Documentación del Proyecto

Nombre del Proyecto: Aplicación para gestión de datos meteorológicos

Autor: Erlin Francisco Sapeg Soriano

Fecha: 17/02/2025

Asignatura: Desarrollo web en entorno servidor

Curso: 2024 – 2025

Grupo: 2WET

Índice

1. Descripción del proyecto

Este proyecto está compuesto de distintos módulos que conforman un servicio informativo de varios parámetros relacionados a previsiones meteorológicas.

En éste módulo se diseña una API para la recolección de datos meteorológicos a partir de estaciones de la AEMET. El objetivo principal es almacenar y analizar datos climáticos proporcionados, utilizando las tecnologías aprendidas en este módulo.

2. Requisitos

Para ejecutar el proyecto, es necesario contar con las siguientes herramientas:

- **XAMPP** (para el servidor MySQL y Apache)
- **PHP**
- **Laravel**
- **Composer**
- **MySQL Workbench**
- **Postman, Insomnia o navegador web** (para probar la API, opcional)

3. Arquitectura del Proyecto

El proyecto está basado en Laravel y sigue el patrón MVC (Modelo-Vista-Controlador):

- **Modelo:** Representa la base de datos y las entidades.
- **Vista:** No aplica en este caso (API REST).
- **Controlador:** Gestiona las peticiones HTTP.

Estructura del proyecto:

```
├── app/
│   ├── Console/
│   │   ├── Commands/
│   │   │   ├── Kernel.php
│   ├── Http/
│   │   ├── Controllers/
│   │   │   ├── Controller.php
│   │   │   ├── RecolectaInventario.php
```

```
│   ├── Models/
│   │   ├── Datos.php
│   │   ├── EstacionBd.php
│   │   ├── EstacionInv.php
│   │   ├── Inventario.php
│   │   ├── InventarioEst.php
│   │   ├── LogAcceso.php
│   │   ├── Rol.php
│   │   ├── Stat.php
│   │   ├── User.php
│   │   ├── Usuario.php
│   │   ├── UsuarioRol.php
│   ├── Providers/
│   │   ├── AppServiceProvider.php
├── boostrap/
│   ├── cache/
│   │   ├── .gitignore
│   ├── app.php
│   │   ├── providers.php
├── config/
│   ├── app.php
│   ├── auth.php
│   ├── cache.php
│   ├── database.php
│   ├── filesystems.php
│   ├── logging.php
│   ├── mail.php
│   ├── models.php
```

```
│   ├── queue.php
│   ├── sanctum.php
│   ├── services.php
│   ├── session.php
├── database/
│   ├── factories/
│   │   ├── UserFactory.php
│   ├── migrations/
│   ├── seeders/
│   │   ├── DatabaseSeeder.php
│   ├── .gitignore
│   ├── node_modules
├── public/
│   ├── .htaccess
│   ├── favicon.ico
│   ├── index.php
│   ├── robots.txt
├── resources/
│   ├── css/
│   │   ├── app.css
│   ├── js/
│   │   ├── app.js
│   │   ├── boostrap.js
│   ├── views/
│   │   ├── welcome.blade.php
├── routes/
│   ├── api.php
│   ├── console.php
```

```
│   ├── web.php
├── scripts/
├── storage/
│   ├── app/
│   ├── framework/
│   ├── logs/
├── tests/
│   ├── Feature/
│   │   ├── Example.php
│   ├──Unit/
│   │   ├── ExampleTest.php
│   ├── TestCase.php
├── vendor
```

4. Base de Datos

La base de datos está compuesta por las siguientes tablas:

Tabla: datos

Campo	Tipo	Descripción
id	INT	Clave primaria
id_estacion	INT	Relación con la tabla estaciones
idema	VARCHAR(10)	Identificador de la estación
fecha	DATETIME	Fecha y hora de los datos
ta	FLOAT	Temperatura
hr	FLOAT	Humedad relativa
prec	FLOAT	Precipitaciones

Tabla: estacion_bd

Campo	Tipo	Descripción
id	INT	Clave primaria
estado	VARCHAR(10)	Identifica si la estación está activa

Tabla: estacion_inv

Campo	Tipo	Descripción
id	INT	Clave primaria
nombre	VARCHAR(64)	Recoge el nombre de las estaciones
idema	VARCHAR(8)	Identificador de la estación
provincia	VARCHAR(32)	Recoge la provincia en la que se encuentra la estación
latitud	FLOAT	Latitud de la estación
longitud	FLOAT	Longitud de la estación
altitud	INT	Altitud de la estación

Tabla: log_acceso

Campo	Tipo	Descripción
id	INT	Clave primaria
Id_usuario	INT	Clave foránea que guarda el id del usuario
fecha_reg	DATETIME	Fecha del registro
descripcion	VARCHAR(128)	Calve foránea que referencia a id_usuario

Tabla: rol

Campo	Tipo	Descripción
id	INT	Clave primaria auto incremental
descripcion	VARCHAR(64)	Descripción del rol

Tabla: stats

Campo	Tipo	Descripción
id	INT	Clave primaria
id_estacion	INT	Relación con la tabla estaciones
idema	VARCHAR(10)	Identificador de la estación
fecha	DATETIME	Fecha y hora de los datos
ta	FLOAT	Temperatura
hr	FLOAT	Humedad relativa
prec	FLOAT	Precipitaciones

Tabla: usuario

Campo	Tipo	Descripción
id	INT	Clave primaria auto incremental
nombre	VARCHAR(32)	Nombre del usuario
passwd	VARCHAR(32)	Contraseña asignada al usuario
email	VARCHAR(48)	Correo electrónico del usuario

Tabla: usuario_rol

Campo	Tipo	Descripción
id	INT	Clave primaria
Id_usuario	INT	Clave foránea de usuario(id)
Id_rol	INT	Clave foránea de rol(id)
Fecha_creacion	TIMESTAMP	Registra la fecha de creación

5. Endpoints de la API

5.1 Obtener todas las estaciones

Endpoint: `GET /api/getAllStations`

Descripción: Retoma todas las estaciones disponibles en la API y las guarda en la base de datos.

5.2 Obtener datos de todas las estaciones

Endpoint: `GET /api/datos`

Descripción: Retorna todos los datos de la API sobre las estaciones almacenadas en la base de datos.

5.3 Procesar datos

Endpoint: `GET /api/procesar`

Descripción: Retorna los datos almacenados en la tabla ‘datos’ y los devuelve.

5.4 Almacenar datos

Endpoint: `GET /api/almacenar`

Descripción: Obtiene y almacena en la tabla 'stats’ los datos meteorológicos de todas las estaciones registradas.

6. Funcionamiento del Proyecto

6.1 Instalación

1. Clonar el repositorio:

   git clone https://github.com/Kevin-Zurita/Trabajo-Estadisticas.git

2. Instalar dependencias:

Composer install

3. Configurar el archivo .env:
   DB_CONNECTION=mysql
   DB_HOST=127.0.0.1
   DB_PORT=3306
   DATABASE=medio_ambiente
   DB_USERNAME=root
4. Añadir una configuración para la API KEY:
   AEMET_API_KEY=Tu_API_KEY
5. Ejecutar la migraciones:
   php artisan migrate
6. Ejecutar servidor:

   php artisan serve

6.2 Ejecución del recolector de estaciones

Para la obtención de estaciones desde la API de AEMET, se ha creado el siguiente controlador desde laravel:

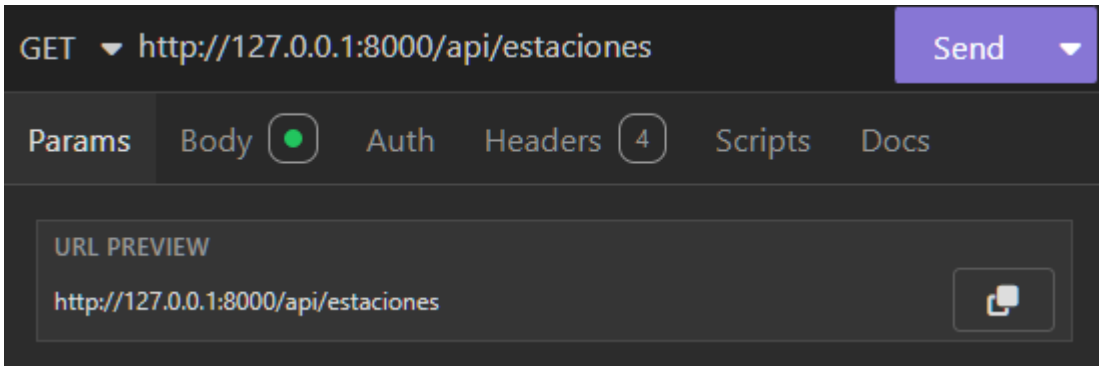
php artisan make:controller RecolectaInventario

Y a continuación se ha diseñado la siguiente función getAllStations():

6.2.3 getAllStations()

- **Propósito:** Obtiene todas las estaciones meteorológicas desde la API de AEMET y las guarda en la base de datos.
- **Pasos:**
  - Se conecta a la API de AEMET con una clave de acceso.
  - Obtiene la URL de los datos de estaciones.
  - Descarga los datos de la URL obtenida y los decodifica en formato JSON.
  - Filtra 10 estaciones con provincias únicas y las guarda en la base de datos (`estaciones_inv`).
  - Registra en los logs las provincias encontradas y devuelve un mensaje con el número de estaciones guardadas.
- **Manejo de errores:** Si ocurre un error en la API o el formato de datos es incorrecto, devuelve un mensaje de error en JSON.

Para obtener los datos desde la API de AEMET, se debe ejecutar el endpoint GET `/api/estaciones`. Esto se puede hacer con Postman, Insomnia o desde un navegador web:



Si se realiza con éxito parecerá el siguiente mensaje:

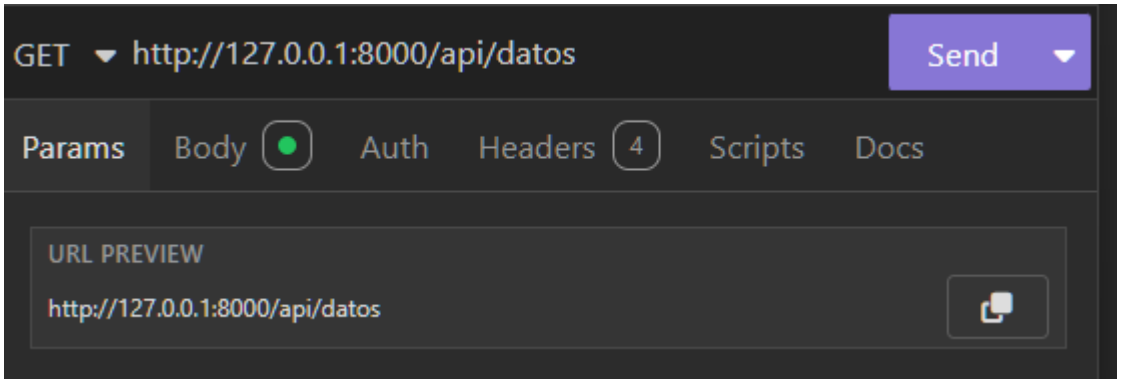
6.3 Ejecución del recolector de datos

A continuación se diseña la función para recolectar los datos:

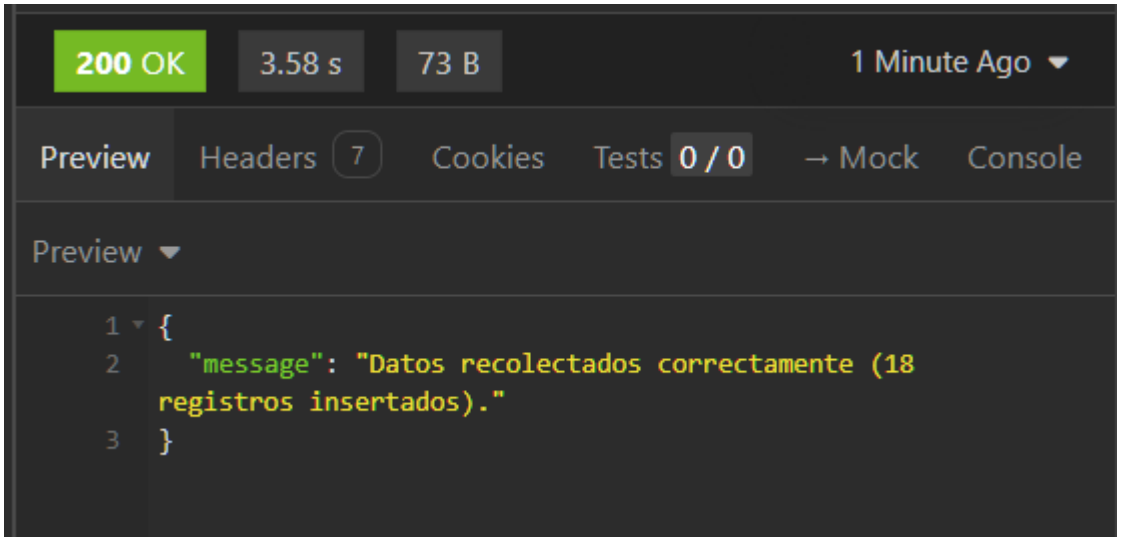
6.3.1 recolectaDatos()

- **Propósito:** Obtiene datos climáticos en tiempo real para cada estación guardada y los almacena en la tabla `datos`.
- **Pasos:**
  - Obtiene todas las estaciones desde la base de datos.
  - Para cada estación, consulta la API de AEMET para obtener sus datos meteorológicos.
  - Filtra los registros nuevos (evitando insertar datos duplicados).
  - Inserta los datos en la tabla `datos` si son nuevos.
  - Retorna un mensaje indicando cuántos datos se han guardado.
- **Manejo de errores:**
  - Si una estación no tiene datos, registra una advertencia en los logs.
  - Si hay un error en la API o en el formato de datos, lo registra en los logs y continúa con la siguiente estación.

Para obtener los datos desde la API de AEMET, se debe ejecutar el endpoint GET `/api/datos`. Esto se puede hacer con Postman, Insomnia o desde un navegador web:



Si se realiza con éxito aparecerá el siguiente mensaje:



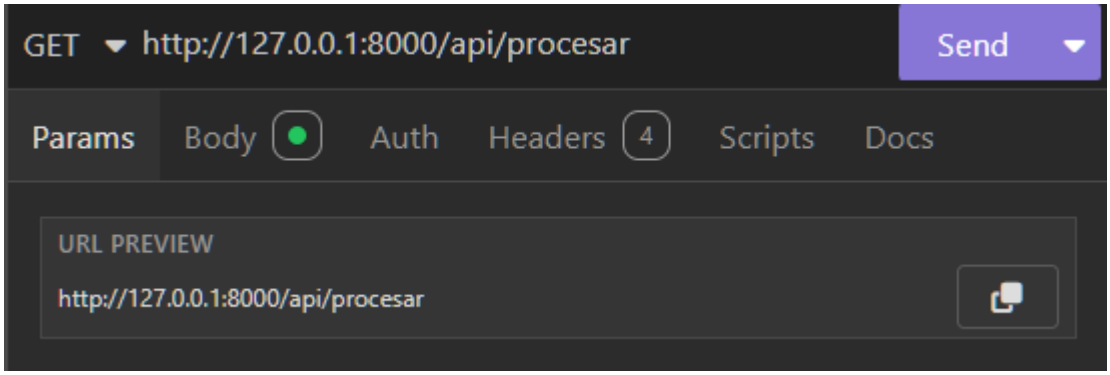
6.4 Ejecución del procesamiento de datos

La función que se encarga de éste procedimiento es procesaStat().

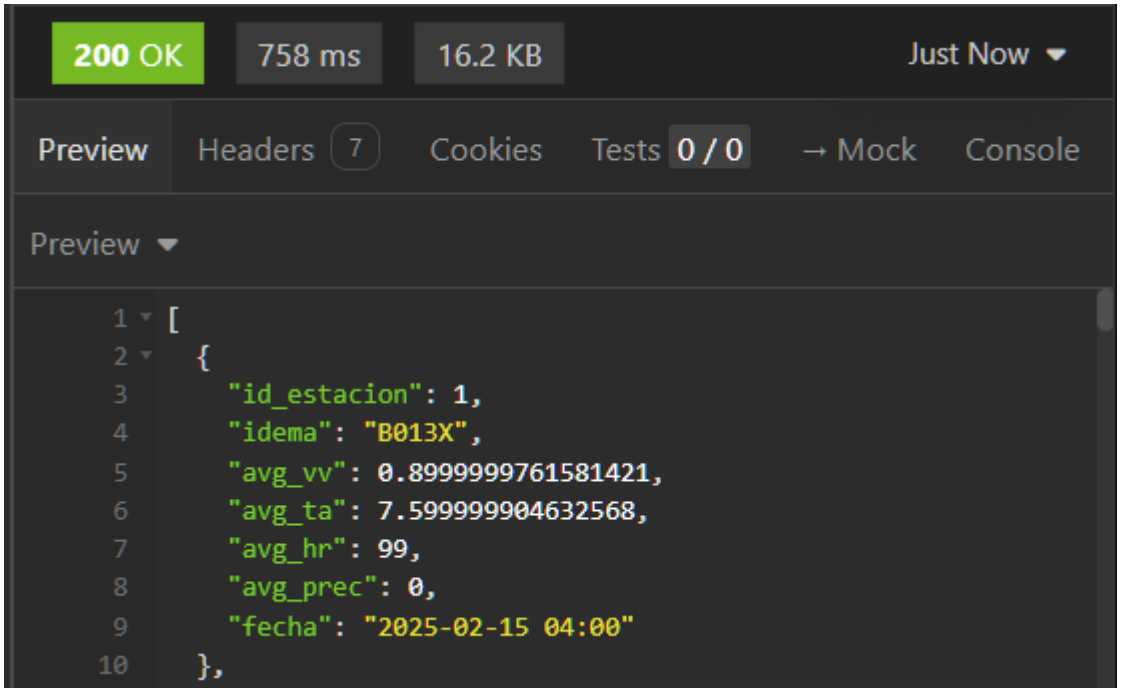
6.4.1 procesaStat()

- **Propósito:** Calcula los promedios de las variables meteorológicas (`vv`, `ta`, `hr`, `prec`) agrupados por estación y fecha.
- **Pasos:**
  - Consulta la base de datos `datos` y agrupa por estación y fecha.
  - Calcula los promedios de cada variable (`vv`: velocidad del viento, `ta`: temperatura, `hr`: humedad relativa, `prec`: precipitación).
  - Retorna los datos con los promedios calculados en formato JSON.
- **Manejo de errores:** Si no hay datos disponibles, devuelve un mensaje indicando que no hay nada para procesar.

Para obtener los datos desde la API de AEMET, se debe ejecutar el endpoint GET `/api/procesar`. Esto se puede hacer con Postman, Insomnia o desde un navegador web:



Lanzará el siguiente resultado con los datos de cada estación a cada hora:



6.5 Ejecución del almacenamiento de datos

Después la función almacenarStats($datosEstacion) guardará los datos en la base de datos y por último se ejecuta la función procesaYalmacena().

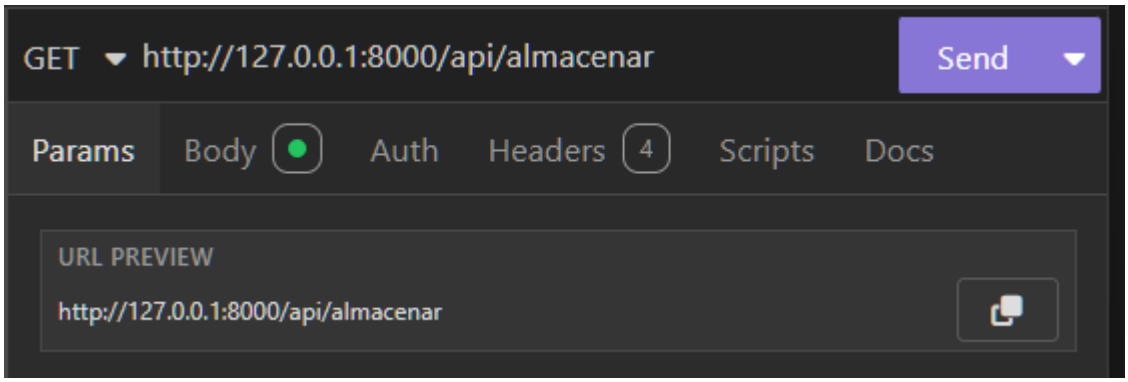
6.5.1. almacenarStats($datosEstacion)

- **Propósito:** Guarda los datos estadísticos procesados en la tabla `stats`.
- **Pasos:**
  - Recorre los registros generados por `procesaStat()`.
  - Inserta los valores promedio en la tabla `stats`.
  - Registra en los logs los valores procesados para cada estación.
  - Retorna un mensaje de confirmación en JSON.

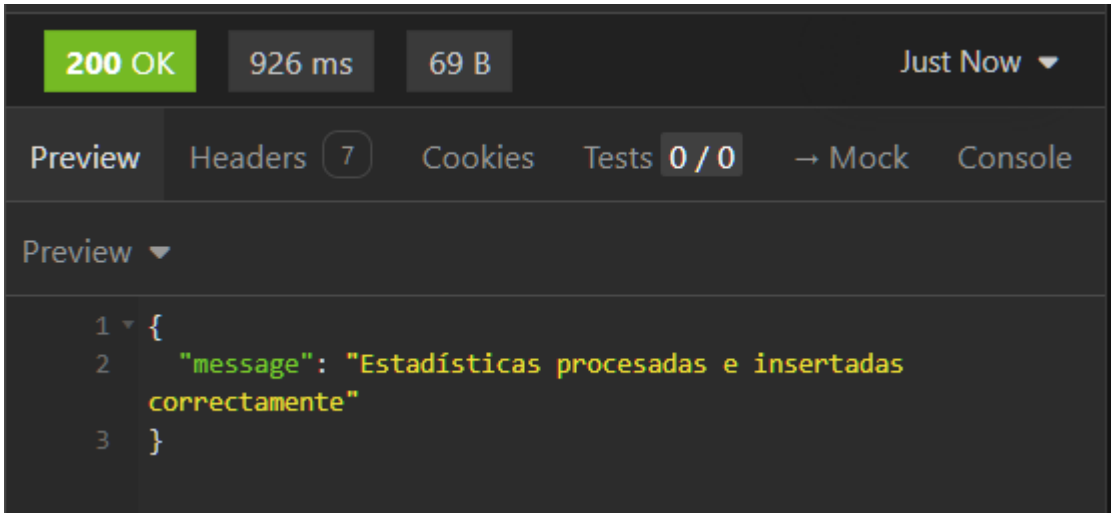
6.5.2. procesaYalmacenaStat()

- **Propósito:** Es una combinación de `procesaStat()` y `almacenarStats()`.
- **Pasos:**
    - Ejecuta `procesaStat()` para obtener los promedios climáticos.
    - Llama a `almacenarStats()` para guardar esos datos en la base de datos.

Para obtener los datos desde la API de AEMET, se debe ejecutar el endpoint GET `/api/almacenar`. Esto se puede hacer con Postman, Insomnia o desde un navegador web:



Dará el siguiente mensaje:



7. Configuración de Schedule

Laravel usa el programador de tareas (`schedule:run`) para ejecutar comandos automáticamente. En nuestro caso, configuramos dos tareas en route/`console.php` que llamarán a nuestras dos funciones:

- `recolecta:datos` → Se ejecuta una vez cada 6 horas para recolectar datos.
- `procesar:estadisticas` → Se ejecuta semanalmente para procesar estadísticas.

En `routes/console.php` definimos los comandos de la siguiente manera:

```php
<?php
use App\Http\Controllers\RecolectaInventario;
use Illuminate\Support\Facades\Schedule;

Schedule::call(callback: function (): void {
    (new RecolectaInventario())->recolectaDatos();
})->everySixHours();

Schedule::call(callback: function (): void {
    (new RecolectaInventario())->procesaYalmacenaStat();
})->weekly();
```

A continuación, configuramos una tarea programada con el Programador de Tareas de Windows:

- Nombre: Aemet Api.
- Se ejecuta si el usuario ha iniciado sesión o no (para que funcione en segundo plano).
- Se ejecuta con los privilegios más altos.
- El disparador repite la tarea cada 6 horas.
- En el apartado “Programa o script”, dentro de la pestaña “Acciones”, hemos escrito la ruta **C:\xampp\php\php.exe.**
- Agregamos en el apartado “Argumentos”: **artisan schedule:run.**
- En “Iniciar en (opcional)” agremamos la ruta del proyecto: **C:\Users\Usuario\Desktop\Trabajo-Estadisticas\Trabajo_Estadisticas.**

8. Problemas y Soluciones

Error: `No se encontraron datos para las estaciones`

Solución: Verificar que la API de AEMET esté en funcionamiento y que las estaciones estén correctamente registradas en la base de datos.

Error: `MySQL shutdown unexpectedly`

Solución:

1. Detener MySQL en XAMPP.
2. Desinstalar Xampp y reinstalar
3. Reiniciar MySQL.

Error: `Failed to Connect to MySQL at 127.0.0.1:3306`

Solución:

Asegurarse de que MySQL está corriendo y que los datos de conexión en `.env` son correctos.

Error: No commands are ready to run

En un principio, se configuró un `schedule` en `Kernel.php` para que `recolectaDatos` se ejecutara una vez al día y `procesaYalmacenaStat` una vez a la semana. Además, se programó el `schedule:run` en el programador de tareas de Windows. Sin embargo, al ejecutar el comando, Laravel no detectaba los comandos programados.

Solución:

Se cambiaron los comandos en **routes/console.php** y se eliminó el archivo **Kernel.php** para que los comandos no causaran conflictos

9. Conclusión

Este proyecto implementa una API para la recolección de datos meteorológicos usando Laravel y MySQL. Se han considerado problemas comunes y soluciones para su correcto funcionamiento. Futuras mejoras podrían incluir una interfaz gráfica para visualizar los datos y la implementación de tareas automatizadas para la recolección periódica de datos.