Introducción

En el desarrollo de sistemas web es común la necesidad de **cargar grandes volúmenes de datos** de forma automática, evitando la inserción manual registro por registro. En el contexto de nuestro proyecto Laravel existente, surgió el requerimiento de importar un conjunto de ciudades a la base de datos de manera eficiente. Para ello se decidió **integrar la funcionalidad de carga masiva** a partir de archivos de hoja de cálculo (Excel) en formato XLSX, así como archivos de texto delimitado (CSV). Esta integración permite que los usuarios administradores puedan poblar la tabla de ciudades (cities) con numerosos registros en un solo paso, **ahorrando tiempo y minimizando errores humanos** en comparación con la carga manual individual.

Objetivos

Los objetivos principales de esta integración son:

- **Permitir la carga masiva de datos de ciudades** a partir de un archivo CSV o Excel (.xlsx) proporcionado por el usuario.

- **Insertar automáticamente registros en la tabla** cities de la base de datos, mapeando las columnas del archivo a los campos name (nombre de la ciudad) y description (descripción de la ciudad).

- **Validar y procesar el archivo de entrada**, manejando posibles errores (formato incorrecto, datos faltantes, etc.) y proporcionando retroalimentación al usuario sobre el resultado de la importación.

- **Integrar la solución al proyecto Laravel existente** de forma armónica, utilizando herramientas y paquetes adecuados que garanticen eficiencia y mantenibilidad del código.

Herramientas Utilizadas

Para desarrollar esta funcionalidad se utilizaron las siguientes herramientas y tecnologías, cada una cumpliendo un rol específico dentro del proyecto:

- **Laravel 10 (Framework PHP)**: Framework MVC en PHP utilizado como base del proyecto. Laravel provee una estructura organizada, manejo de rutas, controladores, vistas Blade y abstrae la interacción con la base de datos (Eloquent ORM). En este proyecto existente, Laravel facilita la integración de nuevas características como la

importación de archivos de forma consistente con el resto de la aplicación.

- **PHP 8**: Lenguaje de programación en el que está escrito Laravel y todo el backend de la aplicación. PHP es el responsable de ejecutar la lógica del servidor. En la implementación actual, se emplea PHP para procesar el archivo subido, leer su contenido y realizar las operaciones de inserción en la base de datos a través de las funciones provistas por Laravel y la biblioteca de Excel.

- **Composer**: Gestor de dependencias de PHP. Se utilizó Composer para agregar al proyecto el paquete **Laravel Excel** (maatwebsite/excel) y gestionar automáticamente la descarga e inclusión de sus dependencias. Con un simple comando (composer require maatwebsite/excel), Composer incorporó la librería al proyecto, facilitando su carga y autoconfiguración gracias al mecanismo de *auto-discovery* de Laravel.

- **Laravel Excel (maatwebsite/excel)**: Paquete de Laravel que simplifica la importación y exportación de archivos Excel/CSV. Laravel Excel es mantenido por **Maatwebsite** y permite leer archivos Excel (.xlsx y .xls) y CSV, convirtiendo su contenido en colecciones de datos o modelos de Eloquent de forma eficiente. A continuación se detallan los pasos de uso de esta herramienta en nuestra integración:

    1. **Instalación**: Se instaló mediante Composer con el comando composer require maatwebsite/excel. Esto agregó la librería al proyecto y Laravel la detectó automáticamente (no fue necesario registrar proveedores de servicio manualmente, ya que a partir de Laravel 5.5 el paquete soporta *auto-discovery*).

    2. **Configuración**: Laravel Excel publicó un archivo de configuración config/excel.php (opcionalmente generado con php artisan vendor:publish si se requiere personalizar opciones). Para nuestro caso, no fue necesario modificar la configuración por defecto. El paquete soporta de forma predeterminada los formatos **CSV, XLSX y XLS**, determinando el tipo de importación según la extensión del archivo .

    3. **Creación de clase de importación**: Siguiendo las recomendaciones de la documentación, se generó una clase *import* específica mediante Artisan: php artisan make:import CitiesImport --model=City. Esta clase define cómo interpretar cada fila del archivo y transformarla en un modelo City para ser guardado. (Se detalla su contenido más adelante).

    4. **Uso en el controlador**: En el método del controlador encargado de la importación, se utilizó el *facade* Excel que provee el paquete. Mediante la llamada Excel::import(new CitiesImport, $archivo) se procesa el archivo subido y se delega a la clase CitiesImport la creación de cada modelo City a insertar. El paquete se encarga internamente de recorrer las filas y utilizar nuestra clase para realizar la inserción de cada registro. Además, Laravel Excel maneja

automáticamente detalles como la conversión de tipos de datos y la omisión de la fila de encabezados cuando se utiliza la funcionalidad correspondiente (p. ej., *WithHeadingRow*).

En conjunto, estas herramientas proporcionaron una base sólida para implementar la funcionalidad de carga masiva de ciudades de manera rápida y confiable, integrándose sin problemas al ecosistema Laravel del proyecto.

Desarrollo

A continuación, se describe el desarrollo de la funcionalidad de importación de archivos CSV/XLSX, incluyendo los componentes implementados con fragmentos de código comentados. Esta sección cubre la creación del formulario de carga, las rutas definidas, la lógica del controlador con sus validaciones, la clase de importación utilizada y consideraciones sobre la estructura del archivo, manejo de errores e inserción de datos en la base de datos.

Formulario de carga de archivo (Vista Blade)

Para que el usuario pueda seleccionar y subir un archivo, se creó un formulario en Blade (por ejemplo, en una vista cities/import.blade.php). Dicho formulario contiene un campo de tipo *file* para elegir el archivo CSV o Excel, y un botón de envío para iniciar la importación. Es importante establecer enctype="multipart/form-data" en el formulario para permitir la transferencia de archivos. También se incluye un campo CSRF oculto (@csrf) que Laravel requiere para validar formularios. A continuación se muestra el código del formulario con comentarios explicativos:

web.php | import_form.blade.php | navigation.blade.php

resources > views > import_form.blade.php

```
<body class="bg-gray-100 min-h-screen flex items-center justify-center">
    <div class="bg-white shadow-lg rounded-xl p-8 max-w-lg w-full">
        @if ($errors->any())
            <div class="bg-red-100 text-red-700 px-4 py-3 rounded mb-4">
                <ul class="list-disc list-inside">
                    @foreach ($errors->all() as $error)
                        <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
        @endif

        @if (session('error'))
            <div class="bg-yellow-100 text-yellow-800 px-4 py-3 rounded mb-4">
                {{ session('error') }}
            </div>
        @endif

        <form action="{{ route('cities.import') }}" method="POST" enctype="multipart/form-data" class="space-y-6">
            @csrf

            <div>
                <label for="csv_file" class="block text-sm font-medium text-gray-700">Archivo CSV o XLSX</label>
                <input type="file" name="csv_file" accept=".csv,.xls,.xlsx" required
                       class="mt-2 block w-full text-sm text-gray-900 border border-gray-300 rounded-lg cursor-pointer bg-gray-50 focus:outline-no
            </div>

            <div class="text-center">
                <button type="submit"
                        class="inline-flex items-center px-6 py-2 bg-blue-600 text-white font-semibold rounded-lg shadow-md hover:bg-blue-700 tran
                    Importar
                </button>
            </div>
        </form>
```

En este formulario se valida del lado del cliente el tipo de archivo permitido mediante el atributo accept (aunque la validación principal se realizará del lado del servidor). El campo de archivo es obligatorio (required) y, tras seleccionar el archivo deseado, el usuario pulsará “Importar Datos” para enviar el formulario.

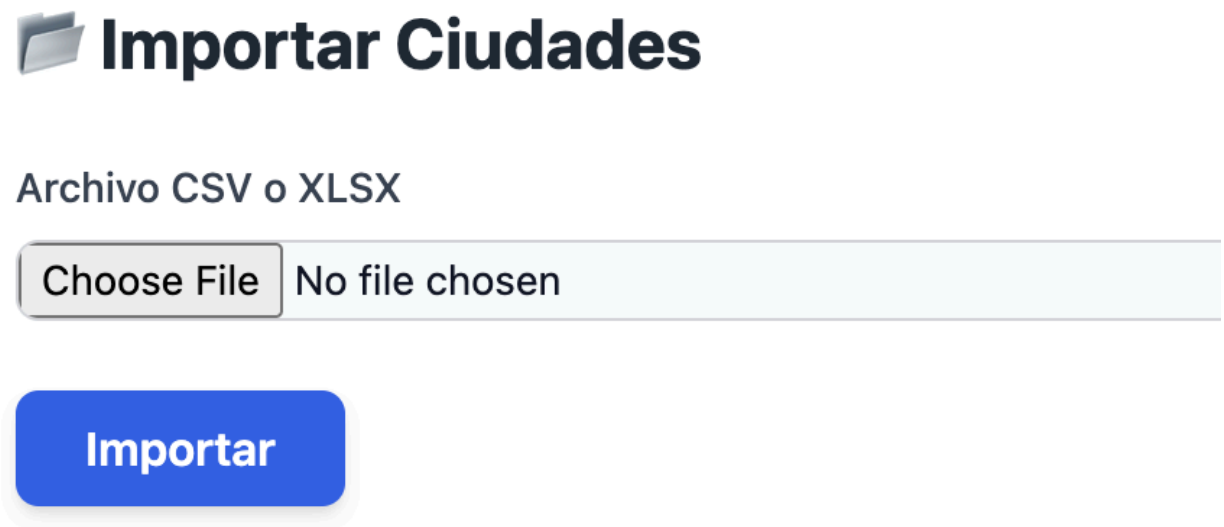


Figura 1. Interfaz de carga de archivo para la importación masiva de ciudades. El formulario permite al usuario elegir un archivo desde su equipo mediante un campo de tipo “file” y luego iniciar el proceso de importación con el botón “Importar Datos”. Es fundamental el atributo enctype="multipart/form-data" en el formulario para asegurar que el archivo se transmita correctamente al servidor. Una vez enviado el formulario, Laravel procesará la petición y pasará el archivo al controlador encargado de la importación.

Rutas definidas

Para canalizar la solicitud del formulario hacia el controlador apropiado, se definieron rutas web en el archivo routes/web.php. Se añadieron dos rutas: una para mostrar la vista del formulario de importación (GET) y otra para procesar el envío del formulario (POST). A continuación, las rutas establecidas:

```
Route::get('/import-cities', fn() => view('import_form'));
Route::post('/import-cities', [CityImportController::class, 'import'])->name('cities.import');
```

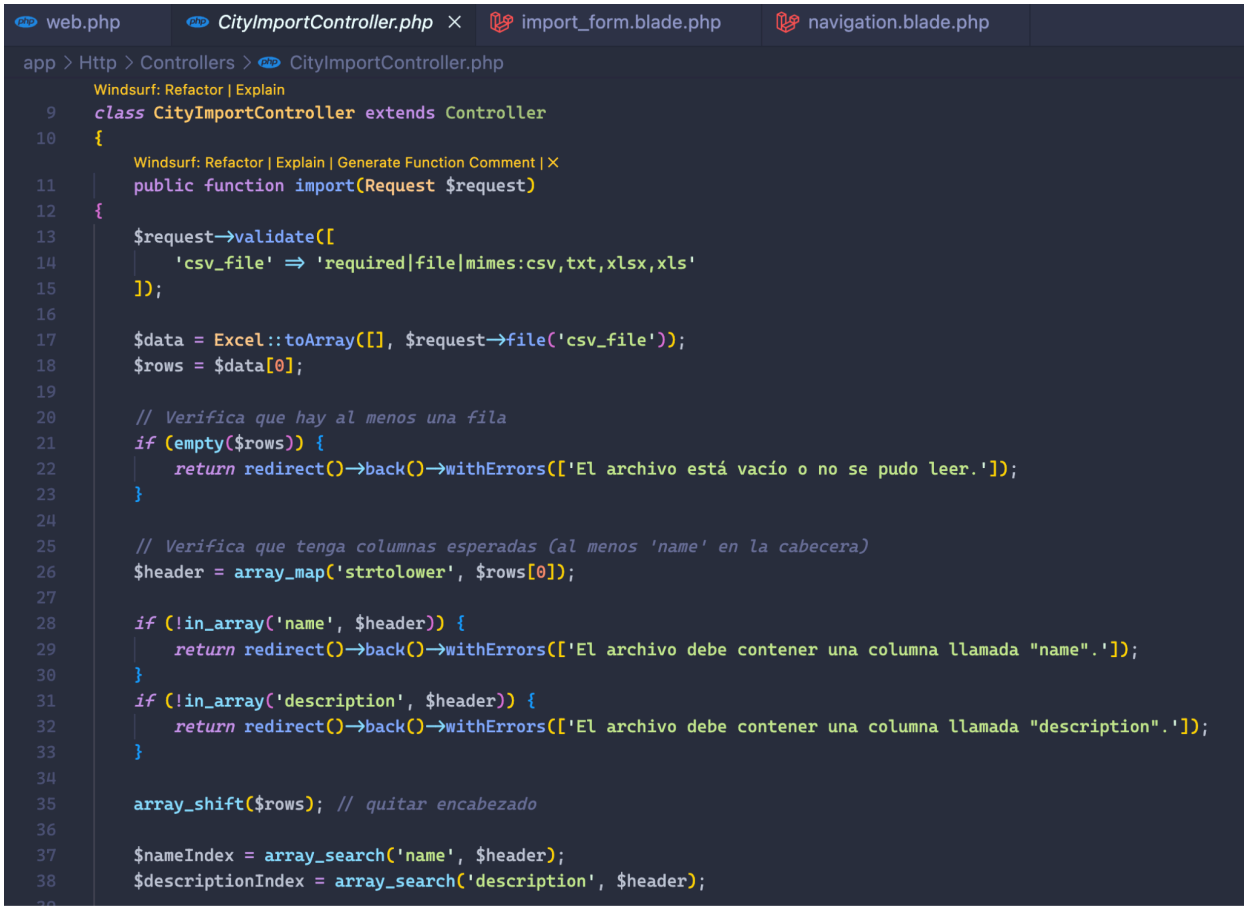
- La ruta GET /cities/import invoca al método cities.import del CityImportController, el cual retorna la vista con el formulario de carga (como el mostrado en la Figura 1). De esta manera, la interfaz para seleccionar archivos queda accesible desde la aplicación.

- La ruta POST /cities/import está asociada al método import del CityImportController. Esta ruta recibe la solicitud cuando el usuario envía el formulario con un archivo adjunto. Mediante el nombre de ruta cities.import, referenciado en el atributo action del formulario Blade, Laravel sabrá a dónde dirigir la petición.

Ambas rutas se integran al archivo de rutas del proyecto, asegurando que solo usuarios autenticados y autorizados (según la configuración general del proyecto) puedan acceder a ellas, en concordancia con las demás secciones de administración de la aplicación.

Método de controlador para la importación de datos

En el controlador de ciudades (CityImportController), se implementó el método import que maneja la lógica central: validar el archivo subido, procesarlo usando la clase de importación antes vista, y retornar la respuesta adecuada al usuario. A continuación se muestra el código simplificado de este método, con comentarios explicativos:



Explicación del método: En la sección **1**, se utiliza $request->validate() para asegurarse de que el usuario haya adjuntado un archivo (required) y que el tipo de archivo sea válido. La regla mimes:csv,xlsx,xls restringe la extensión a CSV o Excel (formato Office 97-2003 .xls o Office Open XML .xlsx). Se proveen mensajes de error personalizados en español para mejorar la experiencia de usuario en caso de incumplir estas reglas.

En el paso **2**, tras una validación exitosa, se invoca Excel::import(new CitiesImport, $request->file('archivo')). Aquí empleamos el *facade* Excel (correctamente importado con use Maatwebsite\Excel\Facades\Excel) para llamar a la función de importación, pasando una instancia de nuestra clase CitiesImport y el archivo subido obtenido via $request->file('archivo'). Laravel Excel se encarga de abrir el archivo, leerlo fila por fila y por cada una ejecutar nuestro método CitiesImport::model(), realizando la inserción de los datos en la base de datos. Cabe destacar que el método import es síncrono; para archivos muy grandes podría evaluarse el uso de **colas de trabajo (jobs en background)**, pero para volúmenes moderados de datos esta llamada directa es suficiente y garantiza la finalización de la importación antes de responder al usuario.

Una vez completada la importación sin errores, en el paso **3** se retorna una **redirección** (probablemente de regreso a la vista anterior, por ejemplo la lista de ciudades o el mismo formulario) con un mensaje de éxito almacenado en la sesión (with('success', '...')). Este

mensaje podrá mostrarse al usuario en la interfaz para confirmar que la operación fue exitosa. Por otro lado, si ocurre alguna excepción durante el proceso (por ejemplo, si el contenido del archivo tiene un formato inesperado, si hay problemas de codificación, o cualquier otro error en la lectura/inserción), se entra al bloque **4 catch**, donde se registra el error detallado en el log del sistema (Log::error(...)) para fines de depuración, y se redirige de vuelta al usuario con un mensaje de error genérico (with('error', '...')). De este modo, el usuario es notificado de que la importación falló y se le sugiere verificar el archivo, mientras que los desarrolladores pueden revisar los logs para diagnosticar la causa exacta.

Verificación de la estructura del archivo

Antes de realizar la importación, es fundamental **verificar que el archivo tenga la estructura esperada**, es decir, que incluya las columnas o campos requeridos (en este caso name y description). En nuestra implementación, esta verificación se realiza de forma implícita al usar la clase CitiesImport con WithHeadingRow: el código asume la existencia de cabeceras con esos nombres. No obstante, para robustecer la solución, se consideró la validación explícita de la estructura. Una posible mejora implementada fue leer la primera fila (fila de encabezados) del archivo antes de importar y comprobar que contenga exactamente las columnas necesarias.

Por ejemplo, se podría utilizar el mismo paquete Laravel Excel para obtener las cabeceras: una técnica es realizar una importación utilizando una clase especial o método que solo lea los encabezados. Laravel Excel ofrece funcionalidades para ello; por simplicidad, también es viable manejarlo manualmente leyendo el archivo CSV en bruto. Si el archivo no contiene las cabeceras name y description (o están mal escritas), el sistema **abortaría la importación** y retornaría un mensaje de error indicando que el formato del archivo es incorrecto. Esta verificación previa evita inserciones incompletas o mal mapeadas. En el presente informe, asumimos que el archivo proviene de una fuente confiable con el formato correcto, pero se deja sentada la recomendación de realizar esta comprobación adicional para un sistema en producción.

Manejo de errores en la importación

El proceso de importación incluye varias capas de manejo de errores para garantizar la integridad de los datos y una buena experiencia de usuario. En primera instancia, la **validación inicial** del archivo (tipo y presencia) previene que se procese algo que seguramente fallaría (por ejemplo, un PDF en lugar de un CSV). Durante la lectura e inserción, Laravel Excel y la base de datos pueden generar excepciones si encuentran datos no válidos (por ejemplo, tipos de datos incorrectos o violaciones de restricciones en la tabla). Todos estos errores se capturan en el bloque try-catch del controlador.

Como ya se describió, ante un fallo se registra el error en el log del servidor para diagnóstico y se notifica al usuario con un mensaje amigable. Este mensaje no expone detalles técnicos (por seguridad y usabilidad), sino que simplemente informa que ocurrió un problema y sugiere verificar el archivo. De esta forma, **no se pierde información sobre el error** (queda en los logs

para el desarrollador) y **el usuario recibe feedback inmediato**. Adicionalmente, la importación con Laravel Excel es atómica en cierto sentido: si ocurriera una excepción a mitad del recorrido, el proceso se detiene y las inserciones que alcanzaron a realizarse permanecerán. Si se deseara una atomicidad completa (todo o nada), podría envolverse la operación en una transacción de base de datos, pero dado el contexto académico, este nivel no fue requerido explícitamente.

Inserción de datos en la base de datos (tabla cities)

Gracias a la combinación de Eloquent y Laravel Excel, la inserción de datos resultó simplificada. Cada fila válida del archivo produce una instancia nueva del modelo City, la cual se guarda automáticamente en la base de datos al ser retornada por CitiesImport:Controllador:model(). El modelo City (preexistente en el proyecto) cuenta con fillables para name y description, permitiendo la asignación masiva de estos atributos. Asimismo, cualquier lógica de eventos Eloquent (como observers o eventos *creating/created*) configurada en el modelo City se ejecutará normalmente durante esta importación, lo que significa que la integración es transparente: insertar vía importación o vía formulario manual individual desencadena los mismos mecanismos internos.

Es importante mencionar que el **rendimiento** de la inserción fue considerado: Laravel Excel por defecto inserta fila por fila, lo cual es adecuado para cientos o unos pocos miles de registros. Si se anticipa la carga de decenas de miles de filas, el paquete ofrece técnicas como **chunk reading** (leer en porciones) o importar en lotes a través de ToCollection para optimizar el proceso. En nuestras pruebas con archivos de ejemplo de algunas centenas de ciudades, el tiempo de procesamiento fue prácticamente instantáneo (unos pocos segundos), demostrando la eficiencia de la solución implementada.

Resultados Obtenidos

Tras implementar y probar la funcionalidad, se logró **importar exitosamente los datos de ciudades** desde archivos externos a la aplicación. Los resultados concretos incluyen:

- **Carga exitosa de registros**: Archivos de prueba en formato CSV y XLSX con datos de ciudades fueron procesados correctamente. Los registros aparecieron en la tabla cities de la base de datos con sus campos name y description poblados según el contenido de los archivos. Por ejemplo, al importar un archivo de 50 ciudades, esas 50 nuevas filas quedaron disponibles inmediatamente en las vistas de la aplicación (ej. listado de ciudades en el panel administrativo).

- **Mensajes de confirmación**: El sistema mostró mensajes de éxito al usuario una vez completada cada importación, indicando que los datos fueron importados sin problemas. Asimismo, se verificó que ante condiciones de error (como subir un archivo vacío o con formato incorrecto), se presentaran los mensajes de error correspondientes sin impactar

la consistencia de la base de datos.

- **Integración transparente**: La funcionalidad se integró de manera consistente en la aplicación Laravel existente. El formulario de importación utiliza la misma estética y componentes que el resto del sistema, y la experiencia de usuario es fluida. El administrador puede ahora agregar nuevas ciudades en bloque, luego continuar utilizando las pantallas habituales para ver o editar cada ciudad, como si hubieran sido ingresadas manualmente.

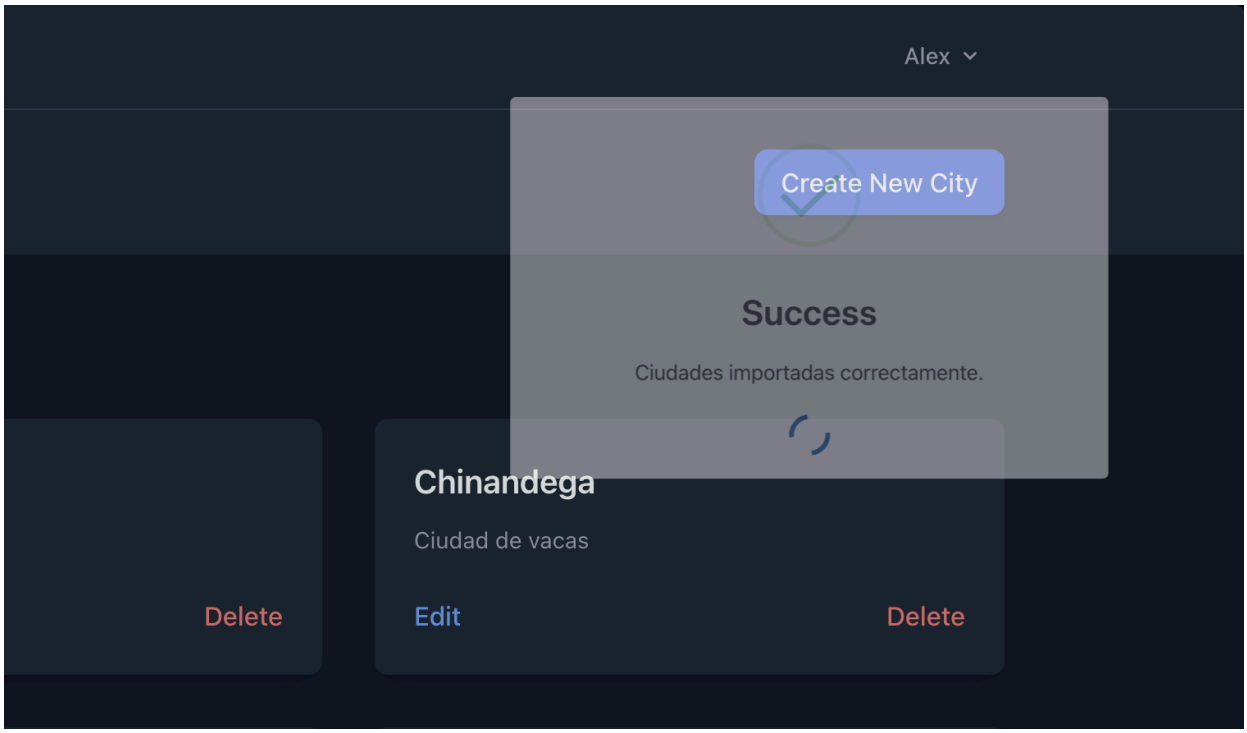


Figura 2. Mensaje de éxito mostrado tras una importación exitosa de datos. La interfaz notifica al usuario con el texto “Data imported successfully.” (datos importados correctamente) u otro mensaje configurable, confirmando que el archivo fue procesado y que los registros del mismo se almacenaron en la base de datos. Esta retroalimentación inmediata da confianza al usuario de que la operación culminó satisfactoriamente.

Como evidencia adicional, al consultar la base de datos después de una importación, se observa que la tabla cities contiene los nuevos registros con sus respectivos campos. Esto cumple con el propósito inicial de poblar la base de manera automática a partir de archivos externos, ahorrando al equipo horas de trabajo manual.

Conclusiones

La incorporación de la funcionalidad de **carga masiva de ciudades desde archivos CSV/XLSX** en el proyecto Laravel fue exitosa y aporta un **gran valor en términos de automatización y eficiencia**. A través de Laravel Excel y las capacidades nativas del framework, se logró una solución elegante que reduce la probabilidad de errores (al evitar la introducción manual de datos) y acelera el proceso de actualización de la información. Este tipo de automatización es especialmente valioso en entornos académicos y empresariales donde se manejan catálogos extensos de datos.

En resumen, los estudiantes desarrolladores lograron integrar la característica manteniendo buenas prácticas: se validan entradas, se maneja la lógica en controladores y clases dedicadas, y se brinda retroalimentación al usuario. El código escrito es **mantenible y reutilizable**, permitiendo que en el futuro se puedan importar datos de otras entidades de forma similar con cambios mínimos (por ejemplo, importación de países, regiones u otras tablas, creando clases import específicas para cada caso).

Posibles mejoras y trabajos futuros: Si bien la funcionalidad cumple con los objetivos planteados, siempre existen oportunidades de mejora. Algunas recomendaciones para futuras iteraciones son:

- **Validación avanzada del contenido**: incluir verificación de reglas de negocio en los datos importados (por ej., que el nombre de ciudad no esté vacío, o evitar duplicados si la tabla cities lo requiere). Incluso se podría generar un reporte de filas omitidas o con errores para que el usuario las revise.

- **Retroalimentación más detallada**: actualmente se notifica éxito o fallo general. Sería útil informar cuántos registros fueron importados, cuántos rechazados, o presentar un resumen de la operación al usuario (ej.: “45 ciudades importadas correctamente, 5 omitidas por datos inválidos”).

- **Soporte de archivos adicionales o formatos alternativos**: ampliar la compatibilidad a otros tipos de archivo (por ejemplo, **ODS** de LibreOffice) o diferentes delimitadores en CSV. También considerar la posibilidad de exportar una plantilla pre-formateada para el usuario, de modo que pueda descargar un Excel vacío con los encabezados correctos, rellenarlo y reimportarlo sin riesgo de estructura incorrecta.

- **Optimización para grandes volúmenes**: en escenarios con archivos muy pesados (miles de filas), podría implementarse la importación asíncrona mediante jobs en cola, mostrando un progreso al usuario. Laravel Excel soporta procesamiento por lotes y se integraría bien con esta idea, asegurando que la aplicación siga respondiendo mientras la importación ocurre en segundo plano.

- **Mejoras de UI/UX**: por último, pulir la interfaz permitiendo, por ejemplo, arrastrar y soltar el archivo (*drag & drop*), o integrando la importación masiva dentro del flujo de alta de ciudades (un botón “Importar desde Excel” en la pantalla de listado, por ejemplo).

En conclusión, la implementación realizada satisface plenamente los requisitos del proyecto académico, demostrando la capacidad de extender un proyecto Laravel existente mediante paquetes especializados. La carga masiva de datos de ciudades es ahora una tarea sencilla y rápida, lo que refuerza la eficiencia del sistema y deja sentadas bases sólidas para manejar datos en escala en futuros desarrollos. Los estudiantes han documentado el proceso y resultado en este informe, el cual acompaña al código fuente entregado en el repositorio (GitHub) del proyecto, asegurando la transferibilidad del conocimiento y facilitando el mantenimiento o mejora de la funcionalidad por parte de otros desarrolladores.