UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN SISTEMA WEB BASADO EN UNA ARQUITECTURA DE MICROSERVICIOS PARA LA GENERACIÓN DE REPORTES DE DATOS HIDROMETEREOLÓGICOS DE LAS ESTACIONES CONVENCIONALES DEL INAMHI.

Trabajo de titulación previo a la obtención del

Título de: Ingenieros de sistemas

AUTORES: JUAN CARLOS APOLO CUMBICOS

BYRON BLADIMIR TACO TACO

TUTORA: PAULINA ADRIANA MORILLO ALCÍVAR

Quito - Ecuador

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍADEL TRABAJO DE TITULACIÓN

Nosotros Juan Carlos Apolo Cumbicos con documento de identificación Nº 1719353367 y Byron Bladimir Taco Taco, Nº 1723518567 manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir, o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 02 de agosto del año 2022

Atentamente,

…… ……....

Juan Carlos Apolo Cumbicos

1719353367

Byron Bladimir Taco Taco

1723518567

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros Juan Carlos Apolo Cumbicos con documento de identificación Nº 1719353367 y Byron Bladimir Taco Taco, Nº 1723518567, expresamos nuestra voluntad propia y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: "Desarrollo de un sistema web basado en una arquitectura de microservicios para la generación de reportes de datos hidrometereológicos de las estaciones convencionales del INAMHI.", el cual ha sido desarrollado para optar por el título de Ingeniero de Sistemas, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con los manifestado, suscribimos este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 02 de agosto del año 2022

Atentamente,

…… ……..

Juan Carlos Apolo Cumbicos

1719353367

Byron Bladimir Taco Taco

1723518567

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Paulina Adriana Morillo Alcívar con documento de identificación Nº 1715646574, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO DE UN SISTEMA WEB BASADO EN UNA ARQUITECTURA DE MICROSERVICIOS PARA LA GENERACIÓN DE REPORTES DE DATOS HIDROMETEREOLÓGICOS DE LAS ESTACIONES CONVENCIONALES DEL INAMHI realizado por Juan Carlos Apolo Cumbicos con documento de identificación Nº 1719353367 y Byron Bladimir Taco Taco, Nº 1723518567, obteniendo como resultado final el trabajo de titulación bajo la opción proyecto técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 02 de agosto del año 2022

Atentamente,

.....

Ing. Paulina Adriana Morillo Alcívar, MsC

1715646574

ÍNDICE GENERAL

INTRODUCCIÓN 1
ANTECEDENTES 1
PROBLEMA DE ESTUDIO 2
JUSTIFICACÍON 3
GRUPO OBJETIVO (Beneficiarios) 4
OBJETIVOS 4
Objetivo General 4
Objetivos Específicos: 4
METODOLOGÍA 5
CAPITULO I 8
1. MARCO TEORICO 8
1.1. Meteorología 8
1.2. Hidro-meteorología 8
1.3. Cálculos Hidrometeorológicos 9
1.3.1. Cálculo de la precipitación Diaria 9
1.3.2. Cálculo de la Evaporación 10
1.3.3. Cálculo de la Temperatura 12
1.3.4. Cálculo de la Nubosidad Diaria 13
1.3.5. Cálculo de la Tensión del vapor (Hp), la Humedad Relativa (%) y el Punto de Rocío (°C) 13
1.3.6. Cálculo del Viento 14
1.4. Servicios Web 15
1.5. Interfaz de programación de aplicaciones (API) 16
1.6. HERRAMIENTAS DE DESARROLLO 17
1.6.1. GitHub 17
1.6.2. Zenhub 17
1.6.3. Postman 18
1.6.4. Motor de base de datos PostgreSQL 18
1.6.5. Python 18
1.6.6. Django 19
1.6.7. Mixins 19
1.6.8. Swagger 19
1.6.9. React 20

1.6.10. Material UI 20
1.6.11. Plotly 20
1.6.12. JMeter 21
CAPITULO II 21
2. DISEÑO DE LA APLICACIÓN 21
2.1. Estado actual del sistema de gestión de la información del INAMHI 21
2.2. Alcance del proyecto 21
2.3. Análisis y levantamiento de requerimientos del sistema 22
2.3.1. Análisis de factibilidad. 22
2.3.2. Viabilidad Técnica. 22
2.3.3. Viabilidad Económica. 24
2.3.4. Viabilidad Operacional. 25
2.3.5. Características de usuarios. 26
2.3.6. Restricciones del sistema 27
2.3.7. Requerimientos funcionales y no funcionales 27
2.4.3.1 Requerimientos funcionales. 28
2.4.3.2 Requerimientos no funcionales 31
2.4. Diagramas de desarrollo de la aplicación 34
2.4.1. Diagrama general BackEnd 34
2.4.2. Diagrama FrontEnd 35
2.4.3. Diagramas Entidad Relación 36
2.4.4. Diagramas de Casos de USO 36
2.4.5. Caso de uso de Ingreso de datos 37
2.4.6. Caso de uso de Visualización de información 37
2.4.7. Diagramas de secuencia (ingreso de datos) 38
2.4.8. Diagrama de secuencia (actualización de datos) 39
2.4.9. Diagrama de secuencia (visualizar gráficas y exportar datos) 40
2.4.10. Diagrama de componentes 40
2.4.11. Diagrama de clases 41
CAPITULO III 43
3. DESARROLLO DE LA APLICACIÓN 43
3.1. Product Backlog 44

3.2. Sprint 1 Selección de herramientas para el desarrollo de la API 45
3.3. Sprint 2 Visualización del modelado de la base de datos 46
3.4. Sprint 3 Desarrollo de diagramas UML 47
3.5. Sprint 4 Instalación de aplicaciones para el backend 48
3.6. Sprint 5 Desarrollo de los modelos zerializables y views 51
3.6.1. Creación de clases para los modelos 52
3.6.2. Creación de las clases para los serializers. 53
3.6.3. Creación de las clases Views 54
3.6.4. Creación de clase Padre (Principal). 55
3.6.5. Creación de las clases hijas 56
3.6.6. Creación de las urls 56
3.7. Sprint 6 Implementación de cálculos meteorológicos 57
3.7.1. Cálculo de la precipitación 58
3.8. Sprint 7 Implementación de Swagger 66
3.8.1. Instalación de Swagger. 66
3.9. Sprint 8 Preparación del entorno Front-End 68
3.9.1. Instalación de NODEJS 69
3.10. Sprint 9 Instalación de librerías para Front-End 69
3.10.1. Creación de la aplicación en React. 70
3.10.2. Instalación de la librería material ui. 71
3.10.3. Instalación de complementos de material-ui. 71
3.10.4. Instalación de material-table. 72
3.11. Sprint 10 Desarrollo de la interfaz gráfica FrondEnd 73
3.12.1 Creación de una variable constante para las peticiones al servidor backend. 74
3.12.2 Creación de la función GET. 75
3.12.3 Creación de la función Post. 75
3.12.4 Creación de la función Put. 76
3.12.5 Creación de la función DELETE. 77
3.12.6 Creación del Hook useForm. 78
3.12.7 Formulario de ingreso de datos. 79

3.12.8 Componente table. 80
3.12.9 Componente Plot 82
CAPITULO IV 85
4 PRUEBAS Y VALIDACIÓN DEL SISTEMA 85
4.1. Pruebas unitarias 85
4.2. Pruebas de integración 88
4.2.1. Prueba del Método GET 88
4.2.2. Prueba del método POST 89
4.2.3. Prueba del método PUT 91
4.2.4. Prueba del método DELETE 93
4.3. Pruebas de aceptación 94
4.4. Pruebas de estrés 94
4.4.1. Creación de Group tree en JMeter 95
4.4.2. Configuración de tree Group 96
4.4.3. Creación de HTTP Request 97
4.4.4. Configuración de HTTP Request en JMeter 98
4.4.5. Creación de View Resul Tree en JMeter 99
4.4.6. Resultado de View Resul Tree 100
4.4.7. Visualización de resultados en Graph Result 101
CONCLUSIONES 103
RECOMENDACIONES 104
REFERENCIAS 105

ÍNDICE DE FIGURAS

Figura 1. *Hoja de libreta hidrometereológica.* 10
Figura 2. *Diagrama general Back-End* 34
Figura 3. *Diagrama de FrontEnd* 35
Figura 4. *Diagrama de modelo entidad-relación* 36
Figura 5. *Caso de uso de Ingreso de datos* 37
Figura 6. *Caso de uso de Visualización de información* 38
Figura 7. *Diagramas de secuencia (ingreso de datos)* 38
Figura 8. *Diagrama de secuencia (actualización de datos)* 39
Figura 9. *Diagrama de secuencia (visualizar gráficas y exportar datos)* 40
Figura 10. *Diagrama de componentes* 41
Figura 11. *Diagrama de los modelos* 42
Figura 12. *Diagrama de clases serielizables.* 42
Figura 13. *Diagrama de clases conectada a la principal.* 43
Figura 14. *Lista de actividades (producto Backlog)* 44
Figura 15. *Selección de herramientas (sprint1)* 45
Figura 16. *Revisión de documentos del INAMHI Sprint 2* 46
Figura 17. *Desarrollo de diagramas UML sprint 3* 47
Figura 18. *Instalación de aplicaciones para el backend, Sprint 4* 48
Figura 19. *Comandos para crear la variable de entorno.* 49
Figura 20. *Comandos para crear el proyecto y la aplicación.* 50
Figura 21. *Configuración para conectarse a la base de datos* 51
Figura 22. *Contenido para la codificación. Sprint 5* 52
Figura 23. *Comandos para crear clases de los modelos* 52
Figura 24. *Ejemplo de una clase model.* 53
Figura 25. *Ejemplo de una clase serielizer.* 54
Figura 26. *Implementación de código de la clase principal.* 55
Figura 27. *Ejemplo de código para clases hijas* 56
Figura 28. *Codificación para las peticiones http.* 57
Figura 29. *Actividad para realizar el Sprint 6* 57
Figura 30. *Código del cálculo de la precipitación.* 58

Figura 31. *Código de validación de la precipitación.* 59
Figura 32. *Validación para valores iguales de la precipitación.* 60
Figura 33. *Código para el cálculo del índice fijo.* 61
Figura 34. *Código para el cálculo de la evaporación suma diaria.* 62
Figura 35. *Código para el cálculo de la temperatura del aire suma diaria* 62
Figura 36. *Código para el cálculo de la nubosidad diaria.* 63
Figura 37. *Código para el cálculo de la tensión vapor, humedad relativa, punto de rocío…* 64
Figura 38. *Código para el cálculo del viento diario recorrido.* 65
Figura 39. *Actividades para el Sprint 7* 66
Figura 40. *Instalación de swagger.* 67
Figura 41. *Código para configurar el openapi.* 67
Figura 42. *Diagrama de la documentación de la API.* 67
Figura 43. *Actividades para el sprint 8* 68
Figura 44. *Instalación de librerías para Front-End* 70
Figura 45. *Creación de la aplicación en React.* 70
Figura 46. *Instalación de material-ui.* 71
Figura 47. *Instalación de complementos de material-ui.* 71
Figura 48. *Instalación de material-table.* 72
Figura 49. *Código complementario de material-ui (icons)* 73
Figura 50. *Listado de actividades del Sprint 10* 73
Figura 51. *Url de dirección del servidor backend.* 74
Figura 52. *Código de la función Get.* 75
Figura 53. *Código de la función Post* 75
Figura 54. *Código de la función Put.* 76
Figura 55. *Código de la función Delete.* 77
Figura 56. *Hook useForm* 78
Figura 57. *Ventana de ingreso de datos.* 79
Figura 58. *Implementación de MaterialTable.* 81
Figura 59. *Ventana para visualizar información.* 82
Figura 60. *Implementación del componente Plotly.* 82
Figura 61. *Gráfico de barras de los datos.* 83

Figura 62. *Instalación de pytest.* 85
Figura 63. *Muestra de ubicación del archivo pytest.ini* 85
Figura 64. *Modificación de contenido en el archivo pytest.ini* 86
Figura 65. *Envió de pruebas unitarias.* 87
Figura 66. *Llamada a un endpoint con el método GET.* 89
Figura 67. *Llamada a un endpoint con el método POST* 90
Figura 68. *Muestra exitosa de datos ingresados.* 90
Figura 69. *Llamada a un endpoint con el método PUT* 91
Figura 70. *Muestra exitosa de la actualización de datos.* 92
Figura 71. *Llamada a un endpoint con el método DELETE* 93
Figura 72. *Muestra exitosa de la eliminación de datos con respecto al id 5* 94
Figura 73. *Creación de grupo de pruebas* 95
Figura 74. *Prueba de peticiones al servidor con cierto número de usuarios en un tiempo determinado.* 96
Figura 75. *Creación de HTTP request para las peticiones al servidor.* 97
Figura 76. *Configuración de la url del servidor a realizar la prueba de carga y estrés.* 98
Figura 77. *Creación para la vista del árbol de resultados* 99
Figura 78. *Resultado de la petición realizada* 100
Figura 79. *Resultado en gráfico* 101

INDICIE DE TABLAS

Tabla 1. *Roles definición y personas a cargo.* 5

Tabla 2 *Simbología de Viento* 14

Tabla 3. *Requisitos del Hardware* 22

Tabla 4. *Requisitos del Software* 23

Tabla 5. *Costos considerados para el desarrollo del proyecto* 24

Tabla 6. *Descripción de usuarios* 26

Tabla 7. *Modelo de matriz del levantamiento de los requerimientos* 27

Tabla 8. *Descripción de las Apertura de las urls.* 28

Tabla 9. *Peticiones de tipos REST.* 29

Tabla 10. *Métricas requeridas* 30

Tabla 11. *Requisito funcional 4* 31

Tabla 12. *Compatibilidad de Sistemas Operativos* 32

Tabla 13 *Disponibilidad de la aplicación* 32

Tabla 14. *Requisito de arquitectura a ser usado.* 33

Tabla 15. *Características del servidor* 95

RESUMEN

En este proyecto se desarrolló un sistema web para la gestión de información del Instituto Nacional de Meteorología e hidrología del Ecuador, el cual consta de dos módulos, uno para la generación de reportes históricos y otro para la validación de los cálculos meteorológicos de las estaciones convencionales de dicha entidad. Estos módulos formarán parte de un sistema de gestión de información integral que estará basado en una arquitectura de microservicios, la cual facilitará la integración de todo el software en el futuro. La metodología que se siguió para el desarrollo del proyecto fue Scrum, a través del uso de la herramienta ZenHub para la gestión de las actividades. Así, el primer paso consistió en el análisis de factibilidad, que determinó la viabilidad técnica, económica y operacional del proyecto, de este modo se definieron los requerimientos del sistema. Posteriormente, se procedió con el diseño de los diagramas UML de la aplicación, usando las herramientas Visual Paradigm y Draw.io. Luego, se realizó la construcción de la aplicación usando los *frameworks* Django y React, para el *BackEnd y FrontEnd*, respectivamente. Finalmente, se realizaron pruebas unitarias, de estrés y de carga que permitieron observar el rendimiento general del sistema. De acuerdo con los resultados en las pruebas unitarias todas las peticiones realizadas a los *endpoints* devolvieron los datos solicitados. En cuanto a las pruebas de estrés y carga se observó que el número máximo de usuarios que se pueden conectar simultáneamente fue de 1782178 u/m, no fue necesario llevar al límite las pruebas ya que el aplicativo va a ser usado exclusivamente por funcionarios del INAMHI, esto limita el acceso a un número reducido de usuarios. De esta forma, se concluye que la aplicación es funcional y que cumple con los requerimientos solicitados por la institución.

ABSTRACT

In this project, a web system was developed for the information management of the "Instituto Nacional de Meteorología e Hidrología" of Ecuador, which consists of two modules, one for the generation of historical reports and the other for the validation of the meteorological calculations of the conventional stations of said entity. These modules will form part of a comprehensive information management system based on a microservices architecture. This architecture will facilitate the integration of all software in the future. The methodology followed for the development of the project was Scrum, using the ZenHub tool for the management of activities. Therefore, the first step was a feasibility analysis, which determined the technical, economic, and operational feasibility of the project. In this way, the requirements of the system were defined. Subsequently, the design of the UML diagrams of the application was made using the Visual Paradigm and Draw.io tools. Then, the construction of the application was carried out using the Django and React frameworks, for the BackEnd and FrontEnd, respectively. Finally, unit, stress, and load tests were performed that allowed one to observe the general behavior of the system. According to the unit test results, all requests made to the endpoints returned the requested data. Regarding the stress and load tests, the results show that the maximum number of users that could be connected simultaneously was 1782178 u/m. Although the tests were not taken to the limit, since the application will be for the exclusive use of INAMHI officials, that reduces access to a small number of users. In this way, the application is functional, and the requirement of the institution was fulfilled.

INTRODUCCIÓN

ANTECEDENTES

El Instituto Nacional de Meteorología e Hidrología (INAMHI) es un instituto científico encargado de recolectar y difundir los datos históricos, actuales y futuros acerca del tiempo, clima y recursos hidrometereológicos del Ecuador. Además, el INAMHI es miembro de la Organización intergubernamental de las Naciones Unidas para la Meteorología, conocida como OMM, esto le permite seguir estándares internacionales en busca de mejorar sus predicciones del tiempo, a través del uso de tecnología. Por lo tanto, actualmente, la institución cuenta con herramientas de hardware y software que le permiten llevar a cabo la adquisición y almacenamiento de los datos para una posterior construcción de modelos de predicción de tiempo. Sin embargo, la arquitectura que maneja su sistema informático fue implementada en el año 2011 y es de tipo monolítica, este tipo de estructura dificulta el mantenimiento del software y la implementación de nuevas funcionalidades. Por esta razón, a partir del 2021 el INAMHI empezó a migrar su sistema a una arquitectura modular conocida como microservicios, esta arquitectura le permitirá mejorar la gestión de su información.

En el desarrollo de software se considera una arquitectura monolítica a la unión de distintos módulos que al empaquetarse forman un solo sistema. La ejecución de este sistema depende de la funcionalidad de cada uno de sus componentes, es decir, si alguno de los módulos falla el programa no tendrá éxito en su compilación. Además, todos los módulos del sistema deben alojarse en un mismo equipo (Augusto, 2017). Por otro lado, la arquitectura de microservicios consiste en un conjunto de módulos de software que trabajan de forma independiente. Generalmente se ejecutan en diferentes entornos y pueden ser desarrollados en distintos lenguajes de programación,

permitiendo a las empresas realizar una mejor gestión de su información, y que sus sistemas sean escalables y mantenibles en el tiempo (López & May, 2017).

PROBLEMA DE ESTUDIO

Actualmente, el INHAMI usa un programa de escritorio para la gestión de los datos hidrometeorológicos que recolecta. Sin embargo, este programa no fue implementado bajo el paradigma orientado a objetos ni basado en una arquitectura modular, lo que dificulta el cumplimiento de los estándares normados por la OMM. Además, algunas de las herramientas informáticas que se utilizan en la adquisición y procesamiento de datos, no garantizan los principios fundamentales de escalamiento y mantenimiento de software. Por esta razón, el INAMHI ha optado por migrar sus aplicaciones a una arquitectura de microservicios, mediante *frameworks* de desarrollo modernos como React y Django, con el fin de mejorar el acceso y procesamiento de sus datos.

JUSTIFICACÍON

A pesar de que el INAMHI cuenta con herramientas que le permiten recolectar, procesar y almacenar datos, el sistema informático usado, hoy por hoy, no ha recibido ninguna actualización desde su creación. Por lo tanto, se hace necesario la construcción de una nueva plataforma que cumpla con los estándares de desarrollo de software actuales, para mejorar su adaptabilidad a las nuevas tecnologías y disminuir los cuellos de botella generados en el procesamiento de los datos. Por otro lado, de acuerdo con políticas de estado, todas las instituciones públicas están obligadas a usar software libre en sus sistemas. Ergo, el INAMHI ha optado por la modernización de sus sistemas de gestión de la información, aplicando una arquitectura modular y herramientas de software libre, que le permitirán separar el modelo de negocio de la interfaz gráfica de usuario y liberarse, en lo posible, de licencias de software costosas.

GRUPO OBJETIVO (BENEFICIARIOS)

El desarrollo del módulo de reporte de datos y validación será una parte esencial del sistema completo de gestión de información del INAMHI. Particularmente, este módulo beneficiará al departamento de información de la institución, debido a que ayudará a mejorar análisis de los datos que servirán como entrada para los modelos de predicción de tiempo.

OBJETIVOS

Objetivo General

- Desarrollar una aplicación web para la visualización de datos hidrometereológicos de estaciones convencionales del INAMHI.

Objetivos Específicos:

- Analizar los requerimientos establecidos por el departamento de información del INAMHI para desarrollar la aplicación web.
- Diseñar una aplicación web aplicando una arquitectura modular para la visualización de los datos obtenidos de las estaciones convencionales.
- Construir un módulo de validación de los datos inconsistentes obtenidos de la base de datos del INAMHI.
- Generar un módulo de reportes de los datos históricos mediante el uso de gráficos estadísticos.
- Realizar pruebas de rendimiento y estrés de los módulos desarrollados.

METODOLOGÍA

Para el desarrollo de este proyecto se utilizó el marco de trabajo SCRUM, debido a la gran incertidumbre que se tiene para la construcción del módulo y la flexibilidad de los requerimientos solicitados por parte de la institución.

Las metodologías ágiles facilitan el manejo de proyectos de desarrollo de software debido a sus beneficios y la gran flexibilidad que brindan para nuevos requerimientos en el producto, a su vez ayuda a tener una colaboración directa con los *stackeholders*, en este caso con el departamento de información del INAMHI (CASTILLO & CEVALLOS, 2021)**.**

SCRUM es considerado un marco de trabajo de desarrollo incremental iterativo en el que se puede emplear un conjunto de técnicas y procesos para trabajar colaborativamente en equipo, obteniendo mejores resultados de un proyecto donde los entornos son complejos y los requisitos son cambiantes, y sobretodo, en proyectos en los que se debe realizar entregas parciales (Sprint) del producto final, ejecutadas en fases cortas que son de dos a cuatro semanas. A su vez, ayuda a que los equipos se auto-organicen. (KNOWLEDGE, 2013).

En Scrum los roles son personas que están a cargo de los proyectos, en la Tabla 1 se muestra la definición de los roles considerados en este proyecto.

Tabla 1.

Roles definición y personas a cargo.

Roles	Definición	Persona a cargo
Product Owner	Persona que representa al cliente, es decir, es el encargado de ver todas las necesidades del usuario para posteriormente comunicar al scrum máster y	Ing. Darwin Rosero

	al *team development*, para que conjuntamente desarrollen dicha necesidad.	
Scrum Máster	Es el moderador entre el *Product Owner* y el *Team development*, encargado de eliminar dificultades que afecten a la entrega del producto además gestiona el proceso Scrum, también facilita reuniones y eventos si lo considera pertinente.	Msc. Paulina Morillo.
Development team	Personas altamente capacitadas para dar solución a la necesidad o requisitos del cliente, el equipo suele estar formado de entre 3-9 personas auto-organizados para proporcionar un entregable.	- Ing. Tatiana Merino - Sr. Juan Apolo - Sr. Byron Taco
Users	Personas que usaran el producto.	- Ing. José Luis - Ing. Tatiana Merino.

Nota*: Roles SCRUM, definición y personas que ocupan el cargo respectivamente. Elaborado por: (Apolo & Taco).*

Los aspectos de SCRUM son actividades programadas que se van completando de acuerdo con el avance del proyecto. Los aspectos utilizados en este proyecto se describen a continuación:

- ***Product Backlog***: Lista de requerimientos obtenidos del cliente por parte del *Product Owner*, que darán cumplimiento a la solicitud del cliente.
- ***Sprint Planning Meeting***: Es la primera reunión que se lleva a cabo para planificar el entregable de la primera fase hasta el final del producto, es decir, se obtiene una lista de funcionalidades tomadas, el *Product Backlog*.

- ***Sprint Backlog***: Es la lista de funcionalidades obtenidas del *Product Backlog*, es decir, son un conjunto de requisitos que se deben presentar en un tiempo estimado de 1-4 semanas más conocidas como *Sprint*.
- ***Sprint***: Es el proceso de desarrollo de la necesidad del cliente, divididas en un módulo funcional incremental, aquí intervienen el *Scrum Master* y *el Development Team*.

Después de definir todos los aspectos que conlleva el uso de SCRUM, se utilizó técnicas como las reuniones (presenciales y virtuales) y entrevistas para la recolección de información que permitió establecer los requerimientos del sistema. Posteriormente, se realizó el análisis de los requerimientos y la planificación del diseño y desarrollo del sistema. Esto se llevó a cabo mediante herramientas de gestión de proyectos como ZenHub y teams, los cuales permitieron planificar los plazos de cada actividad y realizar entregas continuas de las actividades establecidas en el *product Backlog*. De esta forma, se garantizó que el sistema web cumpla con las necesidades y demandas del usuario.

CAPITULO I

1. MARCO TEORICO

En este capítulo se exponen los conceptos básicos de temas relacionados con la meteorología, el desarrollo del servicio web (API) de hidrometereológica y la interfaz gráfica de usuario (UI). Además, se describen las herramientas que se utilizan para el desarrollo del producto y los instrumentos usados en la gestión del proyecto.

1.1. Meteorología

Es la ciencia que se encarga de estudiar los fenómenos atmosféricos a corto plazo, además, de sus distintas propiedades (viento, lluvia, temperatura, presión atmosférica entre otros factores) dadas en la atmósfera terrestre en un determinado tiempo y lugar, estos eventos en la actualidad es utilizada para predecir el tiempo climático que beneficia a múltiples actividades económicas, además, de reducir riesgos con respecto a los desastres hidrometereológicos (Huracán, deslices de tierra, inundaciones, etc) (Marisol Andrades, 2012).

1.2. Hidro-meteorología

Es una rama que forma parte de las ciencias de la atmosfera (meteorología e hidrología) que estudia el estado del agua (evaporación, condensación y precipitación) conocidas como fases atmosféricas y terrestres (ciclo hidrológico), además, se encarga de analizar la transferencia de agua y energía entre la superficie terrestre y la atmósfera, algunos ejemplos atmosféricos son: los huracanes, las inundaciones y las tormentas de nieve o granizo (Huan Wu, 2016).

1.3. Cálculos Hidrometeorológicos

La información de los cálculos que se presentara a comunicación fue brindada por la entidad el INAMHI. Dichos cálculos son estándares mundiales y sirven para predecir los cambios climáticos que se dan en el medio ambiente.

1.3.1. Cálculo de la precipitación Diaria

La precipitación es un proceso hidrometereológico que consiste en la caída de agua ya sea liquida (lluvia) o solida (granizo) hacia la superficie terrestre. La unidad de medida de la precipitación es el milímetro. En la Figura 1 se muestran los valores de precipitación que se ingresan manualmente en las libretas meteorológicas ordinarias, se consideran las observaciones de las 07:00, 13:00 y 19:00 horas del día. La Ecuación (1) muestra la forma de calcular la precipitación diaria, que es igual a la suma de los valores de precipitación de las 13h00, de las 19h00 del día actual, más la precipitación de las 7h00 del siguiente día.

$$\boldsymbol{Suma\_diaria} = valor13h + valor19h + valor07h\_diasiguiente \quad \textbf{(1)}$$

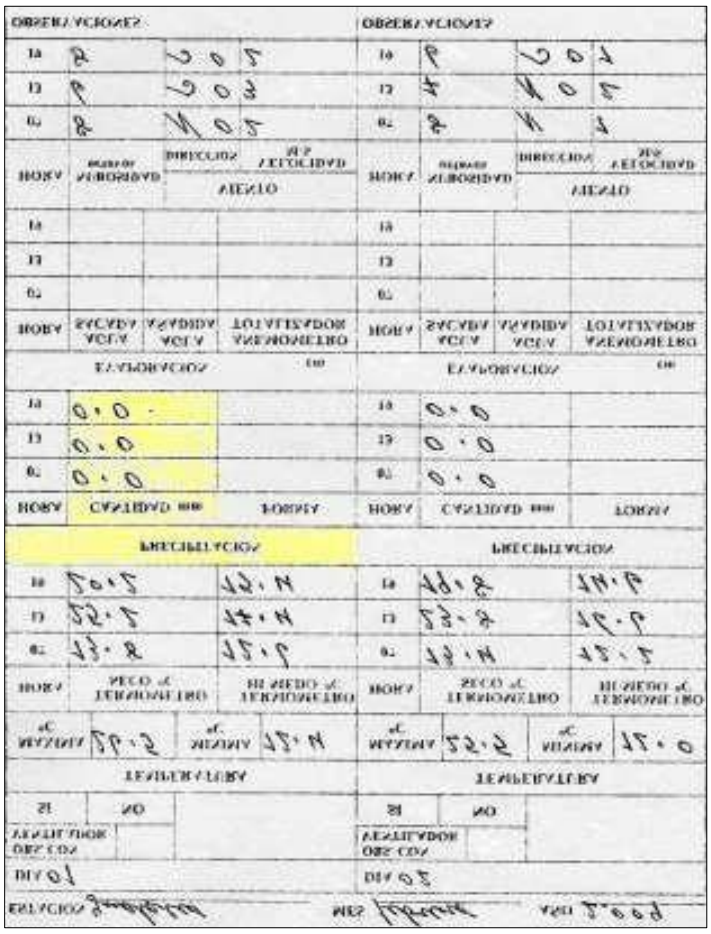
Sin embargo, es necesario considerar las validaciones que se describen a continuación:

- Se puede digitar 999.9 equivalente a nulo y 888.8 equivalente a traza.
- En el caso de que, el valor de las de las 07:00 del día siguiente y valor de las 13:00 del día siguiente es cero, si se puede calcular la precipitación de ese día y del siguiente día
- Si el valor de las 13:00 del día siguiente es mayor a 0 no se debe calcular el valor diario porque puede ser un acumulado.
- Si falta solo el valor de las 19h00, si se puede calcular la lluvia diaria:
- Si no existen datos de las 13:00 y 19:00 si se puede obtener la precipitación diaria.

- Si no existen los valores de las 13:00, 19:00 y 7:00 del día siguiente se anula el valor de este día y del siguiente, los valores de trazas son considerados como CERO, el valor máximo en 24 horas se obtiene de las sumas diarias cuando el mes es completo.

Figura 1.

Hoja de libreta hidrometereológica.



***Nota**: Ejemplo de toma de datos de las estaciones convencionales mediante la librera meteorológica. Elaborado por: INAMHI.*

1.3.2. Cálculo de la Evaporación

La Evaporación es un proceso en el cual el agua se evapora y sube a la atmósfera terrestre, que suele medirse en milímetros. En el campo de la meteorología existen diferentes tipos de cálculos

de evaporación En el caso del INAMHI se manejan tres tipos de evaporación las cuales se listan a continuación:

1.3.2.1. Evaporación índice fijo (agua añadida, agua sacada)

La evaporación tipo índice fijo (7:00, 13:00 y 19:00), estos valores no se usan para generar la evaporación diaria, mensual, etc; estos datos deben usarse para calcular la evaporación real a las 7:00, 13:00 y 19:00.

1.3.2.2. Cálculo de la evaporación de las 7 horas

Si el agua añadida de las 7:00 o agua sacada de las 7:00 o la precipitación de las 7:00 es nula, el resultado es nulo.

Caso contrario

$$EVA07h=PREC07h - \frac{aguasacada07h}{20} + \frac{aguaañadida07h}{20} \quad \textbf{(2)}$$

En la Ecuación **(2)** se muestra el cálculo de la evaporación índice fijo de las 7:00 se debe utilizar la misma fórmula para calcular la evaporación de las 13:00 y 19:00

Nota: El agua añadida y agua sacada son valores netamente ingresados y no necesitan ningún calculo previo.

1.3.2.3. Evaporación calculada

Si los datos ingresados de la evaporación calculada son los mismos a los ingresados en las diferentes horas (07h00, 13h00 y 19h00), no se deben someter a las fórmulas de cálculo previas. Algunas de las validaciones para este cálculo se muestran a continuación:

- Se puede digitar 999.9 equivalente a nulo y 888.8 equivalente a traza.
- En el caso de que, el valor de las de las 07:00 del día siguiente y valor de las 13:00 del día siguiente es cero, si se puede calcular la evaporación de ese día y del siguiente día.

- Si el valor de las 13:00 del día siguiente es mayor a 0 no se debe calcular el valor diario porque puede ser un acumulado.
- Si falta solo el valor de las 19:00, si se puede calcular la evaporación diaria:
- Si no existen datos de las 13:00 y 19:00 si se puede obtener la evaporación diaria.
- Si no existen los valores de las 13:00, 19:00 y 7:00 del día siguiente se anula el valor de este día y del siguiente, los valores de trazas son considerados como CERO, el valor máximo en 24 horas se obtiene de las sumas diarias cuando el mes es completo.

$$Svap\_diaria = valor13h + valor19h + valor07h\_diasiguiente \quad \textbf{(3)}$$

En la Ecuación (3) se observa la fórmula de la evaporación diaria esta es igual a la suma de los valores de precipitación de las 13:00, de las 19:00 del día actual, más la precipitación de las 7:00del siguiente día.

1.3.3. Cálculo de la Temperatura

Los valores de temperatura máxima, mínima diaria, datos de las 3 observaciones del termómetro seco y húmedo también son tomados desde las libretas meteorológicas los valores de temperatura máxima deben estar entre 0 y el récord máximo actualmente se está considerando entre 0 y 45 o nulo, la temperatura mínima debe estar entre la mínima récord y 30 actualmente se ingresa entre -9 y 30, la temperatura del termómetro seco y húmedo de las 7 horas entre -9 y 30 o nulo (99.9)y finalmente la temperatura del termómetro seco y húmedo de las 13 horas y 19 horas entre -9 y 45 o nulo (999.9).

En el caso de la temperatura se considera la temperatura media diaria, a través de la Ecuación **(4)**. Esta fórmula considera todos los valores del mismo día, si uno de los valores es nulo, no se puede calcular el valor diario.

$$\text{Temperatura\_media\_diaria} = \frac{\text{Valor07h} + \text{valor13h} + \text{valor19h}}{3} \qquad \textbf{(4)}$$

1.3.4. Cálculo de la Nubosidad Diaria

La nubosidad es considerada a la aparición de una o varias nubes en la atmósfera terrestre, los valores para calcularla de esta deben estar entre 0 y 9, también puede ingresar el 99 equivalente a nulo. Para este cálculo se consideran los valores del mismo día, el valor resultante es el promedio de las 3 lecturas.

$$\text{Nubosidad\_media\_diaria} = (\text{Valor07h} + \text{Valor13h} + \text{Vaor19h}) / 3 \qquad \textbf{(5)}$$

En la Ecuación (5) se muestra la fórmula para el cálculo de la nubosidad media diaria se debe coliderar todos los valores del mismo día, si uno de los valores de nubosidad registrados es nulo, no se debe calcular el valor diario.

1.3.5. Cálculo de la Tensión del vapor (Hp), la Humedad Relativa (%) y el Punto de Rocío (°C)

La tensión de vapor es tendencia al cambio de las moléculas de agua de líquido a gaseoso, para este cálculo se consideran los valores del termómetro seco y húmedo (tanto para las 7:00, 13:00, 19:00). Se puede calcular la tensión de vapor de saturación con respecto a termómetro seco (TVs) y termómetro húmedo (TVh). De acuerdo con la ecuación **(6)**, donde se consideran los parámetros del termómetro seco y húmedo, los cuales son ingresados directamente sin ningún cálculo previo.

$$\text{Tension\_vapor} = TVh - 0.000662 * \text{presión} * (ts - th) * (1 + 0.00115 * th) \quad \textbf{(6)}$$

Por otra parte, la Humedad Relativa es un indicador que proporciona el porcentaje de aire húmedo en el ambiente, verificando si el aire está seco o no.

En la Ecuación **(7)** se muestra la fórmula para la humedad relativa la cual es calculada previamente de la tensión de vapor.

$$\text{hr} = \frac{\text{TensiónVapor}}{\text{TVs}} * 100\% \quad \textbf{(7)}$$

Finalmente, el Punto de Rocío es un indicador en el cual se muestra el porcentaje de temperatura, en el cual el agua empieza a condensarse. El punto de rocío se obtiene usando la Ecuación **¡Error! No se encuentra el origen de la referencia.** siendo e_s la tensión de vapor.

$$\text{Punto de Rocío} = 237.3 * \frac{\ln(e_s) - 1.80842}{19.0778 - \ln(e_s)} \quad \textbf{(8)}$$

1.3.6. Cálculo del Viento

El viento es una respuesta a la presión o la temperatura que se da de forma natural, para su cálculo se debe considerar que sus valores en el anemómetro están en el rango de 1 a 9999999999 y verificar que el valor ingresado sea mayor que el anterior.

Los valores de dirección de viento que piden ser almacenado, se detallan en la Tabla 2 *Simbología de Viento*.

Tabla 2

Simbología de Viento

Simbología	Dirección
'N '	Norte
'NE'=,	Noreste

E '	Este
''SE'	Sureste
'S '	Sur
'W '	Oeste
'SW'	Sur oeste
'NW'	Noroeste
'C '	Calma
'99'	Nulo

$$\text{Recorrido de viento diario} = \text{recviento7hdiasiguiente} - \text{recviento7h} \quad \textbf{(9)}$$

En la Ecuación **¡Error! No se encuentra el origen de la referencia.** se muestra el cálculo del r ecorrido de viento diario, este se obtiene restando el valor de las 7:00 del día siguiente menos el valor del 7:00 del día a calcular.

1.4. Servicios Web

Los servicios web (*web service*) son aplicaciones modulares auto-contenidas que pueden soportar interacciones inter-operables máquina a máquina, utilizando protocolos de transferencia http y https, los cuales son estándares de internet para el intercambio de datos entre distintas aplicaciones, es decir, se puede localizar e invocar a través de una red, que a su vez, ayudan a construir aplicaciones del lado del cliente (*FrontEnd*) independientemente del lenguaje de programación en el que se haya desarrollado el servicio web (Estévez, Lamela, & Morales, 2016).

Desde la perspectiva del negocio, los servicios web permiten a las entidades integrar a sus aplicaciones sin la necesidad de preocuparse de cómo está desarrollada, dónde reside o sobre qué sistema operativo se está ejecutando.

Los estándares más comunes empleados en el servicio web son:

- XML Lenguaje Extensible de Etiquetado, es un formato estandarizado para los datos, definido como un lenguaje marcado para documentos que posee información estructurada.
- SOAP (*Simple Object Access Protocol*) es un protocolo que permite comunicarse intercambiando datos entre varios dispositivos basado en lenguaje XML (Estévez, Lamela, & Morales, 2016).
- WSDL (*Web Services Description Languaje*). Se usa para describir la interfaz pública de servicios web y permite que un servicio y un cliente establezcan la forma de comunicación (requisitos del protocolo y formato de mensajes), a través de un documento procesable por dispositivos (Estévez, Lamela, & Morales, 2016).
- REST (*Representacional State Transfer*) es un estilo arquitectónico recomendado para sistemas distribuidos en la web al momento de realizar una comunicación entre cliente y servidor, debido a que trabaja con el protocolo HTTP donde no se requiere de una capa de transporte extra como lo hace SOAP, los métodos que se utilizan son GET, POST, PUT, DELETE, etc (Estévez, Lamela, & Morales, 2016).

1.5. Interfaz de programación de aplicaciones (API)

Una Interfaz de programación de aplicaciones (API) es un tipo de tecnología usada para la comunicación e intercambio de información entre diferentes aplicaciones, es decir, permite utilizar los servicios web de otras entidades sin la necesidad de conocer su infraestructura, esto facilita la

obtención de información, la cual puede ser utilizada y plasmada en otro software, donde se encargará de interpretarla para así generar un resultado amigable para el usuario (García, 2019).

El acceso a las APIs se establece en función de los usuarios a los cuales esta dirigidos, estas pueden ser:

- Internas o privadas.
- Externas o públicas.
- *Partner*.
- APIs de servicios Web.
- APIs basadas en librerías.

1.6. HERRAMIENTAS DE DESARROLLO

1.6.1. GitHub

GitHub es Herramienta con servicio en la nube que sirve para alojar proyectos de desarrollo de software en colaboración con otras personas, permitiendo tener un control de versiones, donde los colaboradores realizan cambios, este software facilita el seguimiento y avance del producto (aplicación) (Paredes, 2015).

1.6.2. Zenhub

Zenhub es una herramienta para la gestión de tareas y actividades de proyectos que se llevan a cabo en GitHub, en el que se puede notificar de cualquier error, colocando mensajes en las actividades respectivas, como se lo maneja también en Trello, Zenhub realiza un seguimiento y avance de los *commits* que se efectúa directamente en el repositorio donde se está trabajando, de igual manera permite ver en qué estado se encuentra la tarea asignada a los colaboradores (Gutiérrez, 2017).

1.6.3. Postman

Es una herramienta que permite realizar peticiones HTTP y HTTPS sobre cualquier API de una manera muy sencilla, Postman otorga la posibilidad de testear solicitudes a través de una interfaz gráfica de usuario recibiendo y enviando diferentes tipos de respuestas, además, soporta distintos tipos de formato de datos como JSON, XML, HTML. Esta aplicación contiene diferentes métodos para la interacción con los *endpoints* (urls de una API), los más comunes son GET, PUT, POST y DELETE (Postman, 2022).

1.6.4. Motor de base de datos PostgreSQL

PostgreSQL nació a mediados de la década del 1980, iniciado el proyecto por Michael Stonebraker, quien empezó un proyecto con el nombre de Post Ingres que más tarde simplemente se llamaría Postgres. PostgreSQL es un gestor de bases de datos relacional de código abierto, que soporta grandes cantidades de datos, su desarrollo es llevado a cabo por una gran comunidad de desarrolladores que hacen de este sistema una de las opciones más sólidas a nivel de bases de datos, pues permite ejecutar optimizaciones de consultas avanzadas y sobre todo dispone de varios tipos de datos como, por ejemplo, *varchar, integer, datetimezone*, etc. Este software está disponible en más de 34 plataformas y dispone de una versión nativa para Windows (Chávez D. P., 2010).

1.6.5. Python

Es un lenguaje de programación de alto nivel que utiliza el paradigma orientada a objetos, posee una estructura versátil que ayuda a generar un código legible y pueda ser entendido por cualquier persona con unos conocimientos básicos en programación, además, de que tiene una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas, Python soporta el uso de módulos y paquetes, por lo que se puede desarrollar programas modulares, los cuales puedan ser reutilizables en varios proyectos. También, dispone de grandes librerías para

análisis de datos como Pandas, NLTK (*Natural Language ToolKit*) entre otras. Python fue creado por el informático Guido van Rossum quien subió la primera versión de este lenguaje a USENET en febrero de 1991 (Chávez R. D., 2016).

1.6.6. Django

Es un *framework* gratuito de código abierto escrito en el lenguaje Python, está diseñado para desarrollar sitios web de forma fácil y rápida, esto lo hace con la ayuda de diferentes componentes y librerías que simplifican la codificación del software, además maneja la arquitectura Modelo Vista *Template* (MTV). Django dispone de una gran comunidad de programadores experimentados que ofrecen diferentes actualizaciones implementando nuevas funcionalidades (Garcia, 2015).

1.6.7. Mixins

Según (Montilla, 2014) los *mixins* son clases que contienen un conjunto de métodos y atributos que ayudan a simplificar las consultas típicas a los distintos gestores de bases de datos relacionales y no relacionales, a su vez pueden ser implementados en diferentes clases evitando reescribir código, esto se da gracias a que Python permite realizar la herencia múltiple.

1.6.8. Swagger

Es una herramienta de código abierto que sirve para describir, producir, consumir, visualizar y sobre todo brindar un esquema de documentación de una API, esta herramienta crea una plataforma que ordena los distintos métodos (GET, POST, PUT, DELETE) empleados que facilita la visualización de los mismos para que otros desarrolladores puedan realizar peticiones a las APIs con los parámetros correspondientes implementados en dichas funciones, también permite testear la funcionalidad del *backend* y observar los distintos tipos de respuesta enviados por el servidor (Abad, 2017).

1.6.9. React

React.js es una librería JavaScript, que permite separar en componentes las interfaces de usuario orientadas a la web. Fue creado por el ingeniero de Facebook Jordan Walke y se lanzó en mayo de 2013, es una herramienta de código abierto, utilizada por su rendimiento, flexibilidad y su enfoque declarativo, react se acopla fácilmente a diferentes tecnologías y sobre todo al migrar a nuevas versiones. React al utilizar una arquitectura basada en componentes ayuda a aumentar la reutilización de código y facilita al mantenimiento de proyectos a gran escala, además, posee una comunidad con más de 1500 colaboradores en GitHub, esta librería es usada por grandes empresas como Netflix, Instagram, Facebook, WhatsApp, PayPal, Microsoft, la BBC, etc (Julián Adams, Erick Pérez, 2018).

1.6.10. Material UI

Según (Mejías, 2020) es un proyecto *open source* que implementa el lenguaje visual de componentes de React que incorpora *Material Design* de Google, en cambio para (Martin & Ollé, 2020) es una librería de Interfaz de Usuario cuya finalidad es ser utilizada dentro de aplicaciones de React que permiten reutilizar sus componentes siguiendo la doctrina de diseño del motor de búsqueda más utilizado en el mundo, esta herramienta se puede obtener a través de gestores de paquetes npm o yarn.

1.6.11. Plotly

Es una librería de JavaScript *open source* que sirve para la generación de graficas estadísticas, está construida sobre la biblioteca d3.js, cuenta con 20 diferentes tipos de gráficos preconfigurados listos para ser usados según la conveniencia de los programadores, además posee diferentes funcionalidades útiles que permiten la manipulación de la información plasmada en las gráficas generadas (Seville, 2017).

1.6.12. JMeter

Es un software *open source* construido en java usado para medir el rendimiento de diferentes tipos de aplicaciones ya sean de escritorio u orientadas a la web. *JMeter* permite realizar él envió simultaneo de múltiples peticiones para generar una carga pesada en el servidor, esto lo hace con el fin de comprobar cuál es el pico máximo de solicitudes que este soporta (K.K, 2009).

CAPITULO II

2. DISEÑO DE LA APLICACIÓN

2.1. Estado actual del sistema de gestión de la información del INAMHI

El estado actual en el INAMHI para generar reportes estadísticos de las estaciones convencionales se lleva a cabo a través de un software poco intuitivo desarrollado bajo una arquitectura monolítica con una interfaz gráfica poco amigable para el usuario, a su vez este software se encuentra conectado a una base de datos MongoDB, la cual trabaja con datos no estructurados y genera una complicación al momento de la manipulación y obtención de datos, por lo cual, teniendo en cuenta estos antecedentes se procederá al levantamiento de requerimientos para desarrollar un software que se adapte de una mejor manera a las necesidades de la institución.

2.2.Alcance del proyecto

El software que se desarrollará posee un solo perfil de usuario, puesto que, está destinado para el uso exclusivo de los servidores públicos de INAMHI por el motivo que posee información, la cual es exclusiva y a su vez esta solo puede ser manipulada por dicha entidad. El sistema proporcionará los siguientes servicios. La creación del primer módulo deberá generar reportes diarios para la entidad de una manera gráfica, eliminando el paradigma de aplicaciones monolíticas existentes en el INAMHI actualmente.

2.3. Análisis y levantamiento de requerimientos del sistema

En este capítulo se describe el análisis de requerimientos que servirá para el levantamiento de los requerimientos funcionales y no funcionales del sistema. También incluye el análisis de factibilidad (viabilidad técnica, económica y operacional). También incluye los requerimientos funcionales y no funcionales dictados por el INAMHI.

2.3.1. Análisis de factibilidad.

La factibilidad contiene tres aspectos necesarios para el análisis de un proyecto, las cuales ayudaron a la toma de decisión del desarrollo: viabilidad técnica, económica y operacional.

2.3.2. Viabilidad Técnica.

Para la construcción de este proyecto se utilizarán tecnologías de código abierto actuales, las cuales están detalladas en la Tabla 4. Estas tecnologías permitirán acceder a los diferentes servicios web de una manera más rápida, A su vez se utilizarán diferentes recursos de hardware ver Tabla 3**Tabla 4**. El sistema es considerado escalable debido a que es una aplicación modular la cual se puede incorporan con otros sistemas sin ninguna dificultad y así cumplir las diferentes necesidades que requiere el INAMHI.

Tabla 3.

Requisitos del Hardware

Requisitos del hardware			
Dispositivo	Cantidad	Uso	Descripción
Laptop Dell Inspirion G3 3579	1	Desarrollo, investigación, pruebas y documentación	-Procesador: Intel Core-I7. - Memoria: 16 GB -Discos: Solido 120GB y 1TB.

			- Arquitectura: 64 Bits.
Monitor	2	Desarrollo, investigación, pruebas y documentación	Monitor Samsung de 20”.
CPU	1	Desarrollo, investigación, pruebas y documentación	-Procesador: Intel Core-I5. - Memoria: 8 GB -Discos: 1TB. - Arquitectura: 64 Bits.
Router	2	Red de conexión a internet.	-Velocidad: 60 MB

Nota: *Requisitos de hardware necesarios para el desarrollo del sistema web. Elaborado por: (Apolo & Taco, 2022).*

Tabla 4.

Requisitos del Software

Requisitos del software			
Producto	Cantidad	Uso	Descripción
Windows 10	2	Sistema operativo	-licencia: Educativo. -Año: 2021.
PostgreSQL	1	Software de Base de Datos	-Licencia: GPL -Versión: 14
Python	1	Lenguaje de programación	-Licencia: GPL -Versión: 3
Django	1	*Framework* de desarrollo de Python	-Licencia: GPL -Versión: 3

Visual Studio	1	IDE de desarrollo	-Licencia: GPL
Git Hub	1	Repositorio de código y versionamiento.	-Licencia: GPL.
Google Chrome	1	Explorador web	-Licencia: GPL
Zoom	1	Herramienta para reuniones virtuales.	-Licencia: Gratuita
Microsoft Office	1	Herramienta para documentación	-Licencia: Educativo

***Nota**: Requisitos del software para el desarrollo del sistema web. Elaborado por: (Apolo & Taco, 2022).*

2.3.3. Viabilidad Económica.

Al haber realizado un preanálisis por el INAMHI se logró obtener los costos que se ocuparan en la construcción del aplicativo ver Tabla 5, observando los costos se puede deducir que el proyecto es rentable y escalable. El valor de los recursos (hardware) está cubierto debido a que los equipos se adquirieron con anterioridad y los softwares mencionados en la Tabla 4 son *open source* a excepción del sistema operativo Windows 10, que de igual manera está bajo una licencia educativa. Los costos de acceso al servicio de Internet son contratados por los autores con pago mensual y está cubiertos por los mismos.

Tabla 5.

Costos considerados para el desarrollo del proyecto

Costos de software				
Recurso	Descripción	Cantidad	Precio unitario	Total.
Sistema Operativo	Windows 10	2	$16.90	$33.8
Office	Microsoft Office	2	$67	$134

Lenguajes de Programación	*Open Source*	1	$0	$0
Github	*Open Source*	1	$0	$0
PostgreSQL	*Open Source*	1	$0	$0
Visual Studio	*Open Source*	2	$0	$0
Costos de Hardware				
Laptop DELL	-Core I7 -RAM: 16GB	1	$1250	$1250
CPU	-Core I5 -Ram: 8GB	1	$900	$900
Monitores	Monitor Samsung Monitor LG	2	$100	$200
Costos de servicios				
Servicio de Internet	Internet de 40MB	1	$35	$35
Servicio de Internet	Internet de 40MB	1	$39	$39
TOTAL				$2591.80

***Nota**: Costos de equipos, programas y servicios para la construcción del proyecto*

Elaborado por: (Apolo & Taco, 2022)

2.3.4. Viabilidad Operacional.

En la actualidad la institución cuenta con un sistema poco amigable e intuitivo para la visualización del reporte de datos, el aplicativo está desarrollado en una arquitectura monolítica que dificulta el soporte, mantenimiento y actualización del software. La implementación de la API solucionará las dificultades mencionadas anteriormente, ya que su integración será construida con los *frameworks* Django y React que ayudaran al desarrollo de un sitio web basados en los principios de diseño de

usabilidad correspondiente al FrontEnd. El objetivo de desarrollar con una arquitectura de microservicio es poder adaptar múltiples módulos y que estos puedan ser usador por diferentes desarrolladoras, esto a su vez facilita el mantenimiento, ya que en un futuro se podrán agregar nuevos requerimientos sin la necesidad de parar todo el sistema. Las personas que trabajan en la institución son profesionales de distintas ramas como las BDD, *BackEnd* y *FrotEnd*, los cuales ayudaran a la implementación de la API separando su modelo de negocio con la experiencia de usuario, es decir, cada especialista se encargara del desarrollo (funciones nuevas) por separado indistintamente del lenguaje o *framework* de su preferencia.

2.3.5. Características de usuarios.

En la Tabla 6 se describe las diferentes características de los usuarios que utilizarán el sistema, esta descripción se la lleva a cabo para conocer el nivel de experticia en el manejo del sistema, de cada uno de los perfiles de usuario y así poder adaptar el software para que sea amigable y usable.

Tabla 6.

Descripción de usuarios

Tipo de usuario	Características	Descripción
Servidores públicos INAMHI	Formación	- Usuarios con nivel de estudios superiores que laboran en la institución pública, dedica a la visualización de datos de las estaciones hidrometereológicas convencionales.
	Habilidades	- Visualizar la información de las estaciones convencionales e interpretarla.

		- Informar acerca de los datos pocos comunes ingresados en el sistema por el departamento de información, de las estaciones convencionales.
	Actividades	-Generar reportes diarios y mensuales de las estaciones convencionales.

Nota: *Descripción de los usuarios que usaran el sistema web. Elaborado por: Juan Apolo y Byron Taco*

2.3.6. Restricciones del sistema

El usuario solamente podrá visualizar los datos de manera gráfica mas no modificar, debido a que las modificaciones la realizan el departamento de información del INAMHI.

El usuario simplemente tendrá acceso al servicio API para la lectura de datos que serán devueltos en formato JSON, el usuario también podrá generar los reportes en formatos: csv y pdf.

2.3.7. Requerimientos funcionales y no funcionales

Para el levantamiento de los requerimientos funcionales y no funcionales se hace uso de la herramienta de diagramado de matrices en las cuales se describirán todos los componentes, servicios y restricciones que se prestarán en el sistema. En Tabla 7 se puede visualizar el modelo de matriz que se usará para la descripción de dichos requerimientos.

Tabla 7.

Modelo de matriz del levantamiento de los requerimientos

Sistema Web Especificación de Requerimientos		
Código	**Nombre**	**Grado de necesidad**
Referencia de Requerimiento	Nombre del requerimiento	Importancia del requerimiento

Descripción	Descripción del requerimiento			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Entradas del Requerimiento	Fuentes de las entradas	Salidas del requerimiento	Donde se lleva la salida	Restricciones a tener en cuenta
Proceso	Descripción detallada de las actividades que realiza el requerimiento.			
Efecto colateral	Efectos generados a otros proceso o sistemas, si es el caso			

***Nota**: Este es el modelo matriz que se usara para la descripción de los diferentes requerimientos*

Elaborado por: (Apolo & Taco, 2022)

2.4.3.1 Requerimientos funcionales.

Requisito funcional 01.

En la Tabla 8 se detalla el primer requerimiento funcional el cual consiste en que el sistema a crear podrá ser accedido en todos los navegadores de los diferentes dispositivos como computadoras, celulares y tablets que estén dentro de la red local del departamento de información del INAMHI.

Tabla 8.

Descripción de las Apertura de las urls.

Sistema Web Especificación de Requerimientos Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RF01	Apertura a las URL´s locales		Alto	
Descripción	Se colocarán las respectivas url´s para obtener la información			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Dirección URL del servidor	*Network.* Software de cliente HTTP requests	Software de cliente HTTP *requests*	Software de cliente HTTP	El acceso será para servidores públicos del INAMHI
Proceso	El sistema por el momento contendrá la dirección local, pero en lo posterior podrá acceder con el nombre de un dominio en la internet.			

Efecto colateral	Información solo para servidores públicos del INAMHI.

***Nota**: Identificación de urls para las peticiones HTTP. Elaborado por: (Apolo & Taco, 2022)*

Requisito funcional 02

Como siguiente requerimiento funcional el software deberá ser capaz de permitir el ingreso de datos históricos de las estaciones convencionales hidrometeorológicas, así como también permitirá la actualización eliminación y la visualización de dichos datos en una interfaz amigable para el usuario. En la Tabla 9 se puede observar a detalle el segundo requerimiento funcional.

Tabla 9.

Peticiones de tipos REST.

Sistema Web Especificación de Requerimientos Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RF 02	Peticiones de tipos *REST*.		Alto-medio	
Descripción	El servidor permitirá peticiones de tipo *GET, POST* y *PUT*.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Identificación del tipo de petición.	Software de cliente HTTP *requests*	Confirmación en la pantalla	Software de cliente HTTP (POSTMAN)	Conocimiento del software a usar para consumir las API's e identificación del tipo de petición al server.
Proceso	El Usuario solamente tendrá la opción para visualizar los datos que esta publicada como un servicio, pero para pruebas es necesario realizar peticiones de tipo select, update, insert y delete.			
Efecto colateral	No aplica			

***Nota:** Se describe las peticiones tipo rest hacia el api. Elaborado por: (Apolo & Taco, 2022)*

Requisito funcional 03

El programa deberá ser intuitivo y amigable con todos los usuarios que vaya a utilizarlo permitiéndoles acoplarse de una manera rápida a las diferentes interfaces como la de ingreso de datos, la generación de graficas de datos estadísticos, etc, esto permitirá poner en producción el software lo más pronto posible una vez se haya finalizado. En la (Tabla 10) se describe de manera detalla este requerimiento.

Tabla 10.

Métricas requeridas

Sistema Web Especificación de Requerimientos Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RF 03	Vínculos de Navegación		Medio	
Descripción	El sistema debe contener usabilidad y navegabilidad para facilidad del usuario.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Textos e iconos	Hojas de estilo	Vinculo entendible	Usuario	Los vínculos deben ser imágenes con texto agradable al usuario.
Proceso	No aplica			
Efecto colateral	El usuario podrá navegar de una manera fácil y sencilla.			

Nota: Vínculos para la navegación en el módulo de reportes. Elaborado por: (Apolo & Taco, 2022)

Requisito funcional 04

En la Tabla 11 se describe como último requerimiento el INAMHI ha pedido que el programa deberá contener múltiples opciones, en las cuales se puedan mostrar reportes y gráficos estadísticos,

dependiendo de lo que requieran los usuarios. En la Tabla 11 se describe el ultimo requerimiento funcional solicitado por el INAMHI.

Tabla 11.

Requisito funcional 4

Sistema Web Especificación de Requerimientos Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RF 04	Items para generar reportes		Alto-Medio	
Descripción	El sistema debe contener un *dashboard* el cual contenga las opciones para generar reportes de las estaciones convencionales en formatos csv y pdf.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Items	*Dashboard*	Navegador	Carpetas de descargas	El formato de los archivos son xlsx, csv y pdf.
Proceso	Los ítems estarán situados en el *dashboard* para poder generar los reportes pertinentes.			
Efecto colateral	Ninguna			

***Nota**: RF para los items de la generación de los reportes. Elaborado por: (Apolo & Taco, 2022)*

2.4.3.2 Requerimientos no funcionales.

Requisito no funcional 01

En la Tabla 12 se detallan la compatibilidad que deberá llevar el software con los diferentes sistemas operativos existentes, aunque, como es un aplicativo orientado a la web, no será necesario tomaran en cuenta los navegadores que son compatibles ya que React se encarga de aquello, en los que podrá correr la aplicación sin ningún inconveniente de compatibilidad.

Tabla 12.

Compatibilidad de Sistemas Operativos

Sistema Web Especificación de Requerimientos NO Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RNF 01	Sistemas Operativos		Medio	
Descripción	El aplicativo podrá interactuar con cualquier sistema operativo simplemente con la ayuda de un navegador compatible con esta para así consumir de los servicios, también se puede hacer uso de un software como el Postman para probar el API.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Navegador	No aplica	No aplica	No aplica	No aplica
Proceso	La aplicación deberá de ser independientemente del SO que usa			
Efecto colateral	No aplica			

***Nota**: RNF Compatibilidad para el uso del sistema web. Elaborado por: (Apolo & Taco, 2022)*

Requisito no funcional 02

En la Tabla 13 se describe la disponibilidad del sistema de una manera detalla en la que el usuario puede hacer uso del servicio web.

Tabla 13

Disponibilidad de la aplicación

Sistema Web Especificación de Requerimientos NO Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RNF 02	Disponibilidad		Alto	
Descripción	El sistema debe estar disponible 24/7 para el usuario pueda hacer uso de los servicios en el momento que este lo requiera.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**

Infraestructura	Arquitectura del sistema	No aplica	No aplica	Infraestructura adecuada.
Proceso	El usuario podrá ingresar y consumir los servicios en cualquier momento.			
Efecto colateral	Los administradores deben alojarlo en un servidor donde exista disponibilidad continua de los mismos.			

Nota: *Condiciones para la disponibilidad del servicio API**.*** *Elaborado por: (Apolo & Taco, 2022)*

Requisito no funcional 03

En la Tabla 14 se describe la arquitectura requerida que se debe de implementar para lograr realizar mejoras en el software permitiendo escalar nuevas necesidades haciendo mantenible a la aplicación.

Tabla 14.

Requisito de arquitectura a ser usado.

Sistema Web Especificación de Requerimientos NO Funcionales				
Código	**Nombre**		**Grado de necesidad**	
RNF 03	Modularidad		Alto	
Descripción	La arquitectura de la API debe ser modular, que pueda brindar escalabilidad y flexibilidad ante cualquier requerimiento posterior.			
Entradas	**Fuentes**	**Salidas**	**Destino**	**Restricciones**
Código	Código fuente	No aplica	No aplica	Versionamiento del código fuente
Proceso	Modificación y mejorar ante la última versión del software.			
Efecto colateral	No aplica			

Nota: *Arquitectura a ser implementada en la API. Elaborado por: (Apolo & Taco, 2022)*

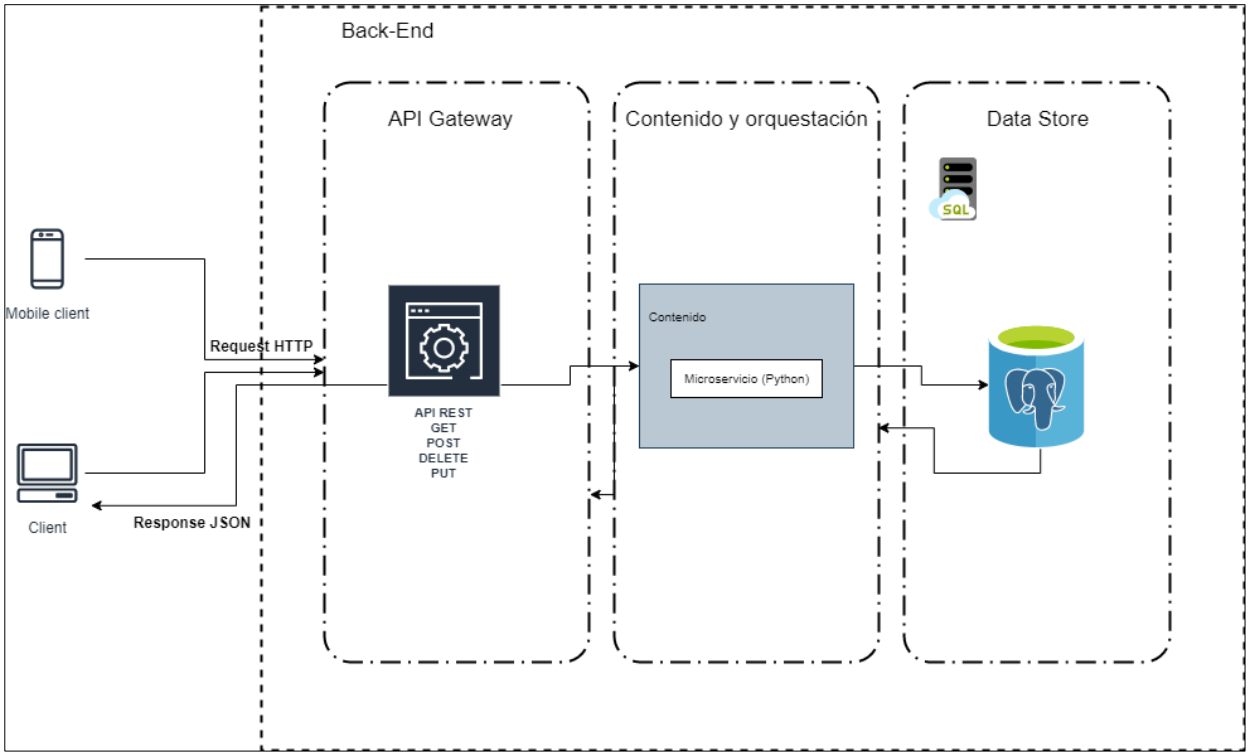
2.4. Diagramas de desarrollo de la aplicación

2.4.1. Diagrama general BackEnd

En la Figura 2 se observa el diseño de la arquitectura del *backend* que debe contener un API, donde se detalla 3 niveles que son requeridos para realizar una petición de tipo HTTP *request,* la cual es solicitada por medio del cliente (*frontend)* a través de los endpoints, que se conecta al contenido y esto, a su vez a la base de datos, retornando la información solicitada en un formato de tipo JSON y esta es proyectada hacia el usuario final de manera gráfica.

Figura 2.

Diagrama general Back-End



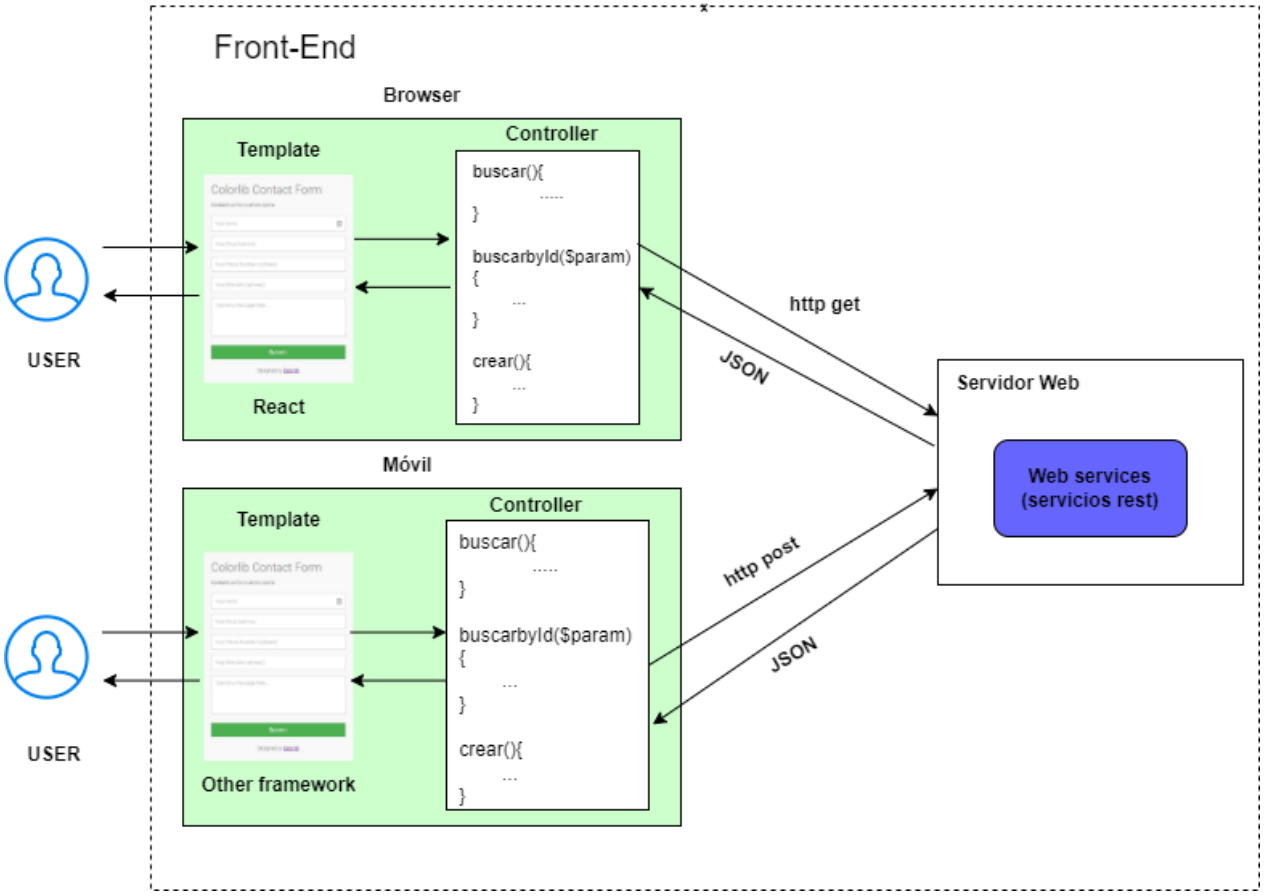
Nota: *Arquitectura de diseño de back-end de un API. Elaborado por: (Apolo & Taco, 2022)*

2.4.2. Diagrama FrontEnd

En la Figura 3 se puede observar la comunicación entre el *frontend* y el *backend*, donde el usuario puede realizar diferentes peticiones, como el envío de datos a través de formularios y también obtener la información que se encuentra almacena en una base de datos indistintamente del tipo de formato (xml, html, json) que esté implementado en un servicio web.

Figura 3.

Diagrama de FrontEnd



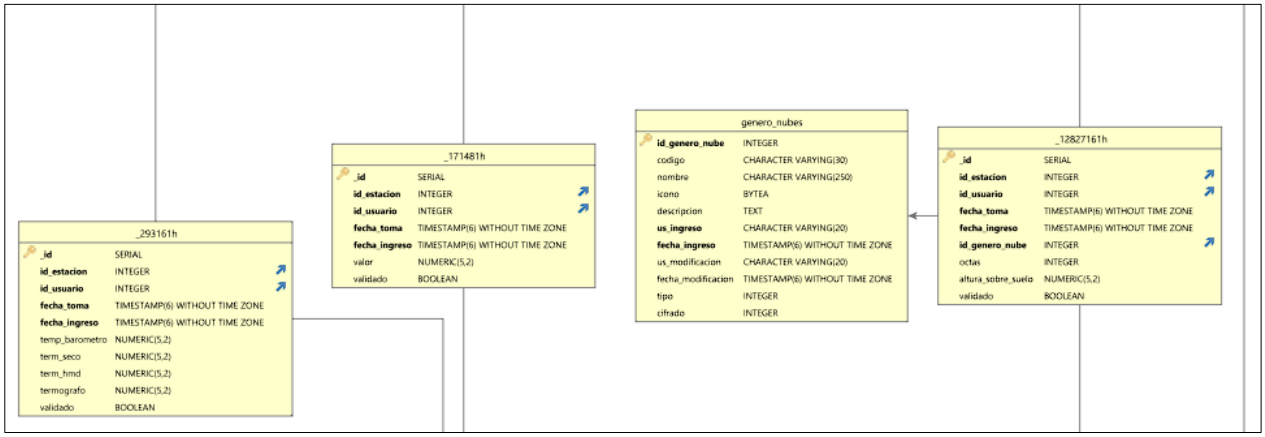
Nota: *Arquitectura de FrontEnd (Interfaz gráfica de usuario). Elaborado por: (Apolo & Taco, 2022)*

2.4.3. Diagramas Entidad Relación

El modelo entidad relación fue heredado por los servidores públicos del INAMHI (Figura 4), el cual fue migrado desde el motor de base de datos MongoDB a Postgres, puesto que este es un gestor de base de datos relacional con el cual se puede llevar un mejor manejo de datos y así adaptarse mejor a las necesidades que posee dicha institución. En la Figura 4 se muestra el diagrama entidad relación en escala reducida por confidencialidad no se puede mostrar todo el contenido del modelo entidad-relación de la base de datos.

Figura 4.

Diagrama de modelo entidad-relación



Nota: *Modelado de base de datos de estaciones convencionales. Elaborado por: (Apolo & Taco, 2022)*

2.4.4. Diagramas de Casos de USO

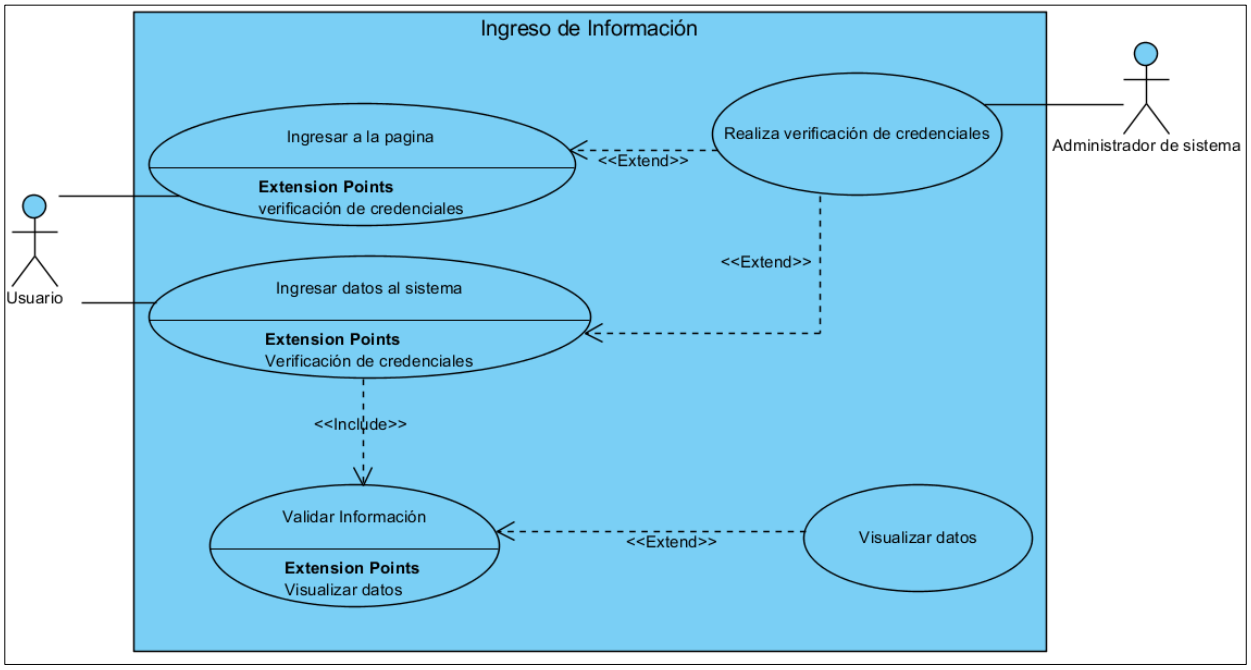
Para el desarrollo de los módulos se ha considerado un solo actor como un usuario sin privilegios, en la cual interactúa con el sistema para el ingreso de datos de las estaciones convencionales y para visualizar dicha información de manera gráfica.

2.4.5. Caso de uso de Ingreso de datos

En la Figura 5 se describe las distintas opciones que ofrece el sistema al usuario, una de ella es el ingreso a la aplicación para la visualización de la información hidrometeorológica que se encuentra almacenada en la base de datos, otro escenario que se observa es el ingreso de datos al sistema el cual es independiente del módulo de validación que se encuentra implementado en el servidor.

Figura 5.

Caso de uso de Ingreso de datos



***Nota**: diagrama caso de uso para el ingreso de datos obtenido de las estaciones convencionales.*

Elaborado por: (Apolo & Taco, 2022)

2.4.6. Caso de uso de Visualización de información

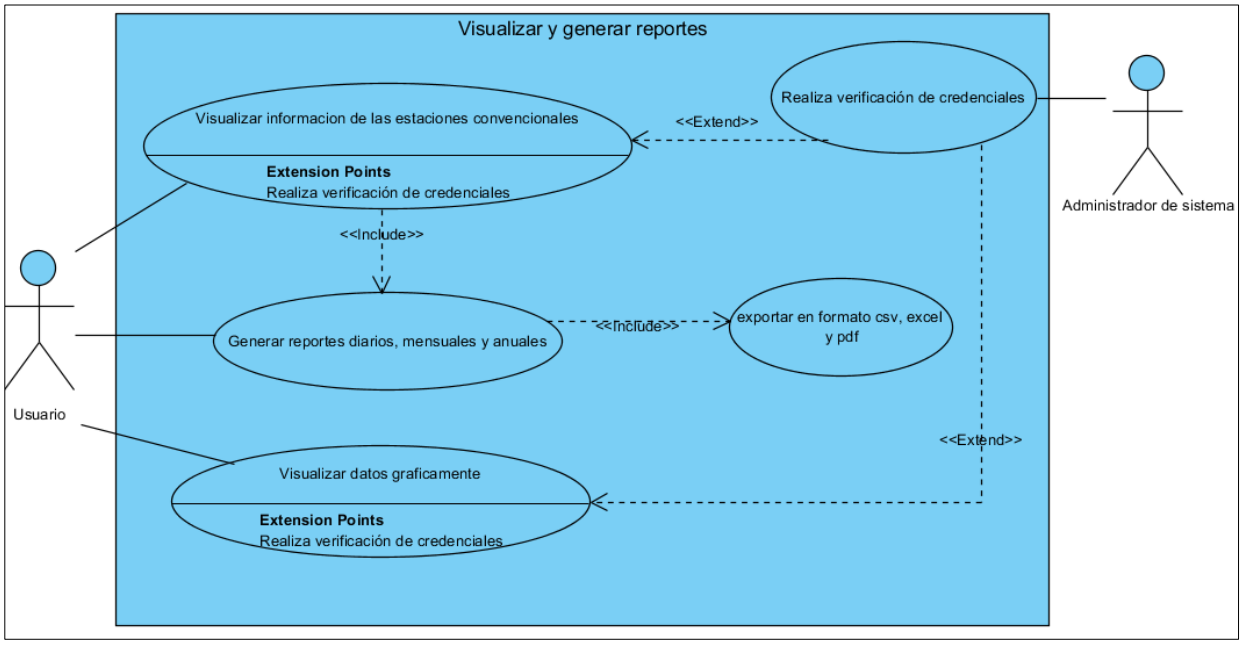
En la Figura 6 se puede observar las diferentes interacciones que se podrá llevar a cabo en el sistema web, las cuales son:

- Visualizar Información: Ver de manera gráfica o en tablas los datos ingresados anteriormente por el usuario.

- Exportar Información: Obtener la información en un cierto formato seleccionado por el usuario.

Figura 6.

Caso de uso de Visualización de información



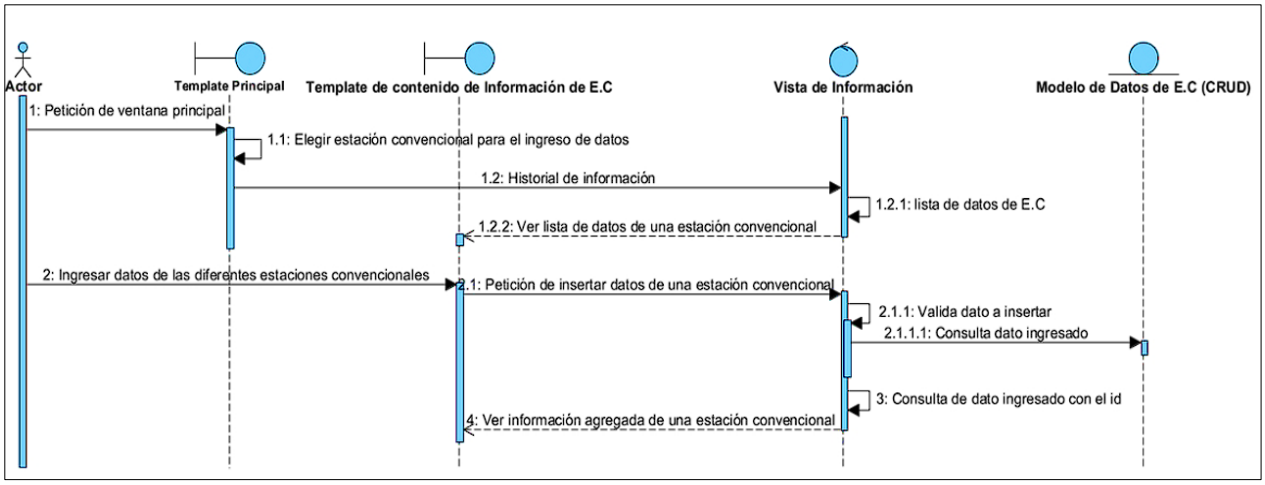
***Nota**: diagrama de caso de uso para visualización y exportación de la información de las estaciones convencionales. Elaborado por: (Apolo & Taco, 2022)*

2.4.7. Diagramas de secuencia (ingreso de datos)

En la Figura 7 se puede observar el diagrama de secuencia para el ingreso de datos, donde se describe la interacción del usuario final con la ventana principal del aplicativo, a su vez se detalla los pasos que lleva a cabo el servidor para insertar y validar la información antes de ser ingresa en el sistema retornando diferentes mensajes dependiendo si la ejecución culmino con éxito o no.

Figura 7.

Diagramas de secuencia (ingreso de datos)



Nota: *Diagrama de secuencia para el ingreso de datos de las estaciones convencionales.*

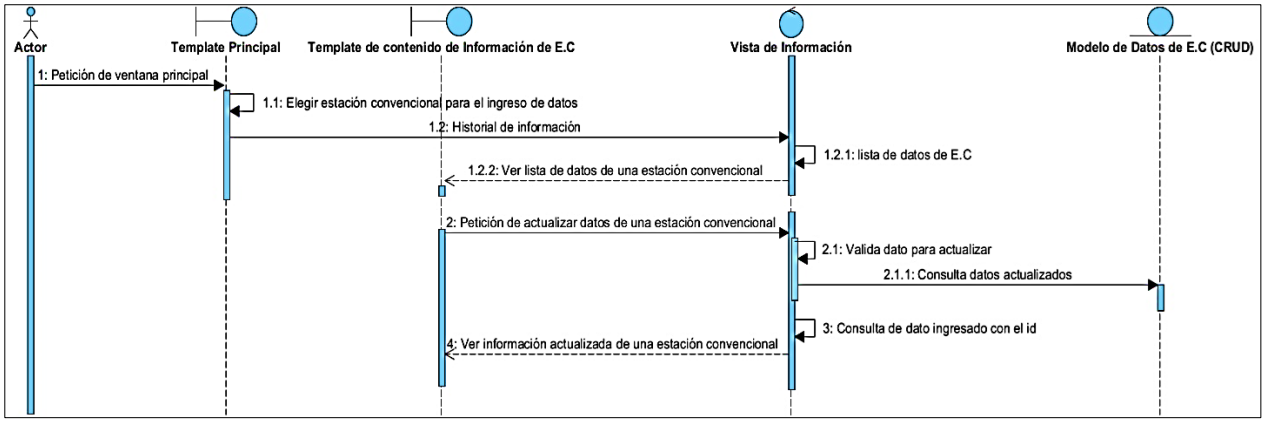
Elaborado por: (Apolo & Taco, 2022)

2.4.8. Diagrama de secuencia (actualización de datos)

La Figura 8 representa el diagrama de secuencia para la actualización de información de las estaciones convencionales del INAMHI, que se puede mirar en la interfaz gráfica(navegador), donde se puede seleccionar, editar y enviar los datos hacia el servicio web, el cual internamente los validad en el controlador (*view*) del servidor (*backend*).

Figura 8.

Diagrama de secuencia (actualización de datos)



Nota: *Diagrama de secuencia para la actualización de datos de las estaciones convencionales*

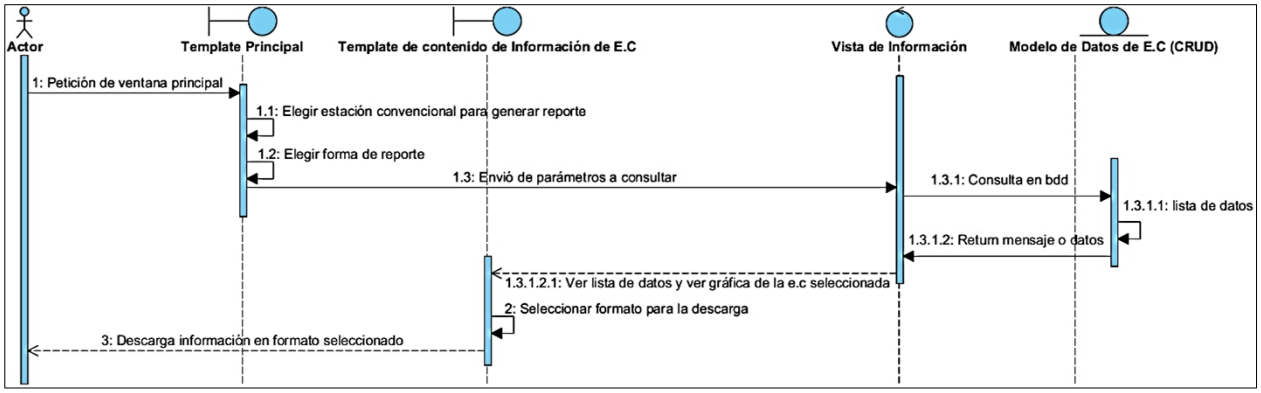
Elaborado por: (Apolo & Taco, 2022)

2.4.9. Diagrama de secuencia (visualizar gráficas y exportar datos)

En la Figura 9 se visualiza el proceso que realiza el usuario para enviar peticiones al servidor, el cual retorna la respectiva información requerida para la visualización de manera gráfica o ha manera de tabla de reporte, esto dependerá de lo que el usuario requiera, a su vez se podrá llevar a cabo la exportación de datos en formatos csv o pdf.

Figura 9.

Diagrama de secuencia (visualizar gráficas y exportar datos)



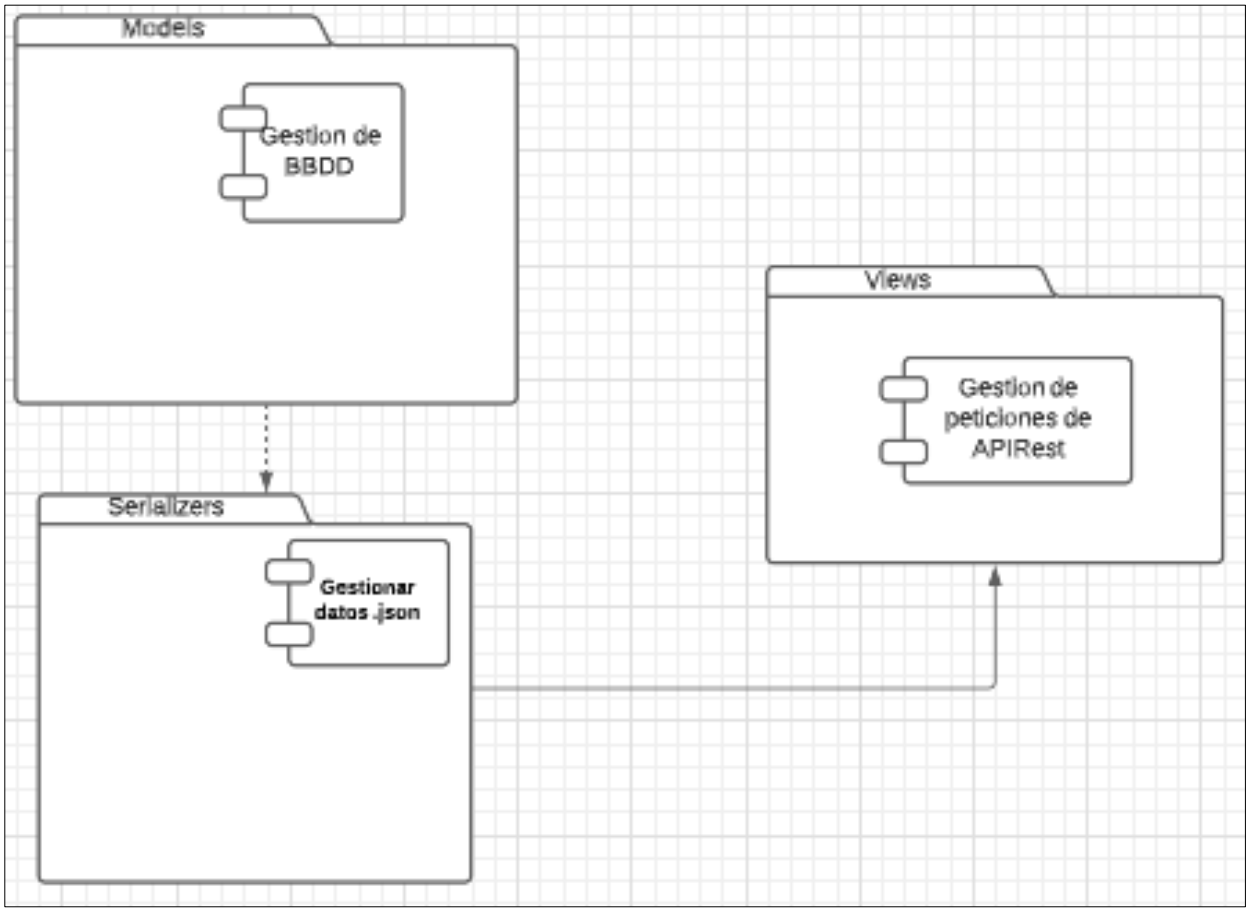
Nota: *Diagrama de secuencia para la visualización gráfica y exportación de datos de las estaciones convencionales. Elaborado por: (Apolo & Taco, 2022)*

2.4.10. Diagrama de componentes

En el diagrama mostrado en la Figura 10 se muestran los distintos componentes del sistema, los cuales se comunican entre sí, como primera instancia se tiene a los modelos, cuya función es llevar la gestión de la base de datos, como segundo componente están los *serializables,* los cuales son los encargados de trasformar la información receptada por los modelos y transformada a un formato de datos general en este caso JSON, como último componente están las *views* la cuales generan las diferentes consultas SQL a la base de datos.

Figura 10.

Diagrama de componentes



Nota: *Diagrama de componentes de la interacción entre los models, serializers y las views.*

Elaborado por: (Apolo & Taco, 2022)

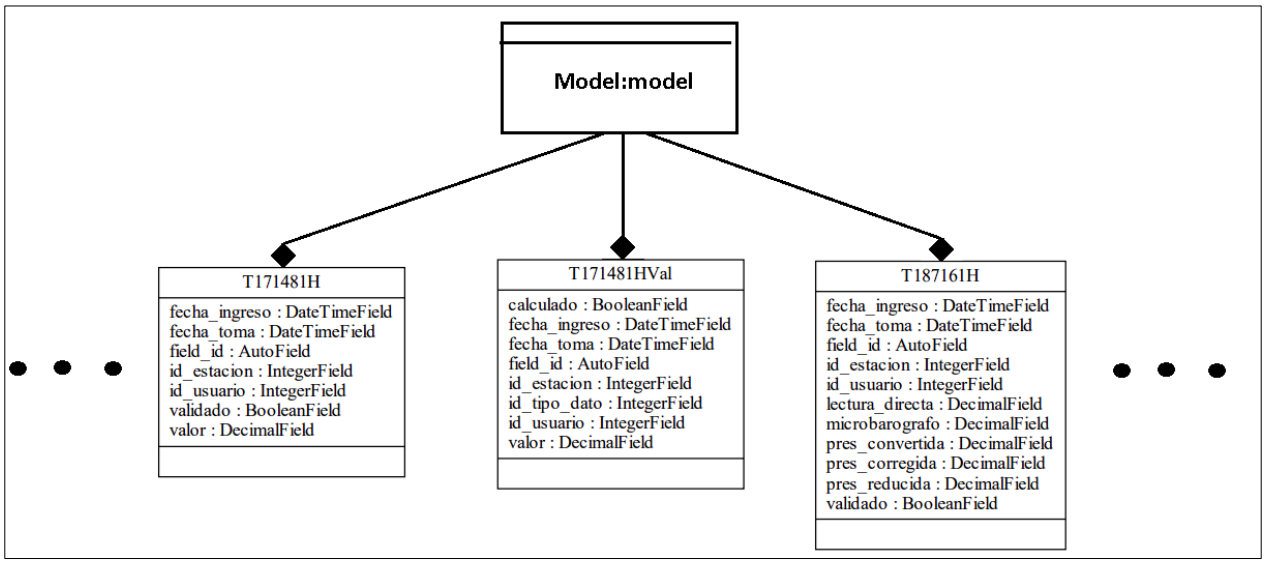
2.4.11. Diagrama de clases

El diagrama de clases está dividido en 3 figuras diferentes, las cuales se las detallaran a continuación.

En la Figura 11 se visualiza un fragmento del diagrama de clases, en este se describen los modelos con sus respectivos atributos, también se observa como estos interactúan con las distintas clases meta, heredadas directamente de Django, las cuales internamente proveen diferentes herramientas que ayudan con la comunicación directa de la base de datos

Figura 11.

Diagrama de los modelos

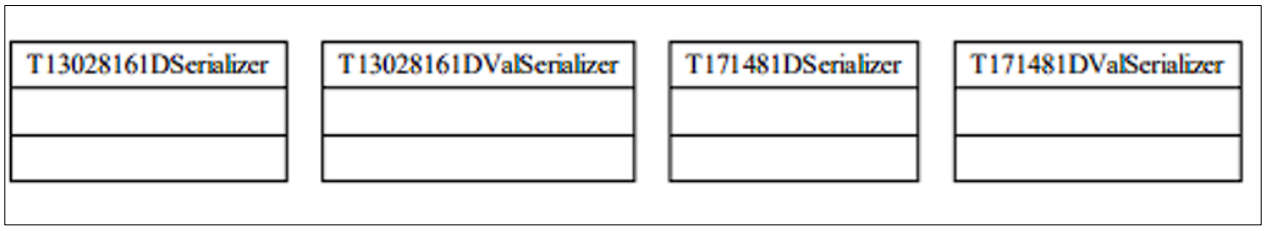


Nota: *Diagrama de clases de los modelos contenidos en el API. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 12 se observa algunas de las clases *serializables*, estas se encuentran aisladas de todo el diagrama, debido a que estos componentes son independientes y su única función es recibir la información limpian traída por los modelos y transformarla al formato nativo de Python para después utilizarla normalmente como se lo maneja en condigo del mismo lenguaje.

Figura 12.

Diagrama de clases serielizables.

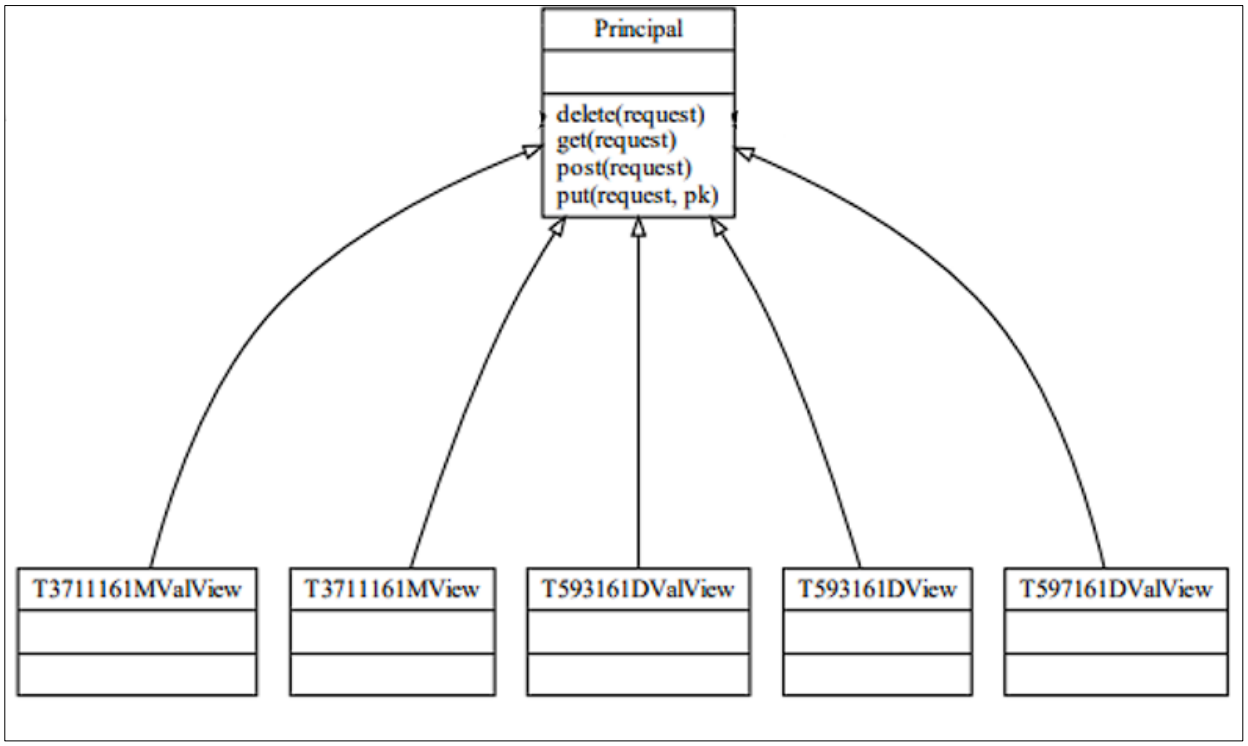


Nota: *Diagrama de clases de los serializers contenidos en el API. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 13 se puede observar algunas de las muchas vistas que posee el software, las cuales están conectadas hacia una clase principal, esto se da por el motivo de que todos los componentes están heredando los distintos métodos (get, post, put, delete) contenidos en dicha clase, esto ayuda a una mejor organización de código ya que lo hace más corto y legible para los programadores.

Figura 13.

Diagrama de clases conectada a la principal.



Nota: *Diagrama de clases que heredan todas de la clase Principal que contiene las views del API*

Elaborado por: (Apolo & Taco, 2022)

CAPITULO III

3. DESARROLLO DE LA APLICACIÓN

En este capítulo se describe la construcción y desarrollo de la API, y también de la parte visual conocida como el lado del cliente (*FrontEnd*) para las cuales, anteriormente se realizó el análisis

de requisitos establecidos en los capítulos anteriores que ayudó a la construcción del *product backlog* que se describe en cada uno de los *Sprint* a lo largo de esta sección.

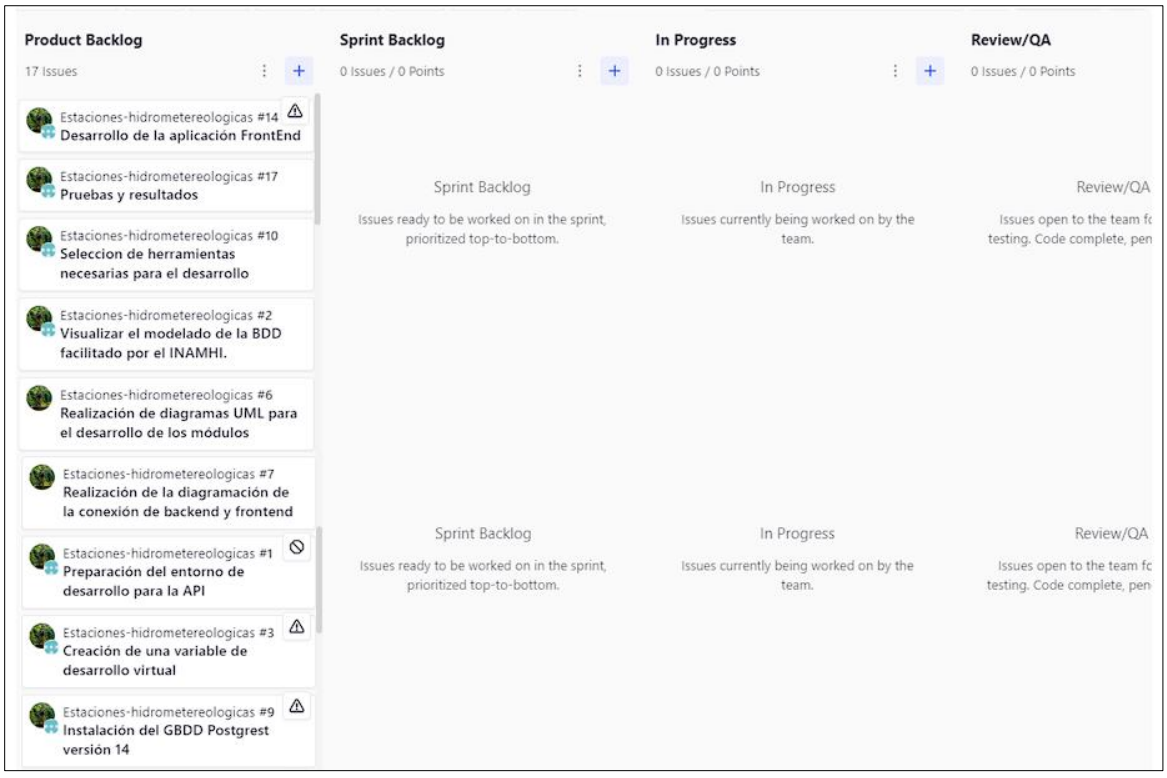
Para la construcción del proyecto web se utilizó la arquitectura Modelo Vista *Template* (MVT) juntamente con la ayuda de los *frameworks* Django para el desarrollo del microservicio y React que comprende la interacción con el usuario final.

3.1. Product Backlog

El marco de trabajo SCRUM recomienda listar las actividades obtenidas del análisis de requerimiento y colocarlas en el *product backlog* como se puede observar en la Figura 14, la cual, permite realizar Sprint de estas tareas, llevando una mejor gestión de las tareas para las entregas incremental continuas del software, de igual manera, para la coordinación de los *Sprint* se utilizó la herramienta ZenHub.

Figura 14.

Lista de actividades (producto Backlog)



Nota: *Listado de las actividades establecidas en el producto backlog para la realización de los Sprint. Elaborado por: (Apolo & Taco, 2022)*

3.2. Sprint 1 Selección de herramientas para el desarrollo de la API

En el *Sprint* 1 se realiza la selección de herramientas para el desarrollo de la API y de la interfaz de usuario para la web (Figura *15*) en base a las distintas características que ofrecen estas, donde los desarrolladores se sientan cómodos al trabajar. Estas herramientas seleccionadas ofrecen flexibilidad, a parte, son aplicaciones de código abierto, de las cuales se listan a continuación:

- VS code e Atom. IDE utilizada para desarrollo del servicio web.
- Postman. Herramienta que permite realizar las peticiones a los *endpoints*.
- PgAdmin4. Software para la interacción y visualización directa con datos de Postgres.
- DbVisualizer. Aplicación que sirve para visualizar diagramas del modelo entidad relación.
- ZenHub. Software que ayuda a gestionar las actividades de usuarios, además, permite integrar con la herramienta de control de versiones GitHub.
- GitHub. Aplicación utilizada para almacenar código desarrollado, el cual, también permite realizar versiones de softwares.

Figura 15.

Selección de herramientas (sprint1)

Nota: *Contenidos de subtareas del sprint 1. Elaborado por: (Apolo & Taco, 2022)*

3.3. Sprint 2 Visualización del modelado de la base de datos

En la Figura 16 se muestra el *sprint* 2 referente a la documentación y el modelado de la base de datos. Esta documentación fue facilitada por la institución para tener una mayor comprensión de los requerimientos necesarios para llevar a cabo el proyecto. El modelado contribuyó a la construcción de la API, debido a que la mayoría de las tablas manejan los mismos tipos de datos, es decir, eran repetitivos. La revisión de la documentación también contribuyó a la utilización de clases genéricas de Django, lo que, facilitó la construcción de una clase principal, de la cual heredan las demás clases.

Figura 16.

Revisión de documentos del INAMHI Sprint 2

Nota: *Actividad a realizar en el Sprint 2 (Visualización del modelado de la base de datos)*

Elaborado por: (Apolo & Taco, 2022)

3.4. Sprint 3 Desarrollo de diagramas UML

En el *sprint* 3 (Figura 17) se realizó los respectivos diagramas antes de la construcción y desarrollo de los módulos, para el cual se utilizó Visual Paradigm UML y Draw.io para la representación gráfica del consumo de las API's hasta la entrega de información requerida por el usuario, teniendo así una manera más clara y concisa panorámica de la creación del software.

En los diagramas que se pueden visualizar en la Figura 2 se puede observar las conexiones de la petición a la API pasando por el microservicio desarrollado en Python, que, a su vez se comunica con la base de datos y este retorna el resultado en un formato JSON. En la Figura 3 se observa mediante la interfaz gráfica de usuario las peticiones a la API que pasan mediante una *APIServices* (*Controller front*) conectando al microservicio que devuelve los datos sugeridos, mostrando de una manera amigable al usuario.

Figura 17.

Desarrollo de diagramas UML sprint 3

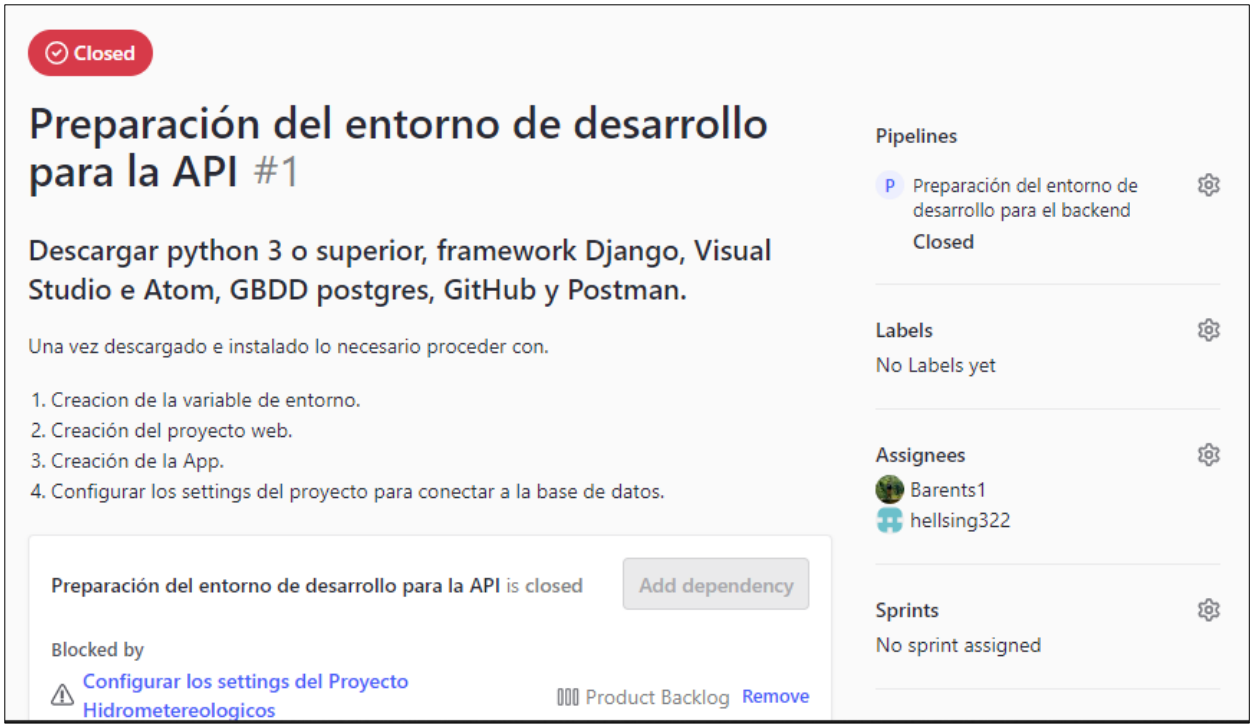
Nota: *Tabla de contenido de las actividades desarrollada en el sprint 3. Elaborado por: (Apolo & Taco, 2022)*

3.5. Sprint 4 Instalación de aplicaciones para el backend

El *sprint* 4 tuvo como objetivo instalar todo lo necesario para el desarrollo del sistema web, donde posteriormente se creó el proyecto y la aplicación con los comandos respectivos, después de haber ejecutado con éxito los comandos se procedió a configurar el archivo settings.py como se muestra en la Figura 19.

Figura 18.

Instalación de aplicaciones para el backend, Sprint 4



Nota: *Tabla de contenido de las actividades del sprint 3. Elaborado por: (Apolo & Taco, 2022)*

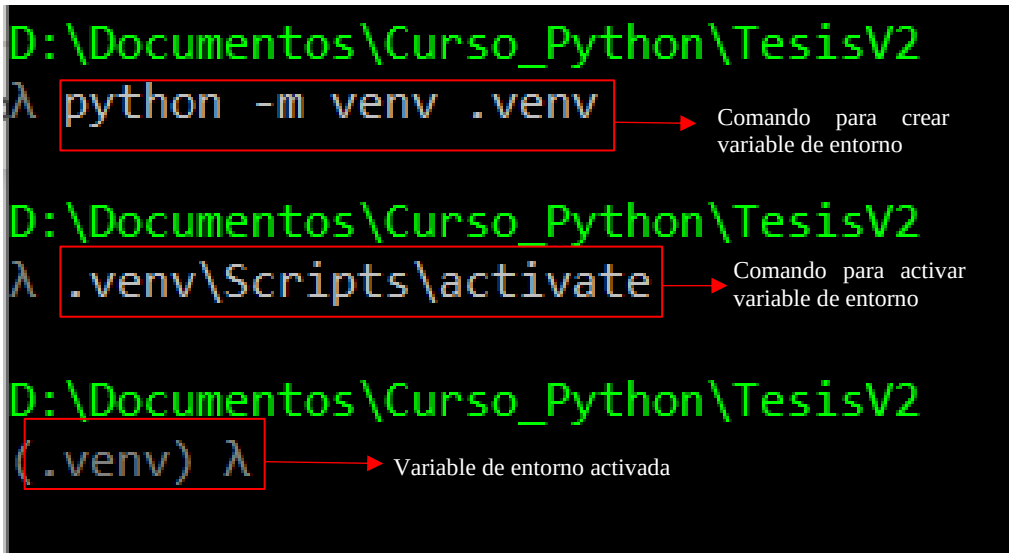
1. Instalación de software para el desarrollo.
    a. IDE de desarrollo, GBDD, GitHub, Python3 y Postman.
2. Creación de la variable de entorno y activarla.

    Creación del entorno de desarrollo para evitar el conflicto con la versión de Python2.7, posteriormente se activa la misma, para la creación del proyecto.

En la Figura 19 se observa los comandos y el procedimiento que se deben llevar a cabo para instalar y activar la variable de entorno.

Figura 19.

Comandos para crear la variable de entorno.



Nota: *Comandos para crear y activar la variable de entorno*

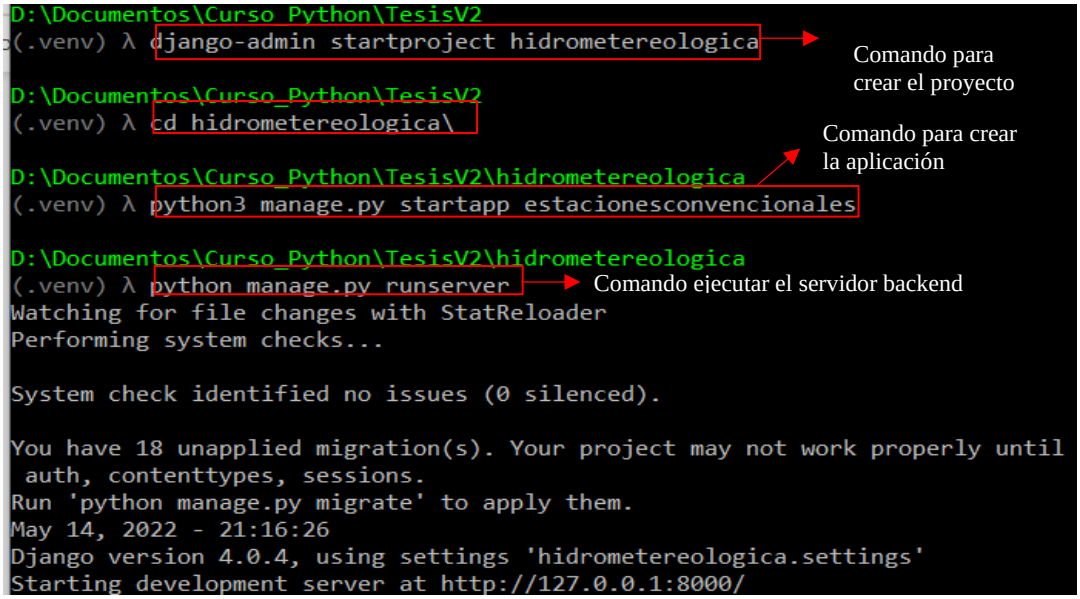
Elaborado por: (Apolo & Taco, 2022)

3. Creación del proyecto web, la aplicación y configuración de los settings.py.

En la Figura 20 se observa los comandos para la creación y ejecución de un proyecto dentro del entorno de desarrollo.

Figura 20.

Comandos para crear el proyecto y la aplicación.



Nota: *Comandos para crear el proyecto y la aplicación de la API*

Elaborado por: (Apolo & Taco, 2022)

Configuración del archivo settings.py del proyecto hidrometereológicas (Figura 21), colocando en el *Engine* el gestor de base de datos a usar, seguidamente de options donde se coloca los esquemas a usar y por último el nombre, user, password, host y puerto de la base de datos.

Figura 21.

Configuración para conectarse a la base de datos

Nota: *Archivo settings.py para configurar los parámetros para la conexión a la BDD*

Elaborado por: (Apolo & Taco, 2022)

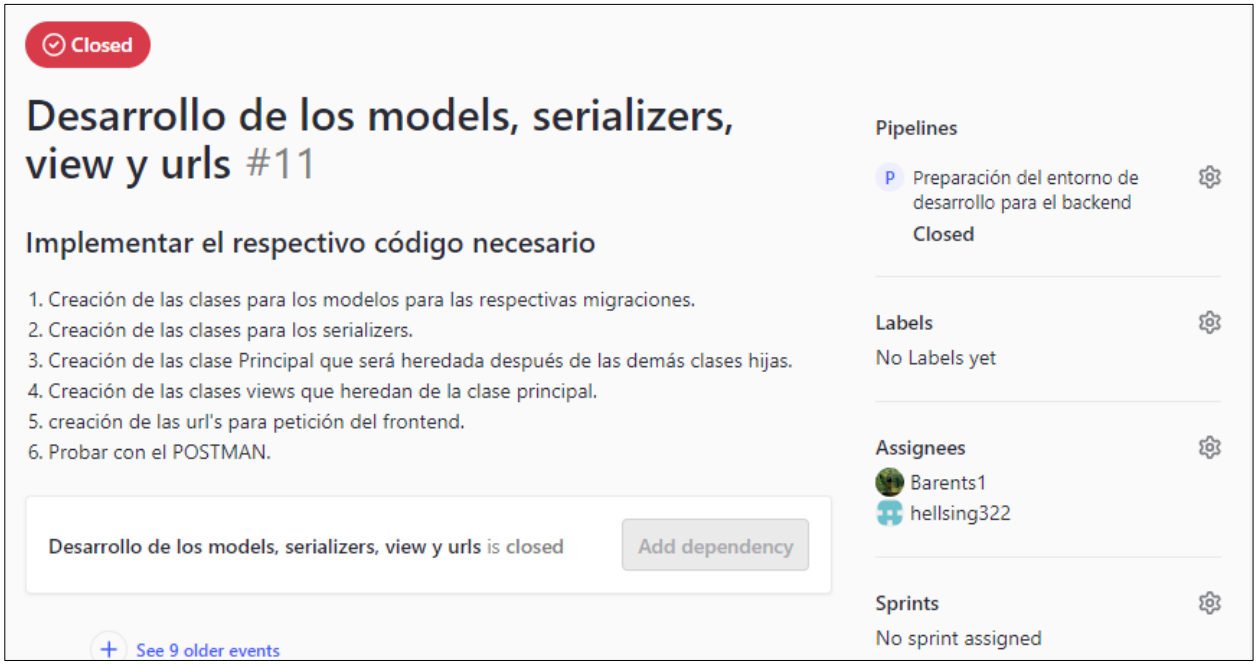
3.6. Sprint 5 Desarrollo de los modelos zerializables y views

En el quinto *sprint* (Figura 22) se implementó en primera instancia las respectivas clases de los modelos que contiene los atributos respectivos definidos en el modelado de la base de datos, como segundo punto se implementaron las clases de los serializadores, que convierte las consultas de los modelos a datos nativos de Python, en el tercer ítem se implementó una clase Principal en las *views* para que pueda ser heredada por las demás clases hijas, las cuales fueron creadas en el cuarto paso, seguidamente en el quinto punto se realizó la creación de las respectivas urls para que se pueda

realizar las respectivas peticiones y por último se comprobó con la herramienta Postman realizando llamadas a los *endpoints* desarrollados.

Figura 22.

Contenido para la codificación. Sprint 5



Nota: *Tabla de actividades planificadas para el sprint 5 (6 ítems)*

Elaborado por: (Apolo & Taco, 2022)

3.6.1. Creación de clases para los modelos

El archivo models.py es generado al momento de crear la aplicación en donde se alojará las diferentes clases que representan las tablas de la base de datos. En este caso teniendo una base de datos facilitada por el INAMHI se realizó una ingeniería inversa, utilizando la herramienta inspectdb de Django ver Figura 23, la cual facilitó la creación de los modelos que contienen los atributos respectivos de cada entidad de la base de datos.

Figura 23.

Comandos para crear clases de los modelos

```
PS C:\Users\aluca\Desktop\proyectoFinal\DjangoApi> python manage.py inspectdb > ApiRestEstacionesConvencionales/model
```

Nota: *Comando para realizar ingeniería inversa el cual crea las respectivas clases de los modelos a través de gestor de base de datos. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 24 se puede observar el modelo contenido en una clase con sus respectivos atributos, que a su vez hereda de la clase *models* de Django la cual contiene los métodos requeridos para la creación de las tablas en la base de datos.

Figura 24.

Ejemplo de una clase model.

```
from django.db import models


class T1073161H(models.Model):
    field_id = models.AutoField(db_column='_id', primary_key=True)  # Field renamed bec
    id_estacion = models.IntegerField()
    id_usuario = models.IntegerField()
    fecha_ingreso = models.DateTimeField()
    valor = models.DecimalField(max_digits=5, decimal_places=2, blank=True, null=True)
    validado = models.BooleanField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = '"test_conv\".\"_1073161h"'
```

Nota: *Clase T1073161h con sus respectivos atributos implementado en el archivo models.py. Elaborado por: (Apolo & Taco, 2022)*

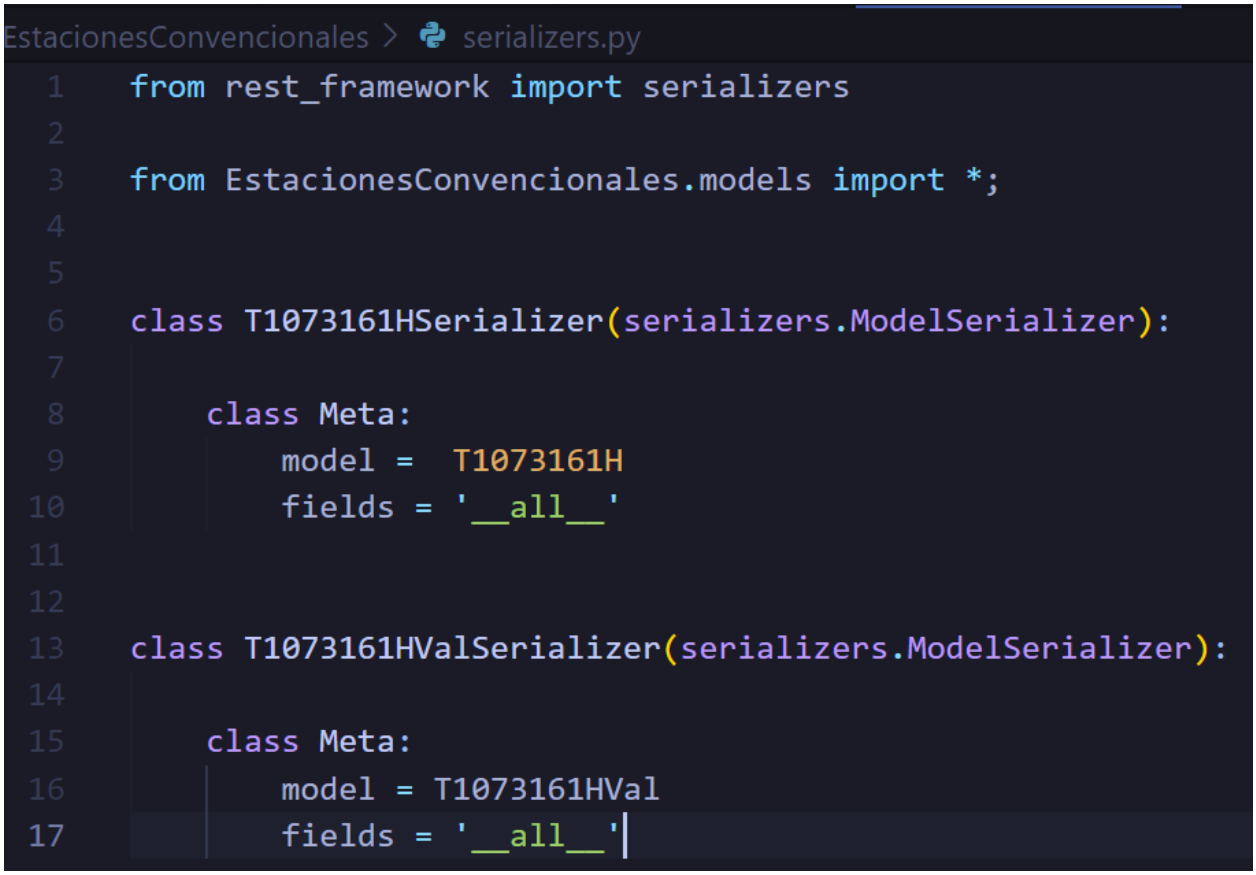
3.6.2. Creación de las clases para los serializers.

El archivo *serializer.py* se creó manualmente y este es usado para crear clases (Figura 25), las cuales pueden trasformar la información compleja a datos nativos de Python, los cuales serán usados posteriormente para representar fácilmente un formato JSON o XML.

Nota Se debe importar la librería *serializers* de *rest_framework* de django para utilizar los métodos que ayudan con el proceso de trasformación de datos.

Figura 25.

Ejemplo de una clase serielizer.



Nota: *Bloques de código de las clases serializers para el intérprete de consultas e instancias de los modelos la cual permiten convertir a datos nativos de Python. Elaborado por: (Apolo & Taco, 2022).*

3.6.3. Creación de las clases Views

Con la creación de la aplicación Estaciones Convencionales se genera un archivo *Views.py* de forma automática, con el que se ha trabajado usando vistas orientadas a clases en las que se pueden heredar las clases *mixin* y *generic* que ayudan con la simplificación de las consultas enviadas a la base de datos

3.6.4. Creación de clase Padre (Principal).

En la Figura 26 se observa la clase principal que contiene los siguientes métodos (get, post, put, delete), con los cuales se gestiona la (obtención, ingreso, edición, eliminación) información en la BDD, con la ayuda de los métodos que provienen de las clases *mixins* que simplifican las consultas SQL. Otra funcionalidad implementada en la clase principal son los logs que ayudan a verificar en que momento se realizó una petición y de que tipo hacia los *endpoints*.

Figura 26.

Implementación de código de la clase principal.

```
class Principal(mixins.ListModelMixin, mixins.CreateModelMixin, mixins.RetrieveModelMixin, mixins.DestroyModelMixin,
                mixins.UpdateModelMixin, generics.GenericAPIView):
    queryset:any
    serializer_class:any


    def __init__(self, serial,model):

        self.__class__.queryset = model.objects.all()
        self.__class__.serializer_class = serial

    def post(self,request, *args, **kwargs):
        loggers.logger_start(f'{datetime.now()}: Iniciando Ejecucion post logger\n')

        result = self.create(request, *args, **kwargs)
        if result.status_code == 201:
            loggers.logger_start(f'{datetime.now()}: Dato Ingresado\n')
        loggers.logger_finaly(f'{datetime.now()}: Proceso de ingreso finalizado, estado: {result.status_code}\n')
        return result

    def get(self, request, *args, **kwargs):
        loggers.logger_start(f'{datetime.now()}: Iniciando Ejecucion list logger\n')

        result = self.list(request, *args, **kwargs)
        if result.status_code == 200:
            loggers.logger_start(f'{datetime.now()}: Datos obtenidos\n')
        loggers.logger_finaly(f'{datetime.now()}: Proceso de obtencion de datos finalizado, estado: {result.status_code}\n')
        return result
    def put(self,request, pk, *args, **kwargs):
        loggers.logger_start(f'{datetime.now()}: Iniciando Ejecucion update logger\n')

        result = self.update(request, *args, **kwargs)
        if result.status_code != 400:
            loggers.logger_start(f'{datetime.now()}: Dato actualizado \n')
        loggers.logger_finaly(f'{datetime.now()}: Proceso update de datos finalizado, estado: {result.status_code}\n')
        return result

    def delete(self,request, *args, **kwargs):
        loggers.logger_start(f'{datetime.now()}: Iniciando Ejecucion delete logger\n')
        result = self.destroy(request, *args, **kwargs)
        if result.status_code == 204:
            loggers.logger_start(f'{datetime.now()}: Dato eliminado \n')
        loggers.logger_finaly(f'{datetime.now()}: Proceso delete de datos finalizado, estado: {result.status_code}\n')
        return result
```

Nota: *Clase Principal que hereda métodos de la clase mixins para la implementación del CRUD.*

Elaborado por: (Apolo & Taco, 2022)

3.6.5. Creación de las clases hijas

Las clases hijas mostradas en la Figura 27 están simplificadas gracias al uso de la herencia, de tal manera, que solo necesita enviar 2 parámetros a la clase principal (Figura 26), la cual le provee de los métodos anteriormente mencionados. Esta herencia se la realiza para una correcta optimización y reducción de código en el aplicativo.

Figura 27.

Ejemplo de código para clases hijas

```
class T12827161HView(Principal):

    def __init__(self):
        Principal.__init__(self,T12827161HSerializer,T12827161H)


class T12827161HValView(Principal):

    def __init__(self):
        Principal.__init__(self,T12827161HValSerializer,T12827161HVal)


class T13028161DView(Principal):

    def __init__(self):
        Principal.__init__(self,T13028161DSerializer,T13028161D)
```

Nota: *Clases que heredan de la clase Principal pasándole como parámetros las clases serializers y models respectivamente. Elaborado por: (Apolo & Taco, 2022).*

3.6.6. Creación de las urls

El archivo urls.py que se creó manualmente, donde se encuentran las diferentes rutas, que permiten el uso de los recursos del programa, y estas tienen la capacidad de admitir peticiones, las cuales

interactúan con los diferentes componentes como son las vistas (*views*) esto se puede observar en la Figura 28. Se usan URLS para que las aplicaciones *Frontend* puedan acceder a los diferentes recursos del *Backed* realizando peticiones HTTP O HTTPS.

Figura 28.

Codificación para las peticiones http.

```
urlpatterns=[


    path('t12827161h/', views.T12827161HView.as_view(), name='t12827161h'),
    path('t12827161h/<int:pk>',views.T12827161HView.as_view(),name='t12827161h'),
```

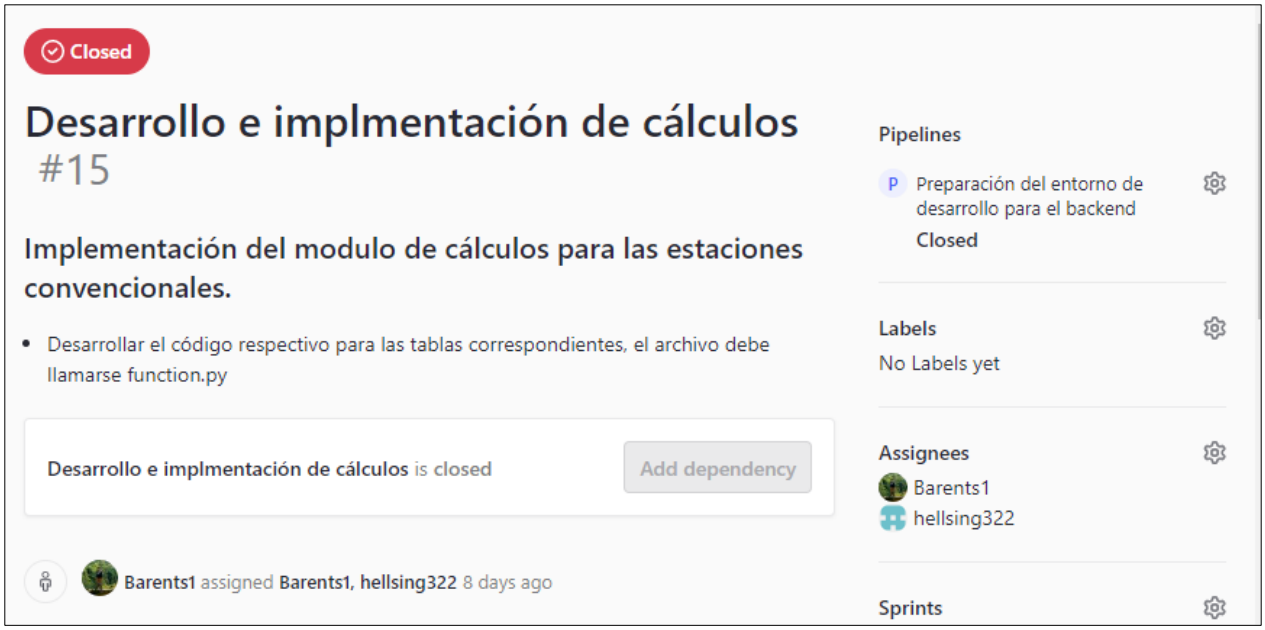
Nota: *Bloque de código de las urls para el servicio web. Elaborado por: (Apolo & Taco, 2022)*

3.7. Sprint 6 Implementación de cálculos meteorológicos

En el *Sprint* 6 se realizó la respectiva codificación para los cálculos diarios de las estaciones convencionales, también se implementó validaciónes para verificar valores nulos o acomulados que puede contener los campos con los que se trabaja para calcular los valores necesitados de cada estación hidrometereológica.

Figura 29.

Actividad para realizar el Sprint 6



Nota: *Tabla de contenido para el sprint 6 para la implementación de cálculos de las estaciones convencionales. Elaborado por: (Apolo & Taco, 2022)*

3.7.1. Cálculo de la precipitación

Con la creación del archivo *funtion.py* se implementó la clase cálculos con los distintos métodos, los cuales contienen cada una de las fórmulas que se utilizan más adelante:

Por medio de la Figura 30.

Código *del cálculo de la precipitación.*se puede visualizar la fórmula y las validaciones respectivas de la precipitación diaria que fueron previamente establecidas. En la Figura 30 se observa la codificación del cálculo de la precipitación diaria el cual fue previamente establecido.

Figura 30.

Código del cálculo de la precipitación.

Nota: *Bloque de código para el cálculo de la precipitación diaria. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 31 se observa la validación y verificación de datos existentes correspondientes a la precipitación diaria Estas condiciones comprueban si los datos a utilizar para el cálculo existen o no en la base de datos y de ser el caso generen su respectiva respuesta.

Figura 31.

Código de validación de la precipitación.

```
#Compruebo que la consulta de valor de las 13h no sea nula y la almaceno en una variabl
        if valor13h.exists()==True :
            for a in valor13h:
                valor13ht=a['valor']
                valor13ht=float(valor13ht)
#En el caso de no encontrar valores en la consulta genero una alerta con la fecha y el v
        else:
            valor13ht='no existe valor 13 horas fecha :  '+str(ultima_fecha)

#De igual manera compruebo el valor de las 19h del dia actual
        if valor19h.exists()==True :
            for a in valor19h:
                valor19ht=a['valor']
                valor19ht=float(valor19ht)
        else:
            valor19ht='no existe valor 19 horas fecha :'+str(ultima_fecha)
```

Nota: *Bloque de código para validar que la información exista en la base de datos.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 32 se observa la validación que se realiza en el caso de que sí existiera 3 valores iguales 9.9, se considera que los valores son acumulados.

Figura 32.

Validación para valores iguales de la precipitación.

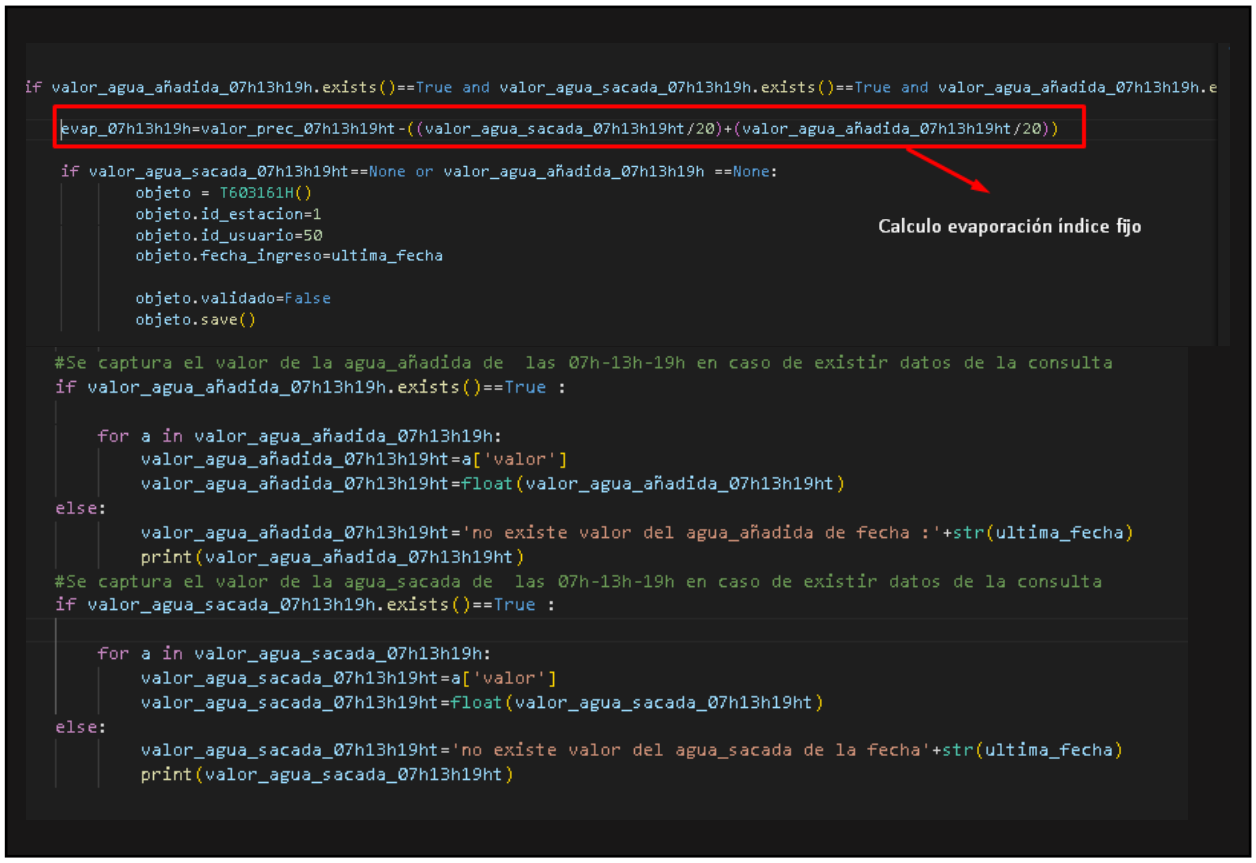
```
#Se comprueba si los tres valores diarios son nulos para no calcular el dia actual y el siguiente
        if valor13ht == 9.9 and valor19ht==9.9 and valor07ht==9.9:

            print("Existe valor acumulado")


#Guardo mis datos en la base de datos con valor nulo por que se a generado una exepcion
            ultima_fecha =ultima_fecha+timedelta(1)
            objeto = T171481D()
            objeto.id_usuario=0
            #falta agregar la descripcion ...
            objeto.fecha_ingreso=fecha_guardada
            #valor nulo
            objeto.diario=False
            objeto.tipo_dato=1
            objeto.save()
```

Nota: *Bloque de código que valida los valores acumulados si fuese el caso de obtener datos con las cantidades 9.9. Elaborado por: (Apolo & Taco, 2022).*

Por medio de la Figura 39 se puede visualizar la fórmula y las validaciones respectivas de la evaporación índice fijo que fueron previamente establecidas en la sección 1.4

Figura 33.

Código para el cálculo del índice fijo.



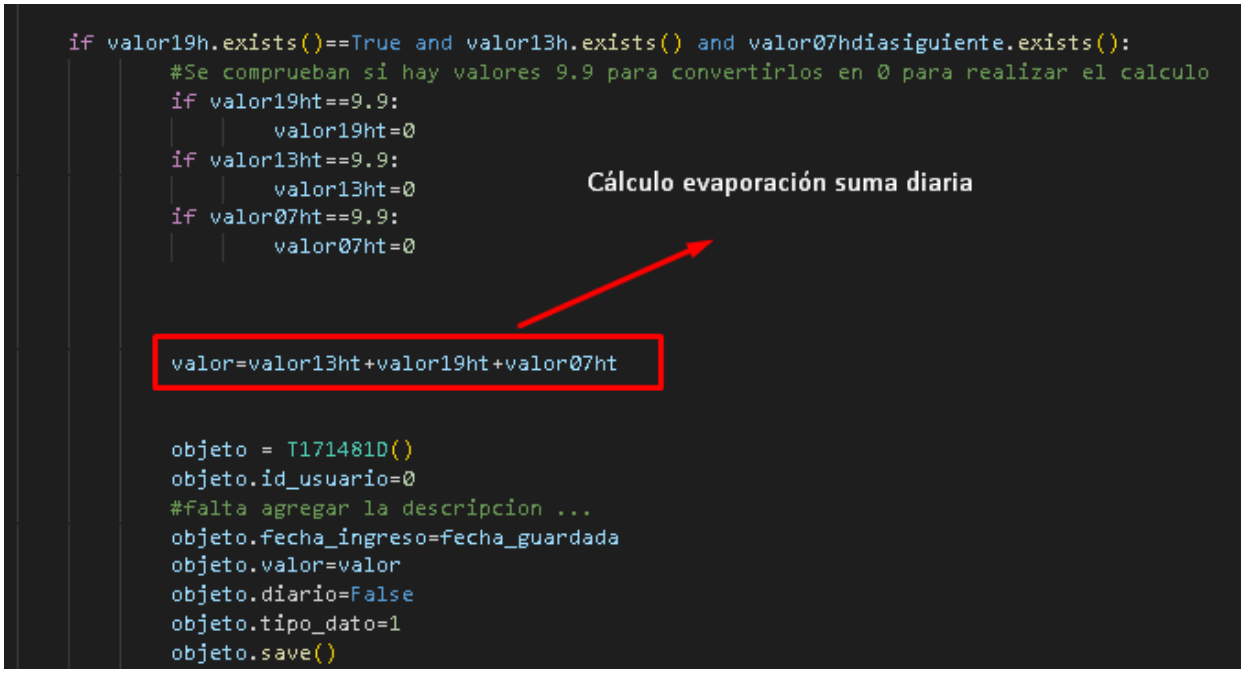

```
if valor_agua_añadida_07h13h19h.exists()==True and valor_agua_sacada_07h13h19h.exists()==True and valor_agua_añadida_07h13h19h.e
    evap_07h13h19h=valor_prec_07h13h19ht-((valor_agua_sacada_07h13h19ht/20)+(valor_agua_añadida_07h13h19ht/20))

    if valor_agua_sacada_07h13h19ht==None or valor_agua_añadida_07h13h19h ==None:
            objeto = T603161H()
            objeto.id_estacion=1
            objeto.id_usuario=50
            objeto.fecha_ingreso=ultima_fecha

            objeto.validado=False
            objeto.save()
    #Se captura el valor de la agua_añadida de  las 07h-13h-19h en caso de existir datos de la consulta
    if valor_agua_añadida_07h13h19h.exists()==True :

        for a in valor_agua_añadida_07h13h19h:
            valor_agua_añadida_07h13h19ht=a['valor']
            valor_agua_añadida_07h13h19ht=float(valor_agua_añadida_07h13h19ht)
    else:
            valor_agua_añadida_07h13h19ht='no existe valor del agua_añadida de fecha :'+str(ultima_fecha)
            print(valor_agua_añadida_07h13h19ht)
    #Se captura el valor de la agua_sacada de  las 07h-13h-19h en caso de existir datos de la consulta
    if valor_agua_sacada_07h13h19h.exists()==True :

        for a in valor_agua_sacada_07h13h19h:
            valor_agua_sacada_07h13h19ht=a['valor']
            valor_agua_sacada_07h13h19ht=float(valor_agua_sacada_07h13h19ht)
    else:
            valor_agua_sacada_07h13h19ht='no existe valor del agua_sacada de la fecha'+str(ultima_fecha)
            print(valor_agua_sacada_07h13h19ht)
```

Nota: *Bloque de código donde se implementa el cálculo de la evaporación índice fijo y las respectivas validaciones en el caso de que existan valores nulos. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 34 se puede visualizar la fórmula y las validaciones respectivas de la evaporación suma diaria que fueron previamente establecidas en la sección 1.4.

Figura 34.

Código para el cálculo de la evaporación suma diaria.




```
if valor19h.exists()==True and valor13h.exists() and valor07hdiasiguiente.exists():
        #Se comprueban si hay valores 9.9 para convertirlos en 0 para realizar el calculo
        if valor19ht==9.9:
                valor19ht=0
        if valor13ht==9.9:
                valor13ht=0
        if valor07ht==9.9:
                valor07ht=0


        valor=valor13ht+valor19ht+valor07ht


        objeto = T171481D()
        objeto.id_usuario=0
        #falta agregar la descripcion ...
        objeto.fecha_ingreso=fecha_guardada
        objeto.valor=valor
        objeto.diario=False
        objeto.tipo_dato=1
        objeto.save()
```

Nota: *Bloque de código donde se implementa el cálculo de la evaporación suma diaria y las respectivas validaciones en el caso de que existan valores nulos para las 3 distintas horas.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 35 se puede visualizar la fórmula y las validaciones respectivas de la temperatura media diaria del aire que fueron previamente establecidas en la sección 1.4.

Figura 35.

Código para el cálculo de la temperatura del aire suma diaria

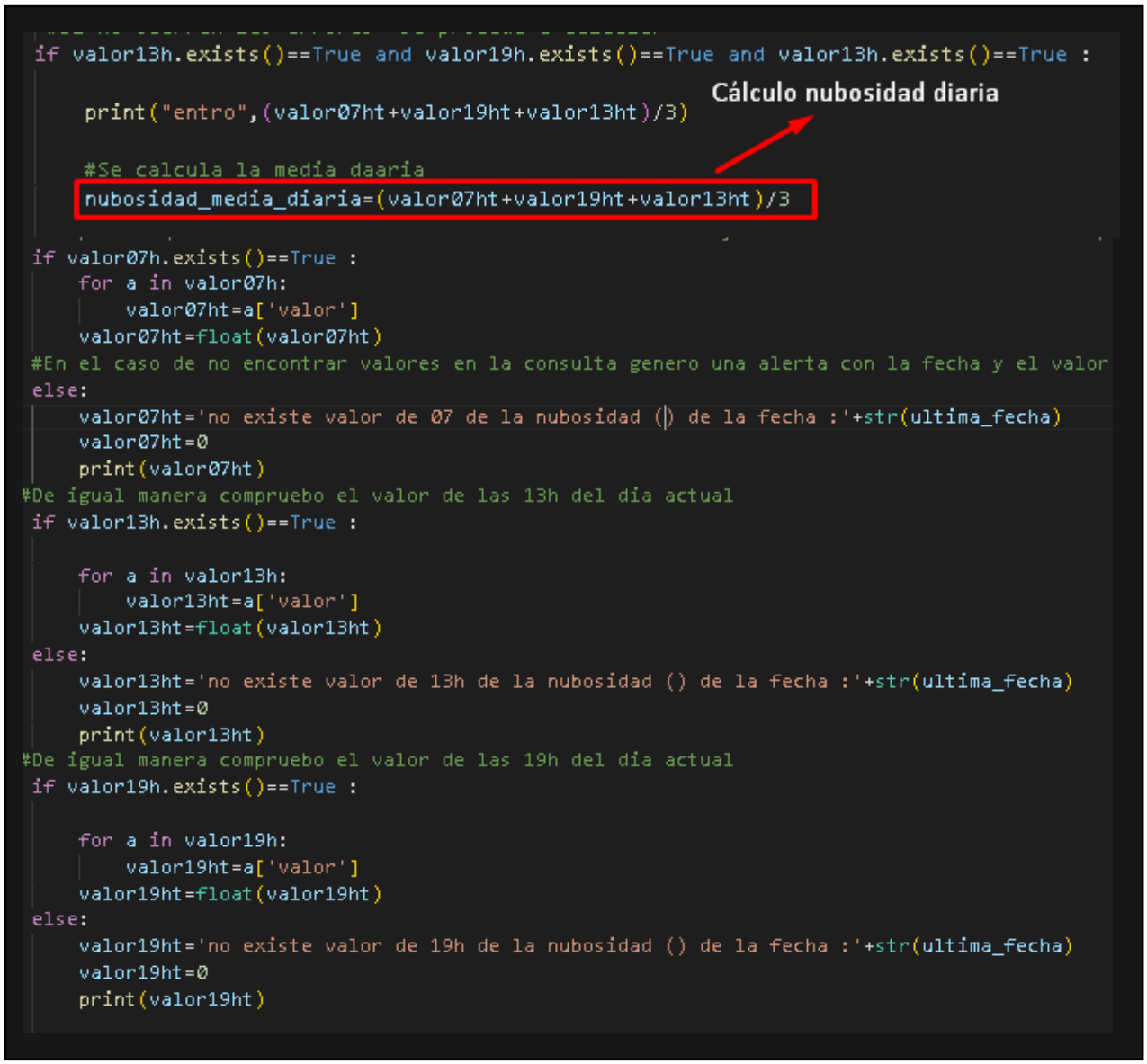

```
 #Si no ocurren las errores  se procede a calcular
if valor13h.exists()==True and valor19h.exists()==True and valor13h.exists()==True :

    print("entro",(valor07ht+valor19ht+valor13ht)/3)

    #Se calcula la media daaria
    temperatura_media_diaria=(valor07ht+valor19ht+valor13ht)/3

    print(valor07ht)
#De igual manera compruebo el valor de las 13h del dia actual
 if valor13h.exists()==True :

    for a in valor13h:
        valor13ht=a['valor']
    valor13ht=float(valor13ht)
 else:
    valor13ht='no existe valor de 13h de la temperatura (Aire) de la fecha :'+str(ultima_fecha)
    valor13ht=0
    print(valor13ht)
#De igual manera compruebo el valor de las 19h del dia actual
 if valor19h.exists()==True :

    for a in valor19h:
        valor19ht=a['valor']
    valor19ht=float(valor19ht)
 else:
    valor19ht='no existe valor de 19h de la temperatura (Aire) de la fecha :'+str(ultima_fecha)
    valor19ht=0
    print(valor19ht)
```

Nota: *Bloque de código donde se implementa el cálculo de la temperatura media diaria del aire seguidamente de la validación para valores nulos. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 36 se puede visualizar la fórmula y las validaciones respectivas de la nubosidad diaria que fueron previamente establecidas.

Figura 36.

Código para el cálculo de la nubosidad diaria.

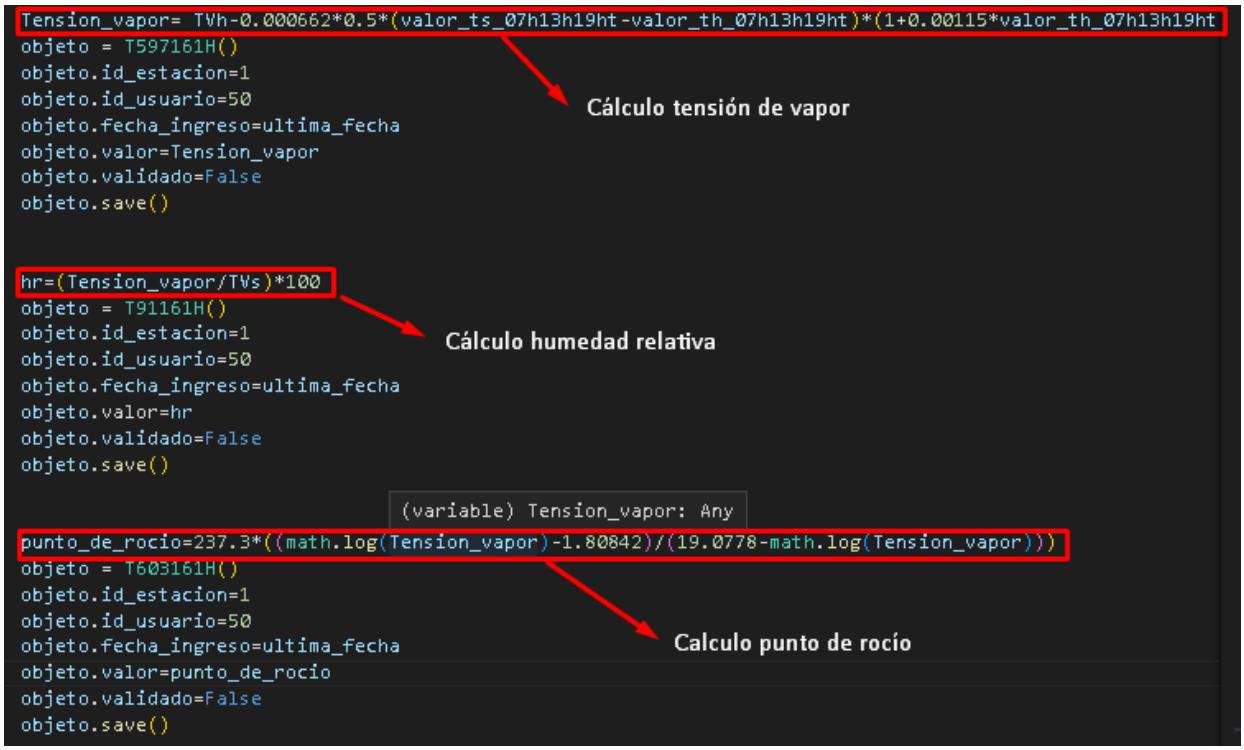


Nota: *Bloque de código donde se implementa el cálculo de la nubosidad diaria seguidamente de las respectivas validaciones en el caso de contener valores nulos. Elaborado por: (Apolo & Taco, 2022)*

En la Figura 37 se puede visualizar la fórmula y las validaciones respectivas de la tensión de vapor, humedad relativa, punto de roció lo cuales fueron previamente establecidos.

Figura 37.

Código para el cálculo de la tensión vapor, humedad relativa, punto de rocío.



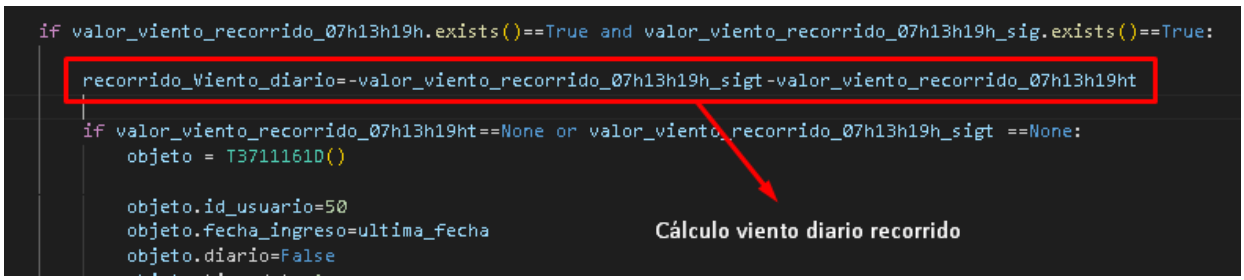
Nota: *Bloque de código donde se implementa el cálculo de la tensión de vapor, humedad relativa y punto de recio en el cual se guardan en diferentes tablas según les corresponda.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 38 se puede visualizar la fórmula con la validación respectiva del viento recorrido diaria que fueron previamente establecidas.

Figura 38.

Código para el cálculo del viento diario recorrido.



Nota: *Bloque de código donde se implementa el cálculo del viento diario recorrido.*

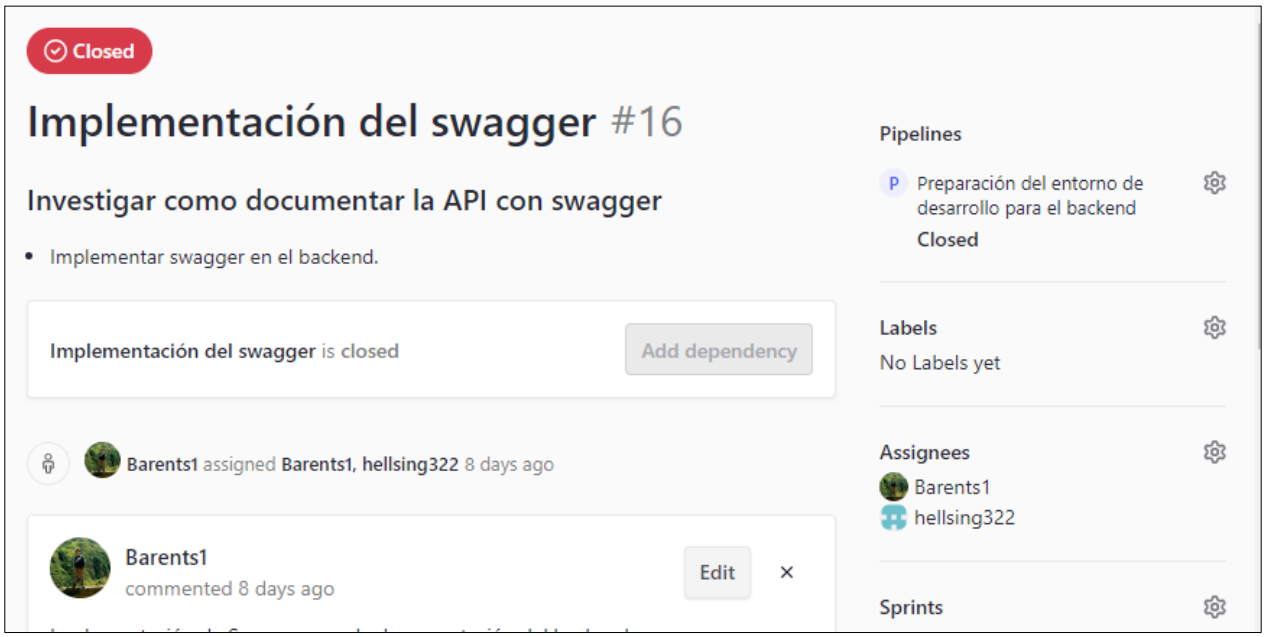
Elaborado por: (Apolo & Taco, 2022)

3.8. Sprint 7 Implementación de Swagger

En el séptimo *sprint* se realizó una investigación para entender e implementar *Swagger* para la documentación de la API, esto facilitó el desarrollo de las interfaces graficas del usuario. *Swagger* también ayudo a verificar errores presentados en los *endpoints* donde se estaba enviando parámetros de una tabla con parámetros distintos al que se requería ingresar la información.

Figura 39.

Actividades para el Sprint 7



Nota: *Tabla de contenido de actividades para el sprint 7. Elaborado por: (Apolo & Taco, 2022).*

3.8.1. Instalación de Swagger.

Para la implementación de la herramienta *swagger* se necesitó ejecutar el comando mostrado en la Figura 40.

Figura 40.

Instalación de swagger.

```
:\Users\aluca\Desktop\Hidrometereologicas>pip install drf-yasg
```

Nota: *Comando para instalar swagger. Elaborado por: (Apolo & Taco, 2022)*

Una vez instalado *swagger* se debe importar la librería *apenapi* y añadir su configuración junto con sus respectivos *paths,* en el archivo urls de proyecto tal y como se muestra en la Figura 41.

Figura 41.

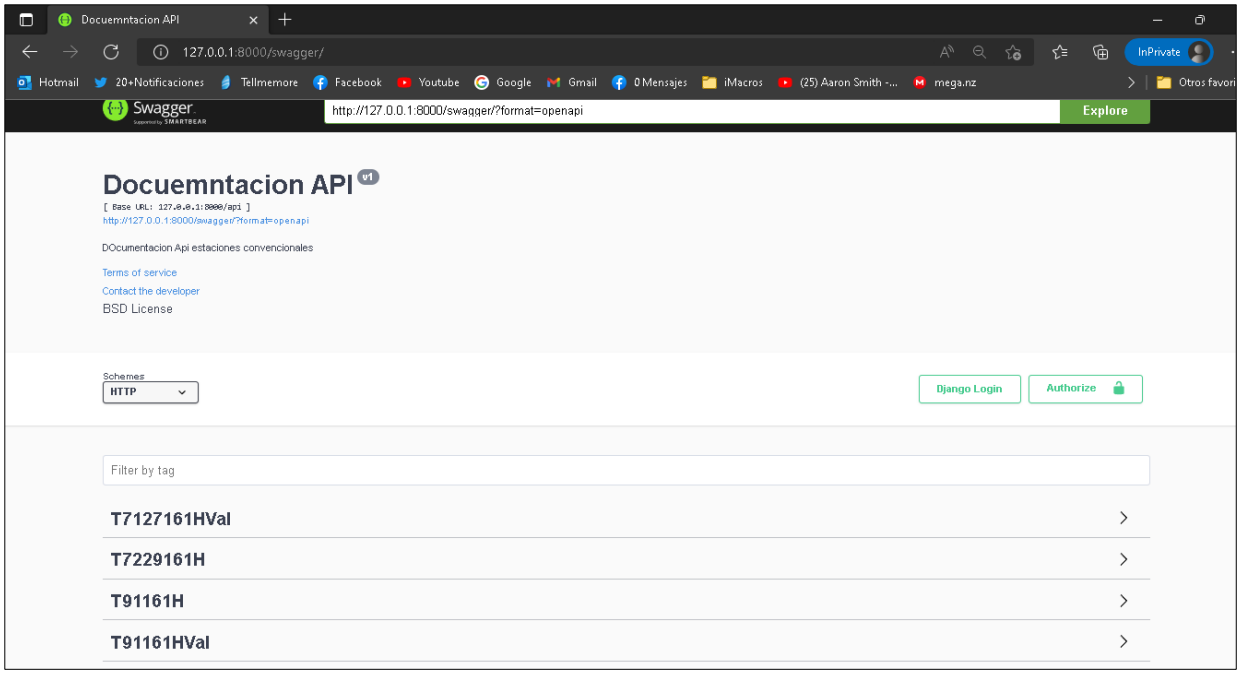
Código para configurar el openapi.

Nota: *Bloque de configuración de los paths en la url del proyecto. Elaborado por: (Apolo & Taco, 2022).*

Finalmente se debe iniciar el servidor y posteriormente dirigirse a la url configurada como se muestra en la Figura 42.

Figura 42.

Diagrama de la documentación de la API.



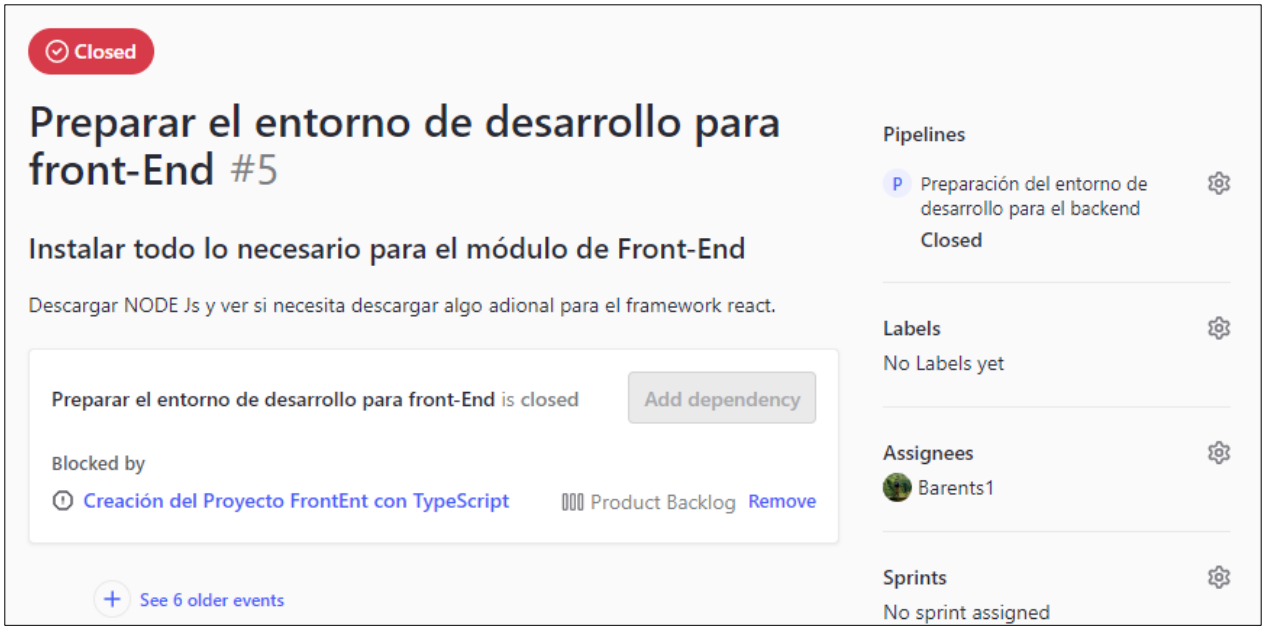
Nota: *Interfaz de swagger implementada y mostrada en el navegador. Elaborado por: (Apolo & Taco, 2022)*

3.9.Sprint 8 Preparación del entorno Front-End

En el octavo *Sprint* se realizó una pequeña investigación para desarrollar el módulo de *FrontEnd*, el cual fue necesario instalar NODEJS debido a que usa librerías y dependencias de esta herramienta para que React pueda trabajar sin ningún inconveniente.

Figura 43.

Actividades para el sprint 8



Nota: *Tabla de contenido de actividades para el Sprint 8. Elaborado por: (Apolo & Taco, 2022)*

3.9.1. Instalación de NODEJS

Para la descarga de *NODEJS* es necesario abrir un navegador y colocar la siguiente url: Descarga | Node.js (nodejs.org).

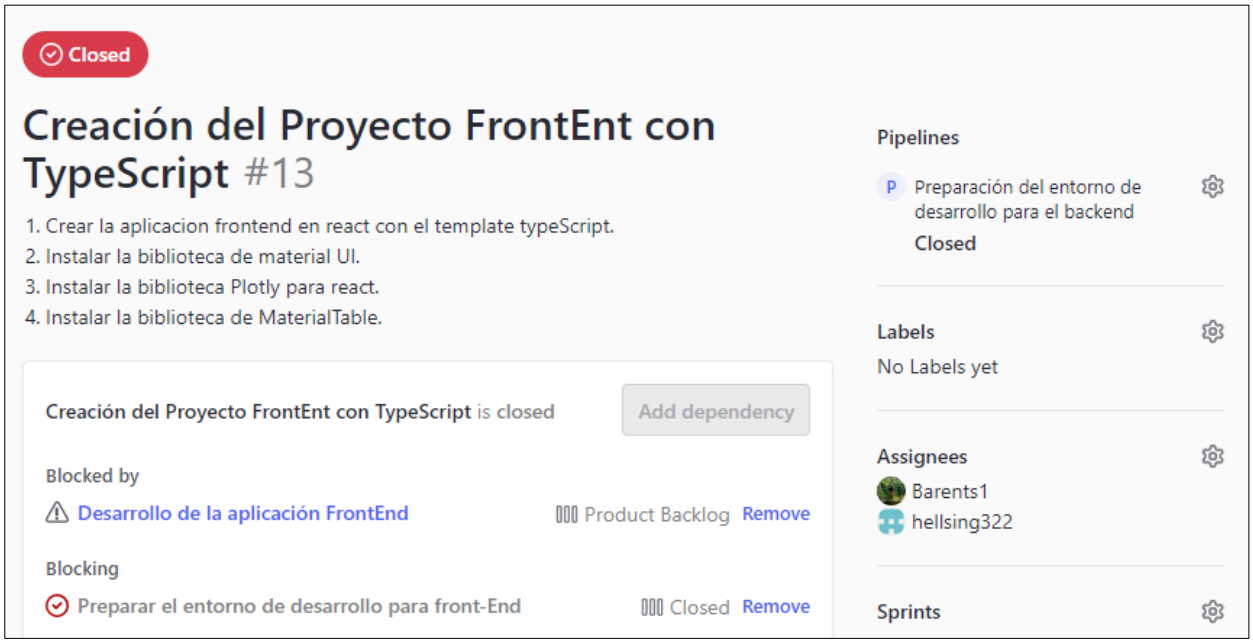
Una vez abierta la página, se debe seleccionar la opción de descarga según el sistema operativo en que se desarrollará el aplicativo, en este caso se desarrolla bajo la plataforma de Windows y se procede con la descarga de la versión recomendada. Finalmente, se realiza la instalación estándar siguiendo los pasos que se presentan en el asistente de instalación del programa.

3.10. Sprint 9 Instalación de librerías para Front-End

En el noveno *Sprint* se creó el proyecto con la extensión TypeScript, el cual corresponde al desarrollo de la interfaz gráfica de usuario comúnmente llamado por los desarrolladores *FrontEnd* que habitualmente React acepta la escritura del código en JavaScript, pero para este caso se utiliza typescript. Posteriormente se instalaron algunas librerías para una mejor experiencia de usuario.

Figura 44.

Instalación de librerías para Front-End



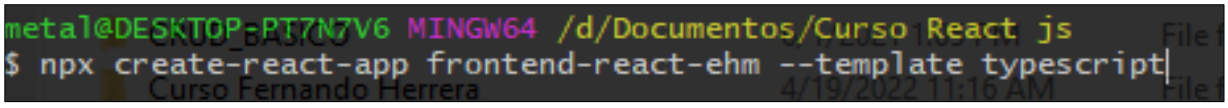
Nota: *Tablero de las actividades para el Sprint 9. Elaborado por: (Apolo & Taco, 2022)*

3.10.1. Creación de la aplicación en React.

En los requerimientos establecidos por el INAMHI para el desarrollo del *frontend* se realizó con la extensión .tsx reemplazando el .jsx común para el desarrollo de este tipo de aplicaciones.

Figura 45.

Creación de la aplicación en React.



Nota: *Comando para la utilización de tipado estricto en typescript. Elaborado por: (Apolo & Taco, 2022)*

3.10.2. Instalación de la librería material ui.

En el navegador colocar *install* material ui y seleccionar la opción de instalación, en este caso se selecciona la primera opción debido a que se utiliza un sistema operativo Windows, la segunda opción es para entornos Mac.

Figura 46.

Instalación de material-ui.

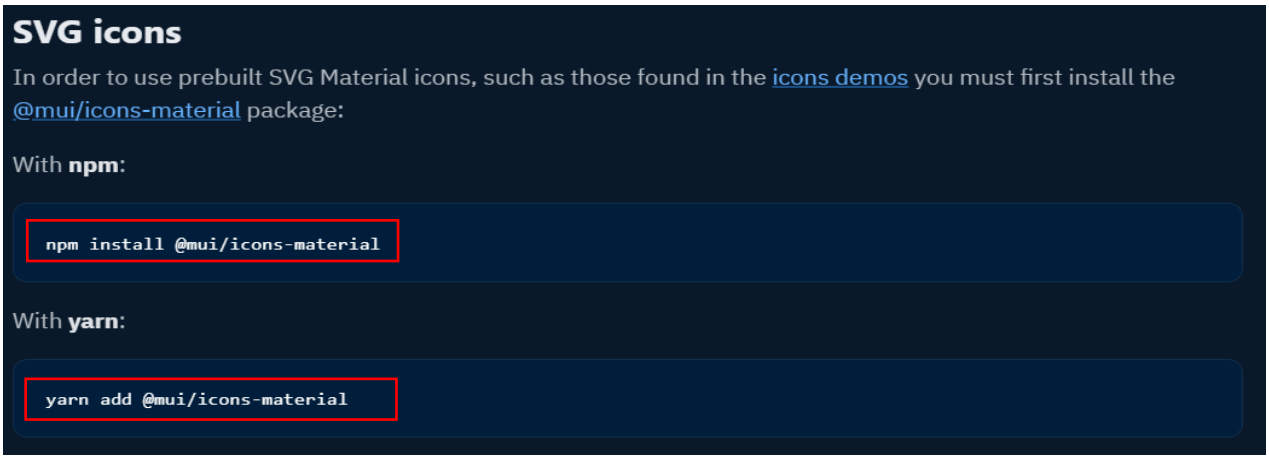
Nota: *Comando para instalar Material-UI en diferentes S.O. Elaborado por: (Apolo & Taco, 2022)*

3.10.3. Instalación de complementos de material-ui.

Es necesario la instalación de iconos de material-ui para una mejor experiencia de usuario, si se presentan errores al momento de la descargar ejecutar el comando *npm i fixed –force*.

Figura 47.

Instalación de complementos de material-ui.



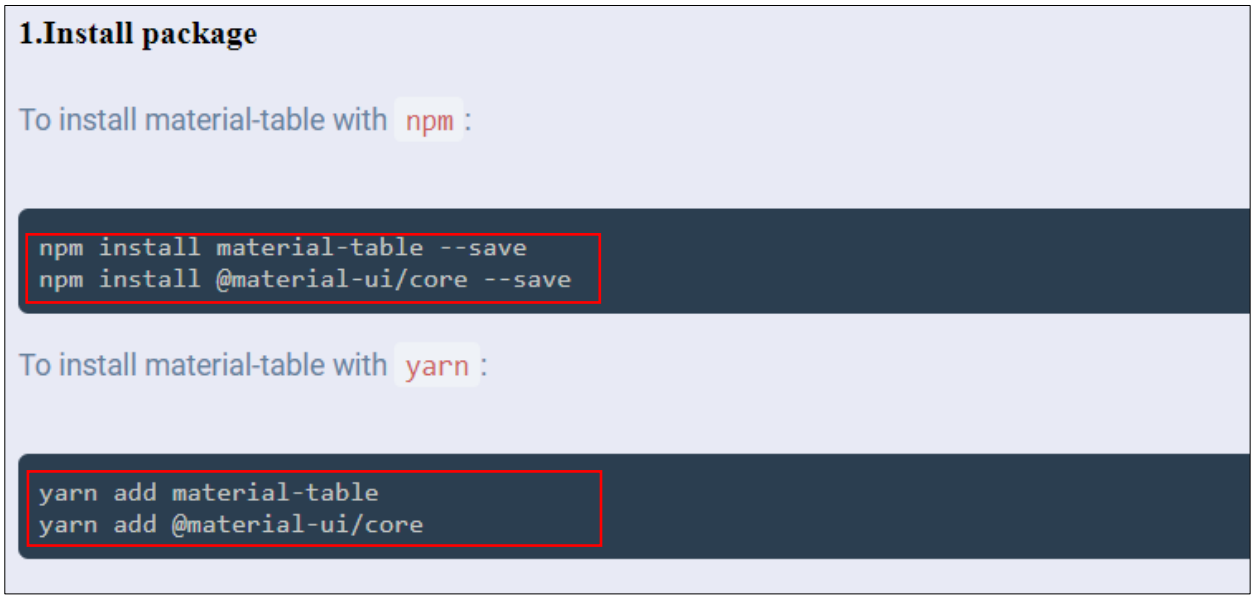
Nota: *Comando para instalar los componentes de icons. Elaborado por: (Apolo & Taco, 2022).*

3.10.4. Instalación de material-table.

Para la visualización de datos en tablas se usará material-table para una mejor comodidad y experiencia de usuario, debido que ofrece opciones de exportación de datos y paginación controlada en el *frontend*. Seleccionar la opción según el sistema operativo

Figura 48.

Instalación de material-table.



Nota: *Comandos para instalar Material-Tables dependiendo del S.O en el que se emplee.*

Elaborado por: (Apolo & Taco, 2022)

Agregar el siguiente código como se muestra en la Figura 49.

Código complementario de material-ui (icons), en el *index* de la carpeta publica del proyecto, la cual ayudará a tener una mejor visualización de iconos sin desbordes.

Figura 49.

Código complementario de material-ui (icons)

```
<!-- Codigo para visualizar los iconos -->
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />
```

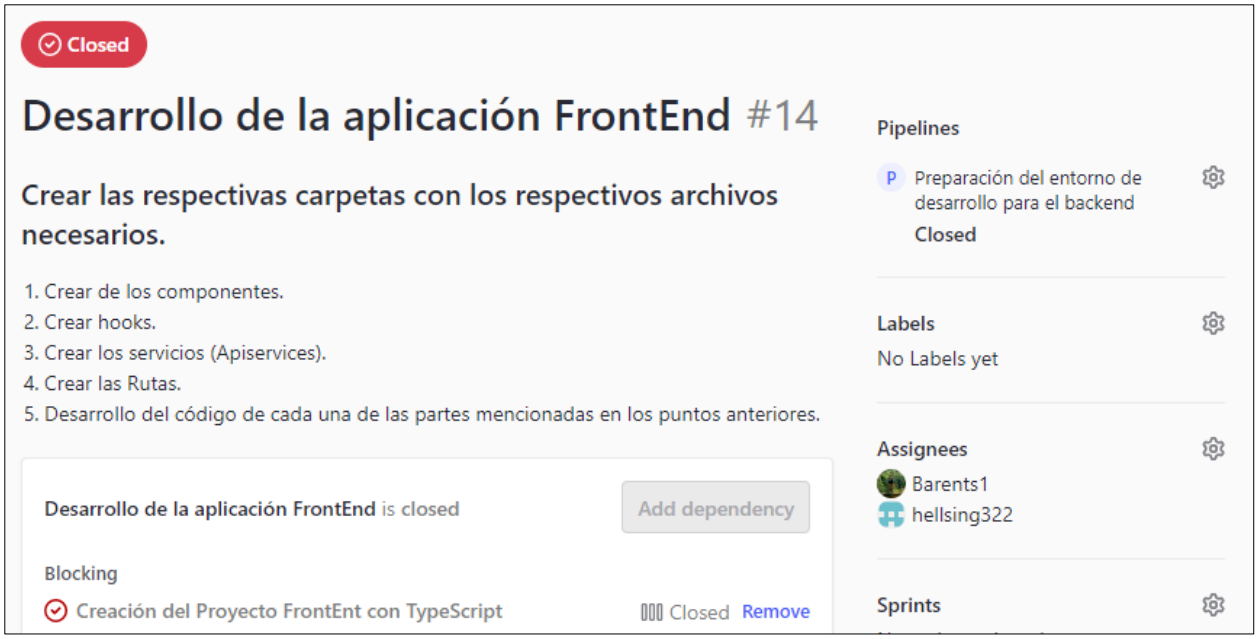
Nota: *Línea de código html para mostrar iconos sin desbordes. Elaborado por: (Apolo & Taco, 2022)*

3.11. Sprint 10 Desarrollo de la interfaz gráfica FrondEnd

En el décimo *sprint* se realizó la implementación del código correspondiente a todo el módulo de la interfaz gráfica de usuario iniciando por la creación de una variable constante, el cual contiene la ruta del servidor al que se debe apuntar, después se crearon los *hooks*, componentes, métodos para posteriormente reutilizarlo.

Figura 50.

Listado de actividades del Sprint 10



Nota: *Listado de contenido correspondiente al sprint 10.*

Elaborado por: (Apolo & Taco, 2022)

3.12.1 Creación de una variable constante para las peticiones al servidor backend.

Url de configuración de la conexión al servidor en el caso de que exista un cambio de la dirección url simplemente se debe cambiar en este apartado.

Figura 51.

Url de dirección del servidor backend.

```
1
2   export const apiUrl = {
3       API_URL:"http://127.0.0.1:8000/api/"
4   }
```

Nota: *Constante apiUrl que contiene la dirección del servidor backend. Elaborado por: (Apolo & Taco, 2022).*

3.12.2 Creación de la función GET.

La función obtener, proporciona datos requeridos dependiendo del nombre de la tabla a la cual se realiza la petición a la API. La solicitud se realiza mediante el *fetch* de tipo GET convirtiendo los datos en formato JSON.

Figura 52.

Código de la función Get.

```
const obtener = async () => {
  let resul = await fetch(direccion,{ method: 'GET',
                                      mode: 'cors',
                                      cache: 'default' })
                          .then(Response => Response.json())
                          .then(data => {
                            // console.log(data);
                            return data;
                          });
  const results:[] = resul.results;
  setNumberOfPages(Math.ceil((resul.count)/10));
  // console.log(direccion);
  setData17(results);
}
```

Nota: *Bloque de código para obtener datos provenientes del servidor.*

Elaborado por: (Apolo & Taco, 2022)

3.12.3 Creación de la función Post.

Función que facilita el ingreso de datos mediante la información ingresada por el usuario en los formularios que posteriormente son enviados al servidor el cual retorna un mensaje exitoso o fallido de ser el caso, al no ser validados en el servidor.

Figura 53.

Código de la función Post

```
const setPost = async () => {
   await fetch(direccion, {
    method:'POST',
    headers:{
        'Accept':'application/json',
        'Content-Type':'application/json'
    },
    body:JSON.stringify(datoSeleccionado)
  })
  .then(res=>res.json())
  .then((result)=>{
      alert('Ingresado exitosamente');
      obtener();
  },(error)=>{
      alert('Failed');
  })
}
```

Nota: *Bloque de código para insertar datos recibido desde un formulario el cual es enviado al servidor. Elaborado por: (Apolo & Taco, 2022)*

3.12.4 Creación de la función Put.

En la Figura 54.

Código de la función Put. se observa la función asíncrona, la cual es encargada de realizar la actualización de un cierto dato seleccionado por el usuario, que recibe el parámetro de la url con el id respectivo que almacena en la variable dirección, el cual retorna un mensaje exitoso o erróneo respectivamente.

Figura 54.

Código de la función Put.

```
const setPut = async()=>{
  const resul = await fetch(direccion,{
          method:'PUT',
          headers:{
              'Accept':'application/json',
              'Content-Type':'application/json'
          },
          body:JSON.stringify(datoSeleccionado)
        })
        .then((result)=>{
            alert('Actualizado con exito');
            obtener();
        },(error)=>{
            alert('Failed');
        })
  return resul;
}
```

Nota: *Bloque de código para actualizar información. Elaborado por: (Apolo & Taco, 2022).*

3.12.5 Creación de la función DELETE.

La función que se visualiza en la Figura 55 realiza una petición a la API de tipo *DELETE*, que devuelve una alerta con un mensaje exitoso o erróneo en el caso de no encontrar el dato para su eliminación.

Figura 55.

Código de la función Delete.

```
const Delete = async()=>{
    fetch(direccion,{
      method:'DELETE',
      headers:{
          'Accept':'application/json',
          'Content-Type':'application/json'
      }
    })
    .then(res=>res.json())
    .then(result=>{
        alert(`Eliminado con éxito`);
        obtener();
    })
    .catch((error)=>{
        alert(error);
    })
}
```

Nota: *Bloque de código para eliminar información. Elaborado por: (Apolo & Taco, 2022)*

3.12.6 Creación del Hook useForm.

El *hook* que se muestra a continuación utiliza unos valores iniciales, dentro de ella contine un método para cambiar y actualizar el estado de los valores del input mediante el evento *change* de Mozilla Developer Network (MDN), donde se desestructura los valores y se actualiza los datos ingresados por el usuario en los campos de textos, retornando dichos cambios. Los valores iniciales son recibidos desde los componentes que hacen uso de este *hook*.

Figura 56.

Hook useForm

```
export function useForm<T>(initState:T) {

    const [datoseleccionado, setDatoseleccionado] = useState(initState);
    const handleChange = ({target}:any) => {
        const { name, value} = target;
        setDatoseleccionado( prevState => ({
            ...prevState,
            [name]:value
        }));
    }

    return {
        datoseleccionado,
        handleChange,
        setDatoseleccionado,
    }
}
```

Nota: *Bloque de código para el cambio iterativo de valores de los campos de textos.*

Elaborado por: (Apolo & Taco, 2022)

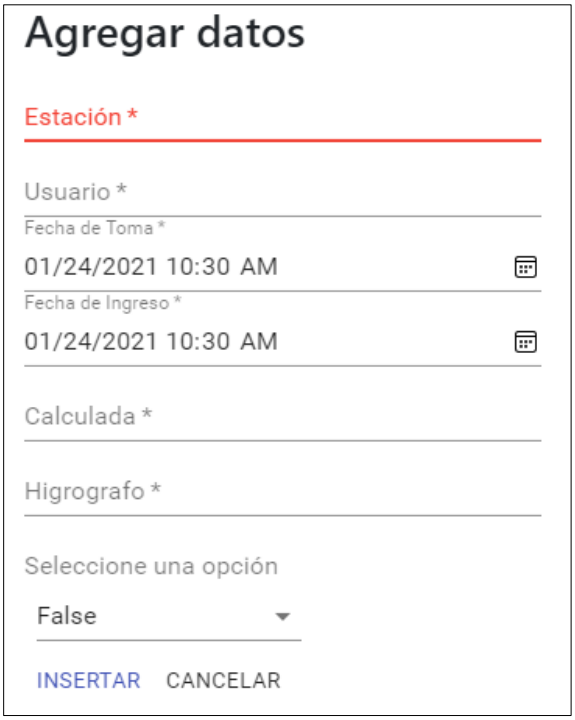
3.12.7 Formulario de ingreso de datos.

La interfaz de usuario fue desarrollada con la ayuda de los componentes de material-ui, para el ingreso de varios datos dependiendo de la tabla en la que se desea insertar, como se muestra en la Figura 57.

Ventana de ingreso de datos., en la mayoría existe el campo de tipo fecha donde se ha colocado valores por defecto que el cliente debe seleccionar.

Figura 57.

Ventana de ingreso de datos.



Nota: *Interfaz gráfica de usuario para el ingreso de datos.*

Elaborado por: (Apolo & Taco, 2022)

3.12.8 Componente table.

En la Figura 58 se puede observar el código implementado conjuntamente con el componente material-table que contiene varios parámetros de los cuales, los más principales son la data y *columns* con las que se trabaja para poderlas visualizar la información de manera amigable en los navegadores, también contiene acciones para la edición y eliminación de datos, otra opción que provee este componente es la exportación de datos mostradas en la tabla, de igual manera posee la opción de paginación, la cual se configuró para manejar hasta un máximo de 1097 registros debido a que son 3 datos diarios que se deben ingresar en la mayoría de tablas, este valor se lo definió ya que el año contiene 365 días que multiplicados por los 3 datos diarios da como resultado 1095.

Figura 58.

Implementación de MaterialTable.

```
<MaterialTable
    title="Datos"
    data={getResultado}
    columns={colums}
    actions={[
        {
            icon:'edit',
            tooltip:"editar",
            onClick:(event, rowData)=>selectData(rowData, opcion1)
        },
        {
            icon:'delete',
            tooltip:"eliminar",
            onClick:(event, rowData)=>selectData(rowData, opcion2)
        }
    ]}
    options={{
        exportButton:true,
        // paging: false,
        pageSizeOptions: [5, 10, 20, 50, 100,1097],
        actionsColumnIndex: -1,
    }}
    localization={{
        header:{
            actions: "Acciones"
        }
    }}
/>
```

Nota: *Bloque de código con los respectivos parámetros que debe contener el componente Material-table. Elaborado por: (Apolo & Taco, 2022)*

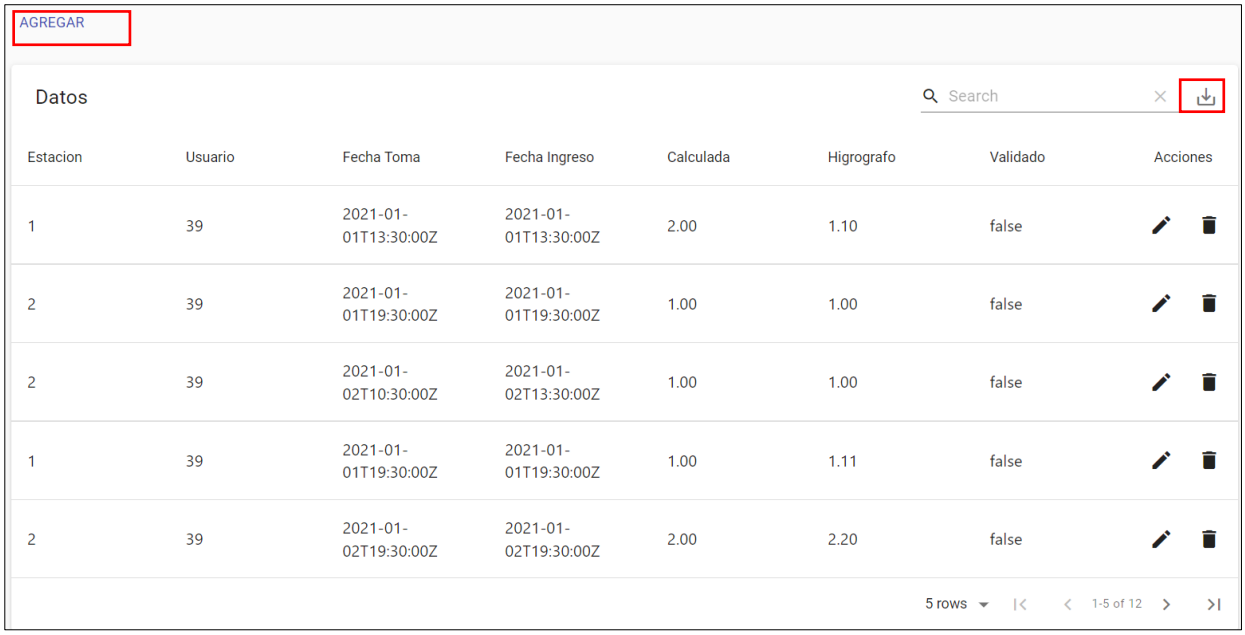
Ventana para visualizar la información.

En la Figura 59 se observa el listado de los datos de una de las tablas, y en la esquina superior izquierda un botón agregar, que al dar clic se abrirá un modal (Figura 57) para que el usuario pueda

realizar el ingreso de datos hidrometereológicos leídos en las estaciones convencionales, a su vez también se tiene la opción para descargar la información con las opciones de formato csv o pdf.

Figura 59.

Ventana para visualizar información.



Nota: *Ventana de muestra de datos en una tabla con algunas opciones de eventos.*

Elaborado por: (Apolo & Taco, 2022)

3.12.9 Componente Plot

Para mostrar los resultados de manera estadística se implementó la librería *plotly* para react, que contiene el parámetro data donde se le coloca el tipo de grafica a mostrar, *x* como etiqueta de fecha e *y* como los valores calculados proveniente de la API, también se agregó el parámetro *layout* que contiene el alto, ancho y titulo para la gráfica.

Figura 60.

Implementación del componente Plotly.

```
<div className={classes.root}>
    <div className={classes.container} >
        <Plot
            data={[
                // { type: 'bar', values:  valores, labels:  fechas },
                { type: 'bar', x: fechas, y: valores, marker:{color: 'rgb(33, 196, 76'} },
            ]}
            // layout={{ font: {size: 18}, title: 'A Fancy Plot' }}
            layout={{  width:1000, height:500, title: 'Datos Historicos T91161' }}
            config = {{responsive: true}}
        />
    </div>
</div>
```

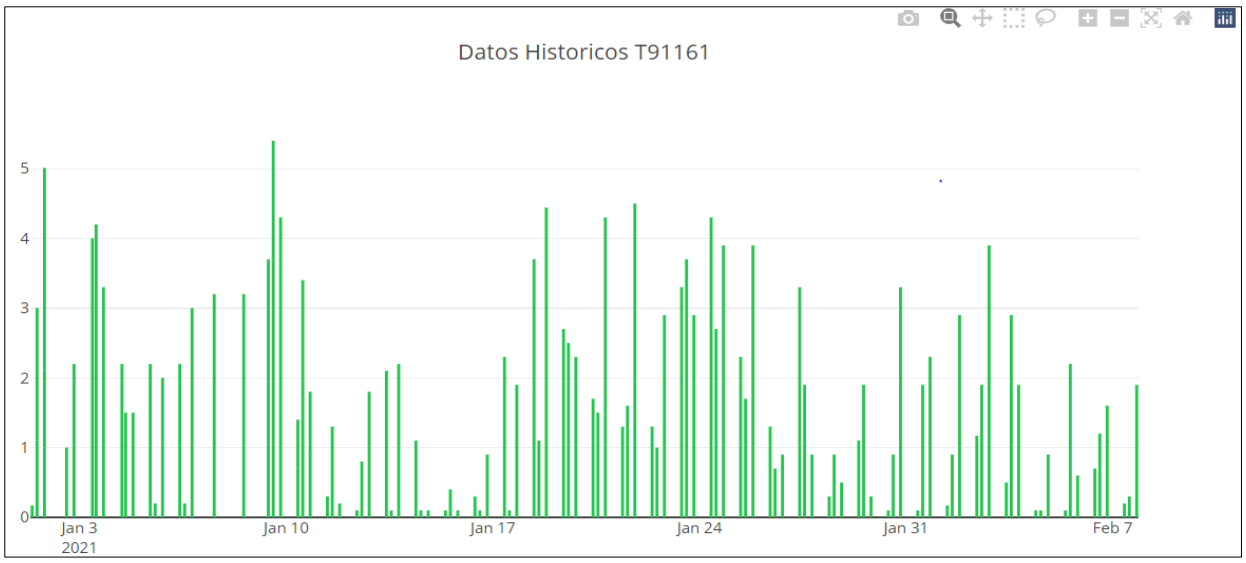
Nota: *Bloque de código para implementar las respectivas graficas estadísticas.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 61 se puede observar la gráfica estadística referente a los datos de la humedad relativa del aire y como esto ha ido cambiando a lo largo de los meses, de igual manera se puede visualizar de la misma forma para los demás datos hidrometereológicos, las cuales son útiles para el INAMHI.

Figura 61.

Gráfico de barras de los datos.



Nota: *Resultado de datos de la tabla T91161. Elaborado por: (Apolo & Taco, 2022)*

CAPITULO IV

4 PRUEBAS Y VALIDACIÓN DEL SISTEMA

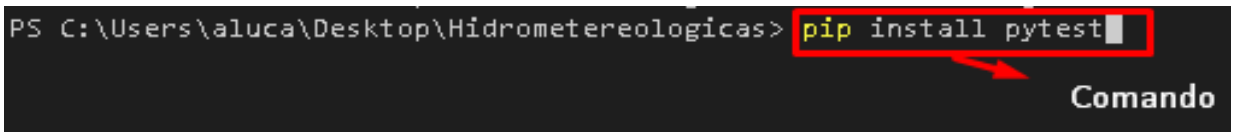
4.1. Pruebas unitarias

Para la ejecución de las pruebas unitarias se hace el uso de la librería *pytest* de Python, la cual se trae alguno de los modelos de la aplicación para verificar su correcto funcionamiento. Se enviará peticiones correctas e incorrectas para observar el comportamiento de los componentes individuales del sistema.

En primer lugar, se procede a instalar *pytest* con el comando mostrado en la Figura 62

Figura 62.

Instalación de pytest.



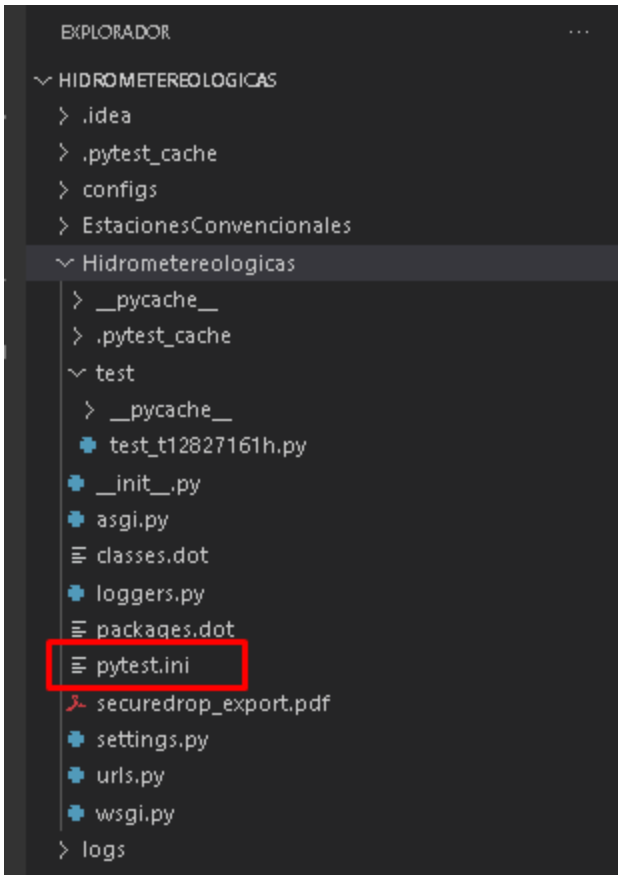
Nota: *Comando para instalar la librería pytest para realizar pruebas unitarias.*

Elaborado por: (Apolo & Taco, 2022)

Una vez instalada la librería es necesario configurarla y crear un archivo en el directorio raíz del proyecto con el nombre *pytest.ini* (Figura 65).

Figura 63.

Muestra de ubicación del archivo pytest.ini

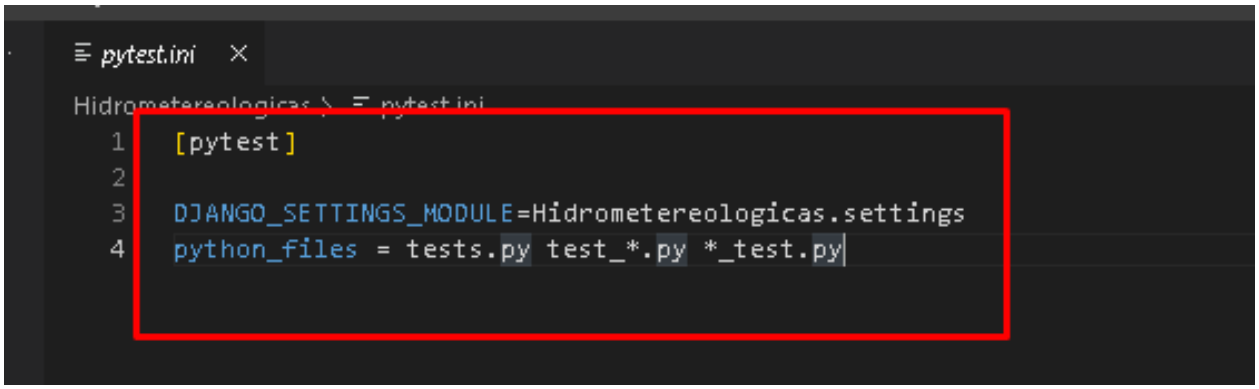


Nota: *Creación del archivo pytest.ini. Elaborado por: (Apolo & Taco, 2022)*

Una vez creado el archivo pytest.ini se procede a sobrescribirlo y añadir los siguientes segmentos de código.

Figura 64.

Modificación de contenido en el archivo pytest.ini



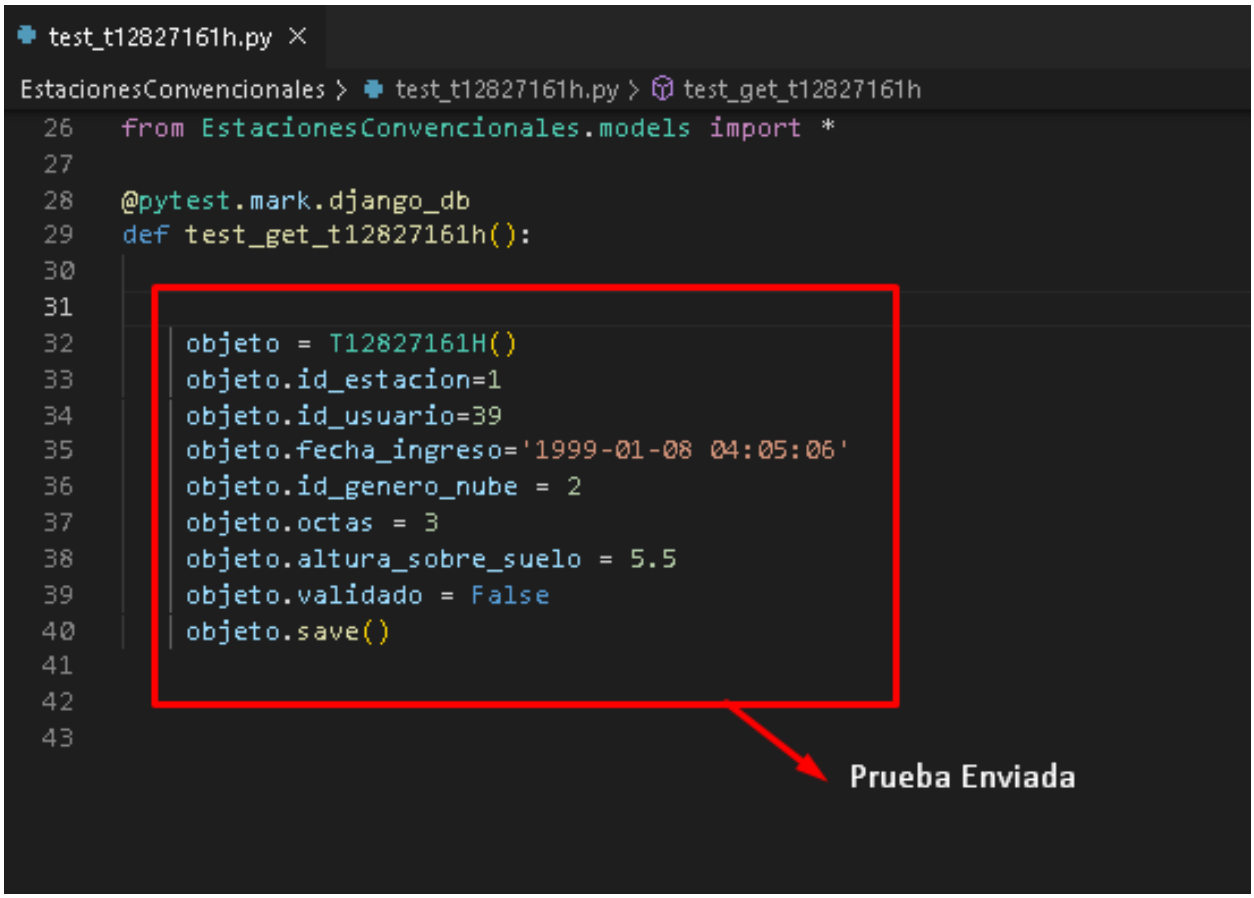
Nota: *Bloque de código para configurar las pruebas al proyecto hidrometereologicas.*

Elaborado por: (Apolo & Taco, 2022)

Una vez configurada la librería se procede a crear una carpeta con el nombre de *tests*, en la cual se alojarán las pruebas que se llevarán a cabo de los distintos componentes, se debe crear archivos diferentes para todos los modelos existentes con el siguiente formato (test_nombre_modelo.py).

Figura 65.

Envió de pruebas unitarias.



Nota: *Bloque de código del envió de datos para la prueba hacia la tabla T12827161H*

Elaborado por: (Apolo & Taco, 2022)

4.2. Pruebas de integración

Para este tipo de pruebas se usa la herramienta Postman para comprobar que todos los componentes del *ApiRest* (*Backed*) funcione correctamente de manera conjunta, por lo cual se realizan solicitudes HTTP esperando una respuesta positiva. En las siguientes figuras se visualiza como se obtiene estas respuestas acertadas de la API usando las diferentes peticiones (get, post, put, delete).

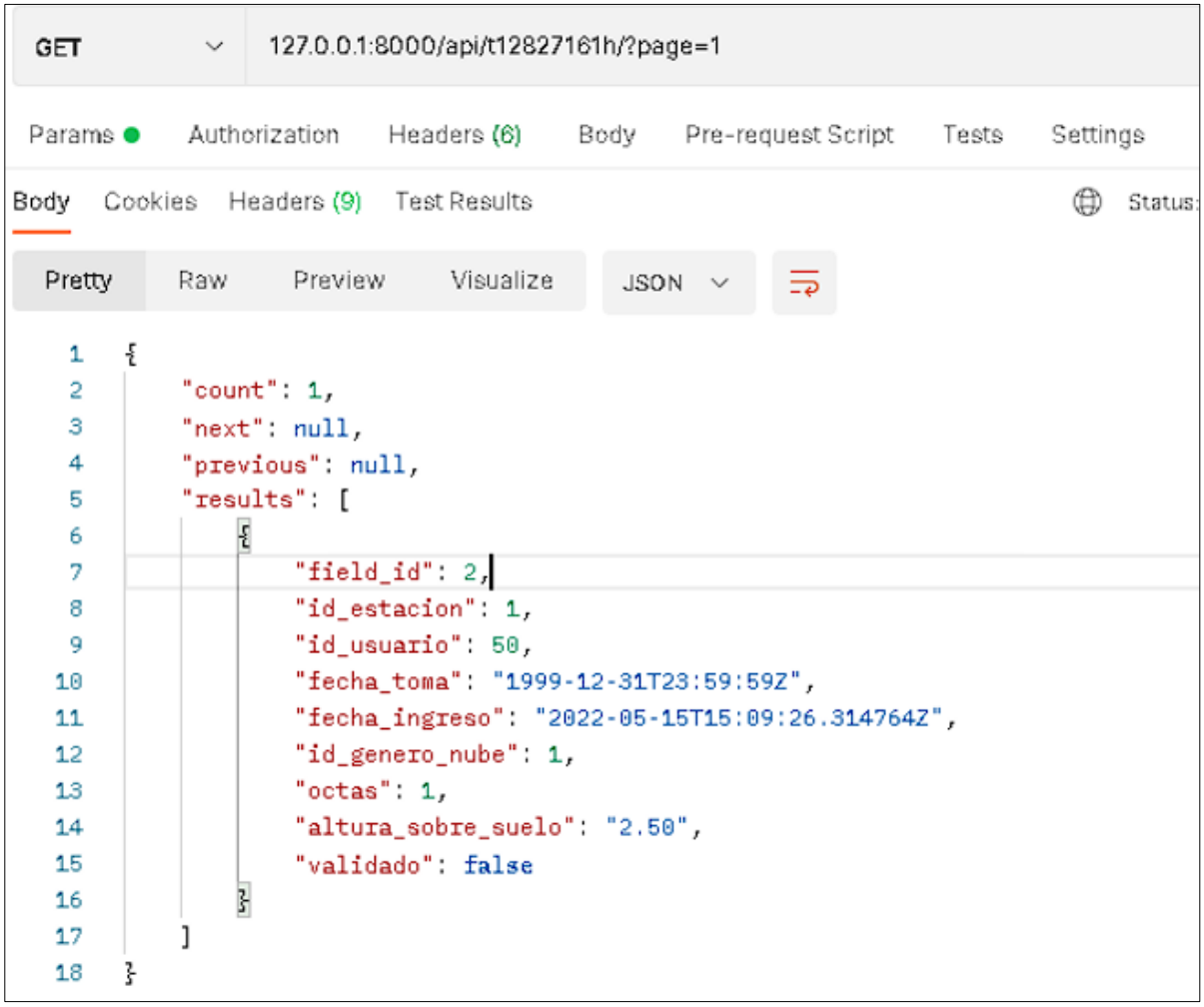
4.2.1. Prueba del Método GET

En la Figura 52 se observa que se realiza una petición a un *endpoint* haciendo uso del método GET que ofrece Postman, retornando una respuesta con todo un listado de información enviado por el

servicio web, a parte de la información recibida también se obtiene los parámetros *count*, *previous* y *next*, el cual ayuda a la implementación de paginación en el *frontend*.

Figura 66.

Llamada a un endpoint con el método GET.



Nota: *Petición a un enpoint mediante Postman haciendo uso del método GETy proyectado en un formato JSON. Elaborado por: (Apolo & Taco, 2022)*

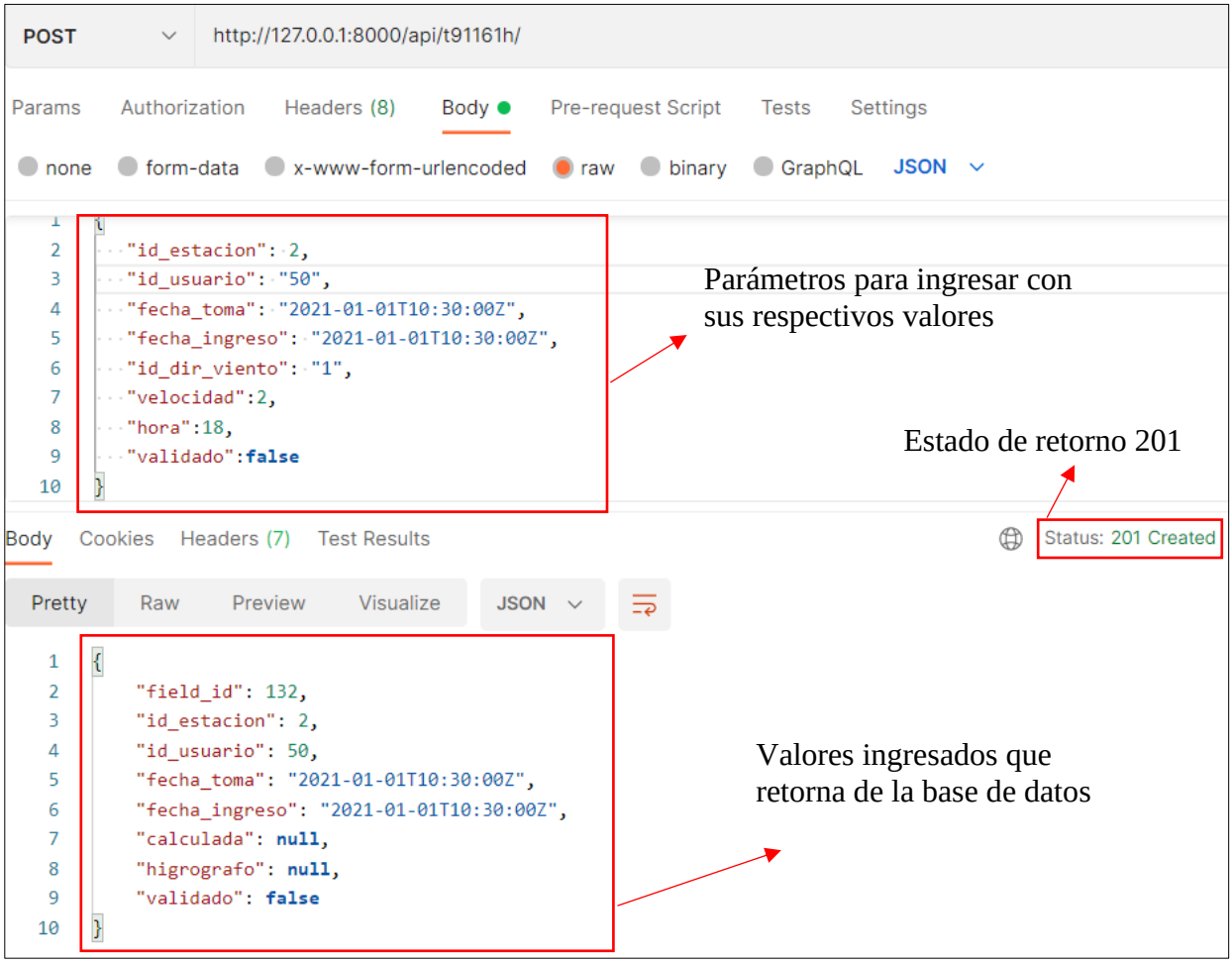
4.2.2. Prueba del método POST

En la Figura 53 se observa el envío de datos colocados en el body, cave recalcar que dependerá de los parámetros en el que se desea agregar la información, estos son enviadas hacia un *enpoint* y

esto, a su vez, retorna un estado 201 que significa creado exitosamente, además retorna los valores ingresados con su respectivo clave primaria (id).

Figura 67.

Llamada a un endpoint con el método POST

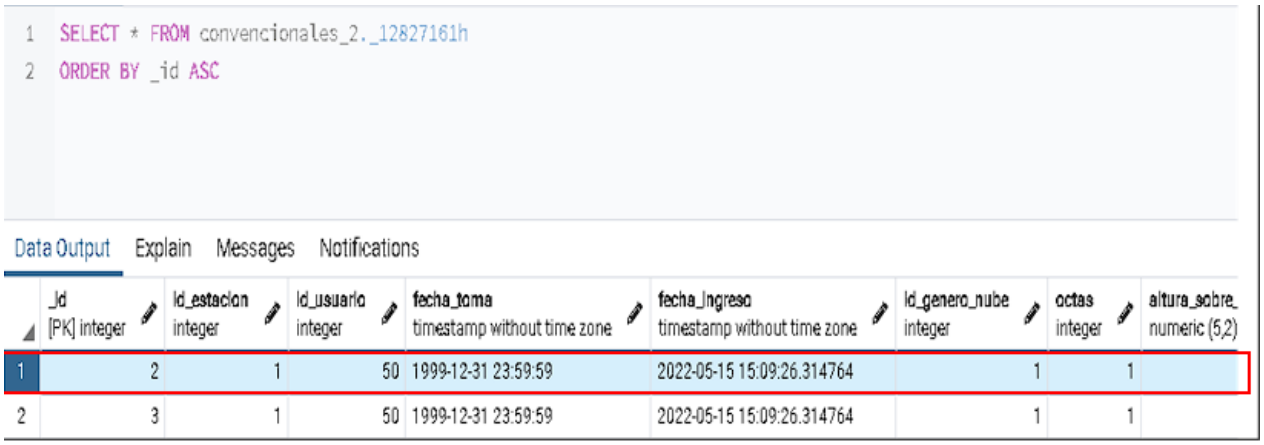


Nota: *Petición a un enpoint mediante Postman haciendo uso del método GET y proyectado en un formato JSON. Elaborado por: (Apolo & Taco, 2022).*

En la Figura 68 se visualiza la información realizada por medio de una consulta a la base de datos de la tabla respectiva donde anteriormente se envió los datos mediante Postman.

Figura 68.

Muestra exitosa de datos ingresados.



```
1  SELECT * FROM convencionales_2._12827161h
2  ORDER BY _id ASC
```

Data Output | Explain | Messages | Notifications

	_id [PK] integer	id_estacion integer	id_usuario integer	fecha_toma timestamp without time zone	fecha_ingreso timestamp without time zone	id_genero_nube integer	octas integer	altura_sobre_ numeric (5,2)
1	2	1	50	1999-12-31 23:59:59	2022-05-15 15:09:26.314764	1	1	
2	3	1	50	1999-12-31 23:59:59	2022-05-15 15:09:26.314764	1	1	

Nota: *Consulta de tipo select a la tabla de _12827161h que se encuentra en el esquema convencionales_2. Elaborado por: (Apolo & Taco, 2022)*

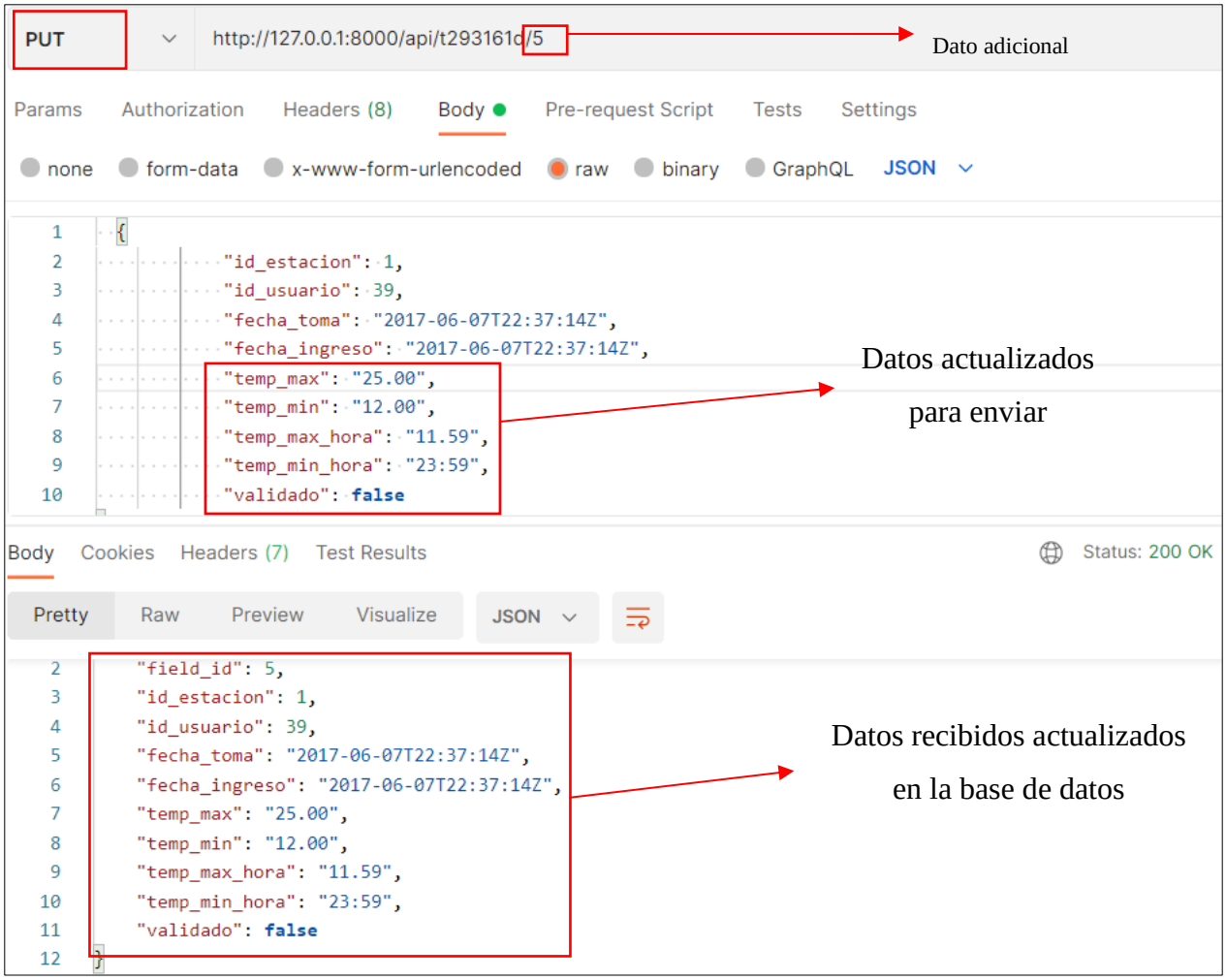
4.2.3. Prueba del método PUT

En la Figura 69.

Llamada a un endpoint con el método PUT se presenta una petición de tipo PUT hacia el *endpoint* de las estaciones convencionales, pasando como un valor adicional al final de la url, el cual viene a ser el id de ciertos datos que contiene la base, a su vez, también se debe agregar los parámetros con el valor correspondiente a ser actualizados donde el servidor devuelve un estado exitoso (200) o fallido (204).

Figura 69.

Llamada a un endpoint con el método PUT



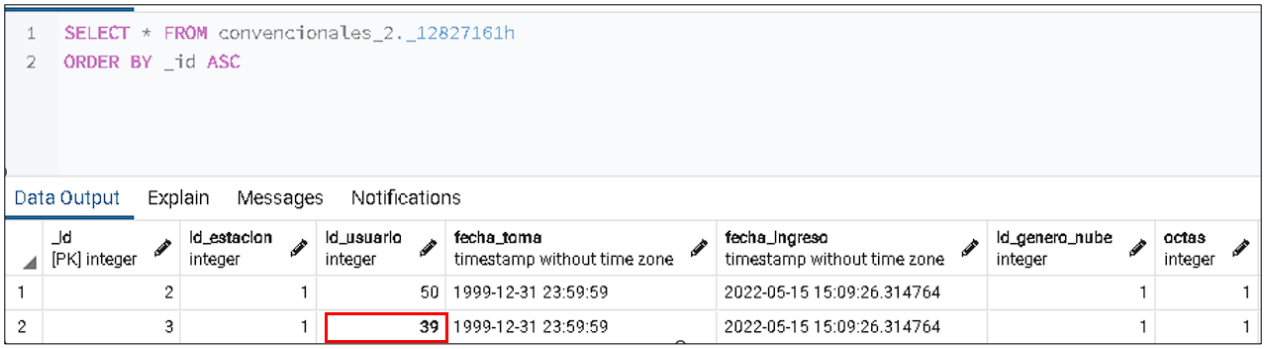
Nota: *Envió de datos con el método PUT hacia el servicio web para actualizar información de un determinado id.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 70 se realiza una consulta a la tabla _12827161h, en él se observa el valor actualizado en la columna de id_usuario que anteriormente tenía el valor de 50, teniendo así un resultado exitoso ante la prueba realizada con herramienta Postman.

Figura 70.

Muestra exitosa de la actualización de datos.



	_id [PK] integer	id_estacion integer	id_usuario integer	fecha_toma timestamp without time zone	fecha_ingreso timestamp without time zone	id_genero_nube integer	octas integer
1	2	1	50	1999-12-31 23:59:59	2022-05-15 15:09:26.314764	1	1
2	3	1	39	1999-12-31 23:59:59	2022-05-15 15:09:26.314764	1	1

Nota: *Consulta para obtener todos los datos que contiene la tabla _12827161h obteniendo como resultado 2 filas. Elaborado por: (Apolo & Taco, 2022)*

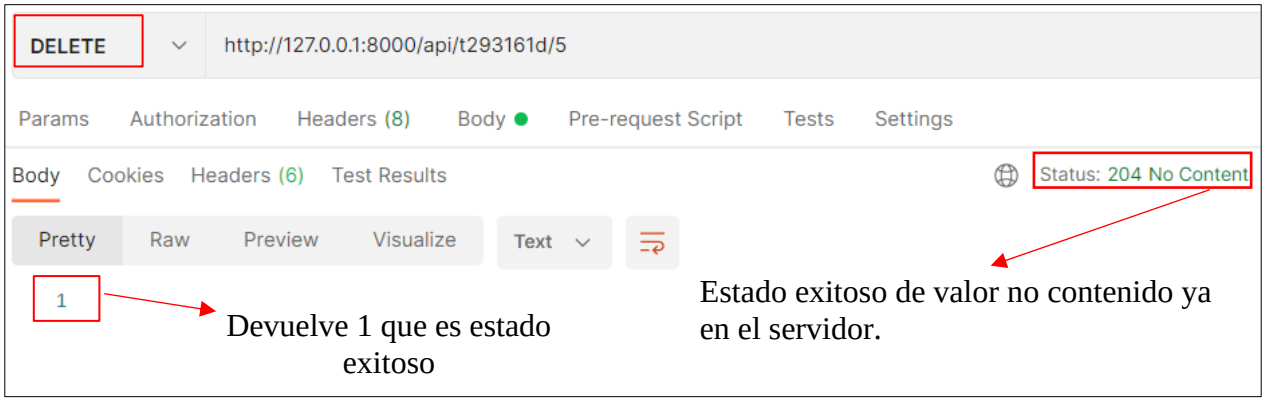
4.2.4. Prueba del método DELETE

En la Figura 71.

Llamada a un endpoint con el método DELETE se hace uso del método DELETE que ofrece la herramienta, el cual, como su propio nombre lo dice es utilizado para eliminar un dato en específico, de igual manera que el método PUT se envía el valor del id mediante la url donde el servidor retornara un estado dependiendo si la petición se realizó con éxito o no.

Figura 71.

Llamada a un endpoint con el método DELETE



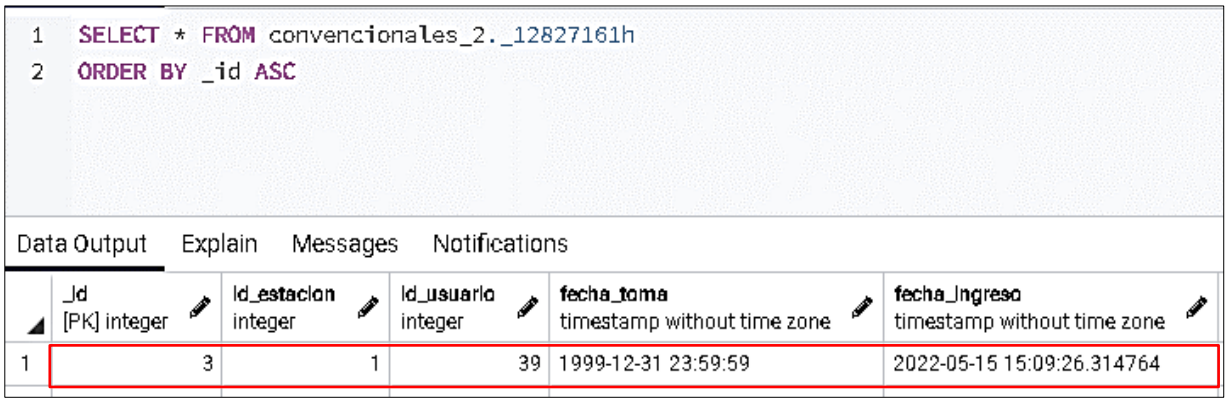
Nota: *Uso del método DELETE para eliminar información de la base de datos.*

Elaborado por: (Apolo & Taco, 2022)

En la Figura 72 se realiza una consulta de tipo *select * from* convencionales_2._12827161h donde retorna todos los valores que contiene la tabla en este caso solamente quedando un solo dato, en la cual se confirma la funcionalidad del método DELETE empleado en la API.

Figura 72.

Muestra exitosa de la eliminación de datos con respecto al id 5



Nota: *Consulta para obtener todos los datos que contiene la tabla _12827161h obteniendo como resultado una fila. Elaborado por: (Apolo & Taco, 2022)*

4.3. Pruebas de aceptación

Los módulos construidos en este proyecto formaran parte de un software completo que se irá construyendo con el tiempo, por este motivo los servidores públicos del INAMHI no requirieron realizar pruebas de aceptación puesto que este tipo de pruebas se las desarrolla cuando el software está prácticamente terminado y se requiere verificar si este cumple con las necesidades y expectativas del usuario final.

4.4. Pruebas de estrés

En este caso las pruebas de estrés son realizadas únicamente por rutina, ya que, el aplicativo no será desplegado en un servidor web público, sino en uno privado. Esto significa que no recibirá una carga excesiva de peticiones, ya que, será usado internamente por los servidores públicos del INAMHI. Para la ejecución de estas pruebas se ha usado JMeter con sus diferentes herramientas,

las cuales ayudaron a medir el rendimiento del sistema(estrés). En la Tabla 15 se pueden observar las características que tiene el servidor en el cual se ejecutaran las pruebas de estrés.

Tabla 15.

Características del servidor

Servidor	
Componente	Descripción
CPU	Hiperconvrgente 3.5Ghz
RAM	16 GB
Núcleos	4

***Nota**: Descripción de componentes del servidor en el cual se ejecutaron las pruebas*

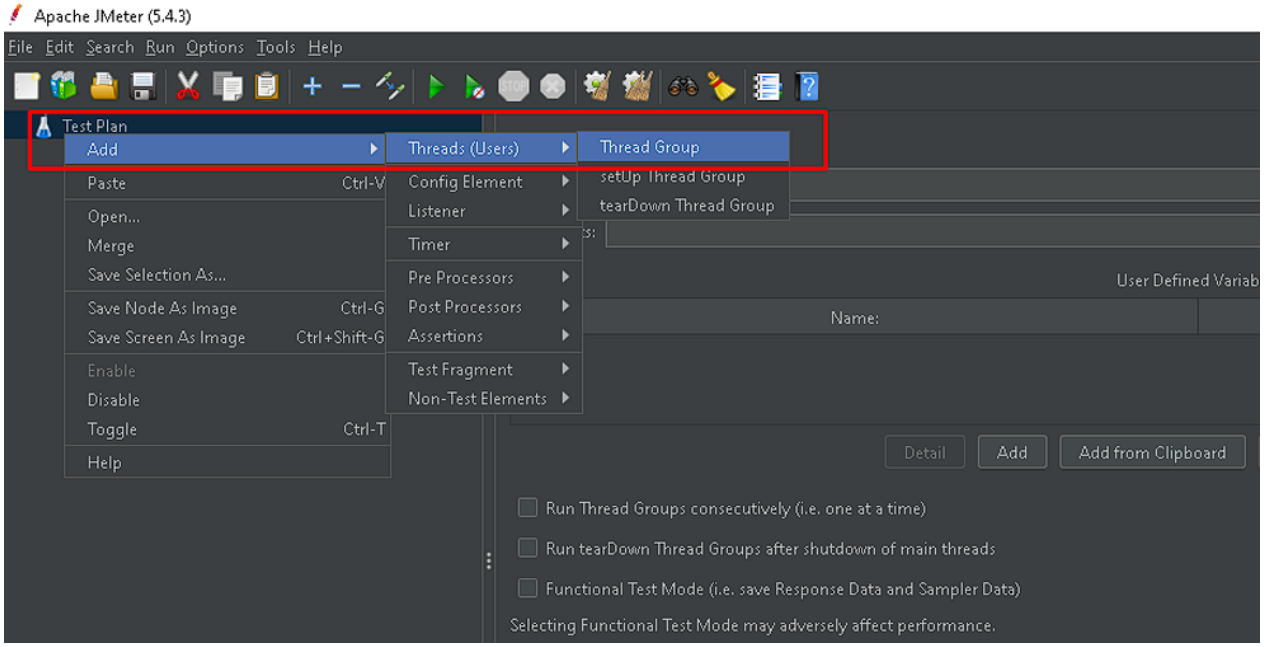
4.4.1. Creación de Group tree en JMeter

Para empezar las pruebas de estrés es necesario crear un árbol de usuarios en *JMeter*, ver la Figura 78.

Resultado de la petición realizada, esto ayudara a simular en número de usuarios que accederán simultáneamente al servidor.

Figura 73.

Creación de grupo de pruebas



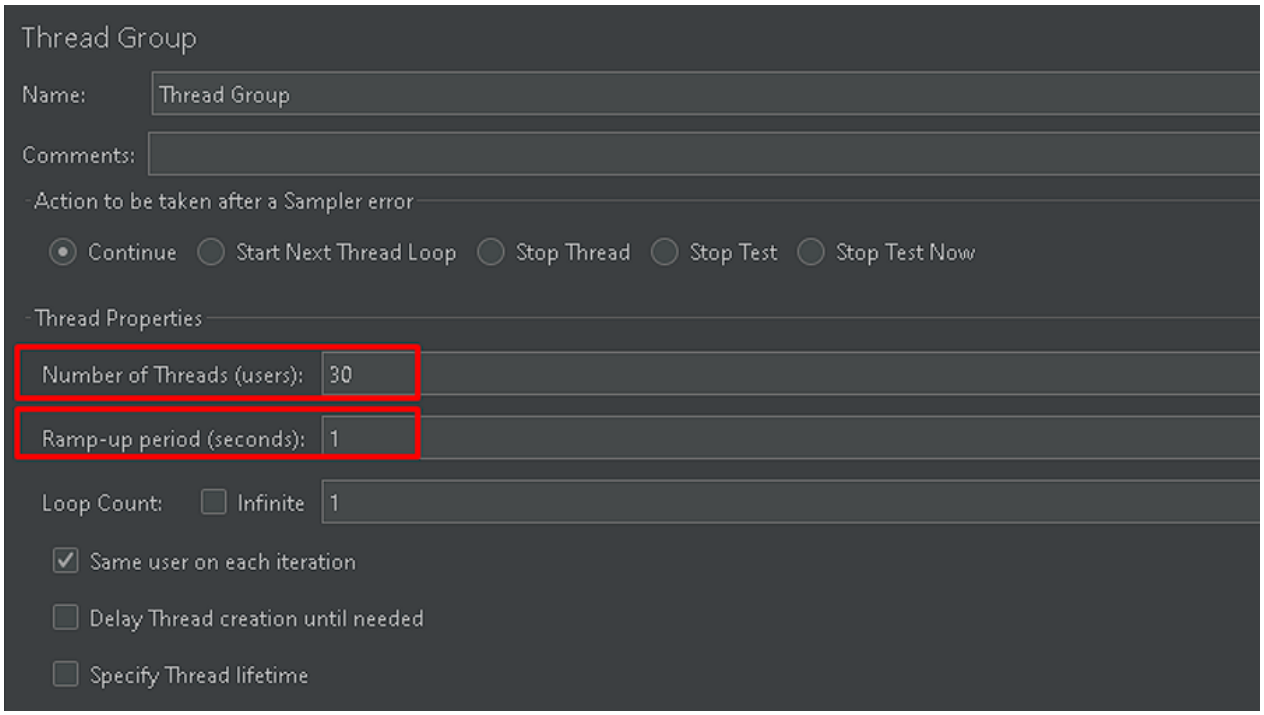
Nota: *Creación del grupo de pruebas en JMeter. Elaborado por: (Apolo & Taco, 2022)*

4.4.2. Configuración de tree Group

Para configurar esta herramienta es necesario ingresar el número de usuarios simulados que intentaran acceder al servidor, a su vez se debe ingresar el intervalo de tiempo, en el cual se realzaran la peticiones. En este caso solo se ha elegido 30 usuarios, ya que el software solo será utilizado por pocas personas seleccionadas del INAMHI, también se ingresado el valor de intervalo de un segundo puesto que este es el tiempo mínimo que los usuarios pueden encontrar controversias al momento de realizar peticiones simultáneamente.

Figura 74.

Prueba de peticiones al servidor con cierto número de usuarios en un tiempo determinado.



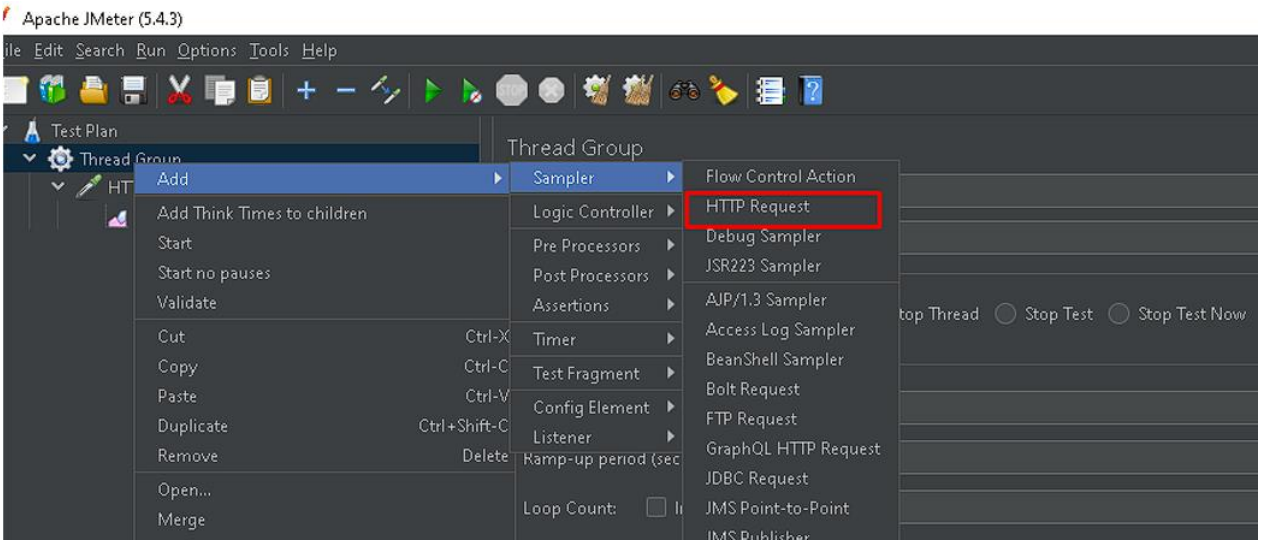
Nota: *Configuración de para la prueba de estrés con 30 usuarios ingresando en 1 segundo. Elaborado por: (Apolo & Taco, 2022)*

4.4.3. Creación de HTTP Request

En la Figura 75 se muestra cómo crear un HTTP *Request* en *JMeter*, esta herramienta ayuda a simular el envío de múltiples peticiones al servidor y así comprobar cuál es el límite de carga que este soporta.

Figura 75.

Creación de HTTP request para las peticiones al servidor.



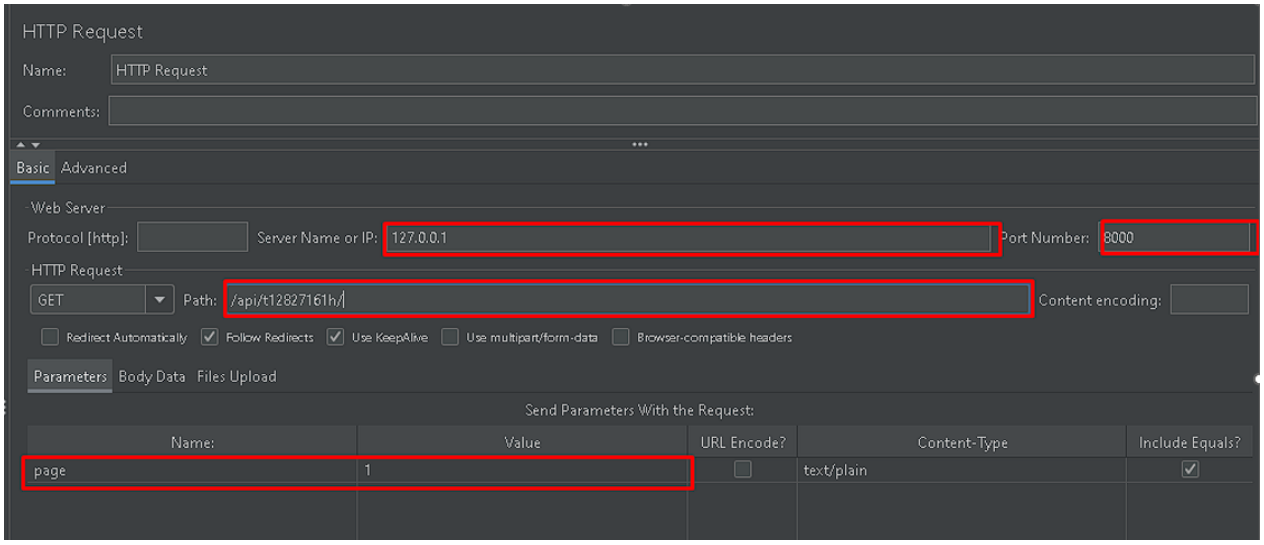
Nota: *Configuración para llamar a un determinado endpoint. Elaborado por: (Apolo & Taco, 2022)*

4.4.4. Configuración de HTTP Request en JMeter

Para configurar esta herramienta en necesario ingresar la url del servidor, a su vez, se debe ingresar el puerto donde se está ejecutando la aplicación, también se debe ingresar la estructura que lleva las peticiones al momento de ser receptadas por el *backend*. En la Figura 76 se observa cómo se configuró JMeter para el envío de peticiones al servidor.

Figura 76.

Configuración de la url del servidor a realizar la prueba de carga y estrés.



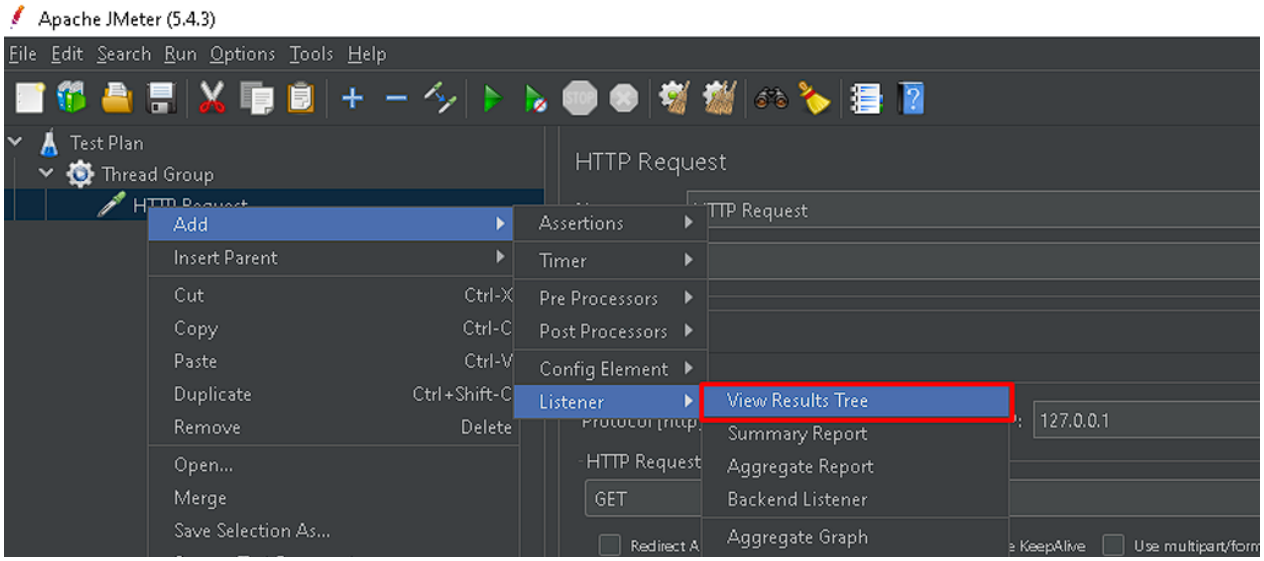
Nota: *Llamada al endpoint /api/t12827161h. Elaborado por: (Apolo & Taco, 2022)*

4.4.5. Creación de View Resul Tree en JMeter

En la Figura 77 se observa cómo crear un *View Resul Tree* en JMeter, esta herramienta únicamente sirve para observar los resultados obtenidos al momento de ejecutar las pruebas de rendimiento en el servidor.

Figura 77.

Creación para la vista del árbol de resultados

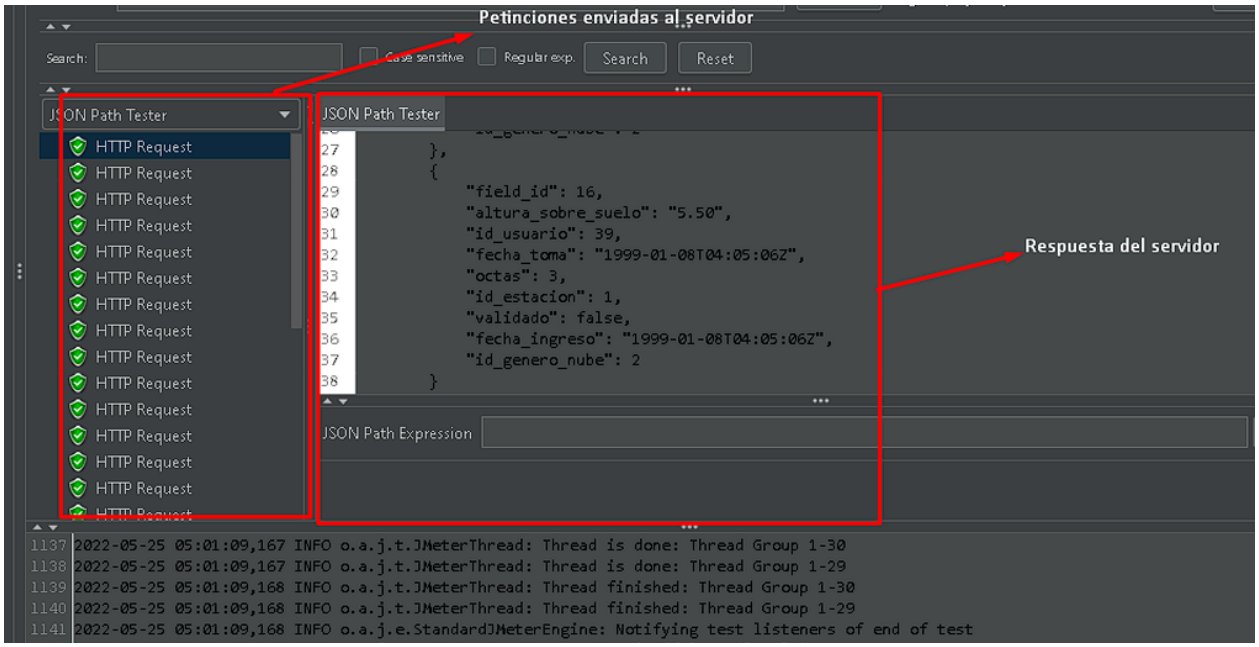


4.4.6. Resultado de View Resul Tree

Se puede observar las múltiples peticiones enviadas al servidor sin error alguno, esto se debe a que las misma no son problema para ser receptadas por el servidor de manera simultánea, lo cual notifica que el programa podrá ser usado por al menos 30 usuarios sin ningún problema de controversia en él envió de solicitudes.

Figura 78.

Resultado de la petición realizada



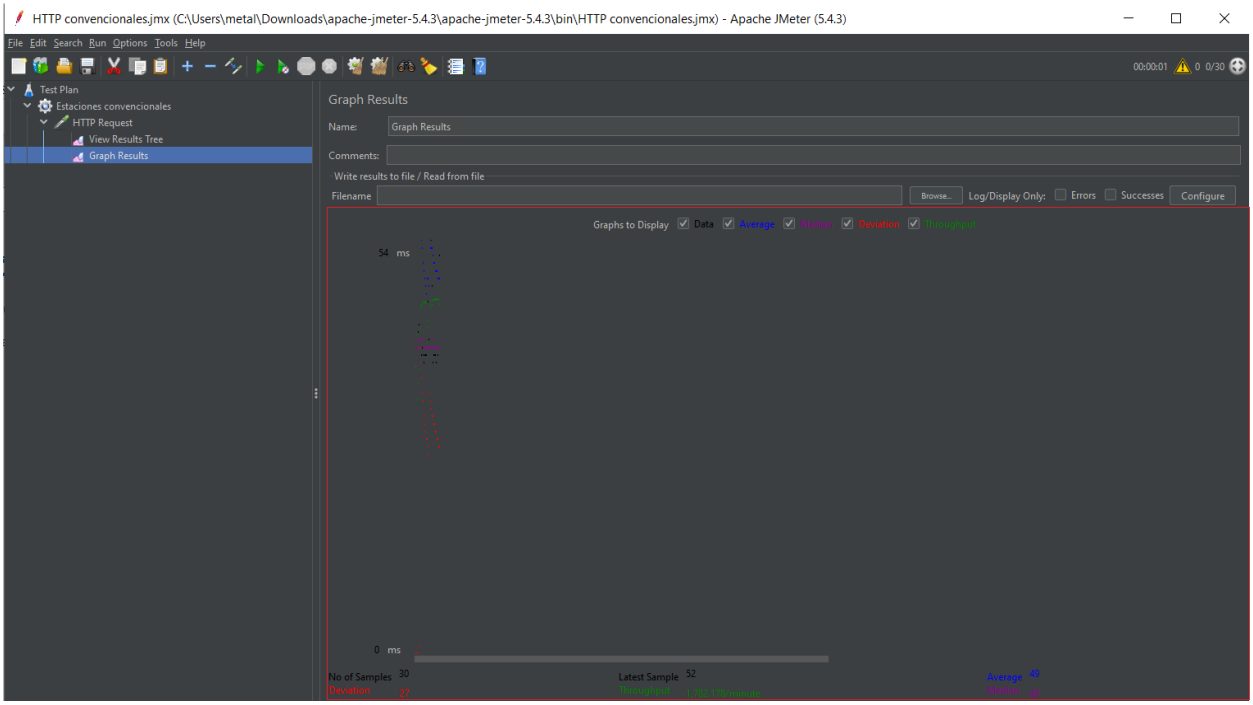
Nota: *árbol de resultado realizada al servidor. Elaborado por: (Apolo & Taco, 2022).*

4.4.7. Visualización de resultados en Graph Result

En la Figura 79 se observa una gráfica de información de los resultados obtenidos en el momento de realizar las pruebas, se puede observar diferentes parámetros como el promedio, desviación, media y rendimiento, los cuales esta identificados con diferentes colores, estos elementos indican el tiempo que se toma el servidor en ejecutar o resolver cada una de las peticiones enviadas por los usuarios y sobretodo, cuál es el número máximo de usuarios/minuto que se pueden conectar.

Figura 79.

Resultado en gráfico



Nota: *resultado gráfico de la data obtenida en un determinado tiempo promedio. Elaborado por: (Apolo & Taco, 2022)*

CONCLUSIONES

- En este trabajo se desarrolló una aplicación web para la visualización de datos hidrometereológicos de estaciones convencionales del INAMHI, esta aplicación cumple satisfactoriamente con los requisitos solicitados por la institución y con las pruebas de rendimiento y estrés a las que fue sometida.
- La aplicación web se diseñó usando una arquitectura modular. De esta forma, cuenta con un módulo para la validación de los datos inconsistentes en la base de datos del INAMHI y un módulo para generar reportes de los datos históricos mediante el uso de gráficos estadísticos.
- Del desarrollo del proyecto se concluye que la utilización del *framework* Django simplifica la codificación del software a través de diferentes componentes. Además, su arquitectura modelo vista *template* permite separar la lógica de negocio con la interfaz gráfica de usuario, ayudando a tener un mayor control y organización en el código de los aplicativos, y permitiendo que los sistemas sean mantenibles y escalables en el tiempo.
- Por otra parte, la librería React es una excelente opción para el desarrollo *FrontEnd*, ya que es capaz de separar de manera individual los elementes visuales que contiene un sitio web, lo que permite llevar una mejor jerarquía en la construcción de estos. React también permite que los componentes trabajen de manera autónoma, de modo que se actualizan solo si están siendo manipulados por los usuarios finales. Esto ayuda a reducir el consumo de recursos de los dispositivos y evita recargar de manera completa la aplicación para visualizar cambios de los eventos realizados.
- Al utilizar el material-UI, la implementación de la interfaz gráfica de usuario fue más sencilla, gracias a la cantidad de componentes que brinda para la creación de páginas web,

con estilos amigables y empleando los principios de funcionalidad de diseño de la *UI-UX*. Esta librería también facilitó la implementación de una paginación personalizada combinando código propio y código de la herramienta desarrollada por Material-UI.

RECOMENDACIONES

- Para el desarrollo de cualquier proyecto web es necesario realizar un análisis de requerimientos adecuado y completo, para que no sea necesario aumentar funcionalidades, no contempladas en el diseño, que puedan complicar la fase de construcción de la aplicación.
- Se deben realizar las pruebas unitarias a todos los componentes de proyecto durante el desarrollo, aunque demande más tiempo, ya que, permitirá encontrar y mitigar los errores no contemplados en el diseño y construcción del sistema y, también evitará que se acumulen los errores hasta el final del proyecto.
- La base de datos (DB) usada en un proyecto debe pasar por al menos tres formas normales iniciales, para que durante la fase de construcción de la aplicación no existan controversias o redundancia de datos. También es recomendable generar un análisis minucioso al esquema entidad relación para determinar tablas faltantes y evitar cambios posteriores a la DB.
- El uso de *Swagger* permite documentar una aplicación, facilitando la construcción de futuros módulos. Esta herramienta incluye documentación detallada de los modelos, las vistas y los serializarles de la API desarrollada. En el caso de este proyecto, la documentación servirá como guía para futuros desarrollos, disminuyendo la curva de comprensión de la lógica del sistema.

- En algunos proyectos no es necesario llevar las pruebas de estrés a niveles extremos, ya que esto depende del número de usuarios que accederá a la aplicación. En el caso de este proyecto, la aplicación será utilizada para uso interno, únicamente por los funcionarios de la institución. Si se desea desplegar la aplicación para todo el público, es recomendable realizar pruebas de rendimiento robustas para identificar el límite de saturación del sistema y ajustar la capacidad del sistema al nuevo número de usuarios finales.

REFERENCIAS

Abad, R. (4 de abril de 2017). *Ramón Abad y punto net*. Obtenido de Ramón Abad y punto net: http://ramonabadypuntonet.org/2017/04/04/que-es-swagger/

Augusto, A. C. (2017). Propuesta Metodológica Para Migración De Sistemas. *Propuesta Metodológica Para Migración De Sistemas.* Escuela Politécnica Nacional, Quito, Ecuador.

CASTILLO, J., & CEVALLOS, C. S. (2021). CREACIÓN DE UN REPOSITORIO WEB PARA DATOS DE ESTACIONES METEOROLOGICAS. *CREACIÓN DE UN REPOSITORIO WEB PARA DATOS DE ESTACIONES METEOROLOGICAS.* UNIVERSIDAD POLITÉCNICA SALESIANA, Quito.

Chávez, D. P. (2010). RESPALDO AUTOMATICOS DE BASES DE DATOS POSTGRES. *RESPALDO AUTOMATICOS DE BASES DE DATOS POSTGRES.* UNIVERSIDAD DE GUAYAQUIL, GUAYAQUIL.

Chávez, R. D. (2016). *ANÁLISIS DEL FRAMEWORK DJANGO PARA IMPLEMENTAR.* Ibarra: Universidad del Norte.

Espinoza, M. E., & Ramires Parra, O. E. (2018). Modelamiento de la precipitacion en las zona urbana de la ciudad de cuenca. *Modelamiento de la precipitacion en las zona urbana de la ciudad de cuenca.* Universidad politecnica salecina, Cuenca, Ecuador. Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/15799/1/UPS-CT007737.pdf

Estévez, S. P., Lamela, N. R., & Morales, C. A. (2016). API de servicios web orientados a accesibilidad. *API de servicios web orientados a accesibilidad.* Universidad Complutense de Madrid, Madrid. Obtenido de

https://eprints.ucm.es/id/eprint/38686/1/Memoria_API%20de%20servicios%20web%20de%20accesibilidad.pdf

García, C. d. (2019). Guía práctica para la publicación de Datos Abiertos usando APIs. *Guía práctica para la publicación de Datos Abiertos usando APIs.* Madrid. Obtenido de https://datos.gob.es/sites/default/files/doc/file/guia_publicacion_apis.pdf

Garcia, S. (2015). Escribe tu primera aplicación Web usando Django. *Escribe tu primera aplicación Web usando Django.* Django Software Corporation, Boston. Obtenido de https://www.academia.edu/14337010/Tutorial_de_introducci%C3%B3n_a_Django_Escribe_tu_primera_aplicaci%C3%B3n_Web_usando

Gutiérrez, J. D. (2017). DESARROLLO DE UNA APLICACIÓN MÓVIL QUE PERMITA LA INTERACCIÓN. *DESARROLLO DE UNA APLICACIÓN MÓVIL QUE PERMITA LA INTERACCIÓN.* UNIVERSIDAD PONTIFICIA BOLIVARIANA, Medellín.

Huan Wu, M. H. (2016). Hydrometeorological Hazards: Monitoring, Forecasting,. *Hydrometeorological Hazards: Monitoring, Forecasting,.* University of Maryland, College Park. Obtenido de Pilcomayo.net: https://downloads.hindawi.com/journals/amete/2016/2367939.pdf

Julián Adams, Erick Pérez. (2018). *DESARROLLO DEL MÓDULO DE REGISTRO DE USUARIOS.* GUAYAQUIL: ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL.

K.K, K. V. (2009). *Software Testing Tools: Covering WinRunner, Silk Test, LoadRunner, JMeter.* Dreamtech, London, United Kingdom.

KNOWLEDGE. (2013). *CONOCIMIENTO DE SCRUM (GUÍA SBOK).* Arizona: SCRUMstudy. Obtenido de https://www.tenstep.ec/portal/images/pdfs/Suscripciones_TenStep/Silver/SCRUMstudy_GUIA_SBOK_espanol.pdf

López, D., & May, E. (2017). Arquitectura de Software basada en Microservicios para. *Arquitectura de Software basada en Microservicios para.* Universidad Técnica del Norte, Instituto de Posgrados, Ibarra, Imbabura, Ecuador.

Marisol Andrades, C. M. (2012). Fundamentos de Climatología. *Fundamentos de Climatología.* Universidad de la Rioja, Quito.

Martin, R. N., & Ollé, G. N. (2020). Agilizando los cambios de UI-UX sobre el ambiente productivo mediante Figma. *Agilizando los cambios de UI-UX sobre el ambiente productivo mediante Figma.* Universidad Nacional de la Plata, Buenos Aires.

Mejías, P. U. (22 de junio de 2020). *Tech Club Tajamar*. Obtenido de Tech Club Tajamar: https://techclub.tajamar.es/material-ui-caching-en-react/#:~:text=Material%2DUI%20es%20una%20biblioteca,js.

Montilla, I. F. (19 de Diciembre de 2014). *IFFM*. Obtenido de IFFM: https://iffm.me/cosas-que-he-aprendido-mixins.html

Paredes, C. L. (2015). *ESTUDIO DEL IMPACTO DEL USO DEL SISTEMA DE CONTROL DE RIESGO.* ESPOCH: Riobamba.

Postman, M. A. (22 de 04 de 2022). *Postman.com*. Obtenido de Postman.com: https://www.postman.com/product/what-is-postman/

Seville, R. (2017). *SAP OpenUI5 for Mobile BI and Analytics.* Espresso Tutorials Gmbh.