STOP GASPI

COMMENCER A ARRETER DE GASPILLER

Sebastien Lludrigas

Projet réalisé dans le cadre de la formation développeur web de l'école O'clock

I. TABLE DES MATIERES

II. COMPETENCES DU REFERENTIEL

A. ACTIVITE TYPE 1

CP1 – MAQUETTER UNE APPLICATION

Après une séance de brainstorming pour savoir comment nous allions appréhender la création des wireframes de l'application nous avons décidé d'utiliser le logiciel en ligne Whimsical pour sa rapidité de prise en main, les éléments prédéfinis et également sa gratuité.

J'ai pris la main sur Whimsical pour réaliser tous les wireframes en étant guidé par tous les membres de l'équipe qui me faisait part de leurs points de vue du travail que j'étais en train de réaliser et de leurs désirs quant à l'UI de l'application.

Alexis s'est quant à lui occupé de rédiger la totalité des commentaires des wireframes.

CP2 – REALISER UNE INTERFACE UTILISATEUR WEB STATIQUE ET ADAPTABLE

À partir des wireframes nous avons déterminé avec Gregory, mon collègue Dev Front, les différentes étapes de l'intégration de l'application.

Contrairement à la tendance actuelle en responsive design qui s'oriente vers du Mobile-First, c'est-à-dire intégration d'abord pour le format mobile puis ensuite adaptation pour le format ordinateur de bureau, nous avons décidé de faire l'inverse Desktop-First donc et pour plusieurs raisons :

- Nous avons imaginé au départ l'application comme étant une application principalement desktop.
- Nous avons remarqué après avoir essayé que développer en Mobile-First puis faire l'adaptation pour le Desktop produisait un code CSS plus complexe, plus long et plus difficile à mettre en place.
- Nous avons jugé plus pratique de partir d'un contenu plus riche permis par le développement Desktop-First pour aller vers un contenu plus basique qu'exige le format mobile que l'inverse.
  En effet, retirer des éléments ou en simplifier certains pour s'adapter au format mobile une fois le développement Desktop terminé, nous a semblé plus simple efficace que de faire l'inverse, à l'image du proverbe "Qui peut le plus peut le moins".
- Nous avons remarqué que souvent, une adaptation d'un responsive Mobile pour le Desktop produisait un rendu plus pauvre pour ce dernier, cela découle peut-être de notre précédent point.

Il a été décidé que nous utiliserions la technologie SCSS pour les nombreuses fonctionnalités quelle apporte en plus au CSS classique ainsi que pour son utilisation plus intuitive liée au fait que l'arborescence du code est très proche de celle du code HTML.

La librairie Kawaï a aussi été sélectionné car elle correspondait bien à l'image que nous voulions donner de notre application, à la fois pratique et efficace mais aussi conviviale et mignonne avec une dose d'humour, car plus une nouvelle habitude est plaisante à implémenter plus la probabilité qu'on l'adopte définitivement sera grande.

CP3 – DEVELOPPER UNE INTERFACE UTILISATEUR WEB DYNAMIQUE

Afin de créer une interface utilisateur particulièrement dynamique et réactive nous avons cherché quels étaient les technologies front-end les plus performantes du marché à l'heure actuelle.

Si la plus utilisé est le framework Angular, nous avons finalement porté notre choix sur la librairie **React** qui offre l'avantage d'être modulaire - on peut donc commencer avec les fonctionnalités de base sans embarquer tout un ensemble de dépendances dont nous n'aurions pas besoin –

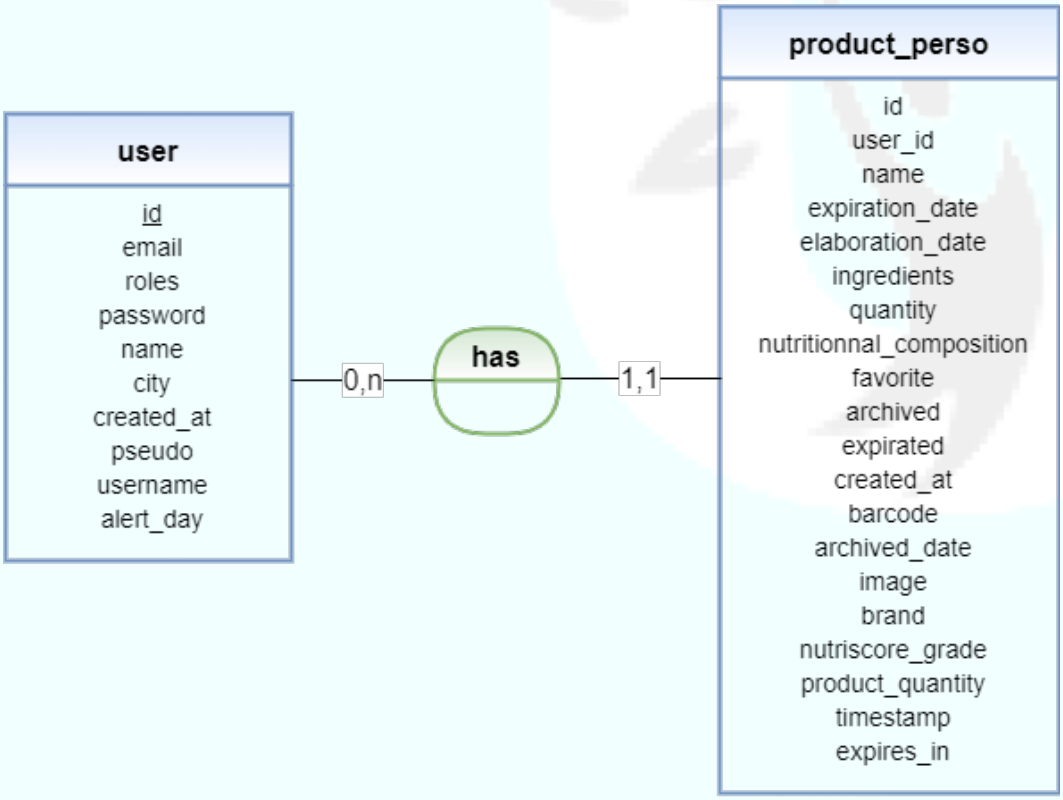
tout en étant très performante.

De plus, le sucre syntaxique JSX utilisé dans cette librairie, fusion en quelque sorte entre le langage Javascript, le HTML et même le CSS, nous a beaucoup plu.
Cette librairie a aussi l'avantage de permettre l'utilisation de Javascript natif sans ajouter de couches d'abstraction supplémentaire ce qui rend son utilisation plus intuitive et permet de voir vraiment ce qui se passe "Dans les coulisses" ce qui est très pratique lors de la recherche d'éventuels bugs.

B. ACTIVITE TYPE 2

CP5 – CREER UNE BASE DONNEES

Nous avons tout d'abord réfléchi au données dont nous aurions besoin pour réaliser l'application.
Nous avons donc fait un brainstorming pour assembler toutes les idées de l'équipe, en prenant en compte les fonctionnalités attendus.
Fort de ce recueil des données nous avons construit le MCD suivant :



Pour les cardinalités, comme on peut le voir sur le MCD, il a été décidé qu'un utilisateur aurait au minimum 0 produit et plusieurs au maximum, et un produit appartiendrait au minimum à un utilisateur et au maximum à un utilisateur également.
Cette cardinalité a été pensée ainsi car même si les produits de consommation que l'on achète sont les mêmes pour tous, les caractéristiques que va y ajouter l'utilisateur rendront ce produit unique.

Nous avons créé ensuite le dictionnaire des données qui permet de pouvoir choisir les types et les spécificités de chaque champ de chaque table.

Dictionnaire de données

user

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de notre utilisateur
email	VARCHAR(180)	NOT NULL, UNIQUE	Le mail de l'utilisateur
roles	LONGTEXT	NOT NULL	Le rôle de l'utilisateur
password	VARCHAR(255)	NOT NULL	Le mot de passe HASHE de l'utilisateur
name	VARCHAR(50)	NOT NULL	Le nom de l'utilisateur
city	VARCHAR(50)	NULL	La ville de l'utilisateur
created_at	DATETIME	NOT NULL, DEFAULT NEW DATETIME	La date de création de l'utilisateur
pseudo	VARCHAR(50)	NOT NULL	Le pseudo de l'utilisateur
username	VARCHAR(255)	NOT NULL, UNIQUE	Le mail de l'utilisateur
alert-day	SMALLINT(6)	NULL	Le jour d'alerte email de l'utilisateur

product_perso

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du produit
user_id	INT	FOREIGN KEY, NOT NULL, UNSIGNED	L'identifiant de l'utilisateur
name	VARCHAR(50)	NOT NULL	Le nom du produit
expiration_date	DATETIME	NOT NULL, DEFAULT NEW DATETIME	La date de péremption du produit
elaboration_date	DATETIME	NULL	La date d'élaboration/confection du produit
ingredients	LONGTEXT	NULL	Les ingrédients entrant dans la composition du produit
quantity	SMALLINT, UNSIGNED	NULL	La quantité de produit disponible
nutritional_composition	VARCHAR(255)	NULL	Le composition nutritionnelle
favorite	TINYINT	NULL	Si le produit est en favorite
archived	TINYINT	NULL, DEFAULT NEW DATETIME	
expirated	TINYINT	NULL	Si le produit est périmé ou non
created_at	DATETIME	NOT NULL, DEFAULT NEW DATETIME	La date de création du produit
barcode	VARCHAR(15)	NULL	La code barre du produit
archived_date	DATE	NULL	date d'archivage du produit
image	VARCHAR(255)	NULL	image du produit
brand	VARCHAR(55)	NULL	image du produit
nutriscore_grade	VARCHAR(1)	NULL	nutriscore
product_quantity	VARCHAR(255)	NULL	quantité du produit
timestamp	VARCHAR(50)	NULL	
expires_in	SMALLINT(6)	NULL	

La création de la base de données avec Symfony se fait directement avec la CLI.

Dans le fichier .env au niveau de la variable d'environnement DATABASE_URL on saisit les informations relatives à notre système de gestion de base de données afin que Symfony sache où il doit créer la base de données.

L'ORM **Doctrine** a été utilisé pour ce projet, il va permettre de faire le lien entre notre application et notre base de données en reflétant dans celle-ci les manipulations que l'on va faire sur nos objets.

La commande php bin/console doctrine:database:create va créer par l'intermédiaire de Doctrine notre base de données dans le SGBD que l'on a renseigné.

CP6 – DEVELOPPER LES COMPOSANTS D'ACCES AUX DONNEES

La connexion à la base de données est automatique avec Symfony comme dit plus haut à partir du moment où on a renseigné les informations relatives à notre SGBD dans la variable d'environnement DATABASE_URL du fichier .env

Chaque Entity, qui représente une classe dont nous avons besoin sera créée par l'intermédiaire de Doctrine en exécutant la commande php bin/console make:entity.

Cette commande créera également le Repository correspondant à l'Entity qui nous permettra de faire des sélections sur les données de la table représentée par l'Entity.

Dans les questions qui suivent l'exécution de cette commande on aura l'occasion de renseigner le type de chaque champ de notre table représenté dans l'Entity par des propriétés.

On exécutera ensuite la commande php bin/console make:migration qui va créer le fichier de migration dans lequel sera contenu les scripts SQL permettant de créer les tables.

Puis pour finir la commande php bin/console doctrine:migrations:migrate qui va lancer les scripts de migration et la création de nos tables.

Chaque Entity aura un Repository qui lui correspondra et servira à faire une sélection de données, le Read des quatre opérations de base de la persistance des données.
On le passera en tant que dépendance de la méthode qui en a besoin et on pourra ensuite l'utiliser dans cette même méthode.

Pour les opérations de manipulations des données, le Create Update et Delete, on utilisera le Manager de Symfony que l'on passera comme dépendance de la méthode qui en a besoin, comme pour le Repository et on pourra ensuite l'utiliser.

CP7 – DEVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB ET WEB MOBILE

Symfony utilise une variante du modèle MVC au niveau de son architecture.

Nous avons :

- Les **Controllers** qui permettent d'écouter une route en particulier, contrairement à une application en PHP natif qui utilise un front controller dans lequel est listé toutes les routes de l'application, dans Symfony chaque Controller est renseigné de la route qui le déclenche par l'intermédiaire d'annotations qui est un commentaire utilisant le système de docblock de PHP.
  On sait donc directement en regardant le code du Controller, l'url de la route qui le déclenche.
  Le Controller va donc :
  - Ecouter une route
  - Analyser la requête http que le navigateur a envoyé à cette route
  - Créer la réponse http adéquate

    - Renvoyer la réponse au navigateur
- Les **Entity** qui sont le reflet d'une table de la base de données
- Les **Repository** qui vont permettre de faire des sélections dans leur table correspondante.
  Ils vont être utilisé à l'intérieur de chaque Controller pour la récupération des données.
- Le **Manager**, qui va permettre de manipuler les lignes de la table dans laquelle il est utilisé.
  Il va être utilisé à l'intérieur de chaque Controller pour la manipulation des données.

C'est au niveau des Entity et des Repository que la différence se fait par rapport au MVC classique, on pourrait voir le Modèle du MVC comme la fusion des Entity et des Repository de Symfony.

La sécurité dans notre application est gérée par l'intermédiaire du SecurityBundle qui pourra être configuré dans le fichier security.yaml.

Dans ce fichier il y a différentes sections :

- Providers indiquera quel est le fournisseur d'utilisateur, Entity pour nous et quel est la propriété qui permettra de le récupérer, pour notre application ça sera l'email.
- Encoders permettra de configurer l'encodage des mots de passe.
- Firewalls permettra d'authentifier l'utilisateur via le Json Web Token .
- Access-control permettra de vérifier le rôle de l'utilisateur.

III. RÉSUME DU PROJET

L'application stopOgaspi a pour but d'aider les ménages à consommer de façon plus pragmatique et plus écologique, en cela elle s'inscrit profondément dans la philosophie du développement durable qui prends de plus en plus d'importance depuis quelques années.

Nous avons remarqué dans les propres comportements des personnes de l'équipe ainsi que dans celui de leurs relations, une perte quotidienne au niveau des denrées alimentaires.

En effet, lorsque nous achetons de la nourriture nous vérifions la date de péremption de chaque aliment, mais par la suite une fois les courses rangées, il nous faut fournir régulièrement un effort de mémoire et d'attention pour ne pas dépasser cette date.

Les préoccupations et les tâches récurentes de la vie quotidienne ont tôt fait de capter cette attention nécessaire et nous nous retrouvons chaque semaine avec plus ou moins de denrées alimentaires à jeter du fait du non-respect de leur date de péremption.

Il en va de même en ce qui concerne les plats faits maison.
Qui ne s'est pas déjà retrouvé avec un plat qu'il avait fait lui-même, stocké dans un tupperware au frigo, qui date de plusieurs jours et qui n'est manifestement plus

consommable.
Nous ne nous rappelons parfois même plus le jour où l'on avait préparé ce plat.

Quelle perte et quel manque de pragmatisme quant à la gestion de notre nourriture...

C'est ce constat qui nous a donné l'idée de développer l'application stopOgaspi.
Celle-ci doit nous permettre de pouvoir rapidement et facilement tenir la comptabilité de la nourriture que nous achetons et que nous préparons.

De plus, elle doit également nous permettre de nous libérer de la charge mentale qui accompagne le fait de garder en mémoire le moment où il faudra consommer tel ou tel plat.

Pour cela, il faudra qu'un système d'alerte mis en place au moment de l'enregistrement de la nourriture nous prévienne, un ou plusieurs jours à l'avance selon un paramétrage personnalisé, que l'aliment va bientôt ne plus pouvoir être consommé.

Ce faisant, nous pourrons décider soit de consommer l'aliment soit même de l'offrir à une autre personne, une chose est sûre c'est que nous ne jetterons plus cet aliment ou au moins nous ne le jetterons plus à cause de l'oubli de sa date de péremption.

Nous apporterons donc notre "part du colibri" selon la légende amérindienne racontée par Pierre Rahbi, à la diminution du gaspillage alimentaire mondial.

IV. CAHIER DES CHARGES

A. CONCEPTUALISATION DE L'APPLICATION

MINIMUM VIABLE PRODUCT

Le minimum viable product, comme le laisse entendre sa traduction littérale en français, est la version de l'application avec le minimum de fonctionnalités possibles pour quelle soit utilisable par une personne lambda.

Le but est de raccourcir au maximum un cycle de la boucle Construit-Mesure-Apprend afin de se confronter le plus tôt possible au retour du client pour en tirer des leçons empiriques qui nous permettront d'améliorer l'application.

Ne disposant que de 4 semaines pour développer l'application nous avons opté pour le **MVP**

Suivant :

- Pouvoir créer un compte utilisateur
- Pouvoir se connecter
- Pouvoir se déconnecter
- Pouvoir saisir des produits à la main
- Pouvoir scanner le code-barres des produits afin de faciliter la saisie de produit
- Pouvoir modifier les informations d'un produit existant

- Pouvoir supprimer un produit
- Avoir accès à un Dashboard personnalisé qui permettra :
    - de visualiser les informations de son profil et de les mettre à jour
    - de paramétrer son alerte mail
    - de supprimer son compte

SPÉCIFICATIONS FONCTIONNELLES

Les spécifications fonctionnelles de l'application sont les suivantes :

- Création d'un compte utilisateur
- Connexion à un compte utilisateur
- Ajout d'un produit selon 3 modes possibles :
    - En scannant le code-barres d'un produit via la caméra d'un smartphone ou la webcam d'un ordinateur
    - En entrant directement le code-barres dans un champ dédié
    - En saisissant à la main un produit dans le cas d'un plat fait maison
- Saisie de la date de péremption du produit enregistré
- Modification de la date de péremption du produit enregistré
- Modification de la quantité du produit enregistré
- Suppression du produit enregistré
- Consultation du nutri-score du produit enregistré
- Mise à jour des informations personnelles
- Paramétrage de la date à laquelle l'utilisateur voudra recevoir une alerte par mail qui lui indiquera qu'un produit en particulier est sur le point d'être périmé
- Suppression du compte utilisateur

ARBORESCENCE DE L'APPLICATION/WORKFLOW

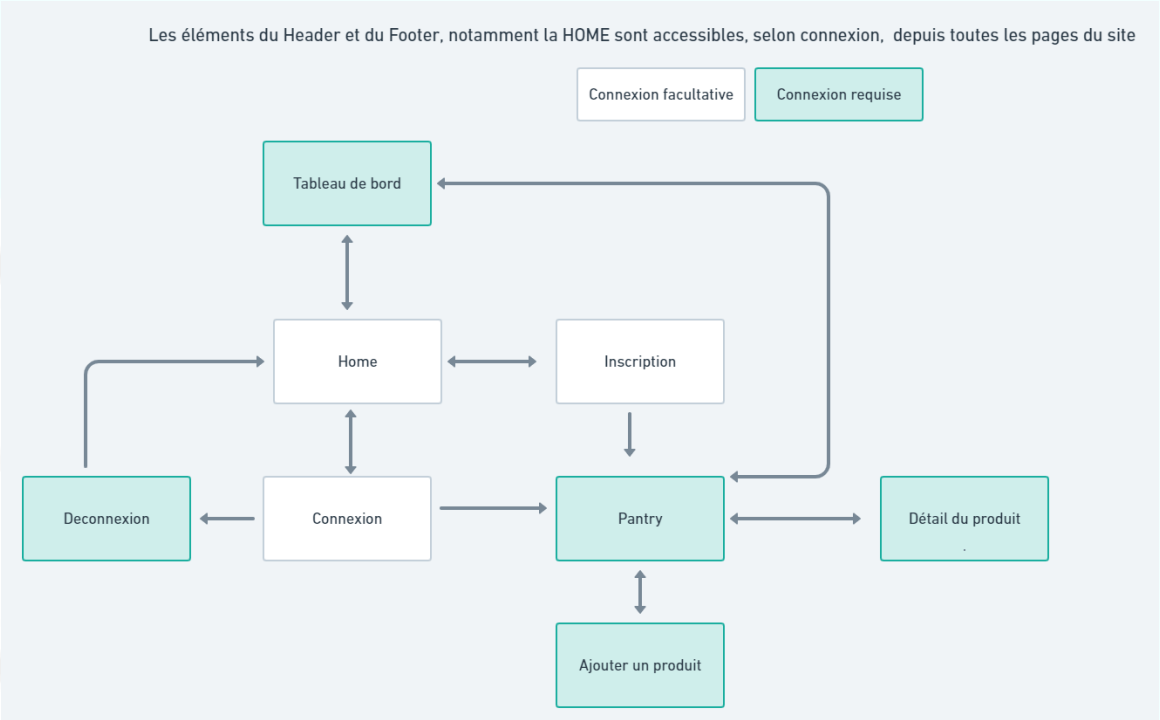
L'utilisation de l'application en tant que visiteur non-inscrit et/ou non-connecté permettra d'accéder à :

- La page d'accueil
- La page de connexion
- La page d'inscription

L'utilisation de l'application en tant que membre déjà inscrit permettra d'accéder à :

- La page de déconnexion
- La page du Pantry (garde-manger) et donc aux détails d'un produit
- Les pages d'ajout d'un produit
- La page du tableau de bord

Schéma du workflow :



B. MISE EN PLACE DE L'APPLICATION

WIREFRAMES

Les wireframes ou maquette fonctionnelle de l'application permettent de visualiser l'articulation entre les différentes pages de l'application.

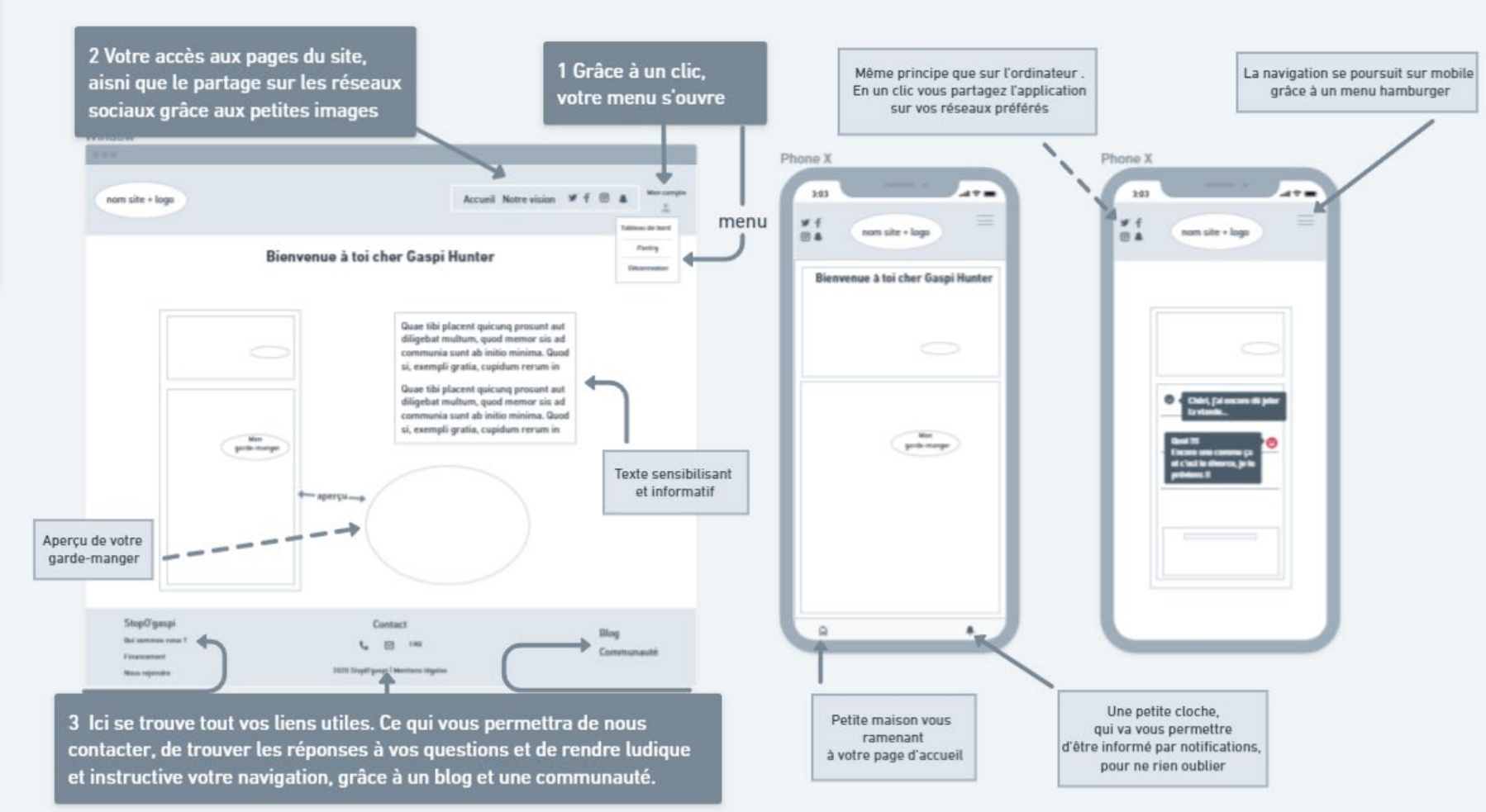
Réalisés en équipe, ils ont permis de concrétiser et de synthétiser d'une manière visuelle les différents points de vue que chaque membre de l'équipe avait sur ce que devait être l'application.

Il était par ailleurs entendu que ces wireframes n'étaient un premier jet de l'idée que se faisait le groupe de ce à quoi devrait ressembler l'application, nous savions que nous aurions sûrement à modifier certaines frames au cours du développement et éventuellement en enlever ou en rajouter, ce qui s'est d'ailleurs avéré exact.

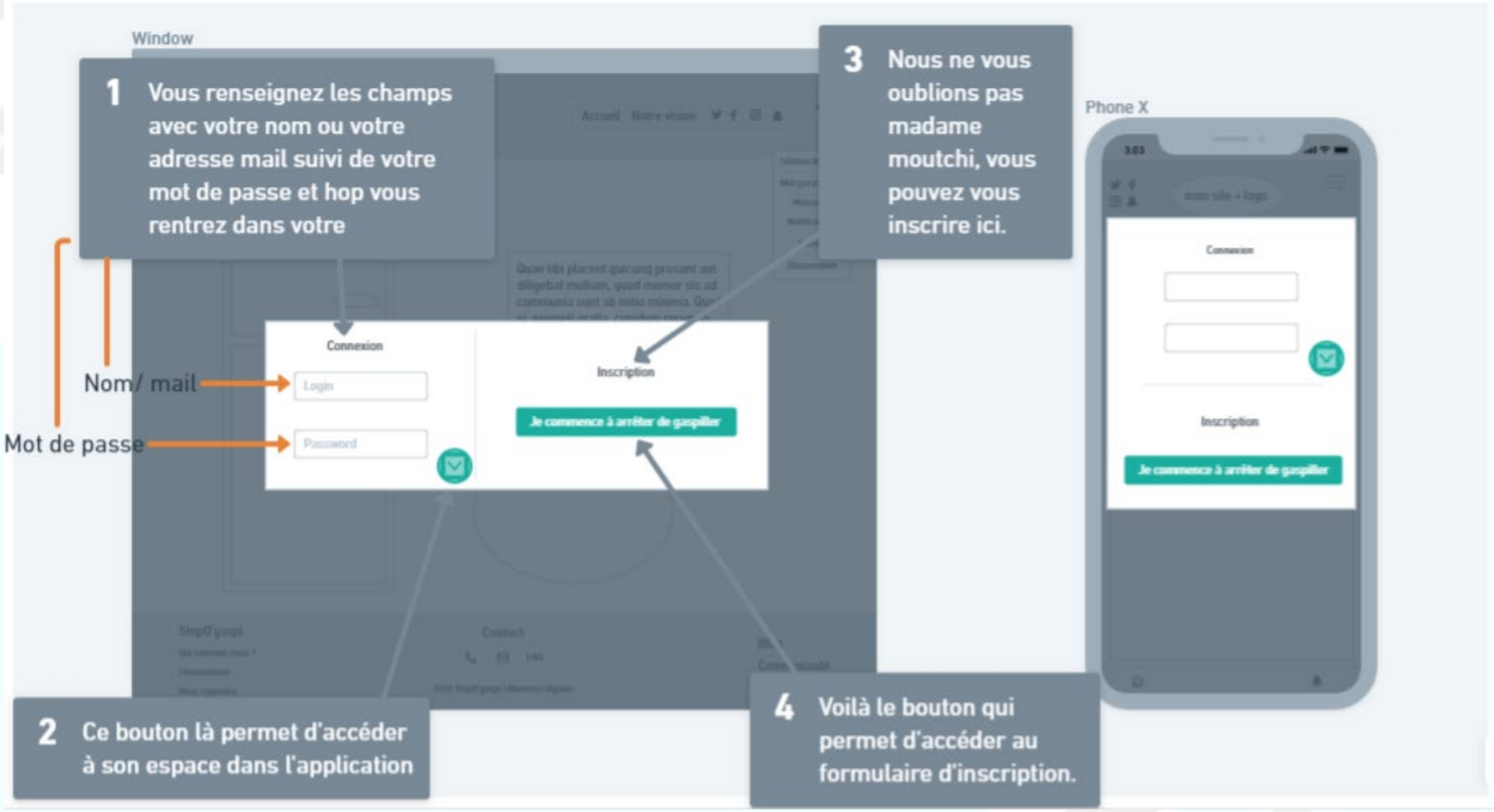
Ce n'est pas surprenant car un wireframes est une représentation basse-fidélité de l'application finale à la différence de la maquette ou même du prototype (maquette dynamique), qui eux sont des représentations hautes fidélité de l'application finale.

Voici les différents wireframes réalisés :

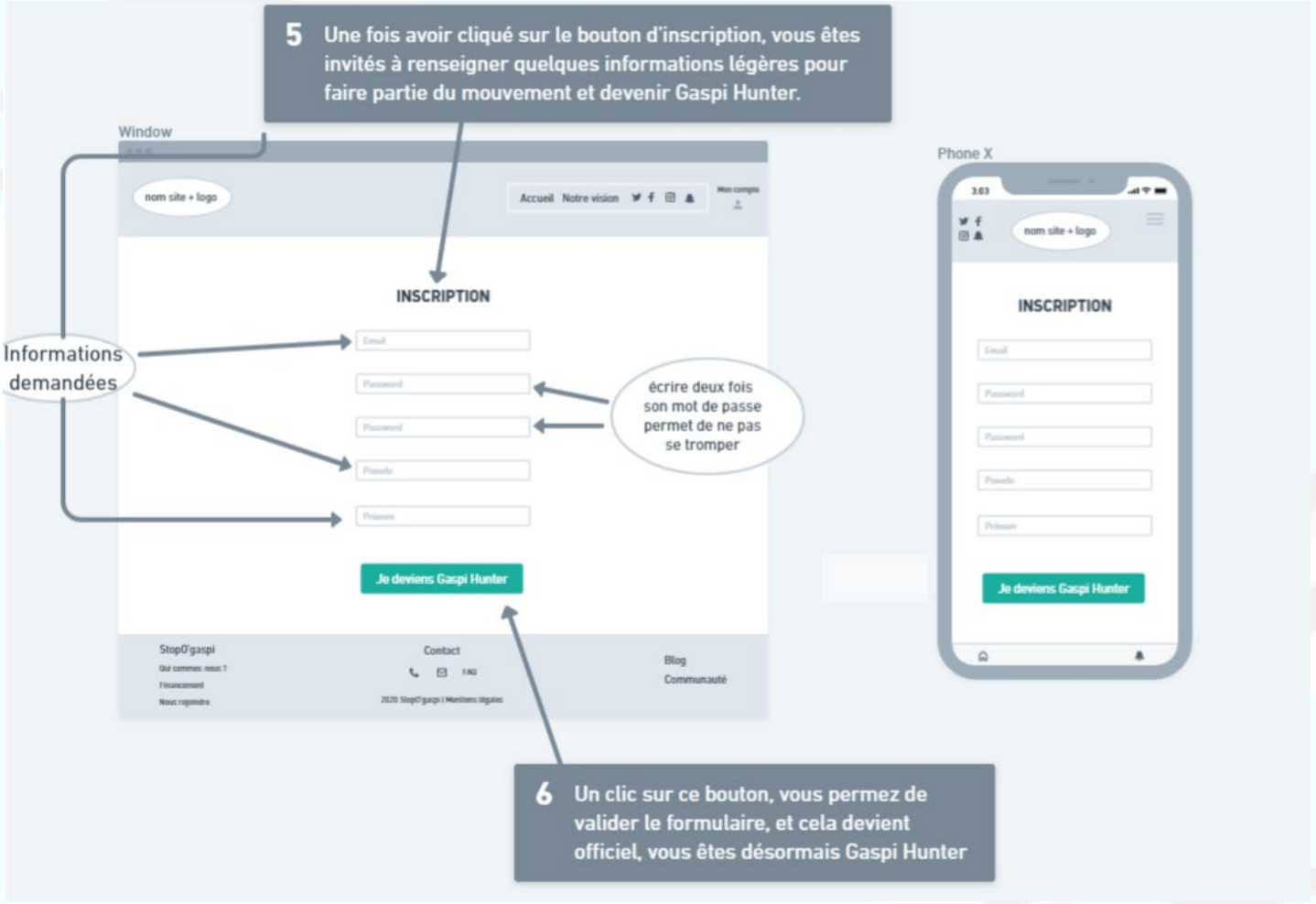
Page d'accueil



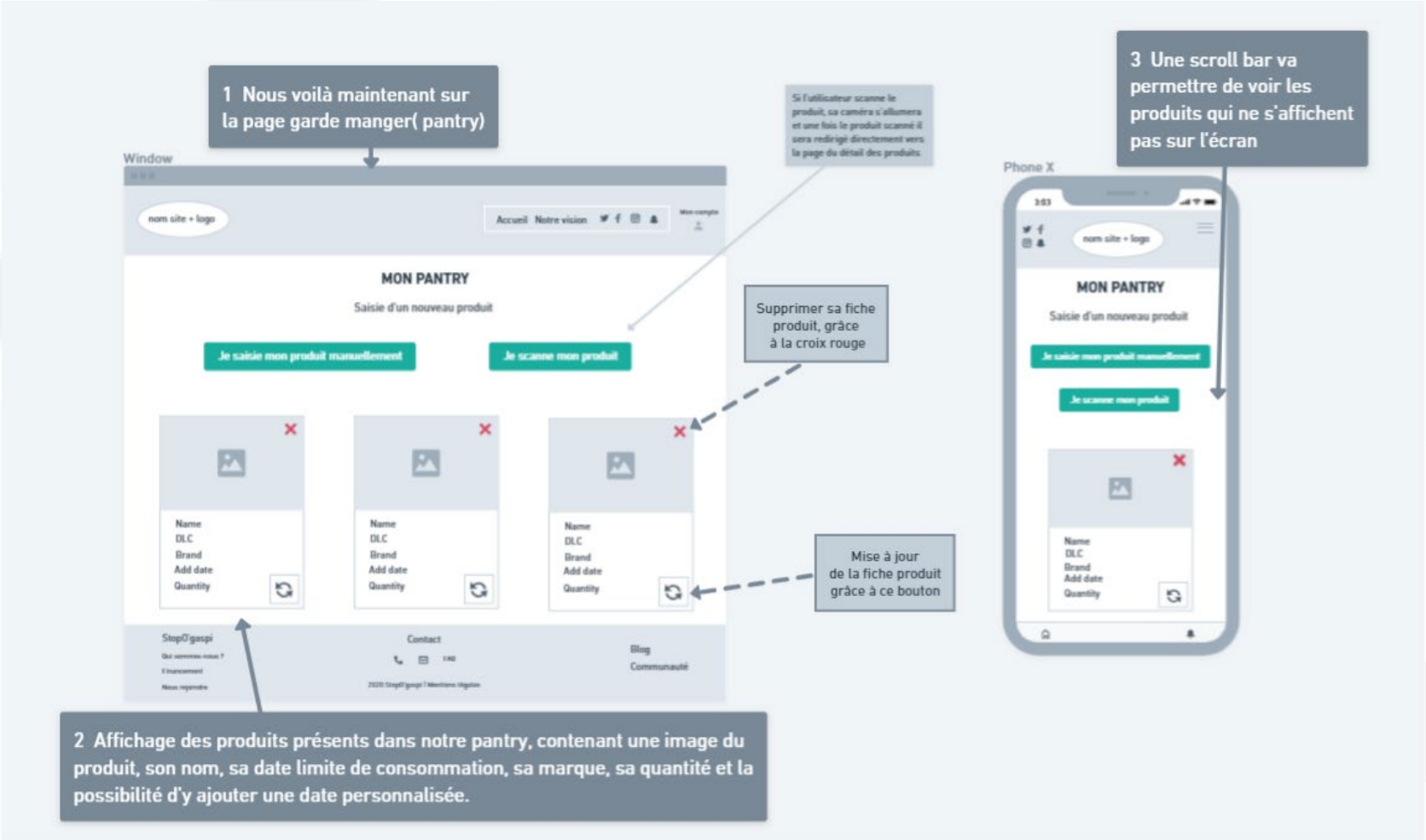
Page de connexion



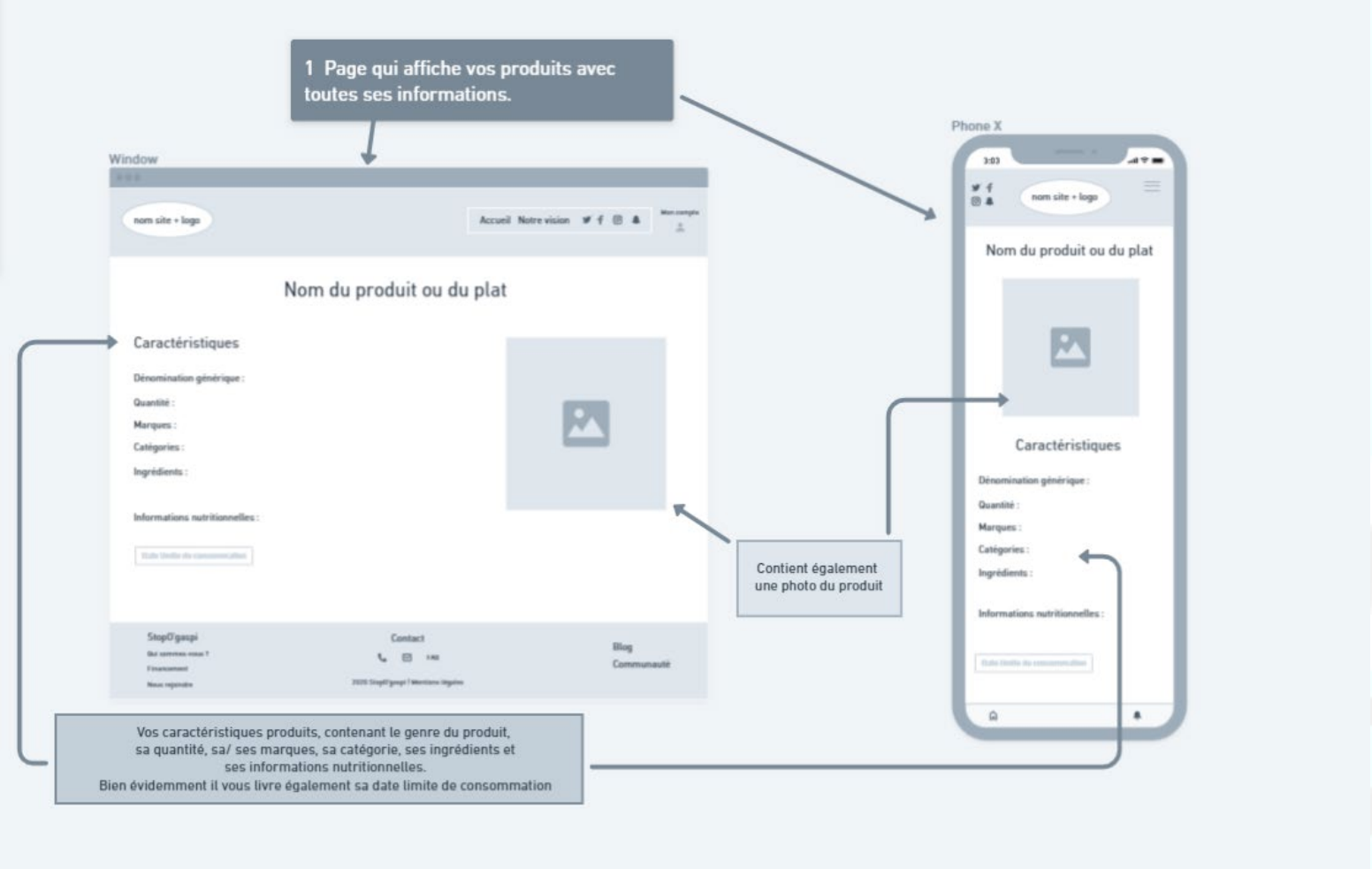
Page d'inscription



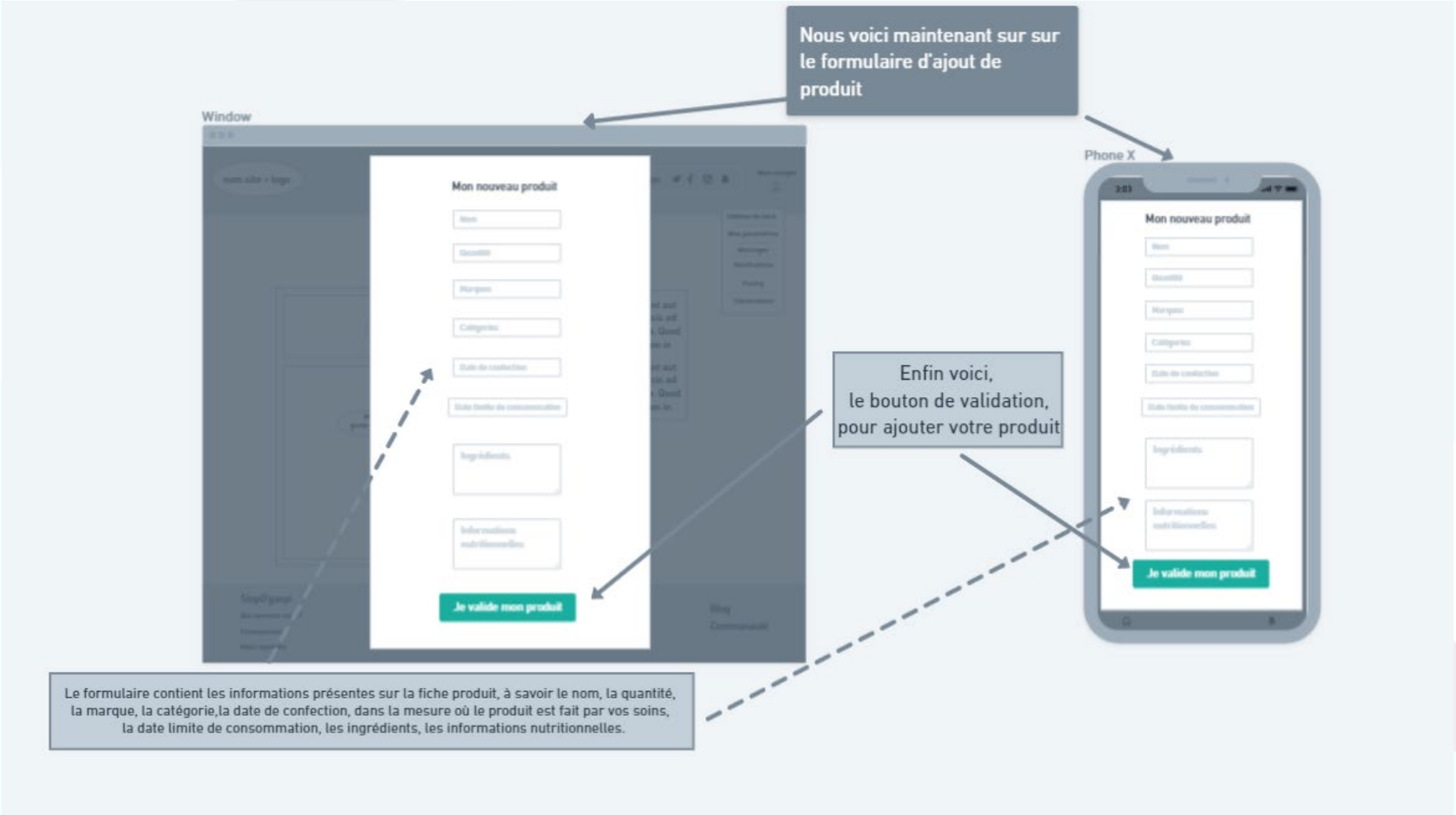
Page Pantry (garde-manger)



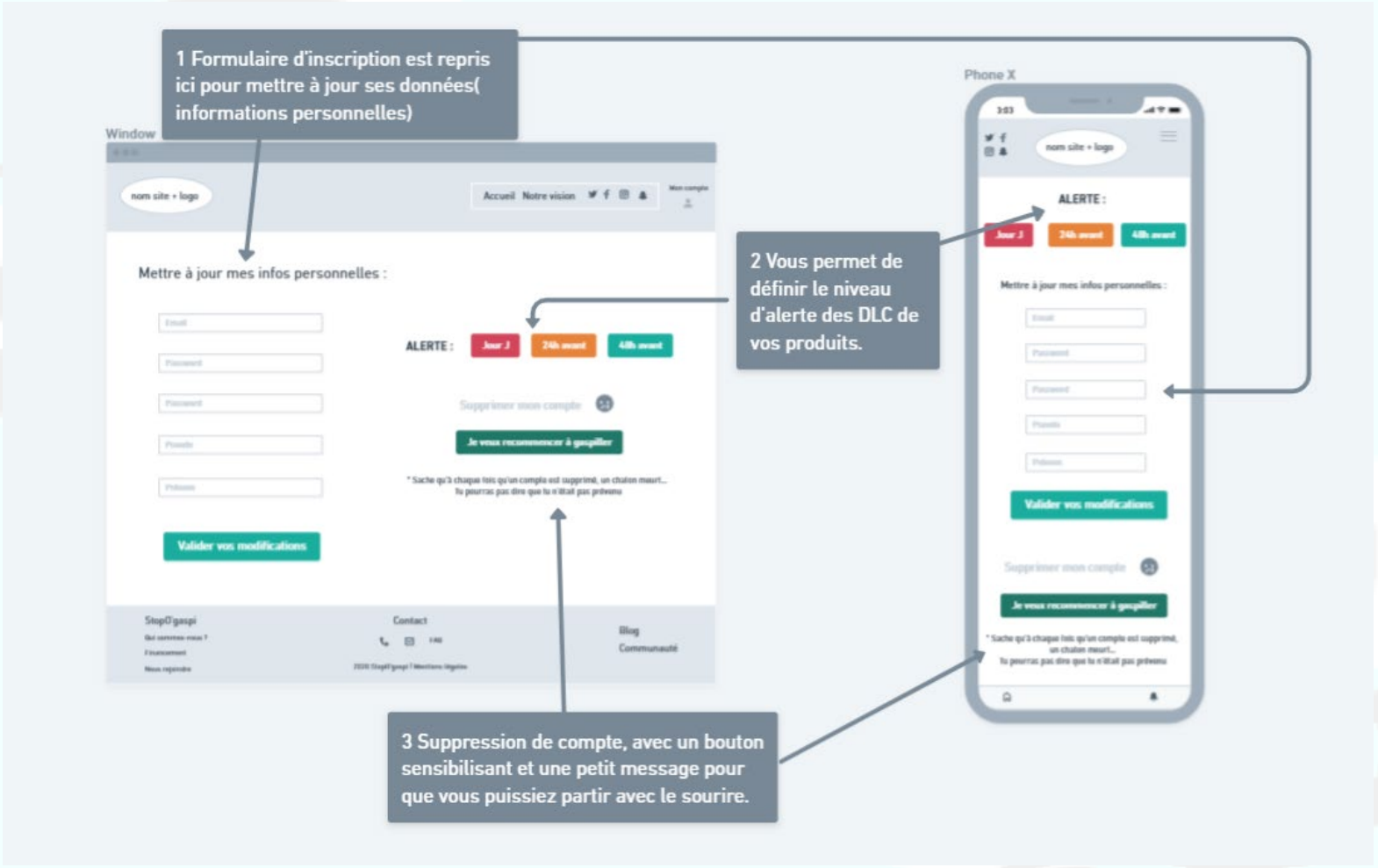
Page fiche produit



Page d'ajout d'un produit



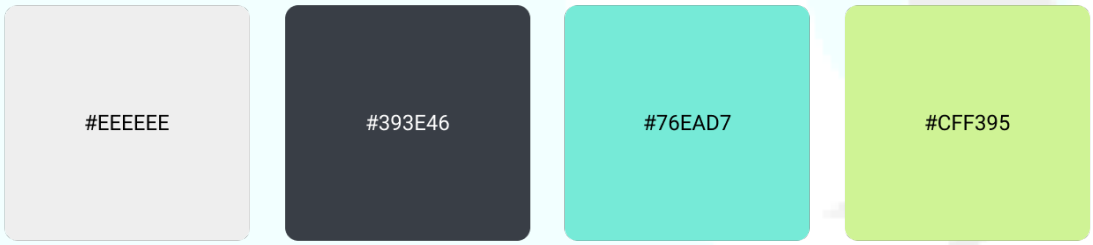
Page tableau de bord



CHARTE GRAPHIQUE

La charte graphique permet de maintenir une cohérence visuelle du site.
Elle permet aussi de développer plus rapidement car à chaque fois que l'on code le style d'un nouveau composant on utilise les variables SCSS qui contiennent les couleurs de la charte.

Après une nouvelle séance en équipe où l'on a comparé nos idées et fait des essais en colorisant certains des wireframes réalisés, nous sommes tombés d'accord sur cette charte graphique :



C. UTILISATEURS ET USER STORIES

PUBLIC VISÉ

La cible de notre application est relativement large car sa vocation est justement d'impacter le plus grand nombre et l'alimentation concerne tout un chacun, cependant nous pensons que les enfants et les adolescents seront surement le public le moins concerné car la plupart du temps ce n'est pas ce public qui s'occupe de la gestion de l'alimentation et de la préparation des repas dans un foyer.

Nous avons tout de même créé une application dont l'interface utilisateur peut séduire les enfants, grâce au design Kawaï et à la touche humoristique notamment.

On peut tout à fait imaginer un enfant préparant un gâteau avec ses parents et utilisant l'application afin de saisir sa date de fabrication ou bien rangeant les courses avec les parents et s'occupant d'entrer chaque aliment dans l'application avec le scanner de code-barres.

Malgré tout, nous avons répertorié le type de public suivant comme étant le principal :

- La ménagère (homme ou femme).
- Le foyer familial.
- Les citoyens faisant partie d'un mouvement écologiste, les végétariens, les végans.

USER STORIES

Les user stories ou scénarios de utilisateurs permettent de lister les fonctionnalités d'une application du point de vue de l'utilisateur final, elles expliquent comment une fonctionnalité va lui apporter de la valeur.

Nous avons utilisé l'application Trello qui permet grâce au système de tableau Kanban de gérer de façon intuitive les différentes user stories, nous reparlerons plus loin du type de méthode de gestion de projet que nous avons adopté qui justifie l'utilisation de ces tableaux.

Pour l'instant nous avons cette liste de user stories :

User stories

En tant que	Je veux	Afin de (si besoin/nécessaire)
visiteur	je veux pouvoir accéder à la page d'accueil	-
visiteur	je veux pouvoir m'inscrire en renseignant mon mail et un mdp	-
visiteur	je veux pouvoir me connecter	-
utilisateur	je veux pouvoir ajouter un produit à l'aide du scanner de code-barres	-
utilisateur	je veux pouvoir ajouter un produit en entrant son code-barres manuellement	-
utilisateur	je veux pouvoir ajouter un produit entièrement à la main	-
utilisateur	je veux pouvoir modifier un produit	-
utilisateur	je veux pouvoir supprimer un produit	-
utilisateur	je veux que mes produits soient triés par date de péremptions croissante	-
utilisateur	je veux voir facilement l'avancement de peremption de mes produits (barre coloré, icone ou autres)	-
utilisateur	je veux être averti par mail de l'approche de la péremption des mes produits	-
utilisateur	je veux pouvoir regler le moment de l'alerte (J-2, J-1 ou jour J)	-
utilisateur	je veux pouvoir modifier mes informations personnelles	-
utilisateur	je veux pouvoir supprimer mon compte	-
utilisateur	je veux pouvoir modifier le trie de mes produits	-
utilisateur	je veux pouvoir trier mes produits par categories	-
utilisateur	je veux pouvoir créer ma liste de courses	-
utilisateur	je veux pouvoir visualiser la barre de progression de la date de péremption de mes produits	-
utilisateur	je veux avoir un compte rendu de mon gaspillage	-
utilisateur	je veux être informé de la dangerosité de la consommation de mon produit après date de peremption	-
utilisateur	je veux savoir quoi faire de mes produits périmés	-
utilisateur	je veux recevoir un avertissement si un autre utilisateur à eu un probleme grave sur un produit de mon pantry	-

V. SPÉCIFICATIONS TECHNIQUES

A. VERSIONNING

Pour la totalité du projet nous avons décidé d'utiliser le logiciel de gestion de version Git ainsi que le site GitHub qui permet de centraliser son code et le rendre accessible à l'ensemble de l'équipe.

Afin que le travail en équipe puisse se faire de manière cohérente et productive, nous avons décidé de mettre en place un workflow Git auquel se tenir tout le long du développement du projet.

Il existe de nombreux workflows Git déjà disponibles certains relativement complexes, cependant nous avons décidé de rester le plus simple possible ceci afin de ne pas ajouter une couche de complexité supplémentaire à notre projet ce qui aurait eu un impact certains sur la tenue des délais qui nous étaient imposés.

Nous nous sommes néanmoins inspiré des bonnes pratiques existantes sur la gestion de version Git pour mettre en place notre propre workflow qui est le suivant :

- Ne jamais coder directement sur la branche Master
- Pour le développement de chaque fonctionnalité, une branche sera créée qui portera le nom du développeur suivi du nom de la fonctionnalité
- Créer un commit pour chaque étape du développement de la fonctionnalité
- Dés qu'une fonctionnalité est terminée on "push" notre branche sur GitHub
- On créer une Pull Request afin que le Git Master de notre équipe puisse réviser le code
- Le Git Master fait alors un merge de la branche de fonctionnalité sur la branche Master
- Il fait les corrections appropriés au niveau du code sur l'interface du site GitHub si jamais il y a des conflits

B. TECHNOLOGIES DU BACK

Le framework **Symfony** dans sa version 4 a été utilisé pour le back-end accompagné de nombreux bundles tel que :

- **Doctrine** : ORM (object-relational mapping) est une interface qui se place entre la base de données et l'application et qui permet de manipuler une base de données à partir d'objets PHP.
  Il a l'avantage de fournir un ensemble de méthode qui permettent de faire des requêtes à la base de données sans avoir à utiliser le langage SQL.
- **Twig** moteur de template permettant de gérer les vues de l'application
- **EasyAdmin** permettant de mettre en place un back-office rapidement

- **Swift Mailer** permettant de gérer l'envoi de mail.

C. TECHNOLOGIES DU FRONT

Pour le front les deux librairies principale qui ont été utilisée est **React** ainsi que **Redux**.

Accompagnées d'autres librairies plus secondaire tel que :

- **Quagga** qui permet d'implémenter un scanner de code-barres
- **Axios** permettant de gérer plus facilement les requêtes HTTP
- **Classnames** permettant de gérer les classes conditionnelles
- **React Kawaï** qui permet d'utiliser des composants d'illustrations au format svg
- **Sass** et **sass-loader** permettant d'utiliser le SCSS avec webpack

D. SECURITE

Nous avons utilisé le **SecurityBundle** pour la sécurité globale de l'application côté back.

Le **NelmioCorsBundle** pour la gestion des CORS.

Le **LexikJWTAuthenticationBundle** pour la gestion des Json Web Token.

Côté front aucune dépendance n'a été utilisé en particulier, la gestion de la sécurité se faisant uniquement côté back.

E. AUTRES SERVICES

Pour le déploiement des deux parties de notre application nous avons utilisé :

- IONOS société d'hébergement web
- Le logiciel FileZilla permettant de transmettre nos fichiers à nos hébergeurs via le protocole FTP
- o2switch société d'hébergement web

VI. ORGANISATION DE L'EQUIPE

A. PRÉSENTATION DE L'EQUIPE, REPARTITION ET DETAILS DES ROLES

- Laurie Reinette la **Product Owner**, est la responsable du produit, de l'application, elle représente le client et a une vision d'ensemble du projet.

  C'est la décisionnaire finale en ce qui concerne les choix à faire pour la réalisation de l'application car elle a la connaissance des besoins métiers.

- Alexis Bertrand le **Scrum Master**, le responsable et le facilitateur du projet, il s'assure que le framework Scrum est bien respecté.
  C'est une sorte de coach qui permet aux différents membres de l'équipe d'appliquer la méthode Scrum.
  Il est en charge de l'animation du Daily Scrum et gère l'outil de suivi Trello pour notre équipe.
- Gregory Broyer le **Git Master**, c'est lui qui gère le versionning du projet, il s'occupe notamment de vérifier les Pull Requests et de faire les merge sur la branche Master.
- Aurélien Romain le **Lead dev back**, c'est lui qui pilote le développement du back-end, il est le responsable du bon fonctionnement de cette partie.
- Sebastien Lludrigas le **Lead dev front**, c'est lui qui pilote le développement du back-end, il est le responsable du bon fonctionnement de cette partie.

B. ORGANISATION DU TRAVAIL

Pour l'organisation du travail nous avons décidé d'adopter le type **semi -itératif** qui est la base utilisée dans les **méthodes agiles**.

Ce type de méthode permet de réaliser des cycles cours de développement, des "itérations", qui permettent de tester régulièrement les fonctionnalités afin d'éviter les dérives du projet.

Parmi les méthodes agiles notre choix s'est portée sur le framework **Scrum**.

Ce cadre de travail permet d'organiser le développement d'un projet sous forme de **Sprint** qui sont des cycles de développement d'une durée déterminé et qui ne change pas au cours du sprint.
Cela permet de donner une base à l'équipe pour pouvoir cibler plus précisément au fil des sprints ce qu'il est réellement possible de faire en un temps donné.

Nous avions 4 semaines pour réaliser notre projet nous avons donc divisé ce temps en 4 sprints d'une semaine chacun.

Au début du projet nous avons mis en place le **Product Backlog** qui permet de regrouper toutes les tâches à réaliser au cours du projet.

Au début de chaque cycle une réunion appelée **Sprint planning** permet de créer un document – **le Sprint Backlog** - qui regroupera la totalité des tâches à réaliser pendant le sprint.

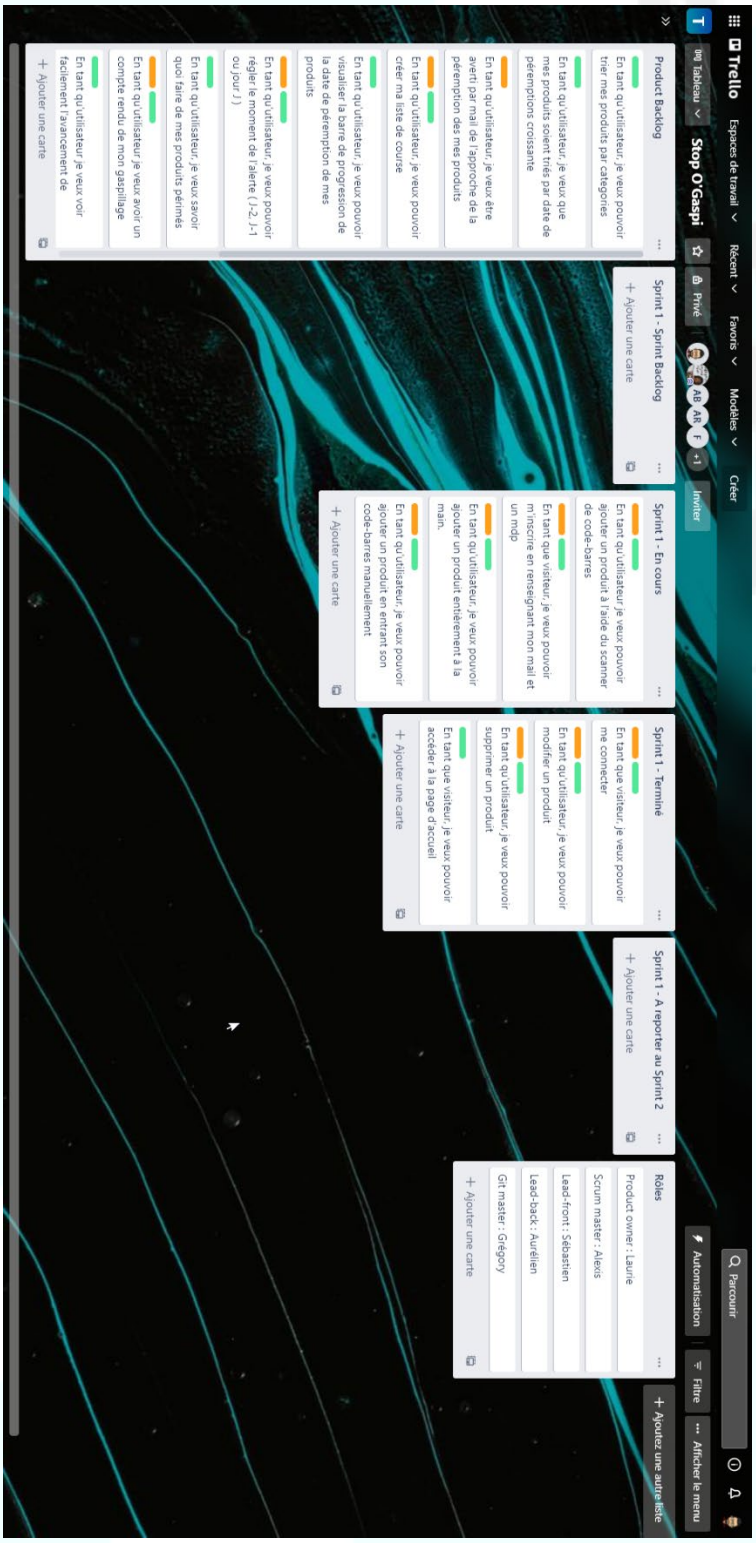
Ensuite, tous les matins se tiendra une réunion d'une quinzaine de minutes animée par le Scrum Master – le **Daily Scrum** – où chaque membre de l'équipe fera un résumé de ce qu'il a fait la veille, une prévision de ce qu'il fera aujourd'hui et les éventuels points durs qu'il rencontre.

À la fin de chaque sprint se déroule une réunion - la **Sprint Review** – qui permet de résumer le déroulement du sprint, en parlant des tâches du Backlog qui ont été réalisées et celles qui ne l'ont pas été, des points durs rencontrés et de l'état actuel du Backlog à la fin du sprint.

Pour finir le cycle une **Sprint Retrospective** est organisée qui permet d'avoir une vision haute sur le sprint précédent en analysant ce qui s'est bien passé et mal passé afin de mettre en place un plan d'amélioration pour les sprints suivants.

Pour gérer cette organisation nous avons utilisé Trello qui met à disposition des tableaux Kanban pouvant être modifié par chacun des membres de l'équipe.

Ces tableaux organisent la gestion de chaque sprint sous forme de colonnes chacune représentant l'état des users stories.
Voici une capture d'écran du tableau que nous avons utilisé durant le sprint 1 :



C. OUTILS UTILISÉS

Les outils utilisés par notre équipe tout au long du développement du projet sont les suivants :

- **Visual Studio Code** :
  Éditeur de code open source personnalisable grâce à une grande quantité de plugins que l'on peut ajouter selon ses besoins.
  Il permet en outre de faire du pair programming ce qui s'est avéré très pratique durant notre projet plus particulièrement lorsque un débogage nécessitait l'aide d'un autre membre de l'équipe en direct.

- **Google Chrome, Mozilla Firefox** :
  Navigateurs web permettant d'afficher la compilation de notre code.
  Ces deux navigateurs offrent en outre des outils de développement très poussés qui facilitent notamment le débogage du code.

- **Git** :
  Logiciel de gestion de version déjà mentionné plus haut qui nous a permis de faire le versionning de notre code**.**

- **GitHub** :
  Site web que nous avons utilisé comme serveur distant pour centraliser et sauvegarder le code source de l'équipe, son interface nous a aussi permis de gérer plus facilement les Pull Requests régulières ainsi que les merge sur la branche master.

- **Discord** :
  Plateforme de messagerie instantanée qui nous a permis de faire nos réunions quotidiennes de Daily Scrum et de faire nos partage d'écrans.

- **Slack** :
  Plateforme de travail collaboratif dans lequel nous avions différents espaces de travail attribués par notre centre de formation.
  Nous a permis de communiquer régulièrement avec nos professeurs ainsi qu'avec les autres étudiants de notre promotion.

D. PROGRAMME DES SPRINTS

SPRINT 0 :

- Création du cahier des charges

- Répartition des rôles individuels inhérents à la méthodologie Scrum
- Création des documents relatifs à la base de données : MCD, MLD, dictionnaire de données.
- Création des wireframes
- Création du trello
- Carnets de bord : permet de conserver un journal de nos réalisations personnelles, des bugs rencontrés et des solutions trouvés, liens vers les ressources utilisées.

SPRINT 1 & 2 :

- Travail sur le code proprement dit
- Redéfinition des enjeux au fur et à mesure que le travail avance afin de réussir à livrer le MVP prévu à la fin des 4 sprints

SPRINT 3 :

- Pas d'ajout de nouvelles fonctionnalités
- Finir les fonctionnalités en cours
- Tests des fonctionnalités (recettage)
- Mettre en production
- Préparation pour la démonstration finale de fin de projet

VII. PRODUCTION PERSONNELLES

A. DÉROULÉ DES SPRINTS

SPRINT 0

J'ai commencé par rechercher les différentes fonctionnalités possibles de notre application :

- Proposer à l'utilisateur des recettes contenant les produits qui arrive à leur dernier de date de péremption.
- Mettre en place une barre de progression qui changerait de couleur au fur et à mesure que la différence entre la date de péremption du produit et la date courante diminuerait :
    - Vert -> plus de 1 jours
    - Orange -> 1 jours
    - Rouge -> 0 jours, donc produit à consommer le jour même
- Gamifier l'application afin de la rendre plus ludique : quid des fonctionnalités permettant cette gamification ?

J'ai ensuite recherché les différentes applications existantes qui répondait au même besoin auquel devait répondre stopOgaspi, afin de pouvoir nous inspirer de leurs fonctionnalités ainsi que de leur interface utilisateur.

Mon idée était également d'aller voir les reviews de ces applications sur le Google Play Store afin de repérer les doléances des utilisateurs.
Cela me permettrait d'avoir des idées pour intégrer dans stopOgaspi des fonctionnalités qui manque à ces autres applications et également d'éviter les défauts relevés par ces utilisateurs.

Voici une liste non-exhaustive des applications que j'ai trouvées :

- À Consommer - Suivi Aliments
  https://play.google.com/store/apps/details?id=com.peytu.bestbefore&hl=fr

- Expiration de produits
  https://play.google.com/store/apps/details?id=com.bfp.apps_expiring_things&hl=fr

- Food Checklist - Groceries Expiration and Shopping
  https://play.google.com/store/apps/details?id=com.chestersw.foodlist&hl=fr

- My Kitchen: Expiry Dates, Shopping List, Pantry
  https://play.google.com/store/apps/details?id=com.steljoy.smartfridge

- Reeking: date de péremption des produits
  https://play.google.com/store/apps/details?id=com.zerobased.reeking

J'ai également commencé à rechercher une API qui permettrait de récolter des informations sur un produit à partir de son code-barres.

J'ai retenu le site Open Food Facts – France https://fr.openfoodfacts.org/ qui propose une reconnaissance de produits sur son site mais également une API dont on pourrait se servir dans notre application.

Le fonctionnement de l'API est relativement simple, il suffit d'ajouter le code-barre du produit dans l'URL de l'API pour récupérer toutes les informations au sujet de ce produit.

J'ai ensuite commencé ma recherche d'une librairie qui permettrait d'implémenter un scanner de code-barre dans notre application.

En effet, lors d'une de nos réunions de groupe du sprint 0 l'idée du scanner de code-barre a émergée car sur beaucoup des applications dont j'ai mis le lien plus haut cette fonctionnalité était présente et était plébiscité par les commentaires sur le Google Play Store.

Il a été décidé que je me chargerai de l'implémentation de cette fonctionnalité qui pourrait ajouter une valeur ajoutée non négligeable à notre projet.

À ce moment là je n'étais pas du tout sûr d'avoir les compétences pour implémenter cette fonctionnalité.

J'ai donc en première recherche trouvé les librairies et ressources suivantes :

- Librairie react-barcode-reader :
  https://www.npmjs.com/package/react-barcode-reader

- Librairie expo-barcode-scanner :
  https://docs.expo.io/versions/latest/sdk/bar-code-scanner/

  que j'ai pu tester sur CodeSandbox :

  https://codesandbox.io/s/l2kq076x3m?file=/src/index.js

- Librairie QuaggaJS :
  https://serratus.github.io/quaggaJS/

- Librairie React Webcam :
  https://github.com/mozmorris/react-webcam

J'ai également effectué des recherches Google en anglais avec cette question :

How to integrate a barcode reader on a React app ?

Plusieurs résultats ont retenu mon attention :

- https://medium.com/@yushulx/how-to-build-web-barcode-scanner-using-react-and-webcam-d36ba26bddfd
- https://www.dynamsoft.com/codepool/web-barcode-scanner-react-webcam.html
- https://www.npmjs.com/package/react-barcode-reader
- https://stackoverflow.com/questions/53857109/barcode-scanner-for-a-react-application

À ce moment là je n'étais pas encore décidé sur une librairie en particulier, il fallait encore que je teste ces différentes librairies dans le code React de base que nous n'avions pas encore créé, rendez-vous donc au sprint 1 pour commencer à faire ces tests.

Une réunion avec Gregory nous a permis de déterminer les étapes de codage du front.
Il a été décidé que Gregory s'occuperait plus du développement du design de l'application via le SCSS et la librairie Kawaï, quant à moi je consacrerai la majeure partie de mon temps à l'algorithmie et à la mise en place de l'architecture générale de React et de Redux.

Nous avons également tout au long de ce sprint 0 réalisé avec toutes l'équipe :

- Le cahier des charges
- Les documents relatifs à la base de données : MCD, MLD, dictionnaire de données.
- Les wireframes
- Le tableau Kanban en utilisant Trello

À la fin de ce premier sprint nous avons fait notre première Sprint Retrospective en présentant à toute la promotion le travail effectué.

SPRINT 1

Après la réalisation du Sprint Planning et du Sprint Backlog nous avons fait une autre réunion d'équipe afin d'échanger sur la mise en place du workflow Git et sur la structure du/des repositories sur GitHub.

Il a été décidé que nous aurions 2 repositories un pour le back et un autre pour le front.

On s'est ensuite divisé en deux groupes, un pour le front et un pour le back.

J'ai expliqué à Alexis le Scrum Master et à Gregory le code de l'environnement Redux car même si cette partie était à ma charge il fallait qu'ils comprennent un minimum le fonctionnement afin qu'ils arrivent à s'orienter dans le code de l'application.

Mise en place de la base du code React et de l'environnement Redux :

J'ai décidé d'utiliser la librairie React avec la librairie React Redux afin de pouvoir centraliser la gestion de l'état de l'application dans un store.

En effet, lorsqu'une application commence à devenir relativement importante la gestion de l'état peut commencer à devenir complexe, on est alors obligé d'utiliser le processus qui est appelé le **prop-drilling**.
Cela consiste à passer les props de composant à composant afin d'atteindre le composant principal où est géré le state de l'application.
Les composants à travers desquels est passé la props servent en quelque sorte de "passe-plat" car ils n'ont pas besoin d'utiliser la props pour leur fonctionnement propre.

Utiliser ce type de processus peut fortement diminuer la scalabilité d'une application car il rend l'ajout de nouvelles fonctionnalités et de nouveaux composants de plus en plus difficile à gérer.

Pour résoudre ce type de problématique il existe ce qu'on appelle le concept de state management ou gestionnaire d'état.

L'idée derrière ce concept est de gérer le state de notre application qui doit être notre source unique de vérité, en dehors des composants de l'application.
Chaque composant pourra accéder à ce gestionnaire d'état indépendamment et donc récupérer à tout moment directement l'état de l'application afin de se mettre à jour en cohérence avec celui-ci.

Étant donné que notre application ne pouvait pas devenir, en l'espace de 4 semaines de développement, suffisamment importante pour que le type de difficulté rencontrées avec le prop-drilling devienne handicapant, j'ai néanmoins choisi d'utiliser quand même un gestionnaire d'état car je trouvais presque plus simple d'implémenter ce concept que de gérer ne serait-ce que quelques composants avec le prop-drilling.

Il était également possible que nous désirions continuer à faire grandir l'application après notre formation et à ce moment là les problématiques évoquées plus haut apparaitrait et il faudrait alors refactoriser considérablement le code existant pour pouvoir intégrer le gestionnaire d'état ce qui serait une perte de temps considérable.

La librairie React intègre son propre gestionnaire d'état qui est React Context mais ayant vu plus profondément React Redux lors de notre formation j'ai décidé d'opter pour celui-ci.

Par ailleurs, d'après un des membres de la Team React, Sebastian Markbåge, React Context est plus adapté pour gérer les changements d'état à basse fréquence tel qu'un changement de thème ou une authentification, mais n'est pas prêt à remplacer Redux pour la gestion de tous les autres changements d'états.
source : https://github.com/facebook/react/issues/14110

La façon la plus classique de commencer à créer une application React est d'utiliser Create React App qui configure automatiquement l'environnement de développement.

J'ai cependant décidé d'utiliser le boiler plate de démarrage que nous avons créé durant notre formation.

J'ai ajouté les librairies que l'on avait déjà décidé d'utiliser, y compris React Redux, librairie permettant de faciliter les liens entre Redux et React.

Dans le dossier src qui est à la racine de notre projet j'ai ensuite ajouté les dossiers suivants nécessaires à l'utilisation de Redux :

- Le dossier **actions** :
  C'est le dossier qui contiendra toutes les actions qui vont modifier le state global

- Le dossier **containers** :
  C'est le dossier qui contiendra les assistants des composants de présentation, c'est eux qui permettront de connecter nos composants au store de Redux

- Le dossier **middlewares :**

C'est le dossier qui contiendra les middlewares chargés de gérer les actions asynchrones

- Le dossier **reducers** :
  C'est le dossier qui contiendra le state ou état de notre application à un moment donné

- Le dossier **store :**
  C'est le dossier dans lequel on créer le store global de notre application

Première étapes de l'implémentation du scanner :

En ce qui concerne l'implémentation du scanner de code-barres, après de multiples essais j'ai opté pour la librairie **QuaggaJS**, c'est cette librairie qui semble s'intégrer le mieux à notre application.

Il a tout de même fallu que je fasse pas mal de modification des codes proposés afin de garder une cohérence avec le reste du code de notre application.

En effet tous les codes que j'ai trouvés utilisaient les composants basés sur les classes alors que l'on avait décidé d'utiliser uniquement des composants basés sur les fonctions car on gérait le state avec redux et si jamais nous devions implémenter un state local dans un composant nous utiliserions le hook d'état useState().

Un autre problème était que ce code fonctionnait très bien avec la caméra de mon téléphone mais avec celle de mon pc la capture du code-barres ne se produisait qu'une fois sur dix environ.
J'ai découvert ensuite que ce problème venait de la différence de résolution entre la web cam de mon pc et celle de mon téléphone, en remplaçant ma web cam actuelle par une plus performante ce problème s'est un peu estompé.

J'ai pu donc commencé à ce stade là à récupérer le code-barre scanné, à le stocker dans le state et à envoyer les premières requêtes vers l'API de Open Food Facts.

Pour envoyer ces requêtes, j'ai utilisé le concept de middlewares qui permet d'utiliser des fonctions asynchrone AJAX avec Redux.

J'ai aussi commencé à trier les faux positifs du scanner en ne conservant que les scans de code-barres contenant 13 et 8 caractères ce qui correspond au code-barres des produits alimentaires répertoriés dans l'API.

J'ai également mis en place les messages à l'utilisateur lorsque le scan avait échoué, le produit n'existait pas dans la BDD et aussi les messages de réussite du scanner.

Autres réalisations :

J'ai également profité de la création du premier middleware pour envoyer les premières données en POST à notre back-end afin de faire les premiers tests de communication.

À cette occasion nous nous sommes penchés sur la problématique des **CORS** ou Cross-Origin resource Sharing que je développerai dans le chapitre sur la veille technologique à la fin de ce dossier.

Nous avons également effectué un travail en groupes afin de déterminer les différents endpoints de notre back-end.

SPRINT 2

Une nouvelle réunion d'équipe nous as permis de formaliser les données au sujet des produits que le front devait envoyer au back.

Fusion des données d'Open Food Facts et des données utilisateur

Comme la première étape lors du scan est d'aller interroger l'API Open Food Facts et qu'une fois les informations du produit récupérées la deuxième étape est de récupérer d'autres informations issues de l'utilisateur, comme la date limite de consommation et la quantité par exemple, il fallait mettre en place un système qui permette de fusionner ces deux sources d'informations.

La solution que j'ai mise en place a été de récupérer les données d'Open Food Facts et celle de l'utilisateur dans le state et ensuite de fusionner toutes ces données dans un objet JSON que je passe à la requête AJAX envoyée en POST à notre back-end.

Il a également fallu modifier la date limite de consommation entrée par l'utilisateur afin de la transformer en format ISO pour quelle soit acceptée par Symfony.

J'ai également créé un formulaire d'ajout d'un produit à la main différent de celui utilisé pour l'ajout d'un produit scanné car les données sont d'entrée sont différentes et le back doit gérer ces deux types de produits de manière différente également.

Ces deux types de produits disposeront également de deux types d'actions différentes afin de bien les distinguer.

Tri des produits dans le Pantry

Il a été décider de trier les produits dans le Pantry en fonction de leur date limite de consommation et également de leur octroyer un code couleur toujours en fonction de cette date.
Pour ce faire j'ai créé un dossier utils qui est consacré aux fonctions utilitaires qu'on peut importer dans n'importe quel composant.

Dans ce fichier j'ai créé une fonction qui convertit la date courante et la date limite de consommation en timestamp (nombre de millisecondes écoulées depuis le 1er janvier 1970) afin d'avoir un nombre à manipuler au lieu d'une date classique, et j'ai ensuite attribués des classes différentes en fonction de la différence entre la date courante et la date limite de consommation.

J'ai ensuite importé cette fonction pour l'utiliser dans le composant s'occupant de gérer les cartes correspondant à chaque produit.

Autres réalisations :

Utilisation des Json Web Token que je développerai en détail dans le chapitre veille technologique.

J'ai passé du temps avec Laurie pour déboguer un problème de format des données envoyées au back pour l'ajout d'un nouveau produit.
Après pas mal d'essais de format de données différents il s'est avéré que le soucis venait d'une mauvaise configuration de Symfony pour la réception de ces données et non d'un problème de formatage de ces données au niveau du front.

J'ai aussi commencé à implémenter la suppression d'un produit lors de cette fin de sprint.

SPRINT 3

Mise en place de la fonctionnalité d'inscription en utilisant le hook useState afin de gérer les messages d'erreur en utilisant des states locaux à la place du reducer de Redux.

Ici, l'utilisation du state global n'est pas justifiée car il n' y a pas besoin de partager les données du state de ce composant avec les autres composants de l'application.

Mise en place de la fonctionnalité de suppression d'un produit, de la mise à jour de la date limite de consommation et de la quantité en utilisant un composant différent pour chacune de ces mise à jour afin de garder les fichiers de composant le plus léger possible.
Une fenêtre modale apparait avec un overlay à chaque action de mise à jour.

Mise en ligne du front sur l'hébergeur IONOS.
On a rencontré une difficulté à ce moment là du fait que le front était en https et l'API en http.
Le front ne voulait pas récupérer de ressources sur un site non sécurisé, donc en attendant que l'équipe du back mette en place un certificat SSL sur leur hébergeur nous avons autoriser le contenu non sécurisé sur le navigateur pour pouvoir continuer à travailler.

Une autre difficulté, inhérente à la mise en production du site, que nous avons rencontré était que si nous rafraichissions une autre page de la home, on tombait sur une page 404 not found.
La résolution de cette difficulté sera développée dans le chapitre recherche et traduction d'extrait de ce dossier.

J'ai également mis en place la fonctionnalité de modification du nombre de jour avant la date de péremption pour l'alerte email sur le tableau de bord de l'utilisateur.

La connexion automatique lors de l'inscription a également été implémentée afin d'améliorer l'expérience utilisateur.

À la demande du Product Owner, je me suis mis l'avant dernier jour du sprint à développer la fonctionnalité de mise à jour des informations utilisateur.
Je me suis servi de la fonctionnalité de mise à jour des informations utilisateurs du site Amazon pour développer celle de notre site, notamment au niveau de la mise à jour du mot de passe où il est demandé d'entrer l'actuel mot de passe puis deux fois le nouveau mot de passe afin que la mise à jour soit acceptée.

À la fin de ce dernier Sprint, nous avons fait la présentation de notre projet sur la chaîne YouTube de l'école O'clock que l'on peut retrouver à cette adresse : https://www.youtube.com/watch?v=n_Jo3Pcf87c&t=548s&ab_channel=O%27clock#t=9m00s

B. FOCUS FONCTIONNALITÉS

Dans cette partie nous allons étudier en détail plusieurs fonctionnalités importantes de notre application que j'ai personnellement développés.

Pour cela, je vais en premier lieu faire une présentation globale du code de l'application afin de ne pas avoir à répéter les même choses à chaque présentation de fonctionnalité.

Pour commencer à l'intérieur du dossier src nous avons créé un fichier index.js qui est le point d'entrée de l'application.

Point d'entrée de l'application

On commence par faire un import de :

- **React** dans le fichier afin de pouvoir utiliser le **JSX**, le sucre syntaxique de React qui permet de faciliter l'écriture du code**.**
- la méthode **render** de **react-dom** permettant de faire le rendu de l'élément React de base dans le nœud DOM racine.
- la méthode **Provider** de **react-redux**, c'est un High Order Component qui ne va donc pas servir à modifier l'interface utilisateur mais à fournir les informations du store à tous ses composants enfants, donc à tous les composants de l'application. React Redux utilise pour cela le Contexte fourni par React.

```
import React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
```

On importe ensuite la méthode **BrowserRouter** de **react-router-dom** qui va permettre d'interagir avec la barre d'adresse du navigateur via son API History.

```
// Import de la librairie react-router-dom
import { BrowserRouter as Router } from 'react-router-dom';
```

On passe ensuite aux imports locaux :

- App qui est le composant parent de tous les autres composants de l'application
- store qui est le dossier où est créé le store Redux

```
// Import locaux
import App from 'src/containers/App';
import store from 'src/store';
```

On créer ensuite l'élément React racine qui contient l'ensemble de l'application.

```
// Création de l'élément React racine (celui qui contient l'ensemble de l'app)
const rootReactElement = (
  <Provider store={store}>
    <Router>
      <App />
    </Router>
  </Provider>
);
```

On cible ensuite le nœud du DOM qui va contenir React.

```
// Container de l'élément react racine
const target = document.getElementById('root');
```

Ce nœud est présent dans le fichier index.html que l'on a placé dans le dossier assets du dossier src.

```
    </head>
    <body>
        <div id="root"></div>
    </body>
</html>
```

Et on passe ces deux constantes à la méthode render() de react-dom afin de déclencher le rendu de React.

```
// Déclenchement du rendu de React
render(rootReactElement, target);
```

Store de React redux

On a créé un fichier index.js dans le dossier store où se trouve le store de React Redux.

On importe :

- La méthode **createStore** qui va permettre de créer le store global
- La méthode **applyMiddleware** qui va permettre d'ajouter un ou plusieurs middlewares à notre store.
  Les middlewares permettent essentiellement d'utiliser des méthodes asynchrones à l'intérieur de nos différentes actions.
  En effet, pour des raisons de fiabilité du comportement de notre application on ne peut pas directement modifier son state global avec des actions asynchrones car le retard potentiel de certaines requêtes http par rapport à une action synchrone classique risquerait de créer des incohérences au niveau de la mise à jour du state.

- La méthode **composeWithDevTools** de l'extension redux devtools, qui nous permettra d'utiliser un outil supplémentaire de débogage très pratique directement dans notre navigateur. Cet outil nous permettra entre autres de connaître l'état de l'application à un moment donné et même de pouvoir revenir dans le temps grâce à

sa timeline afin de voir les différents états qu'a traversé l'application au fil des différentes actions qui se sont succédé.

```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
```

On continue en important le **reducer** qui permet de manipuler le state de notre application en le modifiant via des actions reçues et en renvoyant le nouvel état du state au store.

Important, le reducer doit être une fonction pure, elle doit donc être déterministe et toujours retourner les même résultats en sortie avec les mêmes arguments en entrée.
Ce qui signifie quelle ne doit pas dépendre de variable en dehors de sa portée qui serait donc susceptible de changer au fil du temps et donc de renvoyer pour une entrée identique un résultat différent.

```
import reducer from 'src/reducers/';
```

On importe également les middlewares évoqués plus haut.

```
import dataMiddleware from 'src/middlewares/dataMiddleware';
import userMiddleware from 'src/middlewares/userMiddleware';
```

On passe nos middlewares au dev tools.

```
// on combine devTools avec les middlewares
const enhancers = composeWithDevTools(
  applyMiddleware(
    dataMiddleware,
    userMiddleware,
  ),
);
```

Puis on passe l'ensemble de ces arguments à la fonction de création de notre store et on l'exporte pour pouvoir l'utiliser dans le Provider vu plus haut.

```
const store = createStore(
  reducer,
  enhancers,
);

export default store;
```

Action creators

Dans un dossier actions on stockera toutes les actions que l'on pourra utiliser dans l'application sous forme de fonction qui retourneront un objet d'action avec une propriété indiquant le type de l'action et le cas échéant un ou plusieurs payloads qui seront des informations supplémentaires passées en argument de la fonction.

Dans ce fichier on créera des constantes avec le type de chaque action afin que notre linter, ESLint ici, nous prévienne en cas d'erreur ou de faute de frappe.

Exemple de constante et d'action correspondante :

notre constante

```
export const UPDATE_USER_FIELD = 'UPDATE_USER_FIELD';
```

et l'action qui y correspond, avec 2 payloads

```
export const updateUserField = (newValue, name) => ({
  type: UPDATE_USER_FIELD,
  newValue,
  name,
});
```

Reducer

Dans le dossier reducer nous aurons un fichier index.js qui permettra de créer notre ou nos reducers.

On y importe la méthode **combineReducers** de Redux ainsi que nos différents reducers.

```
import { combineReducers } from 'redux';

import burgerReducer from 'src/reducers/burger';
import myaccountReducer from 'src/reducers/myaccount';
import userReducer from 'src/reducers/user';
```

On combine les différents reducers et on les exporte avec un export par défaut que l'on récupèrera dans notre méthode createStore vu plus haut.

```
const rootReducer = combineReducers({
  // nom du reducer qui gère cette partie du state
  burger: burgerReducer,
  myaccount: myaccountReducer,
  user: userReducer,
});

export default rootReducer;
```

SCANNER DE CODE-BARRES

Nous allons maintenant étudier l'implémentation du scanner de code-barres.

La documentation de la librairie Quagga nous donne les exemples de code à mettre en place dans le composant dédié au scanner.
Je ne vais pas développer ici l'étude de ce code car cela nous amènerait trop loin et n'est pas l'objet de ce dossier.
Il nous suffit juste de savoir qu'il y a 5 méthodes principales :

- **Quagga.init()** initialisera la librairie en fonction des paramètres que l'on aura configurés, et demandera l'accès à la caméra du device.
  Une callback vérifiera si il n'y a pas eu d'erreur et si c'est le cas, appellera la méthode **Quagga.start()** qui démarrera le flux vidéo et commencera à décoder les images
- **Quagga.onProcessed()** chargé du processus de décodage des images proprement dit.
- **Quagga.onDetected()** qui sera appelé lorsque un motif de code-barres sera détecté
- **Quagga.stop()** interrompra le processus de décodage et déconnectera la caméra.

J'importe tout d'abord :

- React et useEffect
- PropTypes de la librairie prop-types.
  Cette librairie va nous permettre de valider le type des props utilisés dans le composant. Cette vérification ne se fera qu'en mode développement, c'est une aide pour le développeur.
  Ici par exemple, je vérifie que la props handleScan est bien une fonction et j'ajoute la propriété isRequired à la fin pour m'assurer de recevoir un message d'avertissement supplémentaire dans la console si la props n'est pas fournie

```
Scanner.propTypes = {
  handleScan: PropTypes.func.isRequired,
};
```

- Et la librairie Quagga

```
import React, { useEffect } from 'react';
import PropTypes from 'prop-types';
import Quagga from 'quagga';
```

Je passe Quagga.init(), Quagga.onProcessed() et Quagga.onDetected() dans le hook useEffect() avec un tableau de dépendances en second argument ce qui permettra de ne lancer ces méthodes qu'au premier rendu du composant.

Je passe Quagga.stop() dans la fonction de retour de useEffect() afin que le processus de décodage soit stoppé lors du démontage du composant.

```
const Scanner = ({ handleScan }) => {
  const init = () => {
    Quagga.init(
      {...
      },

      (err) => {
        if (err) {
          console.log(err);
          return;
        }
        Quagga.start();
      },
    );
  };

  const onDetected = (result) => {
    const code = result;

    Quagga.stop();
    return handleScan(code);
  };

  const onProcessed = (result) => {...
  };

  useEffect(() => {
    let isMounted = true;
    if (isMounted) {
        init();
        Quagga.onDetected(onDetected);
        Quagga.onProcessed(onProcessed);
    }

    return () => {
      isMounted = false;
      Quagga.stop();
    };
  }, []);

  return <div id="interactive" className="viewport smallView" />;
};
```

Le return du composant Scanner affichera la fenêtre où la caméra s'exécutera.

onDetected() retournera en argument de la fonction handleScan() le code-barres qui aura été détecté.

J'exporte à la fin de mon fichier mon composant Scanner afin de pouvoir l'utiliser.

```
export default Scanner;
```

Ce composant Scanner est importé dans le fichier index.js du dossier ScanProduct qui contient le composant ScanProduct.

Dans ce composant ScanProduct on importe le composant Scanner.

```
import Scanner from './Scanner';
```

Et on le place dans une div conditionnelle qui ne l'affichera que lorsque l'on cliquera sur la croix, qui déclenchera le basculement de la props modal à true.

```
{modal && (
  <div className="modal">

    <div className="modal-body">
      <div className="modal-content">
        <i className="fas fa-times camCroix" onClick={toggleModal} />
        <Scanner handleScan={onDetected} />
      </div>
    </div>
  </div>
)}
```

La fonction handleScan du composant scanner récupèrera le code-barres détecté et la passera à la fonction onDetected.

Cette fonction onDetected sera récupérée par le container du composant ScanProduct.

Ce container peut lire les propriétés du state grâce à la fonction mapStateToProps.

```
// mapStateToProps
// si j'ai besoin de lire des informations dans le state
const mapStateToProps = (state) => ({
  // nom de la prop à remplir : élément à récupérer dans le state
  scanCode: state.user.scanCode,
  modal: state.user.modal,
  scanDatas: state.user.scanDatas,
  status: state.user.status,
  productFound: state.user.productFound,
  currentProduct: state.user.currentProduct,
  barCode: state.user.barCode,
  isLogged: state.user.isLogged,
});
```

Et peut aussi dispatcher des actions vers le store grâce à la fonction mapDispatchToProps.
C'est ici que nous retrouvons la fonction onDetected qui a récupérée le code-barres.

```
// mapDispatchToProps
// si j'ai besoin de dispatcher une action vers le store
const mapDispatchToProps = (dispatch) => ({
  // nom de la prop à remplir: fonction qui dispatch l'action
  onChange: (newValue, name) => {
    console.log(`le champ ${name} a la valeur : ${newValue}`);
    dispatch(onChange(newValue, name));
  },
  onChangeBarCode: (newValue) => {
    console.log(`la nouvelle valeur du code-barres est : ${newValue}`);
    dispatch(onChangeBarCode(newValue));
  },
  catchBarCode: (newValue) => {
    dispatch(catchBarCode(newValue));
  },
  handleAddProduct: () => {
    dispatch(handleAddProduct());
  },
  toggleModal: () => {
    dispatch(toggleModal());
  },
  toggleScanInfo: () => {
    dispatch(toggleScanInfo());
  },
  onDetected: (result) => {
    dispatch(onDetected(result));
  },
  cleanUp: () => {
    dispatch(cleanUp());
  },
});
```

Ce container sera connecté au store par la fonction connect() qui prendra en paramètre mapStateToProps et mapDispatchToProps elle renverra une autre fonction qui prendra en paramètre le composant ScanProduct.
C'est ainsi que la liaison entre le composant de présentation ScanProduct, son container et le store sera effectuée.

```
// création de l'assistant
export default connect(mapStateToProps, mapDispatchToProps)(ScanProduct);
```

Ensuite le middleware dataMiddleware interceptera l'action onDetected.

Ce middleware est une triple fléchée qui prendra en premier paramètre le store, puis renverra une deuxième fonction qui prendra en paramètre next et cette fonction renverra une troisième fonction qui prendra en paramètre action permettant de passer l'action au middleware suivant.

```
const datasMiddleware = (store) => (next) => (action) => {
```

Dans ce middleware on va utiliser l'instruction switch qui va nous permettre de traiter plusieurs cas qui correspondront chacun à une action différente.

Dans le case de l'action ON_DETECTED qui nous occupe on va commencer par récupérer le code-barres stocké dans le state, sa valeur par défaut est une chaîne vide tant que le code-barres n'a pas été enregistré dans le state.

Et on va ensuite récupérer le code-barres qui vient d'être scanné.

```
switch (action.type) {
  case ON_DETECTED: {
    // Je récupère le code-barres qui est enregistré dans le state actuel
    const {
      scanCode,
    } = store.getState().user;

    // Je récupère dans barCode le code-barres du produit qui vient d'être scanné via le payload
    // de l'action ON_DETECTED
    const barCode = action.result.codeResult.code;
```

Ensuite je vérifie si le code-barres stocké actuellement dans la state est différent du code-barres qui vient d'être scanné ET si le code-barres qui vient d'être scanné contient 13 caractère.

Si c'est le cas, je fais ma requête vers l'API d'Open Food Facts en lui passant dans l'url le code-barres scanné.

```
// J'exécute la requête seulement si le code-barres qui vient
// d'être scanné contient 13 chiffres et si il est différent
// du code-barres qui est actuellement enregistré dans le state
if ((scanCode !== barCode) && (barCode.length === 13)) {
  // faire une requête vers l'API
  axios.get(`https://world.openfoodfacts.org/api/v0/product/${barCode}.json`)
```

J'utilise ces deux conditions afin d'optimiser les performances du scanner car :

- En ne faisant la requête que si le code-barres scanné est différent du code-barres stocké dans le state j'évite de faire partir plusieurs requêtes vers l'API avec le même code-barres, car la capture du scanner est si rapide que très souvent le même code-barres est scanné plusieurs fois d'affilée et fait donc partir plusieurs requêtes en moins d'une seconde.
  En conditionnant le départ de la requête à l'unicité du code-barres, même si plusieurs code-barres identiques sont scannés très rapidement je ne fais partir au final qu'une seule requête vers l'API.
- La quasi-totalité des produits alimentaires ont un code-barres EAN à 13 chiffres, donc en conditionnant l'exécution de la requête à ce type de code-barres j'élimine beaucoup de faux positif et j'améliore ainsi le fonctionnement de l'application.
  J'ai conscience qu'en faisant cela je risque de ne pas pouvoir scanner certains produits mais l'imperfection de la librairie utilisée m'oblige pour l'instant à avoir recours à ce stratagème pour que l'expérience utilisateur de l'application soit suffisamment bonne.

J'exécute ensuite la requête et dispatch son résultat vers le state grâce à l'action productRecovery, je récupère les éventuelles erreurs dans catch au cas où il y aurait eu un problème.

Puis j'appelle next en lui passant l'action en paramètre afin de passer l'action au middleware suivant.

```
// J'exécute la requête seulement si le code-barres qui vient
// d'être scanné contient 13 chiffres et si il est différent
// du code-barres qui est actuellement enregistré dans le state
if ((scanCode !== barCode) && (barCode.length === 13)) {
  // requête vers l'API d'Open Food facts
  axios.get(`https://world.openfoodfacts.org/api/v0/product/${barCode}.json`)
    .then((response) => {
      console.log(response.data);

      store.dispatch(productRecovery(response.data));
    })
    .catch((error) => {
      console.warn(error);
    });
}
```

Le résultat de la requête est maintenant envoyé dans le reducer.

C'est une fonction qui va prendre en paramètre le state initial et l'action, puis qui va par l'intermédiaire de l'instruction switch, traiter différents cas correspondant chacun à une action différente, un peu à la façon du middleware.

```
const user = (state = initialState, action = {}) => {
  switch (action.type) {
```

Dans l'action productRecovery qui nous intéresse, je teste plusieurs cas :

- Le cas où la propriété status_verbose, renvoyée par l'API est égale à product found. Le produit a donc bien été identifié.

```
case PRODUCT_RECOVERY: {
  if (action.datas.status_verbose === 'product found') {
    return {
      ...state,
      currentProduct: action.datas,
      status: 'product found',
      barCode: '',
      scanCode: '',
      modal: false,
    };
  }
```

J'utilise le spread operator qui permet de récupérer la totalité du contenu du state et de le déverser dans un nouvel objet sans modifier l'objet d'origine ce qui est indispensable au bon fonctionnement de React car si l'on mutait directement le state sera pourrait conduire à des comportements imprévisibles de notre application et rendrait le débogage difficile.

J'ajoute ensuite dans ce nouvel objet les propriétés du state que je veux modifier, elles écraseront dans ce nouvel objet les propriétés ayant le même nom et donc les remplaceront.

En particulier, je passe la valeur product found à la propriété status, ce qui va nous permettre de pouvoir gérer conditionnellement l'affichage du composant ScanProduct.

Je passe également les données d'Open Food Facts concernant le produit scanné dans la propriété currentProduct.

- Le cas où status_verbose est égale à product not found.

```
else if (action.datas.status_verbose === 'product not found') {
  return {
    ...state,
    status: 'product not found',
    barCode: '',
    scanCode: '',
    modal: false,
  };
}
```

- Sinon je passe la valeur invalid code à la propriété status.

```
return {
  ...state,
  status: 'invalid code',
  barCode: '',
  scanCode: '',
  modal: false,
};
```

Nous revenons maintenant dans le composant ScanProduct qui va être directement impacté par ces modifications au niveau du state.

Lorsque status est sur product found on fait apparaitre les deux div suivantes :

Une où on indique à l'utilisateur quel produit il vient de scanner.

```
{(status === 'product found') && (
<div className="add-product">
  <div className="scanSuccess">
    <i className="fas fa-times scan-info croixScanSuccesFirst" onClick={toggleScanInfo} />
    <p>Je viens de scanner le produit  <br />
      <mark>{currentProduct.product.product_name_fr}</mark><br />
      de la marque
      <mark>{currentProduct.product.brands}</mark><br />
      <br />
    </p>
  </div>
  <div className="arrow-down-success" />
</div>
)}
```

Et une autre qui vient du composant Infosproduct où on a mis en place un formulaire qui va permettre à l'utilisateur d'entrer la date limite de consommation du produit ainsi que la quantité.

Celle de ScanProduct dans laquelle on a importé InfosProduct :

```
{(status === 'product found') && (
  <div className="input-dlc ">
    <InfosProduct onChange={onChange} handleAddProduct={handleAddProduct} />
  </div>
)}
```

Celle de InfosProduct :

```
const infosProduct = ({ handleAddProduct, onChange }) => {
  const handleSubmit = (evt) => {
    evt.preventDefault();
    handleAddProduct();
  };

  const handleChange = (evt) => {
    // console.log(evt.target.value);
    onChange(evt.target.value, evt.target.name);
  };

  return (
    <div className="infos-product">
      <h2 className="infos-product-title">Votre scan a réussi</h2>
      <form className="infos-product-form" onSubmit={handleSubmit}>
        Entrez maintenant la date limite de consommation<br />
        <input
          type="date"
          className="infos-product-input"
          onChange={handleChange}
          name="dlc"
        />
        ainsi que la quantité, puis validez pour enregistrer<br />

        <input
          type="number"
          className="infos-product-input"
          onChange={handleChange}
          name="quantite"
        />
        <button
          type="submit"
          className="infos-product-button"
        >
          Valider
        </button>
      </form>
      <div className="img_barcode" />
    </div>
  );
};
```

Ce qui nous donne l'IHM suivante :

Une fois que l'utilisateur a validé le formulaire, la fonction handleAddProduct() va déclencher dans le container de ScanProduct l'action correspondante.

```
handleAddProduct: () => {
  dispatch(handleAddProduct());
},
```

Qui va être interceptée dans le dataMiddleware, où je vais commencer par récupérer les données d'Open Food Facts concernant le produit scanné que j'ai stocké dans la propriété currentProduct, puis la quantité et la date limite de consommation saisies par l'utilisateur.

```
case HANDLE_ADD_PRODUCT: {
  // Récupération des données du state
  const {
    currentProduct,
    quantite,
    dlc,
  } = store.getState().user;
```

Je récupère le token correspondant à l'utilisateur dans le localStorage du navigateur, nous approfondirons cet aspect lors de l'étude de la fonctionnalité de connexion, puis je convertis la date au format ISO afin qu'elle soit acceptée par le back.

```
const token = localStorage.getItem('token');

// On convertit la date du produit ajouté par l'utilisateur en date au format ISO
// afin quelle soit acceptée par Symfony
const date = new Date(dlc);
const expDate = date.toISOString();
```

Puis j'envoi ces données à notre API afin qu'elle stocke le nouveau produit de l'utilisateur actuellement connecté.

```
axios.post(`${localAPI}/user/product/add/scan`, {
  // Création et envoi du nouvel objet JSON avec les données d'open food + les données
  // rentrées par le user au format JSON determiné par le back

  name: currentProduct.product.product_name_fr,
  brand: currentProduct.product.brands,
  image: currentProduct.product.image_front_thumb_url,
  product_quantity: currentProduct.product.quantity,
  ingredients: currentProduct.product.ingredients_text,
  quantity: parseInt(quantite, 10),
  nutriscore_grade: currentProduct.product.nutriscore_grade,
  barcode: currentProduct.code,
  expiration_date: expDate,
}, {
  headers: { Authorization: `Bearer ${token}` },
})
  .then((response) => {
    console.log(response);
    store.dispatch(addProductToPantry(response.data));
  })
  .catch((error) => {
    console.warn(error);
  });
```

L'API me renvoi en réponse la totalité des produits de l'utilisateur que je vais envoyer à mon state via la fonction addProductToPantry().

```
case ADD_PRODUCT_TO_PANTRY:
  return {
    ...state,
    userProducts: action.datas,
    status: 'product added',
  };
```

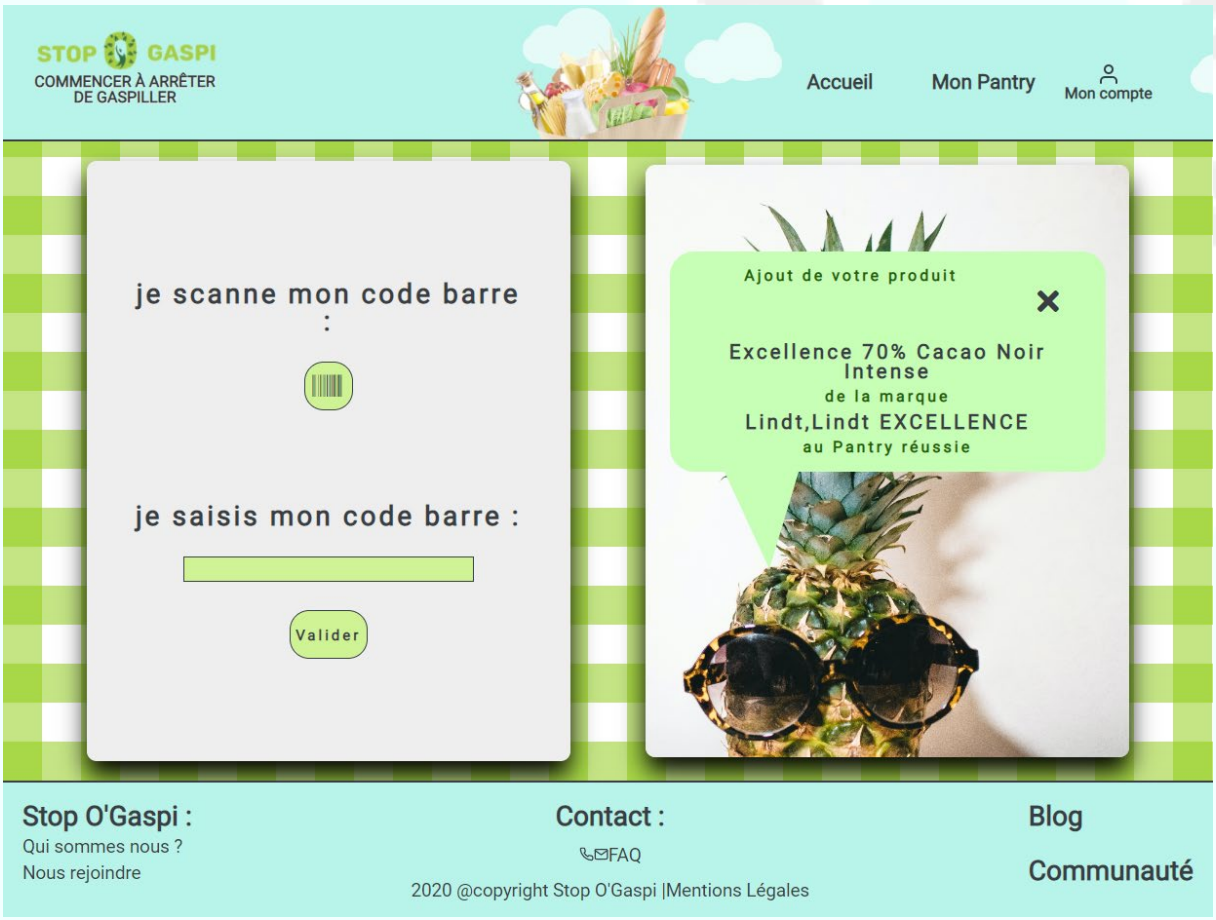
Je passe alors la propriété status à product added, ce qui va mettre à jour l'affichage du composant ScanProduct.

```
{(status === 'product added') && (
<div className="add-product">
  <div className="scanSuccess">
    <i className="fas fa-times scan-info" onClick={toggleScanInfo} />
    <p>Ajout de votre produit<br />
      <mark>{currentProduct.product.product_name_fr}</mark><br />
      de la marque<br />
      <mark>{currentProduct.product.brands}</mark><br />

      <Link to="pantry">
        au Pantry réussie
      </Link>
    </p>
  </div>
  <div className="arrow-down-success" />
</div>
)}
```

Le texte "au Pantry réussie" est placé dans une balise Link qui permet à l'utilisateur d'accéder directement au Pantry après l'ajout de son produit.

Ce qui va donner l'IHM suivante :



À partir de là, l'utilisateur aura le choix entre scanner un nouveau produit et repartir dans le cycle d'actions que l'on vient de décrire en détail ou aller voir les caractéristiques de son produit dans le Pantry.

INSCRIPTION

Nous allons maintenant étudier la fonctionnalité d'inscription.

Dans le composant Registration nous allons importer le hook useState permettant de créer un state local dans un functionnal component que nous allons uniquement utiliser pour gérer les messages d'erreurs du formulaire d'inscription.

```
import React, { useState } from 'react';
```

Nous allons déclarer plusieurs variables d'état dont nous aurons besoin pour gérer les messages d'erreurs au cours de la saisie des informations dans le formulaire et également lors de la soumission du formulaire.

```
// Variables d'état permettant la vérification lors de l'évènement blur, donc au cours
// du remplissage du formulaire
const [equality, setEquality] = useState(true);
const [numberCharacters, setNumberCharacters] = useState(true);

// Variables d'état permettant la vérification lors de l'évènement submit, donc au moment
// de la validation du formulaire
const [equalitySubmission, setEqualitySubmission] = useState(true);
const [charactersSubmission, setCharactersSubmission] = useState(true);
```

On créer les constantes qui correspondront aux messages d'erreurs.

```
const errorPasswords = 'Les mots de passe ne sont pas identiques !';
const errorNumberCharacters = 'Votre mot de passe doit contenir 8 caractères au minimum !';
const errorEqualitySubmit = 'Les mots de passe doivent être identiques pour l\'envoi du formulaire !';
const errorNbCharactersSubmit = 'Les mots de passe doivent contenir 8 caractères au minimum pour l\'envoi du
                                formulaire !';
```

On vérifie la valeur des différentes variables d'état lors de la soumission du formulaire, si elles sont toutes à true et qu'aucun de nos champs est vides ont envoi le formulaire, sinon on modifie les variables d'états afin d'afficher le message d'erreur approprié.

```
const handleSubmitLoggin = (evt) => {
  evt.preventDefault();
  if (
    (equality && numberCharacters)
      && email.length !== 0
      && name.length !== 0
      && City.length !== 0
      && password.length !== 0
      && verifPassword.length !== 0
      && pseudo.length !== 0
  ) {
    setEqualitySubmission(true);
    setCharactersSubmission(true);
    handleRegistration();
  }
  else if (equality && !numberCharacters) {
    setEqualitySubmission(true);
    setCharactersSubmission(false);
  }
  else if (!equality && numberCharacters) {
    setEqualitySubmission(false);
    setCharactersSubmission(true);
  }
  else {
    setEqualitySubmission(false);
    setCharactersSubmission(false);
  }
};
```

Notre mettons en place une fonction :

```
const handleChange = (evt) => {
  // console.log(evt.target.name);
  onChangeRegistration(evt.target.value, evt.target.name);
};
```

Et les différentes valeurs des attributs values (exemple pour le label Nom) :

```
<div className="user-contain">
  <input
    type="text"
    name="registrationName"
    required
    onChange={handleChange}
    value={name}
  />
  <label>Nom</label>
</div>
```

De telle sorte que notre formulaire devient un composant contrôlé, c'est-à-dire qu'à tout moment, la valeur de chaque champ du formulaire est pilotée par React et non par le DOM.

Nous aurons donc à chaque frappe l'information saisie par l'utilisateur disponible dans le state de redux ce qui nous permettra de la manipuler plus facilement et dans un autre composant que celui du formulaire le cas échéant.

Nous allons nous servir de cela pour récupérer les valeurs saisies dans le userMiddleware déclenché par l'action HANDLE_REGISTRATION elle-même déclenchée par la fonction handleRegistration.

```
case HANDLE_REGISTRATION: {
  const {
    registrationEmail,
    registrationName,
    registrationCity,
    registrationPassword,
    registrationVerifPassword,
    registrationPseudo,
  } = store.getState().user;
```

Nous allons ensuite exécuter la requête avec toutes ces informations dans le body.

```
axios.post(`${localAPI}/login/signon`, {
  email: registrationEmail,
  name: registrationName,
  City: registrationCity,
  password: registrationPassword,
  verifPassword: registrationVerifPassword,
  pseudo: registrationPseudo,
})
```

Puis utiliser la réponse de la requête pour dispatcher la fonction automaticConnection() qui remplira les propriétés username et password du state et dispatcher également la fonction login().

```
.then((response) => {
  console.log(response);
  store.dispatch(automaticConnection());
  store.dispatch(logIn());
})
```

Si on a des erreurs on enverra les messages correspondants en fonction des réponses du back au formulaire d'inscription.

```
.then((response) => {
  console.log(response);
  store.dispatch(automaticConnection());
  store.dispatch(logIn());
})
.catch((error) => {
  console.warn(error.response.status);
  if (error.response.status === 404) {
    store.dispatch(catchError('L\'inscription a échoué, veuillez réessayer.'));
  }
  else if (error.response.status === 403) {
    store.dispatch(catchError('Cette adresse email a déjà un compte, veuillez réessayer ou vous connecter.'));
  }
});

next(action);
break;
```

La fonction login() déclenchera l'action LOG_IN correspondante dans le même middleware où on récupérera les valeurs des propriétés username et password du state qui viennent d'être mise à jour par la fonction automaticConnection().

```
const userMiddleware = (store) => (next) => (action) => {
  switch (action.type) {
    case LOG_IN: {
      const { username, password } = store.getState().user;
      console.log(`l'email est :${username} et le password est : ${password}`);
```

On exécutera alors la requête permettant de connecter l'utilisateur.

```
axios.post(`${localAPI}/login_check`, {
  username,
  password,
})
```

On récupérera la réponse de la requête qui contiendra le **JWT** ou Json Web Token que l'on stockera dans le localStorage du navigateur.

Ce JWT sera envoyé maintenant à chaque requête vers l'API que fera l'utilisateur afin de vérifier son identité et autoriser ou non l'exécution de cette requête.

Ce JWT a également une date d'expiration paramétrée par l'API ce qui sécurisera notre front en obligeant l'utilisateur à se reconnecter au bout de cette date d'expiration si il veut faire une nouvelle requête ou bien simplement consulter ses données.

```
.then((response) => {
  // console.log(response);
  localStorage.setItem('token', response.data.token);
  console.log(response.data, 'line 42');
  store.dispatch(getAllProducts());
})
```

On dispatchera la fonction getAllProducts() qui déclenchera l'action GET_ALL_PRODUCTS puis la requête permettant de récupérer tous les produits triés par ordre de date limite de consommation croissante.

À l'intérieur de cette requête on récupérera le JWT stocké qui permettra d'identifier l'utilisateur et on le passera dans le header de la requête, puis on dispatchera la fonction fillPantry() qui remplira le state avec tous les produits de l'utilisateur.

```
case GET_ALL_PRODUCTS: {
  const token = localStorage.getItem('token');
  axios.get(`${localAPI}/user/product/all/order-by-date`, {
    headers: { Authorization: `Bearer ${token}` },
  })
    .then((response) => {
      console.log(response, 'line 90');
      store.dispatch(fillPantry(response.data));
    })
    .catch((error) => {
      console.warn(error);
      console.log(error.message);
      if (error.message === 'Request failed with status code 401') {
        store.dispatch(logOut());
      }
    });

  next(action);
  break;
}
```

Si il y une erreur avec le code 401 Unauthorized c'est que l'identification de l'utilisateur a échouée, par mesure de sécurité on deconnectera donc l'utilisateur en dispatchant la fonction logOut() qui supprimera le token du localStorage.

```
case LOG_OUT: {
  localStorage.removeItem('token');

  return {
    ...state,
    isLogged: false,
    successfulRegistration: false,
  };
}
```

De retour dans la requête de connexion, un deuxième .then() dispatchera une troisième fonction fetchUserInfos() une fois la deuxième requête terminée.

```
.then(() => {
  store.dispatch(fetchUserInfos());
})
```

Celle-ci déclenchera l'action FETCH_USER_INFOS qui exécutera une nouvelle requête.

```
case FETCH_USER_INFOS: {
  const token = localStorage.getItem('token');

  axios.get(`${localAPI}/user`, {
    headers: { Authorization: `Bearer ${token}` },
  })
    .then((response) => {
      console.log(response.data, 'line 72');
      store.dispatch(saveUser(response.data));
    })
    .catch((error) => {
      console.warn(error);
    });

  next(action);
  break;
}
```

Celle-ci dispatchera la fonction saveUser() qui déclenchera l'action SAVE_USER qui remplira les informations utilisateurs au niveau du state permettant notamment de renseigner celles-ci dans son tableau de bord.

De retour dans la requête de connexion, si il y a des erreurs et en fonction de l'erreur un message sera envoyé à l'utilisateur.

```
.catch((error) => {
  console.warn(error.response.status);
  if (error.response.status === 404) {
    store.dispatch(catchError('La connexion a échoué, veuillez réessayer.'));
  }
  else if (error.response.status === 401) {
    store.dispatch(catchError('Identifiants invalides, veuillez réessayer.'));
  }
});

next(action);
break;
```

ALERT DAY

Pour finir cette partie focus fonctionnalités, nous allons étudier la modification du nombre de jour avant le jour de date limite de consommation où le mail d'alerte sera envoyé à l'utilisateur.

Dans le composant Dashboard nous avons une partie réservée aux jours d'alertes, pour que cela soit plus clair voici l'IHM correspondante :

Le jour où le mail est envoyé a une bordure noire, donc le jour de la date limite de consommation pour cette IHM.

Pour changer ce jour il suffira à l'utilisateur de cliquer sur le jour qu'il a choisi.

Au nveau du code voici la partie correspondante à notre capture :

```
<div className="alert_dashboard">
  <h2>ALERTE MAIL :</h2>
  <div
    className={alertDayValue === 0 ? 'alert_J ok' : 'alert_J'}
    onClick={() => {
      alertChange(0);
    }}
  >
    Jour J
  </div>
  <div
    className={alertDayValue === 1 ? 'alert_24 ok' : 'alert_24'}
    onClick={() => {
      alertChange(1);
    }}
  >
    24h Avant
  </div>
  <div
    className={alertDayValue === 2 ? 'alert_48 ok' : 'alert_48'}
    onClick={() => {
      alertChange(2);
    }}
  >
    48h Avant
  </div>
</div>
<div className={displayTempModal ? 'modalInfo' : 'modalInfo hide'}>
  Modification enregistrée !
</div>
```

Le ternaire dans chaque className permet de modifier la couleur de la bordure en fonction de la valeur de la propriété alertDayValue courante du state.

Au click sur une des div d'un jour la fonction alertChange() est appelée avec la valeur du jour sur lequel on a cliqué en paramètre, ce qui va déclencher l'action ALERT_CHANGE correspondante qui va être interceptée dans le userMiddleware.

Dans cette action on récupère le token stocké dans le localStorage et la valeur du jour choisi, ici on n'a pas besoin de récupérer cette valeur dans le state car on l'a directment passée en paramètre de la fonction appelée.

```
case ALERT_CHANGE: {
  const token = localStorage.getItem('token');
  const alertLevel = action.value;
```

On exécute ensuite la requête en passant le jour choisi dans le body et le token dans le header.

```
axios.post(`${localAPI}/user/edit/alertday/${alertLevel}`, {
  alertLevel,
}, {
  headers: { Authorization: `Bearer ${token}` },
})
```

Puis on dispatche la fonction changeAlertDay() avec la valeur renvoyée par l'API ce qui va déclencher l'action CHANGE_ALERT_DAY qui va mettre à jour cette valeur dans le state.

```
.then((response) => {
  // console.log(response.data);
  console.log(response.data);
  store.dispatch(changeAlertDay(response.data.alert_day));
```

Cette action va également passer la valeur de la propriété displayTempModal à true ce qui va faire apparaitre le message de succès "Modification enregistrée !" que l'on peut voir plus haut dans la capture du code du composant Dashboard.

```
case CHANGE_ALERT_DAY:
  return {
    ...state,
    alertDayValue: action.value,
    displayTempModal: true,
  };
```

De retour dans la requête de modification du jour, 2 secondes après le dispatch de la fonction changeAlertDay par l'intermédiaire de la méthode setTimeOut() on dispatche la fonction closeModal() qui va déclencher l'action CLOSE_MODAL qui va passer dans le state la propriété displayTempModal à false afin de faire disparaitre le message de succès "Modification enregistrée !".

```
      setTimeout(() => {
        store.dispatch(closeModal());
      }, 2000);
    })
    .catch((error) => {
      console.warn(error);
    });

next(action);
break;
```

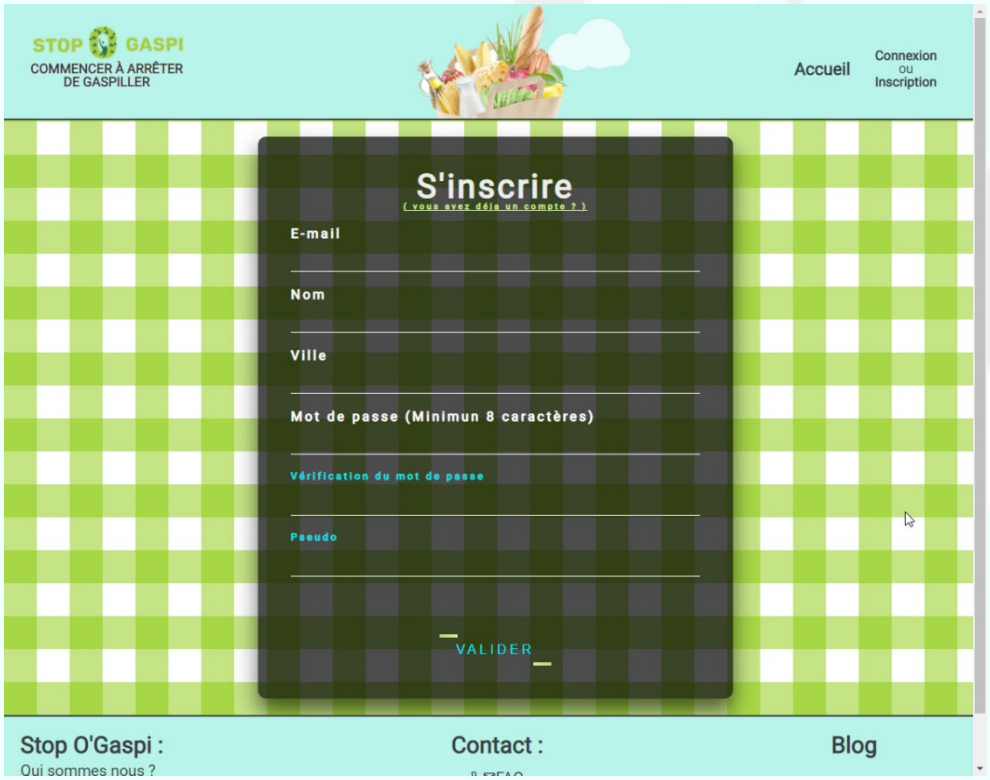
VIII. JEU D'ESSAI

Pour le chapitre de tests fonctionnels, nous allons réaliser les actions suivantes :

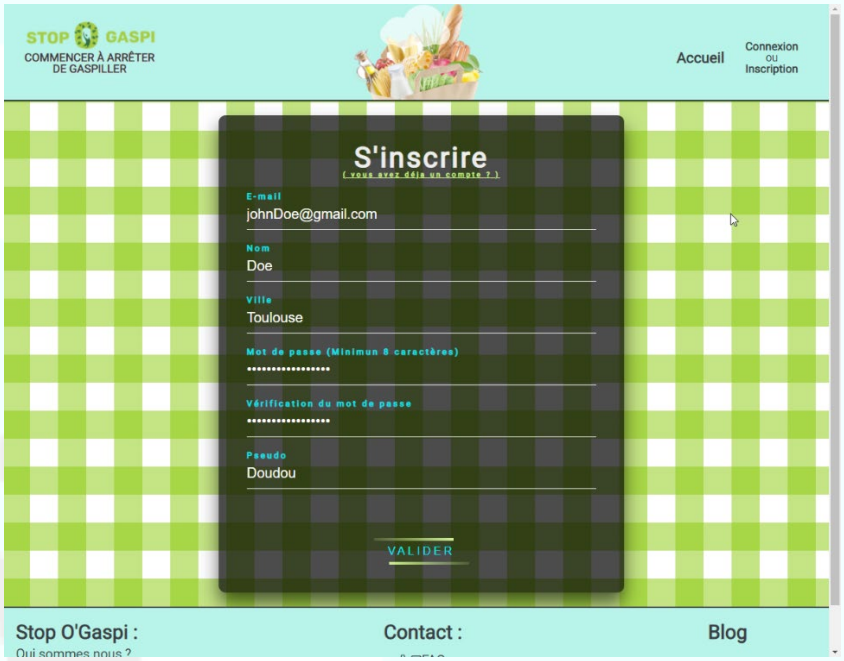
- S'inscrire
- Vérifier la présence du nouvel utilisateur et ses informations
- Entrer un produit à l'aide du scanner de code-barres
- Vérifier la présence du produit et ses informations
- Modifier un produit
- Vérifier la prise en compte de la modification

INSCRIPTION

On se rend sur la page d'inscription :



On rentre les informations du nouvel utilisateur :

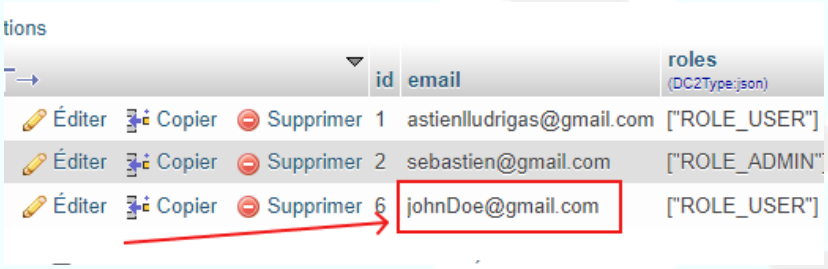


Les informations sont les suivantes :

- E-mail : johnDoe@gmail.com
- Nom : Doe
- Ville : Toulouse
- Mot de passe : motdepasseJohnDoe
- Pseudo : Doudou

On valide l'inscription, on est redirigé sur le Pantry qui est vide pour l'instant :

On vérifie dans phpMyAdmin que l'utilisateur a bien été créé :

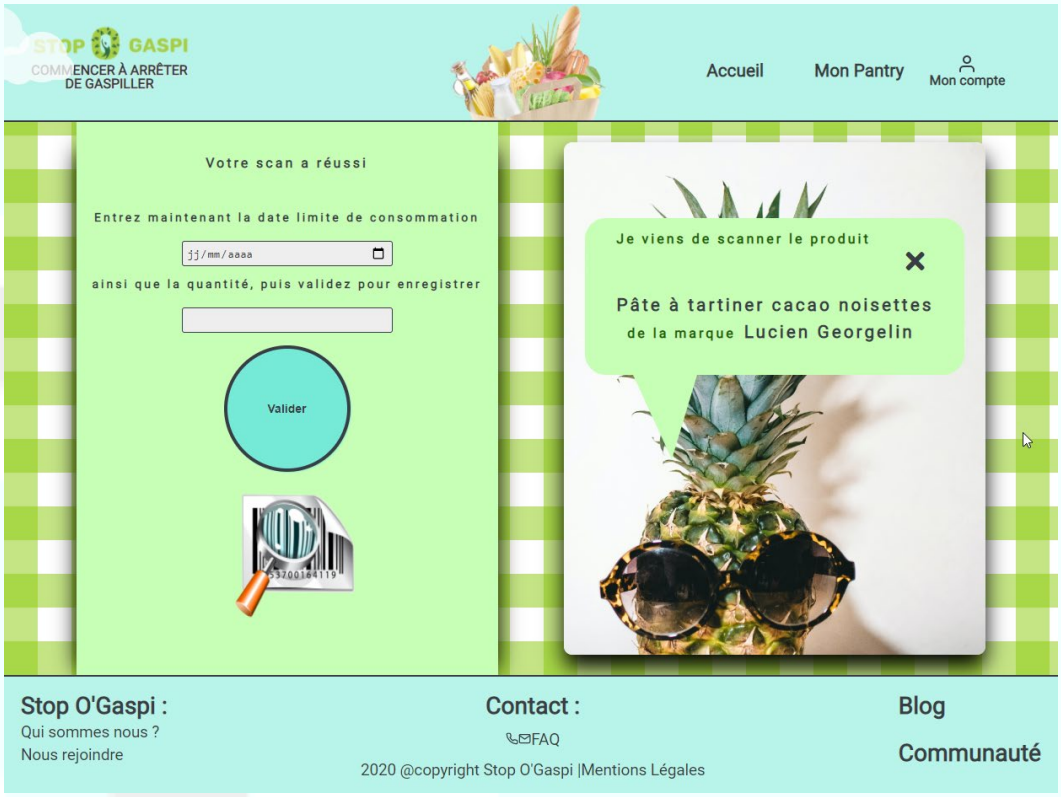


C'est bien le cas, et avec toutes ses informations :

password	name	city	created_at	pseudo	username	alert_day
$2y$13$y0RdEK0awY5tiYdzjovOH.V9YB9ozlikad2q1JFv6CW...	Sebastien	Nice	2021-08-05 09:03:52	Elon Musk	astienlludrigas@gmail.com	2
$2y$13$Q9qU8h.DwEm.X1edK6mGs.5JNY5hqeW3...WGxmW.64IU...	Sebastien	Nice	2021-08-05 09:49:10	Elon Musk	sebastien@gmail.com	1
$2y$13$Z5nnxZTqpQrlblYMX.kkTOoBvFe/1HrUQDu/NTwvs/7...	Doe	Toulouse	2021-11-07 11:09:55	Doudou	johnDoe@gmail.com	1

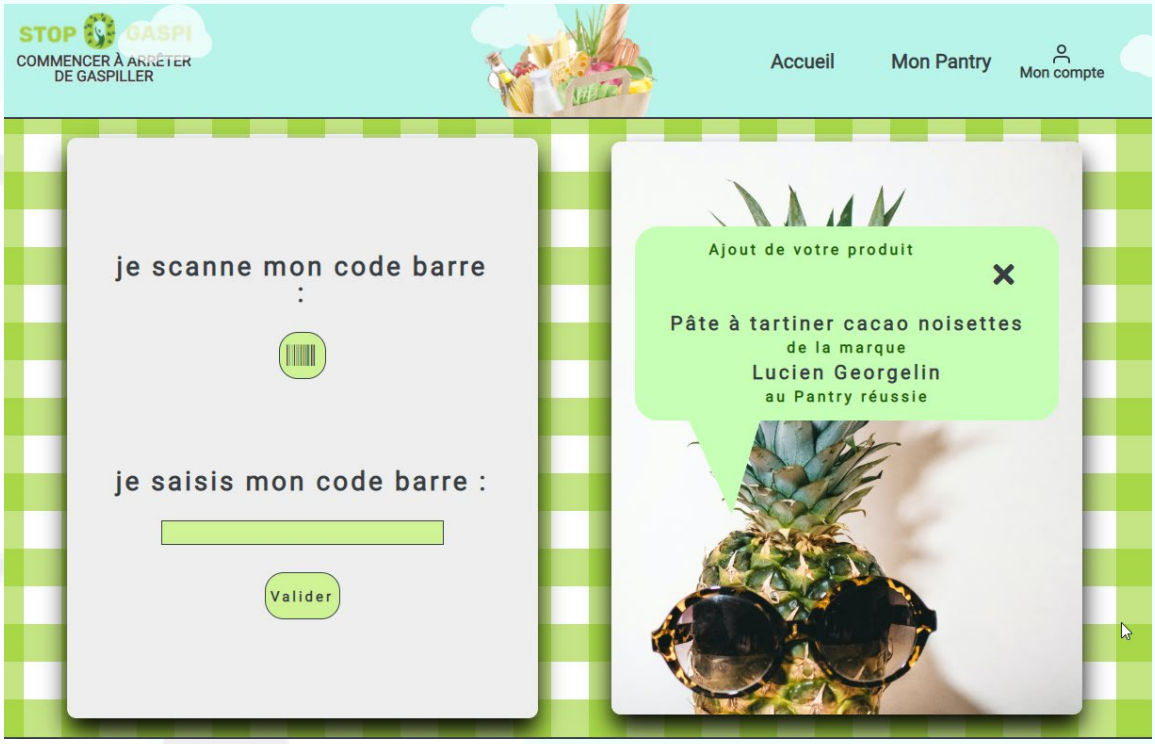
SCAN D'UN PRODUIT

On scanne maintenant un produit :

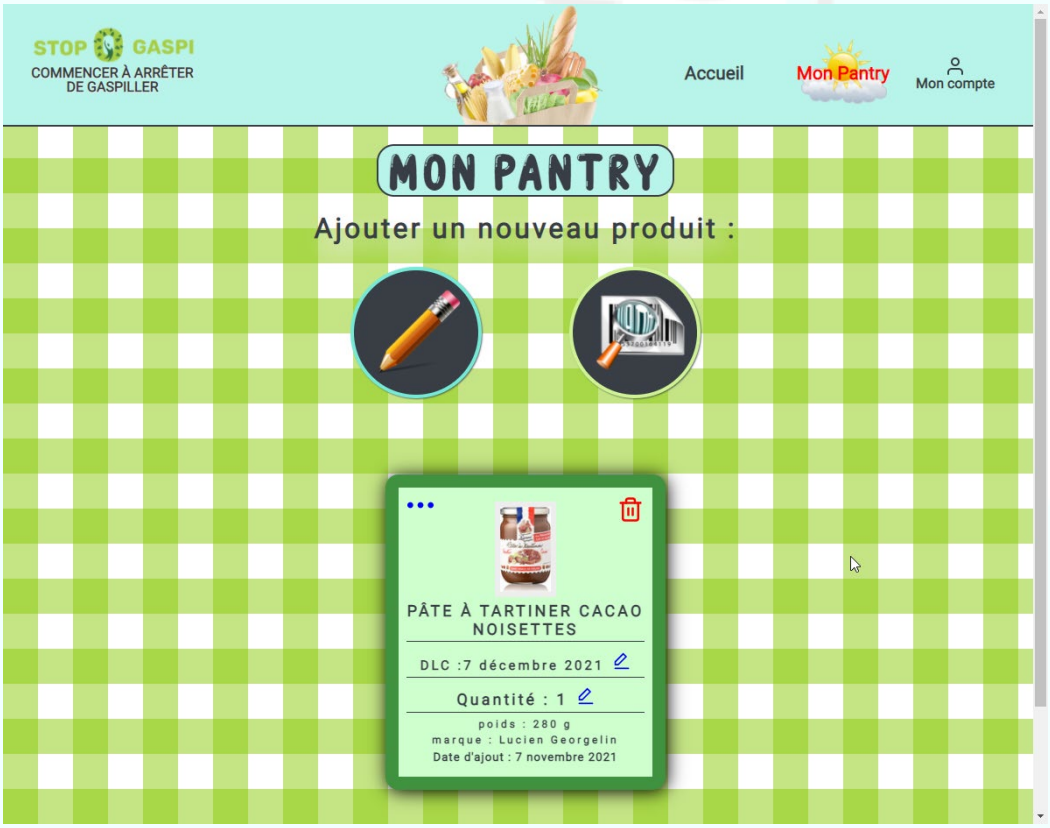


On ajoute les informations sur le produit :

On valide :



On va sur le Pantry pour voir notre produit :



Il s'y trouve bien et avec les bonnes informations.

On vérifie maintenant dans phpMyAdmin :

L'ajout à bien été pris en compte.

MODIFICATION D'UN PRODUIT

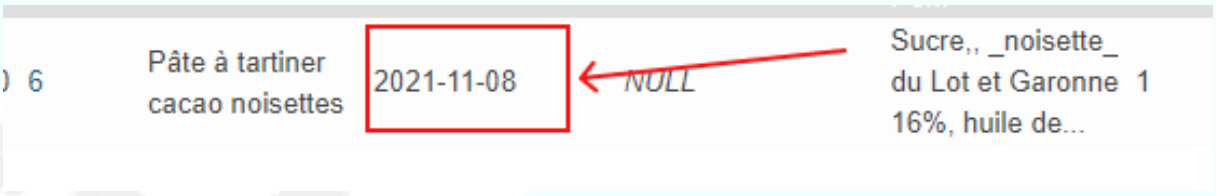
On modifie maintenant le produit que l'on vient d'ajouter en changeant sa date limite de consommation pour le lendemain afin de voir également le changement de couleur de sa bordure.

On modifie la DLC en cliquant sur la partie DLC de la carte :

On la passe donc au 08/11/2021 :

On valide et on constate le changement de place et d'apparence de la carte avec la nouvelle DLC :

On vérifie le changement dans phpMyAdmin :



La modfification a bien été prise en compte avec la nouvelle date limite de consommation.

IX. VEILLE TECHNOLOGIQUE

On a principalement fait de la veille au niveau de la sécurité sur 2 points :

- **CORS** : Cross-Origin Resource Sharing

  Le standard CORS permet via des entêtes http à des navigateurs de charger des ressources à partir d'un autre domaine que celui à partir duquel la première ressource a été chargée.

  Il s'agit de réguler le partage de ressources entre deux domaines d'origines différentes afin de sécuriser ces échanges.
  Pour ce faire, des entêtes http spécifie les domaines à partir duquel les ressources ont le droit d'être chargées, ensuite lorsque le navigateur veut récupérer ces ressources il envoi d'abord une requête en utilisant la méthode OPTIONS, celle-ci fera la demande à la ressource visée et si la ressource n'autorise pas le partage pour ce domaine l'envoi de la deuxième requête sensée récupérer les données ne sera pas autorisée.

  L'équipe du Back a donc mis en place les entêtes http nécessaires à la communication entre notre front-end et notre back-end, à l'aide du bundle NelmioCorsBundle.

  La principale ressource utilisée pour ce travail a été la documentation du bundle : https://github.com/nelmio/NelmioCorsBundle

  Ainsi que cette question sur stackoverflow : https://stackoverflow.com/questions/54544855/allow-cors-on-symfony-4

- **JWT** :

  J'ai utilisé les JWT ou Json Web Token pour sécuriser la connexion d'un utilisateur.

  Ce token sous forme d'objet JSON est composé de trois parties :

  - Un entête
  - Une charge utile
  - Une signature numérique

  Son utilisation est la suivante :

  1. Envoi des informations de connexion fourni par l'utilisateur au back-end en POST.
  2. Vérification de la validité des informations par le back, si elles sont correctes le back envoi au front un JWT qu'il aura généré.
  3. Le front stocke ce JWT dans le local storage du navigateur
  4. Maintenant, chaque requête que le front enverra au back devra être accompagnée de ce JWT, si il est valide la requête sera exécutée par le back, sinon un message d'erreur sera renvoyé et la requête n'aboutira pas.

Le back pourra paramétrer la durée de vie du JWT à volonté.

L'avantage par rapport au session storage c'est que les données stockées dans le local storage persisteront même après la fermeture du navigateur, ce qui permettra à l'utilisateur de revenir sur le site et d'être automatiquement connecté si la durée de vie du JWT paramétrée par le back n'a pas expirée.

Un Hook React, useEffect, permettra de lancer la vérification de l'existence du token et de sa validité à chaque fois que l'utilisateur chargera la page d'accueil du site.

J'ai également mis en place le système de déconnexion qui supprime le token enregistré dans le local storage et bascule l'interface utilisateur en mode déconnecté tout en redirigeant l'utilisateur vers la page de connexion.

Les principales ressources utilisées pour mettre en place les JWT sont côté front :

- Ce tutoriel : https://www.alibabacloud.com/blog/how-to-implement-authentication-in-reactjs-using-jwt_595820
- Ainsi que ce cours sur Udemy : https://www.udemy.com/course/react-v16-redux-avance/

Côté back :

- Ce tutoriel sur YouTube : https://www.youtube.com/watch?v=XT4oy1d1j-g&ab_channel=OverSeasMedia
- Et la documentation officielle du bundle LexikJWTAuthenticationBundle : https://github.com/lexik/LexikJWTAuthenticationBundle

X. RECHERCHE ET TRADUCTION D'EXTRAIT

Nous avons eu un soucis avec le front lors du déploiement de notre site sur l'hébergeur IONOS.
Nous avons constaté que lorsque l'on rafraichissait une autre page que la page d'accueil avec un F5 ou un Ctrl + Shift + R on tombait sur une 404 not found.

On a alors effectué une recherche Google pour essayer de comprendre ce qui se passait et trouver une solution.

La recherche Google était la suivante : React page refresh problem in production

Dans les multiples résultats de recherche c'est ce lien vers Stack Overflow qui a été le plus pertinent et qui nous a finalement permis de régler le problème.

La cause du problème venait du fait que pour le serveur de production une seule url existe, celle de la page d'accueil de notre site, donc lorsqu'on rafraichissait une autre page, l'url de la requête http envoyée au serveur n'était plus celle de la page d'accueil mais celle de la page du routing virtuel créé par React Router.

Par exemple, le Pantry est situé sur l'url : https://stop-o-gaspi.ocrprojects.com/pantry mais le serveur ne connait que https://stop-o-gaspi.ocrprojects.com donc lorsqu'on rafraichit la page du Pantry ou si on copie-colle son url dans un autre navigateur et que l'on fait la requête http avec cette url le serveur nous renvoi une 404 not found.

Le routing avec React Router est lancé une fois que l'application a fini de charger et ce n'est qu'à ce moment là que la page Pantry existe mais seulement sur le navigateur car cette page n'existe pour ainsi dire qu'à l'intérieur de notre code javascript et le serveur n'en a pas connaissance.

Pour régler le problème nous avons mis en place un dossier .htaccess dans le dossier de notre projet sur le serveur avec des règles de réécriture d'url.
Ces règles nous permettent de conserver l'historique du navigateur tout en renvoyant au niveau du serveur toutes les routes vers index.html, quelque soit ce qu'il y a après le / qui suit l'url de notre site.

React Router utilisant l'historique du navigateur va pouvoir ensuite rajouter le chemin d'origine demandé, ce qui nous mènera à la bonne page.

Le texte en anglais qui a permis la compréhension du problème est le suivant :

TEXTE EN ANGLAIS

Server-side vs Client-side

The first big thing to understand about this is that there are now 2 places where the URL is interpreted, whereas there used to be only 1 in 'the old days'. In the past, when life was simple, some user sent a request for http://example.com/about to the server, which inspected the path part of the URL, determined the user was requesting the about page, and then sent back that page.

With client-side routing, which is what React-Router provides, things are less simple. At first, the client does not have any JS code loaded yet. So the very first request will always be to the server. That will then return a page that contains the needed script tags to load React and React Router etc. Only when those scripts have loaded does phase 2 start. In phase 2, when the user clicks on the 'About us' navigation link, for example, the URL is changed locally only to http://example.com/about (made possible by the History API), but no request to the server is made. Instead, React Router does its thing on the client-side, determines which React view to render, and renders it. Assuming your about page does not need to make any REST calls, it's done already. You have transitioned from Home to About Us without any server request having fired.

So basically when you click a link, some Javascript runs that manipulates the URL in the address bar, without causing a page refresh, which in turn causes React Router to perform a page transition on the client-side.

But now consider what happens if you copy-paste the URL in the address bar and e-mail it to a friend. Your friend has not loaded your website yet. In other words, she is still in phase 1. No React Router is running on her machine yet. So her browser will make a server request to http://example.com/about.

And this is where your trouble starts. Until now, you could get away with just placing a static HTML at the webroot of your server. But that would give 404 errors for all other URLs when requested from the server. Those same URLs work fine on the client-side, because there React Router is doing the routing for you, but they fail on the server-side unless you make your server understand them.

Combining server- and client-side routing

If you want the http://example.com/about URL to work on both the server- and the client-side, you need to set up routes for it on both the server- and the client-side. Makes sense right?

And this is where your choices begin. Solutions range from bypassing the problem altogether, via a catch-all route that returns the bootstrap HTML, to the full-on isomorphic approach where both the server and the client run the same JS code.

TRADUCTION PERSONNELLE EN FRANÇAIS

Côté serveur – côté client

La première chose importante à comprendre à ce sujet est qu'il y a maintenant deux endroits où l'url est interprétée, alors qu'il y en avait qu'un seul dans l'ancien temps.

Dans le passé, quand la vie était simple, un utilisateur envoyait une demande pour http://example.com/about au serveur, lequel inspectait la partie chemin de l'URL, déterminait que l'utilisateur demandait la page à propos, et ensuite renvoyait cette page.

Avec le routage côté client, celui que fournit React-Router, les choses sont moins simples.

Au début, le client n'a pas encore chargé le code JS. Donc la toute première requête sera toujours adressée au server.
Cela renverra ensuite une page contenant les balises de script nécessaires pour charger

React et react Router, etc... Ce n'est que lorsque ces scripts sont chargés que la phase 2 démarre.

En phase 2, quand l'utilisateur clique sur le lien "À propos de nous", par exemple, l'url est changée seulement localement en http://example.com/about (rendu possible par l'API History), mais aucune demande n'est faite au serveur.

Au lieu de cela, React Router fait son travail côté client, détermine quelle vue React il doit restituer et il la restitue.
En supposant que votre page " À propos de nous " n'ait pas besoin de faire d'appels REST, le travail est déjà terminé.
Vous êtes passé de "Accueil" à "À propos de nous" sans qu'aucune demande de serveur n'ait été déclenchée.

Donc fondamentalement, lorsque vous cliquez sur un lien, certains scripts Javascript s'exécutent qui manipulent l'URL dans la barre d'adresse, sans provoquer de rafraichissement de la page, ce qui à son tour oblige React Router à effectuer une transition de page côté client.

Mais maintenant considérez ce qui se passe si vous copiez-collez l'URL dans la barre d'adresse et l'envoyez par mail à un ami. Votre ami n'a pas encore chargé votre site Web. En d'autres termes, elle est toujours en phase 1. Aucun router React n'est exécuté sur sa machine. Donc son navigateur fera une requête de serveur à http://example.com/about.

Et c'est là que votre problème commence. Jusqu'à présent, vous pouviez simplement placer un code HTML statique à la racine Web de votre serveur. Mais cela donnerait des erreurs 404 pour toutes les autres URL lorsqu'elles sont demandées par le serveur. Ces mêmes URL fonctionnent bien côté client, car React Router effectue le routage pour vous, mais elles échouent côté serveur à moins que vous ne les fassiez comprendre à votre serveur.

Combiner le routage côté serveur et côté client.

Si vous voulez que l'URL http://example.com/about fonctionne à la fois côté serveur et côté client, vous devez configurer des routes pour elle à la fois côté serveur et côté client.
Cela fait sens, n'est-ce pas ?

Et c'est là que commence vos choix.
Les solutions vont du contournement complet du problème, via une route attrape-tout qui renvoie le code HTML d'amorçage, à l'approche isomorphe complète où le serveur et le client exécutent le même code JS.

XI. CONCLUSION

Le travail sur ce projet m'a permis pour la première fois de me confronter aux difficultés du travail en équipe sur un projet de développement, car jusque là je n'avais quasiment travaillé qu'en solitaire sur tous les challenges et les parcours durant ma formation, mis à part les quelques ateliers de pair-programming que je n'avais d'ailleurs jamais apprécié.
J'ai en effet besoin de m'isoler pour pouvoir me concentrer pleinement et utiliser toutes mes capacités de réflexion.

J'ai pu m'apercevoir que le métier de développeur n'est pas seulement coder et réfléchir sur son code mais aussi savoir gérer le relationnel dans une équipe en faisant des concessions et en écoutant ce que les autres ont à dire. Accepter que les autres pensent différemment de nous et travaille également d'une façon différente, donc lâcher la bride et accepter de ne pas tout controller.

Ce projet m'a aussi mis au défi personnellement car on m'a attribué la tâche la plus ardu qui était celle d'implémenter le scanner de code-barres. On comptait sur moi car on pensait que j'étais le meilleur pour mener à bien l'implémentation de cette fonctionnalité et cela n'a fait que me rajouter encore plus de pression... De l'avis même de nos formateurs, mettre en place toutes les fonctionnalités que l'on avait prévu plus le scanner ne serait surement pas possible compte-tenu du temps qui nous était imparti, et ils avaient raison, du moins si on se basait sur des journées de travail de 8h 5 jours/semaine...

J'ai travaillé dur 7jour/7 et plus de 12h/jour parfois pour mener à bien ce projet et faire mentir nos formateurs, j'ai voulu donner le maximum de moi-même pour montrer à mon équipe qu'ils avaient eu raison de me faire confiance et je pense avoir réussi.
Cette expérience m'a appris à me faire confiance et à ne jamais préjuger de mon manque de compétences pour relever un nouveau défi et cela m'a beaucoup servi lors de ma première expérience professionnelle. Je doute encore toujours pas mal mais j'ai maintenant des exemples de réalisations personnelles qui me permettent de ne pas me laisser décourager.

Quoi qu'il en soit, ce fut une expérience enrichissante sur pleins de points, et même si il y a eu des moments difficiles de remise en question personnelles j'en garde un très bon souvenir.

XII. REMERCIEMENTS

Je tiens à remercier mon mentor OpenClassrooms Florent l'Hélias qui a été pour ainsi dire mon premier professeur dans mon apprentissage du développement.
C'est lui qui le premier à su me donner confiance en moi et a réussi à me faire croire que j'étais fait pour ça. Sa bienveillance, sa gentillesse et son accessibilité m'ont permis de surmonter les moments de doute et de découragements que je n'ai pas manqué de rencontrer à mes débuts.

Je remercie également toutes l'équipe de l'école O'clock et plus particulièrement nos formateurs pour nous avoir transmis leur savoir dans la joie et la bonne humeur ce qui nous permettait de décompresser dans les moments les plus difficiles.
Un grand merci à l'équipe pédagogique également pour la qualité de cette formation si riche et intéressante.

Je tiens enfin à remercier Lauryne Riera la femme de ma vie, qui m'a soutenu inconditionnellement pendant toute ma reconversion et sans qui rien n'aurait été possible.