ChatGPT

Land Selling System – Full-Stack Django Tutorial

This tutorial walks through building a simple **Land Selling System** using Django (Python), MySQL, HTML/CSS, and Bootstrap. We'll cover environment setup, database configuration, creating models, user authentication, admin customizations, views/templates, and handling static/media files. Each step includes code examples and explanations suitable for beginners.

1. Setting Up the Development Environment

- **Install Python 3:** Ensure you have Python 3.10+ installed (from python.org or your OS package manager).
- **Create a Virtual Environment:** It's best to isolate project dependencies. Use Python's built-in `venv` module to make an environment. For example:

```
python -m venv env
```

Activate it with `source env/bin/activate` on macOS/Linux or `env\Scripts\activate.bat` on Windows. This dedicated virtual environment keeps packages separate per project [1]. - **Install Django and MySQL Connector:** Inside the activated venv, install Django and the MySQL driver. Django can be installed via pip, and MySQL requires a DB API driver (e.g. `mysqlclient`) [2] [3]. For example:

```
pip install Django
pip install mysqlclient
```

This ensures Django and MySQL connectivity. (For Windows or Python 3.12+, you may need alternative drivers like PyMySQL if `mysqlclient` fails.)

- **Install MySQL Server:** Install and run MySQL/MariaDB on your machine. Create a database for the project (e.g. `landdb`) and a user with permissions. In MySQL prompt:

```
CREATE DATABASE landdb;
CREATE USER 'landuser'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON landdb.* TO 'landuser'@'localhost';
```

This prepares a MySQL database for Django.

2. Creating the Django Project and App

- **Start a New Project:** In your project folder, run:

```
django-admin startproject landselling
cd landselling
```

This creates a new Django project named `landselling`.

- **Create an App:** Inside the project, create an app to hold listings and other logic:

```
python manage.py startapp listings
python manage.py startapp accounts
```

Add these apps to `landselling/settings.py` under `INSTALLED_APPS`. For example:

```
INSTALLED_APPS = [
    # default apps...
    'django.contrib.auth',
    'django.contrib.staticfiles',
    # our apps:
    'listings',
    'accounts',
]
```

- **Configure Database Settings:** Edit `landselling/settings.py` to use MySQL. For example:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'landdb',
        'USER': 'landuser',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Set `ENGINE` to `'django.db.backends.mysql'` and provide your DB name, user, and password [4]. Ensure you installed the `mysqlclient` package, as Django requires it to talk to MySQL [2].

- **Apply Migrations:** Run migrations to create the auth tables and others:

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

This sets up the database tables.

3. Defining Models

Create models for **LandListing** and a user **Profile** (extending Django's User):

- **LandListing Model (in `listings/models.py`):** Stores land details.

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

class LandListing(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    location = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.ImageField(upload_to='land_images/', blank=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.title} - {self.location}"
```

Here, `price` uses `DecimalField`, `image` uses `ImageField` (requires the Pillow library, install via `pip install Pillow`), and `owner` links to the default `User` model. These fields capture basic land info.

- **User Profile Model (in `accounts/models.py`):** (Optional) Add extra user info.

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

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(blank=True)
    phone = models.CharField(max_length=20, blank=True)

    def __str__(self):
        return self.user.username
```

This `Profile` model uses a one-to-one link to extend `User`. You can customize fields as needed. Don't forget to create migrations after defining models:

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

4. User Registration and Authentication

Django includes a robust auth system with built-in models, forms and views [5].

- **Registration View:** Create a view to register new users (e.g. in `accounts/views.py`):

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = UserCreationForm()
    return render(request, 'accounts/register.html', {'form': form})
```

This uses Django's `UserCreationForm`. On successful save, it redirects to the login page.

- **Login/Logout:** Use Django's built-in auth views for login and logout [6]:

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('login/', auth_views.LoginView.as_view(template_name='accounts/
login.html'), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    # other URLs...
]
```

Create templates `accounts/register.html` and `accounts/login.html`. Be sure to include `django.contrib.auth` and `django.contrib.sessions` in `INSTALLED_APPS` (these are included by default in a new project) so that the authentication system works [5].

- **Require Login for Certain Views:** For pages like "add new land", decorate the view with `@login_required` so only logged-in users can access it.

5. Customizing the Admin Panel

Django's admin interface lets us manage models easily. In `listings/admin.py` and `accounts/admin.py`:

- **Register Models:**

```
from django.contrib import admin
from .models import LandListing

@admin.register(LandListing)
class LandListingAdmin(admin.ModelAdmin):
    list_display = ('title', 'location', 'price', 'owner')
```

```
from django.contrib import admin
from .models import Profile

@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ('user', 'phone')
```

The `list_display` option controls which fields show up in the admin list view [7]. If you don't set `list_display`, the admin shows only the `__str__()` of each item. By defining it, we see multiple columns (e.g. title, price, etc.) for each `LandListing` [7].

- **Superuser:** Create an admin user to access the site:

```
python manage.py createsuperuser
```

Then run the server and log into `/admin` to see and manage Listings and Users.

6. Views and URL Routing

Create views to handle each page:

- **Home Page (Listings):** Show all land listings. In `listings/views.py`:

```
from django.shortcuts import render
from .models import LandListing

def home(request):
    lands = LandListing.objects.all().order_by('-created_at')
    return render(request, 'listings/home.html', {'lands': lands})
```

- **Listing Detail Page:** Show details of one listing. In `listings/views.py`:

```
from django.shortcuts import get_object_or_404

def listing_detail(request, pk):
    land = get_object_or_404(LandListing, pk=pk)
    return render(request, 'listings/detail.html', {'land': land})
```

- **Add New Listing:** Provide a form to add a land (login required). In `listings/views.py`:

```
from django.contrib.auth.decorators import login_required
from .forms import LandForm  # we'll create this form

@login_required
def add_land(request):
    if request.method == 'POST':
        form = LandForm(request.POST, request.FILES)
        if form.is_valid():
            land = form.save(commit=False)
            land.owner = request.user
            land.save()
            return redirect('home')
    else:
        form = LandForm()
    return render(request, 'listings/add_land.html', {'form': form})
```

Create `LandForm` in `listings/forms.py` based on `LandListing` model (or use a `ModelForm`).

- **Register URLs:** In `landselling/urls.py` and app `urls.py` files, map these views:

```
from django.urls import path, include

urlpatterns = [
    path('', include('listings.urls')),  # home and listing detail
    path('accounts/', include('accounts.urls')),  # register, login,
logout
]
```

And in `listings/urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('listing/<int:pk>/', views.listing_detail,
name='listing_detail'),
```

```
    path('add/', views.add_land, name='add_land'),
]
```

7. Templates with Bootstrap

- **Base Template:** Create a `base.html` that all pages extend. Include Bootstrap CSS/JS via CDN for responsive design. For example:

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Land Selling{% endblock %}</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
bootstrap.min.css">
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>
<body>
    <!-- Navbar -->
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'home' %}">LandSelling</a>
        <div class="collapse navbar-collapse">
          <ul class="navbar-nav ms-auto">
            {% if user.is_authenticated %}
              <li class="nav-item"><a class="nav-link" href="{% url
'add_land' %}">Add Land</a></li>
              <li class="nav-item"><a class="nav-link" href="{% url
'logout' %}">Logout</a></li>
            {% else %}
              <li class="nav-item"><a class="nav-link" href="{% url
'login' %}">Login</a></li>
              <li class="nav-item"><a class="nav-link" href="{% url
'register' %}">Register</a></li>
            {% endif %}
          </ul>
        </div>
      </div>
    </nav>
    <div class="container mt-4">
      {% block content %}{% endblock %}
    </div>
    <!-- Bootstrap JS Bundle -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/
bootstrap.bundle.min.js"></script>
```

```
</body>
</html>
```

This uses Bootstrap classes (navbar, container, etc.) for a responsive layout.

- **Home Page Template:** In `listings/templates/listings/home.html`:

```
{% extends 'base.html' %}
{% block title %}Home – LandSelling{% endblock %}
{% block content %}
<h1>Available Lands</h1>
<div class="row">
  {% for land in lands %}
    <div class="col-md-4 mb-3">
      <div class="card">
        {% if land.image %}
          <img src="{{ land.image.url }}" class="card-img-top"
alt="{{ land.title }}">
        {% endif %}
        <div class="card-body">
          <h5 class="card-title">{{ land.title }}</h5>
          <p class="card-text">{{ land.location }} – ${{ land.price }}</
p>
          <a href="{% url 'listing_detail' land.pk %}" class="btn btn-
primary">View Details</a>
        </div>
      </div>
    </div>
  {% empty %}
    <p>No listings available.</p>
  {% endfor %}
</div>
{% endblock %}
```

This loops over `lands`, using Bootstrap grid and cards.

- **Listing Detail Template:** In `listings/templates/listings/detail.html`:

```
{% extends 'base.html' %}
{% block content %}
<h2>{{ land.title }}</h2>
<p><strong>Location:</strong> {{ land.location }}</p>
<p><strong>Price:</strong> ${{ land.price }}</p>
<p>{{ land.description }}</p>
{% if land.image %}
  <img src="{{ land.image.url }}" class="img-fluid"
alt="{{ land.title }}">
```

```
{% endif %}
{% endblock %}
```

- **Add Land Template:** In `listings/templates/listings/add_land.html`:

```
{% extends 'base.html' %}
{% block content %}
<h2>Add New Land</h2>
<form method="post" enctype="multipart/form-data">
  {% csrf_token %}
  {{ form.as_p }}
  <button class="btn btn-success">Submit</button>
</form>
{% endblock %}
```

The form uses `multipart/form-data` for image upload.

- **Register/Login Templates:** In `accounts/templates/accounts/register.html` (and similarly for login), include the Django form:

```
{% extends 'base.html' %}
{% block content %}
<h2>Register</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button class="btn btn-primary">Sign Up</button>
</form>
{% endblock %}
```

Notice we use `{% load static %}` in `base.html` and `{% static 'path' %}` to link CSS or images, leveraging Django's static-files system [8].

8. Static and Media Files

- **Static Files (CSS/JS):** Define `STATIC_URL = '/static/'` in `settings.py` (default). Put CSS/JS in `static/` folders (e.g. `appname/static/appname/style.css`). In templates, load static and refer to them: e.g. `<link href="{% static 'appname/style.css' %}" rel="stylesheet">` [8]. Django's staticfiles app will serve these during development automatically [8].

- **Media Files (Uploads):** In `settings.py`, add:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

This tells Django where to save uploaded files. In `urls.py`, add a URL pattern to serve media during development:

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... your url patterns ...
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

This uses the helper function to serve files under `MEDIA_ROOT` at `MEDIA_URL` [9]. Now when a user uploads an image to a `LandListing`, it will be saved in `media/land_images/` and accessible via its URL.

9. Running Migrations and Testing

- **Apply Migrations:** Every time you change models, run:

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

This creates/updates database tables.

- **Run the Development Server:** Start the Django server:

```
python manage.py runserver
```

Visit `http://127.0.0.1:8000/` in your browser. You should see the home page (even if empty). Test creating an account (register/login) and adding land listings. Check that images upload and display correctly.

- **Basic Validation:** Django's forms handle validation. For example, `UserCreationForm` enforces password rules and matching. In `LandForm` (a `ModelForm` for `LandListing`), Django will enforce non-empty fields by default (unless you set `blank=True`). If a form is invalid, Django returns the same template with `form.errors` displayed under each field. For example:

```
{{ form.non_field_errors }}
{% for field in form %}
  {{ field.label_tag }} {{ field }}
  {{ field.errors }}
{% endfor %}
```

This shows any error messages. You can also add model validators (e.g. `price = models.DecimalField(..., validators=[MinValueValidator(0)])`) to enforce rules.

- **Error Handling:** Use Django's error pages or handle exceptions in views if needed. For a beginner tutorial, relying on form validation and requiring login (`@login_required`) covers most common cases (e.g. redirecting anonymous users to login).

By following these steps, you'll have a basic Land Selling system where users can register/login, post new land listings with images, view listings on the home page, and see details. The admin site lets you manage all listings and users with customized list displays [7]. You can further extend this by adding search, filters, more user profile fields, or deploying to a live server.

Sources: We used official Django documentation and tutorials to guide setup (e.g. installing via `venv`, linking MySQL, static/media configuration) [2] [3] [4] [8] [9] [5] [7]. These references provide the detailed background for each step.

---

[1] Django Create Virtual Environment
https://www.w3schools.com/django/django_create_virtual_environment.php

[2] [3] [4] How to install Django | Django documentation | Django
https://docs.djangoproject.com/en/5.2/topics/install/

[5] [6] Django Tutorial Part 8: User authentication and permissions - Learn web development | MDN
https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Authentication

[7] The Django admin site | Django documentation | Django
https://docs.djangoproject.com/en/5.2/ref/contrib/admin/

[8] [9] How to manage static files (e.g. images, JavaScript, CSS) | Django documentation | Django
https://docs.djangoproject.com/en/5.2/howto/static-files/