TUSHABE TRINITY FRANCESCO

STEPS TO CREATE A DJANGO WEB APPLICATION

Django is used as the web framework to build the backend of the web application. Django provides a convenient and powerful set of tools to handle various aspects of web development, including database management, URL routing, template rendering, and user authentication. Below is a sample documentation that explains how Django is used in the project:

Project Title: Spare Parts Management Web Application

Introduction

The BeRealMotors Spare Parts Company Django Web Application is a software solution designed to assist the company in managing their spare parts inventory and sales records. The application provides features for the manager and workers to add and update spare parts, record sales, issue receipts, and manage user roles.

Prerequisites(Requirements)

Before setting up the application, ensure you have the following prerequisites installed:

- Python (version 3.x)
- Pip (Python package manager)

Installation

Set up Virtual Environment

We recommend using a virtual environment to isolate the project dependencies. Open a terminal or command prompt and follow these steps:

 # Create a new directory for the project

mkdir spare_parts_project

cd spare_parts_project

 # Create and activate the virtual environment

pip install virtualenv

virtualenv venv(the virtual environment name)

 # On Windows

venv\Scripts\activate

 # On macOS or Linux

source venv/bin/activate

Install Django and Related Libraries

With the virtual environment activated, install Django and other required libraries using pip:

pip install django #this installs the latest version of django

Pip install django==3.0.8 # this specifies the version of django

pip install django-filter # For search and filtering functionality

pip install django-crispy-forms # For forms

pip install crispy-bootstrap4 # For bootstrap Form designs

pip install django-bootstrap-v5 #For using Bootstrap

Steps to Create the Project

Create the Project Directory: In the terminal or command prompt, navigate to the desired location and create a new directory for the project:

Set Up Virtual Environment: Create and activate a virtual environment to isolate the project's dependencies

source venv/bin/activate *-for linux/macos*

Create the Django Project: Run the following command to create a new Django project (e.g., spareparts):

django-admin startproject spareparts

Create the App: Create a new Django app (e.g., spareapp) within the project:

python manage.py startapp spareapp

Configure the Project Structure: The project structure will include the main project directory, the app directory, and necessary settings and configuration files.

Edits in settings.py

Installed Apps: Add the newly created app (spareapp) to the INSTALLED_APPS list in settings.py. This ensures that Django recognizes the app and includes it in the project

Database Configuration: By default, Django uses SQLite as the database. You can configure a different database if required (e.g., PostgreSQL, MySQL). Update the DATABASES setting in settings.py accordingly

Static URLs: Configure the static URLs to serve static files (CSS, JS, Images)

Templates Directory: Configure the location of the template directory to store the HTML templates for the app:

os.path.join(BASE_DIR, 'templates')

Templates

Django's Templates are used to generate HTML pages with dynamic content.

- index.html: Home page template.
- all_spare_parts.html: Template to list all spare parts.
- add_spare_part.html: Template to add new spare parts.
- sale.html: Template to record sales.

Edits in urls.py

Create App URLs: In the urls.py of the app (spareapp), define the URL patterns specific to the app's views. You can include the URL patterns of the app in the main project's urls.py using the include function.

In spareapp/urls.py

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

urlpatterns = [
    path('', views.index, name='index'),
    # Add more app-specific URL patterns here...
]
```

In spareparts/urls.py:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
      path('', include('spareapp.urls')),  # Include the app's URL patterns
]
```

Database Migration

Django's migration system is used to manage changes to the database schema. To create the necessary database tables and fields, perform the following steps:

Make Migrations:

- Make changes to your models in the models.py file
- Run the following command to generate migration files:

  python manage.py makemigrations

Apply Migrations:

After generating the migration files, run the following command to apply the changes to the database:

python manage.py migrate

Creating Super User

To access the Django admin interface and manage the application, you need to create a superuser account. Perform the following steps:

Create Super User:

- Run the following command and follow the prompts to create a superuser:

  python manage.py createsuperuser

Views

Views are Python functions that handle HTTP requests and return HTTP responses.

- index: Home page view function.
- all_spare_parts: View function to list all spare parts.
- add_spare_part: View function to add new spare parts.
- sale: View function to record sales.

Models

Django's Models represent the database tables and are used to define the data structure of the application.

- Category: Represents the categories of spare parts.
- Branch: Represents the branches of the company.
- SparePart: Represents the spare parts in the inventory.
- Buyer: Represents the buyers of spare parts.
- Sale: Represents the sales transactions.

Models.py

Models for spare parts inventory and sales records. Fields include category, part_name, part_number, date_of_arrival, price, total_quantity, country_of_origin, branch, etc.

Forms

Django's Forms are used to handle user input, validate data, and perform form processing.

- AddForm: Model form for adding new spare parts. Includes fields for part_name, part_number, date_of_arrival, price, total_quantity, country_of_origin, and branch.
- SaleForm: Model form for recording sales. Includes fields for quantity, amount_paid, buyer, and part_name.

Forms.py

Model forms for adding new spare parts and recording sales. Fields include part_name, part_number, date_of_arrival, price, quantity, amount_paid, buyer, part_name, etc.

User Authentication and Roles

Django's built-in authentication system is used to handle user authentication and login.

User authentication is enforced with @login_required decorator. Roles and permissions are set to restrict access to views.

Receipt Generation

Mechanism to generate receipts after sales, displaying sale details.

Search and Filtering

Implemented search and filtering functionality for spare parts and sales records.

Admin Panel

Django admin interface for manager to manage categories, branches, and other models.

Summary of the Commands ran in terminal in the time of development:

virtualenv venv
source venv/bin/activate
pip install django==3.0.8
django-admin startproject spareparts
python manage.py startapp spareapp
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
python manage.py runserver
deactivate
pip install django-crispy-forms

pip install crispy-bootstrap4
pip install django-bootstrap-v5