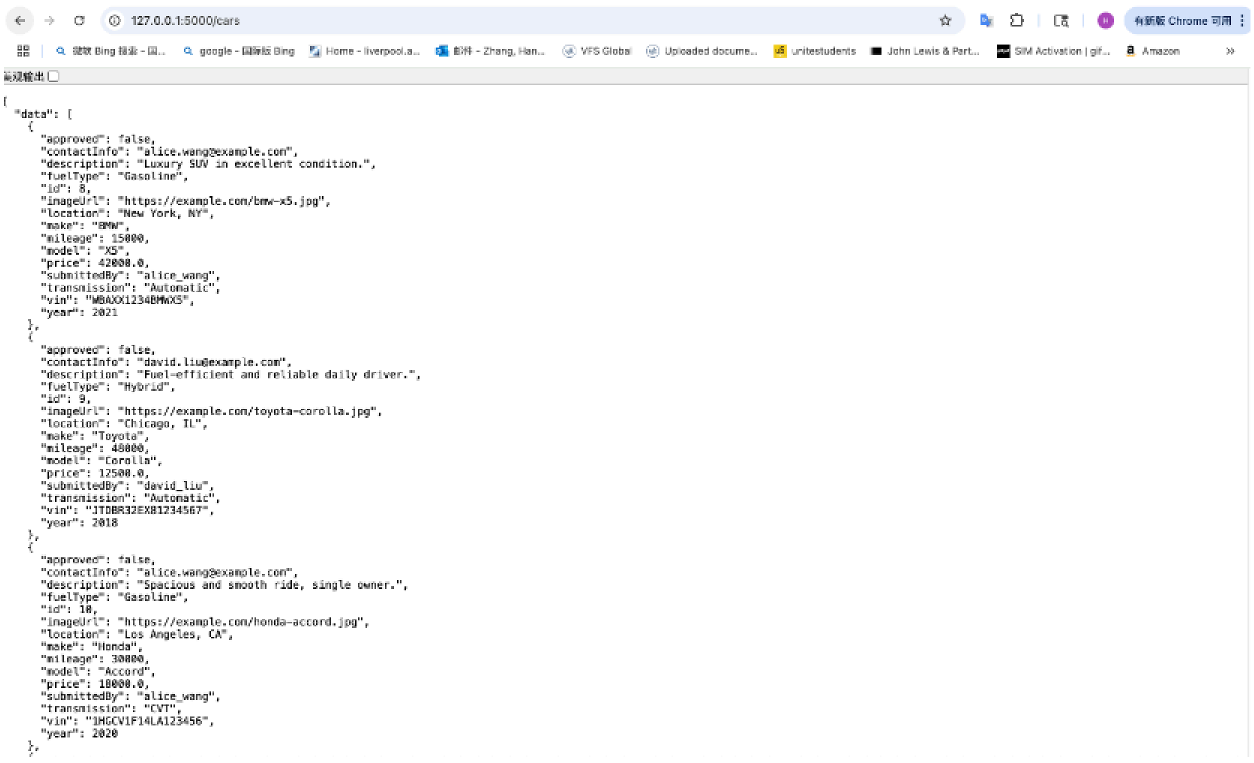
POST http://127.0.0.1:5000/cars

```
  "model": "Model 3",
  "year": 2022,
  "price": 35000.0,
  "mileage": 12000,
  "fuelType": "Electric",
  "transmission": "Automatic",
  "vin": "5YJ3E1EA5JF123456",
  "location": "San Francisco, CA",
  "contactInfo": "david.liu@example.com",
  "description": "Electric sedan with Autopilot, like new.",
  "imageUrl": "https://example.com/tesla-model3.jpg",
  "submittedBy": "david_liu"
}
```

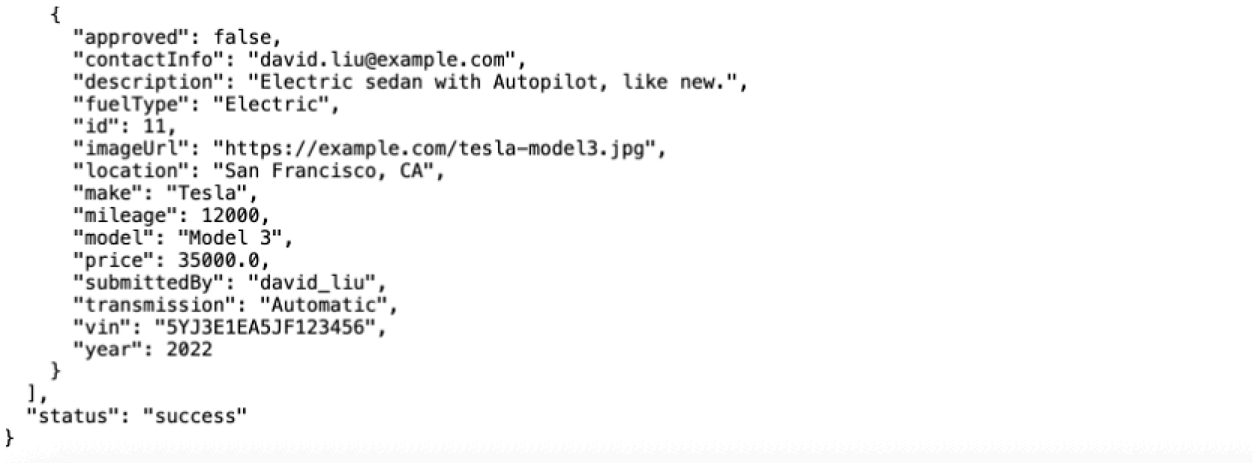
Status: 201 Created Time: 7 ms Size: 682 B

```
      "id": 11,
      "imageUrl": "https://example.com/tesla-model3.jpg",
      "location": "San Francisco, CA",
      "make": "Tesla",
      "mileage": 12000,
      "model": "Model 3",
      "price": 35000.0,
      "submittedBy": "david_liu",
      "transmission": "Automatic",
      "vin": "5YJ3E1EA5JF123456",
      "year": 2022
    },
    "status": "success"
}
```

```
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 13:43:01] "DELETE /cars/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:04] "DELETE /cars/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:08] "DELETE /cars/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:52:26] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:54:10] "POST /cars HTTP/1.1" 201 -
```



```
{
  "data": [
    {
      "approved": false,
      "contactInfo": "alice.wang@example.com",
      "description": "Luxury SUV in excellent condition.",
      "fuelType": "Gasoline",
      "id": 8,
      "imageUrl": "https://example.com/bmw-x5.jpg",
      "location": "New York, NY",
      "make": "BMW",
      "mileage": 15000,
      "model": "X5",
      "price": 42000.0,
      "submittedBy": "alice_wang",
      "transmission": "Automatic",
      "vin": "WBAXX1234BMWX5",
      "year": 2021
    },
    {
      "approved": false,
      "contactInfo": "david.liu@example.com",
      "description": "Fuel-efficient and reliable daily driver.",
      "fuelType": "Hybrid",
      "id": 9,
      "imageUrl": "https://example.com/toyota-corolla.jpg",
      "location": "Chicago, IL",
      "make": "Toyota",
      "mileage": 48000,
      "model": "Corolla",
      "price": 12500.0,
      "submittedBy": "david_liu",
      "transmission": "Automatic",
      "vin": "JTDBR32EX81234567",
      "year": 2018
    },
    {
      "approved": false,
      "contactInfo": "alice.wang@example.com",
      "description": "Spacious and smooth ride, single owner.",
      "fuelType": "Gasoline",
      "id": 10,
      "imageUrl": "https://example.com/honda-accord.jpg",
      "location": "Los Angeles, CA",
      "make": "Honda",
      "mileage": 30000,
      "model": "Accord",
      "price": 18000.0,
      "submittedBy": "alice_wang",
      "transmission": "CVT",
      "vin": "1HGCV1F14LA123456",
      "year": 2020
    },
```



```
    {
      "approved": false,
      "contactInfo": "david.liu@example.com",
      "description": "Electric sedan with Autopilot, like new.",
      "fuelType": "Electric",
      "id": 11,
      "imageUrl": "https://example.com/tesla-model3.jpg",
      "location": "San Francisco, CA",
      "make": "Tesla",
      "mileage": 12000,
      "model": "Model 3",
      "price": 35000.0,
      "submittedBy": "david_liu",
      "transmission": "Automatic",
      "vin": "5YJ3E1EA5JF123456",
      "year": 2022
    }
  ],
  "status": "success"
}
```

3. **Get Users**

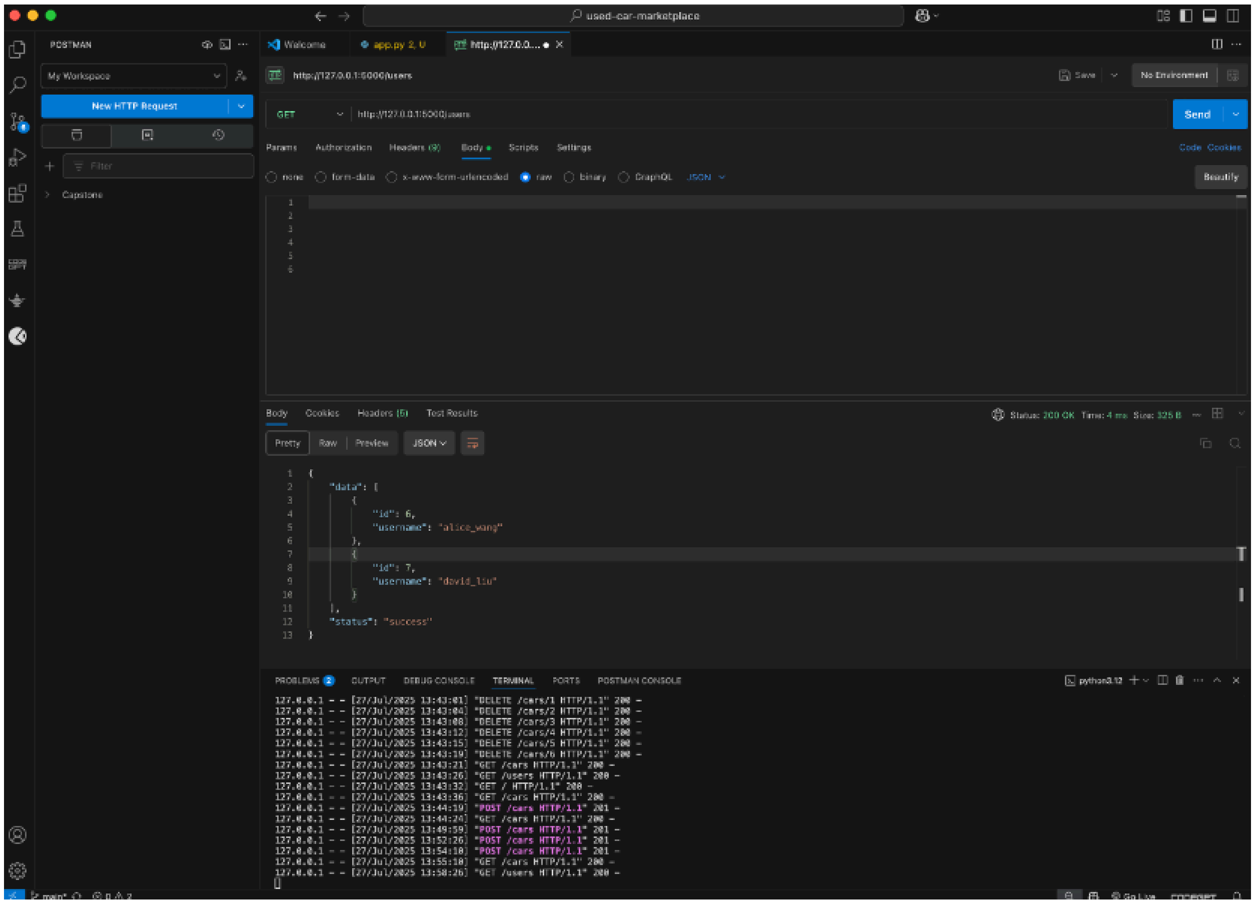
Endpoint: GET/users

Request JSON: N/A

Expected Result:

Status code: 200

Resopnse JSON:


GET http://127.0.0.1:5000/users

Status: 200 OK Time: 4 ms Size: 325 B

```
{
    "data": [
        {
            "id": 6,
            "username": "alice_wang"
        },
        {
            "id": 7,
            "username": "david_liu"
        }
    ],
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:43:01] "DELETE /cars/1 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:04] "DELETE /cars/2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:08] "DELETE /cars/3 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:52:26] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:54:18] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:55:18] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:58:26] "GET /users HTTP/1.1" 200 -
```

4. **Get Cars**

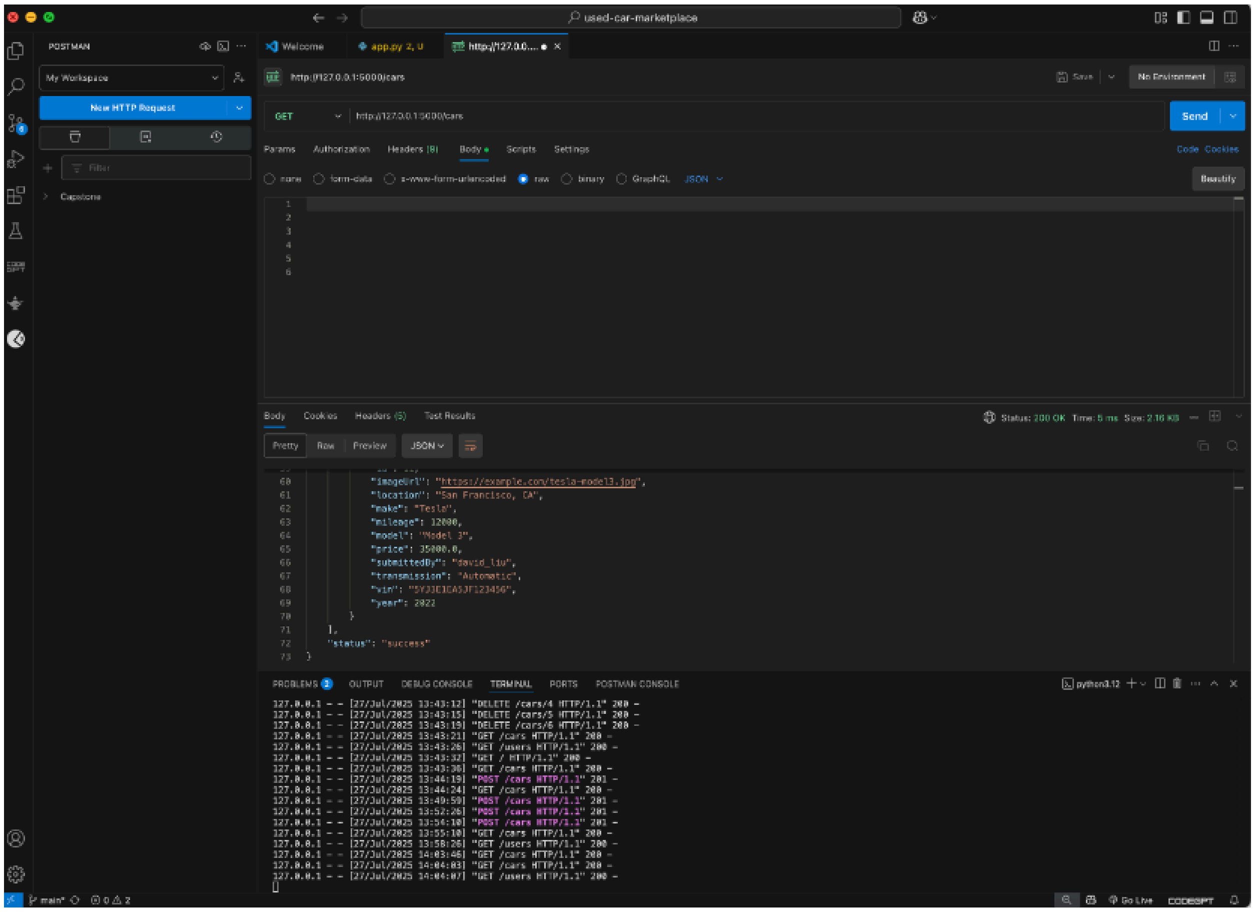
Endpoint: GET/cars

Request JSON: N/A

Expected Result:

Status code: 200

Response JSON:


GET http://127.0.0.1:5000/cars

Status: 200 OK Time: 5 ms Size: 2.16 KB

```
            "imageUrl": "https://example.com/tesla-model3.jpg",
            "location": "San Francisco, CA",
            "make": "Tesla",
            "mileage": 12000,
            "model": "Model 3",
            "price": 35000.0,
            "submittedBy": "david_liu",
            "transmission": "Automatic",
            "vin": "5YJ3E1EA5JF123456",
            "year": 2022
        }
    ],
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:43:12] "DELETE /cars/4 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:15] "DELETE /cars/5 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:19] "DELETE /cars/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:21] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:52:26] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:54:10] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:55:10] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:58:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:03:46] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:03] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:07] "GET /users HTTP/1.1" 200 -
```

used-car-marketplace
POSTMAN
My Workspace
New HTTP Request
Capstone
Welcome
app.py
Params Authorization Headers (8) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Send
Body Cookies Headers (5) Test Results
Pretty Raw Preview JSON
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

5. Put Users

Test Case – Update User Information(ID=6)

Endpoint: PUT/users/1

Request JSON:

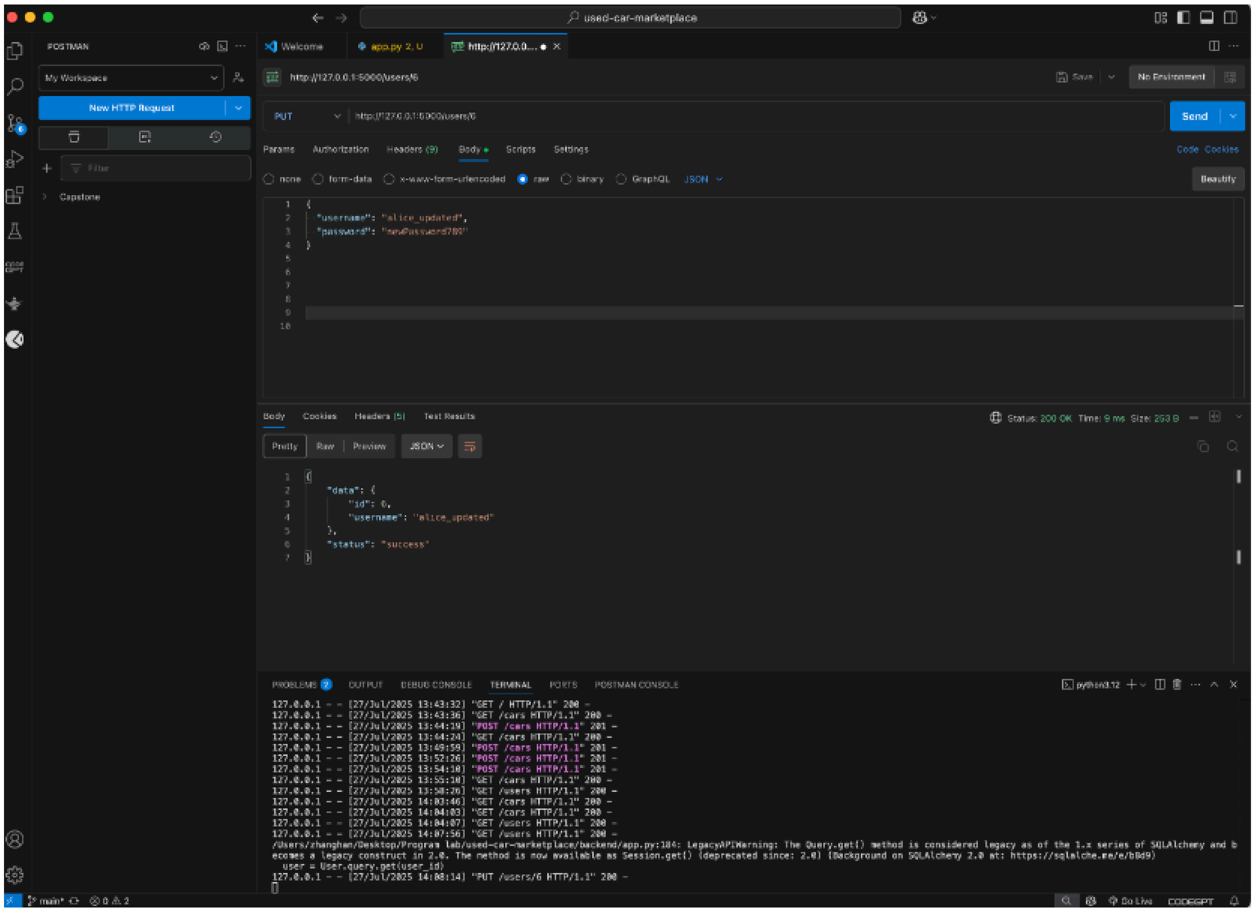
```
{
  "username": "alice_updated",
  "password": "newPassword789"
}
```

Expected Result:

Status Code: 200

Response JSON:

Result: Passed



http://127.0.0.1:5000/users/6

PUT http://127.0.0.1:5000/users/6

```
{
  "username": "alice_updated",
  "password": "newPassword789"
}
```

Status: 200 OK Time: 9 ms Size: 253 B

```
{
    "data": {
        "id": 6,
        "username": "alice_updated"
    },
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:43:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:43:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:44:19] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:44:24] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:49:59] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:52:26] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:54:10] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:55:10] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:58:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:03:46] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:03] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:07] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:07:56] "GET /users HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:184: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 14:08:14] "PUT /users/6 HTTP/1.1" 200 -
```

6. **Put Cars**

Test Case – Update Car Info (ID = 8)

Endpoint: PUT /cars/8

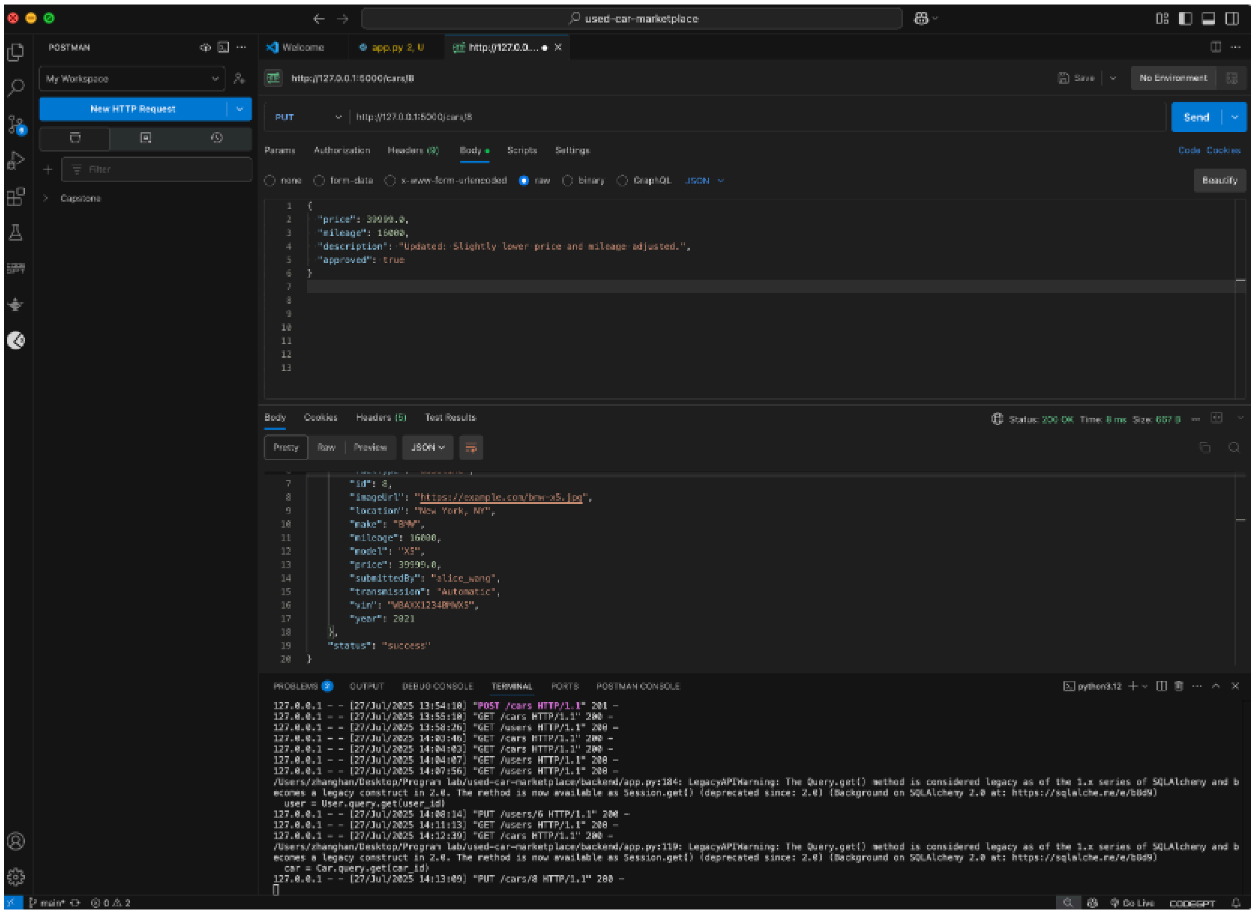
Request JSON:

```
{
  "price": 39999.0,
  "mileage": 16000,
  "description": "Updated: Slightly lower price and mileage adjusted.",
  "approved": true
}
```

Expected Result:

Status Code: 200

Response JSON:



PUT http://127.0.0.1:5000/cars/8

```
{
  "price": 39999.0,
  "mileage": 16000,
  "description": "Updated: Slightly lower price and mileage adjusted.",
  "approved": true
}
```

Status: 200 OK

```
        "id": 8,
        "imageUrl": "https://example.com/bmw-x5.jpg",
        "location": "New York, NY",
        "make": "BMW",
        "mileage": 16000,
        "model": "X5",
        "price": 39999.0,
        "submittedBy": "alice_wang",
        "transmission": "Automatic",
        "vin": "WBAXX12348MWX5",
        "year": 2021
    },
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 13:54:10] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 13:55:10] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 13:58:26] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:03:46] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:03] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:04:07] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:07:56] "GET /users HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:184: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 14:08:14] "PUT /users/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:11:13] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:12:39] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:119: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 14:13:09] "PUT /cars/8 HTTP/1.1" 200 -
```

7. Delete Users and Cars

Test Case 1 – Delete Car (ID = 12)

Endpoint: DELETE /cars/12

Request JSON: N/A

Expected Result:

Result: Passes

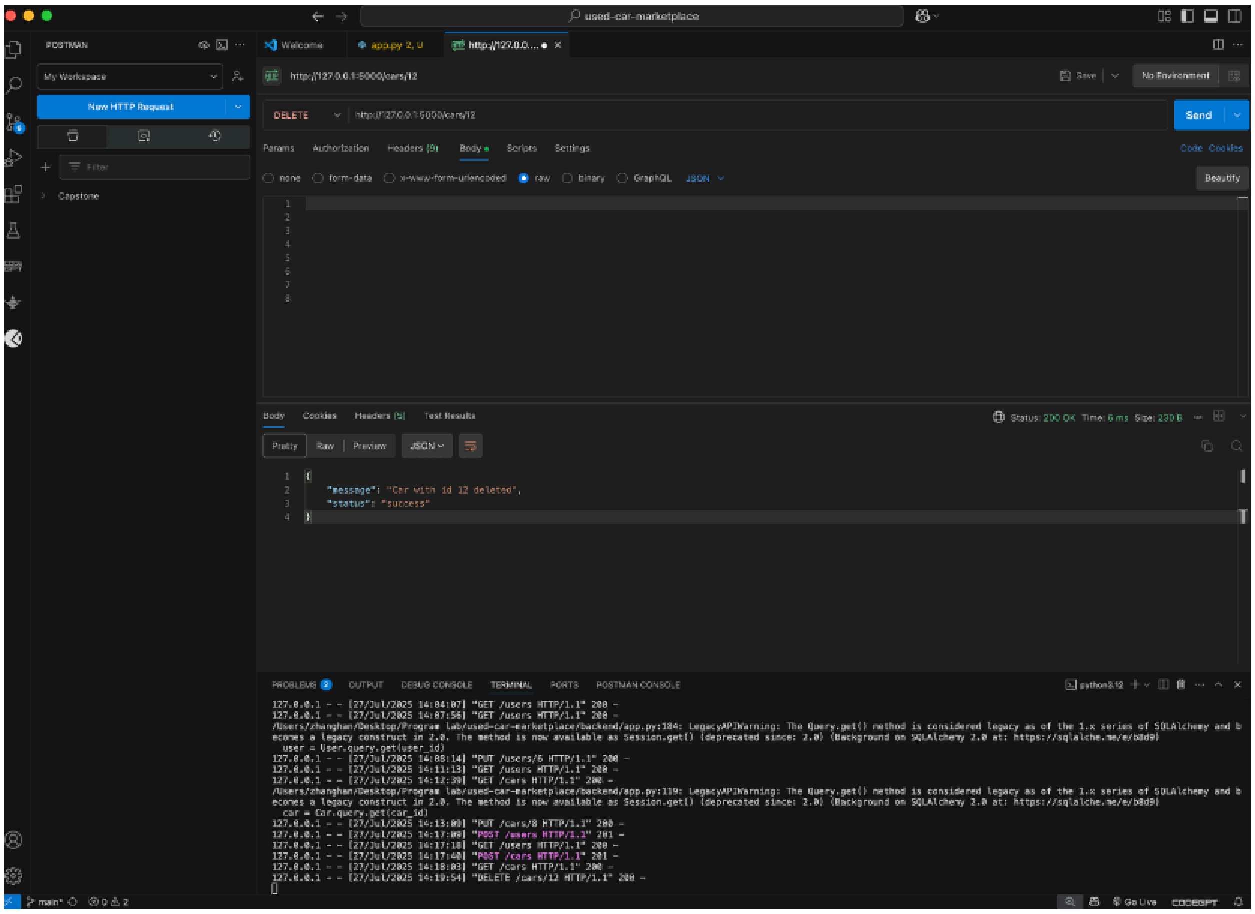
Test Case 2 – Delete User (ID = 8)

Endpoint: DELETE /users/8

Request JSON: N/A

Expected Result:

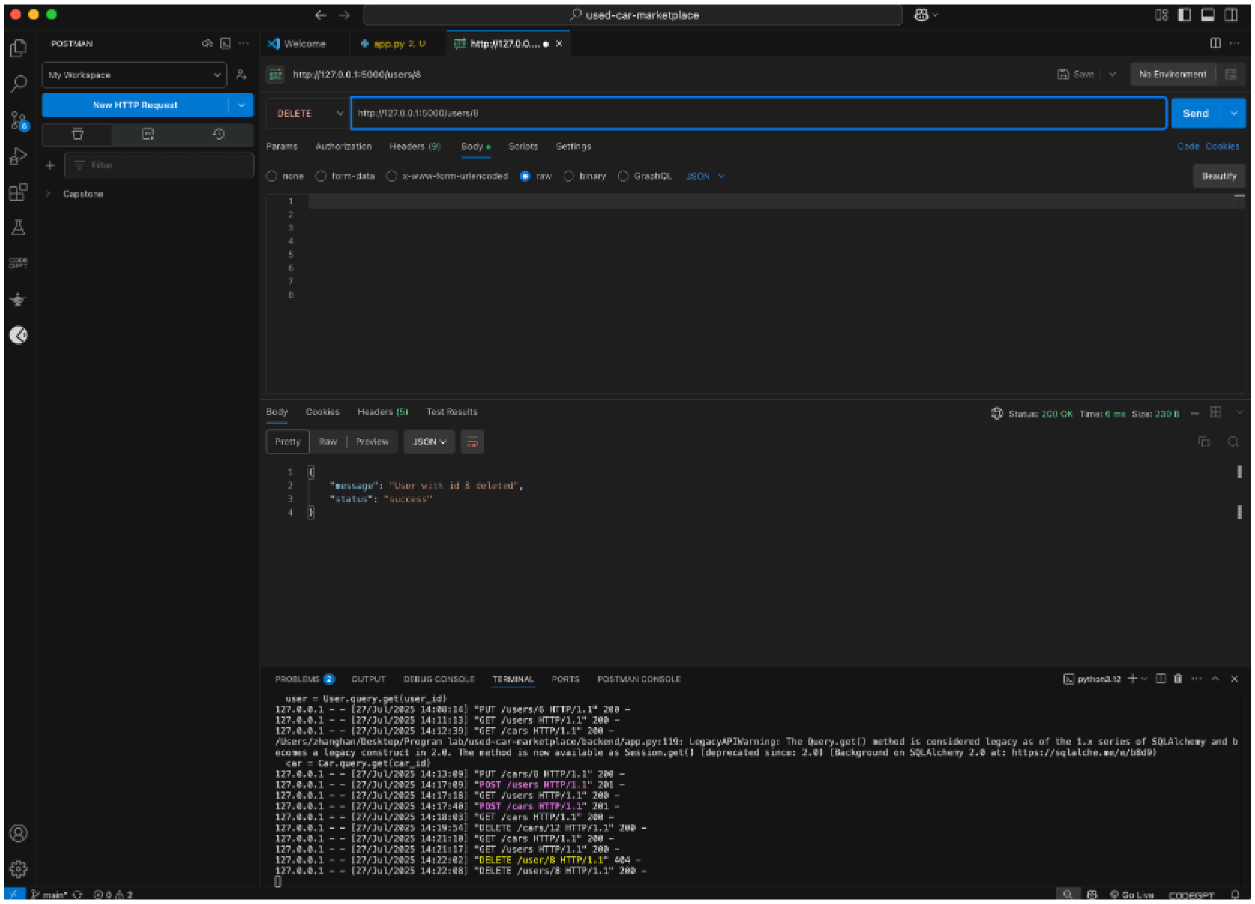
Result: Passed



```
DELETE    http://127.0.0.1:5000/cars/12
```

```
{
    "message": "Car with id 12 deleted",
    "status": "success"
}
```

```
127.0.0.1 - - [27/Jul/2025 14:04:07] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:07:56] "GET /users HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:184: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 14:08:14] "PUT /users/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:11:13] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:12:39] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:119: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 14:13:00] "PUT /cars/8 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:17:09] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 14:17:18] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:17:40] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 14:18:03] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:19:54] "DELETE /cars/12 HTTP/1.1" 200 -
```



DELETE http://127.0.0.1:5000/users/8

Status: 200 OK Time: 6 ms Size: 230 B

```
{
    "message": "User with id 8 deleted",
    "status": "success"
}
```

```
  user = User.query.get(user_id)
127.0.0.1 - - [27/Jul/2025 14:08:14] "PUT /users/6 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:11:13] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:12:39] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:119: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 14:13:09] "PUT /cars/8 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:17:09] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 14:17:18] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:17:40] "POST /cars HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 14:18:03] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:19:54] "DELETE /cars/12 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:21:10] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:21:17] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 14:22:02] "DELETE /user/8 HTTP/1.1" 404 -
127.0.0.1 - - [27/Jul/2025 14:22:08] "DELETE /users/8 HTTP/1.1" 200 -
```

8. **Add Admin user**

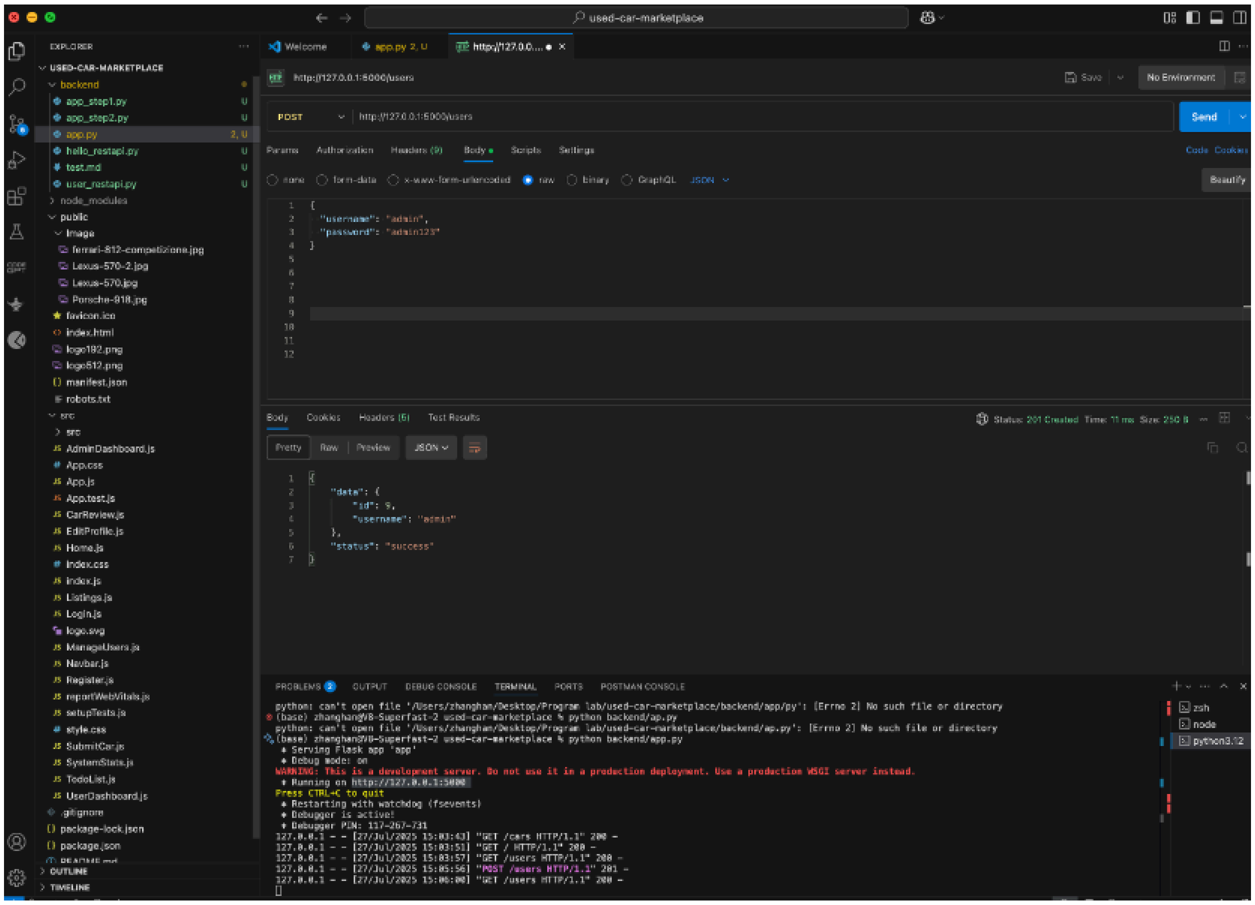
Purpose: Ensure a new user (admin) can be successfully created.

Endpoint: POST /admin/users

Input JSON:

```
{
  "username": "admin",
  "password": "admin123"
}
```

Expected Response:



POST http://127.0.0.1:5000/users

Body (raw, JSON):

```
{
  "username": "admin",
  "password": "admin123"
}
```

Status: 201 Created Time: 11 ms Size: 250 B

```
{
    "data": {
        "id": 9,
        "username": "admin"
    },
    "status": "success"
}
```

```
python: can't open file '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app/py': [Errno 2] No such file or directory
(base) zhanghan@VB-Superfast-2 used-car-marketplace % python backend/ap.py
python: can't open file '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/ap.py': [Errno 2] No such file or directory
(base) zhanghan@VB-Superfast-2 used-car-marketplace % python backend/app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 15:03:43] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:57] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:05:56] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 15:06:00] "GET /users HTTP/1.1" 200 -
```

9. **Endpoint: POST/login**

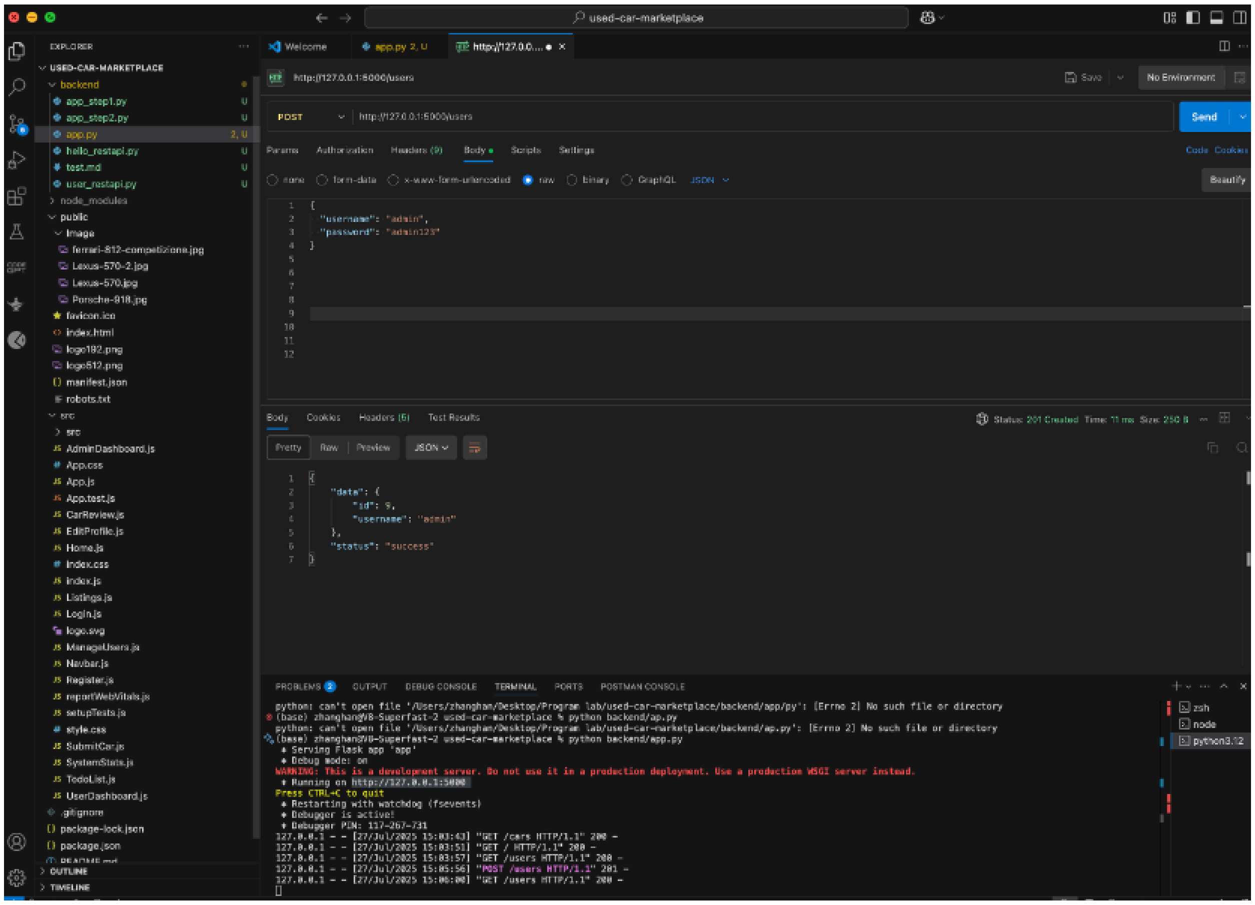
Description: Log in and check if the user is an administrator

Method: POST

Request JSON:

```
{
  "username": "admin",
  "password": "admin123"
}
```

Method: POST


POST http://127.0.0.1:5000/users

```
{
  "username": "admin",
  "password": "admin123"
}
```

Status: 201 Created Time: 11 ms Size: 250 B

```
{
    "data": {
        "id": 9,
        "username": "admin"
    },
    "status": "success"
}
```

```
python: can't open file '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app/py': [Errno 2] No such file or directory
(base) zhanghan@VB-Superfast-2 used-car-marketplace % python backend/ap.py
python: can't open file '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/ap.py': [Errno 2] No such file or directory
(base) zhanghan@VB-Superfast-2 used-car-marketplace % python backend/app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 15:03:43] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:57] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:05:56] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 15:06:00] "GET /users HTTP/1.1" 200 -
```

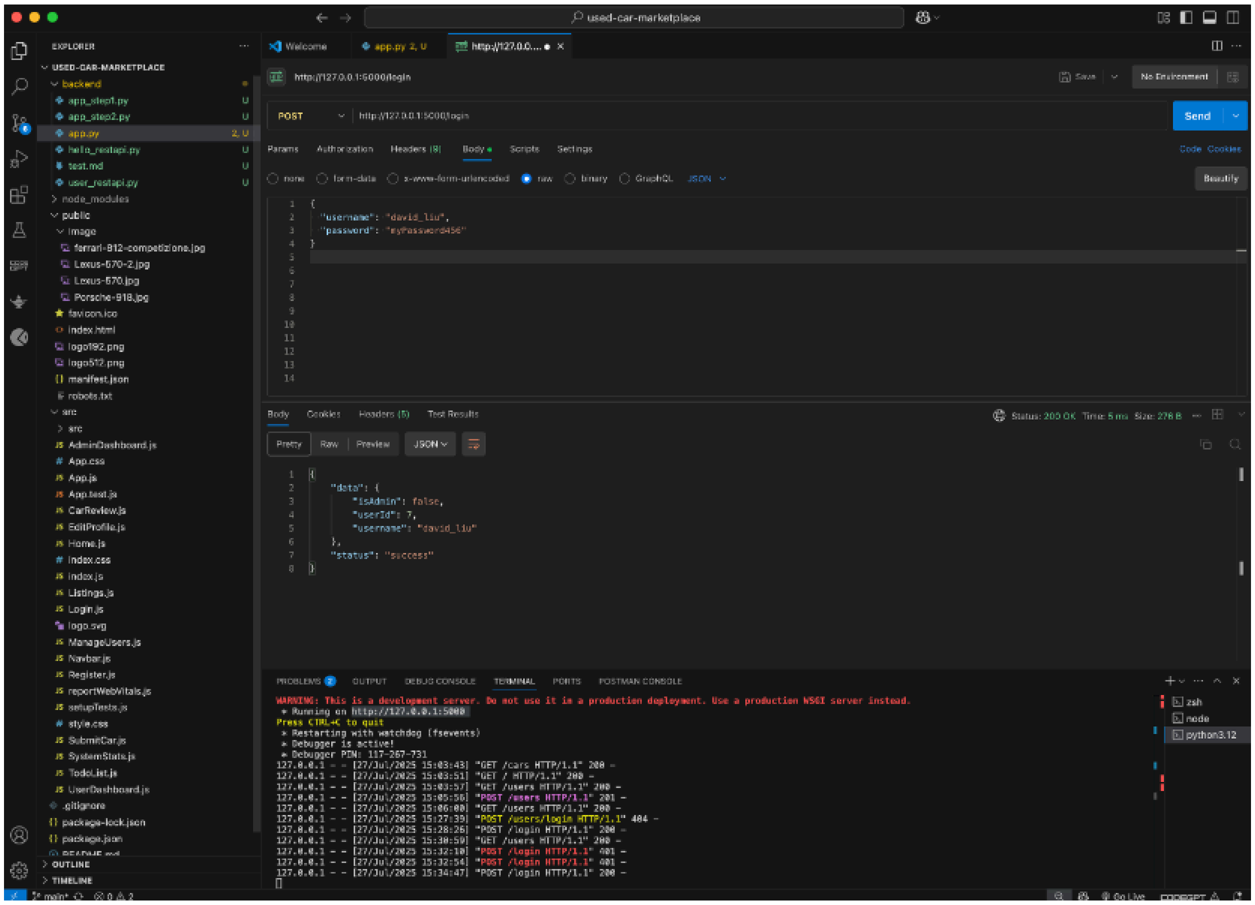
Request JSON:

{

"username": "david_liu",

"password": "myPassword456"

}


POST http://127.0.0.1:5000/login

```
{
  "username": "david_liu",
  "password": "myPassword456"
}
```

Status: 200 OK Time: 5 ms Size: 276 B

```
{
    "data": {
        "isAdmin": false,
        "userId": 7,
        "username": "david_liu"
    },
    "status": "success"
}
```

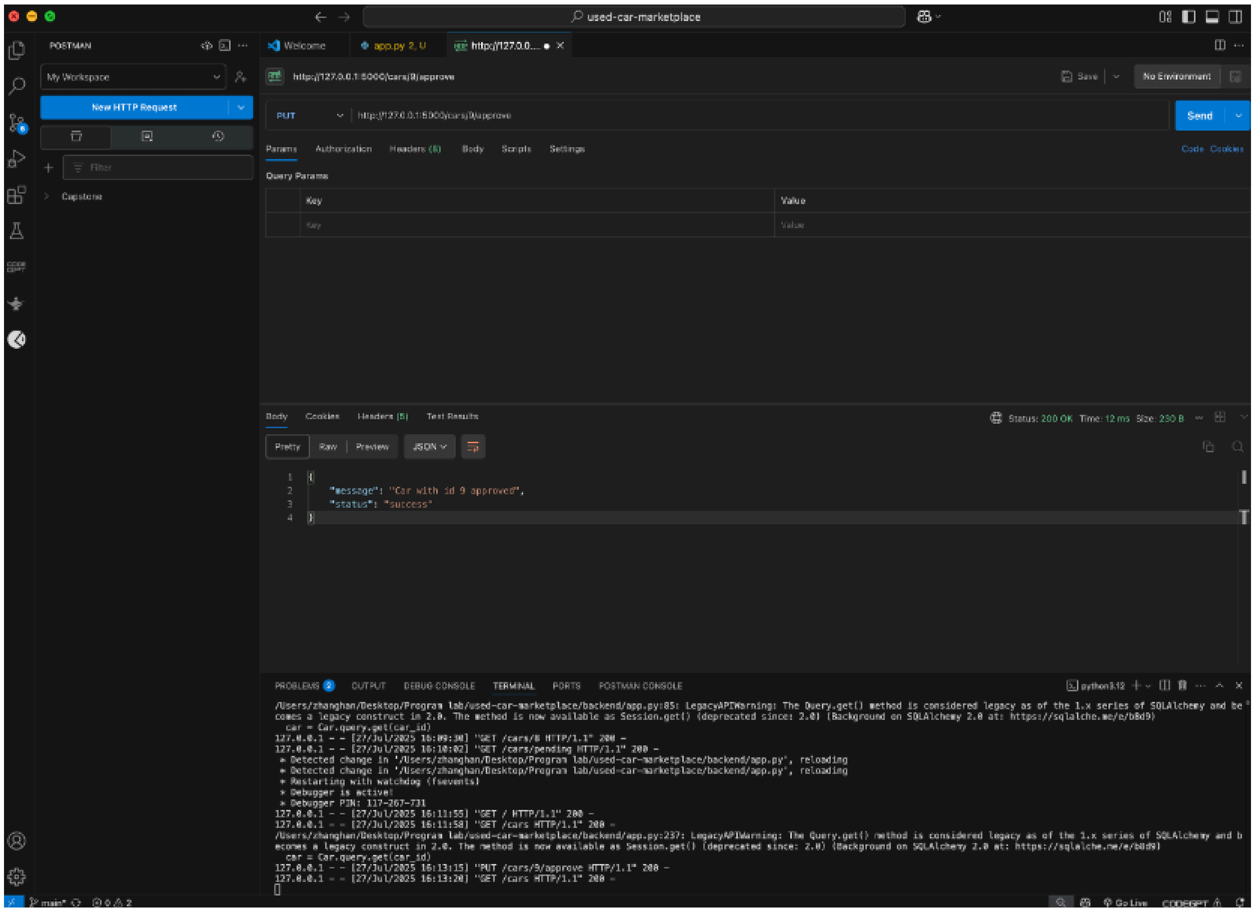
```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 15:03:43] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:03:57] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:05:56] "POST /users HTTP/1.1" 201 -
127.0.0.1 - - [27/Jul/2025 15:06:00] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:27:39] "POST /users/login HTTP/1.1" 404 -
127.0.0.1 - - [27/Jul/2025 15:28:26] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:30:59] "GET /users HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 15:32:10] "POST /login HTTP/1.1" 401 -
127.0.0.1 - - [27/Jul/2025 15:32:54] "POST /login HTTP/1.1" 401 -
127.0.0.1 - - [27/Jul/2025 15:34:47] "POST /login HTTP/1.1" 200 -
```

10. Endpoint: PUT /cars/9/approve

Description: Approve a specific car listing by its ID. Used by administrators to mark a vehicle as approved.

Method: PUT

Response:



http://127.0.0.1:5000/cars/9/approve

PUT http://127.0.0.1:5000/cars/9/approve

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results — Status: 200 OK Time: 12 ms Size: 230 B

Pretty | Raw | Preview | JSON

```
{
    "message": "Car with id 9 approved",
    "status": "success"
}
```

```
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:85: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and be
comes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 16:09:30] "GET /cars/8 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:10:02] "GET /cars/pending HTTP/1.1" 200 -
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:11:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:11:58] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:237: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 16:13:15] "PUT /cars/9/approve HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:13:20] "GET /cars HTTP/1.1" 200 -
```

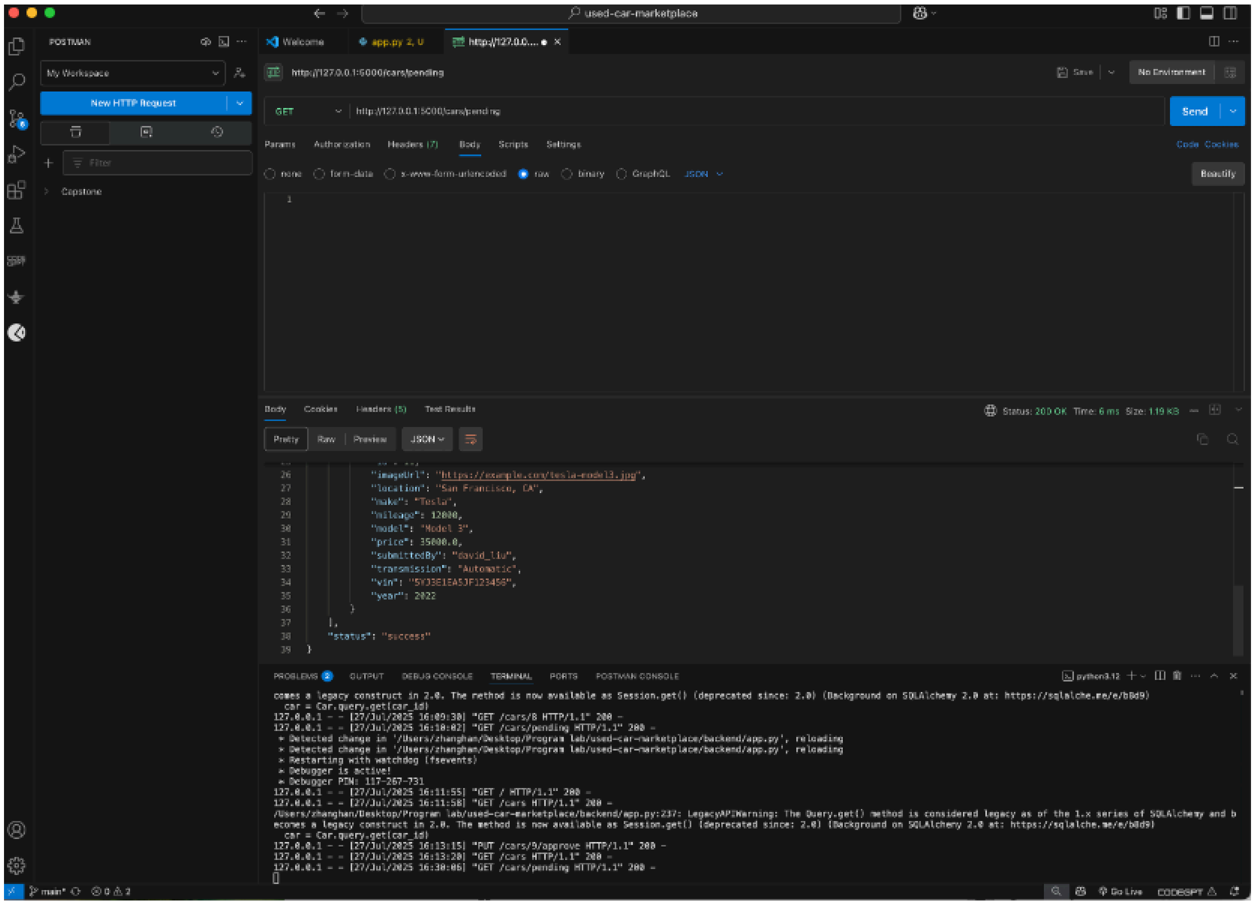
11.Get Pending Car Listings

Description: Retrieve all car listings that have not been approved (approved = false). This endpoint is used by administrators to review cars awaiting approval.

Endpoint: GET /cars/pending

Request Body: N/A

Response:



GET http://127.0.0.1:5000/cars/pending

Status: 200 OK Time: 6 ms Size: 1.19 KB

```
            "imageUrl": "https://example.com/tesla-model3.jpg",
            "location": "San Francisco, CA",
            "make": "Tesla",
            "mileage": 12000,
            "model": "Model 3",
            "price": 35000.0,
            "submittedBy": "david_liu",
            "transmission": "Automatic",
            "vin": "5YJ3E1EA5JF123456",
            "year": 2022
        }
    ],
    "status": "success"
}
```

```
comes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 16:09:30] "GET /cars/8 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:10:02] "GET /cars/pending HTTP/1.1" 200 -
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:11:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:11:58] "GET /cars HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:237: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and b
ecomes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 16:13:15] "PUT /cars/9/approve HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:13:20] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:30:06] "GET /cars/pending HTTP/1.1" 200 -
```

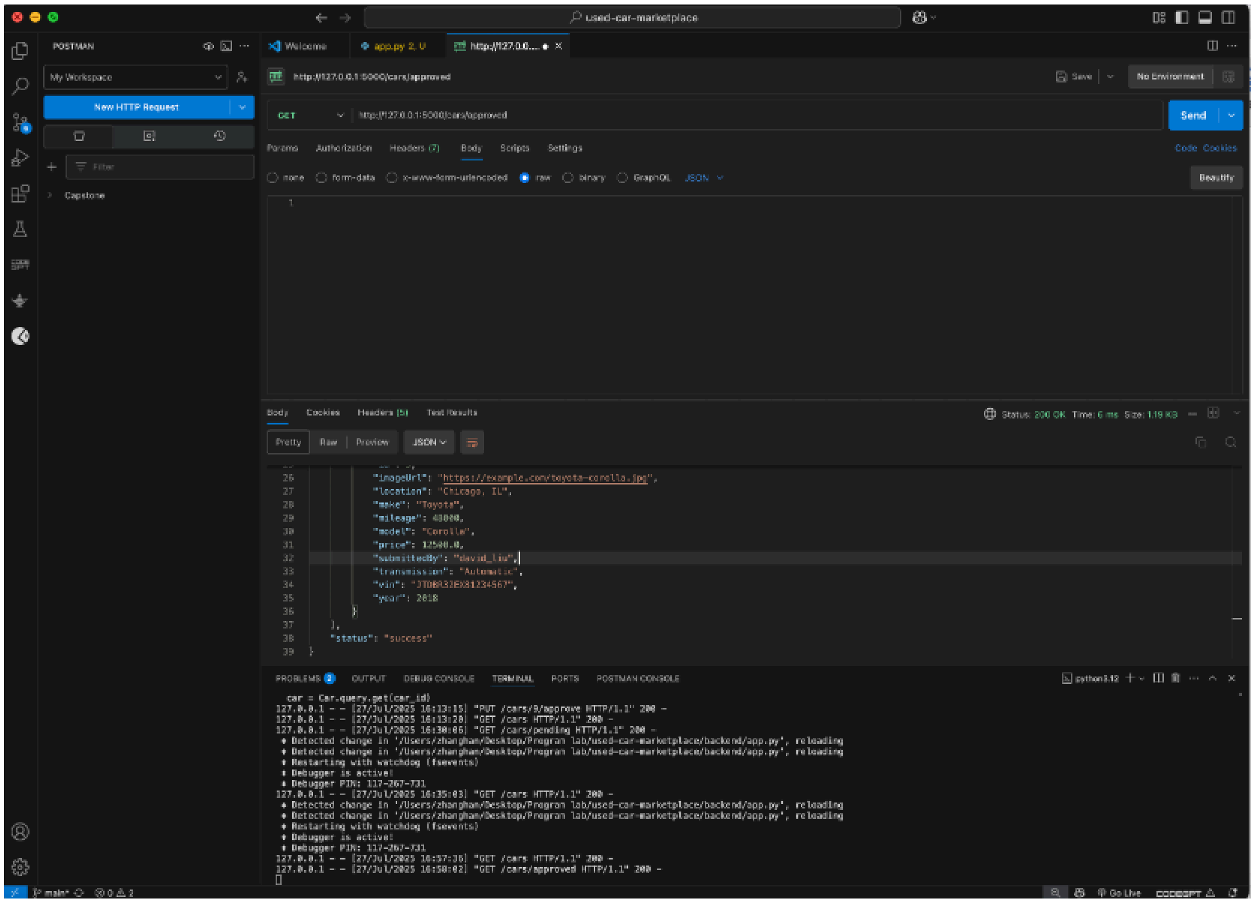
12. Get All Approved Cars

Description: Retrieve all car listings that have been approved. Used on the public listings page.

Endpoint: GET /cars/approved

Request Body: N/A

Response Example:



GET http://127.0.0.1:5000/cars/approved

```
            "imageUrl": "https://example.com/toyota-corolla.jpg",
            "location": "Chicago, IL",
            "make": "Toyota",
            "mileage": 40000,
            "model": "Corolla",
            "price": 12500.0,
            "submittedBy": "david_liu",
            "transmission": "Automatic",
            "vin": "JTDBR32EX81234567",
            "year": 2018
        }
    ],
    "status": "success"
}
```

```
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 16:13:15] "PUT /cars/9/approve HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:13:20] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:30:06] "GET /cars/pending HTTP/1.1" 200 -
 * Detected change in '/Users/zhangham/Desktop/Progran lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhangham/Desktop/Progran lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:35:03] "GET /cars HTTP/1.1" 200 -
 * Detected change in '/Users/zhangham/Desktop/Progran lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhangham/Desktop/Progran lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:57:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:58:02] "GET /cars/approved HTTP/1.1" 200 -
```

13. Get Car Detail by ID

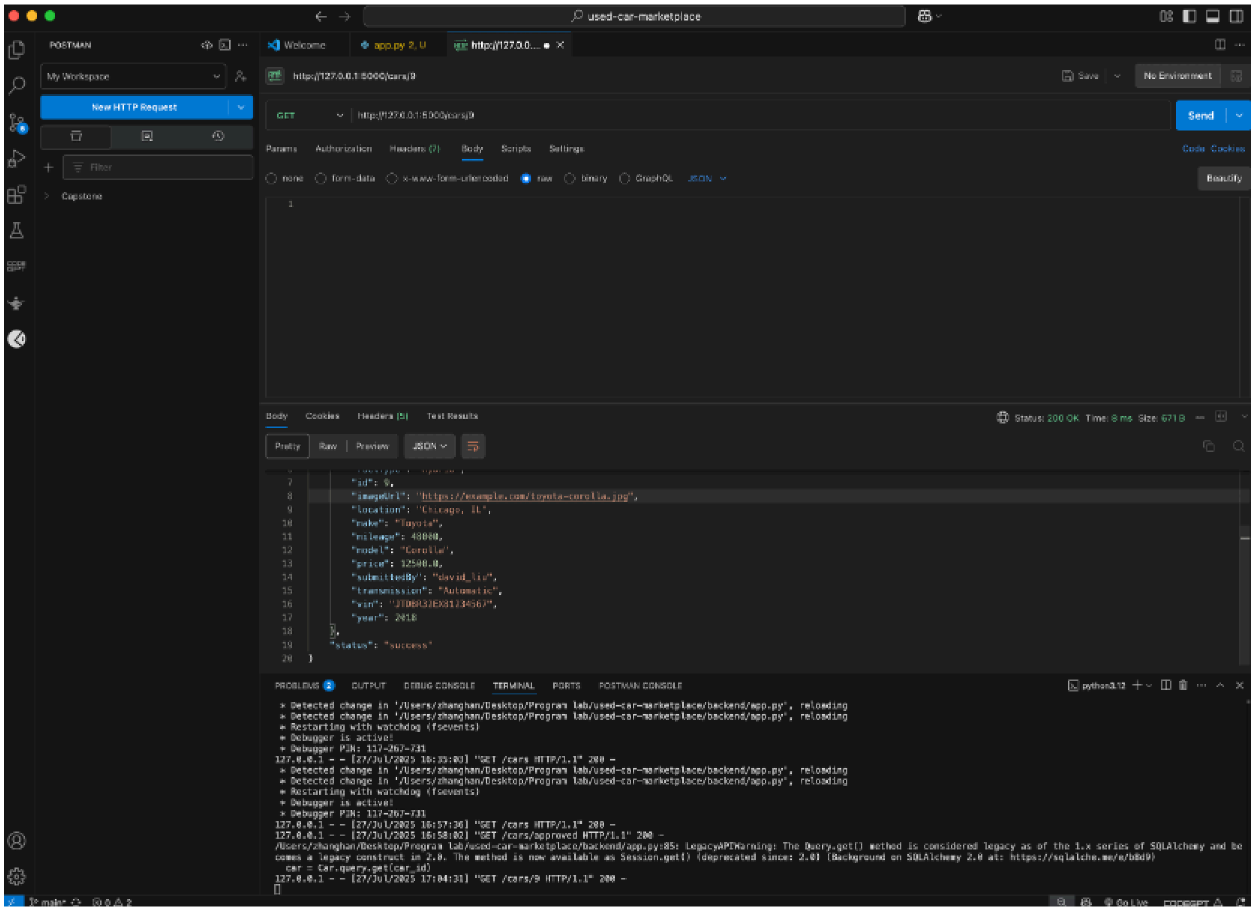
Description: Retrieve detailed information for a specific car by ID. Used when clicking a listing card to view details.

Endpoint: GET /cars/<id>

Request Body: N/A

Example: GET /cars/9

Response (Success):



GET http://127.0.0.1:5000/cars/9

Status: 200 OK Time: 8 ms Size: 671 B

```
            "id": 9,
            "imageUrl": "https://example.com/toyota-corolla.jpg",
            "location": "Chicago, IL",
            "make": "Toyota",
            "mileage": 48000,
            "model": "Corolla",
            "price": 12500.0,
            "submittedBy": "david_liu",
            "transmission": "Automatic",
            "vin": "JTDBR32EX81234567",
            "year": 2018
        },
        "status": "success"
    }
```

```
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:35:03] "GET /cars HTTP/1.1" 200 -
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Detected change in '/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py', reloading
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 117-267-731
127.0.0.1 - - [27/Jul/2025 16:57:36] "GET /cars HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2025 16:58:02] "GET /cars/approved HTTP/1.1" 200 -
/Users/zhanghan/Desktop/Program lab/used-car-marketplace/backend/app.py:85: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and be
comes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  car = Car.query.get(car_id)
127.0.0.1 - - [27/Jul/2025 17:04:31] "GET /cars/9 HTTP/1.1" 200 -
```

The uploaded API test document provides full coverage of backend endpoints with detailed requests, expected vs. actual results, all marked as Passed. Representative cases include:

User endpoints

POST /users — Create new users (alice_wang, david_liu) → Passed

PUT /users/{id} — Update username/password → Passed

DELETE /users/{id} — Delete a user → Passed

GET /users — Retrieve users → Passed

Car endpoints

POST /cars — Add BMW X5, Toyota Corolla, Honda Accord, Tesla Model 3 → Passe

PUT /cars/{id} — Update car info (price, mileage, approval) → Passed

DELETE /cars/{id} — Remove listing → Passed

GET /cars — Retrieve all cars → Passed

GET /cars/{id} — Retrieve car details → Passed

Admin endpoints

POST /admin/users — Create admin account → Passed

POST /login — Authenticate users and admin → Passed

PUT /cars/{id}/approve — Approve pending car → Passed

GET /cars/pending — Fetch pending listings → Passed

GET /cars/approved — Fetch approved listings → Passed

All integration tests produced the expected outputs with HTTP 200/201 responses.

1.4.3 End-to-End Flows

These flows demonstrate complete user-to-admin workflows. Each step is validated by API test cases in the uploaded file.

Flow A — User submission to approval

User registers (POST /users).

User submits a car (POST /cars).

Admin approves it (PUT /cars/{id}/approve).

Car appears in approved list (GET /cars/approved).

→ Result: Passed

Flow B — Admin updates listing

Admin updates car info (PUT /cars/{id}).

Updated details visible in subsequent GET /cars/GET /cars/{id}.

→ Result: Passed

Flow C — Deletion

Admin deletes a car (DELETE /cars/{id}).

Car no longer appears in GET /cars/approved.

Admin deletes a user (DELETE /users/{id}).

Deleted user cannot log in.

→ Result: Passed

1.4.4 Manual Test Cases

These tests simulate real user interactions through the frontend. Expected outcomes are based on the documented UI features.

ID	User Action	Expected Result	Actual Result	Status
MT-1	Open Home Page	Welcome message and navigation buttons visible	Displayed correctly	Passed
MT-2	Register New User	User created, success message/redirect shown	Works as expected	Passed
MT-3	Login with User Account	User dashboard accessible, role = user	Login successful	Passed
MT-4	Submit Car Form	Car submission accepted → appears in Admin pending	Works as expected	Passed
MT-5	Login with Admin	Admin dashboard visible, can view pending cars	Works as expected	Passed
MT-6	Approve Pending Car	Selected car moves from Pending to Approved	Works as expected	Passed
MT-7	Browse Approved Listings	Approved cars display with correct details (year, price, mileage, VIN, etc.)	Works as expected	Passed
MT-8	Admin Manage Users	Admin deletes a user; deleted user cannot log in	Works as expected	Passed
MT-9	System Statistics Page	Counts of users/cars/pending/approved update correctly	Works as expected	Passed

1.4.5 Post-Deployment Smoke Tests

These are minimal validation steps to confirm system health after deployment.

ID	Test Step	Expected Result	Actual Result	Status
ST-1	Call GET /cars/approved	HTTP 200, returns JSON array (can be empty)	Works as expected	Passed
ST-2	Admin login → GET /cars/pending	HTTP 200, returns list of pending cars	Works as expected	Passed
ST-3	Load Home/Browse/Login pages	Pages load without errors; no 5xx or CORS issues	Works as expected	Passed
ST-4	Run SELECT 1 on PostgreSQL	Query succeeds, confirms DB availability	Works as expected	Passed

2. System Setup Instructions

The Used Car Marketplace system consists of three major components:

- Frontend: React SPA (Azure Web App, Node.js 22 LTS)
- Backend: Flask REST API (Azure Web App, Python 3.10 + SQLAlchemy)
- Database: PostgreSQL (Azure PostgreSQL Flexible Server in cloud or local PostgreSQL instance)

The following instructions describe how to set up and run the system from scratch.

2.1 Prerequisites

Operating System

Windows 10/11, macOS, or Ubuntu 22.04+ (Linux recommended for deployment)

Required Tools

Git

Node.js 22 LTS

(includes npm)

Python 3.10+

PostgreSQL 14+

with psql CLI

Azure CLI

(optional for cloud deployment)

Postman

or curl (optional for API tests)

Cloud Resources

Azure Web App (Linux) for Frontend

Azure Web App (Linux) for Backend

Azure Database for PostgreSQL – Flexible Server

2.2 Repository Layout

```
/frontend       # React SPA
  package.json
  .env.example
/backend        # Flask REST API
  app.py
  requirements.txt
  .env.example
  models/
/docs           # documentation
```

2.3 Environment Variables

Backend (/backend/.env)

```
# Flask
FLASK_ENV=development
SECRET_KEY=replace_with_random_secret
```

```
# Database connection

DATABASE_URL=postgresql://<user>:<password>@<host>:5432/usedcars

# For Azure PG add ?sslmode=require

# # DATABASE_URL=postgresql://<user>%40<server>:<password>@<server>.postgres.database.azure.com:5432/usedcars?sslmode=require


# CORS allowed origins (frontend URLs)

CORS_ALLOWED_ORIGINS=http://localhost:3000,https://<frontend-app>.azurewebsites.net
```

Frontend (/frontend/.env)

```
REACT_APP_API_BASE_URL=http://localhost:5000

# For cloud:

# REACT_APP_API_BASE_URL=https://<backend-app>.azurewebsites.net
```

Do not commit .env files. On Azure, configure these under App Service → Configuration → Application settings.

2.4 Database Setup (PostgreSQL)

Option A — Local PostgreSQL

Create role and database:

```
-- Start psql as superuser
psql -U postgres

-- Create an application role and database
CREATE ROLE usedcars_user WITH LOGIN PASSWORD 'changeme';
CREATE DATABASE usedcars OWNER usedcars_user;
GRANT ALL PRIVILEGES ON DATABASE usedcars TO usedcars_user;

-- IMPORTANT: connect to the target database before creating tables
\c usedcars

-- (Recommended) switch to the app role so tables are owned by it
-- \c usedcars usedcars_user
```

Create schema:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role VARCHAR(20) DEFAULT 'user'
);

CREATE TABLE cars (
  id SERIAL PRIMARY KEY,
  make VARCHAR(50),
  model VARCHAR(50),
  year INT,
  price NUMERIC,
  mileage INT,
  fuelType VARCHAR(50),
  transmission VARCHAR(50),
  vin VARCHAR(50) UNIQUE,
  location VARCHAR(100),
  contactInfo VARCHAR(100),
  description TEXT,
  imageUrl TEXT,
  submittedBy VARCHAR(50),
  approved BOOLEAN DEFAULT FALSE
);
```

Design note (v1.0)

The field submittedBy stores the submitter's username as plain text. There is no foreign key to users in v1.0.

Deleting a user does not cascade to car listings; this is a business-rule decision, not a DB constraint.

If needed in the future, this can be migrated to a user_id foreign key without affecting the current version.

If you created the tables as postgres (not usedcars_user) — grant privileges to the app role:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO usedcars_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO usedcars_user;
```

Validation:

```
SELECT 1;                              -- connectivity
\dt                                    -- should list: users, cars
SELECT tablename, tableowner           -- optional: check owners (should be usedcars_user)
FROM pg_tables WHERE schemaname='public';
```

```
-- optional: quick sanity checks
SELECT COUNT(*) FROM users;
SELECT COUNT(*) FROM cars;
```

Option B — Azure PostgreSQL Flexible Server

Create server and usedcars database in Azure Portal.
Enable “Allow access to Azure services” or whitelist outbound IPs of your Web Apps.
SSL required → ensure ?sslmode=require in DATABASE_URL.
Validate with Azure Cloud Shell:

```
psql "host=<server>.postgres.database.azure.com port=5432 dbname=usedcars
user=<admin>@<server> password=<pw> sslmode=require" -c "SELECT 1;"
```

Seed Admin Account

To avoid being locked out, seed admin via API:

```
curl -X POST "$API_BASE/admin/users" \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"Admin#123"}'
```

2.5 Backend Setup (Flask API)

Clone repository:

```
git clone <repo-url>
cd backend
```

Create virtual environment:

```
python -m venv venv
source venv/bin/activate   # macOS/Linux
venv\Scripts\activate      # Windows
```

Install dependencies:

```
 pip install -r requirements.txt
```

(Flask, Flask-CORS, SQLAlchemy, psycopg2-binary, gunicorn, python-dotenv)

Configure environment variables in .env (see 2.3).

Run locally:

```
export FLASK_APP=app.py   # Windows: set FLASK_APP=app.py
flask run
```

→ Backend runs at http://127.0.0.1:5000.

Validate:

```
curl http://localhost:5000/cars
```

Should return 200 OK with JSON array.

2.6 Frontend Setup (React)

Clone repository:

```
git clone <repo-url>
cd frontend
```

Install dependencies:

```
npm install
```

Configure .env (see 2.3).

Run locally:

```
npm start
```

→ Frontend runs at http://localhost:3000.

Build for production:

```
npm run build
```

Output: /frontend/build.

2.7 Deployment on Azure

Backend (Python 3.10)

1.Create Azure Web App (Linux, runtime = Python 3.10).

2.Configure App Settings:

- DATABASE_URL=…
- SECRET_KEY=…
- CORS_ALLOWED_ORIGINS=https://<frontend>.azurewebsites.net

3. Startup Command:

```
gunicorn app:app --bind=0.0.0.0:${PORT}
```

4.Deploy via GitHub Actions or Zip deploy.

5.Validate:

```
curl https://<backend-app>.azurewebsites.net/cars/approved
```

Frontend (Node.js 22)

1.Create Azure Web App (Linux, runtime = Node 22).

2.Configure App Settings:

```
REACT_APP_API_BASE_URL=https://<backend>.azurewebsites.net
```

3.Deploy build output (/build). Use GitHub Actions or zip upload.

4. Startup Command:

```
npx serve -s build -l ${PORT}
```

5.Validate: visit https://<frontend-app>.azurewebsites.net.

2.8 Validation (System Level)

1. Frontend home page loads without errors.
2. Register user → login works.
3. Submit car → appears in Admin pending.
4. Admin approves car → appears in Browse listings.
5. API endpoint GET /cars/approved returns 200.
6. Database SELECT COUNT(*) FROM cars WHERE approved=true; matches UI count.

2.9 Logs and Monitoring

- Backend logs: Azure App Service → Monitoring → Log stream
- Frontend logs: Browser console + App Service access logs
- Database logs: Azure portal → Server logs and metrics

2.10 Common Pitfalls & Fixes

- CORS error → Add frontend domain to CORS_ALLOWED_ORIGINS.
- 500 on DB calls → Wrong DATABASE_URL or missing ?sslmode=require.
- Frontend hitting wrong API → Update REACT_APP_API_BASE_URL and rebuild.
- Backend not starting on Azure → Missing startup command (gunicorn app:app).

2.11 Quick Test Data (for validation)

```
API_BASE=http://localhost:5000

# Create admin
curl -X POST $API_BASE/admin/users \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"Admin#123"}'

# Register user
curl -X POST $API_BASE/users \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"Pw#12345"}'

# Submit car
curl -X POST $API_BASE/cars \
  -H "Content-Type: application/json" \
  -d '{"make":"Toyota","model":"Corolla","year":2020,"price":12000,
       "mileage":30000,"fuelType":"Gasoline","transmission":"Automatic",
       "vin":"JTDBR32E720000001","location":"Seattle, WA",
       "contactInfo":"alice@example.com","description":"Clean car",
       "imageUrl":"https://picsum.photos/640","submittedBy":"alice"}'

# Approve car
curl -X PUT $API_BASE/cars/1/approve
```

3. Issue Diagnosis, Research, Resolution, and Sharing

This section records major issues encountered during the development and deployment of the Used Car Marketplace system, along with diagnosis, research process, and resolutions.

Issue 1 — Database Connection Error on Azure

- Description: Backend API returned 500 Internal Server Error after deployment. Expected behavior was valid JSON response.
- Environment: Azure Web App (Python 3.10), Azure PostgreSQL Flexible Server.
- Steps to Reproduce: Deploy backend without sslmode=require in the connection string; call GET /cars.
- Diagnosis: Azure PostgreSQL requires SSL connections by default.
- Research Process: Azure PostgreSQL documentation, StackOverflow answers on psycopg2 SSL.
- Resolution: Updated DATABASE_URL to include ?sslmode=require. Restarted backend.
- Outcome Verification: GET /cars/approved returned 200 OK with JSON array.

Issue 2 — CORS Error Between Frontend and Backend

- Description: Browser console showed CORS errors when frontend tried to fetch backend APIs.
- Environment: Frontend: React (Azure Web App), Backend: Flask (Azure Web App).
- Steps to Reproduce: Deploy frontend and backend separately; load Browse page; check DevTools console.
- Diagnosis: Flask-CORS configuration did not include the deployed frontend domain.
- Research Process: Flask-CORS documentation, Azure App Service configuration docs.
- Resolution: Added environment variable CORS_ALLOWED_ORIGINS=https://frontend.azurewebsites.net and updated backend CORS configuration.

- Outcome Verification: Frontend successfully fetched data; no CORS errors in console.

Issue 3 — SQLAlchemy 2.0 Deprecation Warning

- Description: Logs displayed: LegacyAPIWarning: The Query.get() method is considered legacy….
- Environment: Local and Azure backend with SQLAlchemy 2.0.
- Steps to Reproduce: Call DELETE /cars/{id} or any endpoint using Car.query.get().
- Diagnosis: SQLAlchemy 2.0 deprecated Query.get() in favor of Session.get().
- Research Process: SQLAlchemy migration guide, blog posts on API changes.
- Resolution: Replaced Car.query.get(car_id) with db.session.get(Car, car_id).
- Outcome Verification: Warnings disappeared, endpoints continued working.

Issue 4 — Invalid Image URLs in Car Listings

- Description: Vehicle listings displayed broken images when invalid URLs were submitted.
- Environment: Local and Azure React frontend.
- Steps to Reproduce: Submit a car with imageUrl=http://invalid-link.com.
- Diagnosis: No validation or fallback handling for image URLs.
- Research Process: MDN documentation for <img> error handling, frontend validation best practices.
- Resolution: Added frontend validation for HTTPS URLs and provided fallback placeholder image.
- Outcome Verification: Invalid links rejected; placeholder displayed if image fails.

Issue 5 — Backend Startup Failure on Azure

- Description: Azure deployment logs showed: No module named app.
- Environment: Backend on Azure Web App (Python 3.10).

- Steps to Reproduce: Deploy backend without startup command.
- Diagnosis: Azure default startup configuration could not locate the Flask app.
- Research Process: Azure App Service deployment guide, Gunicorn documentation.
- Resolution: Set Startup Command to gunicorn app:app --bind=0.0.0.0:${PORT}.
- Outcome Verification: Backend started correctly; GET /cars returned 200.

Issue 6 — Local Database Authentication Error

- Description: Backend failed with password authentication failed for user "usedcars_user".
- Environment: Local Flask + PostgreSQL.
- Steps to Reproduce: Start backend with wrong DB password.
- Diagnosis: Incorrect user password or missing privileges.
- Research Process: PostgreSQL authentication error documentation, DBA forums.
- Resolution: Reset user password, recreated role with proper privileges, updated DATABASE_URL.
- Outcome Verification: Backend connected successfully to local database.

Issue 7 — Wrong API Base URL in Frontend

- Description: Frontend returned 404 or network errors for all requests.
- Environment: React frontend.
- Steps to Reproduce: Deploy frontend with REACT_APP_API_BASE_URL still pointing to http://localhost:5000.
- Diagnosis: API base URL misconfigured for production.
- Research Process: React environment variables guide, Azure deployment logs.
- Resolution: Set REACT_APP_API_BASE_URL=https://backend.azurewebsites.net, rebuilt frontend.
- Outcome Verification: Frontend successfully communicated with backend APIs.

Issue 8 — React Page Refresh Returns 404

- Description: Visiting https://frontend.azurewebsites.net/listings directly returned 404.
- Environment: React frontend on Azure Web App.
- Steps to Reproduce: Navigate directly to non-root route.
- Diagnosis: React SPA requires fallback to index.html.
- Research Process: React Router deployment docs, Azure static site guides.
- Resolution: Configured serve -s build -l ${PORT} as startup command.
- Outcome Verification: Refreshing routes worked without 404 errors.

Issue 9 — Form Validation Failure

- Description: Submitting cars with missing or negative values caused backend 500 errors.
- Environment: Local React + Flask.
- Steps to Reproduce: Submit car with empty price field.
- Diagnosis: Frontend did not validate required fields; backend did not handle None values.
- Research Process: HTML5 form validation docs, Flask request validation examples.
- Resolution: Added required attributes in React form; backend returned 400 for invalid data.
- Outcome Verification: Invalid submissions rejected gracefully.

Issue 10 — PostgreSQL Foreign Key Constraint Error

- Description: Deleting a user failed: violates foreign key constraint on table cars.
- Environment: Local PostgreSQL.
- Steps to Reproduce: Delete user who still had cars in database.
- Diagnosis: Cars table referenced user ID with foreign key constraint.

- Research Process: PostgreSQL foreign key docs, SQL cascade delete examples.
- Resolution: Modified delete logic to remove associated cars before user deletion.
- Outcome Verification: Users and related cars deleted without errors.

Issue 11 — Node.js Version Mismatch

- Description: Running npm install failed with Unsupported engine.
- Environment: Local frontend development.
- Steps to Reproduce: Install dependencies using Node 16.
- Diagnosis: React project required Node.js 22 LTS.
- Research Process: Checked package.json engines field, official Node.js docs.
- Resolution: Installed Node 22 via nvm, reinstalled dependencies.
- Outcome Verification: npm start worked successfully.

Issue 12 — Backend Performance Issues on Azure

- Description: Backend became slow, with worker timeout logs under load.
- Environment: Backend on Azure App Service (Free plan).
- Steps to Reproduce: Send multiple concurrent requests.
- Diagnosis: Azure free tier lacked CPU/memory capacity; Gunicorn workers timing out.
- Research Process: Gunicorn tuning documentation, Azure performance docs.
- Resolution: Upgraded App Service plan to B1, tuned worker count.
- Outcome Verification: Backend handled concurrent requests smoothly.

Issue 13 — Password Stored in Plaintext

- Description: Initially, user passwords were saved as plaintext in the database.
- Environment: Local development.

- Steps to Reproduce: Register new user and inspect DB.
- Diagnosis: Backend did not implement password hashing.
- Research Process: Flask-Security best practices, Werkzeug security module docs.
- Resolution: Added generate_password_hash when saving password, check_password_hash on login.
- Outcome Verification: DB stored only hashed passwords; login verification worked.

4. System Usage Guide

This guide is intended for end users of the Used Car Marketplace system. It explains how to access the application, navigate its features, and complete the main workflows.

4.1 Accessing the Application

Hello, Let me show you our DriveEase Used Car Platform!

https://usedcar-frontend-azftfpbjeeghg4gz.centralus-01.azurewebsites.net

Test Accounts:

Admin:

Username: admin

Password: admin123

Normal user: You may register your own test account through the Register page.

4.2 Navigating Key Features

Home Page – Welcome to the Used Car Marketplace

This is the landing page of our platform, designed to give users a simple and inviting entry point.

At the top, we have a clear welcome message: "Welcome to the Used Car Marketplace!"

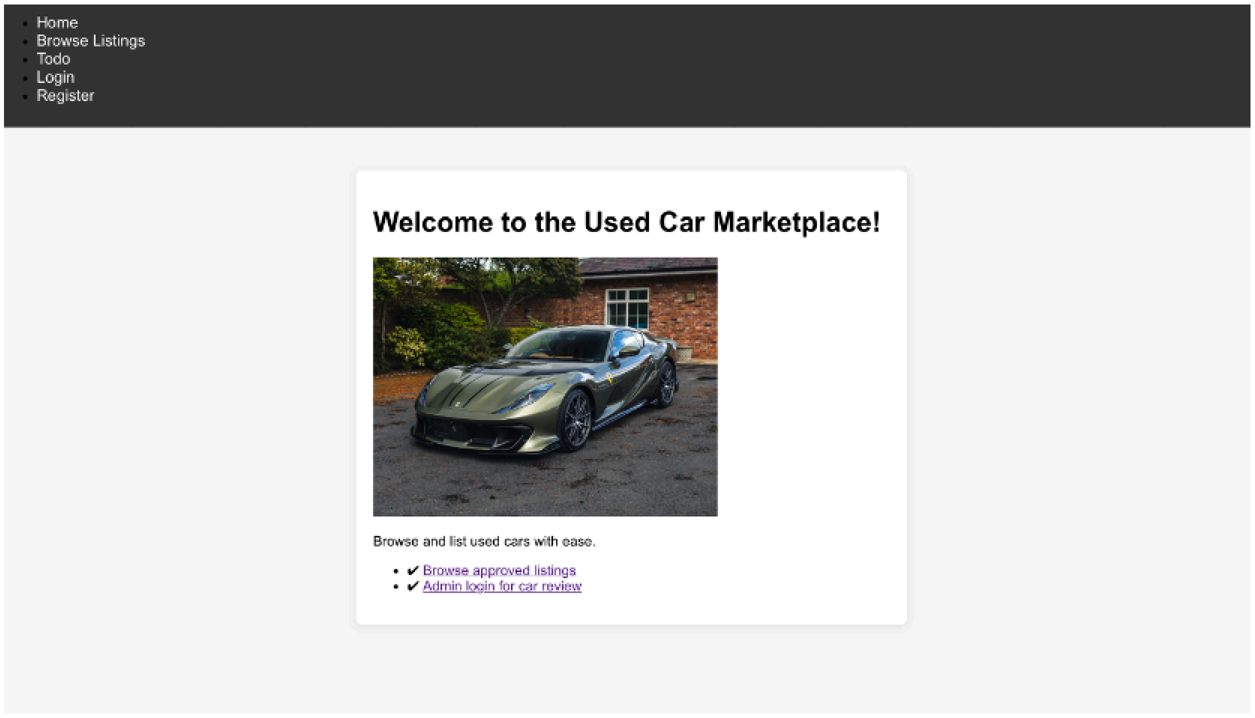
The center features a high-quality car image to create visual appeal and reinforce the automotive theme.

Below the image, users can choose to:

Browse approved listings – view cars that have passed the admin review process.

Admin login for car review – access the admin portal to approve or reject new car submissions.

The page layout is clean and intuitive, ensuring both buyers and administrators can quickly navigate to their desired actions.



Welcome to the Used Car Marketplace!

Browse and list used cars with ease.

- ✔ Browse approved listings
- ✔ Admin login for car review

Browse Car Listings Page

This page allows users to view all approved car listings currently available on the platform.

Each listing includes key vehicle details such as year, price, mileage, transmission type, fuel type, VIN, and location.

Seller contact information is displayed for direct communication between buyers and sellers.

Listings also show who submitted the car and a description field that can include updates like price changes or mileage adjustments.

In this example:

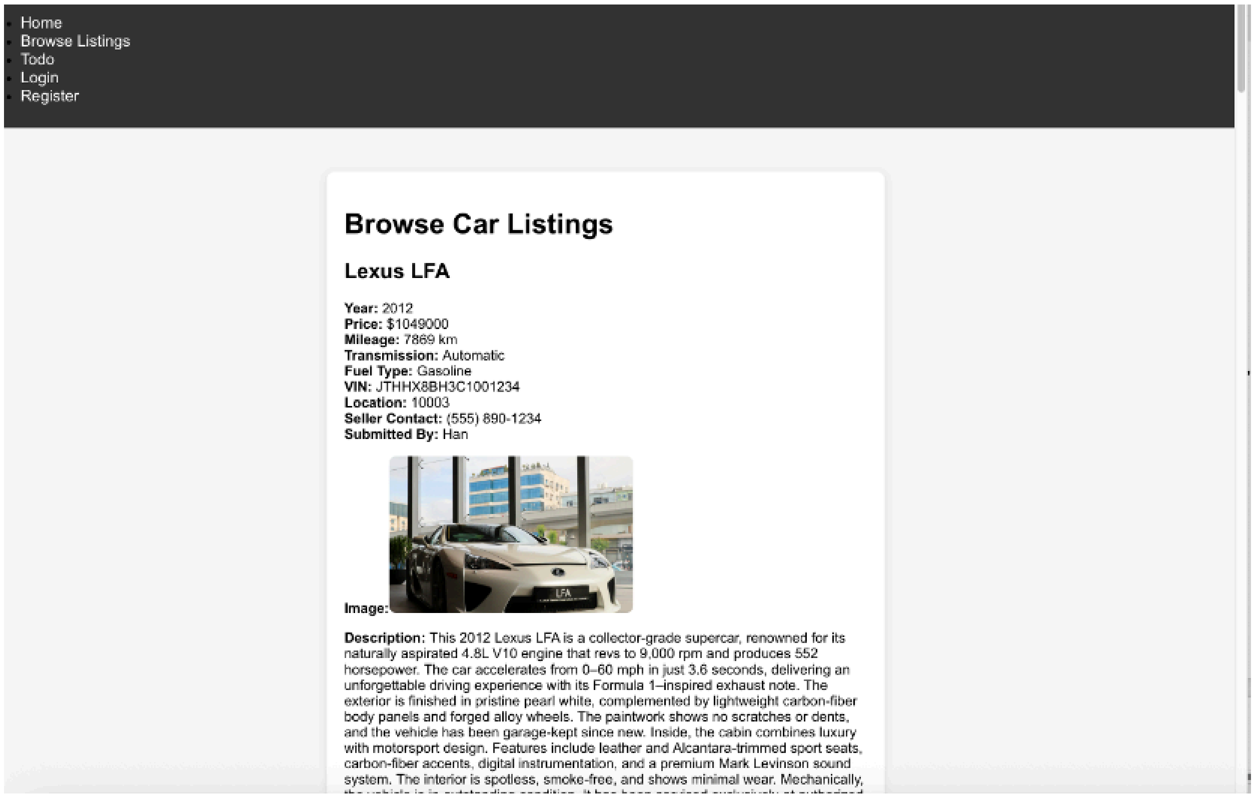
1）2012 Lexus LFA — $1,049,000 — 7,869 km

Collector-grade LFA with the 4.8L naturally aspirated V10 (552 hp), 0–60 mph in 3.6 s. Pristine exterior with carbon-fiber panels and forged wheels; garage-kept, no dents or scratches. Smoke-free interior with leather/Alcantara and Mark Levinson audio. Clean history and serviced at authorized Lexus facilities. Very low mileage; only ~500 units produced worldwide. Includes VIN and seller contact for verification. Ideal for collectors seeking a rare, investment-grade supercar.

2）2021 Lexus ES 350 — $35,999 — 35,786 km

Well-maintained midsize luxury sedan with a 3.5L V6 and 8-speed automatic. Clean title, no accident history; smoke-free cabin in excellent condition. Recent routine maintenance (oil, brakes, rotations); tires and brakes in good shape. Features include dual-zone climate, infotainment, premium audio, and active safety (lane departure alert, adaptive cruise, blind-spot monitoring). Balanced choice for buyers seeking comfort, reliability, and lower running costs.

The page layout focuses on clarity and transparency, ensuring buyers can quickly assess and compare different vehicles.



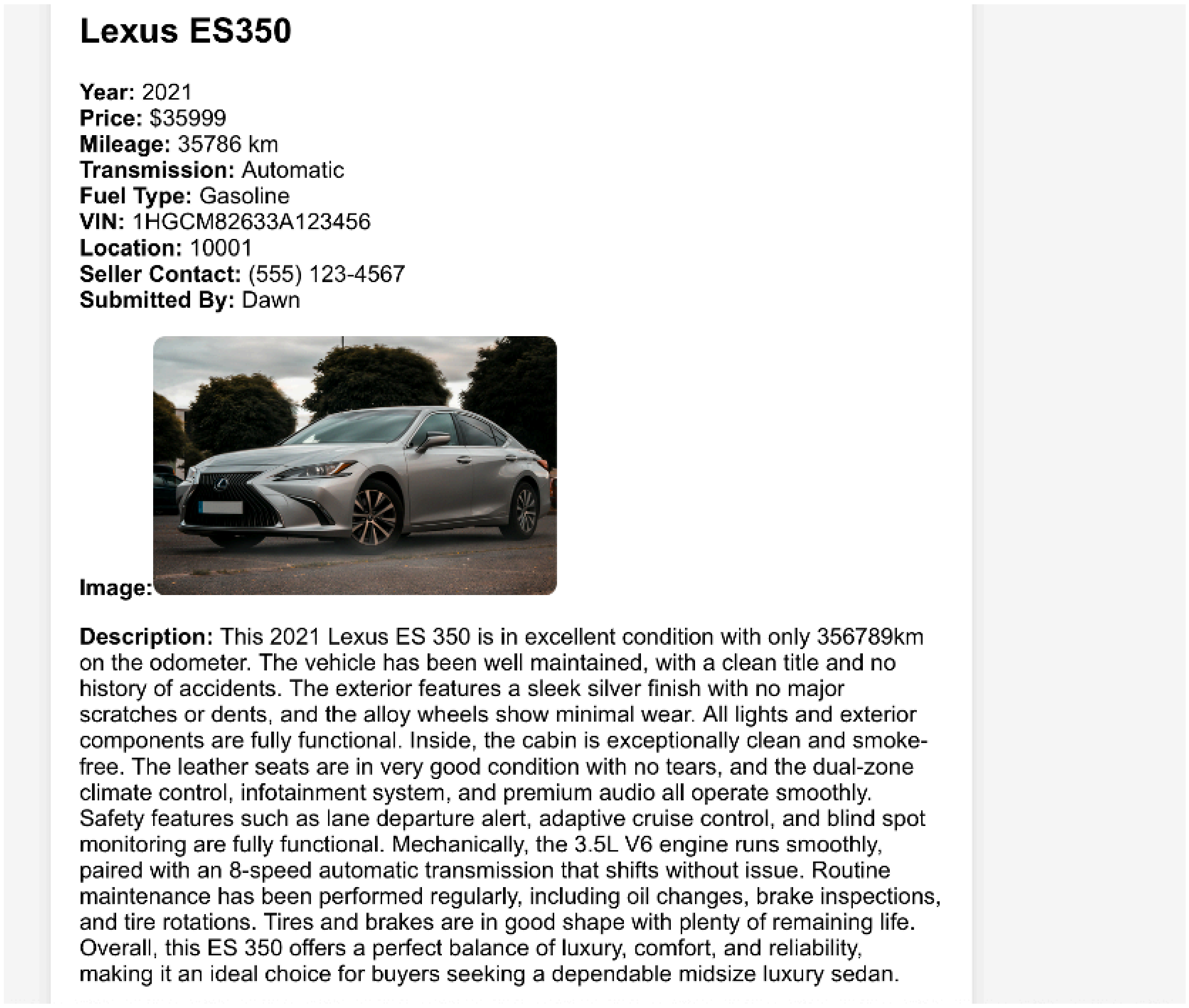
Browse Car Listings

Lexus LFA

Year: 2012
Price: $1049000
Mileage: 7869 km
Transmission: Automatic
Fuel Type: Gasoline
VIN: JTHHX8BH3C1001234
Location: 10003
Seller Contact: (555) 890-1234
Submitted By: Han

Image:

Description: This 2012 Lexus LFA is a collector-grade supercar, renowned for its naturally aspirated 4.8L V10 engine that revs to 9,000 rpm and produces 552 horsepower. The car accelerates from 0–60 mph in just 3.6 seconds, delivering an unforgettable driving experience with its Formula 1–inspired exhaust note. The exterior is finished in pristine pearl white, complemented by lightweight carbon-fiber body panels and forged alloy wheels. The paintwork shows no scratches or dents, and the vehicle has been garage-kept since new. Inside, the cabin combines luxury with motorsport design. Features include leather and Alcantara-trimmed sport seats, carbon-fiber accents, digital instrumentation, and a premium Mark Levinson sound system. The interior is spotless, smoke-free, and shows minimal wear. Mechanically,



Lexus ES350

Year: 2021
Price: $35999
Mileage: 35786 km
Transmission: Automatic
Fuel Type: Gasoline
VIN: 1HGCM82633A123456
Location: 10001
Seller Contact: (555) 123-4567
Submitted By: Dawn

Image:

Description: This 2021 Lexus ES 350 is in excellent condition with only 356789km on the odometer. The vehicle has been well maintained, with a clean title and no history of accidents. The exterior features a sleek silver finish with no major scratches or dents, and the alloy wheels show minimal wear. All lights and exterior components are fully functional. Inside, the cabin is exceptionally clean and smoke-free. The leather seats are in very good condition with no tears, and the dual-zone climate control, infotainment system, and premium audio all operate smoothly. Safety features such as lane departure alert, adaptive cruise control, and blind spot monitoring are fully functional. Mechanically, the 3.5L V6 engine runs smoothly, paired with an 8-speed automatic transmission that shifts without issue. Routine maintenance has been performed regularly, including oil changes, brake inspections, and tire rotations. Tires and brakes are in good shape with plenty of remaining life. Overall, this ES 350 offers a perfect balance of luxury, comfort, and reliability, making it an ideal choice for buyers seeking a dependable midsize luxury sedan.

Car Maintenance Checklist Page

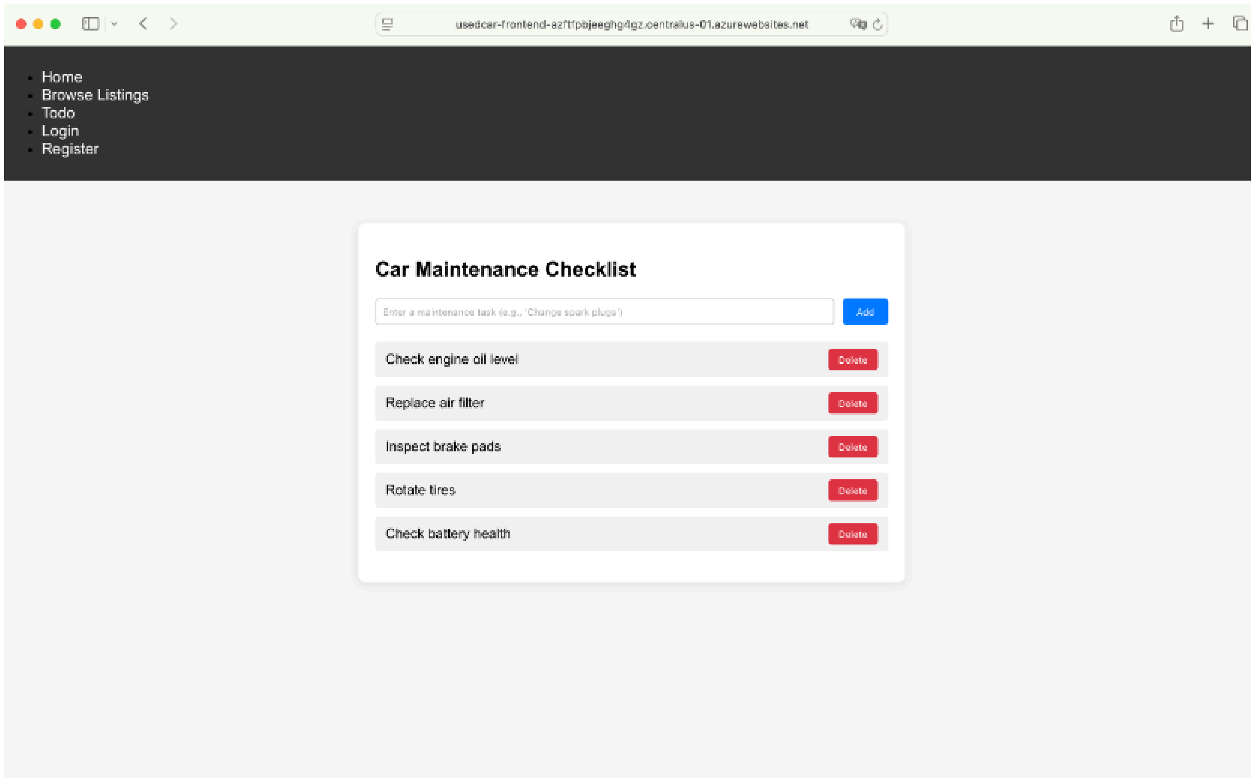
This page provides users with a simple and interactive tool to keep track of essential car maintenance tasks.

At the top, there’s an input field where users can enter a new maintenance task (e.g., “Change spark plugs”) and add it to the list with the Add button.

The checklist displays existing maintenance items such as checking engine oil level, replacing the air filter, inspecting brake pads, rotating tires, and checking battery health.

Each task has a Delete button, allowing users to remove completed or unnecessary tasks from the list.

This feature helps car owners stay organized and ensures timely maintenance, extending the vehicle’s lifespan and safety.



Login Page

This page provides secure access to both user and admin functionalities.

Users enter their username and password, then select their role from the dropdown menu — either User or Admin.

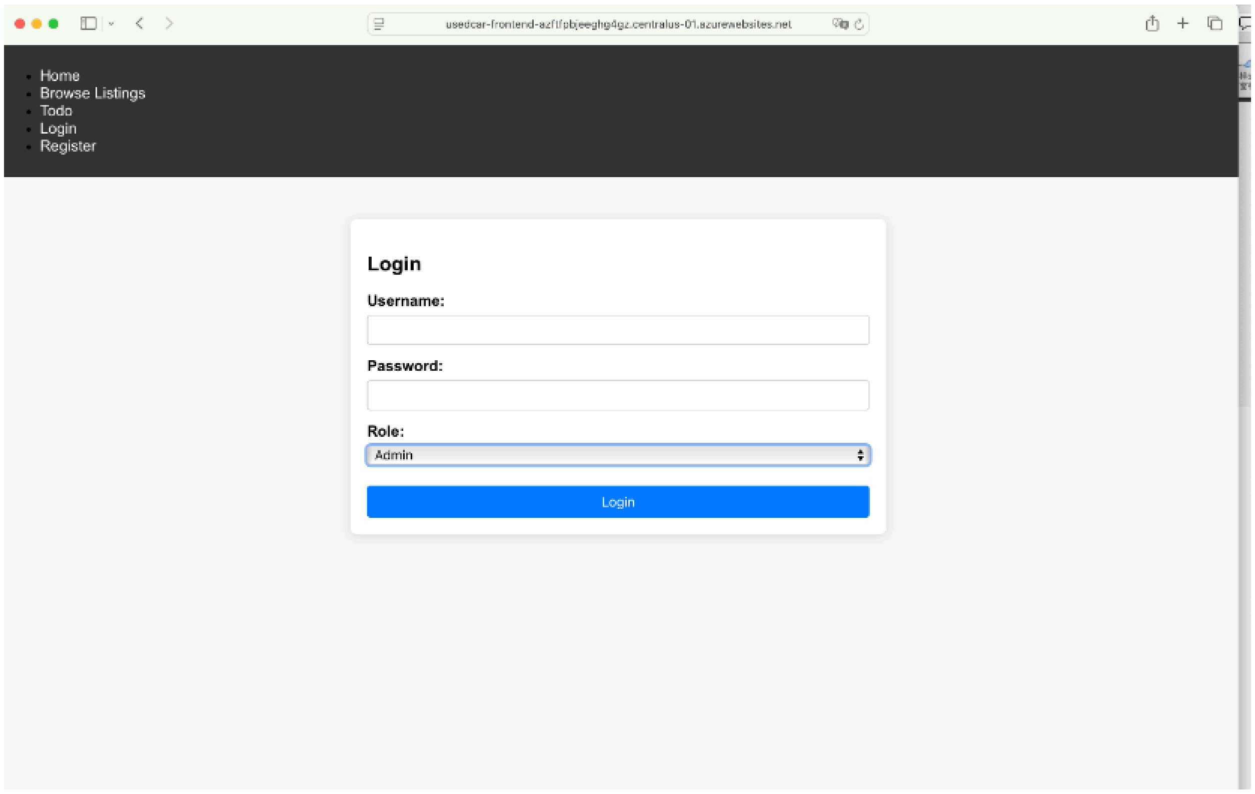
For demonstration and simplicity, the platform includes only one administrator account:

Username: admin

Password: admin123

Having a single admin account keeps the site lightweight, easier to manage, and more transparent for this capstone project.

This design choice allows for straightforward testing of admin functionalities, such as approving car listings and managing users.


usedcar-frontend-azftfpbjeeghg4gz.centralus-01.azurewebsites.net

- Home
- Browse Listings
- Todo
- Login
- Register

Login

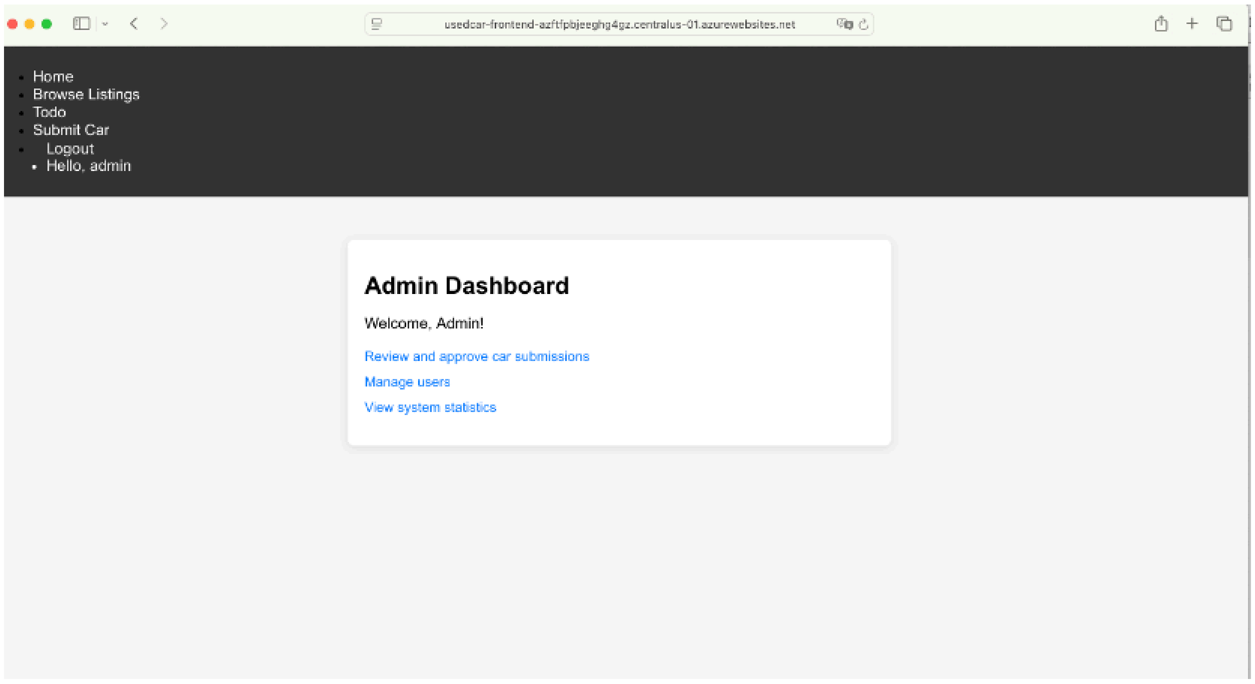
Username:

Password:

Role: Admin

Admin Dashboard & Features Overview

Once logged in as an admin, the user is directed to the Admin Dashboard. From here, the admin has access to three core management functions:


usedcar-frontend-azftfpbjeeghg4gz.centralus-01.azurewebsites.net

- Home
- Browse Listings
- Todo
- Submit Car
- Logout
- Hello, admin

Admin Dashboard

Welcome, Admin!

Review and approve car submissions

Manage users

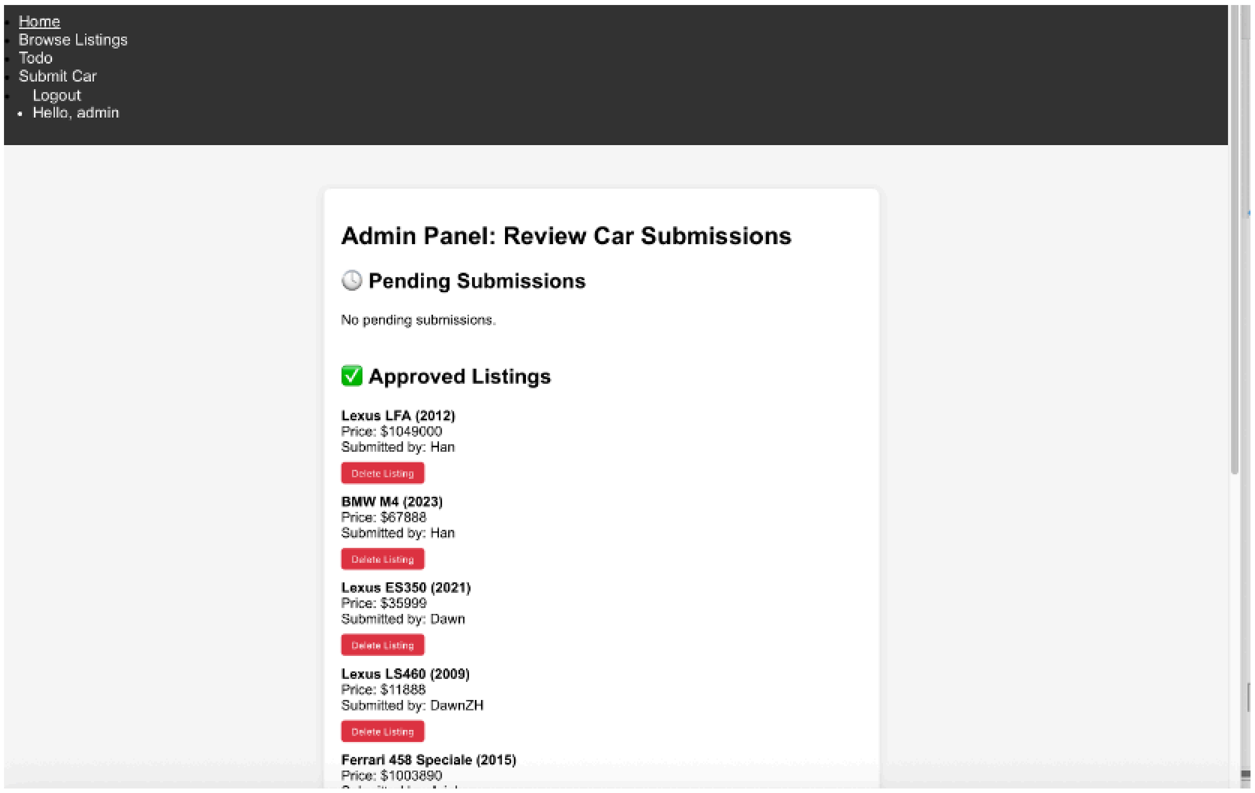
View system statistics

Review and Approve Car Submissions

Admins can see pending submissions awaiting approval and a list of all approved listings.

Each approved listing displays key details such as car model, year, price, and submitter.

The admin can remove any listing instantly by clicking the Delete Listing button.



Admin Panel: Review Car Submissions

🕓 Pending Submissions

No pending submissions.

✅ Approved Listings

Lexus LFA (2012)
Price: $1049000
Submitted by: Han

BMW M4 (2023)
Price: $67888
Submitted by: Han

Lexus ES350 (2021)
Price: $35999
Submitted by: Dawn

Lexus LS460 (2009)
Price: $11888
Submitted by: DawnZH

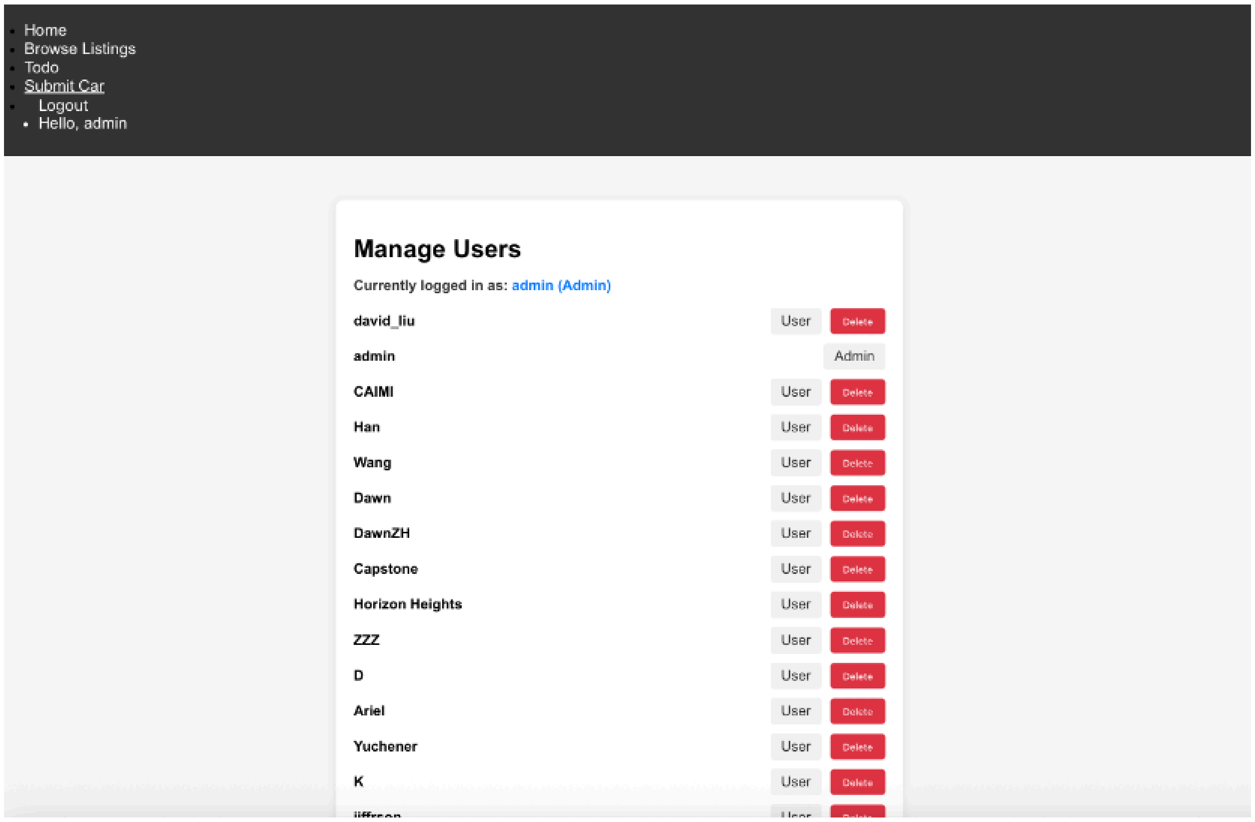
Ferrari 458 Speciale (2015)
Price: $1003890

Manage Users

Displays all registered users along with their roles (User or Admin).

The admin can delete any user account from the system with one click.

For this demo, there is only one admin account to maintain simplicity and transparency.



Manage Users

Currently logged in as: admin (Admin)

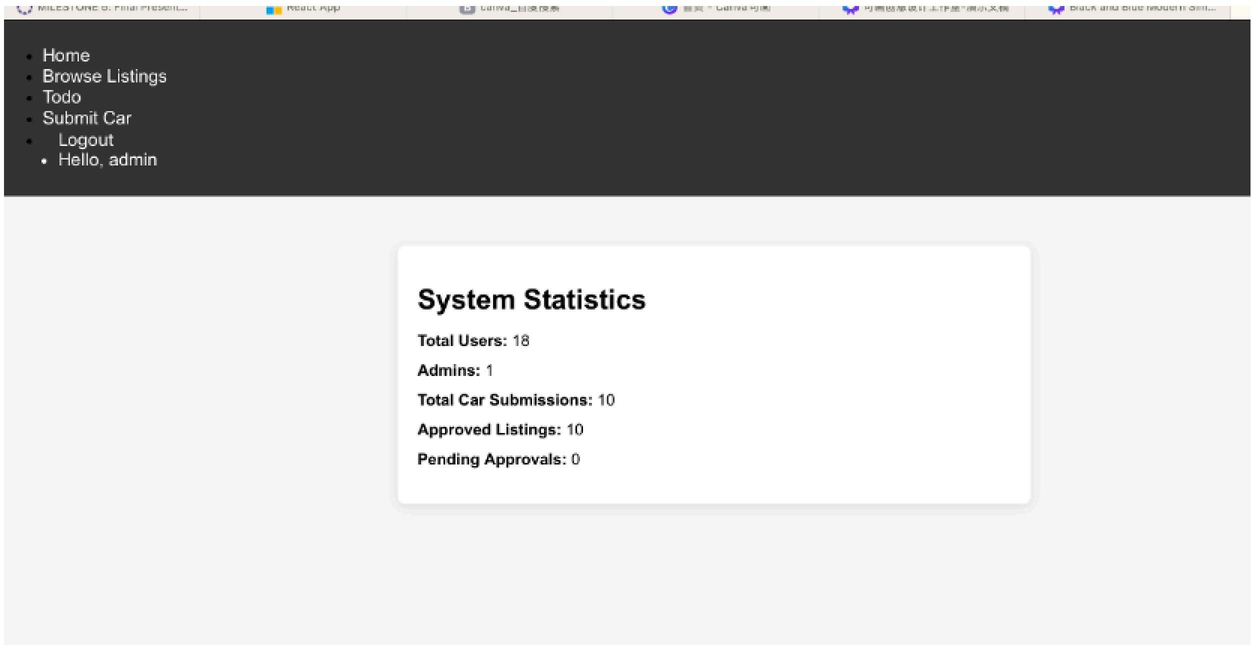
david_liu	User Delete
admin	Admin
CAIMI	User Delete
Han	User Delete
Wang	User Delete
Dawn	User Delete
DawnZH	User Delete
Capstone	User Delete
Horizon Heights	User Delete
ZZZ	User Delete
D	User Delete
Ariel	User Delete
Yuchener	User Delete
K	User Delete

View System Statistics

Shows real-time platform data, including total users, number of admins, total car submissions, approved listings, and pending approvals.

This provides the admin with an at-a-glance overview of system activity and workload.

This admin interface ensures that the platform remains secure, transparent, and well-maintained, allowing quick review of submissions, user management, and overall system monitoring in one central place.



System Statistics

Total Users: 18

Admins: 1

Total Car Submissions: 10

Approved Listings: 10

Pending Approvals: 0

Submit a Used Car Page

This page allows registered users to submit detailed information about a used car they want to sell.

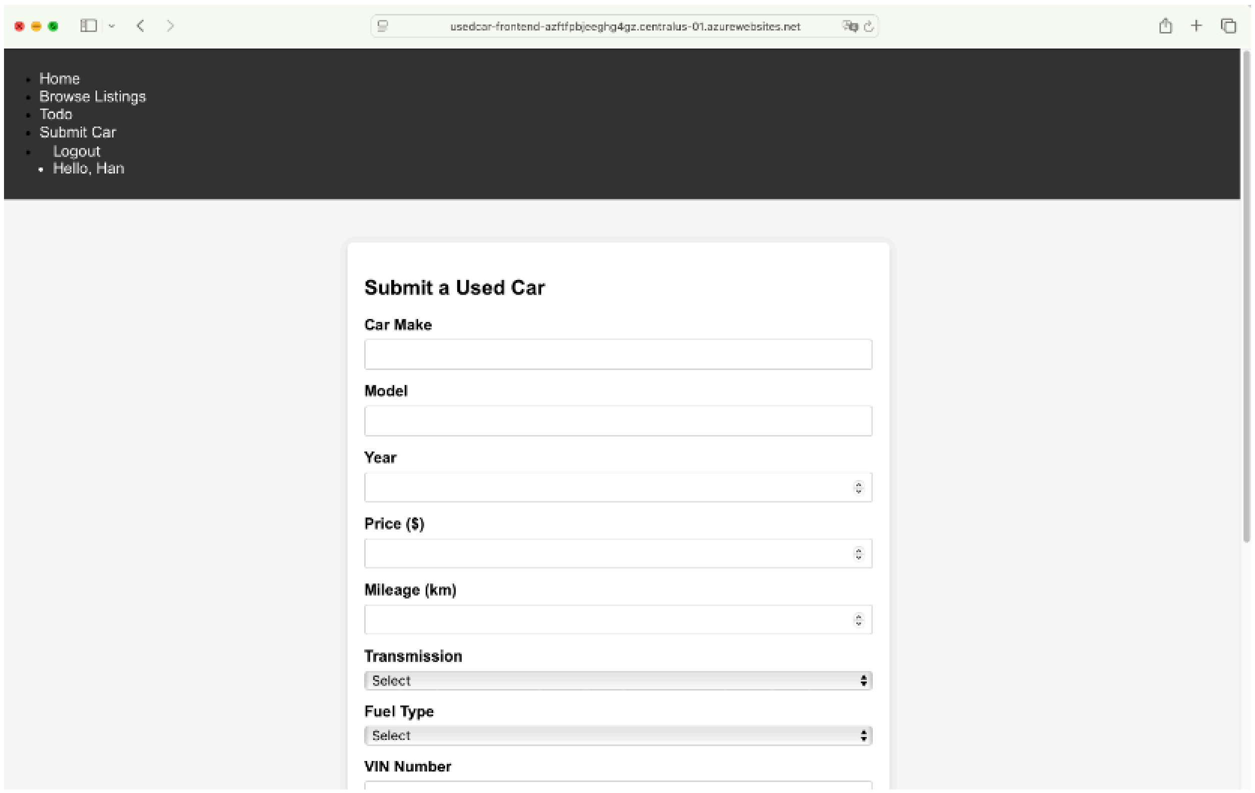
The form includes fields for Car Make, Model, Year, Price, Mileage, Transmission Type, Fuel Type, and VIN Number.

Each field ensures that the admin receives all necessary details to evaluate the submission.

Once submitted, the listing will be sent to the admin dashboard for review and approval before being published in the marketplace.

This process maintains quality control and ensures all published listings are accurate, complete, and trustworthy.

Users can submit a URL link to the vehicle’s images, and once approved by the admin, the listing will be displayed on the platform.


usedcar-frontend-azftfpbjeeghg4gz.centralus-01.azurewebsites.net

- Home
- Browse Listings
- Todo
- Submit Car
- Logout
- Hello, Han

Submit a Used Car

Car Make

Model

Year

Price ($)

Mileage (km)

Transmission: Select

Fuel Type: Select

VIN Number

Register Page

This page allows new users to create an account on the platform.

The form includes three essential fields:

Username – a unique identifier for the user.

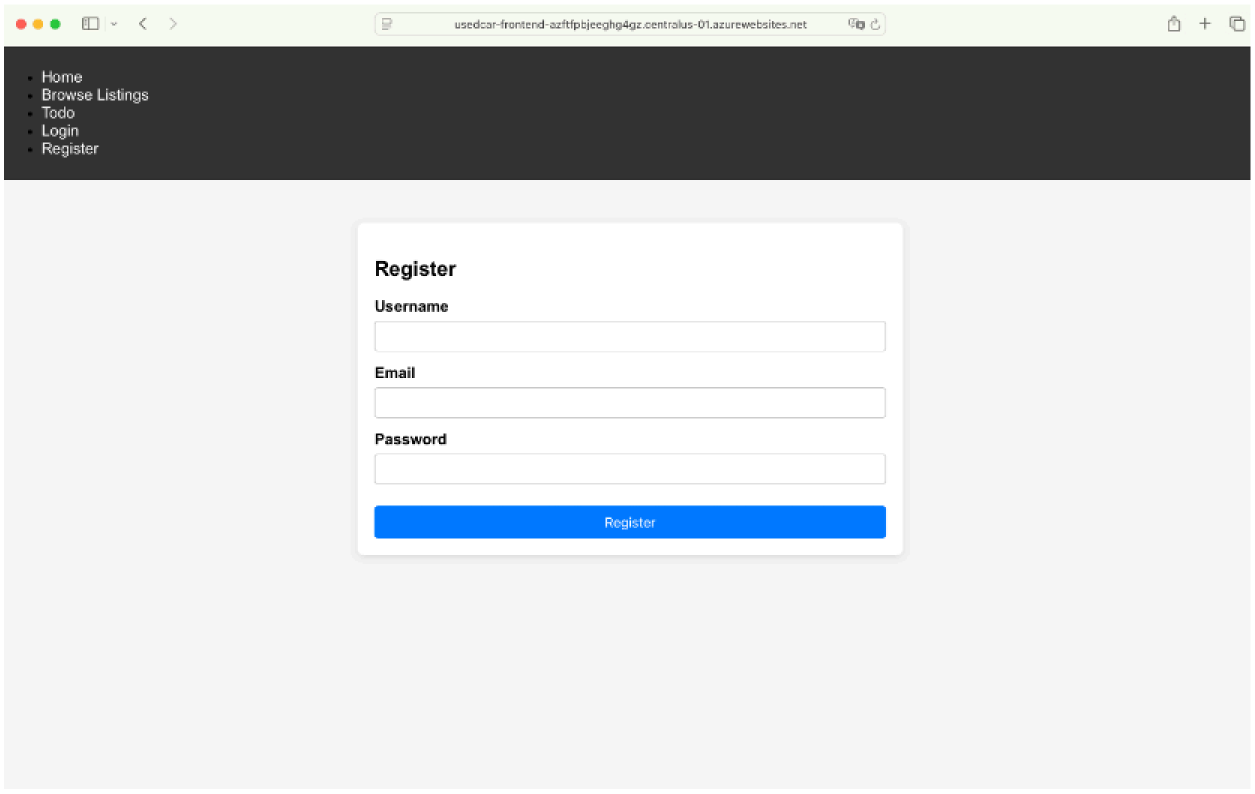
Email – used for account verification and communication.

Password – to secure the account.

Once the form is completed and the Register button is clicked, the system stores the user's information in the database.

After registration, users can log in to access features such as submitting a used car listing, browsing approved listings, and managing their own submissions.

This straightforward registration process ensures quick onboarding while maintaining security for all accounts



4.3 Main Workflows

Workflow 1: User Registration and Login

Navigate to Register page.

Enter username and password → click Submit.

Navigate to Login page.

Enter credentials → redirected to user account.

Workflow 2: Submit a Car for Sale

Log in as a normal user.

Go to Submit a Used Car page.

Fill out car details and provide a valid HTTPS image URL.

Submit → car appears in Admin Pending list.

Workflow 3: Approve a Car (Admin)

Log in as admin.

Open Admin Dashboard → Pending Cars.

Select a pending listing and click Approve.

The car moves to Approved list.

Workflow 4: Browse Approved Cars

Navigate to Browse Listings.

Approved cars display with all submitted details and images.

4.4 Known Limitations

Currently, there is only one default admin account (admin/admin123).

Images must be provided as valid HTTPS URLs; invalid links may show broken thumbnails.

The Maintenance Checklist is frontend-only and does not persist data in the database.

The system is optimized for demonstration purposes and may not scale under heavy concurrent traffic.

Password reset and email verification are not implemented.

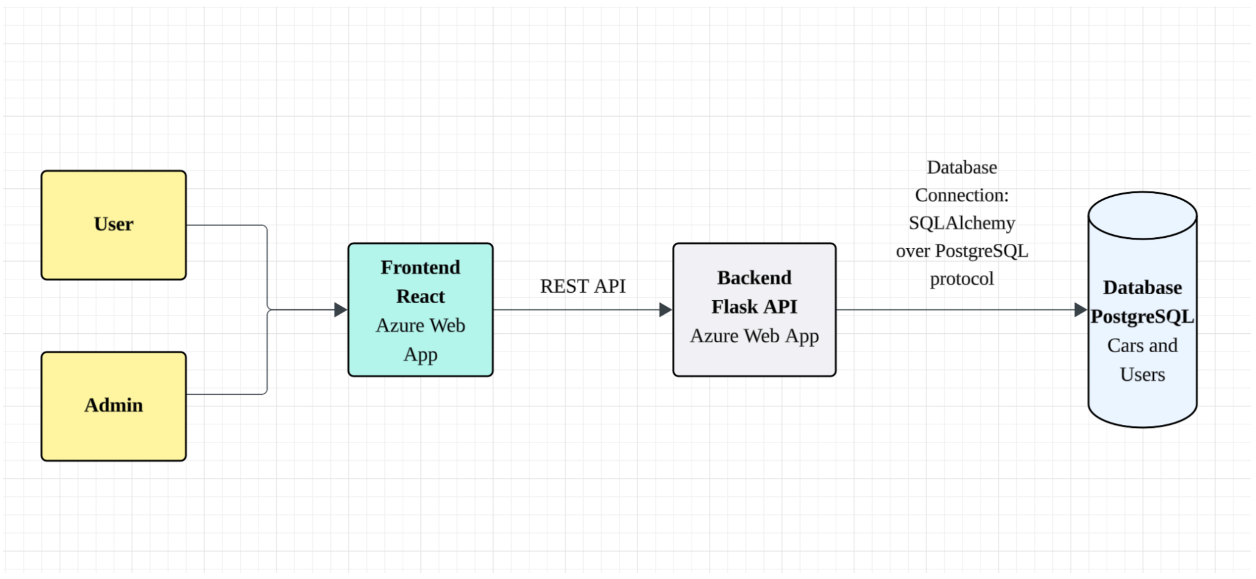
4.5 Support & Contact

- Email: z15684581213@outlook.com (Mon–Fri 9:00–18:00 CT, response within 1 business day)

- When reporting an issue, please include: your test account, page/feature, time, steps to reproduce, and a screenshot/error message.

5. Architecture Diagram

5.1 High-level diagram



Solid arrows: synchronous request/response over HTTPS or PostgreSQL protocol.

Teal box: user-facing UI (frontend).

Grey box: server-side services (backend).

Cylindrical shape: persistent data store (PostgreSQL).

Yellow boxes: user personas (User/Admin).

5.2 Components & roles

Users / Admin
Role: End users (buyers/sellers) and administrators (moderation, approvals, user management).
Frontend — React (Azure Web App, Node 22)
Role: Single-page application that renders UI, performs client-side validation, and calls backend REST APIs over HTTPS.
Backend — Flask API (Azure Web App, Python 3.10)

Role: Business logic and data access; authentication; CRUD for users and car listings; admin actions (approve/reject, manage users, stats).
Database — Azure PostgreSQL Flexible Server
Role: Persistent storage for users and cars tables.
Connectivity: SQLAlchemy over the PostgreSQL protocol (SSL enforced in production).
External APIs
None currently. Images in listings are referenced via user-provided HTTPS URLs (no uploads).

5.3 Communication flows

Browser → Frontend
User/Admin accesses the React app over HTTPS.

Frontend → Backend (REST over HTTPS)
Endpoints include /users, /login, /cars, /cars/{id}/approve, /cars/approved, /cars/pending, etc.

Backend → Database (PostgreSQL)
SQLAlchemy ORM manages connections and queries to PostgreSQL with SSL in production.

Security notes
CORS on the backend allows only the deployed frontend origins.
Secrets (DB URL, Flask SECRET_KEY) are provided via environment variables; no secrets in source control.

5.4 Environments & Deployment

Production Environment

The Used Car Marketplace system is fully deployed to Azure App Service with the following components:

Frontend (React SPA) hosted on Azure Web App.

Backend (Flask API) deployed on Azure Web App (Python 3.10) using Gunicorn.

Database (PostgreSQL) on Azure PostgreSQL Flexible Server, accessed via SQLAlchemy with SSL enabled.

Production Access URL:
👉 https://usedcar-frontend-azftfpbjeeghg4gz.centralus-01.azurewebsites.net/

Access via QR Code

End users can also scan the QR code below to directly open the application on a mobile device:

Deployment Notes

Configuration: Environment variables (DATABASE_URL, SECRET_KEY, API base URL) managed in Azure App Service.

Frontend Build: Compiled with npm run build, served with Node.js.

Backend Startup: Runs with Gunicorn:

```
gunicorn app:app --bind=0.0.0.0:${PORT}
```

Monitoring: Azure App Service Log Stream for backend/frontend logs; Azure Portal for database health.

Local Environment (Development)

Frontend: Runs with React development server on http://localhost:3000.

Backend: Runs with Flask development server on http://localhost:5000.

Database: Local PostgreSQL instance on localhost:5432.

Usage:

Developers run npm start (frontend) and flask run (backend).

CORS allows http://localhost:3000.

.env files point API base URL to http://localhost:5000.

Purpose: Used for feature development and debugging before deployment.

Staging Environment (Optional, Pre-Production)

Frontend: Deployed to Azure Web App (Node.js runtime).

Backend: Deployed to Azure Web App (Python 3.10, Gunicorn).

Database: Azure PostgreSQL Flexible Server (staging instance).

Usage:

Accessible at staging URLs (e.g., https://<fe-staging>.azurewebsites.net).

CORS configured for staging frontend domain.

Secrets managed through Azure App Service → Configuration.

Purpose: Used for smoke testing and integration validation before production release.

6. Deployment Pipeline Overview

The system can be deployed using a simple CI/CD pipeline with the following stages:

Build Stage

Frontend: run npm install and npm run build to generate production-ready static files.

Backend: install Python dependencies from requirements.txt.

Test Stage

Run unit tests and API test scripts (manual or automated) to verify functionality.

Ensure that both backend endpoints and frontend build succeed without errors.

Staging Deployment

Deploy build artifacts to an Azure staging slot or a separate staging environment.

Run smoke tests (basic API calls, frontend load check) to confirm stability.

Approval Gate

After staging validation, a team member or administrator approves promotion to production.

Production Deployment

Swap slots (if using Azure deployment slots) or redeploy the latest build to production.

Environment variables (DATABASE_URL, SECRET_KEY, CORS_ALLOWED_ORIGINS) are injected from Azure App Service configuration.

Rollback Procedure

If the new deployment fails, restore the previous working build via Azure App Service deployment history.

Database rollback is handled through PostgreSQL backups if schema changes are applied.

7. Security Considerations

The application implements basic but effective security mechanisms appropriate for a course project:

Authentication

Users log in with a username and password.

Passwords are stored in the database hashed using Werkzeug security functions, never in plaintext.

Authorization

Two roles exist:

Admin: can review and approve pending cars, manage users, and view system statistics.

Normal User: can register, log in, submit cars, and browse approved listings.

Role checks are enforced in backend API endpoints.

Transport Security

All production traffic runs over HTTPS (Azure App Service provides TLS by default).

Configuration & Secrets

Sensitive values (DATABASE_URL, SECRET_KEY) are stored in Azure App Service Configuration, not hardcoded in source code.

CORS Policy

Backend only accepts requests from the deployed frontend domain to prevent unauthorized cross-origin access.

Security Considerations

- Never log secrets/tokens; redact sensitive values in logs.

- Enforce HTTPS in production; reject plain HTTP.

- CORS: allow only your frontend origin(s); do not use "*".

- Store secrets via environment variables / Key Vault; JWT/SECRET length ≥ 32 bytes.