GOVERNMENT OF INDIA
MINISTRY OF SKILL DEVELOPMENT & ENTREPRENEURSHIP

Skill India

NATIONAL SKILL TRIANING INSTITUTE (W), VIDYANAGER

PROJECT DOCUMENTATION

TOPIC: Automated Flood Detection and Alert System for Villagers

🚀 MISSION CONTROL PANEL 🛠️🌊

Meet the brilliant minds powering the Flood Alert System!
Each crew member plays a crucial role in ensuring accuracy, safety, and efficiency.

👤 Team Member		🎯 Responsibility
A. Vinay		📥 Data Collection
🛠️ B. Anita		🔍 Model Training
🌐 B. Deepthi		📱 Web Integration
🔊 B. Sukanya		📲 Alert & Communication

PROJECT BY-
AI Programming Assistant Trade

FLOOD ALERT
FLOODING IS POSSIBLE. BE PREPARED.

Table of Contents

PROBLEM STATEMENT 3
📌 Summary of Impact (1925–2025) 6
ABOUT THE PROJECT 7
OBJECTIVES OF THE PROJECT 7
SYSTEM ARCHITECTURE OVERVIEW 7
WORKFLOW 8
TECHNOLOGIES USED 8
ADVANTAGES OF THE SYSTEM 8
USER INTERFACE FEATURES 9
💬 CUSTOMER SUPPORT SYSTEM 9
✅ MODIFICATIONS & ENHANCEMENTS IMPLEMENTED 10
🚀 JOURNEY THROUGH THE CODE: UNDERSTANDING (app.py) 12
🔹 1. Importing Required Libraries 12
🔹 2. App & API Configuration 12
🔹 3. Load Data from CSV 12
🔹 4. Load Pretrained Model 13
🔹 5. Weather Fetching Function 13
🔹 6. Send Alert via SMS 13
🔹 7. Routes (Flask Views) 13
8. Start the Flask App 14
DATASET DESCRIPTIONS 15
1. index.html – Home Page 19
2. no_flood.html – Safe Zones Display 22
🌧️ 6. Manual Flood Risk Checker – Documentation .... **Error! Bookmark not defined.**
PROJECT ARCHITECTURE OVERVIEW 28
🚀 COMPLETE SYSTEM DOCUMENTATION: app.py + Templates + Features 28
🔄 SYSTEM EXECUTION FLOW 28
📄 HTML Templates (Explained Point-Wise) 29
Machine Learning Model (flood_model.pkl) 29
🌐 External Integrations 30
🗺️ System Architecture (Text View) 30
✅ Final Notes 30

PROBLEM STATEMENT

🌍 CONTEXT & BACKGROUND

India, with its vast geography and diverse climate zones, faces seasonal monsoons that often lead to flooding in various regions — particularly in low-lying rural areas, urban drainage-challenged cities, and settlements near rivers. Floods not only destroy property and infrastructure but also lead to loss of lives, displacement of families, disease outbreaks, and disruption of local economies.

While the government has established flood early warning systems at national and state levels, many of these systems still suffer from limitations such as:

- **Delayed alerts** due to manual coordination.
- **Non-personalized communication**, not targeting individuals or specific localities.
- **Lack of accessibility** for rural and underprivileged populations without smartphones or internet access.
- **Language barriers**, making alerts ineffective in regions where the population does not understand English or Hindi well.

These gaps pose a major threat to preparedness and timely evacuation in vulnerable areas.

🚨 THE PROBLEM

Despite the availability of real-time weather data and the rise of AI/ML solutions, there is **no widely deployed system** that:

- Uses **live meteorological data** to intelligently **predict flood risks**.
- **Automatically alerts individuals** in potentially affected areas.
- **Delivers those alerts in local languages** (e.g., Telugu, Tamil, Bengali), making them understandable to every recipient.
- Provides an **interactive, low-bandwidth web interface** that works across devices and locations.

This creates a **disconnect between modern technology and real-life usability**, particularly for rural communities in India.

Technical Challenge

Flood forecasting is a complex task due to:

- The **dynamic nature of weather systems**.
- **Variability in local terrain** (e.g., slope, river proximity).
- Lack of **integrated platforms** combining real-time data, AI, language translation, and communication APIs.
- **User data management** – identifying who lives in risk-prone areas and how to reach them.

Any viable solution must integrate:

- Meteorological APIs (e.g., OpenWeatherMap),
- Machine Learning Models for prediction,
- Multilingual support for alerts,
- Web-based dashboards or apps for public access,
- And reliable telecom/SMS delivery systems (e.g., Twilio).

Proposed Solution

This project proposes the **Flood Alert & Weather Monitoring System** — a **smart, scalable, and user-friendly platform** that can:

✓ Use real-time weather data (rainfall, humidity, wind speed)
✓ Predict whether conditions are likely to lead to flooding
✓ Alert registered users in flood-prone areas via SMS
✓ Translate messages into their **regional/local languages**
✓ Allow city-specific weather checking through an intuitive web interface

The system is powered by:

- A trained **Decision Tree classifier** that learns from historical flood conditions,
- An API from OpenWeatherMap for **live environmental data**,
- **Twilio** for SMS-based communication,
- **Google Translate API** for regional message delivery,
- A **Flask-based web interface** using Bootstrap for responsiveness and visual clarity.

Real-World Implications

With real-time alerts and actionable information, this system can:

- **Save lives** by enabling timely evacuation and preparation,
- **Increase public awareness** about environmental threats,
- **Enhance governmental or NGO preparedness** in disaster management,
- **Provide a tech-based solution** that is cost-effective and easily deployable

Major Flood Events in India (1925–2025)

 The **Great Musi Flood of 1908** in Hyderabad occurred on September 28–29, killing over 15,000 people and causing damages worth **£1.25 million** — approximately **₹1,060 crore** in today's value.

 In 1987, **Bihar's Kosi River flooded**, killing around 1,399 people and 5,300 livestock. The estimated loss was **₹68 million**, or **₹6.8 crore**, in crops and public property.

 The **1998 Gujarat Flood** affected 21 districts, killed 156 people, and led to extensive damage to homes and agriculture — financial loss not precisely documented.

 The **1992 India–Pakistan Floods** killed approximately 2,496 people and submerged over 12,600 villages. Estimated damages were around **$1 billion**, which is approximately **₹8,300 crore**, and **3.3 million people** were evacuated.

 On July 26, 2005, **Mumbai received 944 mm of rain in 24 hours**, causing about 1,094 deaths and **₹450 crore** in property damage. Over 14,000 homes were affected.

 Also in 2005, **Gujarat experienced a major flood**, killing 131 people, flooding around 7,200 villages, and making **1.76 lakh people** homeless.

 On August 18, 2008, the **Kosi flood in Bihar** displaced **33 lakh people**, killed 527, and caused economic losses of **$1.2 billion**, or around **₹10,000 crore**.

 The **Ladakh flash flood** on August 6, 2010, killed around 255 people and damaged **71 villages and towns**.

 In 2012, **Assam's flood disaster** displaced **17 lakh people**, inundated thousands of hectares, and killed 124 people.

 The **Uttarakhand flood of 2013** led to more than **5,700 deaths** due to cloudbursts and landslides across nine districts.

 In 2014, **Jammu & Kashmir floods** killed 284 people and damaged large areas of infrastructure and documents as many villages were submerged.

 The **Chennai flood in 2015** (Nov–Dec) killed over 500 people (422 in Tamil Nadu alone), displaced **18 lakh people**, and caused damages worth **₹14,602 to ₹50,000 crore** (≈ **$2.2 to $7.5 billion**).

 The **Kerala flood of 2018** killed 483 people, displaced **12 lakh people**, and caused losses of about **₹30,000 crore**.

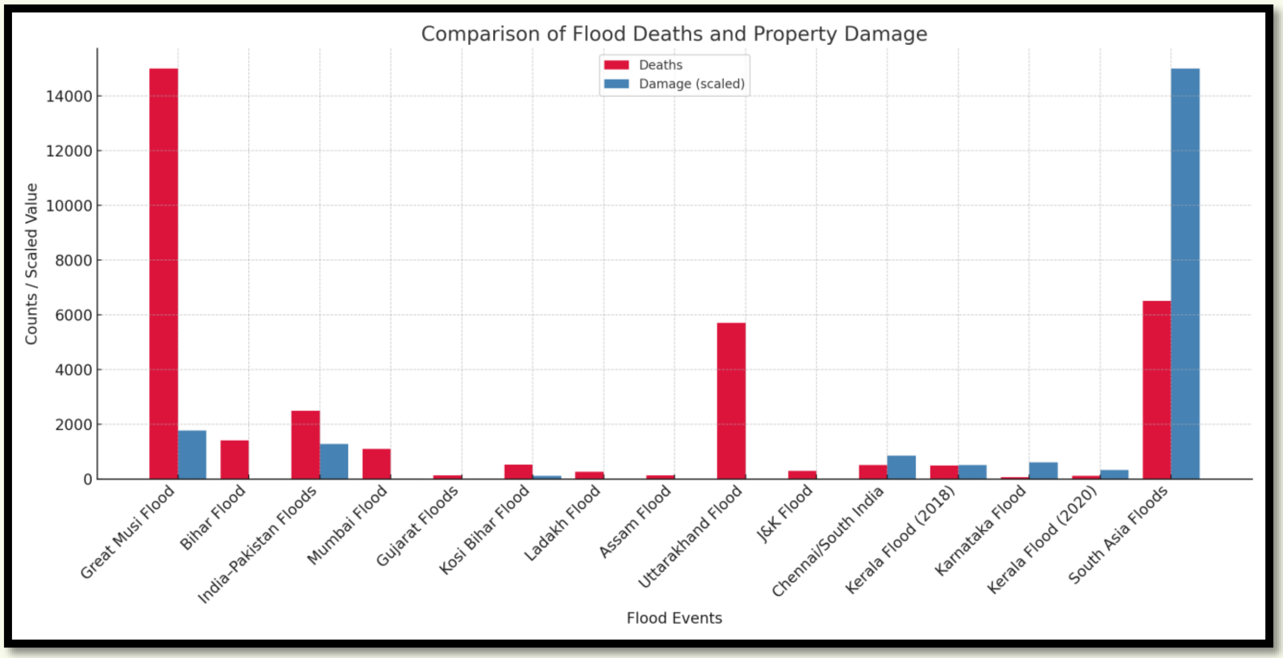
 From June to August 2020, **Kerala suffered another flood**, killing 104 people (plus 12 missing), and causing damages worth **₹19,000 crore**.

⍰ The **Karnataka flood of August 2019** killed 61 people (15 missing), destroyed over **56,000 houses**, displaced **nearly 7 lakh people**, and led to estimated losses of **₹35,160 crore** (≈ **$4.95 billion**).

⍰ The **2020 South Asian floods** from May to October caused **6,511 deaths across the region**. India alone accounted for a loss of **₹7.35 lakh crore** (≈ **$88.5 billion**) out of the global **₹8.7 lakh crore** ($105 billion).

📌 Summary of Impact (1925–2025)

- **Fatalities**: Tens of thousands during individual events; hundreds of thousands cumulatively.
- **Economic Damage**: Multi-trillion INR—₹30,000–₹50,000 crore per major flood (2015 Chennai, 2018 Kerala) and $88 billion losses in 2020.
- **Displacement & Infrastructure Loss**: Millions displaced; thousands of villages/suburbs severely damaged.
- **Climate Influence**: Increasing flood frequency and severity linked to climate change and extreme weather patterns



ABOUT THE PROJECT

BACKGROUND & MOTIVATION

Floods have historically been among the most devastating natural disasters, causing loss of life, property damage, and economic disruption.
As the effects of climate change continue to intensify, rainfall patterns are becoming increasingly unpredictable, leading to more frequent and severe flooding events.
Traditional flood alert systems often lack real-time responsiveness, local language communication, and accessibility for rural or low-income populations.

This project — the Flood Alert & Weather Monitoring System — aims to address these challenges by combining machine learning, real-time weather APIs, user data, translation services, and SMS communication into a unified platform that can assess risk and alert users automatically and efficiently.

OBJECTIVES OF THE PROJECT

1. To predict flood risk in Indian cities using live weather data (rainfall, humidity, wind speed).
2. To alert users in their regional language via SMS if they are in an affected area.
3. To allow users to check real-time weather details for specific cities.
4. To create an interactive and visually appealing web application using Flask and Bootstrap.
5. To build a scalable system that can incorporate more data and features over time.

SYSTEM ARCHITECTURE OVERVIEW

The project follows a modular architecture with the following key components:

- **Weather Data Retrieval**: Fetches live weather metrics from the OpenWeatherMap API.
- **Machine Learning Model**: A decision tree classifier trained on historical weather data to predict flood risk.
- **Alert System**: Automatically sends SMS alerts via Twilio to users in risk-prone areas, with language translation provided by Google Translate.
- **Web Interface**: A Flask-based responsive web interface that allows users to initiate flood checks or retrieve weather data for a city.
- **User Data Management**: Reads from user_data.csv to identify affected users and their preferred languages.

WORKFLOW

1. The user opens the home page and selects either "Check Flood Risk" or "Check City Weather".
2. If "Check Flood Risk" is selected:
    - The system iterates over all cities listed in the user database.
    - For each city, weather data is fetched from OpenWeatherMap.
    - The decision tree model predicts whether a flood risk exists.
    - If risk is "Severe" or "Extreme", an SMS alert is sent to all users in that city.
3. If "Check City Weather" is selected:
    - The user selects a city from a dropdown.
    - The system fetches and displays weather details including temperature, humidity, rainfall, and wind speed.

TECHNOLOGIES USED

- **Flask** —
    - Backend and Web Framework
- **Scikit-learn** —
    - For building the ML decision tree model
- **Pandas** —
    - For reading and manipulating CSV data
- **OpenWeatherMap API** —
    - For fetching real-time weather conditions
- **Twilio API** —
    - For sending SMS messages to users
- **Googletrans** —
    - For translating alert messages into native languages
- **HTML/CSS + Bootstrap** —
    - For creating responsive and styled frontend interfaces

ADVANTAGES OF THE SYSTEM

- **Real-Time Data**:
  Live weather updates provide timely predictions.
- **Localization**:
  Alerts are translated and personalized to each user.
- **Accessibility**:
  SMS-based alerts ensure outreach to non-internet users.
- **Scalability**:
  More data and predictive models can be integrated easily.
- **Easy-to-Use Interface**:
  Clean UI for checking flood risk or city weather.

USER INTERFACE FEATURES

- **Landing Page**:
  - Offers two options — flood check and city weather check.
- **Flood Risk Result**:
  - Displays at-risk cities and affected user names.
- **Safe Zone Message**:
  - Informs that no risk has been detected.
- **City Weather Checker**:
  - Dropdown interface to select and view city weather.
- **Background Animation**:
  - Moving video-based weather backgrounds to enhance visual engagement.

💬 CUSTOMER SUPPORT SYSTEM

To ensure that users—especially first-time visitors or those unfamiliar with technology—can navigate and understand the system easily, we have integrated an **interactive Customer Support FAQ module** into the homepage of the Flood Alert & Weather Monitoring System.

🔷 Key Features and Benefits:

✅ One-Click Accessibility

- The customer support section can be accessed instantly with a **prominent button labeled "💬 Customer Support"** on the landing page.
- This button is centrally positioned below the primary actions ("Check Flood Risk" and "Check City Weather") to ensure visibility and ease of use.

✅ Dynamic Question Dropdown

- Once clicked, the module reveals a **sleek dropdown menu** built using **Bootstrap's modern styling**.
- The dropdown contains **15 frequently asked questions** that address the most common user concerns, categorized across:
-
  - Flood alert process
  - Prediction accuracy
  - Registration and charges
  - Alert language customization
  - Emergency guidance
  - Data security
  - Technical support
  - Service coverage

✓ Instant, Interactive Answers

- When a user selects a question from the dropdown, a relevant and pre-written answer is immediately displayed **without refreshing or navigating away from the page**.
- This is achieved using **JavaScript's onchange() event listener** to dynamically update the content in a <div> container (#faq-answer).
- Answers are shown in a **stylized information box**, making them visually distinct and easy to read.

✓ Fully Responsive Design

- The module is fully responsive and works seamlessly across:
  - Mobile devices
  - Tablets
  - Desktops
- The combination of **clean layout**, **rounded edges**, and **subtle shadows** gives it a **professional and welcoming appearance**.

✓ User-Centric and Accessible

- By removing the need to email or call for basic support, this feature makes it easy for users to:
  - Quickly find answers on their own
  - Understand what actions to take during a flood alert
  - Feel more confident using the system
- It's particularly useful for **non-tech-savvy users**, **rural populations**, and **elderly citizens** who may be unfamiliar with modern web services.

✓ MODIFICATIONS & ENHANCEMENTS IMPLEMENTED

To modernize the system and improve the user experience, the following major updates were implemented:

◆ Front-End Enhancements:

- Created a **fully responsive homepage** using Bootstrap and custom CSS for modern aesthetics.
- Integrated **animated backgrounds** using video or images depending on user preference or browser support.
- Styled buttons with **gradient colors**, **hover animations**, and **neumorphic design elements**.
- Designed an **interactive customer support** section with smart dropdown Q&A.

◆ City Weather Interface:

- Upgraded the /city_weather page with a **card-based layout**, **timestamp visibility**, and clear color indicators for risk status.

◆ Flood Risk Alerts Page:

- Used a modern table/list layout with styled checkboxes for affected cities.
- Enhanced alert visibility with color-coded warnings.

◆ SMS Status Result Page:

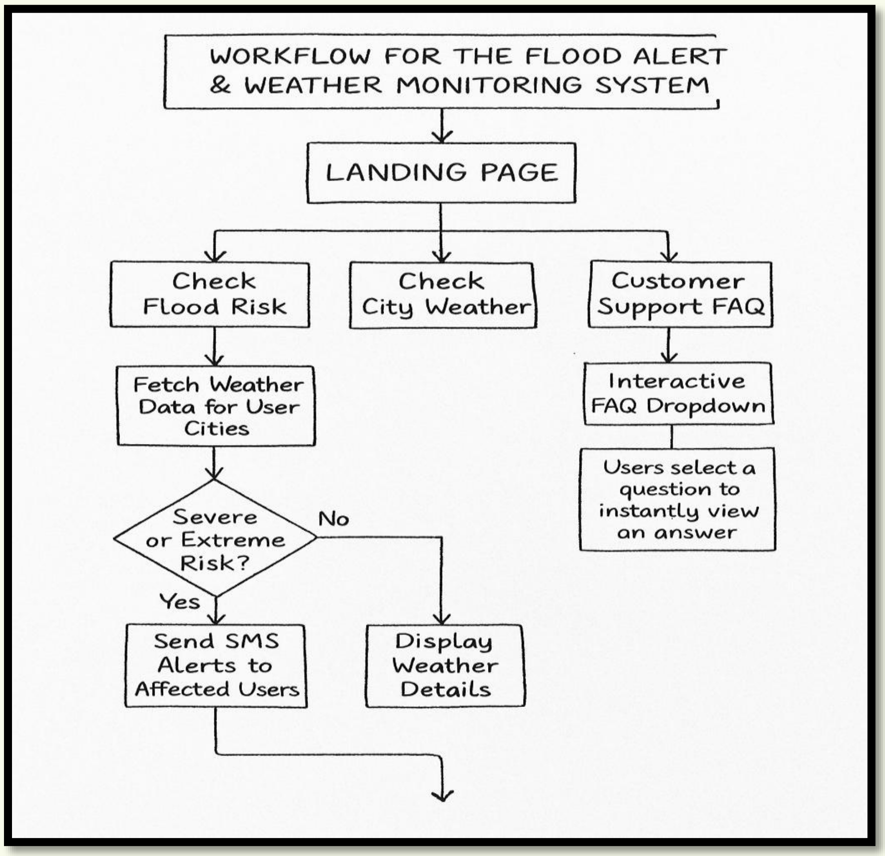
- Replaced plain lists with **Bootstrap-styled alert cards**, improving readability and UX.

◆ Static Resources Organization:

- Set up standard folders:
  - /static/images for weather-related images.
  - /static/videos for background video support.
  - /static/css for custom style sheets (if needed).

◆ Backend Improvements:

- Modified /city_weather to return real-time **timestamped weather details**.
- Standardized weather display to ensure **clarity, speed, and responsiveness**.


WORKFLOW FOR THE FLOOD ALERT & WEATHER MONITORING SYSTEM
LANDING PAGE
Check Flood Risk
Check City Weather
Customer Support FAQ
Fetch Weather Data for User Cities
Interactive FAQ Dropdown
Users select a question to instantly view an answer
Severe or Extreme Risk?
No
Yes
Send SMS Alerts to Affected Users
Display Weather Details

🚀 JOURNEY THROUGH THE CODE: UNDERSTANDING (app.py)

This document walks through each part of your Flask-based **Flood Alert System** backend logic, helping you understand how everything works—step by step.

◆ 1. Importing Required Libraries

```
from flask import Flask, render_template, request
import pandas as pd
import requests
import pickle
from deep_translator import GoogleTranslator
from twilio.rest import Client
from datetime import datetime
import os
```

- Flask: Creates the web app and routes.
- pandas: Reads CSV files (flood_data.csv, user_data.csv).
- requests: Sends HTTP requests to fetch weather data.
- pickle: Loads the saved ML model.
- deep_translator: Translates flood alert messages into different languages.
- twilio: Sends SMS messages to users.
- datetime: Adds current timestamp for weather info.
- os: Checks file existence.

◆ 2. App & API Configuration

```
app = Flask(__name__)
```

- Starts the Flask app.

```
API_KEY = "..."; TWILIO_SID = "..."; ...
```

- Sets API keys for:
  - OpenWeatherMap (real-time weather).
  - Twilio (SMS service).

◆ 3. Load Data from CSV

```
flood_data = pd.read_csv("flood_data.csv")
user_data = pd.read_csv("user_data.csv")
```

- Reads flood_data.csv (used to train the model).
- Reads user_data.csv (contains user details).
- Column names are cleaned using .str.strip().str.lower().

◆ 4. Load Pretrained Model

```
if not os.path.exists("flood_model.pkl"):
    raise FileNotFoundError("❌ Model not found.")
```

- Checks if the trained model (flood_model.pkl) is available.
- If not, it raises an error.

```
with open("flood_model.pkl", "rb") as file:
    flood_model = pickle.load(file)
```

- Loads the model using pickle.

◆ 5. Weather Fetching Function

```
def fetch_weather(city):
```

- Sends request to OpenWeatherMap.
- Retrieves: **rainfall, humidity, wind speed, temperature, description**.
- If data not available, raises an exception.

◆ 6. Send Alert via SMS

```
def send_alert_sms(name, phone, message, lang='te', real=False):
```

- Translates the alert to the user's language (default: Telugu).
- Sends the SMS via Twilio **only if** real=True.
- Returns alert details.

◆ 7. Routes (Flask Views)

🏠 / – Home

```
@app.route("/")
def index():
    return render_template("index.html")
```

- Loads the home page UI.

🌧️ /check_flood – Auto Flood Detection

```
@app.route("/check_flood", methods=["POST"])
def check_flood_risk():
```

- For each city in user_data.csv:
  - Fetch weather via API.
  - Predict flood **type** and **risk** using model.
  - If risk is **severe/extreme**:
    - Send SMS to users.
    - Store details in status[].

- Displays:
    - affected.html if flood detected.
    - no_flood.html if safe.

/manual_input – Manual Input Form

```
@app.route("/manual_input")
```

- Loads manual_input.html form with city dropdown.

/manual_check – Manual Risk Check

```
@app.route("/manual_check", methods=["POST"])
def manual_check():
```

- Takes user-entered **rainfall, humidity, wind speed**.
- Predicts flood type and risk.
- If severe/extreme, sends alerts to users in that city.
- Shows:
    - result.html if risky.
    - no_flood.html if safe.

/city_weather – Weather Info by City

```
@app.route("/city_weather", methods=["GET", "POST"])
```

- Lets users check weather for a selected city.
- Shows: rainfall, humidity, wind speed, temperature.
- Adds current timestamp.

8. Start the Flask App

```
if __name__ == "__main__":
    app.run(debug=True)
```

- Runs the web app in debug mode for development.

DATASET DESCRIPTIONS

1. flood_data.csv – Weather-Based Flood Analysis Data

This dataset is the foundation for training a machine learning model to predict flood risk levels based on meteorological parameters. It contains **historical or synthetically generated records** representing various weather conditions and their corresponding flood effects. The columns in this dataset are:

Column Name	Description
rainfall	The amount of rainfall measured in millimeters (mm) for a given hour/day. It's one of the most critical indicators used in flood prediction.
humidity	The atmospheric humidity percentage during the observation. Higher humidity can indicate heavy rain possibilities.
wind_speed	Wind speed in meters per second (m/s). In severe weather conditions, high wind speeds often accompany heavy rains and storms.
flood effects	The label or output class representing the level of flood severity. It may include categories like "None," "Mild," "Moderate," "Severe," or "Extreme."

✳ Usage in Model:

This dataset is used to train a **Decision Tree Classifier**. The input features (rainfall, humidity, wind_speed) are passed into the model to predict the flood effects. Based on the predictions (severe or extreme), the application takes action, such as sending SMS alerts to users in affected areas.

Example Row:

Rainfall	Humidity	Wind_Speed	Flood Effects
34.5	86	5.2	Severe

This row would train the model to associate high rainfall and humidity with a severe flood risk.

2. user_data.csv – User Location and Contact Database

This dataset contains user-specific information necessary for identifying potential flood victims and delivering alerts via SMS. Each row represents an individual user with essential demographic and locational details.

Column Name	Description
name	Full name of the user, used in personalized alert messages.
phone	The mobile number of the user. Alerts are sent to this number using the Twilio API.
address	The user's complete address. This may not be directly used in the model but is valuable for reference.
city	The user’s city or location. Weather data is fetched from OpenWeatherMap based on this city to assess the flood risk.
near_river	A boolean or categorical field (like Yes/No) indicating whether the user's residence is near a river or flood-prone zone. Useful for prioritizing alerts.

✳ Usage in App:

- In the **/check_flood** route, all cities from this dataset are looped through.
- Real-time weather data is fetched for each city.
- Based on the model prediction, users in “at-risk” cities are sent SMS alerts.
- The interface also uses this dataset to populate dropdowns for weather checks by city.

Example Row:

Name	Phone	Address	City	Near_River
Priya Rao	+91-9876543210 123	MG Road, Hyderabad	Hyderabad	Yes

This row enables the system to:

- Fetch Hyderabad weather data,
- Predict flood risk,
- And if necessary, send an alert to Priya via Twilio with a message like:
  "Dear Priya Rao, Heavy rain & flood risk in your area. Stay safe."

How These Datasets Work Together

The **flood_data.csv** serves as the training data for the model to understand what combinations of weather features are likely to result in floods. The **user_data.csv** provides the list of real users, where they live, and how they can be contacted.

The application uses these two datasets synergistically:

- The model predicts risk,
- The interface visualizes it,
- The system sends real-time alerts.

Together, these datasets empower your system to not only predict natural disasters but also take proactive steps to minimize their impact through communication.

create_model.py – Model Training & Saving Script

This file is responsible for **building and saving the prediction model**.

Key Steps:

1. **Load Historical Data**
   - Loads flood_data.csv which contains past weather data and corresponding flood outcomes.
2. **Clean the Dataset**
   - Ensures column names are clean (e.g., all lowercase, no extra spaces) for consistency.
3. **Select Inputs and Outputs**
   - **Inputs (X):** rainfall, humidity, wind_speed
   - **Outputs (y):** flood_type (e.g., flash, river) and flood_effects (e.g., low, moderate, severe)
4. **Train the Model**
   - A DecisionTreeClassifier is used inside a MultiOutputClassifier to **predict two things at once**: flood type and severity.
5. **Save the Trained Model**
   - The trained model is saved into a file named **flood_model.pkl** using the pickle module.
   - This saved file will later be reused without retraining the model every time.
6. **Output**
   - Prints a success message like ✓ Model saved to flood_model.pkl successfully.

💾 What is flood_model.pkl?

- It is a **serialized file** that stores your trained flood prediction model.
- It **remembers the patterns** from flood_data.csv.
- This file is used by your main application (app.py) to **predict flood type and risk** based on current weather conditions.
- You only need to create it **once**, unless your data changes

✓ check_model.py – Model Testing Script

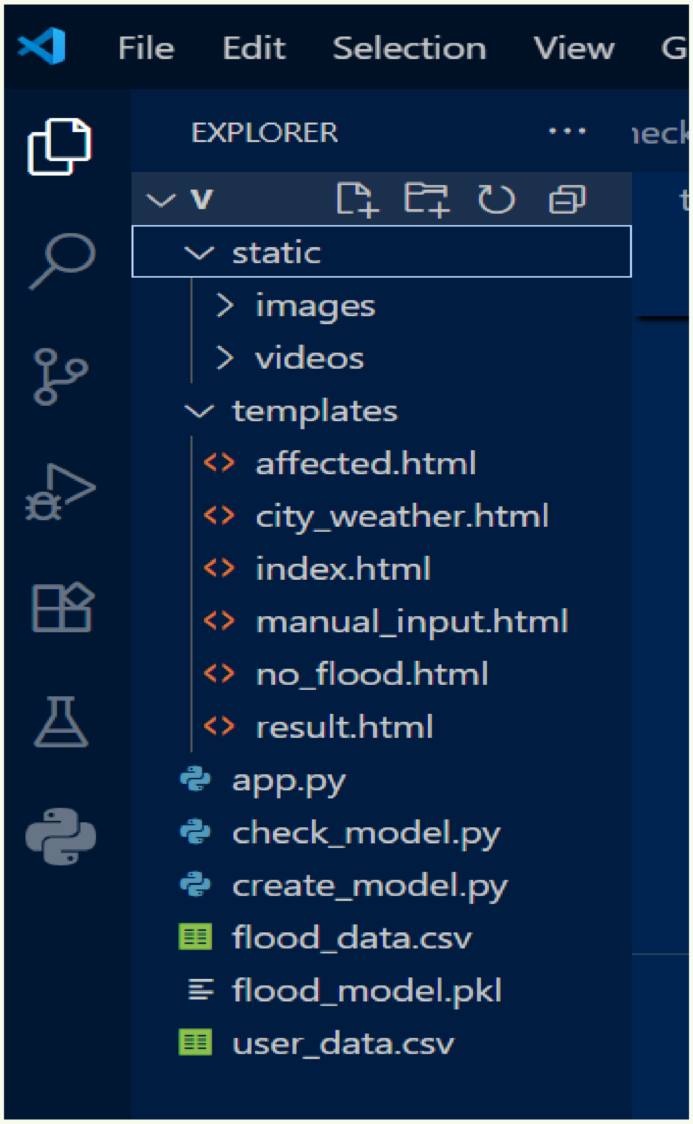
This script is used to **test if the saved model is working properly** with example inputs.

Key Steps:

1. **Load the Model**
   - Loads flood_model.pkl so you can make predictions without retraining.
2. **Prepare Sample Weather Input**
   - Defines test data like rainfall, humidity, and wind speed to simulate real conditions.

3. **Predict Using the Model**
   - The model predicts both:
     - **Flood Type** (e.g., river, flash)
     - **Flood Effect/Severity** (e.g., low, moderate, severe)
4. **Display Results**
   - Prints the prediction to verify that the model is working correctly.

FILE STRUCTURE:



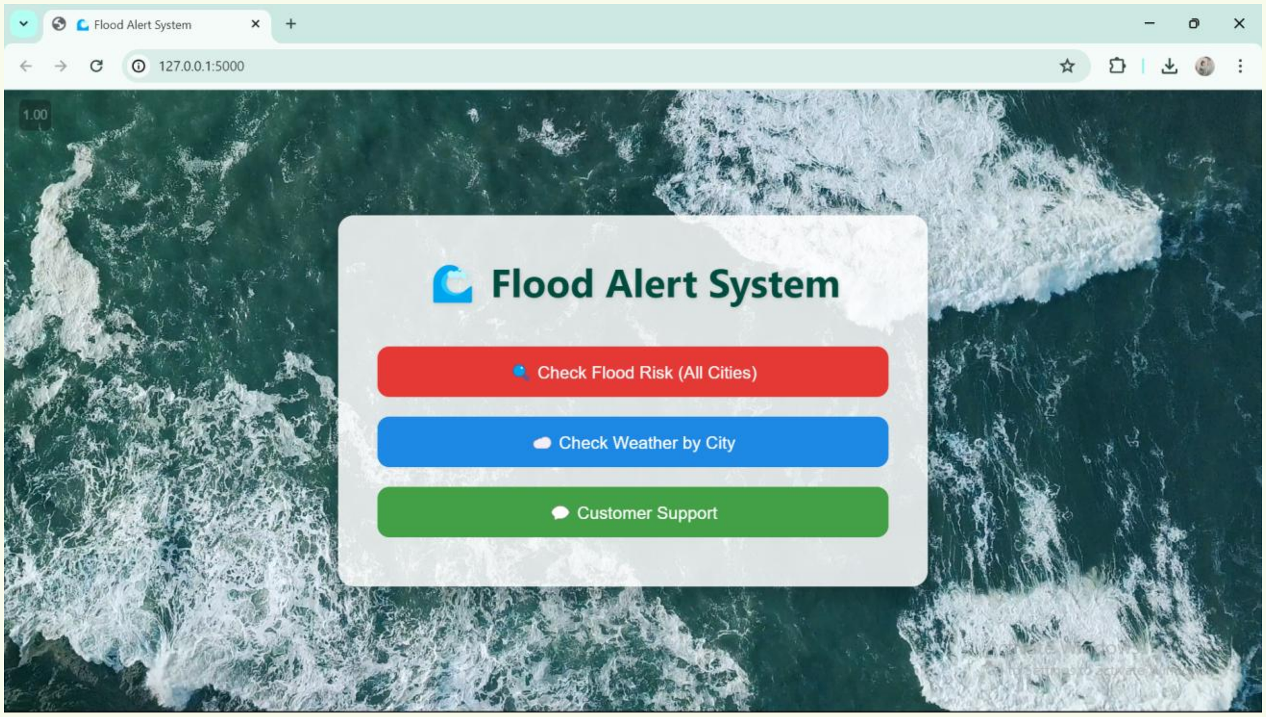
```
v
  static
    images
    videos
  templates
    affected.html
    city_weather.html
    index.html
    manual_input.html
    no_flood.html
    result.html
  app.py
  check_model.py
  create_model.py
  flood_data.csv
  flood_model.pkl
  user_data.csv
```

FLOOD ALERT SYSTEM - TEMPLATE FILE DOCUMENTATION

1. index.html – Home Page

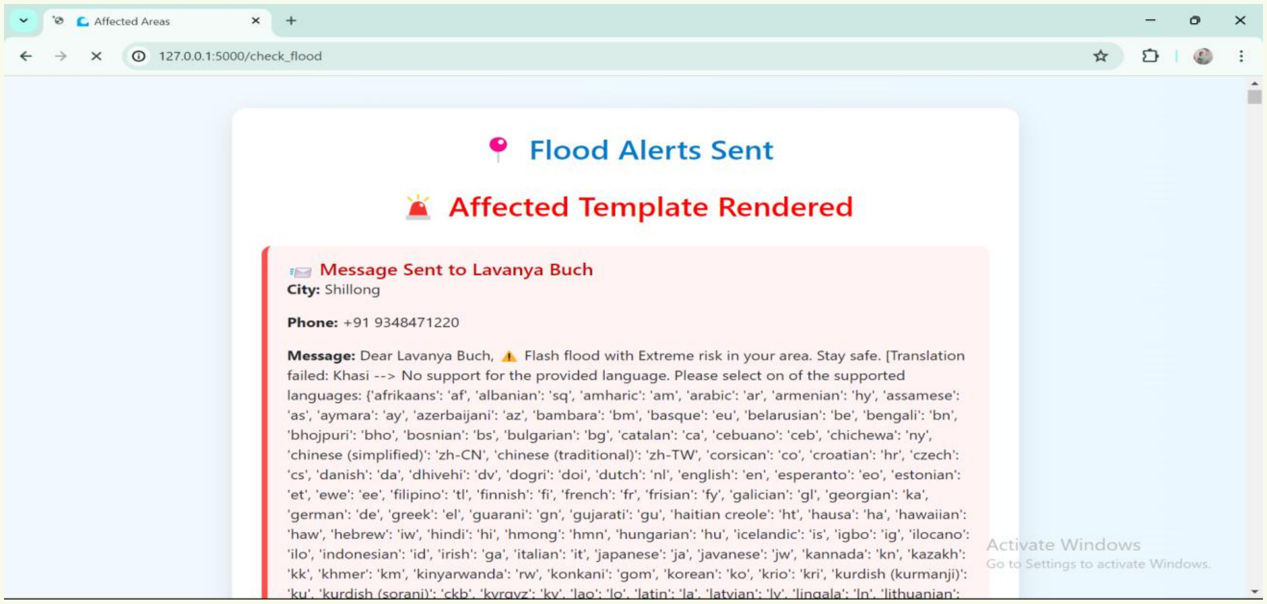
This is the **landing page** of the Flood Alert & Weather Monitoring System and serves as the user's **first point of interaction** with the application. It is designed to be clean, responsive, and visually engaging using Bootstrap 5 and custom CSS.

The home page provides **three primary actions** for users, each accessible through clearly labeled buttons:



✓ 1. 🌊 Check Flood Risk

- This button sends a **POST** request to the /check_flood route.
- When clicked, it activates the server-side flood detection mechanism.
- The backend system:
  - Fetches **real-time weather data** for each city listed in the user_data.csv file.
  - Applies a **Decision Tree machine learning model** to predict the flood risk based on features like rainfall, humidity, and wind speed.



Flood Alerts Sent

Affected Template Rendered

Message Sent to Lavanya Buch
City: Shillong

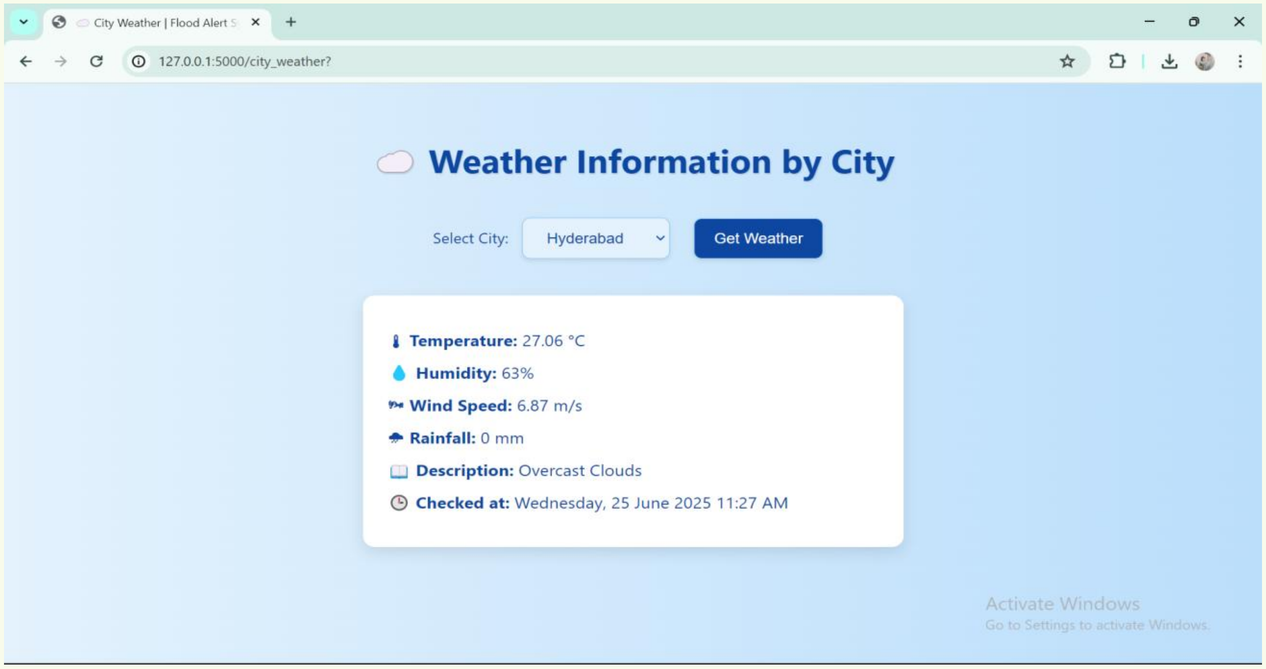
Phone: +91 9348471220

Message: Dear Lavanya Buch, ⚠️ Flash flood with Extreme risk in your area. Stay safe. [Translation failed: Khasi --> No support for the provided language. Please select on of the supported languages: {'afrikaans': 'af', 'albanian': 'sq', 'amharic': 'am', 'arabic': 'ar', 'armenian': 'hy', 'assamese': 'as', 'aymara': 'ay', 'azerbaijani': 'az', 'bambara': 'bm', 'basque': 'eu', 'belarusian': 'be', 'bengali': 'bn', 'bhojpuri': 'bho', 'bosnian': 'bs', 'bulgarian': 'bg', 'catalan': 'ca', 'cebuano': 'ceb', 'chichewa': 'ny', 'chinese (simplified)': 'zh-CN', 'chinese (traditional)': 'zh-TW', 'corsican': 'co', 'croatian': 'hr', 'czech': 'cs', 'danish': 'da', 'dhivehi': 'dv', 'dogri': 'doi', 'dutch': 'nl', 'english': 'en', 'esperanto': 'eo', 'estonian': 'et', 'ewe': 'ee', 'filipino': 'tl', 'finnish': 'fi', 'french': 'fr', 'frisian': 'fy', 'galician': 'gl', 'georgian': 'ka', 'german': 'de', 'greek': 'el', 'guarani': 'gn', 'gujarati': 'gu', 'haitian creole': 'ht', 'hausa': 'ha', 'hawaiian': 'haw', 'hebrew': 'iw', 'hindi': 'hi', 'hmong': 'hmn', 'hungarian': 'hu', 'icelandic': 'is', 'igbo': 'ig', 'ilocano': 'ilo', 'indonesian': 'id', 'irish': 'ga', 'italian': 'it', 'japanese': 'ja', 'javanese': 'jw', 'kannada': 'kn', 'kazakh': 'kk', 'khmer': 'km', 'kinyarwanda': 'rw', 'konkani': 'gom', 'korean': 'ko', 'krio': 'kri', 'kurdish (kurmanji)': 'ku', 'kurdish (sorani)': 'ckb', 'kyrgyz': 'ky', 'lao': 'lo', 'latin': 'la', 'latvian': 'lv', 'lingala': 'ln', 'lithuanian':

- If the predicted risk is **Severe** or **Extreme**, an **SMS alert is sent** to users in the affected cities using Twilio, translated to their **preferred language** using Google Translate. If no flood is predicted, a "Safe Zone" message is shown.

✅ 2. ☀️ Check City Weather

- This button triggers a **GET** request to the /city_weather route.
- The user is redirected to a new page where they can select a **specific city** from a dropdown menu.
- The system fetches and displays:
  - Current temperature
  - Humidity
  - Wind speed
  - Rainfall



Weather Information by City

Temperature: 27.06 °C

Humidity: 63%

Wind Speed: 6.87 m/s

Rainfall: 0 mm

Description: Overcast Clouds

Checked at: Wednesday, 25 June 2025 11:27 AM

- Weather description (e.g., cloudy, rainy)

Timestamp is also shown for when the data was retrieved.

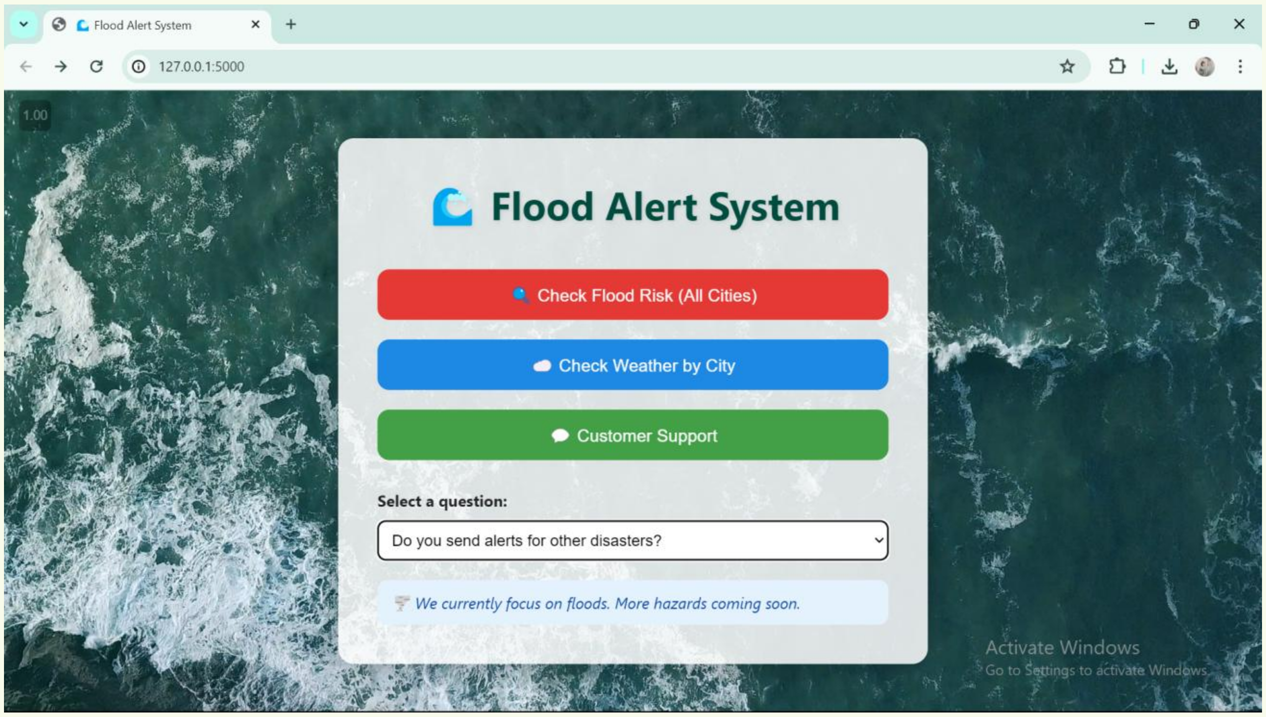
✅ 3. 💬 Customer Support Integration (FAQ)

To enhance the experience, the page also includes an option for **Customer Support**. A small button or link (e.g., "Need Help?") can direct the user to an FAQ section or support form. This feature is vital for non-technical users who might be unsure about why a particular city is flagged or what the SMS will contain. It helps improve user trust and provides a fallback during technical glitches or confusion

The FAQ section may answer questions like:

- "Can I change the alert language?"
- "Why did I receive this warning?"
- "What do I do after receiving an alert?"

Having **support built-in** ensures that users aren't left helpless during critical moments.



We currently focus on floods. More hazards coming soon.

2. no_flood.html – Safe Zones Display

This page is rendered **only when the flood prediction system detects no high-risk zones**. It acts as a **reassurance screen** for users, letting them know that all cities are currently safe based on the latest weather data and machine learning predictions.

When is it triggered?

- This page is triggered in the /check_flood route **after evaluating every city** using the decision tree classifier.
- If the model does **not detect any "Severe" or "Extreme" flood risk** in any of the cities listed in user_data.csv, the application **renders no_flood.html** instead of showing flood alerts.

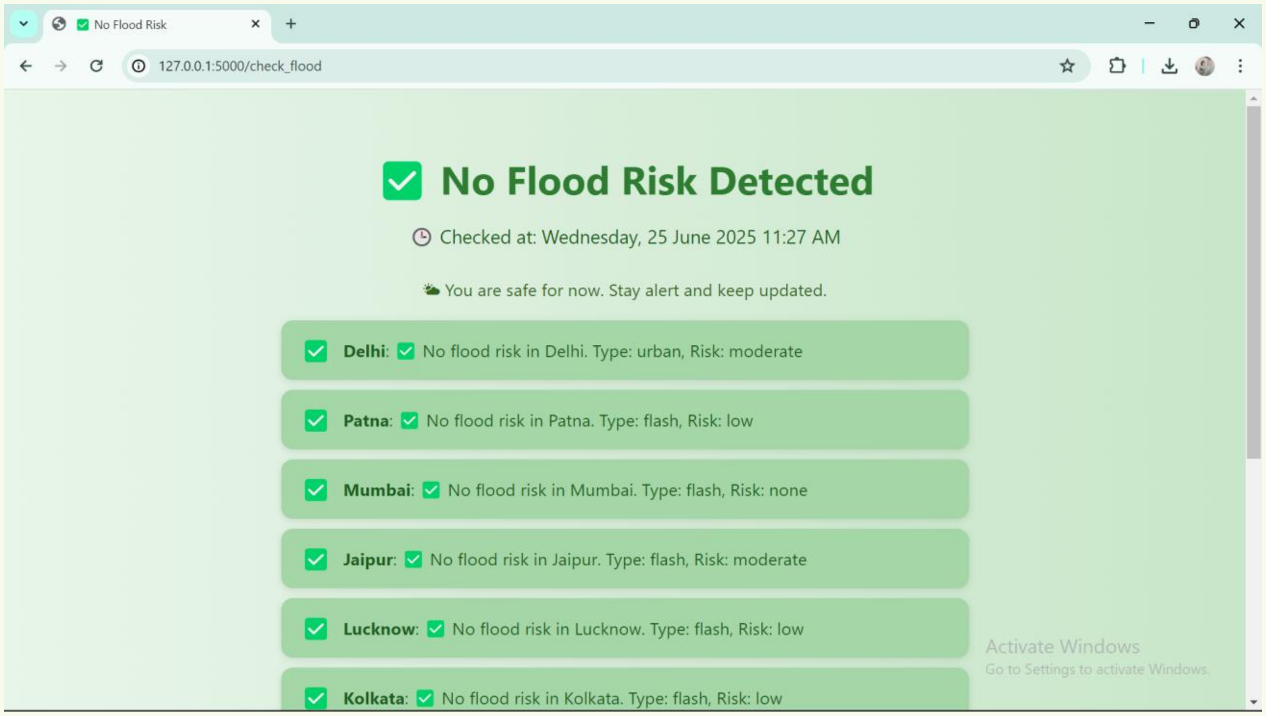
💬 Message to Users

The core message is delivered through a **friendly, calm, and optimistic tone** using a combination of **emojis** and **positive reinforcement** statements such as:

- ✅ **All regions are safe!**
- 🚫 **No flood risk detected.**
- 🌻 **Be prepared, stay aware — but today is clear!**

These messages are wrapped inside a visually appealing notification box with:

- Light green or blue backgrounds
- Rounded corners
- Larger font and icons for readability



✅ No Flood Risk Detected

🕒 Checked at: Wednesday, 25 June 2025 11:27 AM

🌥 You are safe for now. Stay alert and keep updated.

✅ **Delhi**: ✅ No flood risk in Delhi. Type: urban, Risk: moderate

✅ **Patna**: ✅ No flood risk in Patna. Type: flash, Risk: low

✅ **Mumbai**: ✅ No flood risk in Mumbai. Type: flash, Risk: none

✅ **Jaipur**: ✅ No flood risk in Jaipur. Type: flash, Risk: moderate

✅ **Lucknow**: ✅ No flood risk in Lucknow. Type: flash, Risk: low

✅ **Kolkata**: ✅ No flood risk in Kolkata. Type: flash, Risk: low

📱 User Experience Goals

The goal of this template is to:

- **Inform users calmly** that there is no flood risk at the moment
- Avoid unnecessary fear or anxiety
- Maintain a **trustworthy and transparent communication channel**

It contributes to the overall **empathy-driven design philosophy** of the application — focusing on safety, simplicity, and user reassurance.

🌊 3. result.html – Flood Risk Alert Result Page

The result.html page is one of the most important parts of your project — it acts like the **alert dashboard** 🚨 where users (or admins) are informed when flood risks have been detected across any city in India.

This page gets triggered **after a user clicks the "🌊 Check Flood Risk" button** on the homepage (index.html). Behind the scenes, your system checks weather data using OpenWeatherMap's API 📡, runs it through a trained decision tree classifier 🌳, and decides whether the risk in each city is *Safe*, *Severe*, or *Extreme*.

If even **one city is found at risk**, the result.html page will be shown. Otherwise, the user is redirected to the more relaxing no_flood.html page ☺.

✅ Purpose of the Page

The result.html page serves as a **confirmation screen** to show:

- ✔️ **Which cities are at risk**
- 📨 **That alerts were sent** to users living in those cities
- **That the prediction model worked properly**

This keeps users **informed** and builds trust that the system is taking real action, not just showing visuals.

📋 How It Works

The Flask backend gathers all cities from the database (user_data.csv) and checks real-time weather for each. If any city's conditions (rainfall 🌧️, humidity 💧, wind speed 🌬️) are dangerous, it:

1. Predicts "Severe" or "Extreme" 🌩️ using the model
2. Translates the alert message into the user's native language 🌐 (via Google Translate)
3. Sends SMS messages via Twilio 📲
4. Passes the affected city list (alerts) to result.html

Now result.html uses **Jinja templating** to loop through that list and display each city clearly like:

📍 Hyderabad – Alert Sent!
📍 Chennai – Alert Sent!
📍 Mumbai – Alert Sent!

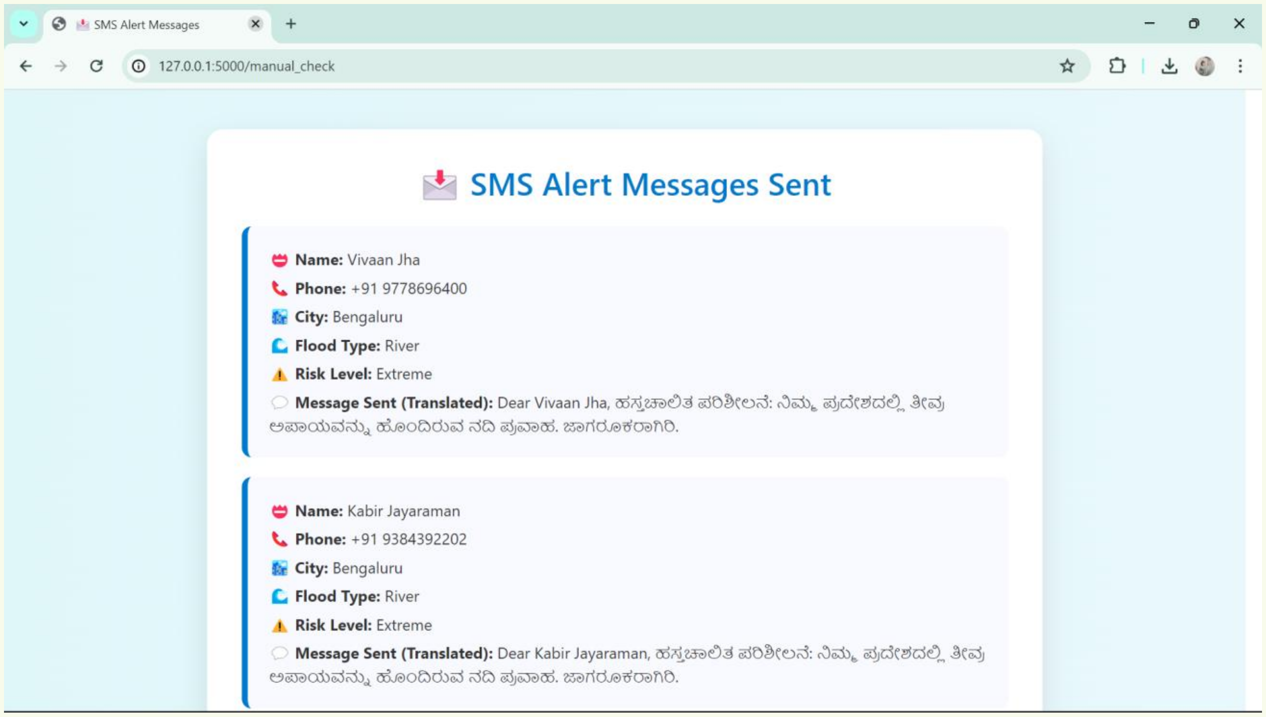
Each alert message reassures that users in these cities have been **warned in time**.

💡 User Interface & Feel

The visual feel is important here. Using Bootstrap, you can wrap the messages in:

- Red alert boxes
- ✅ Green confirmations
- 🏙 City cards with risk level info

This makes the page easy to scan and understand — even for someone with no technical background.



📩 SMS Alert Messages Sent

Name: Vivaan Jha
Phone: +91 9778696400
City: Bengaluru
Flood Type: River
Risk Level: Extreme
Message Sent (Translated): Dear Vivaan Jha, ಹಸ್ತಚಾಲಿತ ಪರಿಶೀಲನೆ: ನಿಮ್ಮ ಪ್ರದೇಶದಲ್ಲಿ ತೀವ್ರ ಅಪಾಯವನ್ನು ಹೊಂದಿರುವ ನದಿ ಪ್ರವಾಹ. ಜಾಗರೂಕರಾಗಿರಿ.

Name: Kabir Jayaraman
Phone: +91 9384392202
City: Bengaluru
Flood Type: River
Risk Level: Extreme
Message Sent (Translated): Dear Kabir Jayaraman, ಹಸ್ತಚಾಲಿತ ಪರಿಶೀಲನೆ: ನಿಮ್ಮ ಪ್ರದೇಶದಲ್ಲಿ ತೀವ್ರ ಅಪಾಯವನ್ನು ಹೊಂದಿರುವ ನದಿ ಪ್ರವಾಹ. ಜಾಗರೂಕರಾಗಿರಿ.

🌍 4. affected.html – Affected Cities & Alert Confirmation Page

The affected.html page is an essential part of the **Flood Alert & Weather Monitoring System**. It appears immediately after the system identifies cities with potential flood threats and prepares to alert affected users. This page acts as a **final review dashboard** before actual SMS alerts are sent, giving administrators or users a clear view of which cities are impacted and which users will receive notifications.

🎯 Main Purpose

The key purpose of this page is to:

- ✅ Display a list of cities where a flood risk has been predicted.
- 👥 Show the users in those cities who are registered in the system.
- 🌐 Display each user's preferred language so alerts can be customized.
- ☑️ Allow the admin or user to select or deselect which users to notify.
- 🚀 Provide a "Send Alerts" button that initiates the SMS alert process.

By giving manual control before sending automated alerts, this page **enhances reliability and reduces false alerts**, which is crucial during emergency situations.

📋 How It Fits in the Workflow

Once the user on the homepage clicks "🌊 Check Flood Risk," the system evaluates real-time weather data for all cities in the user database. Using a machine learning model, it classifies each city as "Safe," "Moderate," "Severe," or "Extreme." If any city is labeled as "Severe" or "Extreme," it is added to a list of flood-prone areas.

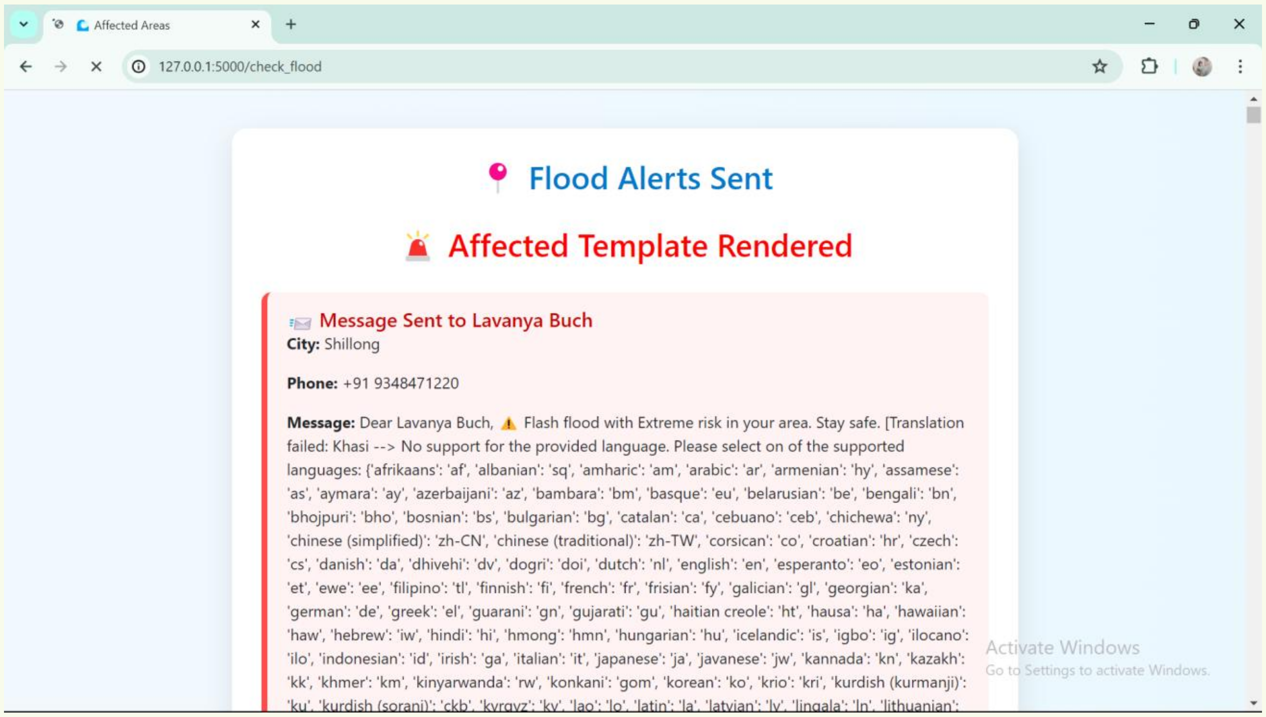
Then, for each such city, the system checks the database (user_data.csv) for registered users living there and prepares a list of these users along with their mobile numbers and preferred languages. That's where affected.html comes in — it displays this entire summary before alerts are sent out.

What the User Sees

Users or administrators visiting this page will see:

- 🏙️ A list of affected cities
- 👤 Usernames and associated phone numbers (optional)
- 🌐 Languages such as Hindi, Telugu, Tamil, etc., in which the alert will be translated
- ✅ A checkbox for each entry to allow selective alerting
- 📨 A button to send the actual alerts via SMS

This design ensures transparency and allows real-time verification of flood warnings before mass notifications are sent out.



📍 Flood Alerts Sent

🚨 Affected Template Rendered

📨 Message Sent to Lavanya Buch
City: Shillong

Phone: +91 9348471220

Message: Dear Lavanya Buch, ⚠️ Flash flood with Extreme risk in your area. Stay safe. [Translation failed: Khasi --> No support for the provided language. Please select on of the supported languages: {'afrikaans': 'af', 'albanian': 'sq', 'amharic': 'am', 'arabic': 'ar', 'armenian': 'hy', 'assamese': 'as', 'aymara': 'ay', 'azerbaijani': 'az', 'bambara': 'bm', 'basque': 'eu', 'belarusian': 'be', 'bengali': 'bn', 'bhojpuri': 'bho', 'bosnian': 'bs', 'bulgarian': 'bg', 'catalan': 'ca', 'cebuano': 'ceb', 'chichewa': 'ny', 'chinese (simplified)': 'zh-CN', 'chinese (traditional)': 'zh-TW', 'corsican': 'co', 'croatian': 'hr', 'czech': 'cs', 'danish': 'da', 'dhivehi': 'dv', 'dogri': 'doi', 'dutch': 'nl', 'english': 'en', 'esperanto': 'eo', 'estonian': 'et', 'ewe': 'ee', 'filipino': 'tl', 'finnish': 'fi', 'french': 'fr', 'frisian': 'fy', 'galician': 'gl', 'georgian': 'ka', 'german': 'de', 'greek': 'el', 'guarani': 'gn', 'gujarati': 'gu', 'haitian creole': 'ht', 'hausa': 'ha', 'hawaiian': 'haw', 'hebrew': 'iw', 'hindi': 'hi', 'hmong': 'hmn', 'hungarian': 'hu', 'icelandic': 'is', 'igbo': 'ig', 'ilocano': 'ilo', 'indonesian': 'id', 'irish': 'ga', 'italian': 'it', 'japanese': 'ja', 'javanese': 'jw', 'kannada': 'kn', 'kazakh': 'kk', 'khmer': 'km', 'kinyarwanda': 'rw', 'konkani': 'gom', 'korean': 'ko', 'krio': 'kri', 'kurdish (kurmanji)': 'ku', 'kurdish (sorani)': 'ckb', 'kyrgyz': 'ky', 'lao': 'lo', 'latin': 'la', 'latvian': 'lv', 'lingala': 'ln', 'lithuanian':

⛅ 5. city_weather.html – City-Based Weather Monitoring Page

The city_weather.html page is a critical component of the **Flood Alert & Weather Monitoring System**. It enables users to access **real-time weather updates** for any city included in the system. This page acts as a **self-service tool** for users who want to monitor current weather conditions without waiting for an automated flood alert.

🎯 Purpose of the Page

This page allows users to **manually check weather conditions** in any city listed in the user database. While the flood detection algorithm works automatically in the background, this page gives users the power to verify conditions themselves, adding a layer of **user control and transparency**.

For example, a farmer, commuter, or local official might want to check if the rainfall or humidity levels in their area are approaching dangerous thresholds. This page provides a simple interface for doing exactly that.

🔄 Workflow and How It Works

When the user clicks the "☀️ Check City Weather" button on the homepage, they are directed to this page. The workflow goes as follows:

1. **Dropdown Display**: A list of all unique cities from the user_data.csv file is displayed in a dropdown menu.
2. **City Selection**: The user selects a city and submits the form.
3. **API Fetching**: The backend uses the OpenWeatherMap API to retrieve current weather data for the selected city.
4. **Display of Weather Info**: If data is successfully retrieved, it shows:

- 🌡️ Temperature (°C)
- 💧 Humidity (%)
- 🌧️ Rainfall (mm)
- 🌬️ Wind Speed (m/s)
- ☁️ Weather Description (e.g., Clear Sky, Light Rain)

All of this is presented clearly so that users can interpret the values without technical expertise.

Real-Time Timestamp

One of the key features of the city_weather.html page is the **timestamp** that accompanies every weather result. The page displays a line like:

"Data retrieved on: Thursday, 20 June 2025, 05:45 PM"

This helps users **trust the freshness of the data**, knowing that what they are viewing is not cached or outdated.

📉 Error Handling and Validation

If a city is selected that cannot be found by the weather API (e.g., due to misspelling or an API issue), the page intelligently displays an error message instead of crashing. This ensures a **robust and user-friendly** experience.

🎨 Design and User Interface

The layout is built using **Bootstrap**, ensuring that it's fully **responsive** and works on desktops, tablets, and phones. The background is usually styled with soft gradients or weather-themed images, enhancing readability.

Data is displayed in **well-designed cards or containers**, using icons or color cues (e.g., blue for rain, red for high temperature). This visual clarity ensures that even users with limited literacy or tech background can make sense of the information.

💡 Practical Impact

This page equips users with **decision-making power**. Instead of waiting passively for alerts, they can proactively monitor their city's weather. Whether for daily planning, agriculture, or flood preparedness, this tool ensures that users stay informed and ahead of the risk.

PROJECT ARCHITECTURE OVERVIEW

🚀 COMPLETE SYSTEM DOCUMENTATION: app.py + Templates + Features

Your project is a real-time, multilingual **Flood Prediction and Alert System** powered by Flask, Machine Learning, OpenWeather API, Twilio SMS, and Google Translate. It follows a **modular and user-friendly structure**.

🔄 SYSTEM EXECUTION FLOW

1. 🏠 Home Page – index.html

- Triggered via the / route.
- Displays five action buttons:
  - ✅ **Check Flood Risk**
  - ✏️ **Manual Flood Check**
  - ⛅ **Check City Weather**
  - **Customer Support**
  - 🚀 **Auto Flood Risk Detection**

2. ✅ Flood Risk Detection (Auto) – /check_flood

- For each city in user_data.csv:
  - Fetch weather via OpenWeatherMap API.
  - Predict flood **type** and **effect** using flood_model.pkl.
- If **"Severe"** or **"Extreme"**:
  - Translated alerts sent via SMS using Twilio.
  - ➡️ Loads affected.html (user alerts shown).
- If no flood risk:
  - ➡️ Loads no_flood.html (safe cities shown).

3. ⛅ Check City Weather – /city_weather

- User selects a city.
- Real-time weather is fetched.
- ➡️ Displayed on city_weather.html:
  - Temperature, Rainfall, Humidity, Wind Speed, Description.

4. Customer Support – /support

- Provides dropdown of common flood-related queries like:
  - What to do before a flood?
  - What precautions to take?
  - Emergency contacts?
- On selection, answer is shown instantly using:
  - ➡ support.html template.

HTML Templates (Explained Point-Wise)

- **index.html**
  - Entry point with all main options for users.
  - Launchpad for accessing features.
- **manual_input.html**
  - City dropdown + weather inputs form.
  - Posts data to /manual_check.
- **result.html**
  - Lists users who received flood alerts manually.
  - Shows city, flood type, and effect.
- **affected.html**
  - Shows users notified during automatic flood detection.
  - Helps visualize real-time alerts.
- **no_flood.html**
  - Simple message: No flood risk detected.
  - Lists cities checked and their safe status.
- **city_weather.html**
  - User selects city and views live weather data.
  - Includes time-stamped results.
- **support.html**
  - FAQ-like dropdown for user questions.
  - Displays answer dynamically without reloading.

Machine Learning Model (flood_model.pkl)

- Trained using:
  - flood_data.csv → Features: Rainfall, Humidity, Wind Speed.
  - Labels: flood_type and flood_effects.

- Trained using:

  MultiOutputClassifier(DecisionTreeClassifier())

- Saved using pickle and loaded in app.py.

🌐 External Integrations

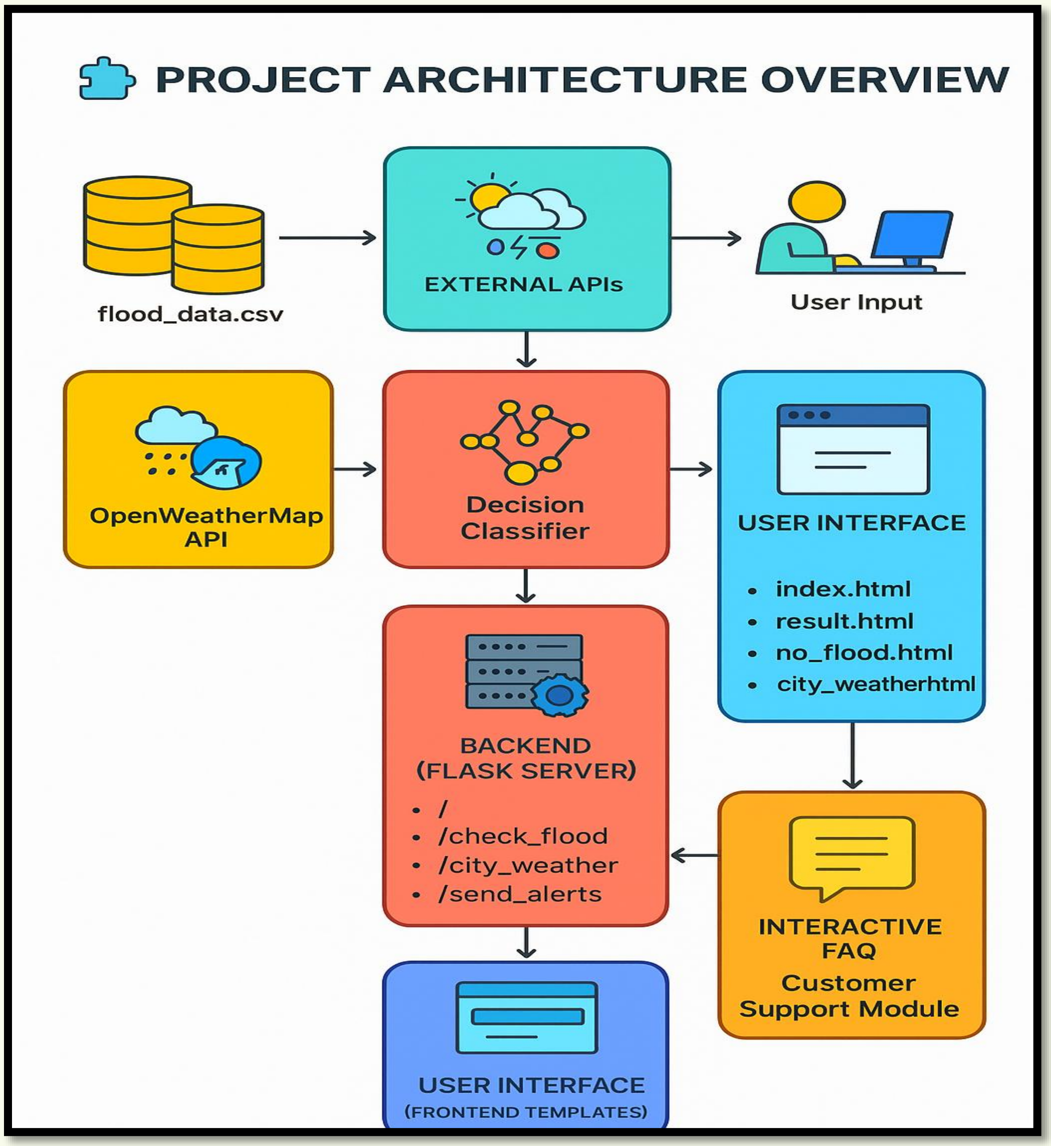
Service	Purpose
OpenWeatherMap	Real-time weather inputs
Twilio API	Sends SMS alerts to users
Google Translate	Translates alerts to local language

🗺️ System Architecture (Text View)

```
User → index.html
   ├── /check_flood → affected.html / no_flood.html
   ├── /manual_input → /manual_check → result.html / no_flood.html
   ├── /city_weather → city_weather.html
   └── /support → support.html
```

✅ Final Notes

- app.py handles logic, routes, model, and API calls.
- HTML templates ensure smooth user interaction.
- The system:
  - Predicts flood risk.
  - Sends timely, translated alerts.
  - Offers weather insights.
  - Provides safety guidance via customer support.



Project Conclusion – Flood Alert System 🌊📢

The Flood Alert System efficiently uses weather data to predict flood risks and send timely SMS alerts, helping reduce damage and save lives. It demonstrates how technology can enhance disaster preparedness and community safety.

🔧 Future Enhancements:

- IoT sensors for real-time data.
- Mobile app for instant alerts.
- Multilingual message support.
- AI-based flood prediction.
- Map-based risk visualization.

"Just like a flood alert system, I believe in reaching out before it's too late." 🌊📢