Manual de Funcionamiento: Módulo de Consulta Sisbén

1. Introducción

Este documento describe el funcionamiento, la arquitectura y las mejores prácticas para el módulo de consulta de datos del Sisbén. El objetivo de la aplicación es permitir a los usuarios buscar información detallada sobre los hogares y sus integrantes a partir de un número de documento.

La aplicación está diseñada para ser rápida, segura y escalable, capaz de manejar grandes volúmenes de datos.

2. Funcionamiento de la Consulta

La interfaz principal presenta un campo de búsqueda donde el usuario puede realizar consultas.

Pasos para Realizar una Consulta:

1. **Ingresar Documento**: El usuario introduce el número de documento de la persona que desea consultar en el campo de búsqueda.
2. **Presionar "Consultar"**: Al hacer clic en el botón "Consultar" o presionar Enter, la aplicación procesa la solicitud.
3. **Visualización de Resultados**:
    - **Resumen Global**: En la parte superior, se muestra un resumen con la fecha del último corte de datos y el número total de hogares registrados en el sistema.
    - **Resultados por Hogar**: Si se encuentra la persona, el sistema mostrará una o más "tarjetas", una por cada hogar al que pertenece la persona.
    - **Detalle de cada Hogar**: Cada tarjeta de hogar está dividida en tres secciones:
        - **Información del Hogar (Verde)**: Muestra datos generales como la dirección, comuna, barrio y el teléfono de contacto.
        - **Titular del Hogar (Morado)**: Muestra el nombre y documento del jefe de hogar asignado.
        - **Integrantes del Hogar**: Una tabla detallada con todos los miembros del hogar, incluyendo su ID, nombre, documento, teléfono y clasificación del Sisbén. La persona consultada aparece resaltada en la tabla para una fácil identificación.
    - **Sin Resultados**: Si el número de documento no se encuentra en la base de datos, se mostrará un mensaje indicándolo.

3. Componentes Técnicos

El módulo sigue la arquitectura estándar de Laravel (MVC - Modelo, Vista, Controlador).

- **Ruta (** `routes/web.php` **):**
  Define las URLs de la aplicación. La ruta principal es `GET /sisben/consulta/{cedula?}`, que dirige las solicitudes al `SisbenController`. El parámetro `{cedula?}` es opcional para permitir tanto la carga inicial de la página como las búsquedas directas por URL.

- **Controlador**
  (`app/Http/Controllers/SisbenController.php` **):**
  Es el cerebro de la aplicación. El método `search()` se encarga de:

  1. Validar que el número de documento recibido sea seguro y tenga un formato válido.
  2. Buscar en la base de datos todos los hogares asociados a ese documento.
  3. Para cada hogar, agrupar la información: datos generales, titular y lista de miembros.
  4. Calcular el total de hogares en el sistema (contando los jefes de hogar).
  5. Enviar todos los datos procesados a la vista para que los muestre.

- **Modelo (** `app/Models/SisbenData.php` **):**
  Representa la tabla `t_hogares_sisben` de la base de datos. Define la conexión a la tabla, los campos que se pueden llenar (`$fillable`) y cómo se deben tratar ciertos tipos de datos (ej: fechas). Actúa como un puente seguro y eficiente entre el controlador y la base de datos, protegiendo contra inyecciones

SQL gracias al ORM Eloquent.

- **Vista (** `resources/views/sisben/consulta.blade.php` **)**:
  Es la interfaz de usuario. Escrita con Blade, el motor de plantillas de Laravel, y estilizada con Tailwind CSS. Se encarga de renderizar dinámicamente la información que recibe del controlador, mostrando el formulario, los resúmenes y las tarjetas de resultados.

4. Estructura de la Base de Datos

La eficiencia de la consulta depende de una estructura de base de datos bien diseñada. La tabla principal es `t_hogares_sisben`.

```
CREATE TABLE `sisben_consulta` (
  -- Llave primaria: Un identificador único para cada re
  `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,

  -- Columnas de la tabla, limpias y con los tipos de da
  `ide_ficha_origen` VARCHAR(100) DEFAULT NULL,
  `cod_barrio` INT DEFAULT NULL,
  `cod_comuna` INT DEFAULT NULL,
  `dir_vivienda` VARCHAR(512) DEFAULT NULL,
  `fec_corte` DATETIME DEFAULT NULL,
  `num_personas_hogar` TINYINT UNSIGNED DEFAULT NULL,
  `identificacion_tentativa_num_hogares` BIGINT NOT NULL
  `num_tel_contacto` BIGINT DEFAULT NULL,
  `asignacion_jefe_hogar` VARCHAR(2) DEFAULT NULL COMMEN
  `ide_persona` BIGINT NOT NULL,
  `nombre_persona_hogar_concatenado` VARCHAR(512) DEFAUL
  `num_documento` VARCHAR(25) NOT NULL,
  `cod_calificacion` VARCHAR(20) DEFAULT NULL,
  `clasificacion` VARCHAR(50) DEFAULT NULL,

  -- ---- DEFINICIÓN DE LLAVES E ÍNDICES ----
  PRIMARY KEY (`id`),
  -- Regla de Integridad: Impide duplicados de persona p
  UNIQUE KEY `uq_hogar_persona` (`identificacion_tentati
  -- Índice para Búsqueda Rápida por Documento (La consu
  INDEX `idx_num_documento` (`num_documento`),
  -- Índice para Búsqueda Rápida de Integrantes de un Ho
  INDEX `idx_hogar` (`identificacion_tentativa_num_hogar
  -- Índice para calcular rápidamente el total de hogare
  INDEX `idx_jefe_hogar` (`asignacion_jefe_hogar`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_
```

Importancia de los Índices: Los índices (`INDEX`) son cruciales. Son como el índice de un libro: permiten a la base de datos encontrar información muy rápidamente sin tener que leer la tabla entera. El índice `idx_num_documento` es el que hace que la consulta principal por cédula sea casi instantánea, incluso con

millones de registros.

5. Estrategia de Carga de Datos Masivos

Para una carga inicial de **más de un millón de registros**, la estrategia de importación es fundamental para la eficiencia y la estabilidad del sistema.

¿Por qué NO es viable crear un módulo de carga en Laravel para esta escala?

Crear un módulo en Laravel (por ejemplo, un comando de Artisan que lea un archivo CSV y cree registros uno por uno) es **extremadamente ineficiente** para grandes volúmenes de datos por las siguientes razones:

1. **Lentitud Extrema**: Por cada registro, Laravel ejecuta una sobrecarga de operaciones (instanciar el modelo, ejecutar eventos, construir y enviar la consulta SQL). Para un millón de registros, este proceso podría tardar **horas**, en comparación con los **minutos o segundos** de una carga directa.
2. **Alto Consumo de Recursos**: El proceso consumiría una cantidad masiva de memoria RAM y tiempo de CPU del servidor de PHP, llevando fácilmente a errores de `memory_limit` o `max_execution_time`.
3. **Complejidad Innecesaria**: Para hacerlo funcionar de forma robusta, se requerirían implementaciones complejas como colas (queues) y trabajos en segundo plano (background jobs), lo cual es excesivo para una tarea de carga de datos.

La Mejor Opción: Carga Directa en la Base de Datos

La forma más profesional, rápida y eficiente es utilizar las herramientas nativas del motor de la base de datos.

Ventajas:

- **Velocidad**: La operación se realiza a bajo nivel, directamente en el motor de la base de datos, que está optimizado para estas tareas.
- **Eficiencia**: El consumo de recursos del servidor de aplicación (PHP) es prácticamente nulo.
- **Atomicidad**: La carga puede realizarse como una única transacción, garantizando la integridad de los datos (o se carga todo, o no se carga nada).

Ejemplo de comando en MySQL/MariaDB:

```
LOAD DATA INFILE '/ruta/al/archivo/datos_sisben.csv'
INTO TABLE sisben_consulta
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
(ide_ficha_origen, cod_barrio, ...); -- Listar todas las
```

¿Cuándo SÍ es viable crear un módulo de carga en Laravel?

Un módulo de carga dentro de la aplicación es la opción correcta en los siguientes escenarios:

1. **Volúmenes de Datos Pequeños a Medianos**: Es ideal para cargas de **hasta unos pocos miles de registros (ej. 1 a**

10,000). Con un buen diseño (usando "chunks" o trozos), podría soportar hasta 50,000 registros, pero el tiempo de procesamiento ya sería considerable. Para más de eso, se recomienda usar colas.

2. **Cargas Realizadas por Usuarios Finales**: Cuando un administrador necesita subir un archivo Excel o CSV con nuevos datos a través de una interfaz web.
3. **Lógica de Negocio Compleja por Fila**: Si cada registro necesita pasar por validaciones complejas de Laravel, relacionarse con otros modelos, o disparar eventos específicos de la aplicación antes de ser guardado.

En resumen, para la carga masiva inicial, la herramienta correcta es la que ofrece la base de datos. Para cargas incrementales, más pequeñas y que requieren la lógica de la aplicación, un módulo en Laravel es la solución perfecta.