**💰 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

**💼 Expense Tracker Project – Database Modeling in Django**

In this project, we focus on accurately recording and managing users' expenses using Django's powerful ORM. The database schema revolves around three core models:

* CurrentBalance
* TrackingHistory
* RequestLogs

**🧱 Step 1: Define Django Models**

📄 **File**: models.py

from django.db import models

class CurrentBalance(models.Model):

current\_balance = models.FloatField(default=0)

def \_\_str\_\_(self):

return str(self.current\_balance)

class TrackingHistory(models.Model):

current\_balance = models.ForeignKey(CurrentBalance, on\_delete=models.CASCADE)

amount = models.FloatField()

expense\_type = models.CharField(choices=(('CREDIT', 'CREDIT'), ('DEBIT', 'DEBIT')), max\_length=200)

description = models.CharField(max\_length=200)

created\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"The amount is {self.amount} for {self.description} expense type is {self.expense\_type}"

class RequestLogs(models.Model):

request\_info = models.TextField()

request\_type = models.CharField(max\_length=100)

request\_method = models.CharField(max\_length=100)

created\_at = models.DateTimeField(auto\_now\_add=True)

**🧠 Step 2: Understanding the Models**

🔹 **CurrentBalance**

* Maintains the user's total balance.
* current\_balance is a FloatField.

🔹 **TrackingHistory**

* Keeps track of all income or expense entries.
* Fields:
  + current\_balance: ForeignKey to the CurrentBalance model
  + amount: Expense amount
  + expense\_type: CREDIT or DEBIT
  + description: Description of the expense/income
  + created\_at: Auto-updated timestamp

🔹 **RequestLogs**

* Stores all HTTP request logs for debugging or tracking purposes.
* Includes request content, type, method, and timestamp.

**⚙️ Step 3: Apply Migrations**

Use the following commands to apply the model changes to the database:

python manage.py makemigrations

python manage.py migrate

**🛠️ Step 4: Register Models in Admin Panel (Optional)**

📄 **File**: admin.py

from django.contrib import admin

from .models import CurrentBalance, TrackingHistory, RequestLogs

admin.site.register(CurrentBalance)

admin.site.register(TrackingHistory)

admin.site.register(RequestLogs)

🔐 This allows you to manage model records easily through the Django Admin UI.

**🧾 Step 5: Add Expense Form in Template**

📄 **File**: index.html

<!-- Expense Tracker Form -->

<form method="POST" id="form">

{% csrf\_token %}

<div class="form-control">

<label for="text">Description</label>

<input type="text" name="description" id="text" placeholder="Enter description..." />

</div>

<div class="form-control">

<label for="amount">Amount <br />

<small>(-100 = expense, 100 = income)</small></label>

<input type="number" name="amount" id="amount" placeholder="Enter amount..." />

</div>

<button class="btn" type="submit">Add transaction</button>

</form>

**🧠 Step 6: Define Views for Handling Expense Tracking**

📄 **File**: views.py

from django.shortcuts import render, redirect

from .models import \*

from django.contrib import messages

def index(request):

if request.method == "POST":

description = request.POST.get('description')

amount = request.POST.get('amount')

current\_balance, \_ = CurrentBalance.objects.get\_or\_create(id=1)

expense\_type = "CREDIT"

if float(amount) < 0:

expense\_type = "DEBIT"

if float(amount) == 0:

messages.success(request, "Amount cannot be zero")

return redirect('/')

tracking\_history = TrackingHistory.objects.create(

amount=amount,

expense\_type=expense\_type,

current\_balance=current\_balance,

description=description

)

current\_balance.current\_balance += float(tracking\_history.amount)

current\_balance.save()

print(description, amount)

return redirect('/')

return render(request, 'index.html')

**🖥️ Output in Terminal (Example)**

Groceries -100.0

Bonus 2500.0

This output confirms that the form data has been successfully processed and saved.

**🧾 Django Expense Tracker - Form Logic & UI Display**

**📌 Objective:**

Build an Expense Tracker in Django that can:

* Add new transactions (Credit/Debit)
* Display current balance, income, and expenses
* Show transaction history
* Delete transactions

**🏗️ Step 1: Handle Transaction Form Submission**

**🔧 Logic in views.py**

from django.shortcuts import render, redirect

from django.contrib import messages

from .models import CurrentBalance, TrackingHistory

def index(request):

if request.method == "POST":

description = request.POST.get('description')

amount = request.POST.get('amount')

current\_balance, \_ = CurrentBalance.objects.get\_or\_create(id=1)

expense\_type = "CREDIT"

if float(amount) < 0:

expense\_type = "DEBIT"

if float(amount) == 0:

messages.success(request, "Amount cannot be zero")

return redirect('/')

tracking\_history = TrackingHistory.objects.create(

amount=amount,

expense\_type=expense\_type,

current\_balance=current\_balance,

description=description

)

current\_balance.current\_balance += float(tracking\_history.amount)

current\_balance.save()

return redirect('/')

current\_balance, \_ = CurrentBalance.objects.get\_or\_create(id=1)

income = 0

expense = 0

for tracking\_history in TrackingHistory.objects.all():

if tracking\_history.expense\_type == "CREDIT":

income += tracking\_history.amount

else:

expense += tracking\_history.amount

context = {

'income': income,

'expense': expense,

'transactions': TrackingHistory.objects.all(),

'current\_balance': current\_balance

}

return render(request, 'index.html', context)

**🖼️ Step 2: Update the HTML Template (index.html)**

**✅ Add Transaction Form + Display History**

<!-- index.html -->

<h1>Expense Tracker</h1>

<div class="balance-container">

<h2>Your Balance</h2>

<h2>${{ current\_balance.current\_balance }}</h2>

</div>

<div class="inc-exp-container">

<div>

<h4>Income</h4>

<p class="money plus">+${{ income }}</p>

</div>

<div>

<h4>Expenses</h4>

<p class="money minus">-${{ expense }}</p>

</div>

</div>

<h3>History</h3>

<ul id="list" class="list">

{% for transaction in transactions %}

<li class="{% if transaction.expense\_type == 'DEBIT' %} minus {% else %} plus {% endif %}">

{{ transaction.description }} <span>{{ transaction.amount }}</span>

<a class="delete-btn" href="{% url 'delete\_transaction' transaction.id %}">

<i class="fas fa-trash-alt"></i>

</a>

</li>

{% endfor %}

</ul>

<h3>Add new transaction</h3>

<form method="POST">

{% csrf\_token %}

<div class="form-control">

<label for="text">Description</label>

<input type="text" name="description" placeholder="Enter description..." />

</div>

<div class="form-control">

<label for="amount">Amount</label>

<input type="number" name="amount" placeholder="Enter amount..." />

</div>

<button class="btn" type="submit">Add transaction</button>

</form>

**🗑️ Step 3: Handle Delete Transactions**

**🧠 Logic in views.py**

def delete\_transaction(request, id):

tracking\_history = TrackingHistory.objects.filter(id=id)

if tracking\_history.exists():

current\_balance, \_ = CurrentBalance.objects.get\_or\_create(id=1)

transaction = tracking\_history[0]

current\_balance.current\_balance -= transaction.amount

current\_balance.save()

tracking\_history.delete()

return redirect('/')

**🛣️ Step 4: Add Delete URL in urls.py**

from django.urls import path

from . import views

urlpatterns = [

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

path('delete-transaction/<id>/', views.delete\_transaction, name='delete\_transaction'),

]