PROJECT REPORT ON EXPENSE TRACKER WEB APPLICATION USING DJANGO

DIPLOMA IN COMPUTER ENGINEERING

TEAM MEMBER'S:

1.	Angothu Adhisheshu	(21007-CS-053)
2.	V Nikhil Kumar Reddy	(21007-CS-033)

ACKNOWLEDGEMENT

We wish to express my healthy gratitude to **Dr.k.Chandra Shekar** garu **M.Tech**, principal secretary and correspondent of **KDR GOVT POLYTECHNIC, Wanaparthy** for providing all the facilities required for the Industrial Training.

We wish to take this opportunity to express our sincere thanks to **Smt. B.Arunima mam**, H.O.D of Computer Science Department for the interest technical support and suggestions during the seminar leading to our Success.

We are obliged and great full to my guide **Venkatesh sir and Uday sir** of CSE department for his valuable suggestions and sagacious guidance in all respects during the technical of our training. We hereby express my sincere thanks regards to all teaching staff, parents and friends who extend their help hand in the accomplished of the Industrial Training.

DECLARATION

I hereby declare that the work which is being presented in this Industrial Training Report titled "**DIPLOMA INDUSTRIAL TRAINING**" towards the partial fulfilment for the award of **DIPLOMA IN COMPUTER ENGINEERING** is an authentic record of my own work carried out under the Guidance of **S. KAUSHIK** Manager of **National Small Industries Corporation (NSIC), ECIL, Hyderabad** for his valuable suggestions.

The matter embodied here, has been submitted by me for the award of any other Degree or Diploma.

ANGOTHU ADHISHESHU
(21007-CS-053)

V NIKHIL KUMAR REDDY

(21007-CS-033)

CONTEXT

Expense Tracker web Application Using Django

Abstract:

- Expense Tracker is a day-to-day expense management system designed to easily and efficiently track the daily expenses of unpaid and unpaid staff through a computerized system that eliminates the need for manual paper tasks that systematically maintains records and easily accesses data stored by the user.

- Expense Tracker is a software application that helps users keep track of their expenses. The application provides a user interface to add, view and delete

- An expense tracker is a software application or tool designed to help individuals or businesses monitor, record, and manage their financial transactions and expenditures. This tool serves as a digital ledger, allowing users to keep a detailed account of their spending habits, budgets, and overall financial health.

- This practical demonstration will help you to understand how to implement CRUD (CREATE, READ, UPDATE and DELETE) operations functionality in Python with DBSQLITE3 database. For any application which is associated with a database, we perform some operations for inserting the record [C= Create], reading the records [R=Read], updating the existing record with new value [U=Update] or deleting the record from the database [D=Delete]. So, These four operations are essentials when talking about any application where we have to use a database.

INTRODUCTION

In this python django project, we will create an expense tracker that will take details of our expenses. While filling the signup form a person will also need to fill in the details about the income and the amount he/she wants to save.

Some people earn on a daily basis, so their income can also be added on a regular basis. Details of expenses will be shown in the form of a pie chart on a weekly, monthly, and yearly basis. Installation of django is a must to start with the Expense Tracker project.

EXISTING SYSTEM:

- The purpose of writing this is for beginners who are curious about backend development or front-end developers who want to learn database technology with server-side programming language.
- As we know that server-side language like python, java, and many others are not sufficient for backed developers and even a backend developer needs to be more knowledgeable in database technologies.
- So let’s start with the basics of SQLite Database with python.
- I chose the SQLite database is just an option, one can apply the same knowledge with any other database also like Mysql and Oracle or any other. the best part of database technology is all the databases are very similar for SQL concepts accept few new Databases.

PROPOSED SYSTEM:

- The Modern Expense Tracker System is a comprehensive financial management solution designed to help users effortlessly manage their expenses, budgets, and financial goals. The system leverages cutting-edge technologies to provide an intuitive and efficient user experience.

- Enable users to log, categorize, and track their expenses seamlessly.

- Provide insights into spending patterns through visual analytics and reports.

- Secure user accounts with authentication mechanisms. Define user roles for personalized access.

- Log individual expenses with details such as description, amount, and category.

- Intuitive and responsive design for a seamless user experience. Accessibility across multiple devices, including desktop and mobile.

- The proposed system will follow a Model-View-Controller (MVC) architecture using the Django web framework.

- The Modern Expense Tracker System aims to provide users with a powerful and user-friendly tool for managing their finances. With its robust features, secure architecture, and future expansion possibilities, this system is poised to become an essential tool for individuals seeking financial wellness.

- ❖ We are not using any python package to make our job easy for learning these four basic operations.

CRUD

C: Create

R: Read

U: Update

D: Delete

Software Requirements:

- ➢ SOFTWARE – PYTHON
- ➢ IDLE – VISUAL STUDIO OR PYCHARM
- ➢ DATABASE - DBSQLITE3
- ➢ LANGUAGES - HTML, CSS, FONTS
- ➢ OPERATING SYSTEM - WINDOWS 11

Hardware Requirements:

- ➢ Intel i5 or equivalent processor
- ➢ Minimum 4GB RAM
- ➢ 500GB SSD storage
- ➢ Ethernet or Wi-Fi connectivity

Python

Certainly! Python is a high-level, interpreted, and general-purpose programming language. Here's some comprehensive information about Python.

History:

- Python was created by Guido van Rossum and was first released in 1991.
- It is influenced by languages such as ABC, Modula-3, C, C++, and other scripting languages.

Key Features:

- ❖ **Readable and Expressive Syntax:**
  - ➢ Python emphasizes code readability and allows developers to express concepts in fewer lines of code than might be possible in languages like C++ or Java.

- ❖ **Interpreted and Interactive:**
  - ➢ Python is an interpreted language, which means that the source code can be executed directly without the need for compilation.

  - ➢ It also supports an interactive mode, allowing for testing and debugging of snippets of code.

- ❖ **Dynamically Typed:**
  - ➢ Python is dynamically typed, meaning that the data type of a variable is interpreted at runtime.

- ❖ **Object-Oriented:**
    - ➢ Python supports both procedural and object-oriented programming paradigms.

- ❖ **Extensive Standard Library:**
    - ➢ Python comes with a large standard library that supports many common programming tasks, such as connecting to web servers, reading/writing files, and more.

- ❖ **Community and Ecosystem:**
    - ➢ Python has a large and active community. The Python Package Index (PYPI) hosts a vast array of third-party libraries and frameworks that extend the capabilities of Python.

Use Cases:

- ❖ **Web Development:**
  Popular frameworks like Django and Flask are used for building web applications.

- ❖ **Data Science and Machine Learning:**
  Python is widely used for data analysis and machine learning. Libraries like NumPy, Pandas, and scikit-learn are essential tools in this domain.

- ❖ **Automation and Scripting:**
  Python is commonly used for scripting and automation tasks due to its simplicity and readability.

❖ **Artificial Intelligence:**
Python is a preferred language for AI development. Libraries like TensorFlow and PY Torch are extensively used for deep learning.

❖ **Scientific Computing:**
Python is used in scientific research and engineering for tasks such as simulations and data analysis.

❖ **Desktop GUI Applications:**
With libraries like Tkinter, PyQt, and wxPython, Python is used for building desktop applications.

Versions:

❖ **Python 2 vs. Python 3:**

- Python 2 was the legacy version and reached its end of life in 2020.
- Python 3 is the current and future version, with ongoing updates and improvements.

Development Environment:

❖ **IDEs (Integrated Development Environments):**

- Popular IDEs for Python include PyCharm, VSCode, Jupyter Notebooks, and IDLE.

❖ **Virtual Environments:**

- Python developers often use virtual environments to manage project dependencies and isolate project environments.

Django

Django is a high-level web framework for building web applications using the Python programming language. It follows the model-view-controller (MVC) architectural pattern and encourages the "Don't Repeat Yourself" (DRY) principle.

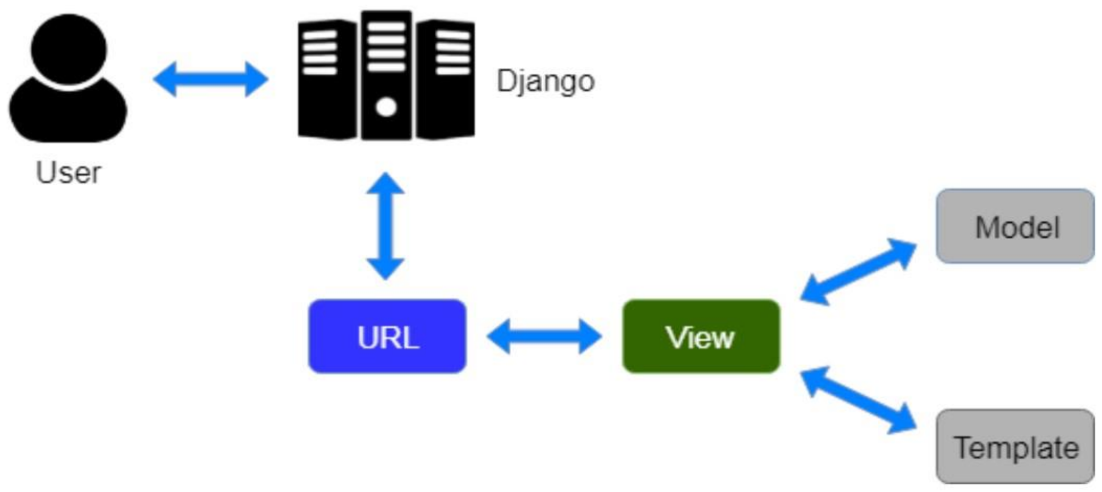
Django provides a set of tools and conventions for handling common web development tasks, such as database interaction, URL routing, and form handling, making it easier for developers to build robust and scalable web applications.

It also includes an administration panel for managing the application's data and a built-in authentication system. Overall, Django aims to simplify and streamline the web development process.

- Django makes it easier to built better web apps more quickly and with less code.

- Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. Its free and open source.

Django offers several advantages that make it a popular choice for web development:



Advantages:

- ❖ **Rapid Development:** Django provides a high-level, clean, and pragmatic design that allows for rapid development. Its built-in features and conventions help developers focus on building functionalities rather than dealing with repetitive tasks.

- ❖ **ORM (Object-Relational Mapping):** Django's ORM system allows developers to interact with databases using Python objects, abstracting away the complexity of SQL queries. This simplifies database operations and promotes code reusability.

- ❖ **Built-in Admin Panel:** Django includes a powerful and customizable admin interface, which automatically generates an admin panel for managing database records. This is a significant time-saver for developers during the development phase.

- ❖ **Security Features:** Django comes with built-in security features to help developers build secure web applications. It includes protection against common web vulnerabilities, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

- **Community and Documentation:** Django has a large and active community, which means there's a wealth of resources, tutorials, and third-party packages available. The official documentation is comprehensive and regularly updated.

- **URL Routing and Middleware:** Django's URL routing system is flexible and allows for clean and organized URL patterns. Middleware support enables developers to process requests globally before they reach the application's views.

- **Third-party Packages:** The Django ecosystem has a rich collection of third-party packages that extend its functionality. These packages cover a wide range of features, from authentication and authorization to API development and more.

These advantages collectively contribute to Django's popularity and its effectiveness in building robust and scalable web applications.

Disadvantages:

- **Learning Curve:** Django has a steeper learning curve, especially for beginners in web development. The framework comes with many features and abstractions, which might be overwhelming for newcomers.

- **Size of Codebase:** Django projects can have a larger codebase compared to simpler frameworks. This might be seen as a disadvantage for small projects or applications with minimal requirements.

- **Overhead for Simple Projects:** For small or straightforward projects, Django's full-stack capabilities may introduce unnecessary complexity. Simpler frameworks might be more suitable for such cases.

- **ORM Limitations:** While Django's Object-Relational Mapping (ORM) is powerful, it may not be as flexible as raw SQL in certain situations. Some developers prefer more control over database queries.

- **Version Compatibility:** Upgrading to newer versions of Django might introduce compatibility issues with existing projects or third-party packages. This can be a challenge, particularly for long-term projects.

- **Resource Intensive:** Django may be considered resource-intensive for simple projects or applications with low traffic. For scenarios where resource efficiency is critical, a more lightweight framework might be preferable.

- **JavaScript Integration:** While Django has improved its support for modern JavaScript frameworks, it might not be the first choice for developers who want to build highly interactive, single-page applications with a front-end framework like React or Vue.js.

FRONTEND

FRONT END -- TEMPLATES (html file-base, index, edit)

BUSINESS LOGIC -- VIEWS (functions-index, create, edit, update, delete)

BACK END -- MODEL (database-table-member)

HTML (Hypertext Markup Language):

1. **Purpose:**
   - Used for structuring content on the web.
   - Defines the structure of web pages using a markup language.

2. **Syntax:**
   - Consists of tags enclosed in angle brackets.
   - Tags define elements such as headings, paragraphs, links, images, etc.

CSS (Cascading Style Sheets):

1. **Purpose:**
   - Used for styling and layout of web pages.
   - Separates content from presentation.

2. **Syntax:**
   - Consists of selectors and declarations.
   - Selectors target HTML elements, and declarations define the style.

JavaScript:

1. **Purpose:**
    - Programming language for making web pages interactive.
    - Supported by all major web browsers.

2. **Syntax:**
    - C-style syntax with object-oriented capabilities.
    - Can be embedded directly into HTML.

Bootstrap:

1. **Purpose:**
    - Front-end framework for building responsive and mobile-first web pages.
    - Developed by Twitter.

2. **Features:**
    - Consists of HTML, CSS, and JavaScript components.
    - Provides a grid system, navigation bars, modals, and other UI components.

3. **Usage:**
    - Easily integrates with existing web projects.
    - Helps in creating consistent and visually appealing designs.

Database:

Django, by default, is configured to use the SQLite database as its default database engine when you start a new project. This is set up in the **settings.py** file of the Django project. The **DATABASES** setting in this file contains the configuration for the default database.

Here's a breakdown of the default database settings in Django:

1. **Database Engine (ENGINE):** The default engine for Django is set to **'django.db.backends.sqlite3'**, indicating the use of the SQLite database engine.

2. **Database Name (NAME):** The **NAME** parameter specifies the name of the SQLite database file. By default, it is set to **'db.sqlite3'**. The database file is created in the project's root directory.

3. **Other Optional Parameters:** The **USER**, **PASSWORD**, **HOST**, and **PORT** parameters are not required for SQLite since it is a serverless and file-based database. For other database engines, these parameters would be relevant.

- ✓ This configuration uses the **sqlite3** engine and specifies the database file as **'db.sqlite3'** in the project's base directory.

- ❖ It's important to note that while SQLite is the default, Django is highly flexible and supports various other database engines such as PostgreSQL, MySQL, Oracle, and more. Developers can easily switch to a different database engine by modifying the **DATABASES** setting in **settings.py**. For example, switching to PostgreSQL might involve changing the **engine** to **'django.db.backends.postgresql'** and configuring the appropriate database connection details.

Expense Tracker

Python Expense Tracker Project

In this python django project, we will create an expense tracker that will take details of our expenses. While filling the signup form a person will also need to fill in the details about the income and the amount he/she wants to save. Some people earn on a daily basis, so their income can also be added on a regular basis. Details of expenses will be shown in the form of a pie chart on a weekly, monthly, and yearly basis. Installation of django is a must to start with the Expense Tracker project.

Project Prerequisites

Sound knowledge of django framework, HTML, CSS, JavaScript and python is required before starting this Expense Tracker project of Python.

Project File Structure

1) Install django framework
2) Create a project and an app
3) Settings.py
4) Models.py
5) Admin.py
6) Urls.py
7) App.py
8) Forms.py
9) Views.py

Block Diagram

```
● ExpenseTracker
|   +---● ExpenseTracker (project folder)
|   |--● _pycache_
|   |--wsgi.py
|   |--settings.py
|   |--urls.py
|   |--asgi.py
|   |
|   +---● home (app folder)
|   |-- ● migrations (includes files related to migrations)
|   |--admin.py
|   |-- models.py
|   |-- forms.py
|   |-- urls.py
|   |-- views.py
|   |--app.py
|   |--test.py
|   |
|   +--● static
|   |   |--images
|   |   |--JavaScript's (script files can be create)
|   |   |--CSS (CSS files can be create)
|   |
|   +--● templates
|  |--● home (html files can be create)
|  |  |--base.html
|  |  |--register.html
|  |  |--login.html
|  |
|  +-- ●db.sqlite3
|  +-- ●manage.py
```

1. Install django framework:

To begin with the project, you need to install django on your system. To install django, write the following command on cmd or terminal window.

```
Pip install django
```

2. Create a project and an app:

We will create a new project named Expense Tracker and an app to start the project. Write the following command on the terminal window.

```
django-admin startproject ExpenseTracker
python manage.py startapp home
python manage.py runserver
```

Create a template and static folder to store your files. Template folder will contain all the html files. Static folder will contain all the CSS files, images and JavaScript files.

3.Settings.py

The term **project** describes a Django web application. The project Python package is defined primarily by a settings module, but it usually contains other things. For example, when you run **django-admin startproject Expense Tracker** you'll get a **Expense Tracker** project directory that contains a **Expense Tracker** Python package with **settings.py**, **urls.py**, **asgi.py** and **wsgi.py**. The project package is often extended to include things like fixtures, CSS, and templates which aren't tied to a particular application.

A **project's root directory** (the one that contains **manage.py**) is usually the container for all of a project's applications which aren't installed separately.

The term **application** describes a Python package that provides some set of features. Applications may be reused in various projects.

Applications include some combination of models, views, templates, template tags, static files, URLs, middleware, etc. They're generally wired into projects with the **INSTALLED_APPS** setting and optionally with other mechanisms such as URLconfs, the **MIDDLEWARE** setting, or template inheritance.

It is important to understand that a Django application is a set of code that interacts with various parts of the framework. There's no such thing as an **Application** object. However, there's a few places where Django needs to interact with installed applications, mainly for configuration and also for introspection. That's why the application registry maintains metadata in an **AppConfig** instance for each installed application.

There’s no restriction that a project package can’t also be considered an application and have models, etc. (which would require adding it to **INSTALLED_APPS**).

Configuring applications

To configure an application, create an **apps.py** module inside the application, then define a subclass of **AppConfig** there.

When **INSTALLED_APPS** contains the dotted path to an application module, by default, if Django finds exactly one **AppConfig** subclass in the **apps.py** submodule, it uses that configuration for the application. This behavior may be disabled by setting **AppConfig.default** to **False**.

If the **apps.py** module contains more than one **AppConfig** subclass, Django will look for a single one where **AppConfig.default** is **True**.

If no **AppConfig** subclass is found, the base **AppConfig** class will be used.

Alternatively, **INSTALLED_APPS** may contain the dotted path to a configuration class to specify it explicitly:

Code of Settings.py

```
"""
Django settings for ExpenseTracker project.

Generated by 'django-admin startproject' using Django 3.0.5.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""
```

```python
from pathlib import Path
import os
from django.contrib.messages import constants as messages
from django.core import mail

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '4--(d0^o%3vqt#-c(hf+8)a$95z8gbo57xol5pft!%xpve9_zd'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'home.apps.HomeConfig',

]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'ExpenseTracker.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'ExpenseTracker.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
STATICFILES_DIRS=[os.path.join(BASE_DIR,"static"),]

MESSAGE_TAGS = {
    messages.ERROR:"danger"
}
#SMTP Configuration

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''

MEDIA_ROOT = os.path.join(BASE_DIR,'static/img')
MEDIA_URL = '/img/'
```

4. Models.py

Database connectivity is done with the help of models.py. Create the following models in models.py file in the app of your project.

In Django, the **models.py** file plays a crucial role in defining the data models for your application. The models you define in this file represent the structure of your database tables. Django's Object-Relational Mapping (ORM) system allows you to interact with the database using Python objects, making database operations more Pythonic and abstracting away the underlying SQL queries.

Here are the key aspects of **models.py** in Django:

1. **Defining Models:**
   - Models are Python classes that subclass **django.db.models.Model**.
   - Each class attribute represents a database field.

2. **Field Types:**
   - Django provides various field types such as **Char Field**, **Integer Field**, **DateField**, **Foreign Key**, etc.
   - Field types define the type of data that can be stored in a field.

3. **Primary Keys and Relationships:**
   - The **id** field is automatically added to every model as a primary key.
   - Relationships between models are defined using fields like **ForeignKey** and **ManyToMany Field**.

4. **Model Methods:**
   - You can define methods within your models to perform specific actions related to the model.

5. **Django Admin Integration:**
   - Models defined in **models.py** are used by the Django Admin interface, allowing you to manage and interact with your database through a user-friendly web interface.
   - Registering models in the **admin.py** file enables their visibility in the Django Admin.

6. **Database Migrations:**
   - After defining or modifying models, you need to create and apply migrations to update the database schema.
   - Use commands like **python manage.py makemigrations** and **python manage.py migrate**.

The **models.py** file serves as the blueprint for your database schema. It's where you define the data structures that your application will interact with, and it's a fundamental part of the Django ORM system.

Code of Model.py

```
from django.db import models
from django.utils.timezone import now
from django.contrib.auth.models import User
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.db.models import Sum
#Create your models here.
SELECT_CATEGORY_CHOICES = [
    ("Food","Food"),
    ("Travel","Travel"),
    ("Shopping","Shopping"),
    ("Necessities","Necessities"),
    ("Entertainment","Entertainment"),
    ("Other","Other")
 ]
ADD_EXPENSE_CHOICES = [
     ("Expense","Expense"),
     ("Income","Income")
 ]
PROFESSION_CHOICES =[
    ("Employee","Employee"),
    ("Business","Business"),
    ("Student","Student"),
    ("Other","Other")
]
class Addmoney_info(models.Model):
    user = models.ForeignKey(User,default = 1, on_delete=models.CASCADE)
    add_money = models.CharField(max_length = 10 , choices = ADD_EXPENSE_CHOICES )
    quantity = models.BigIntegerField()
    Date = models.DateField(default = now)
    Category = models.CharField( max_length = 20, choices = SELECT_CATEGORY_CHOICES
, default ='Food')
    class Meta:
        db_table:'addmoney'

class UserProfile(models.Model):
    user = models.OneToOneField(User,on_delete=models.CASCADE)
    profession = models.CharField(max_length = 10, choices=PROFESSION_CHOICES)
    Savings = models.IntegerField( null=True, blank=True)
    income = models.BigIntegerField(null=True, blank=True)
    image = models.ImageField(upload_to='profile_image',blank=True)
    def __str__(self):
       return self.user.username
```

5. Admin.py

It will help register the tables in the database.

In Django, the **admin.py** file is used to configure the Django Admin interface. The Django Admin is a built-in, powerful, and customizable web-based interface that allows administrators and developers to manage and interact with the data stored in the database. By creating entries in the **admin.py** file, you can control how your models are displayed and manipulated in the admin interface.

Here's what you can do in **admin.py**:

1. **Register Models:**
   - Use the **admin.site.register()** function to register your models. This makes them accessible in the Django Admin interface.

2. **Customize Model Display:**
   - You can customize how your model instances are displayed in the admin interface by creating a custom **Model admin** class.

3. **Filtering and Search:**
    - You can enable filtering and search capabilities for your models.

4. **Actions:**
    - Define custom actions that can be performed on selected objects in the admin interface.

5. **Inline Models:**
    - Display related models as inline forms within the parent model's detail view.

6. **Customizing Detail Views:**
    - Customize the detail views of your models in the admin interface.

7. **Overriding Templates:**
    - You can override the default templates used by the Django Admin to change the look and feel of the interface.

The **admin.py** file provides a way to tailor the Django Admin to the specific needs of your application. By customizing the display, behaviour, and functionality of your models in the admin interface, you can create a more user-friendly and efficient experience for those managing the application.

Code of Admin.py

```python
from django.contrib import admin
# Register your models here.
from .models import Addmoney_info

class Addmoney_infoAdmin(admin.ModelAdmin):
    list_display=("user","quantity","Date","Category","add_money")
admin.site.register(Addmoney_info,Addmoney_infoAdmin)
```

To store these models in the database, run the following command:

```
python manage.py make migrations
python manage.py migrate
```

For accessing the database, create the superuser. To create a superuser run the following command on your terminal window.

```
python manage.py createsuperuser
```

6. Urls.py

In Django, the **urls.py** file is used to define the URL patterns for your web application. It plays a crucial role in mapping URLs to views and helps in organizing the structure of your web application. The **urls.py** file is typically found in each Django app and may also exist in the project's main directory.

Here are the key aspects of the **urls.py** file:

1. **URL Patterns:**
   - The primary purpose of **urls.py** is to define URL patterns. Each pattern is associated with a specific view function or another URL pattern.

2. **Path Function:**
   - The **path()** function is commonly used to define URL patterns. It takes a route string and associates it with a view function.

- The **name** parameter is used to create a reference to the URL pattern, making it easier to refer to in templates or code.

3. **Regular Expressions with re_path:**
   - For more complex URL patterns, you can use the **re_path()** function, which allows you to use regular expressions in your URL patterns.

4. **Include Function:**
   - The **include()** function is used to include other URL patterns from different modules or apps.

5. **Namespaces:**
   - Namespaces are used to avoid conflicts when including URLs from different apps. They are defined in the **app_name** attribute in the **urls.py** file of the app.

6. **Passing Parameters to Views:**
   - You can capture parts of the URL and pass them as parameters to your views using angle brackets <>.

7. **View Functions:**
   - Views are Python functions or classes that handle HTTP requests and return HTTP responses.

8. **URL Reverse Resolution:**
   - The **reverse()** function in Django is used to generate URLs from URL patterns by using their names.

9. **Middleware and Decorators:**
   - You can use middleware or decorators to apply logic or checks to certain URL patterns or views.

10. Namespacing and App Structure:

- Organizing your URL patterns using name spacing and separating them into individual apps helps maintain a clean and modular project structure.
-

Code of Urls.py

```
from django.contrib import admin
from django.urls import path
from django.urls import include
from . import views
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('', views.home, name='home'),
    path('index/', views.index, name='index'),
    path('register/',views.register,name='register'),
    path('handleSignup/',views.handleSignup,name='handleSignup'),
    path('handlelogin/',views.handlelogin,name='handlelogin'),
    path('handleLogout/',views.handleLogout,name='handleLogout'),
    path('reset_password/',auth_views.PasswordResetView.as_view(template_name =
"home/reset_password.html"),name='reset_password'),
    path('reset_password_sent/',auth_views.PasswordResetDoneView.as_view(template_na
me="home/reset_password_sent.html"),name='password_reset_done'),
    path('reset/<uidb64>/<token>/',auth_views.PasswordResetConfirmView.as_view(templ
ate_name ="home/password_reset_form.html"),name='password_reset_confirm'),
    path('reset_password_complete/',auth_views.PasswordResetView.as_view(template_na
me ="home/password_reset_done.html"),name='password_reset_complete'),
    path('addmoney/',views.addmoney,name='addmoney'),
    path('addmoney_submission/',views.addmoney_submission,name='addmoney_submission'
),
    path('charts/',views.charts,name='charts'),
    path('tables/',views.tables,name='tables'),
    path('expense_edit/<int:id>',views.expense_edit,name='expense_edit'),
    path('<int:id>/addmoney_update/', views.addmoney_update, name="addmoney_update")
,
    path('expense_delete/<int:id>',views.expense_delete,name='expense_delete'),
    path('profile/',views.profile,name = 'profile'),
    path('expense_month/',views.expense_month, name = 'expense_month'),
    path('stats/',views.stats, name = 'stats'),
    path('expense_week/',views.expense_week, name = 'expense_week'),
    path('weekly/',views.weekly, name = 'weekly'),
    path('check/',views.check,name="check"),
    path('search/',views.search,name="search"),
    path('<int:id>/profile_edit/',views.profile_edit,name="profile_edit"),
    path('<int:id>/profile_update/',views.profile_update,name="profile_update"),
    path('info/',views.info,name="info"),
    path('info_year/',views.info_year,name="info_year"),
    path('layout2/',views.layout2, name='layout2'),
    path('layout3/',views.layout3, name='layout3'),
    path('page1/',views.page1, name='page1'),
    path('page2/',views.page2, name='page2'),
    path('settings/',views.settings, name='settings'),
    path('activity/',views.activity, name='activity'),
]
```

By defining URL patterns in the **urls.py** file, you establish the routing for your Django application, connecting URLs to views and creating a logical structure for navigating through your web application.

7.App.py

In a standard Django project created using **startapp**, the **app.py** file is not a default or required file. Instead, Django apps typically have an **apps.py** file that contains configuration settings for the app. The main work of **apps.py** is to define an **AppConfig** class and specify various settings for the app. Here's a typical structure:

```
from django.apps import AppConfig

class HomeConfig(AppConfig):
    name = 'home'
```

1. **Custom Initialization or Configuration:** Developers might choose to create an **app.py** file to perform custom initialization or configuration when the app is loaded. This could include setting up global variables, connecting to external services, or performing other setup tasks.

2. **Utility Functions:** It could contain utility functions that are used across multiple modules within the app.

3. **Middleware Configuration:** In some cases, developers might use **app.py** to configure middleware for the app.

4. **Signals and Event Handling:** If your app needs to handle signals or events during the lifecycle of the Django application, developers might use **app.py** for setting up event handlers.

8.Forms.py

In a Django application, the **forms.py** file is where you define your forms. Forms are a crucial part of web development, allowing you to handle user input, validate it, and interact with your models or perform other actions. Here are the main works or purposes of the **forms.py** file:

```python
from django import forms
from .models import Expense

class ExpenseForm(forms.ModelForm):
    class Meta:
        model = Expense
        fields = ['amount', 'category', 'description', 'date']
```

1) Defining Forms

2) Validation

3) Rendering HTML Forms

4) Handling Form Submissions

5) Custom Form Logic

9. Views.py

In Django, the **views.py** file is where you define the logic for processing HTTP requests and returning HTTP responses. Views handle user input, interact with the database if needed, and render templates to produce the final HTML that is sent to the client's browser. The **views.py** file is a critical component of the Model-View-Controller (MVC) architectural pattern used by Django.

Here are the key aspects of the **views.py** file:

1. **View Functions:**
    - Views in Django are typically implemented as Python functions.

2. **Request and Response Objects:**
    - The **request** object represents the incoming HTTP request, and the view function uses it to access parameters, headers, and other information.
    - The view function returns an HTTP response object, which can be an HTML page, a redirect, or other types of responses.

3. **Render HTML Templates:**
    - Views often use the **render** function to generate HTML content from templates.

4. **Business Logic:**
    - Views can contain the application's business logic, which involves processing data, interacting with the database using models, and preparing the context for rendering templates.

5. **Context Data:**
   - The **context** parameter in the **render** function is used to pass data to the template.

6. **Class-Based Views (CBVs):**
   - Django also supports class-based views, which provide a more object-oriented approach to organizing view logic.

7. **Middleware and Decorators:**
   - Middleware and decorators can be used to add additional functionality to views, such as authentication checks, logging, or custom processing.

8. **Asynchronous Views:**
   - Django supports asynchronous views using the **async def** syntax for handling asynchronous tasks.

9. **RESTful Views:**
   - For building APIs, Django provides additional tools such as the Django REST framework, which introduces serializers and class-based views tailored for RESTful interactions.

❖ By defining views in **views.py**, you specify how your application responds to different URL patterns, handle user input, and dictate the logic for generating the appropriate HTTP responses. This separation of concerns enhances code organization and maintainability in your Django project.

a. Importing modules:

- <u>Render:</u> It returns the HTTP response object and combines the template with the dictionary that is mentioned in it.

- <u>HTTP Response:</u> It displays a text response to the user.
- <u>Redirect:</u> It redirects the user to the specified URL.
- <u>Messages:</u> It helps to store and display messages to the user on the screen.
- <u>Authenticate:</u> It verifies the user.
- <u>User:</u> This model handles authentication as well as authorization.
- <u>Session:</u> It helps the user to access only their data. Without sessions, every user's data will be displayed to the user.
- <u>Paginator:</u> It is used to manage paginated data.
- <u>datetime:</u> It is used to get the current date and time.

b. Login and Index function:

➢ home() is a function that allows the user to access the dashboard once the user is logged in. index() function contains the backend of the dashboard page.

- filter(): Query set is filtered by filter().
- get(): Single unique object can be obtained with get().
- order by(): It orders the query set.

c. Other Functions:

➢ The first function redirects the user to the page where we can enter our expenses and income. Profile() function redirects the user to the profile page where information of the user is displayed. Profile_edit() redirects to the page where information of the user can be edited. These pages can only be accessed if the user is logged.

d. Updating Profile:

- Profile update() function performs the backend of the edit profile form. User.objects.get() gets all the information of the user then all the updated information is saved again. This function is performed by save().

e. Signup, Login, and Logout backend:

- Handle signup() function handles the backend of signup form. Uname, fname, lname, email , pass1, pass2, income, savings and profession will store the information of the form in these variables.

- Various conditions are there to sign up . The username should be unique, pass1 and pass 2 should be the same and also the length of the username should be maximum 15 characters. Handle Login() handles the backend of the login page. If the information entered by the user is correct, the user will be redirected to the dashboard. handle Logout() handles the backend of logout.

- error(): This function gives the error message on the screen if a condition is not satisfied.
- Len():This function returns the length of the string, array, dictionary etc.
- success():If a condition is satisfied, it displays the message that is specified in the parentheses.

f. Add Money Form and Add Money Update Backend:

- Addmoney_submission() handles the backend of the form we filled for our daily expenses. addmoney_update() saves the information of the form after we have edited .

g. Expense Edit and Expense Delete Backend:

- ➢ Expense_edit() form redirects the user to the edit form and also extracts the details of the user from the database and displays it on the screen. expense_delete() helps in deleting the expenses.

h. Monthly, weekly , yearly expense Backend:

- ➢ Expense month() function gets the data of the expenses of the current month. Get category() function gets the category (expense/income) from the database. Get expense category amount() fetches the amount from the database of the category(expense). stats() function calculates the overall expenses and savings made by the user in a month. expense week() and info year() performs the same function as expense month() but on a weekly basis. weekly() gets the amount saved in a month and also the overall expenses of a user.

<u>COMMANDS TO RUN DJANGO PROJECT:</u>

- ❖ <u>Python manage.py runserver</u> - We are running Django server using manage.py module with python language.
  Localhost i.e., **127.0.0.1:8000** it will open at Web Browser-html page (Home Page).

- ✓ *<u>http://127.0.0.1:8000</u>* – 127.0.0.1 – Local Host 8000 – Port Number
  Deployment- (online) – domain name: Web Server – example: facebook.com, amazon.in, etc.

❖ url(r’^$’, views.index_redirect, name=’index redirect’),

r-direct, ^ - starting point, $ - ending point, in the views module we are calling index_redirect function, name of the function is index_redirect.

http://127.0.0.1:8000/^$ - views.index_redirect

example: > math.pi – math – module and pi – function

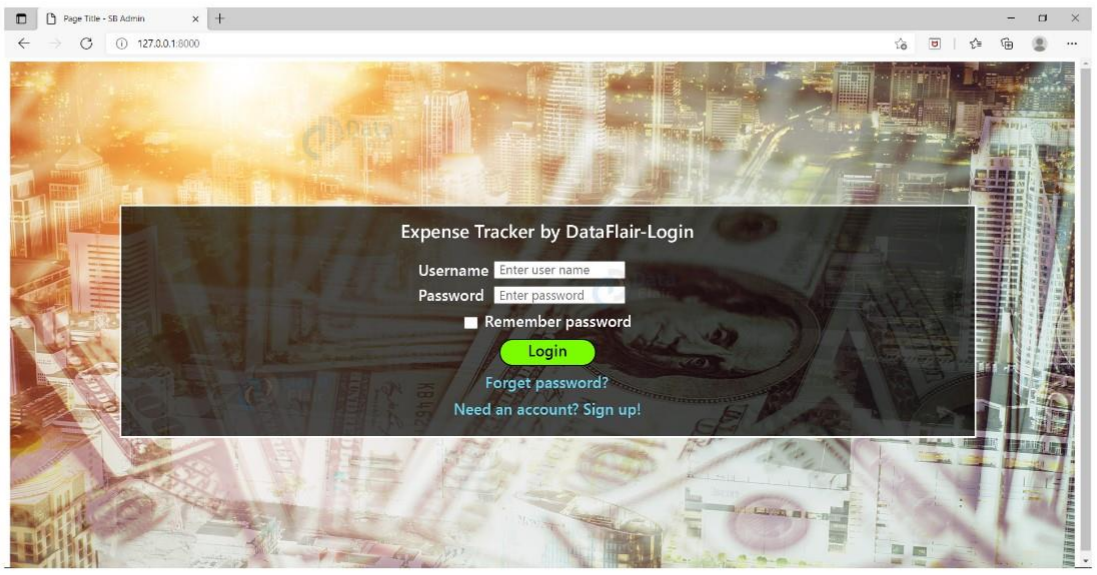
url(r’^crud/’, include(‘crud.urls’)), - http://127.0.0.1:8000/crud/ now the cursor is in the crud app in urls of the above line.

url(r’^admin/’ admin.site.urls),

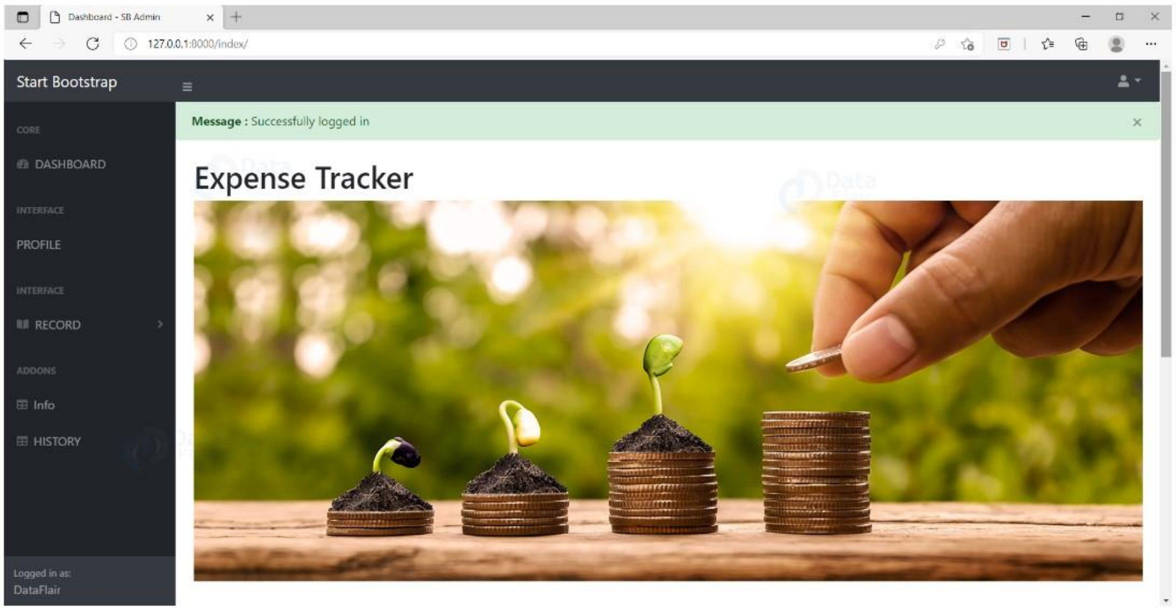
OUTPUTS:

Python Expense Tracker Output:

Login Form:

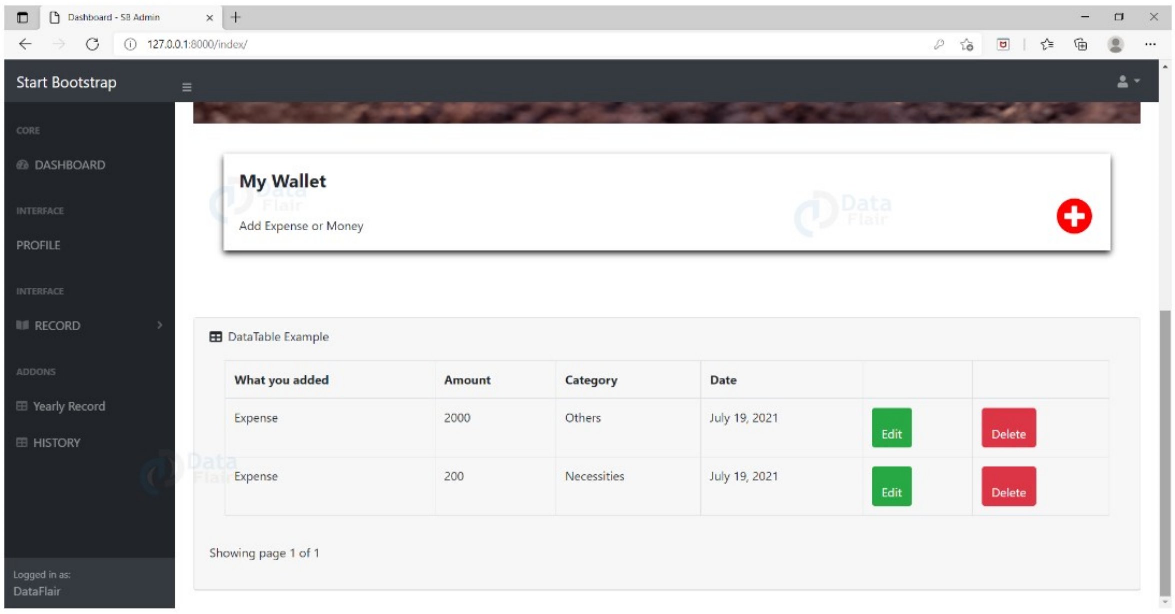


Dashboard:



Message : Successfully logged in

Expense Tracker



My Wallet

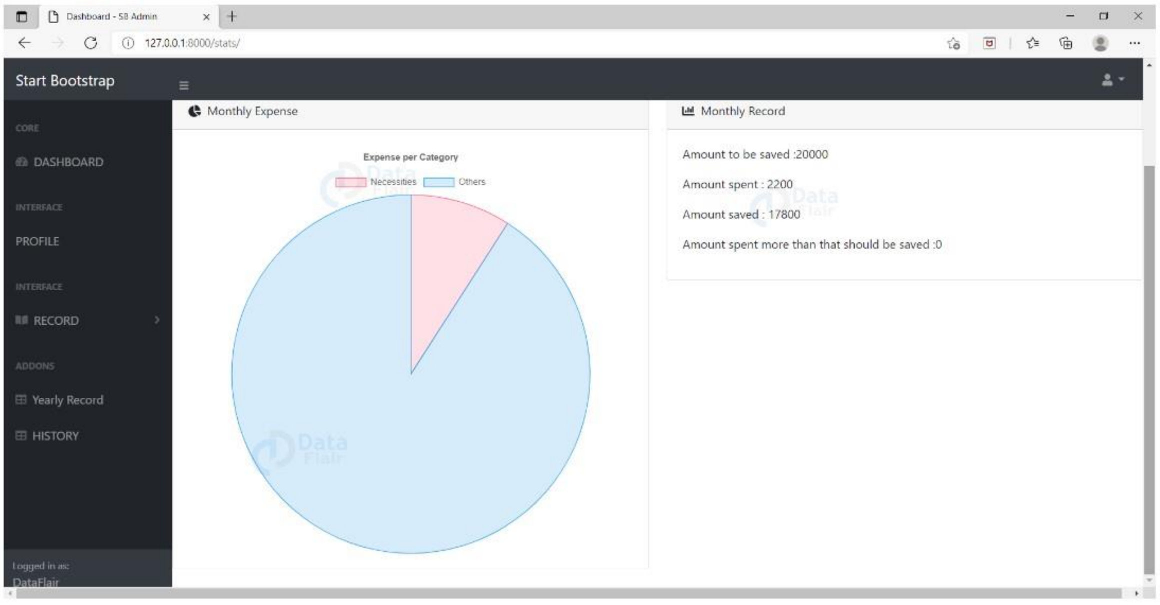
Add Expense or Money

DataTable Example

What you added	Amount	Category	Date		
Expense	2000	Others	July 19, 2021	Edit	Delete
Expense	200	Necessities	July 19, 2021	Edit	Delete

Showing page 1 of 1

Monthly Expense Page:



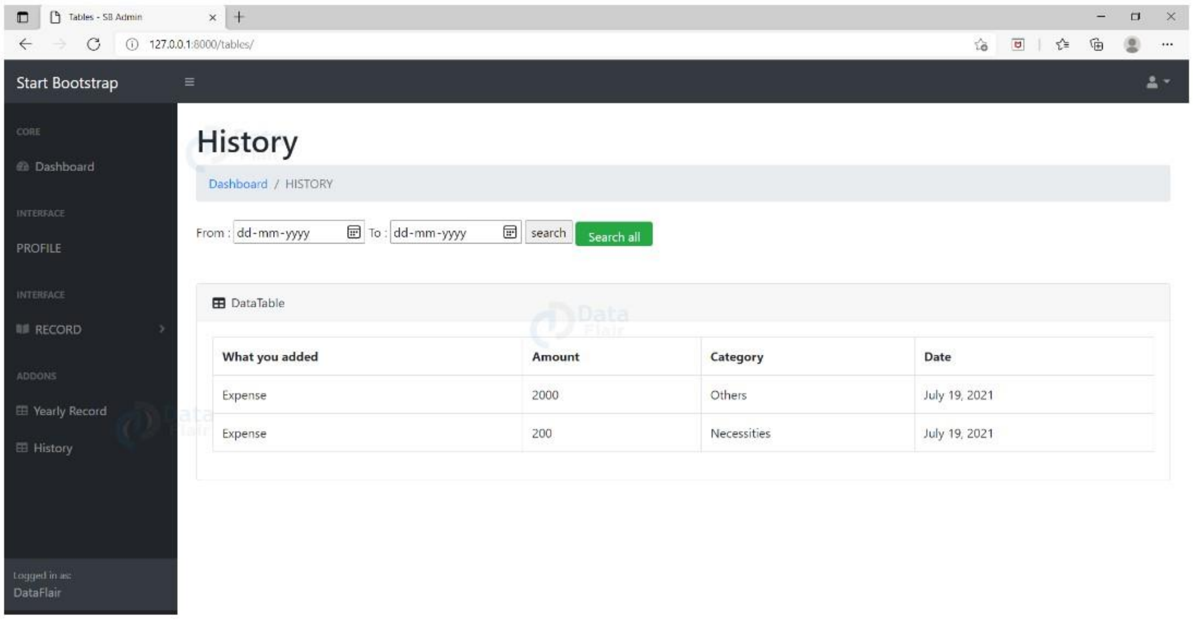
Amount to be saved :20000

Amount spent : 2200

Amount saved : 17800

Amount spent more than that should be saved :0

History Page:



What you added	Amount	Category	Date
Expense	2000	Others	July 19, 2021
Expense	200	Necessities	July 19, 2021

Conclusion:

- We have successfully created the expense tracker project in python. We learned a variety of concepts while making this project.
- In conclusion, our Expense Tracker is designed to empower you in taking control of your finances, providing a seamless and efficient way to manage your expenses and achieve your financial goals.
- As you embark on your financial journey, we are committed to continuously enhancing the Expense Tracker with new features, improvements, and a steadfast commitment to user satisfaction. Your feedback is invaluable to us, so please don't hesitate to share your thoughts.
- Thank you for choosing our Expense Tracker. Here's to a future of financial wellness and success!