UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE

COMPUTACIÓN

MIGRACIÓN DE LA LIBRETA HIDROMETEOROLÓGICA PRINCIPAL DEL INAMHI A UNA ARQUITECTURA DE MICROSERVICIOS

Trabajo de titulación previo a la obtención del

Título de: Ingeniero en Ciencias de la Computación

AUTOR: Fabián Sebastián Garrido Mena

TUTOR: Paulina Adriana Morillo Alcívar

Quito-Ecuador

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

DEL TRABAJO DE TITULACIÓN

Yo, Fabián Sebastián Garrido Mena con documento de identificación Nº 1724781966 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo 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, 16 de Marzo del año 2022

Atentamente,

.....

Fabián Sebastián Garrido Mena

1724781966

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

Yo, Fabián Sebastián Garrido Mena con documento de identificación N° 1724781966, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del proyecto técnico: "Migración de la Libreta Hidrometeorológica Principal del Inamhi a una Arquitectura de Microservicios", el cual ha sido desarrollado para optar por el título de Ingeniero en Ciencias de la Computación, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

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

Quito, 16 de Marzo del año 2022

Atentamente,

.....

Fabián Sebastián Garrido Mena

1724781966

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Paulina Adriana Morillo Alcívar con documento de identificación Nº 1715646574, docente de la Universidad Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: MIGRACIÓN DE LA LIBRETA HIDROMETEOROLÓGICA PRINCIPAL DEL INAMHI A UNA ARQUITECTURA DE MICROSERVICIOS realizado por Fabián Sebastián Garrido Mena con documento de identificación Nº 1724781966, 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, 16 de Marzo del año 2022

Atentamente,

.....

Ing. Paulina Adriana Morillo Alcívar, MsC

1715646574

Dedicatoria

Dedico está tesis a la persona que pueda leerla, tal vez le pueda servir este documento como información, pero quisiera decirle algo más importante, si a veces no sabes valorar lo que tienes, entonces por favor pido que siempre valores lo que tienes y sepas equilibrar bien tu vida profesional, estudiantil y personal, ya que no existe ninguna garantía de que te vaya bien en el mundo exterior, por más recto que seas o cualquier cosa que creas, el mundo de afuera es bello pero se debe conocer como vivirlo, no te sientas solo si crees que no hay solución a tu situación, a veces es bueno aplicar la frase que se dice comúnmente ”para todo se tiene solución excepto la muerte”, porque es cierto, yo solo acotaría que a veces hay varias soluciones y debes buscarlas con paciencia, aunque a veces no coincida con todos lo criterios se que la vas a encontrar, no hay una única solución a tus problemas, tal vez en algunas operaciones matemáticas, pero en tu vida puede haber varias soluciones. Finalmente querido lector quiero decirte que estimo mucho que hayas leído la dedicatoria, y te invito a seguir viviendo feliz y estudiar mucho con astucia, no solo para ganar prestigio, tu puedes y sigue adelante que no estás solo, como puedes leer, si este criterio que te estoy mostrando es parecido al tuyo, y no has encontrado a nadie que comparta este criterio, pues te demuestro que no eres el único, solo debes buscar a estás personas, ya verás que todo saldrá bien.

Fabián Sebastián Garrido Mena

AGRADECIMIENTOS

Quiero agradecer a Dios, debido a que soy creyente, y por eso le agradezco mi vida, y mis estudios, que son privilegios, que he podido tener y he podido sobrellevar, gracias a mi amigos, familia y novia.

También un agradecimiento especial a mi tutora, por guiarme en la realización de la tesis.

Fabián Sebastián Garrido Mena

Índice General

Índice de Tablas

Índice de Figuras

RESUMEN

Este proyecto muestra el desarrollo de la migración del *backend* de la libreta hidrometeorológica, del Instituto Nacional de Meteorología e Hidrología del Ecuador, a una arquitectura de microservicios, empleando herramientas de software libre como Python, Django, PostgreSQL como lenguaje de programación, *framework* de desarrollo y gestor de base de datos, respectivamente.

El proyecto está enfocado en la capa de persistencia del software, es decir, que la aplicación va a permitir el acceso, la modificación, la eliminación de los datos de la libreta hidrometeorológica, a través de la implementación de una API que también posibilita el intercambio de información por medio de archivos en formato JSON. De igual forma, los módulos de petición se realizaron utilizando el protocolo HTTP y SQL siguiendo la granularidad de *software* fina.

Gracias al desarrollo de este proyecto es posible que las aplicaciones del *Frontend* y la capa de datos se manejen de forma independiente. Además, la aplicación siguió el patrón de diseño modelo vista controlador, lo que favorece el mantenimiento y la escalabilidad de la API. De acuerdo a los resultados de las pruebas de funcionamiento y de código se puede concluir que la aplicación cumple con los criterios de aceptación de los requerimientos solicitados por la institución.

Palabras Clave: arquitectura de software, base de datos, *backend*, microservicios, hidrometeorológica

ABSTRACT

This project shows the development of the migration of the backend of the hydrometeorological notebook, of the National Institute of Meteorology and Hydrology of Ecuador, to a microservices architecture, using free software tools such as Python, Django, PostgreSQL as the programming language, development framework, and database manager, respectively.

The project is focused on the persistence layer of the software, that is, the application will allow access, modification, deletion of data from the hydrometeorological notebook, through the implementation of an API that also enables the exchange of information through files in JSON format. Similarly, the request modules were made using the HTTP and SQL protocol following the fine software granularity.

Thanks to the development of this project, it is possible for the Frontend applications and the data layer to be managed independently. In addition, the application followed the model-view-controller design pattern, which favors the maintainability and scalability of the API. According to the results of the performance and code tests, it can be concluded that the application meets the acceptance criteria of the requirements requested by the institution.

Keywords: software architecture, database, backend, microservices, hydrometeorological

INTRODUCCIÓN

La meteorología es un campo de la ciencia que se encarga del estudio de la atmósfera y los fenómenos asociados a ella. En el Ecuador, el organismo encargado de su análisis es el Instituto Nacional de Meteorología e Hidrología (INAMHI), que busca entre otras cosas, hacer mediciones y adquirir datos meteorológicos para implementar modelos de predicción. Esta institución cuenta con una libreta hidrometeorológica donde se almacenan diferentes datos, para diferentes aplicaciones en la web. La arquitectura actual de esta libreta es monolítica, con una base de datos (DB) no relacional, lo que dificulta la manipulación de los datos y la implementación de nuevas aplicaciones, y su integración con otros sistemas. Por lo tanto, se pretende realizar la migración de esta libreta a una arquitectura mucho más versátil y flexible.

La arquitectura de microservicios, se basa en la arquitectura de software que consiste en la organización de un sistema informático teniendo en cuenta las relaciones de los componentes del sistema (Maier, Emery, y Hilliard, 2004). La arquitectura de microservicios, son aplicaciones individuales que ofrecen servicios en diferentes proyectos, pueden estar escritos en diferentes lenguajes de programación, a su vez, estar en diversas partes del mundo por consiguiente el mantenimiento puede ser realizado por diferentes especialistas (Alfonso y Contreras, 2018). Las empresas anteriormente realizaban actualizaciones, que tomaban mucho tiempo y por eso las realizaban, una o dos veces al año. Sin embargo, en la actualidad se manejan otras arquitecturas, una de ellas la arquitectura de microservicios, con el fin de ser más competitivos, al realizar actualizaciones en periodos de tiempo más cortos (O'Connor, Elger, y Clarke, 2016).

A nivel mundial, han habido varias experiencias exitosas sobre el uso de microservicios en empresas de gran magnitud como Netflix. Esto valida la implementación de sistemas con este patrón de arquitectura. (Mas'ad Nasra, 2016). Netflix, por ejemplo, sufrió una interrupción

del servicio debido a una importante corrupción de la base de datos. Por tal razón, tomaron la decisión de separar la aplicación, pasando de servicios monolíticos a múltiples servicios independientes para aumentar la escalabilidad, la confiabilidad y la disponibilidad del software (Paul Durzan, 2020).

También hay ejemplos en América Latina, como la empresa Financial Systems Company SAC, donde tuvieron que migrar su sistema tradicional de cobranza a una arquitectura de microservicios, en el aplicativo web de cobranza digital, para expandir sus servicios. De este modo, pueden soportar envíos masivos de mensajes por canal, sin impactar en el rendimiento del sistema y manteniendo una alta disponibilidad (Villaizán, 2019).

En el Ecuador, en la región sierra, provincia de Imbabura, en el año 2017 la Universidad Técnica del Norte realizó una investigación del proceso de desarrollo de software. La investigación nos muestra, la dificultad en el mantenimiento o cambio en las funcionalidades de las aplicaciones, debido a su naturaleza monolítica (López Hinojosa, 2017).

Por lo tanto, después de la revisión de la literatura, este trabajo propone llevar cabo la migración del *backend* de la libreta hidrometeorológica del INAMHI, a una arquitectura de microservicios, usando lenguajes de código abierto como Django, para la implementación de la API (*Application Programming Interface*) y PostgreSQL para la implementación de la base datos.

Problema

La arquitectura actual de la libreta hidrometeorológica (LH) del INAMHI, es una arquitectura monolítica. Esto impide su correcto mantenimiento, escalamiento e integración con otros sistemas y aplicaciones. Por consiguiente, este trabajo responderá a la pregunta de ¿Cómo realizar la migración de la libreta hidrometeorológica del INAMHI a una arquitectura más versátil y flexible como la de microservicios? Actualmente, en el INAMHI se recolectan datos en las dife-

rentes estaciones hidrometeorológicas. Está acción se la realiza mediante su aplicativo web LH. La ausencia de una arquitectura de software enfocada a servicios, provoca pérdidas de información, dificulta el mantenimiento de la aplicación, ya que la actualización del sistema conlleva la utilización de más recursos al tener que actualizar cada componente. Esto a su vez, limita la innovación y mejoramiento continuo de la aplicación, prolongado el tiempo de desarrollo ante cualquier cambio.

El aporte del presente trabajo será analizar los componentes del aplicativo LH del INAMHI, para poder realizar el cambio a una arquitectura de microservicios, para lo cual se realizará una aplicación *backend* mediante una API, que pueda modificar los datos, en las 35 tablas de la base de datos del INAMHI, y de está manera el INAMHI podrá utilizar la API, en cualquier aplicación *Frontend* que realice en un futuro.

Objetivo General

Realizar la migración de la libreta hidrometeorológica principal del INAMHI a una arquitectura de microservicios.

Objetivo Específicos

- Analizar los requerimientos para la migración del aplicativo LH a una arquitectura de microservicios.
- Diseñar y construir el sistema con Kanban.
- Desarrollar la migración de una arquitectura monolítica a una arquitectura de microservicios.

- Evaluar el aplicativo LH con la arquitectura de microservicios mediante métricas.

Capítulo 1

Fundamentos teóricos

1.1 Meteorología

1.1.1 Definición

La meteorología consiste en el estudio de la atmósfera y se encarga de descifrar sus propiedades y fenómenos. La atmósfera tiene varias variables como presión atmosférica, temperatura, etc. Hoy en día, la meteorología es un ciencia avanzada, que toma todas la variables de la atmósfera junto con el conocimiento de la Física y la tecnología moderna para predecir el tiempo hasta con una semana de antelación (Jiménez, Capa, y Lozano, 2004).

1.1.2 Clasificación de la meteorología

La meteorología se divide en diferentes ramas como: Meteorología dinámica, meteorología física y meteorología sinóptica. La meteorología dinámica trata los movimientos y termodinámica de la atmósfera. La meteorología física estudia los efectos físicos de la atmósfera como calor, evaporación, etc. La meteorología sinóptica estudia las fluctuaciones que tiene el ambiente diariamente en una región específica o globales, comparándolas con observaciones que se han hecho anteriormente a nivel mundial (Sadourny, 2006).

1.1.2.1 Hidrometeorología

Es la ciencia estrechamente ligada a la meteorología, la hidrología, y la climatología. Por lo tanto estudia el ciclo del agua en el planeta, también compone el estudio de las fases atmosféricas y terrestres del ciclo hidrológico y como estás fases se relacionan (Centro de Investigación

Aplicada en Hidrometeorología (CRAHI-UPC), s.f.).

1.1.3 Importancia de la meteorología

En todos los países del mundo, la importancia de la meteorología radica en el valor socio-económico, de los mismos, ya que según la cuantificación que se ha realizado en Estados Unidos, por cada dólar invertido en meteorología se gana cien dólares más. El estudio de la meteorología se utiliza para proteger la vida, y la propiedad de las personas frente a catástrofes naturales (Acuña Valverde y Robles Sánchez, 2015).

Un caso particular donde se evidencia la importancia de la meteorología es el transporte aéreo o marítimo, que gracias a las predicciones de las variables meteorológicas, puede planificar adecuadamente sus rutas para evitar o prepararse frente a condiciones adversas del clima (Sadourny, 2006).

De igual forma, el estudio del clima y su análisis tiene una gran influencia en la agricultura. Aunque la producción de los cultivos, depende también de la genética y el estado sanitario de la plantación, para evitar plagas indeseadas. La predicción del clima puede ayudar a que la producción de alimentos agrícolas no se destruyan ayudando a los agricultores, a establecer fechas de cultivo con mayor precisión, por está razón los elementos del clima determinan si un producto agrícola será rentable o no, de acuerdo a la estación y a la época del año. (Rodríguez y León, 2012).

Por otra parte, la meteorología también es de suma importancia para la exploración espacial, junto con la optimización de los modelos meteorológicos terrestres de área limitada, estos modelos son propicios para probarlos en la atmósfera de Marte. Además, sirve para la interpretación de datos de restricciones ambientales que contiene el planeta rojo y para medir los riesgos en la etapa de entrada, descenso, y aterrizaje a este planeta (Pla-Garcia y C.R.Rafkin, 2016).

En el caso del Ecuador, el INAMHI se encarga de medir las variables del clima e informar a la ciudadanía de sus posibles cambios. Además, esta institución intercambia información con otros país para mejorar los modelos de pronóstico del clima. (INAMHI, 2021).

1.1.4 Meteorología e internet

La red de internet es un elemento clave de la sociedad actual, que cada vez cuenta con herramientas para generar y compartir información. Desde esta perspectiva, la meteorología se ha convertido en una de las ciencias que más ha utilizado internet, especialmente, para el intercambio de datos entre los diferentes centros de medición y de elaboración de datos meteorológicos. De la misma forma, las estaciones meteorológicas han dejado disponible al público, información detallada de meteorología, que durante años permanecían en las manos de unas pocas personas expertas en este campo. Por está razón, las estadísticas que muestran los sitios web más visitados, muestran a los sitios de meteorología en los primeros lugares de popularidad. (Oldani, 2020).

1.1.5 La predicción del tiempo su importancia

La predicción del tiempo consiste en medir variables como la temperatura, la presión, la humedad, etc, para predecir el clima y prevenir o alertar a la población sobre situaciones críticas, desastres naturales, etc. Por lo tanto, un pronóstico del clima acertado puede minimizar las pérdidas y brindar mayor seguridad frente a cambios climáticos extremos

La predicción meteorológica se puede hacer de manera estadística, pero la forma más efectiva, es mediante la resolución de ecuaciones matemáticas, que se forman mediante la observación de las leyes físicas que tiene la atmósfera (Jiménez y cols., 2004). De esta forma, para realizar una predicción del tiempo de acuerdo a las técnicas de modelización numérica establecidas por los meteorólogos, se tienen que resolver ecuaciones diferenciales, que se definen mediante la

conservación de la masa, de la energía y del momento en la atmósfera. De está manera, con una observación en un tiempo (t), se obtiene el valor de las variables en un tiempo posterior ($t + \Delta t$), los pasos para realizar está predicción son: a) tomar los datos de las observaciones, b) ver la calidad de esas observaciones, c) observar la condiciones iniciales en una serie de puntos geográficos que conforman la rejilla del modelo, d) resolver las ecuaciones obtenidas de cada punto de la rejilla para un cierto tiempo, e) repetir la resolución de ecuaciones hasta llegar al tiempo deseado un día, horas, etc, f) representar gráficamente en forma de mapa de isolíneas (Jiménez y cols., 2004).

Los organismos e instituciones encargados de realizar las predicciones, realizan también los procesos de medición de las variables y la adquisición de los datos meteorológicos a lo largo del tiempo. Una buena medición asegura la calidad en los datos y esto a su vez, sirve para generar modelos de predicción más confiables.

1.2 Arquitectura basada en Microservicios

1.2.1 Definición de Microservicio

Para poder definir el concepto de microservicios, es importante determinar dónde y cómo surgió este término. En este sentido, según Martin Fowler el término microservice, microservicio en español, se originó en un taller de arquitectos de software, realizado en Venecia en 2011, donde se observó que el software desarrollado estaba compuesto por pequeños servicios independientes que podían comunicarse a través de mecanismos ligeros como una API bien definida(EMag, 2015).

La arquitectura de microservicios establece la construcción de un software como un conjunto de servicios, donde cada uno crece de manera independiente del otro y pueden ser escritos en lenguajes diferentes, que pueden estar en diversas partes del mundo, y realizar mantenimientos

con equipos de desarrollo distintos (Alfonso y Contreras, 2018).

1.2.2 Definición de Arquitectura de Software

Consiste en la organización fundamental, de un sistema, sus componentes, las relaciones entre ellos y el entorno, y los principios que orientan su diseño y evolución (Maier y cols., 2004).

1.2.3 Diferencias entre SOA y Microservicios

La arquitectura de microservicios está basada en *Service Oriented Architectures* (SOA). SOA consiste en construir aplicaciones enfocada a servicios, de una forma que se pueda integrar varias aplicaciones para un resultado en común, para realizar SOA se debe definir las interfaces de las aplicaciones que se van integrar , sea que estén realizadas en diferente hardware, diferente sistema operativo o lenguaje de programación, estás interfaces se conectan entre sí, para complementar su funcionamiento y llegar al resultado final (Bolo, s.f.).

La principal diferencia entre SOA y microservicios es el nivel de granularidad, es decir el principio de reutilizar unidades más pequeñas en el software. De esta forma, SOA utiliza una granularidad gruesa enfocada al negocio, con protocolos de comunicación más rigurosos, en cambio los microservicios utilizan una granularidad fina enfocada a la tarea o proceso particular, comunicados con mecanismos más livianos como protocolos HTTP, mediante APIs RESTFull (Alshuqayran, Ali, y Evans, s.f.). La infraestructura de microservicios se representa mediante una API. En la Figura 1 se muestra como una API o también conocida como api *layer* se conecta con diferentes módulos de una aplicación.

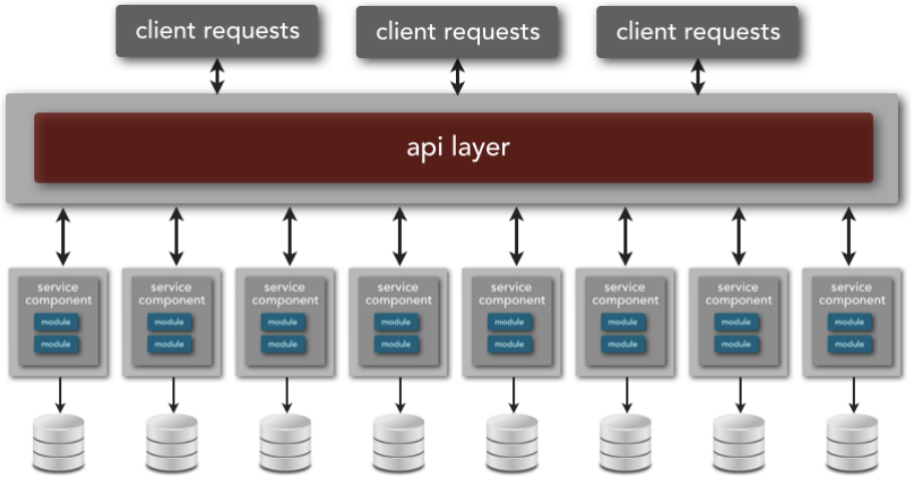

Figura 1: Infraestructura de una API
Fuente: (Richards, 2015)

1.2.4 Interfaz de programación de aplicaciones

Se puede entender como interfaz de programación de aplicaciones o en inglés *Application Programming Interface* (API) al conjunto de definiciones y protocolos que sirven para el desarrollo e integración del software en las aplicaciones. Las APIs permiten la comunicación e interacción entre varios módulos de software de forma más eficiente, evitando refactorizar la arquitectura del código inicial (Plaza Estévez, Ramírez Lamela, y Acosta Morales, 2016).

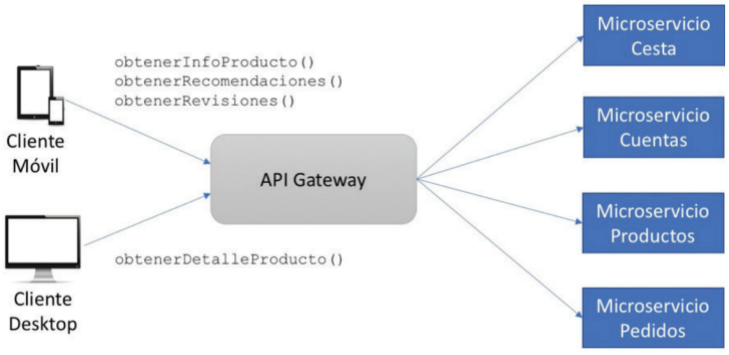

Figura 2: Esquema de una API Gateway

Fuente: (Martínez, 2018)

En la Figura 3 se observa el ecosistema de una API, el cual muestra mediante ejemplos de empresas, como las diferentes compañías se pueden conectar entre sí, compartiendo recursos. En este caso, la API de PayPal permite que cualquier comercio electrónico pueda realizar pagos en línea sin necesidad de programar su propio código. A su vez, en la Figura 2 se observa el esquema de una API Gateway.

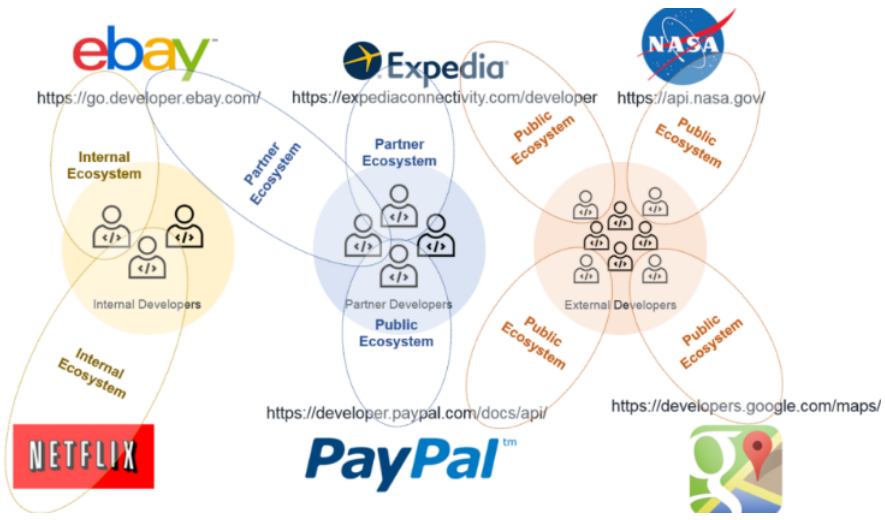

Figura 3: Ecosistema de una API

Fuente: (Weir, 2019)

1.2.5 RESTful

La arquitectura RESTful es un modelo de datos compartido que utiliza cuatro operaciones: GET (Consultar), PUT (Modificar), POST (Crear) y DELETE (Borrar) (IBM, 2020).

1.2.6 Modelo Vista Controlador

MVC es un patrón de diseño de software que separa las funcionalidades de una aplicación en tres capas, la primera se trata del modelo la cual contiene los datos y el estado del sistema, seguido de la segunda (vista), que es la encargada de interactuar con el usuario, por último la capa controlador que es intermediaria entre las dos primeras capas y donde se establece la lógica

de la aplicación. Utilizar este patrón de diseño disminuye el tiempo de trabajo en el desarrollo y mantenimiento de una aplicación (Albertos Gómez y Díaz Fernandez, 2018).

1.2.7 Postman

Postman es una plataforma que se utiliza para usar y crear APIs de forma más sencilla y para facilitar el trabajo ordenado en equipos de desarrollo (Postman Inc, 2022). En la Figura 4 se muestra como funciona la aplicación.

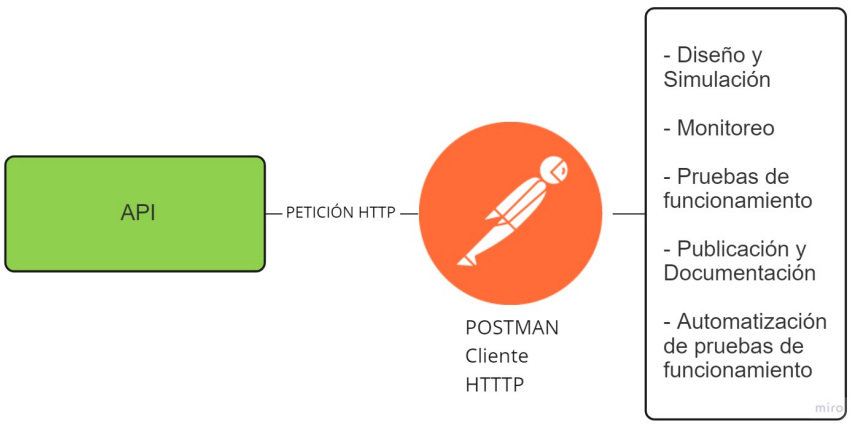

Figura 4: Funcionalidades de Postman

Elaborado por: Fabián Garrido

1.3 Base de datos relacionales

La bases de datos relacionales (DB) son herramientas que sirven para gestionar la información. Las DB guardan la información en tablas relacionadas y su principal ventaja consiste en que se puede acceder a los datos de forma rápida y eficiente a través de una consulta (Sánchez, 2016).

Existen diferentes tipos de gestores de bases de datos relacionales algunos de ellos son: PostgreSQL, MySQL, MariaDB, Oracle, etc.

Las bases de datos relacionales se diferencian de las no relacionales por su alta integridad transaccional, es decir, tienen una estructura bien definida, donde se establecen las propiedades de sus datos por adelantado, al igual que las relaciones existentes entre sus tablas. (Cordova y Cuzco, 2013).

1.3.1 Lenguaje de Consulta Estructurado

El Lenguaje de Consulta Estructurado (SQL) se utiliza para gestionar y consultar los datos, cumple con un estándar internacional reconocido por organismos como la ISO y ANSI (Microsoft, s.f.).

1.3.2 Sistema Gestor de Base de Datos PostgreSQL

Un Sistema Gestor de Base de Datos (SGBD) trata de un conjunto de programas que trabajan de forma transparente para el usuario y sirven para gestionar los datos de una DB. La función principal de un SGBD es brindar a las personas la facilidad de recuperar y almacenar su información (Silberschatz, Korth, y Sudarshan, S. (Instituto Indio de Tecnología, 2002). En la Figura 11 se aprecia el funcionamiento de un SGBD.

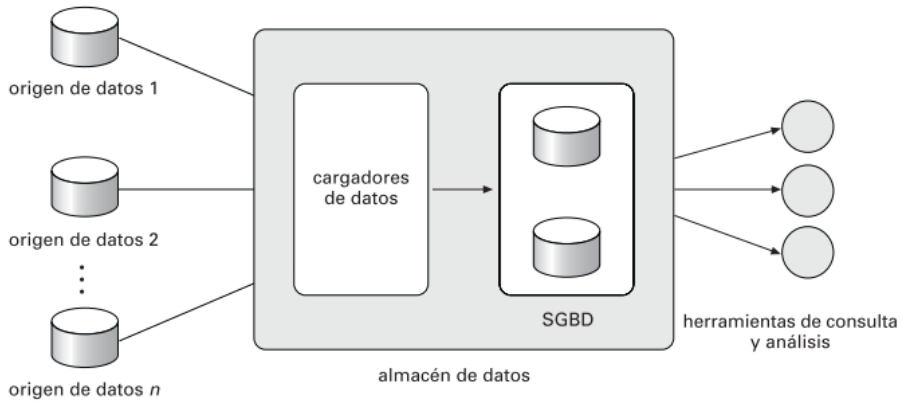

Figura 5: Arquitectura de los almacenes de datos

Fuente: (Silberschatz y cols., 2002)

PostgreSQL es un SGBD catalogado como robusto, potente y estable, debido a su infraestructura que utiliza hilos en paralelo para el procesamiento de consultas en la base de datos, lo que además brinda mayor rapidez al realizar cualquier consulta (Zea Ordóñez, Molina Ríos, y Redrován Castillo, 2017).

Adicionalmente, PostgreSQL es un sistema de código abierto, que actualmente está en el *ranking* mundial, entre los mejores puntuados, ocupando el cuarto lugar (DB-Engines, s.f.).

1.4 *Framework* de desarrollo Django

Django es un marco de trabajo web basado en Python que facilita el desarrollo de una aplicación, usando las librerías y funcionalidades de este lenguaje de programación.

La arquitectura de Django sigue el patrón de diseño Modelo Vista Controlador (MVC). De modo que cada módulo de Django es independiente y puede ser modificado sin afectar a los demás. Por lo tanto, un diseñador HTML podrá modificar el diseño de la página sin afectar el código en el *backend* de una aplicación. Así mismo, un administrador de base datos podrá hacer cambios a la DB sin tener que alterar otros archivos (Foundation, 2012).

Por otra parte, existen otros *frameworks* de desarrollo web para python como Flask, Pyramid, Bottle, CherryPy, Tornado, etc. Sin embargo, los más populares son Flask y Django. Aunque ambos funcionan de forma similar, Django tiene una estructura más rígida, lo que facilita su mantenimiento (Ghimire, 2020). Además, los desarrolladores suelen escoger a Django principalmente para evitar errores de seguridad y para que la aplicación sea más escalable.

1.5 Herramienta de gestión de proyectos Kanban

Kanban es un sistema de señales visuales que sirve para controlar la producción y mantener activo el abastecimiento de tareas en un proyecto. Puede enviar diferentes tipos de tareas acorde

a las prioridades del proyecto. Además, permite establecer equipos de trabajo, los cuales podrán priorizar las tareas y eliminarlas si no tienen valor, reduciendo costes de producción en cualquier proyecto (Castellano Lendínez, 2019). En la Figura 6 se puede apreciar un ejemplo del tablero Kanban.

Figura 6: Tablero Kanban

Elaborado por: Fabián Garrido

Capítulo 2

Marco metodológico

En está sección, se presenta los requerimientos del proyecto, la arquitectura de la aplicación y el proceso de construcción de la API mediante Python con Django.

2.1 Análisis de requerimientos

2.1.1 Alcance

Este aplicativo tiene como objetivo proporcionar, una versión *backend*, de la libreta hidrometeorológica del INAMHI, con el lenguaje de programación Python.

Concretamente, el aplicativo permitirá la conexión de la LH con futuras aplicaciones *Frontend*, y tendrá la facilidad de trabajar con funciones *Create, Read, Update and Delete* (CRUD).

2.1.2 Requerimientos específicos

El desarrollo de este proyecto cuenta con cuatro requerimientos específicos, que se determinaron bajo la tutela de los interesados.

- R1: Crear el proyecto usando el lenguaje de programación en Python y el *framework* de desarrollo Django.
- R2: Crear una base de datos usando PostgreSQL y conectarla con la API.
- R3: Migrar los datos a un nuevo formato JSON (*JavaScript Object Notation*) que facilite la consulta, carga y lectura de los datos.

- R4: Construir los métodos GET, POST, PUT y DELETE en gestionar la base de datos a través de la API.

Los criterios de aceptación de estos requerimientos se detallan en la tabla 1.

Tabla 1: Criterio de aceptación de los requerimientos específicos

Requerimientos	Criterios de aceptación
R1	Uso del modelo Vista Controlador (MVC) para la construcción de la arquitectura de la aplicación.
	Utilización de un marco de trabajo web (Django)
	Utilización de lenguaje de programación Python, versión 3 en el desarrollo del proyecto.
R2	Uso del esquema denominado ”convencionales” de la base de datos.
	Utilización de la versión 13 de PostgreSQL.
	Manejo privado de los datos de la base de datos de la LH.
R3	Estructura del formato JSON adecuada y con los atributos y secciones acorde a la información que se va a manejar.
R4	Peticiones construidas sobres clases de Python.
	Uso del lenguaje SQL.
	Peticiones validadas y que acepten el ingreso de parámetros .

2.1.3 Requerimientos funcionales

Debido a que este trabajo incluye únicamente el desarrollo del *backend* de la aplicación, los usuarios que van a utilizar la API serán aplicaciones *frontend*. En este sentido se detallan los requerimientos funcionales en las Tablas 2 y 3, respectivamente.

Tabla 2: Conexión con la base de datos

Descripción	Consulta y visualización de los datos del INAMHI
Datos de Entrada	Datos de búsqueda (nombre de la tabla, id, fecha)
Proceso	Obtención de datos requeridos
Salida	Mensaje de aceptación / Error

Tabla 3: Gestión de peticiones

Descripción	Consulta, modificación y eliminación de los datos
Datos de Entrada	Datos de búsqueda (nombre de la tabla, id, fecha) y Estructura del formato JSON
Proceso	Modificación de la base de datos
Salida	Mensaje de aceptación / Error

2.1.4 Requerimientos no funcionales

Respecto a los requerimientos no funcionales se solicita que la aplicación sea altamente mantenible y que el código sea fácilmente entendible. De igual manera, se requiere que la API sea altamente escalable y que pueda funcionar con cualquier esquema de la base de datos.

2.2 Arquitectura y diseño de la aplicación

Con el objetivo de facilitar el ingreso de datos del INAMHI, se realiza una aplicación *backend*, que realiza las funciones GET, POST, PUT y DELETE para manipular los datos del esquema denominado ”convencionales” de la DB. La realización de estas funciones se hizo junto con la herramienta Kanban.

En la Figura 7 se muestra la arquitectura diseñada de la API, la cual consiste en dos componentes, por un lado, la estructura JSON que sirve para transformar los atributos de la DB a un JSON y por otro lado, el componente de peticiones que realiza un CRUD con los datos de las tablas.

En el componente de peticiones se construyen los *endpoints*, es decir, se crean las direcciones url del proyecto, en los cuales se establecen los parámetros a ingresar y el contenido mediante el *body* que lleva los datos de la petición (como un contenido de un formulario de una página HTML). El body es el cuerpo de una página html, este contenido se lo ingresará de manera

manual y para está acción se utiliza Postman.

El diseño de la aplicación continua con la gestión de los datos, los cuales son manipulados por el *body* de Postman, consecuentemente la información se muestra de forma actualizada en la DB.

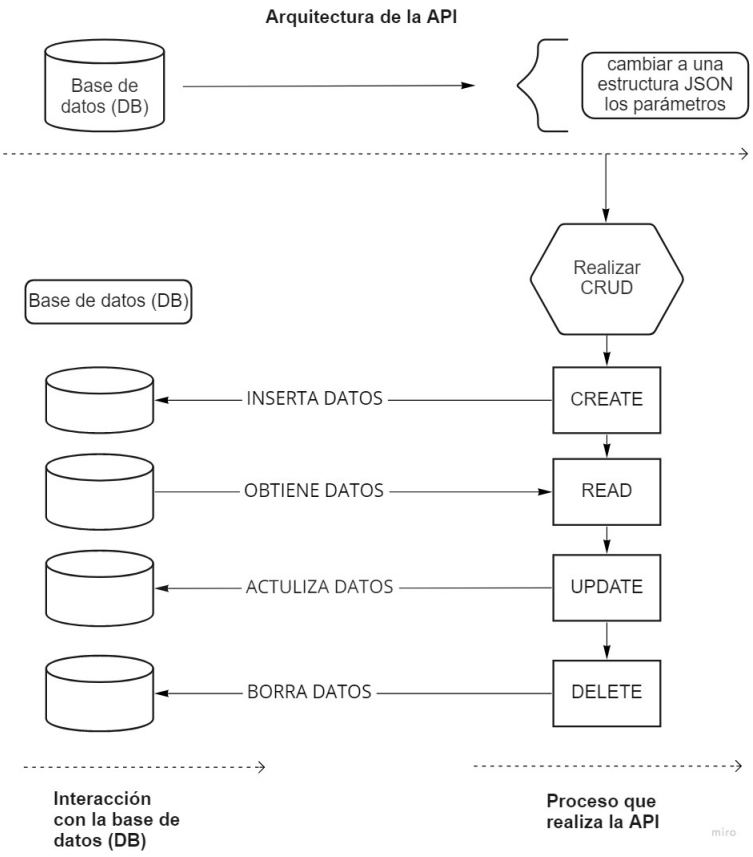

Figura 7: Arquitectura de la API

Elaborado por: Fabián Garrido

2.3 Desarrollo de la API

Es importante mencionar que el INAMHI utiliza una sintaxis numérica para establecer los nombres de las tablas de la base de datos, estás tablas pertenecen a diferentes categorías meteorológicas. Sin embargo, el presente proyecto trabajará solo con el esquema de tablas denominado ”convencionales” de la base de datos.

En la Figura 8 se muestra los pasos para realizar un *endpoint*. Se comienza estableciendo la dirección que se va usar en el proyecto tal que, está dirección sea la dirección Padre para todos los *endpoints* que se puedan subdividir. Luego, se debe establecer una clase de Pyhton, en este caso se ha establecido la case denominada ”InamhiView”, que contiene los métodos GET, POST, PUT o DELETE que son lo que realizan la gestión en la DB a través del protocolo http.

Figura 8: Pasos para crear un *endpoint*

Elaborado por: Fabián Garrido

Después de establecer la clase de Python, se inserta en la dirección Padre los parámetros que se deben ingresar en el endpoint, los cuales son el nombre de la tabla y la fecha, ambos de tipo string. En el último parámetro también se permite ingresar el id de la tabla que se desea consultar o modificar.

Posteriormente, se debe establecer que peticiones http que se van a realizar. De acuerdo a la solicitud del INAMHI en la Tabla 4 se muestra la descripción y funcionalidades de cada petición.

Tabla 4: Peticiones HTTP con sus respectivas funciones

GET	Imprime todos los datos de las tablas. Imprime los datos de una tabla en específico. Imprime los datos de una fecha específica junto con cualquier tabla del esquema denominado ”convencionales” de la DB.
POST	Ingresa todos los datos del conjunto de tablas mediante el body de Postman.
PUT	Actualiza los mismos datos que se ingresan en la petición POST.
DELETE	Borra los datos de una fila de la tabla de acuerdo a un ID.

Seguidamente se definen la sintaxis SQL y la creación de los métodos CRUD: *INSERT* (*Create*), *SELECT* (*Read*), *UPDATE*, *DELETE*, para poder manipular y almacenar los registros de cada tabla de la DB, la sintaxis que se utiliza en todas la tablas se muestra en la Figura 9, por consiguiente el método *SELECT* se define como se muestra en la Figura 9(A), en cual se observa la estructura de la consulta en formato JSON *array*, para posteriormente unir las consultas de todas las tablas en varios JSON array e imprimir en el *endpoint* correspondiente.

Luego, se describe la consulta que se debe ingresar para realizar una búsqueda de una fecha específica. Esto permite imprimir solo los registros que se encuentren en esa fecha (Figura 9(B)). La Figura 9(C) corresponde a la consulta para actualizar los registros de cada tabla. La Figura 9(D) menciona como se debe insertar los datos en la tablas. En la Figura 9(E) se muestra como se debe eliminar datos de una tabla.

Por último, se establece la recolección de los datos JSON en la clase denominada ”Inamhi-View” junto con su respectivas validaciones en las peticiones HTTP. Para llevar a acabo la acción de recolectar los datos JSON del *body* de Postman se necesita llamar al módulo import json de Python, como se muestra en la Figura 10(A). Después se accede a todas la variables del *body* de Postman Figura 10(B) y se inserta esa variable entre comillas simples, para guardar el valor en una variable de Python.

Consulta por JSON (A)

```
= "SELECT array_to_json(array_agg(p.*)) FROM convencionales._1073161h p"
```

Consulta por JSON con fecha (B)

```
= "SELECT array_to_json(array_agg(p.*)) FROM "
+="convencionales._1073161h p where fecha_ingreso="+"'"+search+"'"
```

Actualizar tabla (C)

```
 = """update convencionales._1073161h """
+= """set id_estacion=%s,  id_usuario=%s,  fecha_toma=%s, """
+= """ fecha_ingreso=%s,  valor=%s,  validado=%s where"""
+= """id_temp_int_baro = %s"""
```

Insertar Datos (D)

```
="""INSERT INTO convencionales._1073161h"""
+="""(id_estacion,id_usuario, fecha_toma, fecha_ingreso"""
+=""", valor,validado)VALUES (%s,%s,%s,%s,%s,%s)"""
```

Borrar Datos (E)

```
query = """delete from convencionales._1073161h """
query+= """where id_temp_int_baro = %s"""
```

Figura 9: Sintaxis CRUD

Elaborado por: Fabián Garrido

Módulo JSON (A)

```
#obtener valores del Json
jd=json.loads(request.body)
```

Del body del navegador a una variable (B)

```
id_estacion=jd['id_estacion_1073161h']
```

Figura 10: Recolección de datos JSON

Elaborado por: Fabián Garrido

La aplicación *backend* tendrá la capacidad de gestionar los datos del INAMHI acorde a las peticiones anteriormente descritas, las cuales se describen en el diagrama de caso de uso de la Figura 11.

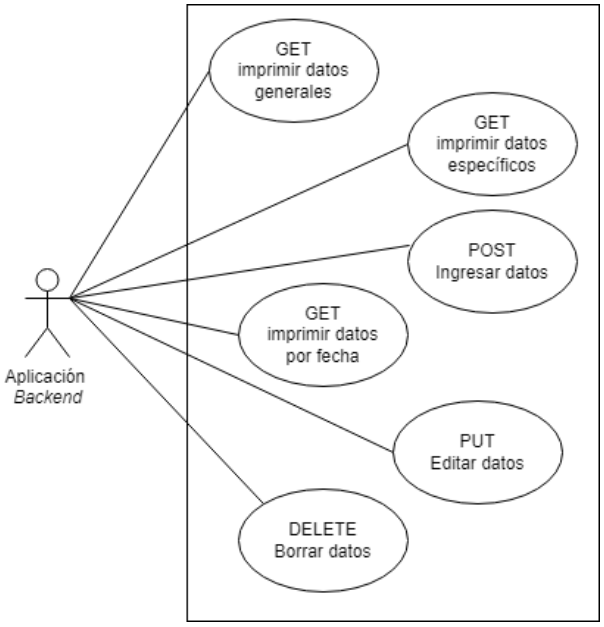

Figura 11: Peticiones de la API

Elaborado por: Fabián Garrido

El INAMHI tendrá las funcionalidades que muestra el diagrama de caso de uso en la Figura 12, estás funcionalidades se podrán aplicar a futuras aplicaciones *frontend* que se realice. Estás aplicaciones *frontend* podrán estar en diferentes sistemas operativos o dispositivos, en la Figura 13 se observa el funcionamiento que podrán tener estos dispositivos con la API junto con la base de datos del INAMHI.

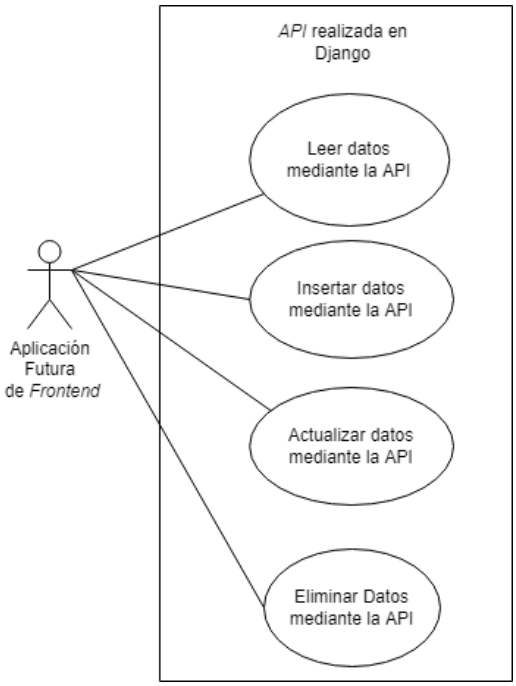

Figura 12: Aplicación Futura del INAMHI

Elaborado por: Fabián Garrido

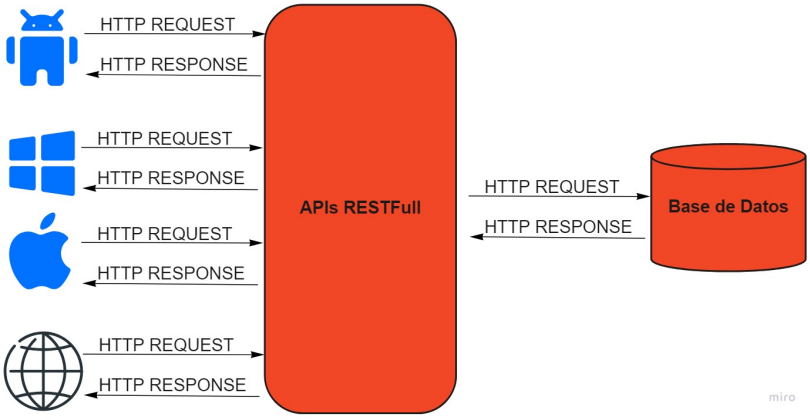

Figura 13: Versatilidad de la API del INAMHI

Elaborado por: Fabián Garrido

Capítulo 3

Pruebas y resultados

Para evaluar el funcionamiento de la API se llevaron a cabo pruebas funcionales con la herramienta Postman, que permite testear los HTTP *requests*. A su vez se inspeccionó el código con SonarCloud y finalmente se documentó la API con Postman.

Las pruebas se desarrollaron de forma local, en un computador con las características mostradas en la Tabla 5 y la Tabla 6.

Modelo	Procesador	Memoria RAM
Aspire F5-573G	Intel Core i5-7200U	12288 MB

Tabla 5: Características de la computadora

Sistema Operativo	Python	Django	Base de Datos	Postman
Windows 10	3.9.7	3.2.4	PostgreSQL 13	8.9.1

Tabla 6: Características del Software

3.1 Pruebas funcionales

Se realizó las pruebas funcionales de las peticiones GET, seguido de las peticiones POST, PUT y DELETE.

3.1.1 Peticiones GET

Para la petición GET se estableció la dirección del *endpoint* para imprimir todos los datos de las tablas del esquema denominado ”convencionales” de la DB, como muestra en la Figura 14 el *status* del *endpoint* fue un éxito, ya que se pudo mostrar todos los datos de las tablas en formato JSON, separando cada tabla con un corchete (Figura 14).

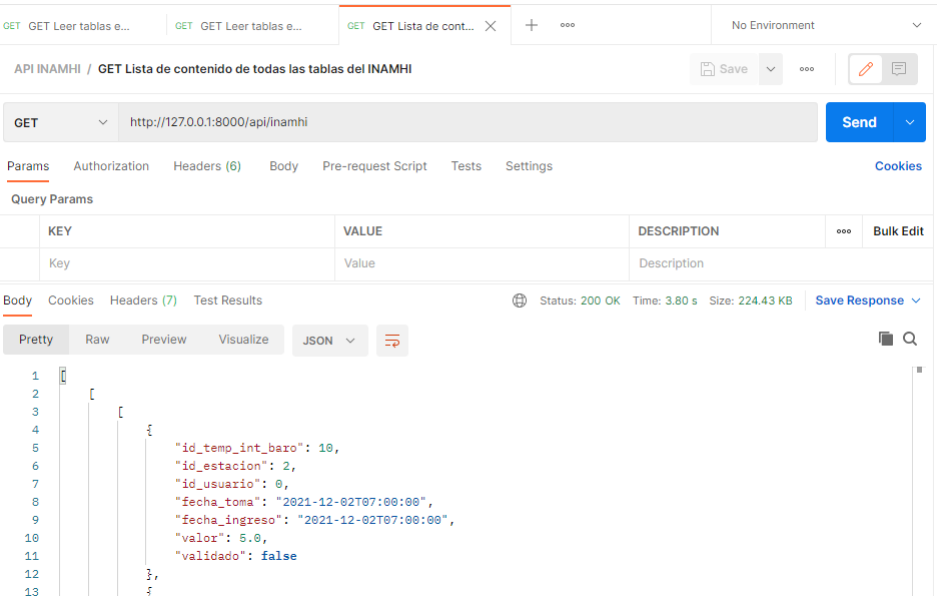

Figura 14: *Endpoint* para todas la tablas

Elaborado por: Fabián Garrido

Por otra parte, la segunda petición GET muestra los datos de una tabla en específico, enviando como parámetro el nombre de una tabla. En el ejemplo de la Figura 15 se prueba con la tabla _734161h, logrando leer los datos y de esta forma se puede leer cualquier tabla del esquema denominado ”convencionales” de la DB.

La última petición GET del proyecto consiste en imprimir los datos de una tabla en específico, pero en este *endpoint* también se hace un filtro con una fecha particular, por ende el resultado tendrá dos filtros, por nombre de la tabla y por una fecha en particular. La forma que se prueba este *endpoint* se muestra en la Figura 16, en el cual se establece el nombre de la tabla _1073161h

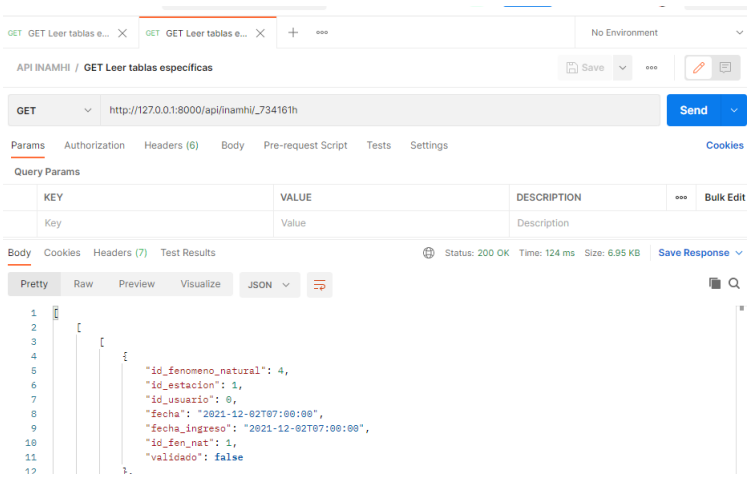

Figura 15: *Endpoint* para una tabla en específico

Elaborado por: Fabián Garrido

como primer parámetro, y la fecha 2021-12-02T07:00:00 como segundo parámetro. Para unir la fecha y la hora es necesario usar el caracter ”T”, en vez del espacio en blanco y este formato debe coincidir tanto en el *endpoint* como en la DB.

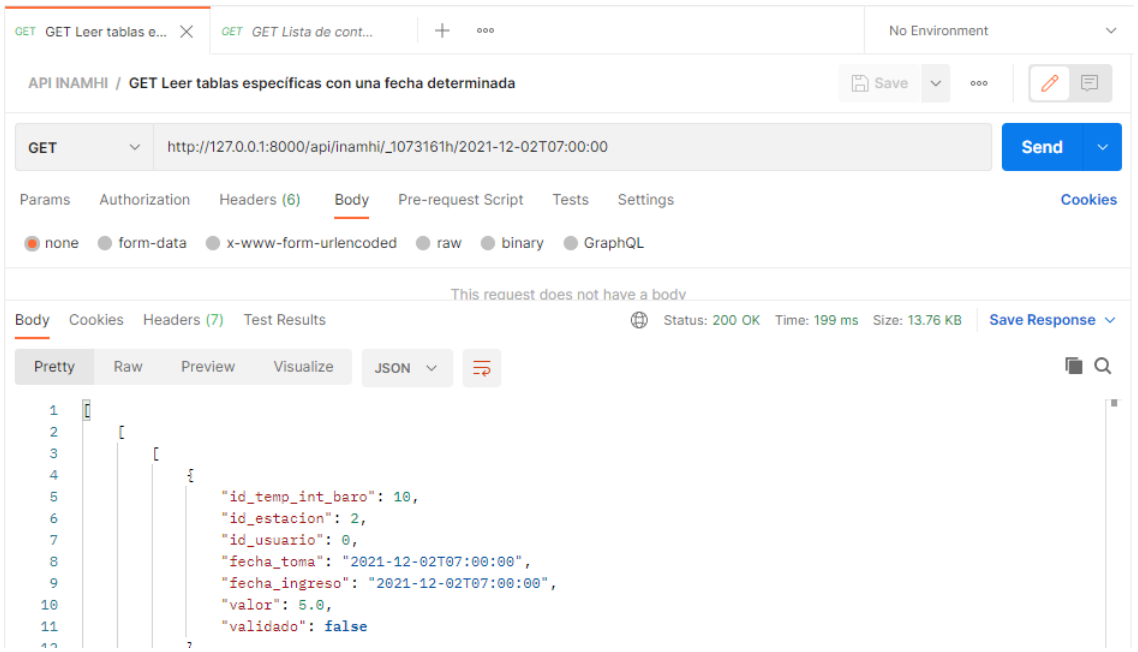

Figura 16: *Endpoint* para una tabla en específico y una fecha determinada

Elaborado por: Fabián Garrido

3.1.2 Petición POST

Para comprobar el *endpoint* POST se ingresa en formato JSON datos de prueba correspondiente a las 35 tablas de la libreta hidrometeorológica, mediante el *body* de Postman. Para las pruebas de funcionamiento, el formato que se ingreso en todas las tablas es como se muestra en la Figura 17.

Figura 17: *Endpoint* para ingresar los datos en todas las tablas

Elaborado por: Fabián Garrido

3.1.3 Petición PUT

Para la prueba de funcionamiento de la petición PUT se ingresa en el *endpoint* dos parámetros, el primero es el nombre de la tabla, en este caso en la Figura 18 se prueba con la tabla _734161h, y el segundo parámetro es el id, es decir la fila de la tabla, en la Figura 18 se prueba con la fila 1. Finalmente se debe establecer en el *body* de Postman, los atributos de la tabla junto con los nuevos valores que se requiera actualizar, como muestra la Figura 18.

Figura 18: *Endpoint* para editar datos

Elaborado por: Fabián Garrido

3.1.4 Petición DELETE

En la prueba de funcionamiento de la petición DELETE se ingresa en el *endpoint* dos parámetros, el primero es el nombre de la tabla, en este caso en la Figura 19 se prueba con la tabla _734161h, y el segundo parámetro es el id, en la Figura 19 se prueba con la fila 2. De está manera se ha eliminado la fila dos de la tabla _734161h de la DB.

3.2 Pruebas de Código con SonarCloud

En la Tabla 7 se resume del análisis de las 4162 líneas de código realizado por SonarCloud. De acuerdo a los resultados no se encontró ningún error de funcionalidad ni de vulnerabilidad. Sin embargo, hubieron 162 líneas de código apestoso, que aunque que se pueden refactorizar, estos cambios no son imprescindibles para el funcionamiento de la aplicación, como por ejemplo las impresiones de consola, cierre de conexión de base de datos y la repetición del nombre de variables (Figura 20).

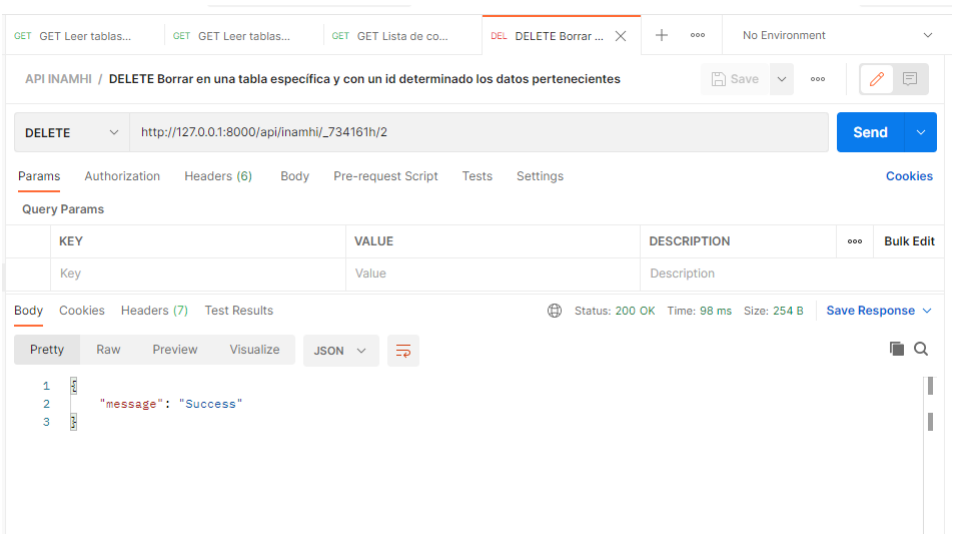

Figura 19: *Endpoint* para eliminar

Elaborado por: Fabián Garrido

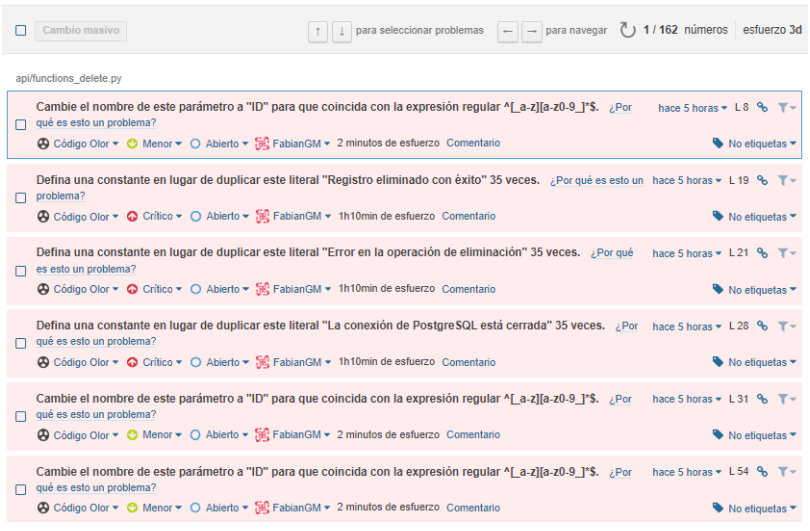

Figura 20: Error de variable e impresión de consola

Elaborado por: Fabián Garrido

Carpetas del Proyecto	Líneas de Código	Errores	Vulnerabilidades	Código Apestoso	Security Hotspots	Duplications
api	4,162	0	0	162	1	0.0%
Proyecto_API	88	0	0	0	2	0.0%
manage.py	15	0	0	0	0	0.0%

Tabla 7: Análisis del código

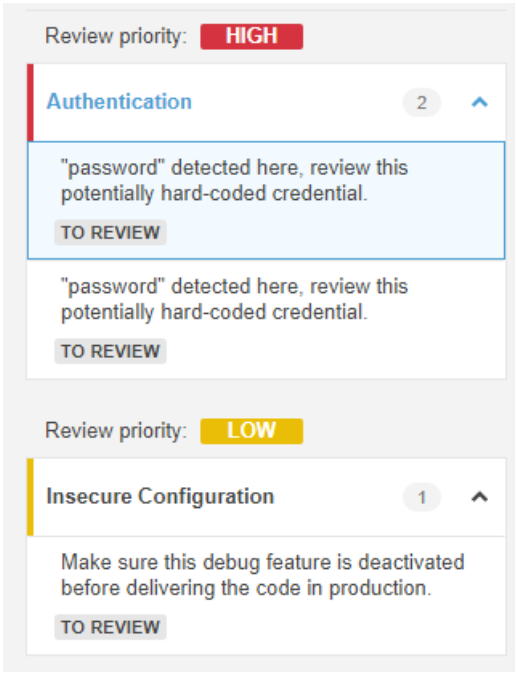

Figura 21: *Security Hotspots*

Elaborado por: Fabián Garrido

También en la columna de Security Hotspots de la Tabla 7, se muestra ciertos puntos de seguridad que se debe revisar en las funciones de conexión con PostgreSQL, como muestra la Figura 21 se detectó un archivo de Pyhton con los datos de acceso a la DB que podrían corregirse en un trabajo futuro usando el estándar JSON Web Token.

3.3 Documentación de la API

La documentación de la API se hizo con Postman y la publicación del código se realizó en GitHub. Por motivos de privacidad solo se muestra la estructura del informe entregado al INHAMI a través de la Figura 22.

De igual forma, el proyecto en GitHub también fue creado de forma privada, por esta razón el código del desarrollo no es publico, ni se puede reutilizar, pero se puede mostrar las estructura de los ficheros en Python del proyecto montado en Django (Figura 23).

3.3.1 Descripción de la API con Postman

La Figura 22(A) muestra las peticiones solicitadas por el INAMHI en el *backend*. A su vez, estás peticiones podrán ser importadas de manera local mediante Postman, junto con el enlace generado La (Figura 22(B)) muestra una descripción de la API y las funciones de sus peticiones, igualmente la Figura 22(C) muestra como está documentada cada petición. Finalmente, la Figura 22(D) muestra una descripción en formato HTML de los *endpoints* realizados.

3.3.2 Respaldo de la API con GitHub

En la Figura 23 se puede observar la publicación en GitHub y también la entrega de un respaldo PDF del informe de la API, junto con un documento txt con los paquetes utilizados en el proyecto, los cuales están presentes en la Tabla 8.

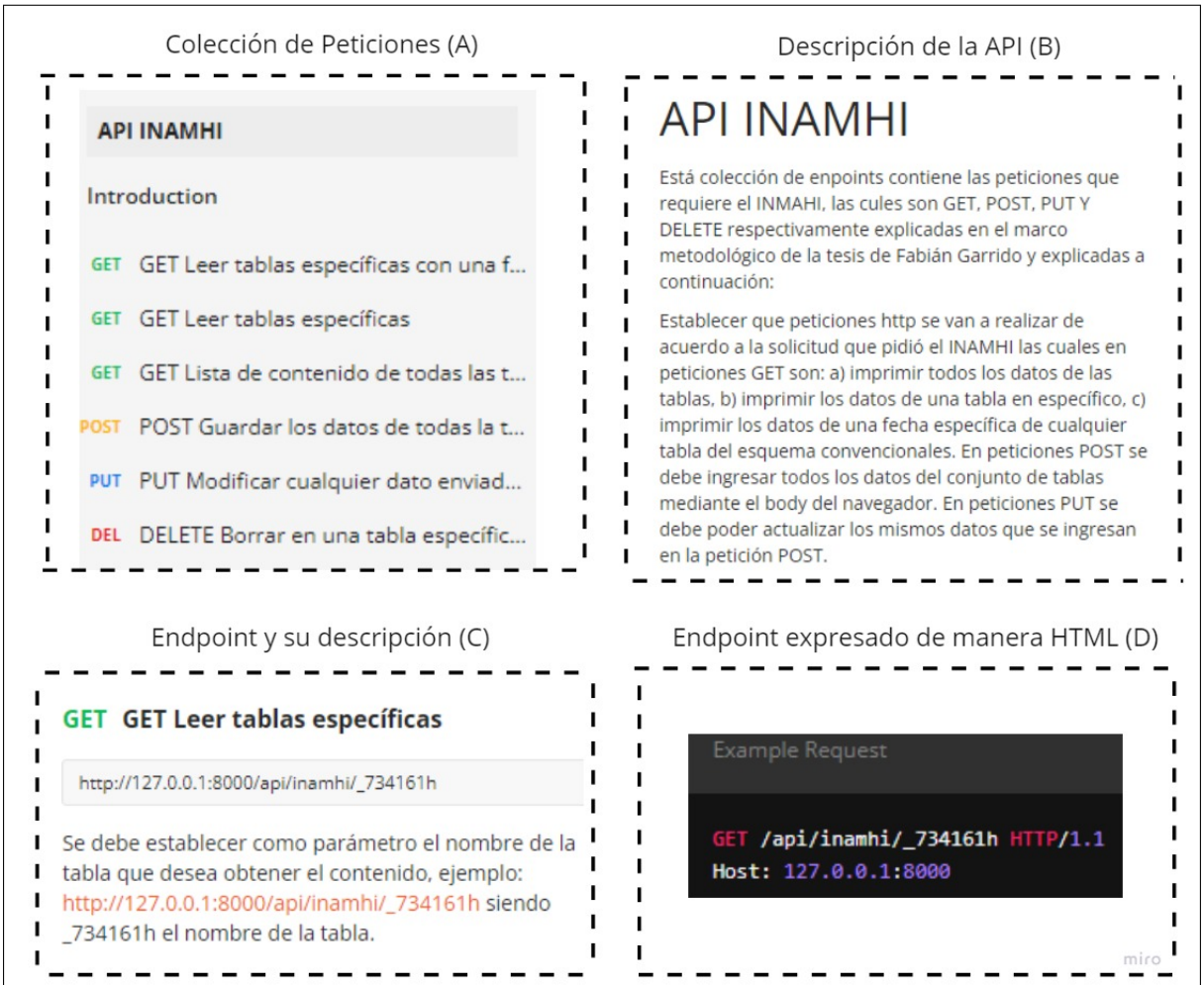

Figura 22: Estructura de la Documentación

Elaborado por: Fabián Garrido

Figura 23: Código en GitHub

Elaborado por: Fabián Garrido

Paquetes	Versión
asgiref	3.4.1
Django	3.2.4
pip	21.3.1
psycopg2	2.9.2
pytz	2021.3
setuptools	59.0.1
wheel	0.37.0

Tabla 8: Paquetes Utilizados

3.4 Análisis de Resultados

Cada prueba funcional de las peticiones mostró sus resultados de manera correcta. Sin embargo, el análisis del código muestra código denominado ”apestoso”, el cual no es relevante para el funcionamiento de la aplicación y tampoco se trata de advertencias de seguridad sobre la conexión con la base de datos. Si, en futuros desarrollos, se debe incluir nuevas tablas al proyecto es recomendable seguir la distribución de los módulos de Django, como muestra la Figura 24, la cual muestra el patrón de 3 capas (MVC). El módulo connection.py representa la capa controlador, todos los módulos nombrados functions.py junto con el nombre que se requiera dar conforman la capa modelo y finalmente, el módulo views.py representa a la capa vista.

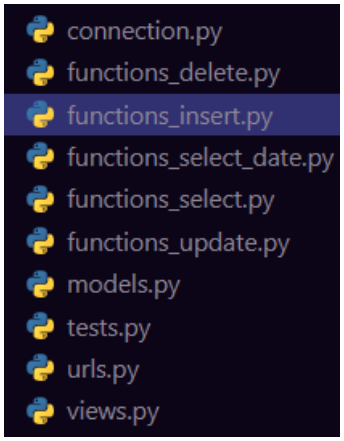

Figura 24: MVC Proyecto INAMHI

Elaborado por: Fabián Garrido

Conclusiones

En este trabajo se ha realizado con éxito la migración de la libreta hidrometeorológica del INAMHI a una arquitectura de microservicios y se han obtenido las siguientes conclusiones:

- Fue posible construir el *backend* de la aplicación a través de herramientas de software libre como Django y python en el entorno de desarrollo y como lenguaje de programación, respectivamente. Logrando establecer con éxito los *endpoints* requeridos por el INAMHI.
- Al analizar los requerimientos fue posible definir los criterios de aceptación para dar cumplimiento a los objetivos del proyecto.
- Se utilizó la planificación de las tareas con la herramienta KANBAN, la cual facilitó la gestión de tareas, en especial, la parte de construcción de la API.
- Al realizar la migración de la LH de una arquitectura monolítica a microservicios fue necesario establecer un nuevo formato de intercambio de datos JSON, lo que le da mayor versatilidad a la aplicación.
- La aplicación pasó satisfactoriamente la evaluación funcional y de código realizada con los protocolos HTTP. Sin embargo, existen fragmentos de código, que de ser necesario se podrían refactorizar.

Recomendaciones

Del desarrollo de este proyecto se desprenden las siguientes recomendaciones:

- Las pruebas de la aplicación se llevaron a cabo en un servidor local, por lo que se recomienda hacer prueba en un servidor en línea para constatar el correcto funcionamiento de la API.
- Es necesario mantener las sentencias SQL separadas en módulos para facilitar su lectura.
- Como trabajo futuro se recomienda implementar un sistema de colas, y de está manera utilizar la aplicación de forma paralela, en varias aplicaciones distintas.

Referencias

Acuña Valverde, D., y Robles Sánchez, D. (2015). Manual de meteorología y gestion de la información climática. *Instituto de Montaña*, 1–168.

Albertos Gómez, E., y Díaz Fernandez, J. (2018). *Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura* (Tesis Doctoral no publicada).

Alfonso, D., y Contreras, B. (2018). Arquitectura de Microservicios. *Tecnología Investigación y Academia*, *6*(1), 36–46.

Alshuqayran, N., Ali, N., y Evans, R. (s.f.). A Systematic Mapping Study in Microservice Architecture. , 8. doi: 10.1109/SOCA.2016.15

Bolo, M. (s.f.). ARQUITECTURA DE INTEGRACIÓN ORIENTADA A SERVICIOS. , 19–46.

Castellano Lendínez, L. (2019). KANBAN. METODOLOGÍA PARA AUMENTAR LA EFICIENCIA DE LOS PROCESOS KANBAN. METHODOLOGY TO INCREASE PROCESS EFFICIENCY. *3C Tecnología*$_{G}losasdeinnovacinaplicadasalapyme, 8(1), 30 - -41.doi$:

Centro de Investigación Aplicada en Hidrometeorología (CRAHI-UPC). (s.f.). *Qué es la hidrometeorología?* Descargado 2021-09-17, de `http://www.crahi.upc.edu/index.php?option=com_content&view=article&id=75%3Aque-es-la-hidrometeorologia-&catid=36&Itemid=85&lang=eshttp://www.crahi.upc.edu/index.php?option=com_content&view=article&id=75:que-es-la-hidrometeorologia-&catid=36&Itemid=85&lan`

Cordova, R., y Cuzco, B. (2013). *Análisis comparativo entre bases de datos relacionales con bases de datos no relacionales* (Tesis Doctoral). Descargado de `http://dspace.ups.edu.ec/bitstream/123456789/6977/1/UPS-CT003639.pdf`

DB-Engines. (s.f.). *DB-Engines Ranking.* Descargado 2022-02-09, de `https://db-engines`

.com/en/rankinghttps://db-engines.com/en/ranking%0Ahttp://db-engines.com/en/ranking_trend

EMag. (2015). Architectures you Always Wondered About. *Infoq Microservices*(31).

Foundation, D. S. (2012). *Django, La guia Definitiva*.

Ghimire, D. (2020). *Comparative study on Python web frameworks: Flask and Django* (Tesis Doctoral no publicada).

IBM. (2020). *Concepts of RESTful JSON web services - IBM Documentation.* Descargado 2021-09-17, de https://www.ibm.com/docs/en/cics-ts/5.2?topic=services-concepts-restful-json-web

INAMHI. (2021). *Valores / Misión / Visión – Instituto Nacional de Meteorología e Hidrología.* Descargado 2022-02-15, de https://www.inamhi.gob.ec/valores-mision-vision/

Jiménez, R. M. R., Capa, Á. B., y Lozano, A. P. (2004). *Meteorología Y Climatología*. Descargado de https://cab.inta-csic.es/uploads/culturacientifica/adjuntos/20130121115236.pdf

López Hinojosa, J. D. (2017). *ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB DE LA ASAMBLEA NACIONAL* (Tesis Doctoral no publicada).

Maier, M. W., Emery, D., y Hilliard, R. (2004, jan). ANSI/IEEE 1471 and systems engineering. *Systems Engineering*, *7*(3), 257–270. Descargado de https://onlinelibrary.wiley.com/doi/full/10.1002/sys.20008https://onlinelibrary.wiley.com/doi/abs/10.1002/sys.20008https://onlinelibrary.wiley.com/doi/10.1002/sys.20008 doi: 10.1002/sys.20008

Martínez, D. R. (2018). *Microservicios Un enfoque integrado*. Descargado de https://books

.google.com.ec/books?id=2o6fDwAAQBAJ&printsec=frontcover&dq=TIPO+DE+MICROSERVICIOS&hl=
es&sa=X&redir_esc=y#v=onepage&q=TIPODEMICROSERVICIOS&f=false

Mas'ad Nasra, R. (2016). *Migración de un Sistema Monolitico a una Arquitectura de Microservicios* (Tesis Doctoral). Descargado de https://repositorio.usm.cl/handle/11673/19938

Microsoft. (s.f.). *Access SQL: conceptos básicos, vocabulario y sintaxis.* Descargado 2022-01-25, de https://support.microsoft.com/es-es/office/access-sql-conceptos-b{á}sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671#bm1https://support.microsoft.com/es-es/office/access-sql-conceptos-b{á}sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-

O'Connor, R. V., Elger, P., y Clarke, P. M. (2016). Exploring the impact of situational context - A case study of a software development process for a microservices architecture. *Proceedings - International Conference on Software and System Process, ICSSP 2016*, 6–10. doi: 10.1145/2904354.2904368

Oldani, J. (2020). *La meteorología*. Editorial De Vecchi, S.A. Descargado de https://books.google.com.ec/books?id=21HhDwAAQBAJ&pg=PT97&dq=meteorolog{í}a&hl=es&sa=X&ved=2ahUKEwi70NavtrL0AhXbSDABHSeCAkIQ6AF6BAgHEAI#v=onepage&q=meteorolog{í}a&f=false

Paul Durzan. (2020). *El crecimiento de los microservicios y por qué es importante para su negocio.* Descargado 2021-09-16, de https://www.datacenterdynamics.com/es/opinion/el-crecimiento-de-los-microservicios-y-por-qu{é}-es-importante-para-su-negocio/

Pla-Garcia, J., y C.R.Rafkin, S. (2016). Meteorología mesoescalar en Marte. *Física de la Tierra*,

28(0), 129–161. doi: 10.5209/rev_fite.2016.v28.53901

Plaza Estévez, S., Ramírez Lamela, N., y Acosta Morales, C. (2016). *API de servicios web orientados a accesibilidad* (Tesis Doctoral). Descargado de https://eprints.ucm.es/38686%0Ahttps://goo.gl/E3A6BL

Postman Inc. (2022). *What is Postman?* Descargado 2022-02-02, de https://www.postman.com/product/what-is-postman/

Richards, M. (2015). *Microservices vs. ServiceOriented Architecture*.

Rodríguez, M. A., y León, C. M. (2012). *FUNDAMENTOS DE CLIMATOLOGÍA*.

Sadourny, R. (2006). *¿Podemos creer en la meteorología?* Madrid: Ediciones Akal, S.A. Descargado de https://books.google.com.ec/books?id=2PBvu1lBQqIC&pg=PA8&dq=meteorolog{í}a+importancia&hl=es&sa=X&ved=2ahUKEwiBpuKdqrL0AhWEZzABHVtrAekQ6AF6BAgKEAI#v=onepage&q=meteorolog{í}aimportancia&f=false

Sánchez, Ó. (2016). *Aplicaciones informáticas de bases de datos relacionales. Microsoft Access 2016*. Descargado de https://books.google.com.ec/books?id=pRCkCwAAQBAJ&printsec=frontcover&dq=base+de+datos+relacionales&hl=es&sa=X&redir_esc=y#v=onepage&q=basededatosrelacionales&f=false

Silberschatz, A. B. L., Korth, H. F. B. L., y Sudarshan, S. (Instituto Indio de Tecnología, B. (2002). *Fundamentos de bases de datos*.

Villaizán, H. R. (2019). *Arquitectura de software basada en microservicios para implementación de la aplicación web de cobranza digital en Financial Systems Company SAC* (Tesis Doctoral). Descargado de https://repositorio.continental.edu.pe/handle/20.500.12394/6387https://repositorio.continental.edu.pe/bitstream/20.500.12394/6387/1/IV_FIN_103_TSP_Villaizan_Yamamoto_2019.pdf

Weir, L. (2019). *Enterprise API Management: Design and deliver valuable business APIs*. Descargado de https://books.google.com.ec/books?id=0OikDwAAQBAJ&printsec=frontcover&dq=API&hl=es&sa=X&redir_esc=y#v=onepage&q&f=false

Zea Ordóñez, M. P., Molina Ríos, J. R., y Redrován Castillo, F. F. (2017). *ADMINISTRACIÓN DE BASES DE DATOS CON POSTGRESQL*. Descargado de https://books.google.com.pe/books?id=5-mkDgAAQBAJ&pg=PA5&hl=es&source=gbs_selected_pages&cad=2#v=onepage&q=robustez&f=false doi: 10.17993/ingytec.2017.18