

The Superior University

|  |  |  |
| --- | --- | --- |
| Name: Muhammad Bilal Sajid | Roll No: 018 | Course: PAI Lab |
| Semester: 4th | Section: BSAI 4A | Department: |
| Submitted To: | Total Marks: | Date: |

**Lab 7 & 8**

**lab 7 = backend code (python / flask code)**

**lab 8 = backend + frontend code (python/flask + html,css,js code)**

**Enhanced Weather App Documentation**

**1. Introduction**

The Enhanced Weather App is a comprehensive Flask-based web application that provides detailed weather information for locations worldwide. Building upon the basic version, this enhanced application includes multiple advanced features such as weather forecasts, geolocation, unit conversion, favorites system, interactive weather maps, and performance optimization through caching.

The application uses the WeatherAPI.com service to fetch real-time weather data and presents it in a user-friendly, responsive interface that works well on both desktop and mobile devices.

**2. New Features Overview**

**2.1 Weather Forecasts**

* 7-day weather forecast with daily conditions
* Detailed information including temperature ranges, humidity, wind, and precipitation chance
* Collapsible hourly forecasts for each day
* Visual weather condition icons

**2.2 Geolocation**

* Automatic location detection using the browser's Geolocation API
* One-click weather information for the user's current location
* Comprehensive error handling for location permission issues

**2.3 Temperature Unit Toggle**

* Easy switching between Celsius and Fahrenheit
* Consistent unit conversion across all temperature data
* Complementary unit conversions (km/h to mph, km to miles)
* Persistent unit preference during navigation

**2.4 Favorites System**

* Save frequently checked locations
* Quick access to favorite locations from the home page
* Add/remove favorites with a single click
* Local storage-based persistence between sessions

**2.5 Weather Maps**

* Interactive weather maps with multiple data layers
* Temperature, precipitation, wind, pressure, and cloud coverage visualizations
* Color-coded legends for data interpretation
* Location-specific map views

**2.6 Performance Optimization**

* Server-side caching to reduce API calls
* Configurable cache timeout
* Improved application responsiveness
* Reduced external API dependency

**3. Technical Implementation**

**3.1 Backend Architecture**

The application uses Flask as its web framework and is structured around several key components:

* **Route Handlers**: Process user requests and render appropriate templates
* **API Integration**: Communicate with WeatherAPI.com to fetch weather data
* **Data Processing**: Transform API responses into structured data for templates
* **Caching System**: Store and retrieve weather data to minimize API calls
* **Error Handling**: Manage API errors and provide user-friendly messages

The backend implements a decorator-based caching system that stores API responses in memory with a configurable timeout. This significantly reduces the number of API calls for frequently requested locations.

**3.2 Frontend Design**

The frontend is built with HTML, CSS, and JavaScript, with a focus on:

* **Responsive Design**: Adapts to different screen sizes and devices
* **Intuitive Navigation**: Clear pathways between different sections
* **Visual Hierarchy**: Emphasizes important information through layout and typography
* **Interactive Elements**: Dynamic components that respond to user actions
* **Consistent Styling**: Unified color scheme and design language throughout

The application uses Font Awesome icons and a custom CSS framework to create a modern, clean interface that prioritizes readability and usability.

**3.3 Data Flow**

1. **User Input**: User enters a city name or uses geolocation
2. **Server Processing**: Flask routes the request to the appropriate handler
3. **Cache Check**: System checks if the requested data is in the cache
4. **API Request**: If not cached, the system requests data from WeatherAPI.com
5. **Data Processing**: Raw API data is transformed into a structured format
6. **Template Rendering**: Processed data is passed to HTML templates
7. **Client-Side Enhancements**: JavaScript adds interactivity to the rendered page

This flow ensures efficient data retrieval while maintaining a responsive user experience.

**4. Code Structure**

**4.1 Main Application (app.py)**

The main application file contains:

* Flask application initialization
* Route definitions for different pages
* Caching decorator implementation
* API integration functions
* Data processing utilities

Key functions include:

* get\_weather\_data(): Fetches and caches weather data from the API
* process\_current\_weather(): Transforms current weather data
* process\_forecast\_data(): Processes forecast information
* Route handlers for home, weather, maps, and API endpoints

**4.2 Templates**

The application uses three main templates:

* **index.html**: Home page with search functionality and favorites list
* **weather.html**: Detailed weather information with current conditions and forecast
* **maps.html**: Interactive weather maps with different data layers

Each template includes responsive design elements and JavaScript for client-side interactivity.

**4.3 Static Assets**

The CSS file (style.css) provides:

* Base styling for all pages
* Responsive layout rules
* Component-specific styles
* Animation and transition effects
* Media queries for different screen sizes

**5. User Interface and Experience**

**5.1 Home Page**

The home page features:

* A prominent search bar for city lookup
* Geolocation button for current location weather
* List of saved favorite locations
* Navigation to the weather maps section

**5.2 Weather Details Page**

The weather details page includes:

* Current weather conditions with temperature and icon
* Detailed metrics (humidity, wind, pressure, etc.)
* Air quality information when available
* 7-day forecast with expandable hourly details
* Unit toggle between Celsius and Fahrenheit
* Favorite toggle for the current location
* Weather map preview
* Weather alerts (when applicable)

**5.3 Weather Maps Page**

The maps page provides:

* Interactive map with zoom and pan capabilities
* Layer selection for different weather data types
* Color-coded legend for data interpretation
* Automatic centering on user location or searched city

**6. Running the Application**

**6.1 Prerequisites**

* Python 3.6 or higher
* Flask and Requests libraries
* Internet connection for API access

**6.2 Installation**

1. Install required packages:

pip install flask requests

1. The WeatherAPI.com API key is hardcoded in the application.
2. For full map functionality, replace YOUR\_OPENWEATHERMAP\_KEY in maps.html with a valid OpenWeatherMap API key.

**6.3 Execution**

Run the application with:

python app.py

Access the application by opening a web browser and navigating to:

http://127.0.0.1:5000/





