Working with Databases & CLI
CSV Importer and Laravel API

Hatim Salhi

February 21, 2026

Contents

1 Introduction

The objective of this project is to practice backend development concepts including:

- Building a Command Line Interface (CLI) application using Symfony Console
- Importing data from a CSV file into a MySQL database
- Using PDO and RAW SQL with transactions
- Exporting and transferring database data
- Creating a REST API using Laravel
- Implementing filtering, sorting, pagination and basic authentication

Two main parts were implemented:

1. Symfony CLI application for CSV import
2. Laravel API connected to the imported data

2 Part 1 – Symfony CLI CSV Importer

2.1 Project Setup

The project was initialized using Composer and the following packages were installed:

- symfony/console
- league/csv
- symfony/dotenv

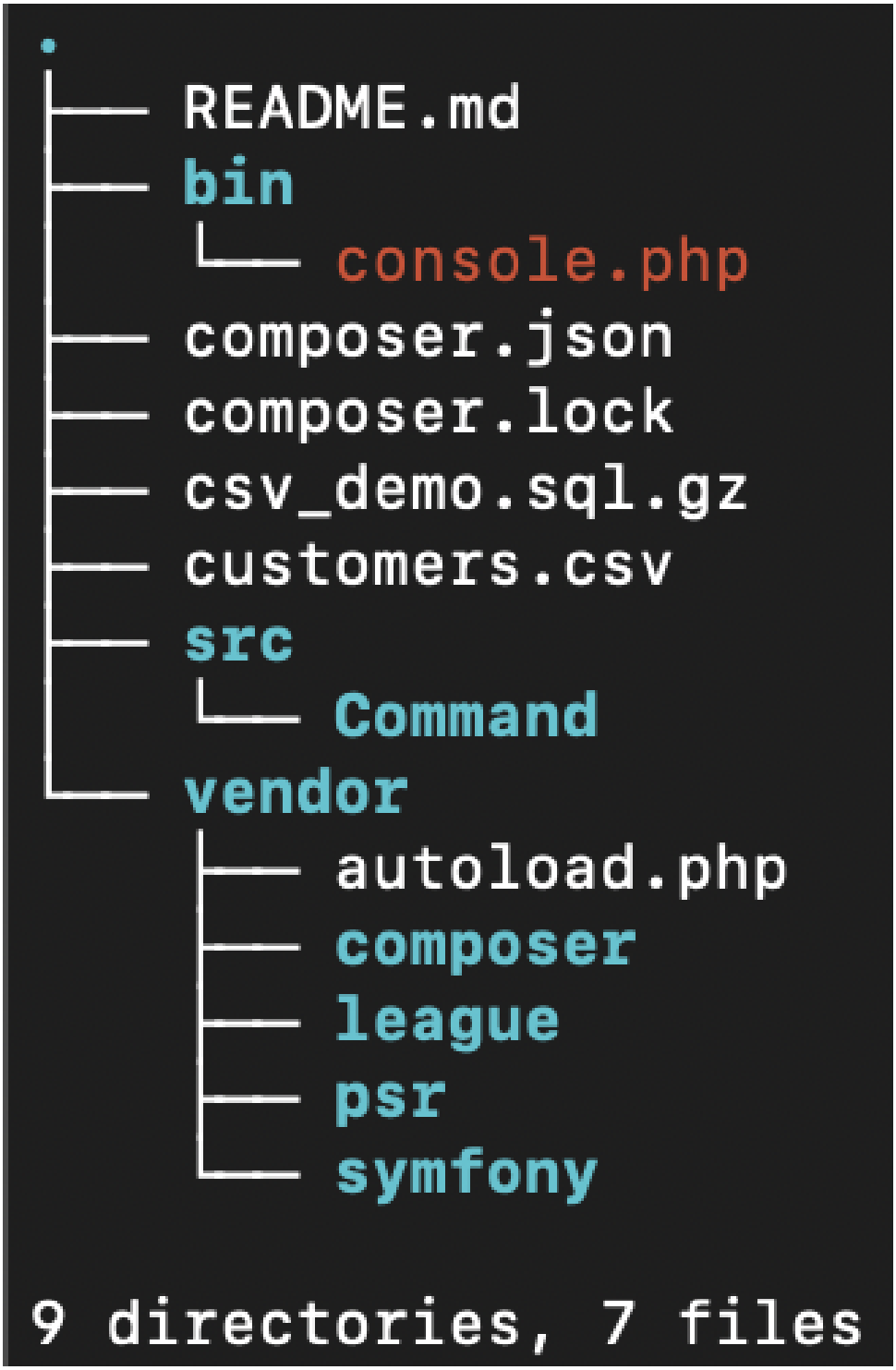

Figure 1: Project structure

2.2 Database Creation

A MySQL database named `csv_demo` was created. A user `csv_user` was created and granted privileges on the database.

```
1 CREATE DATABASE csv_demo;
2
3 CREATE USER 'csv_user'@'localhost' IDENTIFIED BY 'password';
```

```
GRANT ALL PRIVILEGES ON csv_demo.* TO 'csv_user'@'localhost';
FLUSH PRIVILEGES;
```

2.3 Table Design

A `customers` table was created to store the CSV data.

```
CREATE TABLE customers (
    id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    csv_index INT,
    customer_id INT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    company VARCHAR(150),
    city VARCHAR(100),
    country VARCHAR(100),
    phone1 VARCHAR(50),
    phone2 VARCHAR(50),
    email VARCHAR(190),
    website VARCHAR(190),
    subscription_date DATE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.4 CSV Import Logic

The League CSV Reader was used to read the CSV file. Data was inserted using PDO prepared statements.

All inserts were wrapped inside a database transaction:

```
$pdo->beginTransaction();

try {
    foreach ($records as $record) {
        $stmt->execute([...]);
    }
    $pdo->commit();
} catch (Exception $e) {
    $pdo->rollBack();
}
```

This guarantees that either all rows are inserted or none.

2.5 Execution Time Measurement

Execution time was measured using:

```
$start = hrtime(true); // nanoseconds
```

Figure 2: Execution Start

```
$end = hrtime(true);
$seconds = ($end - $start) / 1_000_000_000;
```

Figure 3: Execution End

2.6 CLI Execution

```
→  csv-importer git:(main) php bin/console.php app:import-csv customers.csv
Inserted rows: 10000
Execution time: 1.3151 seconds
```

Figure 4: CSV Execution Result

3 Export and Transfer of Data

3.1 Export using mysqldump

```
mysqldump -u csv_user -p csv_demo > csv_demo.sql
```

Figure 5: Database export

3.2 Compression

```
csv-importer gzip -9 csv_demo.sql
```

Figure 6: Compressed SQL file

4 Part 2 – Laravel REST API

4.1 Laravel Setup

A new Laravel application was generated and connected to the existing database.

4.2 Customer Model

```
class Customer extends Model {
    protected $table = 'customers';
    public $timestamps = false;
}
```

4.3 Custom Header Middleware

A middleware adds the header:

`x-api-version: v1`

```php
public function handle(Request $request, Closure $next): Response
{
    $response = $next($request);
    return $response->header('x-api-version', env('API_VERSION', 'v1'));
}
```

Figure 7: API version header

4.4 Basic Authentication

API routes are protected using Basic Auth with credentials stored in `.env`.

4.5 Filtering, Sorting and Pagination

The endpoint supports:

- filter[name]
- filter[email]
- sort[name]
- sort[email]
- page[number]
- page[size]

Example request:

```
curl -u admin:admin123 \
"http://127.0.0.1:8000/api/users?filter[name]=Eladio&page[number]=1&
    page[size]=10"
```

Figure 8: API JSON response (filtered)

5 Testing Tools

API was tested using:

- curl
- Postman

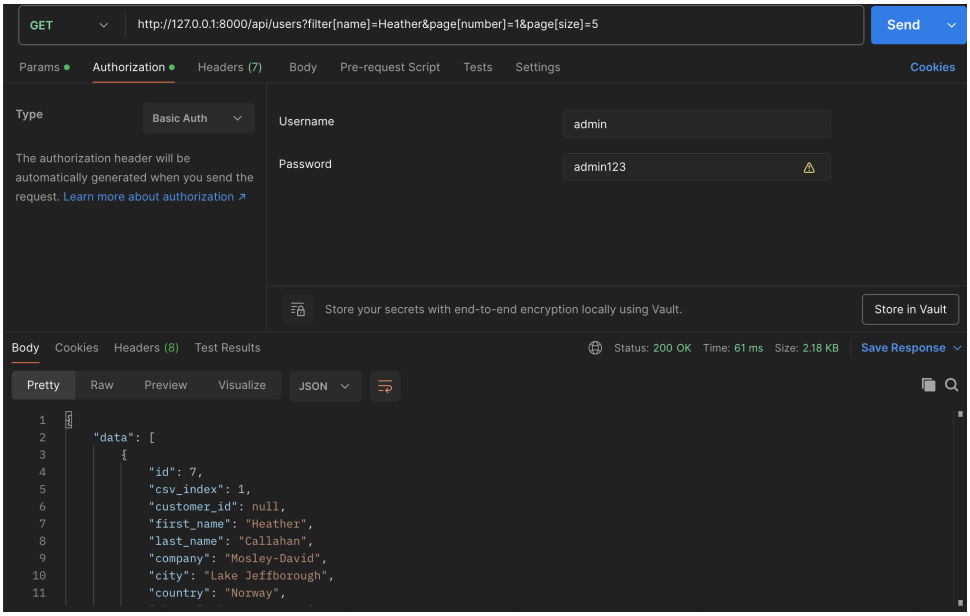

Figure 9: Postman testing

6 Conclusion

This project demonstrated how to:

- Build a CLI tool with Symfony
- Import large CSV files safely
- Use database transactions
- Export and transfer databases
- Create a secure Laravel REST API
- Implement filtering, sorting and pagination

These skills reflect real-world backend development practices.