**Application JAM**

**Ghinda Cristina-Maria**

# Introduction

Dans le cadre de la formation « EXPERT EN INFORMATIQUE ET SYSTEME D'INFORMATION », pour pouvoir valider les compétences du BLOC 3 - Conception et développement d’une application informatique, une application (nommé application Jam) doit être modifiée et rendue sous forme d’un dépôt Git ou un Zip sans les dépendances.

Un document avec cinq questions pour nous guider dans les démarches effectuées pour répondre à cette demande a été également fourni.

Ces cinq demandes sont :

1. Migration de l'application Jam Symfony en API :

Nous vous demandons de réaliser une migration de l'application Jam vers une architecture API, afin de permettre la consommation du service depuis divers dispositifs. Vous êtes libre de faire les choix techniques qui vous semblent appropriés. Veuillez étayer avec précision les raisons motivant cette migration dans votre document PDF.

2. Développement d'un front-end pour consommer l'API :

Vous avez la liberté d'utiliser React ou Vue.JS, par exemple, pour développer le front-end qui consomme l'API.

3. Phases de tests de l'application Jam :

Dans votre document PDF, veuillez détailler les différentes phases de tests que vous prévoyez pour l'application Jam, ainsi que leur mise en œuvre effective dans l'application.

4. Gestion d'un changement de version TLS majeur :

Expliquez dans votre document comment vous organisez la transition vers une version

TLS majeure pour l'application Jam.

5. Implémentation d'un filtre à facettes :

Vous devrez mettre en place un filtre à facettes sur le projet, ce qui pourrait nécessiter d'enrichir le modèle de données existant. Veuillez expliquer comment vous comptez réaliser cette implémentation dans votre document PDF.

Dans ce document, nous allons voir en détail les différentes étapes suivis pour répondre à ces questions.

# La migration d'une application Symfony vers une architecture basée sur une API (Application Programming Interface)

Le choix de migrer une application Symfony vers une API peut apporter plusieurs avantages. Par exemple :

1. **La séparation des préoccupations** : En utilisant une API, vous pouvez séparer la logique métier de l'application de l'interface utilisateur. Cela permet de développer différentes interfaces utilisateur (applications web, mobiles, IoT, etc.) qui partagent la même API, facilitant ainsi la maintenance et la mise à jour de l'application.
2. **L’interopérabilité** : Une API bien conçue permet à une application d'interagir plus facilement avec d'autres systèmes, services ou applications tierces. Cela peut être essentiel si nous allons devoir intégrer l’application avec d'autres systèmes ou si nous souhaitons ouvrir vos données **et** fonctionnalités à des partenaires externes.
3. **Évolutivité** : Les API sont plus évolutives et flexibles en termes de gestion de la charge et d'ajout de nouvelles fonctionnalités. Vous pouvez développer de nouvelles fonctionnalités de manière indépendante sans perturber le fonctionnement de l'ensemble de l'application.
4. **Expérience utilisateur améliorée** : Une API bien conçue peut fournir une expérience utilisateur plus rapide et réactive, car elle permet de réduire la charge de traitement côté serveur et d'effectuer des opérations en parallèle.
5. **Réutilisation des données et des fonctionnalités** : En exposant votre application sous forme d'API, vous pouvez réutiliser les données et les fonctionnalités de votre application dans divers contextes, ce qui peut économiser du temps et des ressources de développement.
6. **Sécurité renforcée** : Une API peut offrir un niveau de sécurité supplémentaire en permettant un contrôle plus précis sur les autorisations d'accès aux données et aux fonctionnalités. Vous pouvez mettre en place des mécanismes d'authentification et d'autorisation spécifiques à l'API.
7. **Compatibilité multiplateforme** : Avec une API, vous pouvez développer des clients pour différentes plateformes (iOS, Android, applications web, etc.) tout en offrant une expérience cohérente aux utilisateurs.
8. **Facilitation des mises à jour** : Les mises à jour de l'application peuvent être plus faciles à gérer, car elles peuvent être déployées indépendamment de l'interface utilisateur. Cela réduit le risque d'interruption du service pour les utilisateurs finaux.
9. **Scalabilité** : Les API sont plus faciles à mettre à l'échelle horizontalement pour gérer une augmentation du trafic. Vous pouvez ajouter de nouveaux serveurs ou ressources pour répondre aux besoins en matière de performances.
10. **Adaptabilité aux tendances technologiques** : Les architectures basées sur des API sont bien adaptées aux tendances technologiques actuelles, telles que les applications à architecture microservices, le développement sans serveur (serverless), et les architectures cloud.

**Pour répondre à la demande de migration de l’application Symfony, JAM, vers une architecture basée sur une API,** j’ai fait le choix de me diriger vers DJANGO Rest Framework, une extension populaire et puissante du framework Django qui facilite la création de services web RESTful (Representational State Transfer) en utilisant Python

Pour expliquer ce choix, je vais présenter les avantages que ce framework apporte :

1. **Performance et extensibilité** : Django REST framework se distingue par ses performances élevées et sa capacité à gérer des charges de trafic importantes.
2. **Rapidité de développement** : Django REST framework offre des outils puissants qui simplifient la création de mon API REST. J'apprécie particulièrement les classes de sérialisation qui me permettent de convertir facilement des objets Python en formats de données comme JSON ou XML. De plus, les vues basées sur des classes facilitent la gestion des opérations CRUD (Create, Read, Update, Delete), et les fonctionnalités pour la validation des données et l'authentification sont un vrai plus.
3. **Sécurité** : La sécurité est essentielle pour moi, et Django REST framework le comprend bien. Il intègre des fonctionnalités solides pour protéger mon API, notamment des mécanismes d'authentification robustes tels que l'authentification basée sur des jetons (token-based authentication) et OAuth. Il assure également une protection contre les attaques CSRF (Cross-Site Request Forgery) et autres menaces de sécurité.
4. **Personnalisabilité** : J'aime avoir le contrôle, et c'est l'un des points forts de Django REST framework. Je peux personnaliser chaque aspect de mon API pour qu'elle corresponde exactement à mes besoins spécifiques.
5. **Écosystème et communauté** : Je trouve très bénéfique de travailler avec un framework qui dispose d'une grande communauté de développeurs, de nombreuses ressources en ligne, ainsi que de bibliothèques tierces et de plugins. Cela me fait gagner un temps précieux dans le développement de l’API.
6. **Gestion des versions** : La gestion des versions est essentielle pour garantir la compatibilité avec les clients existants. Avec Django REST framework, je peux gérer différentes versions de mon API de manière fluide.
7. **Compatibilité multiplateforme** : Je sais que mon API sera compatible avec une variété de clients, que ce soient des applications web, des applications mobiles, ou même des clients en ligne de commande.
8. **Testabilité** : La possibilité de mettre en place des tests unitaires et de tests d'intégration est un atout majeur pour assurer la qualité de mon backend.
9. **Évolutivité** : Mon application va évoluer avec le temps, et je suis ravi de savoir que je pourrai développer mon API de manière modulaire pour répondre à ces évolutions.

Pour résumer, Django REST framework est un choix judicieux pour moi afin de développer le backend de cette application. Il permet de créer une API RESTful robuste, personnalisable, tout en garantissant la sécurité, la testabilité, et en profitant de l'écosystème Django."

# Développement d'un front-end pour consommer l'API

En ce qui concerne le developpement d’un front-end qui consommera l’api Django REST, j’ai fait le choix d’utiliser Vue.js avec TypeScript. Cette décision repose sur plusieurs facteurs clés.

Concernant le choix de vue.js :

* **Facilité d'apprentissage** : Vue.js est un framework JavaScript convivial. Ayant déjà une solide base en HTML, CSS et JavaScript, j'ai trouvé son apprentissage particulièrement fluide, ce qui a permis d'accélérer le développement initial de l'application.
* **Réactivité** : La réactivité de Vue.js est impressionnante. La gestion automatique des mises à jour de l'interface utilisateur lorsque les données de l'API changent simplifie grandement la gestion des états dynamiques.
* **Composants réutilisables** : J'apprécie beaucoup le concept de composants réutilisables dans Vue.js. Cela nous permet de créer une interface utilisateur modulaire et de réutiliser efficacement des éléments d'interface dans différentes parties de notre application.
* **Intégration en douceur** : Une autre raison pour laquelle j'ai opté pour Vue.js est sa facilité d'intégration dans des projets existants. Cela signifie que nous pouvons commencer par l'utiliser pour des fonctionnalités spécifiques sans perturber l'ensemble de notre application.
* **Écosystème robuste** : Vue.js dispose d'une communauté active et d'un écosystème croissant de bibliothèques et de plugins. Cela nous offre de nombreuses options pour étendre les fonctionnalités de notre application de manière efficace.
* **Vue CLI** : L'interface en ligne de commande de Vue (Vue CLI) offre des outils puissants pour faciliter le développement, le test et le déploiement de notre application Vue.js. Elle simplifie considérablement le cycle de vie du projet.
* **Flexibilité** : Vue.js est suffisamment souple pour s'adapter à nos besoins. Que nous développions une petite application simple ou une application web plus complexe, Vue.js peut être ajusté en fonction de nos exigences.
* **Documentation solide** : La documentation de Vue.js est extrêmement complète, ce qui facilite la recherche de solutions aux problèmes rencontrés et la résolution de questions techniques.
* **Performance** : Vue.js offre de bonnes performances grâce à ses mécanismes d'optimisation. Cela garantira une expérience utilisateur fluide, même lorsque l'application deviendra plus complexe.

Concernant le choix d’ajout du typescript :

* **Sécurité accrue** : TypeScript est un sur-ensemble de JavaScript qui ajoute la vérification statique des types. Cela signifie que nous pouvons détecter et corriger les erreurs potentielles au moment de la compilation, ce qui améliore considérablement la fiabilité de notre code. Dans le contexte de la consommation d'une API, cela réduit le risque de bugs liés à la manipulation incorrecte des données reçues de l'API.
* **Productivité améliorée** : TypeScript offre une meilleure aide à la saisie, une autocomplétion plus précise et une documentation intégrée dans les éditeurs de code. Cela accélère le développement en réduisant les erreurs de saisie et en offrant une expérience de développement plus fluide.
* **Maintenance simplifiée** : À mesure que notre application se développe, la maintenance devient un enjeu majeur. TypeScript facilite la gestion de projets de grande envergure grâce à une meilleure organisation du code, une réduction des conflits de noms et une documentation plus claire.
* **Collaboration efficace** : TypeScript améliore la lisibilité du code pour les développeurs qui travaillent sur le projet, ce qui facilite la collaboration. En outre, les types explicites rendent le code plus compréhensible pour toute l'équipe de développement.
* **Interopérabilité** : TypeScript est compatible avec JavaScript, ce qui signifie que nous pouvons progressivement ajouter du code TypeScript à notre projet existant. Cela nous donne la flexibilité d'adopter TypeScript à notre propre rythme.
* **Écosystème** : TypeScript bénéficie de la popularité croissante et de l'adoption généralisée. Il est largement pris en charge par des outils, des bibliothèques et des frameworks, y compris Vue.js, ce qui garantit une compatibilité harmonieuse avec notre stack technologique actuelle.
* **Réduction des erreurs** : En détectant les erreurs au moment de la compilation, TypeScript réduit les risques d'erreurs coûteuses et de comportements inattendus dans notre application, ce qui contribue à la robustesse de notre front-end.

En somme, l'ajout de TypeScript à notre front-end basé sur Vue.js renforce la sécurité, la productivité, la maintenance, la collaboration et la qualité de notre code, tout en améliorant la fiabilité de notre application. Je suis convaincu que cette combinaison nous permettra de développer un front-end exceptionnel pour notre application qui consomme une API.

# Phases de tests de l'application Jam

Pour assurer la qualité et la fiabilité d'une application il est essentiel de mettre en place un processus de test rigoureux.

Voici les différentes phases de tests que nous pouvons envisager pour l’application Jam réalisé avec Django REST et Vue.js 3 :

 **Tests Unitaires** :

* **Backend (Django)** :
  + Tests unitaires pour les vues (API endpoints) en utilisant le module de test Django.
  + Tests unitaires pour les fonctions et les services utilisés dans le backend.
* **Frontend (Vue.js 3)** :
  + Tests unitaires pour les composants Vue.js en utilisant des frameworks de test tels que Jest, Vue Test Utils, ou Testing Library Vue.
  + Tests unitaires pour les fonctions utilitaires et les services frontend.

 **Tests d'Intégration** :

* **Backend-Frontend Integration** :
  + Assurez-vous que le frontend et le backend communiquent correctement en testant les interactions entre les deux. Cela peut inclure des tests pour vérifier que les requêtes HTTP sont correctement traitées et que les données sont transmises de manière appropriée.

 **Tests de Compatibilité** :

* Assurez-vous que l'application fonctionne correctement sur différents navigateurs (Chrome, Firefox, Safari, Edge) en effectuant des tests de compatibilité.
* Testez également la compatibilité avec différentes résolutions d'écran et appareils mobiles.

 **Tests de Performance** :

* Mesurez les performances de l'application en utilisant des outils comme Lighthouse, PageSpeed Insights ou WebPageTest.
* Identifiez les goulots d'étranglement de performances et optimisez-les.

 **Tests de Sécurité** :

* Effectuez des tests de sécurité pour identifier les vulnérabilités potentielles de l'application, notamment les failles de sécurité courantes comme les injections SQL, les attaques XSS, etc.
* Utilisez des outils de test de sécurité automatisés et effectuez des audits manuels.

 **Tests de Documentation** :

* Vérifiez que la documentation de l'application (backend et frontend) est complète, précise et à jour.
* Testez les exemples de code et les tutoriels pour vous assurer qu'ils fonctionnent correctement.

 **Tests de Gestion des Erreurs** :

* Testez la gestion des erreurs et des exceptions à tous les niveaux de l'application, en veillant à ce que les messages d'erreur soient informatifs et utiles pour les utilisateurs.

En mettant en œuvre des phases de tests de ce genre, nous pouvons garantir la qualité, la sécurité et les performances de notre application Django REST avec Vue.js 3 en TypeScript, offrant ainsi une expérience utilisateur fiable et optimale.

# Gestion d'un changement de version TLS majeur

# Implémentation d'un filtre à facettes

Un filtre à facettes (ou facette) est un moyen de permettre aux utilisateurs de filtrer et de trier des données de manière interactive dans une interface utilisateur, généralement sous forme de liste ou de tableau. Ce type de fonctionnalité est couramment utilisé dans les applications Web pour faciliter la recherche et la navigation à travers de grandes quantités de données.

La mise en place d’un filtre à facette permet aux utilisateurs de parcourir et de trouver rapidement et facilement les produits de confiture en fonction de leurs préférences, de leur budget, de leurs besoins alimentaires spécifiques, de la disponibilité des produits, etc. Lorsque les utilisateurs sélectionnent un ou plusieurs filtres, la liste des produits affichée est mise à jour en temps réel pour correspondre à leurs critères de recherche.

Pour garantir la mise en place d'un filtre à facettes assurant une expérience utilisateur performante, il a été nécessaire d'enrichir la base de données initiale. Dans la suite, nous examinerons à la fois la base de données initiale et la nouvelle base de données enrichie, ainsi que les étapes suivies pour générer cette dernière.

# Base de données existante, création d’une nouvelle base de données et améliorations apportées

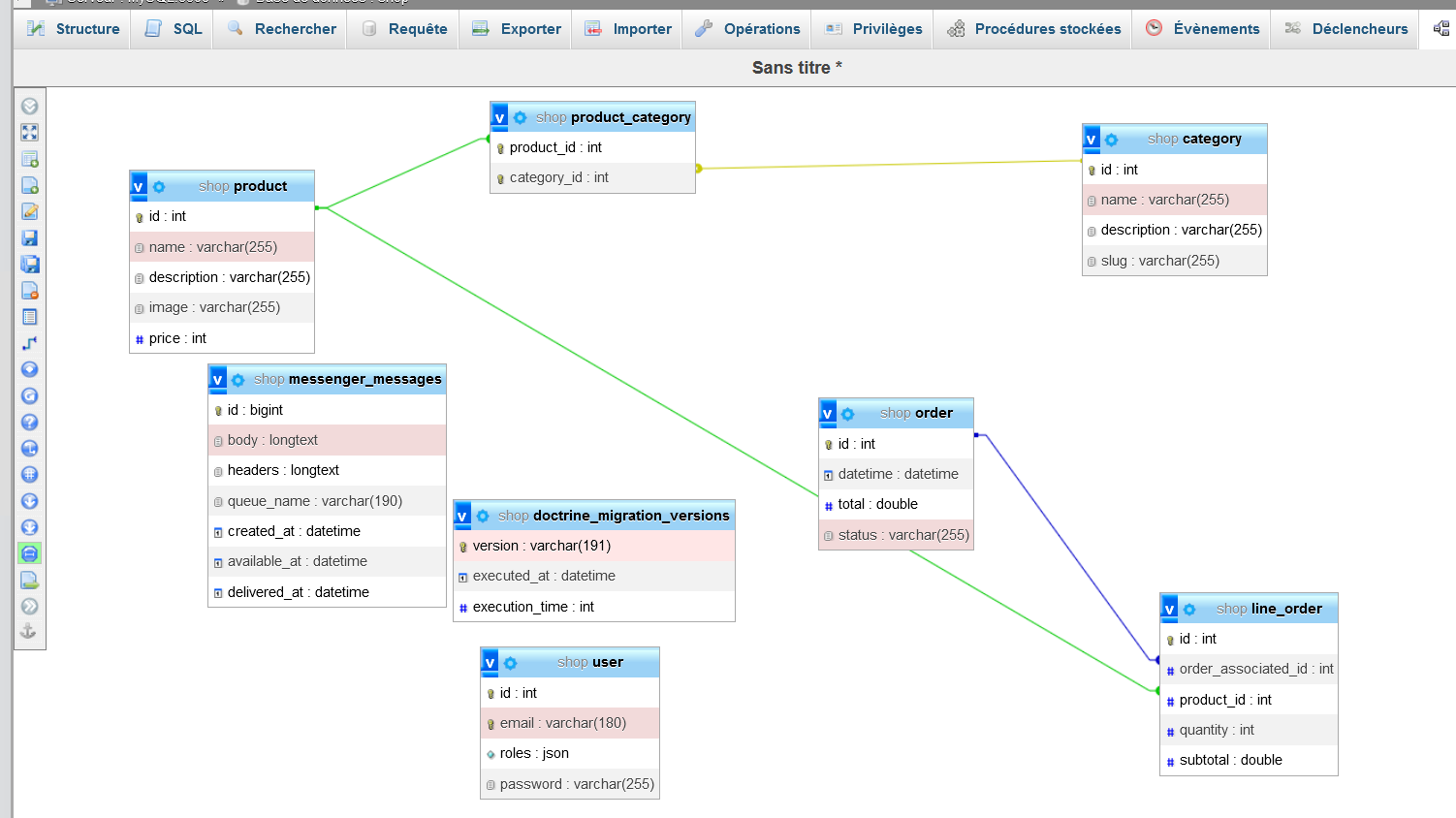
Pour construire une nouvelle base de données pour cette application, j’ai choisi d’utiliser le logiciel MySQL Workbench (un outil de conception de base de données graphique, pour concevoir le modèle de base de données).

A partir d’un modèle conçu dans MySQL Workbench nous allons pouvoir créer une base de données dans le cadre de notre backend DJANGO, en suivant les étapes suivantes :

1. **Conception du modèle de base de données dans MySQL Workbench :** Dans MySQL Workbench, nous concevons notre modèle de base de données. Cela implique la création de la base de données, des tables, la définition de colonnes, la mise en place de relations entre les tables, la spécification des clés primaires et des clés étrangères, ainsi que d'autres propriétés nécessaires pour notre base de données. De plus, nous insérons des données initiales dans les tables si nécessaire.
2. **Exportation du modèle MySQL Workbench :** Une fois que la conception est terminée, nous exportons le modèle depuis MySQL Workbench vers un script SQL. Ce script SQL contient toutes les instructions nécessaires pour créer la structure de la base de données, y compris les tables et les contraintes.
3. **Configuration de la base de données dans Django :** Dans notre projet Django, nous configurons la connexion à la base de données en modifiant les paramètres appropriés dans le fichier settings.py. Nous spécifions le moteur de base de données (MySQL), le nom de la base de données, le nom d'utilisateur, le mot de passe, l'hôte et le port de la base de données MySQL. Dans MySQL Workbench, nous pouvons utiliser l'option "Forward Engineering" pour établir la connexion à notre application Django et pour générer la structure de la base de données à partir de notre modèle. Cela simplifie le processus de création de la base de données en synchronisant le modèle de données de notre application Django avec la base de données MySQL, garantissant ainsi la cohérence entre la structure de la base de données et notre modèle conceptuel. Une fois que cette configuration est correctement réalisée, Django peut créer, gérer et interagir avec la base de données de manière transparente, facilitant ainsi le développement de notre application.
4. **Création de modèles Django :** Nous créons des modèles Django qui correspondent aux tables que nous avons conçues dans MySQL Workbench. Chaque modèle Django est une classe Python qui définit les champs de la table et les relations avec d'autres tables. Nous utilisons le module models de Django pour définir ces modèles. Si nous avons utilisé le Forward engeneering, nous pouvvons générer les models à partir de la bdd Myqsl Workech, avec la comande python3.7 manage.py inspectdb > back\_app/models.py
5. **Génération de migrations :** Nous utilisons la commande python manage.py makemigrations pour générer des fichiers de migration Django basés sur nos modèles. Ces fichiers contiennent les instructions nécessaires pour créer les tables correspondantes dans la base de données.
6. **Application des migrations :** À l'aide de la commande python manage.py migrate, nous appliquons les migrations pour créer les tables réelles dans la base de données MySQL. (à exécuter cette étape si on ne fait pas le forward engineering de l’tape 3)
7. **Génération de fixtures :** Une fois que la base de données est prête, nous pouvons générer des fixtures Django à partir des données introduites dans les tables de MySQL Workbench. Nous utilisons la commande python manage.py dumpdata --format json > back/fixtures/initial-data.json. Les fixtures sont des fichiers JSON, XML ou YAML contenant des données initiales pour notre application. Nous pouvons ensuite insérer ces données initiales dans la base de données à l'aide de la commande python /back/manage.py loaddata initial-data.json. À noter que si nous ajoutons de nouvelles données ou apportons des modifications au fichier de fixtures, Django est intelligent et saura préserver les données existantes dans les tables. Il effectuera uniquement les mises à jour nécessaires et ajoutera de nouvelles données lorsque cela est requis, sans supprimer les données préexistantes. Cette fonctionnalité garantit que nos données existantes restent intactes tout en permettant des mises à jour en douceur.
8. **Utilisation de la base de données :** Avec la structure de la base de données en place, nous pouvons utiliser Django pour interagir avec la base de données. Nous pouvons créer, lire, mettre à jour et supprimer des données en utilisant les modèles Django et les requêtes SQL générées automatiquement par Django.

Ce processus de création de base de données à partir d'un modèle MySQL Workbench dans Django permet de bénéficier de l'interface graphique conviviale de MySQL Workbench pour la conception initiale, tout en utilisant la puissance de Django pour la gestion de la base de données et le développement d'applications Web.

La base de données existante dans l’application JAM Symfony a la structure présentée dans l’image suivante :



Comme pour répondre à la question 5, un filtre à facettes doit être mise en place et le modèle de données existant doit être enrichie, la base de données que je vais utiliser pour cette application aura la structure suivante :

Une image contenant texte, capture d’écran, nombre, diagramme

Description générée automatiquement

Un filtre à facettes est une fonctionnalité de recherche et de filtrage avancée qui permet aux utilisateurs de raffiner leurs résultats en fonction de critères spécifiques.

À partir de notre nouvelle base de données, nous avons la possibilité de mettre en place une gamme complète de filtres, offrant ainsi une expérience utilisateur encore plus riche et personnalisée. Voici quelques exemples de filtres que nous pouvons proposer :

* **Filtre par Catégorie** :
  + On va se servir de la table "Category" pour montrer une liste des catégories disponibles.
  + Quand un utilisateur choisit une catégorie, on trie les produits en utilisant la clé étrangère "category" dans la table "Product".
* **Filtre par Marque** :
  + Pour ça, on utilise la table "Brand" pour présenter une liste de marques dispo.
  + Quand l'utilisateur sélectionne une marque, on trie les produits en utilisant la clé étrangère "brand" dans la table "Product".
* **Filtre par Saveur** :
  + On affiche une liste des saveurs dispo en se servant de la table "Flavor".
  + Quand un utilisateur choisit une saveur, on trie les produits avec la clé étrangère "flavor" dans la table "Product".
* **Filtre par Type de Contenant** :
  + On montre une liste des types de contenant avec la table "TypeContenant".
  + Quand l'utilisateur sélectionne un type de contenant, on trie les produits avec la clé étrangère "type\_contenant" dans la table "Product".
* **Filtre par Ingrédient** :
  + Si on veut que les utilisateurs puissent trier par ingrédient, on peut utiliser la table "Ingredient".
  + Pour ça, on crée une relation entre la table "Product" et "Ingredient" via la table "ProductHasIngredient". Quand l'utilisateur choisit un ingrédient, on trie les produits qui contiennent cet ingrédient grâce à la table de liaison "ProductHasIngredient".
* **Filtre par Prix** :
  + Pour que les utilisateurs puissent définir un prix min et max, on utilise le champ "price" dans la table "Product". Ils pourront ainsi trier les produits en fonction du prix.
* **Filtre par Promotion** :
  + Si on veut aider les utilisateurs à trouver des produits en promo, on peut se servir du champ "promotion" dans la table "Product".
* **Filtre par Stock Disponible** :
  + Pour ça, on utilise la table "StockDisponible" pour présenter les options de stock disponibles.
  + Quand l'utilisateur choisit une option, on trie les produits avec la clé étrangère "stock\_disponible" dans la table "Product".