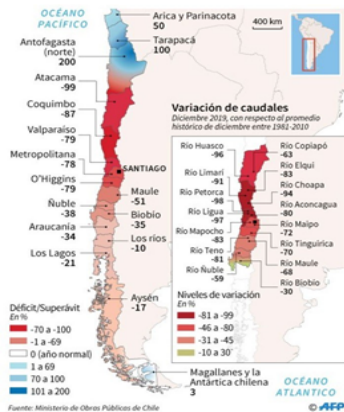


"Documento de Diseño"

API REST



Docente: Sebastián Salazar Molina
Fecha de entrega: 30/10/2021
GRUPO D

Integrantes: Rodrigo Alvarez Abello
Esteban Hernandez Muñoz
Facundo Alexandre Buchelli

Tabla de contenidos

1. Introducción	2
2. Objetivo General	2
3. Objetivos específicos	2
4. Alcances y limitaciones	2
5. Resolución de la problemática	3
5.1. Descripción del problema	3
5.2. Solución planteada	3
5.2.1. Obtención de datos históricos	3
5.2.1.1. Scraping y Web Scraping	3
5.2.1.1.1. Scraping de recursos estáticos	4
5.2.1.1.2. Scraping de recursos dinámicos	6
5.2.2. Modelamiento de datos	7
5.2.3. Estimación de indicadores meteorológicos	8
5.2.4. Exposición de datos	8
6. Tecnología utilizada	13
6.1. Dependencias	13
6.2. Dependencias de desarrollo	15
7. Conclusión	15
Anexo	16
1 - Instalación de PostgreSQL en Ubuntu 20.04LTS	16
2 - Instalación Docker en Ubuntu 20.04LTS	17

1. Introducción

En un mundo cada vez más golpeado por el cambio climático, cada vez se hacen más frecuentes grandes catástrofes, que ponen en jaque a nuestra sociedad y nos obligan a buscar soluciones nuevas e innovadoras para enfrentar estos episodios de crisis.

La sequía está golpeando a nuestro país, con mediciones históricas sobre déficit de lluvias y desertización. Chile se enfrenta a la peor sequía en más de medio siglo. Desde que en 1950 comenzaron las mediciones, el país, que ostenta la presencia del COP 25, nunca había mostrado una sequía tan prolongada, un déficit de lluvias y una desertización como la presente.

No es desconocido para nadie que nuestro país atraviesa una de las mayores mega sequías de la última década y un sostenido aumento de las temperaturas. La disminución de las lluvias tiene enormes consecuencias en la generación de electricidad, nuestra agricultura y la vida diaria, la falta de agua, se nota.

Debido a esto, en el siguiente informe se presenta la documentación para la creación de un sistema que funcione mediante una API de tipo rest, para que este nos entregue la información necesaria de las estaciones para poder estimar las posibles precipitaciones y temperaturas que estas tendrán en el día o en el futuro.

2. Objetivo General

Desarrollar un sistema que permita acceder a los datos históricos de las estaciones, a través de un API REST simple de consumir y de operaciones que permitan estimar la información de meteorología de los distintos lugares de nuestro país.

3. Objetivos específicos

1. Mostrar las características e indicadores meteorológicos solicitados para las estaciones.
2. Realizar procedimientos de scraping y web scraping para poblar una base de datos con los datos históricos de los indicadores solicitados.
3. Validación de tokens y credenciales de usuarios para acceder al sistema.
4. Utilizar operaciones del protocolo HTTP para la exposición de datos.

4. Alcances y limitaciones

- Alcances:
 - El sistema necesitará de un login de usuario y contraseña para la utilización de este.
 - El sistema permitirá crear usuarios.
 - El sistema entregará la información de la temperatura máxima, mínima y las precipitaciones de las estaciones.

- El sistema permitirá realizar búsquedas de los indicadores mencionados anteriormente.
 - El sistema permitirá generar estimaciones de los indicadores mencionados en el punto anterior para el día actual (en el momento de ejecución de la operación de estimación) a partir de los datos históricos.
 - El poblamiento de la base de datos es a través de scraping de fuentes estáticas y web scraping de fuentes dinámicas.
 - El sistema se actualizará diariamente con nuevos datos de precipitaciones y temperatura para las estaciones registradas.
- Limitaciones:
- La API entrega indicadores de temperatura y precipitación solamente para las estaciones registradas en la base de datos (aquellas obtenidas de las fuentes de información).
 - La operación de estimación a partir de los datos históricos no es de alta precisión y puede tener un alto error ya que se basa en un método estadístico simple (promedios móviles) que puede ser utilizado para predicciones meteorológicas pero no es el más adecuado para una predicción real de precisión ante un sistema tan complejo.
 - No se pueden obtener estimaciones de localidades puntuales de Chile en manera precisa, la operación de estimación estimará los indicadores solicitados para la estación registrada más cercana a la latitud y longitud entregada.

5. Resolución de la problemática

5.1. Descripción del problema

Se solicita crear una API que exponga indicadores meteorológicos de temperatura y precipitaciones y permita realizar predicciones de dichos indicadores para el día en que se solicita en una ubicación específica entregada mediante latitud y longitud. El sistema debe contener una gran cantidad de datos (datos históricos) de los indicadores solicitados y debe ser capaz de generar continuidad en los indicadores a partir de su actualización diaria.

5.2. Solución planteada

5.2.1. Obtención de datos históricos

5.2.1.1. Scraping y Web Scraping

Para el poblamiento de la base de datos con los indicadores relacionados a las temperaturas y precipitaciones desde 1950 al 2005 (considerados como recursos estáticos) se utilizaron métodos manuales de obtención de datos, es decir, se descargaron manualmente los archivos excel entregados por la fuente y colocados dentro de la carpeta del proyecto. Posteriormente se aplicó un método automatizado con librerías de análisis de archivos excel para la obtención de datos a partir del contenido de los archivos .xls entregados por la fuente y posteriormente ingresados a la base de datos.

Este proceso de recolección y poblamiento de la base de datos es realizado mediante un script que debe ser ejecutado por la persona que va a setear el sistema de forma manual cuando lo considere necesario, normalmente una sola vez al instalar el sistema en el ambiente donde vaya a correr la API.

Para el poblamiento de la base de datos con los indicadores entregados por la Dirección General de Aeronáutica Civil, se utilizó la API del gobierno que expone los recursos de precipitación, temperaturas y precipitaciones, entre otros indicadores del 2015 hasta el 2021 (hasta el primer semestre de 2021 a fecha de este documento). Para la obtención de los identificadores de los recursos a los cuales se tuvieron que consultar a la API se realizó un web scraping con el objetivo de obtener todas las URL's (endpoints) pertinentes.

Además de la recolección y almacenamiento de los datos históricos, se creó un servicio de web scraping que se ejecuta diariamente vía un "cron job" para la obtención de los indicadores diarios expuestos por la Dirección General de Aeronáutica Civil. Estos recursos fueron catalogados como "recursos dinámicos" ya que se actualizan diariamente.

El proceso de recolección, tratamiento y almacenamiento de los datos se mostrará con más detalle en los apartados siguientes, tanto para los recursos estáticos como los dinámicos.

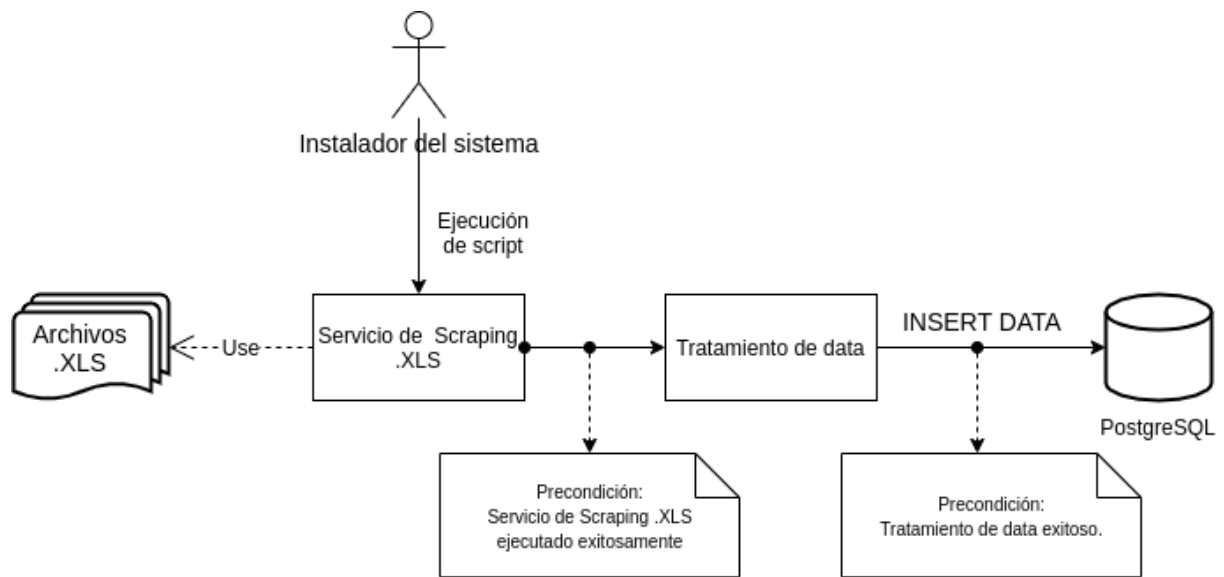
5.2.1.1.1. Scraping de recursos estáticos

Figura 1

ID	Recurso	Fuente	Método utilizado
RE1	Indicadores meteorológicos 1950-2005. Departamento de Geofísica de la Universidad de Chile.	http://www2.dgf.uchile.cl/AC T19/baseDMC/index.html	Descarga manual de fuentes + Tratamiento automatizado de archivos .xls (scraping de archivos excel)
RE2	Indicadores meteorológicos 2015-2021 (hasta la fecha). Gobierno de Chile, Dirección General de Aeronáutica Civil.	https://datos.gob.cl/organizacion/direccion_general_de_aeronautica_civil	Web Scraping de endpoints de recursos de API del gobierno + Solicitudes a la API de gobierno posterior

Fuente: Creación propia

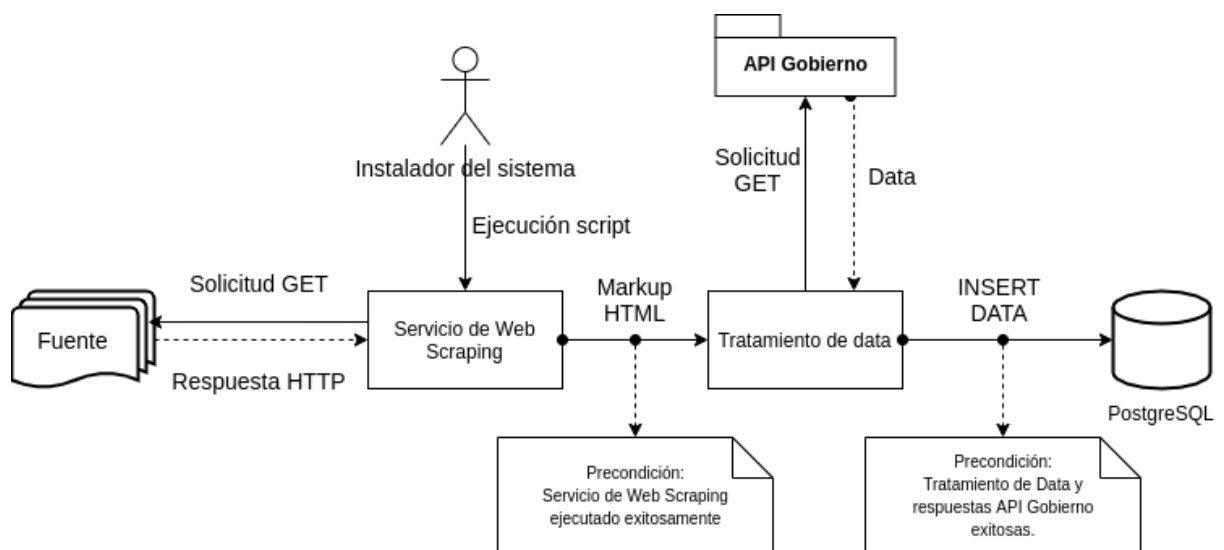
Figura 2



Descripción del flujo de scraping para la obtención de datos meteorológicos del recurso RE1.

Fuente: Creación propia.

Figura 3



Descripción del flujo de scraping para la obtención de datos meteorológicos del recurso RE2.

Fuente: Creación propia.

5.2.1.1.2. Scraping de recursos dinámicos

Tarea programada mediante un "cron job".

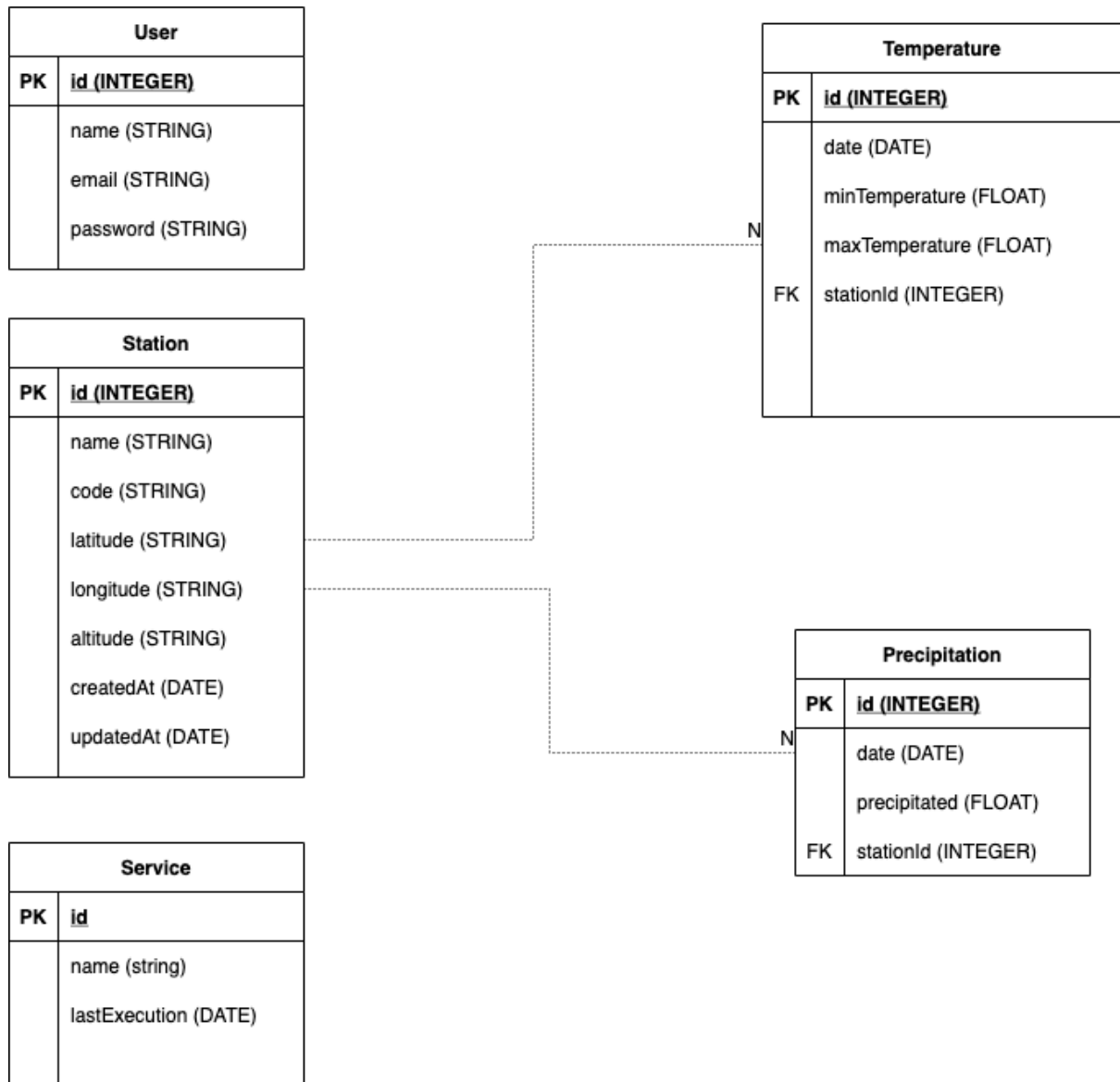
Figura 3

ID	Recurso	Fuente
RD1	Indicadores meteorológicos diarios. Gobierno de Chile, Dirección General de Aeronáutica Civil.	https://climatologia.meteochile.gob.cl/application/diario/boletinClimatologicoDiario/actual

Fuente: Creación propia

5.2.2. Modelamiento de datos

Explicación del modelo de datos aquí, justificando su forma...



5.2.3. Estimación de indicadores meteorológicos

Para realizar la estimación de los indicadores meteorológicos (temperatura mínima, temperatura máxima y precipitaciones) en una ubicación dada (latitud y longitud) se utilizó, sobre los datos históricos, un método estadístico denominado “Media móvil exponencial”.

El método de la media exponencial es una herramienta matemática utilizada como un indicador de tendencias que permite realizar predicciones para un período a partir de datos anteriores.

La media móvil exponencial, es una variación del promedio móvil simple (PMS):

$$PMS = \text{suma de } n \text{ temperaturas} / \text{cantidad } n \text{ de temperaturas}$$

donde

- I. n : Número de periodos que se desea promediar
- II. PMS : Promedio móvil simple de datos

El método de la media móvil exponencial es entonces similar al método de promedios móviles simples pero contiene un factor de suavizado que le asigna más peso (importancia) a los últimos datos registrados.

$$EMA(t) = \text{ValorHoy} * K + EMA(t - 1)*(1-K);$$

$$K = 2 / (N + 1);$$

donde

- I. t : Valor actual.
- II. t-1 : EMA anterior.
- III. K: Es el factor de suavizado.
- IV. N: Cantidad total de datos en el periodo seleccionado.

5.2.4. Exposición de datos

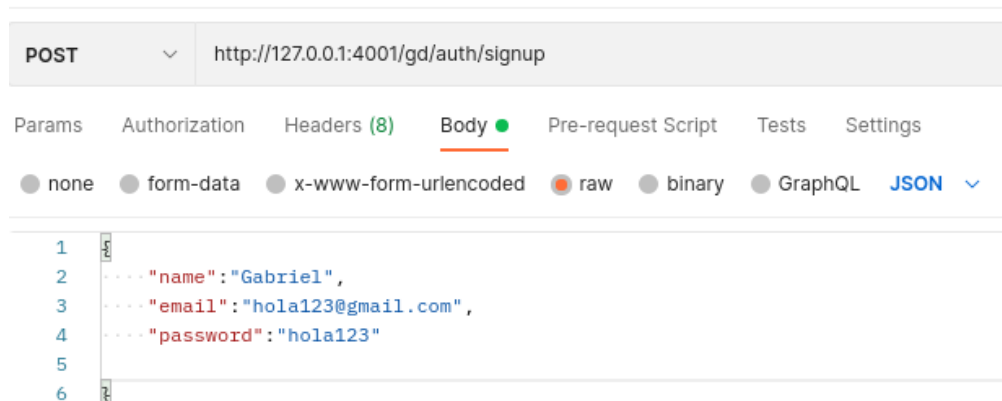
La forma en la cual se exponen los datos es a través de rutas de operaciones HTTP, en las cuales se usan para consultar la información necesaria según el criterio del cliente. Para realizar las consultas a la API, todas deben estar precedidas por el prefijo “/dg”.

Se utilizará el programa POSTMAN, el cual sirve para hacer consultas a distintas API mediante las operaciones disponibles que existen de HTTP.

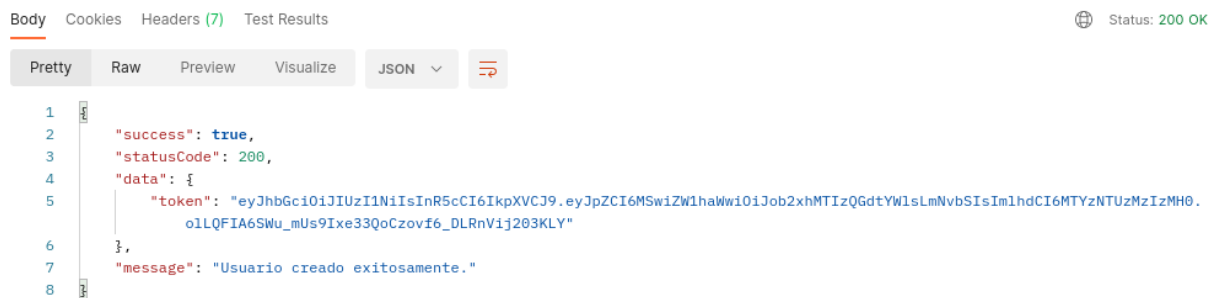
Para comenzar con la exposición de los datos es necesario registrar el usuario con el cual se comenzará a solicitar la información. Para realizar esto es necesario operar con el método POST de HTTP e ir a la ruta:

localhost:4001/dg/auth/signup

Esta ruta registra un nuevo usuario al sistema. Devuelve un Bearer Token para la autenticación del usuario. El modo de ingreso de datos para la creación del usuario es el siguiente:



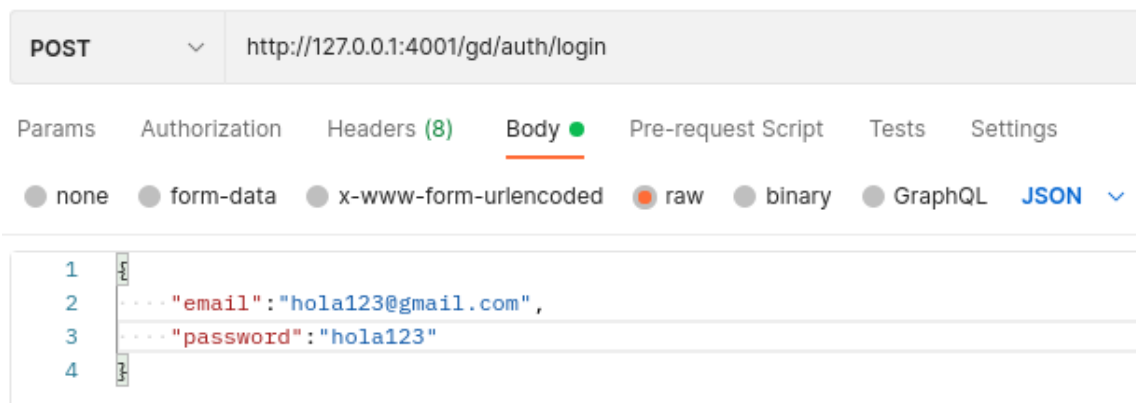
Y lo que retornará al enviar estos datos, será lo siguiente:



Luego, es necesario autenticar el usuario creado, para realizar las consultas de información a la API:

`localhost:4001/dg/auth/login`

Esta ruta es la principal para lograr hacer consultas a la API, ya que con esta operación se realiza la autenticación de los usuarios registrados en la base de datos, la cual entregará un token necesario para hacer las consultas de información sobre las estaciones.



La respuesta de la API a esta consulta será la siguiente:



The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (7), and Test Results. The 'Body' tab is selected, displaying a JSON response in 'Pretty' format. The status is '200 OK'. The JSON response is as follows:

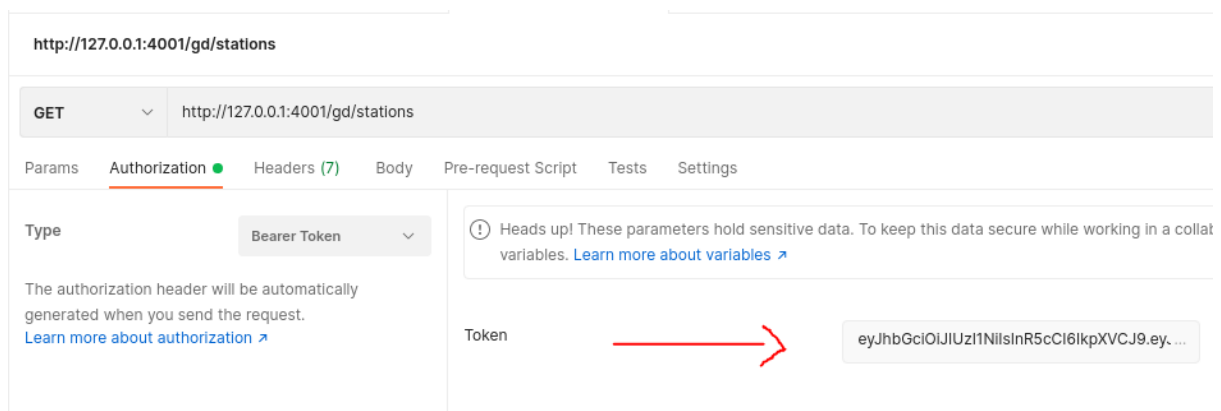
```
1 {
2   "success": true,
3   "statusCode": 200,
4   "data": {
5     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiOiJ0b2xhMTIzQGdtYWlsLmNvbSI6Im1ldCI6MTYzNTUzMzgweMX0.8fIEeSvhQcQrwb71qsUAvYz3G8pX8XFIzUSD_8Pi8gE"
6   },
7   "message": "Inicio de sesion exitoso."
8 }
```

Con el token de autenticación que nos entregará la API, será posible hacer la consulta de información de las estaciones, tanto como específicas o la estimación de la temperatura.

- Consulta de información

1.- **Estaciones:** Para realizar la consulta de las estaciones que están inscritas en nuestra base de datos, es necesario copiar el token obtenido en el inicio de sesión explicado anteriormente. Esta ruta utiliza el método GET de HTTP. Se ingresa a la pestaña de autenticación, se le pega el token copiado, y se realiza la consulta.

localhost:4001/gd/stations



Y la API mostrará las estaciones con sus respectivos parámetros en JSON, siendo esta, de la siguiente manera:

```

Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "success": true,
3   "data": {
4     "stations": [
5       {
6         "id": 1,
7         "name": "Chacalluta, Arica Ap.",
8         "code": "180005",
9         "latitude": "18&deg 21' 20'' S",
10        "longitude": null,
11        "altitude": "50",
12        "createdAt": "2021-10-29T17:32:49.168Z",
13        "updatedAt": "2021-10-29T17:32:49.168Z"
14      },
15      {
16        "id": 2,
17        "name": "Chacalluta, Arica Ap.",
18        "code": "180005",

```

1.1.- **Estación por código:** Esta función hace el retorno de una estación en específico según su código con el cual están registrados en la base de datos. Recibe el parámetro del código de estación. Es necesario ingresar el token de autenticación. Utiliza el método GET de HTTP.

localhost:4001/gd/stations/:StationCode

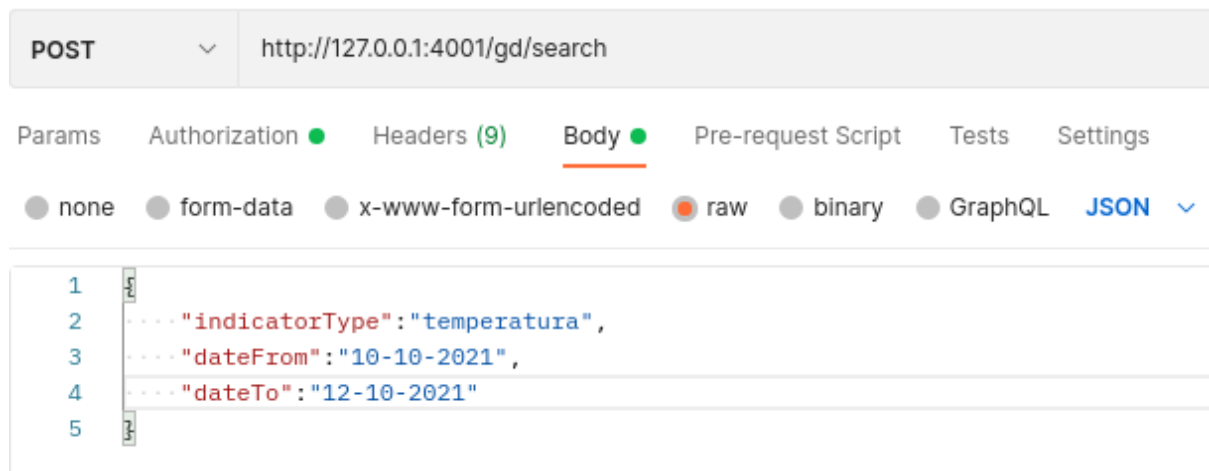
En donde “:StationCode”, es el código de la estación que se está solicitando la información.

```

GET http://127.0.0.1:4001/gd/stations/200006
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "success": true,
3   "data": {
4     "station": {
5       "id": 4,
6       "name": "Diego Aracena Iquique Ap.",
7       "code": "200006",
8       "latitude": "20&deg 32' 57'' S",
9       "longitude": null,
10      "altitude": "48",
11      "createdAt": "2021-10-29T17:32:50.583Z",
12      "updatedAt": "2021-10-29T17:32:50.583Z"
13    }
14  }
15 }

```

2.- **Búsqueda:** Esta operación trae un listado del tipo de indicador seleccionado entre dos fechas. Utiliza el método POST de HTTP. Es necesario el token de autenticación. El tipo de indicador que puede recibir la ruta puede ser temperatura, precipitacion, temperaturaMaxima, temperaturaMinima.

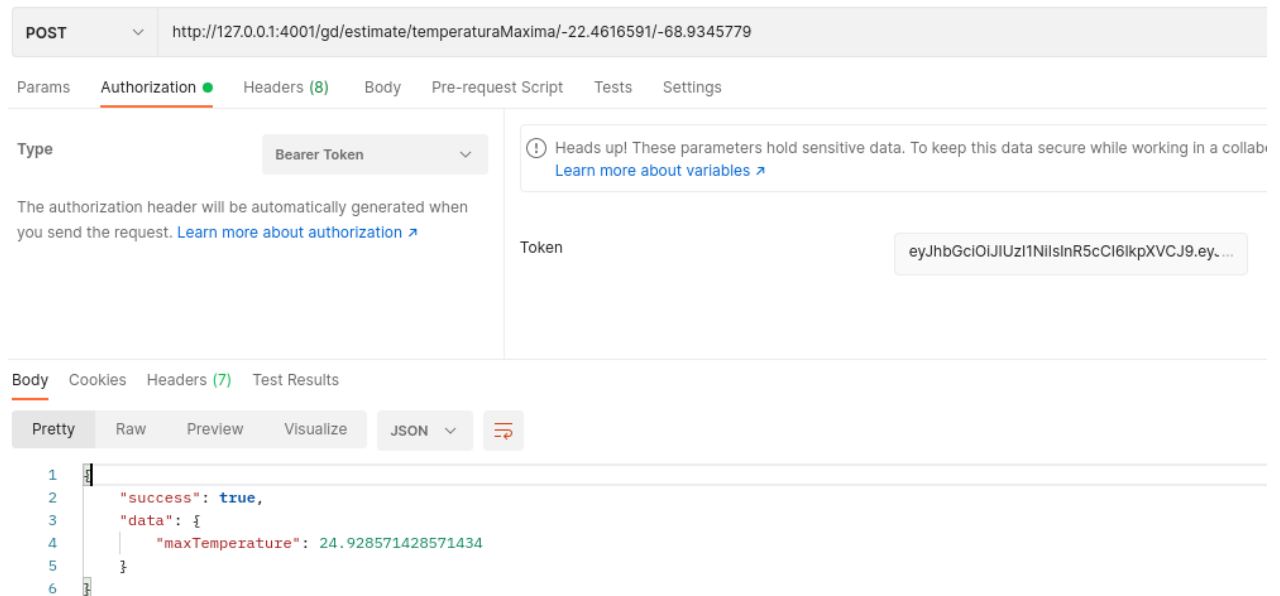


Y el listado que entregará la API será lo siguiente:



3.- **Estimación:** Esta operación hace la estimación para predecir una fecha futura según los datos históricos de los cuales se disponen. Los parámetros que recibe esta función son el indicador solicitado, la latitud y la longitud. Utiliza el método POST de HTTP. Se necesita autenticación para poder hacer la consulta de información.

El tipo de indicador puede ser temperatura, precipitación, temperaturaMaxima, temperaturaMinima.



6. Tecnología utilizada

¿Por qué utilizar Node.js? Se determinó implementar la API en Node JS debido a múltiples beneficios y funcionalidades las cuales fueron sus similitudes con JavaScript que hace que dicho lenguaje sea fácil de aprender ya que el equipo de trabajo posee conocimientos en JavaScript. Node.js puede ejecutarse en una variedad amplia de servidores, su rendimiento permite crear trabajos de gran calidad y disminuye el margen de experimentar errores técnicos, permite crear aplicaciones altamente escalables e innovadoras por lo que lo hace ideal para el desarrollo web. Node js es ideal para manejar aplicaciones de alto tráfico en cuanto a usuarios y eventos. Node.js es una buena opción de aplicaciones basadas en API REST, ya que JavaScript se utiliza tanto en el frontend como en el backend de los sitios. Así, un servidor puede comunicarse fácilmente con el frontend a través de APIs REST utilizando Node.js. Además, Node.js proporciona paquetes como Express.js y Koa que facilitan aún más la creación de aplicaciones web. Otro factor importante por el cual se escogió esta tecnología fue porque Node.js posee una manera sencilla de poder hallar soluciones ante posibles problemas que se nos presenten durante el proceso de desarrollo por tener una comunidad grande y de mucha actividad.

6.1. Dependencias

ID	Nombre	Descripción	Uso en la API
1	@googlemaps/google-maps-services-js	Wrapper para la utilización de la API de Google Maps desde NodeJS	En script para agregar valores de latitud y longitud exacta de las estaciones

2	@hapi/boom	Librería para la generación de objetos de errores HTTP	Dentro de los controladores para lanzar errores HTTP y dentro del middleware de validación de errores
3	axios	Cliente HTTP basado en Promesas para NodeJS y el navegador.	Dentro de los scripts de web scraping para realizar peticiones GET a los recursos de información
4	bcryptjs	Versión optimizada de bcrypt para JavaScript. Librería para encriptar contraseñas.	Para encriptar contraseñas de los usuarios que se registran en el sistema.
5	celebrate	Middleware para Express que funciona como wrapper para la librería Joi de validación de objetos en JavaScript.	Como middleware para la validación de estructura de las peticiones HTTP.
6	cheerio	Librería para tratar con markup y que provee una API para realizar operaciones sobre él.	Dentro de los scripts de web scraping para traer el markup de los recursos solicitados.
7	cors	Librería que permite habilitar CORS en el framework Express.	Para la habilitación de CORS en el servidor.
8	dotenv	Librería que permite cargar las variables de entorno del sistema al objeto "process.env" de NodeJS	Para la definición y la utilización de variables de entorno dentro del código.
9	express	Framework para el desarrollo de aplicaciones web.	Como marco de trabajo para el desarrollo de la aplicación y de las estructuras básicas de la aplicación web.
10	express-jwt	Middleware para Express que permite cargar al objeto de respuesta de Express "req.user" el usuario autenticado bajo las rutas solicitadas.	Para la autenticación de los usuarios dentro de las diferentes rutas de la API.
11	joi	Middleware de validación de las estructuras de objetos de JavaScript.	Para la validación de las estructuras de las peticiones que se realizan a la API.
12	jsonwebtoken	Librería para la transmisión segura de datos entre dos partes en JSON.	Para la autenticación de los usuarios dentro de las diferentes rutas de la API.
13	lodash	Librería de utilidades para JavaScript.	Se utilizan utilidades en algunas operaciones dentro de los scripts de Web Scraping y Scraping que requieren tratamiento de datos posterior.
14	moment-timezone	Librería para tratar con fechas dentro de una zona horaria especificada.	Se utilizan en todas partes donde se requiere la utilización de fechas y la operación con ellas.
15	morgan	Middleware para Express que funciona como logger para las peticiones HTTP que entran a la API.	Logger de peticiones para la API.

16	nanoid	Generador de identificadores únicos para JavaScript.	Dentro de los scripts de Web Scraping para identificar a las estaciones que no tienen identificadores asociados en sus fuentes.
17	node-cron	Módulo para programar tareas en NodeJS basado en el crontab de GNU	Para programar la tarea de hacer web scraping al boletín diario de indicadores meteorológicos.
18	pg	Cliente PostgreSQL para NodeJS	Para interactuar con la base de datos PostgreSQL
19	sequelize	ORM para las bases de datos PostgreSQL, MySQL, MariaDB, SQLite y SQL Server en NodeJS	Para interactuar con la base de datos PostgreSQL
20	swagger-jsdoc	Librería para leer anotaciones en JSDocs y transformarlas a documentación Swagger	Para la anotación y generación de documentación Swagger.
21	swagger-ui-express	Módulo para servir documentación Swagger autogenerada mediante un endpoint Express.	Para la generación de documentación Swagger.
22	xlsx	Módulo para escribir y leer distintos formatos de spreadsheets	Para leer los documentos xlsx y realizar scraping de los datos históricos almacenados en ese formato.

6.2. Dependencias de desarrollo

ID	Nombre	Descripción	Uso en la API
1	eslint	Linter para JavaScript	Asegurar ciertas reglas y estructuras de escritura en el código encontrando y corrigiendo errores de estilos de codificación y de buenas prácticas.
2	eslint-config-airbnb-base	Reglas para el Linter	Reglas de Linter de buenas prácticas de estilos de codificación de Airbnb
3	eslint-config-kentcdodds	Reglas para el Linter	-
4	eslint-plugin-prettier	Plugin Prettier para el Linter	-
5	nodemon	Herramienta para reiniciar el servidor Express cuando hay cambios	Herramienta para reiniciar el servidor Express cuando hay cambios
6	prettier	Formateador de código	Para formatear el código al guardar los archivos en base a lo configurado en el Linter
7	sequelize-cli	CLI de Sequelize para la generación de Modelos, Migraciones y Seeders	Como utilidad para generar elementos de la ORM rápidamente.

7. Conclusión

Node JS cumplió con nuestras expectativas, permitiéndonos realizar un desarrollo rápido y cómodo en cuanto a las funciones requeridas en este trabajo. Se logró acceder a los datos históricos de las estaciones, en el caso de los Scraping estáticos fue mediante el método de scraping de archivos Excel, web scraping de endpoints de recursos de API del gobierno y las solicitudes a la API de gobierno posterior. Por otro lado tenemos los Scraping de recursos dinámicos en donde la tarea programada para este caso fue mediante un "cron job" utilizando como recursos los indicadores meteorológicos diarios y la dirección general de Aeronáutica Civil. Otro punto importante a destacar, fue el haber conseguido realizar consultas a distintas API mediante el uso de POSTMAN a través de las operaciones disponibles de HTTP. Para la exposición de datos se debe registrar al usuario con el Método POST e ir a localhost:4001/dg/auth/signup para que este registre el nuevo usuario al sistema y devuelva un Bearer Token, este token se deberá conservar a nuestro cliente web y remitir en las peticiones posteriores que se hagan a la API. Una vez realizado el proceso de autenticar al usuario creado en la ruta principal para lograr el óptimo funcionamiento de realizar consultas a la API, se entrega un token que permite realizar las consultas de información de las estaciones o la estimación de la temperatura. Por todos estos logros y aprendizajes adquiridos en este proceso de desarrollo, es que se adquiere la resolución al problema principal propuesto a este proyecto, dejando a suficiencia de uso para el desarrollo móvil o fines estadísticos, los datos de la recopilación de información histórica.

Anexo

1 - Instalación de PostgreSQL en Ubuntu 20.04LTS

Pasos seguidos para la instalación de PostgreSQL ([Guía de instalación](#))

```
# Crear el archivo de configuración del repositorio:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt
$(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
```

```
# Importar la key para firmar el repositorio:
wget --quiet -O -
https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key
add -
```

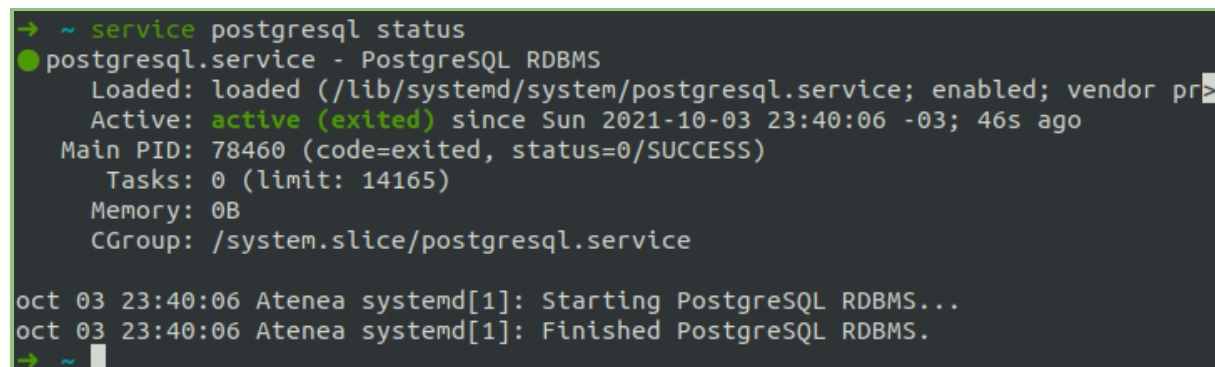
```
# Actualizar paquetes:
sudo apt-get update
```

```
# Instalar la última versión de PostgreSQL:
# (En caso de querer una versión específica, usar 'postgresql-12'
o similar especificando la versión requerida)
sudo apt-get -y install postgresql
```

La instalación va a pedir una contraseña, anotarla porque vamos a necesitarla para configurar el servidor en las variables de entorno (archivo .env), no es necesario que sea compleja si la base de datos va a estar local.

Para verificar que haya quedado bien instalado:

```
service postgresql status
```



```
→ ~ service postgresql status
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor pr>
   Active: active (exited) since Sun 2021-10-03 23:40:06 -03; 46s ago
   Main PID: 78460 (code=exited, status=0/SUCCESS)
     Tasks: 0 (limit: 14165)
    Memory: 0B
   CGroup: /system.slice/postgresql.service

oct 03 23:40:06 Atenea systemd[1]: Starting PostgreSQL RDBMS...
oct 03 23:40:06 Atenea systemd[1]: Finished PostgreSQL RDBMS.
→ ~
```

Resultado esperado al correr "service postgresql status"

Si del comando anterior, se obtiene un resultado de status **Active: inactive (dead)**, ejecutar el siguiente comando:

```
service postgresql start
```

2 - Instalación Docker en Ubuntu 20.04LTS

<https://docs.docker.com/engine/install/ubuntu/>