Creating a prototype of e-commerce Web application using ReactJS, MUI, Laravel, Laravel Sail, Orchid, Akeneo, MySQL and Elasticsearch, along with Docker, Git, GitHub Actions CI/CD, Kanban, Asana and AsciiDoc.

Table of Contents

This project aims to make use of some widely used Web technologies. The tool thus developed remains rudimentary and is of course not suitable for professional use.

Problem and Overview of the Solution

Problem

A fictitious store without a Web site selling a few dozen products that are particularly sought after by Internet users would like to have a showcase site to display these products. No major

> extension of the catalog is planned in the next ten years because it is a niche market. With the deployment of this site, the store will enjoy a greater presence on its market, thanks to the visibility that this showcase site would be able to provide on the Web. For reasons that only the owner of the store knows, he absolutely wants to use the Akeneo PIM, but he would like to be able to manage the prices of his products. However, Akeneo is not made for that…

This project addresses this problem by providing:

- A Web material user interface with ReactJS and MUI;
- The Akeneo PIM, which allows the store to create his products - by definition without handling prices, stocks, orders and invoices for example;
  - We fix this by making products prices manageable in Laravel Orchid and binding the later to Akeneo as you can read below.
- Elasticsearch to allow store's customers to search for certain products;
- A REST system consisting of a Laravel server with an Orchid administration panel, and MySQL to store the associated prices, defined within Orchid, to the Akeneo products. Entries in this MySQL table will be created upon Akeneo hooks triggering;
  - We argue that a relational database like MySQL is sufficient and preferable towards non-relational, highly scalable, ones like MongoDB because the store only have a few products to display, which don't need to be distributed accross cluster nodes thus.
- Docker containers sharing a same Docker network, for: (1/4) Elasticsearch, (2/4) MySQL, (3/4) Akeneo and (4/4) Laravel (including Orchid);
- Git and GitHub to handle project versioning;
- GitHub Actions (including CI/CD-oriented GitHub Actions Workflows) for tests automation for example;
- Agile method Kanban using Asana;
- AsciiDoc for the current documentation you are reading.

Installations

Docker

We have installed Docker Engine (v20.10.17, build 100c701) and Docker Compose (v2.6.0) by following both documentations:

- https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository
  - Using the Docker APT repository (section "Install using the repository");
- https://docs.docker.com/compose/install/compose-plugin/#install-using-the-repository
  - Using the Docker Compose APT repository (section "Install using the repository").

We chose to let the Docker daemon run as root (which it does by default), **i.e.**: we didn't opt-in to Rootless mode for the Docker daemon. Section "Docker daemon attack surface" of the documentation https://docs.docker.com/engine/security/#docker-daemon-attack-surface was taken into consideration when making this choice. No specific decision on this risk has been taken, having judged that it was not necessary because 1) our personal Ubuntu user is of course trusted and 2) we don't put this project in production (thus, only we are involved during the whole lifetime of the project).

- The Docker daemon involved in the CLI `docker` command script is `root` in order to be able to bind to a Unix socket. If interaction with this Unix socket by the mandatory of the Docker daemon is needed by the CLI `docker` command, it implies to run the last prefaced with `sudo`;
- More generally: the CLI `docker` command always need to be run with `sudo`.

To avoid using `sudo` each time we need to use the `docker` CLI commands, or each time a software needs to do so (like Akeneo setup for example, which moreover needs to bind to the Unix socket), we have added our personal Linux user to the `docker` group that was created by the Docker installation. It provides also an access to another Unix socket, the one that is created by the Docker daemon when it starts; this Unix socket is usable by the Unix group `docker` (thus, Akeneo setup scripts for example will be able to bind to this socket).

The package `docker-compose`, required by the Akeneo setup, was installed too, using `sudo apt-get install docker-compose`. The Docker documentation could not be involved for this operation, since Akeneo setup requires `docker-compose` and not `docker-compose-plugin`. It means that to use Docker Compose, we can either use the CLI commands: 1) `docker compose` (which is the latest version), or 2) `docker-compose` (v1.25.0, build unknown) (required by Akeneo setup).

Akeneo

We have installed Akeneo PIM Community Standard Edition Buccaneer Bunny (v6.0.32) by following both documentations:

- https://github.com/akeneo/pim-community-standard
- https://docs.akeneo.com/6.0/install_pim/docker/installation_docker.html

Akeneo was run in development mode. As no change in Akeneo is expected, it is not versioned (it is Git-ignored).

Laravel

We have installed Laravel Framework (v9.19.0) with Laravel Sail (allowing the use of Docker containers) by following this documentation: https://laravel.com/docs/9.x#getting-started-on-linux . The Laravel environment file is Git-ignored for security reasons, among others.

Laravel Orchid

We= Exx have installed Laravel Orchid (v12.6.2) by following this documentation: https://orchid.software/en/docs/installation .

ReactJs

We have installed ReactJS (v18.2.0) with Node.js node v18.4.0 (npm v8.12.1) (installed with nvm (v0.39.1)) by following these documentations:

- encodeURI('https://github.com/nvm-sh/nvm#install—update-script')
- https://nodejs.dev/learn/how-to-install-nodejs
- https://reactjs.org/docs/create-a-new-react-app.html#create-react-app

MUI

We have installed Material UI (v5.8.6) by following this documentation: https://mui.com/material-ui/getting-started/installation/#npm using `npm`. We have let the default styling engine, which is: `emotion`.

We have installed @mui/icons-material (v11.9.3) by following this documentation: https://mui.com/material-ui/getting-started/installation/#svg-icons using `npm`.

Elasticsearch

We have installed Elasticsearch (v8.2.0) using Docker by following this documentation: https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html .

Git

We have installed Git (v2.25.1) as a Ubuntu repository package by following this documentation: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git .

We have defined a `.gitignore` file for:

- Akeneo, as no code change must occur for the PIM;
- Some directories and files of Laravel: those which are defined by the Laravel `.gitignore` default file;
- Some directories and files of ReactJS: those which are defined by the ReactJS `.gitignore` default file.

GitHub Actions CI/CD (stands for: "Continuous Integration and Continuous Deployment")

After having pushed the repository of this project to GitHub using Git and the SSH Github authentication, we have defined these GitHub Actions for, among others, CI/CD purposes:

- @todo

Asana and Agile Kanban method

After having created the corresponding project in Asana, we have applied the Agile method named "Kanban" by creating a useful and visual, schedule of the tasks that were completed, to be completed, and currently being completed, with some additional features provided by the Asana software.

AsciiDoc (using Asciidoctor) and Asciidoctor-PDF

We have installed Asciidoctor (v2.0.10) and Asciidoctor-PDF (v1.5.0.alpha.17.dev) as Ubuntu repository packages by following these documentations (**resp.** Asciidoctor, and Asciidoctor-PDF):

- https://docs.asciidoctor.org/asciidoctor/latest/install/linux-packaging/
- https://github.com/asciidoctor/asciidoctor-pdf#install-asciidoctor-pdf

Execution workflow

To be run, this project requires to run in parallel: Elasticsearch, the Laravel server and Akeneo, as three distinct Docker sets of containers, and the ReactJS application. The instructions to run these softwares can be found in their respective documentation, whose links are written in the previous section "Installations". No significant configuration options were set for these softwares to run.

Development workflow

Without going as far as using the Agile SCRUM method since the team is limited to a single individual "administrator, designer, developer and tester" and because it is a project initiated for a fictitious client and a fictitious problem, we still proceeded with a prototyping and iterations workflow whose tasks were planned in Asana using the Kanban agile method. For example, the communication between Laravel and Akeneo, which relies on Akeneo Webhooks, was done by iterations on the basis of a prototype, which was improved and densified in terms of functionalities at each iteration, in order to reach a version that was always more stable and complete, even "finished" with regard to the project’s problem. We used this prototyping and iterative workflow for development, but also for systems administration (**i.e.:** handling installation and configuration of softwares), conception, documentation, writing of the Readme, etc.

Of course, Git (commits, branches, tags, etc.) was used to version the project. We made the choice to not create separate GitHub repositories for 1) the ReactJS application and 2) the Laravel server; instead, only one is used. Therefore, we created two main Git branches: one for the ReactJS application, the other for the Laravel server. We did not create the commonly used main Git branches `dev`, `prod` since this project will never (must not) reach the production state.