Project Report: Change Wallpaper Based on Weather

Operating System (SWE3001) - F1+TF1

Team Member:

Divya P - 23MIA1099

Project Report: Change Wallpaper Based on Weather

Introduction

In the modern digital workspace, personalizing user experience enhances productivity and engagement. This project aims to dynamically change the desktop wallpaper based on real-time weather conditions using OpenWeatherMap API. The implementation ensures cross-platform compatibility for Windows, Linux, and macOS, providing an automated and visually appealing experience.

Abstract

This project leverages real-time weather data to modify the desktop background accordingly. By fetching weather conditions via an API and mapping them to predefined wallpapers, it delivers a dynamic visual representation of the external environment. Using platform-specific commands, the project integrates seamlessly with different operating systems, enhancing user engagement through automation.

Objective

- To develop an automated system that updates the desktop wallpaper based on real-time weather conditions.
- To ensure cross-platform compatibility for Windows, Linux, and macOS.
- To provide a seamless and visually dynamic user experience.
- To enhance personalization using predefined wallpapers mapped to weather conditions.

Functional Breakdown

1. Fetching Weather Data

- The function `get_weather(api_key, city)` retrieves weather data using the OpenWeatherMap API.
- The API request is sent via the `requests` library, and the response is parsed in JSON format.
- If the request is successful, the function extracts the main weather condition (e.g., clear, rain, snow) and returns it in lowercase.
- If the request fails, an error message is printed.

2. Mapping Weather Conditions to Wallpapers

- A dictionary named `wallpapers` stores mappings between weather conditions and corresponding image file paths.
- The function `main()` determines the wallpaper based on the weather condition.
- If a matching wallpaper is found, it is selected; otherwise, a default wallpaper is used.

3. Changing the Desktop Wallpaper

- The function `set_wallpaper(image_path)` updates the desktop wallpaper based on the OS.
- The `platform.system()` function is used to detect the operating system.
- Based on the OS, different methods are used to apply the wallpaper:
    - **Windows:** Uses the `ctypes` library to call `SystemParametersInfoW`.
    - **Linux:** Uses the `gsettings` command to change the GNOME desktop background.
    - **macOS:** Uses an AppleScript command to modify the Finder's desktop picture setting.

Operating System Integration

The project ensures compatibility across different operating systems using the following approach:

1. Windows Integration

- The `ctypes.windll.user32.SystemParametersInfoW(20, 0, image_path, 0)` function updates the desktop wallpaper.
- This method directly interacts with the Windows API to modify the system settings.

2. Linux Integration

- The command `gsettings set org.gnome.desktop.background picture-uri file://{image_path}` is executed using `os.system()`.
- This method works for GNOME-based Linux distributions. Other desktop environments (e.g., KDE, XFCE) may require different commands.

3. macOS Integration

- The command `osascript -e "tell application \"Finder\" to set desktop picture to POSIX file \"{image_path}\""` is executed.
- This method uses AppleScript to update the wallpaper setting.

Potential Improvements

1. Enhancing OS Compatibility

- For Windows, registry updates or third-party libraries like `pywallpaper` could improve compatibility.
- For Linux, dynamically detecting the desktop environment would allow broader support beyond just GNOME.
- For macOS, `osascript` works but might require additional permissions.

2. Weather & Time-Based Wallpapers

- The wallpaper will change dynamically based on the weather conditions of a selected location.
- It will also update based on the time of day (e.g., morning, afternoon, evening, night) for a more immersive experience.

3. Error Handling and Logging

- Implement exception handling to manage failed API requests gracefully.
- Use logging instead of print statements for better debugging and tracking.

4. User Customization

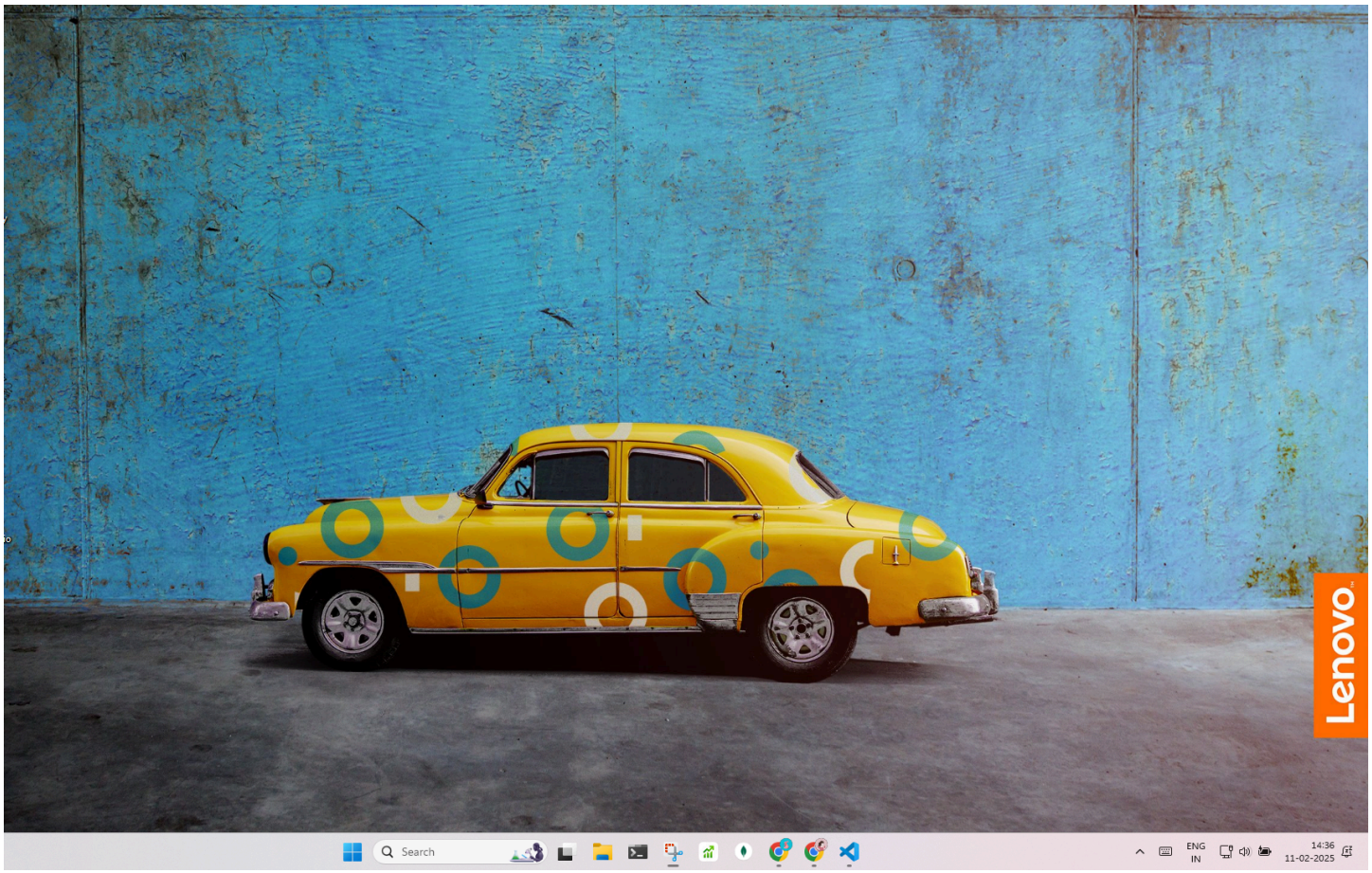
- Allow users to specify a custom directory for wallpapers.Use a background service or cron job to automate execution.
- Provide a GUI for setting preferences, such as choosing different weather sources or selecting update intervals.

5. Automating Updates

- Implement a scheduler to update the wallpaper at regular intervals.
- Use a background service or cron job to automate execution without user intervention.

CODE LINK: https://drive.google.com/file/d/1PW1UODIZgco6g8-b5FuO6dix119OMnv8/view?usp=sharing

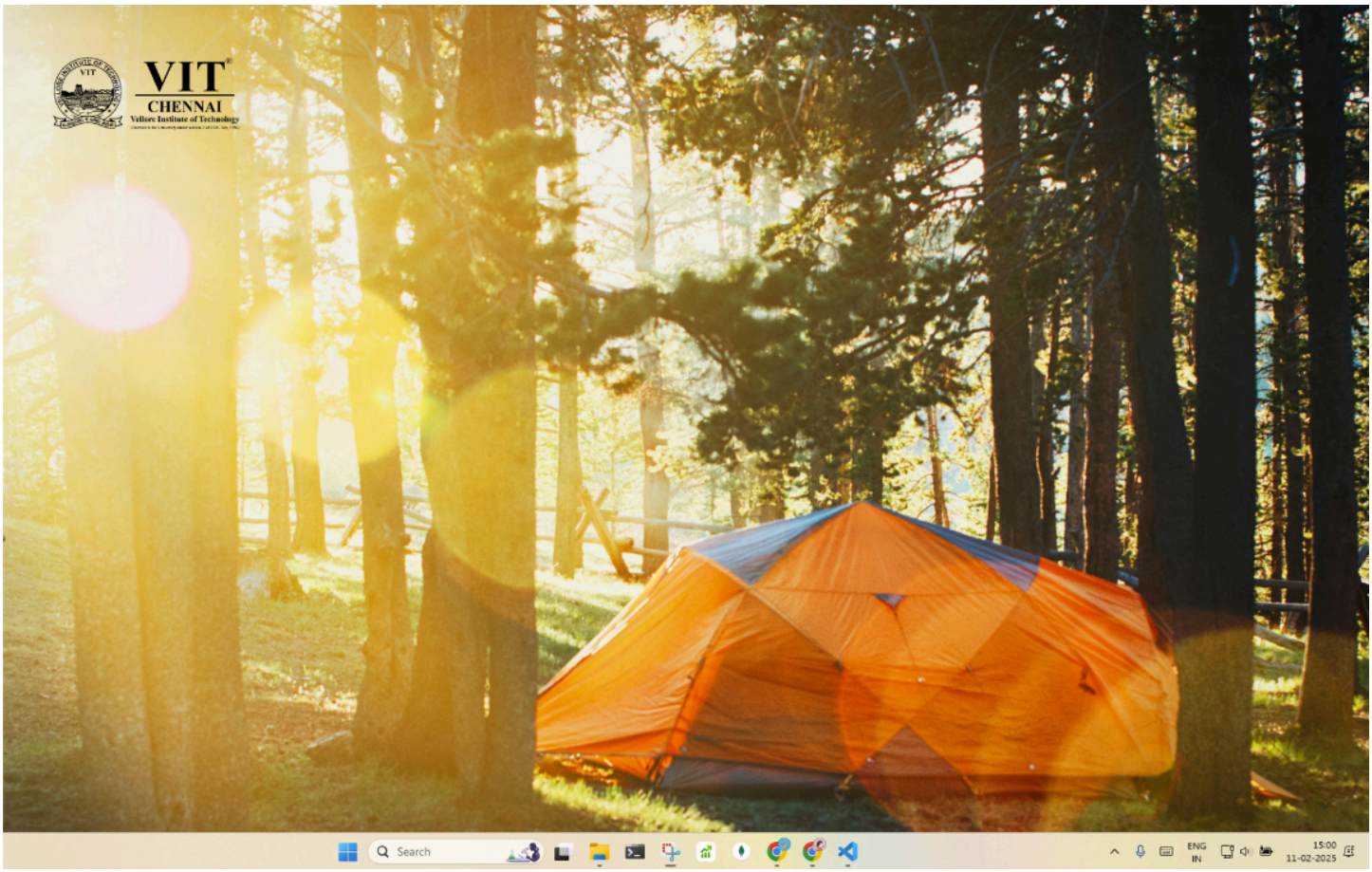
Home page before running the code



Output after running the code

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE
PS C:\Users\91944\Downloads\TRIAL> & C:/Users/91944/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/91944/Downloads/TRIAL/trail.py
Wallpaper changed to C:\Users\91944\Downloads\TRIAL\wallpapers\clear.jpg
PS C:\Users\91944\Downloads\TRIAL>
```

The home page changed based on Chennai weather at 15:00 as it is clear i.e. sunny. the wallpaper changed to a sunny image.



Comparison with Existing Research & Applications

Existing Research and Applications

Several applications and projects exist that provide dynamic wallpaper updates based on weather and time. Some notable examples include:

1. **Living Weather & Wallpapers HD** - Provides animated wallpapers that reflect real-time weather conditions and forecasts.
2. **Dynamic Wallpaper by daspartho** - An open-source project that fetches weather data and updates the wallpaper using images from Unsplash.
3. **24 Hour Wallpaper** - Changes wallpapers based on the time of day, enhancing the visual experience with daylight transitions.

How This Project is Unique

- **Predefined Local Wallpaper Repository:** Unlike applications that fetch images from external sources, this project uses a locally stored set of wallpapers. This ensures faster access and allows users to customize their own wallpapers without relying on third-party services.
- **Cross-Platform Compatibility:** The project supports Windows, Linux, and macOS using OS-specific commands, making it more versatile than some alternatives that focus on a single OS.
- **Privacy-Focused Approach:** No external wallpaper services are required, ensuring user privacy and reducing unnecessary API calls.
- **Lightweight and Customizable:** The application is open-source and easy to modify, allowing users to tailor it according to their preferences.

Conclusion

This project successfully integrates weather-based wallpaper updates with cross-platform support. By leveraging API data and system-specific commands, it dynamically adapts to changing conditions. Unlike existing applications, this approach prioritizes privacy, customization, and OS flexibility. Future improvements could enhance compatibility, automation, and user customization for a more seamless experience. The project demonstrates the practical application of API-driven automation, enhancing the user's digital environment based on real-world factors.