Recipe Meal Planner using Django

Last Updated : 25 Sep, 2024

In this article, we will create the Recipe Meal Planner using Django step-by-step. Generally, we will implement the CRUD (Create, Read, Update, Delete) operations, allowing users to add the recipe name, day, and the recipe itself. Additionally, we will establish a login system, requiring users to register and log in before creating the recipe meal planner. Once the user has added all the daily recipe information, they simply need to click a single button. Subsequently, a PDF form will be generated, which the user can save for future reference.

Recipe Meal Planner using Django

Here, we will create the step-by-step Recipe Meal Planner using Django.

Create Project Folder

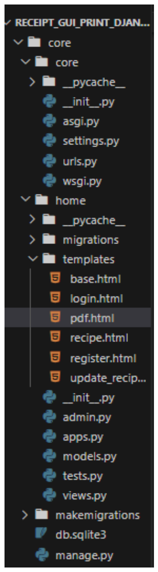
To start the project and app use this command

```
django-admin startproject core
cd core
python manage.py startapp home
```

Now add this app to the **'settings.py'**

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "home",
]
```

File Structure



```
RECEIPT_GUI_PRINT_DJAN...
  core
    core
      __pycache__
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    home
      __pycache__
      migrations
      templates
        base.html
        login.html
        pdf.html
        recipe.html
        register.html
        update_recip...
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      views.py
    makemigrations
    db.sqlite3
    manage.py
```

Building a **recipe meal planner** is a practical way to learn Django. If you want to enhance your skills and build more feature-rich applications, the **Django Web Development-Basics to Advance** **Course is** an ideal resource.

Setting Necessary Files

models.py: Here, the below code defines a Django model named `Recipe` with fields for user, day, name, and description. The `user` field is a foreign key to the built-in `User` model, allowing a null value on deletion. The default values for `day`, `name`, and `description` are set to 'something'.

Python

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

#Create mode for receipt
class Recipe(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)
    day = models.CharField(max_length=100, default='something')
    name = models.CharField(max_length=100,
default='something')
    description = models.CharField(max_length=100,
default='something')
```

views.py: Here, the below code defines a Django application for recipe management, including functionalities for creating, updating, and deleting recipes. It also handles user authentication with login, registration, and logout features. Additionally, there is a basic PDF generation feature for recipes, with the ability to search and filter recipes by day.

Python

```python
#import all libraries
from django.shortcuts import render, redirect
from .models import Recipe
from django.http import HttpResponse, JsonResponse,
HttpResponseRedirect
from django.contrib import messages
from django.contrib.auth import login, authenticate
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.contrib.auth import logout

#create recipes page
@login_required(login_url='/login/')
def recipes(request):
    if request.method == 'POST':
        data = request.POST
        day = data.get('day')
        name = data.get('name')
        description = data.get('description')
        Recipe.objects.create(
            day = day,
            name=name,
            description=description,
        )
        return redirect('/')

    queryset = Recipe.objects.all()
    if request.GET.get('search'):
        queryset = queryset.filter(
            day__icontains=request.GET.get('search'))

    context = {'recipes': queryset}
    return render(request, 'recipe.html', context)

#Update the recipes data
@login_required(login_url='/login/')
def update_recipe(request, id):
    queryset = Recipe.objects.get(id=id)

    if request.method == 'POST':
        data = request.POST
        day = data.get('day')
        name = data.get('name')
        description = data.get('description')

        queryset.day = day
        queryset.name = name
        queryset.description = description
        queryset.save()
        return redirect('/')

    context = {'recipe': queryset}
    return render(request, 'update_recipe.html', context)

#delete the recipes data
@login_required(login_url='/login/')
def delete_recipe(request, id):
    queryset = Recipe.objects.get(id=id)
    queryset.delete()
    return redirect('/')

#login page for user
def login_page(request):
    if request.method == "POST":
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            user_obj =
User.objects.filter(username=username)
            if not user_obj.exists():
                messages.error(request, "Username not
found")
```

```
                return redirect('/login/')
            user_obj = authenticate(username=username, password=password)
            if user_obj:
                login(request, user_obj)
                return redirect('recipes')
            messages.error(request, "Wrong Password")
            return redirect('/login/')
        except Exception as e:
            messages.error(request, "Something went wrong")
            return redirect('/register/')
    return render(request, "login.html")

#register page for user
def register_page(request):
    if request.method == "POST":
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            user_obj = User.objects.filter(username=username)
            if user_obj.exists():
                messages.error(request, "Username is taken")
                return redirect('/register/')
            user_obj = User.objects.create(username=username)
            user_obj.set_password(password)
            user_obj.save()
            messages.success(request, "Account created")
            return redirect('/login')
        except Exception as e:
            messages.error(request, "Something went wrong")
            return redirect('/register')
    return render(request, "register.html")

#logout function
def custom_logout(request):
    logout(request)
    return redirect('login')

#Generate the Bill
@login_required(login_url='/login/')
def pdf(request):
    if request.method == 'POST':
        data = request.POST
        day = data.get('day')
        name = data.get('name')
        description = data.get('description')

        Recipe.objects.create(
            day = day,
            name=name,
            description=description,

        )
        return redirect('pdf')
    queryset = Recipe.objects.all()

    if request.GET.get('search'):
        queryset = queryset.filter(
            day__icontains=request.GET.get('search'))

    context = {'recipes': queryset}
    return render(request, 'pdf.html', context)
```

Creating GUI

login.html: Below, HTML code is a concise Bootstrap-based login form for a job portal, featuring input fields for username and password. Success messages are displayed using Bootstrap's alert, and a link is included for users to create a new account.

HTML

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
```

```
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
    <title>Job Portal</title>
</head>
<body><br><br><br><br>


   <br><br>

    <div class="container bg-white col-md-2 card shadow p-3
" id="log">
        <div class="login-form">
            {% if messages %}
            {% for message in messages %}
            <div class="alert alert-success {{ message.tags
}} mt-4"
                 role="alert">
                {{ message }}
            </div>
            {% endfor %}
            {% endif %}
            <form action="" method="post">
                {% csrf_token %}
                <h4   >  Login </h4>
                <div class="">
                    <input type="text"  name="username"
                           placeholder="Username" required
                        >
                </div>
                <div class="mt-2">
                    <input type="password"  name="password"
                           placeholder="Password" required>
                </div>
                <div class="mt-2">
                    <button   >Login</button>
                </div>
                <br>
            </form>
```

```
                <p  ><a href="{% url 'register' %}" >Create an
                    Account.</a></p>
            </div>
        </div>

</body>

</html>
```

register.html: The provided HTML code creates a registration form for a job portal using Bootstrap. It includes input fields for username and password, a registration button, and a link to the login page. Bootstrap's alert component is utilized for displaying success messages.

HTML

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/cs
s/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCO
mLASjC" crossorigin="anonymous">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">
    <title>Job Portal</title>
</head>

<body>
    <body>
        <br> <br><br><br><br><br>

        <div class="container bg-white mx-auto col-md-2
card shadow p-3">
```

```
            <div class="login-form">
                {% if messages %}
                {% for message in messages %}
                <div class="alert alert-success {{ message.tags }}" role="alert">
                    {{ message }}
                </div>
                {% endfor %}
                {% endif %}
                <form action="" method="post">
                    {% csrf_token %}
                    <h4 > Register </h4>
                    <div >
                        <input type="text" name="username" placeholder="Username" required>
                    </div>

                    <div class="mt-2">
                        <input type="password" name="password" placeholder="Password" required>
                    </div>
                    <div class="mt-2">
                        <button >Register</button>
                    </div>
                </form>
                <p ><a href="{% url 'login' %}">Log In </a></p>
            </div>
        </div>

    </body>

</html>
```

recipe.html: The Django template extends a base HTML file and presents a form for adding recipe data. It includes a button to generate a PDF recipe plan and displays a table of existing recipes with options to delete or update. The styling includes a hover effect for link color change.

HTML

```
{% extends "base.html" %}
{% block start %}

<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
<style>
.ok{
    color: white;
    text-decoration: none;
  }
  .ok:hover{
    color: white;
    text-decoration: none;
  }

</style>

<div class="container mt-3 col-6">
  <br><br>
    <form class="col-6 mx-auto card p-3 shadow-lg"
method="post" enctype="multipart/form-data">
        {% csrf_token %}
        <h4> Recipe </h4>
        <hr>
        <div class="form-group">
          <label for="exampleInputEmail1">Day-Time </label>
          <input type="text" name="day" required>
        </div>
        <div class="form-group">
          <label for="exampleInputEmail1">Recipe </label>
          <input name="name" type="text" required>


         </div>
        <div class="form-group">
          <label for="exampleInputPassword1">Description
</label>
          <!-- <input name="description" type="text"
rows="10" cols="50" required> -->

          <textarea  name="description"  type="text"
rows="5" cols="30"></textarea>
        </div>
        <button type="submit" class="">Add Data</button>
    </form>
    <hr>
    <div class="class mt-5">
        <form action="">
          <button > <a  href="{% url 'pdf' %}">Generate Plan
</a></button>
        </form>

        <table class="table mt-6">
            <thead>
                <tr>
                    <th scope="col">S.No. </th>
                    <th scope="col">Day-Time </th>
                    <th scope="col">Recipe Name </th>
                    <th scope="col">Description </th>
                    <th scope="col">Actions</th>
                </tr>
            </thead>
            <tbody>
                {% for recipe in recipes %}
                <tr>
                    <th scope="row">{{forloop.counter}}</th>
                    <td>{{recipe.day}}</td>
                    <td> {{recipe.name}}</td>
                    <td>{{recipe.description}}</td>
                    <td>
                        <a href="/delete_recipe/{{recipe.id
}}" >Delete </a>
                        <a href="/update_recipe/{{recipe.id
}}">Update </a>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>


    {% endblock %}
```

update_recipe.html: The Django template, extending a base HTML file, displays a form for updating recipe data. It pre-fills fields with existing data and allows users to modify day, recipe name, and description. The styling uses Bootstrap, creating a centered card with a shadow effect.

HTML

```html
{% extends "base.html" %}
{% block start %}

<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
<style>

</style>


<div class="container mt-5 col-5">

  <form class="col-6 mx-auto card p-3 shadow-lg" method="post" enctype="multipart/form-data">
    {% csrf_token %}


    <div class="form-group">
      <label for="exampleInputEmail1">Day-Time </label>
      <input type="text" name="day" value="{{recipe.day}}" required>
    </div>
    <div class="form-group">
      <label for="exampleInputEmail1">Recipe </label>
      <input name="name" type="text" value="{{recipe.description}}"
        required>

    </div>
    <div class="form-group">
      <label for="exampleInputPassword1">Description </label>
        <textarea  name="description"  type="text" rows="5" cols="30" value="{{recipe.description}}"></textarea>
    <br>
    <br>

```

```
        <button type="submit" >Update Data</button>
      </form>


    </div>

    {% endblock %}
```

pdf.html: Below, HTML document defines a Recipe Meal Planner webpage with Bootstrap styling. It includes a table displaying recipe details and a button to generate a PDF using the html2pdf library. The styling features a clean layout with a card container and a green-themed table. JavaScript functionality is added to trigger PDF generation on button click.

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Recipe Meal Planner</title>

    <!-- Add Bootstrap CSS Link -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.2/dist/css/
bootstrap.min.css" rel="stylesheet">
    <!-- Add html2pdf library -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/html2pdf.js/0.10
.1/html2pdf.bundle.js"></script>
    <style>
        body {
            background-color: #f8f9fa;
        }

        .recipe-container {
            padding: 20px;
            margin-top: 30px;
            background-color: #ffffff;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }

        .recipe-header {
            color: black;
        }

        .recipe-table th,
        .recipe-table td {
            text-align: center;
            border: 1px solid #dee2e6;
            padding: 8px;
        }

        .recipe-table th {
            background-color: #70e78c;
            color: #fff;
        }

        .generate-pdf-btn {
            margin-top: 20px;
        }
    </style>
</head>

<body>

    <div class="container recipe-container col-md-8">
        <div class="card">
            <div class="card-body">
                <h2 class="recipe-header">Recipe Meal
Planner</h2>
                <br><br>
                <table class="table recipe-table">
                    <thead class="recipe-table-head">
                        <tr>
                            <th>Day-Time</th>
                            <th>Recipe Name</th>
                            <th>Description</th>
                        </tr>
                    </thead>
                    <tbody>
```

```
                        {% for recipe in recipes %}
                        <tr>
                            <td>{{recipe.day}}</td>
                            <td>{{recipe.name}}</td>
                            <td>{{recipe.description}}</td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>

                <button class="btn btn-danger  generate-pdf-btn" onclick="generatePDF()">Generate PDF</button>
            </div>
        </div>
    </div>

    <!-- Add Bootstrap JS and Popper.js Scripts -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.2/dist/js/bootstrap.min.js"></script>

    <script>
        function generatePDF() {
            var element = document.querySelector('.recipe-container');
            html2pdf(element);
        }
    </script>
</body>

</html>
```

base.html: The HTML template serves as a base for Django views, with a dynamic title based on the variable `page`. It includes a block for content rendering, allowing customization in extending templates.

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>{{page}}</title>
</head>
<body>

    {% block start %}
    {% endblock %}

    <script>
        console.log('Hey Django')
    </script>
</body>

</html>
```

admin.py : Here we are registering our models.

Python

```python
from django.contrib import admin
from .models import *
from django.db.models import Sum

admin.site.register(Recipe)
```

urls.py : Here, the Django URL patterns include routes for user authentication (login, logout, register), recipe handling (view, update, delete), and a PDF generation endpoint. These paths are associated with corresponding views from the 'home' app.

Python

```python
from django.contrib import admin
from django.urls import path
from home import views

urlpatterns = [
    path('logout/', views.custom_logout, name="logout"),
    path('pdf/', views.pdf , name='pdf'),
    path('admin/', admin.site.urls),
    path('login/' , views.login_page, name='login'),
    path('register/', views.register_page, name='register'),

    path('', views.recipes, name='recipes'),
    path('update_recipe/<id>', views.update_recipe, name='update_recipe'),
    path('delete_recipe/<id>', views.delete_recipe, name='delete_recipe'),
]
```

Deployment of the Project

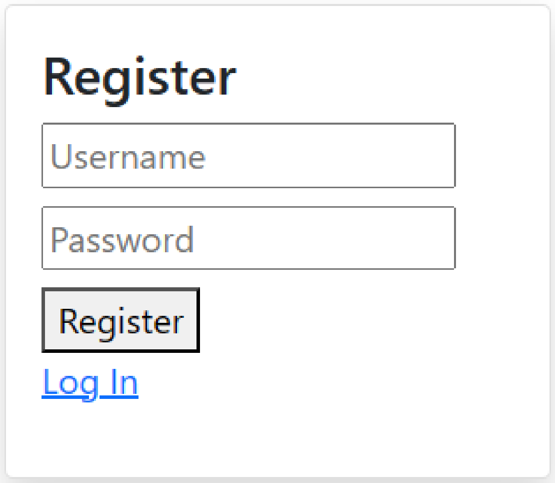
Run these commands to apply the migrations:

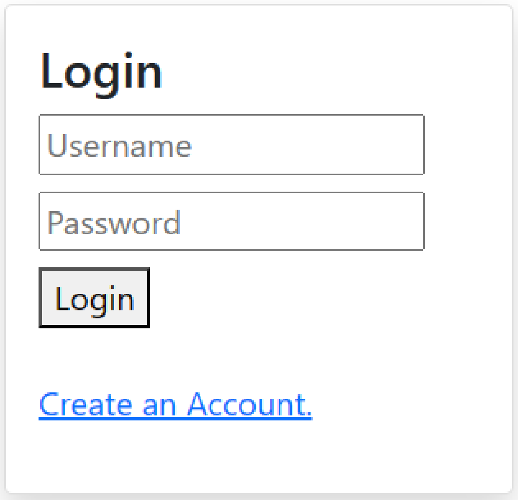
```
python3 manage.py makemigrations
python3 manage.py migrate
```

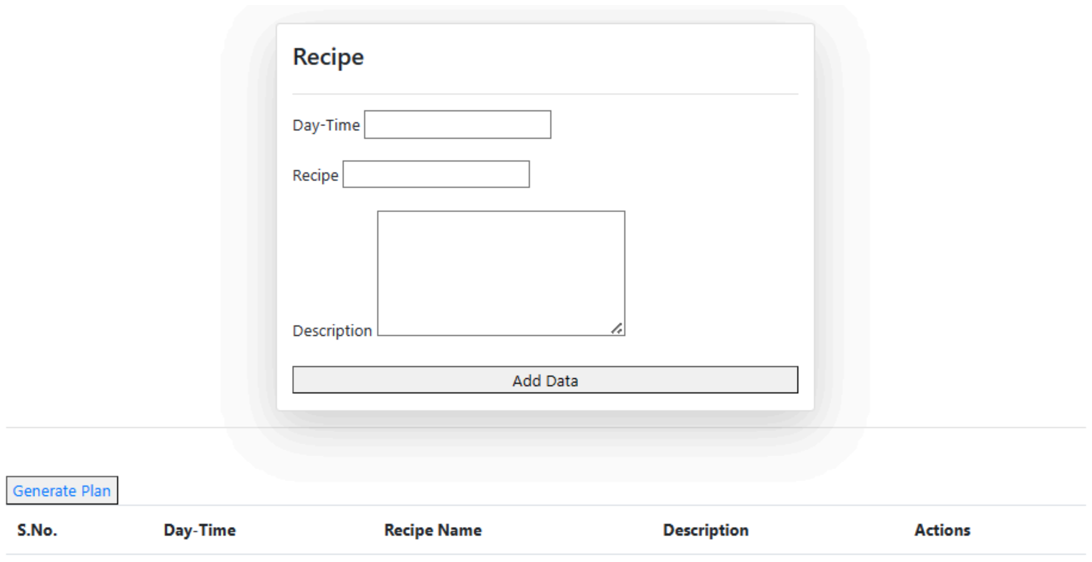
Run the server with the help of following command:

```
python3 manage.py runserver
```

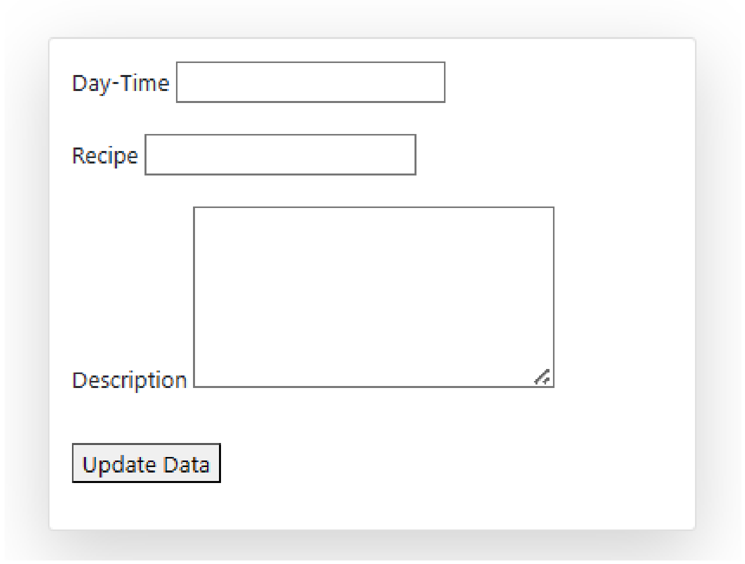
Output

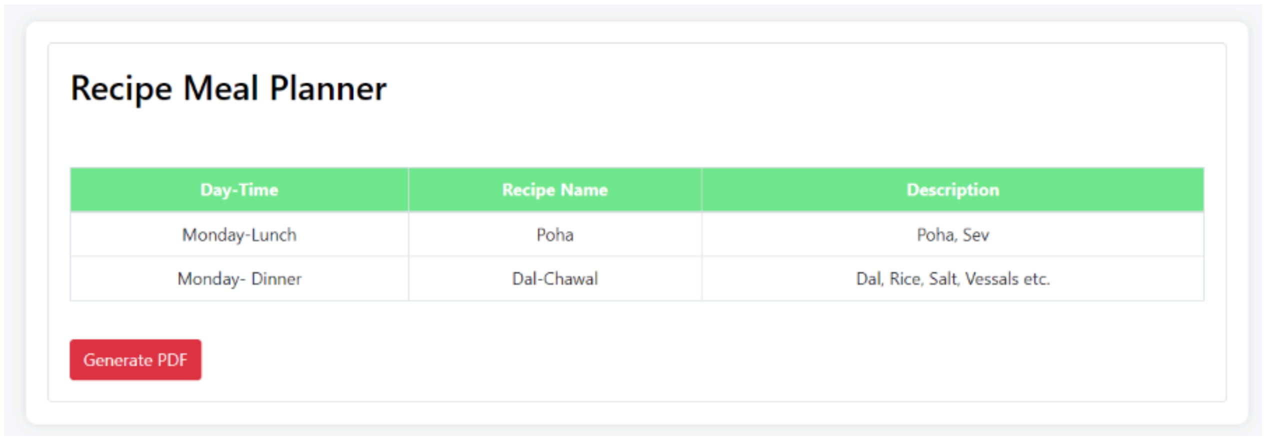






S.No.	Day-Time	Recipe Name	Description	Actions




Recipe Meal Planner

Day-Time	Recipe Name	Description
Monday-Lunch	Poha	Poha, Sev
Monday- Dinner	Dal-Chawal	Dal, Rice, Salt, Vessals etc.

S soura...

Previous Article **Next Article**

Portfolio Showcase using Django

Similar Reads

Build a Recipe App using MVVM Architecture with Kotlin in Android

In this article, we will make a recipe app that displays a list of Indian recipes using the retrofit library and MVVM architecture. Model — View — ViewModel (MVV...

7 min read

Recipe Recommendation System Using Python

In this article, we will explore how to build a Recipe Recommendation System using Streamlit and OpenAI. We will create the GUI using the Streamlit library...

2 min read

Recipe Generator using Next.js

In this tutorial, we'll create a Recipe Generator App using Next.js, a React framework that allows users to generate random recipes. This type of applicatio...

4 min read

How to create a food recipe app using ReactJS ?

We are going to make a food recipe app using React.js. Pre-requisite:React hooksReact componentsJavaScript ES6API CSSApproach: Here in this app we...

4 min read

Recipe Finder using ReactJS

In this project article, we will be creating a recipe finder application using the React library. We have given the application the name "GeeksforGeeks Recipe...

5 min read

Daily Activity Planner App using MERN Stack

This project is a web-based Daily Planner application built using React for the frontend and Node.js with Express for the backend. It allows users to schedule...

7 min read

Workout Planner using MERN Stack

With a busy schedule, it is very important to prioritize our health and fitness. In this article, a walkthrough to creating a Workout Planner using MERN (Mongod...

15+ min read

Recipe Manager with MERN Stack

The Recipe Manager is a popular MERN Stack-based project used for managing various recipes and creating, editing, or deleting recipes. This article will cover t...

13 min read

Design a Recipe App in HTML CSS & JavaScript

We will create an attractive and styled application that gives the recipe of food, and dishes entered by the user. The user needs to enter the required food name...

6 min read

Project Idea | (Trip Planner)

This project is basically an application which helps friends, colleagues or relatives who live at far off places (generally in another country) plan a trip together to a...

2 min read

Article Tags : Django Geeks Premier League Project Python +2 More

Practice Tags : python

GET IT ON Google Play | Download on the App Store