**💰 Django Project: Expense Tracker**

Managing personal finances efficiently has become increasingly important. Whether you're saving for a vacation, planning retirement, or staying within a budget — **tracking expenses is essential**.

Let’s build an **Expense Tracker** web application using Django to simplify this process.

**🧾 Project Overview**

The **Expense Tracker** app will have the following key features:

* 🔐 **User Authentication**: Register, login, logout, and manage accounts securely.
* 💸 **Expense Management**: Add, edit, delete, and view personal expenses.
* 🗂️ **Expense Categories**: Classify expenses (e.g., Food, Rent, Travel) for better analysis.
* 📱 **Responsive Design**: Compatible across devices — desktops, tablets, and phones.

**⚙️ Setting Up the Project**

**🔧 1. Environment Setup**

**✅ Install Python**

Make sure Python is installed on your system. Download it from [python.org](https://www.python.org/).

**✅ Install Django**

Use pip to install Django:

pip install django

**✅ Create Virtual Environment**

Isolate your project dependencies:

python -m venv myenv

Activate it:

cd myenv/Scripts/activate # Windows

OR

source myenv/bin/activate # macOS/Linux

**🏗️ 2. Create Django Project**

django-admin startproject expensetracker

cd expensetracker

**🗃️ 3. Initialize Git Repository**

git init

Create a .gitignore file to exclude:

\*.pyc

\_\_pycache\_\_/

myenv/

**🧩 4. Create Django App**

python manage.py startapp tracker

Add the app in settings.py:

INSTALLED\_APPS = [

...

'tracker',

]

**🌐 Version Control with Git**

**📝 Add Changes**

git add .

**✅ Commit Changes**

git commit -m "Initial project setup with Django Expense Tracker"

**☁️ Create Remote Repository**

Use platforms like **GitHub** or **GitLab**, then add the remote:

git remote add origin <your-repo-url>

**🚀 Push to Remote**

git push origin master

**🌿 Working with Branches**

**🌱 Create a Feature Branch**

git checkout -b feature/expenses

➡️ Replace feature/expenses with your feature name (e.g., feature/authentication).

**🔨 Implement Your Feature**

Write code, add templates, update models — develop your feature in this branch.

**📌 Stage & Commit**

git add .

git commit -m "Implemented expense management feature"

**☁️ Push the Feature Branch**

git push origin feature/expenses

**🔄 Switch Between Branches**

* To switch to main branch:

git checkout master

* To return to feature branch:

git checkout feature/expenses

**🔃 Merge or Rebase Changes**

**✅ Merge Feature to Master**

git checkout master

git merge feature/expenses

**🔁 Rebase Instead (Optional, Clean History)**

git checkout feature/expenses

git rebase master

**🎨 Designing the UI of the Expense Tracker**

To build a visually appealing and user-friendly **Expense Tracker**, we'll integrate a **pre-designed UI** from CodePen and wire it up with Django's backend logic.

**📌 Project Goal**

Integrate a designer-made HTML/CSS UI into your Django application so that you can:

* ✅ Focus on **functionality** while keeping the interface elegant
* ✅ Quickly prototype and test frontend-backend connections

**🪜 Step-by-Step Integration**

**🔧 Step 1: Setup Django Project**

Ensure you have completed:

* ✅ Installed Django
* ✅ Created a project and app
* ✅ Configured templates and static files

(Refer to the "Setting Up the Project" section earlier.)

**🗂️ Step 2: Add Template Folder**

Inside your Django app (e.g., tracker), structure your templates:

tracker/

├── templates/

│ └── index.html

Paste the HTML from CodePen into index.html.

**🌐 Example: index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Expense Tracker</title>

<link rel="stylesheet" href="/media/css/styles.css">

</head>

<body>

<h1>Expense Tracker</h1>

<!-- [UI content from CodePen goes here] -->

</body>

</html>

✅ Make sure the href points to the correct location of your static CSS file.

**🎨 Step 3: Add CSS to Static Directory**

Organize static assets like so:

public/

└── static/

└── css/

└── styles.css

Copy CodePen CSS into styles.css.

**⚙️ Step 4: Create View and URL**

**views.py**

from django.shortcuts import render

def index(request):

return render(request, 'index.html')

**tracker/urls.py**

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

]

**expensetracker/urls.py**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('tracker.urls')),

]

**📁 Step 5: Folder Structure for Static Files**

Terminal commands to create folders:

mkdir public

mkdir public/static

mkdir public/static/css

Place styles.css inside public/static/css.

**⚙️ Step 6: Static & Media Settings in settings.py**

import os

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'public/static')]

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'public/static')

**🌍 Step 7: Serve Static & Media in Development**

Update expensetracker/urls.py:

from django.conf import settings

from django.conf.urls.static import static

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

urlpatterns += staticfiles\_urlpatterns()

**🚀 Step 8: Run the Server**

Start your server:

python manage.py runserver

View the UI at:

http://127.0.0.1:8000/

**🧠 Final Structure Recap**

project\_root/

├── public/

│ └── static/

│ └── css/

│ └── styles.css

├── tracker/

│ └── templates/

│ └── index.html

├── tracker/

│ ├── views.py

│ ├── urls.py

├── expensetracker/

│ ├── urls.py

│ ├── settings.py