



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Herramienta Automática para la Gestión
de Información Meteorológica de la
Misión Global Precipitation
Measurement (GPM) de la NASA**

Autor: Amanda Parreño Sierra

Tutor(a): José Antonio Mateo Cortés

Madrid, julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Herramienta Automática para la Gestión de Información Meteorológica de la Misión Global Precipitation Measurement (GPM) de la NASA

Julio 2025

Autor: Amanda Parreño Sierra

Tutor:

José Antonio Mateo Cortés

Arquitectura y Tecnología de Sistemas Informáticos

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado (TFG) se centra en el desarrollo de una aplicación web para gestionar la información meteorológica proporcionada por la misión Global Precipitation Measurement (GPM) de la NASA. Se exploran diversas fuentes de datos y métodos de obtención, incluyendo la descarga directa y el uso de OPeNDAP para crear URLs dinámicas que faciliten la descarga de datos.

El enfoque de este proyecto reside en la creación de una herramienta funcional y accesible para cualquier persona interesada en la información meteorológica proporcionada por la misión GPM de la NASA. Su objetivo es ofrecer una interfaz amigable para visualizar y solicitar datos meteorológicos bajo demanda.

A lo largo del documento, se detallan las tecnologías seleccionadas para la implementación, el diseño de alto nivel de la aplicación, incluyendo prototipos, y el diseño de bajo nivel, que sienta las bases para la implementación futura. Además, se definen la estructura de la base de datos y los servicios REST necesarios para el funcionamiento de esta.

En resumen, este TFG representa un esfuerzo integral para crear una herramienta útil y accesible que contribuya al acceso y comprensión de la información meteorológica proporcionada por la misión GPM de la NASA, facilitando así la investigación y el trabajo científico en esta área.

Abstract

This Final Degree Project focuses on the development of a web application to manage meteorological information provided by the Global Precipitation Measurement (GPM) mission of NASA. Various data sources and acquisition methods are explored, including direct download and the use of OPeNDAP to create dynamic URLs to facilitate data retrieval.

The focus of this project lies in creating a functional and accessible tool for anyone interested in meteorological information provided by NASA's GPM mission. Its aim is to provide a user-friendly interface for visualizing and requesting meteorological data on demand.

Throughout the document, the selected technologies for implementation are detailed, along with the high-level design of the application, including prototypes, and the low-level design, which lays the groundwork for future implementation. Additionally, the database structure and REST services necessary for the application's operation are defined.

In summary, this project represents a comprehensive effort to create a useful and accessible tool that contributes to accessing and understanding meteorological information provided by NASA's GPM mission, thereby facilitating research and scientific work in this field.

Tabla de contenido

1	Introducción	1
1.1	Contexto	1
1.2	Objetivos	1
1.3	Estado del Arte	2
1.3.1	Earthdata Search	3
1.3.2	Geovanni	3
2	Desarrollo de la Aplicación	6
2.1	Metodología Adoptada	6
2.2	Diseño Alto Nivel	6
2.2.1	Requisitos del Sistema	7
2.2.2	Diagrama BBDD	16
2.2.3	Arquitectura del Sistema	18
2.2.4	Tecnologías Utilizadas	20
2.3	Diseño de Bajo Nivel	21
2.3.1	Backend	21
2.3.1.1	Acceso a datos con OPeNDAP	22
2.3.1.2	Tecnologías Utilizadas	24
2.3.1.3	Arquitectura Backend	25
2.3.1.4	Rest API	34
2.3.1.5	Base de Datos	39
2.3.1.6	Seguridad	42
2.3.2	Frontend	44
2.3.2.1	Tecnologías Utilizadas	44
2.3.2.2	Arquitectura Frontend	46
2.3.2.3	Interfaz de Usuario	52
3	Resultados y Conclusiones	62
3.1	Evaluación de la Aplicación	62
3.2	Futuras Mejoras	63
4	Análisis de Impacto	65
	Bibliografía	67

1 Introducción

1.1 Contexto

El presente Trabajo de Fin de Grado (TFG) se desarrolla en el marco de formación académica en el Grado de Ingeniería Informática de la Universidad Politécnica de Madrid. Se centra en la aplicación de habilidades en desarrollo de software y diseño web para la gestión y visualización de datos meteorológicos, específicamente en relación con la misión GPM (Global Precipitation Measurement) de la NASA [1]. Se hace hincapié en el nivel 3 GPROF versión 07 [2], que implica cambios significativos con respecto a la versión anterior, como modificaciones en el formato del producto y la inclusión de nuevas variables dimensionales, entre otros ajustes.

1.2 Objetivos

El proyecto busca abordar la recuperación, almacenamiento y visualización de datos meteorológicos utilizando el algoritmo L3GPROF. Este algoritmo proporciona promedios mensuales y diarios de la precipitación y parámetros relacionados, obtenidos a partir del algoritmo de perfilado de precipitación GPROF de nivel 2 para los satélites centrales y de constelación de GPM. Se pretende desarrollar una solución integral que permita a los usuarios acceder a datos meteorológicos de manera sencilla y comprensible, mediante la creación de una API personalizada para gestionar las solicitudes de datos y el diseño e implementación de una base de datos eficiente para almacenar la información recolectada. La aplicación web resultante servirá tanto como ejercicio práctico de desarrollo de software como herramienta útil para aquellos interesados en explorar y aprender sobre el clima y la meteorología.

El objetivo principal es aplicar los conocimientos teóricos adquiridos durante la formación académica en un proyecto práctico y significativo, mejorando las habilidades en el diseño y desarrollo de software, así como en la documentación y presentación de proyectos.

Para alcanzar este objetivo principal, se han establecido los siguientes objetivos específicos:

- Objetivo 1: Recuperar información meteorológica de la misión GPM de la NASA.
- Objetivo 2: Diseñar e implementar la base de datos necesaria para almacenar los datos meteorológicos.
- Objetivo 3: Creación de API personalizada.
- Objetivo 4: Desarrollo del *backend* de la aplicación.
- Objetivo 5: Desarrollar *frontend* para mostrar la información.

Todos estos objetivos se han ido alcanzando según una planificación inicial. Para llevar a cabo esta planificación, se creó una lista detallada de tareas para cada objetivo, y se utilizó también un Diagrama de Gantt dividido en semanas. El Diagrama de Gantt, ampliamente reconocido como un instrumento de gestión de proyectos, se empleó para visualizar de manera clara y concisa las tareas que debían realizarse desde el inicio del proyecto hasta su conclusión. Esto permitió una mejor coordinación de las actividades, una asignación eficiente de

recursos y una monitorización efectiva del progreso del proyecto a lo largo del tiempo.

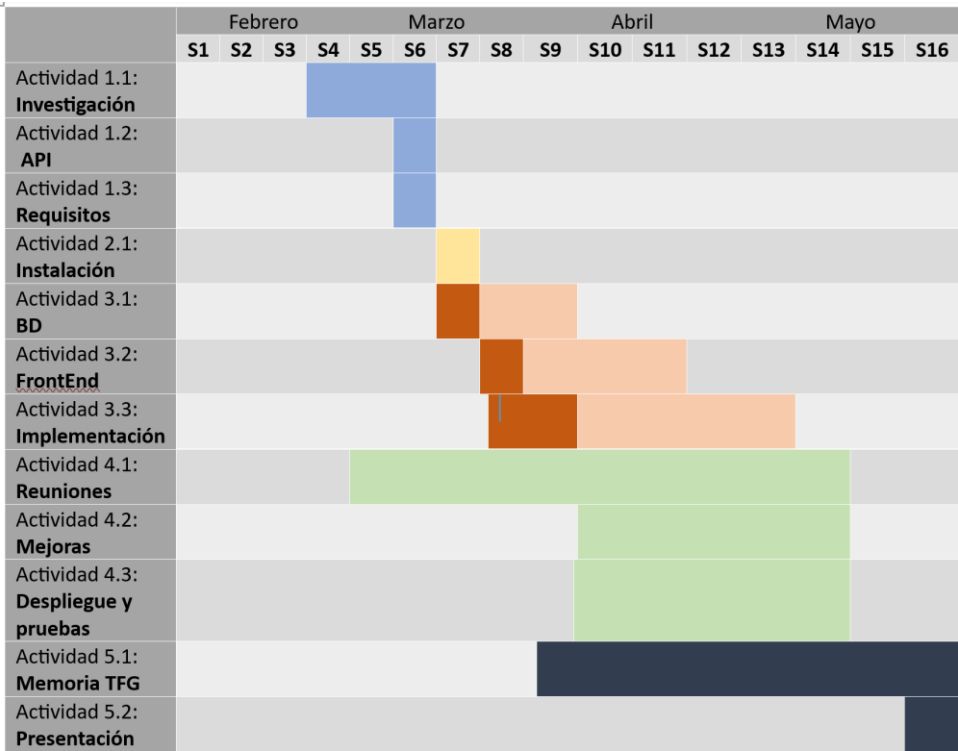


Ilustración 1: Diagrama de Gantt.

Como se puede observar en la Ilustración 1, la Actividad 1 se desarrolló según lo previsto en el plan inicial. Sin embargo, dentro de la Actividad 2 se contemplaba el desarrollo de un pequeño ejemplo, el cual se reconsideró y, finalmente, se decidió no llevar a cabo. Como consecuencia de esta decisión, se realizó una modificación en el diagrama, redistribuyendo las horas asignadas de esta actividad hacia la Actividad 3. El resto del Diagrama de Gantt se ajustó en consecuencia para reflejar con precisión la realidad del proyecto.

1.3 Estado del Arte

En esta sección, se explorará el panorama actual de algunas de las herramientas y plataformas utilizadas en el acceso, recopilación y análisis meteorológico. Aunque existen numerosas aplicaciones con estos propósitos, la comparativa se centrará en dos de ellas: Earthdata Search y Giovanni, ambas plataformas desarrolladas por la NASA. Al comprender el estado actual de las tecnologías disponibles, es posible identificar oportunidades para mejorar y diferenciar nuestra aplicación en desarrollo

1.3.1 Earthdata Search

Earthdata Search [3] es una plataforma desarrollada por la NASA que ofrece acceso a una amplia variedad de datos relacionados con la Tierra, incluidos los datos recopilados por la misión GPM (Global Precipitation Measurement). Esta plataforma permite a los usuarios buscar, descubrir y acceder a una gran cantidad de datos meteorológicos, incluyendo el nivel 3 GPROF versión 07, que proporciona información detallada sobre la precipitación a nivel global.

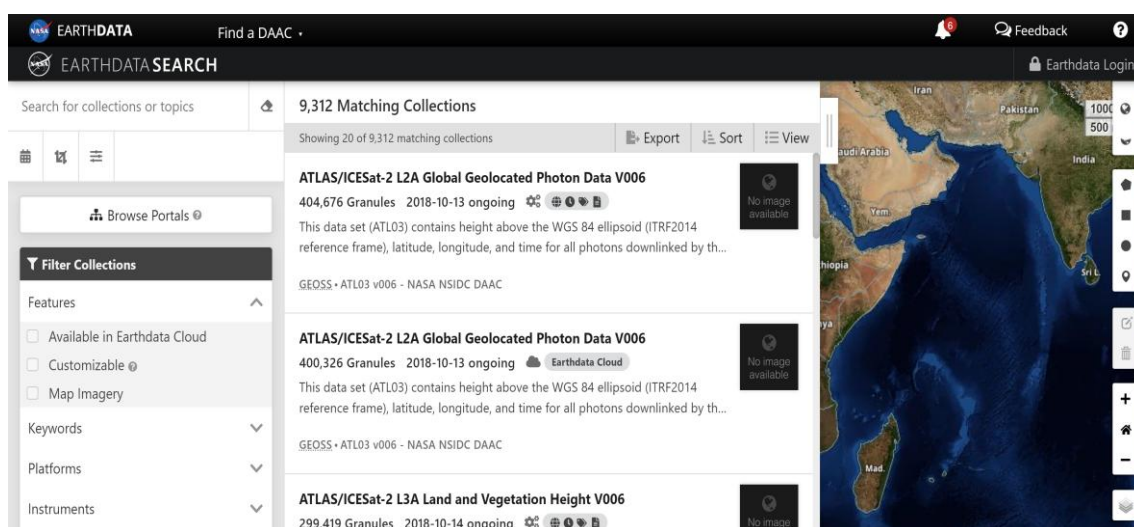


Ilustración 2: Aplicación web Earthdata Search [3].

Aunque es una herramienta ampliamente utilizada para acceder a datos meteorológicos, puede presentar una curva de aprendizaje pronunciada para usuarios menos experimentados. La complejidad de su interfaz puede dificultar la navegación y la recuperación de datos específicos, lo que limita su utilidad para aquellos que no están familiarizados con su funcionamiento. Aunque tiene una interfaz sencilla, como se muestra en la Ilustración 2, los usuarios pueden encontrarse con dificultades para encontrar y acceder a los datos de la misión GPM de manera eficiente, lo que puede ralentizar el proceso de investigación y análisis meteorológico. Además, el acceso a algunos conjuntos de datos en Earthdata Search puede requerir un registro previo en la plataforma, lo que añade una barrera adicional para los usuarios nuevos.

1.3.2 Giovanni

Giovanni [4] también desarrollada por la NASA ofrece acceso a una variedad de herramientas y servicios para la visualización y análisis de datos relacionados con la Tierra. A través de Giovanni, los usuarios pueden explorar y examinar una amplia gama de conjuntos de datos atmosféricos, oceanográficos y

terrestres recopilados por diversas misiones satelitales y sistemas de observación de la Tierra.

En la siguiente ilustración se puede observar parte de los filtros que el usuario puede seleccionar con esta herramienta.

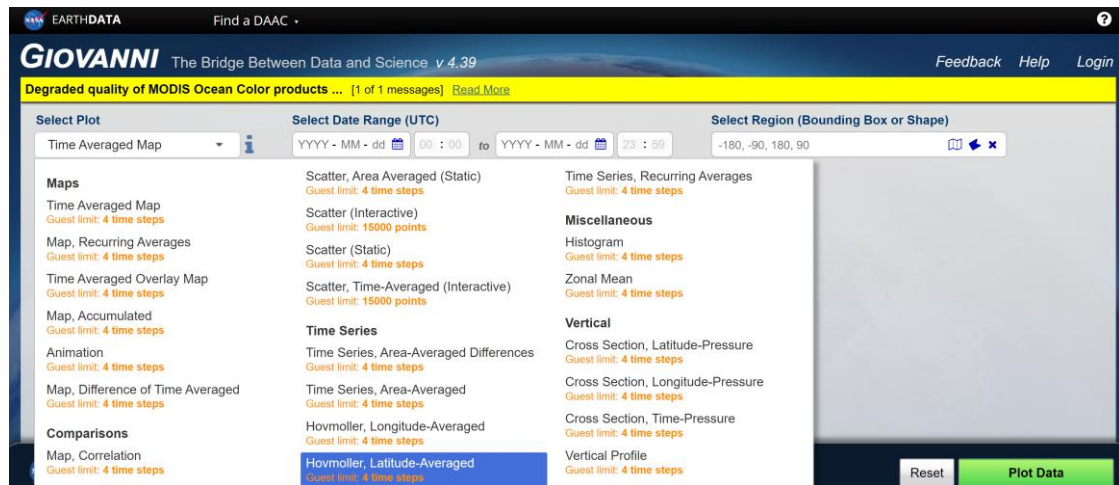


Ilustración 3: Aplicación web Giovanni. Página selección de filtros [4].

Una de las principales ventajas de Giovanni es su conjunto de herramientas de visualización interactiva, que permiten a los usuarios generar gráficos, mapas y diagramas personalizados para explorar y analizar datos meteorológicos. Además, Giovanni ofrece capacidades de análisis avanzadas, como la capacidad de realizar promedios temporales y espaciales, correlaciones y comparaciones entre diferentes conjuntos de datos.

Aunque se han resaltado las ventajas de la herramienta Geovanni, es importante tener en cuenta una limitación significativa: la incapacidad de visualizar datos específicos de cada radiómetro y satélite de la misión GPM de forma individual.

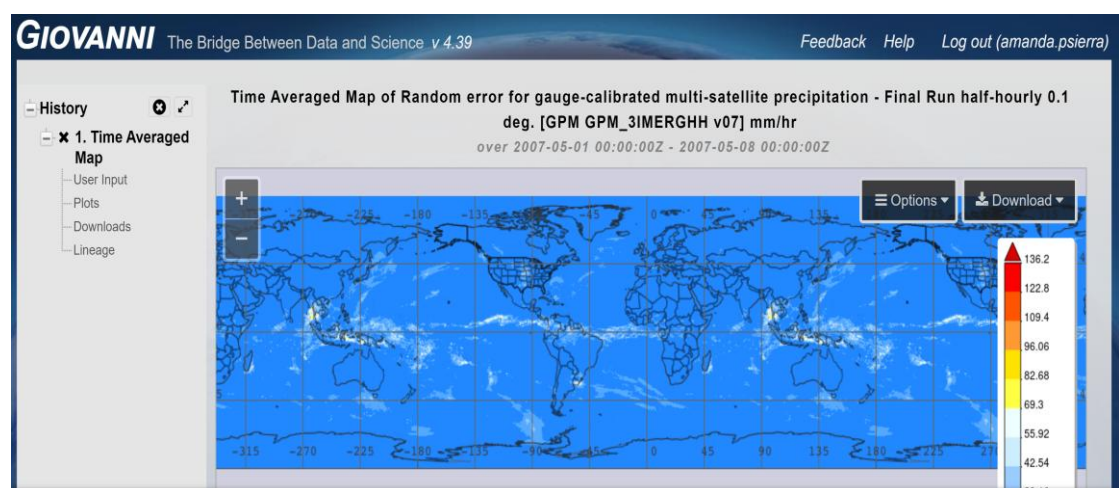


Ilustración 4: Aplicación web Giovanni. Resultado error aleatorio en la precipitación multi-satélite [4].

La Ilustración 4 muestra un mapa promedio en el tiempo del error aleatorio para la precipitación multi-satélite calibrada con sensores en tierra, generado

por la herramienta Giovanni. Este mapa representa la variabilidad del error en la precipitación estimada por los satélites de la misión GPM durante un período específico, en este caso, del 1 al 8 de mayo de 2007. Estos datos provienen del conjunto GPM_3IMERGHH v07, que utiliza el algoritmo IMERG desarrollado por la NASA [5]. Este algoritmo combina estimaciones de precipitación recopiladas durante la operación del satélite TRMM (2000-2015) con estimaciones recientes de precipitación recopiladas por la misión GPM (2014-presente).

También es importante mencionar que la curva de aprendizaje de Giovanni puede ser pronunciada para usuarios menos experimentados, como pasaba con la herramienta mencionada anteriormente Earthdata Search.

Para resolver estas limitaciones la aplicación se centra específicamente en proporcionar acceso a datos meteorológicos de la misión GPM, con un enfoque en la recuperación, almacenamiento y visualización eficientes de estos datos. Se ha diseñado la aplicación con una interfaz intuitiva y fácil de usar, que está especialmente adaptada para satisfacer las necesidades de los usuarios interesados en la investigación y el análisis de precipitaciones globales. Al centrarse en un conjunto de datos específico, se puede ofrecer una experiencia más enfocada y especializada para nuestros usuarios, lo que les permite acceder y analizar datos meteorológicos de manera rápida y eficiente.

2 Desarrollo de la Aplicación

En esta sección, se definirá el proceso de desarrollo de la aplicación para la gestión y análisis de datos meteorológicos. Desde la metodología adoptada hasta las tecnologías utilizadas, se explorará en detalle cada fase del desarrollo. Se realizará un análisis sobre cómo se tradujeron los requisitos del sistema en un diseño de alto nivel, y luego se pasará al diseño de bajo nivel, analizando *backend* y *frontend* por separado. Además, se examinará cómo se estructuró la base de datos para almacenar y gestionar eficientemente los datos meteorológicos.

El objetivo principal de esta sección es proporcionar una visión detallada del enfoque técnico y metodológico empleado para dar vida a la aplicación, destacando las decisiones clave tomadas a lo largo del proceso de desarrollo y su impacto en la funcionalidad y el rendimiento del sistema final.

2.1 Metodología Adoptada

Para el desarrollo de este proyecto, se optó por una metodología de desarrollo iterativo e incremental [6]. Esta elección se basó en la necesidad de adaptarse a los cambios y ajustes que surgían a lo largo del proceso, así como en la posibilidad de realizar entregas graduales y continuas de funcionalidades.

En este enfoque, cada ciclo de trabajo se enfocó en la implementación y prueba de una funcionalidad específica, la cual era desarrollada en el *backend*, probada y una vez que cumpliera con su funcionalidad se desarrollaba el *frontend* y se unía para su prueba final. Si bien no se dividió el proyecto en módulos independientes, se adoptó una estrategia de desarrollo incremental, donde las funcionalidades se construían y agregaban al producto de manera progresiva.

El desarrollo iterativo e incremental ofrece flexibilidad y la capacidad de adaptarse a los cambios, pero también presenta desafíos. Por ejemplo, se requiere una planificación cuidadosa para garantizar la coherencia entre las diferentes partes del sistema y la integración adecuada de las funcionalidades desarrolladas en cada iteración. Sin embargo, al no existir un equipo de trabajo, si no, un único desarrollador, la coordinación y cohesión entre los diferentes componentes del proyecto se simplificaron considerablemente.

Dado que no existía un cliente final en el sentido tradicional, las reuniones de retroalimentación con el tutor desempeñaron un papel crucial. Estas reuniones permitieron revisar el progreso, discutir posibles cambios y ajustes, y tomar decisiones de desarrollo en conjunto.

2.2 Diseño Alto Nivel

En este apartado se establecerán las bases fundamentales sobre las que se ha construido la aplicación web, *GPMDDataExplorer*. El diseño de alto nivel es una fase necesaria en cualquier proyecto de desarrollo para garantizar que el proyecto cumpla con los objetivos y necesidades del usuario final, así como proporcionar una guía clara sobre los pasos a seguir para el equipo de desarrollo.

En primer lugar, se definirán los requisitos del sistema, que especifican las funcionalidades y características que debe tener la aplicación para cumplir con los objetivos. Luego, se presentará el diagrama de la base de datos, que muestra la estructura y las relaciones entre las diferentes entidades de datos que se utilizarán. A continuación, se describirá la arquitectura del sistema, que define la estructura general y la distribución de los componentes del sistema, incluidos el *backend* y el *frontend*. Finalmente, en el apartado de tecnologías utilizadas se pasará a definir aquellas herramientas de soporte que han sido utilizadas en todo el proceso de desarrollo, dejando para más adelante las herramientas específicas tales como lenguajes de programación, frameworks y bibliotecas utilizadas en el *backend* y *frontend* de la aplicación.

2.2.1 Requisitos del Sistema

Objetivos

Para poder comenzar con el desarrollo de la aplicación, es crucial identificar las necesidades que deben ser abordadas por el proyecto. A continuación, se detallan los objetivos principales que se buscan alcanzar:

- **OBJ-1:** La aplicación deberá gestionar las sesiones de los usuarios de manera eficiente, permitiendo un acceso seguro y controlado a la información meteorológica. Esto incluye la creación, mantenimiento y expiración de sesiones para garantizar la seguridad y privacidad de los datos (véase la Tabla 1).

Tabla 1. Objetivo gestión de sesiones

OBJ-1	Gestión de sesiones
Descripción	El sistema debe almacenar y manejar información de las sesiones.
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

- **OBJ-2:** Es fundamental que la aplicación sea capaz de descargar información meteorológica de la misión Global Precipitation Measurement de la NASA según los criterios seleccionados y almacenarlos (véase la Tabla 2).

Tabla 2. Objetivo gestión de descarga

OBJ-2	Gestión de descarga
Descripción	El sistema debe gestionar los procedimientos para descargar y almacenar los datos ofrecidos por la misión GPM de la NASA
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

- **OBJ-3:** La aplicación deberá gestionar de manera eficiente los datos meteorológicos almacenados en la base de datos, permitiendo su fácil acceso y visualización (véase la Tabla 3).

Tabla 3. Objetivo gestión datos meteorológicos

OBJ-3	Gestión de datos meteorológicos
Descripción	El sistema debe gestionar los datos almacenados para su visualización
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

Requisitos de información

Una vez analizados los objetivos, será necesario elaborar los requisitos de información del sistema. Estos requisitos de información están diseñados para satisfacer las necesidades específicas del proyecto, asegurando que el sistema pueda manejar adecuadamente la información y proporcionar una experiencia de usuario óptima

- **RI-1 Sesión:** Este requisito de información se centra en la gestión de sesiones dentro del sistema. Es fundamental para garantizar la seguridad y la experiencia del usuario, ya que define cómo se almacenan y manejan los datos de las sesiones de los usuarios (véase la Tabla 4).

Tabla 4. Requisito de información sesión

RI-1	Sesión
Versión	1
Objetivos asociados	OBJ-1 Gestión de sesiones
Requisitos asociados	RF-1 Inicio sesión RF-2 Expiración sesión
Descripción	El sistema deberá almacenar información de las sesiones
Datos específicos	• Identificador de sesión • Nombre de usuario • Fecha de creación • Fecha de expiración
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

- **RI-2 Datos GPM:** Este requisito de información aborda la gestión de la descarga de datos meteorológicos proporcionados por la misión GPM de la NASA (véase la Tabla 5).

Tabla 5. Requisito de información datos GPM

RI-2	Datos GPM
Versión	1
Objetivos asociados	OBJ-2 Gestión de descarga de datos
Requisitos asociados	RF-3 Producto RF-4 Filtro
Descripción	El sistema debe ser capaz de descargar y almacenar información meteorológica proporcionada por la misión GPM de la NASA según criterios seleccionados por el usuario.
Datos específicos	• Identificador de sesión • Identificador de filtro • Información de producto: - Radiómetro - Satélite • Filtro: - Fechas - Longitud - Latitud

	- Altura - Variables
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

- **RI-3 Datos Meteorológicos:** Este requisito de información se centra en la gestión eficiente de los datos meteorológicos almacenados en la base de datos del sistema (véase la Tabla 6).

Tabla 6. Requisito de información datos meteorológicos

RI-3	Datos Meteorológicos
Versión	1
Objetivos asociados	OBJ-3 Gestión de datos meteorológicos
Requisitos asociados	RF-5 Visualización
Descripción	El sistema debe gestionar de manera eficiente los datos meteorológicos almacenados en la base de datos, permitiendo su fácil acceso y visualización
Datos específicos	• Identificador de filtro • Fecha • Datos coordenadas • Datos variables
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentario	Ninguno

Requisitos funcionales

Los requisitos funcionales son especificaciones detalladas que describen las funciones y características específicas que el sistema debe realizar para cumplir con los objetivos establecidos. Estos requisitos se centran en lo que el sistema debe hacer y cómo debe comportarse en respuesta a entradas específicas del usuario o del entorno. En esta sección, se detallarán las funciones principales que el sistema debe llevar a cabo para satisfacer las necesidades identificadas en los objetivos del proyecto. Cada requisito funcional describirá una acción

específica que el sistema debe realizar, junto con los criterios de aceptación asociados que permitirán verificar su correcto funcionamiento.

- **RF-1 Inicio sesión:** Este requisito funcional aborda la funcionalidad de inicio de sesión dentro del sistema. Es esencial para permitir que los usuarios accedan de manera segura a la aplicación. Se describe el proceso que un usuario debe seguir para iniciar sesión, incluyendo la entrada de su nombre y la generación de un identificador de sesión único (véase la Tabla 7).

Tabla 7. Requisito funcionar inicio sesión

RF-1	Inicio sesión
Versión	1
Objetivos asociados	OBJ-1 Gestión de sesiones
Requisitos asociados	RI-Sesión
Descripción	El sistema debe permitir a los usuarios iniciar sesión proporcionando su nombre.
Precondición	Ninguna
Secuencia normal	1.El usuario accede a la pantalla inicio 2. El usuario ingresa su nombre 3. El sistema inicia sesión con identificador de sesión único y con hora de creación
Postcondición	El usuario accede a la aplicación
Secuencia alternativa	
Rendimiento	Alto
Frecuencia esperada	Máxima
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentarios	Ninguno

- **RF-2 Expiración sesión:** Este requisito funcional trata sobre la expiración automática de sesiones después de un período de inactividad. Su objetivo es garantizar la seguridad de la sesión del usuario al cerrarla automáticamente si no hay actividad durante un tiempo especificado (véase la Tabla 8).

Tabla 8. Requisito funcional expiración sesión

RF-2	Expiración sesión
Versión	1
Objetivos asociados	OBJ-1 Gestión de sesiones
Requisitos asociados	RI-1 Sesión
Descripción	El sistema debe expirar las sesiones después de un cierto tiempo
Precondición	El usuario debe haber iniciado sesión.
Secuencia normal	1.Si el usuario no realiza ninguna actividad durante un período de tiempo predefinido: - El sistema detecta la inactividad - El sistema marca la sesión como expirada -El sistema elimina todos los datos asociados a la sesión
Postcondición	El usuario es redirigido a la pantalla de inicio de sesión, indicando que su sesión ha expirado.
Secuencia alternativa	
Rendimiento	Alto
Frecuencia esperada	Máxima
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentarios	Ninguno

- **RF-3 Producto:** Este requisito funcional se centra en la capacidad del sistema para permitir que los usuarios elijan un producto meteorológico específico para visualizar datos. Describe el proceso que un usuario debe seguir para seleccionar un producto, como un radiómetro o un satélite, y cómo el sistema valida esta selección (véase la Tabla 9).

Tabla 9. Requisito funcional producto

RF-3	Producto
Versión	1
Objetivos asociados	OBJ-2 Gestión de descarga
Requisitos asociados	RI-2 Datos GPM
Descripción	El sistema debe permitir elegir un producto del que visualizar datos
Precondición	El usuario debe haber iniciado sesión.
Secuencia normal	1.El usuario accede a la pantalla de selección de producto. 2. El usuario selecciona radiómetro. 3. El usuario selecciona satélite. 4. El sistema valida la selección del usuario
Postcondición	El usuario es redirigido a la pantalla de selección de filtros
Secuencia alternativa	
Rendimiento	Alto
Frecuencia esperada	Máxima
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentarios	Ninguno

- **RF-4 Filtro:** Este requisito funcional aborda la funcionalidad de filtrado de datos meteorológicos. Describe cómo los usuarios pueden especificar criterios de filtrado, como fechas, rangos de longitud y latitud, y variables meteorológicas, para descargar datos específicos. También incluye la validación de los filtros seleccionados y el proceso de descarga y almacenamiento de datos (véase la Tabla 10).

Tabla 10. Requisito funcional filtro

RF-4	Filtro
Versión	1
Objetivos asociados	OBJ-2 Gestión de descarga
Requisitos asociados	RI-2 Datos GPM
Descripción	El sistema debe permitir al usuario especificar los criterios de filtrado para la descarga de datos meteorológicos. Esto incluye seleccionar fechas, rangos de longitud y latitud, variables meteorológicas, entre otros parámetros
Precondición	El usuario debe haber elegido el producto del que quiere los datos.
Secuencia normal	1.El usuario accede a la pantalla de filtro. 2. El usuario marca los criterios de filtrado. 3. El sistema valida los filtros seleccionados. 4. El sistema descarga y almacena los datos en la base de datos
Postcondición	Los datos son guardados en la base de datos.
Secuencia alternativa	3.1 Si el usuario no introduce los filtros obligatorios se desactivará el botón de comienzo de descarga. 4.1 Si falla la descarga aparece un mensaje de error informando al usuario
Rendimiento	Alto
Frecuencia esperada	Máxima
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta

- **RF-5 Visualización:** Este requisito funcional se refiere a la capacidad del sistema para permitir que los usuarios visualicen datos meteorológicos almacenados de manera clara y comprensible. Describe el proceso de carga y visualización de datos, así como las opciones de interacción del usuario con la visualización de datos. Su objetivo es garantizar que los usuarios puedan comprender y analizar los datos meteorológicos de manera efectiva (véase la Tabla 11).

Tabla 11. Requisito funcional visualización

RF-5	Visualización
Versión	1
Objetivos asociados	OBJ-3 Gestión de datos meteorológicos
Requisitos asociados	RI-3 Datos meteorológicos
Descripción	El sistema debe permitir al usuario visualizar los datos meteorológicos almacenados en la base de datos de manera clara y comprensible. Esto incluye mostrar gráficos, tablas u otras representaciones visuales de los datos meteorológicos.
Precondición	El usuario debe haber seleccionado los datos meteorológicos deseados y aplicado los filtros necesarios.
Secuencia normal	1. El usuario accede a la pantalla de visualización de datos meteorológicos. 2. El sistema carga y muestra los datos meteorológicos según los criterios especificados. 3. El usuario interactúa con la visualización de datos
Postcondición	El usuario puede comprender y analizar los datos meteorológicos visualizados de manera efectiva.
Secuencia alternativa	
Rendimiento	Alto
Frecuencia esperada	Máxima
Importancia	Alta
Urgencia	Máxima
Estabilidad	Alta
Comentarios	Ninguno

2.2.2 Diagrama BBDD

Tras el exhaustivo análisis de los requisitos y funcionalidades que se desea que cumpla la descrita herramienta web, así como de los datos proporcionados por la misión Global Precipitation Measurement (GPM) de la NASA, se procedió a diseñar la base de datos (BBDD). Las conclusiones extraídas de la información de cómo están almacenados los datos meteorológicos de la misión GPM jugaron un papel crucial en la definición de la estructura y las relaciones de la base de datos. El diseño de la BBDD es una etapa crítica en cualquier proyecto de desarrollo de software, ya que sienta las bases para la gestión eficiente de la información.

Inicialmente, se utilizó un enfoque basado en un modelo de datos conceptual. Este enfoque permitió representar de manera abstracta las entidades y relaciones clave que forman parte del sistema, sin preocuparse por detalles de implementación como los tipos de variables o el lenguaje de programación a utilizar. De esta manera, se pudo comprender la estructura lógica de los datos y sus interacciones.

Este modelo representa las entidades principales y sus relaciones en la base de datos. Las tablas representan entidades como sesión, radiómetro, satélite, producto, filtro, variables seleccionadas, valores de capa y datos meteorológicos. Se ha optado por una base de datos relacional para asegurar la integridad y coherencia de los datos. Las relaciones entre estas entidades se establecen mediante claves primarias y claves foráneas para garantizar la integridad referencial y la coherencia de los datos. Todo esto proporciona una visión clara de la estructura de la base de datos y las relaciones entre las diferentes entidades, lo que facilita la comprensión y el diseño del sistema.

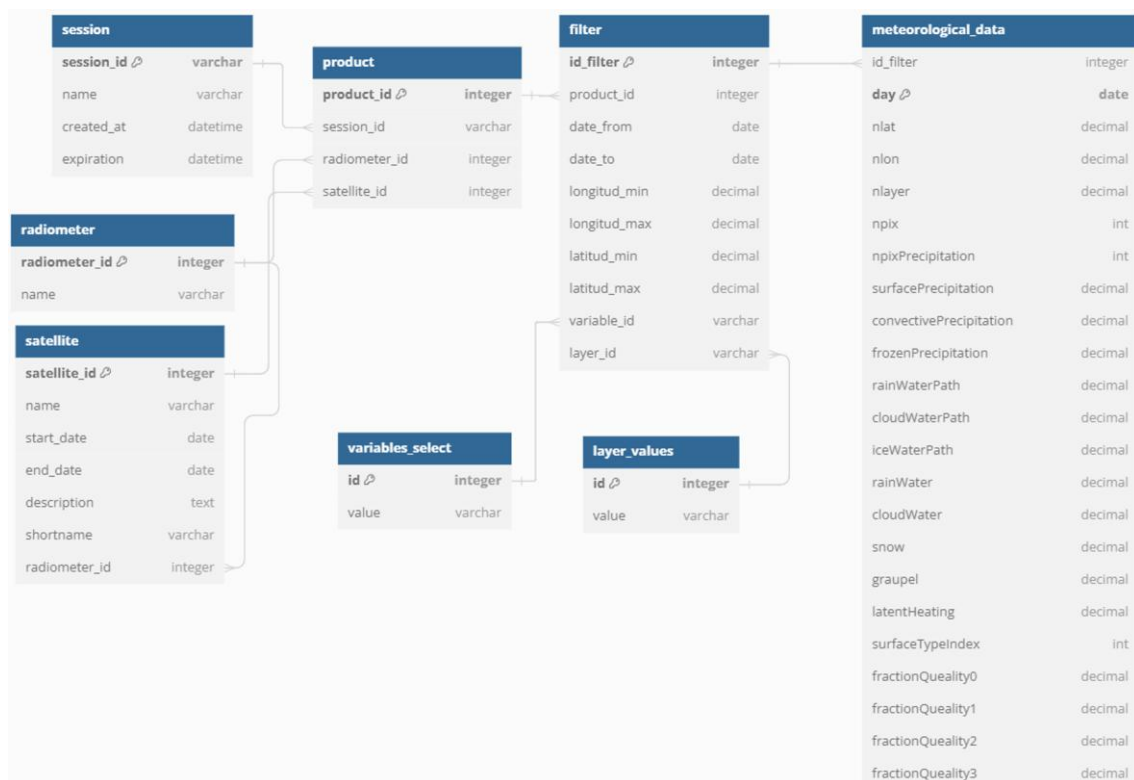


Ilustración 5: Diagrama BBDD del proyecto.

La Ilustración 5 muestra el esquema de la base de datos relacional diseñada para la herramienta web *GPMDDataExplorer*. Este esquema proporciona una representación visual de las entidades y de cómo estas se relacionan entre sí.

Las entidades que se han definido para la base de datos son:

- **Session**: Almacena información relacionada con las sesiones de usuario, como el identificador de sesión, el nombre de usuario, la fecha de creación y la fecha de expiración.
- **Radiometer**: Contiene los diferentes radiómetros utilizados en la misión GPM con un campo id y el nombre del radiómetro.
- **Satellite**: Contiene información sobre los satélites utilizados por cada radiómetro de la misión GPM. Incluye su identificador, nombre, fecha de inicio de actividad, fecha de fin de actividad, descripción y el nombre corto asociado. También incluye un campo para la relación con el radiómetro correspondiente.
- **Product**: Almacena la información necesaria para identificar el usuario, y el producto seleccionado asociándolo a los criterios de filtrado correspondientes. Incluye el identificador de sesión del usuario, los identificadores de radiómetro y satélite seleccionados por el usuario.
- **Filter**: Contiene los criterios de filtrado seleccionados por el usuario para la descarga de los datos meteorológicos. Entre los filtros disponibles se encuentra el rango de fechas, rango de región, variables meteorológicas y capas verticales(layers).

- *Variables_select*: Almacena un listado de todas las variables meteorológicas disponibles que el usuario puede seleccionar para el filtrado.
- *Layer_values*: Contiene las capas verticales de altitud de las que se mostraran los datos.
- *Meteorological_data*: Guarda los datos meteorológicos descargados, incluyendo información sobre la fecha, las coordenadas, y los datos de cada una de las variables seleccionadas.

Al usar un modelo relacional estas relaciones entre entidades se establecen mediante claves primarias y claves foráneas.

Una clave primaria es un atributo o conjunto de atributos dentro de una tabla cuyos valores son únicos para cada fila. Se utilizan para identificar cada registro de la tabla. Como se muestra en la Ilustración 5 se ha utilizado distintas claves primarias como podrían ser *sesión_id* para la entidad *session* o *radiometer_id* para la entidad *radiometer*. En entidad *meteorological_data* se decidió como clave primaria *day* ya que representa la fecha asociada a cada registro.

Una clave foránea es un atributo en una tabla que establece una relación con la clave primaria de otra tabla. Esta clave vincula datos de dos tablas relacionadas. Por ejemplo, se puede observar que la tabla *satellite* establece una relación con la tabla *radiometer* a través de la clave foránea *radiometer_id*, así se vincula cada satélite con su radiómetro correspondiente. De manera similar, se puede observar que en la tabla *product* las claves foráneas *session_id*, *radiometer_id* y *satellite_id* establecen relaciones con las tablas *session*, *radiometer* y *satellite* respectivamente, asociando cada producto con la sesión de usuario y los instrumentos correspondientes.

2.2.3 Arquitectura del Sistema

Para garantizar un desarrollo eficiente y mantenible de la aplicación, se ha optado por utilizar el patrón de diseño Modelo-Vista-Controlador (MVC) [7]. Este enfoque divide la aplicación en tres componentes principales (vistas, modelos y controladores), cada uno con una responsabilidad específica, separando la lógica de la aplicación. Esto facilita la gestión del código y la escalabilidad del sistema. Fue desarrollado a finales de los años 70 por Trygve M. H. Reenskaug, quien trabajaba en el proyecto SmallTalk de Xerox.

El patrón MVC está compuesto por los siguientes elementos:

Modelo

Representa la lógica y los datos de la aplicación, y generalmente se encuentra en el *backend* del sistema. En nuestro caso, el Modelo está compuesto por las clases y métodos encargados de interactuar con la base de datos, gestionando la lógica de la aplicación. Esto incluye la manipulación de datos meteorológicos,

la gestión de sesiones de usuario y la aplicación de filtros sobre los datos obtenidos.

Vista

Es la interfaz gráfica, se encarga de la presentación de la información al usuario y de la interacción con él. En la aplicación, las vistas corresponden a las páginas web y elementos de interfaz gráfica que permiten al usuario interactuar con la aplicación. Esto incluye el inicio de sesión, la elección de producto por parte del usuario, así como el formulario de filtros o la visualización de los datos meteorológicos.

Controlador

El controlador actúa como intermediario entre el modelo y la vista, gestionando las solicitudes del usuario, procesando la entrada de datos y actualizando el modelo en consecuencia. En la aplicación, los Controladores son los encargados de recibir las peticiones HTTP del cliente, llamar a los métodos apropiados del modelo para realizar las operaciones necesarias y seleccionar la vista adecuada para mostrar los resultados al usuario.

En la Ilustración 6 se puede ver el esquema del patrón Modelo-Vista-Controlador (MVC).

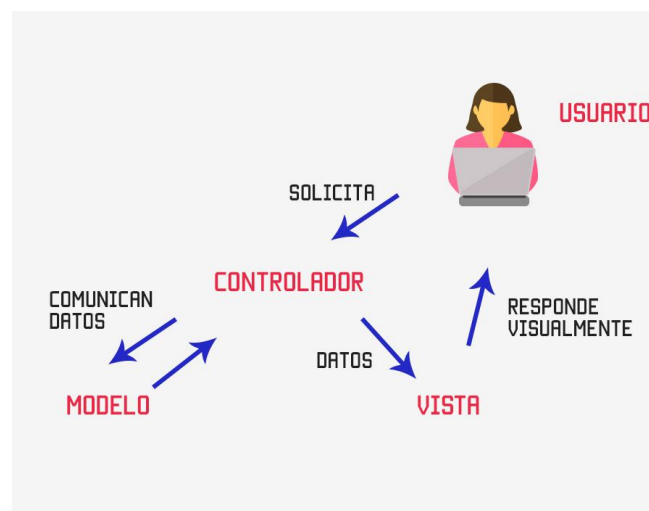


Ilustración 6: Modelo-Vista-Controlador(MVC) [8].

Esta arquitectura basada en el patrón MVC nos permite mantener una separación clara entre la lógica de negocio, la presentación de la información y la gestión de la interacción con el usuario. Esto facilita la modularidad del código, la reutilización de componentes y el mantenimiento a largo plazo de la aplicación.

2.2.4 Tecnologías Utilizadas

En este apartado, se describen las principales tecnologías y herramientas utilizadas en el desarrollo del proyecto, sin entrar en las que son particulares de *backend*, *frontend* y base de datos las cuales serán descritas más adelante. Estas herramientas nombradas a continuación desempeñaron un papel crucial en el proceso de desarrollo, desde el diseño y la implementación hasta la gestión y colaboración con el tutor.

Dbdiagram.io

Una herramienta en línea que permite diseñar y visualizar esquemas de bases de datos de manera intuitiva. Permite crear y editar tablas, establecer relaciones entre ellas y generar automáticamente el código SQL (Structured Query Language) correspondiente. Su interfaz fácil de usar facilita el diseño y mantenimiento de la base de datos. Esta herramienta se utilizó para la creación del diseño de la base de datos.



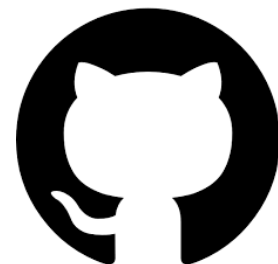
Visual Studio Code



Es un entorno de desarrollo integrado (IDE) altamente personalizable y de código abierto desarrollado por Microsoft. Ofrece características avanzadas para la edición de código. Además, es compatible con una amplia variedad de lenguajes de programación y ofrece una amplia gama de extensiones para personalizar y ampliar sus funcionalidades. Esta herramienta se utilizó para desarrollar la aplicación.

GitHub

Es una plataforma de desarrollo colaborativo basada en la nube que proporciona control de versiones utilizando Git. Permite a los equipos de desarrollo colaborar en proyectos de software. GitHub [9] también ofrece herramientas para la integración continua, el despliegue continuo y la automatización del flujo de trabajo de desarrollo. Se puede acceder al repositorio del proyecto en GitHub a través del siguiente enlace:



<https://github.com/WFreezer/GPMDDataExplorer.git>

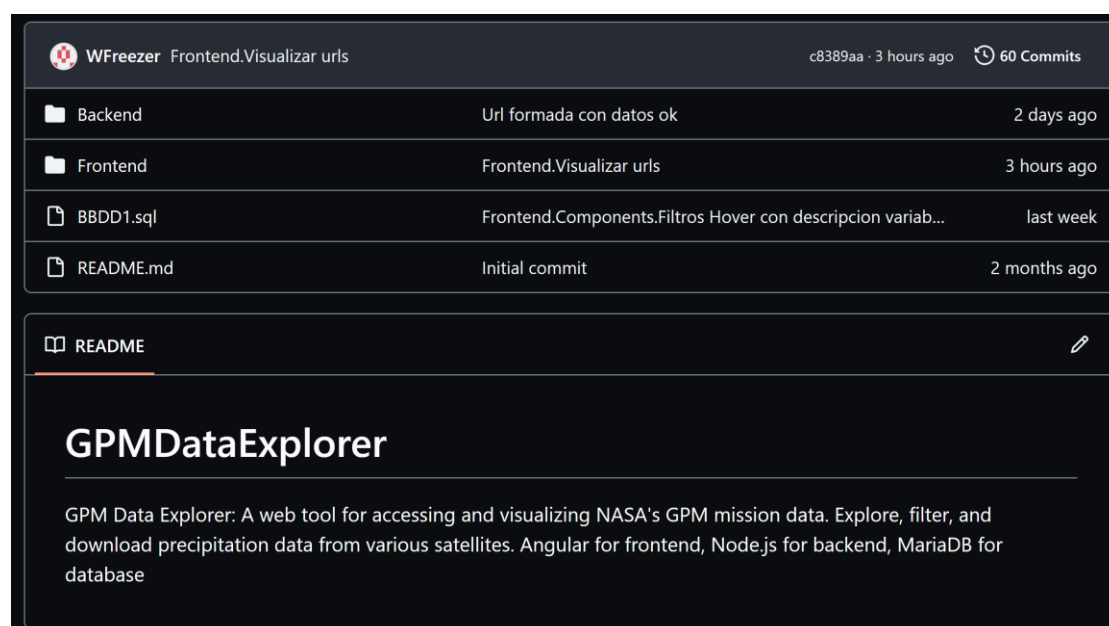


Ilustración 7: Repositorio GitHub del proyecto [9].

2.3 Diseño de Bajo Nivel

En la próxima sección, se presentará una descripción más detallada y técnica de la implementación de la aplicación *GPMDDataExplorer*. Después de establecer la estructura general del proyecto en el apartado anterior, nos centraremos en aspectos más específicos y técnicos de la implementación. Se examinarán tres áreas principales: el *backend*, la base de datos y el *frontend*. Este análisis nos permitirá comprender cómo se han cumplido los requisitos del sistema en términos de arquitectura, tecnologías utilizadas, seguridad y la interacción entre los diferentes componentes.

Es crucial reconocer la importancia del diseño de bajo nivel en el proceso de desarrollo de software. Este nivel de diseño se centra en la implementación específica de los componentes del sistema, detallando aspectos como la estructura de datos, la lógica y la interacción entre los diferentes módulos. Un diseño de bajo nivel bien planificado puede contribuir a la eficiencia, la escalabilidad y la mantenibilidad del proyecto. Por lo tanto, esta sección desempeña un papel fundamental. Durante todo el proceso de desarrollo, se ha tenido como objetivo principal lograr un equilibrio entre la eficiencia, la claridad y la modularidad para garantizar la robustez y sostenibilidad de la aplicación.

2.3.1 Backend

El *backend* o servidor, constituye el núcleo de la aplicación donde se encuentra la lógica de la aplicación web. Cuando el usuario o cliente interactúa con el *frontend* de la aplicación, este envía una solicitud al *backend* en formato HTTP/HTTPS. El *backend* procesa la solicitud, interactúa con la base de datos y la API, y devuelve una respuesta. En los siguientes apartados se analizará la

arquitectura del *backend*, las tecnologías utilizadas para desarrollar esta parte y la implementación de la API REST que permite la comunicación con el *frontend*.

La siguiente figura ilustra cómo interactúan el *frontend*, el *backend* y la base de datos, así como el flujo de las solicitudes HTTP/HTTPS y el papel de la API REST en la comunicación entre estas partes. En la Ilustración 8 se puede ver la arquitectura de interacción entre *frontend* y *backend*.

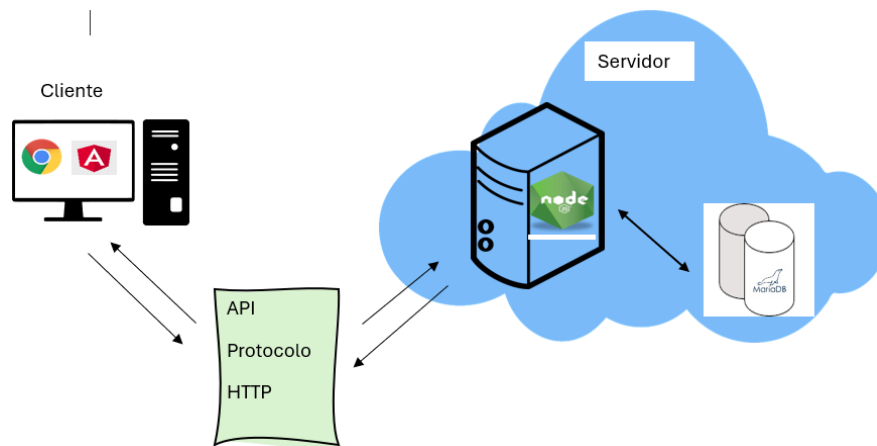


Ilustración 8: Arquitectura de interacción entre frontend y backend.

2.3.1.1 Acceso a datos con OPeNDAP

Antes de profundizar en la arquitectura del *backend*, es importante destacar el proceso de evaluación de las opciones disponibles para la obtención de datos de la misión Global Precipitation Measurement (GPM). Se exploraron diversas alternativas con el objetivo de seleccionar la más adecuada para nuestro proyecto. Tras evaluar estas opciones, se optó por utilizar OPeNDAP (Open-source Project for a Network Data Access Protocol) como el método principal para la obtención de datos de la misión GPM. OPeNDAP ofrece la posibilidad de acceder a los datos en formato ASCII, lo que facilita su manejo y procesamiento sin requerir un conocimiento profundo de formatos de archivo especializados como HDF5. Esta característica resultó fundamental para nuestro proyecto, ya que nos permitió aprovechar los datos de GPM de manera efectiva sin tener que enfrentar una curva de aprendizaje complicada.

OPeNDAP ofrece un servidor o base de datos con los datos de la misión GPM, que incluyen información crucial sobre la precipitación global. La aplicación accede a estos datos a través de URLs específicas, que permiten recuperar información detallada de cada uno de los conjuntos de radiómetro-satélite [10].

Name	Last Modified	Size
<u>GPM_3CMB.07/</u>	2024-03-09T07:10:24GMT	-
<u>GPM_3CMB_DAY.07/</u>	2024-03-08T13:07:22GMT	-
<u>GPM_3DPR.07/</u>	2024-03-10T14:24:55GMT	-
<u>GPM_3DPRD.07/</u>	2024-03-10T14:44:56GMT	-
<u>GPM_3DPR_ASC.07/</u>	2024-03-10T14:31:26GMT	-
<u>GPM_3DPR_DES.07/</u>	2024-03-10T14:41:05GMT	-
<u>GPM_3GCSH.07/</u>	2024-03-08T13:08:09GMT	-
<u>GPM_3GPROFAQAAMSRE_CLIM.07/</u>	2022-08-15T21:27:59GMT	-
<u>GPM_3GPROFAQAAMSRE_DAY_CLIM.07/</u>	2022-08-16T00:08:53GMT	-
<u>GPM_3GPROFF08SSMI_CLIM.07/</u>	2022-08-22T15:08:02GMT	-
<u>GPM_3GPROFF08SSMI_DAY_CLIM.07/</u>	2022-08-22T16:39:41GMT	-
<u>GPM_3GPROFF10SSMI_CLIM.07/</u>	2022-08-22T17:00:40GMT	-
<u>GPM_3GPROFF10SSMI_DAY_CLIM.07/</u>	2022-08-22T18:42:43GMT	-

Ilustración 9: Acceso a datos de GPM por OPeNDAP [11].

Estructura de las URLs

La estructura de las URLs de OPeNDAP se compone de varios elementos que especifican el tipo de datos requeridos, la fecha y la ubicación geográfica. En la Ilustración 9 se muestra el contenido de la URL base de la cual parten todos los conjuntos de datos.

Al agregar "/GPM_3GPROFGPMGMI_DAY.07" a la URL base, accedemos a los datos del radiómetro GMI del satélite GPM. Al incluir también el año y el mes en la URL, como "/2024/02/", se puede obtener datos para un período de tiempo específico. Una vez configurada la URL base, se puede acceder a datos específicos mediante la adición de parámetros a la URL. Por ejemplo, utilizando una serie de parámetros como "latentHeating", "fractionQuality", "surfacePrecipitation", entre otros, se pueden obtener datos detallados sobre la precipitación en una región geográfica específica y en un momento determinado.

Es importante mencionar que los datos se almacenan en un formato de cuadrícula, lo que permite una organización estructurada de la información. En este proyecto web, se realiza la conversión de cada longitud y latitud a la cuadrícula correspondiente.

La aplicación modifica dinámicamente las URLs de OPeNDAP según los parámetros seleccionados por el usuario, lo que permite la recuperación de datos precisos y relevantes para su análisis y visualización. Esta integración con OPeNDAP permite aprovechar una fuente de datos confiable y completa para enriquecer la funcionalidad de la aplicación.

2.3.1.2 Tecnologías Utilizadas

A continuación, se mostrarán las herramientas, lenguajes y tecnologías utilizadas para el desarrollo del *backend* de este proyecto.

Node.js

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome [12]. Permite usar JavaScript para escribir código del lado del servidor, lo que posibilita la creación de aplicaciones escalables y rápidas.

Node.js se eligió para este proyecto debido a su popularidad en el mercado y a que ya era conocido por el equipo de desarrollo. También fue importante para su selección la alta disponibilidad de módulos y paquetes que tiene a través de npm (Node Package Manager) facilitando la incorporación de funcionalidades.

Express

Express es un framework web flexible para Node.js. Proporciona un conjunto de características para construir aplicaciones web de forma fácil y rápida. Proporciona mecanismos para la construcción de APIs, incluyendo la gestión de rutas y la manipulación de peticiones y respuestas HTTP [13].

Express se eligió para este proyecto debido a su facilidad de uso y a su popularidad.

Postman

La herramienta Postman permite probar y documentar la API desarrollada. Tiene una interfaz simple y amigable con la que se pueden crear colecciones para agrupar las distintas solicitudes y peticiones HTTP. En el desarrollo del *backend* del proyecto esta herramienta ha sido especialmente útil para probar los distintos *endpoints* y su correcto funcionamiento.

Aunque más adelante este documento se llevará a cabo un análisis más exhaustivo de las pruebas realizadas en la creación de la API, en la Ilustración 10 se muestra un pequeño ejemplo de una petición HTTP.

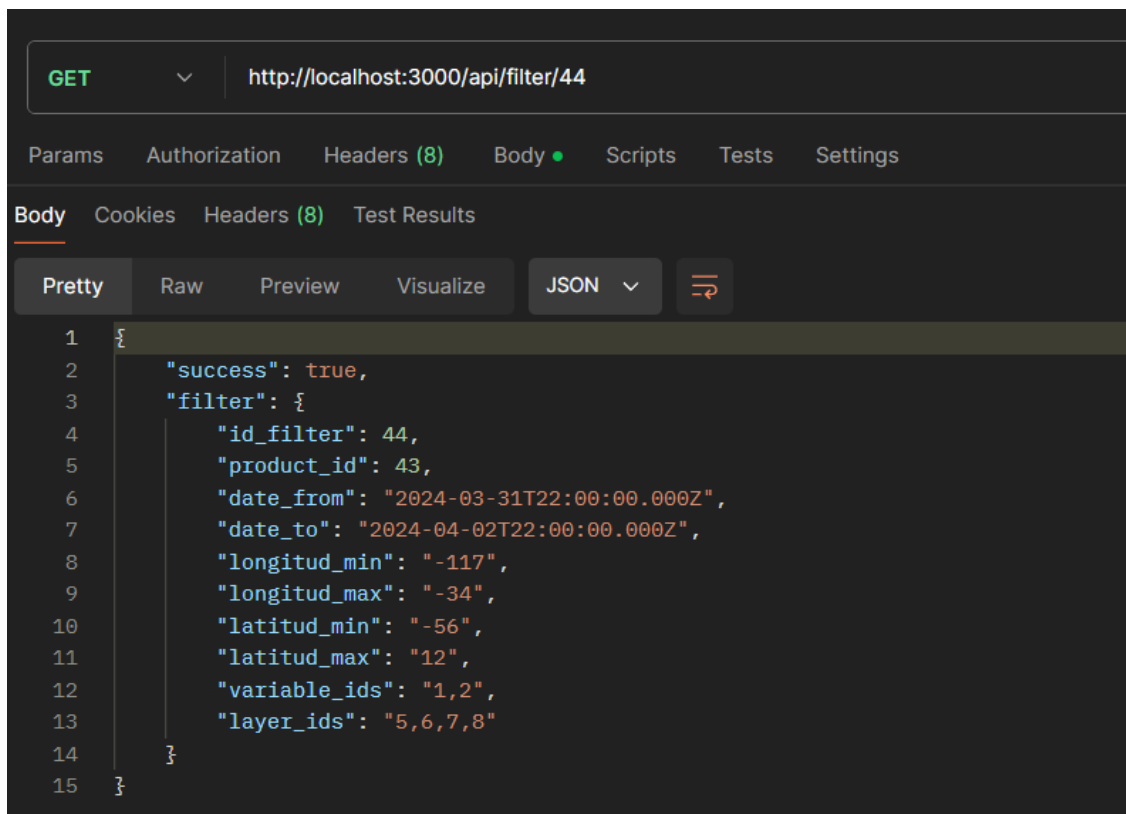


Ilustración 10: Ejemplo petición GET a Postman.

Biblioteca ExcelJS

ExcelJS es una biblioteca de JavaScript que permite crear, leer y manipular archivos Excel (.xlsx) en aplicaciones Node.js. Esta biblioteca fue necesaria para procesar los datos, anteriormente nombrados del documento “Coodenadas.xlsx”.

2.3.1.3 Arquitectura Backend

En esta sección, se explorará la arquitectura del *backend* de la aplicación *GPMDDataExplorer*. Se comenzará presentando una visión general de la estructura de carpetas que conforma el *backend*, seguido de una breve discusión sobre su organización y propósito.

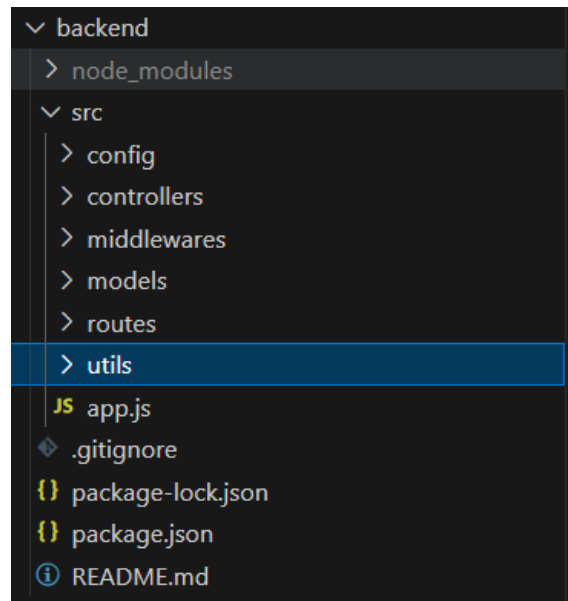


Ilustración 11: Estructura backend.

En la ilustración anterior se muestra la estructura de carpetas del *backend* de la aplicación, siguiendo los estándares de organización recomendados para proyectos Node.js. Esta organización modular de carpetas facilita la navegación y el mantenimiento del código, al mismo tiempo que separa las responsabilidades de los distintos componentes. Como ya se ha mencionado en el apartado “2.2.3 Arquitectura del Sistema” se ha seguido un patrón modelo-vista-controlador para desarrollar la herramienta web *GPMDDataExplorer*.

En un primer vistazo se pueden observar algunos archivos esenciales que son comunes en cualquier proyecto Node.js. Estos archivos desempeñan roles específicos en la configuración y el funcionamiento del proyecto:

- **node_modules:** Esta carpeta almacena las dependencias de Node.js utilizadas en el proyecto. Aquí se encuentran todas las bibliotecas y paquetes necesarios para la ejecución adecuada de la aplicación
- **.gitignore:** Se trata de un archivo que especifica los archivos y directorios que deben ser ignorados por el sistema de control de versiones Git. Esto es necesario para evitar subir al repositorio archivos innecesarios como las dependencias instaladas o archivos de configuración local.
- **package-lock.json:** Fichero generado automáticamente por npm (Node Package Manager), contiene información detallada sobre las versiones de todas las dependencias instaladas. Es el encargado de garantizar que las versiones de las dependencias sean consistentes.
- **package.json:** Este archivo contiene metadatos importantes sobre el proyecto, como el nombre, la versión, las dependencias y los scripts de inicio. Es esencial para describir la configuración y el comportamiento del proyecto, así como para gestionar sus dependencias de manera efectiva.
- **README.md:** Este archivo contiene la documentación del proyecto.

También se puede observar el archivo `app.js`, el cuál funciona como punto de entrada principal del *backend*. En este archivo, se configura y se inicia el servidor. En este archivo, se establecen las bases para el funcionamiento del

servidor utilizando el *framework* Express.js, altamente popular en el desarrollo de aplicaciones web con Node.js

```
// app.js
const express = require('express');
const cors = require('cors');
const db = require('./config/dbconnector');
const routes = require('./routes');
const cleanExpiredSessions = require('./middlewares/cleanExpiredSessions');
const cookieParser = require('cookie-parser');

const app = express();
const port = process.env.PORT || 3000;

// Para analizar solicitudes JSON
app.use(express.json());

// Para analizar solicitudes de URL codificadas
app.use(express.urlencoded({ extended: true }));

// Middleware para permitir solicitudes de otros dominios (CORS)
app.use(cors());

// Middleware para manejo de cookies
app.use(cookieParser());

// Usar cleanExpiredSessions (limpia sesiones expiradas)
app.use(cleanExpiredSessions);

// Montar las rutas
app.use('/api', routes);

// Iniciar el servidor
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

Ilustración 12: Archivo app.js.

Como se puede observar en la ilustración anterior, aquí se configura desde la importación de los módulos necesarios, como express, cors, y cookie-parser, hasta la configuración de middleware esenciales como express.json() para analizar solicitudes JSON y express.urlencoded() para las de URL codificadas, cada línea de código cumple un papel crucial en la preparación del servidor para manejar las solicitudes entrantes.

Además de la configuración básica, el archivo app.js también se encarga de la conexión a la base de datos, asegurando que el servidor esté listo para interactuar con ella cuando sea necesario.

Es importante mencionar que al no utilizar un sistema de sesiones donde haya un *login* y un *register*, si no sesiones que expiran pasado un tiempo, es necesario incluir el middleware cleanExpiredSessions, esencial para mantener la integridad y eficiencia del sistema al eliminar sesiones expiradas antes de procesar cualquier solicitud. Utiliza consultas a la base de datos para identificar y eliminar las sesiones cuya fecha de expiración ha pasado, junto con los datos asociados, como filtros, productos y datos meteorológicos. Este proceso garantiza que la base de datos esté limpia y libre de sesiones obsoletas, lo que contribuye a un rendimiento óptimo del sistema.

El montaje de las rutas definidas en el archivo routes.js es otro aspecto crucial. Al importar y montar estas rutas en el *endpoint* '/api', se define cómo se manejarán las solicitudes HTTP entrantes y cómo se responderá a ellas, estableciendo así la estructura y funcionalidad del *backend* de la aplicación.

Finalmente, con el uso del método app.listen(), se inicia el servidor, escuchando las solicitudes entrantes en el puerto especificado. Una vez en funcionamiento,

se imprime un mensaje en la consola indicando la dirección y el puerto en los que está escuchando el servidor.

Ahora se pasará a definir el contenido de cada una de las carpetas que componen el *backend*, para proporcionar así una visión completa de la arquitectura.

Config

En esta carpeta se ha desarrollado un único archivo, `dbconnector.js`, el cual desempeña un papel crucial en la configuración y gestión de la conexión con la base de datos utilizada en la aplicación.

El archivo `dbconnector.js` comienza importando el módulo `mysql2`, que proporciona las herramientas necesarias para interactuar con una base de datos desde una aplicación Node.js. A continuación, se define un objeto `dbConfig` que contiene la información de configuración necesaria para establecer la conexión a la base de datos. Esto incluye el host de la base de datos, el puerto, el nombre de usuario, la contraseña y el nombre de la base de datos.

Una vez configurado, se crea un *pool* de conexiones utilizando la función `mysql.createPool()` proporcionada por el módulo `mysql2`. Este *pool* gestiona varias conexiones a la base de datos y las reutiliza según sea necesario, lo que mejora la eficiencia y el rendimiento de la aplicación.

Además, se define una función `query()` que permite ejecutar consultas SQL en la base de datos. Esta función devuelve una promesa que se resuelve con los resultados de la consulta o se rechaza con un error si la consulta falla. Dentro de esta función, se obtiene una conexión del *pool* de conexiones, se ejecuta la consulta y se libera la conexión una vez completada la operación.

Finalmente, se exporta la función `query` para que pueda ser utilizada en otros módulos de la aplicación.

Es importante mencionar que la configuración proporcionada en `dbconnector.js` está optimizada para un entorno de desarrollo local. En un entorno de producción, es posible que se requieran ajustes adicionales para garantizar la seguridad y el rendimiento óptimo de la conexión a la base de datos.

Controllers

La carpeta `controllers` desempeña un papel central en la API del *backend* de la aplicación, ya que actúa como la capa intermedia entre las solicitudes HTTP entrantes y la lógica de la aplicación. Cada archivo dentro de esta carpeta corresponde a un controlador específico, diseñado para manejar las solicitudes relacionadas con áreas particulares de la aplicación, como la gestión de filtros, productos, sesiones, entre otros (Ilustración 13).

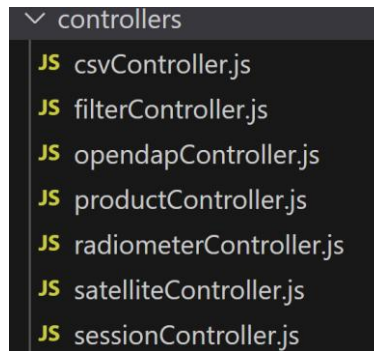


Ilustración 13: Contenido carpeta controllers.

Estos controladores son los encargados de recibir las solicitudes HTTP, procesar los datos proporcionados por el cliente y coordinar las operaciones necesarias para cumplir con la solicitud. Para llevar a cabo estas operaciones, los controladores interactúan estrechamente con los modelos de la aplicación, los cuales representan la estructura de datos y proporcionan métodos para interactuar con la base de datos.

Por ejemplo, en el archivo `productController.js`, se definen funciones de controlador como `createProduct`, `getAllProducts` y `getProductsBySessionId`, las cuales son responsables de crear nuevos productos, obtener todos los productos almacenados y obtener productos asociados a una sesión específica, respectivamente. Estas funciones utilizan métodos proporcionados por el modelo `Product` para realizar operaciones en la base de datos.

Esta separación de responsabilidades entre controladores y modelos sigue el principio de arquitectura MVC.

```
// Controlador para crear un nuevo producto
const createProduct = async (req, res) => {
  try {
    // Obtener los datos del cuerpo de la solicitud
    const { session_id, radiometer_id, satellite_id } = req.body;

    console.log('Datos recibidos:', { session_id, radiometer_id, satellite_id });

    // Verificar si el session_id existe en la tabla session
    const sessionExists = await Session.exists(session_id);

    if (!sessionExists) {
      return res.status(404).json({ message: 'La sesión no existe' });
    }

    console.log('Sesión existe: ' + sessionExists);
    // Crear el nuevo producto en la base de datos
    const newProduct = await Product.create(session_id, radiometer_id, satellite_id);

    console.log('Nuevo producto creado:', newProduct);

    // Enviar respuesta con el nuevo producto creado
    res.status(201).json(newProduct);
  } catch (error) {
    // Manejar errores
    console.error('Error creating product:', error);
    res.status(500).json({ message: 'Error creating product' });
  }
};
```

Ilustración 14: Fragmento de código de `ProductController.js`.

El fragmento de código que aparece en la Ilustración 14 representa un controlador específico, en este caso, el controlador para crear un nuevo producto en la base de datos. Este controlador es responsable de manejar las

solicitudes POST de la API que contienen datos para crear un nuevo producto. Al recibir la solicitud, el controlador extrae los datos necesarios del cuerpo de la solicitud, como el `session_id`, `radiometer_id` y `satellite_id`. Luego, verifica si la sesión asociada existe en la base de datos utilizando un método del modelo `Session`. Si la sesión no existe, devuelve un código de estado 404 junto con un mensaje de error. Si la sesión existe, utiliza un método del modelo `Product` para crear un nuevo producto en la base de datos con los datos proporcionados. Una vez que se crea el producto con éxito, el controlador devuelve un código de estado 201 junto con los detalles del nuevo producto en formato JSON. En caso de que ocurra algún error durante el proceso, el controlador maneja el error adecuadamente y devuelve un código de estado 500 junto con un mensaje de error. Este controlador ejemplifica la interacción entre los controladores y los modelos, donde los controladores utilizan métodos proporcionados por los modelos para interactuar con la base de datos y completar las operaciones solicitadas a través de la API.

Middleware

Esta carpeta contiene un archivo denominado `cleanExpiredSession.js`, el cual se ha mencionado anteriormente. Este middleware desempeña un papel fundamental en la aplicación al limpiar las sesiones expiradas antes de procesar cualquier solicitud. Esta funcionalidad contribuye significativamente a la integridad y eficiencia del sistema, garantizando que la base de datos esté libre de sesiones obsoletas y optimizando el rendimiento de la aplicación.

Es importante comprender el papel de los middlewares en general en una aplicación web. Los middlewares actúan como capas intermedias entre el cliente y el servidor, manipulando solicitudes y respuestas antes de que lleguen a su destino final. Estos componentes desempeñan diversas funciones, como la validación de datos, la autenticación de usuarios, la gestión de sesiones y la seguridad, entre otras.

Models

Aquí se albergan los archivos que definen los modelos de datos utilizados en la aplicación. Estos modelos son representaciones de las entidades principales con las que interactúa la aplicación y están diseñados para interactuar directamente con la base de datos. Los modelos proporcionan métodos para acceder, manipular y gestionar los datos almacenados en la base de datos, lo que los convierte en componentes fundamentales para la funcionalidad de la aplicación.

Cada modelo en la carpeta `models` se corresponde con una tabla específica en la base de datos y define la estructura y el comportamiento de esa tabla. Los modelos son llamados desde los controladores de la aplicación para realizar operaciones relacionadas con la base de datos, como recuperar datos, crear nuevos registros o actualizar información existente. Esto establece una clara separación de responsabilidades en la arquitectura de la aplicación, donde los controladores manejan la lógica y la interacción con el cliente, mientras que los modelos se encargan de la gestión de los datos subyacentes.

A continuación, se presenta una breve descripción de cada modelo en la carpeta `models`:

- **Filter:** Este modelo representa los filtros seleccionados por el usuario y utilizados en la aplicación para filtrar los datos meteorológicos según diversos criterios. Proporciona métodos para recuperar y manipular la información de los filtros almacenados en la base de datos.
- **Product:** El modelo `Product` representa los productos seleccionados por el usuario para obtener los datos, que radiómetro y que satélite dentro de ese radiómetro. Contiene métodos para recuperar todos los productos, obtener productos por ID de sesión y crear nuevos productos.
- **Radiometer:** Este modelo representa los radiómetros disponibles para la recopilación de datos meteorológicos. Aunque la información sobre los radiómetros ya está almacenada en la base de datos, el modelo proporciona métodos para acceder a esta información desde la aplicación. Esto incluye la recuperación de detalles sobre los radiómetros disponibles y la gestión de su información asociada, permitiendo a la aplicación interactuar de manera efectiva con estos dispositivos en el contexto de la recopilación de datos meteorológicos.
- **Satellite:** El modelo `Satellite` representa los satélites disponibles para la recopilación de datos meteorológicos. Al igual que con los radiómetros, la información sobre los satélites ya está almacenada en la base de datos. El modelo proporciona métodos para recuperar esta información desde la aplicación. Esto permite a la aplicación obtener detalles sobre los satélites disponibles y gestionar su información asociada, facilitando así su integración en el proceso de recopilación de datos meteorológicos.
- **Session:** El modelo `Session` representa las sesiones de usuario en la aplicación. Contiene métodos para crear nuevas sesiones y mantener la información de sesión asociada a cada usuario.
- **Csv:** Este modelo facilita la importación y gestión de datos meteorológicos desde archivos CSV en la aplicación. Permite la lectura de archivos CSV y su posterior inserción de los datos en la correspondiente tabla. También proporciona la funcionalidad para la recuperación de estos datos meteorológicos filtrados con el identificador y el día de los datos a recuperar.

Estos modelos desempeñan un papel crucial en el funcionamiento de la aplicación, proporcionando una interfaz entre la lógica de la aplicación y la capa de almacenamiento de datos. Al interactuar directamente con la base de datos, los modelos garantizan la integridad y consistencia de los datos, al tiempo que facilitan la implementación de la lógica de negocio en la aplicación.

Routes

Esta carpeta contiene archivos que definen las rutas de la API para diferentes recursos y funcionalidades de la aplicación. Cada archivo de ruta se encarga de manejar las solicitudes HTTP relacionadas con un tipo específico de recurso o funcionalidad.

Por ejemplo, `productRoutes.js` maneja las solicitudes relacionadas con los productos, mientras que `sessionRoutes.js` maneja las solicitudes relacionadas con las sesiones de usuario.

El archivo `index.js` es el punto de entrada principal que exporta un *router* de Express configurado con todas las rutas definidas en los otros archivos de la carpeta. Esto permite un enrutamiento modular y organizado en la aplicación.

```
const express = require('express');
const router = express.Router();

const sessionRoutes = require('./sessionRoutes');
const productsRoutes = require('./productRoutes');
const radiometerRoutes = require('./radiometerRoutes');
const satelliteRoutes = require('./satelliteRoutes');
const filterRoutes = require('./filterRoutes');
const opendapRoutes = require('./opendapRoutes');
const csvRoutes = require('./csvRoutes');

router.use('/sessions', sessionRoutes);
router.use('/products', productsRoutes);
router.use('/radiometers', radiometerRoutes);
router.use('/satellites', satelliteRoutes);
router.use('/filter', filterRoutes);
router.use('/opendap', opendapRoutes);
router.use('/csv', csvRoutes);

module.exports = router;
```

Ilustración 15: Archivo index.js.

Como se puede observar en la Ilustración 15, `router.use()` se utiliza para montar las rutas definidas en los archivos individuales en rutas base específicas, como `/sessions`, `/products`, `/radiometers`, `/satellites`, `/filter`, `/opendap` y `/csv`. Esto facilita la organización y la gestión de las rutas de la API, agrupándolas por funcionalidad y recursos.

Además de definir la estructura y organización de las rutas de la API, cada archivo de ruta dentro de la carpeta `routes` implementa las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para un tipo específico de recurso o funcionalidad. Tomemos como ejemplo el archivo `productRoutes.js`.

Este archivo define las rutas para interactuar con los productos disponibles en la aplicación. Utilizando el framework `Express.js`, se definen varias rutas HTTP que corresponden a diferentes operaciones en los productos, y se asignan a funciones específicas del controlador de productos (`productController`). En la Ilustración 16 se puede observar las siguientes rutas:

- **Ruta POST /:** Esta ruta permite a los usuarios crear un nuevo producto seleccionado. Cuando se realiza una solicitud POST a esta ruta, el controlador de productos `createProduct` se encarga de procesar la solicitud y crear el producto correspondiente en la base de datos.
- **Ruta GET /:** Esta ruta permite obtener todos los productos disponibles en la aplicación. Al realizar una solicitud GET a esta ruta, el controlador de productos `getAllProducts` se encarga de recuperar todos los productos almacenados en la base de datos y devolverlos como respuesta.

- **Ruta GET /session/:sessionId:** Esta ruta permite obtener los productos asociados a una sesión específica de usuario. Al realizar una solicitud GET a esta ruta con el ID de sesión proporcionado como parámetro, el controlador de productos `getProductsBySessionId` se encarga de recuperar los productos asociados a esa sesión en particular y devolverlos como respuesta.

```
// productRoutes.js

const express = require('express');
const router = express.Router();
const productController = require('../controllers/productController');

// Ruta para crear un producto seleccionado por usuario
router.post('/', productController.createProduct);

// Ruta para obtener todos los productos
router.get('/', productController.getAllProducts);

// Ruta para obtener los productos asociados a una sesión específica
router.get('/session/:sessionId', productController.getProductsBySessionId);

module.exports = router;
```

Ilustración 16: Archivo productRoutes.js.

Estas rutas proporcionan una interfaz clara y estructurada para que los clientes de la API interactúen con los productos en la aplicación. Más adelante en este documento, se ofrecerá una explicación más detallada sobre la API RESTful creada, incluyendo los principios de diseño RESTful que se han seguido y las operaciones disponibles para cada recurso.

Utils

La carpeta Utils contiene archivos y herramientas útiles que se utilizan en la aplicación para tareas específicas. Dentro de esta carpeta se encuentra *Coordenadas.xlsx* que en un primer momento se había planteado almacenar dicho fichero en una carpeta data que contuviese datos necesarios de la aplicación. Este archivo Excel contiene los datos necesarios para representar las coordenadas seleccionadas por el usuario en la vista filtro en su homónima en el sistema de cuadrícula en el que están almacenados los datos en el servidor OPeNDAP. Desde el archivo `excelUtils.js` se accede a leer este fichero *Coordenadas.xlsx* donde se sacan los valores `nlon`, `nlat` y `nlay` :

- **nlat:** Representa el número de intervalos de cuadrícula de 0.25 grados de latitud desde 90° S hasta 90° N. En este caso, hay 720 intervalos, cubriendo toda la latitud terrestre.
- **nlon:** Indica el número de intervalos de cuadrícula de 0.25 grados de longitud desde 180° W hasta 180° E. Con 1440 intervalos, se cubre toda la longitud terrestre.
- **nlayer:** Define el número de capas de perfilado. Cada capa tiene un límite superior, que varía desde 0.5 km hasta 18.0 km sobre la superficie de la Tierra. En total, hay 28 capas.

En la Ilustración 17, mostrada a continuación se muestra la función `calcularNlon` que calcula los valores de `nlon1` y `nlon2` en función de las longitudes proporcionadas (`lon1` y `lon2`) devolviéndolo en un formato compatible para su inserción en la URL necesaria.

```
// Función para calcular nlon1 y nlon2
async function calcularNlon(lon1, lon2) {
  const rutaArchivo = './utils/Coordenadas.xlsx'; // Ruta al archivo Excel

  let nlon1, nlon2; // Variables para almacenar nlon1 y nlon2
  try {
    const workbook = new ExcelJS.Workbook();
    await workbook.xlsx.readFile(rutaArchivo);

    const worksheet = workbook.getWorksheet(1); // Obtener la primera hoja del libro de trabajo

    // Iterar sobre cada fila en la hoja de cálculo, comenzando desde la segunda fila
    worksheet.eachRow({ includeEmpty: true, firstRow: 2 }, (row, rowNumber) => {
      // Obtener los valores de las celdas en la fila como un arreglo
      const LONmax = parseFloat(row.getCell(1).text);
      const LONmin = parseFloat(row.getCell(2).text);
      const nlon = row.getCell(3).text;

      // Verificar si lon1 y lon2 están dentro del rango LONmax y LONmin
      if (lon1 >= LONmax && lon1 < LONmin) {
        nlon1 = nlon; // Asignar el valor de nlon a nlon1
      }
      if (lon2 >= LONmax && lon2 < LONmin) {
        nlon2 = nlon; // Asignar el valor de nlon a nlon2
      }
    });
  } catch (error) {
    console.error('Error al leer el archivo Excel:', error.message);
  }
  return `[${nlon1}:${nlon2}]`;
}
```

Ilustración 17: Fragmento código de `excelUtils.js`.

Por último, el archivo `opendapUtils.js` proporciona funciones para generar URLs base para la descarga y visualización de datos a través de la API OPeNDAP. Estas funciones obtienen información necesaria de los modelos de la base de datos, como productos, satélites y radiómetros, y generan las URLs necesarias para acceder a los datos meteorológicos. Estas URLs son utilizadas por el controlador `opendapController` para crear y manejar las solicitudes de datos meteorológicos a través de OPeNDAP.

2.3.1.4 Rest API

Inicialmente, se consideró utilizar alguna API existente para la obtención de datos de la misión Global Precipitation Measurement (GPM). Sin embargo, ninguna satisfacía completamente las necesidades del proyecto, por lo que se optó por la creación de una API personalizada. Esta decisión se basó en la necesidad de adaptar la API a los requerimientos específicos del proyecto, garantizando así un acceso eficiente y preciso a los datos de GPM a través de OPeNDAP como ya se ha mencionado anteriormente. La API personalizada desarrollada proporciona los servicios necesarios para la interacción entre el *frontend* y el *backend* de la aplicación, permitiendo una gestión eficaz de los datos y una experiencia de usuario óptima.

Para la realización de la API se utilizó Swagger [14], un conjunto de herramientas de código abierto para diseñar, construir, documentar y consumir servicios web RESTful. La especificación OpenAPI, en la que se basa Swagger, permite describir la estructura de la API de forma estandarizada, lo que facilita la comprensión y el uso de esta por parte de los desarrolladores.

Para llevar a cabo una documentación automatizada en node.js se siguieron los siguientes pasos:

1. Selección de herramientas: Se utilizó 'swagger-jsdoc' para generar la documentación de la API a partir de comentarios en el código y 'swagger-ui-express' para servir la documentación de forma interactiva.
2. Configuración del entorno: Se instalaron las dependencias necesarias con el comando 'npm install swagger-jsdoc swagger-ui-express'.
3. Definición de la estructura de la documentación: En el archivo app.js se definieron las opciones de Swagger y se configuró el middleware como se puede observar en la Ilustración 18.

```
const swaggerOptions = {
  swaggerDefinition: {
    openapi: '3.0.0',
    info: {
      title: 'GPMDDataExplorer',
      version: '1.0.0',
      description: 'Documentación de la API del proyecto GPMDDataExplorer para el Trabajo de Fin de Grado de Ingeniería Informática'
    },
    servers: [
      {
        url: 'http://localhost:3000',
        description: 'Servidor de desarrollo'
      }
    ]
  },
  apis: ['./routes/*.js'] // Ruta a los archivos de las rutas
};

const swaggerDocs = swaggerJsDoc(swaggerOptions);
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

Ilustración 18: Fragmento código app.js.

Al acceder a 'http://localhost:3000/api-docs', se puede visualizar la documentación generada automáticamente por Swagger UI (Ilustración 19).



Ilustración 19: Documentación Swagger [14].

4. Definición de *endpoints* en las rutas: En cada archivo de rutas, se añadieron comentarios que describen los *endpoints* utilizando la especificación OpenAPI. Como ejemplo de ello se muestra a continuación el archivo de rutas `sessionRoutes.js`.

```
/**
 * @swagger
 * tags:
 *   - name: Sessions
 *     description: Endpoints relacionados con las sesiones
 */
```

Ilustración 20: Etiquetado de Swagger.

En todos los archivos de rutas, como se muestra en el fragmento cogido como ejemplo (Ilustración 20), se definen etiquetas `tags` para categorizar los *endpoints* de la API. En este caso, todos los *endpoints* relacionados con las sesiones se agrupan bajo la etiqueta `Sessions`. Como se puede ver en la Ilustración 21 también las etiquetas: `CSV`, `Filters`, `OpenDAP`, `Products`, `Radiometers` y `Satellites`.

CSV	Endpoints relacionados con la gestión de archivos CSV	▼
Filters	Endpoints relacionados con los filtros	▼
OpenDAP	Endpoints relacionados con OpenDAP	▼
Products	Endpoints relacionados con los productos	▼
Radiometers	Endpoints relacionados con los radiómetros	▼
Satellites	Endpoints relacionados con los satélites	▼
Sessions	Endpoints relacionados con las sesiones	▼

Ilustración 21: Estructura documentación Swagger [14].

Siguiente a la etiqueta `tag` se encuentra la información de cada uno de los *endpoints*. Siguiendo con el archivo de rutas `sessionRoutes.js`, se puede observar en la ilustración siguiente el bloque que documenta el *endpoint* ‘`GET /api/sessions`’. Este se utiliza para obtener todas las sesiones. La documentación incluye una breve descripción con la etiqueta `summary`, la etiqueta `tag` que categoriza el *endpoint* y las posibles respuestas con la etiqueta `responses`. En este caso, una respuesta exitosa (‘200’) devolverá una lista de sesiones en formato JSON.

```

/**
 * @swagger
 * /api/sessions:
 *   get:
 *     summary: Obtiene todas las sesiones
 *     tags:
 *       - Sessions
 *     responses:
 *       200:
 *         description: Lista de sesiones
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 type: object
 */
router.get('/', sessionController.getAllSessions);

```

Ilustración 22: Etiquetado de Swagger endpoint 'GET /api/sessions'.

También en este archivo de rutas, se encuentra el bloque que documenta el endpoint 'POST /api/sessions' (Ilustración 23). Este se utiliza para crear una nueva sesión. La documentación describe el cuerpo de la solicitud con la etiqueta requestBody, incluyendo el tipo de contenido y el campo requerido, en este caso username. También se detallan las posibles respuestas, una respuesta exitosa ('201') que devolverá la información sobre la sesión creada y una respuesta error en caso de fallo.

```

/**
 * @swagger
 * /api/sessions:
 *   post:
 *     summary: Crea una nueva sesión
 *     tags:
 *       - Sessions
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               username:
 *                 type: string
 *                 description: Nombre de usuario para la sesión
 *     responses:
 *       201:
 *         description: Sesión creada
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 session_id:
 *                   type: string
 *                   description: ID de la sesión creada
 *                 username:
 *                   type: string
 *                   description: Nombre de usuario de la sesión
 *                 expiration:
 *                   type: string
 *                   format: date-time
 *                   description: Fecha y hora de expiración de la sesión
 *       500:
 *         description: Error al crear la sesión
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 message:
 *                   type: string
 *                   description: Mensaje de error
 */
router.post('/', sessionController.createSession);

```

Ilustración 23: Etiquetado de Swagger endpoint 'POST /api/sessions'.

En la siguiente ilustración se muestra como quedaría en Swagger la estructura de lo mencionado anteriormente.

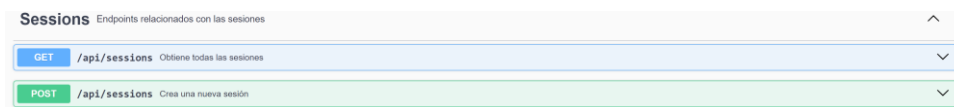


Ilustración 24: Estructura documentación de Sessions [14].

Además de Swagger, se utilizó la herramienta Postman para realizar pruebas más exhaustivas. Esta herramienta permite enviar solicitudes HTTP y recibir respuestas de manera eficiente, facilitando pruebas manuales y automatizadas.

Cuando se integra con la documentación generada por Swagger, Postman ofrece una perspectiva práctica para validar el comportamiento de las APIs en situaciones reales.

En comparación con otras herramientas similares, Postman se destaca por su interfaz intuitiva y sus características avanzadas, que son adecuadas tanto para desarrolladores experimentados como para aquellos que están comenzando. Su uso en proyectos de software no solo mejora la eficiencia del desarrollo, sino que también fortalece la calidad y estabilidad de la aplicación.

Siguiendo con el ejemplo documentado en Swagger realizaremos pruebas sobre los *endpoints* de Session.

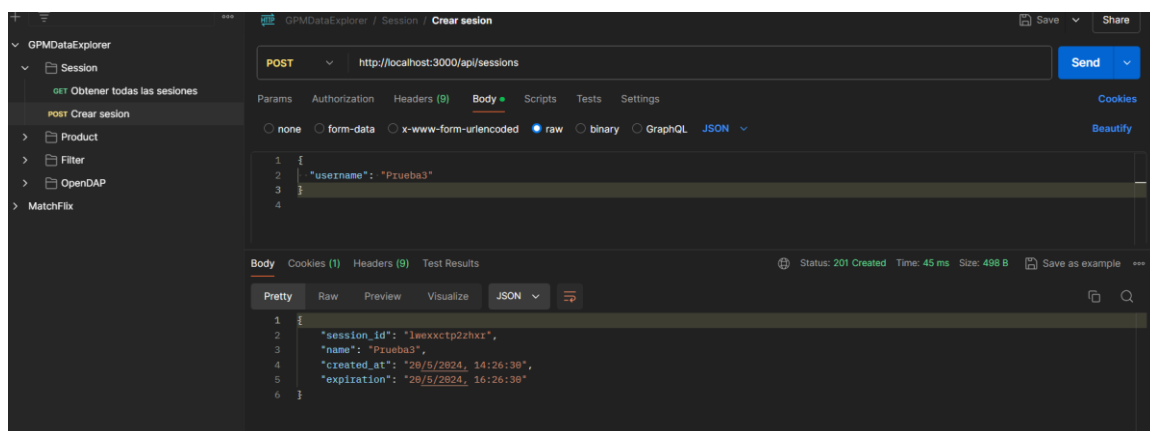


Ilustración 25: Consulta POST en Postman.

En la Ilustración 25, a la izquierda, se presenta la colección de pruebas utilizada para validar y verificar el correcto funcionamiento de la aplicación *GPMDataExplorer*. Esta colección abarca una variedad de escenarios y casos de uso, cubriendo tanto las operaciones básicas como las más complejas de la API.

Además, se muestra un ejemplo de una consulta POST realizada para crear una sesión en la aplicación. En esta solicitud, se proporcionó el nombre de usuario Prueba3. La respuesta del servidor, con un estado 201 creado, incluye los detalles de la sesión recién creada, como su ID, nombre, fecha de creación y fecha de expiración.

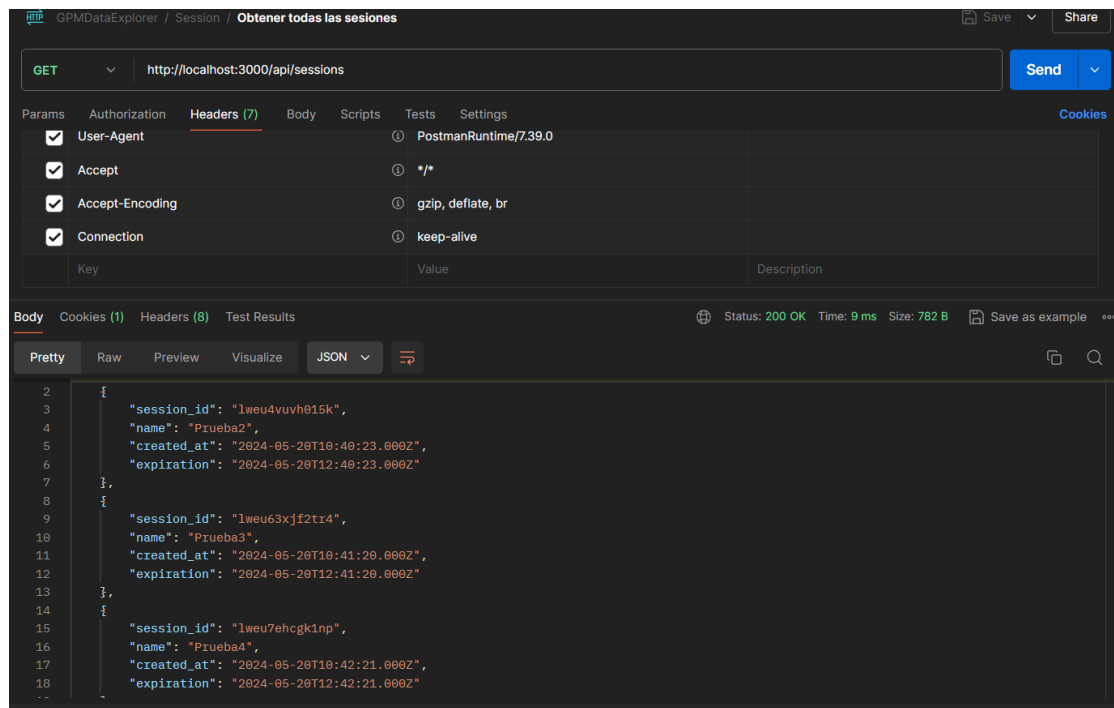


Ilustración 26: Consulta GET en Postman.

En la ilustración anterior se puede observar una consulta GET la cual devuelve una lista detallada de las sesiones disponibles en ese momento en la aplicación. Cada sesión está representada como un objeto JSON con una serie de atributos correspondientes a cada sesión.

2.3.1.5 Base de Datos

En el desarrollo de la base de datos del proyecto se utilizaron dos herramientas fundamentales para la gestión de esta: MariaDB [15] y HeidiSQL [16] .

MariaDB es un sistema de gestión de bases de datos relacionales de código abierto, compatible con MySQL. Su elección se basó en su reputación de proporcionar un alto rendimiento y escalabilidad, lo que lo convierte en una opción sólida para proyectos de diversas escalas. La compatibilidad con MySQL también resultó beneficiosa, ya que permitió aprovechar la amplia documentación y comunidad de usuarios asociada.

HeidiSQL complementó a MariaDB al ofrecer una interfaz gráfica de usuario intuitiva y fácil de usar. Esta herramienta resultó invaluable para la administración y manipulación de la base de datos del proyecto. Su compatibilidad con múltiples sistemas de gestión de bases de datos, incluyendo MariaDB, simplificó considerablemente las tareas de diseño, consulta y mantenimiento de la base de datos.

En el apartado '2.2.2 Diagrama BBDD', se presentó una visión general de la estructura de datos de nuestra aplicación, detallando las entidades principales, sus relaciones y la lógica subyacente en el diseño de la base de datos. Este apartado se centra en cómo esta estructura se implementa y configura dentro del *backend* de la aplicación.

Creación de las tablas

Lo primero que se hizo fue la creación de un *script* SQL el cual sigue la estructura de diseño realizada en la fase previa. Este script incluye la creación de las diferentes tablas con sus respectivos campos, definición de claves primarias y foráneas.

Esto permitió crear físicamente la base de datos y sus tablas en el servidor de MariaDB local. La ejecución del script garantizó que la estructura de la base de datos estuviera correctamente configurada según el diseño establecido.

Inserción de Datos

Tras la creación de las tablas, se procedió a la inserción de datos iniciales necesarios para el correcto funcionamiento de la aplicación. Por ejemplo, se introdujeron datos en las tablas de radiometer (Ilustración 27) y satellite (Ilustración 28) para registrar los distintos radiómetros y satélites disponibles, respectivamente.

```
-- Insertar datos en tabla radiometer
INSERT INTO radiometer (name)
VALUES ('AMSR2'),
       ('ATMS'),
       ('GMI'),
       ('MHS'),
       ('SSM/I'),
       ('TMI'),
       ('AMSR-E'),
       ('AMSU-B'),
       ('SSM/I');
```

Ilustración 27: Inserción de datos tabla radiometer.

```
INSERT INTO satellite (shortname, name, radiometer_id, start_date, end_date, description) VALUES
('GPM_3GPROFGCOMW1AMSR2_DAY_07','GCOMW1', 1,'2022-05-01', '2024-04-05', 'Radiometer Precipitation Profiling L3 1 day 0.25 degree x 0.25 degree V07'),
('GPM_3GPROFGCOMW1AMSR2_DAY_CLIM_07','GCOMW1', 1,'2012-07-02', '2024-01-30', ''),
('GPM_3GPROFNOAA20ATMS_DAY_07','NOAA20', 2, '2022-05-01', '2024-03-25', 'Radiometer Precipitation Profiling L3 1 day 0.25 degree x 0.25 degree V07'),
('GPM_3GPROFNOAA20ATMS_DAY_CLIM_07','NOAA20', 2, '2017-11-29', '2024-01-30', ''),
('GPM_3GPROFNOAA21ATMS_DAY_07','NOAA21', 2, '2023-02-01', '2024-03-25', 'Radiometer Precipitation Profiling L3 1 day 0.25 degree x 0.25 degree V07'),
('GPM_3GPROFNOAA21ATMS_DAY_CLIM_07','NOAA21', 2, '2023-02-01', '2024-01-30', ''),
('GPM_3GPROFNPPATMS_DAY_07','NPP', 2, '2022-05-01', '2024-03-25', 'Radiometer Precipitation Profiling L3 1 day 0.25 degree x 0.25 degree V07'),
('GPM_3GPROFNPPATMS_DAY_CLIM_07','NPP', 2, '2011-11-08', '2024-01-30', '');
```

Ilustración 28: Inserción datos tabla satellite.

Además, se introdujeron datos iniciales en las tablas *variables_select* y *layer_values*. Estas tablas contienen información sobre las variables meteorológicas disponibles y las capas verticales de altitud utilizadas en la aplicación, respectivamente. Aunque estas tablas solo se acceden mediante consultas GET y no son modificadas por la aplicación directamente, desempeñan un papel crucial al proporcionar opciones predefinidas para que los usuarios configuren sus criterios de filtrado y visualización de datos meteorológicos.

Configuración en código

Para interactuar con la base de datos MySQL desde nuestra aplicación web, fue necesario establecer una conexión adecuada y proporcionar funcionalidades para ejecutar consultas SQL de manera segura y eficiente. Para lograr esto,

creamos un módulo llamado `dbconnector.js` dentro de la carpeta `'src/config'`, que contiene toda la lógica relacionada con la conexión.

En la siguiente ilustración se muestra la estructura de configuración.

```
// dbconnector.js
const mysql = require('mysql2');

// Configuración de la conexión a la base de datos
const dbConfig = {
  host: '127.0.0.1',
  port: '3306',
  user: 'root',
  password: '54911f7d',
  database: 'gpmdataexplorer',
};

// Crear un pool de conexiones
const pool = mysql.createPool(dbConfig);

// Función para ejecutar consultas a la base de datos
function query(sql, values) {
  return new Promise((resolve, reject) => {
    pool.getConnection((err, connection) => {
      if (err) {
        reject(err);
        return;
      }

      connection.query(sql, values, (error, results) => {
        connection.release();

        if (error) {
          reject(error);
        } else {
          resolve(results);
        }
      });
    });
  });
}

module.exports = { query };
```

Ilustración 29: Configuración BBDD.

Para interactuar con la base de datos MySQL desde Node.js, se utilizó el módulo `mysql2`. Configuramos la conexión mediante un objeto `dbConfig`, que incluye detalles como la dirección del host, el puerto, el nombre de usuario, la contraseña y el nombre de la base de datos. Es necesario crear un *pool* de conexiones MySQL con esta configuración para gestionar eficientemente múltiples conexiones. Es importante destacar que esta configuración se realizó para un entorno de desarrollo local.

Integración en la Aplicación

La integración de este módulo en la aplicación se realiza mediante su importación en otros archivos donde se requiere interactuar con la base de datos. Por ejemplo, en los modelos de la aplicación, este módulo se utiliza para ejecutar consultas SQL y realizar operaciones en la base de datos de manera segura y eficiente. Los modelos actúan como una interfaz entre la lógica de la aplicación y la base de datos, encapsulando las consultas y operaciones relacionadas con los datos. Esto permite una mayor modularidad y un código más organizado, ya que todas las llamadas a la base de datos y modificaciones se realizan desde los modelos.

2.3.1.6 Seguridad

La seguridad en el desarrollo de aplicaciones web es un aspecto fundamental para proteger los datos y la integridad del sistema. En este apartado, se analizan las medidas de seguridad implementadas en el *backend* de la aplicación *GPMDDataExplorer*. Se abordan tanto las prácticas de seguridad adoptadas en el código como las estrategias utilizadas para proteger los datos y prevenir posibles vulnerabilidades.

GPMDDataExplorer es una herramienta diseñada para visualizar datos meteorológicos proporcionados por la misión Global Precipitation Measurement (GPM) de la NASA. Dado que la aplicación no maneja datos sensibles de los usuarios, como información personal, financiera o profesional, se necesitan menos medidas de seguridad en comparación con aplicaciones que sí lo hacen. Sin embargo, aun así, se implementan medidas robustas para garantizar la confidencialidad, integridad y disponibilidad de los recursos.

Una de las decisiones de seguridad clave en *GPMDDataExplorer* es el manejo de sesiones en lugar de una autenticación donde el usuario tenga que registrarse e iniciar sesión con su cuenta. Dado que la aplicación no requiere información personal de los usuarios y su función principal es proporcionar acceso a datos meteorológicos, el uso de sesiones simplifica el proceso de interacción. Los usuarios pueden comenzar a explorar los datos meteorológicos de inmediato sin la necesidad de registrarse y crear una cuenta.

Este enfoque hacia la seguridad no solo simplifica la experiencia del usuario, sino que también ayuda a reducir los riesgos asociados con el manejo de datos sensibles.

Algunas de las prácticas y estrategias específicas de seguridad adoptadas son:

- **Middleware cors:** Para garantizar una interacción segura entre el *frontend* y el *backend*, que se ejecutan en puertos diferentes, la aplicación web desarrollada implementa políticas de CORS (Cross-Origin Resource Sharing) [17]. Esta práctica de seguridad restringe los recursos a ser solicitados solo desde dominios autorizados, evitando así ataques como Cross-Site Scripting (XSS) [18].
- **Middleware cookie-parser:** Este middleware analiza las cookies adjuntas a las solicitudes HTTP entrantes. Es importante destacar, como se muestra en la siguiente ilustración, que se establece la opción *httpOnly* en *true* al establecer la cookie de sesión, lo que significa que la cookie solo se puede acceder a través del protocolo HTTP y no está disponible para scripts del lado del cliente, lo que ayuda a prevenir ataques XSS (Cross-Site Scripting).

```
// Establecer una cookie de sesión en la respuesta
res.cookie('sessionId', newSession.session_id, { maxAge: 120 * 60 * 1000, httpOnly: true, secure: true });
```

Ilustración 30: Cookie de sesión.

- **Limpieza de sesiones expiradas:** Para mantener la integridad del sistema, se ha desarrollado un middleware (`cleanExpiredSessions`) encargado de eliminar las sesiones expiradas de la base de datos. Esta medida asegura que la base de datos no se sature con sesiones obsoletas, manteniendo así un óptimo rendimiento y funcionamiento de la aplicación.
- **Uso de parámetros de consulta seguros:** Al ejecutar consultas SQL, se utilizan parámetros de consulta para evitar la inserción directa de datos proporcionados por el usuario en la consulta SQL. Esto ayuda a prevenir ataques de inyección SQL.

```
const insertQuery = 'INSERT INTO session (session_id, name, created_at, expiration) VALUES (?, ?, NOW(), ?)';
```

En este ejemplo sacado del modelo de `session`, la sentencia SQL utiliza parámetros de consulta seguros, donde los valores a ser insertados en la tabla no se concatenan directamente en la cadena SQL. En lugar de eso, se utilizan marcadores de posición "?" para representar los valores que serán proporcionados más adelante. Esto asegura que los datos proporcionados por el usuario no puedan alterar la estructura de la consulta SQL.

Su homóloga vulnerable a un ataque de inyección SQL:

```
const insertQuery = `INSERT INTO session (session_id, name, created_at, expiration) VALUES ('${sessionId}', '${name}', NOW(), '${expiration}')`;
```

Los valores proporcionados por el usuario (`sessionId`, `name`, `expiration`) se concatenan directamente en la cadena SQL. Esto abre la puerta a posibles ataques de inyección SQL si los valores proporcionados contienen caracteres especiales que puedan alterar la lógica de la consulta, como comillas simples (`'`), lo que podría permitir a un atacante manipular la consulta y comprometer la seguridad de la base de datos.

- **Manejo de errores:** Se implementa un manejo adecuado de errores en los controladores y modelos para capturar y manejar cualquier error que pueda surgir durante el procesamiento de las solicitudes. Esto evita la divulgación de información sensible sobre el funcionamiento interno de la aplicación en caso de errores no controlados. Se ha utilizado la estructura *try-catch* para capturar cualquier error que pueda ocurrir durante la ejecución del bloque *try*. Si se produce un error, es el bloque *catch* el que lo maneja. En el bloque *catch*, se registra el error en la consola y se envía una respuesta HTTP con un estado 500 (Internal Server Error). Esto hace que la aplicación maneje errores de manera controlada y no revele detalles internos que podrían ser aprovechados.

2.3.2 Frontend

El *frontend* constituye la parte visible de la aplicación *GPMDDataExplorer* y es el punto de interacción principal para los usuarios. A través del *frontend*, los usuarios pueden visualizar y explorar los datos meteorológicos proporcionados por la misión Global Precipitation Measurement (GPM) de la NASA.

Cuando un usuario interactúa con la interfaz de usuario (UI) del *frontend*, se envían solicitudes al *backend* para recuperar datos y realizar operaciones específicas. El *frontend* se encarga de recibir estas respuestas y actualizar la UI en consecuencia, proporcionando una experiencia dinámica y reactiva. Este proceso implica el uso de diversas tecnologías y herramientas que garantizan un rendimiento óptimo y una alta usabilidad.

En los siguientes apartados, se detallará la arquitectura del *frontend*, las tecnologías empleadas para su desarrollo, el diseño de la interfaz de usuario y la forma en que se integra con el *backend*. Esta exploración permitirá comprender cómo se ha construido el *frontend* para cumplir con los objetivos de la aplicación, ofreciendo una plataforma robusta y eficiente para la obtención y visualización de datos meteorológicos.

2.3.2.1 Tecnologías Utilizadas

A continuación, se mostrarán las herramientas, lenguajes y tecnologías utilizadas para el desarrollo del *frontend* de este proyecto.

Angular

El *frontend* de *GPMDDataExplorer* está desarrollado utilizando Angular [19], un framework de desarrollo web basado en TypeScript. Angular proporciona una estructura robusta y modular para construir aplicaciones web y facilita la creación de componentes reutilizables, el enrutamiento, la gestión del estado de la aplicación y la comunicación con el *backend*. Algunas de las principales ventajas de este framework son:

- Modularidad: La estructura modular facilita la adición de nuevas funcionalidades y el mantenimiento del código existente. Cada módulo puede desarrollarse, probarse y depurarse de manera independiente.
- Reutilización de Código: Los componentes y servicios bien definidos pueden reutilizarse en diferentes partes de la aplicación, reduciendo la duplicación de código y mejorando la eficiencia del desarrollo.
- Escalabilidad: La separación clara de preocupaciones permite escalar la aplicación más fácilmente, ya que cada parte de la aplicación puede crecer independientemente sin afectar a otras.

- **Mantenibilidad:** La arquitectura organizada y clara facilita la comprensión del código por parte de otros desarrolladores, mejorando la mantenibilidad a largo plazo.

Además de estas ventajas, ya se tenían conocimientos previos en Angular, lo que facilitó su elección. Aunque estos conocimientos no eran completos, el proyecto ha servido como una excelente oportunidad para seguir ganando experiencia y profundizar en el uso de este framework.

TypeScript

TypeScript es un lenguaje de programación basado en JavaScript que agrega características de tipado estático y otros elementos de programación orientada a objetos. Angular utiliza TypeScript como su lenguaje de programación principal.

HTML y CSS

HTML (HyperText Markup Language) se utiliza para estructurar el contenido de las páginas web, mientras que CSS (Cascading Style Sheets) se utiliza para definir el aspecto visual y el diseño de la interfaz de usuario. Estas tecnologías son fundamentales para crear una experiencia de usuario atractiva y coherente, y se utilizan en conjunto con Angular.

Leaflet

Leaflet [20] es una biblioteca de JavaScript de código abierto ampliamente utilizada para la creación de mapas interactivos en aplicaciones web. En el *frontend* de *GPMDDataExplorer*, Leaflet se utiliza para proporcionar a los usuarios una forma más visual e intuitiva de seleccionar la región sobre la cual desean obtener datos meteorológicos. Esta biblioteca ofrece una amplia gama de funcionalidades para personalizar y controlar la visualización de mapas, como la capacidad de agregar marcadores, superponer capas, y proporcionar controles de navegación fluidos. Además, Leaflet es altamente modular y extensible, lo que permite a los desarrolladores integrar fácilmente complementos y personalizar la funcionalidad según las necesidades específicas del proyecto.

La biblioteca Leaflet se utiliza en el componente Map. En este componente, Leaflet se importa y se utiliza para inicializar y controlar el mapa interactivo. Además, se utiliza para crear rectángulos interactivos mediante el complemento Leaflet.draw, permitiendo a los usuarios dibujar y definir el área de coordenadas para aplicar en el filtro.

Estas son solo algunas de las tecnologías utilizadas en el desarrollo del *frontend* de *GPMDDataExplorer*. La combinación de estas tecnologías proporciona una plataforma robusta y eficiente para la obtención, visualización y exploración de datos meteorológicos, garantizando una experiencia de usuario óptima y satisfactoria.

2.3.2.2 Arquitectura Frontend

La arquitectura del *frontend* de *GPMDDataExplorer* sigue una estructura típica de Angular, un popular framework de desarrollo web conocido por su robustez y flexibilidad. Este enfoque modular facilita el desarrollo, mantenimiento y escalabilidad de la aplicación.

En la estructura de carpetas del *frontend* se pueden encontrar algunos archivos esenciales que desempeñan roles específicos en la configuración y el funcionamiento del proyecto:

- Archivos de configuración: Angular utiliza varios archivos de configuración, como `angular.json` para la configuración del proyecto, `tsconfig.json` para la configuración de TypeScript y `package.json` para la gestión de dependencias y scripts de NPM. Estos archivos permiten personalizar el comportamiento del proyecto y asegurar que todas las dependencias necesarias estén disponibles.
- Directorio `src`: Este es el directorio principal donde reside el código fuente de la aplicación. Dentro de `src`, además de la carpeta `app`, encontramos otros directorios y archivos importantes, como `assets` para recursos estáticos (imágenes, estilos, etc.), `environments` para configuraciones específicas de diferentes entornos (desarrollo, producción, etc.).

Dentro de la carpeta `app`, la estructura se divide en varios archivos y directorios clave que organizan la funcionalidad y lógica de la aplicación:

- `app.module.ts`: Este archivo es el módulo raíz de la aplicación y define los componentes, directivas, servicios y otros módulos que forman parte de la aplicación. Es aquí donde se realiza la configuración principal de Angular, como la importación de otros módulos necesarios y la declaración de los componentes principales (Ilustración 31).

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { CookieService } from 'ngx-cookie-service';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './components/home/home.component';
import { RadiometerselectionComponent } from './components/radiometerselection/radiometerselection.component';
import { FilterComponent } from './components/filter/filter.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    RadiometerselectionComponent,
    FilterComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    AppRoutingModule
  ],
  providers: [CookieService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Ilustración 31: Código de `app.module.ts`.

- `app-routing.module.ts`: Este archivo se encarga de definir las rutas de la aplicación. Angular utiliza un sistema de enrutamiento para navegar entre diferentes vistas o componentes (Ilustración 32).

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from '../components/home/home.component';
import { RadiometerselectionComponent } from '../components/radiometerselection/radiometerselection.component';
import { SatelliteselectionComponent } from '../components/satelliteselection/satelliteselection.component';
import { FilterComponent } from '../components/filter/filter.component';
import { MapComponent } from '../components/map/map.component';
import { DataDownloadComponent } from '../components/data-download/data-download.component';

const routes: Routes = [
  { path: '', component: HomeComponent }, // Esta es la ruta raíz que apunta al componente Home
  { path: 'radiometerselection', component: RadiometerselectionComponent },
  { path: 'satelliteselection/:radiometerId', component: SatelliteselectionComponent },
  { path: 'filter/:productId', component: FilterComponent },
  { path: 'map/:productId', component: MapComponent },
  { path: 'data-download/:id_filter', component: DataDownloadComponent } // Agrega la ruta para el componente DataDownloadComponent
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Ilustración 32: Código de *app-routing.module.ts*.

A continuación, se describirá el contenido de las carpetas components, models y services, cada una desempeña un papel crucial en la estructura y funcionalidad de la aplicación.

Components

Esta carpeta contiene todos los componentes de la interfaz de usuario. Cada componente se encarga de una parte específica de la UI, facilitando la reutilización y el mantenimiento del código. Los componentes en Angular encapsulan tanto la lógica como la vista, permitiendo una separación clara de responsabilidades. Esta separación se realiza con una combinación de una plantilla HTML, estilos CSS y lógica TypeScript por cada componente. Un componente en Angular se puede crear fácilmente utilizando el CLI de Angular, para ello se debe escribir en la terminal el comando:

ng generate component nombre

Esto generará los archivos necesarios para el nuevo componente, incluyendo la plantilla HTML, los estilos CSS y el código TypeScript con su respectiva clase de Tests Unitarios. También declarará el componente en el módulo correspondiente.

Los componentes creados en la aplicación son:

- data-download: Gestiona las URLs de OpenDAP a través de las cuáles los usuarios pueden obtener los datos meteorológicos.
- filter: Permite a los usuarios aplicar diferentes filtros para los datos solicitados.
- home: Proporciona la interfaz principal de inicio para los usuarios.
- map: Encargado de la visualización de mapas interactivos desarrollados con leaflet.
- satelliteselection: Facilita la selección del satélite del que se quieren los datos.
- radiometerselection: Permite la selección del radiómetro del que se quieren los datos.

A continuación, se muestra el código del componente radiometerselection como ejemplo ilustrativo de cómo se organiza y funcionan los componentes.

```

import { Component, OnInit } from '@angular/core';
import { SessionService } from 'src/app/services/session.service';
import { SelectionService } from 'src/app/services/selection.service';
import { Radiometer } from 'src/app/models/radiometer.model';
import { Router } from '@angular/router';

@Component({
  selector: 'app-radiometerselection',
  templateUrl: './radiometerselection.component.html',
  styleUrls: ['./radiometerselection.component.css']
})
export class RadiometerselectionComponent implements OnInit {
  sessionId: string | undefined;
  radiometers: Radiometer[] = [];

  constructor(private router: Router, private sessionService: SessionService, private selectionService: SelectionService) {}

  ngOnInit(): void {
    // Aquí recuperamos el session_id al inicializar el componente
    this.sessionId = this.sessionService.getSessionId();
    console.log('Session id: ' + this.sessionId);
    //Carga radiometers disponibles
    this.loadRadiometers();
  }

  loadRadiometers(): void {
    this.selectionService.getAllRadiometers().subscribe(
      radiometers => {
        this.radiometers = radiometers;
      },
      error => {
        console.error('Error fetching radiometers:', error);
        // Aquí podrías mostrar un mensaje de error al usuario si lo deseas
      }
    );
  }

  selectRadiometer(radiometer: Radiometer): void {
    this.selectionService.setRadiometerName(radiometer.name);
    console.log(radiometer.name);
    // Navegar a la vista SatelliteSelection con el ID del radiómetro como parámetro
    this.router.navigate(['/satelliteselection', radiometer.radiometer_id]);
  }
}

```

Ilustración 33: Fragmento de código de radiometerselection.ts.

La estructura general de la plantilla TypeScript de los componentes creados sigue un patrón similar, que incluye:

- **Importaciones necesarias:** Los componentes suelen importar módulos, servicios y modelos necesarios para su funcionamiento. Por ejemplo, en la Ilustración 33, se importan servicios como SessionService y SelectionService, y modelos como Radiometer para trabajar con datos.
- **Decorador @Component:** Este decorador se utiliza para indicar que la clase es un componente de Angular. En él, se especifican metadatos como el selector (selector), la plantilla HTML (templateUrl) y el contenido HTML en línea (template), y los estilos CSS (styleUrls) y estilos CSS en línea (styles).
- **Clase del componente:** La clase del componente contiene la lógica del componente. Puede implementar interfaces como OnInit para inicializar el componente y manejar ciertos eventos del ciclo de vida del componente. En esta clase se definen propiedades y métodos que son utilizados por la plantilla HTML.
- **Constructor:** El constructor de la clase del componente se utiliza para inyectar los servicios necesarios y otras dependencias. Estas dependencias pueden ser utilizadas en los métodos de la clase para realizar operaciones específicas.

- **Métodos del componente:** En la clase del componente se definen métodos que realizan operaciones específicas, como cargar datos, manejar eventos del usuario, y realizar acciones relacionadas con la lógica de negocio. Por ejemplo, el método `ngOnInit()` se ejecuta al inicializar el componente y se utiliza para realizar inicializaciones necesarias, como cargar datos iniciales.

La plantilla HTML define la estructura y el contenido visual del componente. Puede contener enlaces a propiedades y métodos definidos en la clase del componente utilizando la sintaxis de enlace de datos de Angular, como `{{ }}` para interpolación de cadenas y `()` para enlace de eventos.

Los estilos CSS definen el aspecto visual del componente. Pueden ser estilos aplicados a elementos específicos dentro de la plantilla HTML utilizando selectores de clase, ID o elemento, o estilos globales que afectan a todo el componente.

En conjunto, estos archivos muestran cómo se organiza un componente en Angular, combinando lógica, estructura y estilos para crear una funcionalidad completa y modular dentro de la aplicación (Ilustración 34).

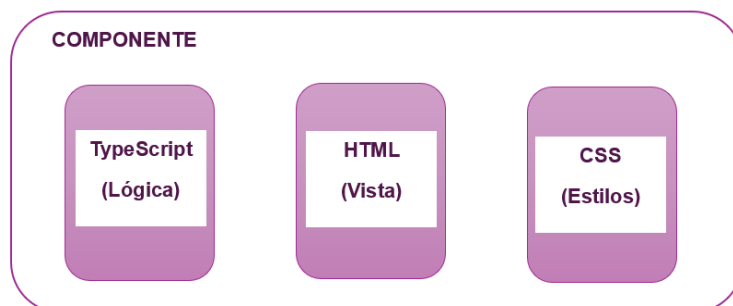


Ilustración 34: Estructura de un componente Angular.

Models

En la carpeta `models` se encuentran las definiciones de las interfaces de datos utilizadas en la aplicación. Estas interfaces representan las estructuras de datos que maneja la aplicación, proporcionando una manera clara y coherente de organizar y manipular la información. Al definir interfaces claras y bien estructuradas, se asegura la coherencia y el manejo adecuado de los datos. Estas interfaces corresponden directamente con las entidades de la base de datos, asegurando que los datos se gestionen de manera uniforme tanto en el *frontend* como en el *backend*.

Para generar una interfaz utilizando Angular CLI, se puede utilizar el siguiente comando en la terminal:

```
ng generate interface nombre-de-la-interfaz
```

En la Ilustración 35 se muestra los modelos creados en el desarrollo de la aplicación.

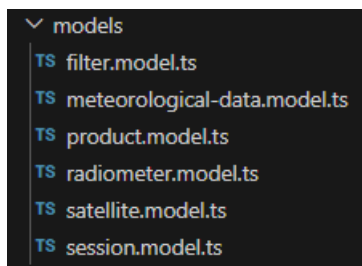


Ilustración 35: Contenido carpeta *models*.

A continuación, se presenta un ejemplo ilustrativo de una de estas interfaces para proporcionar una mejor comprensión de cómo están estructuradas y funcionan.

```
export interface Radiometer {  
  radiometer_id: number;  
  name: string;  
}
```

Ilustración 36: Contenido *radiometerModel.ts*.

La interfaz *Radiometer*, como se muestra en la ilustración anterior, se define en TypeScript y representa los radiómetros que pueden ser seleccionados en la aplicación. Esta interfaz incluye dos propiedades *radiometer_id* y *name*, que corresponden a las columnas de la tabla de radiómetros en la base de datos. Esta interfaz asegura que los datos de los radiómetros sean manejados de manera uniforme y coherente a lo largo de la aplicación.

Services

Los servicios en Angular son clases que se encargan de la lógica de negocio y la comunicación con el *backend*. En la carpeta *services*, se implementan las funciones que realizan llamadas a la API REST del *backend* para recuperar las URLs de OPeNDAP y otros datos necesarios. Los servicios permiten compartir datos y funcionalidades entre diferentes componentes de manera eficiente.

Para generar un servicio utilizando Angular CLI, se puede utilizar el siguiente comando en la terminal:

ng generate service nombre

Los services creados en la aplicación son los mostrados a continuación en la Ilustración 37.

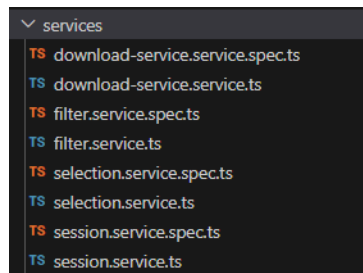


Ilustración 37: Contenido carpeta services.

La estructura general de los servicios creados sigue un patrón similar, que incluye:

- Importaciones necesarias: Los servicios suelen importar módulos y clases necesarias para su funcionamiento, como `HttpClient` para realizar solicitudes HTTP y clases de modelos para trabajar con datos.
- Decorador `@Injectable`: Este decorador se utiliza para indicar que la clase es un servicio y puede ser inyectada en otros componentes o servicios.
- Constructor: Se define un constructor que recibe como parámetro las dependencias necesarias para el servicio, como `HttpClient` para realizar solicitudes HTTP.
- Métodos del servicio: Se definen métodos que realizan operaciones específicas, como obtener datos del *backend*, manejar la lógica de negocio, o almacenar datos en el almacenamiento local del navegador.

Como ejemplo ilustrativo de un servicio se utilizará `SelectionService` (Ilustración 38), que se encarga de gestionar la selección de radiómetros y satélites, y de crear los productos necesarios para luego con los filtros seleccionados por el usuario, obtener los datos meteorológicos.

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Radiometer } from '../models/radiometer.model';
import { Satellite } from '../models/satellite.model';
import { ProductModel } from '../models/product.model';
@Injectable({
  providedIn: 'root'
})
export class SelectionService {
  private apiUrl = 'http://localhost:3000/api';

  private RADIOMETER_NAME_KEY = 'radiometerName';
  private SATELLITE_NAME_KEY = 'satelliteName';

  constructor(private http: HttpClient) {}
  setRadiometerName(name: string): void {
    localStorage.setItem(this.RADIOMETER_NAME_KEY, name);
  }

  getRadiometerName(): string | null {
    return localStorage.getItem(this.RADIOMETER_NAME_KEY);
  }

  setSatelliteName(name: string): void {
    localStorage.setItem(this.SATELLITE_NAME_KEY, name);
  }

  getSatelliteName(): string | null {
    return localStorage.getItem(this.SATELLITE_NAME_KEY);
  }

  //Obtiene un listado de todos los radiómetros
  getAllRadiometers(): Observable<Radiometer[]> {
    return this.http.get<Radiometer[]>(`${this.apiUrl}/radiometers`);
  }

  //Obtiene un listado de todos los satelites de un radiometro
  getAllSatellitesByRadiometerId(radiometerId: number): Observable<Satellite[]> {
    return this.http.get<Satellite[]>(`${this.apiUrl}/satellites/radiometer/${radiometerId}`);
  }

  //Obtiene un radiometro por ID
  getRadiometerById(radiometerId: number): Observable<Radiometer> {
    return this.http.get<Radiometer>(`${this.apiUrl}/radiometers/${radiometerId}`);
  }
}

```

Ilustración 38: Código selectionService.

Como ya se ha indicado este servicio proporciona métodos para interactuar con el *backend* y realizar operaciones relacionadas con la selección de radiómetros y satélites, así como la creación de productos para obtener datos meteorológicos.

2.3.2.3 Interfaz de Usuario

En esta sección, se describirán las distintas pantallas que componen la interfaz de usuario de *GPMDDataExplorer*. Se proporcionará una visión detallada de cada pantalla, destacando las funcionalidades clave y los elementos de diseño que facilitan la interacción del usuario con la aplicación. Además, se incluirán capturas de pantalla para ilustrar visualmente cómo se presentan los datos y las opciones de navegación.

En la Ilustración 39 se muestra el logotipo de *GPMDDataExplorer*, el cual representa de manera simbólica la funcionalidad y la estética de la aplicación.



Ilustración 39: Logo de la aplicación.

Pantalla de Inicio

En la Ilustración 40 se muestra la primera interfaz que el usuario visualiza al acceder a *GPMDDataExplorer*. Esta pantalla tiene un diseño sencillo y centrado, con un recuadro donde el usuario debe introducir su nombre para poder acceder a la aplicación.

En el centro de la pantalla, se presenta un campo de texto donde se solicita al usuario que ingrese su nombre. Además, se incluye un botón de "Start" que, al ser presionado, verifica si el campo de nombre ha sido completado.

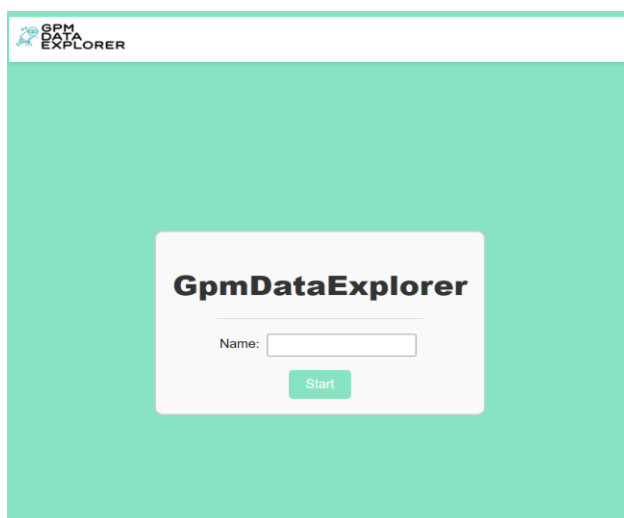
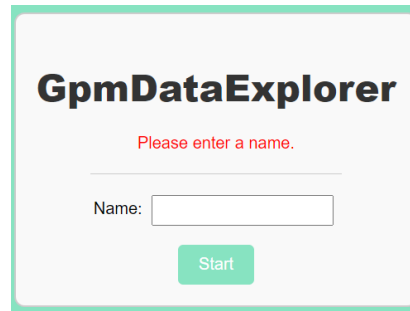


Ilustración 40: Pantalla de inicio.

Es obligatorio que el usuario introduzca un nombre para acceder a la aplicación. Si el usuario intenta proceder sin introducir un nombre, la aplicación mostrará un mensaje de error indicando que debe completar este campo para continuar. Este mensaje de error se presenta en color rojo y se ubica justo encima del campo de texto, asegurando que sea visible y claro para el usuario (Ilustración 41).



GpmDataExplorer

Please enter a name.

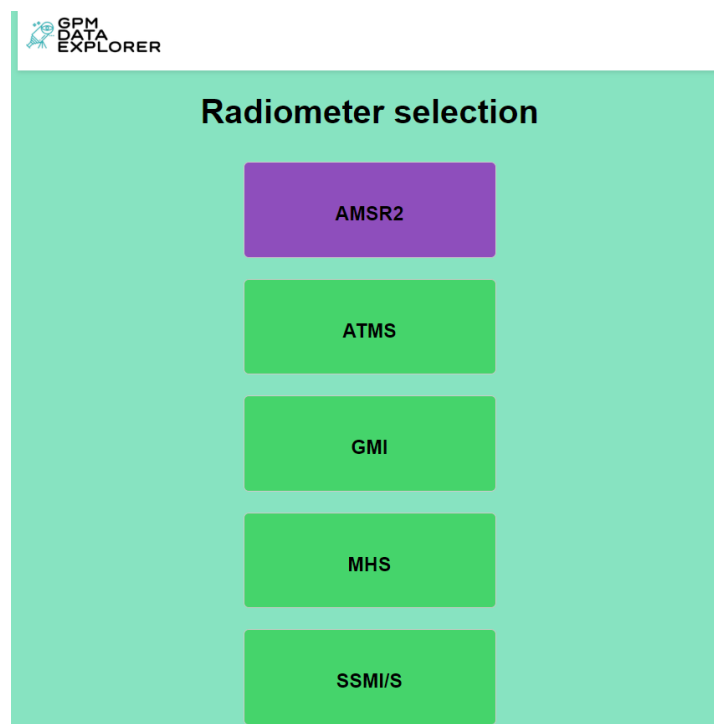
Name:

Start

Ilustración 41: Mensaje de error en la pantalla inicio.

Pantalla de Selección de Radiómetro

Después de la pantalla de inicio, los usuarios son dirigidos a la pantalla de selección de radiómetro (Ilustración 42). Aquí, tienen la oportunidad de elegir uno de los radiómetros disponibles que se almacenan en la base de datos. Esta selección es crucial ya que determina desde qué instrumento se obtendrán los datos meteorológicos para su visualización y análisis.



GPM DATA EXPLORER

Radiometer selection

- AMSR2
- ATMS
- GMI
- MHS
- SSM/I/S

Ilustración 42: Pantalla de selección de radiómetro.

En esta interfaz, se presenta una lista de radiómetros disponibles, cada uno representado como un cuadro que muestra el nombre del radiómetro. Cuando el usuario pasa el ratón sobre un radiómetro, el cuadro cambia de color para indicar visualmente la interacción. Esta funcionalidad mejora la experiencia del usuario al proporcionar una retroalimentación visual clara.

Pantalla de Selección de Satélite Según Radiómetro

Después de que el usuario selecciona un radiómetro en la pantalla anterior, se le presenta la pantalla de selección de satélite. Esta pantalla permite al usuario seleccionar el satélite específico asociado al radiómetro del cual desean obtener los datos meteorológicos. Cada radiómetro puede tener varios satélites asociados, y estos satélites pueden ofrecer diferentes tipos de datos.

En la siguiente ilustración se muestra algunos satélites asociados al radiómetro ATMS. Dependiendo del radiómetro seleccionado, los satélites disponibles variarán. La interfaz también incluye una sección a la derecha que proporciona información descriptiva sobre el conjunto de datos que se obtendrán, ayudando al usuario a tomar una decisión informada.

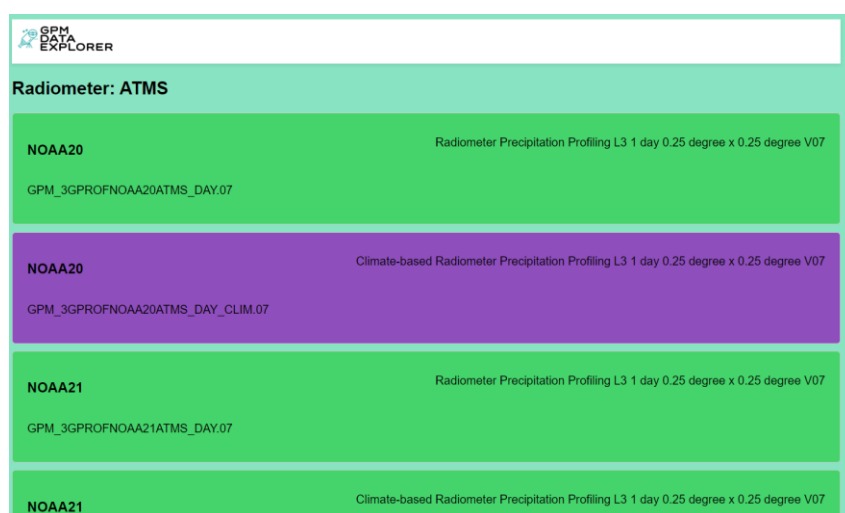


Ilustración 43: Pantalla de selección de satélite.

En la Ilustración 43, se puede observar que aparecen varios satélites aparentemente iguales entre los que elegir. Sin embargo, difieren en el tipo de datos que ofrecen. Por ejemplo, ambos conjuntos de datos tienen una resolución diaria, pero "DAY.07" se centra en promedios o agregados a corto plazo, mientras que "DAY_CLIM.07" se enfoca en promedios climatológicos a largo plazo.

Pantalla de Filtros

En esta pantalla (Ilustración 44), el usuario podrá seleccionar los filtros específicos sobre el conjunto de datos que desea analizar. Utilizando el radiómetro ATMS y el satélite NOAA20 como ejemplo, los filtros disponibles son:

- Rango de fechas
- Rango de región
- Variables de las que se quiere obtener datos
- Altitudes de las que se quiere obtener datos

Cada uno de estos filtros permite al usuario afinar su búsqueda para obtener datos más relevantes y precisos. A continuación, se describirán y mostrarán cada uno de estos filtros

Ilustración 44: Pantalla inicial de filtros.

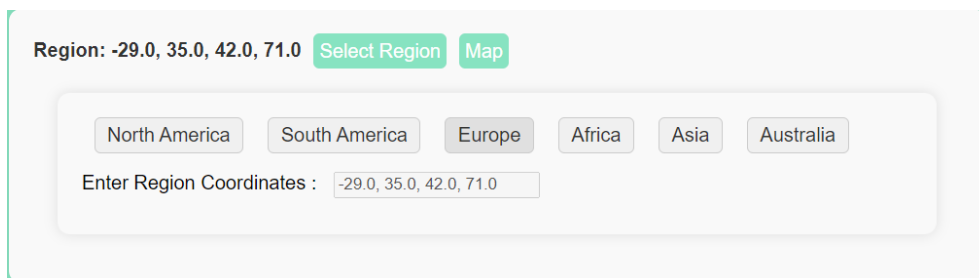
El primer filtro es el rango de fechas (Ilustración 45), donde se muestra el período en el cual el satélite ha estado activo. El usuario puede seleccionar las fechas deseadas dentro de este rango. La interfaz proporciona un calendario desplegable que facilita la selección de las fechas, haciendo el proceso más cómodo y visualmente intuitivo.

Ilustración 45: Filtro de rango de fechas.

El siguiente filtro es el Rango de Región, el cual está restringido a las coordenadas -180,-70,180,70, ya que la misión GPM de la NASA solo ofrece datos dentro de ese rango. El usuario tiene dos opciones para seleccionar la región de interés:

1. Selección de Región Predefinida: El usuario puede seleccionar una región predefinida, como continentes, utilizando el botón "Select Región".
2. Selección Manual en Mapa: El usuario puede seleccionar manualmente la región utilizando el botón "Map". Esto abrirá una pantalla interactiva donde el usuario puede dibujar un rectángulo sobre la región de interés dentro de las coordenadas permitidas.

Si el usuario elige seleccionar una región predefinida se mostrará una lista de opciones, como continentes, y al seleccionar una de ellas, la aplicación ajustará automáticamente las coordenadas para esa región. En la Ilustración 46 se muestra como se ha seleccionado la región Europa.



The screenshot shows a user interface for selecting a region. At the top, it displays 'Region: -29.0, 35.0, 42.0, 71.0' followed by two buttons: 'Select Region' and 'Map'. Below this is a container with six buttons representing continents: 'North America', 'South America', 'Europe', 'Africa', 'Asia', and 'Australia'. The 'Europe' button is highlighted with a grey background. Below the buttons is a text input field labeled 'Enter Region Coordinates :' containing the coordinates '-29.0, 35.0, 42.0, 71.0'.

Ilustración 46: Filtro de región predefinida.

Si el usuario opta por seleccionar manualmente la región, se abrirá una pantalla interactiva de mapa. En esta pantalla, se utiliza la librería Leaflet para proporcionar un mapa interactivo donde el usuario puede dibujar un rectángulo sobre la región de interés.

En la ilustración adjunta se muestra un ejemplo de cómo se vería esta pantalla cuando el usuario selecciona manualmente la región de España. El usuario puede ajustar el rectángulo según sus preferencias, asegurándose de que esté dentro de las coordenadas permitidas.

Una vez que el usuario finaliza la selección, puede guardar las coordenadas y volver a la pantalla de filtro.

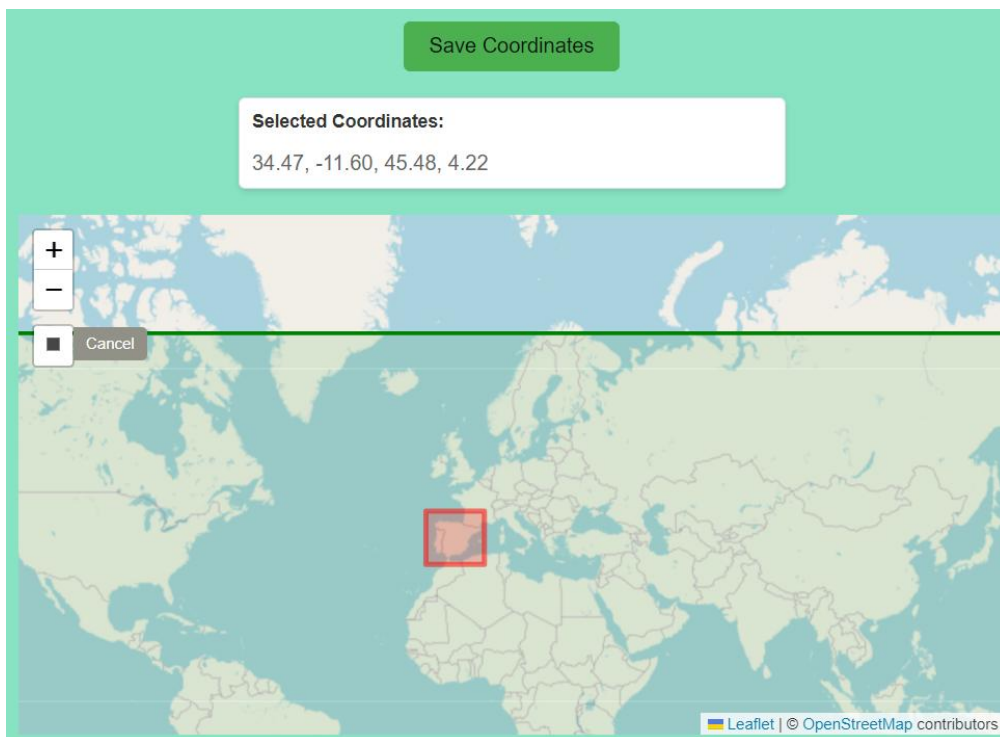


Ilustración 47: Filtro de región manual.

Para facilitar la selección de variables, se proporciona una lista de opciones con una breve descripción de cada variable como se puede observar en la siguiente ilustración. Cuando el usuario pasa el ratón sobre cada variable, se muestra una descripción emergente que proporciona información básica sobre la variable.

Además, para aquellos usuarios que necesiten información más detallada sobre cada variable, se ha incorporado la funcionalidad de descargar un archivo PDF desde la página oficial de la NASA. Este archivo PDF contiene información más detallada sobre las variables meteorológicas disponibles (Download Variable Information PDF).

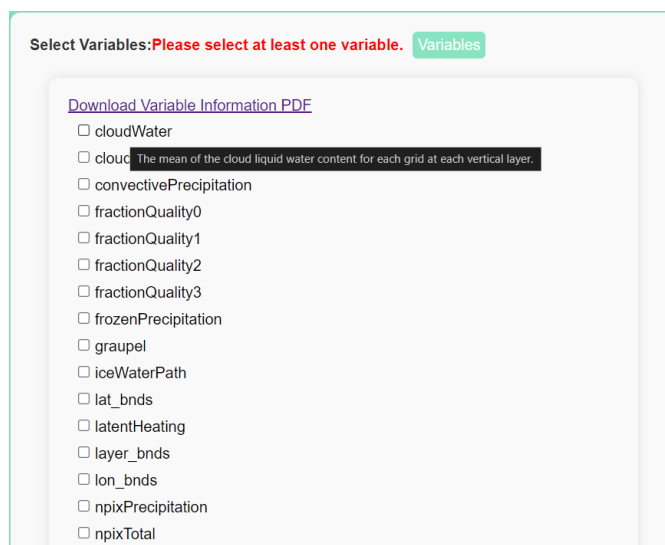


Ilustración 48: Filtro selección de variables.

Por último, el usuario puede especificar las altitudes a las que desea obtener los datos (Ilustración 49). Esto es esencial para estudios que requieren datos a diferentes niveles de la atmósfera. En este apartado aparece un aviso en rojo indicando que las capas seleccionadas deben ser consecutivas.



Ilustración 49: Filtro selección de layers.

Es importante destacar que todos los campos de filtro deben ser completados correctamente para proceder a la siguiente pantalla. Si alguno de los campos no se selecciona, la aplicación mostrará un mensaje de error y no permitirá avanzar, como se muestra en la siguiente ilustración. Esto asegura que el conjunto de datos obtenido sea completo y adecuado para el análisis.

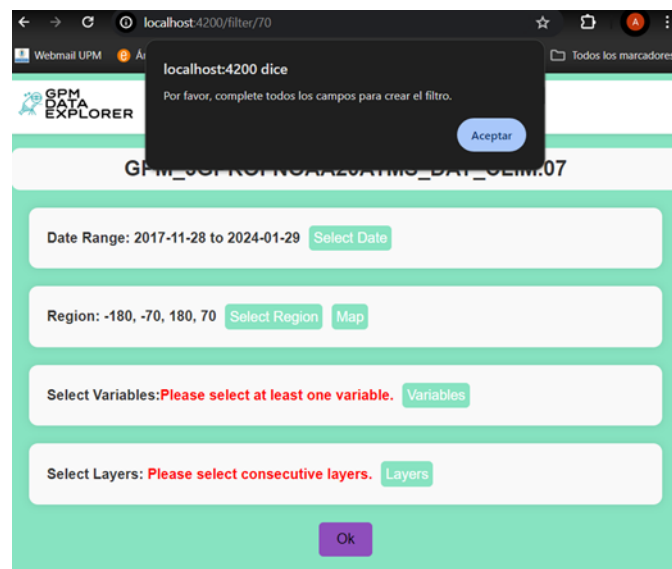


Ilustración 50: Mensaje de error en filtros.

En resumen, la pantalla de filtros ofrece una interfaz intuitiva y detallada para que el usuario pueda definir exactamente los datos que necesita. La

combinación de un calendario desplegable, selección de región, variables y altitudes proporciona una flexibilidad y precisión que son fundamentales para un análisis meteorológico detallado.

Pantalla de URLs OPeNDAP

En esta pantalla, el usuario puede acceder a las URLs necesarias para obtener los datos del servidor OPeNDAP. Estas URLs están generadas en función de los filtros seleccionados previamente en la pantalla de filtro.

Es importante tener en cuenta que para acceder a estos datos del servidor OPeNDAP, es necesario estar registrado en el sitio web oficial de la NASA. Por lo tanto, se ha incluido un enlace directo a la página de registro en esta pantalla para facilitar el proceso de registro a los usuarios que aún no lo han realizado.

Como se puede observar en la Ilustración 51, también se muestra la opción de “View table”, con lo que se generará una tabla con los datos que el usuario solicita.

Así pues, se pone a disposición del usuario dos formas de visualización de datos.

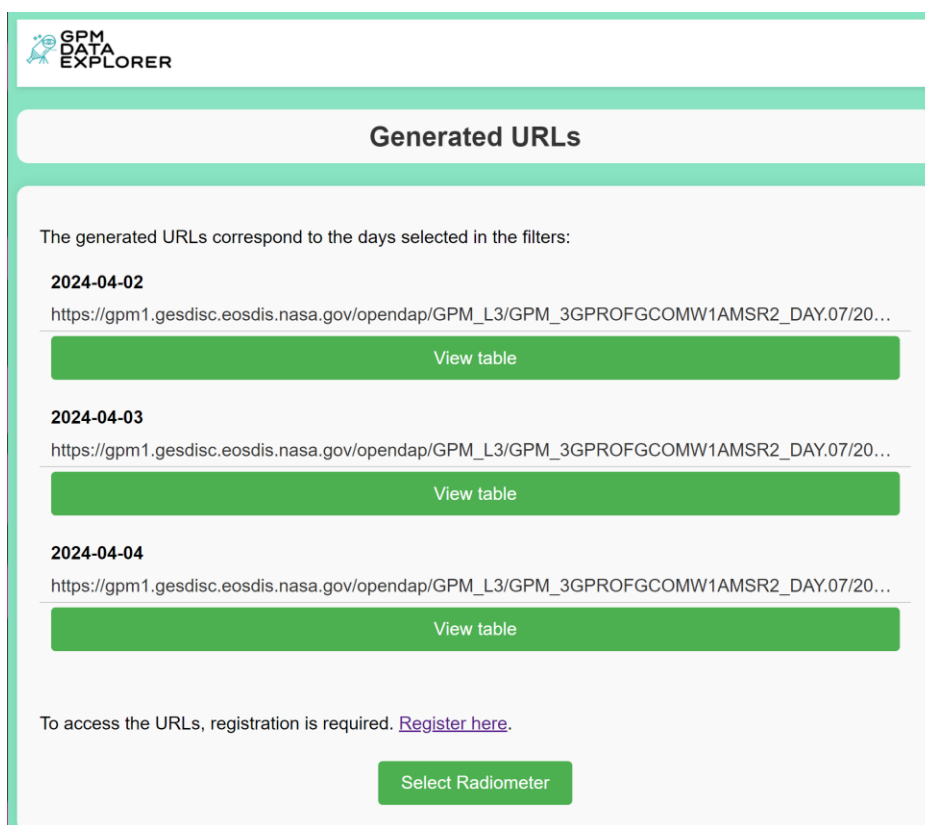


Ilustración 51: Pantalla de URLs generadas con los datos.

Pantalla de Tabla de Visualización

Esta pantalla, como ya se ha mencionado antes, muestra los datos en formato tabla, para la comodidad del usuario (Ilustración 52). Su diseño intuitivo asegura que los usuarios puedan navegar y entender fácilmente la información presentada.



Data for the date: 2024-04-01 (AAAA-MM-DD)

nlat	nlon	nlayer	npix	npixPrecipitation	Surface Precipitation	Convective F
-64.875000	-154.875000	1.750000	41	15	0.0321683140	0.00173
-64.875000	-154.625000	1.750000	39	8	0.0286301770	0.00147
-64.875000	-154.375000	1.750000	38	1	0.0293584430	0.00152
-64.875000	-154.125000	1.750000	40	8	0.0302206040	0.00153
-64.875000	-153.875000	1.750000	43	11	0.0342727500	0.00169
-64.875000	-153.625000	1.750000	37	4	0.0345322500	0.00159
-64.875000	-153.375000	1.750000	42	0	0.0374976200	0.00168
-64.875000	-153.125000	1.750000	49	0	0.0370677970	0.00179
-64.875000	-152.875000	1.750000	47	0	0.0392919850	0.00172
-64.875000	-152.625000	1.750000	43	0	0.0383577940	0.00165

Ilustración 52: Pantalla de visualización de datos .

3 Resultados y Conclusiones

El desarrollo del proyecto *GPMDDataExplorer* ha sido un proceso largo y costoso, especialmente debido al desconocimiento en el campo climático y la complejidad de la documentación proporcionada por la NASA. Gran parte del tiempo se ha invertido en entender y manejar esta documentación, lo cual ha sido un desafío considerable.

Sin embargo, este proyecto ha sido una oportunidad para afianzar muchos de los conocimientos adquiridos durante la carrera de Ingeniería Informática. A lo largo de las distintas fases del proyecto, he podido aplicar y consolidar habilidades en áreas clave como el diseño de sistemas, la gestión de bases de datos, y el desarrollo tanto del *backend* como del *frontend* de una aplicación web completa. Construir una aplicación de esta envergadura era algo que nunca había hecho antes, y enfrentar este desafío ha sido una experiencia muy enriquecedora.

Estoy contenta con el trabajo conseguido y con la capacidad de resolver cada uno de los problemas que surgían a lo largo del proceso. Aunque no es un proyecto perfecto y tiene varios puntos a mejorar, considero que se han cumplido los objetivos en mayor o menor medida.

El proyecto *GPMDDataExplorer* ha demostrado ser una herramienta funcional y útil, aunque con margen para optimizaciones y ampliaciones futuras. En general, estoy satisfecha con el resultado final y con el aprendizaje obtenido durante su desarrollo.

3.1 Evaluación de la Aplicación

La herramienta *GPMDDataExplorer* ha sido desarrollada como un proyecto de fin de grado en Ingeniería Informática, con el objetivo de consolidar los conocimientos adquiridos a lo largo de la carrera. Aunque se trata de un proyecto académico, se ha logrado crear una aplicación funcional que ofrece valiosas capacidades para la exploración de datos meteorológicos.

El desarrollo de *GPMDDataExplorer* ha permitido poner en práctica diversas habilidades técnicas, desde la programación y diseño de interfaces de usuario hasta la integración de servicios web y la gestión de datos. A pesar de estos logros, es importante señalar que, con la participación de un equipo de desarrollo especializado, incluyendo expertos en datos meteorológicos y científicos del sector, la aplicación podría evolucionar significativamente. La colaboración interdisciplinaria podría enriquecer la herramienta con funcionalidades avanzadas y un mayor grado de precisión y utilidad.

Se reconoce que, aunque los resultados obtenidos hasta ahora son satisfactorios, existe un considerable margen para mejoras y expansión. La aplicación tiene el potencial de convertirse en una herramienta robusta y

esencial para la gestión y análisis de datos meteorológicos, siempre y cuando se continúe su desarrollo con una visión a largo plazo.

En el apartado siguiente, se detallan una serie de futuras mejoras que podrían implementarse para incrementar la funcionalidad y la usabilidad de *GPMDDataExplorer*. Estas mejoras están orientadas a maximizar el potencial de la herramienta,

3.2 Futuras Mejoras

A pesar de los logros alcanzados, se identificaron áreas para futuras mejoras que podrían incrementar la funcionalidad y la usabilidad de la aplicación:

Ampliación de funcionalidades

Integrar herramientas adicionales de análisis de datos meteorológicos podría proporcionar un valor añadido significativo a los usuarios. Con la colaboración de profesionales y expertos en meteorología, sería posible identificar las herramientas más beneficiosas y relevantes para los estudios climáticos.

Optimización de la interfaz de usuario

Continuar mejorando la interfaz de usuario a partir de comentarios adicionales puede optimizar aún más la experiencia del usuario, facilitando el acceso a datos y opciones de configuración avanzadas. Al añadir nuevas funcionalidades, podría ser conveniente implementar un sistema de *login* en el que cada usuario tenga su propio perfil. Esto permitiría funcionalidades adicionales, como la capacidad de guardar conjuntos de datos favoritos, personalizar la configuración y almacenar preferencias de usuario. Estas mejoras no solo harían que la interfaz fuera más intuitiva y personalizada, sino que también aumentarían la eficiencia y la satisfacción del usuario al interactuar con la aplicación.

Integración con otras fuentes de datos

Expandir la capacidad de la aplicación para integrar datos de otras misiones y fuentes meteorológicas podría enriquecer la base de datos disponible y ofrecer una visión más completa de las condiciones climáticas globales. Colaborar con agencias meteorológicas, instituciones académicas y otros proveedores de datos podría permitir la incorporación de nuevas fuentes de datos, tales como observaciones terrestres, datos de estaciones meteorológicas y otros satélites climáticos. Esta integración no solo ampliaría el alcance y la precisión de los datos disponibles, sino que también facilitaría análisis comparativos y estudios multidisciplinarios.

Añadir información y documentación técnica

Incorporar información y documentación técnica detallada sobre las misiones, radiómetros, satélites y conjuntos de datos utilizados en la aplicación enriquecería el conocimiento de los usuarios y les permitiría entender mejor el origen y la calidad de los datos. Esta documentación podría incluir

descripciones técnicas, manuales de usuario, datos históricos, especificaciones de los instrumentos y enlaces a recursos externos. Proveer esta información no solo mejoraría la transparencia y la confiabilidad de la aplicación, sino que también serviría como un recurso educativo valioso para investigadores y estudiantes interesados en la meteorología y las ciencias climáticas.

4 Análisis de Impacto

La Agenda 2030 de las Naciones Unidas establece 17 Objetivos de Desarrollo Sostenible (ODS) para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos [21]. En este contexto, el presente Trabajo de Fin de Grado se alinea especialmente con el *ODS 13: Acción por el Clima*. Este objetivo insta a adoptar medidas urgentes para combatir el cambio climático y sus efectos. La lucha contra el cambio climático, incluyendo la reducción de emisiones de gases de efecto invernadero y la adaptación al cambio climático, es una prioridad.

El proyecto desarrollado, una aplicación web para la gestión de información meteorológica de la misión Global Precipitation Measurement (GPM) de la NASA, tiene un impacto directo y significativo en la lucha contra el cambio climático. La aplicación facilita el acceso y análisis de datos meteorológicos, lo cual es crucial para la investigación y la toma de decisiones informadas en relación con el clima.

Al proporcionar una herramienta accesible para la visualización y solicitud de datos meteorológicos, esta aplicación permite a investigadores, científicos y responsables de políticas obtener información precisa y actualizada sobre la precipitación global. Esto es fundamental para:

- **Monitorización de Fenómenos Climáticos:** Permite el seguimiento y análisis de fenómenos climáticos extremos como lluvias torrenciales, sequías y huracanes.
- **Investigación Climática:** Facilita estudios detallados sobre patrones de precipitación y sus cambios a lo largo del tiempo, contribuyendo a la comprensión del cambio climático.
- **Planificación y Adaptación:** Ayuda a planificar estrategias de mitigación y adaptación al cambio climático, esencial para la gestión de recursos hídricos, la agricultura y la planificación urbana.

Como se menciona en el ODS 13, “la agricultura, el turismo, el agua y la energía son sectores particularmente vulnerables a los impactos del cambio climático”, se puede concluir que esta aplicación puede ayudar también a estos sectores.

En el ámbito agrícola, *GPMDDataExplorer* proporciona datos precisos sobre patrones de precipitación, lo cual es crucial para la planificación de cultivos y la gestión eficiente de los recursos agrícolas. Al tener acceso a esta información, los agricultores pueden tomar decisiones informadas sobre cuándo sembrar y cosechar, lo que puede ayudar a optimizar los rendimientos y reducir las pérdidas.

En el sector del turismo, la aplicación ayuda a prever y gestionar los impactos de fenómenos meteorológicos extremos en destinos turísticos. Al proporcionar información detallada sobre el clima y las condiciones meteorológicas, *GPMDDataExplorer* permite a los administradores turísticos tomar medidas proactivas para proteger la seguridad y el bienestar de los turistas, así como para salvaguardar la infraestructura turística.

En cuanto a la gestión del agua, la aplicación facilita la planificación y gestión de recursos hídricos al ofrecer acceso a datos de precipitación y tendencias climáticas. Esta información es fundamental para entender los patrones de lluvia y la disponibilidad de agua, lo que ayuda a los responsables de la gestión del agua a tomar decisiones informadas sobre la asignación de recursos y la implementación de medidas de conservación.

El acceso a datos meteorológicos precisos y detallados también tiene un impacto positivo en la educación y la investigación. La aplicación puede ser utilizada como una herramienta educativa en universidades y centros de investigación, apoyando el *ODS 4: Educación de Calidad*. Estudiantes y académicos pueden utilizar los datos proporcionados por la aplicación para realizar estudios y proyectos de investigación, fomentando una mayor comprensión de los fenómenos meteorológicos y climáticos.

Es importante destacar que, aunque el desarrollo actual de la aplicación no alcanza aún todo su potencial funcional, se perfila como una herramienta con grandes capacidades. Con futuras mejoras y ampliaciones, la aplicación tiene el potencial de convertirse en una plataforma esencial para la investigación y gestión climática. Su evolución puede contribuir aún más significativamente a la acción climática global, apoyando la resiliencia y sostenibilidad en múltiples sectores de la economía y la sociedad española.

Bibliografía

- [1] NASA. (2019). Global Precipitation Measurement (GPM) Mission. [PDF]. Disponible: <https://gpm.nasa.gov/media/418>. [Último acceso: 8/03/2024].
- [2] Precipitation Processing System. (2022). GPM Level 3 GPROF Version 07 Information. [PDF]. Disponible: <https://gpm.nasa.gov/resources/documents/gpm-level-3-gprof-version-07-information>. [Último acceso: 10/03/2024].
- [3] NASA Earthdata Search. (s.f). NASA Earthdata Search. Disponible: <https://search.earthdata.nasa.gov/search>. [Último acceso 12/04/2024]
- [4] NASA Giovanni. (s.f). NASA Giovanni. Disponible: <https://giovanni.gsfc.nasa.gov/giovanni>. [Último acceso: 22/04/2024]
- [5] NASA, " Integrated Multi-satellitE Retrievals for GPM (IMERG)," NASA. [Online]. Disponible: <https://gpm.nasa.gov/data/imerg>. [Último acceso: 12/05/2024].
- [6] Solano-Fernández, E., & Porras-Alfaro, D. (2020). El modelo iterativo e incremental para el desarrollo de la aplicación de realidad aumentada Amón_RA. Tecnología en Marcha, Edición especial, Escuela de Arquitectura y Urbanismo, 165-177. [PDF]. Disponible: <https://doi.org/10.18845/tm.v33i8.5518> [Último Acceso: 10/05/2024].
- [7] López, C. A. (2009). "Cómo mantener el patrón modelo vista controlador en una aplicación orientada a la WEB." Revista Inventionum. [Online]. Disponible: <https://doi.org/10.26620/uniminuto.inventionum.4.7.2009.72-78>. [Último Acceso: 01/05/2024].
- [8] MVC (Model, View, Controller). [Imagen]. Disponible: <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>. [Último acceso: 01/06/2024].
- [9] GitHub, "WFreezer/GPMDDataExplorer", GitHub. [En línea]. Disponible: <https://github.com/WFreezer/GPMDDataExplorer.git>. [Último acceso: 20/06/2024].
- [10] OPeNDAP, "OPeNDAP". [En línea]. Disponible: <https://www.opendap.org/>. [Último acceso: 17/05/2024].
- [11] "Hyrax Data Server (OPeNDAP)". NASA IT Security Warning Banner. [En línea]. Disponible: https://gpm1.gesdisc.eosdis.nasa.gov/opendap/GPM_L3/contents.html. [Último acceso: 10/06/2024].
- [12] Node.js, "Node.js". [En línea]. Disponible: <https://nodejs.org/en>. [Último Acceso: 17/05/2024].
- [13] Express, "Express - Node.js web application framework," Express. [Online]. Disponible: <https://expressjs.com/>. [Último Acceso: 26/05/2024].

- [14] “API Documentation & Design Tools for Teams | Swagger”. API Documentation & Design Tools for Teams | Swagger. [En línea]. Disponible: <https://swagger.io/>. [Último Acceso: 02/05/2024].
- [15] “Documentation - MariaDB.org”. MariaDB.org. [En línea]. Disponible: <https://mariadb.org/documentation/>. [Último Acceso: 19/06/2024].
- [16] HeidiSQL, “HeidiSQL – MariaDB/MySQL”. [En línea]. Disponible: <https://www.heidisql.com/> . [Último acceso: 19/06/2024].
- [17] Mozilla, "Cross-Origin Resource Sharing (CORS)," Mozilla. [En línea]. Disponible en: <https://developer.mozilla.org/es/docs/Web/HTTP/CORS>. [Último acceso: 20/05/2024].
- [18] “Cross-site scripting”. MDN Web Docs. [En línea]. Disponible: https://developer.mozilla.org/es/docs/Glossary/Cross-site_scripting [Último acceso: 15/05/2024].
- [19] Angular.io, «Angular,» 2023. [En línea]. Disponible en: <https://angular.io/>.
- [20] Leaflet.js. (s.f.). Leaflet - a JavaScript library for interactive maps. [En línea]. Disponible en: <https://leafletjs.com/>. [Último acceso: 02/05/2024].
- [21] “Ministerio de Derechos Sociales, Consumo y Agenda 2030 - Agenda 2030”. Ministerio de Derechos Sociales, Consumo y Agenda 2030 - Agenda 2030. [En línea]. Disponible: <https://www.agenda2030.gob.es/agenda2030/index.htm>. [Último acceso: 20/05/2024].