15.11.2018

Ho Jung

Project 8: Pur Beurre

Objectif

The objective of this project was to create a web application, for a start-up restaurant called Pur Beurre located at Montmartre, Paris, which allows users to find healthier food replacements in a single click.

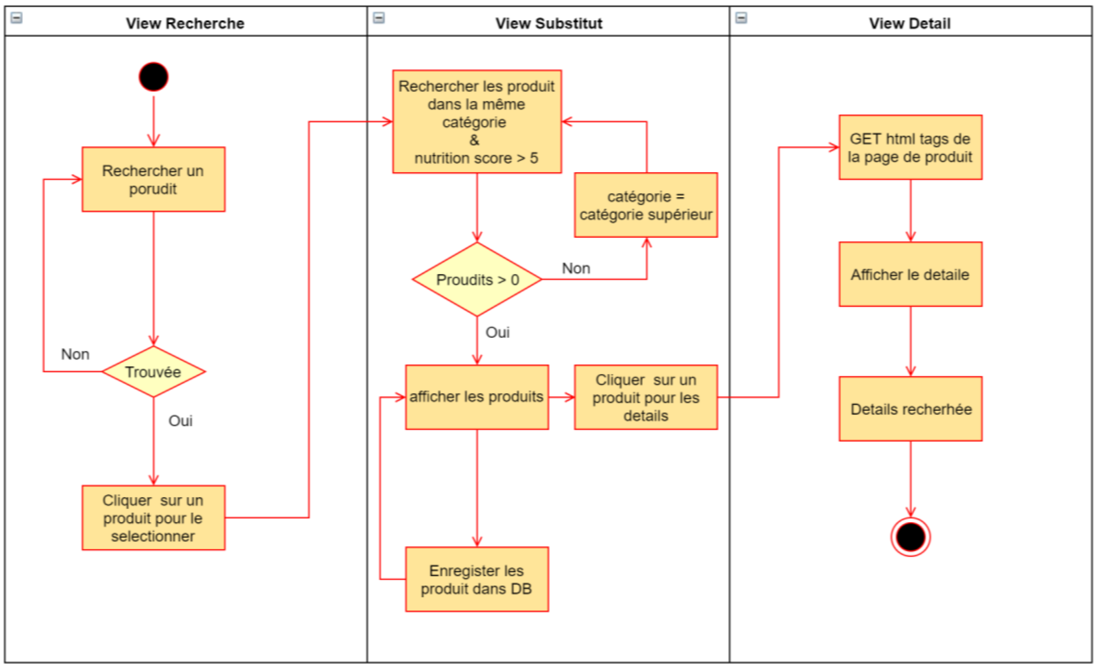
Utilities

The application is built with Django and PostgresSQL.

Django is an open source web framework for python. It offers wide range of features such as ORM, URL routing, MVC layout, Session handling, and Easy database migrations and many more. These ready-to-use features make web developing simple and fast but also powerful enough to handle large websites such as Instagram, Dropbox, Pinterest, Reddit and etc. Django has been rapidly gaining popularity for its ease of use and pragmatic design.

PostgreSQL is an open source RDBMS. It has been around for more than two decades. It has a very strong and active community that constantly improves existing features while strives for new cutting-edge features and security. It is supported by all major cloud service provider. Including Amazon, Google, and Microsoft.

Activity Diagram


View Recherche
View Substitut
View Detail
Rechercher un porudit
Non
Trouvée
Oui
Cliquer sur un produit pour le selectionner
Rechercher les produit dans la même catégorie & nutrition score > 5
catégorie = catégorie supérieur
Prouidts > 0
Non
Oui
afficher les produits
Cliquer sur un produit pour les details
Enregister les produit dans DB
GET html tags de la page de produit
Afficher le detaile
Details recherhée

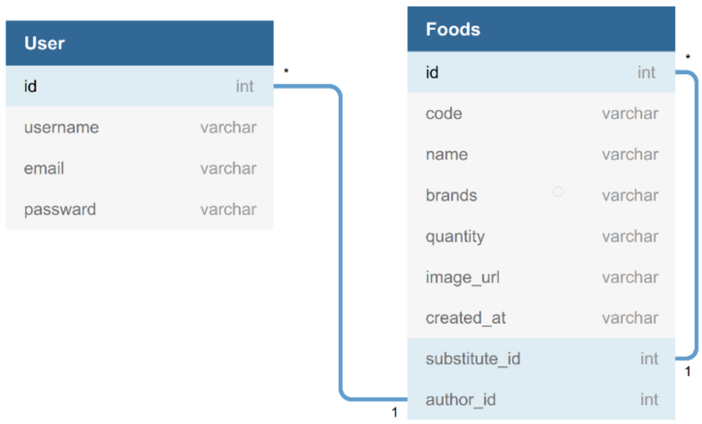
This diagram represents the flow of activities in three different views (search, substitute, and detail).

In search view, a user searches for a product to be replaced. The search view sends a HTTP request to *OpenFoodFacts API* which returns a set of data in JSON. Products are displayed on the template with a unique id by analyzing the JSON. After displaying all the products on the template. It stores the data to session to be used later. When user finds a right product and clicks on the product, it loads substitute page by creating an URL attached with the *id*.

The substitute function inside of the view gets the *id* from its URL and finds a matching product from the data in the session dictionary. The matching product has many attributes. One of them is called ‘*cateories_hierarchy’* which contains a list of categories the product resides. To find healthier replacement products, the view sends a request to get products that are in the same category as the selected product and has the nutrition score less than 5. Open Food Facts gives a product a nutrition grade in accordance to the nutrition score. The score less than 5 would return all the products with nutrition grade greater than ‘C’. If nothing is found, Substitute function repeats the search in the category superior until it returns finds products that meets the criteria. The user can save the product of his or her choice by clicking a save button located at the bottom of each product image. If user clicks on the image or the name of the product, the user is redirected to the detail page.

The detail view gets the product *id* variable from its URL and uses the *id* to sends a GET request directly to the OpenFoodFacts’ product page and gets the HTML source code of the page. Then the detail view parses the source code and finds the nutrition score and level for 100 g. This information is displayed on the template along with a button which directs the user to the original product page of OpenFoodsFacts.

Entity Relationship diagram



Only two database tables are used for this project. Django provides a built-in user table with a password hash field. This user table has a one-to-many relationship with a custom-built table called Foods. Foods table is responsible for storing data of all the products and its substitutes saved by users. In order to create the hierarchical relations between a product and its replacements, a one-to-many relationship is referenced to itself.

Challenges

The most challenging part of this project was working with CSS. I did not have a solid grasp of the language. Properties such as position, display, and flex were bit abstruse until I played with them multiple times to actually see how it displays on the page.

Data analysis was also challenging. It was very tedious job to read and find the set of data I need from the JSON file. On top of that, the data from OpenFoodsFacts were not consistent. There were some attributes with empty sting, null value, or didn’t exist at all. I had to compare different set of data from different products after I’d faced with several unexpected errors from different search results. I had to build a condition that would satisfy all the constraints and covert them into a uniformed object to be stored in the database.