DETALLES DE LA SOLUCIÓN DE LA PRUEBA TÉCNICA.

TECNOLOGÍAS USADAS:

- PHP 8.1
- Laravel 10
- MySQL.
- Paquete Breeze con InertiaJS – React y TailwindCSS.

EXPLICACIÓN DE LA SOLUCIÓN GENERAL:

Al utilizar Laravel y Breeze ya tenía un punto de partida sólido.
Laravel utiliza arquitectura por capas así que aprovechando eso modifiqué y añadí funcionalidad gradualmente:

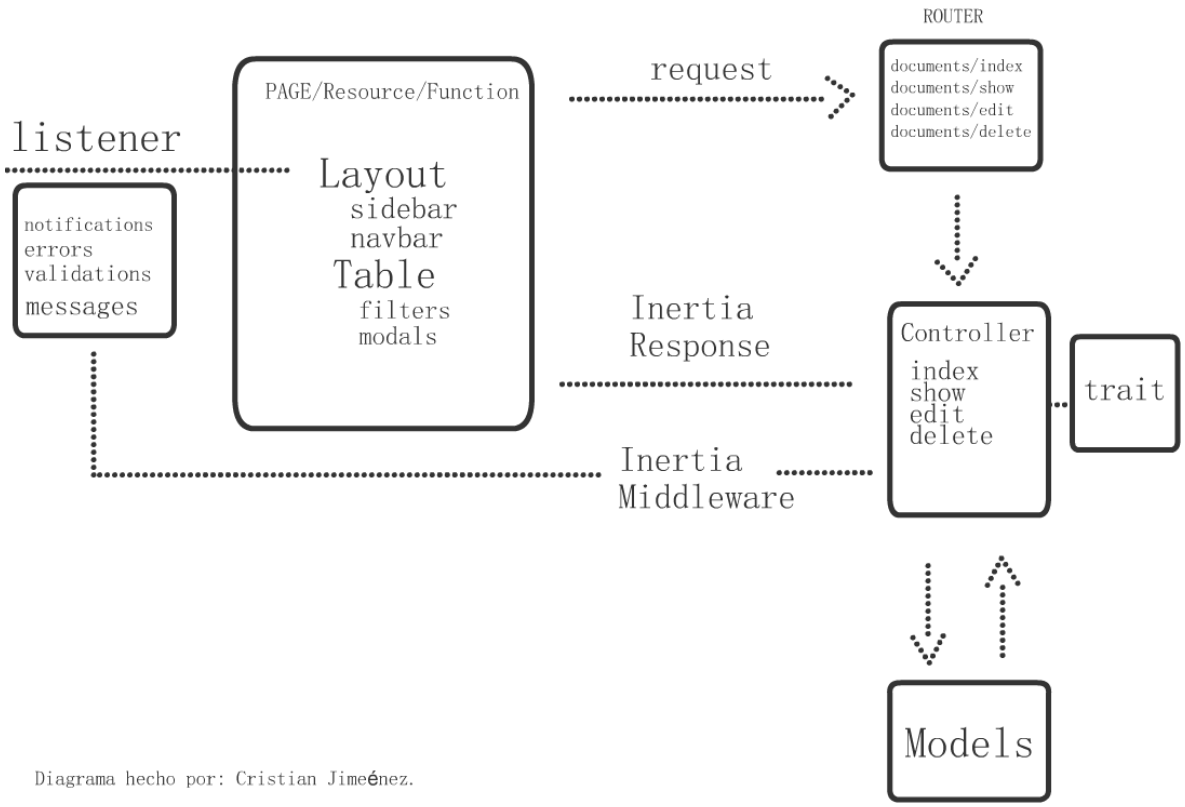
1. Primero, quité servicios de autenticación que no son necesarios para el proyecto y dejé solamente la validación de usuario (username y password).

2. Agregué las migraciones necesarias para crear las tablas y los índices que se pidieron. También agregué dos seeders que son clases que funcionan a modo de plantilla para crear 5 registros en la tabla tip_tipo_doc y 5 registros en la tabla pro_proceso.

3. Agregué modelos para utilizar el ORM Eloquent que tiene Laravel y facilitar el uso de consultas, agregué en el modelo las relaciones y métodos (scopes) para facilitar la toma de datos basada en filtros.

4. Agregué las rutas y el controlador para recibir las peticiones para el CRUD de la tabla doc_documento. Los métodos más complejos los implementé en un Trait que uso en el controlador, de esta manera el controlador principal contiene solo los métodos para renderizar la página correspondiente o responder a la petición.

5. Definí la estructura general del proyecto y la utilicé como punto de partida para todo el proyecto:



EXPLICACIÓN DE SOLUCIONES ESPECÍFICAS:

1. CÓDIGOS CONSECUTIVOS:

1.1. Para solucionar lo de los códigos consecutivos en la columna **doc_codigo** de la tabla **doc_documento** en primera instancia se me ocurrió hacerlo con el ID de cada documento, pero luego me di cuenta que al ingresar documentos con códigos diferentes se aumentarían y no sería consecutivos.

1.2. La solución que se me ocurrió fue crear una tabla (“codigos”) en donde voy almacenado cada código al momento de crear un documento o actualizarlo.

1.3. Finalmente creé un método para actualizar el listado de códigos cuando un documento es modificado o eliminado. Recorro primero todos los documentos con el código viejo (“que se va a eliminar”) y los modifico, hago la misma operación con los códigos (en la tabla codigos).

2. FILTROS DE BÚSQUEDAD Y PAGINACIÓN EN EL LISTADO DE DOCUMENTOS:

2.1. Implementé la funcionalidad de filtros en el listado de documentos para tomar los documentos cuyo tipo de documento o proceso los pueda seleccionar el usuario, también filtrar por nombre y paginar los resultados, el funcionamiento es sencillo, si el usuario envía algún filtro, evalúo en el controlador (en realidad en el trait que implementa el controlador) y con base en ello aplico los filtros al modelo y mando los datos resultantes como respuesta (para esto fue que agregué los scopes locales en el modelo Document de la tabla doc_documento).

2.2. EL ORM Eloquent de Laravel ya trae la funcionalidad para responder con datos paginados, lo único que hice fue crear el componente React mostrar los links.

3. NOTIFICACIONES:

Cuando se crea, se actualiza o se elimina un documento, envío un mensaje flash (mensaje que es válido solo para esa respuesta) y en la vista (componente de página) estoy evaluando si existe alguna notificación, si es así la muestro (Un uso similar al patrón Observer).

4. TABLA DE TIPOS DE DOCUMENTOS Y PROCESOS.

Listé las tablas tip_tipo_doc y pro_proceso usando una estructura similar a la mencionada para la tabla doc_documento, la diferencia es que no agregué la posibilidad de agregar o eliminar (quise hacerlo pero no iba a terminar a tiempo), así que solo agregué la funcionalidad de editar. Cuando se edita un tipo de documento o proceso, actualizo los códigos de los documentos y los códigos de la tabla “codigos”.

5. HELPERS Y PATRONES DE DISEÑO.

5.1. Laravel ya implementa algunos patrones de diseño como: Singleton, Dependency Injection, Facades, Factory, MVC, Observer, entre otros. Así que solo los utilicé (algunos) para mantener la consistencia.

5.2. Creé un archivo llamado Helpers.php en el cuál escribí varias funciones útiles para el desarrollo, luego utilice psr-4 para cargar el archivo y que esté disponible en todo el ciclo de vida del request.