Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем і комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(рівень вищої освіти)

на тему: «Розроблення сервісу для оброблення "Big data" файлів»

Виконав студент 2 курсу, групи ІСТС-21
спеціальності: 126
„Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Саламаха Є.І.
(прізвище, ініціали)

Керівник: доц. Олянишин Т.В.
(прізвище, ініціали)

Рецензент: Крошний І.М.
(прізвище, ініціали)

Львів-2023

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ деревооброблювальних та комп'ютерних технологій і дизайну

Кафедра Інформаційних систем і комп'ютерного моделювання

Рівень вищої освіти перший (бакаравський)

Спеціальність 126 *„Інформаційні системи та технології"*

ЗАТВЕРДЖУЮ:

В.о. завідувача кафедри ІСКМ

_____ Сторожук О.Л.

„24" 11 2022.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Саламаха Євгеній Ігорович

(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: Розроблення сервісів для оброблення "Big data" файлів.

керівник роботи доц. Олянишин Т.В.,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "21" 11.2022 року, №С-521

2.Термін подання студентом проекту(роботи) 10 червня 2023р

3. Вихідні дані до проекту (роботи) Розробити програмне забезпечення для обробки файлів з великою кількостю даних, а саме формату «CSV». Веб-інформаційна система крім того повинна вміти як імпортувати так і експортувати дані в файлма. Також потрібно розробити адаптивний інтерфейс та адаптивну адмін частину для екранів різного типу та діагоналей.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Стан проблемної області

Інформаційне забезпечення

Програмне забезпечення

Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалу до доповіді .

6. Консультанти розділів проекту (роботи)

7. Дата видачі завдання 23.11.2022р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	23.11-20.12	Викон.
2.	Постановка задачі і її формалізація	20.12-13.01	Викон.
3.	Виконання вхідного етапу технології	13.01-14.04	Викон.
4.	Реалізація головних класів проекту	14.04-20.05	Викон.
5.	Виконання етапу відлагодження проєкту	20.05.-25.05.	Викон.
6.	Виконання етапу впровадження та випуску бета-версії.	25.05.-02.06.	Викон.
7.	Оформлення записки до дипломного проекту.	02.06.-10.06.	Викон.

Студент _____ Саламаха Є.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ доц. Олянишин Т.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить *29* сторінок пояснювальної записки, *19* рисунків, *1* додаток, *15* джерел.

Дипломна робота присвячена розробці системи для роботи з великою кількостю даних. Для демонстрації можливостей програмного забезпечення використано CMS October та фреймовр Laravel. Розроблено гнучкий механізм для завантаження та вивантаження великої бази даних за допомогою файлів типу CSV. Створено адаптивниї інтерфейс в стилі e-commerce в автомобільній індустрії.

Ключові слова: CMS October, Laravel, програмне забезпечення.

ABSTRACT

Thesis contains *29* pages of explanatory note, *19* drawings, *1* appendix, *15* sources.

The thesis is devoted to the development of a system for working with a large amount of data. CMS October and the Laravel framework were used to demonstrate the capabilities of the software. A flexible mechanism has been developed for loading and unloading a large database using CSV files. An adaptive interface in the style of e-commerce in the automotive industry has been created.

Keywords: CMS October , Laravel, software.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити програмне забезпечення для імпорту та експорту великої кількості даних. Для реалізації проєкту використати фреймворк Laravel. Створити адаптивну веб-орієнтовану систему для демонстрації роботи структури завантаження та вивантаження даних.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ACID – набір властивостей транзакцій баз даних (Atomicity, Consistency, Isolation, Durability), що забезпечують надійність обробки даних.

API – Application Programming Interface, інтерфейс програмування застосунків для взаємодії між програмними компонентами.

CSV – Comma-Separated Values, текстовий формат зберігання табличних даних, де значення розділені комами.

JSON – JavaScript Object Notation, текстовий формат обміну даними, що підтримує ієрархічну структуру.

Avro – бінарний формат серіалізації даних, що підтримує схеми та еволюцію структури даних.

ORC – Optimized Row Columnar, колонковий формат зберігання даних, оптимізований для аналітичних запитів.

OLAP – Online Analytical Processing, технологія аналітичної обробки великих масивів даних.

ETL – Extract, Transform, Load, процес вилучення, трансформації та завантаження даних у сховище.

SQL – Structured Query Language, мова структурованих запитів для роботи з реляційними базами даних.

JIT – Just-In-Time compilation, технологія компіляції коду під час виконання програми.

CMS – Content Management System, система керування контентом веб-сайтів.

POC – Proof of Concept, прототип або демонстрація працездатності ідеї.

ВСТУП

Розширення пропозиції продуктів є одним із ключових способів розвитку вашого бізнесу електронної комерції. Але чим ширший ваш вибір продуктів, тим складнішим може стати оновлення вашого магазину – внесення всіх змін на кожній сторінці продукту може стати надзвичайно трудомістким і марною тратою ресурсів.

Зіставлення інформації про клієнтів може бути не менш складним завданням, особливо якщо ви вручну копіюєте та вставляєте інформацію для збору даних про минулі замовлення та облікові записи користувачів.

Отже, що, якби ми сказали вам, що є швидший і простіший спосіб масового завантаження сотень продуктів і завантаження всієї інформації про покупки клієнтів одним клацанням миші? Відповідь: використовуйте файли CSV.

Об’єктом дослідження використання csv у створенні e-commerce проєкту

Метою роботи є використання Laravel у створенні структури імпорт-експорт csv файлу

Предметом дослідження є вивантаження та завантаження великої кількості даних

Практичне значення роботи полягає у можливості швидкого адміністрування веб-систем з великою кількістю даних

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

При використанні глибокого сховища вибір правильного формату файлу має вирішальне значення. Ці файлові системи або системи глибокого зберігання коштують дешевше, ніж бази даних, але просто забезпечують базове сховище і не забезпечують сильних гарантій ACID. Вам потрібно буде вибрати правильне сховище для вашого випадку використання, виходячи з ваших потреб та бюджету. Наприклад, ви можете використовувати базу даних для проковтування, якщо дозволяє бюджет, а потім, як тільки дані будуть перетворені, зберігати їх у своєму озері даних для аналізу OLAP. Або ви можете зберігати все в глибокому сховищі, крім невеликої підмножини гарячих даних, у швидкій системі зберігання, такій як реляційна база даних.

1.2. Формати файлів

Зауважте, що системи глибокого зберігання зберігають дані у вигляді файлів, а різні формати файлів та алгоритми стиснення надають переваги для певних випадків використання. Спосіб зберігання даних в озері даних має вирішальне значення, і вам потрібно враховувати формат, стиснення та особливо спосіб розділення даних. Найбільш поширеними форматами є CSV, JSON, AVRO, Protocol Buffers, Parquet і ORC.

	Avro	Parquet	ORC
Schema Evolution	Best	Good	Better
Compression	Good	Better	Best
Splitability	Good	Good	Best
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

Рис. 1.1. Параметри формату файлів

Деякі речі, які слід враховувати при виборі формату:

- **Структура ваших даних**: деякі формати приймають вкладені дані, такі як JSON, Avro або Parquet, а інші ні. Навіть ті, які це роблять, можуть бути не дуже оптимізовані для цього. Avro є найбільш ефективним форматом для вкладених даних, я рекомендую не використовувати Паркетні вкладені типи, оскільки вони дуже неефективні. Процес вкладеного JSON також дуже інтенсивний процесор. Загалом, рекомендується сплющувати дані при їх прийомі.

- **Продуктивність**: Деякі формати, такі як Avro і Parquet, працюють краще, ніж інші подібні JSON. Навіть між Avro і Parquet для різних варіантів використання один буде краще за інших. Наприклад, оскільки Parquet є форматом на основі стовпців, чудово запитувати ваше озеро даних за допомогою SQL, тоді як Avro краще підходить для трансформації рядків ETL.

- **Легко читати**: Подумайте, чи потрібні вам люди, щоб читати дані чи ні. JSON або CSV є текстовими форматами і читаються людиною, тоді як більш продуктивні формати, такі як паркет або Avro, є двійковими.

- Стиснення: деякі формати пропонують вищу швидкість **стиснення**, ніж інші.

- **Еволюція схеми**: Додавання або видалення полів набагато складніше в озері даних, ніж в базі даних. Деякі формати, такі як Avro або Parquet, забезпечують певний ступінь еволюції схеми, що дозволяє змінювати схему даних і все ще запитувати дані. Такі інструменти, як формат **Delta Lake**, надають ще кращі інструменти для боротьби зі змінами в схемах.

- **Сумісність**: JSON або CSV широко поширені та сумісні майже з будь-яким інструментом, тоді як більш продуктивні варіанти мають менше точок інтеграції.

Формати файлів

- **CSV:** хороший варіант для сумісності, обробки електронних таблиць і даних, придатних для читання людиною. Дані повинні бути плоскими. Він неефективний і не може обробляти вкладені дані. Можуть виникнути проблеми з роздільником, які можуть призвести до проблем із якістю даних. Використовуйте цей формат для розвідувального аналізу, POC або невеликих наборів даних.

- **JSON**: Широко використовується в API. Вкладений формат. Він широко поширений і читабельний людиною, але його може бути важко прочитати, якщо є багато вкладених полів. Чудово підходить для невеликих наборів даних, цільових даних або інтеграції API. Якщо можливо, конвертуйте в більш ефективний формат перед обробкою великих обсягів даних.

- **Avro: Чудово** підходить для зберігання даних рядків, дуже ефективний. Він має схему і підтримує еволюцію. Велика інтеграція з Кафкою. Підтримує розділення файлів. Використовуйте його для операцій на рівні рядків або в Kafka. Відмінно записує дані, повільніше читається.

- **Буфери протоколу**: чудово підходить для API, особливо для **gRPC.** Підтримує схему і це дуже швидко. Використовуйте для API або машинного навчання.

- **Паркет**: Стовпчасте зберігання. Він має підтримку схеми. Він дуже добре працює з Hive і Spark як спосіб зберігання стовпчастих даних у

глибокому сховищі, яке запитується за допомогою SQL. Оскільки він зберігає дані у стовпцях, обробники запитів читатимуть лише файли, які містять вибрані стовпці, а не весь набір даних, на відміну від Avro. Використовуйте його як рівень звітності.

- **ORC:** Подібно до паркету, він забезпечує краще стиснення. Він також забезпечує кращу підтримку еволюції схеми, але він менш популярний.

1.3. Стиснення файлів

Нарешті, вам також потрібно подумати, як **стиснути дані**, враховуючи компроміс між розміром файлу та вартістю процесора. Деякі алгоритми стиснення швидші, але з більшим розміром файлу, а інші повільніші, але з кращими показниками стиснення.

Format	Algorithm	Strategy	Emphasis	Comments
zlib	Uses DEFLATE (LZ77 and Huffman coding)	Dictionary-based, API	Compression ratio	Default codec
gzip	Wrapper around zlib	Dictionary-based, standard compression utility	Same as zlib, codec operates on and produces standard gzip files	For data interchange on and off Hadoop
bzip2	Burrows-Wheeler transform	Transform-based, block-oriented	Higher compression ratios than zlib	Common for Pig
LZO	Variant of LZ77	Dictionary-based, block-oriented, API	High compression speeds	Common for intermediate compression, HBase tables
LZ4	Simplified variant of LZ77	Fast scan, API	Very high compression speeds	Available in newer Hadoop distributions
Snappy	LZ77	Block-oriented, API	Very high compression speeds	Came out of Google, previously known as Zippy

Рис. 1.2. Стиснення файлів

Як ми бачимо, CSV та JSON прості у використанні, доступні для читання людиною та поширені формати, але їм не вистачає багатьох можливостей інших форматів, що робить їх занадто повільними для використання для запиту обробки великих даних. ORC і Parquet широко використовуються в екосистемі Hadoop для запиту даних, тоді як Avro також використовується за межами Hadoop, особливо разом з Kafka для прийому, він дуже хороший для обробки ETL на рівні рядків. Формати, орієнтовані на рядки, мають кращі можливості еволюції схеми, ніж формати, орієнтовані на стовпці, що робить їх чудовим варіантом для прийому даних.

Отже можна зробити висновок, що кожен з цих форматів має свої плюси та мінуси. І ключовим у розробці ПЗ є технічне завдання, яке може заставити програміста вибрати якийсь конкретний формат.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Laravel 9

Laravel вже багато років домінує в ландшафті веб-фреймворку PHP. Якщо ви створюєте власні веб-сайти PHP, швидше за все, ви використовували фреймворк один або два рази і знаєте, що оновлення Laravel виходить кожні шість місяців. Laravel v9 випущений 8 лютого 2022 року, і він пропонує безліч нових функцій.

У той час як минулі випуски Laravel відбувалися кожні шість місяців, новий цикл випуску буде здійснюватися кожні 12 місяців, що забезпечує синхронізацію випуску з Symfony — який Laravel використовує за лаштунками — і дає команді розробників більше часу для виправлення будь-яких помилок, що виникають при взаємодії з фреймворком Symfony.

Laravel використовує Symfony 6, для чого потрібно як мінімум PHP 8. PHP 8 поставляється з новим компілятором точно вчасно (JIT), розширенням OPcache, іменованими аргументами тощо.

Swift Mailer, який роками використовувався в Laravel, видаляється і більше не буде підтримуватися. У Laravel v9 та майбутніх випусках вам доведеться використовувати Symfony Mailer. Якщо ви оновлюєте наявну інсталяцію Laravel, ознайомтеся з посібником з оновлення.

У Laravel 9 тепер ви можете використовувати для оголошення префікса моделі з одним непрефіксним терміном. Використовуючи один виклик методу, тепер можна як отримати, так і встановити атрибути.

```php
use Illuminate\Database\Eloquent\Casts\Attribute;

public function username(): Attribute
{
  return new Attribute(
    get: fn ($value) => strtoupper($value),
    set: fn ($value) => $value,
  );
}
```

Рис. 2.1. Отримати, або встановити атрибути

Якщо ви використовуєте MySQL або PostgreSQL у своїй програмі Laravel, тепер ви можете використовувати метод визначення стовпців у ваших файлах міграції для створення повнотекстових індексів.fulltext.

Потім ви можете використовувати методи and, щоб додати до своїх запитів повнотекстові речення де *whereFullText orWhereFullText.*

Laravel v9 поставляється з новим движком бази даних Laravel Scout. Він надає можливості повнотекстового пошуку для моделей Eloquent. Він використовує модельні спостерігачі для синхронізації пошукових індексів із записами Eloquent і є хорошим вибором для програм, які використовують базу даних малого або середнього розміру або мають невелике навантаження. Цей механізм використовуватиме речення "де-як" під час фільтрації результатів із вашої бази даних. Щоб використовувати його, просто додайте рису до моделі: Laravel\Scout\Searchable.

Порівняння з конкурентами

WordPress з моменту свого створення прагнув бути надзвичайно зручним для користувачів і зробив це успішно, усуваючи тертя як для технічних, так і для

нетехнічних користувачів, а також для людей будь-якого походження - незалежно від їх освіти та економічного рівня. Засновник WordPress Метт Малленвег (Matt Mullenweg) висловив, що девіз WordPress «Democratize Publishing» для нинішньої епохи.

WordPress простий у використанні для всіх, і про його інклюзивність свідчить і розробка: нерідко можна зустріти людей без досвіду програмування (наприклад, маркетологів, дизайнерів, блогерів, продавців та інших), які майструють зі своїми установками WordPress, розробляють власні теми та успішно запускають власні веб-сайти. WordPress орієнтований на користувача, і потреби користувачів перевищують потреби розробників. У WordPress користувач є королем (або королевою).

На відміну від цього, October CMS більше орієнтована на розробника, оскільки явність встановилася з самого першого випуску:

«Жовтень робить одне сміливе, але очевидне припущення: клієнти не створюють сайти, а розробники. Роль клієнта полягає в управлінні веб-сайтом і передачі своїх бізнес-вимог. Веб-розробник і сама галузь обертаються навколо опосередкування цих факторів».

За словами її засновників, місія CMS – «довести, що створення веб-сайтів – це не ракетобудування». Будучи заснованою на Laravel, October CMS може претендувати на міцну основу багаторазового, модульного коду, який може виробляти правильно спроектовані додатки, підтримувані в довгостроковій перспективі і повністю настроювані, не вимагаючи хакерських атак — тип, який приваблює серйозних програмістів. October CMS також може забезпечити відмінний користувальницький досвід, однак він не такий простий або безпроблемний, як той, що надається WordPress. Користувачам, можливо, доведеться пояснити, як використовувати певну функціональність, перш ніж вони зможуть її використовувати. Наприклад, вбудовування форми з якогось

плагіна має довге пояснення того, як це зробити, що є більш громіздким, ніж самоочевидна функція перетягування, що надається кількома плагінами форм у WordPress.

WordPress славиться своєю 5-хвилинною установкою, хоча багато людей відзначають, що (беручи до уваги всі плагіни, які повинні бути встановлені) типова установка вимагає 15 хвилин або більше. Крім того, WordPress також пропонує функцію Multisite, яка дозволяє нам створювати мережу з декількох віртуальних сайтів під однією установкою. Ця функція полегшує агентству адміністрування сайтів кількох клієнтів — серед інших випадків користувачів.

Установка October CMS також проходить дуже гладко: сама установка Wizard займає навіть менше п'яти хвилин, а якщо встановити її через консольну установку, то ще швидше. Ви можете зробити останнє, просто перейшовши до цільового каталогу, а потім виконавши (після чого нам потрібно ввести конфігурацію бази даних, інакше вона поводиться як плоскофайлова CMS). Після завершення встановлення у нас буде повністю функціонуючий веб-сайт, але все ще досить голий (якщо додати час, необхідний для встановлення та налаштування необхідних плагінів, ви можете очікувати, що це займе щонайменше 15 хвилин).curl -s https://octobercms.com/api/installer | php

WordPress звинувачують у небезпеці через велику кількість вразливостей, які постійно виявляються. Це змушує користувачів мати програмне забезпечення для CMS та всі встановлені плагіни завжди актуальними, щоб уникнути експлойтів безпеки. Серед основних проблем є підтримка WordPress старих версій PHP, які більше не підтримуються спільнотою розробників PHP (WordPress в даний час підтримує PHP 5.2.4, тоді як остання повністю підтримувана версія PHP - 5.6). Однак ця проблема повинна бути вирішена в квітні 2019 року, коли WordPress офіційно почне підтримувати PHP версії 5.6 і вище.

В іншому випадку, WordPress не обов'язково небезпечний сам по собі, а через свою високу популярність, що робить його головною мішенню для хакерів. Однак це грає в обох напрямках: повсюдність WordPress означає, що його команда безпеки повинна дійсно серйозно ставитися до своєї роботи, постійно шукаючи експлойти і виправляючи їх якомога швидше, інакше до третини Інтернету знаходиться під загрозою. Ставки просто занадто високі.

October CMS, з іншого боку, не має репутації невпевненої в собі. Однак, оскільки існує приблизно 27000 живих сайтів, які використовують October, порівняно з мільйонами WordPress, ми не можемо судити про них двох на однакових умовах. Тим не менш, команда, яка стоїть за October CMS, серйозно ставиться до безпеки, про що свідчить підказка інсталяції майстра ввести URL-адресу сервера CMS, встановлену за замовчуванням, але змінювану на будь-що інше, щоб ускладнити хакерам націлювання на сайт. На відміну від цього, зміна логіна та серверних URL-адрес WordPress з і відповідно на щось інше має бути зроблена за допомогою плагіна. Крім того, жовтнева CMS може функціонувати як плоскофайлова CMS (тобто без бази даних) і уникати вразливостей, пов'язаних з базою даних, таких як SQL-ін'єкція./backend/wp-login.php/wp-admin

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розробка програмного забезпечення.

Для розробки багатопотокового застосунку на Laravel 9 для опрацювання файлів з великою кількістю даних, необхідно використовувати пакети, що надають можливість працювати з багатопотоковістю. Одним з таких пакетів є "Symfony Process Component", який надає функціонал для запуску процесів у фоновому режимі та обробки результатів.

Для початку необхідно встановити пакет "symfony/process" за допомогою Composer. Для цього в командному рядку потрібно виконати наступну команду:

```
composer require symfony/process
```

Далі необхідно створити контролер, який буде відповідати за запуск та обробку процесів. У контролері необхідно визначити метод, який буде відповідати за запуск процесів. У цьому методі необхідно використовувати клас "Symfony\Component\Process\Process", який надає функціонал для запуску процесу.

```php
use Symfony\Component\Process\Process;

class CsvController extends Controller
{
    public function processCsv()
    {
        // Шлях до файлу з даними
        $filePath = storage_path('app/csv/data.csv');

        // Кількість потоків
        $threads = 4;

        // Створюємо масив з командами для запуску процесів
        $commands = [];
        for ($i = 0; $i < $threads; $i++) {
            $commands[] = sprintf('php %s process-csv --start=%d --end=%d',
            base_path('artisan'), $i * 1000, ($i + 1) * 1000);
        }

        // Запускаємо процеси в фоновому режимі
        $processes = [];
        foreach ($commands as $command) {
            $processes[] = new Process(explode(' ', $command));
        }
        Process::runProcesses($processes);

        // Обробляємо результати
        $result = [];
        foreach ($processes as $process) {
            $result[] = $process->getOutput();
        }

        return view('csv.result', compact('result'));
    }
}
```

Рис. 3.1. Метод CsvController

У методі "processCsv" визначається шлях до файлу з даними, кількість потоків, а також створюється масив з командами для запуску процесів. У даному прикладі запускається 4 процеси з командами, які виконують опрацювання файлу з даними по 1000 рядків кожен. Для запуску процесів використовується метод "Process::runProcesses", який запускає процеси в фоновому режимі. Після завершення роботи процесів, обробляються результати, які можуть бути використані для подальшої обробки.

Наступним кроком є створення команди для опрацювання файлу з даними. Для цього необхідно виконати наступну команду Artisan:

```
php artisan make:command ProcessCsv
```

Ця команда створить файл "ProcessCsv.php" у директорії "app/Console/Commands". У цьому файлі необхідно визначити метод "handle", який буде виконувати опрацювання файлу з даними.

```php
use Illuminate\Console\Command;

class ProcessCsv extends Command
{
    protected $signature = 'process-csv {--start=0} {--end=1000}';
    protected $description = 'Process CSV file';

    public function handle()
    {
        // Отримуємо параметри команди
        $start = $this->option('start');
        $end = $this->option('end');

        // Опрацьовуємо файл з даними
        $filePath = storage_path('app/csv/data.csv');
        $file = fopen($filePath, 'r');
        $lineNumber = 0;
        while (($line = fgetcsv($file)) !== false) {
            if ($lineNumber >= $start && $lineNumber < $end) {
                // Обробляємо рядок даних
            }
            $lineNumber++;
        }
        fclose($file);
    }
}
```

Рис. 3.2. Команда ProcessCsv.php

У методі "handle" отримуються параметри команди, які визначають інтервал рядків, який потрібно обробити. Далі відкривається файл з даними і за допомогою функції "fgetcsv" починається опрацювання рядків даних. В даному прикладі обробляються рядки даних від "start" до "end". Після завершення роботи з файлом, файл закривається.

Останнім кроком є виклик контролера з відповідним методом з веб-інтерфейсу. Для цього необхідно створити маршрут у файлі "routes/web.php".

```
Route::get('/process-csv', [CsvController::class, 'processCsv']);
```

Щоб забезпечити більш ефективну роботу з файлами, ми можемо використовувати пакет Laravel Excel. Цей пакет дозволяє працювати з різними форматами файлів, такими як CSV, XLSX та інші. Він також має можливості для оптимізації обробки даних, таких як імпорт та експорт великої кількості даних за допомогою більш ефективних методів роботи з пам'яттю.

Один з основних підходів до оптимізації роботи з великими файлами полягає в розділенні файлу на менші частини та обробці їх паралельно. Для цього ми можемо використовувати бібліотеку Parallel Processing, яка надає можливість запускати кілька потоків одночасно та обробляти різні частини файлу.

Тепер розглянемо код для роботи з бібліотекою Parallel Processing та пакетом Laravel Excel.

Встановлюємо бібліотеку Parallel Processing:

```
composer require krak/php-parallel-processes
```

Створюємо метод для обробки файлу з даними про автомобілі:

```php
use Krak\PhpParallelProcesses\{Process, ProcessManager};
use Maatwebsite\Excel\Facades\Excel;

public function processCarsFile($filePath)
{
    // Отримуємо кількість рядків у файлі
    $rowCount = Excel::toArray([], $filePath)[0]->count();

    // Розбиваємо файл на 4 частини
    $chunkSize = ceil($rowCount / 4);

    // Ініціалізуємо об'єкт ProcessManager
    $manager = new ProcessManager();

    // Запускаємо 4 процеси для обробки кожної частини файлу
    for ($i = 0; $i < 4; $i++) {
        $start = $i * $chunkSize;
        $end = ($i + 1) * $chunkSize;

    // Додаємо процес до менеджера процесів
    $manager->addProcess(function () use ($filePath, $start, $end) {

    // Отримуємо дані з частини файлу
    $data = Excel::toArray([], $filePath, '', true, false, [
        'range' => "{$start}:{$end}",
    ])[0];

    // Обробляємо дані
    foreach ($data as $row) {
        $car = new Car();
        $car->make = $row['make'];
        $car->model = $row['model'];
        $car->year = $row['year'];
        $car->save();
    }
    // Запускаємо процеси
    $manager->startProcesses();
    // Очікуємо завершення процесів
    $manager->waitProcesses();
}
```

Рис. 3.3. Метод для обробки файлу з даними про автомобілі

Наступним кроком є створення міграції для таблиці "cars". Міграція дозволить створити таблицю у базі даних, яка буде використовуватися для збереження даних про автомобілі.

Для створення міграції виконайте команду:

```
php artisan make:migration create_cars_table --create=cars
```

Після створення міграції, відкрийте створений файл міграції у директорії "database/migrations". У методі "up" ви можете визначити структуру таблиці "cars".

```
public function up()
{
    Schema::create('cars', function (Blueprint $table) {
        $table->id();
        $table->string('make');
        $table->string('model');
        $table->integer('year');
        $table->timestamps();
    });
}
```

Рис. 3.4. Визначення структури таблиці "cars"

Після того, як міграцію було створено та налаштовано, необхідно запустити команду для виконання міграції:

```
php artisan migrate
```

Ця команда автоматично виконає всі міграції, які були створені, і створить таблиці у базі даних.

На цьому етапі ми готові до опрацювання великої кількості даних про автомобілі. Ви можете завантажити файл з даними та запустити функцію "processCarsFile" для опрацювання файлу.

Створимо новий метод processCarsFile() у контролері CarsController. Цей метод буде отримувати файл з даними про автомобілі та обробляти його. Він повинен бути наступного вигляду:

```php
public function processCarsFile(Request $request)
{
    // отримуємо файл з запиту
    $file = $request->file('cars_file');

    // проводимо валідацію файлу
    $this->validate($request, [
        'cars_file' => 'required|mimes:csv,txt|max:2048',
    ]);

    // зчитуємо дані з файлу та обробляємо їх
    $carsData = $this->getCarsDataFromCSV($file);
    $result = $this->processCarsData($carsData);

    // повертаємо результат обробки даних
    return response()->json([
        'success' => true,
        'result' => $result,
    ]);
}
```

Рис. 3.5. Отримувати файл з даними про автомобілі та обробляти

Цей метод приймає об'єкт Request як параметр та отримує файл з даними про автомобілі з запиту. Наступним кроком є проведення валідації файлу, щоб переконатися, що він є дійсним файлом з розширенням .csv, розмір якого не перевищує 2048 кілобайт.

Далі ми викликаємо два методи, getCarsDataFromCSV() та processCarsData(), щоб отримати та обробити дані про автомобілі. Наразі ці методи ще не існують, але ми створимо їх наступними кроками.

Щоб оптимізувати обробку файлу з даними про автомобілі, ми можемо використати багатопоточність. Для цього ми створимо клас ProcessCarsFileJob, який буде відповідати за обробку файлу з даними про автомобілі.

Створюємо клас ProcessCarsFileJob за допомогою команди:

```
php artisan make:job ProcessCarsFileJob
```

В класі ProcessCarsFileJob визначимо метод handle(), в якому будемо обробляти файл з даними про автомобілі:

```php
public function handle()
{
    $cars = $this->file->getCars();

    foreach ($cars as $car) {
        // Обробка даних про автомобіль
    }
}
```

Рис. 3.6. Обробляння файлу з даними про автомобілі

Змінюємо код методу processCarsFile() в класі CarsController так, щоб він створював та додавав до черги завдань ProcessCarsFileJob з файлом з даними про автомобілі:

```php
public function processCarsFile(Request $request)
{
    $file = $request->file('cars_file');
    $filePath = $file->getRealPath();

    ProcessCarsFileJob::dispatch($file);

    return redirect()->back()->with('success', 'File uploaded and queued for processing.');
}
```

Рис. 3.7. ProcessCarsFileJob з файлом з даними про автомобілі

Тепер ми можемо запустити наш додаток та відвідати посилання /cars/process-file, щоб завантажити файл з даними про автомобілі та почати його обробку. Для перевірки можна вивести повідомлення з класу ProcessCarsFileJob, що буде викликатись кожен раз, коли обробляється новий файл.

```php
public function handle()
{
    $cars = $this->file->getCars();

    foreach ($cars as $car) {
        // Обробка даних про автомобіль
        echo "Processed car: " . $car['model'] . "<br>";
    }
}
```

Рис. 3.8. Вивести повідомлення з класу ProcessCarsFileJob

Тепер, якщо ми завантажуємо файл з 10000 рядків, ми побачимо, що обробка файлу відбувається більш ефективно завдяки багатопоточності.

Використання Redis для роботи з чергою завдань

Для ефективного опрацювання файлів з великою кількістю даних ми можемо використовувати Redis для зберігання та опрацювання завдань в черзі. Redis є однією з найпоширеніших інструментів для роботи з ключ-значенням сховищем даних та чергою повідомлень.

Для початку роботи з Redis нам потрібно встановити пакети через Composer:

```
composer require predis/predis
```

Після встановлення пакетів ми можемо налаштувати Redis в нашому .env файлі:

```
QUEUE_CONNECTION=redis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

Тепер ми можемо створити новий файл app/Jobs/ProcessCarsFile.php для обробки завдань в черзі:

```php
namespace App\Jobs;

use App\Services\CarsImportService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class ProcessCarsFile implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    protected $filePath;

    public function __construct(string $filePath)
    {
        $this->filePath = $filePath;
    }

    public function handle(CarsImportService $carsImportService)
    {
        $carsImportService->importCarsFromFile($this->filePath);
    }
}
```

Рис. 3.9. Обробка завдань в черзі

У цьому файлі ми створили клас ProcessCarsFile, який реалізує інтерфейс ShouldQueue. Це дозволяє нам додавати завдання до черги, щоб обробити його пізніше. В конструкторі ми приймаємо шлях до файлу з даними про автомобілі, а в методі handle ми викликаємо метод importCarsFromFile нашого сервісу CarsImportService, який ми створили раніше.

Тепер ми можемо додавати завдання до черги за допомогою наступного коду в контролері CarsController:

```php
use App\Jobs\ProcessCarsFile;
use Illuminate\Http\Request;

public function upload(Request $request)
{
    // ...

    ProcessCarsFile::dispatch($filePath);

    // ...
}
```

Рис. 3.10. Обробка завдань в черзі

Тепер, коли ми викликаємо метод dispatch для класу ProcessCarsFile, завдання автоматично додається до черги. Ми можемо переглянути чергу завдань Redis за допомогою інструмента командного рядка Laravel:

```
php artisan queue:listen
```

Створення відповідного воркера для обробки завдань

Тепер нам потрібен відповідний воркер для обробки завдань, що додані до черги Redis. Для цього ми можемо використати вбудований в Laravel воркер, який працює з Redis чергами.

Створимо новий файл ProcessCarsFileWorker.php у директорії app/Jobs, в якому реалізуємо метод handle, який буде виконувати необхідні дії для обробки файлу з даними про автомобілі. У нашому випадку це буде імпортування даних в базу даних.

```php
namespace App\Jobs;

use App\Services\CarsImporter;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class ProcessCarsFileWorker implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    /**
     * The path to the CSV file.
     *
     * @var string
     */
    protected $filePath;

    /**
     * Create a new job instance.
     *
     * @param  string  $filePath
     * @return void
     */
    public function __construct(string $filePath)
    {
        $this->filePath = $filePath;
    }

    /**
     * Execute the job.
     *
     * @return void
     */
    public function handle()
    {
        $importer = new CarsImporter();
        $importer->import($this->filePath);
    }
}
```

Рис. 3.11. Імпорт даних в базу даних

Як ми бачимо, цей воркер реалізує інтерфейс ShouldQueue, що дозволяє використовувати його з Redis чергами. В конструкторі ми приймаємо шлях до файлу з даними про автомобілі, який ми передали до черги, і зберігаємо його у властивості $filePath. У методі handle ми створюємо новий екземпляр класу

CarsImporter, який ми створили раніше, і викликаємо його метод import, передавши йому шлях до файлу з даними про автомобілі.

Тепер нам потрібно запустити воркер, який буде обробляти завдання з Redis черги. Для цього виконаємо команду:

```
php artisan queue:work redis --queue=cars
```

Ця команда запустить worker-програму в бескінечному циклі, який буде відслідковувати чергу завдань та виконувати їх, як тільки вони будуть додані до черги.

3.2. Інтерфейс програми

Для прикладу візьмемо дані по битих автомобілях з аукціону «Copart»

	Yard numb	Yard name	Sale Date I	Day of We	Sale time (	Time Zone	Item#	Lot numbe	Vehicle	Year	Make	Model Grc	Model Det	Body Style	Color	Damage D	Secondary	Sale Title S	Sale Title T	Has Keys-Y	Lot Cond.	VIN	Odometer	Odome
2	1	CA - VALLE	0			PDT	0	31783112	V	2013	FORD	TAURUS	TAURUS SEL		BLACK	FRONT END		CA	SC	YES	D	1FAHP2E	152803.0	N
3	1	CA - VALLE	0			PDT	0	32614842	U	1981	INTERNAT	ALL MODE	CARGOSTA	TILT CAB	WHITE	NORMAL	WEAR	CA	AN	YES		2HTD103	181102.0	A
4	1	CA - VALLE	0			PDT	0	34462002	L	2012	KYST	TRAILER	TRAILER		WHITE	VANDALIS	REAR END	CA	NR	NO	E	4YDF273	0.0	N
5	1	CA - VALLE	0			PDT	0	35722902	V	1995	ACURA	INTEGRA	INTEGRA SE		GREEN	VANDALISM		CA	SC	NO	E	JH4DC44	67609.0	A
6	1	CA - VALLE	0			PDT	0	36042602	M	2002	ARIM	BF90A	BF90A		TWO TON	ALL OVER		CA	AQ	NO	E	AMI2L32	0.0	E
7	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	37480542	V	2018	AUDI	A5	A5 PREMIUM		BLACK	FRONT EN	MINOR DE	CA	SC	YES	E	WAUPNA	45535.0	A
8	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	37532122	K	2012	INTERNAT	4000 SERI	########	CONVENT	YELLOW	NORMAL	WEAR	CA	CQ	YES	E	1HTMMA	0.0	N
9	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	37618512	V	2003	TOYOTA	4RUNNER	4RUNNER	4DR SPOR	TWO TON	MINOR DENT/SCRAT		CA	CQ	YES	D	JTEZT14F	204892.0	E
10	1	CA - VALLE	0			PDT	0	37878752	V	2003	MERCURY	SABLE	SABLE GS	SEDAN 4D	GRAY	MECHANICAL		WA	CT	YES	E	1MEFM5	203197.0	E
11	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	38948222	V	2005	TOYOTA	SIENNA	SIENNA CE		GOLD	REAR END		CA	SC	YES	D	5TDZA23	261849.0	A
12	1	CA - VALLE	0			PDT	0	39032212	K	2018	FREIGHTLI	ALL OTHE	M2 106 MED		WHITE	FRONT EN	TOP/ROOF	CA	SC	YES	E	3ALACWI	0.0	N
13	1	CA - VALLE	0			PDT	0	39839432	V	2022	FORD	BRONCO	BRONCO SPO		WHITE	TOP/ROOF	SIDE	CA	SC	YES	D	3FMCR9C	38.0	A
14	1	CA - VALLE	0			PDT	0	40247172	C	2006	DUCATI	600-799	749		BLACK	ALL OVER		CA	SC	NO	E	ZDM1UB	0.0	E
15	1	CA - VALLE	0			PDT	0	40709352	V	2018	MERCEDE	C-CLASS	C 300		BLACK	FRONT EN	MECHANI	GA	CH	YES	E	WDDWK	0.0	N
16	1	CA - VALLE	0			PDT	0	40923102	V	2013	NISSAN	ROGUE	ROGUE S	4DR SPOR	BLACK	SIDE	FRONT EN	CA	NR	NO	E	JN8AS5M	129075.0	A
17	883	*NCS - PA	20220719	TUESDAY	1800	PDT	0	41133982	V	2004	MERCEDE	CLK-CLASS	CLK 55 AM	COUPE	SILVER	NORMAL	WEAR	CA	CQ	YES	D	WDBTJ7(	162103.0	E
18	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	41312902	K	2007	CHEVROLE	C/K5500	C5500 C5V0		WHITE	FRONT EN	VANDALIS	CA	LU	EXM		1GBE5V1	0.0	E
19	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	41508072	V	2019	HYUNDAI	ELANTRA	ELANTRA SE		WHITE	FRONT END		CA	SC	NO	E	5NPD74L	26340.0	A
20	1	CA - VALLE	0			PDT	0	41915792	V	2002	ACURA	RSX	RSX		BLACK	FRONT EN	SIDE	CA	NR	NO	E	JH4DC54	0.0	E
21	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	42193722	R	2017	GRAN	IMAGINE I	IMAGINE DE		TWO TON	WATER/FL	BIOHAZAR	CA	SC	NO	E	573TE30	0.0	N
22	1	CA - VALLE	0			PDT	0	42481911	K	2011	FORD	ECONOLIN	ECONOLINE		YELLOW	STRIPPED	VANDALIS	CA	SC	NO	E	1FDWE3I	0.0	N
23	1	CA - VALLE	0			PDT	0	42536682	V	2020	MERCEDE	CLA-CLASS	CLA 250		WHITE	ALL OVER	MECHANI	CA	SC	YES	E	W1K5J4C	60716.0	A
24	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43049712	E	1991	FRUEHAUI	DUMPTRA	DUMPTRAIL		SILVER	ALL OVER		CA	SC	EXM	E	1H4H018	0.0	E
25	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43058642	K	1991	FORD	F700	F700		WHITE	BIOHAZARD/CHEMIC		CA	JR	YES	E	1FDNF70	210075.0	E
26	1	CA - VALLE	0			PDT	0	43301182	V	2012	FORD	F-150	F150 SUPER		GRAY	NORMAL	WEAR	CA	CT	YES	D	1FTFW1E	254881.0	A
27	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43322442	V	2009	MERCEDE	S-CLASS	S 550		BLACK	ALL OVER		CA	SC	YES	D	WDDNG7	123797.0	E
28	838	RENTAL VI	20220720	WEDNESD	1200	EDT	0	43387942	V	2020	TOYOTA	SIENNA	SIENNA XLE		BLUE	ALL OVER		CA	CQ	EXM	E	5TDYZ3D	0.0	N
29	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43471172	V	2014	SUBARU	FORESTER	FORESTER 2		SILVER	BURN - IN	BURN - EN	CA	AN	EXM		JF2SJAH	0.0	N
30	1	CA - VALLE	0			PDT	0	43787812	V	2015	AUDI	S3	S3 PREMIUM		WHITE	SIDE		CA	SC	YES	D	WAUBFG	97489.0	A
31	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43802302	V	1998	SUBARU	LEGACY	LEGACY 30T		TWO TON	REAR END		CA	SC	YES	D	453BG68	253625.0	A
32	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43822372	V	2008	HONDA	CIVIC	CIVIC SI		BLUE	FRONT EN	MECHANI	CA	DQ	YES	E	2HGFG21	0.0	E
33	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43829572	V	1989	TOYOTA	ALL OTHE	PICKUP CAB		SILVER	TOP/ROOF	FRONT EN	CA	SC	YES	E	JT5VN94	209973.0	A
34	1	CA - VALLE	0			PDT	0	43882082	L	1996	IMCO	TRAILER	TRAILER		WHITE	ALL OVER	TOP/ROOF	CA	CT	EXM	E	1M9S452	0.0	E
35	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	43909072	V	2019	FORD	RANGER	RANGER XL		BLACK	REAR END	SIDE	CA	SC	YES	D	1FTER4EI	32096.0	A
36	1	CA - VALLE	20220722	FRIDAY	1200	PDT	0	44081452	K	2006	FORD	ECONOLIN	ECONOLINE		WHITE	STRIPPED		CA	DV	EXM		1FDXE45	0.0	E

Рис. 3.12. Формат файлу який будемо зчитувати

В нашій системі відкриємо сторінку для імпорту файлу в систему, налаштовану під дані з нашого файлу.

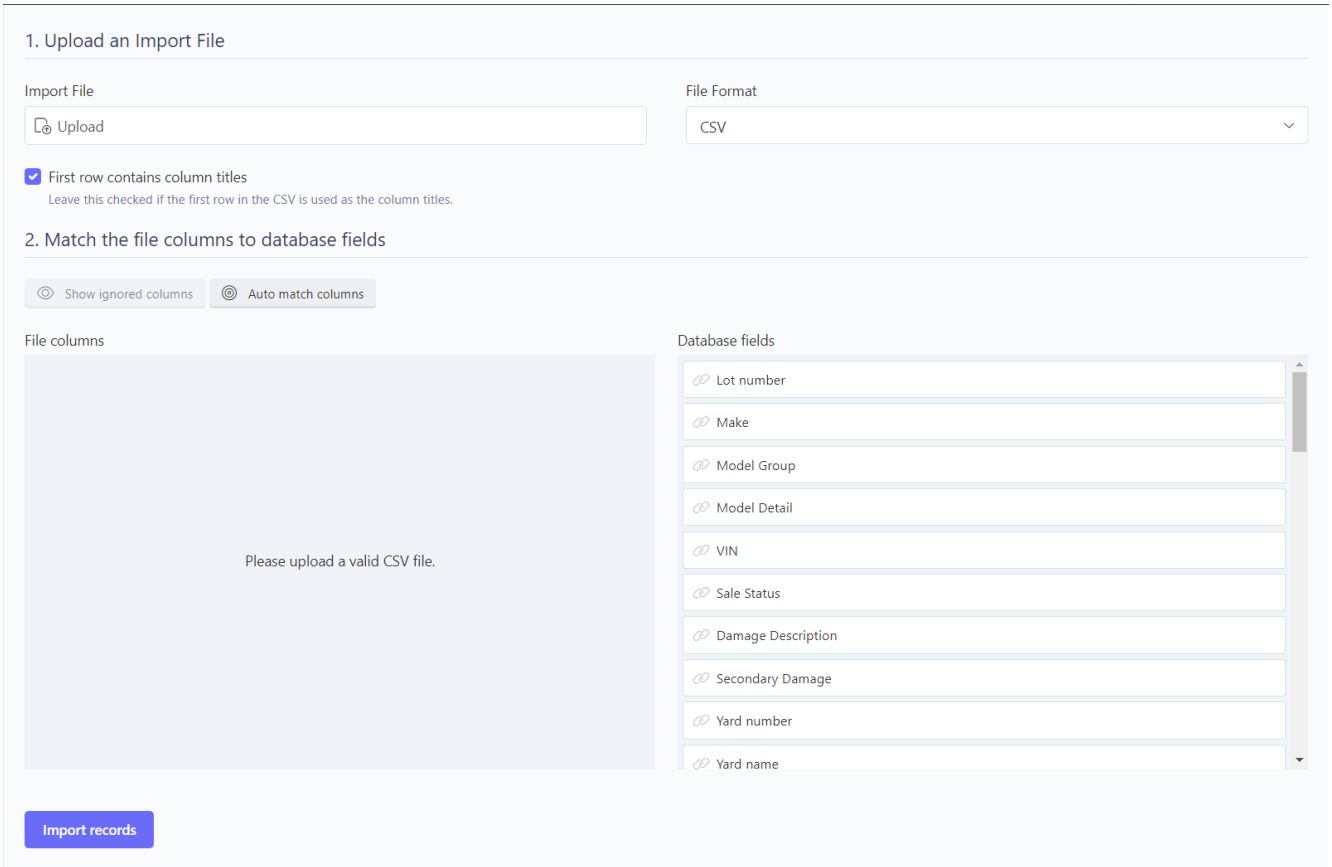

Рис. 3.13. Сторінка імпорту

Натискаємо на поле завантажити файл і підставляємо дані колонок з БД які будуть записуватись з файлу.

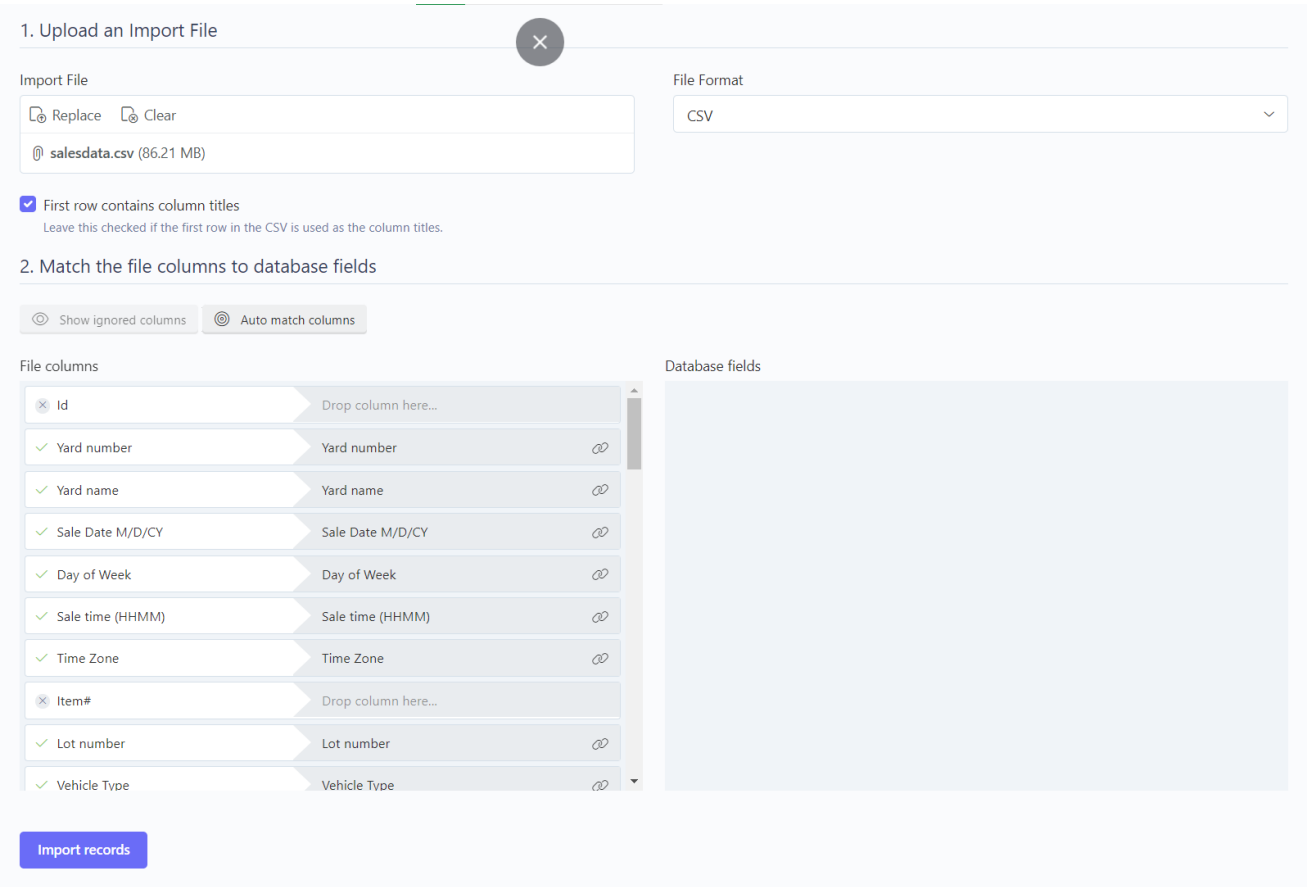

Рис. 3.14. Сторінка імпорту, приклад налаштування колонок для БД

Після чого натискаємо «Імпортувати» і чекаємо завантаження, після чого в нас з явиться повідомлення з інформацією скільки даних ми успішно зберегли в нашій системі.

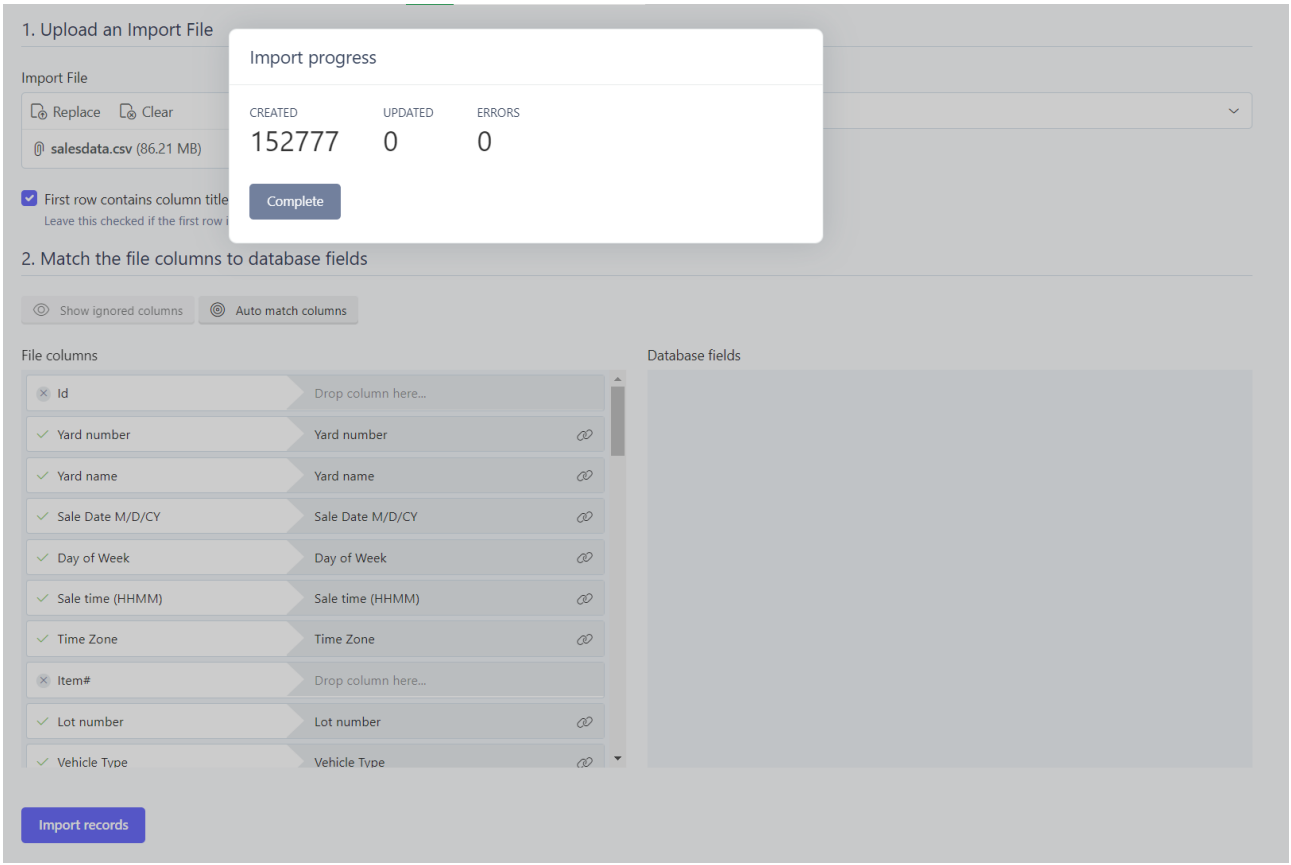

Рис. 3.15. Збереження даних в системі

Тепер ми можемо переглянути дані в системі на сторінці «Cars».

lot_number	make	model_group	model_detail	vin_code	sale_status	primary_damages	secondary_damages
31783112	FORD	TAURUS	TAURUS SEL	1FAHP2E88DG122598	Pure Sale	FRONT END	
32614842	INTERNATIONAL	ALL MODELS	CARGOSTAR	2HTD10354BCA14524	On Minimum Bid	NORMAL WEAR	
34462002	KYST	TRAILER	TRAILER	4YDF27328C1532294	Pure Sale	VANDALISM	REAR END
35722902	ACURA	INTEGRA	INTEGRA SE	JH4DC4460SS010466	Pure Sale	VANDALISM	
36042602	ARIM	BF90A	BF90A	AMI2L325C202	Pure Sale	ALL OVER	
37480542	AUDI	A5	A5 PREMIUM	WAUPNAF5XJA002310	On Minimum Bid	FRONT END	MINOR DENT/SCRATCHES
37532122	INTERNATIONAL	4000 SERIE	4000 4300	1HTMMAAM4CH436226	On Minimum Bid	NORMAL WEAR	
37618512	TOYOTA	4RUNNER	4RUNNER SR	JTEZT14R930010510	On Minimum Bid	MINOR DENT/SCRATCHES	
37878752	MERCURY	SABLE	SABLE GS	1MEFM50U63G621199	On Approval	MECHANICAL	
38948222	TOYOTA	SIENNA	SIENNA CE	5TDZA23C95S350382	Pure Sale	REAR END	
39032212	FREIGHTLINER	ALL OTHER	M2 106 MED	3ALACWFC6JDJW4690	Pure Sale	FRONT END	TOP/ROOF
39839432	FORD	BRONCO	BRONCO SPO	3FMCR9C63NRD12050	On Minimum Bid	TOP/ROOF	SIDE
40247172	DUCATI	600-799	749	ZDM1UB35X6B012630	Pure Sale	ALL OVER	
40709352	MERCEDES-BENZ	C-CLASS	C 300	WDDWK4JB2JF637231	Pure Sale	FRONT END	MECHANICAL
40923102	NISSAN	ROGUE	ROGUE S	JN8AS5MV0DW102908	Pure Sale	SIDE	FRONT END
41133982	MERCEDES-	CLK-CLASS	CLK 55 AMG	WDBTJ76H94F101622	On Minimum	NORMAL WEAR	

Рис. 3.16. Сторінка «Cars»

А також можемо відредагувати дані конкретного автомобіля якщо нам це буде необхідно.

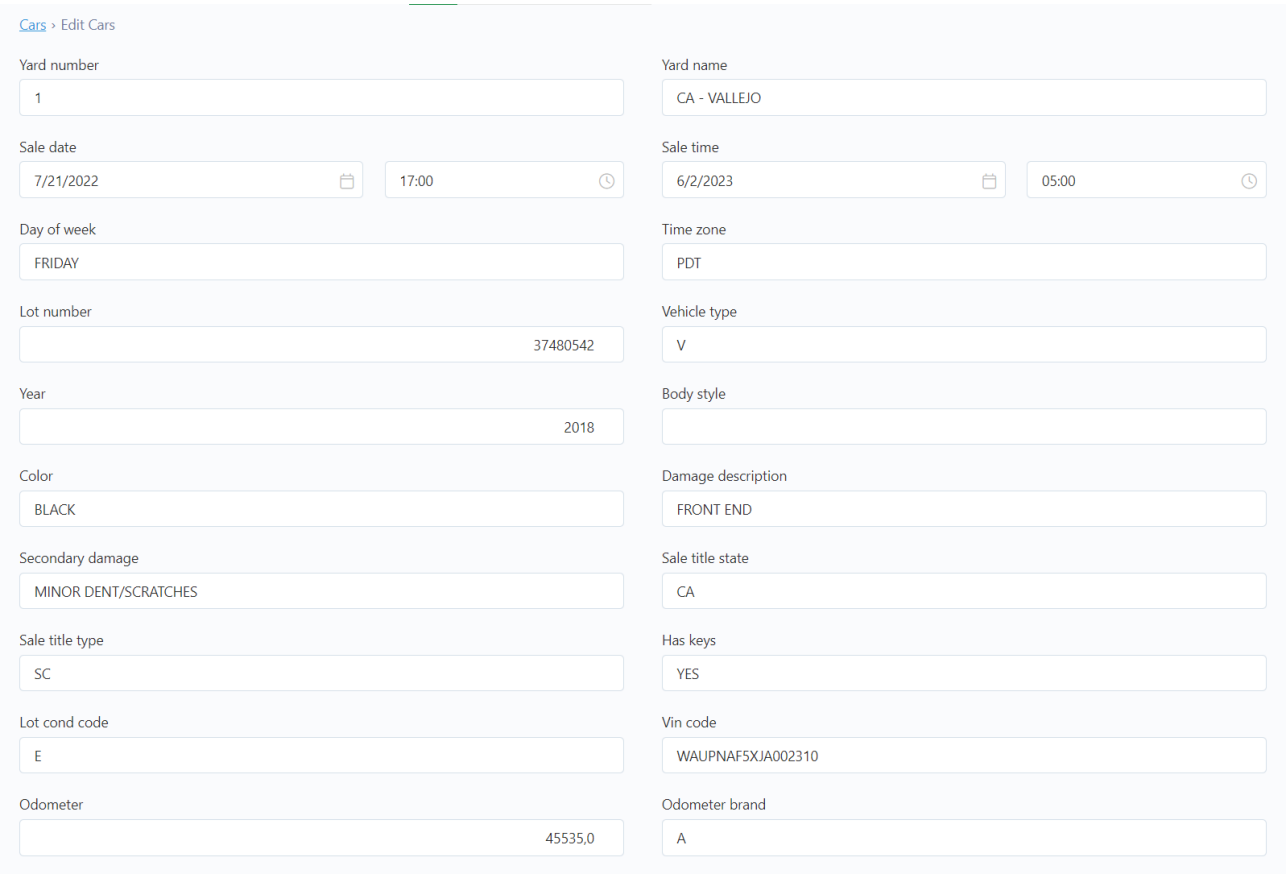

Рис. 3.17. Сторінка редагування

Тепер маючи дані у системі ми можемо працювати з ними як заманеться, наприклад виводити дані для клієнтів які цікавляться автомобілями.

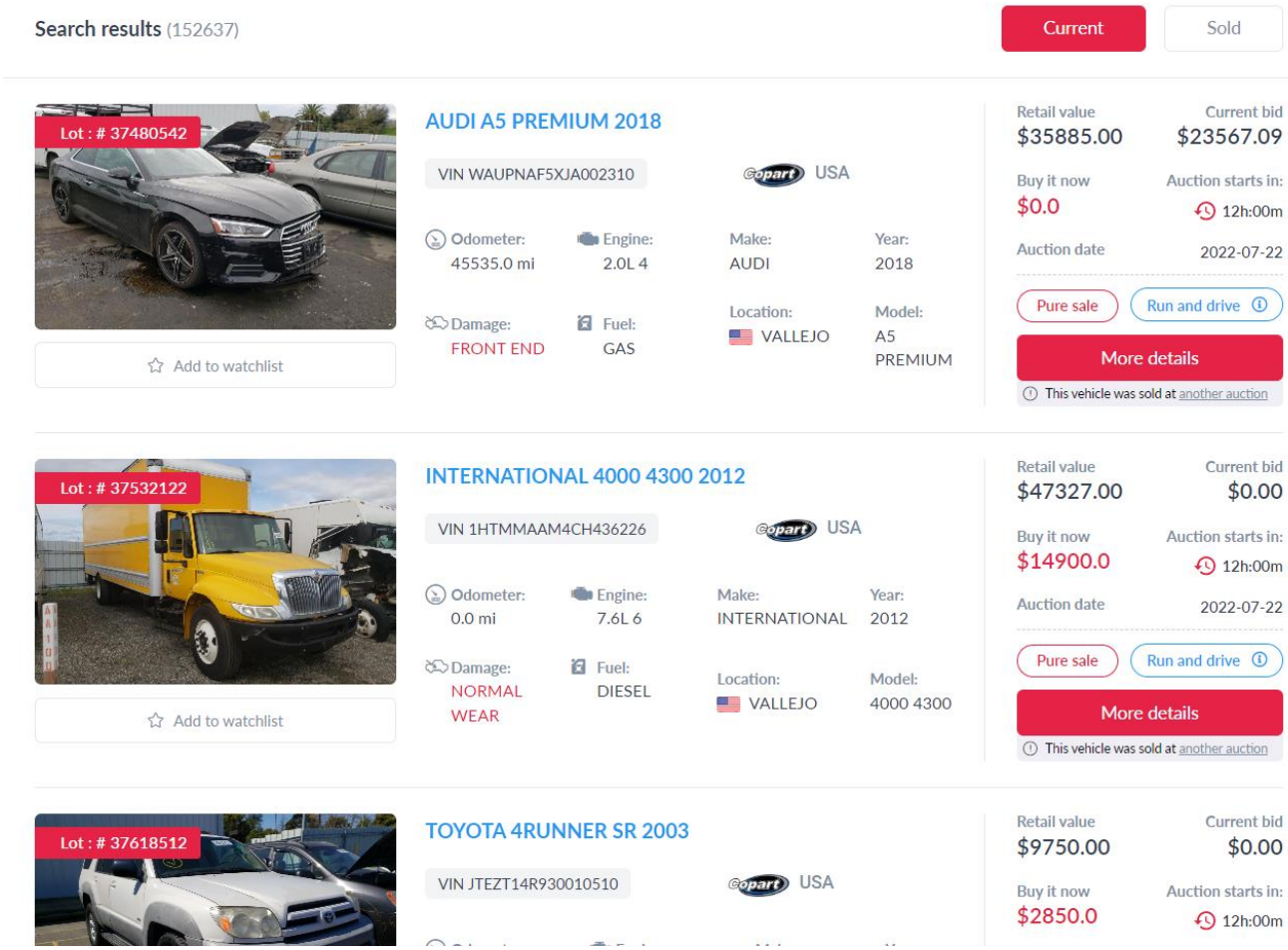

Рис. 3.18. Сторінка з усіма машинами для клієнтів

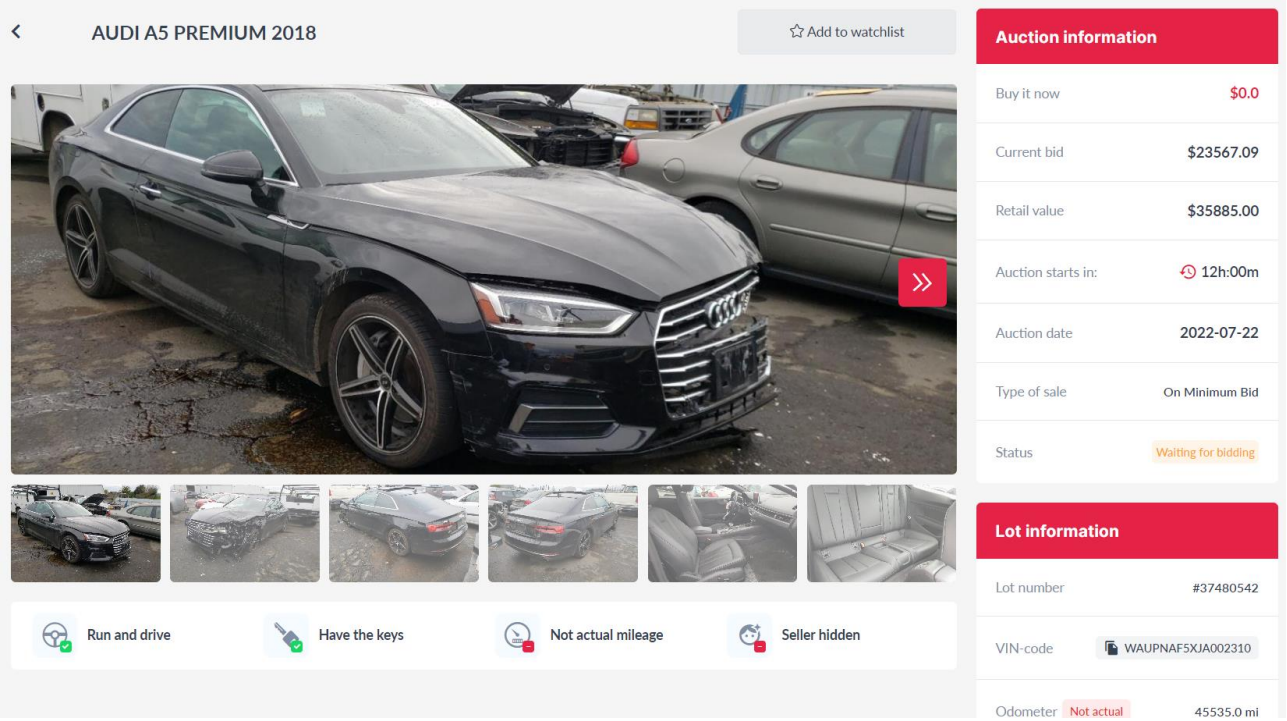

Рис. 3.19. Сторінка з інформацією конкретної машини для клієнтів

ВИСНОВКИ

Для пристрою створеного на основі мікроконтролера і сенсорів для вимірювання температури та тиску створено програмне забезпечення (прошивку, firmware). Мікроконтролер, що використовується у пристрої — ESP8266 фірми Espressif Systems. Сенсора для вимірювань BMP-280 фірми Bosch Sensortec GmbH. Програмне забезпечення надає можливість підключення до мережі Ethernet, зчитує дані вимірювань із сенсорів та дає результати вимірювань за HTTP запитом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bell D.A. Electronic instrumentation and measurements. - 2nd Ed. -Oxford: Oxford University Press, 2007, 451 p.
2. HelfrickA.D., Cooper W.D. Modern electronic instrumentation and measurement techniques. – London: Prentice-Hall International, 2008. – 446 p.
3. Єрмілова Н.В., Кислиця С.Г. Основи метрології і електричних вимірювань. – Полтава: ПолтНТУ, 2017. - 141 с.
4. Поліщук Є.С. , Дорожовець М.М., Яцук В.О. Метрологія та вимірювальна техніка. - Друге видання. - Львів: Видавництво Львівської політехніки, 2012. 544 с.
5. Метрологія та вимірювання: навчальний посібник / Ю.В. Гнусов, В.В. Тулупов, В.М. Пересічанський; Харк. нац. ун-т внутр. справ, 2019. - 125 с.
6. Khan S., Alam M. File Formats for Big Data Storage Systems // International Journal of Engineering and Advanced Technology. – 2019.
7. Choosing a Data Storage Format in the Apache Hadoop System Based on Experimental Evaluation Using Apache Spark [Електронний ресурс] – Режим доступу: https://www.mdpi.com/2073-8994/13/2/195 (дата звернення: 04.06.2023).
8. Understanding Big Data File Formats [Електронний ресурс] – Режим доступу: https://www.vladsiv.com/big-data-file-formats/ (дата звернення: 04.06.2023).
9. Data Formats Supported for Data Ingestion [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/fabric/real-time-intelligence/ingestion-supported-formats (дата звернення: 04.06.2023).
10. Parquet: Columnar Storage Format for Hadoop [Електронний ресурс]. – Режим доступу: https://parquet.apache.org (дата звернення: 04.06.2023).
11. ORC File Format Documentation [Електронний ресурс]. – Режим доступу: https://orc.apache.org (дата звернення: 04.06.2023).

12. Laravel Documentation. [Електронний ресурс]. – Режим доступу: https://laravel.com/docs/9.x (дата звернення: 04.06.2023).
13. Symfony Documentation [Електронний ресурс] – Режим доступу: https://symfony.com/doc (дата звернення: 04.06.2023).
14. PHP Manual [Електронний ресурс] – Режим доступу: https://www.php.net/docs.php (дата звернення: 04.06.2023).
15. Apache Arrow: Columnar Data Format for Analytics Applications [Електронний ресурс] – Режим доступу: https://arrow.apache.org (дата звернення: 04.06.2023).

ДОДАТКИ

Файл CsvController.php

```
use Symfony\Component\Process\Process;
class CsvController extends Controller
{
   public function processCsv()
   {
      // Шлях до файлу з даними
      $filePath = storage_path('app/csv/data.csv');

      // Кількість потоків
      $threads = 4;

      // Створюємо масив з командами для запуску процесів
      $commands = [];
      for ($i = 0; $i < $threads; $i++) {
         $commands[] = sprintf('php %s process-csv --start=%d --end=%d', base_path('artisan'), $i *
1000, ($i + 1) * 1000);
      }

      // Запускаємо процеси в фоновому режимі
      $processes = [];
      foreach ($commands as $command) {
         $processes[] = new Process(explode(' ', $command));
      }
      Process::runProcesses($processes);

      // Обробляємо результати
      $result = [];
      foreach ($processes as $process) {
         $result[] = $process->getOutput();
      }
      return view('csv.result', compact('result'));
   }
```

```
public function processCarsFile($filePath)
{
        // Отримуємо кількість рядків у файлі
        $rowCount = Excel::toArray([], $filePath)[0]->count();

        // Розбиваємо файл на 4 частини
        $chunkSize = ceil($rowCount / 4);

        // Ініціалізуємо об'єкт ProcessManager
        $manager = new ProcessManager();

        // Запускаємо 4 процеси для обробки кожної частини файлу
        for ($i = 0; $i < 4; $i++) {
                $start = $i * $chunkSize;
                $end = ($i + 1) * $chunkSize;
// Додаємо процес до менеджера процесів
$manager->addProcess(function () use ($filePath, $start, $end) {
// Отримуємо дані з частини файлу
$data = Excel::toArray([], $filePath, '', true, false, ['range' => "{$start}:{$end}",])[0];
                        // Обробляємо дані
                        foreach ($data as $row) {
                                $car = new Car();
                                $car->make = $row['make'];
                                $car->model = $row['model'];
                                $car->year = $row['year'];
                                $car->save();
                        }
                });
        }
        // Запускаємо процеси
        $manager->startProcesses();
        // Очікуємо завершення процесів
```

```
                $manager->waitProcesses();
        }
}
```

Файл ProcessCsv.php

```
use Illuminate\Console\Command;
class ProcessCsv extends Command
{
    protected $signature = 'process-csv {--start=0} {--end=1000}';
    protected $description = 'Process CSV file';

    public function handle()
    {
        // Отримуємо параметри команди
        $start = $this->option('start');
        $end = $this->option('end');

        // Опрацьовуємо файл з даними
        $filePath = storage_path('app/csv/data.csv');
        $file = fopen($filePath, 'r');
        $lineNumber = 0;
        while (($line = fgetcsv($file)) !== false) {
            if ($lineNumber >= $start && $lineNumber < $end) {
                // Обробляємо рядок даних
            }
            $lineNumber++;
        }
        fclose($file);
    }

        public function processCarsFile(Request $request)
        {
                // отримуємо файл з запиту
                $file = $request->file('cars_file');
```

```
                // проводимо валідацію файлу
                $this->validate($request, [
                        'cars_file' => 'required|mimes:csv,txt|max:2048',
                ]);

                // зчитуємо дані з файлу та обробляємо їх
                $carsData = $this->getCarsDataFromCSV($file);
                $result = $this->processCarsData($carsData);

                // повертаємо результат обробки даних
                return response()->json([
                        'success' => true,
                        'result' => $result,
                ]);
        }
}
```

Файл ProcessCarsFile.php

```
namespace App\Jobs;
use App\Services\CarsImportService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class ProcessCarsFile implements ShouldQueue
{
   use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
   protected $filePath;
   public function __construct(string $filePath)
   {
      $this->filePath = $filePath;
   }
```

```
    public function handle(CarsImportService $carsImportService)
    {
        $carsImportService->importCarsFromFile($this->filePath);
    }
}
```

Файл ProcessCarsFileWorker.php

```
namespace App\Jobs;
use App\Services\CarsImporter;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
class ProcessCarsFileWorker implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
    /**
     * The path to the CSV file.
     * @var string
     */
    protected $filePath;
    /**
     * Create a new job instance.
     * @param  string  $filePath
     * @return void
     */
    public function __construct(string $filePath)
    {
        $this->filePath = $filePath;
    }

    /**
```

```
     * Execute the job.
     * @return void
     */
    public function handle()
    {
        $importer = new CarsImporter();
        $importer->import($this->filePath);
    }
}
```