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 | Chất lượng không khí |

**MỤC LỤC**

[CHƯƠNG 1. MÔ TẢ BÀI TOÁN 6](#_Toc19670)

[1.1. Giới thiệu 6](#_Toc4885)

[1.2. Chức năng chính 6](#_Toc28676)

[1.3. Yêu cầu phi chức năng 6](#_Toc22622)

[CHƯƠNG 2. PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ HỆ THỐNG 7](#_Toc14157)

[2.1. Phân tích yêu cầu: 7](#_Toc21051)

[1. Xem thời tiết theo tùy chọn: 8](#_Toc6960)

[2. Xem chi tiết thời tiết: 8](#_Toc28743)

[3. Thêm/xóa địa điểm: 8](#_Toc10636)

[4. Tìm kiếm và xem thời tiết theo tên địa điểm: 8](#_Toc15055)

[5. Xem thời tiết theo vị trí hiện tại: 8](#_Toc23489)

[6. Xem chỉ số chất lượng không khí (AQI): 8](#_Toc5871)

[8. Yêu cầu phi chức năng: 9](#_Toc30409)

[2.2. Thiết kế hệ thống: 9](#_Toc8383)

[2.3. Triển khai: 15](#_Toc3390)

[Kiểm tra chức năng lấy thời tiết hiện tại 15](#_Toc29647)

[Kiểm tra chức năng tìm kiếm thành phố 16](#_Toc14948)

[Kiểm tra chức năng dự báo thời tiết theo ngày 16](#_Toc15479)

[Kiểm tra chức năng thêm địa điểm yêu thích 16](#_Toc9907)

[Kiểm tra chức năng xóa địa điểm khỏi danh sách yêu thích 17](#_Toc210)

[Kiểm tra chức năng xem chi tiết thời tiết theo giờ 17](#_Toc10608)

[Kiểm tra lỗi kết nối mạng 18](#_Toc18006)

[Kiểm tra giao diện đa màn hình 18](#_Toc14423)

[Kiểm tra thông tin thời tiết trực tiếp theo vị trí 18](#_Toc8551)

[Kiểm tra tính năng tìm kiếm thành phố 19](#_Toc7534)

[2.4. Vận hành và bảo trì: 19](#_Toc26207)

[CHƯƠNG 3. KẾT QUẢ THỰC HIỆN 20](#_Toc27756)

[3.1. Công nghệ đã sử dụng 20](#_Toc12245)

[3.2. Tiến độ thực hiện 20](#_Toc15451)

[3.3. Hình ảnh sản phẩm 22](#_Toc19506)

[KẾT LUẬN 28](#_Toc3943)

[DANH MỤC TÀI LIỆU THAM KHẢO 28](#_Toc29904)

[PHỤ LỤC 30](#_Toc826)

**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 chỉ số chất lượng không khí(AQI).
* 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 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 quan tâm đến môi trường**: Những người muốn theo dõi chất lượng không khí (AQI) ở nơi mình sống hoặc các địa điểm khác, đặc biệt quan trọng đối với những người có vấn đề về sức khỏe liên quan đến ô nhiễm.
* **Người dùng 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.

**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; xem AQI) 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 chỉ số chất lượng không khí (AQI):

* Cung cấp thông tin về **AQI** để người dùng biết mức độ ô nhiễm không khí tại vị trí hiện tại hoặc các địa điểm đã lưu.
* Hiển thị rõ ràng chỉ số AQI và cảnh báo nếu mức ô nhiễm cao.

**7. 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.

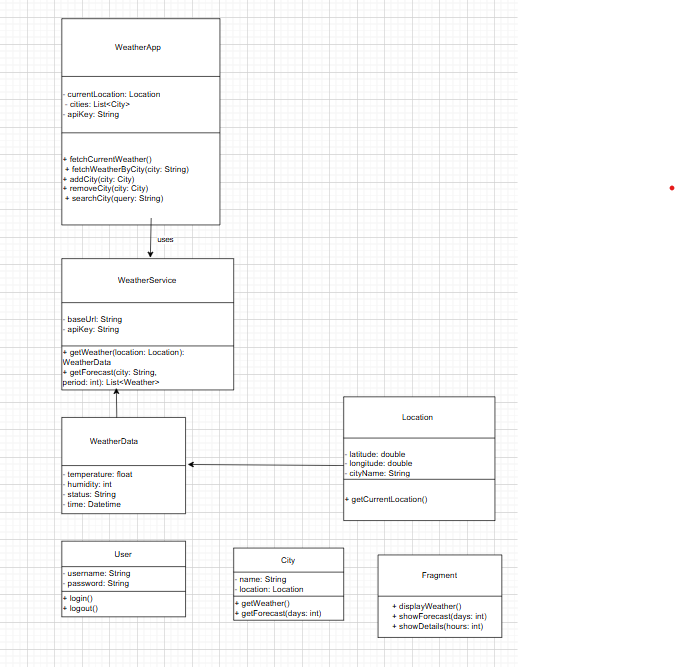
### 8. Yêu cầu phi chức năng:

* **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 và AQI nhanh chóng.
* **Độ tin cậy**: Dữ liệu thời tiết và AQI 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:**

* Tập trung vào các thực thể nghiệp vụ và mối quan hệ giữa chúng.
* Thường không chứa các lớp liên quan đến giao diện người dùng, cơ sở dữ liệu, hay các chi tiết kỹ thuật khác.
* Ví dụ: trong ứng dụng Todo App, biểu đồ lớp cho mô hình miền **chỉ chứa** thực thể **Task** với các thuộc tính như *id, title, description, deadline, priority, completed*.



**Biểu đồ lớp cho MVC:**

* Mô tả kiến trúc của ứng dụng, bao gồm các lớp trong Model, View và Controller.
* Thể hiện mối quan hệ giữa các lớp trong MVC và cách chúng tương tác với nhau.
* Ví dụ: trong ứng dụng Todo App, biểu đồ lớp cho MVC sẽ chứa các lớp như Task, TaskList, TaskView, TaskController, FileHandler (hoặc GoogleSheetHandler).

**Thiết kế kiến trúc (MVC):**

* **Model:**
  + **WeatherData**: Chứa các thông tin chi tiết về thời tiết như nhiệt độ, độ ẩm, điều kiện thời tiết, v.v.
  + **Location**: Đại diện cho một địa điểm mà người dùng có thể thêm để theo dõi thời tiết.
  + **AQIData**: Chứa thông tin về chỉ số chất lượng không khí (AQI).
  + **WeatherRepository**: Chứa logic lấy dữ liệu thời tiết từ các nguồn (có thể là API hoặc cơ sở dữ liệu cục bộ).
* **View:**
  + **WeatherView**: Hiển thị dữ liệu thời tiết dựa trên thông tin từ WeatherData và AQIData.
  + **LocationView**: Hiển thị danh sách các thành phố mà người dùng đã thêm vào
* **Controller:**
  + **WeatherController**: Quản lý các yêu cầu của người dùng liên quan đến dự báo thời tiết và chất lượng không khí.
  + **LocationController**: Quản lý việc thêm/xóa các địa điểm mà người dùng muốn theo dõi.

|  |
| --- |
| **Dựa trên kiến trúc MVC đã chọn, ta có thể xác định các lớp sau:**   * **Lớp Task:**   + **Thuộc tính:** ID, tiêu đề, mô tả, deadline, mức độ ưu tiên, trạng thái.   + **Phương thức:**     - Getters và setters cho các thuộc tính.     - toString(): trả về chuỗi mô tả công việc. * **Lớp TaskList:**   + **Thuộc tính:** List<Task> tasks (danh sách các công việc).   + **Phương thức:**     - addTask(Task task): thêm công việc mới vào danh sách.     - removeTask(int taskId): xóa công việc theo ID.     - getTaskById(int taskId): tìm kiếm công việc theo ID.     - updateTask(Task task): cập nhật thông tin công việc.     - getAllTasks(): lấy danh sách tất cả công việc.     - getTasksByStatus(String status): lọc công việc theo trạng thái.     - getTasksByPriority(String priority): lọc công việc theo mức độ ưu tiên.     - getTasksByDeadline(Date deadline): lọc công việc theo deadline. * **Lớp GoogleSheetHandler:**   + **Thuộc tính:**     - String spreadsheetId: ID của Google Sheet.     - String sheetName: Tên của sheet chứa dữ liệu.     - Sheets service: Đối tượng Sheets dùng để tương tác với Google Sheets API.   + **Phương thức:**     - readTasks(): đọc dữ liệu công việc từ Google Sheet và trả về List<Task>.     - writeTasks(List<Task> tasks): ghi danh sách công việc vào Google Sheet.     - updateTask(Task task): cập nhật công việc trên Google Sheet.     - deleteTask(int taskId): xóa công việc trên Google Sheet. * **Lớp TaskView:**   + **Phương thức:**     - displayMenu(): hiển thị menu chính.     - displayTasks(List<Task> tasks): hiển thị danh sách công việc.     - getInput(String prompt): lấy input từ người dùng.     - showMessage(String message): hiển thị thông báo. * **Lớp TaskController:**   + **Thuộc tính:**     - TaskList taskList: đối tượng TaskList.     - GoogleSheetHandler sheetHandler: đối tượng GoogleSheetHandler.     - TaskView taskView: đối tượng TaskView.   + **Phương thức:**     - handleUserChoice(int choice): xử lý lựa chọn của người dùng từ menu.     - addNewTask(): thêm công việc mới.     - displayTasks(): hiển thị danh sách công việc.     - markTaskAsCompleted(): đánh dấu công việc hoàn thành.     - editTask(): chỉnh sửa công việc.     - deleteTask(): xóa công việc.   **Mối quan hệ giữa các lớp:**   * TaskController sử dụng TaskList và GoogleSheetHandler để quản lý và lưu trữ dữ liệu. * TaskController sử dụng TaskView để tương tác với người dùng. * TaskList chứa danh sách các đối tượng Task.   **Lưu ý:** Biểu đồ lớp có thể được vẽ chi tiết hơn bằng cách sử dụng UML (Unified Modeling Language) để thể hiện rõ ràng các thuộc tính, phương thức, và mối quan hệ giữa các lớp.  BieuDoLopChoMoHinhMVC |

**Thiết kế cơ sở dữ liệu:**

* 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

A screenshot of a computer

Description automatically generated

**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 (ví dụ: Google API Client Library for Java).
  + 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.

### ****Kiểm tra chức năng thêm địa điểm yêu thích****

* **Mục tiêu**: Đảm bảo rằng người dùng có thể thêm thành phố vào danh sách yêu thích.
* **Test case 1**:
  + **Input**: Tìm kiếm và thêm thành phố "Los Angeles" vào danh sách yêu thích.
  + **Expected result**: "Los Angeles" xuất hiện trong danh sách các thành phố yêu thích.
* **Test case 2**:
  + **Input**: Thêm một thành phố đã có trong danh sách yêu thích (ví dụ: "Los Angeles").
  + **Expected result**: Ứng dụng không cho phép thêm lại thành phố này và hiển thị thông báo "Thành phố đã có trong danh sách yêu thích".

### ****Kiểm tra chức năng xóa địa điểm khỏi danh sách yêu thích****

* **Mục tiêu**: Đảm bảo rằng người dùng có thể xóa thành phố khỏi danh sách yêu thích.
* **Test case**:
  + **Input**: Nhấn vào nút xóa bên cạnh "Los Angeles" trong danh sách yêu thích.
  + **Expected result**: "Los Angeles" bị xóa khỏi danh sách.

### ****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.

**2.4. Vận hành và bảo trì:**

* Cài đặt và triển khai:
  + Hướng dẫn người dùng cách chạy ứng dụng từ console (ví dụ: java -jar TodoApp.jar).
  + Ngoài hướng dẫn chạy ứng dụng, cần hướng dẫn người dùng cách tạo Google Sheet và chia sẻ quyền truy cập cho ứng dụng.
  + Có thể cần cài đặt thêm các thư viện cần thiết cho việc kết nối với Google Sheets.
* 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: IntelliJ IDEA
* 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 IntelliJ IDEA.
* Chọn "Create New Project".
* Chọn "Java" làm ngôn ngữ lập trình.
* Chọn JDK phù hợp (ví dụ: JDK 11 hoặc mới hơn).
* Nhập tên dự án (ví dụ: "TodoApp").
* 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:
  + model: chứa các lớp Task và TaskList.
  + view: chứa lớp TaskView.
  + controller: chứa lớp TaskController.
  + handler: chứa lớp GoogleSheetHandler.

**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 Google Sheets API:**

* Mở file pom.xml (nếu bạn đang sử dụng Maven) hoặc build.gradle (nếu bạn đang sử dụng Gradle).
* Thêm dependency cho thư viện Google API Client Library for Java. Ví dụ, trong Maven:
* IntelliJ IDEA 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:
  + **Task:** cài đặt các thuộc tính và phương thức cơ bản.
  + **TaskList:** cài đặt các phương thức để quản lý danh sách công việc.
  + **GoogleSheetHandler:** cài đặt các phương thức để kết nối, đọc, ghi, cập nhật, xóa dữ liệu trên Google Sheet.
  + **TaskView:** cài đặt các phương thức để hiển thị giao diện console và tương tác với người dùng.
  + **TaskController:** cài đặt logic xử lý yêu cầu của người dùng, điều phối các lớp khác để hoàn thành các chức năng.

**B6. Xử lý xác thực Google Sheets:**

* Tạo tài khoản dịch vụ trên Google Cloud Platform và tải xuống file JSON chứa khóa API.
* Tham khảo tài liệu của Google Sheets API để biết cách sử dụng file JSON này để xác thực ứng dụng của bạn.
* Cài đặt code xử lý xác thực trong GoogleSheetHandler.

**B7. Chạy và kiểm thử:**

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

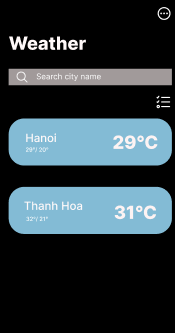
**B8. Triển khai (tùy chọn):**

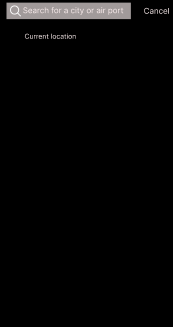
* Nếu muốn đóng gói ứng dụng thành file JAR để dễ dàng chia sẻ và chạy trên các máy khác, bạn có thể sử dụng chức năng "Build Artifacts" của IntelliJ IDEA.

**3.3. Hình ảnh sản phẩm**











**KẾT LUẬN**

**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.

**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).

**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" \l "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 Google Sheet. Bạn có thể sử dụng OAuth2 để cho phép ứng dụng truy cập vào tài khoản Google của bạn.
* 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.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.TextView;

import androidx.annotation.Nullable;

import androidx.appcompat.app.AppCompatActivity;

import com.google.android.material.floatingactionbutton.FloatingActionButton;

public class MainActivity extends AppCompatActivity {

FloatingActionButton floatingActionButton;

TextView next\_24\_hours;

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_weather);

floatingActionButton = findViewById(R.id.floatingActionButton4);

floatingActionButton.setOnClickListener(view -> {

Intent intent = new Intent(MainActivity.this, AddCity.class);

startActivity(intent);

});

next\_24\_hours = findViewById(R.id.next\_24\_hours);

next\_24\_hours.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Intent intent = new Intent(MainActivity.this,Weather24h.class);

intent.putExtra("city","Ha noi");

startActivity(intent);

}

});

}

}

**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);

}

}

}

}