Django Car Price Showing System

A Beginner-Friendly Tutorial with MySQL & Bootstrap

What you will build:

A complete Django web application that allows users to register, log in, and manage car listings. Each car has a name, image, and price. The site uses MySQL as the database and Bootstrap 5 for clean, light-themed pages.

Feature	Included
User Registration & Login	Yes
Add / View / Edit / Delete Cars (CRUD)	Yes
Car Image Upload	Yes
MySQL Database	Yes
Bootstrap 5 Light Theme	Yes
Beginner-Friendly Code	Yes

Table of Contents

1. Introduction & What You'll Learn

Welcome! In this tutorial, you will build a complete **Car Price Showing System** using Django (a popular Python web framework). The system shows a list of cars with their names, images, and prices. Users can register, log in, and manage their own car listings.

This tutorial is designed for **beginners**. Every step is explained in simple words, and the code is kept clean and easy to read. You will learn:

- How to install and set up Django on your computer.
- How to connect Django to a MySQL database.
- How to build models, views, and templates (the heart of Django).
- How to add user registration and login using Django's built-in tools.
- How to perform CRUD operations (Create, Read, Update, Delete).
- How to upload and display images.
- How to design clean, light-themed pages with Bootstrap 5.

TIP

By the end of this tutorial, you will have a working website with login, image uploads, and a MySQL-backed car database. You can use this as a base for any car dealership, vehicle showroom, or product listing site.

2. Prerequisites & Tools You Need

Before we start, please make sure you have the following installed on your computer. Don't worry — installation steps are included in the next section.

Tool	Purpose	Version
Python	The programming language Django uses	3.10 or higher
pip	Installs Python packages (comes with Python)	Latest
MySQL Server	The database that stores users and cars	5.7 or higher
A code editor	VS Code is recommended	Any
Web browser	Chrome, Firefox, or Edge	Any

3. Installation Steps

Step 1: Install Python

Download Python from **python.org** and install it. During installation on Windows, tick the box that says **'Add Python to PATH'**.

Check that Python is installed by opening a terminal (Command Prompt) and typing:

```
python --version
```

Step 2: Install MySQL

Download MySQL Community Server from **dev.mysql.com**. During installation, remember the **root password** — you will need it.

> **TIP**
>
> On Windows, you can also use XAMPP which includes MySQL (MariaDB). It is easier for beginners. Just install XAMPP and start the MySQL module.

Step 3: Create a Project Folder

Open your terminal and create a folder for your project. We will call it **car_price_system**.

```
mkdir car_price_system
cd car_price_system
```

Step 4: Create a Virtual Environment

A virtual environment keeps your project's packages separate from other projects. This is a best practice in Python.

```
# Create a virtual environment named 'venv'
python -m venv venv

# Activate it (Windows)
venv\Scripts\activate

# Activate it (Mac/Linux)
source venv/bin/activate
```

After activation, you will see **(venv)** at the start of your terminal line. That means you are inside the virtual environment.

Step 5: Install Django and MySQL Connector

```
pip install django
pip install mysqlclient
pip install Pillow
```

What each package does:

- **django** — the web framework itself.
- **mysqlclient** — lets Django talk to MySQL.
- **Pillow** — needed for image uploads (ImageField).

NOTE

If **mysqlclient** fails to install on Windows, try this alternative instead: **pip install pymysql**. We will show you how to use it in the database section.

4. Creating the Django Project

Now we create the actual Django project and an app inside it. In Django, a **project** is the whole website, and an **app** is a feature module (like 'cars' or 'users').

Step 1: Start a new Django project

```
django-admin startproject carproject .
```

The dot (**.**) at the end tells Django to create the project in the current folder. This avoids creating an extra nested folder.

Step 2: Create the 'cars' app

```
python manage.py startapp cars
```

This creates a folder called **cars/** with files like models.py, views.py, and admin.py. Each file has a clear job, which we will use later.

Step 3: Register the app

Open **carproject/settings.py** and find the **INSTALLED_APPS** list. Add **'cars'** at the bottom:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'cars',  # our new app
]
```

Step 4: Configure media and static files

We need to tell Django where to save uploaded car images. Add these lines at the bottom of **settings.py**:

```
import os

# For uploaded images (car photos)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# For static files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

# Login redirects
LOGIN_URL = '/login/'
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/login/'
```

5. Connecting MySQL Database

Step 1: Create a database in MySQL

Open a MySQL command prompt (or phpMyAdmin if using XAMPP) and run:

```
CREATE DATABASE car_db CHARACTER SET utf8mb4;
```

Step 2: Update settings.py with database config

Find the **DATABASES** section in **settings.py** and replace it with:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'car_db',
        'USER': 'root',
        'PASSWORD': 'your_mysql_password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Replace **your_mysql_password** with the password you set during MySQL installation. If you use XAMPP, the default is usually empty (leave it as '').

Optional: Using PyMySQL Instead

If **mysqlclient** did not install on your machine, install **pymysql** and add the following at the very top of **carproject/__init__.py**:

```
import pymysql
pymysql.install_as_MySQLdb()
```

Step 3: Run initial migrations

Django comes with built-in tables (for users, sessions, admin, etc.). Create them in your database:

```
python manage.py migrate
```

NOTE

If you get a database connection error, double-check your password, that MySQL is running, and that the database **car_db** exists.

6. Final Folder Structure

After all the steps in this tutorial, your project will look exactly like this. Use this as a map while you work.

```
car_price_system/
■
■■■ venv/                        (your virtual environment)
■
■■■ carproject/                  (the Django project folder)
■   ■■■ __init__.py
■   ■■■ settings.py              (all settings including DB)
■   ■■■ urls.py                  (main URL router)
■   ■■■ wsgi.py
■   ■■■ asgi.py
■
■■■ cars/                        (our app)
■   ■■■ migrations/
■   ■■■ templates/
■   ■   ■■■ cars/
■   ■       ■■■ base.html        (master template with navbar)
■   ■       ■■■ home.html        (list of cars)
■   ■       ■■■ car_form.html    (add / edit form)
■   ■       ■■■ car_detail.html (single car view)
■   ■       ■■■ confirm_delete.html
■   ■       ■■■ register.html
■   ■       ■■■ login.html
■   ■■■ __init__.py
■   ■■■ admin.py                 (admin panel config)
■   ■■■ apps.py
■   ■■■ forms.py                 (Car + Register forms)
■   ■■■ models.py                (Car model)
■   ■■■ urls.py                  (app URLs)
■   ■■■ views.py                 (all view functions)
■
■■■ media/                       (auto-created for uploaded images)
■   ■■■ car_images/
■
■■■ static/                      (CSS, custom JS)
■   ■■■ css/
■       ■■■ style.css
■
■■■ manage.py                    (Django command runner)
■■■ requirements.txt             (list of installed packages)
```

TIP

Templates are stored inside **cars/templates/cars/** (note the double folder). This is a Django convention that prevents naming conflicts between apps.

7. Building the Car Model

A **model** is a Python class that becomes a database table. Each attribute becomes a column. We will create one model called **Car**.

Edit cars/models.py

```
from django.db import models
from django.contrib.auth.models import User


class Car(models.Model):
    """A single car listing."""

    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.ImageField(upload_to='car_images/')
    description = models.TextField(blank=True)
    owner = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name='cars'
    )
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name
```

Explanation of each field:

- **name** — the car name (e.g. Toyota Prius). Up to 100 characters.
- **price** — a decimal number with 2 decimal places (e.g. 25000.00).
- **image** — a photo. Files are saved to **media/car_images/**.
- **description** — optional details about the car.
- **owner** — the user who added the car. Linked to Django's built-in User model.
- **created_at** — the date/time the car was added (set automatically).

Make and apply migrations

Migrations are how Django turns your models into actual database tables. Run these two commands every time you change a model:

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

Register the model in admin.py

Django gives you a free admin panel. To see your cars there, register the model:

```
# cars/admin.py
from django.contrib import admin
from .models import Car

admin.site.register(Car)
```

Create an admin user (superuser)

```
python manage.py createsuperuser
```

Follow the prompts to set a username, email, and password. You will use these later to log into **/admin/**.

8. User Registration & Login

Django includes a complete authentication system. We will use it for login and logout, and write a simple registration form ourselves.

Step 1: Create the forms file

Create a new file **cars/forms.py** and add:

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from .models import Car


class RegisterForm(UserCreationForm):
    """Form for new users to sign up."""
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class CarForm(forms.ModelForm):
    """Form for adding and editing cars."""
    class Meta:
        model = Car
        fields = ['name', 'price', 'image', 'description']
```

Why two forms?

RegisterForm extends Django's built-in **UserCreationForm** so we get password validation for free. **CarForm** is a **ModelForm** — it builds itself from the Car model, saving us a lot of code.

Step 2: Add views for register, login, logout

Open **cars/views.py** and start with the auth views. We will add CRUD views in the next section.

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import login, logout, authenticate
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import RegisterForm, CarForm
from .models import Car


def register_view(request):
    """Sign up a new user."""
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)  # auto-login after register
            messages.success(request, 'Welcome! Your account is ready.')
            return redirect('home')
    else:
        form = RegisterForm()
    return render(request, 'cars/register.html', {'form': form})
```

```
def login_view(request):
    """Log a user in."""
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
        else:
            messages.error(request, 'Invalid username or password.')
    return render(request, 'cars/login.html')


def logout_view(request):
    """Log the user out and send to login page."""
    logout(request)
    return redirect('login')
```

Explanation of the auth views:

- **register_view** — shows the form, saves the new user, and logs them in instantly.
- **login_view** — checks the username/password using **authenticate()**, then calls **login()**.
- **logout_view** — ends the session and redirects to login.
- **messages** — Django's flash message system. We will display them in the template.

9. CRUD Views for Cars

CRUD stands for **Create, Read, Update, Delete**. These are the four basic operations every database app needs. We will write one view function for each.

Add these views to cars/views.py

```
# READ: list all cars (home page)
def home_view(request):
    cars = Car.objects.all().order_by('-created_at')
    return render(request, 'cars/home.html', {'cars': cars})


# READ: single car detail
def car_detail_view(request, pk):
    car = get_object_or_404(Car, pk=pk)
    return render(request, 'cars/car_detail.html', {'car': car})
```

```
# CREATE: add a new car (login required)
@login_required
def car_create_view(request):
    if request.method == 'POST':
        form = CarForm(request.POST, request.FILES)
        if form.is_valid():
            car = form.save(commit=False)
            car.owner = request.user
            car.save()
            messages.success(request, 'Car added successfully.')
            return redirect('home')
    else:
        form = CarForm()
    return render(request, 'cars/car_form.html',
                  {'form': form, 'title': 'Add New Car'})
```

```
# UPDATE: edit an existing car
@login_required
def car_update_view(request, pk):
    car = get_object_or_404(Car, pk=pk, owner=request.user)
    if request.method == 'POST':
        form = CarForm(request.POST, request.FILES, instance=car)
        if form.is_valid():
            form.save()
            messages.success(request, 'Car updated successfully.')
            return redirect('car_detail', pk=car.pk)
    else:
        form = CarForm(instance=car)
    return render(request, 'cars/car_form.html',
                  {'form': form, 'title': 'Edit Car'})
```

```
# DELETE: remove a car
@login_required
def car_delete_view(request, pk):
    car = get_object_or_404(Car, pk=pk, owner=request.user)
    if request.method == 'POST':
        car.delete()
        messages.success(request, 'Car deleted.')
        return redirect('home')
    return render(request, 'cars/confirm_delete.html', {'car': car})
```

Important things to notice

• **@login_required** — only logged-in users can add, edit, or delete cars. The home page is public so anyone can see prices.

• **request.FILES** — required for uploading images. Without it, the image will not be saved.

• **commit=False** — we save the form without writing to the DB so we can attach the current user as the owner first.

• **get_object_or_404(Car, pk=pk, owner=request.user)** — this not only fetches the car, but also makes sure the logged-in user is the **owner**. This stops users from editing or deleting other people's cars.

TIP

If you want any logged-in user to edit any car (not just their own), simply remove **owner=request.user** from the get_object_or_404 calls.

10. URLs Configuration

URLs map web addresses (like **/cars/3/**) to view functions. We need URLs in two places: the project level and the app level.

Step 1: Create cars/urls.py

```
# cars/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home_view, name='home'),
    path('register/', views.register_view, name='register'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('car/new/', views.car_create_view, name='car_create'),
    path('car/<int:pk>/', views.car_detail_view, name='car_detail'),
    path('car/<int:pk>/edit/', views.car_update_view, name='car_update'),
    path('car/<int:pk>/delete/', views.car_delete_view, name='car_delete'),
]
```

Step 2: Update carproject/urls.py

Now connect the app's URLs to the main project, and tell Django how to serve uploaded images during development:

```
# carproject/urls.py
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('cars.urls')),
]

# Serve uploaded images in development
if settings.DEBUG:
    urlpatterns += static(
        settings.MEDIA_URL, document_root=settings.MEDIA_ROOT
    )
```

TIP

The **name=** in each URL is very important. Templates use these names (like **{% url 'home' %}**) instead of hard-coding the URL. If you ever change a URL path, the templates still work.

11. Bootstrap Templates (Light Theme)

Templates are HTML files that Django fills with data. We use Bootstrap 5 from a CDN so there is nothing to download. The colors are kept light, soft, and clean for easy reading.

base.html (master template)

Create **cars/templates/cars/base.html**. Every other page will extend this one.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}Car Showroom{% endblock %}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/
dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    body { background:#f8fafc; color:#1f2937; }
    .navbar { background:#ffffff; border-bottom:1px solid #e5e7eb; }
    .navbar-brand { color:#2563eb !important; font-weight:600; }
    .card { border:1px solid #e5e7eb; border-radius:10px; }
    .btn-primary { background:#2563eb; border-color:#2563eb; }
    .price { color:#10b981; font-weight:600; }
  </style>
</head>
```

```
<body>
  <nav class="navbar navbar-expand-lg">
    <div class="container">
      <a class="navbar-brand" href="{% url 'home' %}">Car Showroom</a>
      <div class="ms-auto">
        {% if user.is_authenticated %}
          <span class="me-3">Hi, {{ user.username }}</span>
          <a href="{% url 'car_create' %}" class="btn btn-primary btn-sm">
            + Add Car</a>
          <a href="{% url 'logout' %}" class="btn btn-outline-secondary btn-sm">
            Logout</a>
        {% else %}
          <a href="{% url 'login' %}" class="btn btn-outline-primary btn-sm">
            Login</a>
          <a href="{% url 'register' %}" class="btn btn-primary btn-sm">
            Register</a>
        {% endif %}
      </div>
    </div>
  </nav>

  <main class="container py-4">
    {% if messages %}
      {% for m in messages %}
        <div class="alert alert-info">{{ m }}</div>
      {% endfor %}
    {% endif %}
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

home.html (car list)

```
{% extends 'cars/base.html' %}
{% block title %}All Cars{% endblock %}

{% block content %}
  <h2 class="mb-4">Available Cars</h2>
  <div class="row g-4">
    {% for car in cars %}
      <div class="col-md-4">
        <div class="card h-100 shadow-sm">
          <img src="{{ car.image.url }}" class="card-img-top"
               style="height:200px; object-fit:cover;" alt="{{ car.name }}">
          <div class="card-body">
            <h5 class="card-title">{{ car.name }}</h5>
            <p class="price mb-2">$ {{ car.price }}</p>
            <a href="{% url 'car_detail' car.pk %}"
               class="btn btn-sm btn-outline-primary">View</a>
          </div>
        </div>
      </div>
    {% empty %}
      <p class="text-muted">No cars yet. Add the first one!</p>
    {% endfor %}
  </div>
{% endblock %}
```

car_detail.html (single car)

```
{% extends 'cars/base.html' %}
{% block title %}{{ car.name }}{% endblock %}

{% block content %}
  <div class="row">
    <div class="col-md-6">
      <img src="{{ car.image.url }}" class="img-fluid rounded">
    </div>
    <div class="col-md-6">
      <h2>{{ car.name }}</h2>
      <h4 class="price">$ {{ car.price }}</h4>
      <p class="text-muted">Added by {{ car.owner.username }}</p>
      <p>{{ car.description }}</p>
      {% if user == car.owner %}
        <a href="{% url 'car_update' car.pk %}"
           class="btn btn-primary">Edit</a>
        <a href="{% url 'car_delete' car.pk %}"
           class="btn btn-outline-danger">Delete</a>
      {% endif %}
    </div>
  </div>
{% endblock %}
```

car_form.html (add / edit)

```
{% extends 'cars/base.html' %}
{% block title %}{{ title }}{% endblock %}

{% block content %}
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card shadow-sm">
        <div class="card-body">
          <h3 class="mb-3">{{ title }}</h3>
          <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {% for field in form %}
              <div class="mb-3">
                <label class="form-label">{{ field.label }}</label>
                {{ field }}
                {% if field.errors %}
                  <small class="text-danger">{{ field.errors|join:', ' }}</small>
                {% endif %}
              </div>
            {% endfor %}
            <button class="btn btn-primary">Save</button>
            <a href="{% url 'home' %}" class="btn btn-link">Cancel</a>
          </form>
        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

NOTE

The attribute **enctype="multipart/form-data"** on the form is required for image uploads. Without it, the image field will be empty when you save.

confirm_delete.html

```
{% extends 'cars/base.html' %}
{% block content %}
  <div class="card p-4 shadow-sm">
    <h4>Delete "{{ car.name }}"?</h4>
    <p class="text-muted">This action cannot be undone.</p>
    <form method="post">
      {% csrf_token %}
      <button class="btn btn-danger">Yes, delete</button>
      <a href="{% url 'car_detail' car.pk %}"
         class="btn btn-link">Cancel</a>
    </form>
  </div>
{% endblock %}
```

login.html

```
{% extends 'cars/base.html' %}
{% block content %}
  <div class="row justify-content-center">
    <div class="col-md-5">
      <div class="card shadow-sm">
        <div class="card-body">
          <h3 class="mb-3">Login</h3>
          <form method="post">
            {% csrf_token %}
            <div class="mb-3">
              <label class="form-label">Username</label>
              <input name="username" class="form-control" required>
            </div>
            <div class="mb-3">
              <label class="form-label">Password</label>
              <input type="password" name="password"
                     class="form-control" required>
            </div>
            <button class="btn btn-primary w-100">Login</button>
          </form>
          <p class="mt-3 text-center">
            No account?
            <a href="{% url 'register' %}">Register here</a>
          </p>
        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

register.html

```
{% extends 'cars/base.html' %}
{% block content %}
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card shadow-sm">
        <div class="card-body">
          <h3 class="mb-3">Create Account</h3>
          <form method="post">
            {% csrf_token %}
            {% for field in form %}
              <div class="mb-3">
                <label class="form-label">{{ field.label }}</label>
                {{ field }}
                {% if field.errors %}
                  <small class="text-danger">
                    {{ field.errors|join:', ' }}
                  </small>
                {% endif %}
              </div>
            {% endfor %}
            <button class="btn btn-primary w-100">Register</button>
          </form>
        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

12. Running the Project

Make sure your virtual environment is active and your MySQL server is running, then start the Django development server:

```
python manage.py runserver
```

Open your browser and go to:

```
http://127.0.0.1:8000/
```

You should see the Car Showroom homepage with an empty list. Click **Register** to create an account, then click **+ Add Car** to add your first car.

Save your installed packages

It is good practice to save the list of packages you installed. This way, anyone (including future-you) can install the same versions:

```
pip freeze > requirements.txt
```

To restore them on another machine: **pip install -r requirements.txt**

13. Testing the Application

Walk through these steps to make sure everything works:

#	Action	Expected Result
1	Visit /register/	Registration form appears
2	Fill the form and submit	Auto-logged in, redirected to home
3	Click '+ Add Car'	Car form appears
4	Add name, price, image, save	Car appears on the home page
5	Click the car	Detail page shows full info
6	Click Edit on your car	Edit form pre-filled
7	Click Delete	Confirm page, then car is removed
8	Logout	Redirected to login page
9	Visit /admin/	Admin panel; superuser can manage all cars

14. Common Errors & Fixes

ModuleNotFoundError: No module named 'MySQLdb'

Either install **mysqlclient** (pip install mysqlclient) or switch to **pymysql** as shown in Section 5.

django.db.utils.OperationalError: Access denied

Wrong MySQL password or user. Update the PASSWORD value in settings.py DATABASES section.

django.db.utils.OperationalError: Unknown database 'car_db'

You did not create the database. Run: CREATE DATABASE car_db; in MySQL.

Image not showing on home page

Make sure you added the static() line at the bottom of carproject/urls.py and that DEBUG=True during development.

'NoneType' object has no attribute 'url' (in template)

The image field is empty for that car. Wrap with: {% if car.image %} ... {% endif %}

CSRF verification failed

Every POST form must include {% csrf_token %} just inside the <form> tag.

Page not found at /static/...

Run: python manage.py collectstatic only when deploying. In development, static files are served automatically.

ImportError: cannot import name 'login'

Check spelling in your import: from django.contrib.auth import login, logout, authenticate

15. Next Steps

Congratulations! You now have a working Car Price Showing System. Here are some ideas to take it further:

- **Search and filter** — let users search cars by name or price range.
- **Pagination** — show 9 cars per page using Django's **Paginator**.
- **Categories** — add a Brand or Type model (Toyota, Honda, etc.).
- **Multiple images** — create a separate CarImage model linked to Car.
- **Email notifications** — send an email when a new car is added.
- **API** — expose the cars as a JSON API using Django REST Framework.
- **Deployment** — host on PythonAnywhere, Railway, or a VPS.
- **Better UI** — add a hero banner, footer, and image lightbox.

TIP

Practice is the key. Try changing things, breaking them, and fixing them. Each error you solve teaches you more than reading any tutorial.

Happy coding!

More tutorials at **egotechworld.com**

PHP | Python | Django | React | Node.js