{EPITECH}

TRINITY - PART TWO_

< DEV WEB />

TRINITY - PART TWO

This project is composed of **three distinct parts**: dev Ops, dev Web and dev App.
The **timing** of deliveries and presentations has already been **fixed**.

To boost competitiveness, a grocery chain needs your dev skills to achieve several goals:

- ✓ They want to improve the customers' user experience, simplify their purchasing process, and enhance their satisfaction. To achieve this, you are tasked with developing an integrated and intuitive **mobile** application.
- ✓ Internally, they aim to optimize product management, streamline the supervision of sales, and facilitate decision-making through the visualization of key performance indicators. You are therefore assigned to implement a **web** application connected to a RESTful API.

WEB Development

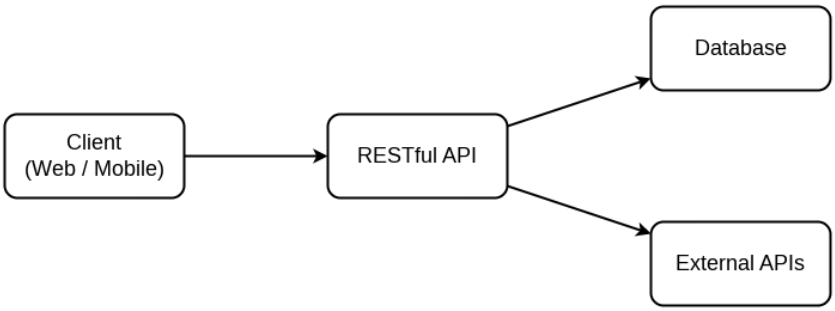
To improve stock and sales managements, optimize in-store operations, and enhance decision-making for managers, you need to develop a web application, with an associated API.

Mainly, your technical allows the managers of a store to:

- ✓ efficiently track products available in stock;
- ✓ easily manage customers and sales;
- ✓ astutely visualize key performance indicators.

API

The **A**pplication **P**rogramming **I**nterface must be **RESTful**.



Each API request must be **secured with a JWT token**.
The API endpoints are provided in the specification section below.

The user's content allows recording some information about a customer, such as:

- ✓ first name,
- ✓ last name,
- ✓ phone number,
- ✓ billing details (address, zip code, city, country).

The product's content allows recording some information about a product, including *at least*:

- ✓ name,
- ✓ price,
- ✓ brand,
- ✓ picture,
- ✓ category,
- ✓ nutritional information,
- ✓ available quantity in stock.

This list is **non-exhaustive**: you can add as many fields as you like, keeping in mind the size of your screen and ensuring a clear display for the user.

Be aware that, in a near future, you'll have to develop a mobile component, with a payment system via PayPal. Therefore, you should consider the PayPal API from now on, in order to set up callback URLs to keep payment information up to date.

Back office

Except the restrictions below, you are free to use any other tool and any language you fancy (nodeJS, python, java, ...). Be aware that you'll be asked to expose and justify your choices.

You are **NOT allowed** to use some tools that does everything for you, such as:

- ✓ API Platform, Strapi or any other framework generating automatically your API;
- ✓ ReactAdmin or any other framework building automatically your back office.

Your product stock management:

- ✓ allows for the creation, reading, updating, and deletion of products in stock;
- ✓ is integrated with the Open Food Facts API to automate product's information update;
- ✓ provides some advanced search and filtering features to facilitate product management.

You have to fetch the products directly from the Open Food Facts API.

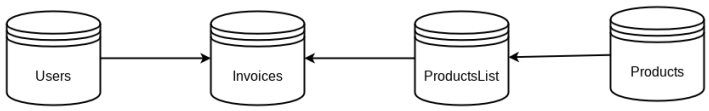
Your user management:

- ✓ allows for the creation, reading, updating, and deletion of user(s);
- ✓ enables viewing of user purchases, including payment history.

Your report management implements at least 5 key performance indicators.

You have to define (and justify!) your KPIs criteria. For instances: the average purchase in the last X hours, the most purchased products, the median of customers' payments, ...

Here is an example of the database schematization, to store the different elements:



This is an example; you can follow it or do anything else that seems relevant.
You'll have to expose and justify your choices regarding your database implementation.

Front office

You are free to:

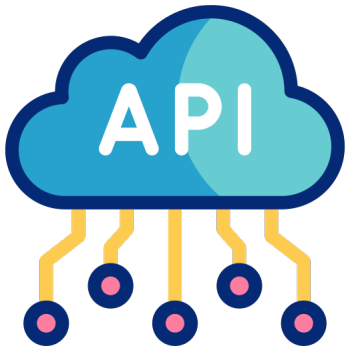
- ✓ use any tool you fancy (React|Angular|Vue, Figma|Sketch, Sass|Less|Stylus ...);
- ✓ design your front as you like (elements, fonts, colors, proportions, ux, ui, ...).

Be aware that you'll be asked to expose and justify your choices.
Be creative but do not neglect user's experience. Distinguish the good, the bad, and the ugly!

Specification details

Routes and endpoints

- ✓ Users:
  - `GET /users` retrieves the users.
  - `POST /users` adds a user.
  - `PUT /users/{id}` updates a user.
  - `DELETE /users/{id}` deletes a user.
- ✓ Invoices:
  - `GET /invoices` retrieves the invoices.
  - `POST /invoices` adds an invoice.
  - `PUT /invoices/{id}` updates an invoice.
  - `DELETE /invoices/{id}` deletes an invoice.
- ✓ Products:
  - `GET /products` retrieves the products.
  - `POST /products` adds a product.
  - `PUT /products/{id}` updates a product.
  - `DELETE /products/{id}` deletes a product.
- ✓ Reports:
  - `GET /reports` generates reports on sales and performance.



Technical challenges

- ✓ **Data security:**
  - secure communications between the web app, the API, and the external services;
  - session management must be protected against CSRF and XSS attacks.
- ✓ **Open Food Facts API integration:**
  - set up requests to automatically retrieve and update product information;
  - handle errors if data from the Open Food Facts API is incomplete or unavailable.
- ✓ **Code architecture:**
  - adopt a modular architecture to allow better maintainability and scalability;
  - use design patterns (such as Model-View-ViewModel or Model-View-Presenter) to separate concerns and facilitate unit testing;
  - implement reusable services and components to optimize development;
  - document the code architecture and data flows to ease onboarding for new developers and collaboration between teams.
- ✓ **Testing:**
  - write unit tests for all critical features of the application to ensure their proper functionality. A minimum of 20% coverage is expected;
  - use testing frameworks like Jest for the frontend and Mocha/Chai for the backend;
  - automate the execution of unit tests as part of the CI pipeline to quickly detect regressions.

Delivery and keynote

During this second review, you are to present and justify:

- ✓ your API;
- ✓ your back office;
- ✓ your front office.

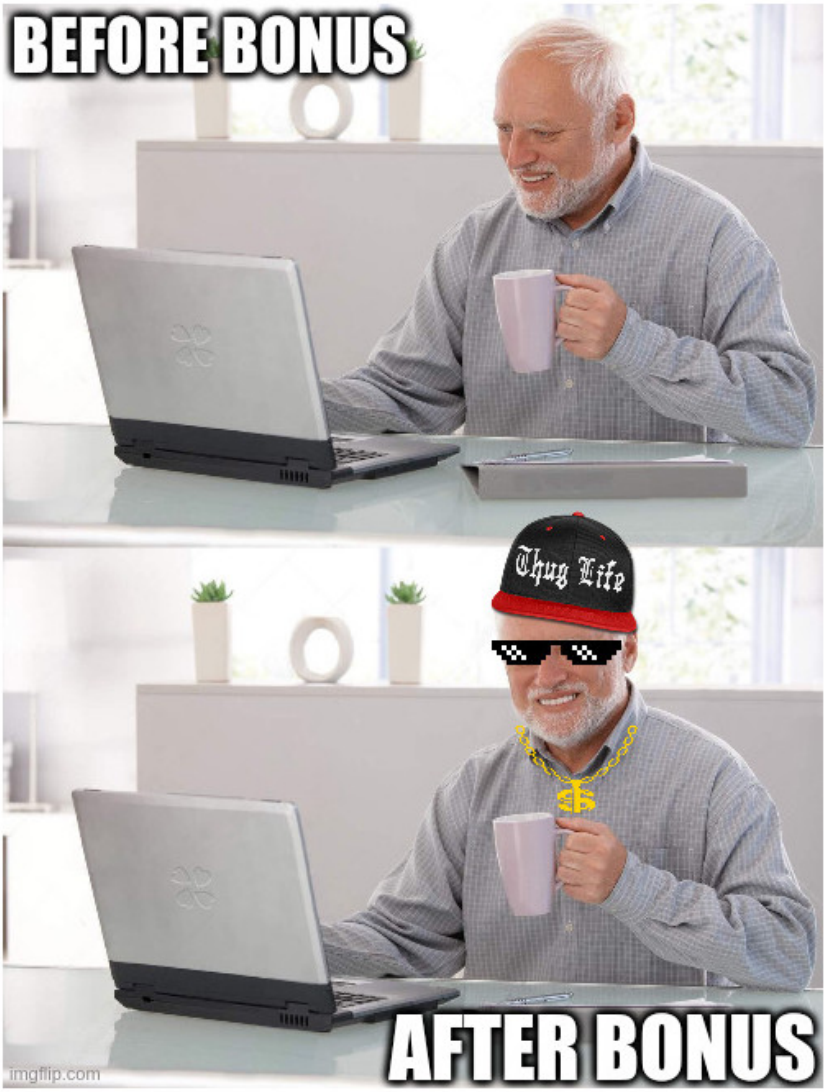
You are expected to deliver:

- ✓ The source code of your WEB application.
- ✓ The technical documentation for:
  - its architecture and its components;
  - your technological choices;
  - the data flows describing how data moves within the application.
- ✓ Some UML diagrams, including:
  - a class diagram showing data structures and their relationships;
  - an activity diagram, illustrating different workflows and business processes.
- ✓ The unit tests report, which should be directly visible in the CI/CD pipeline.

Bonuses

Once you finished the mandatory parts, add some optional features to your project, such as:

- ✓ add a feature to export purchase receipts in PDF format for easy storage;
- ✓ provide an option to send purchase receipts via email directly from the application;
- ✓ add other payment options, such as the local currency for your region;
- ✓ implement a guided tour of your web application;
- ✓ make the report customizable, with awesome dataviz for your KPIs;
- ✓ include modes to improve the accessibility of your interfaces.



v 0.3