



Instituto Politécnico Nacional
Unidad Profesional Interdisciplinaria de Ingeniería
Campus Guanajuato



Proyecto de Investigación

Investigación, desarrollo y análisis de un sistema de información meteorológica automático

para obtener el título de Ingeniero en Aeronáutica

Oswaldo Arias Estrada
Presenta

Arturo Hernández Hernández
Asesor

1. Abstract

The development of an automatic weather station is presented, which is capable of collecting and transmitting weather data to a server for further processing and visualization. The weather station is designed to be inexpensive and easy to build, using off-the-shelf components.

Analysis of the data collected by the weather station is performed to determine the quality of the data, as well as the accuracy of the weather station. The results show that the weather station is capable of collecting data with a high degree of accuracy, and that the data is suitable for use in aviation.

2. Abstracto

Se presenta el desarrollo de una estación meteorológica automática, capaz de recolectar y transmitir datos meteorológicos a un servidor para su posterior procesamiento y visualización. La estación meteorológica está diseñada para ser económica y fácil de construir, utilizando componentes pre-diseñados.

El análisis de los datos recolectados por la estación meteorológica se realiza para determinar la calidad de los datos, así como la precisión de la estación meteorológica. Los resultados muestran que la estación meteorológica es capaz de recolectar datos con un alto grado de precisión, y que los datos son adecuados para su uso en aviación.

3. Dedicatoria

A Mariana, mi novia quien agradezco por su confianza y compañía, ella es la que me ha dado la fuerza para seguir persiguiendo mis sueños y seguir aumentando mis metas, siempre se puede ser mejor, y a su lado se que los dos lo seremos.

A mis padres, Silvia y José, quienes siempre me han alentado y apoyado a lo largo de mi trayectoria académica. Su fe inquebrantable en mí ha sido una fuente constante de motivación.

A mi hermana, Romina, a quien agradezco aguantarme tanto, me llena de orgullo ver sus logros y anticipar todo lo que alcanzará, de ingeniero a futura licenciada, le deseo lo mejor.

4. Agradecimientos

Expreso mi más sincero agradecimiento a:

- A mi amiga, Mireya Arizbeth Mercado Delgado, por siempre creer en mí, por más deprecario que fuera el estado en el que me viera, al final si se pudo.
- A mi asesor de proyecto Arturo Hernández Hernández, por brindarme información, comentarios y apoyo invaluable durante la investigación y el desarrollo del proyecto.
- A mis profesores, por su guía, consejo y aliento a lo largo de mi trayectoria académica.
- A Martin Haug y Laurenz Mädje por el desarrollo de [Typst](#), la herramienta que me ha permitido realizar la escritura de este proyecto de investigación.

Gracias.

5. Resumen

Las estaciones meteorológicas automáticas tienen como objetivo recopilar datos meteorológicos precisos y actualizados de múltiples ubicaciones utilizando sensores y componentes para la recopilación y transmisión de datos a una red de servidores para su posterior procesamiento, análisis y visualización.

El Aviation Weather Center (AWC) es el organismo perteneciente al gobierno de los Estados Unidos de América, responsable de proporcionar información meteorológica a los usuarios de la aviación, incluidos pilotos, aeropuertos y equipos experimentales. Uno de los servicios más empleados del AWC es el servicio de reportes METAR (Meteorological Terminal Aviation Routine Weather Report), que es un formato estandarizado para informar las condiciones climáticas en los aeropuertos.

La información METAR es recolectada a través de múltiples estaciones meteorológicas que pueden ser manuales o automatizadas, las cuales se encuentran en aeródromo y se encargan de tomar observaciones meteorológicas, respecto a la temperatura, punto de rocío, presión, precipitación, así como la dirección y velocidad del viento, entre otras más. La información que proveen estas estaciones es vital para las aerolíneas, pilotos y demás usuarios del espacio aéreo.

El objetivo principal del proyecto es proporcionar datos meteorológicos fiables y precisos a una mayor cantidad de usuarios. Para lograrlo, el diseño de la estación meteorológica se realizará tomando en cuenta aspectos como la asequibilidad, confiabilidad, calidad, así como la facilidad de instalación, administración y mantenimiento de la misma.

A partir de los datos obtenidos por la estación meteorológica, se realizará un análisis de los mismos para determinar la calidad de los datos, así como la precisión de la estación. Se espera que los resultados obtenidos mediante la recolección y análisis de las observaciones nos permitan determinar las posibilidad del uso de alternativas más económicas para la observación meteorológica, así como encontrar un rango de público objetivo interesado en el uso de la tecnología desarrollada.

6. Índice

1. Abstract	2
2. Abtracto	2
3. Dedicatoria	3
4. Agradecimientos	4
5. Resumen	5
6. Índice	6
7. Nomenclatura	10
7.1. Almacenamiento de datos	10
7.2. Análisis de datos	10
7.3. Autoridades	11
7.3.1. Internacionales	11
7.3.2. Estados Unidos de América	11
7.3.3. México	12
7.4. Aviación	12
7.5. Desarrollo	13
7.5.1. Hardware	13
7.5.2. Software	14
7.6. Meteorología	15
7.7. Servicios de datos meteorológicos	16
8. Introducción	17
9. Problemática	19
10. Hipótesis	20
11. Justificación	21
12. Objetivo	22
12.1. General	22
12.2. Específico	22
13. Marco Teórico	23
13.1. Antecedentes	23
13.2. Bases Teóricas	23
13.2.1. Aeroespacial	23
13.2.2. Aeronáutica	24
13.2.3. Aviación	24
13.2.4. Aviónica	24
13.2.5. Electrónica	24
13.2.6. Meteorología	25
13.3. Bases Conceptuales	25
13.3.1. Desarrollo de Software	25
13.3.2. Estaciones METAR	25
13.3.3. Microcontroladores	26

13.3.3.1. Arduino	26
13.3.3.2. ESP32	26
13.3.4. Sensores	26
13.3.5. Sistema de Bases de Datos	27
13.3.6. Sistema de Comunicaciones	27
14. Descripción de Símbolos y Convenciones	28
14.1. Notas	28
14.2. Citas	28
14.3. Referencias	28
14.4. Definiciones	28
14.5. Código	29
15. Desarrollo	30
15.1. Investigación	30
15.1.1. Observación Meteorológica	30
15.1.2. Reportes Meteorológicos. METAR, TAF y NOTAM	30
15.1.3. Estaciones Meteorológicas	37
15.1.4. Estaciones Meteorológicas en Aeropuertos	37
15.1.5. Estaciones Meteorológicas Automáticas	37
15.1.6. Sensores de Condiciones Ambientales	39
15.1.7. Sistemas de Bases de Datos	40
15.1.8. Sistemas de Comunicaciones	40
15.1.9. Servicios de Datos Meteorológicos	41
15.2. Prototipo	43
15.2.1. PocketBase	43
15.2.1.1. Entorno Local	44
15.2.1.2. Entorno Remoto	53
15.2.2. ESP32 - BME680	53
15.2.2.1. Guías, Manuales y Tutoriales	54
15.2.2.2. Componentes	54
15.2.2.3. Funcionalidades	56
15.2.2.4. Pinouts	56
15.2.2.5. Esquemático	59
15.2.2.6. Diagrama de Flujo	59
15.2.2.7. Código	61
15.2.2.8. Pruebas	78
15.2.3. Estación Meteorológica	79
15.2.3.1. Guías, Manuales y Tutoriales	80
15.2.3.2. Componentes	80
15.2.3.3. Funcionalidades	83
15.2.3.4. Pinouts	84
15.2.3.5. Ensamblaje	87

15.2.3.6. Diagrama de Flujo	102
15.2.3.7. Código	103
15.2.3.8. Pruebas	117
15.2.4. Servidor	121
15.2.4.1. Guías, Manuales y Tutoriales	121
15.2.4.2. Características	122
15.2.4.3. Servicios	122
15.2.4.4. Diagrama de Flujo	123
15.2.4.5. Código	124
15.2.4.6. Pruebas	134
15.3. Resultados	140
15.4. Análisis de Resultados	141
15.4.1. Extracción de Datos	141
15.4.2. Refinamiento de Datos	144
15.4.3. Análisis de Datos	146
15.4.3.1. Dewpoint	149
15.4.3.2. Humidity	152
15.4.3.3. Precipitación	155
15.4.3.4. Presión	158
15.4.3.5. Temperatura	161
15.4.3.6. Índice UV	164
15.4.3.7. Velocidad de Viento	167
15.4.4. Análisis de Costos	173
15.4.4.1. Estación	173
15.4.4.2. Estaciones Comerciales	174
16. Conclusión	176
17. Glosario	179
18. Anexos	190
18.1. Normas, Reglamentos y Manuales	190
18.2. Reportes	191
18.2.1. METAR	191
18.2.2. TAF	191
18.2.3. NOTAM	192
18.3. Scripts y Códigos	194
18.3.1. ESP32/BM680	194
18.3.1.1. Blink	194
18.3.1.2. BME680	194
18.3.1.3. BME680 - OLED	196
18.3.1.4. BME680 - OLED - PocketBase	199
18.3.2. Estación Meteorológica	205
18.3.2.1. Blink	205

18.3.2.2. BME280	205
18.3.2.3. VEML6075	206
18.3.2.4. Station	207
18.3.2.5. Station - PocketBase	212
18.3.2.6. Station - PocketBase - SD	220
18.3.3. Servidor	230
18.3.3.1. AccuWeather	230
18.3.3.2. AWC	232
18.3.3.3. Open-Meteo	235
18.3.3.4. OpenWeatherMap	238
18.3.4. Recolección	239
18.3.4.1. RAW Data	239
18.3.5. Procesamiento y análisis de datos	241
18.3.5.1. Reescribir fechas	241
18.3.5.2. Redondeo de fechas	242
18.3.5.3. Interpolación y agrupación	243
19. Listas	245
19.1. Lista de Diagramas	245
19.2. Lista de Esquemas	245
19.3. Lista de Figuras	245
19.4. Lista de Gráficas	247
19.5. Lista de Referencias	247
19.6. Lista de Tablas	248
20. Bibliografía	249

7. Nomenclatura

7.1. Almacenamiento de datos

- **Base de datos.** Conjunto estructurado de información que se almacena y se organiza en un sistema informático. En una base de datos, los datos están organizados en tablas y se pueden acceder, gestionar y actualizar mediante un software de gestión de bases de datos.
- **CSV.** Acrónimo de “valores separados por comas” (Comma-Separated Values, en inglés) y es un formato de archivo utilizado para almacenar datos tabulares. Los datos en un archivo CSV están separados por comas y cada fila representa un registro.
- **JSON.** Acrónimo de “Notación de Objetos JavaScript” (JavaScript Object Notation, en inglés) y es un formato de archivo utilizado para almacenar y transmitir datos. Se utiliza comúnmente en aplicaciones web y móviles para intercambiar datos entre diferentes sistemas.
- **Logs.** Archivos que registran los eventos y actividades que ocurren en un sistema informático. Los logs se utilizan para solucionar problemas y depurar errores en el sistema, así como para monitorizar el rendimiento y la seguridad del sistema.
- **MySQL.** Sistema de gestión de bases de datos relacional (RDBMS) de código abierto. Es uno de los sistemas de bases de datos más populares y se utiliza para gestionar bases de datos de diferentes tamaños y complejidades.
- **PocketBase.** Sistema de gestión de bases de datos simplificado y ejecutado desde un solo archivo, permite a los usuarios almacenar y gestionar datos de una forma fácil y sencilla.
- **RDBMS.** Acrónimo en inglés de “Sistema de Gestión de Bases de Datos Relacionales” (Relational Database Management System). Se refiere a un software diseñado para gestionar bases de datos relacionales, que son aquellas que almacenan y organizan la información en tablas compuestas por filas y columnas, permitiendo la relación y la conexión entre los datos almacenados en distintas tablas.

7.2. Análisis de datos

- **Excel.** Programa de hojas de cálculo desarrollado por Microsoft. Permite a los usuarios crear y gestionar hojas de cálculo con datos numéricos, realizar cálculos y análisis, crear gráficos y tablas dinámicas, y mucho más.
- **Pandas.** Biblioteca de Python que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento. Se utiliza para manipular y analizar datos numéricos y tabulares.

7.3. Autoridades

7.3.1. Internacionales

- **ICAO.** La OACI (Organización de Aviación Civil Internacional, en español) es una agencia especializada de las Naciones Unidas que se dedica a la promoción de la seguridad y la eficiencia de la aviación civil internacional.
- **WMO.** Organización Meteorológica Mundial (World Meteorological Organization, en inglés). Se trata de una agencia especializada de las Naciones Unidas que se dedica a la promoción de la cooperación internacional en el ámbito de la meteorología, la climatología y la hidrología. La WMO se encarga de establecer normas y estándares internacionales para la recopilación, el análisis y la difusión de datos meteorológicos.

7.3.2. Estados Unidos de América

- **AWC.** Acrónimo en inglés de “Aviation Weather Center” (Centro de Clima para Aviación). Es una división de la Administración Nacional Oceánica y Atmosférica (NOAA) en los Estados Unidos que proporciona información meteorológica para la aviación. El AWC emite pronósticos y alertas meteorológicas para ayudar a los pilotos y operadores de aviación a tomar decisiones informadas sobre la seguridad de los vuelos.
- **FAA.** “Federal Aviation Administration” en inglés, es la agencia del gobierno de los Estados Unidos responsable de regular y supervisar la aviación civil en el país. La FAA establece normas y estándares de seguridad para la aviación civil, emite licencias y certificaciones para los pilotos y operadores de aviones, y supervisa el diseño y la construcción de aeronaves y aeropuertos.
- **ICAMS.** Acrónimo en inglés de “Interagency Council for Advancing Meteorological Services”. Es un consejo interinstitucional que se encarga de coordinar y promover la mejora de los servicios meteorológicos en los Estados Unidos.
- **NOAA.** Acrónimo en inglés de la Administración Nacional Oceánica y Atmosférica (National Oceanic and Atmospheric Administration, en inglés). Es una agencia del gobierno de los Estados Unidos que se dedica a la investigación y el monitoreo de la atmósfera, los océanos y el clima. La NOAA es responsable de proporcionar información meteorológica y climática precisa y confiable para el público en general, la industria y el gobierno.
- **NWS.** Acrónimo en inglés del Servicio Meteorológico Nacional (National Weather Service, en inglés), una agencia de la NOAA en los Estados Unidos que se dedica a la recopilación, el análisis y la difusión de información meteorológica para el público en general, la industria y el gobierno. El NWS emite pronósticos y alertas meteorológicas

para ayudar a las personas y las empresas a tomar decisiones informadas sobre la seguridad y la planificación de sus actividades.

7.3.3. México

- **CAPMA.** Acrónimo del Centro de Análisis y Pronóstico Meteorológico Aeronáutico, una agencia gubernamental de México responsable de la recopilación, el análisis y la difusión de información meteorológica para la aviación civil en el país.
- **CONAGUA.** Acrónimo de la Comisión Nacional del Agua, una agencia gubernamental de México que se dedica a la gestión del agua en el país, incluyendo la planificación, la conservación y el aprovechamiento de los recursos hídricos, así como la regulación de la calidad del agua.
- **DGAC.** Acrónimo de la Dirección General de Aeronáutica Civil, una agencia gubernamental de México responsable de la regulación y supervisión de la aviación civil en el país. La DGAC establece normas y estándares de seguridad para la aviación civil, emite licencias y certificaciones para los pilotos y operadores de aeronaves, y supervisa el diseño y la construcción de aeropuertos y aeronaves.
- **SCT.** Acrónimo de la Secretaría de Comunicaciones y Transportes, una agencia gubernamental de México responsable de la regulación y supervisión de los sistemas de comunicaciones y transporte del país, incluyendo la aviación civil, el transporte terrestre y marítimo, y las telecomunicaciones.
- **SMN.** Acrónimo del Servicio Meteorológico Nacional, una agencia gubernamental de México responsable de la recopilación, el análisis y la difusión de información meteorológica en el país. El SMN emite pronósticos y alertas meteorológicas para ayudar a las personas y las empresas a tomar decisiones informadas sobre la seguridad y la planificación de sus actividades.

7.4. Aviación

- **METAR.** Acrónimo de “Meteorological Terminal Aviation Routine Weather Report” (Informe meteorológico rutinario de aviación de terminal). Es un formato estandarizado utilizado para transmitir información meteorológica a los pilotos de aviación. El informe METAR incluye información sobre la visibilidad, la altura de las nubes, la dirección y velocidad del viento, la temperatura, la humedad y la presión barométrica.
- **TAF.** Acrónimo de “Terminal Aerodrome Forecast” (Pronóstico de aeródromo de terminal). Es un informe meteorológico que proporciona información detallada sobre las condiciones meteorológicas esperadas en un aeropuerto en particular durante un período de tiempo específico. El TAF incluye información sobre la visibilidad, la altura de las nubes, la dirección y velocidad del viento, la temperatura, la humedad

y la probabilidad de precipitación. El TAF se emite regularmente para ayudar a los pilotos y operadores de aviación a planificar y ejecutar sus operaciones de manera segura y eficiente.

7.5. Desarrollo

- **Agile.** Enfoque de gestión de proyectos que se enfoca en la flexibilidad, la colaboración y la entrega iterativa e incremental. El enfoque Agile se centra en la adaptabilidad a los cambios y en el trabajo en equipo, con el objetivo de entregar productos de alta calidad de manera más eficiente.
- **Gantt.** Herramienta de planificación de proyectos que utiliza un gráfico de barras para visualizar el progreso de un proyecto a lo largo del tiempo. El gráfico de Gantt muestra las tareas del proyecto en una línea de tiempo, con barras que representan el tiempo dedicado a cada tarea y las dependencias entre ellas.
- **Kanban.** Sistema de gestión de proyectos que se centra en la visualización del flujo de trabajo y la eliminación de los cuellos de botella. El sistema Kanban utiliza tableros visuales para mostrar las tareas del proyecto, con tarjetas que representan cada tarea y que se mueven de una columna a otra a medida que se completan. El objetivo del sistema Kanban es mejorar la eficiencia y la calidad del trabajo mediante la identificación temprana de los problemas y la eliminación de los desperdicios.

7.5.1. Hardware

- **AS3935.** Circuito integrado de detección de tormentas eléctricas utilizado en estaciones meteorológicas y otros dispositivos electrónicos para detectar la presencia de tormentas eléctricas cercanas y emitir alertas.
- **BME280.** Sensor de presión, temperatura y humedad utilizado en estaciones meteorológicas y otros dispositivos electrónicos para medir y monitorear las condiciones ambientales.
- **Data Logger.** Dispositivo electrónico que registra y almacena datos de sensores y otros dispositivos de medición. Los Data Loggers se utilizan en una variedad de aplicaciones, incluyendo la monitorización ambiental, la medición de la calidad del aire, la medición del flujo de agua y la monitorización de la salud de las estructuras.
- **ESP32.** Microcontrolador de bajo costo y alto rendimiento utilizado en una amplia variedad de dispositivos electrónicos, incluyendo estaciones meteorológicas, robots y sistemas de automatización del hogar.
- **MicroMod.** Plataforma modular de electrónica que permite a los diseñadores crear prototipos de dispositivos electrónicos personalizados utilizando una variedad de módulos de hardware interconectables.

- **MicroMod Weather.** Módulo de hardware para la plataforma MicroMod que incluye sensores de temperatura, humedad, presión atmosférica y luz ambiental para la monitorización y el registro de datos meteorológicos.
- **microSD.** Formato de tarjeta de memoria utilizado en dispositivos electrónicos para almacenar y transferir datos.
- **RJ11.** Tipo de conector utilizado para conectar teléfonos, módems y otros dispositivos de comunicación a la red telefónica.
- **SparkFun.** Empresa que diseña y fabrica componentes y kits de electrónica para aficionados y profesionales de la electrónica.
- **VEML6075.** Sensor de radiación ultravioleta utilizado en dispositivos electrónicos para medir la exposición a la radiación UV y proporcionar alertas de exposición excesiva.

7.5.2. Software

- **Arduino.** Plataforma de hardware y software de código abierto utilizada para la creación de proyectos electrónicos interactivos y robótica.
- **Bash.** Intérprete de comandos y shell de Unix utilizado para la programación de scripts de línea de comandos y la automatización de tareas en sistemas operativos basados en Unix.
- **C.** Lenguaje de programación de bajo nivel utilizado para la creación de sistemas operativos, controladores de dispositivos y otras aplicaciones de software de alto rendimiento.
- **Embedido** (Embedded). Sistema informático integrado en un dispositivo electrónico o sistema más grande, utilizado para controlar el funcionamiento del dispositivo o sistema.
- **Git.** Sistema de control de versiones distribuido utilizado para el seguimiento de cambios en el código fuente de software y la colaboración en proyectos de software en equipo.
- **GitHub.** Plataforma de alojamiento de código fuente y colaboración en línea utilizada para el desarrollo de software.
- **GO.** Lenguaje de programación de código abierto utilizado para el desarrollo de aplicaciones de alta escalabilidad y rendimiento.
- **JavaScript.** Lenguaje de programación de alto nivel utilizado para la creación de aplicaciones web y móviles.
- **Microcontrolador.** Chip de computadora programable que integra un procesador, memoria y periféricos de entrada/salida en un solo chip, utilizado para controlar dispositivos electrónicos y sistemas embebidos.

- **NTP.** (Network Time Protocol) es un protocolo de Internet utilizado para la sincronización de relojes de computadora en una red.
- **Python.** Lenguaje de programación de alto nivel utilizado para la creación de aplicaciones web, científicas, educativas y de automatización de tareas.
- **Script.** Archivo de texto que contiene un conjunto de instrucciones o comandos de programación que se ejecutan automáticamente.
- **Shell.** Interfaz de línea de comandos utilizada para interactuar con el sistema operativo y ejecutar programas y scripts en sistemas Unix.
- **UTC.** (Tiempo Universal Coordinado) es un estándar de tiempo global utilizado como referencia para la sincronización de relojes en todo el mundo.

7.6. Meteorología

- **Anemómetro.** Instrumento utilizado para medir la velocidad y dirección del viento.
- **ASOS.** (Automated Surface Observation System) Sistema Automático de Observación del Tiempo a nivel de Superficie, utilizado para la recopilación de datos meteorológicos en tiempo real.
- **AWOS.** (Automated Weather Observation System) Sistema Automático de Observación del Tiempo en General, utilizado para la recopilación de datos meteorológicos en tiempo real.
- **Barómetro.** Instrumento utilizado para medir la presión atmosférica.
- **EMA's.** Estaciones Meteorológicas Automáticas, utilizadas para la recopilación y transmisión de datos meteorológicos.
- **Hygrotermómetro.** Instrumento utilizado para medir la temperatura y la humedad del aire.
- **Pluviómetro.** Instrumento utilizado para medir la cantidad de lluvia que cae en un determinado período de tiempo.
- **Presión atmosférica.** Presión ejercida por el aire en la superficie terrestre.
- **Presión barométrica.** Presión atmosférica medida con un barómetro.
- **Punto de rocío.** Temperatura a la que el vapor de agua comienza a condensarse en el aire.
- **Radiación solar.** Energía emitida por el sol en forma de ondas electromagnéticas.
- **Temperatura.** Medida de la intensidad del calor o frío en un cuerpo.
- **Termómetro.** Instrumento utilizado para medir la temperatura.
- **UV Index.** Índice que indica la intensidad de los rayos UV del sol en un lugar determinado.

- **UVA.** Longitud de onda de la radiación UV que penetra más profundamente en la piel.
- **UVB.** Longitud de onda de la radiación UV responsable del enrojecimiento y quemaduras solares.
- **Veleta.** Instrumento utilizado para medir la dirección del viento.

7.7. Servicios de datos meteorológicos

- **AccuWeather.** Servicio de pronóstico del tiempo y análisis meteorológicos.
- **Open-Meteo.** Plataforma de datos meteorológicos abiertos para el acceso y la investigación.
- **OpenWeatherMap.** Servicio en línea que proporciona pronósticos del tiempo y datos meteorológicos en tiempo real.
- **Weather Underground.** Servicio de pronósticos del tiempo y datos meteorológicos históricos con una red de estaciones meteorológicas en todo el mundo.

8. Introducción

El proyecto se conforma de cuatro partes, la investigación, el prototipo, los resultados y el análisis. Mediante la separación de las partes, se puede tener un mejor control sobre el proyecto y se puede realizar un seguimiento de los avances.

La investigación se centrará en la recopilación de información relativa a los sistemas de recolección de datos meteorológicos, los sensores y los protocolos de comunicación utilizados en la industria. También se realizará una investigación sobre los protocolos de comunicación utilizados en la industria de la aviación para la transmisión de datos meteorológicos, así como de los sistemas automáticos de observación del tiempo (ASOS) y los sistemas automáticos de observación del tiempo en general (AWOS).

La comprensión del uso de APIs, Text Data Servers y demás protocolos de comunicación en la industria de la aviación, así como de los sistemas automáticos de observación del tiempo, será esencial para el desarrollo del sistema de recolección de datos meteorológicos.

El desarrollo del prototipo se centrará en la creación de un sistema de recolección de datos meteorológicos que pueda ser utilizado en la industria de la aviación. El sistema se basará en un microcontrolador con múltiples sensores ambientales integrados, así como un módulo de comunicación inalámbrica. El sistema se diseñará para ser fácilmente instalable y de bajo costo. Además, esperando obtener una ventaja en disponibilidad de datos, el proyecto se prepara para que recopile y envíe datos meteorológicos en tiempo real.

La comparación y selección de tecnologías a emplear en el sistema de recolección de datos meteorológicos se realizará en base a los resultados de la investigación. Cada una de las tecnologías seleccionadas será explicada en detalle, incluyendo su funcionamiento, ventajas y desventajas.

La recolección de datos meteorológicos se realizará mediante sensores y módulos de comunicación inalámbrica. Los sensores se utilizarán para la medición de la radiación ultravioleta, la temperatura, la humedad y la presión atmosférica. Los módulos de comunicación inalámbrica se utilizarán para la transmisión de los datos meteorológicos recopilados a un servidor en línea.

Sistemas de respaldo y redundancia serán implementados para garantizar la disponibilidad de datos. El sistema de respaldo se utilizará para almacenar los datos meteorológicos recopilados en caso de que el servidor en línea no esté disponible. El sistema de redundancia se utilizará para garantizar la disponibilidad de datos en caso de que un sensor falle.

El análisis de resultados se centrará en la evaluación de la precisión del sistema de recolección de datos meteorológicos y la comparación de los resultados con los datos meteorológicos de una estación meteorológica convencional. También se realizará una

evaluación de la precisión de los datos meteorológicos transmitidos por los sistemas automáticos de observación del tiempo y los obtenidos de los servicios de datos meteorológicos en línea.

Se presentará un reporte a partir de los resultados empleando herramientas de visualización de datos, como gráficos de Excel, para facilitar la comprensión de los datos. Estas herramientas nos ayudarán a visualizar y realizar una comparación de precisión entre los datos meteorológicos recopilados y los datos meteorológicos obtenidos de los servicios de datos meteorológicos en línea.

Una conclusión será presentada, donde a partir de los resultados obtenidos se podrá determinar si el sistema de recolección de datos meteorológicos es viable para su uso en la industria de la aviación. Así como el rango de aplicaciones en el que el sistema puede ser utilizado.

9. Problemática

La escasez de datos meteorológicos precisos, confiables y oportunos en la mayor parte del mundo, tienen graves consecuencias para una variedad de sectores e industrias.

Entre las causas, tenemos la limitada cobertura de las existentes estaciones meteorológicas, el alto costo de instalación y mantenimiento.

El impacto de este problema se puede ver en diferentes segmentos de la sociedad, incluidos la agricultura, el transporte y el turismo. Ejemplo, los datos meteorológicos inexactos pueden dar como resultado cosechas deficientes, cancelaciones de vuelos y otras interrupciones que desencadenan en graves consecuencias económicas y sociales.

La industria aeroespacial es uno de los sectores más afectados por la falta de datos meteorológicos precisos y actualizados. Los datos meteorológicos son esenciales para la planificación y ejecución de vuelos seguros, la falta de datos precisos puede dar lugar a retrasos, desvíos y cancelaciones de vuelos.

10. Hipótesis

La implementación de estaciones meteorológicas automatizadas y asequibles en zonas con poca cobertura de datos meteorológicos, mejora el nivel de precisión y actualización de los datos meteorológicos a un reducido costo.

11. Justificación

El desarrollo de un sistema de información meteorológica automatizado se justifica a partir de la necesidad de información meteorológica precisa y actualizada en la industria de la aeroespacial. Los actuales sistemas nacionales se basan en observadores humanos para informar sobre las condiciones climáticas, lo que puede dar lugar a errores de interpretación y a retrasos en la transmisión de los datos.

Trabajamos en innovar en el campo de la recolección de datos meteorológicos, ya que, a diferencia de las estaciones meteorológicas tradicionales, que suelen ser costosas y complejas de operar, se ofrece una solución económica y fácil de operar que se puede implementar en casi cualquier lugar.

Las estaciones meteorológicas automatizadas se han convertido en parte de la columna vertebral de la observación del clima en los Estados Unidos y Canadá, y cada vez son más comunes en todo el mundo debido a su eficiencia y ahorro de costos.

(Wikipedia, 2023)

Los beneficios son numerosos, y van desde una mayor seguridad a nivel personal y empresarial, hasta una mejor planificación y gestión aérea.

Mediante el desarrollo de una estación METAR asequible y automatizada, el proyecto pretende proporcionar información fiable y en tiempo real a pilotos, controladores de tráfico aéreo, así como a otros profesionales de la aviación. Especialmente en México, donde la instalación de estaciones METAR tradicionales suele ser costosa y desafiante debido a las barreras geográficas y climáticas. La actual falta de estaciones METAR automatizadas en México presenta un problema importante para la industria de la aviación, ya que los datos meteorológicos son esenciales para la planificación y la ejecución de vuelos seguros.

Al proporcionar una alternativa rentable a las estaciones METAR tradicionales, un sistema automatizado puede aumentar significativamente la disponibilidad de datos meteorológicos en México, lo que a su vez puede mejorar la seguridad y la eficiencia de la aviación. El mismo sistema puede ser utilizado para recopilar datos meteorológicos en otras industrias, como la agricultura, la minería y el turismo. Estos datos pueden ser utilizados para mejorar la planificación y la gestión de los recursos, mediante el uso de sistemas de análisis de datos. El desarrollo de un sistema METAR automatizado asequible es un paso crucial en la mejora de la seguridad y la eficiencia de la aviación en México.

12. Objetivo

12.1. General

Desarrollar un sistema de información meteorológica automatizado, que permita la recopilación de datos meteorológicos en tiempo real, con un costo reducido y una instalación sencilla.

12.2. Específico

- Investigar y analizar las características de las estaciones METAR tradicionales y automatizadas.
- Implementar un sistema de información meteorológica automatizado.
- Analizar los resultados de la implementación del sistema, mediante la comparación de los datos recopilados con los datos de una estación METAR tradicional y los datos de múltiples servicios de pronóstico del tiempo.
- Presentar el desarrollo y los resultados de la investigación en un informe técnico.

13. Marco Teórico

13.1. Antecedentes

La industria de la aviación depende en gran medida de la información meteorológica precisa y oportuna para garantizar la seguridad de los pasajeros y la tripulación. La disponibilidad de datos meteorológicos actuales y precisos ayuda a los pilotos a tomar decisiones informadas sobre rutas de vuelo, consumo de combustible y enfoques de aterrizaje. La OACI recomienda que todos los aeropuertos tengan un servicio meteorológico que proporcione información meteorológica de acuerdo a los estándares establecidos. (WMO, 2021)

La forma tradicional de proporcionar información meteorológica es a través de la observación manual y la elaboración de informes por parte de meteorólogos o personal capacitado. Sin embargo, este proceso es tardado y costoso, especialmente para los aeropuertos pequeños y con recursos limitados. Además, estos informes pueden tener errores debido a las malas prácticas al momento de la toma o transmisión de los datos. Los sistemas automatizados, por otro lado, pueden reducir significativamente el costo y el tiempo requerido para recolectar y reportar los datos meteorológicos. En los últimos años el crecimiento en el interés en el desarrollo de procesos automatizados, y herramientas tales como microcontroladores que cada vez son más accesibles por su precio y disponibilidad, han permitido el desarrollo de sistemas automatizados de bajo costo, pudiendo implementar estos sistemas a aeródromos pequeños y remotos. (Barani, 2018a)

Las estaciones meteorológicas automatizadas son estaciones meteorológicas que se instalan en lugares remotos y que se conectan a una red de comunicaciones para transmitir datos meteorológicos en tiempo real a un centro de datos. Estas estaciones se utilizan para recopilar datos meteorológicos en lugares donde no hay estaciones meteorológicas convencionales, como aeropuertos, carreteras, puentes, túneles, etc. Estas estaciones son una alternativa rentable a las estaciones meteorológicas tradicionales, ya que pueden reducir significativamente los costos de instalación y mantenimiento. (Barani, 2018b)

13.2. Bases Teóricas

13.2.1. Aeroespacial

La industria aeroespacial es un sector de la economía que se dedica al diseño, la fabricación y la operación de aeronaves y sistemas espaciales. La industria aeroespacial incluye la aviación civil, la aviación militar, la exploración espacial y la investigación

científica. La industria aeroespacial es una de las industrias más importantes del mundo, ya que desempeña un papel crucial en la economía, la seguridad y la exploración del espacio. (Crocker, 2010)

13.2.2. Aeronáutica

La aeronáutica es una disciplina científica y tecnológica que se ocupa del estudio, diseño, construcción y operación de aeronaves, así como de los principios y fenómenos que rigen su vuelo. Esta disciplina interdisciplinaria combina la física, la matemática, la ingeniería y la informática para desarrollar y mejorar la navegación aérea. La aeronáutica abarca áreas como la aerodinámica, la propulsión, la estructura de las aeronaves, la navegación y el control de vuelo. Su objetivo principal es garantizar la seguridad y eficiencia de los vuelos, así como impulsar la innovación y el avance tecnológico en la industria aeroespacial. (Cuesta, 2003)

13.2.3. Aviación

La aviación es el sector que se encarga de la operación de aeronaves, incluidos los vuelos comerciales, los vuelos privados y los vuelos militares. La aviación es una industria de gran importancia, ya que permite el transporte de personas y mercancías a gran velocidad y a largas distancias. (Crocker, 2010)

13.2.4. Aviónica

La aviónica es una rama de la ingeniería aeroespacial que se ocupa del diseño, la fabricación y el mantenimiento de los sistemas electrónicos y de comunicación de las aeronaves. La aviónica incluye sistemas como los de navegación, comunicación, control de vuelo, radar, sistemas de entretenimiento y sistemas de seguridad. La aviónica es una parte fundamental de la aviación, ya que garantiza la seguridad y eficiencia de los vuelos. (Cuesta, 2003)

13.2.5. Electrónica

La electrónica es una rama de la física y la ingeniería que se ocupa del estudio y la aplicación de los dispositivos y circuitos electrónicos. La electrónica abarca áreas como la teoría de circuitos, la electrónica digital, la electrónica analógica, la electrónica de potencia y la microelectrónica. La electrónica es una disciplina fundamental en la industria aeroespacial, ya que se utiliza en el diseño y la fabricación de sistemas de aviónica, sistemas de control de vuelo, sistemas de comunicación y sistemas de navegación. (Crocker, 2010)

13.2.6. Meteorología

La meteorología es la ciencia que estudia los fenómenos atmosféricos, como el clima, el tiempo y los eventos meteorológicos extremos, y sus efectos en la Tierra y en los seres vivos. Esta disciplina se basa en la física, la química y la biología para comprender y predecir los cambios en la atmósfera y sus interacciones con otros sistemas terrestres. La meteorología es fundamental para la planificación y la toma de decisiones en diversas áreas, como la agricultura, la aviación, la navegación marítima, la gestión de desastres y la investigación científica. Los meteorólogos utilizan una variedad de instrumentos y técnicas, como el análisis de datos, la observación satelital, los modelos numéricos y la inteligencia artificial, para recopilar, analizar y pronosticar el clima y el tiempo. El estudio de la meteorología contribuye a mejorar la comprensión de los fenómenos naturales y a desarrollar estrategias para mitigar los impactos de los eventos meteorológicos adversos en la sociedad y el medio ambiente. (Crocker, 2010)

13.3. Bases Conceptuales

13.3.1. Desarrollo de Software

El desarrollo de software es el proceso de creación de programas informáticos. Es un proceso iterativo que involucra la planificación, el diseño, la implementación, la prueba y la documentación de los programas. Este proceso es complejo y requiere una gran cantidad de conocimientos y habilidades técnicas.

13.3.2. Estaciones METAR

Las estaciones METAR son instalaciones meteorológicas utilizadas en aeropuertos para recopilar datos sobre las condiciones atmosféricas. Estas estaciones son empleadas por pilotos, controladores de tráfico aéreo y otros profesionales de la aviación. (Barani, 2018b) Las estaciones METAR convencionales emplean sensores para obtener mediciones meteorológicas, mientras que las estaciones METAR automatizadas utilizan sensores y un microcontrolador para recopilar y procesar los datos.

Las estaciones METAR convencionales suelen estar compuestas por una variedad de sensores que miden parámetros como la temperatura, la humedad, la presión atmosférica, la velocidad y dirección del viento, la visibilidad y las condiciones de precipitación. Estos sensores están ubicados en diferentes partes del aeropuerto y se comunican con una estación central que recopila y procesa los datos.

Por otro lado, las estaciones METAR automatizadas utilizan tecnología de microcontroladores para recopilar y procesar los datos meteorológicos. Estos dispositivos están programados para leer los datos de los sensores y enviarlos a una estación central

a través de una red de comunicaciones. La estación central recibe los datos y los procesa para generar informes meteorológicos en tiempo real.

En resumen, las estaciones METAR son instalaciones meteorológicas utilizadas en aeropuertos para recopilar datos sobre las condiciones atmosféricas. Las estaciones convencionales emplean sensores para obtener mediciones meteorológicas, mientras que las automatizadas utilizan sensores y microcontroladores para recopilar y procesar los datos de manera eficiente y rentable.

13.3.3. Microcontroladores

Un microcontrolador es un circuito integrado que contiene un procesador, memoria y periféricos. (Boxall, 2021) Los microcontroladores son dispositivos de bajo costo, que pueden ser programados para realizar una gran variedad de tareas. Los microcontroladores son ampliamente utilizados en la industria, ya que son pequeños, baratos y fáciles de programar.

13.3.3.1. Arduino

Arduino es un sistema de desarrollo de hardware y software, que permite a los usuarios crear sistemas electrónicos interactivos. Arduino es una plataforma de desarrollo de hardware libre y software, que se utiliza para crear dispositivos electrónicos interactivos. Arduino se basa en un microcontrolador, que es un circuito integrado que contiene un procesador, memoria y periféricos. El microcontrolador es el cerebro del sistema, y es el encargado de procesar los datos y ejecutar las instrucciones del programa. El microcontrolador se conecta a los periféricos, que son los dispositivos que se utilizan para interactuar con el mundo exterior. Los periféricos incluyen sensores, actuadores, pantallas, etc. Arduino se conecta a un ordenador, que es utilizado para programar el microcontrolador y para cargar el programa en el microcontrolador. El programa se carga en el microcontrolador, y el microcontrolador ejecuta las instrucciones del programa. Arduino es un sistema de desarrollo de hardware y software, que permite a los usuarios crear sistemas electrónicos interactivos. (Huang & Runberg, 2017)

13.3.3.2. ESP32

ESP32 es un microcontrolador de bajo costo, que se utiliza para crear sistemas electrónicos interactivos. El ESP32 es compatible con Arduino, por lo que puede ser programado utilizando el entorno de desarrollo de Arduino. (Boxall, 2021)

13.3.4. Sensores

Un sensor es un dispositivo que se utiliza para medir una variable física, como la temperatura, la humedad, la presión, etc. (Huang & Runberg, 2017) Los sensores son ampliamente utilizados en la industria, ya que permiten a los sistemas automatizados

recopilar datos de forma automática. Los sensores son dispositivos de bajo costo, que pueden ser conectados a un microcontrolador para recopilar datos.

13.3.5. Sistema de Bases de Datos

Un sistema de bases de datos es un conjunto de programas que se utilizan para crear, administrar y consultar una base de datos. (Hyde, 2020a) Los sistemas de bases de datos son ampliamente utilizados en la industria, ya que permiten a los usuarios almacenar, administrar y consultar grandes cantidades de datos. Los sistemas de bases de datos son dispositivos de bajo costo, que pueden ser conectados a un ordenador para almacenar, administrar y consultar datos.

13.3.6. Sistema de Comunicaciones

Un sistema de comunicaciones es un conjunto de dispositivos que se utilizan para transmitir datos entre dos o más dispositivos. Los sistemas de comunicaciones son ampliamente utilizados en la industria, ya que permiten a los usuarios transmitir datos entre dispositivos. (Hyde, 2020a)

14. Descripción de Símbolos y Convenciones

Para una mejor lectura del siguiente documento, se han definido los siguientes símbolos y convenciones:

14.1. Notas

Información adicional o aclaraciones sobre un tema específico.

Nota. Este es un ejemplo de una nota.

14.2. Citas

Cita una fuente o referencia en el texto.

*Este es un ejemplo de una cita.
(Autor, 2024)*

La primera cita del documento ya fue presentada en la Sección 11.

14.3. Referencias

Proporcionar información sobre una fuente o referencia citada en el texto.

De modo que la referencia anterior puede ser reescrita de la siguiente forma:

La primera cita del documento ya fue presentada en la **Sección 11**.

14.4. Definiciones

Definición de un término específico.

Ejemplo. *Este es un ejemplo de una definición.*

14.5. Código

El código se utiliza para mostrar ejemplos de código fuente, comandos de terminal, scripts de shell, etc.

.py

```
def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        fib_sequence = [0, 1]
        while len(fib_sequence) < n:
            next_num = fib_sequence[-1] + fib_sequence[-2]
            fib_sequence.append(next_num)
        return fib_sequence

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

def find_prime_fibonacci(n):
    prime_fibonacci = []
    fib_sequence = fibonacci(n)
    for num in fib_sequence:
        if is_prime(num):
            prime_fibonacci.append(num)
    return prime_fibonacci

n = int(input("Enter the number of Fibonacci numbers to generate: "))
prime_fibonacci = find_prime_fibonacci(n)
print("Prime Fibonacci numbers:")
print(prime_fibonacci)
```

15. Desarrollo

La primera parte se dedicará a la investigación de los temas relacionados con el proyecto, como la meteorología, la aviación, el desarrollo de software, los microcontroladores, los sensores, los sistemas de bases de datos y los sistemas de comunicaciones. La segunda parte se dedicará al prototipo del proyecto, que consiste en la creación de un prototipo funcional del proyecto. La tercera parte se dedicará a la realización de pruebas y a la obtención de resultados. La cuarta parte se dedicará al análisis de los resultados obtenidos, y a la elaboración de conclusiones.

15.1. Investigación

15.1.1. Observación Meteorológica

Es la medición cuantitativa y cualitativa de una o más variables meteorológicas y se remiten inmediatamente a un centro recolector de datos mediante mensajes codificados, por la vía de comunicación más rápida disponible. (México, 2015)

15.1.2. Reportes Meteorológicos. METAR, TAF y NOTAM

METAR. Reporte meteorológico estándar que se emite en aeropuertos y otras estaciones meteorológicas. (Crocker, 2010)

Los reportes son emitidos comúnmente a través de estaciones METAR. Dichos reportes pueden ser consultados en la página de la Administración Federal de Aviación (FAA) en Estados Unidos (AWC, 2013a). También se pueden consultar a nivel nacional en la página del Centro de Análisis y Pronósticos Meteorológicos Aeronáuticos (CAPMA) (CAPMA, s. f.).

Tomando en cuenta la “Guía de Referencia” (Weather.gov, 2008) y a partir de la **Referencia 1** se describe cada componente del METAR en la **Tabla 1**

Referencia 1. Ejemplo de reporte METAR.

```
METAR KABC 121755Z AUTO 21016G24KT 180V240 1SM R11/P6000FT -RA BR
BKN015 OVC025 06/04 A2990 RMK A02 PK WND 20032/25 WSHFT 1715 VIS 3/4V1
1/2 VIS 3/4 RWY11 RAB07 CIG 013V017 CIG 017 RWY11 PRESFR SLP125 P0003
60009 T00640036 10066 21012 58033 TSNO $
```

Se anexa un breve ejemplo de la decodificación de un reporte METAR en la **Sección 18.2.1**

Tabla 1. Características de un reporte METAR.

Característica	Descripción	Ejemplo
Tipo de reporte	METAR: informe horario (programado); SPECI: informe especial (no programado).	METAR
Identificador de estación	Cuatro caracteres alfabéticos; Identificador de ubicación de la OACI.	KABC
Fecha y hora de la observación	Todas las fechas y horas en UTC usando un reloj de 24 horas; fecha de dos dígitos y hora de cuatro dígitos; siempre se adjunta con Z para indicar UTC.	121755Z
Modificador de reporte	AUTO para informe totalmente automatizado, sin intervención humana; eliminado cuando el observador lo registra.	AUTO
Dirección y velocidad del viento	Dirección en decenas de grados desde el norte verdadero (primeros tres dígitos); dos dígitos siguientes: velocidad en nudos enteros; según sea necesario Ráfagas (carácter) seguidas de la velocidad máxima observada; siempre adjunto con KT para indicar nudos; 0000KT para calma; si la dirección varía en 60° o más, se notifica un grupo de dirección de viento variable.	21016G24KT
Visibilidad	Visibilidad predominante en millas terrestres y fracciones (espacio entre millas enteras y fracciones); siempre adjunto con SM para indicar millas terrestres; valores menores a 1/4 informados como M1/4 .	1SM

Característica	Descripción	Ejemplo
Visibilidad de pista	Valor RVR de 10 minutos en cientos de pies; informado si la visibilidad predominante es \leq una milla o $RVR \leq 6000$ pies; siempre se agrega FT para indicar pies; valor con el prefijo M o P para indicar que el valor es más bajo o más alto que el valor RVR notificable.	R11/6000FT
Fenómeno	RA : precipitación líquida que no congela; SN : precipitación congelada distinta del granizo; UP : precipitación de tipo desconocido; intensidad antepuesta a la precipitación: ligera - , moderada (sin signo), fuerte - ; FG : niebla; FZFG : niebla helada (temperatura inferior a 0°C); BR : niebla; HZ : neblina; SQ : turbonada; máximo de tres grupos informados; aumentado por observador: FC (nube en embudo/tornado/tromba marina); TS (tormenta); GR (granizo); GS (pequeño granizo; menor de 1/4 de pulgada); FZRA (intensidad; lluvia helada); VA (ceniza volcánica).	-RA BR
Condición de cielo	Cantidad y altura de nubes: CLR (no se detectan nubes por debajo de los 12000 pies); FEW (pocos); SCT (disperso); BKN (roto); OVC (nublado); seguido de altura de 3 dígitos en cientos de pies; o visibilidad vertical (VV) seguida de altura para techo indefinido.	BKN015 OVC025
Temperatura y punto de rocío	Cada uno se informa en grados Celsius enteros utilizando dos dígitos; los valores están separados por un solidus; los valores bajo cero tienen el prefijo M (menos).	06/04

Característica	Descripción	Ejemplo
Altímetro	Altímetro siempre con el prefijo A que indica pulgadas de mercurio; reportado usando cuatro dígitos: decenas, unidades, décimas y centésimas.	A2990
Identificador de observaciones	RMK	RMK
Actividad de tornados	incluir TORNADO, FUNNEL CLOUD o WATERSPOUT, hora de inicio/fin, ubicación, movimiento.	TORNADO B25 N MOV E
Tipo de estación automatizada		A02
Viento máximo	PK WND dddff(f)/(hh)mm; dirección en decenas de grados, velocidad en nudos enteros y tiempo.	PK WND 20032/25
Cambio de dirección del viento	WSHFT dddff(f)/(hh)mm; dirección en decenas de grados, velocidad en nudos enteros y tiempo.	WSHFT 1715
Visibilidad de torre o superficie	TWR VIS vvvv: visibilidad informada por el personal de la torre, SFC VIS vvvv: visibilidad informada por ASOS.	TWR VIS 2 SFC VIS 2
Visibilidad variable prevalente	VIS vvvv Vn v/v: visibilidad variable prevalente.	VIS 3/4V1 1/2
Visibilidad en 2da ubicación	VIS v/v LOC: visibilidad en 2da ubicación.	VIS 3/4 RWY11
Rayos	freq LTG loc: Frecuencia y ubicación de rayos.	FRQ LTG NE

Característica	Descripción	Ejemplo
Inicio y final de precipitación y tormentas	w'w' B (hh)mmE(hh)mm; TSB (hh)mm E (hh)mm	RAB07
Virga	Precipitación que cae de las nubes y se evapora antes de tocar el suelo.	VIRGA
Altura variable de nubes	CIG hhnchn V hxhxhx; informado si el techo en el cuerpo del informe es < 3000 pies y variable.	CIG 013V017
Altura de las nubes en 2da ubicación	CIG hhnchn LOC: altura de las nubes en 2da ubicación.	CIG 017 RWY11
Presión ascendiendo o descendiendo rápidamente	PRESRR ascendiendo rápidamente, PRESFR descendiendo rápidamente.	PRESFR
Presión al nivel del mar	SLP ppp: presión al nivel del mar en decenas de hPa.	SLP125
Precipitación de la última hora	P rrr; en 0,01 pulgadas desde el último METAR; un rastro es P0000.	P0003
Precipitación de las últimas 3 o 6 horas	HRRRR; cantidad de precipitación en 0,01 pulgadas durante las últimas 6 horas notificadas en observaciones a las 00, 06, 12 y 18 UTC y durante las últimas 3 horas en observaciones a las 03, 09, 15 y 21 UTC; un rastro es 60000.	60009
Precipitación de las últimas 24 horas	7RRRR; cantidad de precipitación en 0,01 pulgadas durante las últimas 24 horas; un rastro es 70000.	70003

Característica	Descripción	Ejemplo
Temperatura y punto de rocío por hora	Horas y decimas de grados Celsius	T00640036
Máxima temperatura en las últimas 6 horas	1 hora y decimas de grados Celsius	10066
Mínima temperatura en las últimas 6 horas	2 hora y decimas de grados Celsius	21012
Máxima temperatura en las últimas 24 horas	4 hora y decimas de grados Celsius	400461006
Tendencia de presión	5appp: el carácter a (8) indica que la presión está aumentando.	58033
Estado del sensor	RVRN0: Falta RVR; PWINO: información del identificador de precipitación no disponible; PN0: cantidad de precipitación no disponible; FZRANO: información sobre lluvia helada no disponible; TSNO: información de tormentas no disponible; VISNO (LOC): visibilidad en la ubicación secundaria no disponible, por ejemplo, VISNO RWY06; CHINO (LOC): (indicador de altura de nubes) condición del cielo en la ubicación secundaria no disponible	TSNO
Mantenimiento	\$: Se requiere mantenimiento del sistema.	\$

TAF. Pronóstico de condiciones meteorológicas para un aeropuerto o zona de navegación. (Crocker, 2010)

Referencia 2. Ejemplo de reporte TAF.

```
TAF MML0 020430Z 0206/0306 22010KT P6SM BKN200
FM021900 24015KT P6SM BKN200
FM030400 24008KT 6SM HZ BKN200
```

Este tipo de pronóstico se encuentra fuera de nuestras capacidades, por lo que la información se reservará a la emitida por la Administración Federal de Aviación (FAA) en Estados Unidos a nivel internacional y a nivel nacional a la emitida por el Centro de Análisis y Pronósticos Meteorológicos Aeronáuticos (CAPMA).

Se anexa un breve ejemplo de la decodificación de un reporte TAF en la [Sección 18.2.2](#)

NOTAM. Notificación de aeronavegabilidad. (Crocker, 2010)

Referencia 3. Ejemplo de reporte NOTAM.

```
A1234/06 NOTAMR A1212/06
Q)EGTT/QMXLC/IV/NB0/A/000/999/5129N00028W005
A)EGLL
B)0609050500
C)0704300500
E)DUE WIP TWY B SOUTH CLSD BTN 'F' AND 'R'. TWY 'R' CLSD BTN 'A' AND
'B' AND DIVERTED VIA NEW GREEN CL AND BLUE EDGE LGT. CTN ADZ
```

Este tipo de notificación se encuentra fuera de nuestras capacidades, por lo que la información se reservará a la emitida por la Administración Federal de Aviación (FAA) en Estados Unidos a nivel internacional y a nivel nacional a la emitida por el Centro de Análisis y Pronósticos Meteorológicos Aeronáuticos (CAPMA).

Se anexa un breve ejemplo de la decodificación de un reporte NOTAM en la [Sección 18.2.3](#)

15.1.3. Estaciones Meteorológicas

Instalación que cuenta con el instrumental, equipos y sistemas destinados a medir y registrar regularmente variables meteorológicas. (México, 2015)

El servicio que brindan las estaciones meteorológicas es el de proporcionar información meteorológica de forma continua y en tiempo real. Esta información es de gran utilidad para la toma de decisiones en diversas áreas, como la agricultura, la aviación, la navegación, la industria, la pesca, la minería, la construcción, la seguridad pública, entre otras.

15.1.4. Estaciones Meteorológicas en Aeropuertos

Las estaciones meteorológicas son instaladas en aeropuertos con el objetivo de proporcionar información meteorológica de forma continua y en tiempo real.

15.1.5. Estaciones Meteorológicas Automáticas

Automated Surface/Weather Observing Systems (ASOS/AWOS). El programa de Sistemas Automatizados de Observación de Superficie (ASOS) es un esfuerzo conjunto del Servicio Meteorológico Nacional (NWS), la Administración Federal de Aviación (FAA) y el Departamento de Defensa (DOD). Actualmente hay más de 900 sitios de ASOS en los Estados Unidos. Estos sistemas automatizados recopilan observaciones de forma continua, las 24 horas del día. Los datos de ASOS se archivan en la base de datos Global Surface Hourly. (NCEI, 2021)

Las unidades del Sistema Automatizado de Observación del Tiempo (AWOS) son operadas y controladas por la Administración Federal de Aviación. Estos sistemas se encuentran entre las estaciones meteorológicas automatizadas más antiguas y son anteriores a ASOS. Por lo general, informan a intervalos de 20 minutos y, a diferencia de ASOS, no informan observaciones especiales para condiciones climáticas que cambian rápidamente.

EMA's. Sistema autónomo y automático formado por un conjunto de sensores de medición, dispositivos eléctricos, electrónicos y mecánicos, montados sobre una estructura de soporte, en donde son distribuidos, orientados y conectados al Sistema de Adquisición, Procesamiento y Almacenamiento de Datos (SAPAD) de la estación, con el objetivo de realizar la medición y registro de las variables meteorológicas que imperan en el lugar, y transmitir los datos obtenidos a la oficina central, en donde serán utilizados y almacenados a una base de datos. (SMN, s. f.)

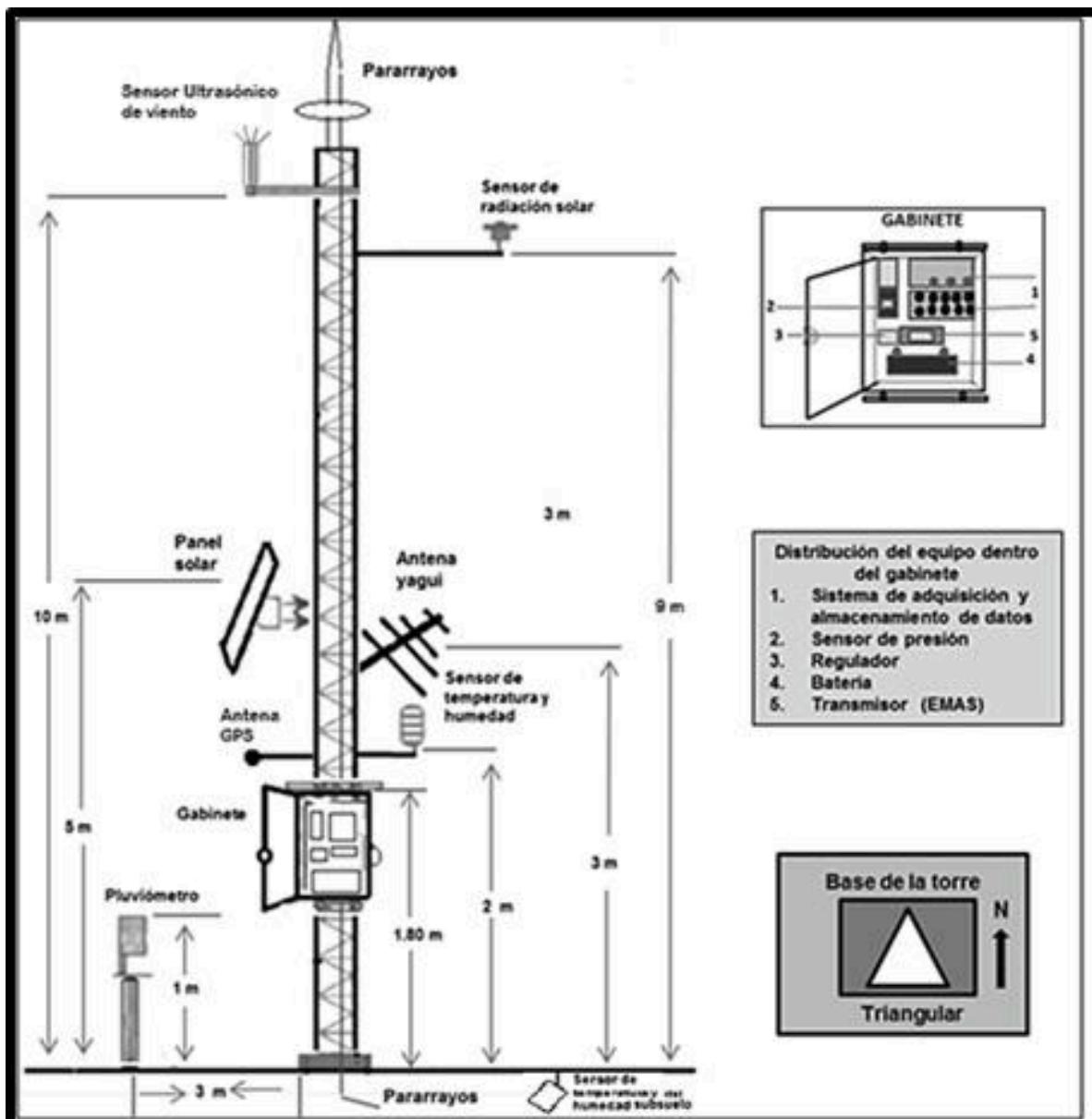
Sensores básicos que integran una EMA:

- Velocidad del viento.
- Dirección del viento.

- Presión atmosférica.
- Temperatura.
- Humedad relativa.
- Radiación solar.
- Precipitación.

La Figura 1 muestra el bosquejo comúnmente utilizado para representar una EMA.

Figura 1. Bosquejo de una EMA (SMN).



15.1.6. Sensores de Condiciones Ambientales

Para evitar la búsqueda exhausta de todos y cada uno de los sensores que se pueden utilizar en una estación meteorológica, se ha decidido solo mencionar los sensores que se utilizarán en el proyecto, los cuales se encuentran en el mercado y que se pueden adquirir fácilmente.

La lista de los sensores requeridos es la siguiente:

- Sensor de Temperatura.
- Sensor de Presión.
- Sensor de Humedad.
- Sensor de Radiación Solar.
- Sensor de Dirección del Viento.
- Sensor de Velocidad del Viento.
- Sensor de Precipitación.

Ideando la forma de integrar todos estos sensores en una sola estación meteorológica, se llegó a la conclusión de que la mejor opción es utilizar un microcontrolador que cuente con un amplio número de puertos de entrada y salida, así como con capacidades WiFi y Bluetooth. El microcontrolador que se utilizará en el proyecto es el ESP32, se hablará más de él en la Sección 15.2.

Para el sensor de temperatura tenemos dos opciones, trabajar con un sensor de temperatura y humedad (DHT22), o trabajar con un sensor de temperatura, presión y humedad (BME280). El sensor de temperatura y humedad (DHT22) es un sensor de bajo costo, pero no es muy preciso, por lo que se decidió utilizar el sensor de temperatura, presión y humedad (BME280). Además de que previamente ya se ha trabajado con el DHT22 y el BME680, por lo que se tiene experiencia con estos sensores.

Por ahora cubrimos temperatura, presión, y humedad con un solo sensor, el BME280. Esto nos permitirá iniciar pruebas previas, puesto que ya se cuenta con un BME680 y un ESP32.

Para la radiación solar hemos optado por el VEML6075 ya que es un sensor de bajo costo y de fácil integración. Además de que es un sensor que no requiere de mucho tiempo de calibración, puesto que nos brinda los valores de UVA, UVB y UV Index directamente.

Para la dirección del viento ocupamos una veleta, para la velocidad del viento ocupamos un anemómetro, y para la precipitación ocupamos un pluviómetro. Después de realizar una búsqueda exhaustiva en el mercado, se llegó a la conclusión de que la mejor opción sería operar un kit con estos tres sensores, el SparkFun Weather Meter Kit. Este kit cuenta con un anemómetro, una veleta y un pluviómetro, además de que se puede integrar con un microcontrolador que se encarga de procesar los datos de los sensores y enviarlos a

un servidor web. Este kit es de bajo costo y de fácil integración, por lo que se decidió utilizarlo.

15.1.7. Sistemas de Bases de Datos

Al momento de trabajar con bases de datos, se debe tener en cuenta que existen diferentes tipos de bases de datos, cada una con sus características y ventajas. En este proyecto se utilizará una base de datos relacional, puesto que es la más utilizada y la que mejor se adapta a las necesidades del proyecto.

En un inicio se pensó en trabajar con una base de datos MySQL o MariaDB, pero al momento de realizar la búsqueda de un servidor que cumpliera con las características necesarias, se llegó a la conclusión de que la mejor opción sería utilizar una base de datos en la nube, puesto que es una base de datos que se encuentra en un servidor remoto, por lo que no es necesario tener un servidor local para almacenar los datos.

Entre las opciones más simples de operar en un servidor remoto, que sean del tipo relacional, contamos con firebase, appwrite y PocketBase.

firebase es una base de datos en la nube, que se puede utilizar de forma gratuita, está bastante documentada, pero al existir en el entorno de Google Cloud Computing (GCP), se debe tener en cuenta que el uso del servicio podría contribuir a costos no previstos.

appwrite es una nueva alternativa a firebase, que se puede utilizar de forma gratuita, está bastante completa, pero al momento no cuenta con una alternativa fácil de trabajar si se desea utilizar en un entorno local.

PocketBase es una base de datos más simplificada, pero con todas las características necesarias para el proyecto, además de que se puede utilizar de forma gratuita, y se puede utilizar en un entorno local y remoto sin mayor dificultad.

15.1.8. Sistemas de Comunicaciones

El protocolo seleccionado para la comunicación entre el microcontrolador y el servidor bien pudo haber sido operado a través de MQTT, pero al momento de analizar las ventajas y desventajas de cada protocolo, se llegó a la conclusión de que el protocolo HTTP es el más adecuado para el proyecto, debido a la experiencia que se tiene en la implementación de este protocolo.

MQTT. Protocolo de red de máquina a máquina que proporciona una forma ligera de enviar mensajes entre dispositivos. Está diseñado para conexiones de alta latencia y baja fiabilidad, y es ideal para los dispositivos de Internet de las cosas (IoT).

HTTPS. (*Hypertext Transfer Protocol Secure*) es una versión segura del protocolo *HTTP*, que se utiliza para la comunicación entre servidores y clientes. Este protocolo se utiliza para la comunicación entre el servidor y el cliente web.

Al momento de realización se cuenta con un servidor gratuito en Oracle (Oracle, s. f.), el cual cuenta con las siguientes características:

- 1 vCPU.
- 1GB de RAM.
- 50GB de almacenamiento.

Se encuentra operando con el sistema operativo Ubuntu Server 22.04 LTS (Ubuntu, s. f.), y se encuentra configurado con los siguientes servicios:

- Cloudflared (Cloudflare, s. f.).
- Sub-dominio redirigido a la IP del servidor.

Para mantenernos dentro de tema, se evitará extenuar hablando sobre los detalles del servidor.

15.1.9. Servicios de Datos Meteorológicos

Durante la búsqueda de servicios de datos meteorológicos, se encontraron varios servicios que se pueden utilizar de forma gratuita, pero al momento de analizar las características de cada uno, se llegó a la conclusión de que los servicios que se utilizarán en el proyecto son los siguientes:

AccuWeather. (AccuWeather, Inc.) es un proveedor de datos meteorológicos, que cuenta con una API que permite obtener datos meteorológicos de forma gratuita, pero con un límite de 50 peticiones por día. Esta API cuenta con una documentación bastante completa, y es de fácil integración. (AccuWeather, s. f.)

API. (*Application Programming Interface*) es un conjunto de reglas y especificaciones que se utilizan para definir cómo se comunican dos aplicaciones entre sí. Las API permiten a las aplicaciones compartir datos y funcionalidades entre sí.

AWC. El Aviation Weather Center, cuenta con un servicio de datos meteorológicos, que permite obtener datos meteorológicos de forma gratuita, no hay un límite esclarecido, pero se deja a criterio del usuario, ya que este debe tomar en cuenta que si realiza

demasiadas peticiones será bloqueado. El sistema usado para la obtención de datos es a través del Text-Data-Server (TDS), el cual es un servicio que permite obtener datos meteorológicos en formato de texto, este es el más complicado puesto que no cuenta con una documentación completa, y es necesario realizar pruebas para obtener los datos deseados. (AWC, 2013b)

Open-Meteo. (Open-Meteo) es un proveedor de datos meteorológicos, que cuenta con una API que permite obtener datos meteorológicos de forma gratuita, pero con un límite de 10,000 peticiones por día, si ese límite es excedido se puede solicitar un aumento de límite. Esta API cuenta con una documentación bastante completa, y es de fácil integración. (Open Meteo, s. f.)

OpenWeatherMap. (OpenWeatherMap) es un proveedor de datos meteorológicos, que cuenta con una API que permite obtener datos meteorológicos de forma gratuita, pero con un límite de 1,000 peticiones por día y 60 peticiones por minuto. Esta API cuenta con una documentación bastante completa, y es de fácil integración. (OpenWeather, s. f.)

Se intento agregar a los servicios empleados el ofrecido por la CONAGUA a través del SMN (Servicio Meteorológico Nacional), pero al momento de realizar las pruebas de su API se obtuvo que las estaciones más cercanas a la ubicación prueba llevaban de 3 a 6 años sin actualizar sus datos meteorológicos, por lo que se descartó su uso. (Méjico, s. f.)

La integración y pruebas se verán más a fondo en la [Sección 15.2.4](#)

15.2. Prototipo

Para llevar a cabo un desarrollo más íntegro del proyecto, se decidió dividir el proyecto en 2 prototipos, uno inicial para familiarizarnos con el entorno de desarrollo y el hardware, y otro prototipo final, que será el que se utilizará para las pruebas en campo.

El prototipo inicial se basa en el uso de un microcontrolador ESP32, el cual cuenta con un sensor BME680, el cual es un sensor de temperatura, humedad, presión y calidad del aire, el cual cuenta con una interfaz I2C, por lo que se puede conectar directamente al microcontrolador, sin necesidad de utilizar un conversor analógico digital (ADC).

I2C. (*Inter-Integrated Circuit*) Es un protocolo de comunicación de dos cables que permite la comunicación entre varios dispositivos a través de dos cables.

Para iniciar debemos de implementar un sistema de bases de datos a través del cual llevar a cabo las pruebas de comunicación y registro de datos. Recordando la [Sección 15.1.7](#) donde hablamos acerca de los sistemas de bases de datos, parte en la cual decidimos que la base de datos que se utilizaría en el proyecto sería PocketBase, debido a que es una base de datos más simplificada, pero con todas las características necesarias para el proyecto, además de que se puede utilizar de forma gratuita en un entorno local y remoto sin mayor dificultad.

La forma más fácil de comenzar es descargar la aplicación PocketBase mínima preconstruida (PocketBase, s. f.)

15.2.1. PocketBase

Al momento de trabajar con PocketBase, tenemos que considerar la advertencia que nos recibe la primera página de su documentación:

Nota. Se debe tomar en cuenta que PocketBase aún está en desarrollo activo y no se garantiza la compatibilidad total con versiones anteriores antes de llegar a v1.0.0.

Las pruebas y el proyecto se empezarán a realizar en la versión v0.14.0, disponible en <https://github.com/pocketbase/pocketbase/releases/tag/v0.14.0>

Aun así, se planea continuar actualizando la versión en uso mientras esta no suponga problemáticas, e igualmente se fijarán las pruebas en el uso de una sola versión en específico, la cual será indicada posteriormente.

15.2.1.1. Entorno Local

Para el entorno local se trabajó en una máquina personal con el sistema operativo Fedora 37, la cual cuenta con una versión de PocketBase preconstruida, por lo que no fue necesario realizar la construcción de la aplicación, solo fue necesario descargarla y ejecutarla. Este mismo sistema operativo posteriormente será actualizado a la versión 38, pero tal proceso no se realizará durante el periodo de pruebas de la aplicación por lo que no afectará el desarrollo del proyecto.

Sistema Operativo. (SO) es el software que controla la computadora y permite la comunicación con el hardware. Se encarga de gestionar los recursos del sistema, como la memoria, el disco duro y el procesador.

Para trabajar con este tipo de aplicaciones binarias es necesario realizar su ejecución desde la terminal de comandos, a continuación, se describe el proceso de ejecución de PocketBase en el entorno local desde la terminal.

Nota. Por la constante actualización y desarrollo de la aplicación PocketBase, las capturas puede que correspondan a más de una versión, empezando desde la versión v0.14.0

Se crea un directorio para trabajar con PocketBase, y nos movemos a este directorio:

```
.sh  
mkdir pocketbase  
cd pocketbase
```

Nota. El código mostrado en el bloque anterior corresponde a la creación de un directorio y a la navegación a este, por lo que no es necesario ejecutarlo en la terminal. Al igual que todos los demás comandos tipo shell (.sh) que se presenten en este documento.

Se descarga la aplicación preconstruida de la versión v0.14.0:

```
.sh  
wget https://github.com/pocketbase/pocketbase/releases/download/v0.15.2/pocketbase_0.15.2_linux_amd64.zip
```

Descomprimimos el archivo descargado:

```
.sh  
unzip pocketbase_0.15.2_linux_amd64.zip
```

Para iniciar PocketBase, se debe de ejecutar el siguiente comando desde el directorio donde se encuentra el archivo ejecutable:

```
.sh  
. ./pocketbase serve
```

Con una correcta ejecución debe obtenerse el siguiente resultado en terminal:

```
.sh  
2023/04/21 14:52:41 Server started at http://127.0.0.1:8090  
→ REST API: http://127.0.0.1:8090/api/  
→ Admin UI: http://127.0.0.1:8090/_/
```

Por su compilación binaria no es necesario realizar la instalación de la aplicación, por lo que se puede ejecutar desde cualquier directorio, y es compatible para ejecución en múltiples sistemas operativos, además de estar completamente integrada para trabajarse en sistemas operativos linux.

Linux. es un sistema operativo de código abierto que se ejecuta en una amplia variedad de dispositivos, desde servidores y computadoras de escritorio hasta teléfonos móviles y tabletas.

Una vez ejecutado el comando, se iniciará el servidor de PocketBase, el cual por defecto se ejecuta en el puerto 8090, por lo que para acceder a la aplicación se debe de ingresar a la siguiente dirección desde un navegador web, http://127.0.0.1:8090/_/ :

Figura 2. Admin UI, http://127.0.0.1:8090/_/ (PocketBase).

The screenshot shows the PocketBase Admin UI login interface. At the top center is the PocketBase logo. Below it, a message reads "Create your first admin account in order to continue". There are three input fields: "Email *", "Password *", and "Password confirm *". A note below the password field says "Minimum 10 characters.". At the bottom is a large black button labeled "Create and login →".

Se llenan los campos de Email, Password y Password confirm con los datos que se nos solicitan, y damos click en el botón Create and login →

Se recibe una pantalla de inicio como la siguiente.

Figura 3. Pantalla inicial (PocketBase).

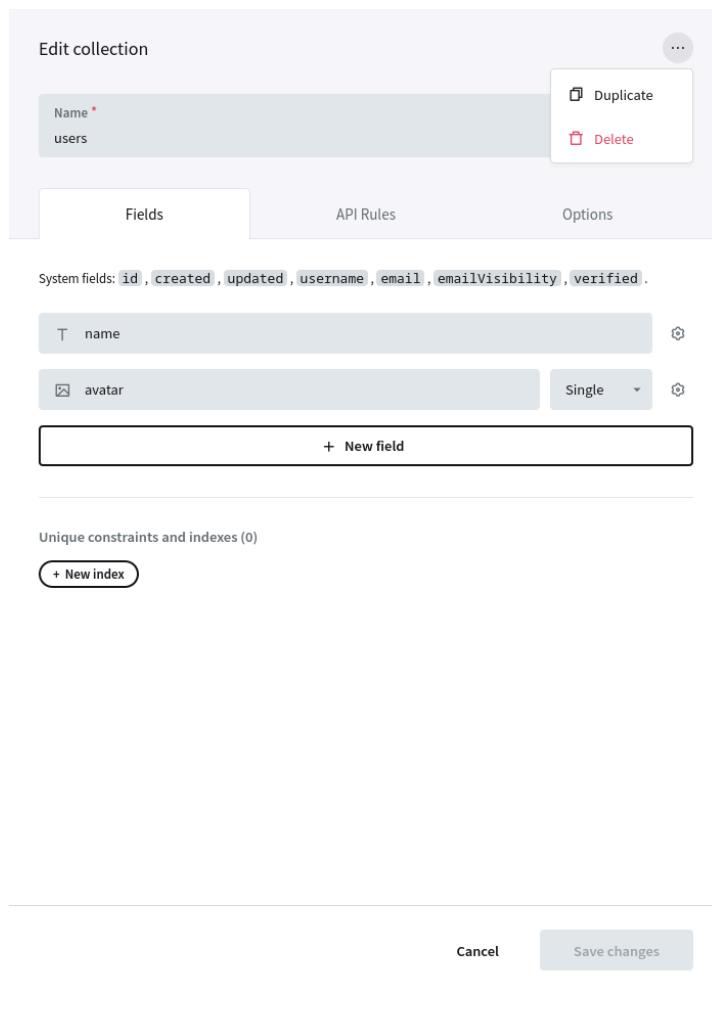
The screenshot shows the PocketBase Admin UI main dashboard. On the left is a sidebar with a collection list, where "users" is selected. The main area shows a table for the "users" collection with columns: id, username, email, name, avatar, created, updated, and an ellipsis. A search bar at the top allows filtering by "created > 2022-01-01". A message at the bottom of the table says "No records found.". At the bottom right of the main area is a button "+ New record". The top right of the screen has buttons for "API Preview" and "+ New record". The bottom right corner of the screen shows "Docs | PocketBase v15.2".

La interfaz para utilizar PocketBase es bastante intuitiva, por lo que no se ahondará en su uso, pero se recomienda revisar la documentación oficial de PocketBase para conocer más acerca de su uso.

Se puede observar que ya existe una “colección” o tabla como se le conoce en otros sistemas de bases de datos, la cual se llama `users`, la cual es la tabla que se crea por defecto al iniciar PocketBase, y la cual se utiliza para el registro de usuarios, por lo que no es necesario crear una tabla para el registro de usuarios. Pero en nuestro caso es innecesario su uso por lo que se procede a eliminarla.

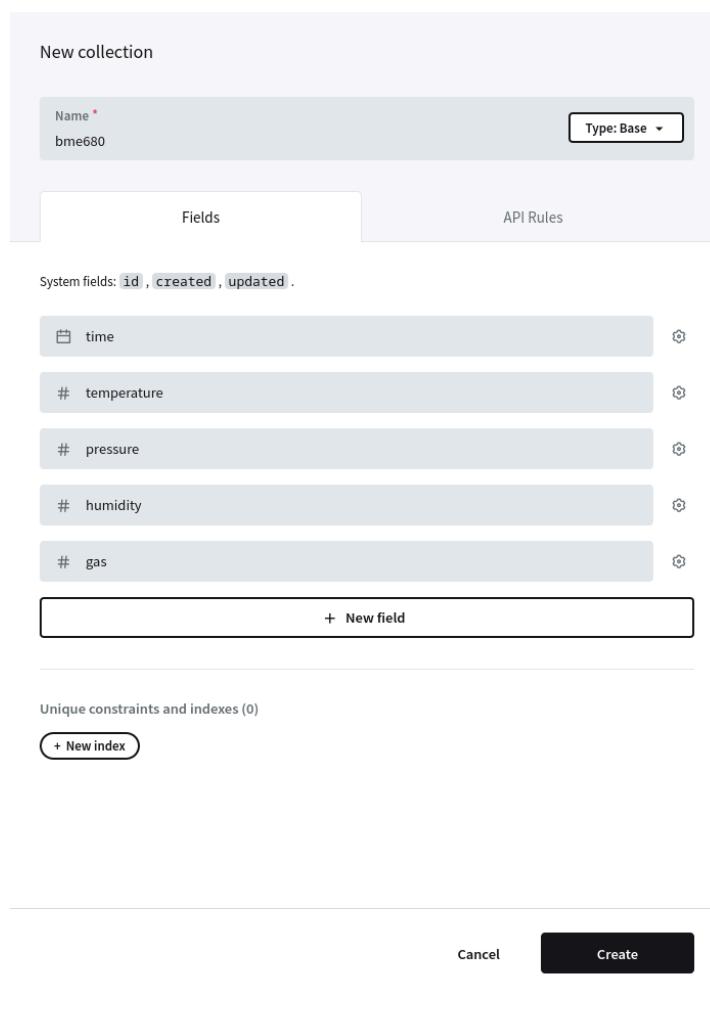
Para eliminar la tabla `users` se debe de dar click en el botón `Delete` que se encuentra en la parte superior derecha de la tabla, y posteriormente confirmar la eliminación de la tabla.

Figura 4. Eliminado de tabla `users` (PocketBase).



Paso siguiente se creará una de las tablas que se utilizarán en el proyecto, la cual se llamará `bme680`, para ello se da click en el botón `New collection` que se encuentra en la parte superior derecha de la pantalla.

Figura 5. Creado de tabla `bme680` (*PocketBase*).

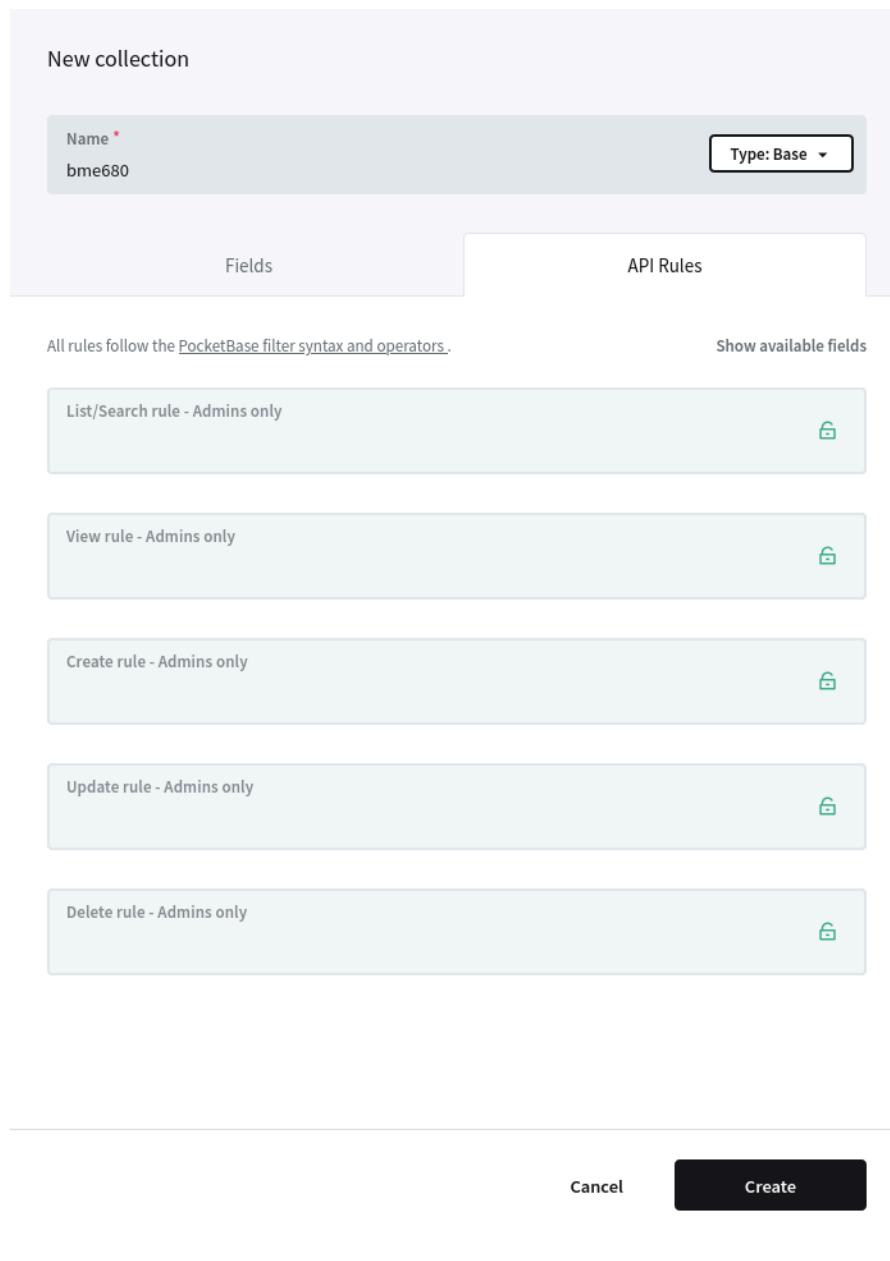


Se llena el campo `Name` con el nombre de la tabla, en este caso `bme680`, y con el fin de acelerar el proceso se llenan también los fields necesarios para la creación de la tabla, en este caso se crearán los siguientes fields:

- `time` de tipo `timestamp`.
- `temperature` de tipo `number`.
- `pressure` de tipo `number`.
- `humidity` de tipo `number`.
- `gas` de tipo `number`.

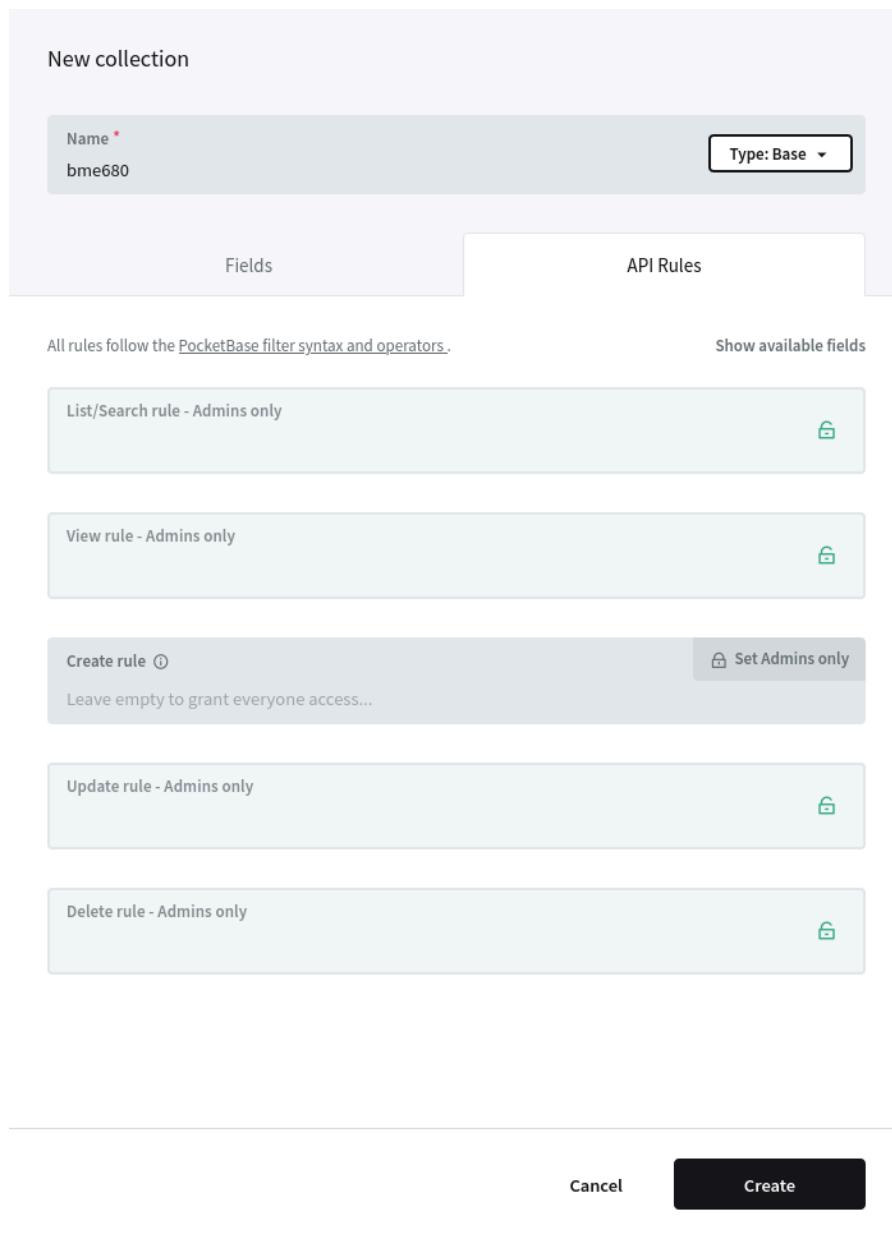
Otro de los aspectos importantes al realizar la creación de la tabla es el campo `API Rules`, el cual se encuentra como un submenú al lado de la parte de `Fields`, en este apartado encontramos las reglas para los diferentes tipos de comunicación que se pueden llevar a través de la API de PocketBase.

Figura 6. Reglas de la API (*PocketBase*).



Se puede observar que por defecto todas las reglas se encuentran bloqueadas para acceso solo a administradores, para simplificar el proceso y no requerir de realizar un sistema de autenticación se pasara a desbloquear la regla de `Create rule` ya que es la única que se utilizará en esta primera parte de recolección de información.

Figura 7. Habilitación de regla `Create` (*PocketBase*).



Finalmente se da click en el botón `Create` para crear la colección.

Nota. Es importante remarcar que a partir de este momento se referirá a lo que comúnmente conocemos como “**tablas**” como “**colección**” con el fin de adaptarnos al entorno de desarrollo de PocketBase.

Las siguientes colecciones a crear son autodescriptivas, se espera que el lector pueda crearlas sin mayor problema, pero en caso de dudas se puede consultar la documentación oficial de PocketBase.

Se añadirán las siguientes colecciones con sus respectivos campos:

- accuweather
 - ▶ time de tipo timestamp .
 - ▶ temperature de tipo number .
 - ▶ realFeelTemperature de tipo number .
 - ▶ realFeelTemperatureShade de tipo number .
 - ▶ relativeHumidity de tipo number .
 - ▶ indoorRelativeHumidity de tipo number .
 - ▶ dewPoint de tipo number .
 - ▶ windDirection de tipo number .
 - ▶ windSpeed de tipo number .
 - ▶ uvIndex de tipo number .
 - ▶ pressure de tipo number .
 - ▶ apparentTemperature de tipo number .
 - ▶ precipitation de tipo number .
- awc
 - ▶ raw_text de tipo string .
 - ▶ station_id de tipo string .
 - ▶ observation_time de tipo timestamp .
 - ▶ temp_c de tipo number .
 - ▶ dewpoint_c de tipo number .
 - ▶ wind_dir_degrees de tipo number .
 - ▶ wind_speed_kt de tipo number .
 - ▶ altim_in_hg de tipo number .
 - ▶ corrected de tipo boolean .
 - ▶ precip_in de tipo number .
 - ▶ metar_type de tipo string .
- open_meteo
 - ▶ time de tipo timestamp .

- ▶ temperature_2m de tipo number .
 - ▶ relative_humidity_2m de tipo number .
 - ▶ dewpoint_2m de tipo number .
 - ▶ apparent_temperature de tipo number .
 - ▶ pressure_msl de tipo number .
 - ▶ surface_pressure de tipo number .
 - ▶ windspeed_10m de tipo number .
 - ▶ windspeed_80m de tipo number .
 - ▶ windspeed_120m de tipo number .
 - ▶ windspeed_180m de tipo number .
 - ▶ winddirection_10m de tipo number .
 - ▶ winddirection_80m de tipo number .
 - ▶ winddirection_120m de tipo number .
 - ▶ winddirection_180m de tipo number .
 - ▶ temperature_80m de tipo number .
 - ▶ temperature_120m de tipo number .
 - ▶ temperature_180m de tipo number .
 - ▶ uv_index de tipo number .
-
- openweathermap
 - ▶ time de tipo timestamp .
 - ▶ temperature de tipo number .
 - ▶ feels_like de tipo number .
 - ▶ pressure de tipo number .
 - ▶ humidity de tipo number .
 - ▶ wind_speed de tipo number .
 - ▶ wind_direction de tipo number .

 - station
 - ▶ time de tipo timestamp .
 - ▶ temperature de tipo number .
 - ▶ humidity de tipo number .
 - ▶ dewpoint de tipo number .
 - ▶ pressure de tipo number .
 - ▶ rainFall de tipo number .
 - ▶ windSpeed de tipo number .
 - ▶ windDirection de tipo number .
 - ▶ uva de tipo number .
 - ▶ uvb de tipo number .
 - ▶ uvindex de tipo number .

Cada una de ellas se debe desbloquear la regla de `Create rule` para poder realizar la inserción de datos a través de la API.

15.2.1.2. Entorno Remoto

Es importante tener el servicio de la base de datos, y el de recolección de datos, funcionando las 24 horas del día, por lo que se requiere de un entorno remoto, en este caso se utilizará un Servidor Virtual Privado (VPS) con el sistema operativo Ubuntu 22.04, el cual se puede adquirir en cualquier proveedor de servicios en la nube, en este caso se utilizarán los brindados por el Oracle Free Tier. (Oracle, s. f.)

Nota. La documentación necesaria para llevar desarrollar la infraestructura del servidor se encuentra en <https://www.oracle.com/mx/cloud/free/>, omitimos la documentación de la creación del servidor, la implementación para la comunicación con los servidores y base de datos, ya que es un tema muy extenso, y no es el objetivo de este documento.

15.2.2. ESP32 - BME680

Para iniciar con el proceso de desarrollo del prototipo se debe tener en cuenta que se requiere de un entorno de desarrollo para el ESP32, en este caso se utilizará el IDE de Arduino, el cual se puede descargar desde su página oficial <https://www.arduino.cc/en/software>

No se repasará el proceso de instalación del IDE de Arduino, pero se recomienda seguir la guía oficial de instalación <https://www.arduino.cc/en/Guide>. Una vez instalado el IDE de Arduino se debe instalar el soporte para el ESP32, para ello se debe seguir la adecuada guía de instalación correspondiente a cada sistema operativo, la cual se puede encontrar en la documentación oficial de Arduino <https://www.arduino.cc/en/Guide/Cores>

Nota. El siguiente proceso utiliza una serie de herramientas de las cuales se espera se tenga una noción, para el desarrollo de código en arduino se estarán siguiendo los principios de los siguientes enunciados.

... Arduino was not originally created for makers, engineers, or hobbyist but for design students in Icrea, Italy, as a learning platform to help them make their projects function without the needing years of electrical engineering courses or tons of math and theory. It was designed to shorten the time from “nothing” to “awesome!” -that is, from idea to physical product- for nontechnical people.

(Huang & Runberg, 2017)

The Arduino project has grown exponentially since its introduction in 2005. It's now a thriving industry, supported by a community of people united with the common bond of creating something new.

(Boxall, 2021)

No se detallará el proceso de instalación ni de drivers ni de las herramientas empleadas, sino que se centrara en desarrollar el producto “as it is”, es decir, se asume que se tiene el conocimiento y las herramientas necesarias para el desarrollo del prototipo.

Nota. El código utilizado se encuentra en los anexos, pero también recibirá actualizaciones en el siguiente repositorio:

https://github.com/0sares10/BME680-ESP32_PocketBase

15.2.2.1. Guías, Manuales y Tutoriales

A continuación, se listan los enlaces a las guías, manuales y tutoriales que se utilizaron para el desarrollo del prototipo.

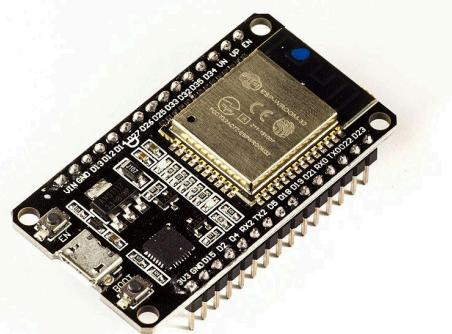
- <https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/> (Last Minute Engineers, s. f.)
- <https://microcontrollerslab.com/bme680-esp32-arduino-oled-display/> (Microcontrollers Lab, s. f.)

15.2.2.2. Componentes

A continuación, se listan los componentes que se utilizaron para el desarrollo del prototipo.

ESP32. Para la comunicación e integración de los sensores se utilizó un microcontrolador ESP32, el cual se puede observar en la siguiente Figura 8.

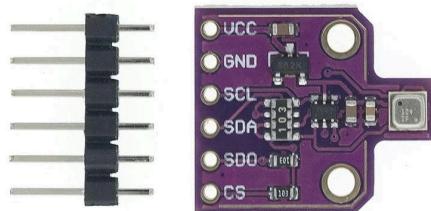
Figura 8. ESP32 Dev Board (*Espressif*).



El ESP32 es un microcontrolador de bajo costo, que cuenta con un amplio número de puertos de entrada y salida, así como con capacidades WiFi y Bluetooth, lo que lo hace ideal para el desarrollo de prototipos de IoT.

BME680. Para la medición de los parámetros ambientales se utilizó un sensor BME680, el cual se puede observar en la siguiente Figura 9 imagen de UNIT ELECTRONICS, <https://uelectronics.com>.

Figura 9. BME680 (UNIT ELECTRONICS).



Pantalla OLED 128 x 64 Pixel 2 Color. Para la visualización de los datos se utilizó una pantalla OLED de 128 x 64 pixeles a 2 colores, imagen en la **Figura 10** propiedad de MEGATRONICA, <https://megatronica.cc>.

Figura 10. Pantalla OLED 128 x 64 Pixel Bi-Color (*MEGATRONICA*).



Los demás componentes son los comunes al momento de desarrollar un prototipo de estas características, protoboard, jumpers y fuente de alimentación.

15.2.2.3. Funcionalidades

A continuación, se listan las funcionalidades que se implementaron en el prototipo.

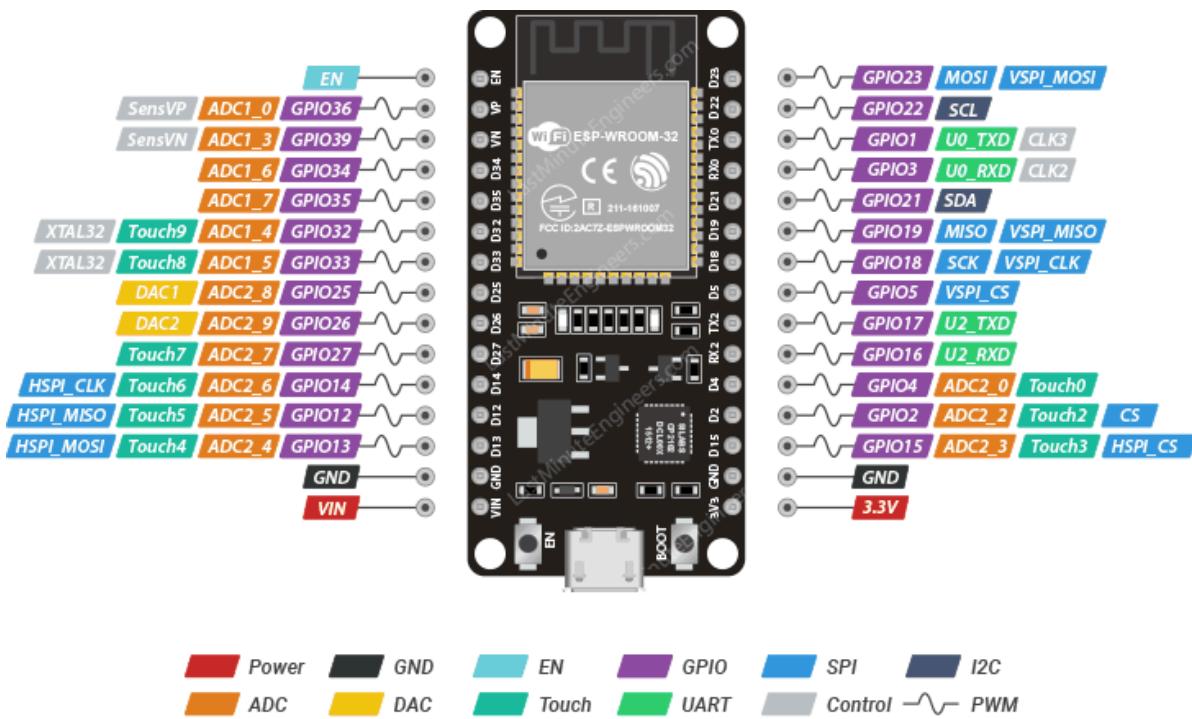
- Medición de parámetros ambientales. El prototipo es capaz de medir los parámetros ambientales de temperatura, humedad, presión y calidad del aire.
- Visualización de datos. El prototipo es capaz de visualizar los datos medidos en una pantalla OLED.
- Almacenamiento de datos. El prototipo es capaz de almacenar los datos medidos en una base de datos en la nube.

15.2.2.4. Pinouts

A continuación, se listan los pinouts utilizados para el desarrollo del prototipo.

ESP32. Para la comunicación e integración de los sensores se utilizó un microcontrolador ESP32, el cual se puede observar en el siguiente **Esquemático 1.**

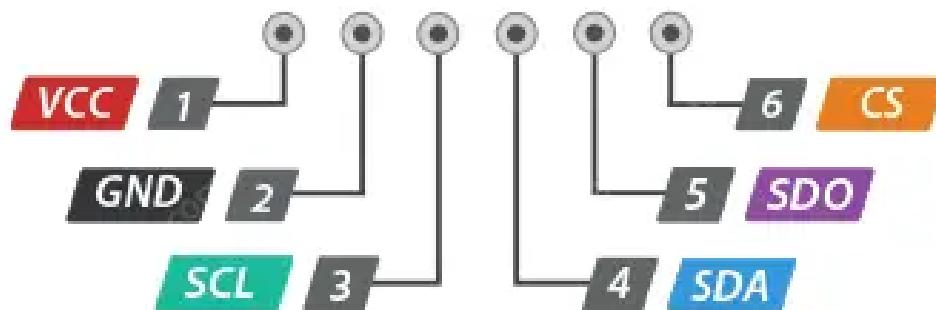
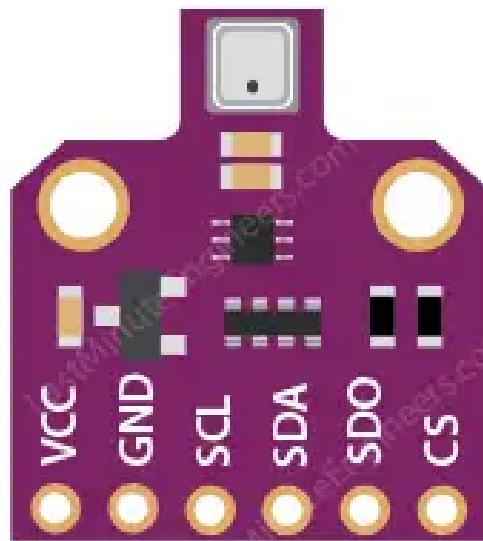
Esquemático 1. Pinout ESP32 (*Last Minute Engineers*).



Pinout. Es un diagrama que muestra la disposición de los pines de un dispositivo, como un microcontrolador, un circuito integrado o un conector. Los pines se numeran y se muestran en un diagrama para facilitar la conexión de los dispositivos.

BME680. Para la medición de los parámetros ambientales se utilizó un sensor BME680, el cual se puede observar en el siguiente **Esquemático 2.**

Esquemático 2. Pinout BME680 (*Last Minute Engineers*).



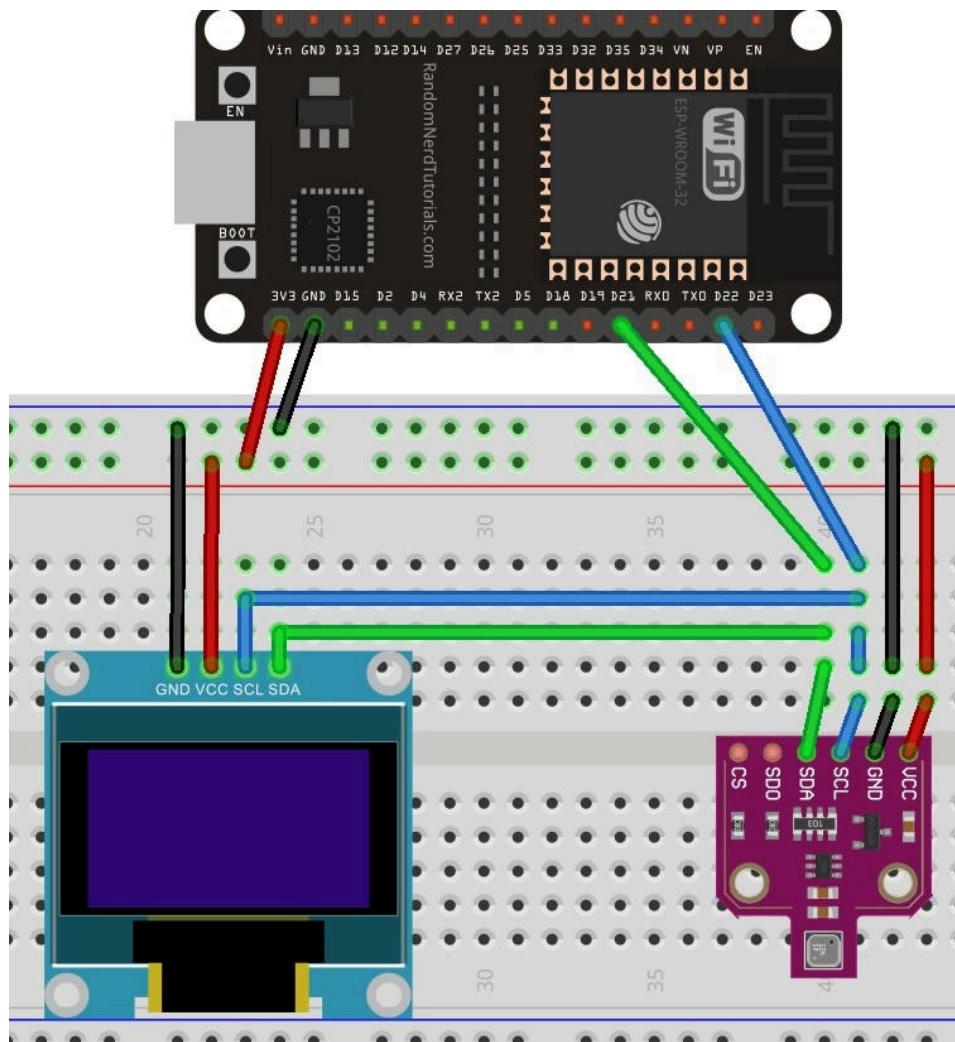
BME680 Module / Pinout

 *Last Minute
ENGINEERS.com*

15.2.2.5. Esquemático

Para el desarrollo del esquemático se utilizó el brindado por Microcontrollers Lab, <https://microcontrollerslab.com/bme680-esp32-arduino-oled-display/>, el cual se puede observar en el siguiente **Esquemático 3.**

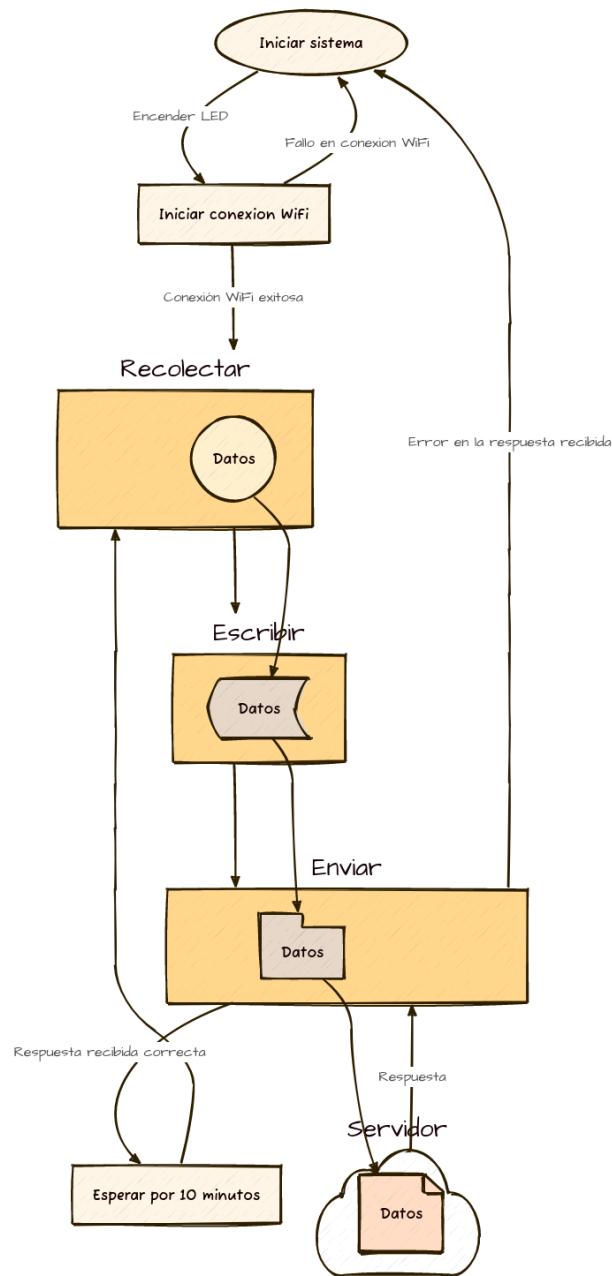
Esquemático 3. Unión de componentes (*Microcontrollers Lab*).



15.2.2.6. Diagrama de Flujo

A continuación, se muestra el siguiente diagrama de flujo, el cual se puede observar en el siguiente **Diagrama 1** hecho con el lenguaje de programación D2 (D2 Lang, s. f.) .

Diagrama 1. Flujo del prototipo del ESP32.



15.2.2.7. Código

Para el desarrollo del código y esperando se siga un avance lineal en la lectura del documento, A continuación, se enuncian las bases para el desarrollo del código.

In the engineering fields, engineers approach a specified problem by following a prescribed set of rules, building custom solution from a combination of predetermined solutions.

(Hyde, 2020b)

The software engineer's job is to create the best possible product given conflicting requirements by making appropriate compromises in a system's design.

(Hyde, 2020b)

Se inicia el desarrollo con un simple código para el parpadeo del LED integrado, con el fin de verificar que el entorno de desarrollo este correctamente configurado.

Nota. Los siguientes códigos serán detallados paso a paso, pero el código en su totalidad puede ser consultado en la [Sección 18.3.1](#)

Blink/Blink.ino [Sección 18.3.1.1](#)

Se define el pin del LED integrado como salida.

```
void setup() {  
    pinMode(LED_BUILTIN, OUTPUT);  
}
```

.ino

.ino Es la extensión de los archivos de código fuente de Arduino.

Se crea un ciclo infinito en el cual se enciende el LED integrado por un segundo y se apaga por un segundo.

```
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

.ino

Verificamos y cargamos el código en el ESP32, cuando el LED integrado parpadee, se puede continuar con el desarrollo.

Una vez verificado el entorno de desarrollo, se procede a la integración del sensor BME680, para ello se utiliza la librería BME680 de Adafruit.

BME680/BME680.ino Sección 18.3.1.2

Incluimos las librerías necesarias para el desarrollo del código.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>
```

.ino

El uso de la librería BME680 de Adafruit requiere de la definición de los pines de comunicación del sensor, en este caso se utilizan los pines SDA y SCL.

```
Adafruit_BME680 bme; // I2C
```

.ino

Se requiere asignar el valor de la presión atmosférica a nivel del mar en milibares, para ello se utiliza el valor de la presión atmosférica estándar al nivel del mar con un valor de 1013.25mb .

```
#define SEALEVELPRESSURE_HPA (1013.25)
```

.ino

Setup es la función que se ejecuta una sola vez al inicio del programa, en ella se inicializan los pines de comunicación del sensor y se verifica que el sensor esté conectado.

Iniciamos una comunicación serial con una velocidad de 115200 baudios, con el objetivo de poder visualizar los datos obtenidos del sensor en el monitor serial.

Baudios. Es una unidad de medida que se utiliza para medir la velocidad de transmisión de datos en un sistema de comunicación.

```
void setup() {  
    Serial.begin(115200);  
    while (!Serial);  
    Serial.println(F("BME680 async test"));  
    ...  
}
```

.ino

Se verifica que el sensor esté conectado, en caso de no estar conectado se imprime un mensaje de error en el monitor serial.

```
void setup() {  
    ...  
    if (!bme.begin()) {  
        Serial.println(F("Could not find a valid BME680 sensor, check  
wiring!"));  
        while(1);  
    }  
    ...  
}
```

.ino

En la función anterior el uso de `while(1);` detiene la ejecución del programa, por lo que obliga a verificar la conexión antes de permitir que se continúe con la ejecución del programa.

Si la ejecución logra continuar es necesario inicializar el sensor, para ello se realiza la configuración de la resolución de la temperatura, humedad, presión y gas.

```
.ino  
  
void setup() {  
    ...  
    // Set up oversampling and filter initialization  
    bme.setTemperatureOversampling(BME680_OS_8X);  
    bme.setHumidityOversampling(BME680_OS_2X);  
    bme.setPressureOversampling(BME680_OS_4X);  
    bme.setIIRFilterSize(BME680_FILTER_SIZE_3);  
    bme.setGasHeater(320, 150); // 320*C for 150 msFlujo  
}
```

Loop es la función que se ejecuta de forma cíclica, en ella se obtienen los datos del sensor y se imprimen en el monitor serial.

Monitor Serial. Es una herramienta que permite visualizar los datos obtenidos del sensor en tiempo real. Se puede acceder a esta herramienta desde el menú de herramientas del IDE de Arduino.

Iniciamos el ciclo con la toma de datos del sensor.

```
.ino  
  
void loop() {  
    // Tell BME680 to begin measurement.  
    unsigned long endTime = bme.beginReading();  
    if (endTime == 0) {  
        Serial.println(F("Failed to begin reading :("));  
        return;  
    }  
    Serial.print(F("Reading started at "));  
    Serial.print(millis());  
    Serial.print(F(" and will finish at "));  
    Serial.println(endTime);  
    ...  
}
```

La previa implementación con el uso de las funciones brindadas por la librería del sensor BME680 de Adafruit, permite que la lectura de los datos del sensor se realice de forma asíncrona, por lo que es necesario esperar a que la lectura de los datos del sensor finalice.

.ino

```
void loop() {
  ...
  Serial.println(F("You can do other work during BME680 measurement."));
  delay(50); // This represents parallel work.
  // There's no need to delay() until millis() >= endTime:
  Flujobme.endReading()
  // takes care of that. It's okay for parallel work to take longer than
  // BME680's measurement time.
  ...
}
```

Desde un inicio podemos observar que uno de los principales problemas con la toma de datos del sensor es el tiempo de espera, que no es instantáneo y mucho menos constante, por lo cual resultará en una desviación de los 10 minutos cada vez que una lectura sea tomada por el sensor.

.ino

```
void loop() {
  ...
  // Obtain measurement results from BME680. Note that this operation
  isn't
  // instantaneous even if milli() >= endTime due to I2C/SPI latency.
  if (!bme.endReading()) {
    Serial.println(F("Failed to complete reading :("));
    return;
  }
  Serial.print(F("Reading completed at "));
  Serial.println(millis());
  ...
}
```

Una vez obtenidos los resultados podemos imprimirlas a través del monitor serial.

.ino

```
void loop() {
  ...
  Serial.print(F("Temperature = "));
  Serial.print(bme.temperature);
  Serial.println(F(" *C"));
```

```
Serial.print(F("Pressure = "));  
Serial.print(bme.pressure / 100.0);  
Serial.println(F(" hPa"));  
  
Serial.print(F("Humidity = "));  
Serial.print(bme.humidity);  
Serial.println(F(" %"));  
  
Serial.print(F("Gas = "));  
Serial.print(bme.gas_resistance / 1000.0);  
Serial.println(F(" KOhms"));  
  
Serial.print(F("Approx. Altitude = "));  
Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));  
Serial.println(F(" m"));  
  
Serial.println();  
delay(2000);  
}
```

Las apropiadas conversiones han sido realizadas para la correcta visualización de los datos obtenidos del sensor, de igual forma el tiempo entre la toma de datos se ha fijado a dos (2) segundos, puesto que, por el momento solo requerimos de observar el correcto funcionamiento del sensor, y aún no realizamos el envío de los datos a ninguna plataforma.

A continuación, se ejemplifica la salida de los datos obtenidos del sensor en el monitor serial.

```
BME680 async test  
Reading started at 0 and will finish at 1000  
You can do other work during BME680 measurement.  
Reading completed at 1000  
Temperature = 24.00 *C  
Pressure = 1013.25 hPa  
Humidity = 50.00 %  
Gas = 0.00 KOhms  
Approx. Altitude = 0.00 m
```

Una vez que se han obtenido los datos del sensor de manera correcta, se procede a la integración de la pantalla OLED al prototipo, para ello se requiere de la librería Adafruit SSD1306, la cual se puede instalar desde el administrador de librerías de Arduino IDE.

Nota. El siguiente código al igual que los previos se puede encontrar en la parte de los anexos, si bien se desarrolló basándose en los ejemplos de las librerías usadas por Adafruit, las correctas atribuciones deben de ser brindadas de acuerdo a la licencia BSD de Adafruit Industries del presente código.

BME680_OLED/BME680_OLED.ino Sección 18.3.1.3

.ino

```
*****
This is a library for the BME680 gas, humidity, temperature & pressure
sensor
```

```
Designed specifically to work with the Adafruit BME680 Breakout
----> http://www.adafruit.com/products/3660
```

```
These sensors use I2C or SPI to communicate, 2 or 4 pins are required
to interface.
```

```
Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing products
from Adafruit!
```

```
Written by Limor Fried & Kevin Townsend for Adafruit Industries.
BSD license, all text above must be included in any redistribution
```

```
***** /
```

BSD License. Es una licencia de software que permite a los usuarios utilizar, modificar y redistribuir el software sin restricciones, siempre y cuando se incluya el aviso de copyright y la licencia original.

A las previas librerías se le agregan las siguientes para el correcto funcionamiento de la pantalla OLED a través de la comunicación I2C.

.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Se definen los pines de la pantalla OLED, y se crea el objeto de la clase `Adafruit_SSD1306`.

```
#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10
```

.ino

Se declara el objeto de la clase `Adafruit_SSD1306`, el cual se encargará de la comunicación con la pantalla OLED.

```
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);
```

.ino

De la función de `void setup()` anteriormente descrita, se agrega la inicialización de la pantalla OLED, al igual que se reduce la frecuencia de baudios a la que se trabajará con el fin de tener una conexión más estable.

```
void setup() {
    Serial.begin(9600);
    Serial.println(F("BME680 test"));

    // by default, we'll generate the high voltage from the 3.3v line
    // internally! (neat!)
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with the I2C
    addr 0x3C (for the 128x32)
    // init done
    display.display();
    delay(100);
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    ...
}
```

.ino

Para la función de `void loop()` se agrega la impresión de los datos obtenidos del sensor en la pantalla OLED.

Primero definimos las coordenadas donde se imprimirá el texto en la pantalla OLED.

.ino

```
void loop() {
    display.setCursor(0,0);
    display.clearDisplay();
    ...
}
```

Después se imprime el texto en la pantalla OLED.

.ino

```
void loop() {
    ...
    display.print("Temperature: "); display.print(bme.temperature);
    display.println(" *C");
    display.print("Pressure: "); display.print(bme.pressure / 100.0);
    display.println(" hPa");
    display.print("Humidity: "); display.print(bme.humidity);
    display.println(" %");
    display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);
    display.println(" KOhms");

    display.display();
    ...
}
```

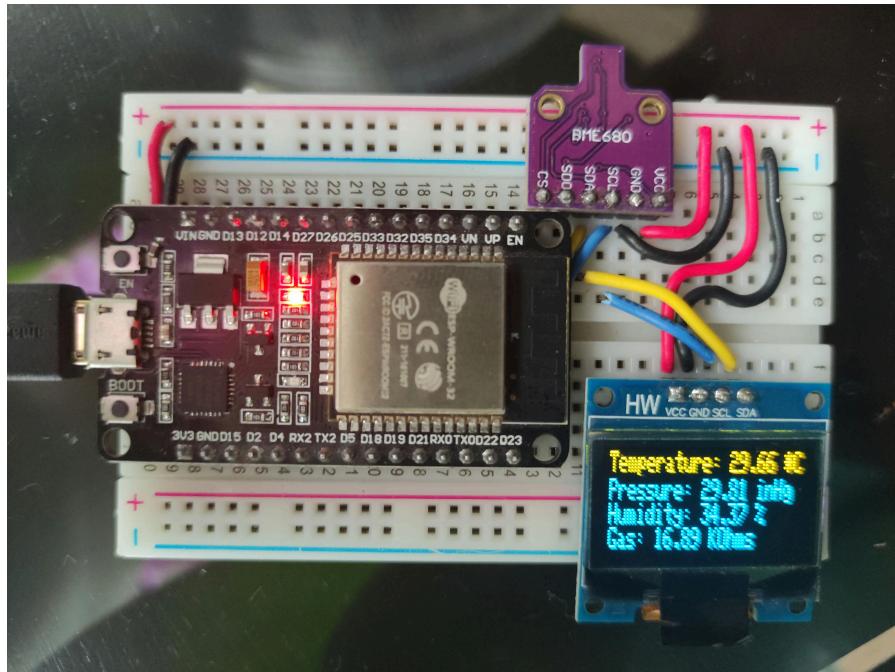
Lo siguiente debería de poder observarse en la [Figura 11](#).

Figura 11. Pantalla OLED con datos del sensor BME680.



El prototipo listo para pruebas se puede observar en la Figura 12.

Figura 12. Prototipo ESP32-BME680.



Aún falta integrar el envío de los datos a la plataforma de PocketBase, pero lo integrado hasta el momento abarca completamente la parte del hardware necesario para el desarrollo del prototipo.

BME680_OLED_PocketBase/BME680_OLED_PocketBase.ino Sección 18.3.1.4

Para la tercera etapa del desarrollo del código que se ejecutara en nuestro prototipo partimos a partir de la última versión del código, en el cual se agrega la librería de WiFi, la cual nos permitirá conectarnos a una red WiFi, la librería de HTTPClient, la cual nos permitirá realizar peticiones HTTP, sys/time para poder obtener la hora del sistema, así como esp_sleep para poder poner a dormir al ESP32.

```
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <esp_sleep.h>
```

.ino

Se define el pin para el LED integrado en el ESP32, el cual nos servirá para indicar cuando se esté realizando la toma y envío de datos a PocketBase.

.ino

```
#define LED 2
```

Se define el nombre de la red WiFi a la que nos conectaremos, así como la contraseña de la misma.

.ino

```
const char* ssid = "SSID"; // Network SSID  
const char* password = "password"; // Network password
```

Nota. Es importante notar que cuando el código llegue a requerir de credenciales o llaves de acceso, la escritura de estas será evitada en el código mostrado.

Se declaran las variables del servidor NTP al que se le realizará la petición para obtener la hora del sistema.

.ino

```
const char* ntpServer = "pool.ntp.org"; // NTP server  
const long gmtOffset_sec = 0; // Offset from GMT  
const int daylightOffset_sec = 0; // Offset from daylight savings time
```

El servidor NTP empleado es `pool.ntp.org`, el cual es un conjunto de servidores NTP distribuidos alrededor del mundo, los cuales se encargan de proveer la hora del sistema. Para más información sobre el servidor NTP `pool.ntp.org` se puede consultar la siguiente liga, <https://www.pool.ntp.org/es/>.

NTP. *Network Time Protocol, es un protocolo de red para sincronizar los relojes de los sistemas informáticos a través de paquetes de datos enviados por la red.*

Para mantener un buen desarrollo del código, se ahondará en crear descripciones para cada una de las funciones que se vayan a emplear, de esta forma se tendrá un mejor

entendimiento del código, se facilitara su mantenimiento, y se podrán detectar errores más fácilmente.

Para la función de `void setup()` se agrega la inicialización de la conexión WiFi, así como la inicialización de la conexión con el servidor NTP.

```
.ino

void setup() {
    ...
    Serial.println("Starting BME680...");

    Serial.println("Starting Wi-Fi..."); // Print a message to the serial
monitor
    wifiConnected = connectToWifi(); // Obtain the Wi-Fi connection status

    if (wifiConnected) {
        Serial.println("Connection to Wi-Fi successful"); // Print a
message to the serial monitor
        Serial.println("Starting time sync..."); // Print a message to the
serial monitor
        configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
        while (!time(nullptr)) {
            delay(1000);
            Serial.println("Waiting for time sync...");
        }
        Serial.println("Time synced");
    } else {
        Serial.println("Connection to Wi-Fi failed"); // Print a message
to the serial monitor
        Serial.println("Proceeding without Wi-Fi..."); // Print a message
to the serial monitor
    }
    ...
}
```

Como se puede observar, se agrega la inicialización de la conexión WiFi, la cual se realiza mediante la función `connectToWifi()`, la cual se encarga de realizar la conexión a la red WiFi, y regresa un valor booleano indicando si la conexión fue exitosa o no.

La implementación anterior implica que sin la conexión WiFi no se podrá obtener la hora del sistema, por lo que se agrega un `if` para verificar si la conexión fue exitosa o no, y en caso de que no lo sea, se imprime un mensaje indicando que se procederá sin la conexión WiFi.

Igualmente se crea un aviso visual que nos permita visualizar el correcto funcionamiento del prototipo sin necesidad de tener una conexión serial al iniciar

```
.ino

void setup() {
    ...
    // By default, we'll generate the high voltage from the 3.3v line
    internally! (neat!)
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // Initialize with the I2C
    addr 0x3C (for the 128x32)
    display.display();
    delay(100);
    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    ...
}
```

Se agrega un mensaje serial indicando que el `setup` ha finalizado.

```
.ino

void setup() {
    ...
    Serial.println("Setup complete");
    ...
}
```

Para la ejecución y funcionamiento del prototipo se debe asegurar que los datos puedan continuar siendo tomados y mostrados aún sin una correcta conexión WiFi.

La implementación de este tipo de sistema redundante es más fácil de llevar a cabo a través del seccionamiento de código a través de funciones, de esta forma se puede tener un mejor control sobre el código, así como una mejor organización.

Para imprimir los datos al serial se crea la función `printToSerial()`, la cual se encarga de imprimir los datos obtenidos del sensor BME680 al serial.

```
.ino

void printToSerial() { // Print to serial monitor
    Serial.print("Temperature = "); Serial.print(bme.temperature);
    Serial.println(" *C");
```

```
Serial.print("Pressure = "); Serial.print(bme.pressure / (20 *  
133.32239)); Serial.println(" inHg");  
Serial.print("Humidity = "); Serial.print(bme.humidity);  
Serial.println(" %");  
Serial.print("Gas = "); Serial.print(bme.gas_resistance / 1000.0);  
Serial.println(" KOhms");  
}
```

Para imprimir los datos al display OLED se crea la función `printToDisplay()`, la cual se encarga de imprimir los datos obtenidos del sensor BME680 al display OLED.

```
.ino  
  
void printToDisplay() { // Print to OLED display  
    display.setCursor(0,0);  
    display.clearDisplay();  
    display.print("Temperature: "); display.print(bme.temperature);  
    display.println(" *C");  
    display.print("Pressure: "); display.print(bme.pressure / (20 *  
133.32239)); display.println(" inHg");  
    display.print("Humidity: "); display.print(bme.humidity);  
    display.println(" %");  
    display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);  
    display.println(" KOhms");  
    display.display();  
}
```

Para checar la conexión WiFi se crea la función `connectToWifi()`, la cual se encarga de verificar si la conexión WiFi sigue activa, en caso de que no lo sea, se intenta reconectar.

```
.ino  
  
bool connectToWifi() { // Connect to the Wi-Fi network  
    Serial.print("Connecting to ");  
    Serial.println(ssid);  
  
    unsigned long startTime = millis(); // Get the current time  
  
    WiFi.begin(ssid, password); // Connect to the network  
  
    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to  
    connect  
        delay(500);  
        Serial.print(".");
    }
}
```

```
        if (millis() - startTime > 60000) { // If it's been more than 1
minute
            Serial.println("");
            Serial.println("WiFi connection timed out");
            return false; // Return false
        }
    }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP()); // Print the local IP address

return true; // Return true if connection was successful
}
```

Para obtener el tiempo actual UTC se crea la función `getUTCTime()`, la cual se encarga de obtener el tiempo actual UTC.

.ino

```
String getUTCTime() { // Get UTC time from NTP server
    struct timeval tv;
    gettimeofday(&tv, nullptr);
    time_t now = tv.tv_sec;
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    char buffer[30];
    sprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.
%03ldZ",
            timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
            timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
            tv.tv_usec / 1000);
    return String(buffer);
}
```

La función para enviar los datos a PocketBase se crea como `sendToPocketBase()`, la cual se encarga de enviar los datos a PocketBase.

.ino

```
void sendToPocketBase() { // Send data to PocketBase
    Serial.println("Sending data to PocketBase...");
```

```
HTTPClient http;

http.begin("https://custom.domain/api/collections/collection_name/
records"); // Specify the URL
http.addHeader("Content-Type", "application/json"); // Specify
content-type header

// Create the JSON payload
String payload = "{\"time\": \"\" + timeUTC + "\", \"temperature\": \""
+ bme.temperature + "\", \"pressure\": \"\" + bme.pressure / (20 *
133.32239) + "\", \"humidity\": \"\" + bme.humidity + "\", \"gas\": \"\" +
bme.gas_resistance / 1000.0 + \"\"}";

int httpCode = http.POST(payload); // Send the request

if(httpCode == 200) { // Check the returning code
    Serial.println("Data sent to PocketBase successfully");
} else { // If the code is not 200, something went wrong
    Serial.print("Error sending data to PocketBase, returned code: ");
    Serial.println(httpCode);

    Serial.println("Restarting ESP32...");
    delay(1000); // Wait for the serial output to finish
    esp_restart(); // Restart the ESP32
}

http.end(); // Close connection
}
```

En la previa función se puede observar que no se ha logrado una correcta implementación en el uso de las variables para la conexión en nuestro caso el URL del servidor de PocketBase debería ser guardado en una variable global, sin embargo, por el momento se ha dejado de esta forma para poder realizar pruebas de envío de datos de la forma más inmediata posible. A causa de los siguientes principios.

“First make it work.” You are out of business if it doesn’t work. **“Then make it right.”** Refactor the code so that you and others can understand it and evolve it as needs change or are better understood. **“Then make it fast.”** Refactor the code for “needed” performance.

(Martin, 2017)

Nota. La modularización del código será una práctica recurrente en el desarrollo del prototipo, por lo que se recomienda tener un buen manejo de las funciones y su implementación.

Finalmente integramos las funciones en el `void loop()` para que se ejecuten de forma continua.

```
.ino

void loop() {
    digitalWrite(LED, HIGH); // Turn the LED on (Note that LOW is the
    voltage level

    display.setCursor(0,0);
    display.clearDisplay();

    if (! bme.performReading()) {
        Serial.println("Failed to perform BME680 reading");
        return;
    }

    printToSerial(); // Print to serial monitor
    printToDisplay(); // Print to OLED display

    if (!wifiConnected) { // If Wi-Fi is not connected, wait 10 minutes
        and try again
        Serial.println("Retrying in 10 minutes...");
        digitalWrite(LED, LOW); // Turn the LED off by making the voltage
        HIGH
        delay(10 * 60 * 1000); // Wait 10 minutes

        esp_restart(); // Restart the ESP32
    }

    timeUTC = getUTCTime(); // Get UTC time from NTP server

    sendToPocketBase(); // Send data to PocketBase

    Serial.println("Updating in 10 minutes...");
    digitalWrite(LED, LOW); // Turn the LED off by making the voltage HIGH

    delay(10 * 60 * 1000); // Wait 10 minutes
}
```

La estructura del código previo nos permite mantener visualmente la información ambiental, y al mismo tiempo reintentar una conexión a internet en caso de que no se haya logrado una conexión exitosa, además de enviar los datos a PocketBase y reiniciar el ESP32 cada 10 minutos.

15.2.2.8. Pruebas

Se llevará a cabo la realización de pruebas en casa para verificar el funcionamiento del prototipo.

Para iniciar debemos monitorear el puerto serial para verificar que el ESP32 se encuentre funcionando correctamente.

A continuación, se muestra el resultado de una correcta ejecución del código en el ESP32.

Referencia 4. Resultados de correcta ejecución en el ESP32.

```
Starting BME680...
Starting Wi-Fi...
Connecting to SSID
Wi-Fi connected
IP address: 192.168.1.XXX
Connection to Wi-Fi successful
Starting time sync...
Time synced
Setup complete
Temperature = 28.11 *C
Pressure = 29.72 inHg
Humidity = 30.14 %
Gas = 9.98 KOhms
Sending data to PocketBase...
Data sent to PocketBase successfully
Updating in 10 minutes...
```

Si se llevó a cabo una correcta ejecución del código, se debería poder observar el registro de los datos en PocketBase.

Figura 13. Datos de la colección BM680 (*PocketBase*).

The screenshot shows a web-based database interface for 'PocketBase'. At the top, there's a navigation bar with 'Collections / bme680' and icons for refresh, search, and settings. To the right are buttons for 'API Preview' and '+ New record'. Below the header is a search bar with the placeholder 'Search term or filter like created > "2022-01-01" ...'. The main area displays a table with the following columns: id, time, temperature, pressure, humidity, gas, created, and updated. A single record is listed:

	id	time	temperature	pressure	humidity	gas	created	updated	...
<input type="checkbox"/>	pbgdut480us2zqx	2023-05-07 23:24:15 UTC	28.11	29.72	30.14	8.98	2023-05-07 23:24:17 UTC	2023-05-07 23:24:17 UTC	→

Nota. En ocasiones la lectura del tiempo del servidor NTP puede fallar, en tal caso la fecha mostrada pasara a ser 1970-01-01 00:00:00, esto se debe a que el tiempo Unix comienza en esa fecha. Así mismo en estos casos es recomendable usar el tiempo de registro en la base de datos de PocketBase denotado como `created`.

Epoch Unix Timestamp es el número de segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 GMT. Para mayor información consultar <https://www.unixtimestamp.com/>.

Con el correcto funcionamiento del prototipo, se dan por finalizadas las pruebas de la etapa de desarrollo.

15.2.3. Estación Meteorológica

Para el desarrollo del prototipo de la estación meteorológica se integran tres sistemas, las guías, manuales y componentes usados se detallarán a continuación.

Nota. El presente desarrollo es desarrollado mediante el uso de la previa metodología, aplicada en el prototipo del ESP32 y el BME680, por lo que se recomienda tener una buena comprensión de la misma. Las librerías y tecnologías usadas pueden explorarse a fondo en su respectiva documentación, además de que el código utilizado se encuentra en los anexos, pero también recibirá actualizaciones en el siguiente repositorio:

https://github.com/0sares10/Sparkfun-MicroMod-Weather_PocketBase

15.2.3.1. Guías, Manuales y Tutoriales

Para el desarrollo del prototipo las siguientes guías, manuales y tutoriales fueron de gran ayuda.

- <https://learn.sparkfun.com/tutorials/micromod-esp32-processor-board-hookup-guide> .
(SparkFun, s. f.-a)
- <https://learn.sparkfun.com/tutorials/micromod-weather-carrier-board-hookup-guide> .
(SparkFun, s. f.-b)
- <https://learn.sparkfun.com/tutorials/weather-meter-hookup-guide> (SparkFun, s. f.-c)
- <https://www.lacrossetechnology.com/products/925-1418> (La Crosse Technology, s. f.)

15.2.3.2. Componentes

Para el desarrollo del prototipo se usaron los siguientes componentes.

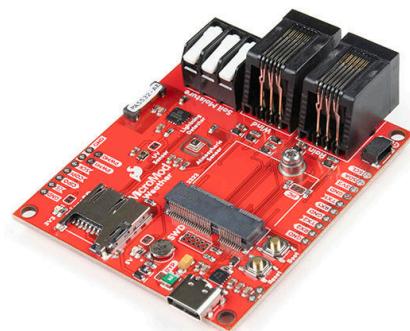
SparkFun MicroMod ESP32 Processor. El procesador principal, manteniéndonos en el entorno del ESP32 nos permitirá trabajar en el mismo ecosistema de desarrollo, además de que el ESP32 cuenta con conectividad Wi-Fi, lo que nos permitirá enviar los datos a PocketBase. (SparkFun, s. f.-d)

Figura 14. MicroMod ESP32 Processor (SparkFun).



SparkFun MicroMod Weather Carrier Board. La placa portadora del procesador, esta placa nos permite conectar el procesador a los sensores y a la computadora. (SparkFun, s. f.-e)

Figura 15. MicroMod Weather Carrier Board (SparkFun).



Weather Meter Kit. El kit de sensores, este kit nos permite medir la velocidad del viento, la dirección del viento y la precipitación. (Sparkfun, s. f.)

Figura 16. Weather Meter Kit (SparkFun).



925-1418 Sensor Weather Shield. Protector exterior diseñado para el sensor de temperatura, presión y humedad. (La Crosse Technology, s. f.)

Figura 17. 925-1418 Sensor Weather Shield (*La Crosse Technology*).



MicroSD. Tarjeta de memoria para el almacenamiento de los datos, esta tarjeta se conecta a la placa portadora.

Nota. Se utilizarán múltiples tarjetas microSD a lo largo del proyecto, por lo que se procede a ilustrar una genérica. En las pruebas realizadas en una tarjeta de 8GB no se obtuvo más allá de un 5% de uso de la memoria de la tarjeta, por lo que se considera que no se tendrá problemas de almacenamiento.

Figura 18. MicroSD (*Kingston*).



15.2.3.3. Funcionalidades

Entre las funcionalidades del prototipo que se implementaran se encuentran:

- Medición de los datos ambientales (temperatura, presión, humedad y punto de rocío) a través del BME280 que se encuentra en la placa portadora.
- Medición de UVA, UVB y UV Index a través del VEML6075 que se encuentra en la placa portadora.
- Medición de la velocidad del viento, la dirección del viento y la precipitación a través del Weather Meter Kit.
- Envío de los datos a PocketBase a través de Wi-Fi.
- Almacenamiento de los datos en microSD en estación como copia de seguridad de la información en caso de que no se pueda enviar a PocketBase, y permitiendo así que pueda operar aún sin una conexión a internet.

Nota. La SparkFun MicroMod Weather Carrier Board cuenta también con el sensor AS3935 Lightning Detector, sin embargo, este no se implementará en el prototipo por el momento.

Se espera que a través de las presentes funcionalidades se pueda obtener una buena cantidad de datos ambientales, los cuales serán de gran utilidad para el análisis de la calidad del aire.

Al momento de realización se planea realizar una prueba de aproximadamente 15 días, para así poder comparar los datos obtenidos y poder determinar si el prototipo es funcional.

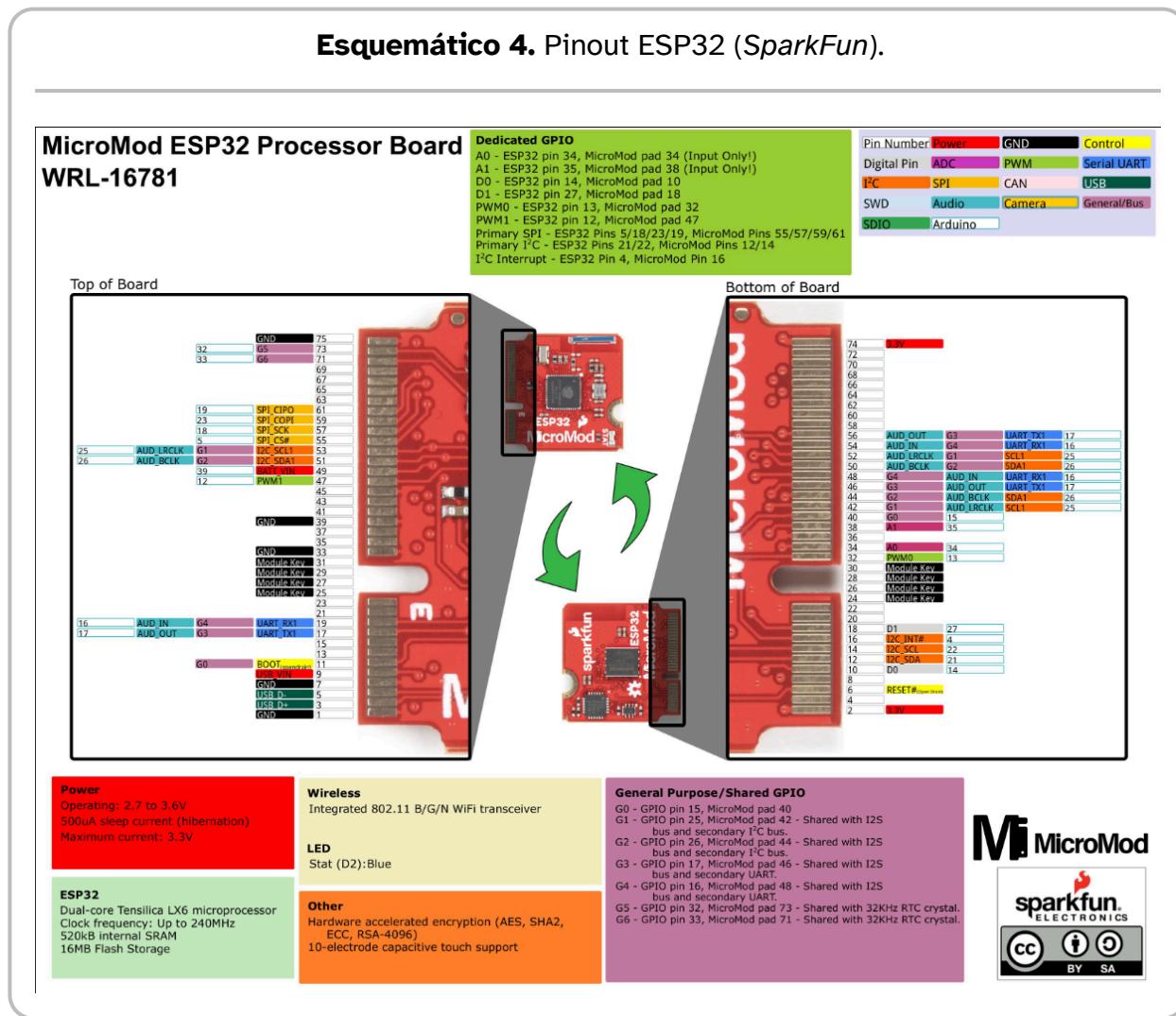
Los datos a recolectar serán los siguientes:

- Temperatura.
- Presión.
- Humedad.
- Punto de rocío.
- UVA.
- UVB.
- UV Index.
- Velocidad del viento.
- Dirección del viento.
- Precipitación.

15.2.3.4. Pinouts

A continuación, se listan los pinouts utilizados para el desarrollo del prototipo.

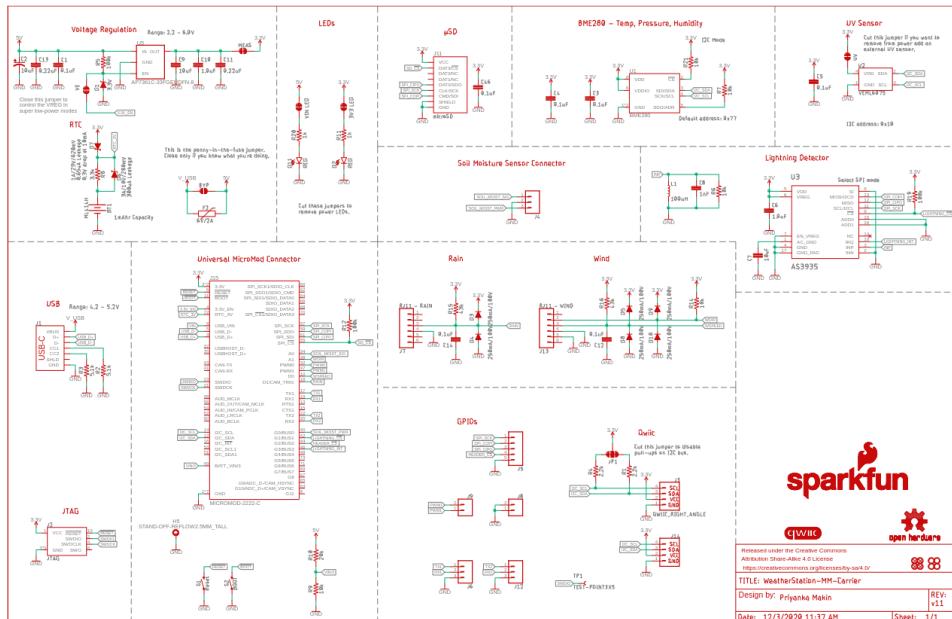
MicroMod ESP32. Para la comunicación e integración de los sensores se utilizó un MicroMod ESP32, el cual se puede observar en el siguiente **Esquemático 4.**



GPIO. General Purpose Input/Output, es un pin que puede ser configurado para ser de entrada o de salida, y puede ser utilizado para múltiples propósitos.

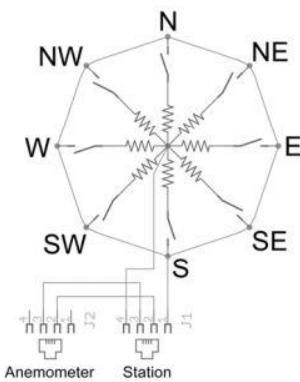
MicroMod Weather Meter Carrier Board. Para la conexión de los sensores se utilizó una MicroMod Weather Meter Carrier Board, la cual se puede observar en el siguiente Esquemático 5.

Esquemático 5. Weather Carrier Board (SparkFun).



Weather Meter Kit (Veleta de viento). Para la conexión de la veleta de viento se utilizó un Weather Meter Kit, el cual se puede observar en el siguiente Esquemático 6.

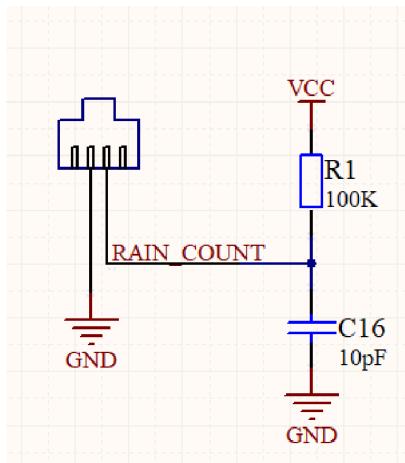
Esquemático 6. Weather Meter Kit Anemómetro (SparkFun).



Nota. Se puede observar que en el esquemático de la veleta de viento se integra el del anemómetro, esto se debe a que ambos comparten el mismo conector RJ11, por lo que se debe de conectar el cableado de ambos sensores al mismo conector. En el cual el anemómetro solo utilizara dos de los cuatro pines, mientras que la veleta de viento utilizara los otros dos.

Weather Meter Kit (Pluviómetro). Para la conexión del pluviómetro se utilizó un Weather Meter Kit, el cual se puede observar en el siguiente [Esquemático 7](#).

Esquemático 7. Weather Meter Kit Pluviómetro (*SparkFun*).



15.2.3.5. Ensamblaje

La primera parte corresponde al armado del Kit de sensores, para ello se siguen las instrucciones de la guía de ensamblaje del kit de sensores. (SparkFun, s. f.-c)

Para el ensamblaje se requiere de las siguientes herramientas:

- Weather Meter Kit.
- Destornillador Phillips.
- Destornillador plano.

Componentes. Cuando el paquete es recibido, se debe contar con los siguientes componentes.

Figura 19. Contenido del Weather Meter Kit (SparkFun).



- Dos (2) tubos metálicos.
- Tres (3) sensores:
 - Pluviómetro.
 - Anemómetro.
 - Veleta.

- Dos (2) abrazaderas.
- Un (1) brazo de montaje central.
- Un (1) brazo de montaje lateral.
- Una (1) bolsa de tornillos y tuercas.
- Un (1) paquete de bridás de plástico.

Todos los sensores son pasivos por lo que requieren de una fuente de alimentación externa, en este caso se usará la placa portadora, por lo que se debe conectar el cable de comunicación a la placa, por lo cual cada sensor externo cuenta con un conector RJ-11.

Pluviómetro. El pluviómetro es el sensor que mide la precipitación, este sensor cuenta con mecanismo que mide la cantidad de agua que cae en el recipiente.

Figura 20. Pluviómetro del Weather Meter Kit (SparkFun).



RJ11. Registered Jack 11, es un conector telefónico de cuatro o seis pines, utilizado para conectar teléfonos y otros dispositivos de telecomunicaciones. En este caso se utiliza para conectar los sensores al microcontrolador.

Nota. El modelo utilizado cuenta con un nivel de burbuja para asegurar que el pluviómetro se encuentre nivelado.

Figura 21. Nivel de burbuja del pluviómetro del Weather Meter Kit (*SparkFun*).



El sensor es un colector de cubo basculante de vaciado automático. Esto significa que por cada 0.011in (0.2794mm) de lluvia que cae en el sensor, la cubeta se inclinará, descargando el agua y cerrando un contacto momentáneo.

El cierre del interruptor se puede medir usando pines de interrupción o un contador digital. Los conductores centrales del conector RJ-11 están conectados al interruptor del medidor.

Anemómetro. El anemómetro es el sensor que mide la velocidad del viento, este sensor cuenta con tres (3) copas que giran con el viento.

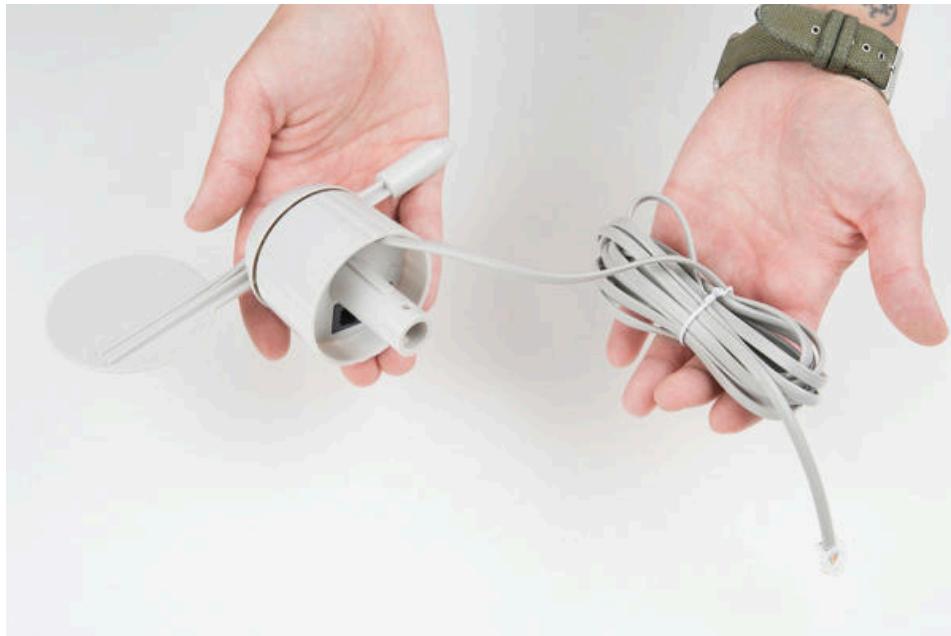
Figura 22. Anemómetro del Weather Meter Kit (SparkFun).



El viento mueve las copas del anemómetro, que a su vez hacen girar un imán cerrado. El imán cierra un interruptor de láminas en cada rotación, lo que se refleja en la salida. Se puede medir esto en los dos conductores internos del conector RJ-11 (pines 2 y 3), usando un contador digital o pines de interrupción en su microcontrolador. Para convertir esto en una velocidad de viento funcional, use la conversión de 1.492mph = 1 cierre de interruptor/segundo. Para aquellos en tierra métrica, esto es 2.4km/h .

Veleta de viento. La veleta indica la dirección en la que sopla el viento.

Figura 23. Veleta de viento del Weather Meter Kit (SparkFun).



Si bien podría pensarse que este sensor es algo simple es el más complejo, A continuación, se describe su funcionamiento, recordar el [Esquemático 6](#)

Dentro de la veleta se encuentran ocho (8) interruptores, cada uno con una resistencia única.

A medida que la veleta gira, un imán cierra los interruptores de lengüeta y puede cerrar dos a la vez debido a su proximidad entre sí. Con el uso de una resistencia externa, se puede crear un divisor de voltaje. Medir la salida de voltaje con un convertidor de analógico a digital en su microcontrolador permite determinar la dirección a la que apunta la veleta.

Como la salida de tensión dependerá del valor de la resistencia externa utilizada, no existe una función de conversión común.

Nota. La dependencia de la salida de las resistencias culminará en el uso de códigos de prueba para calibrar a preferencia el sensor.

Dado que los valores emitidos por la veleta se basan en grados, en teoría, cualquier valor puede representar cualquier dirección. Sin embargo, se recomienda que el valor en el

grado 0 represente el norte para facilitar su uso. También hay indicadores de dirección muy pequeños y apenas visibles en los cuatro lados de la veleta. Al elegir diferentes valores para indicar direcciones, asegurar de marcarlos en consecuencia. Al instalar y colocar el medidor meteorológico, se debe asegurar de que las marcas de dirección estén apuntando en la orientación correcta.

Nota. Tener en cuenta que la veleta “apunta” en la dirección desde la que sopla el viento.

Armazón.

Para comenzar, es necesario unir los dos tubos de metal y deslizarlos juntos.

Figura 24. Unión de tubos Weather Meter Kit (SparkFun).



A continuación, se coloca el brazo central en la parte superior de los tubos. Asegurando la alineación con la muesca en el tubo superior.

Figura 25. Unión del brazo central Weather Meter Kit (SparkFun).



Se usa un tornillo para asegurar el brazo central en su lugar.

Figura 26. Unión del brazo central con tornillo Weather Meter Kit (SparkFun).



Siguiente se monta el anemómetro en el brazo central.

El anemómetro cuenta con una protuberancia que coincide en las muescas del brazo central. Esto ayuda a asegurar que el anemómetro esté conectado correctamente y solo permite que se monte en una dirección.

Figura 27. Unión del anemómetro Weather Meter Kit (SparkFun).



Se desliza el anemómetro en el brazo central hasta que se atore en su lugar.

Figura 28. Unión del anemómetro Weather Meter Kit (*SparkFun*).



El anemómetro es fijado con la ayuda de un tornillo.

Figura 29. Unión del anemómetro con tornillo Weather Meter Kit (*SparkFun*).



Para unir la veleta de viento, se sigue el mismo procedimiento que para el anemómetro. Primero se alinea la protuberancia de la veleta con la muesca del brazo central y después se atora en su lugar.

Figura 30. Unión de la veleta de viento Weather Meter Kit (SparkFun).



Se asegura la veleta de viento con un tornillo.

Figura 31. Unión de la veleta de viento con tornillo Weather Meter Kit (SparkFun).



Para conectar el pluviómetro se necesita el brazo lateral. Esto para mantener el pluviómetro alejado de los otros sensores para garantizar que pueda obtener una medición precisa. Si el pluviómetro está demasiado cerca de los otros sensores, el agua de lluvia puede salpicar en el pluviómetro desde el anemómetro o la veleta de viento.

Se fija el brazo lateral a los tubos de metal.

Figura 32. Unión del brazo lateral Weather Meter Kit (SparkFun).



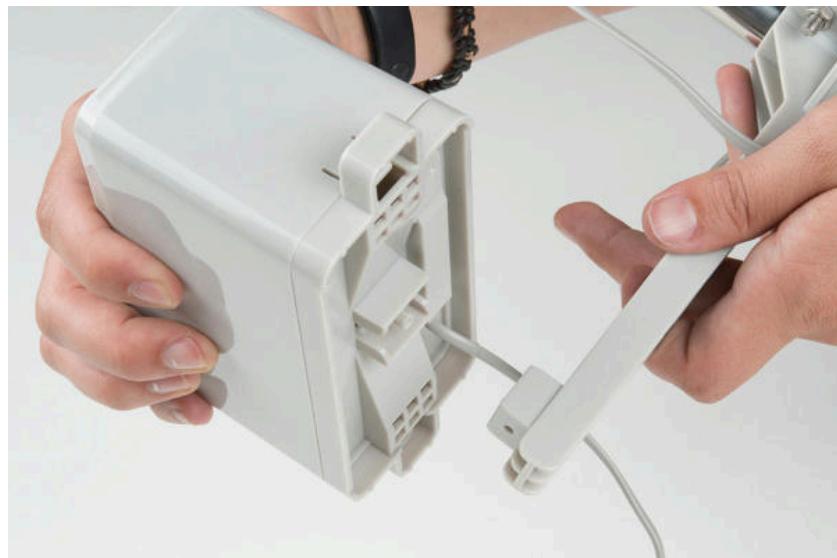
El pluviómetro cuenta también con muescas para asegurar que se monte de forma correcta.

Figura 33. Unión del pluviómetro Weather Meter Kit (SparkFun).



Antes de colocar se debe de alinear para después asegurarlo.

Figura 34. Unión del pluviómetro Weather Meter Kit (*SparkFun*).



Se asegura el pluviómetro con un tornillo. Manteniéndolo fijo y alineado al suelo para su mejor funcionamiento.

Figura 35. Unión del pluviómetro con tornillo Weather Meter Kit (*SparkFun*).



Gestión de cables.

Para la gestión de cables se utilizan abrazaderas de cables y clips integrados en el armazón.

Se desenredan los cables del anemómetro y la veleta de viento. Se sujetan mediante el uso de los clips en la parte inferior del brazo central.

Figura 36. Gestión de cables Weather Meter Kit (*SparkFun*).



Una vez sujetados los cables se conecta el cable más corto que sale del anemómetro a la veleta de viento.

Figura 37. Conexión de anemómetro a veleta Weather Meter Kit (*SparkFun*).



Para asegurar los cables sueltos se utilizan bridás de cables.

Figura 38. Sujeción de cables Weather Meter Kit (*SparkFun*).



De esta forma queda terminado el armazón del Weather Meter Kit.

Figura 39. Armazón terminado Weather Meter Kit (*SparkFun*).



Cuando se desea trabajar con equipo extra como en nuestro caso, se puede utilizar una abrazadera para darle más estabilidad al armazón.

Figura 40. Abrazadera Weather Meter Kit (SparkFun).

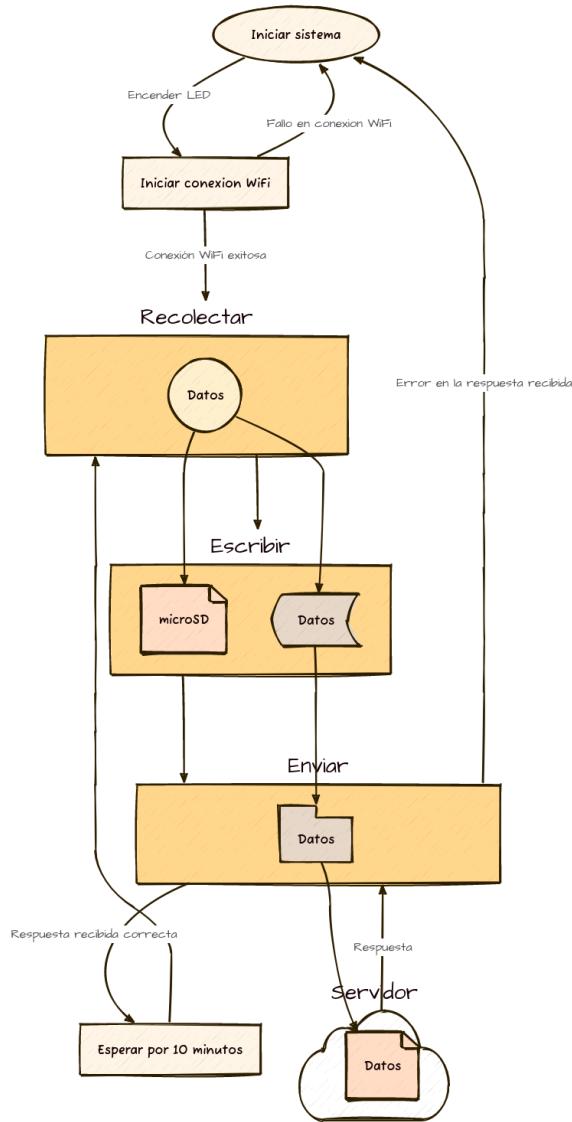


Nota. El procedimiento de la integración del protector de sensores queda pendiente, puesto que la unidad fue modificada para permitir su integración con los sensores utilizados.

15.2.3.6. Diagrama de Flujo

A continuación, se muestra el siguiente diagrama de flujo, el cual se puede observar en el siguiente [Diagrama 2](#) hecho con el lenguaje de programación D2 (D2 Lang, s. f.) .

Diagrama 2. Flujo del prototipo de estación.



15.2.3.7. Código

Se inicia el desarrollo con un simple código para el parpadeo del LED integrado, con el fin de verificar que el entorno de desarrollo este correctamente configurado.

Nota. Los siguientes códigos serán detallados paso a paso, pero el código en su totalidad puede ser consultado en la [Sección 18.3.2](#)

Blink/Blink.ino [Sección 18.3.2.1](#)

Se define el pin del LED integrado como salida.

```
.ino  
// Blink a LED on the MicroMod Weather (ESP32) board  
  
int ledPin = 2; // LED is connected to GPIO2  
  
void setup() {  
    pinMode(ledPin, OUTPUT); // Set GPIO2 to output mode  
    Serial.begin(115200); // Initialize serial port  
}  
  
void loop() {  
    digitalWrite(ledPin, HIGH); // Turn LED on  
    delay(1000); // Wait for 1000 millisecond(s)  
    Serial.println("The LED is on."); // Print a message  
    digitalWrite(ledPin, LOW); // Turn LED off  
    delay(1000); // Wait for 1000 millisecond(s)  
}
```

El primer sensor a integrar será el `BME280` por lo que la primera librería a integrar será `SparkFunBME280.h`.

[BME280/BME280.ino](#) [Sección 18.3.2.2](#)

Se incluye la librería `SparkFunBME280.h` y se define el pin del LED integrado como salida.

Se definen las librerías a usar, así como el objeto `bme280Sensor` para el sensor `BME280` y el valor `RealFloatPressure` para la presión atmosférica.

```
.ino  
  
#include <Wire.h>  
#include "SparkFunBME280.h"  
  
BME280 bme280Sensor; // Create BME280 object  
  
float RealFloatPressure;
```

Se continúa con el método `setup()` en el cual se inicializa el puerto serial, se inicia la comunicación con el sensor `BME280` y se imprime un mensaje de inicio.

Se trabajará a través del puerto serial a una velocidad de `115200` baudios, se inicia la comunicación con el sensor `BME280` empleando el medio de comunicación I2C previamente empleado y se imprime un mensaje de inicio.

Se define el pin del LED integrado como salida.

Se evalúa si el sensor `BME280` responde, en caso de no responder se imprime un mensaje de error y se congela el programa.

```
.ino  
  
void setup() {  
    Serial.begin(115200); // Initialize serial port  
    while (!Serial); // Wait for user to open serial monitor  
  
    Serial.println("MicroMod Weather Carrier Board - BME280 Example");  
    Serial.println();  
  
    Wire.begin(); // Join I2C bus  
  
    bme280Sensor.setReferencePressure(101500); // Set sea level pressure  
    to 101325 Pa (default)  
  
    if (bme280Sensor.begin() == false) { // Connect to BME280  
        Serial.println("BME280 did not respond.");  
        while(1); // Freeze  
    }  
  
    pinMode(LED_BUILTIN, OUTPUT);  
}
```

Se continúa con el método `loop()` en el cual se obtienen los valores de temperatura, humedad y presión atmosférica, se imprimen los valores obtenidos y se congela el programa por `1000` milisegundos.

```
.ino

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.print("Temperature: ");
    Serial.println(bme280Sensor.readTempC(), 2);
    Serial.print("Humidity: ");
    Serial.println(bme280Sensor.readFloatHumidity(), 0);
    Serial.print("Pressure: ");

    RealFloatPressure = bme280Sensor.readFloatPressure() / (20 * 133.32239);
    Serial.println(RealFloatPressure, 2);

    Serial.print("Altitude: ");
    Serial.println(bme280Sensor.readFloatAltitudeMeters(), 1);
    Serial.print("Dewpoint: ");
    Serial.println(bme280Sensor.dewPointC(), 2);

    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

El siguiente sensor a integrar será el `VEML6075` por lo que la siguiente librería a integrar será `SparkFun_VEML6075_Arduino_Library.h`.

`VEML6075/VEML6075.ino` Sección 18.3.2.3

Se incluye la librería `SparkFun_VEML6075_Arduino_Library.h` y se declara el objeto `veml6075` para el sensor `VEML6075`.

```
.ino

#include <SparkFun_VEML6075_Arduino_Library.h>

VEML6075 veml6075; // Create a VEML6075 object
```

Se continúa con el método `setup()` en el cual se inicializa el puerto serial, se inicia la comunicación con el sensor `VEML6075` y se imprime un mensaje de inicio.

Se trabajará a través del puerto serial a una velocidad de `115200` baudios, se inicia la comunicación con el sensor `VEML6075` empleando el medio de comunicación I2C previamente empleado y se imprime un mensaje de inicio.

Se define el pin del LED integrado como salida. Se evalúa si el sensor `VEML6075` responde, en caso de no responder se imprime un mensaje de error y se congela el programa.

```
.ino

void setup() {
    Serial.begin(115200);
    while(!Serial); // Wait for user to open serial monitor

    Serial.println("MicroMod Weather Carrier Board - VEML6075 Example");

    Wire.begin(); // Join I2C bus

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond."); // If the sensor does not
        respond, print an error message
        while(1); // Freeze
    }

    pinMode(LED_BUILTIN, OUTPUT);
    Serial.println("UVA, UVB, UV Index"); // Print the header for the data
    Serial.println();

}
```

Se continúa con el método `loop()` en el cual se obtienen los valores de radiación UV, se imprimen los valores obtenidos y se hace esperar al programa por `1000` milisegundos.

```
.ino

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    // Print the UVA, UVB, and UV Index values
    Serial.println("UVA: " + String(veml6075.uva()));
    Serial.println("UVB: " + String(veml6075.uvb()));
    Serial.println("UV Index: " + String(veml6075.index()));

    digitalWrite(LED_BUILTIN, LOW);
    delay(1000); // Wait 1 second
}
```

Una vez con los suficientes sensores integrados se procede a realizar la integración de los sensores en un código que integre todos los sensores.

Station/Station.ino Sección 18.3.2.4

Se incluyen las librerías `SparkFun_BME280_Arduino_Library.h` y `SparkFun_VEML6075_Arduino_Library.h` y se declaran los objetos `bme280Sensor` y `veml6075` para los sensores `BME280` y `VEML6075` respectivamente.

```
.ino

#include <Wire.h>
#include "SparkFunBME280.h"
#include <SparkFun_VEML6075_Arduino_Library.h>

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object
```

Las variables para almacenar los valores de temperatura, humedad, presión atmosférica, radiación UV, velocidad del viento, dirección del viento y lluvia se declaran como variables globales.

```
.ino

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall
```

Se definen las variables para las operaciones de tiempo y para el cálculo de la dirección y velocidad del viento.

```
.ino

const int duration = 1; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the
remaining minutes and seconds
```

Se definen las variables para los cálculos de la dirección y velocidad del viento.

```
.ino

volatile int windSpeedCount = 0; // Variable to store the number of wind
pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y
values
```

```
float x, y, theta, averageWindDirection, averageWindSpeed; // Variables to store the x, y, theta, and average wind direction and speed
```

Se definen las variables para almacenar los valores de temperatura, humedad, presión atmosférica, radiación UV, velocidad del viento, dirección del viento y lluvia en un `struct`.

```
.ino
```

```
struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
    float uvindex;
};
WeatherData weather;
```

Se definen las variables para almacenar los valores de velocidad y dirección del viento en un `struct`.

```
.ino
```

```
struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};
WindData wind[duration * 6];
```

Se procede a crear funciones de los principales componentes que integran el funcionamiento de la estación para de esta forma realizar un código más legible y ordenado.

Se crean interrupciones para los sensores de velocidad del viento y lluvia.

```
.ino

void rainIRQ()
{
    rainCount++;
    // Serial.println("Rain clicked");
}

// Function is called when the magnet in the anemometer is activated
void windSpeedIRQ()
{
    windSpeedCount++;
    // Serial.println("Wind clicked");
}
```

Se crea una función para obtener la dirección del viento.

```
.ino

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value
from the wind direction sensor

    if ( 10 < reading && reading <= 150) return (288);
    if ( 150 < reading && reading <= 250) return (264);
    if ( 250 < reading && reading <= 400) return (336);
    if ( 400 < reading && reading <= 600) return (312);
    if ( 650 < reading && reading <= 850) return ( 24);
    if ( 850 < reading && reading <= 1050) return ( 0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}
```

Nota. La función previa se obtuvo a partir de un análisis, puesto que la tabla de valores integrada en el sensor de dirección del viento no es la misma que la que se muestra en la hoja de datos del sensor.

A continuación, se muestra dicha tabla de valores para su comparación.

Figura 41. Tabla de valores del sensor de dirección del viento (*SparkFun*).

Direction (Degrees)	Resistance (Ohms)	Voltage (V=5v, R=10k)
0	33k	3.84v
22.5	6.57k	1.98v
45	8.2k	2.25v
67.5	891	0.41v
90	1k	0.45v
112.5	688	0.32v
135	2.2k	0.90v
157.5	1.41k	0.62v
180	3.9k	1.40v
202.5	3.14k	1.19v
225	16k	3.08v
247.5	14.12k	2.93v
270	120k	4.62v
292.5	42.12k	4.04v
315	64.9k	4.33v
337.5	21.88k	3.43v

Se crea una función para calcular la dirección y velocidad del viento.

```
.ino
void getAverageWind(float& averageWindDirection, float& averageWindSpeed)
{
    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // convert angle to radians
        x = wind[i].speed * cos(theta);
        y = wind[i].speed * sin(theta);
        xSum += x;
        ySum += y;
    }
    averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum /
(duration * 6))); // convert radians to degrees
    if (averageWindDirection < 0) averageWindDirection += 360; // convert
negative angles to positive
    averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum /
```

```
(duration * 6), 2)); // calculate average speed  
  
    xSum = 0;  
    ySum = 0;  
}
```

Se crea una función para imprimir los valores de los sensores.

```
.ino  
  
void printWeather() {  
    Serial.print("Temperature: ");  
    Serial.print(weather.temperature, 2); // Temperature in degrees  
    Celsius  
    Serial.print("    Humidity: ");  
    Serial.print(weather.humidity, 2); // Humidity in percent  
    Serial.print("    Dewpoint: ");  
    Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius  
    Serial.print("    Pressure: ");  
    Serial.print(weather.pressure, 2); // Pressure in inches of mercury  
    Serial.print("    Wind Speed: ");  
    Serial.print(weather.windSpeed, 2); // Wind speed in knots  
    Serial.print("    Wind Direction: ");  
    Serial.print(weather.windDirection, 2); // Average wind direction in  
    degrees  
    Serial.print("    Rain Fall: ");  
    Serial.print(weather.rainFall, 2); // Rain fall in inches/hour  
    Serial.print("    UVA: ");  
    Serial.print(weather.uva); // UVA value  
    Serial.print("    UVB: ");  
    Serial.print(weather.uvb); // UVB value  
    Serial.print("    UV Index: ");  
    Serial.println(weather.uvindex); // UV index  
}
```

La función `loop()` se encarga de tomar las mediciones de los sensores y de imprimir los valores en el monitor serial.

```
.ino  
  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED  
    rainCount = 0; // Reset the rain count  
    for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction
```

```
array
    wind[i].reading = false;
    wind[i].direction = 0;
    wind[i].speed = 0;
}
for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of
the report, taking measurements every 10 seconds
    Serial.print("Taking wind measurements, this process will be
finished in: "); // Print a message to the serial monitor to let the user
know the program is working
    remainingMinutes = duration - ((i / 6) + !(i % 6));
    remainingSeconds = 60 - ((i % 6) * 10);
    Serial.print(remainingMinutes);
    Serial.print(":");
    if (remainingSeconds == 60) {
        Serial.println("00");
    } else {
        Serial.println(remainingSeconds);
    }

    windSpeedCount = 0; // Reset the wind speed count
    delay(10 * 1000); // Wait for 10 seconds
    wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the
wind speed
    wind[i].direction = getWindDirection(); // Calculate the wind
direction
    wind[i].reading = true; // Set the reading flag to true
}
weather.rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate
the rain fall in inches/hour
getAverageWind(averageWindDirection, averageWindSpeed); // Get the
average wind direction and speed
weather.windDirection = averageWindDirection; // Set the average wind
direction to the weather data
weather.windSpeed = averageWindSpeed; // Set the average wind speed to
the weather data
weather.temperature = bme280.readTempC(); // Get the temperature in
degrees Celsius
weather.humidity = bme280.readFloatHumidity(); // Get the humidity in
percent
weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees
Celsius
weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); // Convert the pressure from Pascals to inches of mercury

weather.uva = veml6075.uva(); // Get the UVA value
```

```
weather.uvb = veml6075.uvb(); // Get the current time in UTC  
weather.uvindex = veml6075.index(); // Get the UV index  
  
printWeather(); // Print the weather data to the serial monitor  
  
digitalWrite(LED_BUILTIN, LOW); // Turn off the LED  
}
```

Nota. A partir de este punto los siguientes códigos se centrarán en la integración de las dos funciones faltantes omitiendo repetir los de la operación de la estación.

Una vez que la estación meteorológica y el servidor se encuentran funcionando de manera independiente, se procede a realizar la integración de ambos sistemas. Para la parte de la estación meteorológica se presenta A continuación, el código que se encarga de enviar los datos al servidor.

El presente código se puede evaluar en su totalidad en el anexo Station_PocketBase/Station_PocketBase.ino Sección 18.3.2.5

Nota. Si bien la estación empleara el uso de funciones para la conexión a WiFi, así como la obtención del presente tiempo UTC, ambos códigos ya se han tratado con anterioridad por lo que se omitirá su explicación.

La función a implementar será la encargada de enviar los datos al servidor, para ello se empleará la librería `HTTPClient.h` la cual permite realizar peticiones HTTP. La función lleva el nombre de `sendWeatherDataToPocketBase()` y se presenta a continuación.

```
.ino  
  
void sendWeatherDataToPocketBase() {  
    Serial.println("Sending weather data to PocketBase...");  
  
    HTTPClient http;  
  
    // Set the PocketBase endpoint URL  
    http.begin("https://custom.domain/api/collections/collection_name/  
records");  
  
    // Set the HTTP headers
```

```
http.addHeader("Content-Type", "application/json");

// Create the JSON payload
String payload = "{\"time\": \"" + weather.time +
    "\", \"temperature\": " + String(weather.temperature) +
    ", \"humidity\": " + String(weather.humidity) +
    ", \"dewpoint\": " + String(weather.dewpoint) +
    ", \"pressure\": " + String(weather.pressure) +
    ", \"rainFall\": " + String(weather.rainFall) +
    ", \"windSpeed\": " + String(weather.windSpeed) +
    ", \"windDirection\": " + String(weather.windDirection)

+
    ", \"uva\": " + String(weather.uva) +
    ", \"uvb\": " + String(weather.uvb) +
    ", \"uvindex\": " + String(weather.uvindex) + "}";

// Send the POST request with the payload
int httpCode = http.POST(payload);

// Check if the request was successful
if(httpCode == 200) {
    Serial.println("Data sent to PocketBase successfully");
} else {
    Serial.println("Error sending data to PocketBase");
    Serial.print("HTTP code: ");
    Serial.println(httpCode);
}

// Free resources
http.end();
}
```

Se puede notar que el proceso consta en formar un payload en formato JSON con los datos de la estación meteorológica, para posteriormente realizar una petición POST al servidor. En caso de que la petición sea exitosa, se imprimirá un mensaje en el monitor serial, en caso contrario se imprimirá un mensaje de error junto con el código HTTP de la petición.

JSON. Es un formato de texto sencillo para el intercambio de datos. Es fácil de leer y escribir para los humanos y fácil de analizar y generar para las máquinas.

Se procede a implementar el código que implementara la copia de seguridad de la información a la microSD. Para eso se creará un nuevo programa a cargar al ESP32, el cual llevará el nombre de Station_PocketBase_SD/Station_PocketBase_SD.ino

Sección 18.3.2.6

Station_PocketBase_SD/Station_PocketBase_SD.ino Sección 18.3.2.6

La función a implementar llevará el nombre de writeDataToSDCard() y se presenta a continuación.

```
.ino

void writeDataToSDCard() { // Write the weather data to the SD card
    Serial.println("Writing weather data to SD card...");

    for (int i = 0; i < 3; i++) { // Try opening the file up to 3 times
        dataFile = SD.open("/data.csv", FILE_APPEND); // Open the data
        file

        if (dataFile) { // If the file opened successfully, write the data
            dataFile.print(weather.time);
            dataFile.print(",");
            dataFile.print(weather.temperature);
            dataFile.print(",");
            dataFile.print(weather.humidity);
            dataFile.print(",");
            dataFile.print(weather.dewpoint);
            dataFile.print(",");
            dataFile.print(weather.pressure);
            dataFile.print(",");
            dataFile.print(weather.rainFall);
            dataFile.print(",");
            dataFile.print(weather.windSpeed);
            dataFile.print(",");
            dataFile.print(weather.windDirection);
            dataFile.print(",");
            dataFile.print(weather.uva);
            dataFile.print(",");
            dataFile.print(weather.uvb);
            dataFile.print(",");
            dataFile.println(weather.uvindex);
            dataFile.close(); // Close the file
            Serial.println("Data written to SD card successfully");
            return; // Exit the function after successful write
        } else { // If the file did not open successfully, print an error
            Serial.println("Error opening file on SD card. Retrying...");
            delay(500); // Wait for half a second before retrying
        }
    }
}
```

```
        }  
  
    // If all attempts to open the file have failed, print an error  
    message and restart the ESP32  
    Serial.println("Failed to write data to SD card after multiple  
attempts.");  
    Serial.println("Restarting ESP32...");  
  
    delay(1000); // Wait for the serial output to finish  
    esp_restart(); // Restart the ESP32  
}
```

Puede compararse con la función de envío de datos al servidor, la cual también se encarga de formar un payload solo que este en formato CSV, para posteriormente escribirlo en la microSD. En caso de que la escritura sea exitosa, se imprimirá un mensaje en el monitor serial, en caso contrario se imprimirá un mensaje de error y se reiniciara el ESP32.

CSV. *Es un formato de archivo que se utiliza para almacenar datos tabulares. Cada línea del archivo es una fila de la tabla, y los valores de las columnas se separan por comas.*

Debe considerarse que en este caso también se implementa un mecanismo de reintento de escritura, el cual se encarga de intentar abrir el archivo hasta 3 veces, en caso de que no se logre abrir el archivo, se reiniciara el ESP32. Así como la escritura de los datos con una fecha predeterminada en caso de que no se tenga acceso a la red, de esta forma se puede identificar cuando se perdió la conexión a internet y mantener el registro de datos.

15.2.3.8. Pruebas

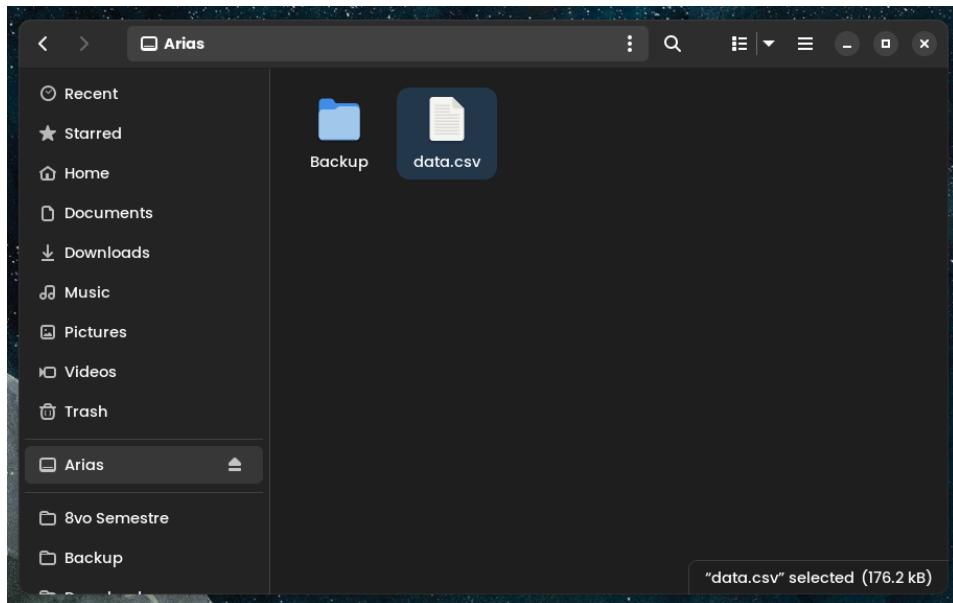
A continuación, se muestra el resultado de una correcta ejecución del código en la estación meteorológica. En la [Referencia 5](#) se puede observar el resultado de la ejecución del código en el monitor serial.

Referencia 5. Resultados de correcta ejecución en la estación meteorológica.

```
Starting MicroMod Weather Station...
Starting SD card...
SD card started
Starting Wi-Fi...
Connecting to SSID
Wi-Fi connected
IP address: 192.168.1.XXX
Starting time sync...
Waiting for time sync...
Time synced
Starting BME280...
BME280 started
Starting VEML6075...
VEML6075 started
Setup complete
Starting report...
Taking wind and rain measurements, this process will be finished in:
9:50
...
Time: 2023-04-30 23:49:34.000Z
Temperature: 30.16
Humidity: 9.85
Dew Point: -4.99
Pressure: 29.69
Rain Fall: 0
Wind Speed: 4.8
Wind Direction: 214.01
UVA: 55.62
UVB: 45.55
UV Index: 0.06
Writing weather data to SD card...
Data written to SD card successfully
Sending weather data to PocketBase...
Data sent to PocketBase successfully
Report complete
```

Si se llevó a cabo una correcta ejecución del código, se debería poder observar el registro de los datos en la microSD.

Figura 42. Archivo guardado en la microSD (*data.csv*).



Abriendo el archivo en un editor de texto se observa lo siguiente.

Figura 43. Contenido del archivo *data.csv* en la microSD.

The screenshot shows a text editor window with the file 'data.csv' open. The content of the file is a large list of data points, each consisting of a timestamp followed by a series of numerical values separated by commas. The data points represent sensor readings over time, with the first few lines showing values like 2023-04-30 23:49:34, 5772, 30, 16, 9, 85, -4, 99, 29, 69, 0, 00, 4, 80, 214, 01, 55, 62, 45, 55, 0, 06 and 2023-04-30 23:59:36, 1912, 29, 22, 10, 49, -4, 88, 29, 69, 0, 00, 10, 00, 287, 29, 51, 17, 41, 25, 0, 05.

Timestamp	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6	Value 7	Value 8	Value 9	Value 10	Value 11	Value 12	Value 13	Value 14	Value 15	Value 16	Value 17	Value 18	Value 19	Value 20	Value 21	Value 22	Value 23	Value 24	Value 25	Value 26	Value 27	Value 28	Value 29	Value 30	Value 31	Value 32	Value 33	Value 34	Value 35	Value 36	Value 37	Value 38	Value 39	Value 40	Value 41	Value 42	Value 43	Value 44	Value 45	Value 46	Value 47	Value 48	Value 49	Value 50	Value 51	Value 52	Value 53	Value 54	Value 55	Value 56	Value 57	Value 58	Value 59	Value 60	Value 61	Value 62	Value 63	Value 64	Value 65	Value 66	Value 67	Value 68	Value 69	Value 70	Value 71	Value 72	Value 73	Value 74	Value 75	Value 76	Value 77	Value 78	Value 79	Value 80	Value 81	Value 82	Value 83	Value 84	Value 85	Value 86	Value 87	Value 88	Value 89	Value 90	Value 91	Value 92	Value 93	Value 94	Value 95	Value 96	Value 97	Value 98	Value 99	Value 100	Value 101	Value 102	Value 103	Value 104	Value 105	Value 106	Value 107	Value 108	Value 109	Value 110	Value 111	Value 112	Value 113	Value 114	Value 115	Value 116	Value 117	Value 118	Value 119	Value 120	Value 121	Value 122	Value 123	Value 124	Value 125	Value 126	Value 127	Value 128	Value 129	Value 130	Value 131	Value 132	Value 133	Value 134	Value 135	Value 136	Value 137	Value 138	Value 139	Value 140	Value 141	Value 142	Value 143	Value 144	Value 145	Value 146	Value 147	Value 148	Value 149	Value 150	Value 151	Value 152	Value 153	Value 154	Value 155	Value 156	Value 157	Value 158	Value 159	Value 160	Value 161	Value 162	Value 163	Value 164	Value 165	Value 166	Value 167	Value 168	Value 169	Value 170	Value 171	Value 172	Value 173	Value 174	Value 175	Value 176	Value 177	Value 178	Value 179	Value 180	Value 181	Value 182	Value 183	Value 184	Value 185	Value 186	Value 187	Value 188	Value 189	Value 190	Value 191	Value 192	Value 193	Value 194	Value 195	Value 196	Value 197	Value 198	Value 199	Value 200	Value 201	Value 202	Value 203	Value 204	Value 205	Value 206	Value 207	Value 208	Value 209	Value 210	Value 211	Value 212	Value 213	Value 214	Value 215	Value 216	Value 217	Value 218	Value 219	Value 220	Value 221	Value 222	Value 223	Value 224	Value 225	Value 226	Value 227	Value 228	Value 229	Value 230	Value 231	Value 232	Value 233	Value 234	Value 235	Value 236	Value 237	Value 238	Value 239	Value 240	Value 241	Value 242	Value 243	Value 244	Value 245	Value 246	Value 247	Value 248	Value 249	Value 250	Value 251	Value 252	Value 253	Value 254	Value 255	Value 256	Value 257	Value 258	Value 259	Value 260	Value 261	Value 262	Value 263	Value 264	Value 265	Value 266	Value 267	Value 268	Value 269	Value 270	Value 271	Value 272	Value 273	Value 274	Value 275	Value 276	Value 277	Value 278	Value 279	Value 280	Value 281	Value 282	Value 283	Value 284	Value 285	Value 286	Value 287	Value 288	Value 289	Value 290	Value 291	Value 292	Value 293	Value 294	Value 295	Value 296	Value 297	Value 298	Value 299	Value 300	Value 301	Value 302	Value 303	Value 304	Value 305	Value 306	Value 307	Value 308	Value 309	Value 310	Value 311	Value 312	Value 313	Value 314	Value 315	Value 316	Value 317	Value 318	Value 319	Value 320	Value 321	Value 322	Value 323	Value 324	Value 325	Value 326	Value 327	Value 328	Value 329	Value 330	Value 331	Value 332	Value 333	Value 334	Value 335	Value 336	Value 337	Value 338	Value 339	Value 340	Value 341	Value 342	Value 343	Value 344	Value 345	Value 346	Value 347	Value 348	Value 349	Value 350	Value 351	Value 352	Value 353	Value 354	Value 355	Value 356	Value 357	Value 358	Value 359	Value 360	Value 361	Value 362	Value 363	Value 364	Value 365	Value 366	Value 367	Value 368	Value 369	Value 370	Value 371	Value 372	Value 373	Value 374	Value 375	Value 376	Value 377	Value 378	Value 379	Value 380	Value 381	Value 382	Value 383	Value 384	Value 385	Value 386	Value 387	Value 388	Value 389	Value 390	Value 391	Value 392	Value 393	Value 394	Value 395	Value 396	Value 397	Value 398	Value 399	Value 400	Value 401	Value 402	Value 403	Value 404	Value 405	Value 406	Value 407	Value 408	Value 409	Value 410	Value 411	Value 412	Value 413	Value 414	Value 415	Value 416	Value 417	Value 418	Value 419	Value 420	Value 421	Value 422	Value 423	Value 424	Value 425	Value 426	Value 427	Value 428	Value 429	Value 430	Value 431	Value 432	Value 433	Value 434	Value 435	Value 436	Value 437	Value 438	Value 439	Value 440	Value 441	Value 442	Value 443	Value 444	Value 445	Value 446	Value 447	Value 448	Value 449	Value 450	Value 451	Value 452	Value 453	Value 454	Value 455	Value 456	Value 457	Value 458	Value 459	Value 460	Value 461	Value 462	Value 463	Value 464	Value 465	Value 466	Value 467	Value 468	Value 469	Value 470	Value 471	Value 472	Value 473	Value 474	Value 475	Value 476	Value 477	Value 478	Value 479	Value 480	Value 481	Value 482	Value 483	Value 484	Value 485	Value 486	Value 487	Value 488	Value 489	Value 490	Value 491	Value 492	Value 493	Value 494	Value 495	Value 496	Value 497	Value 498	Value 499	Value 500	Value 501	Value 502	Value 503	Value 504	Value 505	Value 506	Value 507	Value 508	Value 509	Value 510	Value 511	Value 512	Value 513	Value 514	Value 515	Value 516	Value 517	Value 518	Value 519	Value 520	Value 521	Value 522	Value 523	Value 524	Value 525	Value 526	Value 527	Value 528	Value 529	Value 530	Value 531	Value 532	Value 533	Value 534	Value 535	Value 536	Value 537	Value 538	Value 539	Value 540	Value 541	Value 542	Value 543	Value 544	Value 545	Value 546	Value 547	Value 548	Value 549	Value 550	Value 551	Value 552	Value 553	Value 554	Value 555	Value 556	Value 557	Value 558	Value 559	Value 560	Value 561	Value 562	Value 563	Value 564	Value 565	Value 566	Value 567	Value 568	Value 569	Value 570	Value 571	Value 572	Value 573	Value 574	Value 575	Value 576	Value 577	Value 578	Value 579	Value 580	Value 581	Value 582	Value 583	Value 584	Value 585	Value 586	Value 587	Value 588	Value 589	Value 590	Value 591	Value 592	Value 593	Value 594	Value 595	Value 596	Value 597	Value 598	Value 599	Value 600	Value 601	Value 602	Value 603	Value 604	Value 605	Value 606	Value 607	Value 608	Value 609	Value 610	Value 611	Value 612	Value 613	Value 614	Value 615	Value 616	Value 617	Value 618	Value 619	Value 620	Value 621	Value 622	Value 623	Value 624	Value 625	Value 626	Value 627	Value 628	Value 629	Value 630	Value 631	Value 632	Value 633	Value 634	Value 635	Value 636	Value 637	Value 638	Value 639	Value 640	Value 641	Value 642	Value 643	Value 644	Value 645	Value 646	Value 647	Value 648	Value 649	Value 650	Value 651	Value 652	Value 653	Value 654	Value 655	Value 656	Value 657	Value 658	Value 659	Value 660	Value 661	Value 662	Value 663	Value 664	Value 665	Value 666	Value 667	Value 668	Value 669	Value 670	Value 671	Value 672	Value 673	Value 674	Value 675	Value 676	Value 677	Value 678	Value 679	Value 680	Value 681	Value 682	Value 683	Value 684	Value 685	Value 686	Value 687	Value 688	Value 689	Value 690	Value 691	Value 692	Value 693	Value 694	Value 695	Value 696	Value 697	Value 698	Value 699	Value 700	Value 701	Value 702	Value 703	Value 704	Value 705	Value 706	Value 707	Value 708	Value 709	Value 710	Value 711	Value 712	Value 713	Value 714	Value 715	Value 716	Value 717	Value 718	Value 719	Value 720	Value 721	Value 722	Value 723	Value 724	Value 725	Value 726	Value 727	Value 728	Value 729	Value 730	Value 731	Value 732	Value 733	Value 734	Value 735	Value 736	Value 737	Value 738	Value 739	Value 740	Value 741	Value 742	Value 743	Value 744	Value 745	Value 746	Value 747	Value 748	Value 749	Value 750	Value 751	Value 752	Value 753	Value 754	Value 755	Value 756	Value 757	Value 758	Value 759	Value 760	Value 761	Value 762	Value 763	Value 764	Value 765	Value 766	Value 767	Value 768	Value 769	Value 770	Value 771	Value 772	Value 773	Value 774	Value 775	Value 776	Value 777	Value 778	Value 779	Value 780	Value 781	Value 782	Value 783	Value 784	Value 785	Value 786	Value 787	Value 788	Value 789	Value 790	Value 791	Value 792	Value 793	Value 794	Value 795	Value 796	Value 797	Value 798	Value 799	Value 800	Value 801	Value 802	Value 803	Value 804	Value 805	Value 806	Value 807	Value 808	Value 809	Value 810	Value 811	Value 812	Value 813	Value 814	Value 815	Value 816	Value 817	Value 818	Value 819	Value 820	Value 821	Value 822	Value 823	Value 824	Value 825	Value 826	Value 827	Value 828	Value 829	Value 830	Value 831	Value 832	Value 833	Value 834	Value 835	Value 836	Value 837	Value 838	Value 839	Value 840	Value 841	Value 842	Value 843	Value 844	Value 845	Value 846	Value 847	Value 848	Value 849	Value 850	Value 851	Value 852	Value 853	Value 854	Value 855	Value 856	Value 857	Value 858	Value 859	Value 860	Value 861	Value 862	Value 863	Value 864	Value 865	Value 866	Value 867	Value 868	Value 869	Value 870	Value 871	Value 872	Value 873	Value 874	Value 875	Value 876	Value 877	Value 878	Value 879	Value 880	Value 881	Value 882	Value 883	Value 884	Value 885	Value 886	Value 887	Value 888	Value 889	Value 890	Value 891	Value 892	Value 893	Value 894	Value 895	Value 896	Value 897	Value 898	Value 899	Value 900	Value 901	Value 902	Value 903	Value 904	Value 905	Value 906	Value 907	Value 908	Value 909	Value 910	Value 911	Value 912	Value 913	Value 914	Value 915	Value 916	Value 917	Value 918	Value 919	Value 920	Value 921	Value 922	Value 923	Value 924	Value 925	Value 926	Value 927	Value 928	Value 929	Value 930	Value 931	Value 932	Value 933	Value 934	Value 935	Value 936	Value 937	Value 938	Value 939	Value 940	Value 941	Value 942	Value 943	Value 944	Value 945	Value 946	Value 947	Value 948	Value 949	Value 950	Value 951	Value 952	Value 953	Value 954	Value 955	Value 956	Value 957	Value 958	Value 959	Value 960	Value 961	Value 962	Value 963	Value 964	Value 965	Value 966	Value 967	Value 968	Value 969	Value 970	Value 971	Value 972	Value 973	Value 974	Value 975	Value 976	Value 977	Value 978	Value 979	Value 980	Value 981	Value 982	Value 983	Value 984	Value 985	Value 986	Value 987	Value 988	Value 989	Value 990	Value 991	Value 992	Value 993	Value 994	Value 995	Value 996	Value 997	Value 998	Value 999	Value 1000
-----------	---------	---------	---------	---------	---------	---------	---------	---------	---------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	------------

Misma información que puede ser abierta con alguna hoja de cálculo.

Figura 44. Contenido del archivo data.csv (LibreOffice Calc).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2023-04-30 23:49:34.577Z	20.16	9.85	-4.99	29.69	0	4.8	214.01	55.62	45.55	0.06															
2	2023-04-30 23:49:36.102Z	20.22	10.49	-4.88	29.69	0	10	207.29	51.17	41.29	0.05															
3	2023-05-01 00:09:07.713Z	28.75	10.57	-5.11	29.69	0	11.71	215.06	50.27	41.65	0.05															
4	2023-05-01 00:09:11.713Z	28.75	10.57	-5.11	29.69	0	11.71	215.06	50.27	41.65	0.05															
5	2023-05-01 00:29:41.120Z	28.76	10.92	-4.7	29.7	0	7.87	212.29	43.47	38.65	0.05															
6	2023-05-01 00:39:42.607Z	28.85	10.75	-5.12	29.69	0	6.61	221.49	37.57	33.85	0.04															
7	2023-05-01 00:40:00.592Z	28.85	10.75	-5.12	29.69	0	6.61	221.49	37.57	33.85	0.04															
8	2023-05-01 00:59:45.934Z	28.19	11.5	-4.45	29.69	0	9.23	217.66	18	13	0.02															
9	2023-05-01 01:09:47.532Z	27.94	11.85	-4.57	29.7	0	5.56	215.04	6	1	0															
10	2023-05-01 01:29:50.502Z	25.67	22.01	2.98	29.71	0	1.29	75.51	0	0	0															
11	2023-05-01 01:39:05.047Z	24.71	23.86	2.71	29.72	0	9.76	32.83	0	0	0															
12	2023-05-01 01:40:00.989Z	23.13	28.12	3.71	29.77	0	0.54	45.52	0	0	0															
13	2023-05-01 01:59:56.242Z	23.13	28.12	3.71	29.77	0	0.54	45.52	0	0	0															
14	2023-05-01 01:59:56.242Z	23.13	28.12	3.71	29.77	0	7.82	32.83	0	0	0															
15	2023-05-01 02:09:58.348Z	23.82	26.51	3.46	29.73	0	1.06	34.89	0	0	0															
16	2023-05-01 02:10:00.999Z	23.99	27.59	3.47	29.75	0	0.51	49.19	0	0	0															
17	2023-05-01 02:30:02.076Z	23.99	27.59	3.47	29.75	0	1.11	49.19	0	0	0															
18	2023-05-01 02:40:03.646Z	22.86	28.51	3.73	29.76	0	5.19	47.24	0	0	0															
19	2023-05-01 02:50:04.777Z	23.05	27.51	3.86	29.76	0	1.84	37.74	0	0	0															
20	2023-05-01 03:09:58.662Z	23.13	28.12	3.71	29.77	0	0.54	45.52	0	0	0															
21	2023-05-01 03:10:09.566Z	23.13	28.12	3.71	29.77	0	0.54	45.52	0	0	0															
22	2023-05-01 03:20:30.573Z	23.12	28.15	3.71	29.76	0	0.07	45.5	0	0	0															
23	2023-05-01 03:40:11.072Z	22.68	28.12	3.66	29.79	0	0.03	192	0	0	0															
24	2023-05-01 03:40:11.307Z	22.68	28.12	3.66	29.79	0	0.03	192	0	0	0															
25	2023-05-01 03:50:18.682Z	22.3	29.56	3.76	29.79	0	1.13	229.57	0	0	0															
26	2023-05-01 03:50:18.682Z	22.3	29.56	3.76	29.79	0	1.13	229.57	0	0	0															
27	2023-05-01 04:10:23.702Z	22.04	29.8	3.59	29.82	0	0.02	112.17	0	0	0															
28	2023-05-01 04:20:24.742Z	21.92	29.89	3.53	29.82	0	0.01	192	0	0	0															
29	2023-05-01 04:30:25.772Z	21.92	29.89	3.53	29.82	0	0.01	192	0	0	0															
30	2023-05-01 04:40:33.476Z	21.51	30.39	3.4	29.82	0	0.08	138.51	0	0	0															
31	2023-05-01 04:50:35.282Z	21.26	30.95	3.45	29.82	0	0	0	0	0	0															
32	2023-05-01 05:00:36.282Z	21.26	30.95	3.45	29.82	0	0	0	0	0	0															
33	2023-05-01 05:10:44.512Z	20.93	31.85	3.42	29.82	0	0	0	0	0	0															
34	2023-05-01 05:20:44.642Z	20.72	31.85	3.31	29.82	0	0	0	0	0	0															
35	2023-05-01 05:30:44.642Z	20.72	31.85	3.31	29.82	0	0	0	0	0	0															
36	2023-05-01 05:41:02.665Z	19.75	33.16	3.08	29.82	0	0	0	0	0	0															
37	2023-05-01 05:51:04.062Z	19.95	33.06	3.18	29.82	0	0	0	0	0	0															
38	2023-05-01 06:01:04.062Z	19.95	33.06	3.18	29.82	0	0	0	0	0	0															
39	2023-05-01 06:11:07.072Z	19.57	33.46	3.26	29.8	0	0	0	0	0	0															
40	2023-05-01 06:21:08.702Z	19.44	32.8	2.68	29.8	0	0	0	0	0	0															
41	2023-05-01 06:31:11.402Z	19.44	32.8	2.68	29.8	0	0	0	0	0	0															
42	2023-05-01 06:41:11.402Z	18.93	33.89	2.69	29.8	0	0	0	0	0	0															
43	2023-05-01 06:51:12.602Z	18.88	33.59	2.52	29.79	0	0	0	0	0	0															

Finalmente se comprueba que los datos se encuentran en la base de datos de PocketBase.

Figura 45. Datos en la colección station (PocketBase).

Collections	/ station	...	+ New record
<input type="text"/> Search term or filter like created > "2022-01-01"...			
<input type="checkbox"/> o- id <input type="checkbox"/> time ↑ <input type="checkbox"/> # temperature <input type="checkbox"/> # humidity <input type="checkbox"/> # dewpoint <input type="checkbox"/> # pressure <input type="checkbox"/> # rainFall <input type="checkbox"/> # windSpeed <input type="checkbox"/> # windDirection <input type="checkbox"/> # uva <input type="checkbox"/> # uvb <input type="checkbox"/> # uvindex <input type="checkbox"/> created ...			
<input type="checkbox"/>	jk653p10bal3	2023-04-30 23:59:36 UTC	30.16
<input type="checkbox"/>	zflz7m5dtudtw5	2023-04-30 23:59:36 UTC	29.22
<input type="checkbox"/>	tobi021pymp7syh	2023-05-01 00:09:37 UTC	28.75

Una vez que se ha comprobado que la estación meteorológica funciona correctamente, se puede proceder a realizar la instalación de la estación meteorológica en el lugar donde se llevarán a cabo las mediciones.

Donde se realizará una prueba de su funcionamiento durante una duración aproxima de 15 días, para posteriormente realizar una revisión de los datos obtenidos.

Figura 46. Instalación de la estación meteorológica.



15.2.4. Servidor

A continuación, se presenta el desarrollo del servidor que recolectará los datos de los distintos servicios de información meteorológica, para posteriormente almacenarlos en una base de datos y compararlos con los datos obtenidos por la estación meteorológica.

Nota. La siguiente información tiende a ser más técnica, por lo que se recomienda tener conocimientos básicos de programación y de los servicios de información meteorológica.

15.2.4.1. Guías, Manuales y Tutoriales

Los siguientes manuales y guías fueron empleados para el desarrollo del servidor.

- <https://www.aviationweather.gov/dataserver> (AWC, 2013b)
- <https://docs.python.org/3/> (Python Foundation, s. f.)
- The Linux Command Line (Shotts, 2012)
- Wicked Cool Shell Scripts (Perry & Taylor, 2016)
- Python One Liners (Mayer, 2020)
- <http://www.faqs.org/docs/air/tsshell.html> (Shell Foundation, s. f.)
- <https://open-meteo.com/en/docs/> (Open Meteo, s. f.)
- <https://openweathermap.org/api> (OpenWeather, s. f.)
- <https://developer.accuweather.com/apis> (AccuWeather, s. f.)
- <https://jsoncrack.com/> (JSON Crack, s. f.)
- <https://crontab.guru/> (Crontab Guru, s. f.)
- <https://pocketbase.io/docs/> (PocketBase, s. f.)
- <https://ubuntu.com/server/docs> (Ubuntu, s. f.)
- <https://www.oracle.com/mx/cloud/free/> (Oracle, s. f.)
- <https://www.cloudflare.com/products/tunnel/> (Cloudflare, s. f.)

15.2.4.2. Características

Nota. Para mayor información se recomienda investigar acerca de temas como *Servidores, Linux, Oracle Cloud, etc.*

El presente código se está ejecutando en un servidor de Oracle Cloud, el cual cuenta con las siguientes características.

- **CPU.** 1/8 OCPU.
- **RAM.** 1 GB.
- **Almacenamiento.** 50 GB HDD.
- **Sistema Operativo.** Ubuntu 22.04 LTS. (Ubuntu, s. f.)

Si bien las características del servidor son muy limitadas, esto se debe a que se implementó en el plan de servidores gratuitos de Oracle (Oracle, s. f.) , el cual es suficiente para el desarrollo del prototipo.

15.2.4.3. Servicios

Los servicios a implementar y de los cuales se recolectarán los datos son los siguientes.

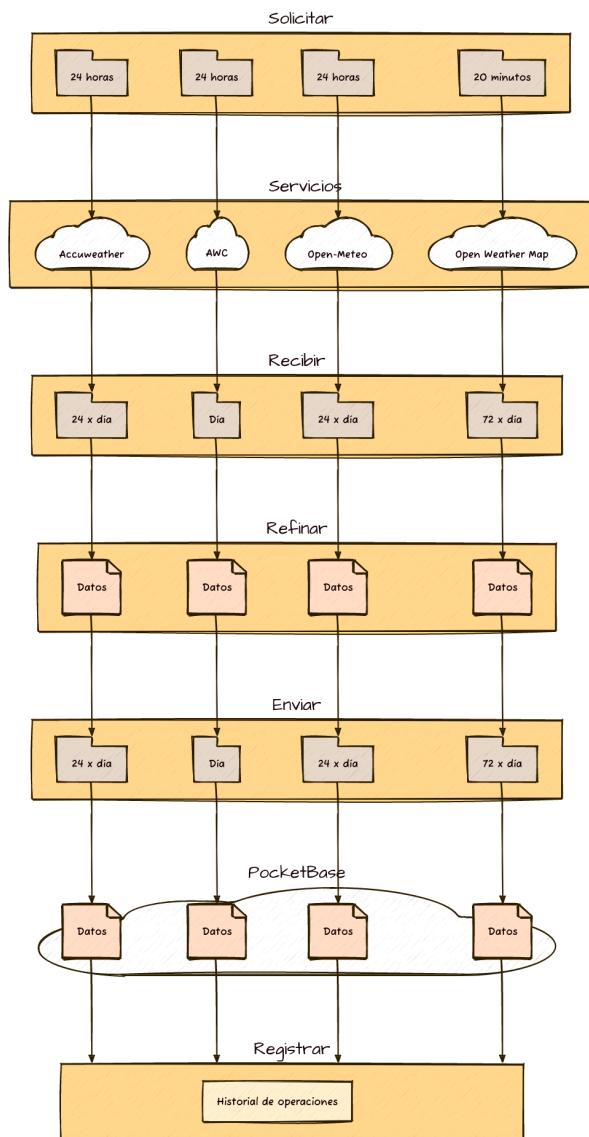
- AccuWeather.
- AWC.
- Open-Meteo.
- OpenWeatherMap.

Cada uno de los servicios listados cuenta con su propio protocolo de comunicación, por lo que se debe de realizar una implementación para cada uno de ellos. Al igual que la información la recolectan en distintos lapsos de tiempo, por lo que se debe de realizar una implementación para cada caso.

15.2.4.4. Diagrama de Flujo

A continuación, se muestra el siguiente diagrama de flujo, el cual se puede observar en el siguiente [Diagrama 3](#) hecho con el lenguaje de programación D2 (D2 Lang, s. f.) .

Diagrama 3. Flujo del prototipo del servidor.



15.2.4.5. Código

La siguiente sección es de las que a simple vista puede parecer simple de realizar, pero en realidad es la más compleja, debido a que se debe de realizar una implementación para cada servicio de información meteorológica, por lo que se debe de realizar una investigación de cada uno de ellos. Ya que si bien transformamos la información recolectada a un mismo formato no toda se implementa de la misma forma.

Nota. Al lector se le ha ahorrado la lectura necesaria, pero es necesario que tenga presente que existe una documentación extensa que fue empleada para la realización de cada implementación, y que, aunque se ha tratado de simplificar la información, es posible que se presente información demasiado técnica.

Nota. El código utilizado se encuentra en los anexos, pero también recibirá actualizaciones en el siguiente repositorio:

https://github.com/Osares10/Weather-Services_PocketBase

El desarrollo de la siguiente programación se implementa en base a los siguientes enunciados.

A computer is like a Swiss Army knife that you can configure for countless tasks...

(Mayer, 2020)

Bash remains a staple tool for anyone working on Unix-like server or workstations...

(Perry & Taylor, 2016)

Bash. Es un lenguaje de programación de scripts que se utiliza en sistemas operativos Unix y Linux.

15.2.4.5.1. AccuWeather

Para la implementación de AccuWeather se emplea el siguiente código.

accuweather.py Sección 18.3.3.1

En la primera parte definimos las librerías a emplear.

```
.py  
import requests  
import json  
from datetime import datetime
```

.py Es la extensión de los archivos de Python. **Python** siendo el lenguaje de programación de alto nivel, interpretado y de propósito general que emplearemos para la comunicación con los servicios de información meteorológica.

Se define la API de AccuWeather, la cual se emplea para realizar las peticiones, entre los parámetros se encuentra la llave de la API, la cual se debe de obtener de la página de AccuWeather (AccuWeather, s. f.), y la llave de la ubicación, la cual se obtiene de la página de AccuWeather (AccuWeather, s. f.).

```
.py  
# Define API endpoint and parameters  
location_key = "YOUR-LOCATION-KEY" # Location key  
api_key = "YOUR-API-KEY" # AccuWeather API key  
pocketbase_api_url = "https://custom.domain/api/collections/collection_name/records" # Pocketbase API endpoint
```

Se definen los `headers` de la petición, los cuales son necesarios para realizar la petición a PocketBase (PocketBase, s. f.).

```
.py  
# Define headers  
headers = {  
    "Content-Type": "application/json",  
}
```

Se crea una función para obtener el tiempo actual, el cual se emplea para imprimir el tiempo de ejecución del script.

```
.py  
  
# Define function to get current time  
def execution_time():  
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")  
  
# Print execution time  
print(execution_time() + " - Executing script...")
```

Script. Es un archivo de texto plano que contiene una serie de instrucciones que se ejecutan secuencialmente. En este caso, el script se emplea para realizar la recolección de datos de AccuWeather y almacenarlos en PocketBase.

Se realiza la petición a AccuWeather (AccuWeather, s. f.), la cual se almacena en la variable `response`.

```
.py  
  
# Make API request  
print(execution_time() + " - Making request to AccuWeather...")  
response = requests.get("http://dataservice.accuweather.com/currentconditions/v1/" + location_key + "/historical/24?apikey=" + api_key +  
"&language=en-us&details=true&metric=true")  
data = json.loads(response.text)  
data = json.dumps(data, indent=4)
```

Se verifica que la petición haya sido exitosa, en caso de serlo se procede a realizar el parseo de la información, la cual se almacena en la variable `new_data`, la cual se emplea para realizar la petición a PocketBase (PocketBase, s. f.).

```
.py  
  
# Check if response was successful  
if response.status_code == 200:  
    print(execution_time() + " - Request to AccuWeather successful!")  
    # Parse JSON response  
    for i in range(0, len(json.loads(data))):
```

```
new_data = {
    "time": datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.000Z'),
    "temperature": json.loads(data)[i]['Temperature']['Metric']
['Value'],
    "realFeelTemperature": json.loads(data)[i]
['RealFeelTemperature']['Metric']['Value'],
    "realFeelTemperatureShade": json.loads(data)[i]
['RealFeelTemperatureShade']['Metric']['Value'],
    "relativeHumidity": json.loads(data)[i]['RelativeHumidity'],
    "indoorRelativeHumidity": json.loads(data)[i]
['IndoorRelativeHumidity'],
    "dewPoint": json.loads(data)[i]['DewPoint']['Metric']
['Value'],
    "windDirection": json.loads(data)[i]['Wind']['Direction']
['Degrees'],
    "windSpeed": json.loads(data)[i]['Wind']['Speed']['Metric']
['Value'],
    "uvIndex": json.loads(data)[i]['UVIndex'],
    "visibility": json.loads(data)[i]['Visibility']['Imperial']
['Value'],
    "pressure": json.loads(data)[i]['Pressure']['Imperial']
['Value'],
    "apparentTemperature": json.loads(data)[i]
['ApparentTemperature']['Metric']['Value'],
    "precipitation": json.loads(data)[i]['Precip1hr']['Metric']
['Value'],
}
# Send data to PocketBase API
print(execution_time() + " - Sending data to PocketBase with date
and time: " + new_data['time'])

response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Data sent to PocketBase
successfully!")
else:
    print(execution_time() + " - Request to PocketBase failed!
with status code: " + str(response.status_code))

else:
    print(execution_time() + "Request to AccuWeather failed! with response
code: " + str(response.status_code))
```

Los datos obtenidos de AccuWeather se almacenan en PocketBase.

15.2.4.5.2. AWC

Se implementa el siguiente código para obtener los datos de AWC.

awc.py Sección 18.3.3.2

En la primera parte definimos las librerías a emplear, entre ellas se encuentra `requests`, la cual se emplea para realizar las peticiones a AWC, `csv`, para leer el archivo CSV, `json`, para parsear la información obtenida de AWC, y `datetime`, para obtener la fecha y hora actual.

Librerías. (Libraries) Son conjuntos de funciones y procedimientos que se pueden utilizar para realizar tareas específicas, como la lectura de archivos CSV, la realización de peticiones HTTP, el parseo de información JSON, entre otros.

```
# Import libraries
import requests
import csv
import json
from datetime import datetime
```

.py

Nota. Para evitar repetir código, solo se explicarán las implementaciones específicas de cada API, para ver el código completo se puede consultar la [Sección 18.3.3.2](#).

Las constantes y variables globales se definen de la siguiente manera. `API_URL` es la URL de la API de AWC, `POCKETBASE_API_URL` es la URL de la API de PocketBase, `data_type` es el tipo de datos a obtener, `airport_code` es el código del aeropuerto, `hours_before_now` es el número de horas antes de ahora para obtener los datos, y `output_format` es el formato de salida de los datos.

.py

```
# Define constants
API_URL = "https://www.aviationweather.gov/adds/dataserver_current/
httpparam" # Aviation Weather Center API URL
POCKETBASE_API_URL = "https://custom.domain/api/collections/collection_
name/records" # PocketBase API URL

# Define variables
data_type = "metars"
airport_code = "XXXX" # Airport code
hours_before_now = "24" # Number of hours before now to fetch data for
output_format = "csv"
```

Se define la función `fetch_csv_data`, la cual recibe como parámetros la URL de la API y los parámetros de la petición. Se realiza la petición a AWC, y se verifica que la petición haya sido exitosa, en caso de serlo se decodifica el contenido de la respuesta, se parsea la información y se retorna.

Parsear. (*Parsing*) Es el proceso de analizar una cadena de símbolos, ya sea en lenguaje natural, lenguajes informáticos o estructuras de datos, conforme a las reglas de una gramática formal. El término análisis proviene del latín *pars*, que significa parte.

.py

```
# Define function to fetch CSV data
def fetch_csv_data(api_url, parameters):
    print(execution_time() + " - Making request to Aviation Weather
Center...")
    # Make the API request
    response = requests.get(api_url, params=parameters)
    # Check for successful request
    if response.status_code == 200:
        # Decode the CSV content
        content = response.content.decode("utf-8")
        # Parse the CSV data and return it
        csv_data = list(csv.reader(content.splitlines(), delimiter=","))
        return csv_data
    else:
```

```
    print(execution_time() + " - Request to Aviation Weather Center  
failed! with status code: " + str(response.status_code))
```

A continuación, se enlista el proceso de refinamiento de los datos obtenidos de AWC. Primero se obtiene la información de AWC, luego se verifica que haya resultados en la información obtenida, en caso de haberlos se parsea la información y se convierte a JSON, y finalmente se envía la información a PocketBase.

```
.py

# Call the function to fetch the CSV data
csv_data = fetch_csv_data(API_URL, parameters)

# Check if there are results in the CSV data
if len(csv_data) <= 6:
    print(execution_time() + " - No results found in CSV data received
from Aviation Weather Center")
else:
    print(execution_time() + " - CSV data received from Aviation Weather
Center successfully!")
    # Parse the remaining rows of the CSV data and convert to JSON
    print(execution_time() + " - Parsing CSV data and converting it to
JSON...")
    for row in csv_data[6:]: # Skip the first 6 rows
        # Get the headers from the first row of the CSV data
        headers = csv_data[5]
        # Create a dictionary with the desired keys and values
        data = {
            "raw_text": row[headers.index("raw_text")],
            "station_id": airport_code,
            "observation_time":
                datetime.strptime(row[headers.index("observation_time")], "%Y-%m-%dT%H:%M:
%SZ").strftime("%Y-%m-%d %H:%M:%S.%000Z"),
            "temp_c": float(row[headers.index("temp_c")]),
            "dewpoint_c": float(row[headers.index("dewpoint_c")]) if
row[headers.index("dewpoint_c")] else 99, # Set to 99 if dewpoint_c is
empty
            "wind_dir_degrees":
                int(row[headers.index("wind_dir_degrees")]) if
row[headers.index("wind_dir_degrees")] else 0,
            "wind_speed_kt": int(row[headers.index("wind_speed_kt")]) if
row[headers.index("wind_speed_kt")] else 0,
            "altim_in_hg": float(row[headers.index("altim_in_hg")]),
            "corrected": bool(row[headers.index("corrected")]),
            "precip_in": float(row[headers.index("precip_in")]) if
```

```
row[headers.index("precip_in")] else 0,  
    "metar_type": row[headers.index("metar_type")],  
}
```

15.2.4.5.3. Open-Meteo

Para Open-Meteo se trabaja sobre el siguiente código.

open-meteo.py Sección 18.3.3.3

Las librerías que se importan son las siguientes. `requests` se utiliza para realizar peticiones HTTP, `json` para trabajar con JSON, `datetime` para trabajar con fechas y horas, y `timedelta` para trabajar con intervalos de tiempo.

```
.py  
import requests  
import json  
from datetime import datetime  
from datetime import date  
from datetime import timedelta
```

El intervalo de tiempo que se utiliza para obtener los datos es de 24 horas, por lo que se obtiene la fecha y hora actual, y se le resta 24 horas para obtener la fecha y hora de inicio del intervalo de tiempo.

```
.py  
# Get previous day  
previous_day = (datetime.utcnow() - timedelta(days=1)).strftime("%Y-%m-%d")
```

Se definen los parámetros de la petición a Open-Meteo. Se define la latitud y longitud de la ubicación, las variables que se desean obtener, la unidad de la velocidad del viento, la unidad de la precipitación, el número de días de pronóstico, y la fecha y hora de inicio y fin del intervalo de tiempo.

```
.py  
params = {  
    "latitude": XX.XX, # Latitude of the location  
    "longitude": XXX.XX, # Longitude of the location  
    "hourly": "temperature_2m, relativehumidity_2m, dewpoint_2m,
```

```
apparent_temperature, rain, pressure_msl, surface_pressure, visibility,  
windspeed_10m, windspeed_80m, windspeed_120m, windspeed_180m,  
winddirection_10m, winddirection_80m, winddirection_120m,  
winddirection_180m, temperature_80m, temperature_120m, temperature_180m,  
uv_index",  
    "windspeed_unit": "kn", # Unit of the wind speed  
    "precipitation_unit": "inch", # Unit of the precipitation  
    "forecast_days": 1,  
    "start_date": previous_day, # Start date of the forecast  
    "end_date": previous_day # End date of the forecast, for one day, use  
the same date as start_date  
}
```

Para la extracción de la información por horas se implementa el siguiente código.

```
.py  
  
# Send data to Pocketbase API one hour at a time  
for hour in range(0, len(json.loads(data)['hourly']['time'])):  
    new_data = {  
        "time": datetime.strptime(json.loads(data)['hourly']['time'][hour], "%Y-%m-%dT%H:%M").strftime("%Y-%m-%d %H:%M:00.000Z"),  
        "temperature_2m": json.loads(data)['hourly']['temperature_2m'][hour],  
        "relativehumidity_2m": json.loads(data)['hourly']['relativehumidity_2m'][hour],  
        "dewpoint_2m": json.loads(data)['hourly']['dewpoint_2m'][hour],  
        "apparent_temperature": json.loads(data)['hourly']['apparent_temperature'][hour],  
        "rain": json.loads(data)['hourly']['rain'][hour],  
        "pressure_msl": json.loads(data)['hourly']['pressure_msl'][hour],  
        "surface_pressure": json.loads(data)['hourly']['surface_pressure'][hour],  
        "windspeed_10m": json.loads(data)['hourly']['windspeed_10m'][hour],  
        "windspeed_80m": json.loads(data)['hourly']['windspeed_80m'][hour],  
        "windspeed_120m": json.loads(data)['hourly']['windspeed_120m'][hour],  
        "windspeed_180m": json.loads(data)['hourly']['windspeed_180m'][hour],  
        "winddirection_10m": json.loads(data)['hourly']['winddirection_10m'][hour],
```

```
        "winddirection_80m": json.loads(data)[ 'hourly' ]
[ 'winddirection_80m'][hour],
        "winddirection_120m": json.loads(data)[ 'hourly' ]
[ 'winddirection_120m'][hour],
        "winddirection_180m": json.loads(data)[ 'hourly' ]
[ 'winddirection_180m'][hour],
        "temperature_80m": json.loads(data)[ 'hourly' ]
[ 'temperature_80m'][hour],
        "temperature_120m": json.loads(data)[ 'hourly' ]
[ 'temperature_120m'][hour],
        "temperature_180m": json.loads(data)[ 'hourly' ]
[ 'temperature_180m'][hour],
        "uv_index": json.loads(data)[ 'hourly'][ 'uv_index'][hour]
    }
```

15.2.4.5.4. OpenWeatherMap

En este script se trabaja con intervalos de tiempo puesto que las peticiones se realizarán cada 20 minutos a diferencia de los demás servicios donde las peticiones se realizan cada 24 horas. Esto debido a que en OpenWeather no se puede obtener la información de un intervalo de tiempo específico, sino que se obtiene la información de un intervalo de tiempo que va desde la fecha y hora actual hasta 3 horas antes de la fecha y hora actual.

openweathermap.py Sección 18.3.3.4

Los parámetros que se definen son los siguientes. Se define la API key de OpenWeatherMap, la latitud y longitud de la ubicación, la url de la API de OpenWeatherMap, la url de la API de Pocketbase, y los headers de la petición a Pocketbase.

```
.py

# Define API endpoint and parameters
api_key = "YOUR-API-KEY" # OpenWeatherMap API key
latitude = XX.XX # Latitude of the location
longitude = XXX.XX # Longitude of the location
url = "https://api.openweathermap.org/data/2.5/weather?lat=" +
str(latitude) + "&lon=" + str(longitude) + "&appid=" + api_key
pocketbase_api_url = "https://custom.domain/collections/collection_name/
records" # Pocketbase API endpoint
headers = {
    "Content-Type": "application/json",
}
```

La estructura de la petición a OpenWeatherMap es la siguiente. Se envía la petición a la API de OpenWeatherMap y se obtiene la respuesta en formato JSON. Se parsea la respuesta y se obtienen los datos que se desean obtener. Se crea un diccionario con los datos que se obtuvieron de la respuesta y se envía la petición a Pocketbase.

```
.py

# Parse JSON response
data = json.loads(response.text)

data = json.dumps(data, indent=4)

new_data = {
    "time": datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.000Z'),
    "temperature": json.loads(data)[ 'main'][ 'temp'],
    "feels_like": json.loads(data)[ 'main'][ 'feels_like'],
    "pressure": json.loads(data)[ 'main'][ 'pressure'],
    "humidity": json.loads(data)[ 'main'][ 'humidity'],
    "wind_speed": json.loads(data)[ 'wind'][ 'speed'],
    "wind_direction": json.loads(data)[ 'wind'][ 'deg'],
}
```

15.2.4.6. Pruebas

La ejecución de las pruebas se llevará a cabo mediante la ejecución de múltiples scripts a distintos tiempos, establecidos en el `crontab` del servidor. Los scripts se ejecutarán cada 24 horas, a las `00:00` horas, y cada 20 minutos, a partir de las `00:00` horas hasta las `23:40` horas dependiendo el caso.

El crontab se configuro con la ayuda de Crontab Guru (Crontab Guru, s. f.) y se estableció de la siguiente manera.

Crontab. Es un archivo de configuración que contiene las tareas programadas para ejecutarse en un sistema Unix o Unix-like. Las tareas programadas se conocen como cron jobs.

```
0      18 * * * python3 /home/ubuntu/weather/accuweather.py >> /home/
ubuntu/weather/accuweather.log
0      18 * * * python3 /home/ubuntu/weather/awc.py >> /home/ubuntu/
weather/awc.log
0      18 * * * python3 /home/ubuntu/weather/open-meteo.py >> /home/ubuntu/
```

```
weather/open-meteo.log
*/20 * * * * python3 /home/ubuntu/weather/openweathermap.py >> /home/
ubuntu/weather/openweathermap.log
```

Se puede observar que todos los scripts registran la información en un archivo `.log` para poder verificar que se ejecutaron correctamente.

.log Es un archivo que registra los eventos que ocurren en un sistema, como la ejecución de un script, la conexión a una base de datos, etc.

A continuación, se presentan los resultados obtenidos en la ejecución de los scripts.

Referencia 6. Resultados de correcta ejecución de `accuweather.py`.

```
2023-04-30 00:00:01 - Executing script...
2023-04-30 00:00:02 - Making request to AccuWeather...
2023-04-30 00:00:10 - Request to AccuWeather successful!
2023-04-30 00:00:14 - Sending data to PocketBase with date and time:
2023-04-30 00:55:00.000Z
2023-04-30 00:00:21 - Data sent to PocketBase successfully!
...
```

Figura 47. Datos en la colección accuweather (PocketBase).

id	time	temperature	realFeelTemperature	relativeHumidity	indoorRelativeHumidity	dewPoint	windDirection	windSpeed	timestamp
un0yutdv0c4plc5	2023-04-30 00:55:00 UTC	21.9	19.6	19.4	28	28	2.7	68	31.2
q9ym76drityekc4	2023-04-30 01:55:00 UTC	19.8	17.6	17.6	33	33	3.2	68	25.6
trrz8v7syuyf1	2023-04-30 02:55:00 UTC	18.1	16.4	16.4	38	33	3.4	90	16.9
91ogahi52nc9cbr	2023-04-30 03:55:00 UTC	16.4	15.2	15.2	42	33	3.5	113	11.7
pfpymat43f02qb	2023-04-30 04:55:00 UTC	14.8	13.6	13.6	46	33	3.4	90	10.9
f8lijdhcgen3dwua	2023-04-30 05:55:00 UTC	13.2	12.1	12.1	51	33	3.2	45	9.8
qiafidmq7fhz63	2023-04-30 06:55:00 UTC	12.1	11.1	11.1	53	32	2.9	45	9.2
jtsliik3y6vu7wuv	2023-04-30 07:55:00 UTC	11.1	10.3	10.3	56	31	2.6	23	8.5
ubh5iq3qeg35ln	2023-04-30 08:55:00 UTC	10.5	9.9	9.9	63	34	3.8	23	7.9
tse8dp2rhayb1q	2023-04-30 09:55:00 UTC	9.8	9.4	9.4	66	34	3.7	0	7.3

Referencia 7. Resultados de correcta ejecución de awc.py .

```
2023-04-30 00:00:01 - Executing script...
2023-04-30 00:00:02 - Making request to Aviation Weather Center...
2023-04-30 00:00:10 - Request to Aviation Weather Center successful!
2023-04-30 00:00:14 - CSV data received from Aviation Weather Center
successfully!
2023-04-30 00:00:14 - Parsing CSV data and converting it to JSON...
2023-04-30 00:00:14 - Sending data to PocketBase with date and time:
2023-04-30 00:56:00.000Z
2023-04-30 00:00:18 - Data sent to PocketBase successfully!
...
```

Figura 48. Datos en la colección awc (*PocketBase*).

	id	raw_text	station_id	observation_time	# temp_c	# dewpoint_c	# wind_dir_degrees	# wind_speed_kt	# altim_in_hg	...
<input type="checkbox"/>	bug9xe3hi5at7zf	MMLO 300056Z 24005KT 10SM SKC 26/M12 A...	MMLO	2023-04-30 00:56:00 UTC	26	-12	240	5	30.159449	Fal →
<input type="checkbox"/>	6sonif4f5l42m9o	MMLO 300147Z 00000KT 8SM SKC 24/M12 A...	MMLO	2023-04-30 01:47:00 UTC	24	-12	0	0	30.171259	Fal →
<input type="checkbox"/>	75asolhuwemxor8	MMLO 300253Z 02010KT 8SM SKC 22/03 A30...	MMLO	2023-04-30 02:53:00 UTC	22	3	20	10	30.200787	Fal →
<input type="checkbox"/>	fs4l11xs2yc0gr	MMLO 300342Z 01007KT 8SM SKC 22/02 A3022	MMLO	2023-04-30 03:42:00 UTC	22	2	10	7	30.221457	Fal →
<input type="checkbox"/>	q366yy3y3t7n0m9	MMLO 300451Z 00000KT 8SM SKC 20/01 A3027	MMLO	2023-04-30 04:51:00 UTC	20	1	0	0	30.268702	Fal →
<input type="checkbox"/>	8sp1py0fc168te	MMLO 300549Z 00000KT 8SM SKC 19/00 A30...	MMLO	2023-04-30 05:49:00 UTC	19	0	0	0	30.268702	Fal →
<input type="checkbox"/>	wk2f8pay8ptjau	MMLO 300804Z RTD 00000KT 8SM SKC 16/02...	MMLO	2023-04-30 08:04:00 UTC	16	2	0	0	30.250984	Fal →
<input type="checkbox"/>	zpe4zjm5lrpdosl	MMLO 301023Z RTD 03008KT 8SM SKC 14/04...	MMLO	2023-04-30 10:23:00 UTC	14	4	30	8	30.239174	Fal →
<input type="checkbox"/>	ugqeb835vxdeegr	MMLO 301055Z 02007KT 8SM SKC 14/04 A3025	MMLO	2023-04-30 10:55:00 UTC	14	4	20	7	30.250984	Fal →
<input type="checkbox"/>	yovn33uogm5dv9d	MMLO 301154Z 03007KT 8SM SKC 14/04 A30...	MMLO	2023-04-30 11:54:00 UTC	14	4	30	7	30.259842	Fal →
<input type="checkbox"/>	f1ozeb784dmhyhkm	MMLO 301247Z 02008KT 10SM SKC 14/04 A3...	MMLO	2023-04-30 12:47:00 UTC	14	4	20	8	30.268702	Fal →
<input type="checkbox"/>	py7k7kkllioms	MMLO 301540Z 00000KT 95M SKC 22/01 A30...	MMLO	2023-04-30 15:40:00 UTC	22	1	0	0	30.31004	Fal →

Referencia 49. Resultados de correcta ejecución de `open-meteo.py`.

```
2023-04-30 00:00:00 - Executing script...
2023-04-30 00:00:01 - Making request to Open-Meteo...
2023-04-30 00:00:02 - Request to Open-Meteo successful!
2023-04-30 00:00:03 - Sending data to PocketBase with date and time:
2023-04-30 00:00:00.000Z
2023-04-30 00:00:04 - Data sent to PocketBase successfully!
...
```

Figura 50. Datos en la colección `open_meteo` (*PocketBase*).

o id	time ↑	# temperature_2m	# relativehumidity_2m	# dewpoint_2m	# apparent_temperat...	# pressure_msl	# surface_pressure	# windspeed_10m	# wind	...
<input type="checkbox"/>	ae61p94lj4mzb6x 2023-04-30 00:00:00 UTC	21.9	34	5.2	17.9	1012.9	799.3	10.7	13.8	→
<input type="checkbox"/>	4fpk33yeyd2tgaf 2023-04-30 01:00:00 UTC	19.3	38	4.8	16.1	1015	799.3	7.2	9.2	→
<input type="checkbox"/>	d9vt00ew419y0f5 2023-04-30 02:00:00 UTC	16.7	46	4.9	14.1	1016.9	799.1	5.1	6.7	→
<input type="checkbox"/>	cifsjpo1tlyfed 2023-04-30 03:00:00 UTC	14.9	52	5	12.1	1018.2	798.9	5.8	7.9	→
<input type="checkbox"/>	s83zb672wwwa4jf 2023-04-30 04:00:00 UTC	13.7	55	4.9	10.7	1019.6	799.2	6.3	9	→
<input type="checkbox"/>	jh8ts0lnh5tt 2023-04-30 05:00:00 UTC	12.8	59	4.9	10.4	1020.6	799.4	4.3	6.6	→
<input type="checkbox"/>	ecljn8dwxsq3j46 2023-04-30 06:00:00 UTC	11.6	64	5	9.9	1019.6	797.8	1.8	3.5	→
<input type="checkbox"/>	lpwzn490gkw4vp 2023-04-30 07:00:00 UTC	11.3	66	5.2	9.9	1019.4	797.4	0.8	1.6	→
<input type="checkbox"/>	53tizj0gd7aj4k 2023-04-30 08:00:00 UTC	10.9	68	5.2	9.7	1018.8	796.7	0.3	1.5	→
<input type="checkbox"/>	d20po8rj7fl13v 2023-04-30 09:00:00 UTC	10.3	70	5.1	9.1	1018.7	796.3	0.2	1.4	→

Referencia 8. Resultados de correcta ejecución de `openweathermap.py`.

```
2023-04-30 23:40:00 - Executing script...
2023-04-30 23:40:00 - Making request to OpenWeatherMap...
2023-04-30 23:40:01 - Request to OpenWeatherMap successful!
2023-04-30 23:40:01 - Sending data to PocketBase with date and time:
2023-04-30 00:57:00.000Z
2023-04-30 23:40:02 - Data sent to PocketBase successfully!
...
```

Figura 51. Datos en la colección openweathermap (PocketBase).

Collections / openweathermap ⚙️ 🕒

Search term or filter like created > "2022-01-01"...

	id	time	temperature	feels_like	pressure	humidity	wind_speed	wind_direction	created	updated	...
<input type="checkbox"/>	2r5xj727yf3ved	2023-04-30 23:40:02 UTC	299.47	299.47	1008	12	4.1	237	2023-04-30 23:40:02 UTC	2023-04-30 23:40:02 UTC	→
<input type="checkbox"/>	frk19v8ttq0asnib	2023-05-01 00:00:02 UTC	299.65	299.65	1008	11	5.23	239	2023-05-01 00:00:05 UTC	2023-05-01 00:00:05 UTC	→
<input type="checkbox"/>	cqf3pymvnoghygs	2023-05-01 00:20:02 UTC	299.47	299.47	1008	12	4.1	237	2023-05-01 00:20:02 UTC	2023-05-01 00:20:02 UTC	→
<input type="checkbox"/>	tm9z0ij3ut1kpyn	2023-05-01 00:40:02 UTC	296.3	295.17	1010	19	3.15	29	2023-05-01 00:40:02 UTC	2023-05-01 00:40:02 UTC	→
<input type="checkbox"/>	964rq90h4cr0tnp	2023-05-01 01:00:02 UTC	296.3	295.17	1010	19	3.15	29	2023-05-01 01:00:03 UTC	2023-05-01 01:00:03 UTC	→
<input type="checkbox"/>	1ezfd3rv1n7dov6	2023-05-01 01:20:02 UTC	296.3	295.17	1010	19	3.15	29	2023-05-01 01:20:02 UTC	2023-05-01 01:20:02 UTC	→
<input type="checkbox"/>	m9t3s3y1h1wbfz	2023-05-01 01:40:02 UTC	291.07	290.09	1013	45	6.02	51	2023-05-01 01:40:02 UTC	2023-05-01 01:40:02 UTC	→
<input type="checkbox"/>	nztw4e2adeiqede	2023-05-01 02:00:05 UTC	290.82	289.9	1014	48	7.07	49	2023-05-01 02:00:06 UTC	2023-05-01 02:00:06 UTC	→
<input type="checkbox"/>	6snq0z75uv96uvx	2023-05-01 02:20:02 UTC	291.07	290.09	1013	45	6.02	51	2023-05-01 02:20:02 UTC	2023-05-01 02:20:02 UTC	→
<input type="checkbox"/>	syav4zrva0bzv2p	2023-05-01 02:40:02 UTC	290.02	289.09	1014	51	2.1	86	2023-05-01 02:40:02 UTC	2023-05-01 02:40:02 UTC	→
<input type="checkbox"/>	nbkh72pmdpjnzrk	2023-05-01 03:00:06 UTC	290.01	289.11	1014	52	2.08	83	2023-05-01 03:00:06 UTC	2023-05-01 03:00:06 UTC	→
<input type="checkbox"/>	8hs4v45fvxt4kmf	2023-05-01 03:20:01 UTC	290.02	289.09	1014	51	2.1	86	2023-05-01 03:20:01 UTC	2023-05-01 03:20:01 UTC	→

15.3. Resultados

- **Accuweather.** 384 registros, 2023-04-30 00:55:00 a 2023-05-15 22:55:00.
- **Aviation Weather Center.** 328 registros, 2023-04-30 00:55:00 a 2023-05-15 23:40:00.
- **BME680.** 2053 registros, 2023-04-30 23:50:07 a 2023-05-16 00:35:00.
- **Open-Meteo.** 384 registros, 2023-04-30 00:00:00 a 2023-05-15 23:00:00.
- **OpenWeatherMap.** 1082 registros, 2023-04-30 23:40:02 a 2023-05-16 00:00:02.
- **Estación.** 2122 (*PocketBase*), 2134 (*microSD*) registros, 2023-04-30 23:49:34 a 2023-05-16 00:00:36.

Tabla 2. Porcentajes de datos obtenidos.

Registro	Intervalo	Esperados	Obtenidos	Porcentaje
Accuweather	01:00	384	384	100%
AWC	01:00	384	328	85.42%
BME680	00:10	2165	2028	93.67%
Open-Meteo	01:00	384	384	100%
OpenWeather	00:20	1082	1082	100%
Estación (PB)	00:10	2164	2122	98.06%
Estación (SD)	00:10	2164	2134	98.61%

Nota. Recordar que las siguientes estadísticas son de datos obtenidos en un periodo de 15 días, que aproximadamente van del 1 al 15 de mayo de 2023, y que los tiempos están en formato UTC.

15.4. Análisis de Resultados

Nota. Para la presente sección al lector se le ha ahorrado la lectura necesaria, pero es necesario que tenga presente que existe una documentación extensa que fue empleada para la realización de cada implementación, y que, aunque se ha tratado de simplificar la información, es posible que se presente información demasiado técnica.

El análisis de los resultados se realiza a través de la comparación de los datos obtenidos por cada fuente de información con los datos obtenidos por la estación meteorológica.

Debido a los distintos tiempos de actualización de cada fuente de información, se realiza una comparación de los datos obtenidos en un intervalo de tiempo de 10 minutos, para los datos no existentes en estos períodos se generarán a partir de extrapolación lineal.

15.4.1. Extracción de Datos

Los datos se encuentran en el servidor, pero es necesario recolocarlos en un archivo de CSV para poder realizar el análisis de los mismos.

Para esto se realiza un script en Python que extrae los datos de cada colección de *PocketBase* y los guarda en un archivo CSV.

Nota. El código utilizado se encuentra en los anexos, pero también recibirá actualizaciones en el siguiente repositorio:

<https://github.com/0sares10/PocketBase-to-CSV>

La implementación de este script se realiza en el siguiente código.

raw.py Sección 18.3.4.1

En la primera parte definimos las librerías a emplear.

```
# This script fetches data from PocketBase and writes it to a CSV file  
  
import requests  
import csv
```

Posteriormente definimos las variables a emplear.

```
POCKETBASE_API_URL = "https://domain.name/api/collections/" # Specify the
# PocketBase API URL here
COLLECTION_NAME = "station" # Specify the collection name here
OUTPUT_CSV_FILE = f"{COLLECTION_NAME}.csv" # Generate the output file
# name
```

Definimos la función que nos permitirá obtener los datos de la colección.

```
# Define function to fetch data from PocketBase using pagination
def fetch_pocketbase_data(api_url):
    page = 1
    per_page = 50
    records = []

    while True:
        params = {
            "page": page,
            "perPage": per_page,
            "sort": "-created"
        }

        response = requests.get(api_url, params=params)

        if response.status_code == 200:
            json_response = response.json()
            records += json_response.get("items", [])

            # Check if there are more pages to fetch
            if json_response["page"] == json_response["totalPages"]:
                break

            # Move to the next page
            page += 1
        else:
            print("Request to PocketBase failed with status code:",
            response.status_code)
            break

    return records
```

Construimos la URL de la API de *PocketBase*.

```
.py  
# Construct the PocketBase API URL for the specified collection  
pocketbase_api_url = f"{POCKETBASE_API_URL}{COLLECTION_NAME}/records"
```

Llama a la función para obtener los datos de *PocketBase*.

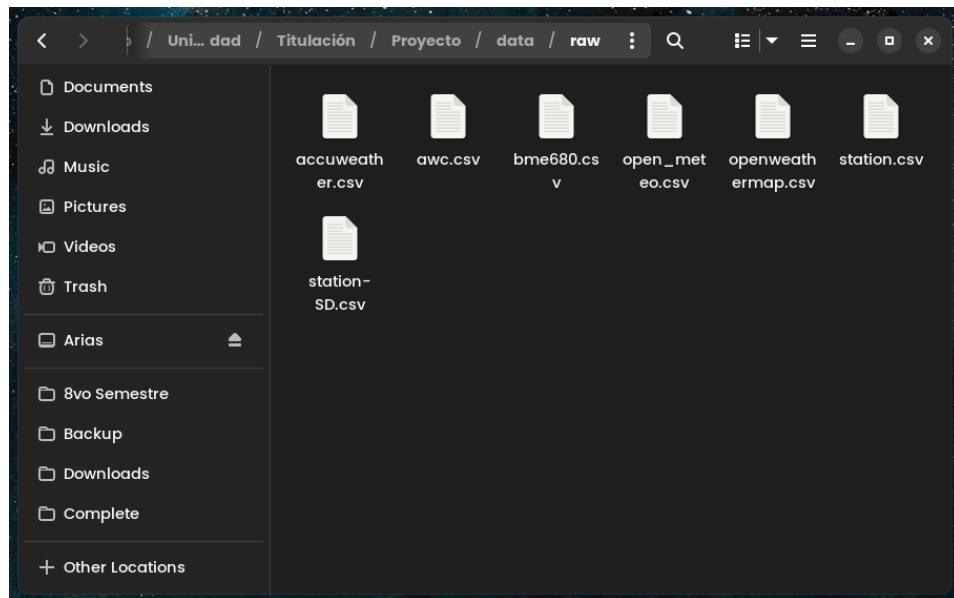
```
.py  
# Call the function to fetch data from PocketBase  
pocketbase_data = fetch_pocketbase_data(pocketbase_api_url)
```

Verifica si se obtuvieron datos de *PocketBase*.

```
.py  
# Check if there are results in the PocketBase data  
if not pocketbase_data:  
    print("No data found in PocketBase")  
else:  
    print("PocketBase data retrieved successfully!")  
    # Extract the keys from the first record to use as CSV headers  
    headers = list(pocketbase_data[0].keys())  
  
    # Open the CSV file for writing  
    with open(OUTPUT_CSV_FILE, mode="w", newline="") as file:  
        writer = csv.DictWriter(file, fieldnames=headers)  
        writer.writeheader()  
  
        # Write each record to the CSV file  
        for record in pocketbase_data:  
            writer.writerow(record)  
  
    print("Data written to CSV file:", OUTPUT_CSV_FILE)
```

A partir de la ejecución del script obtenemos lo siguiente:

Figura 52. Datos de la recolección del script de `raw.py`.



15.4.2. Refinamiento de Datos

Previo al análisis de los datos, se realiza un refinamiento de los mismos, para eliminar los datos que no son necesarios para el análisis, así como para completar los datos faltantes.

Nota. El código utilizado se encuentra en los anexos, pero también recibirá actualizaciones en el siguiente repositorio:

<https://github.com/0sares10/CSV-Data-Refinement>

Para refinar los datos se realizan los siguientes tres procedimientos sobre el archivo CSV.

- Se reescreiben las fechas de los datos, para que se encuentren en formato ISO 8601, aceptado por Excel, de igual forma para las fechas registradas incorrectamente en el campo de `date` se reemplazan por la fecha de `created`.

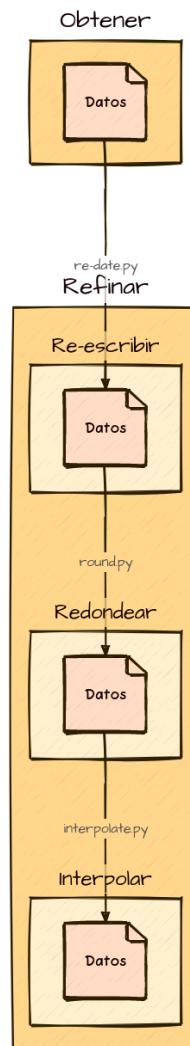
ISO 8601. Formato de fecha y hora internacionalmente aceptado,

`AAAA-MM-DDTHH:MM:SS`.

- Se redondean los tiempos de los datos a 10 minutos, para que coincidan con los tiempos de la estación meteorológica.
- Se rellenan los datos faltantes, a partir de la extrapolación lineal de los datos existentes.

A continuación, se muestran las transformaciones que experimentaran nuestros datos en el siguiente **Diagrama 4** hecho con el lenguaje de programación D2 (D2 Lang, s. f.) .

Diagrama 4. Refinamiento de datos.



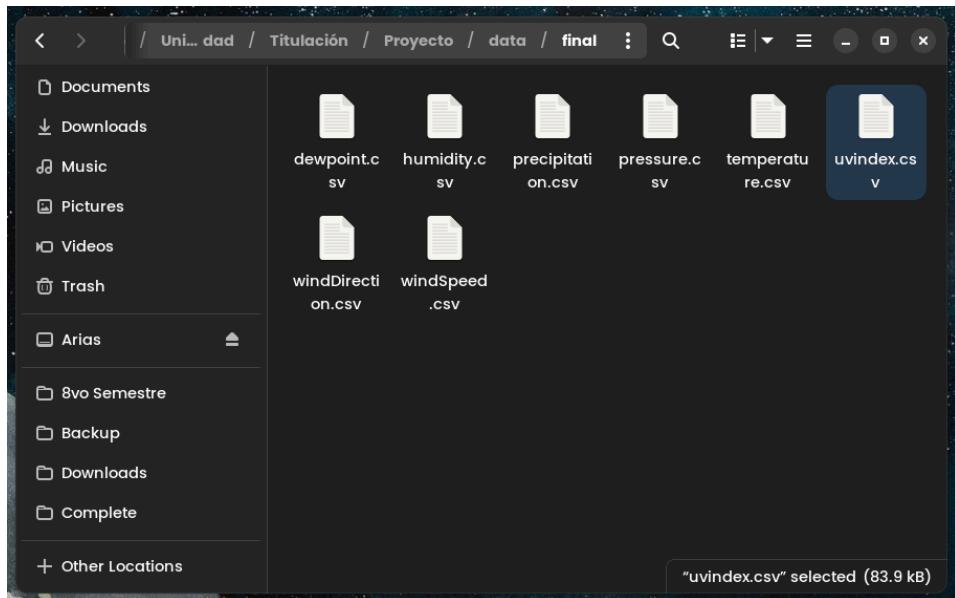
El separar el proceso en múltiples pasos, nos permite realizar el refinamiento de los datos de forma más sencilla, así como realizar pruebas intermedias para verificar que los datos se encuentren en el formato correcto.

Los códigos empleados para cada etapa se pueden encontrar en los siguientes anexos.

- `redate.py` Sección 18.3.5.1
- `round.py` Sección 18.3.5.2
- `interpolate.py` Sección 18.3.5.3 (PyData, s. f.)

A partir de la ejecución de los scripts obtenemos los siguientes archivos.

Figura 53. Resultados del refinamiento de datos.



Estos archivos son los que se utilizarán para el análisis de los datos, ya que cada archivo CSV contiene los datos de una sola variable, pero de todas las colecciones.

Debido al soporte nativo de los archivos CSV por parte de Excel y por la facilidad de uso de los mismos, se decidió utilizar este formato para el análisis de los datos.

Una vez que se han refinado los datos, se procede a realizar el análisis de los mismos.

15.4.3. Análisis de Datos

Para el análisis se desarrolla un archivo Excel, en el cual se realizan los cálculos necesarios para obtener los resultados deseados.

Nota. Se espera que el lector tenga conocimientos básicos de Excel, para poder entender el análisis realizado. Se omite la explicación de las fórmulas utilizadas, ya que se considera que son sencillas de entender. El archivo `Analysis.xlsx` queda a disposición del lector para su análisis a petición.

El archivo Excel se divide en múltiples hojas, cada una para una variable diferente, en las cuales se realiza el análisis de los datos. A continuación, se muestra la hoja de `Temperature` como ejemplo.

Figura 54. Análisis de la variable `Temperature`.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	time	accuweather	awc	bme680	open_meteo	openweathermap	station-SD	station	averag	deviat	minimu	maxim	correct	correct
2	1/5/23 0:00	28.0333333333	28.6363636	26.89	26.5	26.5	28.75	28.75	27.7228	1.02719 -1.22281	1.02719 -1.46584	26.257		
3	1/5/23 0:10	27.7666666667	28.2727273	26.65	25.9166666667	26.41	28.71	28.71	27.4909	1.21913 -1.5742	1.21913 -1.27389	26.217		
4	1/5/23 0:20	27.5	27.9090909	26.47	25.3333333333	26.320000000001	28.76	28.76	27.2932	1.4668 -1.95987	1.4668 -1.02623	26.267		
5	1/5/23 0:30	27.2333333333	27.5454545	26.32	24.75	24.735	28.85	28.85	26.8977	1.95232 -2.16268	1.95232 -0.54071	26.357		
6	1/5/23 0:40	26.9666666667	27.1818182	26.2	24.1666666667	23.15	28.27	28.27	26.315	1.95498 -3.16502	1.95498 -0.53805	25.777		
7	1/5/23 0:50	26.7	26.8181818	26.1	23.5833333333	23.15	28.19	28.19	26.1045	2.0855 -2.9545	2.0855 -0.40753	25.697		
8	1/5/23 1:00	26.1833333333	26.4545455	26.03	23	23.15	27.94	27.94	25.814	2.12602 -2.81398	2.12602 -0.36701	25.447		
9	1/5/23 1:10	25.6666666667	26.0909091	25.96	22.3166666667	23.15	27.75	27.75	25.5263	2.22368 -3.20965	2.22368 -0.26935	25.257		
10	1/5/23 1:20	25.15	25.7272727	25.88	21.6333333333	23.15	25.67	25.67	24.6972	0.97277 -3.0639	1.18277 -1.52026	23.177		
11	1/5/23 1:30	24.6333333333	25.3636364	25.81	20.95	20.535	24.71	24.71	23.816	0.894 -3.281	1.994 -1.59902	22.217		
12	1/5/23 1:40	24.1166666667	25	25.73	20.2666666667	17.92	24.89	24.89	23.259	1.63095 -5.33905	2.47095 -0.86208	22.397		
13	1/5/23 1:50	23.6	24.8333333	25.68	19.5833333333	17.795	23.97	23.97	22.776	1.19405 -4.98095	2.90405 -1.29898	21.477		
14	1/5/23 2:00	23.2	24.6666667	25.64	18.9	17.67	23.82	23.82	22.531	1.28905 -4.86095	3.10905 -1.20398	21.327		
15	1/5/23 2:10	22.8	24.5	25.62	18.65	17.795	23.705	23.705	22.3964	1.30857 -4.60143	3.22357 -1.18446	21.212		
16	1/5/23 2:20	22.4	24.3333333	25.6	18.4	17.92	23.59	23.59	22.2619	1.3281 -4.3419	3.3381 -1.16493	21.097		
17	1/5/23 2:30	22	24.1666667	25.58	18.15	17.395	23.08	23.08	21.9217	1.15833 -4.52667	3.65833 -1.33469	20.587		
18	1/5/23 2:40	21.6	24	25.55	17.9	16.87	22.86	22.86	21.6629	1.19714 -4.79286	3.88714 -1.29588	20.367		
19	1/5/23 2:50	21.2	23.8333333	25.53	17.65	16.865	23.65	23.65	21.7683	1.88167 -4.90333	3.76167 -0.61136	21.157		
20	1/5/23 3:00	20.7666666667	23.6666667	25.5	17.4	16.86	23.71	23.71	21.659	2.05095 -4.79905	3.84095 -0.44208	21.217		
21	1/5/23 3:10	20.3333333333	23.5	25.46	17.0333333333	16.865	23.13	23.13	21.3502	1.77976 -4.48524	4.10976 -0.71327	20.637		
22	1/5/23 3:20	19.9	23.3333333	25.43	16.6666666667	16.87	23.12	23.12	21.2057	1.91429 -4.53905	4.22429 -0.57874	20.627		
23	1/5/23 3:30	19.4666666667	23.1666667	25.41	16.3	16.475	23.09	23.09	20.9998	2.09024 -4.69976	4.41024 -0.40279	20.597		
24	1/5/23 3:40	19.0333333333	23	25.37	15.9333333333	16.08	22.68	22.68	20.6824	1.99762 -4.74905	4.68762 -0.49541	20.187		
25	1/5/23 3:50	18.6	22.6666667	25.31	15.5666666667	16.105	22.3	22.3	20.4069	1.8931 -4.84024	4.9031 -0.59993	19.807		
26	1/5/23 4:00	18.2833333333	22.3333333	25.25	15.2	16.13	22.15	22.15	20.2138	1.93619 -5.01381	5.03619 -0.55684	19.657		
27	1/5/23 4:10	17.9666666667	22	25.16	15.05	16.105	22.04	22.04	20.0517	1.98833 -5.00167	5.10833 -0.50469	19.547		

En la hoja se muestran los datos de la variable `Temperature`, así como los cálculos realizados para obtener los resultados deseados.

Nota. A continuación, se muestra una breve explicación de los cálculos realizados, los cuales fueron realizados para cada variable.

Los cálculos realizados para cada variable son los siguientes.

- **Average**. Promedio de los datos.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n}(x_1 + x_2 + \dots + x_n)$$

- **Deviation**. Desviación simple de los datos.

$$\sigma = x - \bar{x}$$

- **Minimum**. Valor mínimo de la desviación los datos.

$$\min(x) = \min(x_1, x_2, \dots, x_n) - \bar{x}$$

- **Maximum**. Valor máximo de la desviación de los datos.

$$\max(x) = \max(x_1, x_2, \dots, x_n) - \bar{x}$$

- **Correction Factor**. Factor de corrección de los datos, calculado a partir de la desviación simple.

$$C_F = \frac{1}{n} \sum_{i=1}^n \sigma = \frac{1}{n}(\sigma_1 + \sigma_2 + \dots + \sigma_n)$$

- **Correction**. Corrección de los datos, calculada a partir de la desviación simple.

$$C = \sigma - C_F$$

- **Corrected**. Datos corregidos, a partir de la corrección calculada.

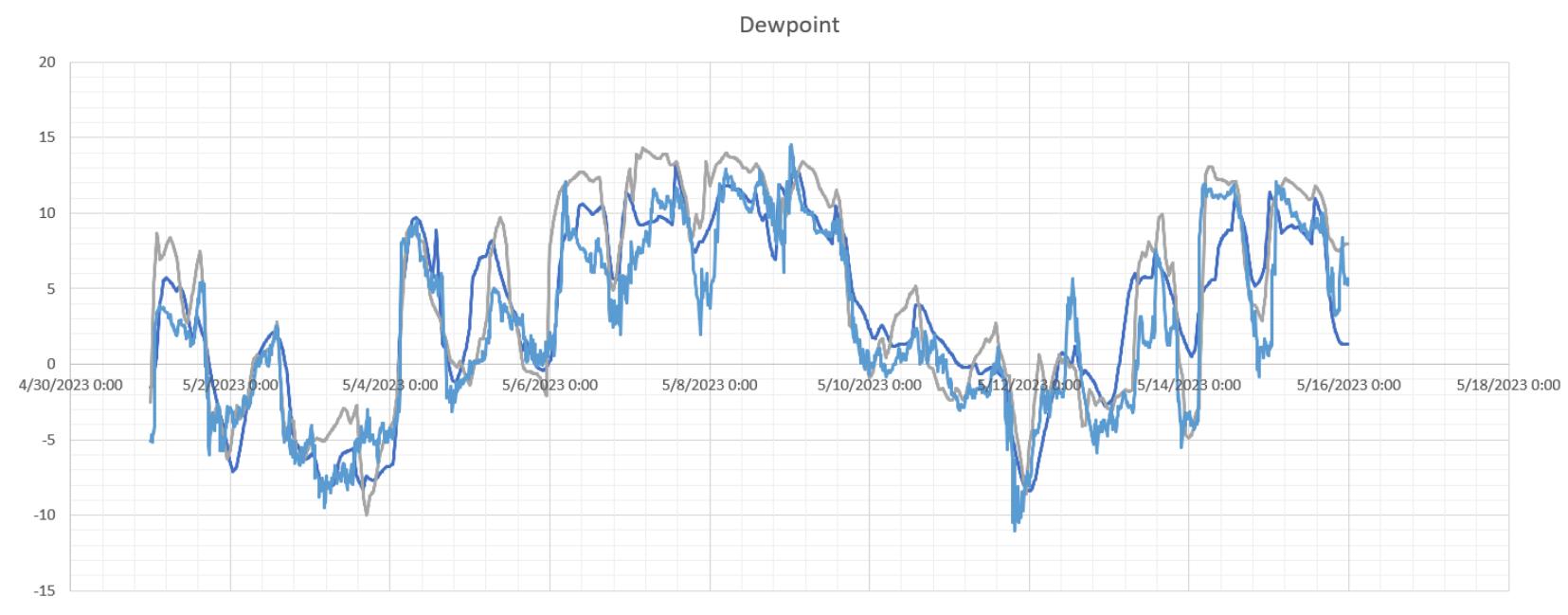
$$x_C = x - C_F$$

A partir de estos cálculos podemos obtener las siguientes gráficas y datos para cada variable.

15.4.3.1. Dewpoint

Para el punto de rocío se obtienen un factor de corrección de -0.6359°C .

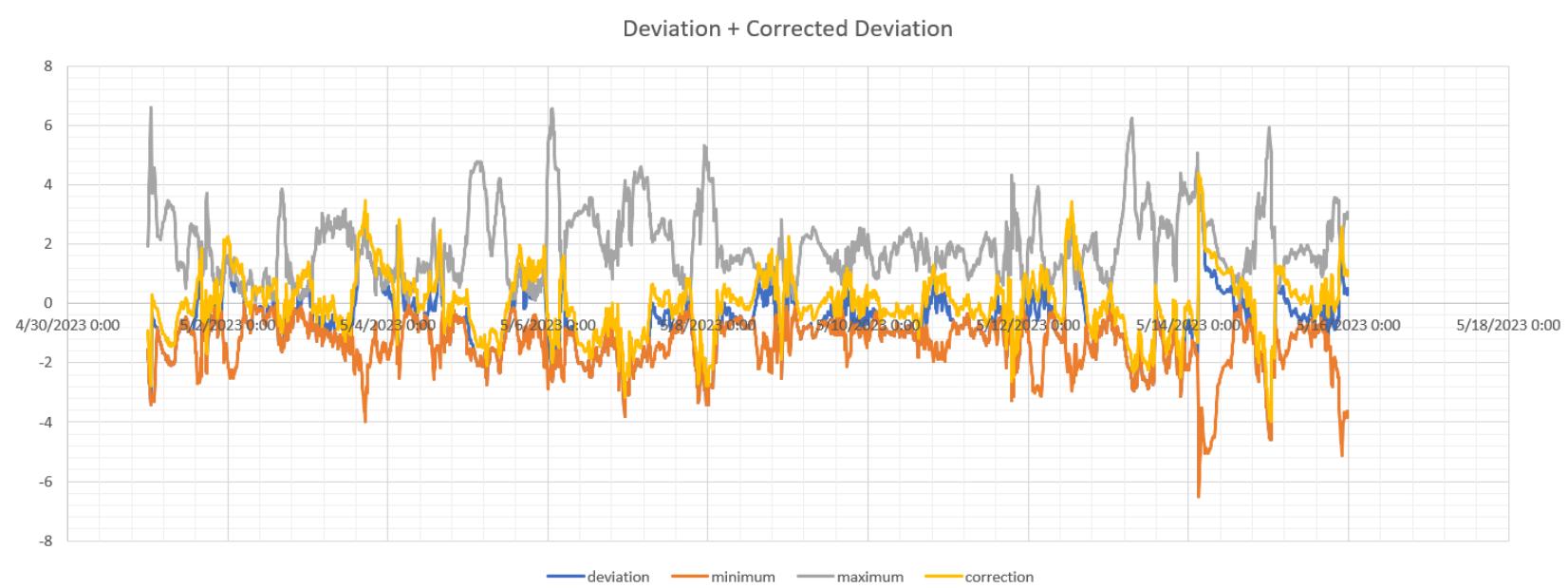
Gráfica 1. Punto de rocío.



Factor de corrección. Valor utilizado para ajustar una medición a un valor de referencia.

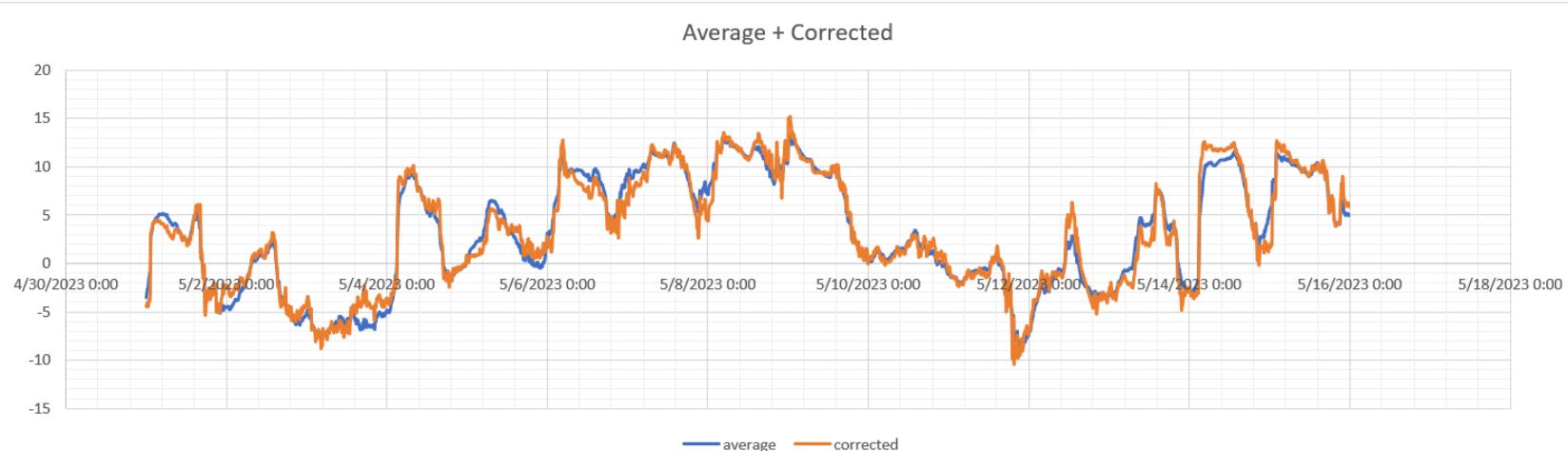
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 2. Desviación del punto de rocío.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

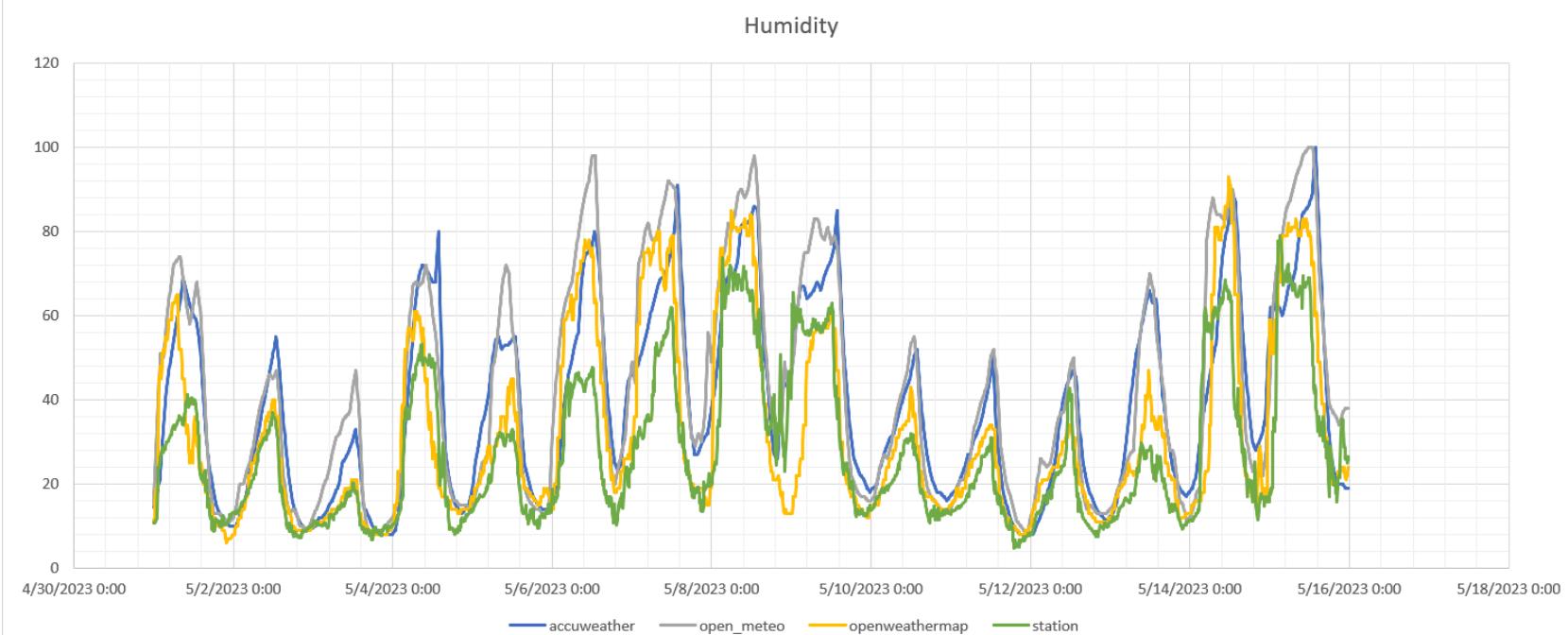
Gráfica 3. Corrección comparada con el promedio del punto de rocío.



15.4.3.2. Humidity

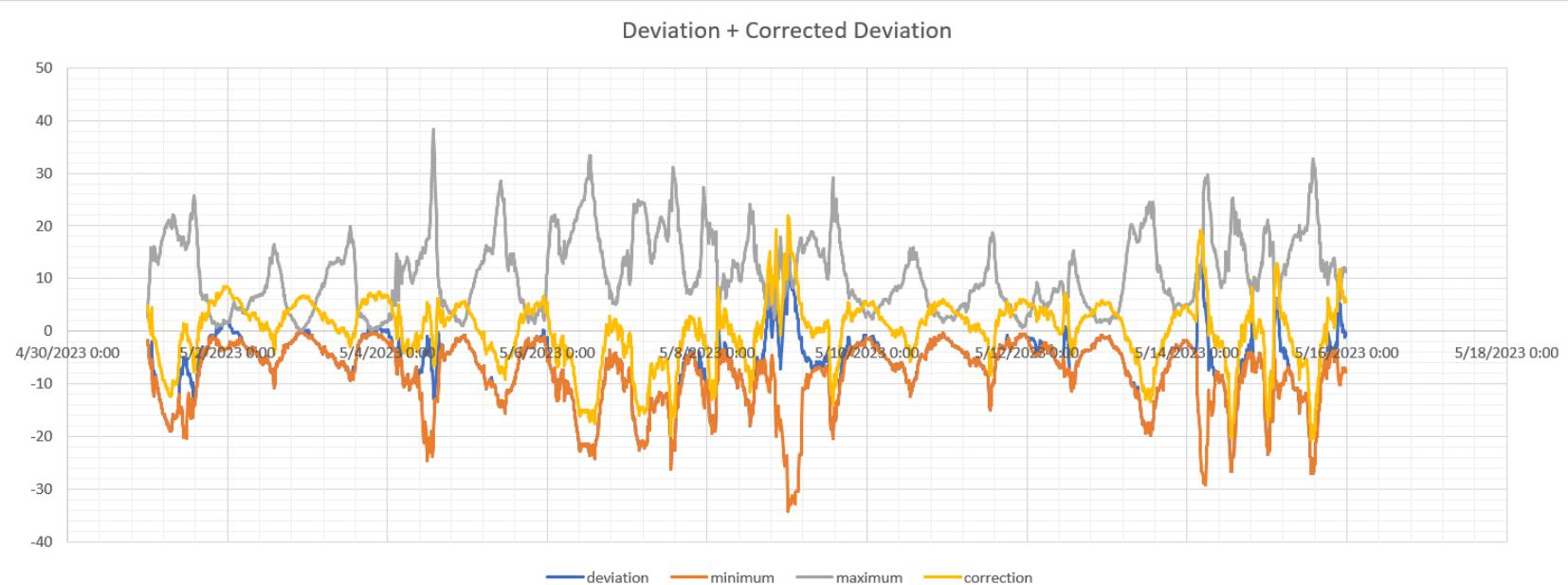
Para la humedad se obtienen un factor de corrección de -6.482 %.

Gráfica 4. Humedad.



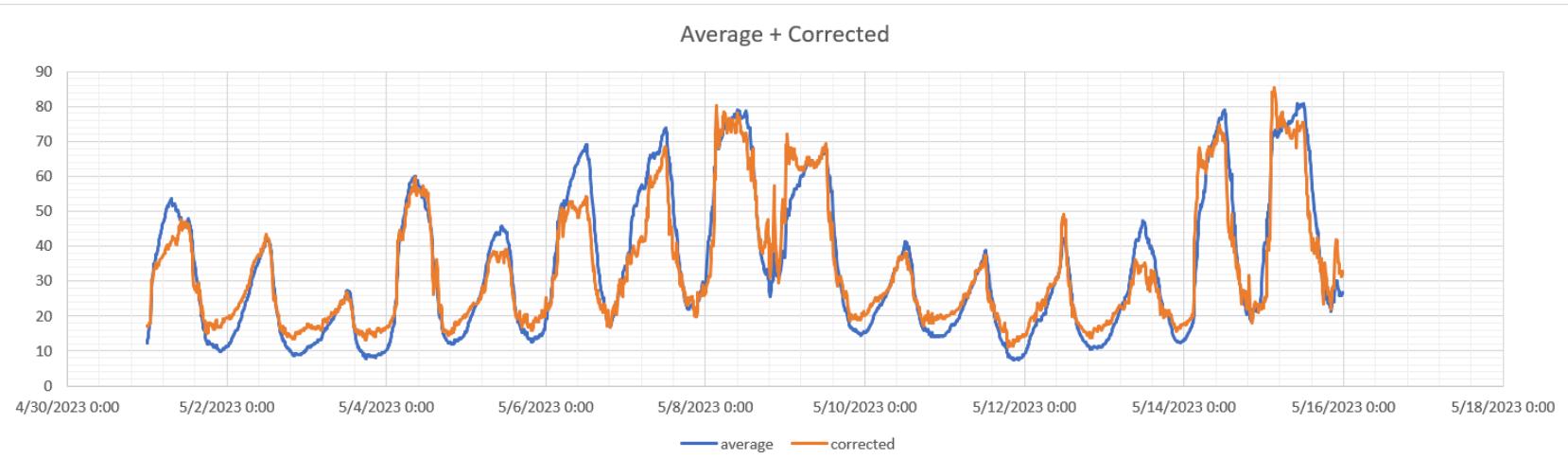
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 5. Desviación de la humedad.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

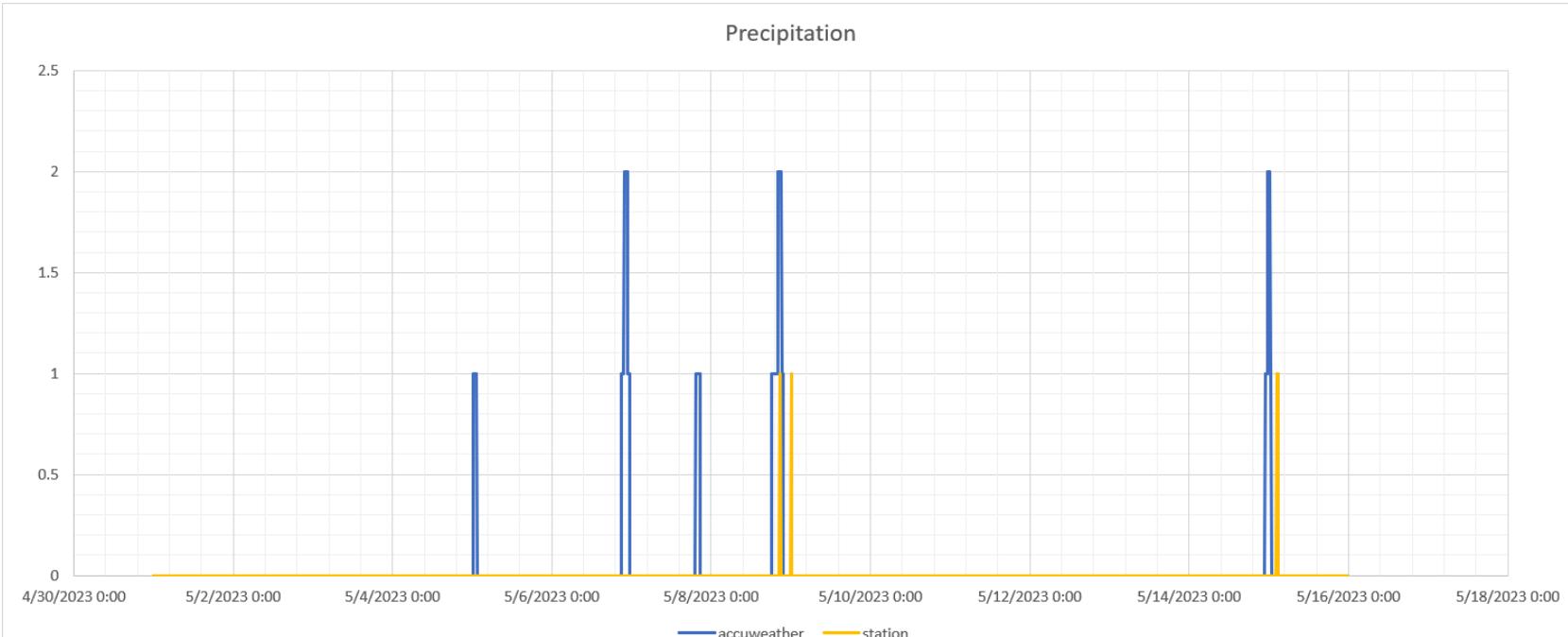
Gráfica 6. Corrección comparada con el promedio de la humedad.



15.4.3.3. Precipitación

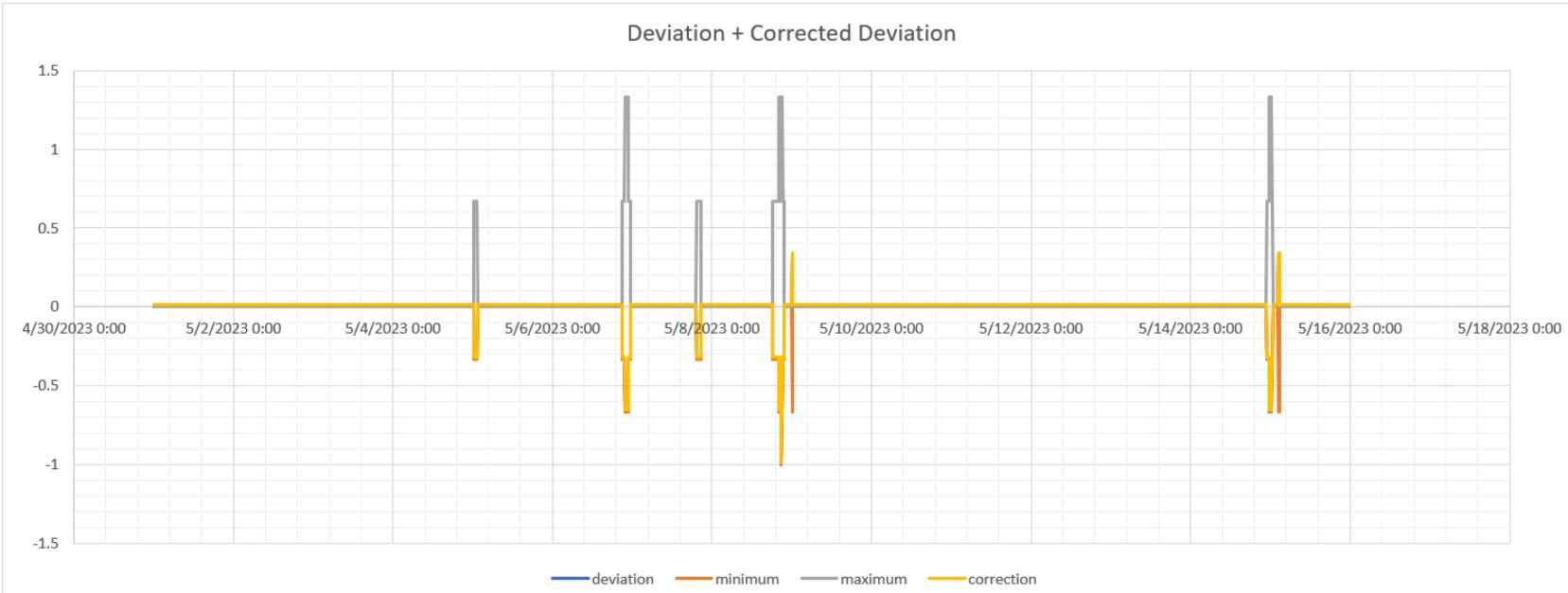
Para la precipitación se obtienen un factor de corrección de -0.0122 in.

Gráfica 7. Precipitación.



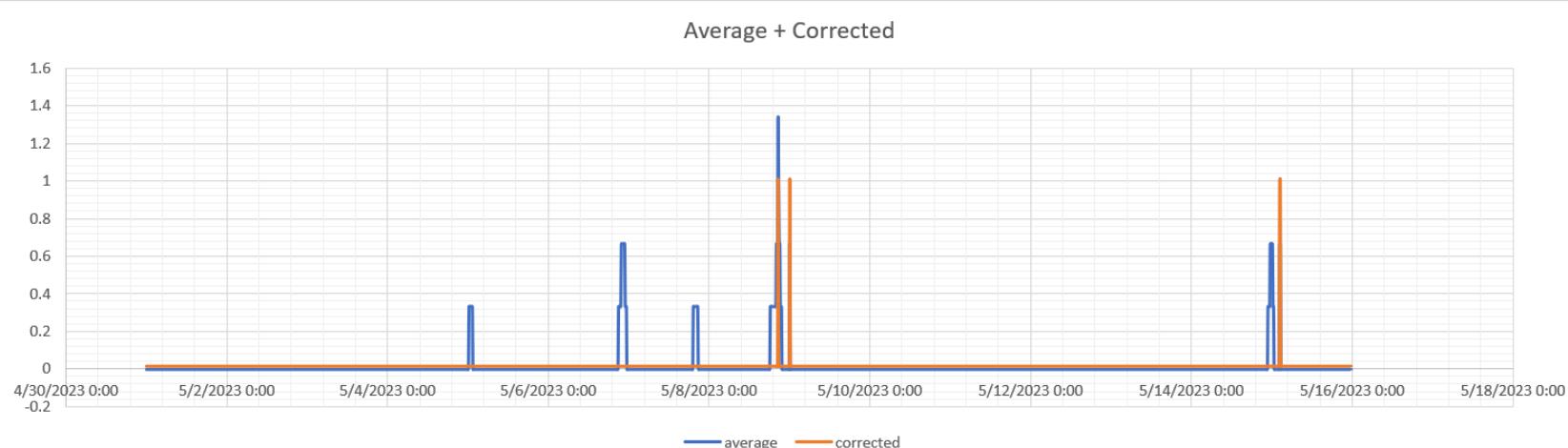
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 8. Desviación de la precipitación.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

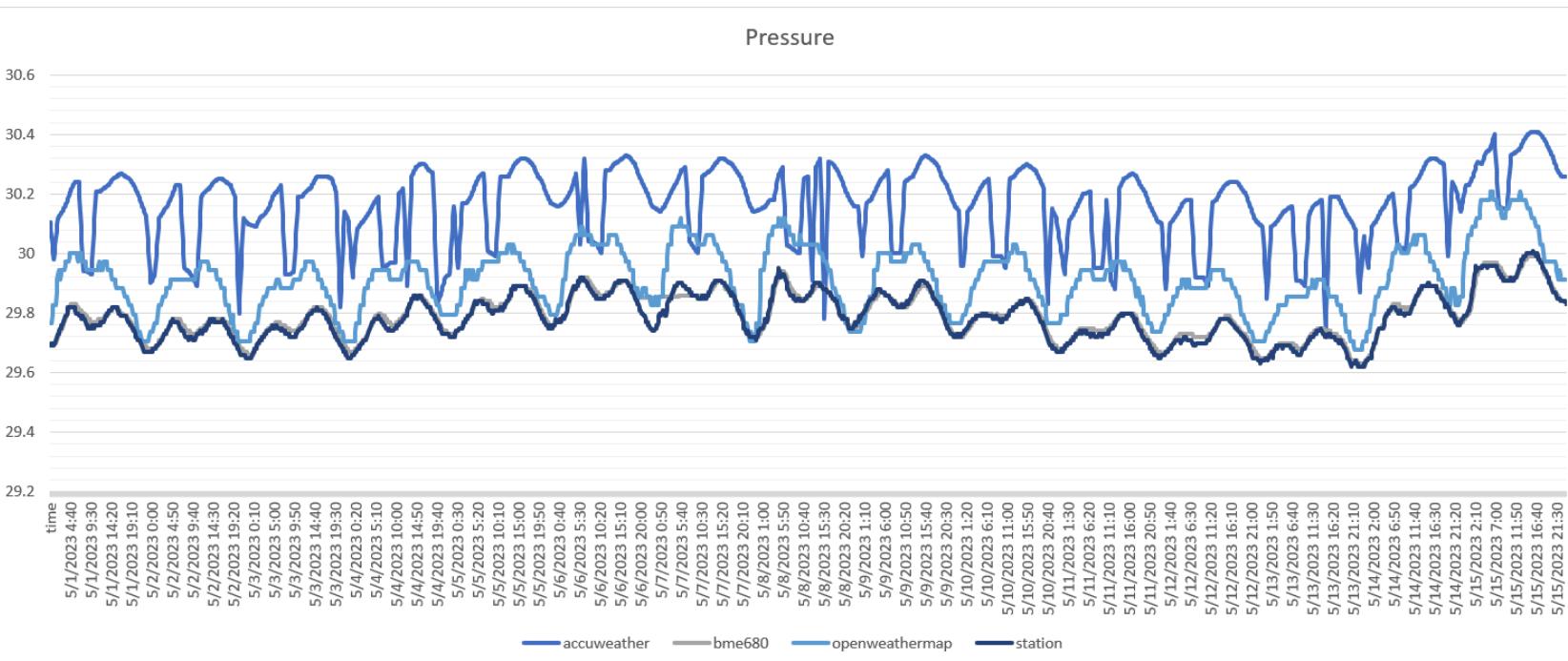
Gráfica 9. Corrección comparada con el promedio de la precipitación.



15.4.3.4. Presión

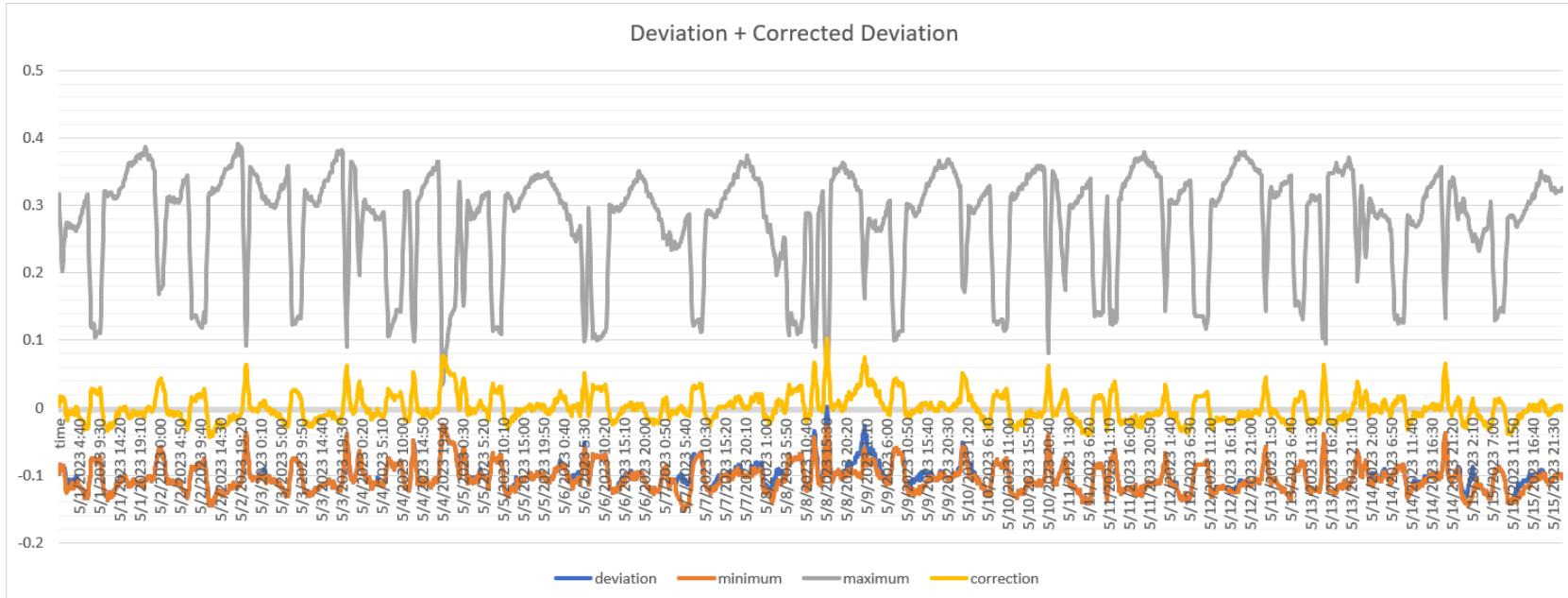
Para la presión se obtienen un factor de corrección de -0.102 in/mmHg .

Gráfica 10. Presión.



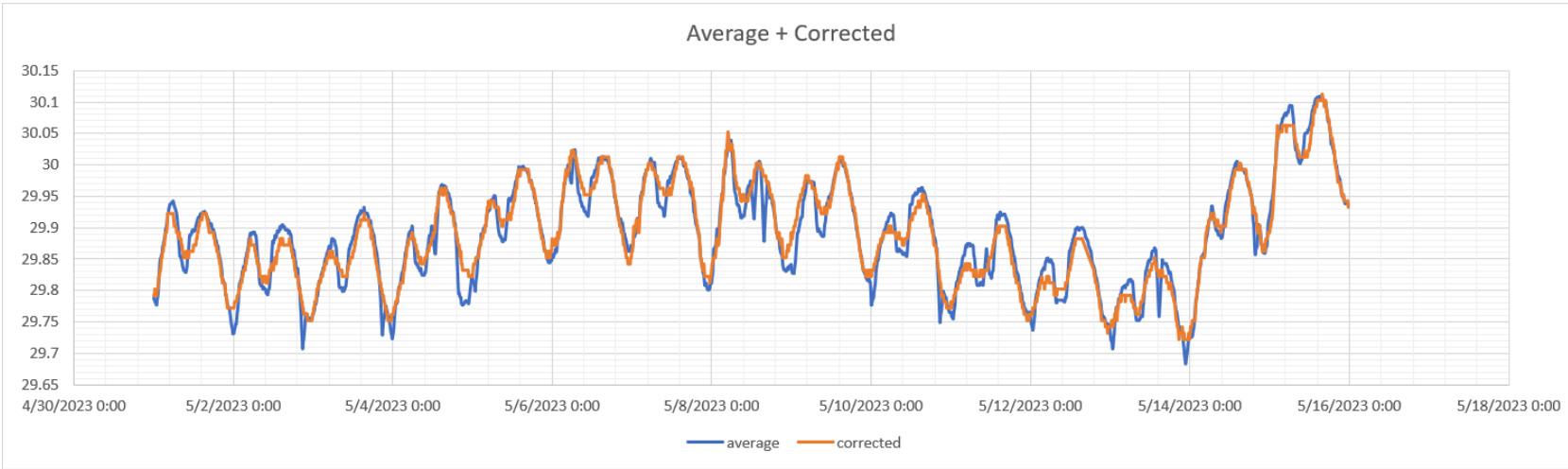
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 11. Desviación de la presión.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

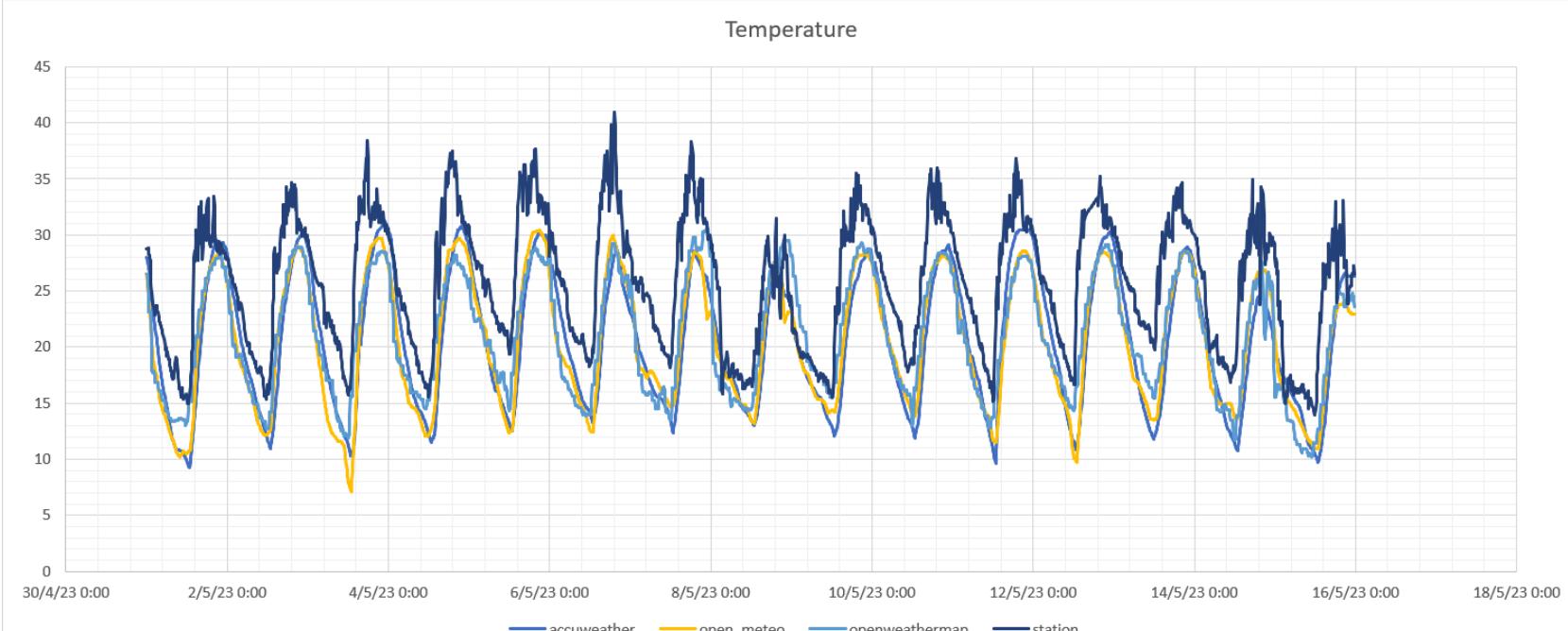
Gráfica 12. Corrección comparada con el promedio de la presión.



15.4.3.5. Temperatura

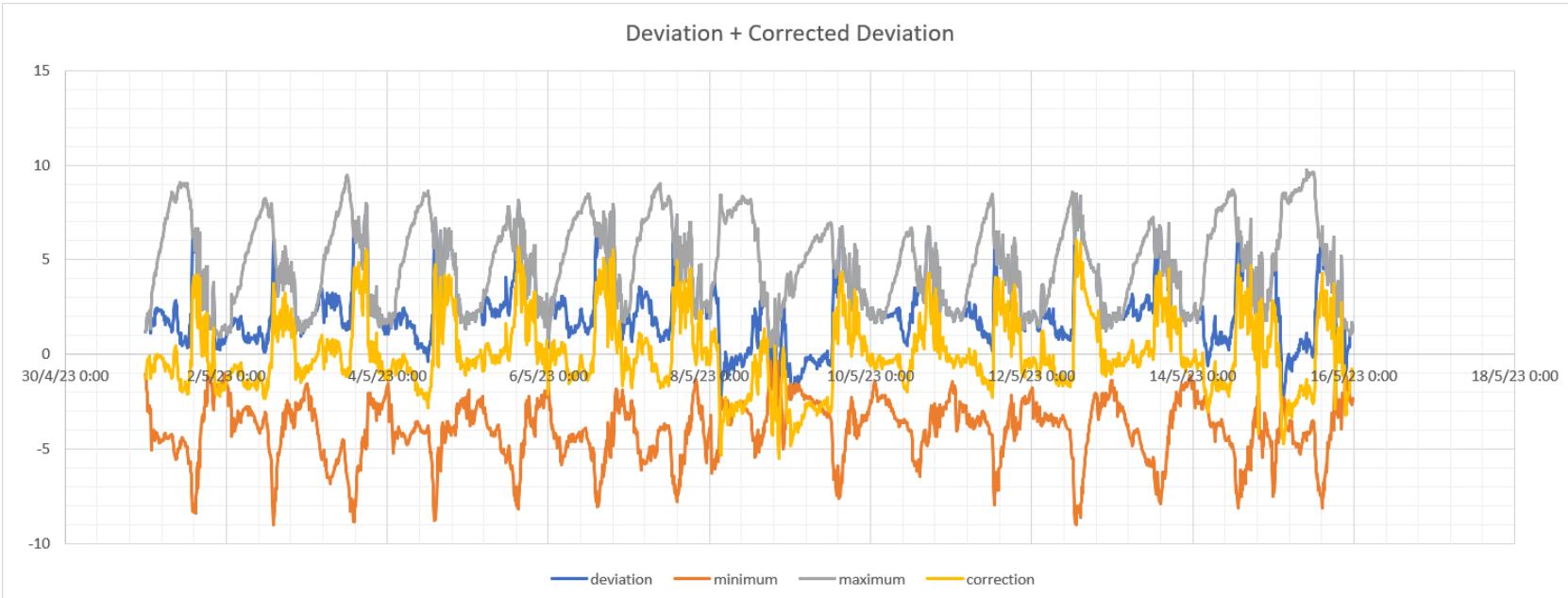
Para la temperatura se obtienen un factor de corrección de 2.4443°C .

Gráfica 13. Temperatura.



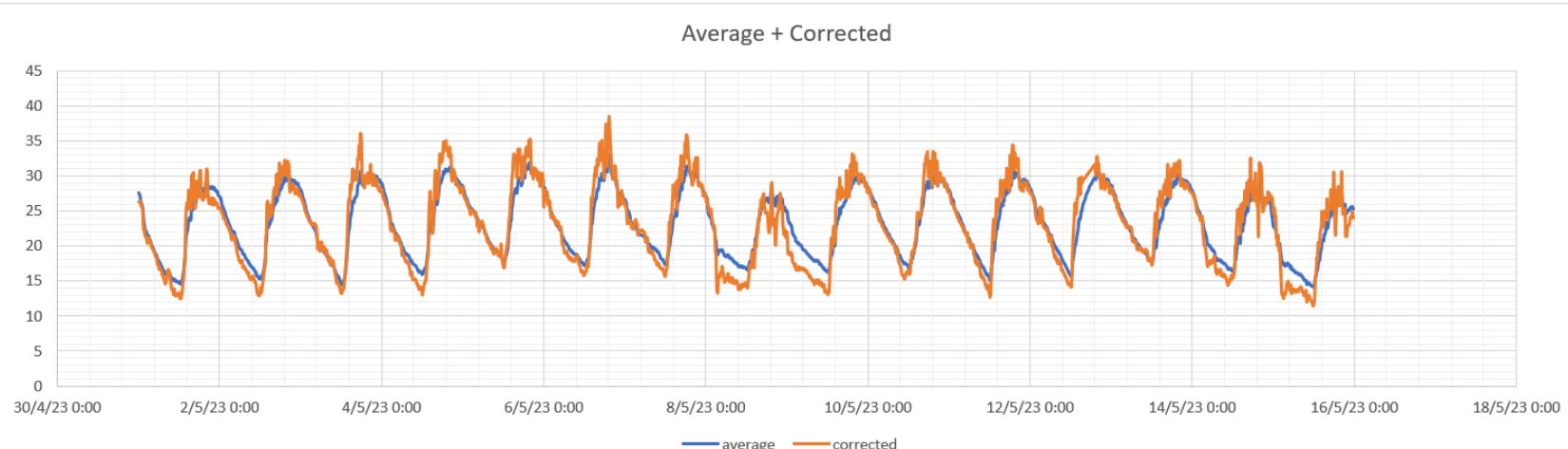
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 14. Desviación de la temperatura.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

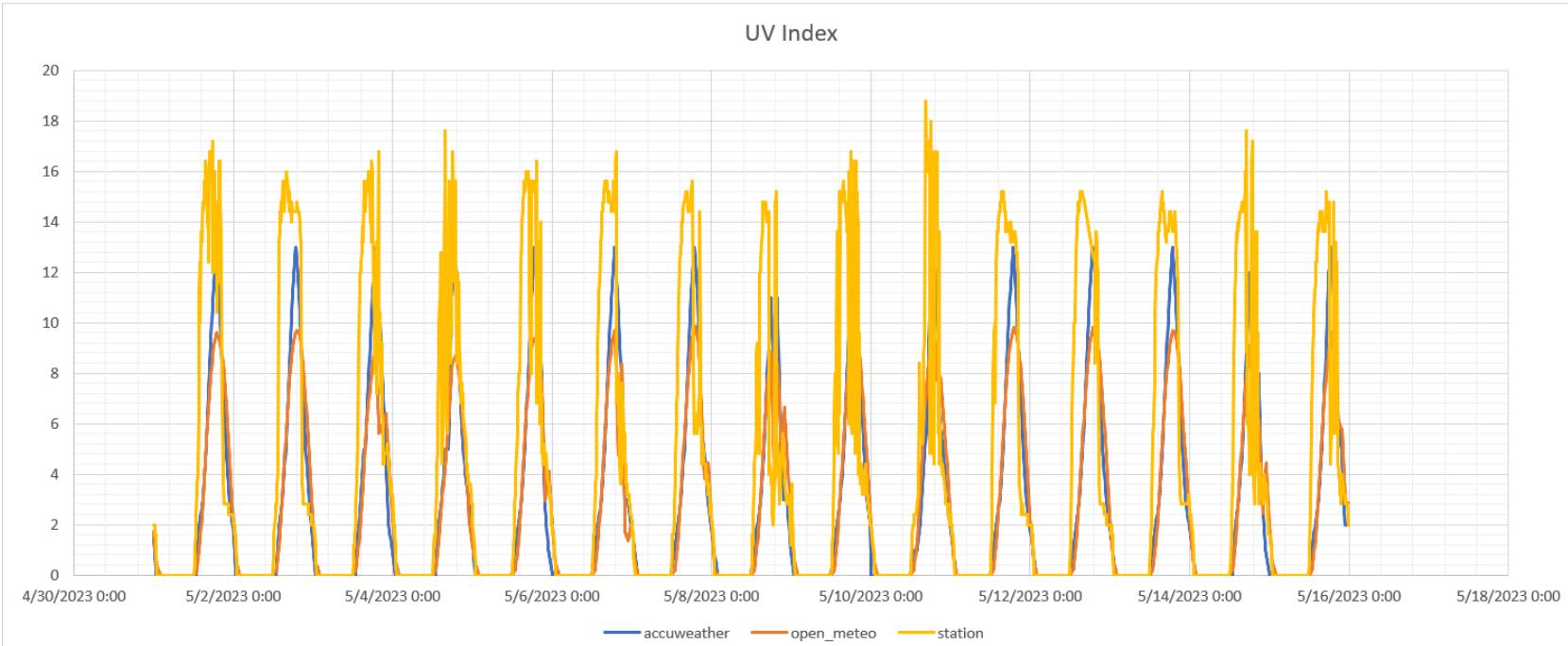
Gráfica 15. Corrección comparada con el promedio de la temperatura.



15.4.3.6. Índice UV

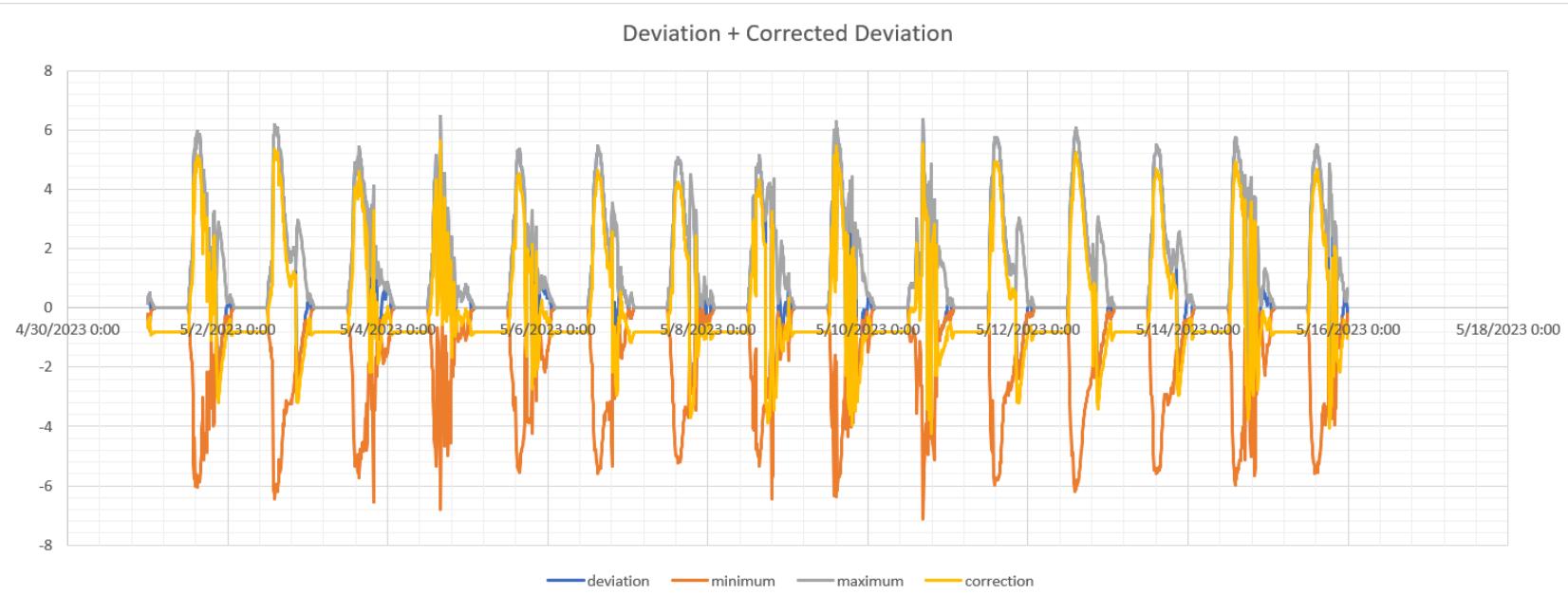
Para el índice UV se obtienen un factor de corrección de 0.82127.

Gráfica 16. Índice UV.



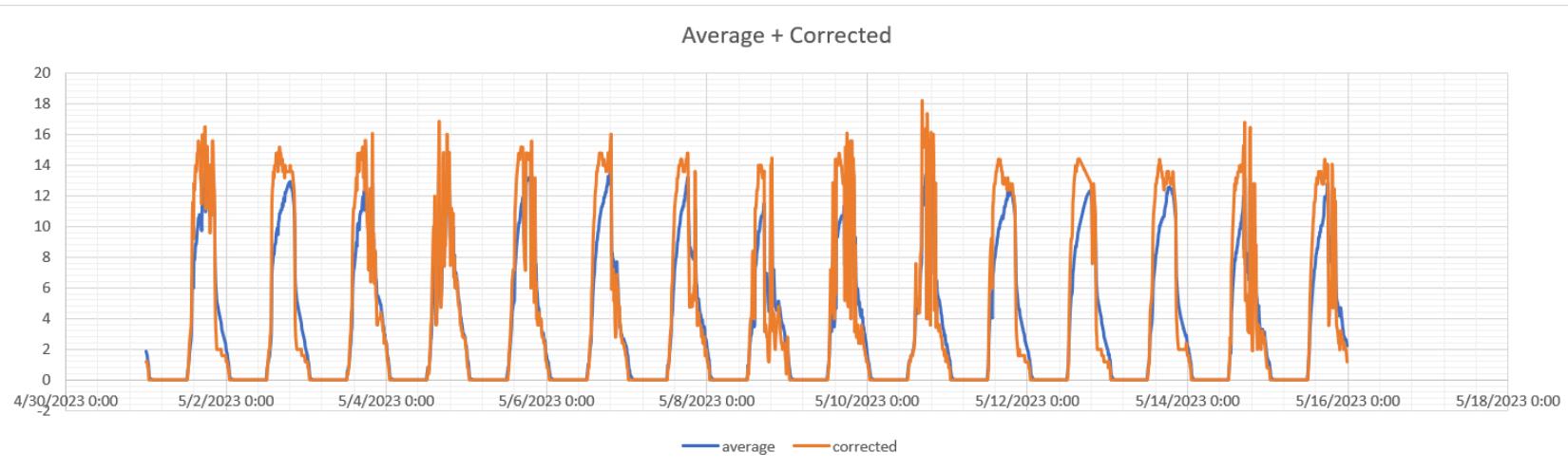
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 17. Desviación del índice UV.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

Gráfica 18. Corrección comparada con el promedio del índice UV.



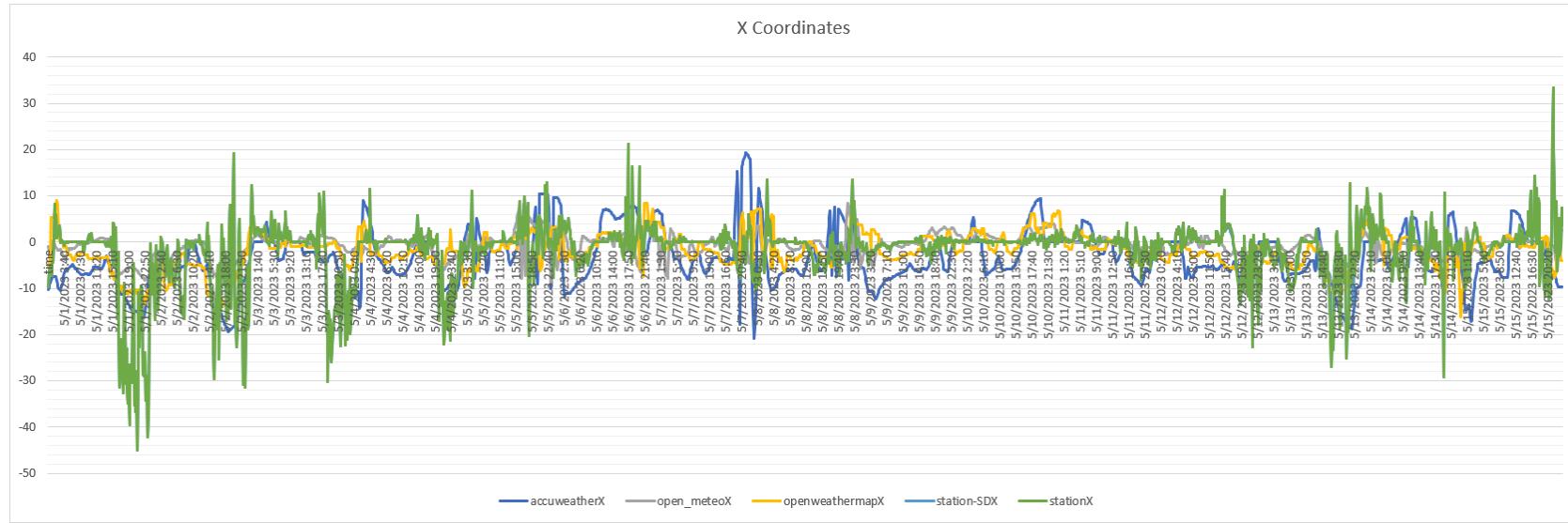
15.4.3.7. Velocidad de Viento

Nota. Para el análisis de la velocidad de viento se tiene que considerar que los datos de la velocidad del viento se encuentran en factores de dirección y velocidad, por lo que se tiene que realizar una conversión a los datos de velocidad de viento en X y Y.

15.4.3.7.1. Coordenadas de Viento en X

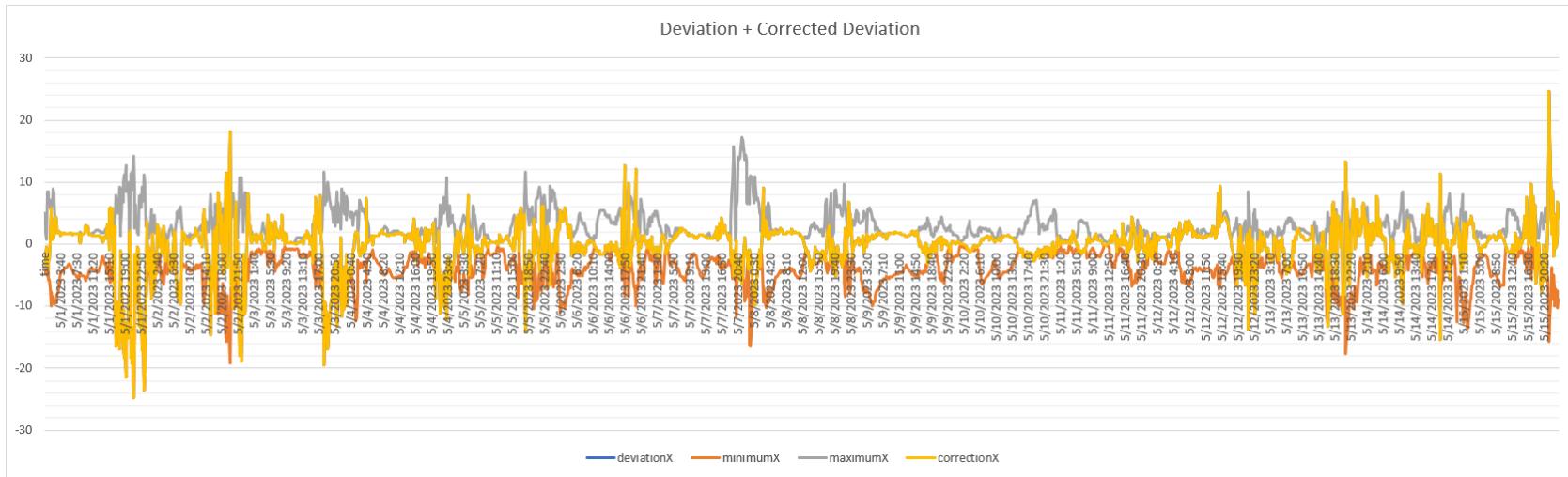
Para las coordenadas de viento en X se obtienen un factor de corrección de -0.216851 knots .

Gráfica 19. Coordenadas de viento en X.



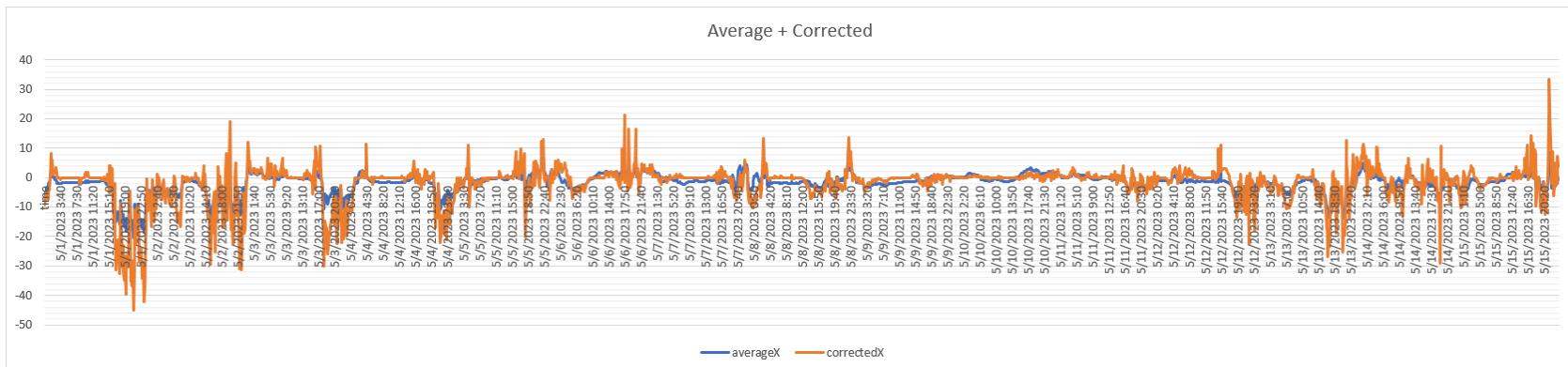
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 20. Desviación de las coordenadas de viento en X.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

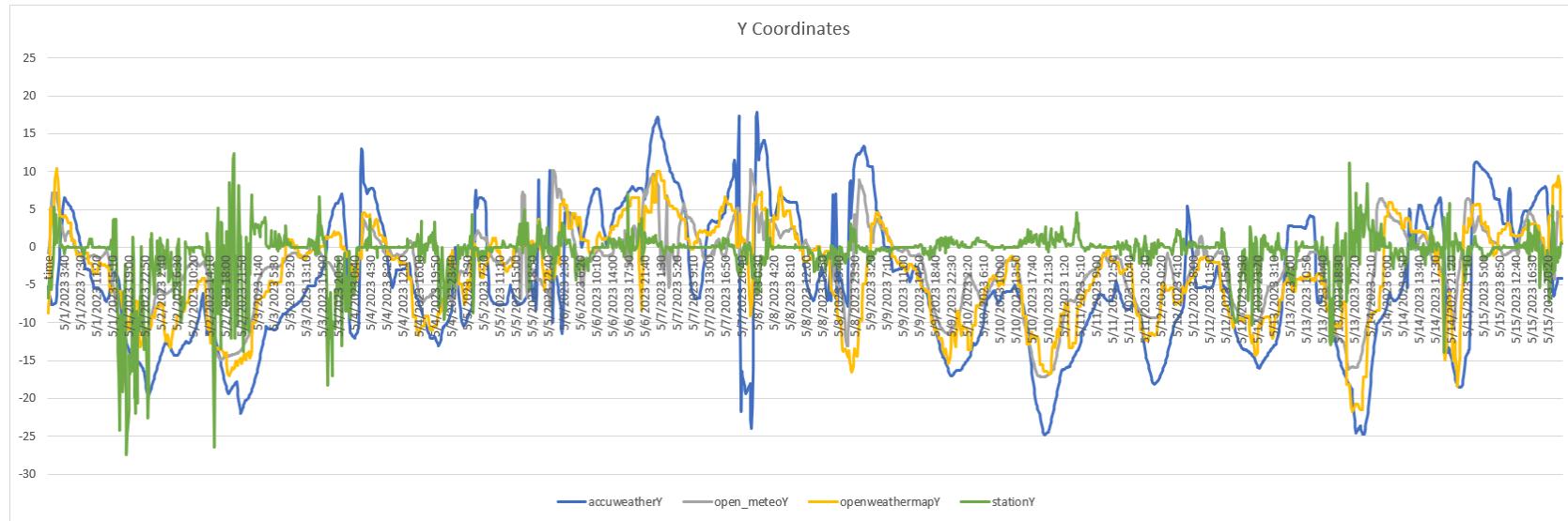
Gráfica 21. Corrección comparada con el promedio de las coordenadas de viento en X.



15.4.3.7.2. Coordenadas de Viento en Y

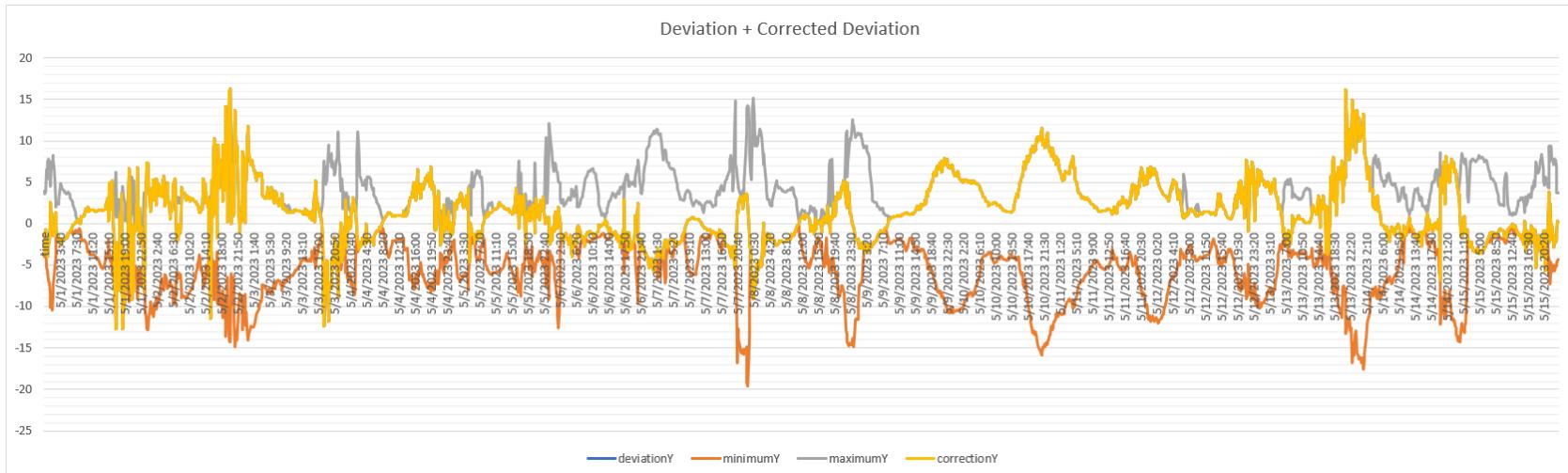
Para las coordenadas de viento en Y se obtienen un factor de corrección de 1.3681405 knots.

Gráfica 22. Coordenadas de viento en Y.



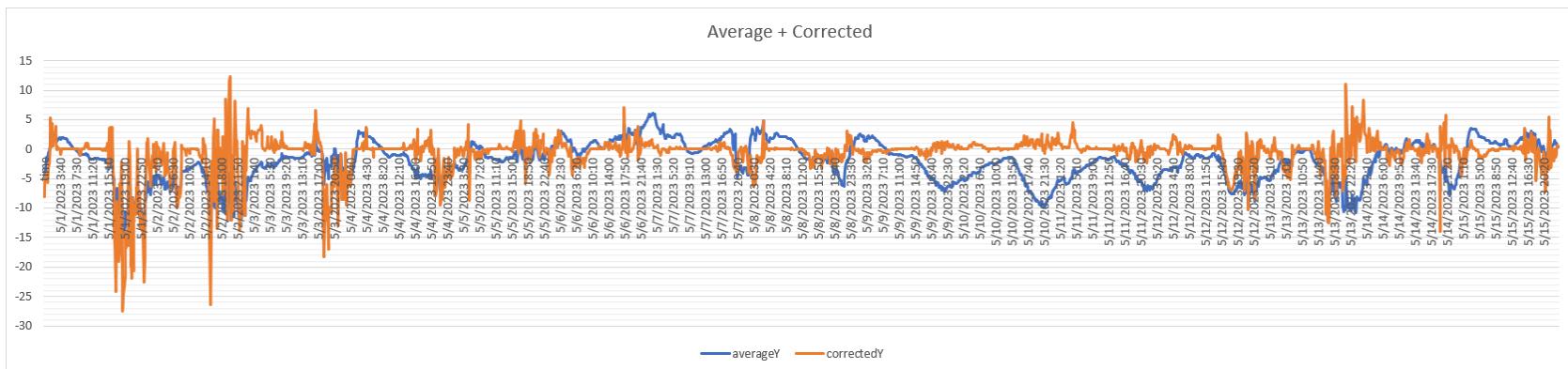
Con su respectiva desviación, comparada contra los mínimos, máximos y la nueva desviación a partir de los datos corregidos.

Gráfica 23. Desviación de las coordenadas de viento en Y.



Resultando así en la gráfica final comparando los datos corregidos contra los datos promedio.

Gráfica 24. Corrección comparada con el promedio de las coordenadas de viento en Y.



15.4.4. Análisis de Costos

Nota. La tasa de cambio utilizada para la conversión de USD a MXN es de 1 USD = 20.00 MXN y para la conversión de EUR a MXN es de 1 EUR = 20.00 MXN.

15.4.4.1. Estación

El costo de la estación elaborada se obtiene a partir de la suma de los costos de los componentes de la estación, los cuales se muestran en la siguiente tabla.

Nota. Facturas y cotizaciones a disposición de petición.

Tabla 3. Costos de la Estación.

Producto	Cantidad	Precio (MXN)
Adaptador de Corriente 5V 3A	1	480.00
Extensión de uso rudo	1	754.53
Kit Estación Meteorológica	1	2001.11
Lector de Tarjetas MicroSD	1	304.50
MicroMod Weather Board	1	1028.64
MicroMod ESP32	1	350.00
MicroMod SD 64GB	1	240.00
Protector de Sensores	1	526.03
Total		5684.81

15.4.4.2. Estaciones Comerciales

Se cotizaron 2 estaciones comerciales.

15.4.4.2.1. BASIC Wireless Weather Station LoRaWAN Set (USA, Europe, Africa, Brazil, Mexico)

La primera estación de la cual se obtuvo una cotización fue la estación BASIC Wireless Weather Station LoRaWAN Set (USA, Europe, Africa, Brazil, Mexico) de la compañía BARANI, la cual tiene un costo de 1,797.00 EUR o aproximadamente 35,940.00 MXN.

Figura 55. Estación BASIC Wireless Weather Station LoRaWAN (BARANI).



<https://barani-design.myshopify.com/products/basic-wireless-weather-station-set>

15.4.4.2.2. MWS-C400

La segunda estación de la cual se obtuvo una cotización fue la estación MWS-C400 de la compañía intellisense, la cual tiene un costo inicial de 7,000.00 USD o aproximadamente 140,000.00 MXN.

Figura 56. Estación MWS-C400 (*intellisense*).



<https://www.intellisenseinc.com/products/weather-stations/mws-c400/>

16. Conclusión

La investigación y construcción de una estación meteorológica ofrecen numerosos beneficios y oportunidades para la comprensión del uso de la tecnología en la meteorología. A lo largo de este proyecto, se han adquirido conocimientos en electrónica, programación y análisis de datos.

La estación meteorológica construida proporciona la capacidad de obtener datos en tiempo real y realizar análisis personalizados. Esto permite una mayor comprensión del clima local y la capacidad de monitorear las condiciones meteorológicas de manera precisa y confiable. Recordar que, al realizar investigaciones y análisis propios, se puede contribuir al conocimiento científico y comprender mejor los patrones climáticos y sus efectos.

Así es como, el proyecto culmina en un análisis de los datos obtenidos por la estación meteorológica construida. Datos que se compararon con los obtenidos de fuentes de información meteorológica como AccuWeather, Aviation Weather Center, Open-Meteo y OpenWeatherMap.

A partir del análisis de los datos, se determinó que la estación meteorológica y sus datos presentan una desviación con respecto a las fuentes de información meteorológica. Por lo tanto, se aplicó una corrección a los datos para obtener resultados más confiables.

De cada una de las variables analizadas, se llegaron a las siguientes conclusiones.

- **Punto de Rocío y Presión.** Los datos de punto de rocío y presión son los más confiables, ya que presentan una desviación mínima con respecto a las fuentes de información meteorológica. Esto puede ser debido a la precisión de los sensores utilizados para medir estos datos, así como a la poca variabilidad de estos datos en el campo.
- **Temperatura y Humedad.** Los datos de temperatura y humedad presentan una desviación mayor con respecto a las fuentes de información meteorológica, por lo que se recomienda realizar una calibración de estos datos cada cierto tiempo para asegurar la confiabilidad de los datos. Es importante tomar en cuenta que la ubicación de la estación puede afectar los datos de temperatura y humedad, por lo que se recomienda colocar la estación en un lugar donde no se vea afectada por la luz solar directa para futuras mediciones.
- **Índice UV.** El dato del índice UV también se ve afectado por la desviación, pero en menor medida. Aunque no es completamente confiable, se puede considerar que es un dato aceptable en términos de precisión.
- **Velocidad del Viento.** Aunque tiene una desviación reducida, es un dato que se ve afectado por la desviación, por lo que se recomienda realizar una calibración de estos datos cada cierto tiempo para asegurar la confiabilidad de los datos. Es importante

tener en cuenta que los datos obtenidos están fuertemente sujetos a las condiciones de la ubicación de la estación.

- **Precipitación.** Fue prácticamente nula durante el periodo de prueba, por lo que no se puede concluir nada sobre este dato.

Además de los resultados obtenidos, la realización de una propia estación meteorológica nos permite personalizar y experimentar con ella. Podemos realizar mejoras y ajustes según nuestras necesidades y preferencias, fomentando la innovación y la exploración de nuevas ideas en el campo de la meteorología y la aeronáutica.

La ventaja de una estación construida en comparación con las estaciones comerciales es el costo. Mientras que las estaciones comerciales tienen un precio de 35,940.00 MXN y 140,000.00 MXN, la estación construida tiene un costo de 5,684.81 MXN. Además, la estación construida no está limitada por restricciones comerciales y se puede modificar para medir más datos, lo que la hace más versátil.

Si bien no se puede considerar la estación construida como un reemplazo a los sistemas de observación meteorológica, su existencia y aplicaciones son prometedoras, debido a las múltiples implementaciones que se pueden llevar a cabo con la misma, desde el reporte de datos meteorológicos en tiempo real hasta la implementación de sistemas de alerta temprana, el enlace de información mediante el sitio [WEATHER UNDERGROUND](#) y todo esto a un costo accesible.

Los resultados obtenidos y las conclusiones alcanzadas no se habrían logrado de la misma manera sin la construcción de la estación meteorológica. Esta experiencia nos permitió comprender mejor los componentes y su funcionamiento, así como implementar mejoras y mejores prácticas en la toma de datos.

La investigación y construcción de una estación meteorológica son una experiencia enriquecedora que ofrece beneficios tanto en el ámbito personal como en el desarrollo de habilidades técnicas. Este proyecto brinda la oportunidad de contribuir al conocimiento científico, comprender mejor el clima local y abrir puertas a oportunidades profesionales en las áreas de la meteorología, la aeronáutica y la electrónica.

Las aportaciones que brinda a las distintas disciplinas son las siguientes.

- **Aeronáutica.** La estación meteorológica brinda información de suficiente fiabilidad para su manejo y operación en aeropuertos y pistas de menor uso, permitiendo una mejor toma de decisiones en cuanto a la operación de aeronaves aún en zonas remotas.
- **Aeroespacial.** La estación provee información de suficiente fiabilidad para su manejo y operación en zonas de lanzamiento de cohetes de menor escala, educativos, amateur y de investigación, permitiendo un mayor control y seguridad en los lanzamientos.

- **Aviónica.** La estación meteorológica brinda la oportunidad de experimentar con sensores y microcontroladores, permitiendo el desarrollo de habilidades técnicas y la exploración de nuevas ideas en el campo de la electrónica en la aviación.
- **Meteorología.** La estación brinda suficiente información para contribuir a la red de estaciones meteorológicas, permitiendo una mejor comprensión de los patrones climáticos y la predicción del clima en zonas locales.

En conclusión, la estación meteorológica que se ha construido representa una alternativa viable a las estaciones comerciales. Aunque los datos que proporciona no son completamente fiables, esto puede no ser un inconveniente en todos los casos, ya que no todas las aplicaciones requieren una precisión extrema. Sin embargo, si se desea aumentar su fiabilidad, se puede realizar una calibración periódica de los sensores para garantizar la precisión de los datos.

Es importante destacar que la estación no debe considerarse equivalente a una estación profesional. No obstante, esto no resta mérito a su utilidad: sigue siendo una opción más económica y versátil para aquellos que buscan monitorear el clima a un nivel básico.

17. Glosario

Definiciones en orden alfabético.

A

AccuWeather. Servicio de pronóstico del tiempo y análisis meteorológicos.

Adafruit. Empresa de hardware y software de código abierto.

Aerodromo. Lugar destinado para el aterrizaje y despegue de aeronaves.

Aeronáutica. Ciencia que estudia la navegación aérea.

Aeronave. Vehículo capaz de volar.

ADC. Convertidor analógico-digital. Dispositivo que convierte una señal analógica en una señal digital.

Agile. Metodología de desarrollo de software.

Agricultura. Uso de técnicas y conocimientos para cultivar la tierra y producir alimentos.

Analógico. Sistema de representación de información que utiliza valores continuos.

Anemómetro. Instrumento utilizado para medir la velocidad y dirección del viento.

API. Interfaz de programación de aplicaciones. Conjunto de reglas y especificaciones que las aplicaciones pueden seguir para comunicarse entre sí.

API Key. Clave de API. Código que se utiliza para autenticar una solicitud a una API.

Appwrite. Plataforma de backend de código abierto para aplicaciones web y móviles.

Arduino. Plataforma de hardware y software libre para el desarrollo de prototipos electrónicos basada en una placa con un microcontrolador y un entorno de desarrollo.

ASOS. Sistema Automático de Observación del Tiempo a nivel de Superficie, utilizado para la recopilación de datos meteorológicos en tiempo real.

Autenticación. Proceso de verificar la identidad de un usuario.

Aviación. Actividades relacionadas con el diseño, desarrollo, producción, operación y uso de aeronaves.

AWC. Centro de pronóstico de la aviación.

AWOS. Sistema de observación meteorológica automatizado.

AWS. Estación meteorológica automática.

B

Backend. Parte de un sistema informático o de una aplicación que el usuario no ve, pero que maneja la funcionalidad de la aplicación.

Barani. Compañía que fabrica estaciones meteorológicas.

Barómetro. Instrumento utilizado para medir la presión atmosférica.

Base de datos. Colección organizada de datos, generalmente almacenada y accesible electrónicamente.

Bash. Interprete de comandos de Unix.

Baudios. Unidad de medida de la velocidad de transmisión de datos.

Binario. Sistema de numeración que utiliza dos dígitos, 0 y 1.

BME280. Sensor de temperatura, humedad y presión atmosférica desarrollado por Bosch.

BME680. Sensor de temperatura, humedad, presión atmosférica y gas desarrollado por Bosch.

Board. Placa de circuito impreso. Placa que contiene componentes electrónicos y se utiliza para conectar otros componentes.

Bug. Error o defecto en un programa de computadora que causa un resultado incorrecto o inesperado.

C

C. Lenguaje de programación de bajo nivel.

Calibración. Proceso de ajuste de un instrumento de medición para que proporcione resultados correctos.

CAPMA. Centro de Análisis y Pronósticos Meteorológicos Aeronáuticos.

Cloudflare. Empresa de servicios de red y seguridad. Dedicada a la entrega de contenido web y a la distribución de red de entrega de contenido.

Codificación. Proceso de transformar información de un formato a otro.

Colección. Conjunto de datos almacenados en una base de datos.

Compilador. Programa que traduce el código fuente escrito por los programadores en un lenguaje que la computadora puede entender.

CONAGUA. Comisión Nacional del Agua.

CPU. Unidad Central de Procesamiento.

Crontab. Archivo que contiene la lista de tareas programadas en un sistema Unix.

CS pin. Chip Select pin. Pin utilizado para seleccionar un dispositivo en un bus de comunicación.

CSV. Formato de archivo que almacena datos en forma de tabla.

D

Data Logger. Dispositivo electrónico que registra y almacena datos de sensores y otros dispositivos de medición.

Depuración. Proceso de identificación y corrección de errores en el código de un programa de computadora.

Desviación. Diferencia entre el valor medido y el valor real.

Dev Board. Placa de desarrollo. Placa de circuito impreso que se utiliza para prototipos y pruebas de hardware.

DGAC. Dirección General de Aeronáutica Civil.

Diagrama. Representación gráfica de un conjunto de datos.

Digital. Sistema de representación de información que utiliza valores discretos.

DNS. Sistema de nombres de dominio.

DOD. Departamento de Defensa de los Estados Unidos.

Dominio. Nombre único que identifica un sitio web en Internet.

D2. Lenguaje de programación de diagramas.

E

EMA's. Estaciones Meteorológicas Automáticas.

ESP32. Microcontrolador de bajo costo y bajo consumo de energía desarrollado por Espressif Systems.

Espressif. Empresa de tecnología de la información.

Esquemático. Diagrama que muestra la conexión de los componentes electrónicos en un circuito.

Estación Meteorológica. Conjunto de instrumentos que se utilizan para medir y registrar las condiciones atmosféricas.

Estación Meteorológica Automática. Estación meteorológica que realiza las mediciones de manera automática.

Estación Meteorológica Comercial. Estación meteorológica que se puede adquirir en el mercado.

Ethernet. Estándar de red de área local. Protocolo de red que se utiliza para la comunicación entre dispositivos en una red local.

Excel. Programa de hojas de cálculo desarrollado por Microsoft.

F

FAA. Administración Federal de Aviación.

Factor de corrección. Valor utilizado para ajustar una medición a un valor conocido.

Fedora. Sistema operativo basado en Linux.

Frecuencia. Número de veces que se repite un fenómeno en un intervalo de tiempo.

Frontend. Parte de un sistema informático o de una aplicación que el usuario ve y con la que interactúa.

Firebase. Plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google.

Firmware. Software que se ejecuta en un dispositivo de hardware.

FPGA. Matriz de puertas programable en campo.

FTP. Protocolo de transferencia de archivos.

G

Gantt. Herramienta de planificación de proyectos que utiliza un gráfico de barras para visualizar el progreso de un proyecto a lo largo del tiempo.

GCP. Google Cloud Platform. Plataforma de servicios en la nube desarrollada por Google.

Git. Sistema de control de versiones.

GitHub. Plataforma de desarrollo colaborativo de software.

GMT. Tiempo Medio de Greenwich.

GND pin. Ground pin. Pin utilizado para conectar a tierra un dispositivo.

GNU. Sistema operativo de código abierto.

GO. Lenguaje de programación de código abierto.

GPIO. General Purpose Input/Output. Pines de propósito general utilizados para la entrada y salida de señales digitales.

GPS. Sistema de posicionamiento global.

H

Hardware. Componentes físicos de un sistema informático.

Humedad. Cantidad de vapor de agua presente en el aire.

Humedad Relativa. Relación entre la cantidad de vapor de agua presente en el aire y la cantidad máxima que podría contener a una temperatura dada.

HTML. Lenguaje de marcado utilizado para crear páginas web.

HTTP. Protocolo de transferencia de hipertexto.

HTTPS. Protocolo de transferencia de hipertexto seguro.

Hygrotermómetro. Instrumento utilizado para medir la temperatura y la humedad del aire.

I

ICAMS. Sistema Integrado de Mantenimiento Colaborativo de la Aviación.

ICAO. Organización de Aviación Civil Internacional.

IDE. Entorno de desarrollo integrado.

Investigación. Proceso de búsqueda de conocimiento.

IP. Protocolo de Internet.

ISO 8601. Formato de fecha y hora internacionalmente aceptado, AAAA-MM-DDTHH:MM:SS .

ISP. Proveedor de servicios de Internet.

I2C. Inter-Integrated Circuit. Protocolo de comunicación de bus serie.

J

JavaScript. Lenguaje de programación interpretado, de alto nivel y orientado a objetos.

JSON. Notación de objetos de JavaScript. Formato de intercambio de datos.

K

Kanban. Sistema de gestión de proyectos que se centra en la visualización flujo de trabajo y la eliminación de los cuellos de botella.

L

LibreOffice. Suite de oficina de código abierto.

Librerías. (Libraries) Son conjuntos de funciones y procedimientos que se pueden utilizar para realizar tareas específicas, como la lectura de archivos CSV, la realización de peticiones HTTP, el parseo de información JSON, entre otros.

Licencia BSD. Licencia de software de código abierto. Permite a los usuarios utilizar, modificar y distribuir el software de forma gratuita.

Linux. Sistema operativo de código abierto basado en Unix. Uno de los sistemas operativos más utilizados en servidores y supercomputadoras.

Logs. Registros de eventos. En este caso, registros de datos meteorológicos.

Loop. Estructura de control en programación que permite repetir un bloque de código varias veces.

M

MariaDB. Sistema de gestión de bases de datos relacional.

METAR. Código de observación meteorológica para la aviación.

Meteorología. Ciencia que estudia la atmósfera terrestre y los fenómenos meteorológicos.

Microcontrolador. Circuito integrado programable que se utiliza para controlar dispositivos electrónicos.

MicroMod. Plataforma modular de electrónica que permite a los diseñadores crear prototipos de dispositivos electrónicos personalizados utilizando una variedad de módulos de hardware interconectables.

microSD. Formato de tarjeta de memoria utilizado en dispositivos electrónicos para almacenar y transferir datos.

Modular. Sistema que se compone de módulos independientes que se pueden combinar para formar un sistema más grande.

MQTT. Protocolo de mensajería ligero para redes de sensores.

MySQL. Sistema de gestión de bases de datos relacional.

N

NOAA. Administración Nacional Oceánica y Atmosférica.

NOTAM. Aviso a los aviadores. Información importante para la seguridad de la aviación.

NTP. Protocolo de tiempo de red.

NWS. Servicio Meteorológico Nacional, de los Estados Unidos.

O

OLED. Pantalla de diodos orgánicos de emisión de luz.

Open-Meteo. Plataforma de datos meteorológicos abiertos para el acceso y la investigación.

OpenWeatherMap. Servicio en línea que proporciona pronósticos del tiempo y datos meteorológicos en tiempo real.

Oracle. Empresa de tecnología de la información.

OS. Sistema operativo.

P

Parametrización. Proceso de ajuste de los parámetros de un sistema para que se ajusten a un conjunto de especificaciones.

Parámetros. Variables que se utilizan para ajustar el comportamiento de un sistema.

Parsear. (Parsing) Es el proceso de analizar una cadena de símbolos, ya sea en lenguaje natural, lenguajes informáticos o estructuras de datos, conforme a las reglas de una gramática formal. El término análisis proviene del latín pars, que significa parte.

Pinout. Diagrama que muestra la disposición de los pines en un conector o dispositivo.

Pluviómetro. Instrumento utilizado para medir la cantidad de lluvia que cae en un determinado período de tiempo.

PocketBase. Plataforma de hardware y software libre para el desarrollo de prototipos electrónicos basada en una placa con un microcontrolador y un entorno de desarrollo.

POST. Petición HTTP que se utiliza para enviar datos a un servidor web.

Precipitación. Cantidad de agua que cae en forma de lluvia, nieve, granizo, etc.

Presión atmosférica. Presión ejercida por el aire en la superficie terrestre.

Presión barométrica. Presión atmosférica medida con un barómetro.

Pronóstico Meteorológico. Predicción del tiempo futuro basada en datos meteorológicos actuales y modelos meteorológicos.

Protoboard. Placa de pruebas utilizada para prototipos electrónicos.

Prototipo. Modelo inicial de un producto o sistema.

Python. Lenguaje de programación interpretado, de código abierto y multiplataforma.

Punto de rocío. Temperatura a la que el vapor de agua comienza a condensarse en el aire.

Q

Query. Consulta. En este caso, una consulta a una base de datos.

R

Radiación. Transferencia de energía en forma de ondas electromagnéticas.

Radiación solar. Energía emitida por el sol en forma de ondas electromagnéticas.

RAM. Memoria de acceso aleatorio.

RDBMS. Sistema de gestión de bases de datos relacionales.

Repositorio. Lugar centralizado donde se almacena y mantiene el código fuente, las pruebas y la documentación de un proyecto de software.

Resistencia. Propiedad de un material que se opone al flujo de corriente eléctrica.

RJ11. Tipo de conector utilizado para conectar teléfonos, módems y otros dispositivos de comunicación a la red telefónica.

S

SCL. Serial Clock. Línea de reloj utilizada en la comunicación I2C.

Script. Archivo de texto que contiene un conjunto de instrucciones o comandos de programación que se ejecutan automáticamente.

SCT. Secretaría de Comunicaciones y Transportes.

SD card. Tarjeta de memoria utilizada en dispositivos electrónicos para almacenar y transferir datos.

SDA. Serial Data. Línea de datos utilizada en la comunicación I2C.

SDO pin. Serial Data Out pin. Pin utilizado para la salida de datos en un bus de comunicación.

Sensor. Dispositivo que detecta y responde a estímulos físicos, químicos o biológicos.

Servidor. Computadora o sistema informático que proporciona servicios a otros dispositivos o programas.

Shell. Interfaz de línea de comandos utilizada para interactuar con el sistema operativo y ejecutar programas y scripts en sistemas Unix.

SMN. Servicio Meteorológico Nacional.

Software. Programas de computadora y datos que proporcionan instrucciones a un sistema informático.

SparkFun. Empresa que diseña y fabrica componentes y kits de electrónica para aficionados y profesionales de la electrónica.

SPECI. Informe Especial No Programado.

SPI. Interfaz de Periférico en Serie. Protocolo de comunicación de bus serie.

SQL. Lenguaje de consulta estructurado.

SSH. Secure Shell. Protocolo de red que se utiliza para la comunicación segura entre dispositivos.

SSL. Secure Sockets Layer. Protocolo de seguridad que se utiliza para proteger la información transmitida a través de Internet.

Sub-dominio. Dominio que forma parte de un dominio principal. Por ejemplo, blog.ejemplo.com es un subdominio de ejemplo.com .

T

TAF. Pronóstico de área terminal.

TDS. Servidor de Datos de Texto.

Temperatura. Medida de la intensidad del calor o frío en un cuerpo.

Terminal. Interfaz de línea de comandos utilizada para interactuar con el sistema operativo y ejecutar programas y scripts en sistemas Unix.

Termómetro. Instrumento utilizado para medir la temperatura.

Timestamp. Marca de tiempo.

U

Ubuntu. Sistema operativo de código abierto basado en GNU/Linux.

Ultravioleta. Radiación electromagnética con una longitud de onda más corta que la luz visible.

UNIX. Sistema operativo de código abierto. Uno de los sistemas operativos más antiguos y utilizados.

URL. Localizador uniforme de recursos. Dirección de un recurso en Internet.

UV Index. Índice que indica la intensidad de los rayos UV del sol en un lugar determinado.

UVA. Longitud de onda de la radiación UV que penetra más profundamente en la piel.

UVB. Longitud de onda de la radiación UV responsable del enrojecimiento y quemaduras solares.

UTC. Tiempo Universal Coordinado.

V

Variable. Elemento de un programa de computadora que puede cambiar su valor durante la ejecución del programa.

VCC. Voltage Common Collector. Pin utilizado para suministrar energía a un dispositivo.

Veleta. Instrumento utilizado para medir la dirección del viento.

VEML6075. Sensor de radiación ultravioleta.

Visibilidad. Distancia a la que un objeto o luz es claramente visible.

VPS. Servidor privado virtual.

W

Weather Underground. Servicio de pronósticos del tiempo y datos meteorológicos históricos con una red de estaciones meteorológicas en todo el mundo.

WiFi. Tecnología de red inalámbrica que permite la conexión a Internet y a otros dispositivos.

WMO. Organización Meteorológica Mundial.

X

XML. Lenguaje de marcado extensible.

Y

YAML. Lenguaje de serialización de datos legible por humanos.

Yaw. Ángulo de rotación alrededor del eje vertical de una aeronave.

Z

Zip. Formato de archivo que comprime uno o más archivos en un solo archivo.

Zona Horaria. Región de la Tierra que sigue la misma hora estándar.

Zulu. Zona horaria utilizada en la aviación y en la navegación marítima.

18. Anexos

18.1. Normas, Reglamentos y Manuales

Regulaciones que rigen la instalación y operación de estaciones meteorológicas automáticas, así como la recopilación y uso de datos meteorológicos.

- **A Guide to Standards and Best Practices.** Guía de estándares y mejores prácticas para la instalación y operación de estaciones meteorológicas automáticas. (ICAMS, s. f.)
- **AC 00-45H.** Advisory Circular. Aviation Weather Services. (FAA, 2016)
- **Manual de Claves.** Manual de claves de la Organización Meteorológica Mundial. (OMM, 2011)
- **METAR and TAF Abbreviations.** Abreviaturas utilizadas en los informes METAR y TAF. (Weather.gov, 2008)
- **METAR Text to Symbol.** Conversión de texto METAR a símbolos meteorológicos. (AWC, s. f.)
- **NMX-AA-166/1-SCFI-2013.** Norma Mexicana. Estaciones Meteorológicas, Climatológicas e Hidrológicas. Parte 1: Especificaciones técnicas que deben cumplir los instrumentos de medición de las estaciones meteorológicas automáticas y convencionales. (México, 2013)
- **NMX-AA-166/2-SCFI-2015.** Norma Mexicana. Estaciones Meteorológicas, Climatológicas e Hidrológicas. Parte 2: Especificaciones técnicas que deben cumplir el emplazamiento y exposición de los instrumentos de medición de las estaciones meteorológicas automáticas y convencionales. (México, 2015)
- **Guía de Instrumentos y Métodos de Observación.** Guía de instrumentos y métodos de observación de la Organización Meteorológica Mundial. (WMO, 2018)
- **Guía del Sistema Mundial de Observación.** Guía del Sistema Mundial de Observación de la Organización Meteorológica Mundial. (OMM, 2010)
- **Guide to Instruments and Methods of Observation.** Guía de instrumentos y métodos de observación de la Organización Meteorológica Mundial. (WMO, 2021)
- **Training Guide in Surface Weather Observations.** Guía de entrenamiento en observaciones meteorológicas de superficie de la Organización Meteorológica Mundial. (NWS, 1998)

18.2. Reportes

Breve decodificación de los reportes METAR, TAF y NOTAM.

18.2.1. METAR

Referencia 9. Ejemplo de reporte METAR.

```
MMLO 020455Z 00000KT 8SM SCT220 24/M04 A3011 RMK 8/002
```

METAR decodificado.

- **Estación:** MMLO (Aeropuerto Internacional del Bajío)
- **Fecha y hora:** 02 de mayo de 2024, 04:55Z
- **Temperatura:** 24.0°C (75°F)
- **Punto de rocío:** -4.0°C (25°F) [HR = 15%]
- **Presión atmosférica (altímetro):** 30.11 pulgadas Hg (1019.7 mb)
- **Viento:** Calma
- **Visibilidad:** 8 millas (13 km)
- **Techo:** al menos 12,000 pies AGL (Sobre el nivel del suelo)
- **Nubes:** nubes dispersas a 22,000 pies AGL (Sobre el nivel del suelo)

18.2.2. TAF

Referencia 10. Ejemplo de reporte TAF.

```
TAF MMLO 020430Z 0206/0306 22010KT P6SM BKN200  
FM021900 24015KT P6SM BKN200  
FM030400 24008KT 6SM HZ BKN200
```

TAF decodificado:

- **Estación:** MMLO (Aeropuerto Internacional del Bajío)
- **Fecha y hora:** 02 de mayo de 2024, 04:30Z

- **Período de pronóstico:** Desde las 06:00 UTC del 02 de mayo de 2024 hasta las 06:00 UTC del 03 de mayo de 2024
 - **Vientos:** Desde el suroeste (220 grados) a 10 nudos (18.5 km/h)
 - **Visibilidad:** Más de 6 millas (10 km)
 - **Nubes:** Nubes rotas a 20,000 pies AGL (Sobre el nivel del suelo)
-
- **Cambios esperados:** A partir de las 19:00 UTC del 02 de mayo de 2024
 - **Vientos:** Desde el oeste-suroeste (240 grados) a 15 nudos (27.8 km/h)
 - **Visibilidad:** Más de 6 millas (10 km)
 - **Nubes:** Nubes rotas a 20,000 pies AGL (Sobre el nivel del suelo)
-
- **Cambios esperados:** A partir de las 04:00 UTC del 03 de mayo de 2024
 - **Vientos:** Desde el oeste-suroeste (240 grados) a 8 nudos (14.8 km/h)
 - **Visibilidad:** 6 millas (10 km)
 - **Clima:** Neblina
 - **Nubes:** Nubes rotas a 20,000 pies AGL (Sobre el nivel del suelo)

18.2.3. NOTAM

Referencia 11. Ejemplo de reporte NOTAM.

A1234/06 NOTAMR A1212/06
Q)EGTT/QMXLC/IV/NB0/A/000/999/5129N00028W005
A)EGLL
B)0609050500
C)0704300500
E)DUE WIP TWY B SOUTH CLSD BTN 'F' AND 'R'. TWY 'R' CLSD BTN 'A' AND 'B' AND DIVERTED VIA NEW GREEN CL AND BLUE EDGE LGT. CTN ADZ

NOTAM decodificado:

- **Serie y número:** A1234 emitido en 2006 (06) Naturaleza del NOTAM: Reemplazando (R) NOTAM A1212 emitido en 2006 (06)
- **FIR:** FIR de Londres (EGTT)
- **Asunto:** Calle de rodaje (MX)
- **Condición:** Cerrada (LC)
- **Tráfico:** NOTAM emitido para vuelos IFR (I) y vuelos VFR (V)

- **Propósito:** NOTAM seleccionado para atención inmediata de los miembros de la tripulación de vuelo (N), para entrada en el PIB (Boletín de Información Pre-vuelo) (B), relacionado con operaciones de vuelo (O)
 - **Alcance:** Aeródromo (A)
 - **Límites:** FL 000 a FL 999 (000/999)
 - **Ubicación geográfica:** 51°29' N 000° 28' W (5129N00028W)
 - **Radio de operación del NOTAM:** 5 NM (005)
-
- **Aeródromo:** Londres Heathrow (EGLL)
 - **Desde:** 05:00 UTC 5 de septiembre de 2006 (060905 0500)
 - **Hasta:** 05:00 UTC 30 de abril de 2007 (070430 0500)
 - **Categoría:** Aeródromos, rutas aéreas y ayudas terrestres
 - **Descripción:** Debido a trabajos en curso (DUE WIP), la calle de rodaje "B Sur" está cerrada entre "F" y "R" (TWY B SOUTH CLSD BTN "F" AND "R"). La calle de rodaje "R" está cerrada entre "A" y "B" (TWY "R" CLSD BTN "A" AND "B") y se desvía a través de una nueva línea central verde y luces azules en los bordes (AND DIVERTED VIA NEW GREEN CL AND BLUE EDGE LGT). Se recomienda precaución (CTN ADZ).

18.3. Scripts y Códigos

18.3.1. ESP32/BM680

18.3.1.1. Blink

Código de prueba para encender y apagar un LED conectado a la placa ESP32.

Blink/Blink.ino

.ino

```
int LED_BUILTIN = 2; // Set LED_BUILTIN pin
void setup() { // Set LED_BUILTIN pin as output
    pinMode (LED_BUILTIN, OUTPUT);
}
void loop() { // Blink LED_BUILTIN
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

18.3.1.2. BME680

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32.

Las librerías usadas son y se encuentran disponibles en:

- Adafruit_BME680 (https://github.com/adafruit/Adafruit_BME680)
- Adafruit_Sensor (https://github.com/adafruit/Adafruit_Sensor)

Wire y SPI son librerías de Arduino para comunicación I2C y SPI respectivamente.

BME680/BME680.ino

.ino

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
```

```
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C

void setup() {
    Serial.begin(115200);
    while (!Serial);
    Serial.println(F("BME680 async test"));

    if (!bme.begin()) {
        Serial.println(F("Could not find a valid BME680 sensor, check
wiring!"));
        while (1);
    }

    // Set up oversampling and filter initialization
    bme.setTemperatureOversampling(BME680_OS_8X);
    bme.setHumidityOversampling(BME680_OS_2X);
    bme.setPressureOversampling(BME680_OS_4X);
    bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
    bme.setGasHeater(320, 150); // 320*C for 150 ms
}

void loop() {
    // Tell BME680 to begin measurement.
    unsigned long endTime = bme.beginReading();
    if (endTime == 0) {
        Serial.println(F("Failed to begin reading :("));
        return;
    }
    Serial.print(F("Reading started at "));
    Serial.print(millis());
    Serial.print(F(" and will finish at "));
    Serial.println(endTime);

    Serial.println(F("You can do other work during BME680 measurement."));
    delay(50); // This represents parallel work.
    // There's no need to delay() until millis() >= endTime:
    bme.endReading()
    // takes care of that. It's okay for parallel work to take longer than
    // BME680's measurement time.

    // Obtain measurement results from BME680. Note that this operation
    isn't
    // instantaneous even if millis() >= endTime due to I2C/SPI latency.
    if (!bme.endReading()) {
```

```
Serial.println(F("Failed to complete reading :("));
    return;
}
Serial.print(F("Reading completed at "));
Serial.println(millis());

Serial.print(F("Temperature = "));
Serial.print(bme.temperature);
Serial.println(F(" *C"));

Serial.print(F("Pressure = "));
Serial.print(bme.pressure / 100.0);
Serial.println(F(" hPa"));

Serial.print(F("Humidity = "));
Serial.print(bme.humidity);
Serial.println(F(" %"));

Serial.print(F("Gas = "));
Serial.print(bme.gas_resistance / 1000.0);
Serial.println(F(" KOhms"));

Serial.print(F("Approx. Altitude = "));
Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
Serial.println(F(" m"));

Serial.println();
delay(2000);
}
```

18.3.1.3. BME680 - OLED

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32 y mostrarlos en una pantalla OLED conectada a la misma placa.

Las librerías usadas son y se encuentran disponibles en:

- Adafruit_BME680 (https://github.com/adafruit/Adafruit_BME680)
- Adafruit_GFX (<https://github.com/adafruit/Adafruit-GFX-Library>)
- Adafruit_Sensor (https://github.com/adafruit/Adafruit_Sensor)
- Adafruit_SSD1306 (https://github.com/adafruit/Adafruit_SSD1306)

BME680/BME680-OLED.ino

.ino

/

This is a library for the BME680 gas, humidity, temperature & pressure sensor

Designed specifically to work with the Adafruit BME680 Breakout
----> <http://www.adafruit.com/products/3660>

These sensors use I2C or SPI to communicate, 2 or 4 pins are required to interface.

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing products
from Adafruit!

Written by Limor Fried & Kevin Townsend for Adafruit Industries.
BSD license, all text above must be included in any redistribution

******/

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK);

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

void setup() {
  Serial.begin(9600);
  Serial.println(F("BME680 test"));
```

```
// by default, we'll generate the high voltage from the 3.3v line
internally! (neat!)
display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with the I2C
addr 0x3C (for the 128x32)
// init done
display.display();
delay(100);
display.clearDisplay();
display.display();
display.setTextSize(1);
display.setTextColor(WHITE);

if (!bme.begin()) {
    Serial.println("Could not find a valid BME680 sensor, check wiring!");
    while (1);
}

// Set up oversampling and filter initialization
bme.setTemperatureOversampling(BME680_OS_8X);
bme.setHumidityOversampling(BME680_OS_2X);
bme.setPressureOversampling(BME680_OS_4X);
bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
bme.setGasHeater(320, 150); // 320*C for 150 ms
}

void loop() {
    display.setCursor(0,0);
    display.clearDisplay();

    if (! bme.performReading()) {
        Serial.println("Failed to perform reading :(");
        return;
    }
    Serial.print("Temperature = "); Serial.print(bme.temperature);
    Serial.println(" *C");
    display.print("Temperature: "); display.print(bme.temperature);
    display.println(" *C");

    Serial.print("Pressure = "); Serial.print(bme.pressure / 100.0);
    Serial.println(" hPa");
    display.print("Pressure: "); display.print(bme.pressure / 100);
    display.println(" hPa");

    Serial.print("Humidity = "); Serial.print(bme.humidity);
    Serial.println(" %");
```

```
display.print("Humidity: "); display.print(bme.humidity);
display.println(" %");

Serial.print("Gas = "); Serial.print(bme.gas_resistance / 1000.0);
Serial.println(" KOhms");
display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);
display.println(" KOhms");

Serial.println();
display.display();
delay(2000);
}
```

18.3.1.4. BME680 - OLED - PocketBase

Código de prueba para obtener datos de temperatura, humedad, presión y calidad del aire del sensor BME680 conectado a la placa ESP32 y mostrarlos en una pantalla OLED conectada a la misma placa. Además, los datos son enviados a una base de datos PocketBase en un servidor remoto.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

BME680/BME680-OLED-PocketBase.ino

```
.ino

/
*****
This is a library for the BME680 gas, humidity, temperature & pressure
sensor

Designed specifically to work with the Adafruit BME680 Breakout
----> http://www.adafruit.com/products/3660

These sensors use I2C or SPI to communicate, 2 or 4 pins are required
to interface.

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products
from Adafruit!

Written by Limor Fried & Kevin Townsend for Adafruit Industries.
BSD license, all text above must be included in any redistribution
```

```
***** */\n\n#include <Wire.h>\n#include <SPI.h>\n#include <Adafruit_Sensor.h>\n#include "Adafruit_BME680.h"\n#include <Adafruit_GFX.h>\n#include <Adafruit_SSD1306.h>\n#include <WiFi.h>\n#include <sys/time.h>\n#include <HTTPClient.h>\n#include <esp_sleep.h>\n\n#define LED 2\n\n#define BME_SCK 13\n#define BME_MISO 12\n#define BME_MOSI 11\n#define BME_CS 10\n\n#define SEALEVELPRESSURE_HPA (1013.25)\n\nAdafruit_BME680 bme; // I2C\n\nAdafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);\n\nString timeUTC;\nbool wifiConnected = true;\n\nconst char* ssid = "Hogwarts"; // Network SSID\nconst char* password = "zV9%E^%tJNd!yaW*"; // Network password\n\nconst char* ntpServer = "pool.ntp.org"; // NTP server\nconst long gmtOffset_sec = 0; // Offset from GMT\nconst int daylightOffset_sec = 0; // Offset from daylight savings time\n\nvoid setup() {\n    Serial.begin(9600);\n    Serial.println("Starting BME680...");\n\n    Serial.println("Starting Wi-Fi..."); // Print a message to the serial\nmonitor\n    wifiConnected = connectToWifi(); // Obtain the Wi-Fi connection status\n\n    if (wifiConnected) {\n\n        // Your code here...\n\n    }\n}\n\nvoid loop() {\n    // Your code here...\n}\n\n// Function prototypes and definitions go here...\n\n// Example function\nvoid exampleFunction() {\n    // Your code here...\n}\n\n// Example function\nvoid anotherFunction() {\n    // Your code here...\n}
```

```
    Serial.println("Connection to Wi-Fi successful"); // Print a
message to the serial monitor
    Serial.println("Starting time sync..."); // Print a message to the
serial monitor
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    while (!time(nullptr)) {
        delay(1000);
        Serial.println("Waiting for time sync...");
    }
    Serial.println("Time synced");
} else {
    Serial.println("Connection to Wi-Fi failed"); // Print a message
to the serial monitor
    Serial.println("Proceeding without Wi-Fi..."); // Print a message
to the serial monitor
}

// by default, we'll generate the high voltage from the 3.3v line
internally! (neat!)
display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with the I2C
addr 0x3C (for the 128x32)
// init done
display.display();
delay(100);
display.clearDisplay();
display.display();
display.setTextSize(1);
display.setTextColor(WHITE);

if (!bme.begin()) {
    Serial.println("Could not find a valid BME680 sensor, check
wiring!");
    while (1); // Freeze the program
}

// Set up oversampling and filter initialization
bme.setTemperatureOversampling(BME680_OS_8X);
bme.setHumidityOversampling(BME680_OS_2X);
bme.setPressureOversampling(BME680_OS_4X);
bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
bme.setGasHeater(320, 150); // 320*C for 150 ms

pinMode(LED, OUTPUT); // Set the LED pin as an output

Serial.println("Setup complete");
```

```
}

void loop() {
    digitalWrite(LED, HIGH); // Turn the LED on (Note that LOW is the
voltage level

    display.setCursor(0,0);
    display.clearDisplay();

    if (!bme.performReading()) {
        Serial.println("Failed to perform BME680 reading");
        return;
    }

    printToSerial(); // Print to serial monitor
    printToDisplay(); // Print to OLED display

    if (!wifiConnected) { // If Wi-Fi is not connected, wait 10 minutes
and try again
        Serial.println("Retrying in 10 minutes...");
        digitalWrite(LED, LOW); // Turn the LED off by making the voltage
HIGH
        delay(10 * 60 * 1000); // Wait 10 minutes

        esp_restart(); // Restart the ESP32
    }

    timeUTC = getUTCTime(); // Get UTC time from NTP server

    sendToPocketBase(); // Send data to PocketBase

    Serial.println("Updating in 10 minutes...");
    digitalWrite(LED, LOW); // Turn the LED off by making the voltage HIGH

    delay(10 * 60 * 1000); // Wait 10 minutes
}

void printToSerial() { // Print to serial monitor
    Serial.print("Temperature = "); Serial.print(bme.temperature);
    Serial.println(" *C");
    Serial.print("Pressure = "); Serial.print(bme.pressure / (20 *
133.32239)); Serial.println(" inHg");
    Serial.print("Humidity = "); Serial.print(bme.humidity);
    Serial.println(" %");
    Serial.print("Gas = "); Serial.print(bme.gas_resistance / 1000.0);
    Serial.println(" KOhms");
```

```
}

void printToDisplay() { // Print to OLED display
    display.setCursor(0,0);
    display.clearDisplay();
    display.print("Temperature: "); display.print(bme.temperature);
    display.println(" *C");
    display.print("Pressure: "); display.print(bme.pressure / (20 *
133.32239)); display.println(" inHg");
    display.print("Humidity: "); display.print(bme.humidity);
    display.println(" %");
    display.print("Gas: "); display.print(bme.gas_resistance / 1000.0);
    display.println(" KOhms");
    display.display();
}

bool connectToWifi() { // Connect to the Wi-Fi network
    Serial.print("Connecting to ");
    Serial.println(ssid);

    unsigned long startTime = millis(); // Get the current time

    WiFi.begin(ssid, password); // Connect to the network

    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to
connect
        delay(500);
        Serial.print(".");
        if (millis() - startTime > 60000) { // If it's been more than 1
minute
            Serial.println("");
            Serial.println("WiFi connection timed out");
            return false; // Return false
        }
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP()); // Print the local IP address

    return true; // Return true if connection was successful
}

String getUTCTime() { // Get UTC time from NTP server
    struct timeval tv;
```

```
gettimeofday(&tv, nullptr);
time_t now = tv.tv_sec;
struct tm timeinfo;
gmtime_r(&now, &timeinfo);
char buffer[30];
snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.
%03ldZ",
         timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
         timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
         tv.tv_usec / 1000);
return String(buffer);
}

void sendToPocketBase() { // Send data to PocketBase
    Serial.println("Sending data to PocketBase...");

    HttpClient http;

    http.begin("https://w.arias.pw/api/collections/bme680/records"); // Specify the URL
    http.addHeader("Content-Type", "application/json"); // Specify content-type header

    // Create the JSON payload
    String payload = "{\"time\": " + timeUTC + ", \"temperature\": " +
        bme.temperature + ", \"pressure\": " + bme.pressure / (20 *
        133.32239) + ", \"humidity\": " + bme.humidity + ", \"gas\": " +
        bme.gas_resistance / 1000.0 + "}";
}

int httpCode = http.POST(payload); // Send the request

if(httpCode == 200) { // Check the returning code
    Serial.println("Data sent to PocketBase successfully");
} else { // If the code is not 200, something went wrong
    Serial.print("Error sending data to PocketBase, returned code: ");
    Serial.println(httpCode);

    Serial.println("Restarting ESP32...");
    delay(1000); // Wait for the serial output to finish
    esp_restart(); // Restart the ESP32
}

http.end(); // Close connection
}
```

18.3.2. Estación Meteorológica

18.3.2.1. Blink

Código de prueba para encender y apagar el LED integrado en la placa.

Blink/Blink.ino

.ino

```
// Blink a LED on the MicroMod Weather (ESP32) board

int ledPin = 2; // LED is connected to GPIO2

void setup() {
    pinMode(ledPin, OUTPUT); // Set GPIO2 to output mode
    Serial.begin(115200); // Initialize serial port
}

void loop() {
    digitalWrite(ledPin, HIGH); // Turn LED on
    delay(1000); // Wait for 1000 millisecond(s)
    Serial.println("The LED is on."); // Print a message
    digitalWrite(ledPin, LOW); // Turn LED off
    delay(1000); // Wait for 1000 millisecond(s)
}
```

18.3.2.2. BME280

Código para leer los datos del sensor BME280.

BME280/BME280.ino

.ino

```
#include <Wire.h>
#include "SparkFunBME280.h"

BME280 bme280Sensor; // Create BME280 object

float RealFloatPressure;

void setup() {
    Serial.begin(115200); // Initialize serial port
    while (!Serial); // Wait for user to open serial monitor
```

```
Serial.println("MicroMod Weather Carrier Board - BME280 Example");
Serial.println();

Wire.begin(); // Join I2C bus

bme280Sensor.setReferencePressure(101500); // Set sea level pressure
to 101325 Pa (default)

if (bme280Sensor.begin() == false) { // Connect to BME280
    Serial.println("BME280 did not respond.");
    while(1); // Freeze
}

pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.print("Temperature: ");
    Serial.println(bme280Sensor.readTempC(), 2);
    Serial.print("Humidity: ");
    Serial.println(bme280Sensor.readFloatHumidity(), 0);
    Serial.print("Pressure: ");

    RealFloatPressure = bme280Sensor.readFloatPressure() / (20 * 133.32239);
    Serial.println(RealFloatPressure, 2);

    Serial.print("Altitude: ");
    Serial.println(bme280Sensor.readFloatAltitudeMeters(), 1);
    Serial.print("Dewpoint: ");
    Serial.println(bme280Sensor.dewPointC(), 2);

    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

18.3.2.3. VEML6075

Código para leer los datos del sensor VEML6075.

VEML6075/VEML6075.ino

.ino

```
#include <SparkFun_VEML6075_Arduino_Library.h>
```

```
VEML6075 veml6075; // Create a VEML6075 object

void setup() {
    Serial.begin(115200);
    while(!Serial); // Wait for user to open serial monitor

    Serial.println("MicroMod Weather Carrier Board - VEML6075 Example");

    Wire.begin(); // Join I2C bus

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond."); // If the sensor does not
        respond, print an error message
        while(1); // Freeze
    }

    pinMode(LED_BUILTIN, OUTPUT);
    Serial.println("UVA, UVB, UV Index"); // Print the header for the data
    Serial.println();

}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    // Print the UVA, UVB, and UV Index values
    Serial.println("UVA: " + String(veml6075.uva()));
    Serial.println("UVB: " + String(veml6075.uvb()));
    Serial.println("UV Index: " + String(veml6075.index()));

    digitalWrite(LED_BUILTIN, LOW);
    delay(1000); // Wait 1 second
}
```

18.3.2.4. Station

Código para leer los datos de los sensores en conjunto.

Station/Station.ino

.ino

```
// Station code for the SparkFun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start
// It should print out the weather data to the serial monitor each
duration minutes
```

```
#include <Wire.h>
#include "SparkFunBME280.h"
#include <SparkFun_VEML6075_Arduino_Library.h>

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

const int duration = 1; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the
remaining minutes and seconds

volatile int windSpeedCount = 0; // Variable to store the number of wind
pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y
values
float x, y, theta, averageWindDirection, averageWindSpeed; // Variables to
store the x, y, theta, and average wind direction and speed

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
    float uvindex;
};

WeatherData weather;

struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};

WindData wind[duration * 6];
```

```
void setup() {
    Serial.begin(115200); // Start the serial monitor
    while (!Serial); // Wait for user to open serial monitor

    Wire.begin(); // Join the I2C bus

    if (bme280.begin() == false) { // Connect to the BME280
        Serial.println("BME280 did not respond."); // Print an error
        message if the BME280 does not respond
        while(1); // Freeze
    }

    if (veml6075.begin() == false) {
        Serial.println("VEML6075 did not respond.");
        while(1);
    }

    pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
    pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as
    an input
    pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
    attachInterrupt(digitalPinToInterruption(windSpeedSensor), windSpeedIRQ,
    FALLING); // Attach the wind speed interrupt
    attachInterrupt(digitalPinToInterruption(rainSensor), rainIRQ,
    FALLING); // Attach the rain interrupt
    interrupts(); // Enable interrupts

    Serial.println("Station complete"); // Print a message to the serial
    monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED
    rainCount = 0; // Reset the rain count
    for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction
    array
        wind[i].reading = false;
        wind[i].direction = 0;
        wind[i].speed = 0;
    }
    for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of
    the report, taking measurements every 10 seconds
        Serial.print("Taking wind measurements, this process will be
        finished in: ");
    } // Print a message to the serial monitor to let the user
```

```
know the program is working
    remainingMinutes = duration - ((i / 6) + !(i % 6));
    remainingSeconds = 60 - ((i % 6) * 10);
    Serial.print(remainingMinutes);
    Serial.print(":");
    if (remainingSeconds == 60) {
        Serial.println("00");
    } else {
        Serial.println(remainingSeconds);
    }

    windSpeedCount = 0; // Reset the wind speed count
    delay(10 * 1000); // Wait for 10 seconds
    wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the
wind speed
    wind[i].direction = getWindDirection(); // Calculate the wind
direction
    wind[i].reading = true; // Set the reading flag to true
}
weather.rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate
the rain fall in inches/hour
getAverageWind(averageWindDirection, averageWindSpeed); // Get the
average wind direction and speed
weather.windDirection = averageWindDirection; // Set the average wind
direction to the weather data
weather.windSpeed = averageWindSpeed; // Set the average wind speed to
the weather data
weather.temperature = bme280.readTempC(); // Get the temperature in
degrees Celsius
weather.humidity = bme280.readFloatHumidity(); // Get the humidity in
percent
weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees
Celsius
weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); // Convert the pressure from Pascals to inches of mercury

weather.uva = veml6075.uva(); // Get the UVA value
weather.uvb = veml6075.uvb(); // Get the current time in UTC
weather.uvindex = veml6075.index(); // Get the UV index

printWeather(); // Print the weather data to the serial monitor

digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}

void printWeather() {
```

```

Serial.print("Temperature: ");
Serial.print(weather.temperature, 2); // Temperature in degrees
Celsius
Serial.print("    Humidity: ");
Serial.print(weather.humidity, 2); // Humidity in percent
Serial.print("    Dewpoint: ");
Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
Serial.print("    Pressure: ");
Serial.print(weather.pressure, 2); // Pressure in inches of mercury
Serial.print("    Wind Speed: ");
Serial.print(weather.windSpeed, 2); // Wind speed in knots
Serial.print("    Wind Direction: ");
Serial.print(weather.windDirection, 2); // Average wind direction in
degrees
Serial.print("    Rain Fall: ");
Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
Serial.print("    UVA: ");
Serial.print(weather.uva); // UVA value
Serial.print("    UVB: ");
Serial.print(weather.uvb); // UVB value
Serial.print("    UV Index: ");
Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value
from the wind direction sensor

    if ( 10 < reading && reading <= 150) return (288);
    if ( 150 < reading && reading <= 250) return (264);
    if ( 250 < reading && reading <= 400) return (336);
    if ( 400 < reading && reading <= 600) return (312);
    if ( 650 < reading && reading <= 850) return ( 24);
    if ( 850 < reading && reading <= 1050) return (  0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}

```

```
void getAverageWind(float& averageWindDirection, float& averageWindSpeed)
{
    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // convert angle to radians
        x = wind[i].speed * cos(theta);
        y = wind[i].speed * sin(theta);
        xSum += x;
        ySum += y;
    }
    averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum /
(duration * 6))); // convert radians to degrees
    if (averageWindDirection < 0) averageWindDirection += 360; // convert
negative angles to positive
    averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum /
(duration * 6), 2)); // calculate average speed

    xSum = 0;
    ySum = 0;
}

void rainIRQ()
{
    rainCount++;
    // Serial.println("Rain clicked");
}

// Function is called when the magnet in the anemometer is activated
void windSpeedIRQ()
{
    windSpeedCount++;
    // Serial.println("Wind clicked");
}
```

18.3.2.5. Station - PocketBase

Código para leer los datos de los sensores en conjunto y enviar los datos a la base de datos de PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

Station_PocketBase/Station_PocketBase.ino

.ino

```
// Station code for the Sparkfun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start
// It should print out the weather data to the serial monitor each
duration minutes
// It should also send the weather data to a PocketBase server every 5
minutes

#include <Wire.h>
#include "SparkFunBME280.h"
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <SparkFun_VEML6075_Arduino_Library.h>

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

const int duration = 4; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the
remaining minutes and seconds

String currentTime; // Variable to store the current time
String minuteString; // Variable to store the minutes from the time
int minuteInt; // Variable to store the minutes as an int

volatile int windSpeedCount = 0; // Variable to store the number of wind
pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y
values
float x, y, theta, averageWindDirection, averageWindSpeed; // Variables to
store the x, y, theta, and average wind direction and speed

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
```

```
float dewpoint;
float pressure;
float rainFall;
float windSpeed;
float windDirection;
float uva;
float uvb;
float uvindex;
};

WeatherData weather;

struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};

WindData wind[duration * 6];

const char* ssid = "Hogwarts"; // Network SSID
const char* password = "zV9%E^%tJNd!yaW*"; // Network password

const char* ntpServer = "pool.ntp.org"; // NTP server
const long gmtOffset_sec = 0; // Offset from GMT
const int daylightOffset_sec = 0; // Offset from daylight savings time

void setup() {
    Serial.begin(115200); // Start the serial monitor
    while (!Serial); // Wait for user to open serial monitor

    connectToWifi(); // Connect to the Wi-Fi network

    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    while (!time(nullptr)) {
        delay(1000);
        Serial.println("Waiting for time sync...");
    }
    Serial.println("Time synced");

    Wire.begin(); // Join the I2C bus

    if (bme280.begin() == false) { // Connect to the BME280
        Serial.println("BME280 did not respond."); // Print an error
message if the BME280 does not respond
        while(1); // Freeze
    }
}
```

```
if (veml6075.begin() == false) {
    Serial.println("VEML6075 did not respond.");
    while(1); // Freeze
}

pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as
an input
pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
attachInterrupt(digitalPinToInterruption(windSpeedSensor), windSpeedIRQ,
FALLING); // Attach the wind speed interrupt
attachInterrupt(digitalPinToInterruption(rainSensor), rainIRQ,
FALLING); // Attach the rain interrupt
interrupts(); // Enable interrupts

Serial.println("Setup complete"); // Print a message to the serial
monitor
Serial.println("Waiting for minute ending in 1 or 6 to start
report"); // Print a message to the serial monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED

    minuteInt = getMinute(); // Get the current minute

    if (minuteInt % 5 == 1) {
        Serial.println("Starting report..."); // Print a message to the
        serial monitor
        rainCount = 0; // Reset the rain count
        for (int i = 0; i < (duration * 6); i++) { // Reset the wind
        direction array
            wind[i].reading = false;
            wind[i].direction = 0;
            wind[i].speed = 0;
        }
        for (int i = 0; i < (duration * 6); i++) { // Loop for the
        duration of the report, taking measurements every 10 seconds
            Serial.print("Taking wind measurements, this process will be
            finished in: "); // Print a message to the serial monitor to let the user
            know the program is working
            remainingMinutes = duration - ((i / 6) + !(i % 6));
            remainingSeconds = 60 - ((i % 6) * 10);
            Serial.print(remainingMinutes);
            Serial.print(":");
            if (remainingSeconds == 60) {
```

```
        Serial.println("00");
    } else {
        Serial.println(remainingSeconds);
    }

    windSpeedCount = 0; // Reset the wind speed count
    delay(10 * 1000); // Wait for a minute
    wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate
the wind speed
    wind[i].direction = getWindDirection(); // Calculate the wind
direction
    wind[i].reading = true; // Set the reading flag to true
}
weather.rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate
the rain fall in inches/hour
getAverageWind(averageWindDirection, averageWindSpeed); // Get the
average wind direction and speed
weather.windDirection = averageWindDirection; // Set the average
wind direction to the weather data
weather.windSpeed = averageWindSpeed; // Set the average wind
speed to the weather data
weather.temperature = bme280.readTempC(); // Get the temperature
in degrees Celsius
weather.humidity = bme280.readFloatHumidity(); // Get the humidity
in percent
weather.dewpoint = bme280.dewPointC(); // Get the dew point in
degrees Celsius
weather.pressure = bme280.readFloatPressure() / (20 *
133.32239); // Convert the pressure from Pascals to inches of mercury

weather.uva = veml6075.uva(); // Get the UVA value
weather.uvb = veml6075.uvb(); // Get the current time in UTC
weather.uvindex = veml6075.index(); // Get the UV index

weather.time = getUTCTime(); // Get the current time in UTC
printWeather(); // Print the weather data to the serial monitor
sendWeatherDataToPocketBase(); // Send the weather data to
PocketBase
}
delay(1000); // Wait for a second
digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}

String getUTCTime() {
    struct timeval tv;
    gettimeofday(&tv, nullptr);
```

```
time_t now = tv.tv_sec;
struct tm timeinfo;
gmtime_r(&now, &timeinfo);
char buffer[30];
snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.%03ldZ",
         timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
         timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
         tv.tv_usec / 1000);
return String(buffer);
}

int getMinute() { // Get the current minute
currentTime = getUTCTime();
minuteString = currentTime.substring(14, 16);
minuteInt = minuteString.toInt();
return minuteInt;
}

void connectToWifi() {
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password); // Connect to the network

while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to
connect
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP()); // Print the local IP address
}

void printWeather() {
Serial.print("Time: ");
Serial.print(weather.time); // Time in UTC
Serial.print("Z Temperature: ");
Serial.print(weather.temperature, 2); // Temperature in degrees
Celsius
Serial.print("Humidity: ");
Serial.print(weather.humidity, 2); // Humidity in percent
Serial.print("Dewpoint: ");
}
```

```
Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
Serial.print("  Pressure: ");
Serial.print(weather.pressure, 2); // Pressure in inches of mercury
Serial.print("  Wind Speed: ");
Serial.print(weather.windSpeed, 2); // Wind speed in knots
Serial.print("  Wind Direction: ");
Serial.print(weather.windDirection, 2); // Average wind direction in
degrees
Serial.print("  Rain Fall: ");
Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
Serial.print("  UVA: ");
Serial.print(weather.uva); // UVA value
Serial.print("  UVB: ");
Serial.print(weather.uvb); // UVB value
Serial.print("  UV Index: ");
Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value
from the wind direction sensor

    if ( 10 < reading && reading <= 150) return (288);
    if ( 150 < reading && reading <= 250) return (264);
    if ( 250 < reading && reading <= 400) return (336);
    if ( 400 < reading && reading <= 600) return (312);
    if ( 650 < reading && reading <= 850) return ( 24);
    if ( 850 < reading && reading <= 1050) return ( 0);
    if (1300 < reading && reading <= 1500) return (216);
    if (1500 < reading && reading <= 1700) return (240);
    if (2000 < reading && reading <= 2200) return ( 72);
    if (2200 < reading && reading <= 2400) return ( 48);
    if (2400 < reading && reading <= 2600) return (168);
    if (2800 < reading && reading <= 3000) return (192);
    if (3000 < reading && reading <= 3200) return (120);
    if (3300 < reading && reading <= 3500) return (144);
    if (3700 < reading && reading <= 3900) return ( 96);
    return (-1);
}

void getAverageWind(float& averageWindDirection, float& averageWindSpeed)
{
    for (int i = 0; i < (duration * 6); i++) {
        theta = radians(wind[i].direction); // convert angle to radians
        x = wind[i].speed * cos(theta);
```

```
y = wind[i].speed * sin(theta);
xSum += x;
ySum += y;
}
averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum /
(duration * 6))); // convert radians to degrees
if (averageWindDirection < 0) averageWindDirection += 360; // convert
negative angles to positive
averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum /
(duration * 6), 2)); // calculate average speed

xSum = 0;
ySum = 0;
}

void rainIRQ()
{
    rainCount++;
    // Serial.println("Rain clicked");
}

// Function is called when the magnet in the anemometer is activated
void windSpeedIRQ()
{
    windSpeedCount++;
    // Serial.println("Wind clicked");
}

void sendWeatherDataToPocketBase() {
    Serial.println("Sending weather data to PocketBase...");
    HttpClient http;

    // Set the PocketBase endpoint URL
    http.begin("https://w.arias.pw/api/collections/station/records");

    // Set the HTTP headers
    http.addHeader("Content-Type", "application/json");

    // Create the JSON payload
    String payload = "{\"time\": \"" + weather.time +
                    "\", \"temperature\": " + String(weather.temperature) +
                    ", \"humidity\": " + String(weather.humidity) +
                    ", \"dewpoint\": " + String(weather.dewpoint) +
                    ", \"pressure\": " + String(weather.pressure) +
                    ", \"rainFall\": " + String(weather.rainFall) +
```

```
    ", \"windSpeed\":" + String(weather.windSpeed) +
    ", \"windDirection\":" + String(weather.windDirection)
+
    ", \"uva\":" + String(weather.uva) +
    ", \"uvb\":" + String(weather.uvb) +
    ", \"uvindex\":" + String(weather.uvindex) + "}";

// Send the POST request with the payload
int httpCode = http.POST(payload);

// Check if the request was successful
if(httpCode == 200) {
    Serial.println("Data sent to PocketBase successfully");
} else {
    Serial.println("Error sending data to PocketBase");
    Serial.print("HTTP code: ");
    Serial.println(httpCode);
}

// Free resources
http.end();
}
```

18.3.2.6. Station - PocketBase - SD

Código para leer los datos de los sensores en conjunto, enviar los datos a la base de datos de PocketBase y guardarlos en una tarjeta SD.

El siguiente código no funciona directamente, las variables de conexión a la base de datos y la red WiFi deben ser reemplazadas por las correspondientes.

Station_PocketBase_SD/Station_PocketBase_SD.ino

```
.ino

// Station code for the Sparkfun MicroMod Weather (ESP32) project
// It's missing AS3935 integration, but it's a good start

#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include "SparkFunBME280.h"
#include <WiFi.h>
#include <sys/time.h>
#include <HTTPClient.h>
#include <SparkFun_VEML6075_Arduino_Library.h>
```

```
#include <esp_sleep.h>

File dataFile; // File to store the weather data

#if defined(ARDUINO_ARCH_APOLL03)
const int chipSelect = CS;
#else
const int chipSelect = SS;
#endif

BME280 bme280; // Instance of the BME280 class
VEML6075 veml6075; // Create a VEML6075 object

int windDirectionSensor = A1; // Analog pin for wind direction
int windSpeedSensor = D0; // Digital I/O pin for wind speed
int rainSensor = D1; // Digital I/O pin for rain fall

bool wifiConnected = true; // Variable to store the Wi-Fi connection status

const int duration = 10; // Duration of the report in minutes

int remainingMinutes, remainingSeconds; // Variables to store the remaining minutes and seconds

volatile int windSpeedCount = 0; // Variable to store the number of wind pulses
volatile int rainCount = 0; // Variable to store the number of rain tips

float xSum = 0, ySum = 0; // Variables to store the sum of the x and y values
float x, y, theta, averageWindDirection, averageWindSpeed, rainFall; // Variables to store the x, y, theta, average wind direction and speed, and the rain fall

struct WeatherData { // Struct to store the weather data
    String time;
    float temperature;
    float humidity;
    float dewpoint;
    float pressure;
    float rainFall;
    float windSpeed;
    float windDirection;
    float uva;
    float uvb;
```

```
    float uvindex;
};

WeatherData weather;

struct WindData { // Struct to store the wind data
    float speed;
    float direction;
    boolean reading;
};
WindData wind[duration * 6];

const char* ssid = "Hogwarts"; // Network SSID
const char* password = "zV9%E^%tJNd!yaW*"; // Network password

const char* ntpServer = "pool.ntp.org"; // NTP server
const long gmtOffset_sec = 0; // Offset from GMT
const int daylightOffset_sec = 0; // Offset from daylight savings time

void setup() {
    Serial.begin(115200); // Start the serial monitor
    while (!Serial); // Wait for user to open serial monitor

    Serial.println("Starting MicroMod Weather Station..."); // Print a
message to the serial monitor

    Serial.println("Starting SD card..."); // Print a message to the
serial monitor
    if (!SD.begin(chipSelect)) { // Check if the SD card is present
        Serial.println("SD card initialization failed"); // Print a
message to the serial monitor
        while(1); // Wait for the user to fix the problem
    }
    Serial.println("SD card started"); // Print a message to the serial
monitor

    Serial.println("Starting Wi-Fi..."); // Print a message to the serial
monitor
    wifiConnected = connectToWifi(); // Obtain the Wi-Fi connection status

    if (wifiConnected) {
        Serial.println("Connection to Wi-Fi successful"); // Print a
message to the serial monitor
        Serial.println("Starting time sync..."); // Print a message to the
serial monitor
        configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
        while (!time(nullptr)) {

```

```
        delay(1000);
        Serial.println("Waiting for time sync...");
    }
    Serial.println("Time synced");
} else {
    Serial.println("Connection to Wi-Fi failed"); // Print a message
to the serial monitor
    Serial.println("Proceeding without Wi-Fi..."); // Print a message
to the serial monitor
}

Wire.begin(); // Join the I2C bus

Serial.println("Starting BME280...");
if (bme280.begin() == false) { // Connect to the BME280
    Serial.println("BME280 did not respond. Please check your wiring
and try again"); // Print an error message if the BME280 does not respond
    Serial.println("Restarting..."); // Print a message to the serial
monitor
    delay(1000); // Wait for the message to be printed
    esp_restart(); // Reset the ESP32
} else {
    Serial.println("BME280 started"); // Print a message to the serial
monitor
}

if (veml6075.begin() == false) {
    Serial.println("VEML6075 did not respond.");
    Serial.println("Restarting..."); // Print a message to the serial
monitor
    delay(1000); // Wait for the message to be printed
    esp_restart(); // Reset the ESP32
} else {
    Serial.println("VEML6075 started");
}

pinMode(LED_BUILTIN, OUTPUT); // Set the LED pin as an output
pinMode(windSpeedSensor, INPUT_PULLUP); // Set the wind speed pin as
an input
pinMode(rainSensor, INPUT_PULLUP); // Set the rain pin as an input
attachInterrupt(digitalPinToInterruption(windSpeedSensor), windSpeedIRQ,
FALLING); // Attach the wind speed interrupt
attachInterrupt(digitalPinToInterruption(rainSensor), rainIRQ,
FALLING); // Attach the rain interrupt
interrupts(); // Enable interrupts
```

```
Serial.println("Setup complete"); // Print a message to the serial
monitor
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED

    Serial.println("Starting report..."); // Print a message to the serial
monitor

    getWindandRainMeasurements(averageWindDirection, averageWindSpeed,
rainFall); // Get the average wind direction and speed

    weather.windDirection = averageWindDirection; // Set the average wind
direction to the weather data
    weather.windSpeed = averageWindSpeed; // Set the average wind speed to
the weather data
    weather.rainFall = rainFall; // Set the rain fall to the weather data
    weather.temperature = bme280.readTempC(); // Get the temperature in
degrees Celsius
    weather.humidity = bme280.readFloatHumidity(); // Get the humidity in
percent
    weather.dewpoint = bme280.dewPointC(); // Get the dew point in degrees
Celsius
    weather.pressure = bme280.readFloatPressure() / (20 * 133.32239); //
Convert the pressure from Pascals to inches of mercury
    weather.uva = veml6075.uva(); // Get the UVA value
    weather.uvb = veml6075.uvb(); // Get the current time in UTC
    weather.uvindex = veml6075.index(); // Get the UV index

    if (!wifiConnected) {
        weather.time = "1970-01-01 00:00:00.000Z"; // Set the time to "No
Wi-Fi" if the ESP32 is not connected to Wi-Fi
        printWeather(); // Print the weather data to the serial monitor

        writeDataToSDCard(); // Write the weather data to the SD card

        Serial.println("Restarting..."); // Print a message to the serial
monitor
        delay(1000); // Wait for the message to be printed
        esp_restart(); // Reset the ESP32
    }

    weather.time = getUTCTime(); // Get the current time in UTC
    printWeather(); // Print the weather data to the serial monitor
```

```
writeDataToSDCard(); // Write the weather data to the SD card

sendWeatherDataToPocketBase(); // Send the weather data to PocketBase

Serial.println("Report complete"); // Print a message to the serial
monitor to let the user know the program is working
digitalWrite(LED_BUILTIN, LOW); // Turn off the LED
}

String getUTCTime() {
    struct timeval tv;
    gettimeofday(&tv, nullptr);
    time_t now = tv.tv_sec;
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    char buffer[30];
    snprintf(buffer, sizeof(buffer), "%04d-%02d-%02d %02d:%02d:%02d.
%03ldZ",
        timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday,
        timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec,
        tv.tv_usec / 1000);
    return String(buffer);
}

bool connectToWifi() { // Connect to the Wi-Fi network
    Serial.print("Connecting to ");
    Serial.println(ssid);

    unsigned long startTime = millis(); // Get the current time

    WiFi.begin(ssid, password); // Connect to the network

    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to
connect
        delay(500);
        Serial.print(".");
        if (millis() - startTime > 60000) { // If it's been more than 1
minute
            Serial.println("");
            Serial.println("WiFi connection timed out");
            return false; // Return false
        }
    }

    Serial.println("");
    Serial.println("WiFi connected");
```

```
Serial.println("IP address: ");
Serial.println(WiFi.localIP()); // Print the local IP address

return true; // Return true if connection was successful
}

void printWeather() {
    Serial.print("Time: ");
    Serial.print(weather.time); // Time in UTC
    Serial.print(" Temperature: ");
    Serial.print(weather.temperature, 2); // Temperature in degrees
    Celsius
        Serial.print(" Humidity: ");
        Serial.print(weather.humidity, 2); // Humidity in percent
        Serial.print(" Dewpoint: ");
        Serial.print(weather.dewpoint, 2); // Dew point in degrees Celsius
        Serial.print(" Pressure: ");
        Serial.print(weather.pressure, 2); // Pressure in inches of mercury
        Serial.print(" Wind Speed: ");
        Serial.print(weather.windSpeed, 2); // Wind speed in knots
        Serial.print(" Wind Direction: ");
        Serial.print(weather.windDirection, 2); // Average wind direction in
    degrees
        Serial.print(" Rain Fall: ");
        Serial.print(weather.rainFall, 2); // Rain fall in inches/hour
        Serial.print(" UVA: ");
        Serial.print(weather.uva); // UVA value
        Serial.print(" UVB: ");
        Serial.print(weather.uvb); // UVB value
        Serial.print(" UV Index: ");
        Serial.println(weather.uvindex); // UV index
}

int getWindDirection() {
    unsigned int reading;
    reading = analogRead(windDirectionSensor); // Read the analog value
from the wind direction sensor

    if ( 10 < reading && reading <= 150) return (288);
    if ( 150 < reading && reading <= 250) return (264);
    if ( 250 < reading && reading <= 400) return (336);
    if ( 400 < reading && reading <= 600) return (312);
    if ( 650 < reading && reading <= 850) return ( 24);
    if ( 850 < reading && reading <= 1050) return ( 0);
    if (1300 < reading && reading <= 1500) return (216);
```

```

        if (1500 < reading && reading <= 1700) return (240);
        if (2000 < reading && reading <= 2200) return ( 72);
        if (2200 < reading && reading <= 2400) return ( 48);
        if (2400 < reading && reading <= 2600) return (168);
        if (2800 < reading && reading <= 3000) return (192);
        if (3000 < reading && reading <= 3200) return (120);
        if (3300 < reading && reading <= 3500) return (144);
        if (3700 < reading && reading <= 3900) return ( 96);
        return (-1);
    }

    void getWindandRainMeasurements(float& averageWindDirection, float&
averageWindSpeed, float& rainFall) {
        rainCount = 0; // Reset the rain count
        for (int i = 0; i < (duration * 6); i++) { // Reset the wind direction
array
            wind[i].reading = false;
            wind[i].direction = 0;
            wind[i].speed = 0;
        }
        for (int i = 0; i < (duration * 6); i++) { // Loop for the duration of
the report, taking measurements every 10 seconds
            Serial.print("Taking wind and rain measurements, this process will
be finished in: ");
            // Print a message to the serial monitor to let the
user know the program is working
            remainingMinutes = duration - ((i / 6) + !(i % 6));
            remainingSeconds = 60 - ((i % 6) * 10);
            Serial.print(remainingMinutes);
            Serial.print(":");
            if (remainingSeconds == 60) {
                Serial.println("00");
            } else {
                Serial.println(remainingSeconds);
            }

            windSpeedCount = 0; // Reset the wind speed count
            delay(10 * 1000); // Wait for a minute
            wind[i].speed = (windSpeedCount * 1.2959) / (10); // Calculate the
wind speed
            wind[i].direction = getWindDirection(); // Calculate the wind
direction
            wind[i].reading = true; // Set the reading flag to true
        }
        rainFall = (rainCount * 0.011 * 60) / (duration); // Calculate the
rain fall in inches/hour
    }
}

```

```
for (int i = 0; i < (duration * 6); i++) {
    theta = radians(wind[i].direction); // Convert angle to radians
    x = wind[i].speed * cos(theta);
    y = wind[i].speed * sin(theta);
    xSum += x;
    ySum += y;
}
averageWindDirection = degrees(atan2(ySum / (duration * 6), xSum /
(duration * 6))); // Convert radians to degrees
if (averageWindDirection < 0) averageWindDirection += 360; // Convert
negative angles to positive
averageWindSpeed = sqrt(pow(xSum / (duration * 6), 2) + pow(ySum /
(duration * 6), 2)); // Calculate average speed

xSum = 0;
ySum = 0;
}

void rainIRQ() // Interrupt called when the magnet in the rain gauge is
activated
{
    rainCount++;
}

void windSpeedIRQ() // Interrupt called when the magnet in the anemometer
is activated
{
    windSpeedCount++;
}

void sendWeatherDataToPocketBase() { // Send the weather data to
PocketBase
    Serial.println("Sending weather data to PocketBase...");

    HTTPClient http; // Create an HTTPClient object

    http.begin("https://w.arias.pw/api/collections/station/records"); // Set the
    // Set the PocketBase endpoint URL

    http.addHeader("Content-Type", "application/json"); // Set the HTTP
    // Set the headers

    // Create the JSON payload
    String payload = "{\"time\": \"" + weather.time +
                    "\", \"temperature\": " + String(weather.temperature) +
                    ", \"humidity\": " + String(weather.humidity) +
```

```
        ", \"dewpoint\":" + String(weather.dewpoint) +
        ", \"pressure\":" + String(weather.pressure) +
        ", \"rainFall\":" + String(weather.rainFall) +
        ", \"windSpeed\":" + String(weather.windSpeed) +
        ", \"windDirection\":" + String(weather.windDirection)
+
        ", \"uva\":" + String(weather.uva) +
        ", \"uvb\":" + String(weather.uvb) +
        ", \"uvindex\":" + String(weather.uvindex) + "}";
}

int httpCode = http.POST(payload); // Send the POST request

if(httpCode == 200) { // Check the returning code
    Serial.println("Data sent to PocketBase successfully");
} else { // If the code is not 200, something went wrong
    Serial.print("Error sending data to PocketBase, returned code: ");
    Serial.println(httpCode);

    Serial.println("Restarting ESP32...");
    delay(1000); // Wait for the serial output to finish
    esp_restart(); // Restart the ESP32
}

http.end(); // Close connection
}

void writeDataToSDCard() { // Write the weather data to the SD card
    Serial.println("Writing weather data to SD card...");

    for (int i = 0; i < 3; i++) { // Try opening the file up to 3 times
        dataFile = SD.open("/data.csv", FILE_APPEND); // Open the data
file

        if (dataFile) { // If the file opened successfully, write the data
            dataFile.print(weather.time);
            dataFile.print(",");
            dataFile.print(weather.temperature);
            dataFile.print(",");
            dataFile.print(weather.humidity);
            dataFile.print(",");
            dataFile.print(weather.dewpoint);
            dataFile.print(",");
            dataFile.print(weather.pressure);
            dataFile.print(",");
            dataFile.print(weather.rainFall);
            dataFile.print(",");
        }
    }
}
```

```
        dataFile.print(weather.windSpeed);
        dataFile.print(",");
        dataFile.print(weather.windDirection);
        dataFile.print(",");
        dataFile.print(weather.uva);
        dataFile.print(",");
        dataFile.print(weather.uvb);
        dataFile.print(",");
        dataFile.println(weather.uvindex);
        dataFile.close(); // Close the file
        Serial.println("Data written to SD card successfully");
        return; // Exit the function after successful write
    } else { // If the file did not open successfully, print an error
        Serial.println("Error opening file on SD card. Retrying...");
        delay(500); // Wait for half a second before retrying
    }
}

// If all attempts to open the file have failed, print an error
message and restart the ESP32
Serial.println("Failed to write data to SD card after multiple
attempts.");
Serial.println("Restarting ESP32...");

delay(1000); // Wait for the serial output to finish
esp_restart(); // Restart the ESP32
}
```

18.3.3. Servidor

18.3.3.1. AccuWeather

Código para obtener los datos de AccuWeather interpretar los datos y enviarlos a PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos deben ser reemplazadas por las correspondientes.

accuweather.py

```
import requests
import json
from datetime import datetime
```

```
# Define API endpoint and parameters
location_key = "3570769" # Location key
api_key = "ACCUWEATHERAPIKEY" # AccuWeather API key
pocketbase_api_url = "https://your.domain/api/collections/accuweather/
records" # Pocketbase API endpoint

# Define headers
headers = {
    "Content-Type": "application/json",
}

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Print execution time
print(execution_time() + " - Executing script...")

# Make API request
print(execution_time() + " - Making request to AccuWeather...")
response = requests.get("http://dataservice.accuweather.com/currentconditi
ons/v1/" + location_key + "/historical/24?apikey=" + api_key +
"&language=en-us&details=true&metric=true")
data = json.loads(response.text)
data = json.dumps(data, indent=4)

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Request to AccuWeather successful!")
    # Parse JSON response
    for i in range(0, len(json.loads(data))):
        new_data = {
            "time": datetime.utcfromtimestamp(json.loads(data)[i]
['EpochTime']).strftime('%Y-%m-%d %H:%M:%S.000Z'),
            "temperature": json.loads(data)[i]['Temperature']['Metric']
['Value'],
            "realFeelTemperature": json.loads(data)[i]
['RealFeelTemperature']['Metric']['Value'],
            "realFeelTemperatureShade": json.loads(data)[i]
['RealFeelTemperatureShade']['Metric']['Value'],
            "relativeHumidity": json.loads(data)[i]['RelativeHumidity'],
            "indoorRelativeHumidity": json.loads(data)[i]
['IndoorRelativeHumidity'],
            "dewPoint": json.loads(data)[i]['DewPoint']['Metric']
['Value'],
```

```
        "windDirection": json.loads(data)[i]['Wind']['Direction']
['Degrees'],
        "windSpeed": json.loads(data)[i]['Wind']['Speed']['Metric']
['Value'],
        "uvIndex": json.loads(data)[i]['UVIndex'],
        "visibility": json.loads(data)[i]['Visibility']['Imperial']
['Value'],
        "pressure": json.loads(data)[i]['Pressure']['Imperial']
['Value'],
        "apparentTemperature": json.loads(data)[i]
['ApparentTemperature']['Metric']['Value'],
        "precipitation": json.loads(data)[i]['Precip1hr']['Metric']
['Value'],
    }
# Send data to PocketBase API
print(execution_time() + " - Sending data to PocketBase with date
and time: " + new_data['time'])

response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Data sent to PocketBase
successfully!")
else:
    print(execution_time() + " - Request to PocketBase failed!
with status code: " + str(response.status_code))

else:
    print(execution_time() + "Request to AccuWeather failed! with response
code: " + str(response.status_code))
```

18.3.3.2. AWC

Código para obtener los datos de AWC interpretar los datos y enviarlos a PocketBase.

```
awc.py .py

# Description: Fetches METAR data from Aviation Weather Center API and
posts it to PocketBase API

# Import libraries
import requests
```

```
import csv
import json
from datetime import datetime

# Define constants
API_URL = "https://www.aviationweather.gov/adds/dataserver_current/
httpparam" # Aviation Weather Center API URL
POCKETBASE_API_URL = "https://your.domain/api/collections/awc/records" #
PocketBase API URL

# Define variables
data_type = "metars"
airport_code = "MMLO" # Airport code
hours_before_now = "24" # Number of hours before now to fetch data for
output_format = "csv"

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Define function to fetch CSV data
def fetch_csv_data(api_url, parameters):
    print(execution_time() + " - Making request to Aviation Weather
Center...")
    # Make the API request
    response = requests.get(api_url, params=parameters)
    # Check for successful request
    if response.status_code == 200:
        # Decode the CSV content
        content = response.content.decode("utf-8")
        # Parse the CSV data and return it
        csv_data = list(csv.reader(content.splitlines(), delimiter=","))
        return csv_data
    else:
        print(execution_time() + " - Request to Aviation Weather Center
failed! with status code: " + str(response.status_code))

# Set up the API request parameters
parameters = {
    "dataSource": data_type,
    "requestType": "retrieve",
    "format": output_format,
    "stationString": airport_code,
    "hoursBeforeNow": hours_before_now,
}
```

```
# Print execution time
print(execution_time() + " - Executing script...")

# Call the function to fetch the CSV data
csv_data = fetch_csv_data(API_URL, parameters)

# Check if there are results in the CSV data
if len(csv_data) <= 6:
    print(execution_time() + " - No results found in CSV data received
from Aviation Weather Center")
else:
    print(execution_time() + " - CSV data received from Aviation Weather
Center successfully!")
# Parse the remaining rows of the CSV data and convert to JSON
print(execution_time() + " - Parsing CSV data and converting it to
JSON...")
for row in csv_data[6:]: # Skip the first 6 rows
    # Get the headers from the first row of the CSV data
    headers = csv_data[5]
    # Create a dictionary with the desired keys and values
    data = {
        "raw_text": row[headers.index("raw_text")],
        "station_id": airport_code,
        "observation_time":
            datetime.strptime(row[headers.index("observation_time")], "%Y-%m-%dT%H:%M:
%SZ").strftime("%Y-%m-%d %H:%M:%S.000Z"),
        "temp_c": float(row[headers.index("temp_c")]),
        "dewpoint_c": float(row[headers.index("dewpoint_c")]),
        "wind_dir_degrees":
            int(row[headers.index("wind_dir_degrees")]) if
            row[headers.index("wind_dir_degrees")] else 0,
        "wind_speed_kt": int(row[headers.index("wind_speed_kt")]) if
            row[headers.index("wind_speed_kt")] else 0,
        "altim_in_hg": float(row[headers.index("altim_in_hg")]),
        "corrected": bool(row[headers.index("corrected")]),
        "precip_in": float(row[headers.index("precip_in")]) if
            row[headers.index("precip_in")] else 0,
        "metar_type": row[headers.index("metar_type")],
    }

    # Set up the headers for the POST request
    headers = {'Content-Type': 'application/json'}

    # Make the POST request to PocketBase API
    print(execution_time() + " - Sending data to PocketBase with date
and time: " + data["observation_time"])
```

```
response = requests.post(POCKETBASE_API_URL,
data=json.dumps(data), headers=headers)

# Check for successful request
if response.status_code == 200:
    print(execution_time() + " - Data sent to PocketBase
successfully!")
else:
    print(execution_time() + " - Request to PocketBase failed!
with status code: " + str(response.status_code))
```

18.3.3.3. Open-Meteo

Código para obtener los datos de Open-Meteo interpretar los datos y enviarlos a PocketBase.

open-meteo.py

.py

```
# Description: This script is used to get the weather data from the
OpenMeteo API and send it to Pocketbase

# Import libraries
import requests
import json
from datetime import datetime
from datetime import date
from datetime import timedelta

# Define API endpoint and parameters
url = "https://api.open-meteo.com/v1/forecast" # OpenMeteo API endpoint
pocketbase_api_url = "https://your.domain/api/collections/open_meteo/
records" # Pocketbase API endpoint
headers = {
    "Content-Type": "application/json",
}

# Get previous day
previous_day = (datetime.utcnow() - timedelta(days=1)).strftime("%Y-%m-
%d")

params = {
    "latitude": 21.01, # Latitude of the location
    "longitude": -101.49, # Longitude of the location
```

```
"hourly":  
    "temperature_2m", relativehumidity_2m, dewpoint_2m, apparent_temperature, rain, pressure_msl, s  
        "windspeed_unit": "kn", # Unit of the wind speed  
        "precipitation_unit": "inch", # Unit of the precipitation  
        "forecast_days": 1,  
        "start_date": previous_day, # Start date of the forecast  
        "end_date": previous_day # End date of the forecast, for one day, use  
the same date as start_date  
    }  
  
# Define function to get current time  
def execution_time():  
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")  
  
# Print execution time  
print(execution_time() + " - Executing script...")  
  
# Make API request  
print(execution_time() + " - Making request to open-meteo...")  
response = requests.get(url, params=params)  
  
# Check if response was successful  
if response.status_code == 200:  
    # Print response  
    print(execution_time() + " - Response from open-meteo was  
successful!")  
  
    # Parse JSON response  
    data = json.loads(response.text)  
  
    # Structure data  
    data = json.dumps(data, indent=4)  
  
    # Send data to Pocketbase API one hour at a time  
    for hour in range(0, len(json.loads(data)['hourly']['time'])):  
        new_data = {  
            "time": datetime.strptime(json.loads(data)['hourly']['time'][hour], "%Y-%m-%dT%H:%M").strftime("%Y-%m-%d %H:%M:00.000Z"),  
            "temperature_2m": json.loads(data)['hourly']['temperature_2m'][hour],  
            "relativehumidity_2m": json.loads(data)['hourly']['relativehumidity_2m'][hour],  
            "dewpoint_2m": json.loads(data)['hourly']['dewpoint_2m'][hour],  
            "apparent_temperature": json.loads(data)['hourly']['apparent_temperature'][hour],  
        }
```

```
        "rain": json.loads(data)['hourly']['rain'][hour],
        "pressure_msl": json.loads(data)['hourly']['pressure_msl']
    [hour],
        "surface_pressure": json.loads(data)['hourly']
    ['surface_pressure'][hour],
        "windspeed_10m": json.loads(data)['hourly']['windspeed_10m']
    [hour],
        "windspeed_80m": json.loads(data)['hourly']['windspeed_80m']
    [hour],
        "windspeed_120m": json.loads(data)['hourly']['windspeed_120m']
    [hour],
        "windspeed_180m": json.loads(data)['hourly']['windspeed_180m']
    [hour],
        "winddirection_10m": json.loads(data)['hourly']
    ['winddirection_10m'][hour],
        "winddirection_80m": json.loads(data)['hourly']
    ['winddirection_80m'][hour],
        "winddirection_120m": json.loads(data)['hourly']
    ['winddirection_120m'][hour],
        "winddirection_180m": json.loads(data)['hourly']
    ['winddirection_180m'][hour],
        "temperature_80m": json.loads(data)['hourly']
    ['temperature_80m'][hour],
        "temperature_120m": json.loads(data)['hourly']
    ['temperature_120m'][hour],
        "temperature_180m": json.loads(data)['hourly']
    ['temperature_180m'][hour],
        "uv_index": json.loads(data)['hourly']['uv_index'][hour]
    }
    # Send data to Pocketbase API
    print(execution_time() + " - Sending data to PocketBase with date
and time: " + new_data["time"])

    response = requests.post(pocketbase_api_url, headers=headers,
data=json.dumps(new_data))

    # Check if response was successful
    if response.status_code == 200:
        print(execution_time() + " - Request to Pocketbase was
successful!")
    else:
        print(execution_time() + " - Request to Pocketbase failed!
with status code: " + str(response.status_code))
    else:
        print(execution_time() + " - Request to open-meteo failed! with status
code: " + str(response.status_code))
```

18.3.3.4. OpenWeatherMap

Código para obtener los datos de OpenWeatherMap interpretar los datos y enviarlos a PocketBase.

El siguiente código no funciona directamente, las variables de conexión a la base de datos deben ser reemplazadas por las correspondientes.

openweathermap.py

.py

```
# Description: This script will make a request to the OpenWeatherMap API
# and save the data to PocketBase
import requests
import json
from datetime import datetime

# Define API endpoint and parameters
api_key = "OPENWEATHERMAPAPIKEY" # OpenWeatherMap API key
latitude = 21.01 # Latitude of the location
longitude = -101.49 # Longitude of the location
url = "https://api.openweathermap.org/data/2.5/weather?lat=" +
      str(latitude) + "&lon=" + str(longitude) + "&appid=" + api_key
pocketbase_api_url = "https://your.domain/api/collections/openweathermap/
records" # Pocketbase API endpoint
headers = {
    "Content-Type": "application/json",
}

# Define function to get current time
def execution_time():
    return datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")

# Print execution time
print(execution_time() + " - Executing script...")

# Make API request
print(execution_time() + " - Making request to OpenWeatherMap...")

response = requests.get(url)

# Check if response was successful
if response.status_code == 200:
    print(execution_time() + " - Request to OpenWeatherMap successful!")
    # Parse JSON response
    data = json.loads(response.text)
```

```
data = json.dumps(data, indent=4)

new_data = {
    "time": datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.000Z'),
    "temperature": json.loads(data)['main']['temp'],
    "feels_like": json.loads(data)['main']['feels_like'],
    "pressure": json.loads(data)['main']['pressure'],
    "humidity": json.loads(data)['main']['humidity'],
    "wind_speed": json.loads(data)['wind']['speed'],
    "wind_direction": json.loads(data)['wind']['deg'],
}

# Make API request to PocketBase
print(execution_time() + " - Sending data to PocketBase with date and
      time: " + new_data['time'])
response = requests.post(pocketbase_api_url, headers=headers,
                           data=json.dumps(new_data))

if response.status_code == 200:
    print(execution_time() + " - Data sent to PocketBase
          successfully!")
else:
    print(execution_time() + " - Request to PocketBase API failed!
          with status code: " + str(response.status_code))
else:
    print(execution_time() + " - Request to OpenWeatherMap failed! with
          status code: " + str(response.status_code))
```

18.3.4. Recolección

18.3.4.1. RAW Data

Script para la recolección de datos sin procesar.

```
raw.py .py

# This script fetches data from PocketBase and writes it to a CSV file

import requests
import csv
```

```
POCKETBASE_API_URL = "https://domain.name/api/collections/" # Specify the
# PocketBase API URL here
COLLECTION_NAME = "station" # Specify the collection name here
OUTPUT_CSV_FILE = f"{COLLECTION_NAME}.csv" # Generate the output file
name

# Define function to fetch data from PocketBase using pagination
def fetch_pocketbase_data(api_url):
    page = 1
    per_page = 50
    records = []

    while True:
        params = {
            "page": page,
            "perPage": per_page,
            "sort": "-created"
        }

        response = requests.get(api_url, params=params)

        if response.status_code == 200:
            json_response = response.json()
            records += json_response.get("items", [])

            # Check if there are more pages to fetch
            if json_response["page"] == json_response["totalPages"]:
                break

            # Move to the next page
            page += 1
        else:
            print("Request to PocketBase failed with status code:",
            response.status_code)
            break

    return records

# Construct the PocketBase API URL for the specified collection
pocketbase_api_url = f"{POCKETBASE_API_URL}{COLLECTION_NAME}/records"

# Call the function to fetch data from PocketBase
pocketbase_data = fetch_pocketbase_data(pocketbase_api_url)

# Check if there are results in the PocketBase data
if not pocketbase_data:
```

```
    print("No data found in PocketBase")
else:
    print("PocketBase data retrieved successfully!")
    # Extract the keys from the first record to use as CSV headers
    headers = list(pocketbase_data[0].keys())

    # Open the CSV file for writing
    with open(OUTPUT_CSV_FILE, mode="w", newline="") as file:
        writer = csv.DictWriter(file, fieldnames=headers)
        writer.writeheader()

        # Write each record to the CSV file
        for record in pocketbase_data:
            writer.writerow(record)

    print("Data written to CSV file:", OUTPUT_CSV_FILE)
```

18.3.5. Procesamiento y análisis de datos

18.3.5.1. Reescribir fechas

Script para reescribir las fechas de los datos, de acuerdo al timestamp de creación y no al proporcionado por la estación.

```
re-date.py .py

# Re-dates time values from given created values.

import os
import csv
from datetime import datetime

def change_date(input_file, output_directory):
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    output_file = os.path.join(output_directory,
    os.path.basename(input_file))

    correction_count = 0
    rows = []

    with open(input_file, "r") as file:
```

```
reader = csv.DictReader(file)
headers = reader.fieldnames

for row in reader:
    time_value = row["time"]
    if time_value.startswith("1970"):
        created_value = row.get("created", "")
        row["time"] = created_value[:24] # Update time value with
    created time
    correction_count += 1

    rows.append(row)

with open(output_file, "w", newline="") as outfile:
    writer = csv.DictWriter(outfile, fieldnames=headers)
    writer.writeheader()
    writer.writerows(rows)

print("Data processing complete for", input_file)
print("Number of corrections:", correction_count)

input_file = "manual/station-SD.csv"
output_directory = "re-dated"

change_date(input_file, output_directory)
```

18.3.5.2. Redondeo de fechas

Script para redondear las fechas de los datos, de acuerdo al valor de 10 minutos más cercano.

round.py

.py

```
# Rounds the time values in the CSV files to the nearest 10 minutes.

import os
import csv
from datetime import datetime, timedelta

def round_to_nearest_ten_minutes(time_str):
    time = datetime.strptime(time_str, "%Y-%m-%d %H:%M:%S.%fZ")
    rounded_time = time - timedelta(minutes=time.minute % 10,
                                    seconds=time.second,
```

```
microseconds=time.microsecond)
return rounded_time.strftime("%Y-%m-%d %H:%M")

input_directory = "re-dated"
output_directory = "rounded"

if not os.path.exists(output_directory):
    os.makedirs(output_directory)

for filename in os.listdir(input_directory):
    if filename.endswith(".csv"):
        input_file = os.path.join(input_directory, filename)
        output_file = os.path.join(output_directory, filename)

        with open(input_file, "r") as file:
            reader = csv.DictReader(file)
            headers = reader.fieldnames

            with open(output_file, "w", newline="") as outfile:
                writer = csv.DictWriter(outfile, fieldnames=headers)
                writer.writeheader()

                for row in reader:
                    row["time"] =
round_to_nearest_ten_minutes(row["time"])
                    writer.writerow(row)

        print("Data processing complete for", filename)

print("All files processed successfully.")
```

18.3.5.3. Interpolación y agrupación

Script para interpolar y agrupar los datos de acuerdo a la variable especificada.

interpolate.py

```
.py

# Combine multiple csv files into a single csv file, interpolating missing
data.

import os
import pandas as pd

def combine_csv_files(file_list, variable, initial_time, final_time):
```

```
combined_data = pd.DataFrame(columns=['time', variable])

for file_path in file_list:
    # Extract filename without extension
    filename = os.path.splitext(os.path.basename(file_path))[0]

    # Read the csv file, parsing 'time' column as datetime
    df = pd.read_csv(file_path, parse_dates=['time'])

    # Check if the variable column exists in the DataFrame
    if variable in df.columns:
        # Filter data based on initial and final time
        df = df[(df['time'] >= initial_time) & (df['time'] <=
final_time)]

        # Create a new column with the data from the variable
        df[filename] = df[variable]

        # Append the relevant columns to the combined data
        combined_data = pd.merge(combined_data, df[['time',
filename]], on='time', how='outer')

    # Set 'time' column as the index
    combined_data = combined_data.set_index('time')

    # Group by index (time) and aggregate the values
    combined_data = combined_data.groupby(combined_data.index).mean()

    # Resample and interpolate missing data
    combined_data =
combined_data.resample('10T').interpolate(method='time')

    # Save the combined data to a new csv file
    combined_data.to_csv(variable + '.csv')

# Example usage
file_list = ['accuweather.csv', 'awc.csv', 'bme680.csv', 'open_meteo.csv',
'openweathermap.csv', 'station-SD.csv', 'station.csv']
variable = 'windSpeed'
initial_time = '2023-04-30 23:00'
final_time = '2023-05-15 23:50'

combine_csv_files(file_list, variable, initial_time, final_time)
```

19. Listas

19.1. Lista de Diagramas

Diagrama 1: Flujo del prototipo del ESP32.	60
Diagrama 2: Flujo del prototipo de estación.	102
Diagrama 3: Flujo del prototipo del servidor.	123
Diagrama 4: Refinamiento de datos.	145

19.2. Lista de Esquemas

Esquemático 1: Pinout ESP32 (<i>Last Minute Engineers</i>).	57
Esquemático 2: Pinout BME680 (<i>Last Minute Engineers</i>).	58
Esquemático 3: Unión de componentes (<i>Microcontrollers Lab</i>).	59
Esquemático 4: Pinout ESP32 (<i>SparkFun</i>).	84
Esquemático 5: Weather Carrier Board (<i>SparkFun</i>).	85
Esquemático 6: Weather Meter Kit Anemómetro (<i>SparkFun</i>).	85
Esquemático 7: Weather Meter Kit Pluviómetro (<i>SparkFun</i>).	86

19.3. Lista de Figuras

Figura 1: Bosquejo de una EMA (<i>SMN</i>).	38
Figura 2: Admin UI, http://127.0.0.1:8090/_/ (<i>PocketBase</i>).	46
Figura 3: Pantalla inicial (<i>PocketBase</i>).	46
Figura 4: Eliminado de tabla <code>users</code> (<i>PocketBase</i>).	47
Figura 5: Creado de tabla <code>bm680</code> (<i>PocketBase</i>).	48
Figura 6: Reglas de la API (<i>PocketBase</i>).	49
Figura 7: Habilitación de regla <code>Create</code> (<i>PocketBase</i>).	50
Figura 8: ESP32 Dev Board (<i>Espressif</i>).	55
Figura 9: BME680 (<i>UNIT ELECTRONICS</i>).	55
Figura 10: Pantalla OLED 128 x 64 Pixel Bi-Color (<i>MEGATRONICA</i>).	56
Figura 11: Pantalla OLED con datos del sensor BME680.	69
Figura 12: Prototipo ESP32-BME680.	70
Figura 13: Datos de la colección BM680 (<i>PocketBase</i>).	79
Figura 14: MicroMod ESP32 Processor (<i>SparkFun</i>).	80
Figura 15: MicroMod Weather Carrier Board (<i>SparkFun</i>).	81
Figura 16: Weather Meter Kit (<i>SparkFun</i>).	81
Figura 17: 925-1418 Sensor Weather Shield (<i>La Crosse Technology</i>).	82
Figura 18: MicroSD (<i>Kingston</i>).	82
Figura 19: Contenido del Weather Meter Kit (<i>SparkFun</i>).	87

Figura 20: Pluviómetro del Weather Meter Kit (<i>SparkFun</i>).	88
Figura 21: Nivel de burbuja del pluviómetro del Weather Meter Kit (<i>SparkFun</i>).	89
Figura 22: Anemómetro del Weather Meter Kit (<i>SparkFun</i>).	90
Figura 23: Veleta de viento del Weather Meter Kit (<i>SparkFun</i>).	91
Figura 24: Unión de tubos Weather Meter Kit (<i>SparkFun</i>).	93
Figura 25: Unión del brazo central Weather Meter Kit (<i>SparkFun</i>).	93
Figura 26: Unión del brazo central con tornillo Weather Meter Kit (<i>SparkFun</i>).	94
Figura 27: Unión del anemómetro Weather Meter Kit (<i>SparkFun</i>).	94
Figura 28: Unión del anemómetro Weather Meter Kit (<i>SparkFun</i>).	95
Figura 29: Unión del anemómetro con tornillo Weather Meter Kit (<i>SparkFun</i>).	95
Figura 30: Unión de la veleta de viento Weather Meter Kit (<i>SparkFun</i>).	96
Figura 31: Unión de la veleta de viento con tornillo Weather Meter Kit (<i>SparkFun</i>).	96
Figura 32: Unión del brazo lateral Weather Meter Kit (<i>SparkFun</i>).	97
Figura 33: Unión del pluviómetro Weather Meter Kit (<i>SparkFun</i>).	97
Figura 34: Unión del pluviómetro Weather Meter Kit (<i>SparkFun</i>).	98
Figura 35: Unión del pluviómetro con tornillo Weather Meter Kit (<i>SparkFun</i>).	98
Figura 36: Gestión de cables Weather Meter Kit (<i>SparkFun</i>).	99
Figura 37: Conexión de anemómetro a veleta Weather Meter Kit (<i>SparkFun</i>).	99
Figura 38: Sujeción de cables Weather Meter Kit (<i>SparkFun</i>).	100
Figura 39: Armazón terminado Weather Meter Kit (<i>SparkFun</i>).	100
Figura 40: Abrazadera Weather Meter Kit (<i>SparkFun</i>).	101
Figura 41: Tabla de valores del sensor de dirección del viento (<i>SparkFun</i>).	110
Figura 42: Archivo guardado en la microSD (<i>data.csv</i>).	118
Figura 43: Contenido del archivo <i>data.csv</i> en la microSD.	118
Figura 44: Contenido del archivo <i>data.csv</i> (LibreOffice Calc).	119
Figura 45: Datos en la colección <i>station</i> (<i>PocketBase</i>).	119
Figura 46: Instalación de la estación meteorológica.	120
Figura 47: Datos en la colección <i>accuweather</i> (<i>PocketBase</i>).	136
Figura 48: Datos en la colección <i>awc</i> (<i>PocketBase</i>).	137
Referencia 49: Resultados de correcta ejecución de <i>open-meteo.py</i> .	137
Figura 50: Datos en la colección <i>open_meteo</i> (<i>PocketBase</i>).	138
Figura 51: Datos en la colección <i>openweathermap</i> (<i>PocketBase</i>).	139
Figura 52: Datos de la recolección del script de <i>raw.py</i> .	144
Figura 53: Resultados del refinamiento de datos.	146
Figura 54: Análisis de la variable <i>Temperature</i> .	147
Figura 55: Estación BASIC Wireless Weather Station LoRaWAN (<i>BARANI</i>).	174
Figura 56: Estación MWS-C400 (<i>intellisense</i>).	175

19.4. Lista de Gráficas

Gráfica 1: Punto de rocío.	149
Gráfica 2: Desviación del punto de rocío.	150
Gráfica 3: Corrección comparada con el promedio del punto de rocío.	151
Gráfica 4: Humedad.	152
Gráfica 5: Desviación de la humedad.	153
Gráfica 6: Corrección comparada con el promedio de la humedad.	154
Gráfica 7: Precipitación.	155
Gráfica 8: Desviación de la precipitación.	156
Gráfica 9: Corrección comparada con el promedio de la precipitación.	157
Gráfica 10: Presión.	158
Gráfica 11: Desviación de la presión.	159
Gráfica 12: Corrección comparada con el promedio de la presión.	160
Gráfica 13: Temperatura.	161
Gráfica 14: Desviación de la temperatura.	162
Gráfica 15: Corrección comparada con el promedio de la temperatura.	163
Gráfica 16: Índice UV.	164
Gráfica 17: Desviación del índice UV.	165
Gráfica 18: Corrección comparada con el promedio del índice UV.	166
Gráfica 19: Coordenadas de viento en X.	167
Gráfica 20: Desviación de las coordenadas de viento en X.	168
Gráfica 21: Corrección comparada con el promedio de las coordenadas de viento en X.	169
Gráfica 22: Coordenadas de viento en Y.	170
Gráfica 23: Desviación de las coordenadas de viento en Y.	171
Gráfica 24: Corrección comparada con el promedio de las coordenadas de viento en Y.	172

19.5. Lista de Referencias

Referencia 1: Ejemplo de reporte METAR.	30
Referencia 2: Ejemplo de reporte TAF.	36
Referencia 3: Ejemplo de reporte NOTAM.	36
Referencia 4: Resultados de correcta ejecución en el ESP32.	78
Referencia 5: Resultados de correcta ejecución en la estación meteorológica.	117
Referencia 6: Resultados de correcta ejecución de <code>accuweather.py</code> .	135
Referencia 7: Resultados de correcta ejecución de <code>awc.py</code> .	136
Referencia 8: Resultados de correcta ejecución de <code>openweathermap.py</code> .	138
Referencia 9: Ejemplo de reporte METAR.	191
Referencia 10: Ejemplo de reporte TAF.	191
Referencia 11: Ejemplo de reporte NOTAM.	192

19.6. Lista de Tablas

Tabla 1: Características de un reporte METAR.	31
Tabla 2: Porcentajes de datos obtenidos.	140
Tabla 3: Costos de la Estación.	173

20. Bibliografía

AccuWeather. *AccuWeather APIs*. Recuperado 11 de abril de 2023, de

<https://developer.accuweather.com/apis>

AWC. *METAR Text to Symbol*. Recuperado 11 de abril de 2023, de

https://www.aviationweather.gov/docs/metar/wxSymbols_anno2.pdf

AWC. (2013a). *AWC - METAR Information*. Aviation Weather Center.

<https://www.aviationweather.gov/metar>

AWC. (2013b). *AWC - Text Data Server*. Aviation Weather Center.

<https://www.aviationweather.gov/dataserver>

Barani, J. (2018a, mayo). *Affordable Auto-METAR for small airports finally a reality*. Barani Design.

<https://www.baranidesign.com/news-innovations-blog/2018/5/26/affordable-auto-metar-for-small-airports-finally-a-reality-at-farnborough-airshow-2018>

Barani, J. (2018b, julio). *What is the difference between METAR and AUTO-METAR?*. Barani Design.

<https://www.baranidesign.com/news-innovations-blog/2018/7/2/what-is-the-difference-between-metar-and-auto-metar>

Boxall, J. (2021). *Arduino Workshop* (2.^a ed.). No Starch Press.

CAPMA. *Centro de Análisis y Pronósticos Meteorológicos Aeronáuticos (CAPMA)*. Recuperado 1 de mayo de 2024, de <http://capma.mx/capma/capma.html>

Cloudflare. *Cloudflare Tunnel*. Cloudflare. Recuperado 15 de abril de 2023, de

<https://www.cloudflare.com/products/tunnel/>

Crocker, D. (2010). *Dictionary of Aviation* (2.^a ed.). A&C Black.

Crontab Guru. *Cron Schedule Expression Generator*. Recuperado 11 de abril de 2023, de

<https://crontab.guru/>

Cuesta, J. García de la. (2003). *Aviation Terminology Terminología Aeronáutica* (1.^a ed.). Ediciones Díaz de Santos.

D2 Lang. *D2 Declarative Diagramming*. Recuperado 16 de mayo de 2023, de <https://d2lang.com/>

FAA. (2016). *Advisory Circular. Aviation Weather Services*.

Huang, B., & Runberg, D. (2017). *The Arduino Inventor's Guide* (1.^a ed.). No Starch Press.

Hyde, R. (2020a). *Write Great Code, Volume 2 Thinking Low-Level* (2.^a ed.). No Starch Press.

Hyde, R. (2020b). *Write Great Code, Volume 3 Engineering Software* (1.^a ed.). No Starch Press.

ICAMS. *A Guide to Standards and Best Practices*. Recuperado 11 de abril de 2023, de

https://www.icams-portal.gov/resources/icams/related_documents/2021_fmh13.pdf

JSON Crack. *JSON Crack*. Recuperado 11 de abril de 2023, de <https://jsoncrack.com/>

La Crosse Technology. *925-1418 Sensor Weather Shield*. La Crosse Technology. Recuperado 15 de mayo de 2023, de <https://www.lacrossetechnology.com/products/925-1418>

Last Minute Engineers. *Getting Data and Time from NTP Server with ESP32*. Last Minute Engineers. Recuperado 11 de abril de 2023, de

<https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/>

Martin, R. (2017). *Clean Architecture A Craftsman's Guide to Software Structure and Design* (1.^a ed.). Prentice Hall.

Mayer, C. (2020). *Python One-Liners* (1.^a ed.). No Starch Press.

Microcontrollers Lab. *BME680 with ESP32 using Arduino IDE (Gas, Pressure, Temperature, Humidity)*. Microcontrollers Lab. Recuperado 11 de abril de 2023, de

<https://microcontrollerlab.com/bme680-esp32-arduino-oled-display/>

México. *Datos Abiertos*. Gobierno de México. Recuperado 11 de abril de 2023, de

<https://datos.gob.mx/>

México. (2013). *NMX-AA-166/1-SCFI-2013*.

https://www.gob.mx/cms/uploads/attachment/file/166835/nmx-aa-166-1-scfi-2013_1_.pdf

México. (2015). *NMX-AA-166/2-SCFI-2015*.

<https://www.gob.mx/cms/uploads/attachment/file/166838/nmx-aa-166-2-scfi-2015.pdf>

NCEI. (2021, marzo). *Automated Surface/Weather Observing Systems (ASOS/AWOS)*. NCEI.

<https://www.ncei.noaa.gov/products/land-based-station/automated-surface-weather-observing-systems>

NWS. (1998). *Training Guide in Surface Weather Observations*.

OMM. (2010). *Guía del Sistema Mundial de Observación*.

OMM. (2011). *Manual de Claves*.

Open Meteo. *Open Meteo Documentation*. Recuperado 11 de abril de 2023, de

<https://open-meteo.com/en/docs/>

OpenWeather. *Weather API*. Recuperado 11 de abril de 2023, de <https://openweathermap.org/api>

Oracle. *Oracle Cloud (modo gratuito)*. Oracle. Recuperado 15 de abril de 2023, de

<https://www.oracle.com/mx/cloud/free/>

Perry, B., & Taylor, D. (2016). *Wicked Cool Shell Scripts* (2.^a ed.). No Starch Press.

PocketBase. *PocketBase - Documentation*. Recuperado 11 de abril de 2023, de

<https://pocketbase.io/docs/>

PyData. *pandas - Python Data Analysis Library*. Recuperado 14 de junio de 2023, de

<https://pandas.pydata.org/docs/>

Python Foundation. *Python Documentation (v3)*. Recuperado 11 de abril de 2023, de

<https://docs.python.org/3/>

Shell Foundation. *An Introduction to Shell Programming*. FAQs.org. Recuperado 11 de abril de 2023, de
<http://www.faqs.org/docs/air/tsshell.html>

Shotts, W. E. (2012). *The Linux Command Line* (2.^a ed.). No Starch Press.

SMN. *Estaciones Meteorológicas Automáticas (EMA's)*. Servicio Meteorológico Nacional. Recuperado 11 de abril de 2023, de

<https://smn.conagua.gob.mx/es/observando-el-tiempo/estaciones-meteorologicas-automaticas-ema-s>

SparkFun. (s. f.-d). *MicroMod ESP32 Processor*. SparkFun. Recuperado 11 de abril de 2023, de
<https://www.sparkfun.com/products/16781>

SparkFun. (s. f.-a). *MicroMod ESP32 Processor Board Hookup Guide*. SparkFun. Recuperado 11 de abril de 2023, de

<https://learn.sparkfun.com/tutorials/micromod-esp32-processor-board-hookup-guide>

SparkFun. (s. f.-b). *MicroMod ESP32 Weather Carrier Board Hookup Guide*. SparkFun. Recuperado 11 de abril de 2023, de

<https://learn.sparkfun.com/tutorials/micromod-weather-carrier-board-hookup-guide>

SparkFun. (s. f.-e). *Weather Carrier Board*. SparkFun. Recuperado 11 de abril de 2023, de
<https://www.sparkfun.com/products/16794>

SparkFun. (s. f.-c). *Weather Meter Hookup Guide*. SparkFun. Recuperado 11 de abril de 2023, de
<https://learn.sparkfun.com/tutorials/weather-meter-hookup-guide>

Sparkfun. *Weather Meter Kit*. SparkFun. Recuperado 11 de abril de 2023, de
<https://www.sparkfun.com/products/15901>

Ubuntu. *Ubuntu Server Documentation*. Recuperado 11 de abril de 2023, de
<https://ubuntu.com/server/docs>

Weather.gov. (2008, diciembre 24). *METAR and TAF Abbreviations*.
https://www.weather.gov/media/wrh/mesowest/metar_decode_key.pdf

Wikipedia,. (2023, enero). *Automated airport weather station*. Wikipedia.
https://en.wikipedia.org/w/index.php?title=Automated_airport_weather_station&oldid=1135363695

WMO. (2018). *Guía de Instrumentos y Métodos de Observación*.

WMO. (2021). *Guide to Instruments and Methods of Observation*. (8.^a ed., Vol. 1).
https://library.wmo.int/doc_num.php?explnum_id=11386