BỘ NÔNG NGHIỆP VÀ PHÁT TRIỂN NÔNG THÔN

**TRƯỜNG ĐẠI HỌC THỦY LỢI**

A blue and white logo

Description automatically generated

**BÀI TẬP LỚN**

**HỌC PHẦN: LẬP TRÌNH ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: Xây dựng ứng dụng Dự báo thời tiết (Weather App)**

|  |  |  |  |
| --- | --- | --- | --- |
| Mã Sinh Viên | Họ và Tên | Ngày Sinh | Lớp |
| 2151060214 | Tống Đức Hải | 07/01/2003 | 63CNTT.NB |
| 2151062709 | Nguyễn Việt Anh | 13/03/2001 | 63CNTT.NB |
| 2051060680 | Hồ Sỹ Sơn | 17/11/2001 | 62TH4 |
| 2151170580 | Lê Ngân Hà | 02/09/2003 | 63KTPM1 |

### 

**Hà Nội, năm 2024**

BỘ NÔNG NGHIỆP VÀ PHÁT TRIỂN NÔNG THÔN

**TRƯỜNG ĐẠI HỌC THỦY LỢI**

A blue and white logo

Description automatically generated

**BÀI TẬP LỚN**

**HỌC PHẦN: LẬP TRÌNH ỨNG DỤNG DI ĐỘNG**

**ĐỀ TÀI: Xây dựng ứng dụng Dự báo thời tiết (Weather App)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mã Sinh Viên | Họ và Tên | Ngày Sinh | Điểm | |
| Bằng Số | Bằng Chữ |
| 2151060214 | Tống Đức Hải | 07/01/2003 |  |  |
| 2151062709 | Nguyễn Việt Anh | 13/03/2001 |  |  |
| 2051060680 | Hồ Sỹ Sơn | 17/11/2001 |  |  |
| 2151170580 | Lê Ngân Hà | 02/09/2003 |  |  |

### 

### CÁN BỘ CHẤM THI 1 CÁN BỘ CHẤM THI 2

**Hà Nội, năm 2024**

**LỜI NÓI ĐẦU**

Trong thời đại công nghệ số phát triển mạnh mẽ, việc tiếp cận thông tin nhanh chóng và chính xác ngày càng trở nên quan trọng. Một trong những nhu cầu thiết yếu của con người là theo dõi và nắm bắt thông tin về thời tiết, không chỉ để phục vụ cho cuộc sống hàng ngày mà còn để dự đoán và lên kế hoạch cho các hoạt động quan trọng. Ứng dụng dự báo thời tiết trên điện thoại di động đã trở thành công cụ không thể thiếu, cung cấp cho người dùng thông tin chi tiết về thời tiết ở bất kỳ địa điểm nào trên thế giới.

Với mục tiêu tạo ra một ứng dụng thời tiết tiện lợi, thân thiện với người dùng và tương thích với đa dạng thiết bị, dự án ứng dụng thời tiết này được xây dựng dựa trên nền tảng API của AccuWeather. Ứng dụng cung cấp các tính năng như xem thời tiết theo ngày, theo giờ, dự báo trong 5 ngày, và cho phép người dùng thêm, xóa, cũng như tìm kiếm thông tin thời tiết theo địa điểm mong muốn. Đặc biệt, ứng dụng còn hỗ trợ hiển thị thời tiết dựa trên vị trí hiện tại của người dùng, giúp tăng cường trải nghiệm sử dụng.

Trong báo cáo này, chúng tôi sẽ trình bày chi tiết về quá trình xây dựng và phát triển ứng dụng dự báo thời tiết, từ khâu thiết kế hệ thống, triển khai tính năng, đến kiểm thử và đánh giá hiệu quả hoạt động. Hy vọng rằng sản phẩm này sẽ mang lại nhiều tiện ích và đáp ứng tốt nhu cầu theo dõi thời tiết của người dùng.

**BẢNG CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **TỪ VIẾT TẮT** | **VIẾT ĐẦY ĐỦ** |
| 1 | CSDL | Cơ sở dữ liệu |
| 2 | AQI | Chỉ số chất lượng không khí |

**MỤC LỤC**

[**CHƯƠNG 1. MÔ TẢ BÀI TOÁN** 7](#_Toc181417224)

[**1.1. Giới thiệu** 7](#_Toc181417225)

[**1.2. Chức năng chính** 7](#_Toc181417226)

[**1.3. Yêu cầu phi chức năng** 7](#_Toc181417227)

[**CHƯƠNG 2. PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ HỆ THỐNG** 8](#_Toc181417228)

[**2.1. Phân tích yêu cầu:** 8](#_Toc181417229)

[1. Xem thời tiết theo tùy chọn: 8](#_Toc181417230)

[2. Xem chi tiết thời tiết: 9](#_Toc181417231)

[3. Thêm/xóa địa điểm: 9](#_Toc181417232)

[4. Tìm kiếm và xem thời tiết theo tên địa điểm: 9](#_Toc181417233)

[5. Xem thời tiết theo vị trí hiện tại: 9](#_Toc181417234)

[7. Yêu cầu phi chức năng: 9](#_Toc181417235)

[**2.2. Thiết kế hệ thống:** 10](#_Toc181417236)

[**2.3. Triển khai:** 18](#_Toc181417237)

[Kiểm tra chức năng lấy thời tiết hiện tại 18](#_Toc181417238)

[Kiểm tra chức năng tìm kiếm thành phố 19](#_Toc181417239)

[Kiểm tra chức năng dự báo thời tiết theo ngày 19](#_Toc181417240)

[Kiểm tra chức năng xem chi tiết thời tiết theo giờ 19](#_Toc181417241)

[Kiểm tra lỗi kết nối mạng 20](#_Toc181417242)

[Kiểm tra giao diện đa màn hình 20](#_Toc181417243)

[Kiểm tra thông tin thời tiết trực tiếp theo vị trí 20](#_Toc181417244)

[Kiểm tra tính năng tìm kiếm thành phố 21](#_Toc181417245)

[Kiểm tra tính năng xem lịch sử thời tiết 21](#_Toc181417246)

[**2.4. Vận hành và bảo trì:** 21](#_Toc181417247)

[**CHƯƠNG 3. KẾT QUẢ THỰC HIỆN** 22](#_Toc181417248)

[**3.1. Công nghệ đã sử dụng** 22](#_Toc181417249)

[**3.2. Tiến độ thực hiện** 22](#_Toc181417250)

[**3.3. Hình ảnh sản phẩm** 26](#_Toc181417251)

[**DANH MỤC TÀI LIỆU THAM KHẢO** 31](#_Toc181417254)

[**PHỤ LỤC** 32](#_Toc181417256)

**CHƯƠNG 1. MÔ TẢ BÀI TOÁN**

**1.1. Giới thiệu**

Ứng dụng Dự Báo Thời Tiết của chúng tôi là một công cụ hữu ích, cho phép bạn nhanh chóng và dễ dàng cập nhật thông tin thời tiết mọi lúc, mọi nơi. Với giao diện đơn giản và thân thiện, ứng dụng giúp bạn theo dõi điều kiện thời tiết theo các khung thời gian linh hoạt và chi tiết.. Ứng dụng sẽ được xây dựng bằng Java và chạy trên giao diện GUI.

**1.2. Chức năng chính**

Ứng dụng Dự Báo Thời Tiết  cần có các chức năng sau:

* Cho phép xem thời tiết theo các tùy chọn 1 ngày, 5 ngày.
* Cho phép xem chi tiết thời tiết trong 1 giờ, 12 giờ tới.
* Cho phép thêm/xóa địa điểm(thành phố) cần xem thời tiết.
* Cho phép tìm kiếm và xem thời tiết theo tên thành phố/địa điểm.
* Cho phép xem thời tiết trực tiếp theo vị trí hiện tại.
* Cho phép xem lịch sử thời tiết theo ngày.

**1.3. Yêu cầu phi chức năng**

* Dễ sử dụng: Giao diện dễ hiểu và dễ sử dụng.
* Hiệu năng: Ứng dụng cần hoạt động nhanh chóng và hiệu quả.
* Độ tin cậy: Dữ liệu công việc cần được lưu trữ an toàn và không bị mất mát.

**CHƯƠNG 2. PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ HỆ THỐNG**

**2.1. Phân tích yêu cầu:**

**Xác định người dùng:**

* **Người dùng phổ thông**: Những người chỉ muốn kiểm tra nhanh tình hình thời tiết hiện tại và dự báo trong ngày hoặc vài ngày tới.
* **Người dùng di chuyển nhiều**: Những người di chuyển giữa các thành phố hoặc quốc gia, có nhu cầu theo dõi thời tiết ở nhiều địa điểm khác nhau để lập kế hoạch di chuyển
* **Người dùng cần thường xuyên theo dõi thời tiết**: Những người có công việc hoặc sinh hoạt cần cập nhật thời tiết thường xuyên như nông dân, ngư dân, hay người làm công việc ngoài trời.
* **Người dùng đi du lịch**: Những người sắp tới phải đi đến địa điểm khác để du lịch hoặc về quê, cần theo dõi thời tiết để đảm bảo sự chuẩn bị.
* **Người có vấn đề sức khỏe nhạy cảm với thời tiết**: Những người bị các bệnh về hô hấp, dị ứng thời tiết, bệnh xương khớp, thể trạng yếu.

**Thu thập yêu cầu:**

Dựa trên mô tả bài toán, ta đã xác định được các chức năng chính (xem thời tiết theo tùy chọn 1 ngày, 5 ngày; xem chi tiết thời tiết 1 giờ, 12 giờ; thêm/xóa địa điểm; tìm kiếm theo tên thành phố; xem thời tiết theo vị trí hiện tại) và yêu cầu phi chức năng (dễ sử dụng, hiệu năng, độ tin cậy) của ứng dụng.

**Phân tích yêu cầu:**

### 1. Xem thời tiết theo tùy chọn:

* Người dùng có thể xem dự báo thời tiết cho:
  + **1 ngày**.
  + **5 ngày**.
* Dữ liệu thời tiết cần hiển thị bao gồm thông tin cơ bản như nhiệt độ, điều kiện thời tiết (nắng, mưa, sương mù...).

### 2. Xem chi tiết thời tiết:

* Cho phép người dùng xem dự báo chi tiết hơn trong:
  + **1 giờ** tiếp theo.
  + **12 giờ** tiếp theo.
* Thông tin chi tiết bao gồm nhiệt độ, lượng mưa, độ ẩm, và tốc độ gió theo từng giờ.

### 3. Thêm/xóa địa điểm:

* Người dùng có thể **thêm** các địa điểm (thành phố) mà họ muốn theo dõi thời tiết.
* Cho phép **xóa** các địa điểm không còn cần thiết.

### 4. Tìm kiếm và xem thời tiết theo tên địa điểm:

* Người dùng có thể **tìm kiếm** tên thành phố hoặc địa điểm để xem thời tiết của nơi đó.
* Tìm kiếm cần nhanh chóng và chính xác, hiển thị các kết quả phù hợp.

### 5. Xem thời tiết theo vị trí hiện tại:

* Ứng dụng sử dụng GPS hoặc thông tin mạng để tự động xác định vị trí hiện tại của người dùng và hiển thị dự báo thời tiết tương ứng.
* Cần hiển thị chính xác và cập nhật thông tin thời tiết theo thời gian thực.

**6. Xem lịch sử thời tiết theo ngày:**

 **Chọn ngày**: Người dùng chọn ngày cụ thể trong quá khứ để xem dữ liệu.

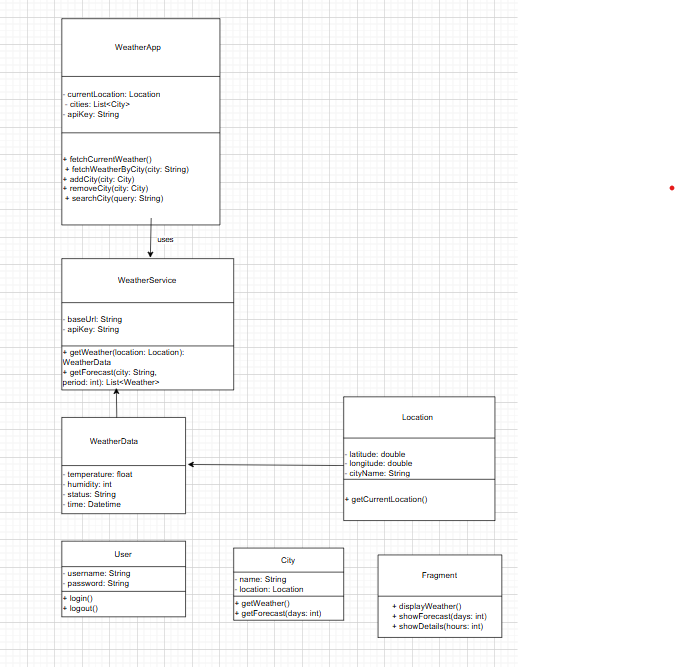
 **Chọn địa điểm**: Người dùng chọn thành phố hoặc vị trí địa lý để xem lịch sử thời tiết.

### 7. Yêu cầu phi chức năng:

* **Dễ sử dụng**: Giao diện cần thân thiện, dễ điều hướng cho người dùng.
* **Hiệu năng**: Ứng dụng cần hoạt động mượt mà, tải dữ liệu thời tiết nhanh chóng.
* **Độ tin cậy**: Dữ liệu thời tiết phải chính xác, cập nhật thường xuyên, và không mất mát thông tin khi sử dụng.

**2.2. Thiết kế hệ thống:**

**Biểu đồ lớp cho mô hình miền:**



**Biểu đồ lớp cho MVC:**

* Mô tả kiến trúc của ứng dụng Weather, bao gồm các lớp trong Model, View và Controller.

**Thiết kế kiến trúc (MVC):**

* **Model:**
  + **WeatherData**: Lớp này chứa các thông tin chi tiết về thời tiết hiện tại cho một thành phố cụ thể.
  + **DailyWeatherData:** Chứa thông tin dự báo thời tiết hàng ngày cho một thành phố.
  + **Forecast:** Lớp này quản lý dữ liệu dự báo thời tiết cho nhiều ngày. Có thể chứa danh sách của DailyWeatherData
  + **HistoryResponse**:Chứa dữ liệu phản hồi lịch sử thời tiết cho một thành phố trong một khoảng thời gian nhất định
  + **HourlyWeatherData:** Chứa thông tin chi tiết về thời tiết cho từng giờ trong một ngày.

**Chức năng của Model:**

* + Lấy dữ liệu thời tiết theo vị trí hiện tại hoặc theo địa điểm người dùng chọn.
  + Cập nhật dữ liệu từ API thời tiết.
  + Quản lý danh sách địa điểm mà người dùng thêm/xóa.
  + Truy xuất dữ liệu lịch sử thời tiết.
* **View:**
  + **CityListViewModel**: Chịu trách nhiệm quản lý danh sách các thành phố mà người dùng đã thêm vào để theo dõi thời tiết. Lớp này sẽ lưu trữ dữ liệu danh sách thành phố và cung cấp các phương thức để thêm, xóa, hoặc lấy danh sách thành phố.
  + **HomeViewModel:** Cung cấp dữ liệu chính cho màn hình chính của ứng dụng, bao gồm thông tin thời tiết hiện tại, vị trí hiện tại của người dùng, và có thể là thông tin tóm tắt dự báo trong ngày.
  + **SearchView**: Tìm kiếm và hiển thị thông tin thời tiết cho một thành phố cụ thể.
  + **Weather24hFragment**: Cung cấp dự báo chi tiết trong 24 giờ tiếp theo.

**Chức năng của View:**

* Nhận dữ liệu từ Controller và hiển thị cho người dùng.
* Cập nhật giao diện theo thời gian thực khi dữ liệu thay đổi.
* **Controller:**
  + **CityListFragment:** Chịu trách nhiệm hiển thị danh sách thành phố mà người dùng đã thêm vào.
  + **HistoryFragment:** Hiển thị dữ liệu thời tiết trong quá khứ, cho phép người dùng xem lịch sử thời tiết cho một thành phố vào ngày cụ thể. Dữ liệu này có thể bao gồm nhiệt độ và các điều kiện thời tiết vào thời điểm đó.
  + **Forecast5DaysFragment:** Cung cấp dự báo thời tiết cho 5 ngày tiếp theo cho thành phố hiện tại hoặc thành phố đã chọn. Dự báo này bao gồm các thông tin như nhiệt độ cao/thấp mỗi ngày, điều kiện thời tiết, và các thông số quan trọng khác.

**Nhiệm vụ của Controller:**

* Nhận yêu cầu từ người dùng (ví dụ: xem thời tiết 5 ngày, thêm địa điểm mới).
* Gọi Model để lấy hoặc cập nhật dữ liệu.
* Chuyển dữ liệu đã xử lý đến View để hiển thị.

|  |
| --- |
| **Dựa trên kiến trúc MVC đã chọn, ta có thể xác định các lớp sau:**  **Model**   * **Lớp WeatherData:**   + **Thuộc tính:**   + temperature: Nhiệt độ hiện tại.   + humidity: Độ ẩm.   + weatherCondition: Điều kiện thời tiết (như nắng, mưa, tuyết, v.v.).   + windSpeed: Tốc độ gió.   + cityName: Tên thành phố   + Phương thức:     - Getters và setters cho các thuộc tính.     - toString(): trả về chuỗi mô tả công việc. * **Lớp DailyWeatherData:**   + **Thuộc tính:**     - date: Ngày của dự báo.     - temperatureHigh: Nhiệt độ cao nhất trong ngày.     - temperatureLow: Nhiệt độ thấp nhất trong ngày.     - weatherCondition: Điều kiện thời tiết cho ngày đó. * **Lớp Forecast:**   + dailyWeatherList: Danh sách các dự báo thời tiết hàng ngày. * **Lớp HistoryResponse:**   + **Thuộc tính:**     - hour: Thời gian trong ngày.     - temperature: Nhiệt độ tại thời điểm đó.     - humidity: Độ ẩm tại thời điểm đó.     - weatherCondition: Điều kiện thời tiết tại thời điểm đó.   **View**   * **Lớp CityListViewModel:**   + **Thuộc tính:**     - cityList: LiveData<List<CityEntity>> - Danh sách các thành phố mà người dùng đã thêm vào.     - weatherData: LiveData<WeatherData> - Thông tin thời tiết cho thành phố đang được theo dõi.     - addCity(city: CityEntity): Phương thức để thêm một thành phố vào danh sách.     - removeCity(city: CityEntity): Phương thức để xóa một thành phố khỏi danh sách.     - fetchWeatherDataForCity(cityName: String): Phương thức để lấy dữ liệu thời tiết cho một thành phố cụ thể.. * **Lớp HomeViewModel:**   + **Thuộc tính:**     - currentWeather: LiveData<WeatherData> - Thông tin thời tiết hiện tại.     - currentLocation: LiveData<Location> - Vị trí hiện tại của người dùng.     - dailyForecast: LiveData<List<DailyWeatherData>> - Danh sách thông tin dự báo thời tiết hàng ngày.     - fetchCurrentWeather(location: Location): Phương thức để lấy thông tin thời tiết hiện tại.     - fetchDailyForecast(location: Location): Phương thức để lấy thông tin dự báo hàng ngày. * **Lớp SearchView:**   + **Thuộc tính:**     - searchQuery: String - Chuỗi tìm kiếm của người dùng.     - searchResults: LiveData<List<CityEntity>> - Danh sách các thành phố tìm kiếm được.     - onSearch(cityName: String): Phương thức để thực hiện tìm kiếm thông tin thời tiết cho một thành phố. * **Lớp Weather24hFragment:**   + hourlyWeatherData: LiveData<List<HourlyWeatherData>> - Danh sách thông tin thời tiết theo giờ trong 24 giờ tiếp theo.   + selectedCity: CityEntity - Thành phố đang được chọn để xem dự báo thời tiết.   + fetchHourlyWeather(cityName: String): Phương thức để lấy thông tin thời tiết theo giờ cho thành phố đã chọn.   **Controller**   * **Lớp CityListFragment:**   + **Thuộc tính:**     - adapter: CityAdapter - Adapter cho RecyclerView hiển thị danh sách thành phố.     - cityListViewModel: CityListViewModel - ViewModel quản lý danh sách thành phố.     - onCreateView(...): Phương thức để khởi tạo giao diện và thiết lập RecyclerView. * **Lớp HistoryFragment:**   + **Thuộc tính:**     - historyData: LiveData<HistoryResponse> - Dữ liệu lịch sử thời tiết cho thành phố đã chọn.     - selectedCity: CityEntity - Thành phố mà người dùng đang xem lịch sử thời tiết.     - fetchHistoryData(cityName: String, date: Date): Phương thức để lấy dữ liệu thời tiết lịch sử cho thành phố và ngày đã chọn. * **Lớp Forecast5DaysFragment:**   + forecastData: LiveData<Forecast> - Dữ liệu dự báo thời tiết cho 5 ngày tới.   + selectedCity: CityEntity - Thành phố mà người dùng đang xem dự báo thời tiết.   + fetchForecastData(cityName: String): Phương thức để lấy dữ liệu dự báo thời tiết cho thành phố đã chọn.fetchHourlyWeather(cityName: String): Phương thức để lấy thông tin thời tiết theo giờ cho thành phố đã chọn   **Mối quan hệ giữa các lớp:**  **1. View và ViewModel**   * **Mối quan hệ**: **One-to-One** * Mỗi lớp View (như CityListFragment, Weather24hFragment, HistoryFragment, Forecast5DaysFragment) sẽ có một ViewModel tương ứng (như CityListViewModel, HomeViewModel). * **Chức năng**: View sẽ gọi ViewModel để lấy dữ liệu và nhận thông tin cập nhật về dữ liệu thời tiết, thành phố, và các thông tin khác cần thiết cho giao diện người dùng.   2. **ViewModel và Model**   * **Mối quan hệ**: **One-to-Many** * ViewModel (như CityListViewModel, HomeViewModel) sẽ quản lý nhiều đối tượng Model (như WeatherData, DailyWeatherData, HourlyWeatherData, HistoryResponse). * **Chức năng**: ViewModel thực hiện việc lấy, lưu trữ và cập nhật dữ liệu từ Model và chuyển giao dữ liệu đó tới View. Nó có thể gọi các phương thức từ Model để lấy thông tin thời tiết, dự báo, và lịch sử.   3. **Fragment (Controller) và ViewModel**   * **Mối quan hệ**: **Many-to-One** * Nhiều Fragment (như CityListFragment, HistoryFragment, Forecast5DaysFragment) có thể sử dụng chung một ViewModel (như CityListViewModel). * **Chức năng**: Các Fragment sẽ tương tác với ViewModel để lấy dữ liệu cần thiết và thực hiện các thao tác như thêm/xóa thành phố, hiển thị thông tin thời tiết, và xem dự báo.   4. **Model và Entity**   * **Mối quan hệ**: **One-to-Many** * Một đối tượng Model (như WeatherData, Forecast) có thể chứa nhiều Entity (như CityEntity). * **Chức năng**: Model sẽ tổ chức và cung cấp thông tin từ nhiều Entity để ứng dụng có thể sử dụng và hiển thị thông tin một cách hiệu quả. |
|  |

**Thiết kế cơ sở dữ liệu:** ERD

* Tạo một Google Sheet mới để lưu trữ dữ liệu.
* Xác định tên sheet và các cột tương ứng với các thuộc tính của công việc (ID, tiêu đề, mô tả, deadline, mức độ ưu tiên, trạng thái).

**Thiết kế giao diện:**

* Hiển thị menu chính với các lựa chọn:
  + 1. Thêm công việc mới
    2. Hiển thị danh sách công việc
    3. Đánh dấu hoàn thành công việc
    4. Chỉnh sửa công việc
    5. Xóa công việc
    6. Thoát
* Sử dụng các thông báo rõ ràng để hướng dẫn người dùng nhập liệu và hiển thị kết quả.
* Minh họa

**2.3. Triển khai:**

* **Viết code:** Sử dụng Java để cài đặt các class trong mô hình MVC, đọc/ghi file, xử lý dữ liệu và hiển thị giao diện console.
  + Cài đặt GoogleSheetHandler sử dụng thư viện API của Google Sheets
  + Xử lý xác thực để ứng dụng có quyền truy cập vào Google Sheet của bạn.
  + Thay đổi code trong TaskController để sử dụng GoogleSheetHandler cho việc đọc/ghi dữ liệu.
* **Kiểm thử:** Viết các test case để kiểm tra các chức năng của ứng dụng, đảm bảo ứng dụng hoạt động đúng theo yêu cầu.

### ****Kiểm tra chức năng lấy thời tiết hiện tại****

* **Mục tiêu**: Đảm bảo rằng ứng dụng có thể lấy được thông tin thời tiết cho vị trí hiện tại của người dùng.
* **Test case**:
  + **Input**: Nhấn vào nút "Xem thời tiết hiện tại".
  + **Expected result**: Ứng dụng hiển thị đúng thông tin thời tiết cho vị trí hiện tại (nhiệt độ, trạng thái thời tiết, độ ẩm, thời gian, v.v.).

### ****Kiểm tra chức năng tìm kiếm thành phố****

* **Mục tiêu**: Đảm bảo rằng người dùng có thể tìm kiếm thời tiết theo tên thành phố.
* **Test case 1**:
  + **Input**: Nhập tên thành phố hợp lệ (ví dụ: "New York") và nhấn nút "Tìm kiếm".
  + **Expected result**: Ứng dụng hiển thị đúng thông tin thời tiết của thành phố "New York".
* **Test case 2**:
  + **Input**: Nhập tên thành phố không hợp lệ (ví dụ: "InvalidCity") và nhấn nút "Tìm kiếm".
  + **Expected result**: Ứng dụng hiển thị thông báo lỗi "Không tìm thấy thành phố".

### ****Kiểm tra chức năng dự báo thời tiết theo ngày****

* **Mục tiêu**: Đảm bảo người dùng có thể xem dự báo thời tiết 1 ngày, 5 ngày tới.
* **Test case**:
  + **Input**: Chọn thành phố và nhấn vào tùy chọn "Dự báo 5 ngày".
  + **Expected result**: Ứng dụng hiển thị dự báo thời tiết cho thành phố đã chọn trong 5 ngày tới, với thông tin chi tiết về nhiệt độ, trạng thái thời tiết cho mỗi ngày.

"Thành phố đã có trong danh sách yêu thích".

### ****Kiểm tra chức năng xem chi tiết thời tiết theo giờ****

* **Mục tiêu**: Đảm bảo rằng người dùng có thể xem chi tiết dự báo thời tiết cho 1 giờ hoặc 12 giờ tới.
* **Test case 1**:
  + **Input**: Chọn tùy chọn "Xem thời tiết 1 giờ tới" cho thành phố "Tokyo".
  + **Expected result**: Ứng dụng hiển thị thời tiết chi tiết cho thành phố "Tokyo" trong 1 giờ tới (thời gian, nhiệt độ, trạng thái).
* **Test case 2**:
  + **Input**: Chọn tùy chọn "Xem thời tiết 12 giờ tới" cho thành phố "Paris".
  + **Expected result**: Ứng dụng hiển thị thời tiết chi tiết trong 12 giờ tới cho "Paris", bao gồm nhiệt độ, độ ẩm, và trạng thái thời tiết.

### ****Kiểm tra lỗi kết nối mạng****

* **Mục tiêu**: Đảm bảo ứng dụng xử lý tốt khi mất kết nối mạng.
* **Test case**:
  + **Input**: Tắt kết nối mạng và nhấn vào "Xem thời tiết hiện tại".
  + **Expected result**: Ứng dụng hiển thị thông báo lỗi "Không có kết nối mạng. Vui lòng thử lại sau".

### ****Kiểm tra giao diện đa màn hình****

* **Mục tiêu**: Đảm bảo giao diện ứng dụng tương thích với các kích cỡ màn hình khác nhau.
* **Test case**:
  + **Input**: Mở ứng dụng trên điện thoại có màn hình nhỏ và máy tính bảng có màn hình lớn.
  + **Expected result**: Giao diện ứng dụng hiển thị đúng và không bị cắt xén hay bị lỗi trên cả hai thiết bị.

### ****Kiểm tra thông tin thời tiết trực tiếp theo vị trí****

* **Mục tiêu**: Đảm bảo ứng dụng lấy thông tin thời tiết theo vị trí hiện tại của người dùng.
* **Test case**:
  + **Input**: Mở ứng dụng và nhấn vào tùy chọn "Xem thời tiết tại vị trí hiện tại".
  + **Expected result**: Ứng dụng hiển thị chính xác thời tiết tại vị trí hiện tại của người dùng.

### ****Kiểm tra tính năng tìm kiếm thành phố****

* **Mục tiêu**: Đảm bảo người dùng có thể tìm kiếm thông tin thời tiết theo tên thành phố.
* **Test case**:
  + **Input**: Nhập tên thành phố hợp lệ vào ô tìm kiếm và nhấn nút tìm.
  + **Expected result**: Hiển thị kết quả thời tiết chính xác của thành phố được tìm.

### ****Kiểm tra tính năng xem lịch sử thời tiết****

* **Mục tiêu**: Đảm bảo người dùng có thể xem lại lịch sử thời tiết các ngày trước.
* **Test case**:
  + **Input**: Nhập tên thành phố và ngày hợp lệ vào ô tìm kiếm và nhấn nút tìm.
  + **Expected result**: Hiển thị kết quả thời tiết chính xác của thành phố được tìm.

**2.4. Vận hành và bảo trì:**

* Cài đặt và triển khai:
  + Hướng dẫn người dùng cách chạy ứng dụng từ console
  + Có thể cần cài đặt thêm các thư viện cần thiết cho việc kết nối với API.
* Bảo trì: Sửa lỗi phát sinh, cập nhật chức năng mới (nếu có) và cải thiện hiệu năng của ứng dụng.

**CHƯƠNG 3. KẾT QUẢ THỰC HIỆN**

**3.1. Công nghệ đã sử dụng**

* Ngôn ngữ lập trình: Java
* Công cụ: Android Studio
* Thư viện: (tùy chọn) có thể sử dụng các thư viện hỗ trợ đọc/ghi file, xử lý dữ liệu,

**3.2. Tiến độ thực hiện**

Link github tới dự án: <https://github.com/hosyson/CSE441_PROJECT>

**Hướng dẫn các bước đã thực hiện:**

**B1. Tạo dự án mới:**

* Mở Android Studio.
* Chọn "Create New Project".
* Chọn "Java" làm ngôn ngữ lập trình.
* Chọn JDK phù hợp
* Nhập tên dự án: CSE441\_PROJECT
* Chọn vị trí lưu trữ dự án.
* Nhấn "Finish" để tạo dự án.

**B2. Tạo các package:**

* Trong cửa sổ "Project", click chuột phải vào thư mục "src".
* Chọn "New" -> "Package".
* Tạo các package sau:

A screenshot of a computer program

Description automatically generated

**B3. Tạo các lớp:**

* Trong mỗi package, click chuột phải và chọn "New" -> "Java Class" để tạo các lớp tương ứng.
* Cài đặt các thuộc tính và phương thức cho từng lớp dựa trên thiết kế đã phân tích.

**B4. Cài đặt thư viện WeatherApiService:**

* Mở build.gradle
* Thêm dependency:
* *dependencies* **{** *implementation*("com.github.bumptech.glide:glide:4.11.0")  
   *implementation*("com.android.volley:volley:1.2.1")  
   *implementation*("com.squareup.picasso:picasso:2.71828")  
   *implementation*(*libs*.*appcompat*)  
   *implementation*(*libs*.*material*)  
   *implementation*(*libs*.*constraintlayout*)  
   *implementation*(*libs*.*navigation*.*fragment*)  
   *implementation*(*libs*.*navigation*.*ui*)  
   *implementation*(*libs*.*lifecycle*.*viewmodel*)  
   *implementation*(*libs*.*lifecycle*.*livedata*)  
   *implementation*(*libs*.*retrofit*)  
   *implementation*(*libs*.*converter*.*gson*)  
   *implementation*(*libs*.*room*.*runtime*)  
   *implementation*(*libs*.*legacy*.*support*.*v4*)  
   *implementation*(*libs*.*play*.*services*.*location*)  
   *compileOnly*(*libs*.*graphview*)  
   *implementation*(*libs*.*lifecycle*.*livedata*.*ktx*)  
   *implementation*(*libs*.*lifecycle*.*viewmodel*.*ktx*)  
   *annotationProcessor*(*libs*.*room*.*compiler*)  
   *implementation*("com.squareup.picasso:picasso:2.8")  
   *implementation*("com.android.support:appcompat-v7:28.0.0")  
   *implementation*("com.android.support:design:28.0.0")  
   *implementation*("com.google.android.gms:play-services-location:18.0.0")  
   *implementation*("com.google.android.gms:play-services-location:21.0.1")  
   *testImplementation*(*libs*.*junit*)  
   *androidTestImplementation*(*libs*.*ext*.*junit*)  
   *androidTestImplementation*(*libs*.*espresso*.*core*)  
   *implementation* ("com.github.bumptech.glide:glide:4.12.0")  
   *implementation* ("com.squareup.retrofit2:retrofit:2.9.0")  
   *implementation* ("com.squareup.retrofit2:converter-gson:2.9.0")  
   *implementation* ("androidx.recyclerview:recyclerview:1.2.1")  
   *implementation* ("androidx.constraintlayout:constraintlayout:2.0.4")  
   *implementation* ("com.google.android.material:material:1.4.0")  
   *implementation* ("androidx.recyclerview:recyclerview:1.2.1")  
   *implementation* ("com.squareup.okhttp3:okhttp:4.9.1")  
   *annotationProcessor* ("com.github.bumptech.glide:compiler:4.12.0")  
  **}**
* Android Studio sẽ tự động tải về và thêm thư viện vào dự án.

**B5. Viết code:**

* Bắt đầu viết code cho từng lớp, thực hiện các chức năng của ứng dụng
  + Lớp API: RetrofitClient, WeatherApiService, WeatherFetcher.
  + Lớp Adapter: CustomAdapter24h, ForecastAdapter, HistoryAdapter, TodayTempItemAdapter.
  + Lớp weather: GraphFragment, WeatherService, MainActivity.
  + Lớp database: CityDao, CityDatabase, CityEntity, WeatherDao, WeatherDatabase, WeatherEntity
  + Lớp Model:
    - currentWeather: Weather, WeatherData, WeatherDescription, WeatherResponse
    - dailyWeather: DailyWeather, DailyWeatherData, DailyWeatherResponse
    - FivedaysWeather: Forecast, Forecast5DaysResponse
    - History: Location, HistoryResponse, ForecastDay, Forecast, Day, Condition
    - hourlyWeather: HourlyWeather, HourlyWeatherData, HourlyWeatherDescription, HourlyWeatherResponse
    - WeatherData24h: WeatherData24h
* Repository:

A screenshot of a computer

Description automatically generated

* Ui:

A screenshot of a computer

Description automatically generated

* Utils: DialogUtils, LocationManager
* Tạo các layout:

A screenshot of a computer program

Description automatically generated

* Navigation: nav\_graph.xml

**B6. Xử lý API, json:**

* Tạo tài khoản dịch vụ AccuWeather API, Weather API và tải xuống file JSON chứa khóa API.
* Tham khảo tài liệu biết cách sử dụng file JSON.
* Cài đặt code xử lý xác thực.

**B7. Chạy và kiểm thử:**

* Chạy ứng dụng từ Android Studio bằng cách click chuột phải vào lớp Main (hoặc lớp chứa phương thức main) và chọn **"Run'"**.
* Kiểm tra các chức năng của ứng dụng, sửa lỗi và hoàn thiện code.

**B8. Triển khai (tùy chọn):**

* Đóng gói ứng dụng thành file JAR để dễ dàng chia sẻ và chạy trên các máy khác

**3.3. Hình ảnh sản phẩm**

**A screenshot of a weather forecast

Description automatically generated** **A screenshot of a weather forecast

Description automatically generated**

**A screenshot of a keyboard

Description automatically generated** **A screenshot of a weather forecast

Description automatically generated**

**KẾT LUẬN**

**1. Ưu điểm**

Ứng dụng thời tiết cung cấp nhiều tính năng tiện ích, giúp người dùng dễ dàng theo dõi thời tiết theo thời gian thực và dự báo cho các ngày tiếp theo. Điểm nổi bật của ứng dụng là tính tương thích với nhiều loại thiết bị, từ điện thoại đến máy tính bảng, và sử dụng kiến trúc một Activity với nhiều Fragment giúp quản lý giao diện một cách linh hoạt. Tính năng định vị vị trí hiện tại cho phép người dùng nhanh chóng xem được tình hình thời tiết xung quanh mà không cần tìm kiếm thủ công. Hơn nữa, việc tích hợp API từ AccuWeather đảm bảo dữ liệu chính xác, đáng tin cậy và liên tục cập nhật.

**2. Nhược điểm**

**Dù có nhiều tính năng hữu ích, ứng dụng thời tiết vẫn tồn tại một số nhược điểm cần khắc phục. Trước hết, việc phụ thuộc vào API của bên thứ ba có thể dẫn đến giới hạn về số lượng yêu cầu hoặc thời gian phản hồi khi sử dụng miễn phí. Hơn nữa, khi thêm nhiều thành phố và cập nhật liên tục, ứng dụng có thể gặp vấn đề về hiệu năng, đặc biệt trên các thiết bị cấu hình thấp. Ngoài ra, giao diện ứng dụng dù tiện dụng nhưng vẫn có thể chưa thân thiện hoặc phù hợp với mọi nhóm đối tượng người dùng, đòi hỏi cần có sự tối ưu hóa hơn về trải nghiệm người dùng (UX/UI).**

**3. Hướng phát triển**

**Trong tương lai, ứng dụng có thể được phát triển thêm nhiều tính năng nâng cao để đáp ứng nhu cầu đa dạng của người dùng. Một trong những hướng đi tiềm năng là tích hợp cảnh báo thời tiết xấu và hiện tượng khí hậu bất thường dựa trên dữ liệu thời gian thực. Ứng dụng cũng có thể cải thiện trải nghiệm người dùng qua việc bổ sung các tính năng như bản đồ thời tiết, hiển thị lượng mưa, độ ẩm và gió. Đặc biệt, để ứng dụng có thể tiếp cận đối tượng người dùng toàn cầu, cần thêm tính năng đa ngôn ngữ và hỗ trợ giao diện tối ưu hơn cho các thiết bị khác nhau.**

**DANH MỤC TÀI LIỆU THAM KHẢO**

## Fragment Tutorial With Example In Android Studio [Fragment Tutorial With Example In Android Studio | Abhi Android](https://abhiandroid.com/ui/fragment#gsc.tab=0)

1. Android Listview in Java with Example [Android Listview trong Java với Ví dụ - GeeksforGeeks](https://www.geeksforgeeks.org/android-listview-in-java-with-example/)
2. Introduction to Activities in Android [Introduction to Activities in Android - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-to-activities-in-android/)

**PHỤ LỤC**

* Xác thực: Cần có bước xác thực để ứng dụng có thể truy cập vào ERD..
* API Google Sheets: Tìm hiểu và sử dụng API của Google Sheets để thực hiện các thao tác đọc, ghi, cập nhật dữ liệu trên sheet.
* Hiệu năng: Tốc độ đọc/ghi dữ liệu có thể phụ thuộc vào tốc độ mạng và API của Google Sheets.
* Bảo mật: Cần bảo mật thông tin xác thực của bạn để tránh rò rỉ thông tin.

**Code minh họa đầy đủ đã thực hiện:**

**MainActivity.java**

package com.example.weather;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

import androidx.navigation.NavController;

import androidx.navigation.Navigation;

import androidx.navigation.ui.NavigationUI;

import androidx.appcompat.widget.Toolbar;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment);

NavigationUI.setupActionBarWithNavController(this, navController);

}

@Override

public boolean onSupportNavigateUp() {

NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment);

return navController.navigateUp() || super.onSupportNavigateUp();

}

}

**WeatherService:**

package com.example.weather;  
  
import Model.FivedaysWeather.Forecast5DaysResponse;  
import Model.currentWeather.WeatherResponse;  
import retrofit2.Call;  
import retrofit2.http.GET;  
import retrofit2.http.Query;  
  
public interface WeatherService {  
 @GET("current.json")  
 Call<WeatherResponse> getCurrentWeather(  
 @Query("q") String cityName,  
 @Query("key") String apiKey);  
  
 @GET("forecast.json")  
 Call<Forecast5DaysResponse> getFiveDayForecast(  
 @Query("q") String cityName,  
 @Query("days") int days,  
 @Query("key") String apiKey);  
}

**RetrofitClient:**

package api;

import retrofit2.Retrofit;

import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitClient {

private static final String BASE\_URL = "https://api.weatherapi.com/v1/";

private static final String API\_KEY = "04b3351e3f114bd7abd233330243110";

private static Retrofit retrofit;

public static Retrofit getClient() {

if (retrofit == null) {

retrofit = new Retrofit.Builder()

.baseUrl(BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.build();

}

return retrofit;

}

public static String getApiKey() {

return API\_KEY;

}

}

**WeatherApiService:** **package api;**

import Model.FivedaysWeather.Forecast5DaysResponse;

import Model.currentWeather.WeatherResponse;

import Model.dailyWeather.DailyWeatherResponse;

import Model.history.HistoryResponse;

import Model.hourlyWeather.HourlyWeatherResponse;

import retrofit2.Call;

import retrofit2.http.GET;

import retrofit2.http.Query;

public interface WeatherApiService {

// Current weather

@GET("current")

Call<WeatherResponse> getCurrentWeatherByCity(

@Query("city") String city,

@Query("key") String apiKey

);

@GET("current")

Call<WeatherResponse> getCurrentWeatherByLocation(

@Query("lat") Double latitude,

@Query("lon") Double longitude,

@Query("key") String apiKey

);

// Hourly weather

@GET("history/hourly")

Call<HourlyWeatherResponse> getHourlyHistoryByCity(

@Query("city") String city,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

@GET("history/hourly")

Call<HourlyWeatherResponse> getHourlyHistoryByLocation(

@Query("lat") Double latitude,

@Query("lon") Double longitude,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

@GET("forecast/hourly")

Call<HourlyWeatherResponse> getHourlyForecastByCity(

@Query("city") String city,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

@GET("forecast/hourly")

Call<HourlyWeatherResponse> getHourlyForecastByLocation(

@Query("lat") Double latitude,

@Query("lon") Double longitude,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

// Daily weather

@GET("forecast/daily")

Call<DailyWeatherResponse> getDailyForecastByCity(

@Query("city") String city,

@Query("days") int days,

@Query("key") String apiKey

);

@GET("forecast/daily")

Call<DailyWeatherResponse> getDailyForecastByLocation(

@Query("lat") Double latitude,

@Query("lon") Double longitude,

@Query("days") int days,

@Query("key") String apiKey

);

@GET("history/daily")

Call<DailyWeatherResponse> getHistoryDailyByCity(

@Query("city") String city,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

@GET("history/daily")

Call<DailyWeatherResponse> getHistoryDailyByLocation(

@Query("lat") Double latitude,

@Query("lon") Double longitude,

@Query("start\_date") String startDate,

@Query("end\_date") String endDate,

@Query("key") String apiKey

);

@GET("forecast.json")

Call<Forecast5DaysResponse> getForecast5DaysResponse(

@Query("key") String apiKey,

@Query("q") String location,

@Query("days") int days

);

@GET("history.json")

Call<HistoryResponse> getHistoricalWeather(

@Query("key") String apiKey,

@Query("q") String location,

@Query("dt") String date

);

}

**Weather24h.java**

package com.example.weather;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.view.PixelCopy;

import android.view.View;

import android.widget.ImageView;

import android.widget.ListView;

import android.widget.TextView;

import androidx.activity.EdgeToEdge;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.graphics.Insets;

import androidx.core.view.ViewCompat;

import androidx.core.view.WindowInsetsCompat;

import com.android.volley.Request;

import com.android.volley.RequestQueue;

import com.android.volley.Response;

import com.android.volley.VolleyError;

import com.android.volley.toolbox.StringRequest;

import com.android.volley.toolbox.Volley;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.Locale;

public class Weather24h extends AppCompatActivity {

ImageView imgback;

TextView txtname;

ListView lv;

CustomAdapter customAdapter;

ArrayList<WeatherData> mangthoitiet;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

EdgeToEdge.enable(this);

setContentView(R.layout.activity\_weather24h);

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

return insets;

});

Anhxa();

Intent intent = getIntent();

String city = intent.getStringExtra("name");

imgback.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

onBackPressed();

}

});

Get24HoursData();

}

private void Anhxa() {

imgback = (ImageView) findViewById(R.id.imageviewBack);

txtname = (TextView) findViewById(R.id.textviewTenthanhpho);

lv = (ListView) findViewById(R.id.listviewWeather24h);

mangthoitiet = new ArrayList<WeatherData>();

customAdapter = new CustomAdapter(Weather24h.this,mangthoitiet);

lv.setAdapter(customAdapter);

}

private void Get24HoursData() {

String url = "https://dataservice.accuweather.com/forecasts/v1/hourly/12hour/1-353412\_1\_AL?apikey=QbGB0fmYi9oCmAPLZSzOb1gwJIBU3CDQ";

RequestQueue requestQueue = Volley.newRequestQueue(Weather24h.this);

StringRequest stringRequest= new StringRequest(Request.Method.GET, url,

new Response.Listener<String>() {

@Override

public void onResponse(String response) {

try {

JSONArray jsonArrayList = new JSONArray(response);

for (int i=0;i<jsonArrayList.length();i++){

JSONObject jsonObjectList = jsonArrayList.getJSONObject(i);

String ngay = jsonObjectList.getString("EpochDateTime");

long l = Long.valueOf(ngay);

Date date = new Date(l\*1000L);

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("EEEE",Locale.ENGLISH);

String Day = simpleDateFormat.format(date);

SimpleDateFormat hourFormat = new SimpleDateFormat("HH:mm a");

String hour = hourFormat.format(date);

JSONObject jsonObjectTemp = jsonObjectList.getJSONObject("Temperature");

String temp = jsonObjectTemp.getString("Value");

int intdoC = (int) ((Double.parseDouble(temp)-32)/1.8);

String stringdoC = String.valueOf(intdoC);

String status= jsonObjectList.getString("IconPhrase");

String icon = jsonObjectList.getString("WeatherIcon");

mangthoitiet.add(new WeatherData(Day,status,icon,hour,stringdoC));

}

customAdapter.notifyDataSetChanged();

} catch (JSONException e) {

throw new RuntimeException(e);

}

}

}, new Response.ErrorListener() {

@Override

public void onErrorResponse(VolleyError error) {

Log.e("Loi tra ve json", "Error occurred: " + error.getMessage());

}

});

requestQueue.add(stringRequest);

}

}

**WeatherData.java**

package com.example.weather;

public class WeatherData {

public String Day;

public String Status;

public String Image;

public String MaxTemp;

public String MinTemp;

public WeatherData(String day,String status,String image,String hour,String minTemp) {

Day=day;

Status=status;

Image=image;

MaxTemp=hour;

MinTemp=minTemp;

}

}

**CustomAdapter.java**

package com.example.weather;

import android.content.Context;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.ImageView;

import android.widget.TextView;

import com.squareup.picasso.Picasso;

import java.util.ArrayList;

public class CustomAdapter extends BaseAdapter {

Context context;

ArrayList<WeatherData> arrayList;

public CustomAdapter(Context context, ArrayList<WeatherData> arrayList) {

this.context=context;

this.arrayList = arrayList;

}

public CustomAdapter(Context context) {

this.context = context;

}

@Override

public int getCount() {

return arrayList.size();

}

@Override

public Object getItem(int i) {

return arrayList.get(i);

}

@Override

public long getItemId(int i) {

return 0;

}

@Override

public View getView(int i, View view, ViewGroup viewGroup) {

LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

view = inflater.inflate(R.layout.item\_listview\_weather24h,null);

WeatherData weather = arrayList.get(i);

TextView txtDay = (TextView) view.findViewById(R.id.textviewThu);

TextView txtTrangThai = (TextView) view.findViewById(R.id.textviewTrangThai);

TextView txtMaxTemp = (TextView) view.findViewById(R.id.textviewMaxTemp);

TextView txtMinTemp = (TextView) view.findViewById(R.id.textviewMinTemp);

ImageView imgStatus = (ImageView) view.findViewById(R.id.imageviewTrangThai);

txtDay.setText(weather.Day);

txtTrangThai.setText(weather.Status);

txtMaxTemp.setText(weather.MaxTemp);

txtMinTemp.setText(weather.MinTemp+"°C");

Picasso.with(context).load("https://developer.accuweather.com/sites/default/files/" + String.format("%02d", Integer.parseInt(weather.Image)) + "-s.png").into(imgStatus);

return view;

}

}

**AddCity.java**

package com.example.weather;

import android.os.Bundle;

import android.view.MotionEvent;

import android.view.View;

import android.view.inputmethod.InputMethodManager;

import android.widget.SearchView;

import android.widget.TextView;

import android.widget.Toast;

import androidx.activity.EdgeToEdge;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.graphics.Insets;

import androidx.core.view.ViewCompat;

import androidx.core.view.WindowInsetsCompat;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;

import java.util.List;

import Model.WeatherData;

import adapter.CityAdapter;

public class AddCity extends AppCompatActivity {

RecyclerView cityRecyclerView;

SearchView searchBar;

CityAdapter cityAdapter;

private List<WeatherData> weatherDataList;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

EdgeToEdge.enable(this);

setContentView(R.layout.activity\_add\_city);

ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

return insets;

});

cityRecyclerView = findViewById(R.id.cityRecyclerView);

searchBar = findViewById(R.id.searchBar);

weatherDataList = new ArrayList<>();

weatherDataList.add(new WeatherData(10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10, 10.0, 10.0, 10, "10/10/10", "10:10", "10", "10"));

cityAdapter = new CityAdapter(weatherDataList);

cityRecyclerView.setAdapter(cityAdapter);

cityRecyclerView.setLayoutManager(new LinearLayoutManager(this));

searchBar.setOnQueryTextFocusChangeListener((v, hasFocus) -> {

if (hasFocus) {

// SearchView gained focus

Toast.makeText(this, "Gained focus", Toast.LENGTH\_SHORT).show();

} else {

// SearchView lost focus

Toast.makeText(this, "Lost focus", Toast.LENGTH\_SHORT).show();

}

});

}

@Override

public boolean dispatchTouchEvent(MotionEvent ev) {

if (ev.getAction() == MotionEvent.ACTION\_DOWN) {

View v = getCurrentFocus(); // Get the currently focused view

if (v != null && searchBar.isFocused()) {

// Check if the touch is outside the SearchView

int[] searchViewLocation = new int[2];

searchBar.getLocationOnScreen(searchViewLocation);

float x = ev.getRawX();

float y = ev.getRawY();

if (x < searchViewLocation[0] || x > (searchViewLocation[0] + searchBar.getWidth())

|| y < searchViewLocation[1] || y > (searchViewLocation[1] + searchBar.getHeight())) {

// Hide the keyboard and clear focus from SearchView

searchBar.clearFocus();

View view = this.getCurrentFocus();

hideKeyboard(view);

}

}

}

return super.dispatchTouchEvent(ev);

}

private void hideKeyboard(View view) {

if (view != null) {

InputMethodManager imm = (InputMethodManager) getSystemService(INPUT\_METHOD\_SERVICE);

if (imm != null) {

imm.hideSoftInputFromWindow(view.getWindowToken(), 0);

}

}

}

}

**ForecastAdapter:**

package adapter;

import android.content.Context;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ImageView;

import android.widget.TextView;

import com.bumptech.glide.Glide; // Thêm import Glide

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.example.weather.R;

import java.util.List;

import Model.FivedaysWeather.Forecast;

import Model.FivedaysWeather.Forecast5DaysResponse;

public class ForecastAdapter extends RecyclerView.Adapter<ForecastAdapter.ForecastViewHolder> {

private Context context;

private List<Forecast> forecastList;

public ForecastAdapter(Context context, List<Forecast> forecastList) {

this.context = context;

this.forecastList = forecastList;

}

@NonNull

@Override

public ForecastViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view = LayoutInflater.from(context).inflate(R.layout.item\_forecast, parent, false);

return new ForecastViewHolder(view);

}

@Override

public void onBindViewHolder(@NonNull ForecastViewHolder holder, int position) {

Forecast forecast = forecastList.get(position);

holder.tvDate.setText(forecast.getDate());

holder.tvTemperature.setText(forecast.getTemperature() + "°C");

holder.tvCondition.setText(forecast.getWeatherCondition());

// Tải biểu tượng thời tiết

String iconUrl = "https:" + forecast.getIconUrl(); // Thêm "https:" vào trước URL nếu cần

Glide.with(context)

.load(iconUrl)

.into(holder.imgWeatherIcon);

}

@Override

public int getItemCount() {

return forecastList.size();

}

public static class ForecastViewHolder extends RecyclerView.ViewHolder {

TextView tvDate;

TextView tvTemperature;

TextView tvCondition;

ImageView imgWeatherIcon;

public ForecastViewHolder(@NonNull View itemView) {

super(itemView);

tvDate = itemView.findViewById(R.id.tvDate);

tvTemperature = itemView.findViewById(R.id.tvTemperature);

tvCondition = itemView.findViewById(R.id.tvCondition);

imgWeatherIcon = itemView.findViewById(R.id.imgWeather);

}

}

}

**HistoryAdapter:**

package adapter;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.example.weather.R;

import java.util.List;

import Model.history.ForecastDay;

public class HistoryAdapter extends RecyclerView.Adapter<HistoryAdapter.HistoryViewHolder> {

private final List<ForecastDay> historyList;

public HistoryAdapter(List<ForecastDay> historyList) {

this.historyList = historyList;

}

@NonNull

@Override

public HistoryViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item\_history\_day, parent, false);

return new HistoryViewHolder(view);

}

@Override

public void onBindViewHolder(@NonNull HistoryViewHolder holder, int position) {

ForecastDay forecastDay = historyList.get(position);

// Kiểm tra xem forecastDay và day có khác null không

if (forecastDay != null && forecastDay.getDay() != null) {

holder.tvDate.setText(forecastDay.getDate() != null ? forecastDay.getDate() : "N/A");

// Lấy nhiệt độ tối đa và tối thiểu

double maxTemp = forecastDay.getDay().getMaxTempC(); // Sử dụng getMaxTempC()

double minTemp = forecastDay.getDay().getMinTempC(); // Sử dụng getMinTempC()

holder.tvTemp.setText(String.format("Max Temp: %.1f°C, Min Temp: %.1f°C", maxTemp, minTemp));

// Lấy điều kiện thời tiết

if (forecastDay.getDay().getCondition() != null) {

holder.tvCondition.setText(String.format("Condition: %s", forecastDay.getDay().getCondition().getText()));

} else {

holder.tvCondition.setText("Condition: N/A");

}

} else {

// Thiết lập giá trị mặc định trong trường hợp dữ liệu bị thiếu

holder.tvDate.setText("N/A");

holder.tvTemp.setText("Temperature: N/A");

holder.tvCondition.setText("Condition: N/A");

}

}

@Override

public int getItemCount() {

return historyList != null ? historyList.size() : 0;

}

public static class HistoryViewHolder extends RecyclerView.ViewHolder {

TextView tvDate, tvTemp, tvCondition;

public HistoryViewHolder(@NonNull View itemView) {

super(itemView);

tvDate = itemView.findViewById(R.id.tvDate);

tvTemp = itemView.findViewById(R.id.tvTemp);

tvCondition = itemView.findViewById(R.id.tvCondition);

}

}

}

**TodayTempItemAdapter:**

package adapter;

import android.app.Activity;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ArrayAdapter;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.annotation.Nullable;

import com.example.weather.R;

import java.util.ArrayList;

import Model.currentWeather.WeatherData;

public class TodayTempItemAdapter extends ArrayAdapter<WeatherData> {

Activity context;

ArrayList<WeatherData> arrayList;

int layoutId;

public TodayTempItemAdapter(Activity context, int layoutId, ArrayList<WeatherData> arrayList) {

super(context, layoutId, arrayList);

this.context = context;

this.arrayList = arrayList;

this.layoutId = layoutId;

}

@NonNull

@Override

public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {

LayoutInflater inflater = context.getLayoutInflater();

convertView = inflater.inflate(layoutId, null);

TextView txtTodayHour = convertView.findViewById(R.id.txtTodayHour);

TextView txtTodayTemp = convertView.findViewById(R.id.txtTodayTemp);

WeatherData weatherDataResponse = arrayList.get(position);

return convertView;

}

}