Conception

**FoodFlow**

(Application web de gestion de cantine / machine distributrice)



Travail remis dans le cadre du cours de  
   
Projet Synthèse  
Par Alexis Provost et Thomas Pelletier  
  
  
  
  
Présenté à: Monsieur Jean-Christophe Demers  
Remis le 23 février 2023

# **Table des matières**

[**Table des matières**](#_84cjd0qmtkbq)

[**1. Maquettes**](#_3tf380j8frt4)

[**Interface client**](#_wgibndvcd4ck)

[**Interface Administrateur**](#_8e3c9ggo75ep)

[**2. Conception UML**](#_7sprlspy6c6f)

[**Diagramme de cas d’usage**](#_u2fs3jzalhge)

[**Diagramme de classes**](#_pzn4xq2oenmp)

[**3. Structure de données externes**](#_56ilv8w9c4oi)

[**Diagramme Postgresql**](#_tx9hhanvxwt4)

[**4. Éléments de conception**](#_iucfmn9j74j7)

[**Structure de données**](#_rmoc3stybi3)

[**Patron de design**](#_occdqtm7yse8)

[**Composite**](#_h9nx8iros3ft)

[**Proxy**](#_lazhjxlbbfoz)

[**Factory Method**](#_2bl4ixgf8t0j)

[**Observer**](#_gjpxr86z0lj1)

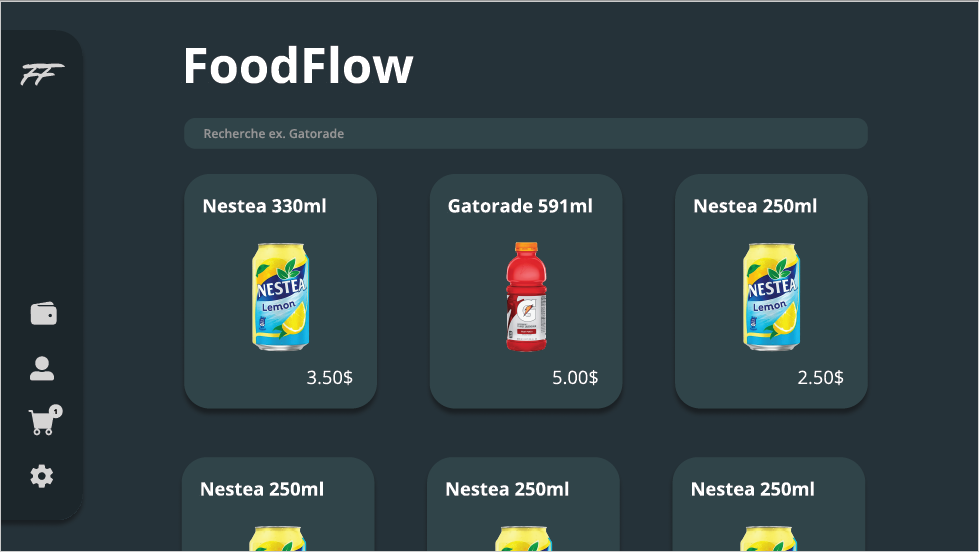
[**Algorithme & Mathématique**](#_eyirrc39j3gm)

[**Expression régulière**](#_i67o0v576ovo)

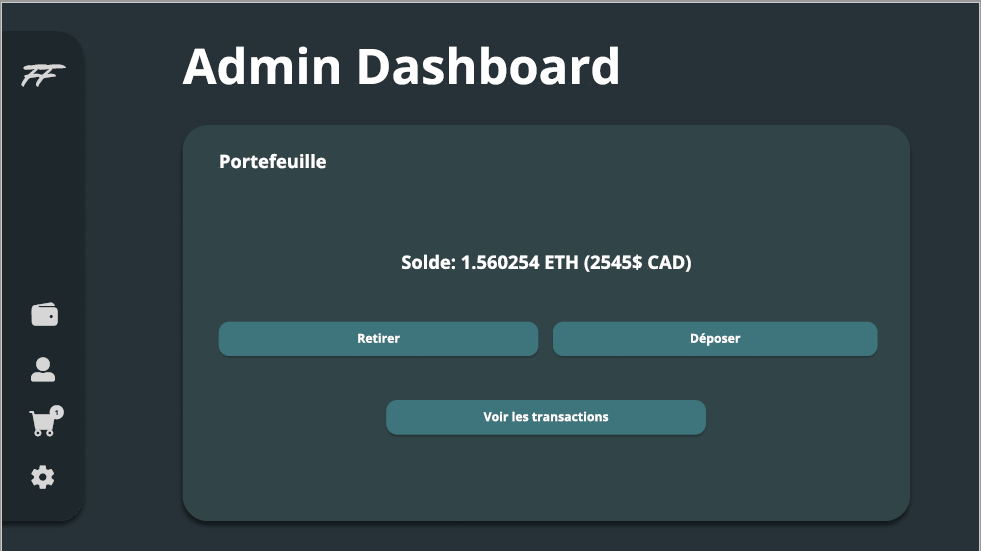
# 1. Maquettes

## Interface client

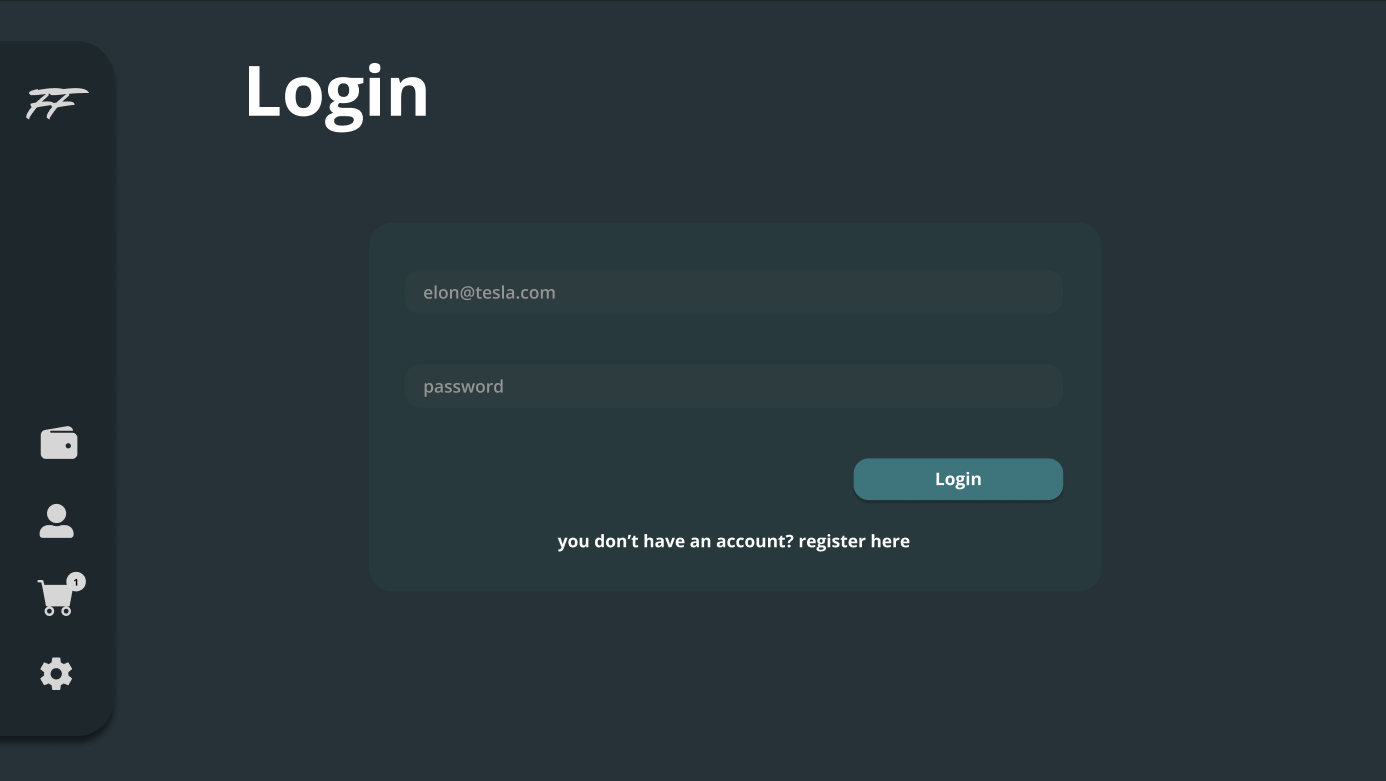
-Page d’accueil



-Page portefeuille

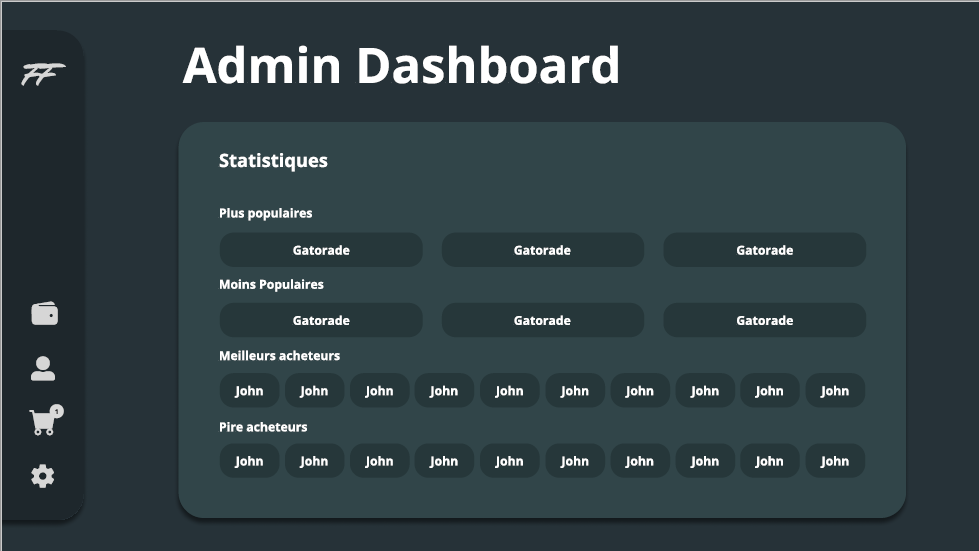


-Page login

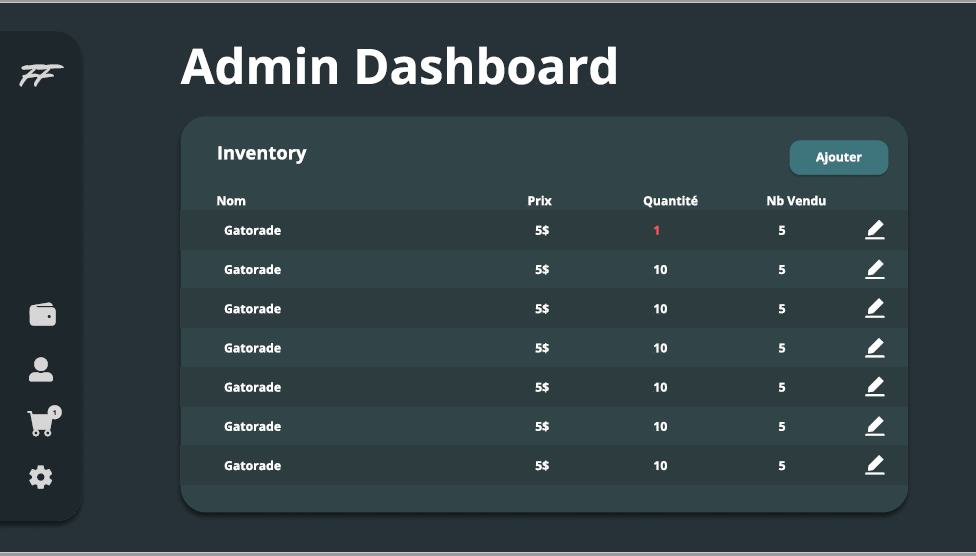


## Interface Administrateur

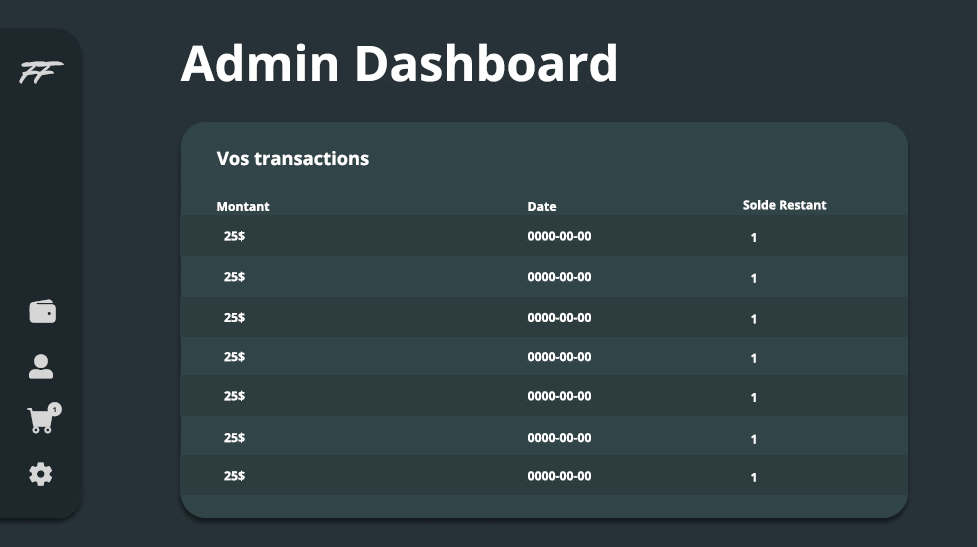
-Page statistiques



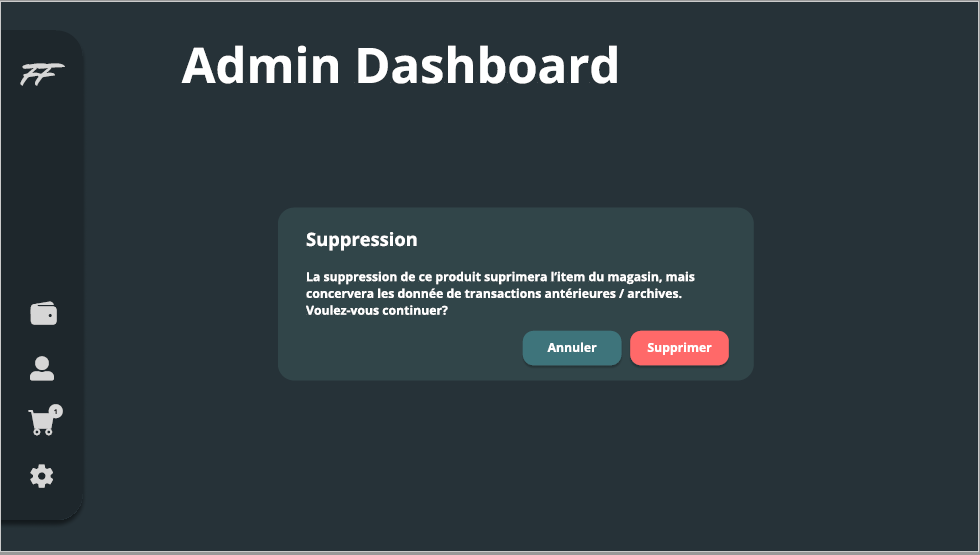
-Page inventaire



-Page gestion des factures



-Suppression de produit fenêtre

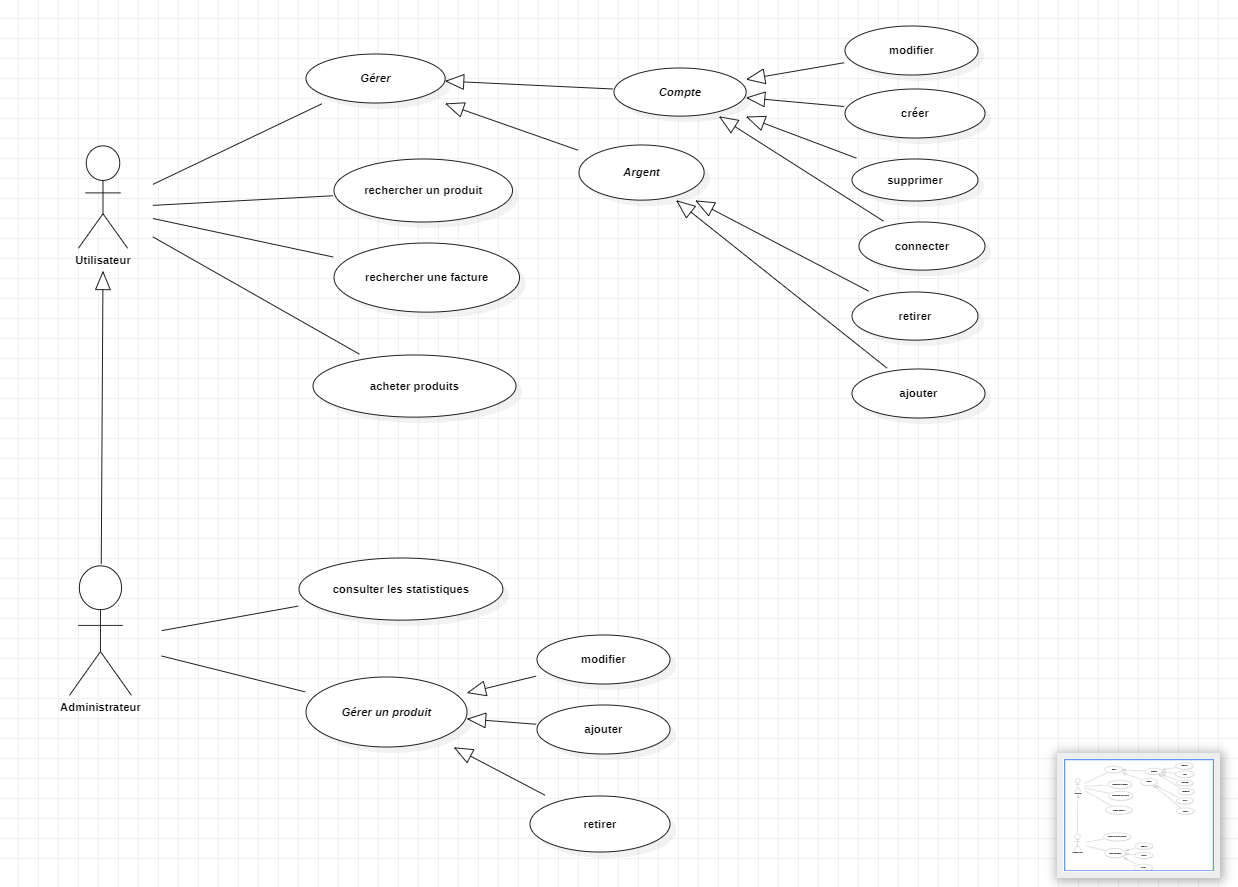


# 

# 2. Conception UML

### 

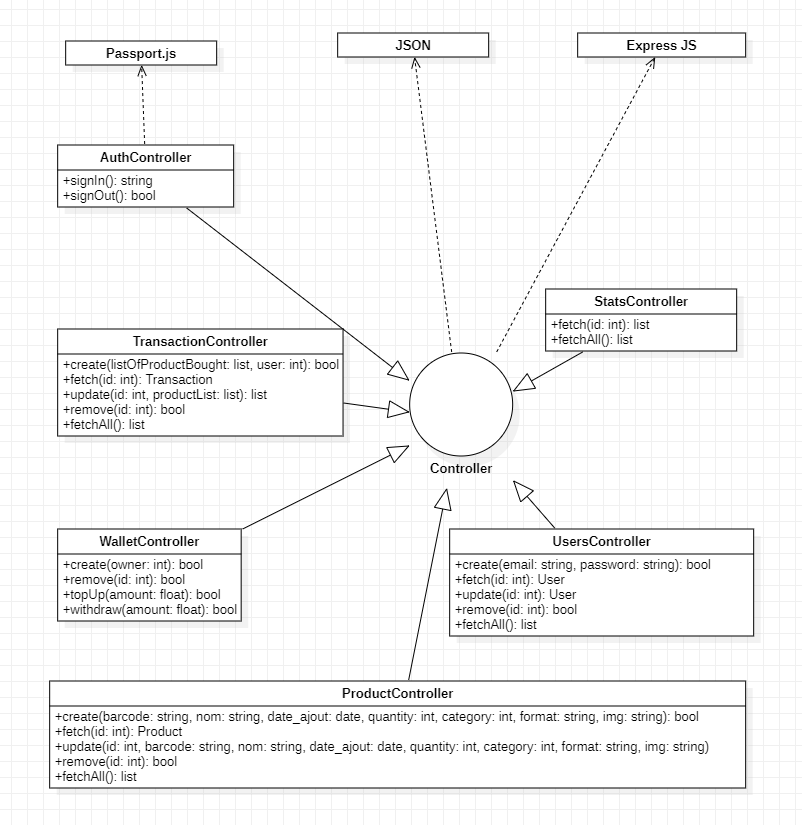
## Diagramme de cas d’usage



### 

### 

## Diagramme de classes

\*Il est à noter ici que certaines fonctions qui sont principalement liées aux calculs des statistiques ou encore aux calculs nécessaires pour l’algorithme se feront directement dans la base de données Postgresql. Elles ne sont donc pas présentes dans ce diagramme de classe qui présente seulement à quoi notre backend Node.js se servant de la librairie Express.js ressemble.

# 

# 3. Structure de données externes

## Diagramme Postgresql

# 

# 4. Éléments de conception

## Structure de données

* **L'ArrayList** est une structure de données très utilisée en programmation, car elle présente plusieurs avantages. Tout d'abord, elle est simple à utiliser et à comprendre. Cette structure de données est basée sur le principe du tableau à taille fixe. Nous avons choisi de l'implémenter par nous même, car elle permet d'ajouter et de supprimer des éléments à tout moment sans affecter les autres éléments présents dans liste. Un autre avantage de l’utiliser est que le temps d'accès à l’information sera le même peu importe la quantité de données dans la liste. En utilisant un système de sous-table et des indices pour situer son contenu, cela permet de calculer rapidement l'emplacement de l'élément dans le tableau en utilisant l'index associé. Enfin, elle peut stocker des objets de tout type de données, ce qui en fait une structure de données très polyvalente. L'ArrayList est une structure de données simple à utiliser et est souple.
* **Linkedlist** est une structure de données qui contient des nœud contenant chacun une valeur ainsi qu’un pointeur ou une référence au prochain qui le suit dans la liste. Cela fait en sorte que la liste est beaucoup plus rapide, car chaque élément connait juste son voisin dans connaître tous les autres dans la liste. Chaque élément est accessible directement par son index. Les avantages de ce type de structure de données est que les éléments de la liste peuvent être enlevés ou ajoutés rapidement sans avoir à déplacer tous les autres éléments contenus dans la liste.
* **La Hashtable** est une structure de données qui se veut être une implémentation du dictionnaire. Elle est une liste dans laquelle les valeurs sont liées à des clés. Elle se veut être très efficace, grâce à ce système de clé, pour facilement faire des recherches, enlever ou ajouter des données à l'intérieur de celle-ci. Dans notre projet, nous comptons utiliser cette structure de données afin de stocker les produits par exemple qui seront affichés dans notre interface graphique. Puisque nous aurons certainement de l'ajout, de la recherche et de la suppression de produit, la hashtable sera alors une solution optimale avec ses fonctions get(), set() et remove().
* **Le String** est une structure de données primitive. Elle peut prendre comme données des caractères et les grouper ensemble. Dans certains langages le string n'est pas considéré comme une structure de donnée, cependant en Javascript celui-ci est bel et bien une structure de donnée. Le String donne accès à plusieurs fonctions utilitaires intéressantes par exemple lors de l'analyse textuelle (indexOf(), split(), charAt(), search(), slice()). Dans notre projet, nous aurons besoin lors de la création de compte de vérifier si les emails et les mots de passe correspondent aux formats habituels. Pour ce faire nous utiliserons certainement les expressions régulières, mais pour vérifier si elles marchent il faudra faire un match(). La fonction match() est justement l'une des fonctions utilitaires de l'objet String. Une autre implémentation de cette structure sera fort probablement présente dans la barre de recherche qui permettra à chaque fois que l'on ajoute un caractère dans la barre de concaténer aux autres déjà présente et faire une recherche à partir de ceux-ci. La fonction concatenate() permet justement de réaliser cette action. Donc le String sera certainement une structure essentielle et importante dans notre projet.

## Patron de design

### Composite

Dans notre projet nous utiliserons comme framework de javascript qui se nomme React. React utilise comme principe un architecture d’interface basé sur des composants qui contiennent des fonctions et des classes qui seront compilées en html et sera affiché sur le site. Cela permet donc facilement d’encapsuler des petites parties d’interface dans des plus grandes parties d’interface qui eux peuvent être utilisées à nouveau avec d’autres petites parties d’interface. Ce principe est à la base même du design pattern de composite. Cela permettra d’éviter de re-programmer au complet des parties de l’interface qui ont déjà été programmées et qui sont similaires et donc améliorera la rapidité avec laquelle on peut construire notre interface. Cela aidera aussi à comprendre facilement la structure de notre interface graphique puisque si l'on structure correctement celle-ci avec ce design pattern, cela devrait ressembler à un arbre que l’on ne fait que remonter pour remonter vers la racine.

### Proxy

Dans notre projet nous allons avoir besoin de faire appel à un système de proxy. Mais avant de commencer qu’est ce qu’un proxy? Une proxy est un serveur qui se situe entre un client et un serveur distant. Lorsque l’utilisateur veut accéder à notre site, il se connecte d’abord au proxy pour lui demander de lui fournir la page web. Il fait alors la requête au serveur web. Le serveur renvoie ensuite la réponse au proxy qui la transfère à au visiteur du site.

Il y a plusieurs raisons d’utiliser un tel système, notamment pour mettre en cache les pages souvent visitées ce qui contribue à la rapidité du site au final. Lors de la demande d’une page, le proxy vérifie si un système de cache est activé, si c’est le cas il répond avec la version caché de la page. S’il ne l’a pas dans le cas d’une page qui n’a pas été visité depuis longtemps ou ce n’est pas une page fixe, il fait la requête au serveur.

L’utilité qui nous sera le plus utilisée sera d'accéder à un même serveur pour plusieurs services en arrière plan. Dans notre projet nous aurons besoin d’avoir accès à notre api avec le même nom de domaine que le site lui-même pour faciliter les droits d'accès par le navigateur. Nous pourrons alors avoir notre api sur le chemin /api et notre site sur le chemin /. La procédure de connexion sera gérée par un service qui ce base sur le patron de conception proxy mais sera réellement un reverse proxy. Contrairement à un proxy traditionnel qui se situe entre un client et un serveur, le reverse proxy se situe entre le serveur Web et les visiteurs de notre site web, en interceptant toutes les demandes et en les envoyant au serveur Web.

### Factory Method

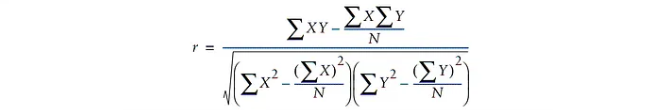
Dans notre projet, nous n’aurons pratiquement aucunement besoin de stocker des données dans notre backend. À vrai dire, la gestion de nos données se fera principalement par des requêtes qui modifieront les données déjà stockées dans la base de données, ajouteront de nouvelles entrées et enlèveront également certaines entrées. Cela se fera cependant pour plusieurs tables différentes qui devront être consultées assez souvent. Ces requêtes seront nécessaires pour mal toutes les tables avec quelques différences dans ce qu’on cherche à retirer vu que les tables ne se ressemblent pas dans ce qu'elles contiennent. Vu que les méthodes sont toutes similaires, il devient donc intéressant de créer une classe interface qui donnera la signature des fonctions qui devront absolument être utilisées par les classes enfants qui feront les requêtes et donnera ainsi à notre code une cohérence. Bien sûr les classes enfant ne feront que override la classe parent afin de combler les petites spécificité lié aux tables que l'ont interrogent. Le design pattern correspondant à cette implémentation est le Factory Method. Ce pattern suggère justement de mettre une classe interface qui contient les fonctions de base de classe qui font la même chose mais qui ont de légères différences dans les attributs de cette classe.

### Observer

Comme dans toute application web, android ou simplement d’ordinateur on retrouve très souvent la présence d'une interface graphique. Cette interface graphique contient des images, du texte, des graphiques, mais également des boîtes pour entrer du texte ou encore des boutons afin que l’utilisateur puisse interagir avec l’application. Cependant, pour pouvoir lire ces intéractions de l'utilisateur avec l’application il faut quelque chose dans l'application voit ces intéractions afin de réagir en conséquence et effectuer les tâches nécessaires lié à l'interaction. Pour ce faire tant en Java qu’en Javascript on utilise ce qu’on appelle des EventListener. Ceux-ci agissent un peu comme observateur et indiquent à l’application quand il y a un événement et quel événement s’est produit. L’application agit ensuite en conséquence. Ceux-ci sont le fruit du design pattern d' observer qui justement règle le problème de lien entre les événements sur l’interface graphique et l’arrière-plan de l’application. Dans notre projet, puisqu’il y aura un interface graphique et un backend, nous devrons donc utiliser des observateurs d’événements afin de savoir lorsqu’un utilisateur clique sur un bouton, un onglet ou un champ.

## Algorithme & Mathématique

Lorsque l’on se trouve sur le web soit sur Amazon ou encore Best Buy et que l’on cherche des produits à acheter on retrouve souvent au bas de la page une liste de produits qui nous sont recommandés afin de faciliter notre recherche ou simplement nous attirer à acheter quelque chose d’autre qui est populaire et risque de nous intéresser. Pour notre application, cette fonctionnalité est assez intéressante. En effet, il serait pratique pour un client de voir les différents produits qui sont souvent achetés afin de donner une suggestion. Cela mousserait probablement les ventes et répondrait au besoin d’efficacité. Afin de pouvoir faire ces recommandations, il faut donc un algorithme qui prendra les informations sur les ventes et selon les goûts des utilisateur donnera un score pour chaque produit afin de déterminer lesquelles correspondrait le mieux au client. Pour ce faire nous comptons utiliser un algorithme de filtrage produit-produit qui utilise comme mathématique pour déterminer le score des produits le calcul de la distance euclidienne.



Ce calcul qui nécessite l’utilisation de la somme, de la racine carré et de l’exposant répond à la nécessité d’avoir des maths qui utilise autres chose que simplement des additions, soustraction, division et multiplication. Cela répond également à la nécessité de posséder un algorithme relativement complexe.

Il est important de mentionner que les calculs liés à cet algorithme seront principalement fait dans la base de données à l’aide de fonctions puisque nous utilisons PostgreSQL.

## Expression régulière

Dans notre projet, nous avons des utilisateurs et des administrateurs qui devront avoir la possibilité de se connecter. Pour ce faire, ceux-ci devront avoir un mot de passe, mais également une adresse courriel. Les adresses courriels et les mot de passe doivent suivre un certain format afin d’être valide et garder une cohérence dans leur stockage. Donc pour ce faire, nous utiliserons les expressions régulières afin de s’assurer que les mots de passe et les courriels qui seront entrés lors de la création du compte correspondent aux formats établis.

# 5.Bibliographie

1. *Fabrique / Factory Method*. (n.d.). Refactoring.guru. Retrieved March 1, 2023, from <https://refactoring.guru/fr/design-patterns/factory-method>
2. *Composite / Composite*. (n.d.). Refactoring.guru. Retrieved March 1, 2023, from https://refactoring.guru/fr/design-patterns/composite
3. ‌*Proxy*. (2014). Refactoring.guru. https://refactoring.guru/design-patterns/proxy
4. *Observer*. (2014). Refactoring.guru. https://refactoring.guru/design-patterns/observer
5. *JavaScript Hash Table – Associative Array Hashing in JS*. (2021, May 11). FreeCodeCamp.org. <https://www.freecodecamp.org/news/javascript-hash-table-associative-array-hashing-in-js/>
6. *JavaScript Data Structures - The String Object*. (n.d.). Www.i-Programmer.info. Retrieved March 1, 2023, from https://www.i-programmer.info/programming/javascript/4138-javascript-data-structures-the-string-object.html
7. *What is Linked List*. (2013, March 8). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-linked-list/
8. ‌*Data Structures in JavaScript – With Code Examples*. (2022, May 16). FreeCodeCamp.org. <https://www.freecodecamp.org/news/data-structures-in-javascript-with-examples/#arrays>
9. Oni, S. (2018, April 30). *Introduction To Recommendation system In Javascript*. Medium. https://becominghuman.ai/introduction-to-recommendation-system-in-javascript-74209c7ff2f7
10. OpenAI. (2022, November 30). *ChatGPT*. Chat.openai.com. <https://chat.openai.com/chat>
11. *Figma*. (2021). Figma. https://www.figma.com/community
12. ‌*Introduction | dbdiagram Docs*. (n.d.). Dbdiagram.io. https://dbdiagram.io/docs