Excel Data Uploader

Software Design Documentation

Author: Charaka Kumarasinghe

2024/03/10

Overview

ExcelDataUploader is a simple application that you can upload Excel and CSV files (xlsx, csv) containing item details that can be used to insert or update the **items** database table.
Once upload an excel file via ***/upload*** route, it will be validated with the correct mime type. If validation fails, it will return an error with a message explaining the details. If successful, it will store the excel file in the app storage and add a record into the database with file information under **excel_files** table. Then it will dispatch a job ProcessExcelData which will be queued in the database queue.

Queued jobs will be run from background worker service, and it will start processing the uploaded excel file. This background job will import excel data from the file into a collection and validates for the data correctness. If required columns for the item table are missing, it will throw **InvalidExcelFile Exception** and update the **excel_files** table with processing error. If rows values are invalid, it will simply ignore those rows and only process valid rows. Once processing of the excel file completes, it will be then stored in the **items** table.

This can be used to insert or update data into the items table. If already existing item codes with data uploaded, they will be updated. Otherwise it will insert them as new records.

Excel file data format example.

Code	Description	Quantity	Price
IT001	Test item 1	5	5.00
IT002	Test item 2	7	8.00

Table1- Example excel file data

Item code should start with the “IT” prefix. Otherwise the row will be ignored as invalid data.
Code, Quantity and Price are mandatory fields

Technologies

This application is developed with PHP and MySQL with docker containerization. It can be deployed in any environment where docker and docker compose are available.

Framework & Software

- Laravel 10.x
- PHP8.2-fpm
- Mysql 8.3
- Nginx

- Docker, Docker compose

Third party libraries and tools

- Supervisord process controller - http://supervisord.org/
- Laravel Excel - https://github.com/SpartnerNL/Laravel-Excel
- Bootstrap CSS - https://getbootstrap.com/

Functional Requirements

- **Setup Laravel Project**

Created a new laravel project named ExcelDataUploader with initial project configurations. In here used Composer to create Laravel framework 10.x project

```
composer create-project laravel/laravel ExcelDataUploader
```

- **Implement Excel Data Upload**

Application has Controller, Service, Model and Request validation classes to handle the file upload. Route for /upload will map the file upload request to handler Controller class via form request validation. Service class has the methods based on domain requirements that are called from the Controller class and store data via Model to database.

- **Implement Queue Processing**

Configured database queue is set up with a background worker to process the uploaded excel files. When excel file upload is successful, the controller will dispatch the ProcessExcelData job to the database queue. Then the job will be background processed to extract the data from excel file and update the items database table.

- **PHPUnit Test**

Configured Laravel unit testing based on PHPUnit testing to cover the scenarios for excel file upload and data processing. Test cases cover the valid and invalid excel file uploads and successful and error data processing.

- **Containerize Application with Docker**

Setup docker-compose.yml to containerize the application with 4 components. App, DB, Web server and the Worker are the main components of the docker deployment. Two separate dockerfiles are there for App and worker to build them with necessary dependencies. PHP-fpm is used to run the app with nginx web server. Supervisord is used to set up laravel queue workers.

Architecture

This section briefs how the application architecture is designed based on the domain requirements.

Deployment

Following diagram highlights the docker deployment architecture of the application with key components.

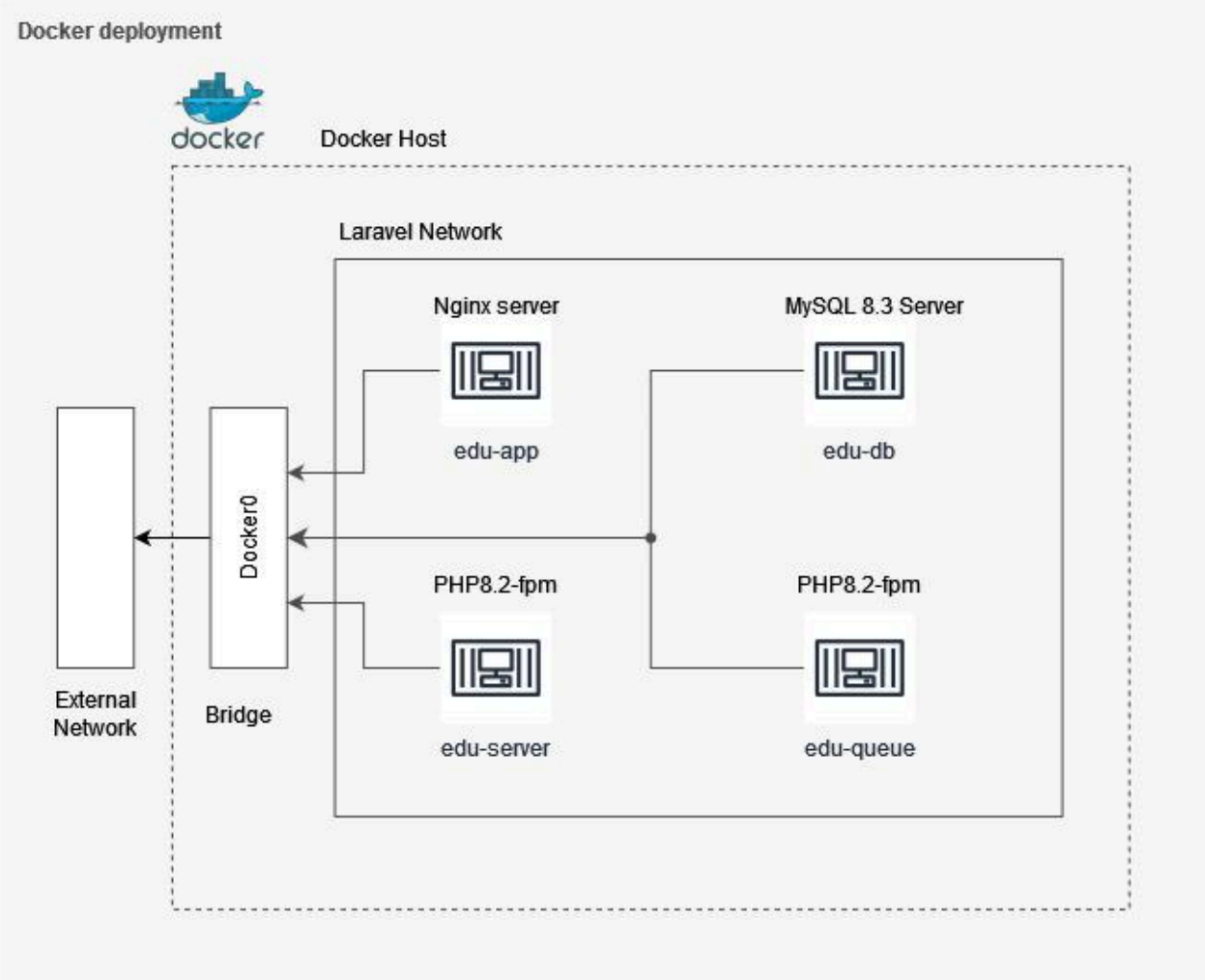

Figure1 - Docker deployment diagram

Components

1. **edu-app** - php-fpm instance that run the laravel application code
2. **edu-server** - nginx instance that handle http request and pass into edu-app
3. **edu-db** - mysql database instance
4. **edu-queue** - application instance that run queue worker with supervisord

Domain

Class diagram

Class diagram emphasizes the key classes of the domain. This includes controller, view, service, job and model classes.

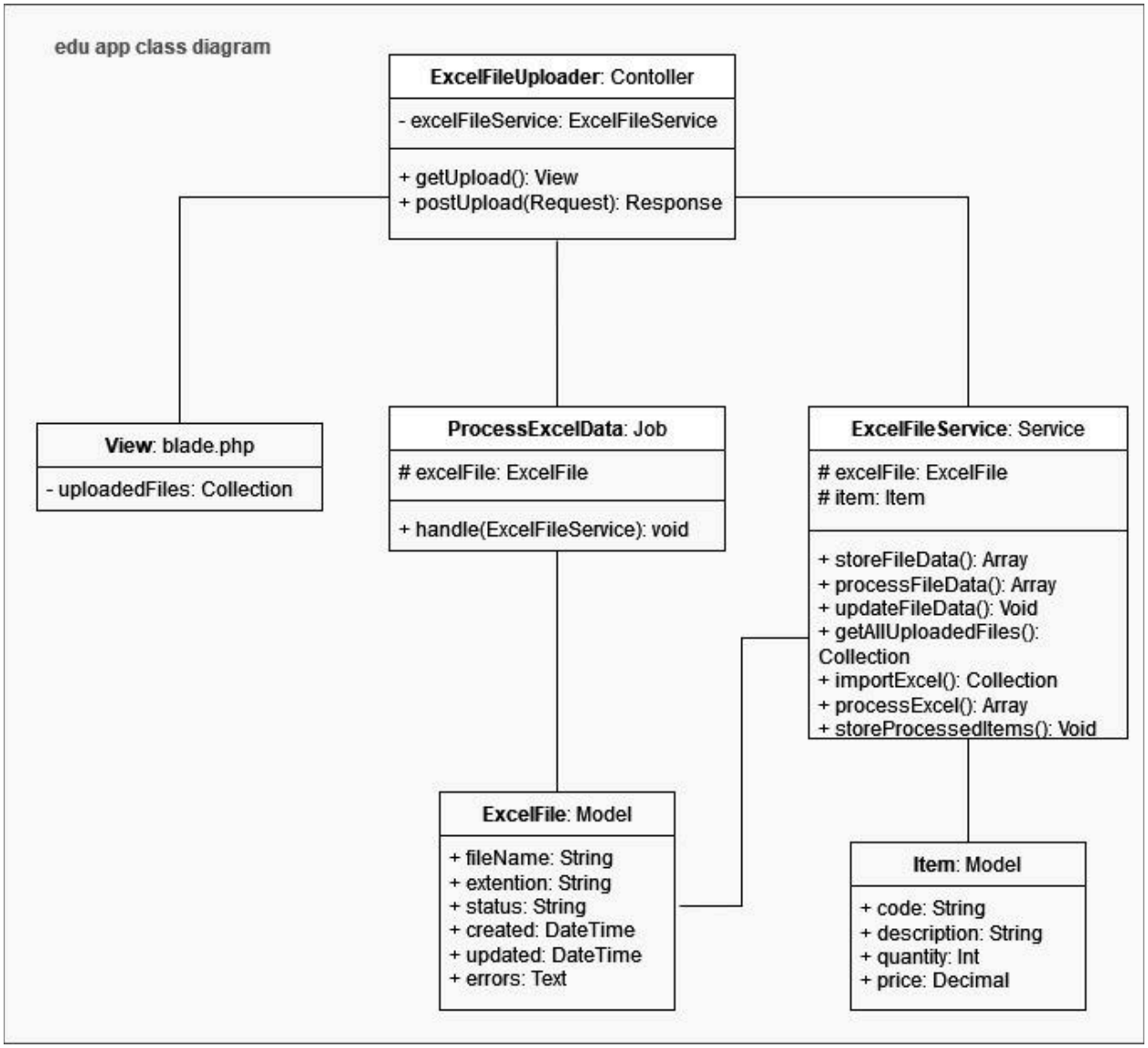

Figure2 - Class diagram

Other classes that are not included in the diagram

- ExcelDataImport - Data structure to map excel data to collection
- InvalidExcelFile - Exception class that throws when excel data is invalid
- DomainConst - Store domain constants

Sequence diagram

Sequence diagram explains how the request response life cycle with job dispatch and background queue work is done.

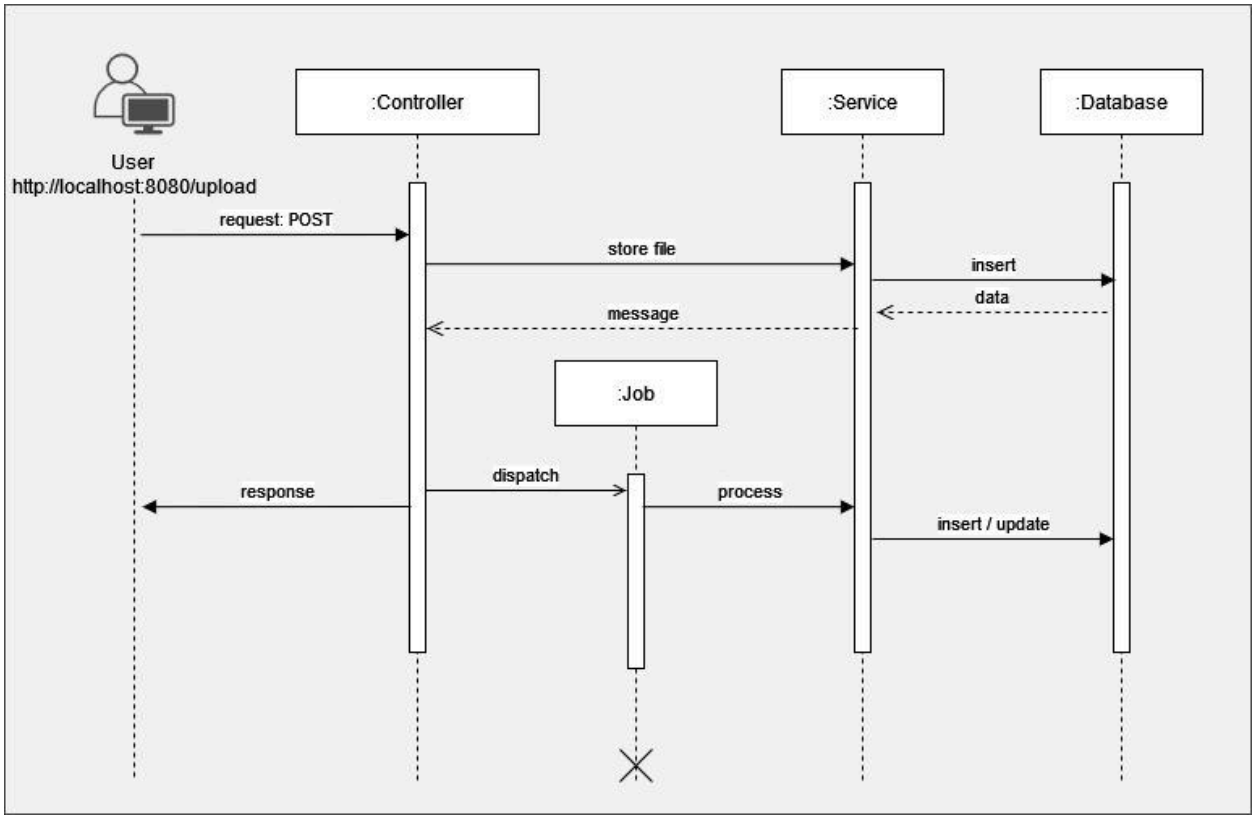

Figure3 - Sequence diagram

Sequence diagram flow

1. User send post request to /upload endpoint with data.
2. Controller validates the request and calls the Service class for file storage.
3. Service class stores the file and inserts the database and fetches results back to the controller.
4. Controller dispatch job to queue on successful file upload and response back to the user with the message.
5. Queued job will be processed and call Service class to process the excel data.
6. Service class stores the processed data in the database.

Setup Guide

ExcelDataUploader repository is hosted in https://github.com/ultimateck/ExcelDataUploader GitHub. Clone the repository to the local machine and go to the project root for next steps.

Project Configurations

Following prerequisites required to set up the application in the docker environment.

- Docker and docker-compose installed.
- Setup .env for configurations.
- Setup .env.testing for test configurations (optional)
- Port 8080 and 3306 available in the host machine. (This can be configured in docker-compose.yml)

Project root consist of following directories and files for dockerization of the application

- docker-compose.yml - all docker services are configured in here
- docker directory
    - mysql/my.cnf - (Optional) Mysql configuration
    - nginx/nginx.conf - Nginx configuration file
    - php/local.ini - PHP configurations
    - Dockerfile.app - edu-app docker file
    - Dockerfile.queue - edu-queue docker file
    - worker.conf - Supervisord configuration

Setup .env

In the project root directory create a new .env file using .env.example file. Apply / modify following configurations in the .env using text editor.

```
# DB configs
DB_CONNECTION=mysql
DB_HOST=edu-db
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root

# Queue configs
QUEUE_CONNECTION=database
```

DB_PASSWORD can be set to any value and it will be used in docker-compose.yml when creating the database.

Run Application

Once the configurations are done you can open a terminal (Linux/Mac) or Windows power shell or GitBash to run the docker-compose commands.

```
# Run docker-compose up command with build
docker-compose up --build
# Run docker-compose up command in background
docker-compose up --build -d
# Run docker-compose up command with force recreate
docker-compose up --build --force-recreate -d
```

These commands will build the docker images for edu-app and edu-queue. Download required images for services. Download the required libraries for Dockerfiles. Mount directories and volumes into the containers. Create a docker network and finally create containers and start up the application.

Run artisan commands

Once docker containers are up for the first time, it needs to set up the application key in the .env file and run the database migrations. In order to run these commands it requires connecting into the laravel application running container (edu-app) shell.

```
# Connect to edu-app interactive shell
docker exec -it edu-app /bin/bash
# If using windows GitBash it require //
docker exec -it edu-app //bin//bash
```

This will login into the edu-app shell with the working directory as /var/www. Make sure you are in the right directory that run the artisan commands.

```
# run artisan key generate
php artisan key:generate

# then run artisan migrate to generate db tables
php artisan migrate
```

Upload Excel File

Once above steps complete, the app will start on the configured port (ex: 8080)
Use a web browser to open the frontend http://localhost:8080/upload and start uploading excel files. Sample files are available under <project root>/tests/Files

Allowed file types for upload are xlsx and csv. Make sure the following required columns are present in the excel.

1. Code
2. Quantity
3. Price

File Upload Screen

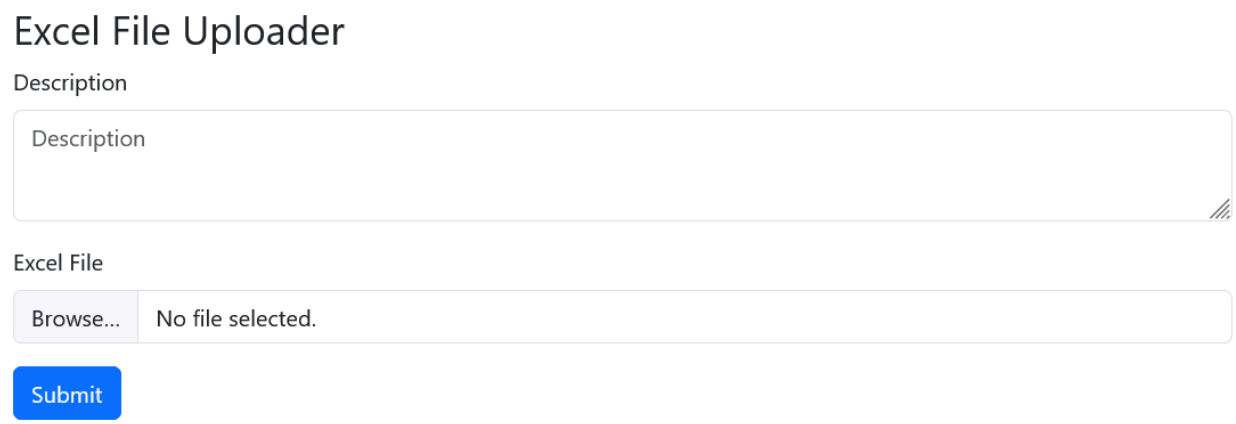

Figure4 - Excel File Upload Form

File Upload Processing Status Screen

Uploaded File Status

Id	Description	File name	Type	Status	Created	Updated	Errors
1	Test Description	items.xlsx	xlsx	PENDING	2024-03-09 09:15:35	2024-03-09 09:15:35	
2	Test Description	items.xlsx	xlsx	COMPLETED	2024-03-09 09:15:36	2024-03-09 09:15:37	
3	Test File	items.xlsx	xlsx	COMPLETED	2024-03-09 09:15:39	2024-03-09 09:15:40	

Figure5 - Excel File Upload Processing Status Table

When the excel file is uploaded to the application, it will set the status as PENDING and queue a job for processing the excel. Once the processing job completes it sets the status of the file as COMPLETE. If processing failed due to validations status will be set as FAILED

Testing

Laravel unit testing alongside PHPUnit testing is written to ensure the correctness of the application functionality. These tests are in the <project root>/tests directory under Feature and Unit directories. Sample files are under <project root>/tests/Files

Run Unit Test

Test cases can be run with artisan command. Simply connect into the edu-app container shell and run the command. Make sure that the current directory is /var/www where project root is mounted.

```
# connect to edu-app shell
docker exec -it edu-app /bin/bash
root@631eb3023819:/var/www#

# run unit tests
php artisan test
```

This will run all test cases under Feature and Unit directories. Optionally setup a testing database with .env.testing and run the test cases in the testing database. (Refer README.md)

Test Cases

Test cases cover the following functions and scenarios.

Function Coverage

- ☑ store file data
- ☑ process file data
- ☑ get all uploaded files
- ☑ import excel
- ☑ process excel
- ☑ process excel with invalid column excel
- ☑ process excel with invalid rows excel

Scenarios Coverage

- ☑ file upload route without data
- ☑ file upload route with invalid mime type
- ☑ file upload route with valid file