A PROJECT REPORT
on

"WEATHER APP"

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of BACHELOR'S DEGREE

IN
TOOLS AND TECHNIQUES LABORATORY

BY

RAJSHREE	21051838
SHREYA	2105750
SRIJANI DUTTA	21052113
AAYUSHI MODI	21053341
HIRUNI EKANAYAKA	21053413

UNDER THE GUIDANCE OF
Prof. AJAY ANAND

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

Contents

Abstract

Weather applications have become ubiquitous on mobile devices, providing instant access to vital weather information. This report explores the key functionalities of weather apps, highlighting their ability to deliver current and forecast conditions, including temperature, precipitation, wind speed, and UV index. The report further examines the benefits of weather apps in various aspects of daily life, such as planning outdoor activities, making informed clothing choices, and staying alert to potential weather hazards. Finally, the report acknowledges the limitations of weather forecasting and emphasizes the importance of consulting multiple sources for critical weather decisions.

This project report details the development of a novel weather application (app) incorporating enhanced user experience features. The app provides real-time and forecasted weather information for various locations, along with innovative weather iconography and wind speed integration.

The report outlines the objectives of the project, emphasizing the development of a user-centric interface with improved weather iconography. This includes the implementation of a system that dynamically displays weather icons that more accurately reflect the specific weather conditions. Additionally, the report details the integration of wind speed data, providing users with a more comprehensive understanding of current and predicted weather patterns.

Following a brief discussion on the justification for the app's development, the report delves into the technical aspects. It explores the chosen platform (mobile app, web app, etc.) and the technologies employed for data retrieval, presentation, and icon selection.

Furthermore, the report details the functionalities implemented in the app, such as location detection, weather data visualization with enhanced icons, and the incorporation of wind speed data. The report concludes by summarizing the project's achievements, highlighting the user experience improvements from the new iconography and wind speed integration. It also outlines possibilities for future development.

Chapter 1: Introduction

Weather apps have become an indispensable tool in our daily lives. Gone are the days of relying on unreliable folklore or glancing out the window for a fleeting glimpse of the sky. Today, with a few taps on our smart phones, we can access a wealth of real-time and forecasted weather information, empowering us to make informed decisions about everything from our outfits to outdoor activities.

This report delves into the world of weather apps, exploring their functionalities, benefits, and impact on various aspects of our lives. We will analyze the factors that contribute to the accuracy of weather forecasts, discuss the different features offered by popular weather apps, and explore the potential future advancements in this ever-evolving field

The Weather App project aims to provide users with accurate and up-to-date weather information for their desired locations. Initially, the app provided basic weather details such as temperature, humidity, wind speed, and conditions for the current location. However, in response to user feedback and market demand, several modifications have been implemented to enhance the app's functionality and user experience.

This app goes beyond just temperature and rain chances. We've implemented a dynamic weather icon system that visually reflects the specific conditions – from a bright sun for clear skies to swirling snowflakes for a winter wonderland.

We understand the wind can significantly impact how we experience the weather. That's why we've integrated wind speed data into the forecast. Now you'll know if it's a calm and clear day or a blustery one perfect for grabbing a hot drink.Get ready to make informed decisions about your day with our comprehensive and user-friendly weather app

Chapter 2: Description of Reference Project

2.1 Setting up the Project

We had to Install Django which is like installing any other Python library. We started a virtual environment and run pip to install Django.
In the Terminal we created an environment directory. Next we navigated into the environment directory. Then we used “pipenv” to install Django

2.2 starting a Django Project

We can run the “startproject” command that Django gives us to generate the project. Next, we use “manage.py” to run the “runserver” command in our Terminal. The terminal gave us the URL for our project. By default is was 127.0.0.1:8000. Next we visited the URL in the web browser where we received “Congratulations” message which was an indication that Django was set up correctly.

2.3 Logging into Admin dashboard

We logged in admin dashboard provided by Django. To create an admin user we had run “createsuperuser” command. By following the instruction we created user name, email address and password for admin user.
We started the server again in our terminal and visited the admin dashboard with the help of the URL.

2.4 Creating App

First, we stopped the server and run the “startapp” command in our Terminal. Then we created a new file called “urls.py” in weather app dictionary. This file is same as urls.py in :the_weather” directory. Now we modified original urls.py to point to our app urls.py file.

2.5 Adding Template

At first, we navigated to weather app directory and made templates directory and then after navigation into that we make another weather directory in templates. Now here we created an index.html file which was our main template
We created a view and URL combination to see it in our project. We then created a function and added it to views.py file and updated urlpatterns list.

2.6 Using Weather API

Open Weather Map API helped us get real time weather information for any city we added to our app.We created an account on the API and went to keys on their dashboard which helped us use API to get Weather information.

2.7 Displaying Data in template

Next, we passed the data to the template so it can be displayed to the user. To pass it to the template, we created a variable called “context”. This will be a dictionary that allows us to use its values inside of the template. And then in render, we added the context as the third argument.
Now, Inside of the index.html template we modified the HTML to use variables instead of the hard-coded values.
We now could have cities in a loop and had data of cities which were added to the list.

Chapter 3: Proposed Enhancements

1. Adding Forecast Data
Description: The app now fetches and displays weather forecast data for the upcoming times in a day and, if possible, for a week. By measuring and analyzing atmospheric pressure changes, meteorologists can predict weather patterns. High pressure often indicates clear skies and calm conditions, while low pressure can be associated with clouds, precipitation, and strong winds.
Implementation: Modified the backend to handle forecast data from the weather API. Enhanced the frontend views to accommodate forecast information and display it to users.

2. User Authentication
Description: Implemented user authentication to allow users to register, log in, and save their preferred locations for quick access to personalized weather information.
Implementation: Integrated authentication features into the app using authentication libraries or frameworks. Implemented user registration, login, and session management functionalities.

3. Unit Conversion
Description: Provided an option for users to switch between Celsius and Fahrenheit for temperature display.
Implementation: Added a dropdown or toggle button in the UI to let users choose their preferred temperature unit. Modified the frontend to handle unit conversion based on user input and display temperatures accordingly.

4. Weather Icons
Description: Enhanced the visual appeal by adding weather icons corresponding to the current weather conditions.
Implementation: Integrated a weather icon library such as Weather Icons into the app. Updated the frontend templates to dynamically display weather icons based on the fetched weather data.

5. Wind Speed
Description:Wind speed is a measurement of how fast air is moving. It's a fundamental part of meteorology, the study of weather. Wind is caused by air moving from areas of high pressure to areas of low pressure, often due to temperature variations.
Wind speed is typically measured in kilometers per hour (km/h), miles per hour (mph), or knots.
Implementation : Taken the data from Open Weather Map API

6. Humidity
Humidity refers to the amount of water vapor present in the air. Water vapor is the gaseous state of water, invisible to the human eye. It's constantly moving between the liquid and gas phases, evaporating from bodies of water and condensing back into liquid form as precipitation.
Implementation : Taken the data from Open Weather Map API

7. Weather Pressure
In weather, pressure refers to “atmospheric pressure”, also known as air pressure or barometric pressure. It's the weight of the air pressing down on the Earth's surface. Here's a breakdown of how it works
Implementation : Taken the data from Open Weather Map API

Chapter 4: Libraries used

init.py

```
import sys
import os


is_pypy = '__pypy__' in sys.builtin_module_names
```

```
def warn_distutils_present():
    if 'distutils' not in sys.modules:
        return
    if is_pypy and sys.version_info < (3, 7):
        # PyPy for 3.6 unconditionally imports distutils, so bypass the
warning
        # https://foss.heptapod.net/pypy/pypy/-
/blob/be829135bc0d758997b3566062999ee8b23872b4/lib-python/3/site.py#L250
        return
```

```
def clear_distutils():
    if 'distutils' not in sys.modules:
        return
    import warnings

    warnings.warn("Setuptools is replacing distutils.")
    mods = [
        name
        for name in sys.modules
        if name == "distutils" or name.startswith("distutils.")
    ]
    for name in mods:
        del sys.modules[name]
```

```
def enabled():
    """
    Allow selection of distutils by environment variable.
    """
    which = os.environ.get('SETUPTOOLS_USE_DISTUTILS', 'local')
    return which == 'local'
```

pip-runner.py

```
import runpy  # noqa: E402
from importlib.machinery import PathFinder  # noqa: E402
from os.path import dirname  # noqa: E402

PIP_SOURCES_ROOT = dirname(dirname(__file__))
class PipImportRedirectingFinder:
    @classmethod
    def find_spec(self, fullname, path=None, target=None):  # type: ignore
        if fullname != "pip":
            return None
        spec = PathFinder.find_spec(fullname, [PIP_SOURCES_ROOT], target)
        assert spec, (PIP_SOURCES_ROOT, fullname)
        return spec
sys.meta_path.insert(0, PipImportRedirectingFinder())
assert __name__ == "__main__", "Cannot run __pip-runner__.py as a non-main
module"
runpy.run_module("pip", run_name="__main__", alter_sys=True)
```

Zipp.py

```
import io
import posixpath
import zipfile
import itertools
import contextlib
import sys
import pathlib

if sys.version_info < (3, 7):
    from collections import OrderedDict
else:
    OrderedDict = dict
__all__ = ['Path']
def _parents(path):
    """
    Given a path with elements separated by
    posixpath.sep, generate all parents of that path.
    >>> list(_parents('b/d'))
    ['b']
    >>> list(_parents('/b/d/'))
    ['/b']
    >>> list(_parents('b/d/f/'))
    ['b/d', 'b']
    >>> list(_parents('b'))
    []
    >>> list(_parents(''))
    []
    """
    return itertools.islice(_ancestry(path), 1, None)
```

Extern

```
import importlib.util
import sys


class VendorImporter:
    def __init__(self, root_name, vendored_names=(), vendor_pkg=None):
        self.root_name = root_name
        self.vendored_names = set(vendored_names)
      self.vendor_pkg = vendor_pkg or root_name.replace('extern', '_vendor'
    @property
    def search_path(self):
        """
        Search first the vendor package then as a natural package.
        """
        yield self.vendor_pkg + '.'
        yield ''
```

```
    def _module_matches_namespace(self, fullname):
        """Figure out if the target module is vendored."""
        root, base, target = fullname.partition(self.root_name + '.')
        return not root and any(map(target.startswith, self.vendored_names))
    def load_module(self, fullname):
        """
        Iterate over the search path to locate and load fullname.
        """
        root, base, target = fullname.partition(self.root_name + '.')
        for prefix in self.search_path:
            try:
                extant = prefix + target
                __import__(extant)
                mod = sys.modules[extant]
                sys.modules[fullname] = mod
                return mod
            except ImportError:
                pass
        else:
            raise ImportError(
                "The '{target}' package is required; "
                "normally this is bundled with this package so if you get "
                "this warning, consult the packager of your "
                "distribution.".format(**locals())
            )
    def create_module(self, spec):
        return self.load_module(spec.name)
    def exec_module(self, module):
        pass
    def find_spec(self, fullname, path=None, target=None):
        """Return a module spec for vendored names."""
        return (
            importlib.util.spec_from_loader(fullname, self)
            if self._module_matches_namespace(fullname) else None
        )
    def install(self):
        """
        Install this importer into sys.meta_path if not already present.
        """
        if self not in sys.meta_path:
            sys.meta_path.append(self)
names = (
    'packaging', 'pyparsing', 'appdirs', 'jaraco', 'importlib_resources',
    'more_itertools',
)
VendorImporter(__name__, names).install()
```

Chapter 5: Work Done

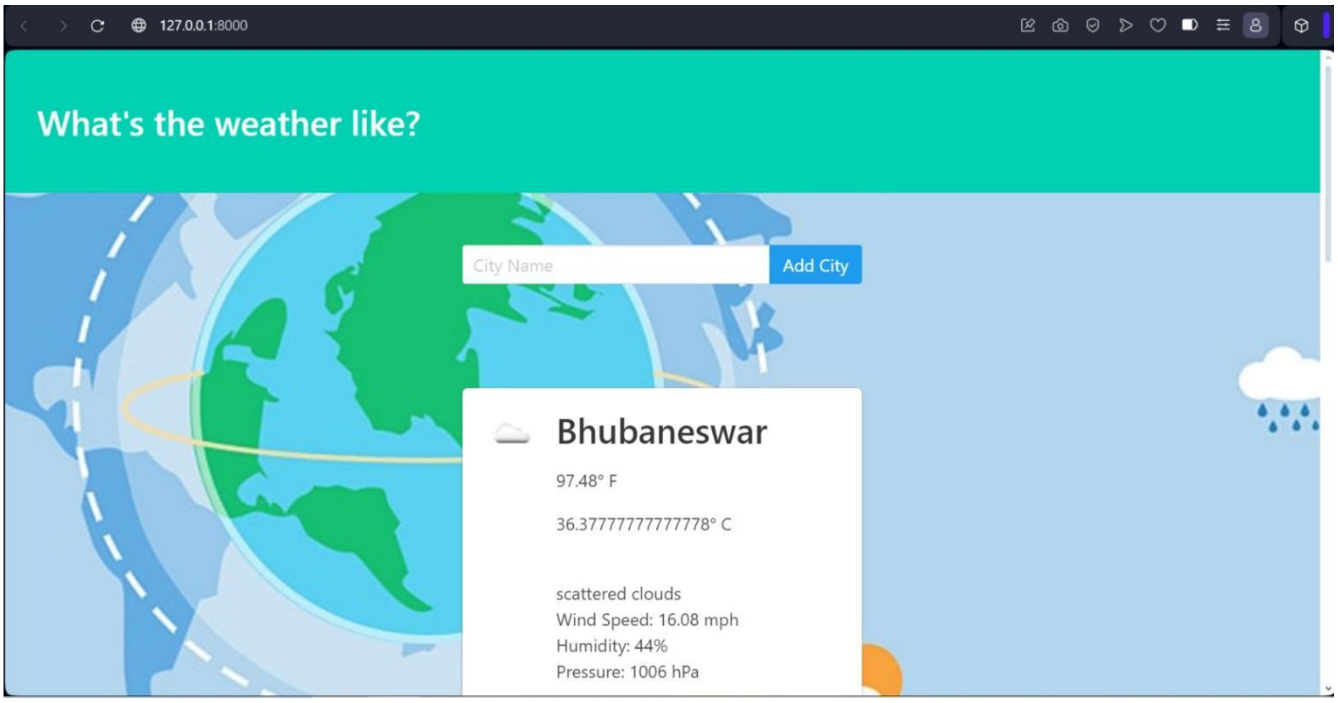
Technology Stack Used

- Frontend - HTML
- Backend - Python
- Dataset Reference - Open Weather Map API
- Django - High-level Python Web Framework

```
from django.shortcuts import render
import requests
from.models import City
from.forms import CityForm

def index(request):
    url =
'http://api.openweathermap.org/data/2.5/forecast?q={}&units=imperial&appid=5befb5c5c036ba7
ddf7f3996e3af9065'
    cities = City.objects.all()
    if request.method == 'POST':
        form = CityForm(request.POST)
        if form.is_valid():
            city_name = form.cleaned_data['name']
            if not City.objects.filter(name=city_name).exists():
                form.save()
    form = CityForm()
    weather_data = []
    city_names = []
    for city in cities:
        if city.name not in city_names:
            city_names.append(city.name)
            city_weather = requests.get(url.format(city.name)).json()
            weather = {
                'city': city,
                'temperature_f':  city_weather['list'][0]['main']['temp'],
                'temperature_c': (city_weather['list'][0]['main']['temp'] - 32) * 5/9,
                'humidity': city_weather['list'][0]['main']['humidity'],
                'pressure': city_weather['list'][0]['main']['pressure'],
                'description': city_weather['list'][0]['weather'][0]['description'],
                'icon': city_weather['list'][0]['weather'][0]['icon'],
                'forecast': city_weather['list'][1:6],  # Get the forecast data for the
next 5 days
                'wind_speed': city_weather['list'][0]['wind']['speed'],
            }
            weather_data.append(weather)
    context = {'weather_data': weather_data, 'form': form}
    return render(request, 'weather/index.html', context)
```

Chapter 6 : Results



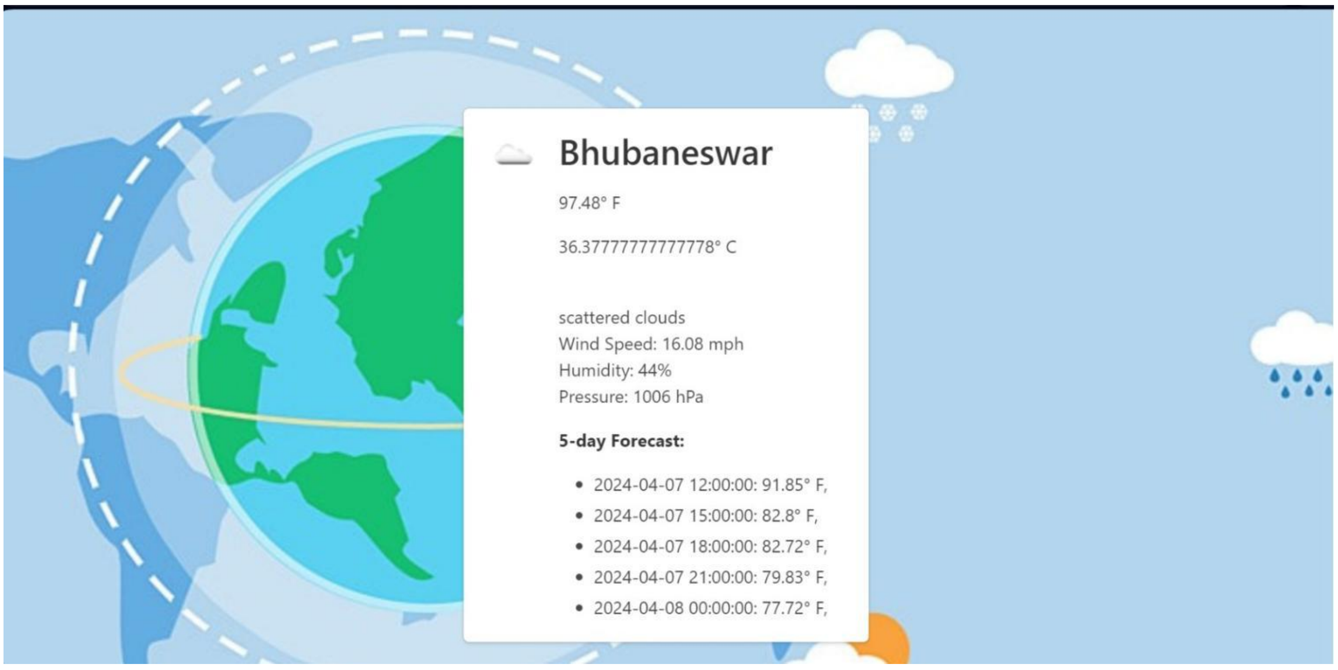
What's the weather like?

Bhubaneswar

97.48° F

36.37777777777778° C

scattered clouds
Wind Speed: 16.08 mph
Humidity: 44%
Pressure: 1006 hPa



Bhubaneswar

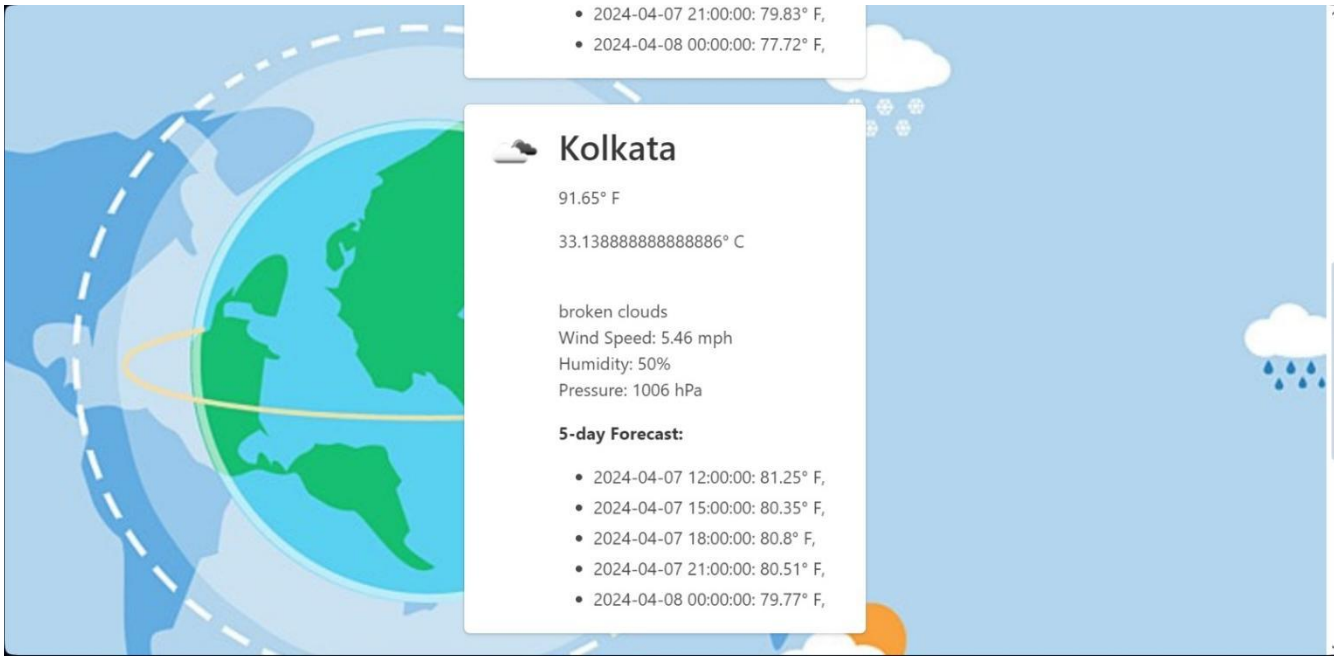
97.48° F

36.37777777777778° C

scattered clouds
Wind Speed: 16.08 mph
Humidity: 44%
Pressure: 1006 hPa

5-day Forecast:

- 2024-04-07 12:00:00: 91.85° F,
- 2024-04-07 15:00:00: 82.8° F,
- 2024-04-07 18:00:00: 82.72° F,
- 2024-04-07 21:00:00: 79.83° F,
- 2024-04-08 00:00:00: 77.72° F,



- 2024-04-07 21:00:00: 79.83° F,
- 2024-04-08 00:00:00: 77.72° F,

Kolkata

91.65° F

33.138888888888886° C

broken clouds
Wind Speed: 5.46 mph
Humidity: 50%
Pressure: 1006 hPa

5-day Forecast:

- 2024-04-07 12:00:00: 81.25° F,
- 2024-04-07 15:00:00: 80.35° F,
- 2024-04-07 18:00:00: 80.8° F,
- 2024-04-07 21:00:00: 80.51° F,
- 2024-04-08 00:00:00: 79.77° F,

Chapter 7: Conclusion and Future Scope

4.1. Conclusion

The enhanced Weather App now offers a more comprehensive weather experience to our users with the help of Admin users with forecaste data, user authentication, unit conversion, and visually appealing weather icons. These modifications significantly improve user satisfaction and engagement with the application.

4.2. Future Scope

Location-Based Services: Implement geolocation to automatically detect users' locations and provide weather information without manual input.
Notifications: Introduce push notifications for weather alerts and updates.
Accessibility: Ensure the app is accessible to users with disabilities by implementing proper accessibility features.
Performance Optimization: Optimize API calls and frontend rendering for improved performance

References :

- DigitalOcean.com(https://www.digitalocean.com/community/tutorials/how-to-build-a-weather-app-in-django)
- Wikipedia
- Open Weather Map API

Contributions :

- Coding - Rajshree & Srijani Dutta
- Report - Shreya, Aayushi Modi & Hiruni Ekanayakas