Hola, Marta!

Aquí tenim un projectet que creiem que ens pots ajudar molt a tenir controlat els logs/metriques de certes parts de FarmaOffice.

Objectiu: Api que recull logs/Metriques i els emmagatzema a una Base de dades.

Hem de desenvolupar una api que recull els logs/Metriques de diferents servidors i els emmagatzema.

Primer cas d'ús:
Volem emmagatzemar tots els canvis de preu que tenim al sistema, ja que això ens permetrà identificar quan hi ha hagut un canvi de preu i qui ho ha canviat.

Tasques a desenvolupar:

1) Endpoint que rep les dades i les emmagatzema a la Base de dades
   -(separar bé la lògica de negoci a un servei no al controlador i del accés a dades)
   ¿Com ho fem? Doncs amb un patró que es diu Repository Pattern:
   -Utilitzar Repository Pattern :
   https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30

   (Això ens permetrà que si el dia de demà canviem d'un Mysql a un PostgreSql o ElasticSearch, la migració a nivell de codi és molt fàcil, només s'ha de fer una classe i ja està, a nivell del servei no cal tocar re)

   Exemple:
   Estructura d'arxius com a exemple (A tenir en compte que les primeres mètriques són preus de productes:

```
app/
├── Http/
│   ├── Controllers/
│   │   ├── MetricController.php
├── Repositories/
│   ├── Interfaces/
│   │   ├── MetricRepositoryInterface.php
│   ├── MetricMysqlRepository.php
├── Models/
│   ├── Metric.php
├── Services/
│   ├── MetricService.php
```

   Esquema Endpoint:

Route -> Controller -> MetricService -> MetricRepositoryInterface ->MetricMysqlRepository -> **BBDD**

Controller: Responsable de gestionar les peticions i enviarles al serveï.
ServiceProvider: Aquí està la lògica de negoci, que vol dir validar que les dades són correctes, per sumes, restes etc si s'han de fer etc
El servei apunta contra la interfície per a emmagatzemar les dades i la interfície apunta cap al LogRepositoryMysql.php que en el nostre cas es on hem de fer la implantació del eloquent contra la BBDD Mysql.

Si el dia de demà hem de canviar de Base de dades, només hem de crear una nova implementació de la interfície y posar les instruccions Insert, Update etc en un Nour LogRepositoryNomNovaBBDD.php

Mes info del Repository Pattern:
https://www.twilio.com/es-mx/blog/repository-pattern-in-laravel-application

2) Dashboard on poder visualitzar les dades (utilitzar Filament: https://filamentphp.com/docs/3.x/panels/installation)

   Aquest dashboard ha de poder filtrar per farmàcia i un cop per farmàcia per producte. Un cop hem filtrat, ha de sortir una gràfica lineal temporal amb el preu del producte al llarg del temps.

   Extra 1:
3) Utilitzar cues per encuar totes les peticions que rebrem a través de l'api, així podem anar processant asyncronament.
4) Afegir gestió de permisos d'accés al dashbaord amb https://jetstream.laravel.com/introduction.html

   Extra 2:
5) Afegir IA amb GPT per predir tendències de les mètriques i suggerir sol·lucións per evitar problemes. Fer resums de tendències de preus/productes

L'esquema de la base de dades és el següent:

```
CREATE TABLE logs_precios (
    id SERIAL PRIMARY KEY,
    product_id INT NOT NULL,
    pharmacy_id INT NOT NULL,
    old_price DECIMAL(10,2),
    new_price DECIMAL(10,2) NOT NULL,
    old_stock INT,
    new_stock INT NOT NULL DEFAULT 0,
    source VARCHAR(255) NOT NULL, -- Pot ser "admin", "system", "API", etc.
    change_date TIMESTAMP DEFAULT NOW()
);
```

Explicació dels camps

- **`id`** → Identificador únic de cada registre de canvi.
- **`product_id`** → ID del producte que ha canviat de preu i/o estoc.
- **`pharmacy_id`** → ID de la farmàcia associada al producte.
- **`old_price`** i **`new_price`** → Valors del preu abans i després del canvi.
- **`old_stock`** i **`new_stock`** → Quantitat d'unitats en estoc abans i després del canvi.
- **`source`** → Origen del canvi (per exemple: "admin", "system", "API").
- **`change_date`** → Data i hora en què s'ha registrat el canvi.