



Dossier de soutenance au titre de Concepteur Développeur d'Applications



Projet présenté :

Nooky est une application destinée aux particuliers qui permet d'échanger des vêtements, chaussures et accessoires de seconde main.

Abstract

Nooky is an application dedicated to fashion exchanges. The goal is to give a second chance to clothes, shoes, and accessories that still reside in our wardrobes but are no longer in use.

Designing and implementing this app has been an enlightening journey in my growth as a software developer. I undertook several key steps, which I'll describe below.

Firstly, I focused on defining the requirements. To comprehend what users would need and enjoy in this future app, I employed the personas marketing technique. This approach facilitated the creation of a comprehensive list of user stories, encapsulating all the functionalities of Nooky.

To maintain project oversight and organizational efficiency, I utilized Trello and adopted the Kanban method, one of the Agile frameworks.

Subsequently, I proceeded to design the application. Graphically, this entailed creating zoning plans, wireframes, and mock-ups. Additionally, I formulated the data model using the Merise method. I delineated the processes for handling this data and defined the architecture while selecting the technology stack: PostgreSQL for the database, the Spring framework for the backend, and the SvelteKit framework for the frontend.

With these foundations in place, I commenced implementation of the functionalities. For the database layer, I initially crafted an SQL script, which I later denormalized for performance optimization. This script was instrumental in database creation. Subsequently, I developed two additional scripts: one for the referential data of the app and another to facilitate testing functionalities (such as fictive users and products).

For the business layer, I structured codes into six packages to distribute responsibilities and ensure code maintainability and security. These packages were ‘config’, ‘controllers’, ‘dto’, ‘entities’, ‘repositories’, and ‘services’. As for the client layer, I organized code into four folders: ‘assets’ for CSS, images, and fonts; ‘components’ for reusable graphical elements; ‘lib’ for client-layer functions; and ‘routes’ for application pages.

I also paid meticulous attention to security aspects during the implementation of functionalities. For the registration and authentication segments of the application, I utilized JWT and bcrypt. When a user registers with Nooky, the password is securely hashed with bcrypt and stored in the database. Upon user login, a JWT is transmitted from the backend to the frontend and stored in the local storage. Furthermore, to secure the application's endpoints, I integrated Spring Security. In my quest for deeper understanding, I researched SQL injections and implemented precautions to mitigate them.

Last but not least, I rigorously tested all functionalities. For the backend, I conducted integration tests using Postman, while for the frontend, I performed end-to-end tests with Playwright. Additionally, upon completion of each functionality, I conducted non-regression tests and User Acceptance Testing (UAT).

This project served as an immensely valuable training ground, enabling me to gain insights and skills in designing and implementing an informatics project. While there are aspects I would have approached differently given more time, I'm largely satisfied with the wealth of knowledge I've acquired throughout this process.

Compétences mises en œuvre dans le cadre du projet

Le projet Nooky réalisé dans le cadre de la formation CDA chez Simplon m'a permis d'aborder les blocs de compétences ci-dessous :

Développer une application sécurisée	Développer des interfaces utilisateur	X
	Développer des composants métier d'une application	X
	Contribuer à la gestion d'un projet informatique	X
Concevoir et développer une application sécurisée organisée en couches	Analyser les besoins et maquetter une application	X
	Définir l'architecture logicielle d'une application	X
	Concevoir et mettre en place une base de données relationnelle	X
	Développer des composants d'accès aux données SQL et NoSQL	X
Préparer le déploiement d'une application sécurisée	Préparer et exécuter les plans de tests d'une application	X

Table des matières

Abstract.....	2
Compétences mises en œuvre dans le cadre du projet.....	3
1 Expression des besoins.....	6
1.1 Description du projet.....	6
1.2 Problématique.....	6
1.3 Contraintes et objectifs.....	6
1.4 Agilité.....	7
1.5 L'analyse des besoins.....	7
1.5.1 Les personas.....	7
1.5.1.1 Lucie, passionnée de mode et éco-responsable.....	8
1.5.1.2 Maxime, l'étudiant à petit budget.....	8
1.5.1.3 Sophie, la maman organisée.....	9
1.5.2 Les fonctionnalités.....	9
1.5.3 Acteurs du projet.....	10
1.5.3.1 Acteur non-enregistré.....	10
1.5.3.2 Acteur authentifié.....	10
1.5.4 Les user stories.....	10
1.5.4.1 Les fonctionnalités liées aux comptes utilisateurs.....	10
1.5.4.2 Les fonctionnalités liées au produit.....	11
1.5.4.3 Les fonctionnalités de recherche.....	11
1.5.4.4 Les fonctionnalités propres aux échanges.....	12
2 Gestion du projet.....	14
2.1 Kanban.....	14
2.2 Definition of done.....	14
2.3 Outil trello.....	14
3 La conception.....	17
3.1 La conception graphique.....	17
3.1.1 Le zoning.....	17
3.1.2 Le wireframe ou maquette fonctionnelle.....	18
3.1.3 L'arborescence du site.....	18
3.1.4 La charte graphique.....	19
3.1.5 La maquette statique.....	20
3.2 La conception des données.....	20
3.2.1 Le MCD.....	21
3.2.2 Le MLD.....	22
3.2.3 Le MPD.....	23
3.3 Les traitements.....	23
3.3.1 Cas d'utilisation.....	23
3.3.2 Diagramme de classes.....	25
3.3.3 Diagramme de séquences.....	26
3.4 L'architecture.....	27
3.4.1 SvelteKit TS.....	27
3.4.2 Java Spring.....	28
3.4.3 PostgreSQL.....	28
4 L'implémentation.....	29
4.1 Client.....	29
4.1.1 SPA.....	29

4.1.2 Structure du projet.....	29
4.1.3 Point d'entrée de l'application.....	29
4.1.4 Le routing.....	30
4.1.5 Les composants graphiques.....	31
4.1.6 Envoi et récupération des données API.....	32
4.1.7 Objets et fonctions utiles.....	32
4.1.8 Exemple d'une fonctionnalité.....	32
4.2 Métier.....	34
4.2.1 Structure du projet.....	34
4.2.2 Les classes entités.....	35
4.2.3 DTO.....	36
4.2.3.1 Les DTO d'entrées.....	36
4.2.3.2 Les DTO de sorties.....	37
4.2.4 Controllers.....	37
4.2.5 Services.....	38
4.2.6 Les ORM.....	39
4.2.7 Repositories.....	40
4.2.8 Sécurité.....	40
4.2.8.1 Configuration des CORS.....	40
4.2.8.2 Sécurisation des routes.....	41
4.2.8.3 JWT.....	42
4.2.8.4 Bcrypt.....	42
4.3 Données.....	43
4.3.1 Le script de création des tables.....	43
4.3.2 Le script d'insertion des données de références.....	44
4.3.3 Le script d'insertion des données de tests.....	45
5 Les tests.....	46
5.1 Tests d'intégration back-end.....	46
5.2 Tests de non-régression.....	47
5.3 Tests des composants graphiques.....	47
5.4 Tests UAT sur Chrome et Mozilla.....	48
6 La veille sécurité – Injection SQL.....	50
6.1 Définition.....	50
6.2 Types d'injection SQL.....	50
6.3 Adaptation du code.....	52
6.3.1 DTO.....	52
6.3.2 JPA.....	52
6.3.3 Requêtes paramétrées.....	52
6.4 Les sources.....	53
Conclusion.....	54
Bibliographie.....	55
Remerciements.....	56
Annexes.....	57

1 Expression des besoins

1.1 Description du projet

Dans le cadre de la formation de Concepteur Développeur d'Applications Java/JavaScript dispensée au sein de l'école Simplon, j'ai travaillé sur la réalisation d'une application dédiée à la valorisation des objets de seconde main.

Le but de cette application est de permettre aux particuliers d'échanger des vêtements, chaussures et accessoires.

Ainsi, un usager inscrit sur la plateforme peut ajouter des produits à son vestiaire, exprimer le souhait d'échanger ses produits contre des produits spécifiques, faire des propositions d'échange et rechercher dans la base du site les produits proposés par les autres utilisateurs.

L'objectif de cette application est de donner une nouvelle vie à des objets que nous n'utilisons plus en les échangeant contre des objets dont nous avons besoin.

1.2 Problématique

Avec les problématiques environnementales auxquelles nous faisons face, de nombreux particuliers aimeraient donner une nouvelle vie aux objets dont ils n'ont plus besoin. Il existe de nombreuses solutions pour le don ou la revente de ces objets, moins pour les échanges.

L'application Nooky se propose de remédier à cela.

1.3 Contraintes et objectifs

Les objectifs du projet Nooky sont définis en tenant compte des contraintes, tout en visant à répondre aux besoins des utilisateurs et à leur offrir une expérience satisfaisante.

En effet, en tant que projet réalisé dans le cadre de ma formation de Concepteur Développeur d'Applications Java/JavaScript à l'école Simplon, je suis soumise à des contraintes budgétaires et des contraintes de temps. Ces limitations nécessitent une gestion efficace des ressources disponibles pour garantir la réalisation du projet dans les délais impartis et avec les moyens alloués.

Je dois donc adapter mon approche de développement. Cela se traduit par une priorisation des fonctionnalités essentielles qui permettent d'atteindre rapidement un produit viable et fonctionnel. Ainsi, je serai en mesure de concentrer mes efforts sur les aspects les plus critiques du projet, tout en laissant la porte ouverte à des évolutions futures.

Les objectifs que je me suis fixée sont les suivants :

- Développer un prototype de type MVP
- Tester l'application

Dans un second temps, j'aimerais déployer l'application et la proposer à plusieurs associations dont je suis membre afin d'avoir un premier retour utilisateur.

1.4 Agilité

J'ai travaillé sur ce projet en appliquant l'agilité.

Pour bien comprendre le besoin des futurs utilisateurs, mettre à plat mes idées et développer l'application, j'ai fait appel à un certain nombre d'outils bien spécifiques.

Tout d'abord, j'ai utilisé la méthode des personas pour définir des profils types d'utilisateurs, ce qui m'a permis de mieux comprendre leurs besoins, leurs motivations et leurs comportements.

Ensuite, j'ai utilisé des user stories pour capturer les besoins des utilisateurs sous forme de scénarios simples. Cela m'a aidé à garder l'utilisateur au centre du processus de développement et à m'assurer que chaque fonctionnalité ajoutée apportait une réelle valeur.

Par ailleurs, nous le verrons un peu plus tard, j'ai utilisé la méthode Kanban pour organiser et visualiser le flux de travail. Un tableau Kanban m'a permis de suivre l'avancement des tâches et de prioriser les activités les plus importantes.

Également, la notion de MVP (Most Valuable Product) a guidé le développement de l'application. Je me suis concentrée sur la livraison des fonctionnalités essentielles pour répondre aux besoins de base des utilisateurs dès le premier déploiement.

Enfin, j'aborderais ce sujet un peu plus loin, j'ai défini une DOD (Definition Of Done). En d'autres termes les critères qui devaient être remplis pour considérer la tâche comme terminée et prête à être livrée.

Tout ceci m'a permis de garder en tête la philosophie agile tout au long du projet et être dans une démarche de livraison continue de valeur.

1.5 L'analyse des besoins

1.5.1 Les personas

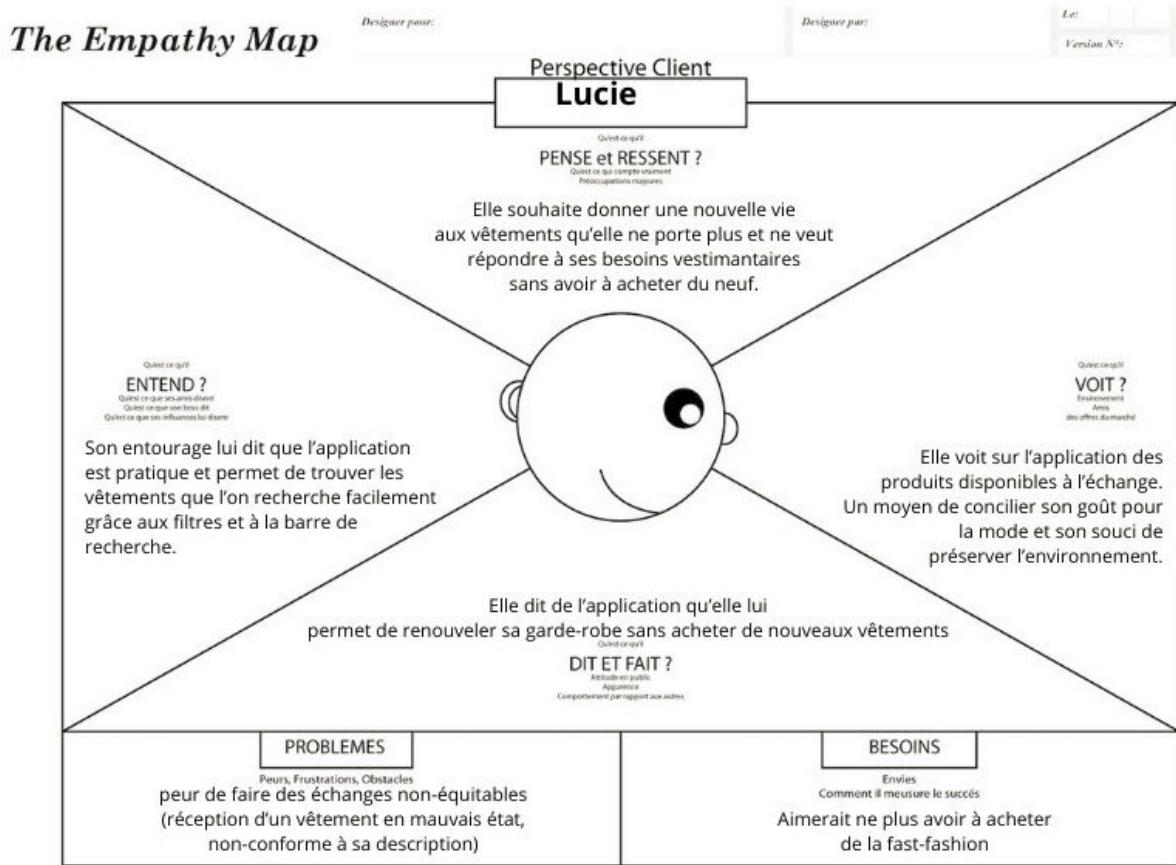
La méthode marketing des personas consiste à définir des profils types d'utilisateurs afin de mieux comprendre leurs besoins, leurs motivations et leurs comportements. Cette approche permet de concevoir une application qui sera plus centrée sur l'utilisateur.

J'ai identifié plusieurs personas et pour appréhender la façon dont ils pourraient interagir avec l'application, j'ai réalisé des empathy maps.

1.5.1.1 Lucie, passionnée de mode et éco-responsable

Lucie est soucieuse de l'environnement et cherche activement des moyens de réduire son empreinte écologique.

Elle souhaite échanger ses vêtements et accessoires pour leur donner une seconde vie plutôt que de les jeter ou de les laisser dormir dans son placard.



1.5.1.2 Maxime, l'étudiant à petit budget

Maxime est étudiant et dispose d'un budget limité pour ses achats vestimentaires.

Il recherche des vêtements de qualité à moindre coût et voit dans l'échange une opportunité d'obtenir de nouveaux articles en dépensant seulement la somme nécessaire à l'envoi du produit.

L'empathy map de Maxime est disponible en annexes, p. 58.

1.5.1.3 Sophie, la maman organisée

Sophie est maman de quatre enfants et travaille. Elle gère les besoins vestimentaires de toute sa famille.

Elle voit dans l'application Nooky une façon pratique de renouveler la garde-robe de ses enfants sans avoir à acheter de nouveaux vêtements à chaque saison.

L'empathy map de Sophie est disponible en annexes, p. 59.

1.5.2 Les fonctionnalités

J'ai listé les fonctionnalités essentielles, puis je les ai organisées en quatre grands blocs.

Les fonctionnalités liées aux comptes utilisateurs :

- Créer un compte utilisateur
- S'authentifier
- Modifier son compte utilisateur
- Supprimer son compte utilisateur

Les fonctionnalités liées aux produits :

- Ajouter un produit dans son vestiaire utilisateur
- Modifier un produit de son vestiaire
- Supprimer un produit de son vestiaire

Les fonctionnalités de recherche :

- Rechercher des produits
- Filtrer les produits
- Consulter les produits disponibles dans le vestiaire d'un autre utilisateur

Les fonctionnalités propres aux échanges :

- Effectuer des propositions d'échange
- Recevoir des propositions d'échange
- Accepter une proposition d'échange
- Refuser une proposition d'échange
- Annuler un échange en cours
- Recevoir les informations nécessaires à l'envoi d'un produit
- Valider la bonne réception d'un produit
- Signaler la non-réception d'un produit

1.5.3 Acteurs du projet

Les seuls utilisateurs présents lors du développement du MVP seront des clients (User).

1.5.3.1 Acteur non-enregistré

Les acteurs non-enregistrés pourront accéder à l'application, consulter les fiches produits, les profils des utilisateurs et rechercher des articles. Cependant, ils ne pourront pas proposer d'échanges ni créer de vestiaire.

1.5.3.2 Acteur authentifié

Les utilisateurs authentifiés de l'application Nooky auront accès à toutes les fonctionnalités après s'être connectés. Ils seront les utilisateurs finaux de l'application. Ils pourront créer un vestiaire associé à leur compte ainsi qu'échanger des vêtements, des chaussures et des accessoires de seconde main.

1.5.4 Les user stories

Les user stories sont des descriptions simples et concises de fonctionnalités du point de vue de l'utilisateur final. Elles permettent de comprendre les besoins et les attentes des utilisateurs, facilitant ainsi la conception et le développement de l'application.

1.5.4.1 Les fonctionnalités liées aux comptes utilisateurs

Création d'un compte utilisateur

En tant qu'utilisateur, je veux pouvoir créer un compte sur l'application Nooky en fournissant des informations telles que mon nom, mon adresse e-mail et un mot de passe, afin de pouvoir publier des articles à échanger et interagir avec d'autres utilisateurs.

Authentification

En tant qu'utilisateur, je veux pouvoir m'authentifier sur l'application Noky à l'aide de mon adresse e-mail et de mon mot de passe, afin d'accéder à mon compte personnel et aux fonctionnalités de l'application.

Consultation du compte

En tant qu'utilisateur je souhaite pouvoir consulter mon compte et mon vestiaire associé afin de vérifier que les informations sont à jour.

Modification du compte utilisateur

En tant qu'utilisateur, je veux pouvoir modifier les informations de mon compte telles que mon e-mail ou mon adresse de livraison, afin de maintenir mes données à jour et de personnaliser mon expérience utilisateur.

Suppression du compte utilisateur

En tant qu'utilisateur, je souhaite pouvoir supprimer mon compte afin de ne plus utiliser l'application.

1.5.4.2 Les fonctionnalités liées au produit

Ajouter un produit dans son vestiaire utilisateur

En tant qu'utilisateur, je veux pouvoir ajouter des produits que je souhaite échanger à mon vestiaire en incluant une photo, des descriptions détaillées et les produits que je recherche en échange, afin de les rendre visibles aux autres utilisateurs.

Modifier un produit de son vestiaire

En tant qu'utilisateur, je souhaite pouvoir modifier les informations d'un produit de mon vestiaire afin de compléter les informations ou retirer d'éventuelles informations erronées.

Supprimer un produit de son vestiaire

En tant qu'utilisateur, je veux pouvoir supprimer un produit de mon vestiaire sur l'application afin de retirer des articles que je ne souhaite plus échanger.

1.5.4.3 Les fonctionnalités de recherche

Rechercher des produits

En tant qu'utilisateur, je veux pouvoir rechercher des produits spécifiques à l'aide d'une barre de recherche afin de trouver des articles qui correspondent à mes besoins et à mes préférences.

Filtrer les produits

En tant qu'utilisateur, je veux pouvoir filtrer les résultats de recherche par catégories, tailles et types afin de trouver plus rapidement les articles qui m'intéressent.

Consulter les produits disponibles dans le vestiaire d'un autre utilisateur

En tant qu'utilisateur, je veux pouvoir consulter l'ensemble des produits disponibles dans le vestiaire d'un autre utilisateur afin d'identifier des articles qui pourraient m'intéresser pour un échange.

1.5.4.4 Les fonctionnalités propres aux échanges

Effectuer des propositions d'échange

En tant qu'utilisateur, je veux pouvoir proposer un échange de produits à un autre utilisateur en sélectionnant un ou plusieurs articles de mon vestiaire et en les soumettant pour considération afin d'initier un échange.

Recevoir des propositions d'échange

En tant qu'utilisateur, je veux pouvoir recevoir les nouvelles propositions d'échange par mail afin de rester informé des activités liées à mon compte.

Accepter une proposition d'échange

En tant qu'utilisateur, je veux pouvoir accepter une proposition d'échange reçue afin de formaliser l'accord et procéder à l'échange des produits.

En tant qu'utilisateur qui a effectué une proposition d'échange, je souhaite être informé par mail que la transaction a été acceptée afin de formaliser l'accord et procéder à l'échange des produits.

Refuser une proposition d'échange

En tant qu'utilisateur, je veux pouvoir refuser une proposition d'échange reçue afin de décliner une offre qui ne m'intéresse pas ou ne correspond pas à mes attentes.

En tant qu'utilisateur qui a effectué une proposition d'échange, je souhaite être informé par mail que ma proposition a été refusée afin de pouvoir proposer mon produit à d'autres utilisateurs.

Annuler un échange en cours

En tant qu'utilisateur, je souhaite pouvoir annuler un échange afin de répondre à un imprévu éventuel.

En tant qu'utilisateur, je souhaite être informé par mail de l'annulation d'un échange afin de pouvoir proposer mon produit aux autres utilisateurs.

Recevoir les informations nécessaires à l'envoi d'un produit

En tant qu'utilisateur, je veux pouvoir recevoir par mail l'adresse de livraison du produit afin de conclure l'échange.

Valider la bonne réception d'un produit

En tant qu'utilisateur qui a reçu un produit, je veux pouvoir confirmer la réception d'un produit échangé sur l'application afin de notifier l'autre utilisateur que l'échange s'est bien déroulé et que le produit est arrivé à bon port.

En tant qu'utilisateur qui a envoyé un produit, je souhaite être informé par mail de la bonne réception du colis envoyé afin de conclure l'échange.

Signaler la non-réception d'un produit

En tant qu'utilisateur, je veux pouvoir signaler la non-réception d'un produit attendu afin de trouver une solution à la transaction incomplète.

En tant qu'utilisateur qui a envoyé un produit, je souhaite être informé par mail de la non-réception du colis envoyé afin de trouver une solution à la transaction incomplète.

2 Gestion du projet

2.1 Kanban

Kanban est une méthode de gestion de projet visuelle qui vise à optimiser le flux de travail. Elle repose sur un tableau divisé en colonnes représentant les différentes étapes du processus, de la planification à la livraison. Les tâches à réaliser sont représentées par des cartes, qui sont déplacées à travers les colonnes en fonction de leur progression. Kanban permet une gestion efficace des priorités, une visualisation claire de l'avancement du projet et une identification rapide des goulets d'étranglement.

Cette façon de gérer le projet de Nooky m'a semblé la plus appropriée étant donné que c'est une application que j'ai conçue et développé seule.

2.2 Definition of done

La Definition of Done (DOD) est une liste de critères définissant les conditions à remplir pour qu'une tâche soit considérée comme terminée. Ces critères peuvent inclure des aspects tels que les tests unitaires effectués, la documentation complétée, le code révisé et approuvé, et toute autre exigence spécifique au projet. La DOD garantit la qualité et la complétude du travail réalisé, aidant ainsi à maintenir des standards élevés tout au long de la réalisation.

Ici, la DOD que je me suis fixée est la suivante.

Pour qu'une fonctionnalité puisse être considérée comme terminée, elle doit répondre aux critères suivants :

- doit être conçue avec un souci de performance
- doit être conçue avec un souci d'éco-conception
- doit être testée (tests d'intégration back-end, tests de non-régression, tests UAT)
- les écrans IHM concernés doivent être responsives
- doit être conforme aux spécifications fonctionnelles initiales

2.3 Outil trello

Trello est un outil de gestion de projet basé sur le principe du tableau Kanban. Il permet de créer des boards personnalisables avec des listes et des cartes afin d'organiser les tâches et collaborer en équipe. Il est particulièrement utile dans le cadre d'un projet géré en agilité et c'est l'outil que j'ai choisi pour avancer dans le développement de l'application Nooky.

Ici, j'ai utilisé un code couleur qui permet d'identifier rapidement le bloc de travail auquel appartient le ticket. Les blocs correspondent aux quatre grands thèmes dans lesquels ont été classées

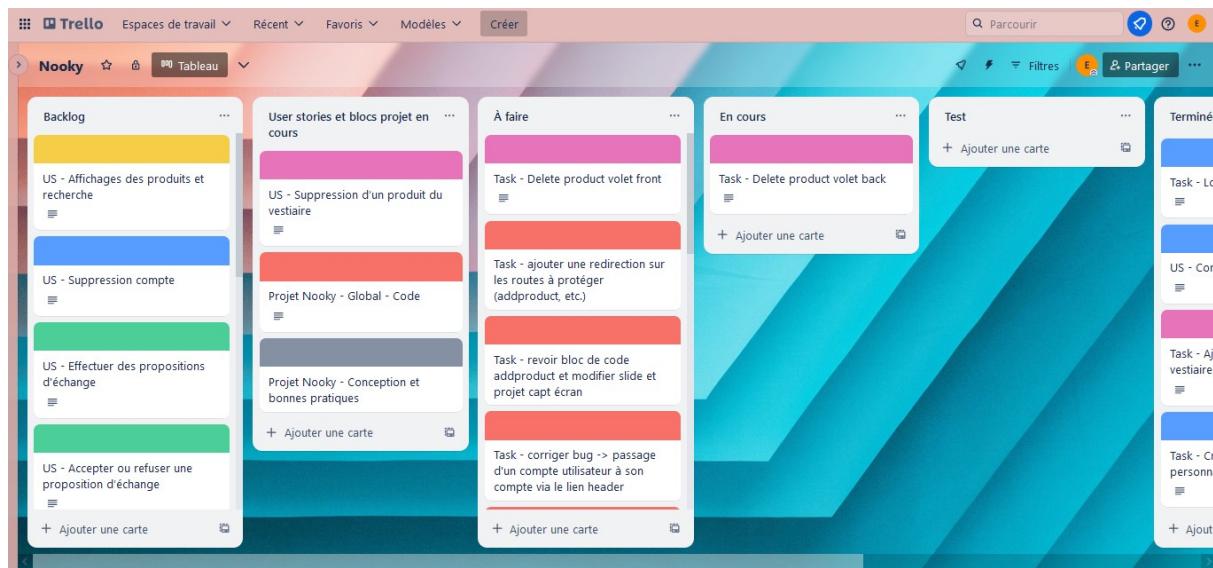
les différentes fonctionnalités. A cela, deux autres blocs ont été ajoutés : « conception et mise en application de bonnes pratiques » et « Implémentation projet global ».

Conception et mise en application de bonnes pratiques	Implémentation – projet global	Les fonctionnalités liées au produit	Les fonctionnalités propres aux échanges	Les fonctionnalités de recherche	Les fonctionnalités liées à l'utilisateur

Mon kanban est structuré en 6 colonnes :

- **Backlog** : contient les user stories à implémenter
- **User stories et blocs projets en cours** : contient les user stories et les blocs projets sur lesquels le travail est actuellement concentré
- **À faire** : contient les tâches correspondant aux user stories et aux blocs projets en cours, découpées en petits tickets
- **En cours** : indique la tâche actuellement en cours de traitement, ne contenant qu'un seul ticket à la fois
- **Tests** : indique que le ticket doit passer par une phase de tests avant d'être considéré comme terminé
- **Terminé** : contient tous les tickets et blocs projets qui sont considérés comme terminés

Ci-dessous, une capture d'écran du kanban.



Ci-dessous, une capture d'écran d'un ticket « user story » et d'un ticket « task ».

The image displays two side-by-side screenshots of a Jira ticket interface. Both tickets have a pink header bar.

User Story Screenshot:

- Title:** US - Suppression d'un produit du vestiaire
- Status:** Dans la liste [User stories et blocs projet en cours](#)
- Notifications:** Suivre
- Description:** En tant qu'utilisateur, je veux pouvoir consulter le profil d'un autre utilisateur sur l'application Nooky, afin de voir ses articles disponibles, ses préférences d'échange et son historique d'échanges.
- Activity:** Afficher les détails
- Links:** Champs personnali...
- Buttons:** Ajouter à la carte, Membres, Étiquettes, Checklist, Dates, Pièce jointe

Task Screenshot:

- Title:** Task - Delete product volet back
- Status:** Dans la liste [En cours](#)
- Notifications:** Suivre
- Description:**
 - créer un end-point personnalisable avec l'id dédié à la suppression du produit
 - ajouter la logique métier de la suppression du produit dans la couche métier
 - tester
- Activity:** Afficher les détails
- Links:** Ajouter des Pow...
- Buttons:** Ajouter à la carte, Membres, Étiquettes, Checklist, Dates, Pièce jointe, Champs personnali..., Power-Ups

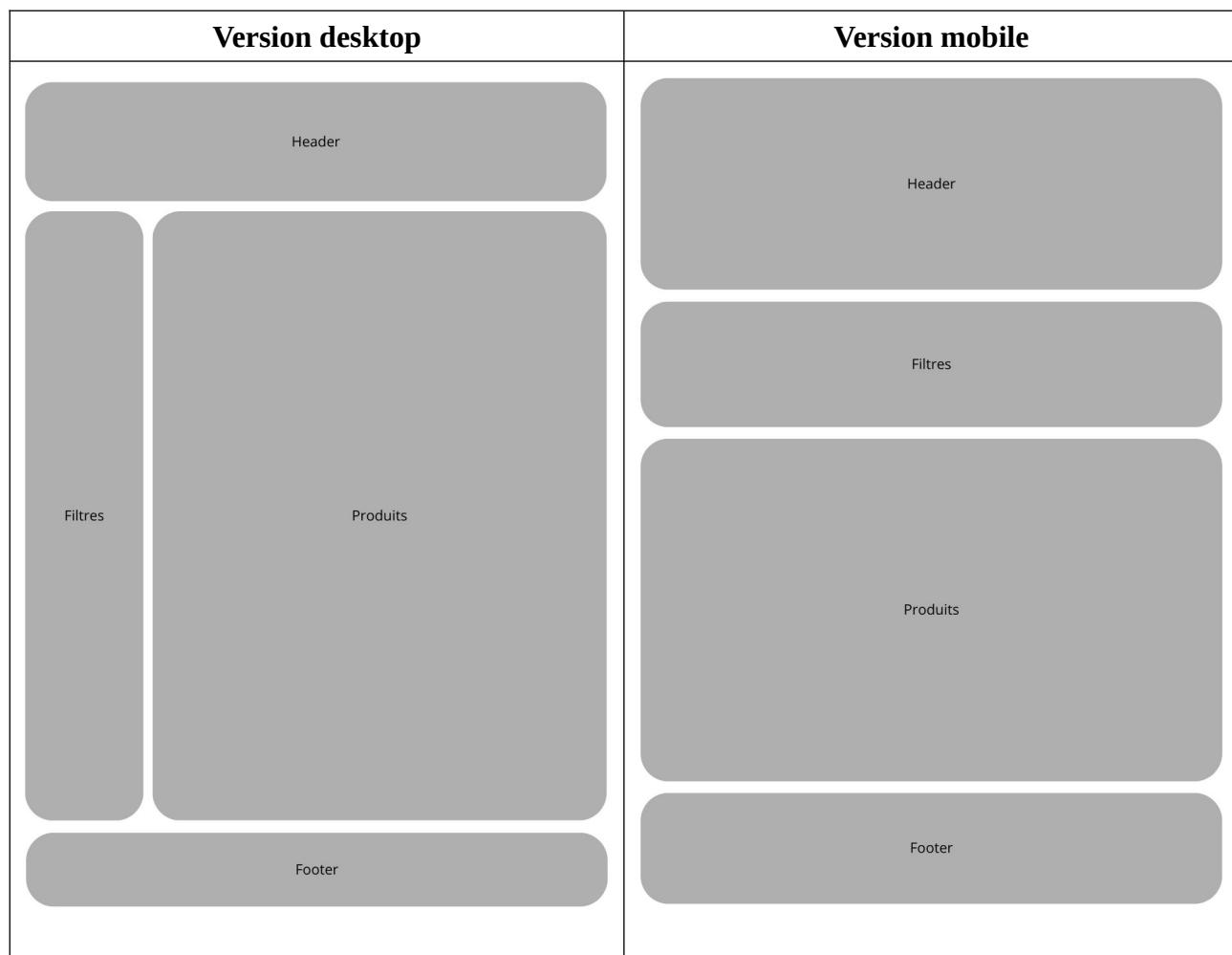
3 La conception

3.1 La conception graphique

3.1.1 Le zoning

Le zoning indique l'emplacement des éléments qui seront présents sur chaque écran du site de façon schématique.

Ci-dessous, voici l'écran Home de l'application.



L'intégralité du zoning, écran par écran, est disponible en annexes, p. 60.

3.1.2 Le wireframe ou maquette fonctionnelle

Le wireframe détaille les éléments qui seront présents sur chaque écran ainsi que leur fonction (bouton, menu, etc.).

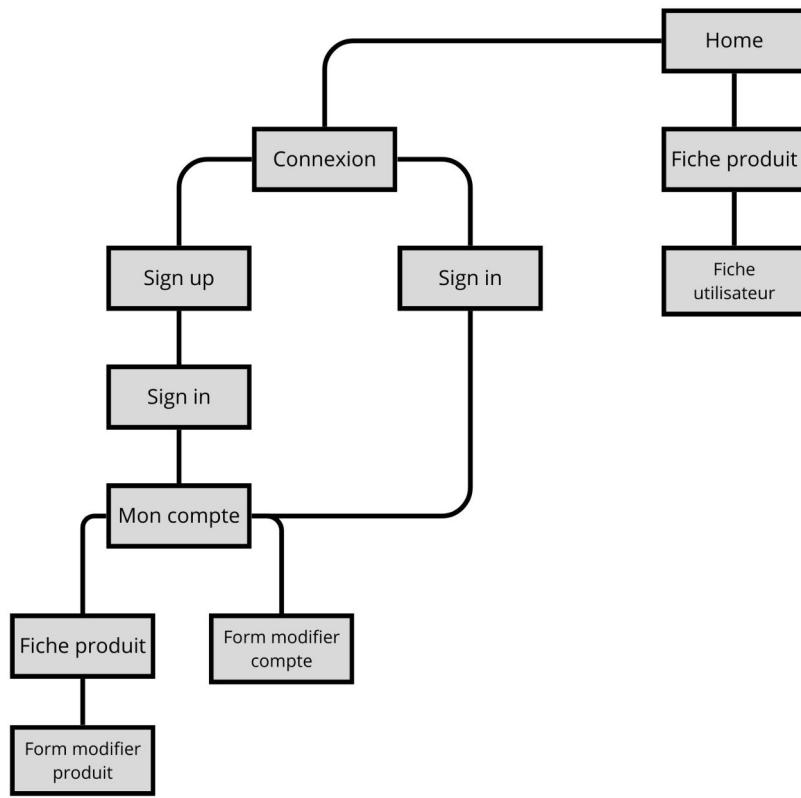
Ci-dessous, voici le wireframe de l'écran Home.

Version desktop	Version mobile
<p>Titre application Sous-titre</p> <p>Logo</p> <p>Produit 1 Produit 2 Produit 3</p> <p>Produit 4 ...</p> <p>Filtres</p> <p>Footer</p>	<p>Titre application Sous-titre</p> <p>Logo</p> <p>Menu</p> <p>barre de recherche</p> <p>Filtre 1</p> <p>Filtre 2</p> <p>Produit 1</p> <p>... autres produits</p> <p>Footer</p>

L'intégralité du wireframe, écran par écran, est disponible en annexes, p. 62.

3.1.3 L'arborescence du site

L'arborescence du site permet de modéliser le sens de navigation dans l'application.



3.1.4 La charte graphique

Ici, la direction donnée pour la charte graphique est de fournir une sélection de couleurs et une typographie qui inspirent la confiance.

Dans le contexte d'une application qui encourage les échanges de produits entre utilisateurs, il est essentiel que ces derniers se sentent en sécurité. C'est pourquoi, j'ai choisi une palette de couleurs comprenant des tons de rose pastel, deux nuances de gris pastel, ainsi qu'un gris anthracite pour créer du contraste. La police d'écriture sélectionnée est Montserrat, reconnue pour son élégance et sa lisibilité grâce à ses angles doux.



Montserrat

Pour renforcer ce sentiment de confiance, j'ai également introduit une mascotte : un Husky souriant nommé Nooky, qui donne son nom à l'application.

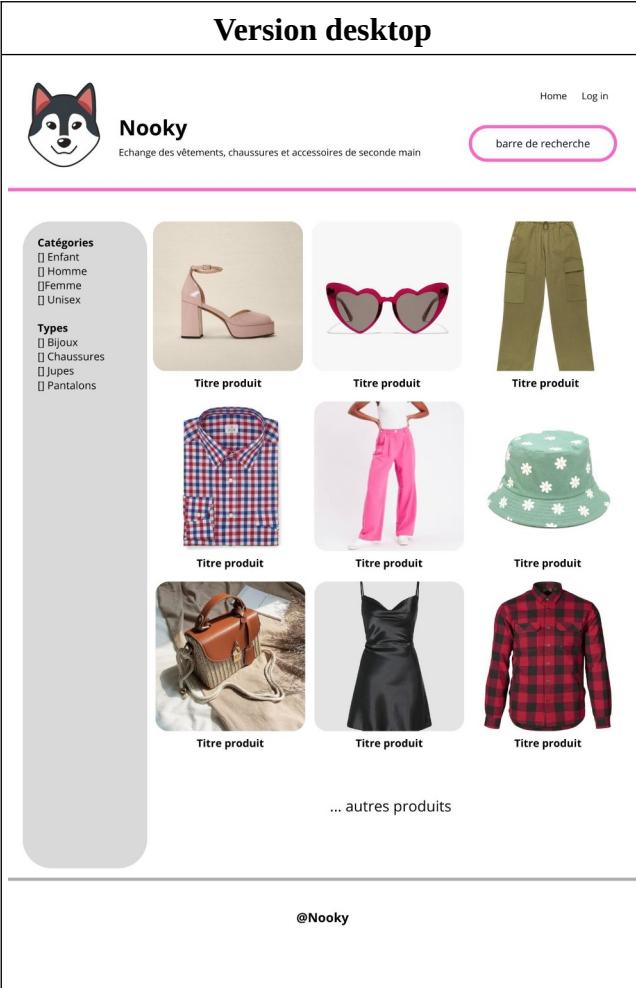
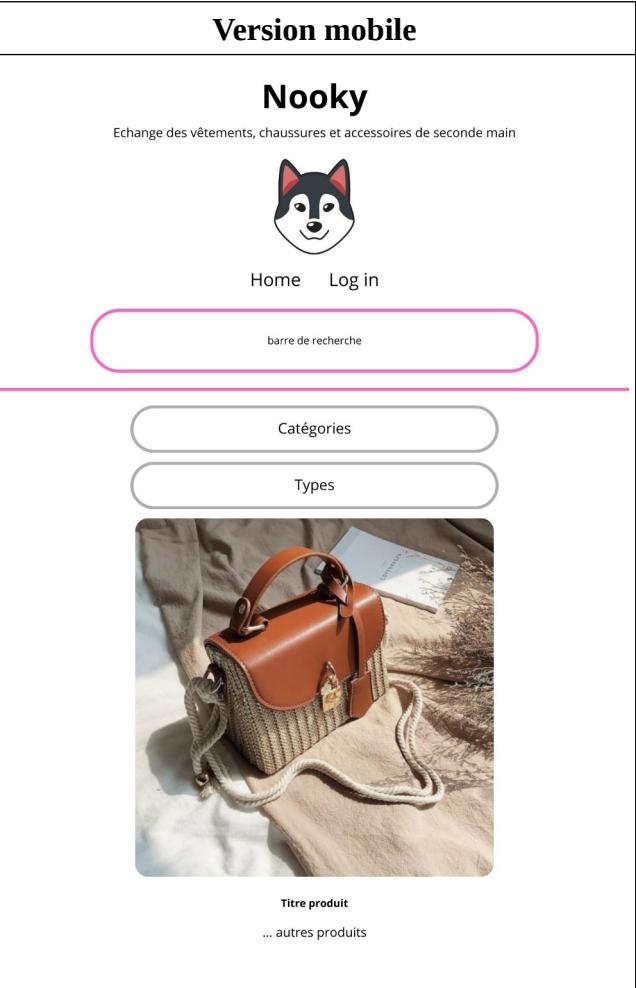
Nooky



3.1.5 La maquette statique

La maquette statique reprend les éléments du wireframe et de la charte graphique.

Ci-dessous, la maquette statique de l'écran home lorsque l'utilisateur n'est pas connecté.

Version desktop	Version mobile
 <p>Le tableau compare la maquette statique pour version desktop et version mobile de l'écran home. La version desktop (à gauche) est plus détaillée, avec une barre de recherche rose, une navigation en haut (Home, Log in), et des filtres pour Catégories (Enfant, Homme, Femme, Unisex) et Types (Bijoux, Chaussures, Jupes, Pantalons). Elle affiche une grille de produits avec des images et des titres de produits. La version mobile (à droite) est plus compacte, avec une barre de recherche rose et des boutons pour Catégories et Types. Elle montre une seule image de produit (une sac à main marron).</p>	

L'intégralité de la maquette statique, écran par écran, est disponible en annexes, p. 66.

3.2 La conception des données

Merise est une méthode de gestion de projet informatique qui a été largement utilisée dans les années 70-80.

Depuis, certains aspects de cette méthode ont été abandonnés, mais la partie dédiée à la conception de bases de données reste d'actualité dans de nombreuses entreprises.

C'est pourquoi j'ai conçu le modèle de données du projet Nooky en suivant trois étapes consécutives :

1. MCD (Modèle Conceptuel de Données)
2. MLD (Modèle Logique de Données)
3. MPD (Modèle Physique de Données)

3.2.1 Le MCD

Un MCD (Modèle Conceptuel de Données) est une abstraction des besoins métiers de l'application.

Dans ce type de schéma, on identifie les objets ou entités nécessaires et on leur attribue les propriétés qui les composent. Parmi ces propriétés, l'une d'elles doit être un identifiant ou discriminant. Ces entités sont associées entre elles par des liens sémantiques, c'est-à-dire un verbe qui qualifie la relation entre les deux entités. Enfin, on indique la cardinalité de chaque côté de l'association entre deux entités, en précisant le nombre minimum et maximum de fois qu'une occurrence de l'entité peut participer à une association.

Dans le MCD de l'application Nooky, j'ai identifié 7 objets essentiels à l'expression des besoins métiers :

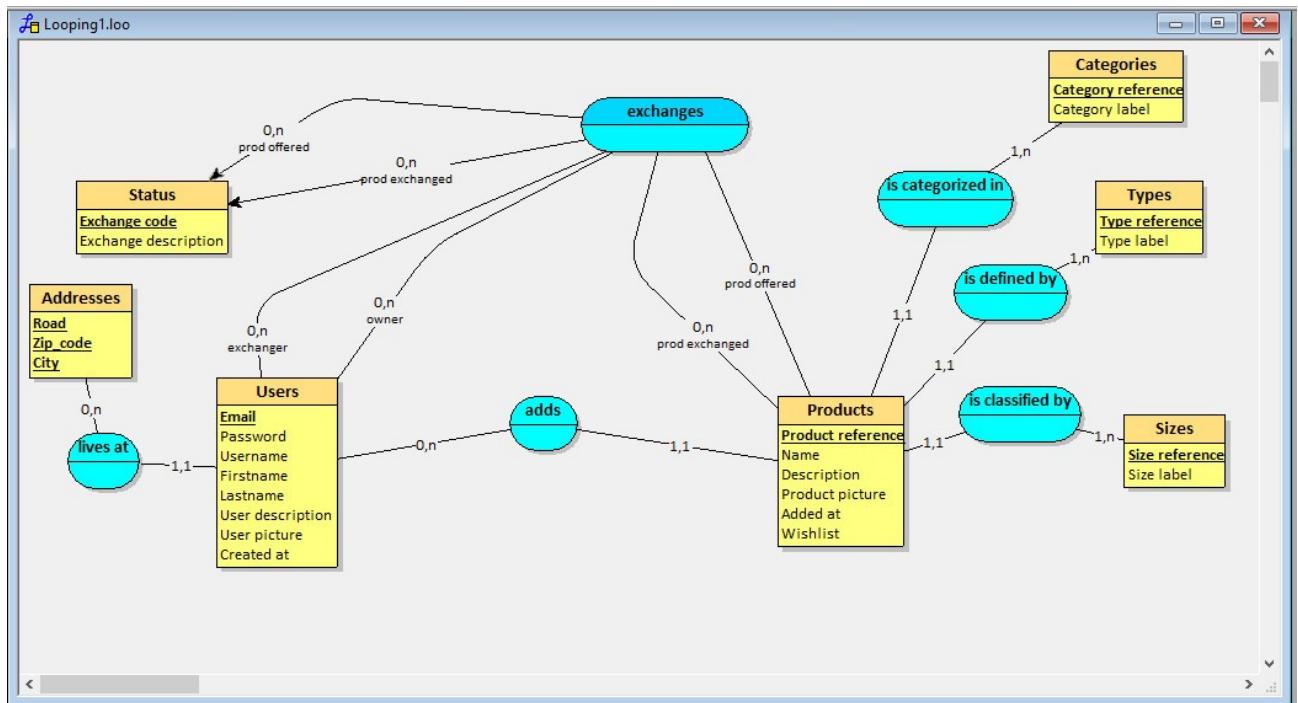
- **Users** représentent les utilisateurs de l'application
- **Addresses** représentent l'adresse physique des utilisateurs, utilisée pour la livraison des produits
- **Products** représentent les produits ajoutés par les utilisateurs
- **Categories** représentent les catégories de produits
- **Types** représentent les types de produits
- **Sizes** représentent les tailles de produits
- **Status** représentent les statuts d'un échange en cours

Une particularité de mon MCD est la relation ternaire entre les entités ‘Users’, ‘Products’ et ‘Status’, associée par le lien sémantique ‘exchanges’. Un lien qui est double à chaque fois.

Il représente l'utilisateur 1, propriétaire du produit 1, qui offre ce produit à l'échange, et l'utilisateur 2, propriétaire du produit 2, qui propose ce produit en échange du produit 1. Les liens vers l'entité ‘Status’ permettent de représenter les différentes étapes de l'échange pour chacun des produits. Par exemple, si l'utilisateur 1 accepte la proposition d'échange de l'utilisateur 2, alors les produits 1 et 2 seront associés au statut « Produit à envoyer à l'adresse de livraison correspondante ».

Une autre particularité de mon MCD est l'identifiant de l'entité ‘Addresses’. Il s'agit d'un discriminant composé des trois propriétés de l'objet : ‘Road’, ‘ZipCode’ et ‘City’.

Ci-dessous le MCD représenté graphiquement.



3.2.2 Le MLD

Avec la méthode Merise, une fois le MCD défini, on peut le transformer en MLD (Modèle Logique de Données).

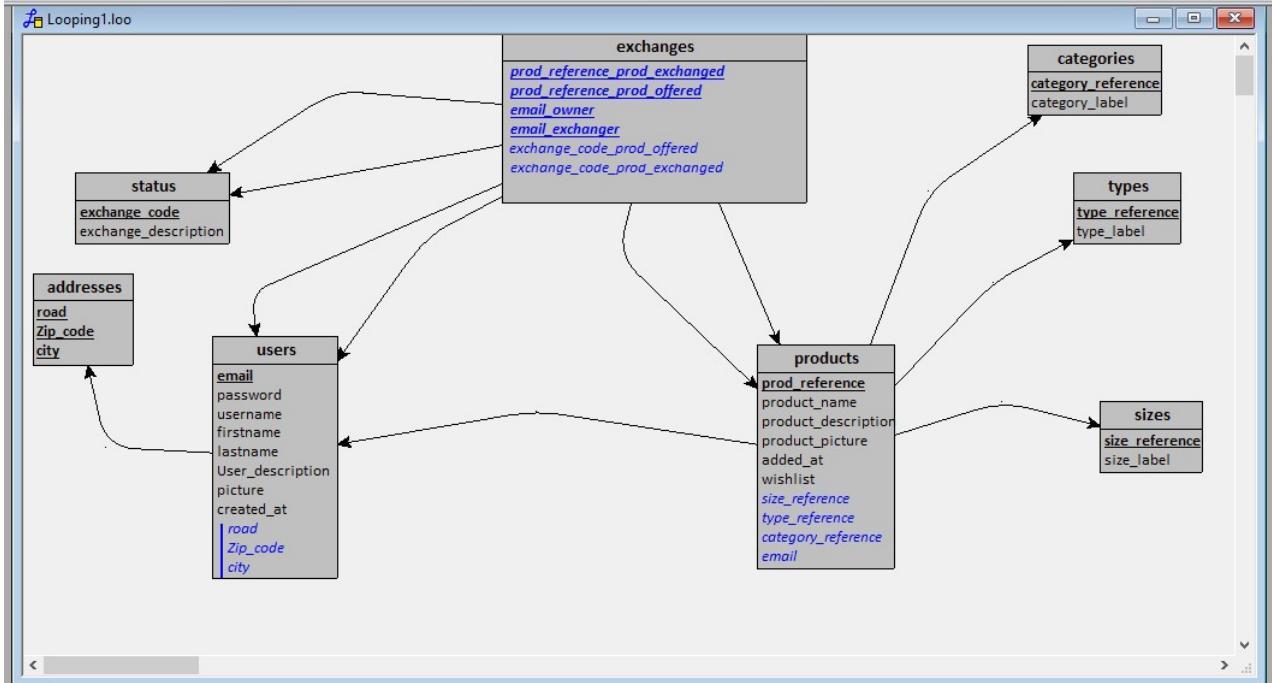
À ce stade, on connaît le type de SGBD que l'on va utiliser et l'on sait si l'on va exploiter une base de données relationnelle ou non-relationnelle.

Dans le cadre du projet Nooky, j'ai choisi un MLD-R, c'est-à-dire un Modèle Logique de Données de type Relationnel.

Ici, les entités du MCD deviennent des tables dans le MLD-R. Les propriétés deviennent des colonnes, les discriminants des clés primaires, et les associations se transforment en clés étrangères. La clé étrangère est placée du côté où la cardinalité est minimale.

Par ailleurs, puisque dans mon MCD, il y avait une cardinalité de type « many to many » (N:N), j'ai ajouté une table de jointure intitulée 'exchanges'.

Ci-dessous, le MLD représenté graphiquement.



3.2.3 Le MPD

Le MPD (Modèle Physique de Données) est une traduction du modèle logique en modèle physique. Lors de la réalisation du MLD, on prend en compte le SGBD-R cible. Ici, il s'agit de PostgreSQL. Le MLD se transforme ainsi en instructions SQL.

Comme nous connaissons le système de gestion qui sera utilisé lors de l'implémentation, nous pouvons typer les colonnes de chaque table.

En annexe, on trouve un schéma des différentes tables du MPD reliées entre elles graphiquement, p. 71.

3.3 Les traitements

Pour bien penser les traitements effectués sur les données, j'ai réalisé des diagrammes UML (Unified Modelling Language) car cela permet de bien analyser les besoins et les processus métier.

3.3.1 Cas d'utilisation

Dans un premier temps, j'ai travaillé sur un use case.

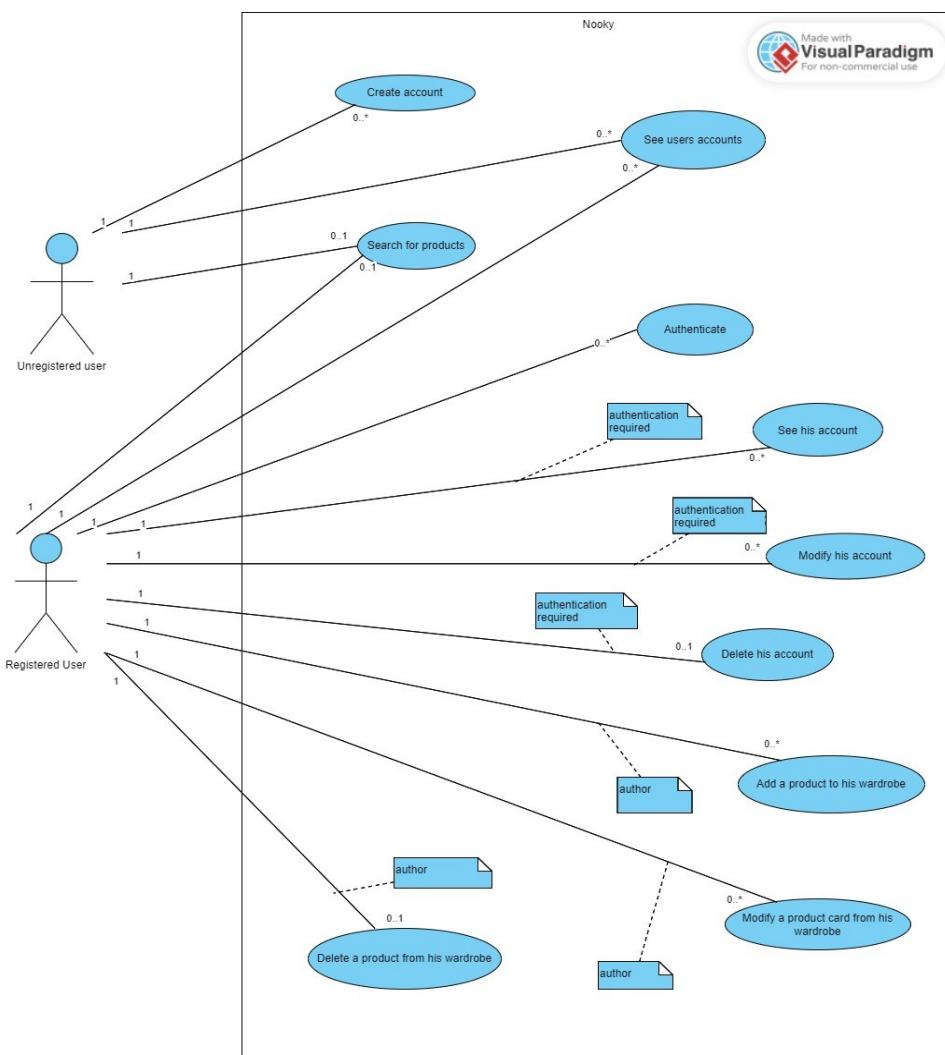
« Un cas d'utilisation, bloc fonctionnel ou cas d'usage (« use-case » en anglais), définit en génie logiciel et en ingénierie des systèmes une manière d'utiliser un système qui a une valeur ou une

utilité pour les acteurs impliqués. Le cas d'utilisation correspond à un ensemble d'actions réalisées par le système en interaction avec les acteurs en vue d'une finalité. L'ensemble des cas d'utilisation permet ainsi de décrire les exigences fonctionnelles d'un système en adoptant le point de vue et le langage de l'utilisateur final. »¹

Présenté pour la première fois par Ivar Jacobson en 1987, cette technique a été intégrée dans les outils UML dans les années 90 et est souvent utilisée dans la conception de programmes orientés objet.

Ci-dessous se trouve un cas d'utilisation réalisé pour l'application Nooky. Ce cas d'utilisation montre comment un utilisateur peut personnaliser son compte et son vestiaire.

Deux acteurs y sont représentés : un utilisateur non-enregistré et un utilisateur enregistré. Le premier peut créer un compte, rechercher des produits et consulter des comptes d'utilisateurs enregistrés. Le second peut rechercher des produits, consulter des comptes d'utilisateurs enregistrés et s'authentifier. Une fois authentifié, il a accès à des fonctionnalités supplémentaires telles que consulter, modifier ou supprimer son compte, ajouter un produit à son vestiaire, modifier les informations d'un produit ou supprimer un produit de son vestiaire.



1 Source : wikipédia, article sur les cas d'utilisation

3.3.2 Diagramme de classes

Dans un deuxième temps, j'ai travaillé sur un digramme de classes.

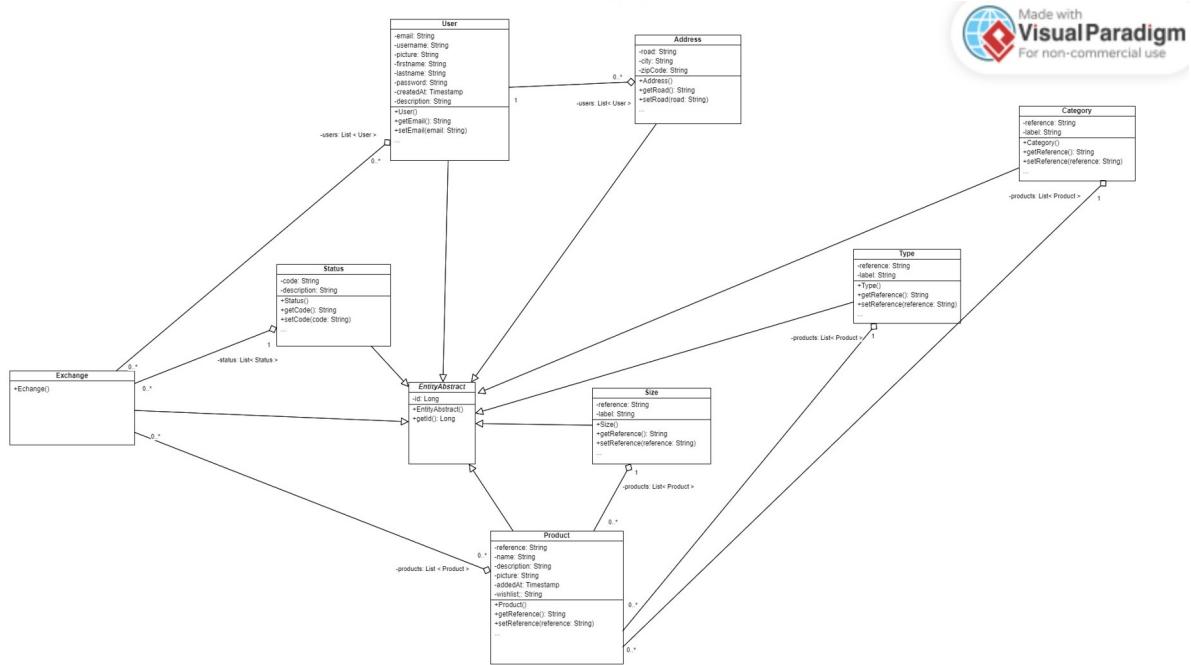
« Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que leurs relations. »²

La réalisation de ce diagramme m'a permis de définir les entités à implémenter dans la couche métier ainsi que les relations entre ces entités.

Ici, j'ai choisi de travailler avec une classe abstraite EntityAbstract qui gère la création d'identifiants. Toutes les autres entités (User, Address, Product, Size, Type, Category, Exchange, Status) héritent de cette classe abstraite.

Les associations entre les classes sont représentées par des liens. Le losange vide indique qu'ici, il s'agit d'aggrégations. De chaque côté de ce lien, on indique la multiplicité. Entre d'autres termes, le nombre de fois que l'occurrence de la classe peut participer à l'association.

Ensuite, pour chaque classe, on indique les attributs et la façon dont ils sont encapsulés. Ici, les attributs sont ‘private’ dans chaque classe. Au-dessous, on place le constructor, puis les getters et les setters. Pour des raisons de lisibilité, je n'ai pas indiqué la totalité des getters et des setters.



En annexe, on retrouve le diagramme de classes présenté en pleine page, p. 72.

2 Wikipédia, article sur les diagrammes de classe

3.3.3 Diagramme de séquences

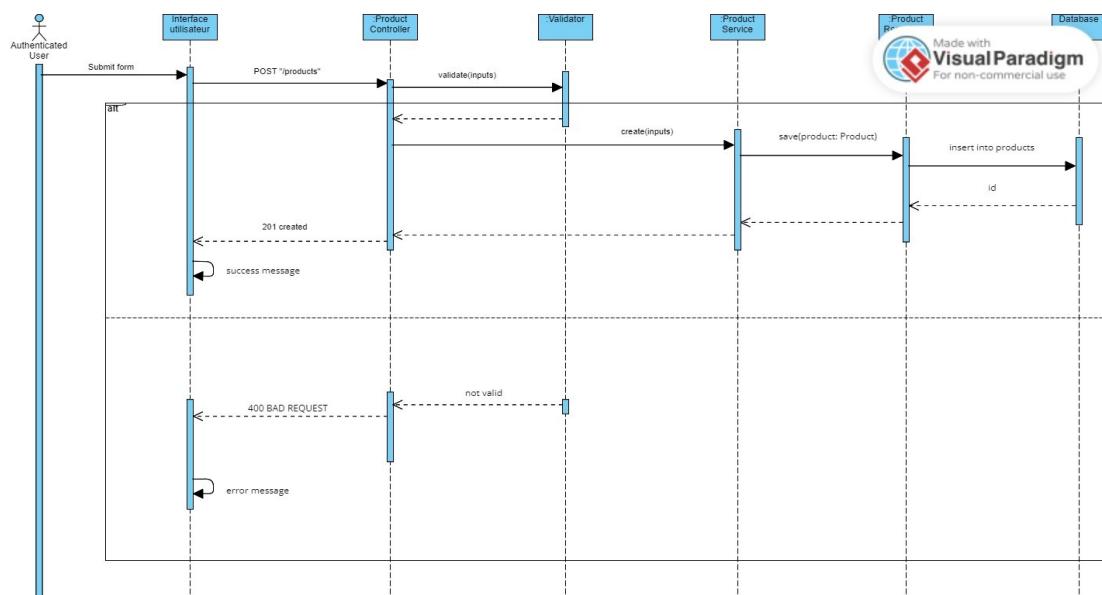
Dans un troisième temps, j'ai élaboré un diagramme de séquences.

Un diagramme de séquences montre comment les objets interagissent entre eux en suivant une séquence de messages dans le temps. Il illustre le déroulement des opérations et les échanges de messages entre les différents composants d'un système. Ce diagramme est utile pour visualiser et analyser le flux de contrôle et de données dans une application ou un processus.

Dans ce diagramme de séquences, j'ai représenté à l'aide des lignes de vie les différentes couches par lesquelles passent les données relatives à la création d'un produit ainsi que les messages transmis d'une couche à l'autre.

Ainsi, on peut voir que lorsqu'un utilisateur authentifié soumet un formulaire d'ajout de produit à son vestiaire côté IHM, les données sont envoyées au contrôleur '/products'. Ensuite, si elles sont validées par le DTO 'CreateProduct', elles passent par la couche service qui fera appel au repository correspondant afin d'être insérées en base de données. Alors, la base de données renvoie l'id au repository, ce, afin d'indiquer que l'insertion des données a bien été effectuée. L'information remonte les différentes couches jusqu'à revenir vers le front avec le code HTTP 201 Created.

Dans le cas où les données ne seraient pas valides et ne correspondraient pas à la forme attendue, le schéma présente le scénario alternatif correspondant avec le message HTTP 400 renvoyé au front.

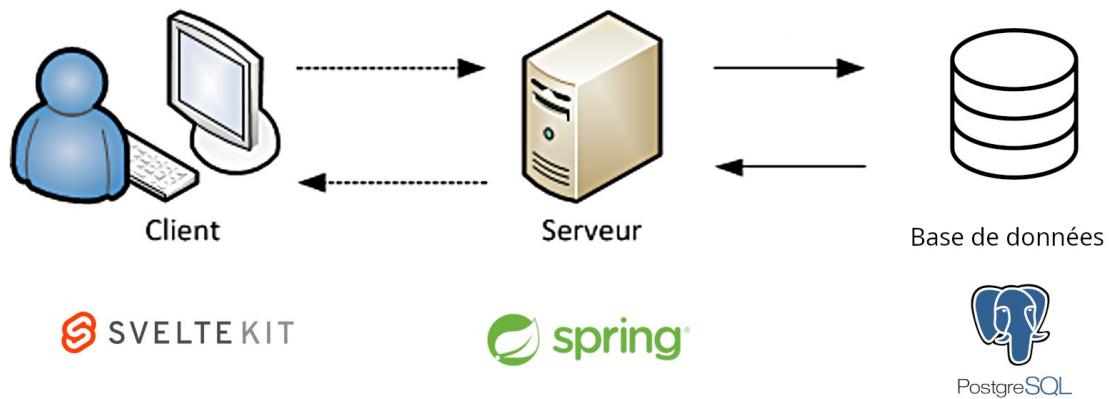


3.4 L'architecture

L'architecture est découpée en trois couches : client, métier, données.

Une architecture en couches permet de séparer les préoccupations et de faciliter la maintenance et l'évolution de l'application. Chaque couche a une responsabilité distincte : la couche client gère l'interface utilisateur, la couche métier contient la logique métier de l'application, et la couche données gère la persistance des données.

La stack choisie est la suivante : SvelteKit TS pour la couche client, Java avec le framework Spring pour la couche métier et PostgreSQL pour la couche données.



3.4.1 SvelteKit TS

SvelteKit est un framework dit SPA (Single Page Application).

Une SPA charge une seule page HTML et met à jour dynamiquement le contenu à mesure que l'utilisateur interagit avec l'application, sans que cela nécessite un rechargement complet de la page. Les avantages incluent une meilleure performance et une expérience utilisateur plus fluide, car seules les parties de la page qui changent sont mises à jour. Cela réduit les temps de chargement et offre une interface plus réactive.

Aussi, c'est un framework qui permet de développer la couche client en architecture par composants. Ce qui offre un code plus modulaire, plus évolutif.

Par ailleurs, SvelteKit est un framework performant et l'utiliser avec le langage TypeScript permet de sécuriser le code en évitant d'éventuelles erreurs de typage.

3.4.2 Java Spring

Le choix du langage Java est induit par le souhait de coder une application en programmation orientée objet. La POO permet la mise en place d'un code modulaire et la réutilisation des classes déjà existantes. Elle facilite ainsi l'évolution et les mises à jour de l'application.

Autre particularité de Java, il s'agit d'un langage au typage statique. La rigueur que cela implique permet d'éviter un certain nombre d'erreurs au moment de la compilation.

Quant au choix d'un framework comme Spring, il a été motivé par le fait qu'il implique une architecture modulaire, qu'il s'accompagne de bibliothèques prêtes à l'emploi, qu'il est populaire, a prouvé sa robustesse et est bien documenté.

3.4.3 PostgreSQL

Le modèle de base de données choisi est un modèle relationnel. Par conséquent, la couche données est en SQL.

Le choix de PostgreSQL s'est imposé car il s'agit d'un SGBD gratuit et open-source et que c'est un SGBD dont j'ai plus connaissance que MySQL. PostgreSQL est reconnu pour sa conformité aux standards SQL et sa capacité à gérer de grandes quantités de données.

4 L'implémentation

4.1 Client

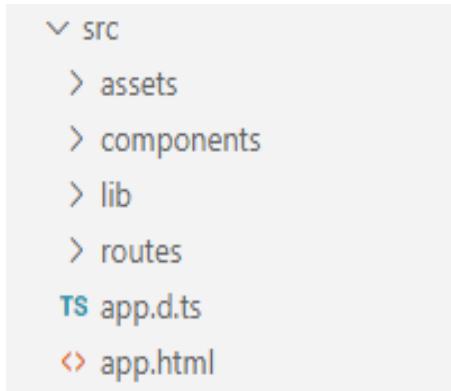
4.1.1 SPA

Comme expliqué précédemment, SvelteKit TS est un framework SPA (Single Page Application). Un framework SPA charge une seule page HTML et met à jour dynamiquement le contenu à mesure que l'utilisateur interagit avec l'application, sans nécessiter un rechargeement complet de la page.

Cela permet une expérience utilisateur plus fluide et des temps de réponse plus rapides.

4.1.2 Structure du projet

Dans le dossier src, on trouve quatre sous-dossiers.



- **Assets** contient les css principaux, les polices d'écriture et images
- **Components** contient les composants graphiques de l'application
- **Lib** contient toutes les fonctions fetch d'appel à l'API, les objets utilisés dans l'application ainsi que toutes les fonctions appelées plusieurs fois
- **Routes** contient les différentes pages de l'application

Dans le dossier 'src', on trouve également le fichier 'app.html' qui est le point d'entrée de l'application.

4.1.3 Point d'entrée de l'application

Ci-dessous, voici la page html qui est le point d'entrée de l'application.

```

src > app.html > ...
1  <!doctype html>
2  <html lang="en">
3      <head>
4          <meta charset="utf-8" />
5          <link rel="icon" href="%sveltekit.assets%/favicon.png" />
6          <meta name="viewport" content="width=device-width, initial-scale=1" />
7          %sveltekit.head%
8      </head>
9      <body data-sveltekit-preload-data="hover">
10         <div style="display: contents">%sveltekit.body%</div>
11     </body>
12 </html>

```

4.1.4 Le routing

```

routes
  account\[id]
    +page.svelte
  addproduct
    +page.svelte
  connexion
    +page.svelte
  error
    +page.svelte
  modifyaccount
    +page.svelte
  modifyproduct\[id]
    +page.svelte
  myaccount\[id]
    +page.svelte
  product\[id]
    +page.svelte
  signin
    +page.svelte
  signup
    +page.svelte
    +page.svelte

```

Avec le framework SvelteKit, le routing est construit de manière particulière.

La route principale, correspondant ici à notre page home, est située à la racine du dossier routes et est intitulée ‘+page.svelte’.

Toutes les autres routes sont déclarées dans des sous-dossiers, chacun contenant un fichier ‘+page.svelte’. Le nom du dossier est le nom du chemin.

Pour un chemin personnalisable, on crée un dossier dont le nom est celui d'une variable placée entre crochets. Par exemple : ‘account/[id]’, ‘myaccount/[id]’, ‘modifyproduct/[id]’, ‘product/[id]’.

Exemple avec application lancée en local :

localhost:5173/account/2

4.1.5 Les composants graphiques

- ✓ components
 - > Card
 - > Filters
 - > Footer
 - > Header
 - > InformationBloc
 - > ProductsBloc
 - > Searchbar
 - ...

Dans le dossier ‘components’, j’ai placé les composants graphiques. Il s’agit d’éléments comme le header utilisés dans différents écrans.

Cette façon de faire permet d’écrire moins de code et d’avoir une application plus facile à faire évoluer ou à mettre à jour. En réutilisant des composants graphiques, on assure une meilleure maintenance, mais aussi une cohérence visuelle à travers toute l’application.

Ci-dessous, exemple de découpage par composant pour l’écran ‘account/[id]’.



Nooky
Échange des vêtements, chaussures et accessoires de seconde main

Composant Header

Home Mon compte Log out

barre de recherche à coder

Mon compte



MarieFashion

Prénom : Marie
Nom de famille : Dubois
Description : Je déniche régulièrement des pépites dans les friperies de la ville de Lyon et recherche accessoires originaux
Inscrit depuis le : 14 05 2024
Habite à : Lyon

Composant InformationBloc

Mon vestiaire



Baskets Blanches



Parapluie transparent



Bague en argent



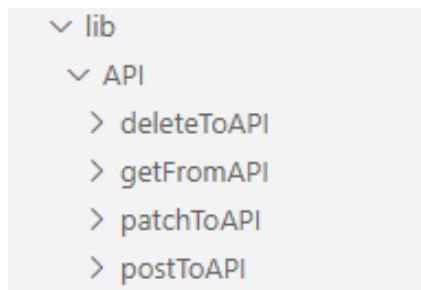
Chemise classique

Composant ProductBloc

Composant Footer

@Nooky

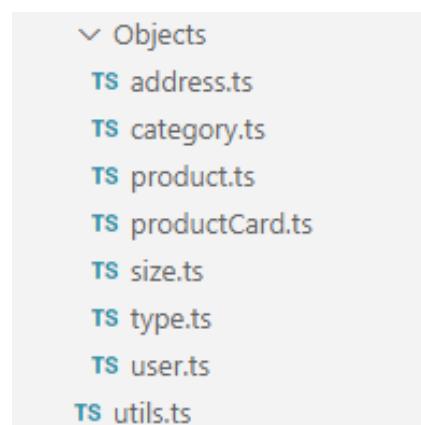
4.1.6 Envoi et récupération des données API



Dans le dossier ‘API’ situé dans ‘lib’, j’ai placé toutes mes requêtes vers l’API.

Celles-ci sont organisées en quatre sous-dossiers, classées par méthodes : get, post, patch et delete.

4.1.7 Objets et fonctions utiles



Dans le dossier ‘Objects’ situé dans ‘lib’, j’ai placé tous les objets que j’utilise dans l’application. Ce dossier contient les définitions des structures de données nécessaires pour le fonctionnement de l’application.

Dans le dossier ‘lib’, j’ai également placé un fichier utils.ts dans lequel j’ai rangé toutes les fonctions utilitaires utilisées à plusieurs endroits de mon application.

4.1.8 Exemple d’une fonctionnalité

Ci-dessous, voici quelques extraits de la fonctionnalité « ajouter un produit à son vestiaire », côté front.

Dans le bloc de code ci-dessous, je peux voir un formulaire dans lequel je récupère dynamiquement les éléments ‘Category’, ‘Type’ et ‘Size’.

La fonction ‘handleSubmit()’ est déclenchée au clic sur le bouton ‘Valider’ (submit).

```

-- 96 <form on:submit={handleSubmit}>
97   <input bind:value={inputOneUser} type="text" placeholder="Entrez le nom du produit" required>
98   <input bind:value={inputTwoUser} type="text" placeholder="Entrez la description du produit">
99   <input bind:value={inputThreeUser} type="text" placeholder="Entrez l'url de l'image">
100  <input bind:value={inputFourUser} type="text" placeholder="Produits souhaités en échange">
101  <select name="category" bind:value={selectedCategory}>
102    <#each categoryList as item {item.id}>
103      <option value={item.id}>{item.label}</option>
104    </each>
105  </select>
106  <select name="size" bind:value={selectedSize}>
107    <#each sizeList as item {item.id}>
108      <option value={item.id}>{item.label}</option>
109    </each>
110  </select>
111  <select name="type" bind:value={selectedType}>
112    <#each typeList as item {item.id}>
113      <option value={item.id}>{item.label}</option>
114    </each>
115  </select>
116  {#if hasError === true}
117    <p class="error-message">Les données envoyées doivent respecter le format ci-dessous :</p>
118    <p class="error-message">Titre de l'article : 50 caractères maximum</p>
119    <p class="error-message">Description de l'article : 400 caractères maximum</p>
120    <p class="error-message">Url de l'image : 1000 caractères maximum</p>
121    <p class="error-message">Wishlist : 200 caractères maximum</p>
122  {/if}
123  <button class="add" type="submit">Valider</button>
124  <p><a href="/myaccount/{userId}">Retour</a></p>
125 </form>

```

```

-- 51 function handleSubmit(event: Event) {
52   event.preventDefault();
53
54   if (!regexInputOneUser.test(inputOneUser)) {
55     hasError = true;
56   } else if (!regexInputTwoUser.test(inputTwoUser)) {
57     hasError = true;
58   } else if (!regexInputThreeUser.test(inputThreeUser)) {
59     hasError = true;
60   } else if (!regexInputFourUser.test(inputFourUser)) {
61     hasError = true;
62   } else {
63     productData.name = inputOneUser;
64     productData.description = inputTwoUser;
65     productData.picture = inputThreeUser;
66     productData.wishlist = inputFourUser;
67     productData.userId = userId;
68     productData.typeId = selectedType;
69     productData.sizeId = selectedSize;
70     productData.categoryId = selectedCategory;
71     postData(productData);
72
73     goto("/myaccount/" + userId);
74   }

```

Dans le bloc de code ci-dessous, je fais un appel fetch méthode Post à mon API via le endpoint products.

On peut voir que cette fonction fait appel à une autre fonction : ‘postData()’.

```

1 import { postData } from '../../../../../utils';
2 import type { Product } from '../../../../../Objects/product';
3
4 export async function postProduct(product: Product) {
5   const url = 'http://localhost:8080/products';
6
7   postData(url, product);
8 }

```

Dans cet extrait de code, je vois la fonction ‘handleSubmit()’.

Dans cette fonction, je récupère les inputs de l’utilisateur. Chaque input est associé au nom de la variable correspondant au DTO ‘CreateProduct.java’ côté back.

Le tout est placé dans une variable ‘productData’ au format json {}.

Ensuite, j’appelle la fonction fetch ‘postProduct()’, puis je bascule sur le compte de l’utilisateur.

```

22  const response = await fetch(url, {
23    method: 'POST',
24    headers: {
25      Accept: 'application/json',
26      'authorization': `Bearer ${token}`,
27      'Content-Type': 'application/json',
28    },
29    body: JSON.stringify(data)

```

C'est la fonction 'postData()' qui gère l'envoi du token vers la couche métier. En effet, il est nécessaire d'être authentifié pour pouvoir ajouter un produit.

4.2 Métier

La couche métier a été codée en Java à l'aide du framework Spring.

Le projet a été configuré comme une application Spring Boot, utilisant la version 3.1.2 du starter parent de Spring Boot. Il a été construit avec Java 17 et comporte plusieurs dépendances clés.

- Spring Data JPA : Pour la gestion des accès aux données et la manipulation des bases de données relationnelles.
- Spring OAuth2 Resource Server : Pour la sécurisation des ressources avec OAuth2.
- Java JWT : Pour la gestion des tokens JWT.
- Spring Validation : Pour la validation des données.
- Spring Web : Pour le développement de services web RESTful.
- PostgreSQL : Le driver pour l'utilisation de PostgreSQL comme base de données.
- Spring Boot Test : Pour les tests unitaires et d'intégration.

La section build contient le plugin spring-boot-maven-plugin pour faciliter la création de l'application exécutable Spring Boot.

4.2.1 Structure du projet

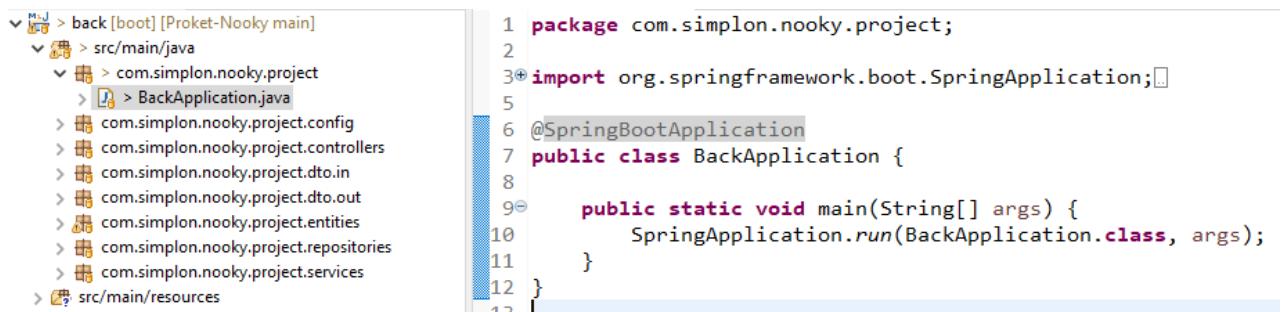
La couche back-end contient un fichier 'BackApplication.java' à la racine qui est le point d'entrée de l'application.

Puis, le projet est structuré en packages ce qui est courant dans les projets Spring Boot.

- Le package **config** contient les configurations spécifiques de l'application, telles que la configuration de sécurité, la configuration JWT, les CORS, etc.
- Le package **controllers** contient les classes contrôleurs qui définissent les ends-points de

mon API REST et gèrent les requêtes HTTP

- Le package **dto.in** contient les objets de transfert de données (DTO) entrants
- Le package **dto.out** contient les objets de transfert de données (DTO) sortants
- Le package **entities** contient les classes d'entités qui représentent les données persistantes de l'application et sont mappées aux tables correspondantes de la base de données
- Le package **repositories** contient les interfaces de repository qui définissent les méthodes pour accéder aux données dans la base de données
- Le package **services** contient les classes de service qui implémentent la logique métier de l'application et coordonnent les opérations entre les contrôleurs et les repositories

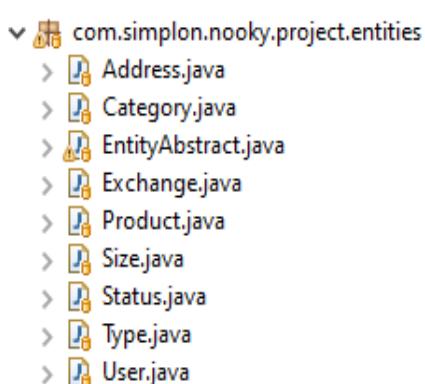


The screenshot shows the project structure and a code editor. The project structure on the left includes packages for BackApplication, config, controllers, dto.in, dto.out, entities, repositories, and services. The code editor on the right contains the main application class:

```
1 package com.simplon.nooky.project;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class BackApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(BackApplication.class, args);
10    }
11 }
12 }
```

4.2.2 Les classes entités

Dans ce package, on retrouve huit entités :



- **Address** représente une adresse associée à un utilisateur
- **Category** représente une catégorie à laquelle un produit peut appartenir
- **Exchange** représente une transaction d'échange entre utilisateurs
- **Product** représente un produit
- **Size** représente une taille associée à un produit
- **Status** représente le statut d'un échange en cours
- **Type** représente un type de produit
- **User** représente un utilisateur de l'application

Ces entités héritent toutes de la classe abstraite ‘EntityAbstract.java’. Cette classe abstraite gère la création d’id de toutes les entités de l’application.

Ci-dessous, la classe abstraite ‘EntityAbstract.java’.

```

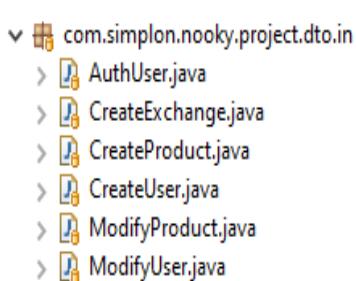
1 package com.simplon.nooky.project.entities;
2
3 import jakarta.persistence.Column;
4
5 @MappedSuperclass
6 public class EntityAbstract {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    @Column(name = "id")
11    private Long id;
12
13    public EntityAbstract() {
14        }
15
16    public Long getId() {
17        return this.id;
18    }
19
20    private void setId(Long id) {
21        // id generated by database
22        this.id = id;
23    }
24}
25
26
27
28 }
```

4.2.3 DTO

L'application a été pensée avec une couche DTO (Data Transfert Object) en entrée (in) et en sortie (out).

Cette couche permet de ne pas exposer les entités.

4.2.3.1 Les DTO d'entrées

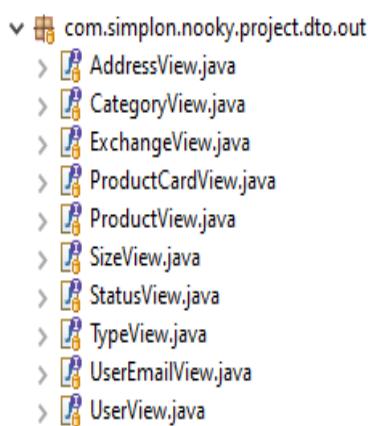


Les données en provenance du front sont tout d'abord validées par une couche DTO.

Chaque DTO va utiliser des contraintes à l'aide d'annotations qui permettent de vérifier que les données envoyées correspondent à la forme attendue en base de données.

Par exemple, un DTO pour la création d'un utilisateur inclut une annotation pour vérifier que l'adresse e-mail est au format correct.

4.2.3.2 Les DTO de sorties



Seules les données nécessaires sont envoyées au front. Ce, pour répondre à des objectifs de sécurité, mais aussi de performance.

Les DTO de sortie sont utilisés pour structurer et filtrer les données avant de les renvoyer au client, garantissant ainsi que seules les informations pertinentes et autorisées sont exposées.

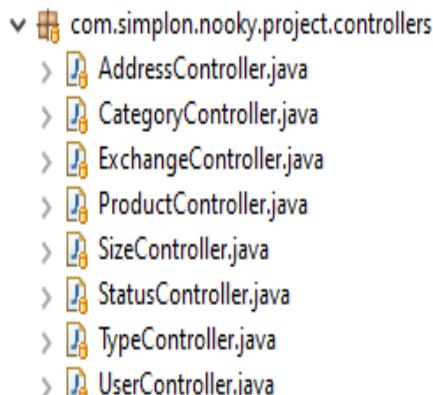
Par exemple, le DTO de sortie pour un profil utilisateur va inclure le nom et l'adresse e-mail de l'utilisateur, mais omettre des informations sensibles telles que le mot de passe.

L'utilisation des DTO permet également de transformer les données si nécessaire, en formatant ou en combinant des champs tout en maintenant une séparation claire entre les modèles de données internes et les représentations publiques.

4.2.4 Controllers

Comme je l'ai expliqué plus haut, dans mon architecture back-end, j'ai adopté une approche basée sur la séparation des responsabilités.

Ainsi, les contrôleurs gèrent les requêtes HTTP et les réponses associées, tandis que les services sont chargés de la logique métier de l'application.



Les contrôleurs agissent comme une interface entre les requêtes des clients et la logique métier sous-jacente, garantissant ainsi une séparation claire des préoccupations et une architecture back-end bien structurée.

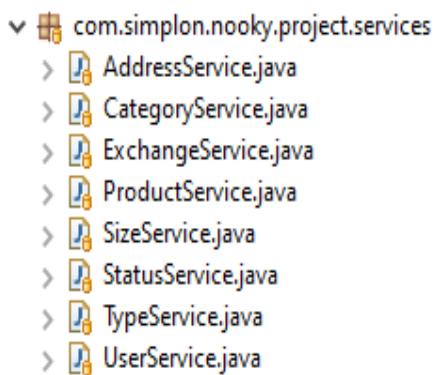
Ci-dessous, un bloc de code extrait de ProductController.java.

```

1 package com.simplon.nooky.project.controllers;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/products")
7 public class ProductController {
8     private final ProductService service;
9
10    public ProductController(ProductService service) {
11        this.service = service;
12    }
13
14    @PostMapping
15    @ResponseStatus(HttpStatus.CREATED)
16    public void createProduct(@RequestBody CreateProduct product) {
17        service.createProduct(product);
18    }

```

4.2.5 Services



La couche service de mon architecture back-end est le pilier de la logique métier de l'application. Elle gère les opérations de traitement des données, telles que la création, la lecture, la mise à jour et la suppression (CRUD).

C'est également à cet endroit que se fait la liaison entre les données DTO (Data Transfer Object) et les entités de l'application, qui sont mappées dans la couche entités et reliées à la base de données via la couche repository.

Cette liaison implique la transformation des données reçues sous forme de DTO en objets entités, puis leur manipulation conformément aux règles métier de l'application.

En effectuant cette transition entre les différentes couches de l'architecture, je veille à ce que les opérations effectuées respectent l'intégrité des données et correspondent aux exigences fonctionnelles définies, contribuant ainsi à la robustesse et à la cohérence du système dans son ensemble.

Ci-dessous un extrait de code présentant la logique métier de la fonction « ajouter un produit ».

```

40  public void createProduct(CreateProduct productCreation) {
41      Product product = new Product();
42
43      Timestamp timestamp = new Timestamp(System.currentTimeMillis());
44
45      synchronized (ProductService.class) {
46          productCounter++;
47      }
48
49      String productRef = "PROD_" + productCounter;
50
51      product.setName(productCreation.getName());
52      product.setReference(productRef);
53      product.setDescription(productCreation.getDescription());
54      product.setPicture(productCreation.getPicture());
55      product.setWishlist(productCreation.getWishlist());
56      product.setAddedAt(timestamp);
57
58      product.setCategory(categoryRepository.getReferenceById(productCreation.getCategoryId()));
59      product.setSize(sizeRepository.getReferenceById(productCreation.getSizeId()));
60      product.setType(typeRepository.getReferenceById(productCreation.getTypeId()));
61      product.setUser(userRepository.getReferenceById(productCreation.getUserId()));
62
63      productRepository.save(product);
64  }

```

4.2.6 Les ORM

Les entités représentent les différents types d'objets métier de l'application, tels que les utilisateurs, les produits ou les échanges.

Ces entités sont mappées aux tables de la bases de données à l'aide d'un ORM explicité dans le code grâce aux annotations.

L'utilisation d'un ORM (Object-Relational Mapping) simplifie la manipulation des données en permettant de travailler avec des objets familiers, plutôt qu'avec des requêtes SQL directes.

L'ORM se charge de traduire les opérations effectuées sur les objets en requêtes SQL, ce qui facilite le développement, la maintenance et l'évolutivité de l'application.

De plus, en définissant des relations entre les entités l'ORM facilite la gestion de la cohérence des données et garantit l'intégrité référentielle au niveau de la base de données.

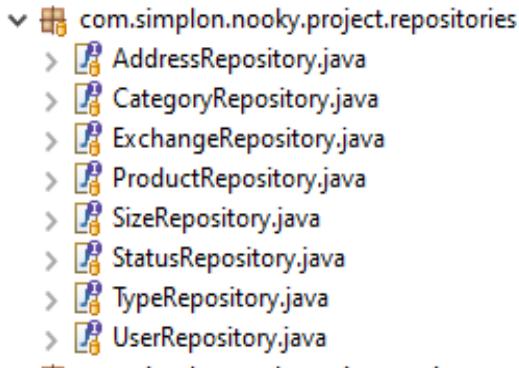
Ci-dessous, l'entité ‘Product’.

```

11 @Entity
12 @Table(name="products")
13 public class Product extends EntityAbstract {
14
15     @Column(name = "reference")
16     private String reference;
17
18     @Column(name = "name")
19     private String name;
20
21     @Column(name = "description")
22     private String description;
23
24     @Column(name = "picture")
25     private String picture;
26
27     @Column(name = "added_at")
28     private Timestamp addedAt;
29
30     @Column(name = "wishlist")
31     private String wishlist;
32
33     @ManyToOne
34     @JoinColumn(name = "category_id")
35     private Category category;
36
37     @ManyToOne
38     @JoinColumn(name = "size_id")
39     private Size size;
40
41     @ManyToOne
42     @JoinColumn(name = "type_id")
43     private Type type;
44
45     @ManyToOne
46     @JoinColumn(name = "user_id")
47     private User user;
48
49     public Product() {
50
51
52     public String getReference() {
53         return this.reference;
54
55     public void setReference(String reference) {
56

```

4.2.7 Repositories



Les repositories agissent comme une interface entre la couche service de l'application et la base de données sous-jacente. Ils encapsulent la logique d'accès aux données, permettant ainsi d'effectuer des opérations de lecture, d'écriture, de mise à jour et de suppression de manière efficace et cohérente. En utilisant les repositories, je peux abstraire les détails spécifiques à la base de données et fournir à la couche service une API cohérente et orientée objet pour interagir avec les données.

4.2.8 Sécurité

La sécurité est un aspect crucial.

Une application sécurisée protège non seulement les données sensibles des utilisateurs mais assure également la confiance et la pérennité de l'application. Les menaces potentielles doivent être anticipées par des pratiques de programmation sécurisées et des configurations appropriées.

4.2.8.1 Configuration des CORS

Les Cross-Origin Resource Sharing (CORS) sont configurés pour autoriser toutes les méthodes

("GET", "POST", "PATCH", "PUT", "DELETE", "OPTIONS") en provenance de toutes les origines. Cette configuration est essentielle pour permettre des interactions entre le front-end et le back-end de l'application.

Voir bloc de code ci-dessous.

```
1 package com.simplon.nooky.project.config;
2
3 import org.springframework.context.annotation.Bean;□
4
5 @Configuration
6 public class CorsConfig {
7
8     @Bean
9     WebMvcConfigurer corsConfigurer() {
10         return new WebMvcConfigurer() {
11             @Override
12             public void addCorsMappings(CorsRegistry registry) {
13                 registry.addMapping("/**")
14                     .allowedOrigins("*")
15                     .allowedMethods("GET", "POST", "PATCH", "PUT", "DELETE", "OPTIONS")
16                     .allowedHeaders("*");
17             }
18         };
19     }
20 }
```

4.2.8.2 Sécurisation des routes

Grâce à la dépendance Spring Security, j'ai pu personnaliser l'accès à chaque route de l'application. Cela permet de s'assurer que seules les personnes autorisées puissent accéder aux fonctionnalités sensibles.

Quelques-uns des endpoints, comme ceux dédiés à la recherche et à la consultation des produits, sont accessibles par tous avec la méthode .permitAll().

```
.authorizeHttpRequests(req -> req
    .requestMatchers(HttpMethod.GET, "/products/all")
    .permitAll())
.authorizeHttpRequests(req -> req
    .requestMatchers(HttpMethod.GET, "/products/{id}")
    .permitAll())
.authorizeHttpRequests(req -> req
    .requestMatchers(HttpMethod.GET, "/products/all/user/{id}")
    .permitAll())
```

Les endpoints liés à la création de compte ou à l'authentification sont accessibles aux utilisateurs non authentifiés avec la méthode .anonymous().

```
.authorizeHttpRequests(req -> req
    .requestMatchers(HttpMethod.POST, "/users/auth")
    .anonymous())
.authorizeHttpRequests(req -> req
    .requestMatchers(HttpMethod.GET, "/users/email/{email}")
    .anonymous())
```

Les endpoints nécessitant une authentification, comme ceux pour ajouter ou modifier des produits, sont protégés avec la méthode `.fullyAuthenticated()`.

```
.authorizeHttpRequests(  
    reqs -> reqs.anyRequest().fullyAuthenticated())
```

4.2.8.3 JWT

Pour l'authentification, j'ai utilisé la dépendance `com.auth0 java-JWT` (Json Web Token). Le JWT est un standard qui permet d'échanger des informations de manière sécurisée entre un client et un serveur sous forme de token.

Lorsqu'un utilisateur se logue et envoie ses données 'email' et 'password' via une requête POST à l'API REST, le back-end génère un token JWT.

Ce token contient trois parties : l'en-tête, la charge utile et la signature.

L'en-tête spécifie le type de token et l'algorithme de chiffrement utilisé. La charge utile (ou payload) contient les informations utilisateur (claims). Dans le cadre de mon projet, il contient son identifiant. La signature assure que le token n'a pas été modifié.

Ce token est ensuite envoyé au client qui le stocke dans le local storage. Le client doit inclure ce token dans les en-têtes des requêtes subséquentes pour accéder aux ressources sécurisées. Le back-end vérifie la validité du token pour authentifier les utilisateurs et autoriser l'accès.

4.2.8.4 Bcrypt

Pour garantir la sécurité des mots de passe des utilisateurs, j'ai utilisé l'algorithme de hachage Bcrypt.

Lorsque l'utilisateur crée un compte ou change son mot de passe, le mot de passe est haché avec Bcrypt avant d'être stocké dans la base de données. Lors de la connexion, le mot de passe fourni par l'utilisateur est haché de la même manière et comparé au haché stocké.

4.3 Données

Dans le cadre de la mise en place de la base de données de l'application, j'ai développé trois scripts distincts. Le premier script est dédié à la création des tables nécessaires à l'application. Le deuxième script se charge de l'insertion de données de référence stables dans les tables 'categories', 'types', 'sizes' pour les produits, ainsi que dans la table 'status' pour les statuts des échanges en cours. Enfin, le troisième script insère des données de test pour valider le bon fonctionnement de l'application.

 schema1-ddl.sql	29/04/2024 16:38	Fichier source SQL	3 Ko
 schema2-data-referential.sql	02/05/2024 11:12	Fichier source SQL	3 Ko
 schema3-data-test.sql	30/04/2024 15:20	Fichier source SQL	19 Ko

4.3.1 Le script de création des tables

Le premier script est conçu pour créer les tables de la base de données. Il définit la structure de chaque table, y compris les colonnes, les types de données et les contraintes. Ce script assure que la base de données dispose de toutes les tables nécessaires pour stocker les informations sur les utilisateurs, les adresses de livraison, les produits, les catégories, les tailles, les types, les échanges et les statuts des échanges.

Lors de l'implémentation, le MPD est dénormalisé si nécessaire. Ici, j'ai ajouté un id numérique pour des soucis de performance. A ce moment-là, les clés primaires du MPD sont devenues des clés secondaires dans chaque table.

Ci-dessous un extrait du script de création des tables. On peut voir que la table 'products' intègre quatre clés étrangères, l'id des tables 'categories', 'sizes', 'types' et 'users'.

```
1 CREATE TABLE "categories" (
2     id SERIAL PRIMARY KEY,
3     reference VARCHAR(10) UNIQUE NOT NULL,
4     label VARCHAR(50) NOT NULL
5 );
6
7 CREATE TABLE "sizes" (
8     id SERIAL PRIMARY KEY,
9     reference VARCHAR(10) UNIQUE NOT NULL,
10    label VARCHAR(50) NOT NULL
11 );
12
13 CREATE TABLE "types" (
14     id SERIAL PRIMARY KEY,
15     reference VARCHAR(10) UNIQUE NOT NULL,
16     label VARCHAR(50) NOT NULL
17 );
18
33    CREATE TABLE "users" (
34        id SERIAL PRIMARY KEY,
35        email VARCHAR(254) UNIQUE NOT NULL,
36        password VARCHAR(100) NOT NULL,
37        username VARCHAR(50) NOT NULL,
38        description VARCHAR(400),
39        picture VARCHAR(1000),
40        firstname VARCHAR(50) NOT NULL,
41        lastname VARCHAR(50) NOT NULL,
42        created_at TIMESTAMP,
43        address_id INT NOT NULL,
44        FOREIGN KEY (address_id) REFERENCES addresses(id)
45    );
46
```

```

47  CREATE TABLE "products" (
48      id SERIAL PRIMARY KEY,
49      reference VARCHAR(1000) UNIQUE NOT NULL,
50      name VARCHAR(50) NOT NULL,
51      description VARCHAR(400),
52      picture VARCHAR(1000),
53      added_at TIMESTAMP,
54      wishlist VARCHAR(200),
55      category_id INT NOT NULL,
56      size_id INT NOT NULL,
57      type_id INT NOT NULL,
58      user_id INT NOT NULL,
59      FOREIGN KEY (category_id) REFERENCES categories(id),
60      FOREIGN KEY (size_id) REFERENCES sizes(id),
61      FOREIGN KEY (type_id) REFERENCES types(id),
62      FOREIGN KEY (user_id) REFERENCES users(id)
63  );

```

En annexe, le script SQL de création des tables est disponible intégralement, p. 73.

4.3.2 Le script d'insertion des données de références

Le deuxième script s'occupe de l'insertion de données de référence dans certaines tables. Il insère les catégories, les types et les tailles des produits dans les tables respectives ‘categories’, ‘types’, et ‘sizes’. Ces données sont stables et ne devraient pas changer fréquemment. De plus, il remplit la table ‘status’ avec les différents statuts possibles pour les échanges en cours, fournissant ainsi des valeurs prédéfinies pour gérer les transactions sur la plateforme.

Ci-dessous, un extrait du script d'insertion des données de référence pour les tables ‘categories’ et ‘sizes’.

```

1  INSERT INTO categories (reference, label) VALUES
2  ('CAT001', 'Enfant'),
3  ('CAT002', 'Femme'),
4  ('CAT003', 'Homme'),
5  ('CAT004', 'Unisex');
6
7  INSERT INTO sizes (reference, label) VALUES
8  ('SIZ001', 'XXS'),
9  ('SIZ002', 'XS'),
10 ('SIZ003', 'S'),
11 ('SIZ004', 'M'),
12 ('SIZ005', 'L'),
13 ('SIZ006', 'XL'),
14 ('SIZ007', 'XXL').

```

4.3.3 Le script d'insertion des données de tests

Le troisième script est utilisé pour insérer des données de test dans la base de données. Ces données de test sont essentielles pour valider le bon fonctionnement de l'application et vérifier que les différentes fonctionnalités se comportent comme prévu. Les données de test incluent des utilisateurs fictifs, des produits d'exemple, et des scénarios d'échange, permettant ainsi de simuler des interactions réelles et de s'assurer que le système répond correctement aux attentes.

Ci-dessous, un extrait du script d'insertion de données de tests dans la table 'products'.

```
35  INSERT INTO products (reference, name, description, picture, added_at, wishlist, category_id, size_id, type_id, user_id) VALUES
36  ('PROD001', 'T-Shirt Noir', 'T-Shirt noir basique', 'https://vision-naire.com/cdn/shop/products/2-Tshirt-visionnaire-TCPR-noir\_25
37  (SELECT id FROM categories WHERE reference = 'CAT003'),
38  (SELECT id FROM sizes WHERE reference = 'SIZ003'),
39  (SELECT id FROM types WHERE reference = 'TYP035'),
40  (SELECT id FROM users WHERE email = 'user1@example.com')),
41  ('PROD002', 'Chemise à carreaux', 'Chemise à carreaux rouge et bleue, tissus de qualité', 'https://www.cottonsociety.com/173158-t
42  (SELECT id FROM categories WHERE reference = 'CAT003'),
43  (SELECT id FROM sizes WHERE reference = 'SIZ002'),
44  (SELECT id FROM types WHERE reference = 'TYP014'),
45  (SELECT id FROM users WHERE email = 'user3@example.com')),
46  ('PROD003', 'Baskets Blanches', 'Baskets blanches à lacets', 'https://www.pasdegeant.fr/media/catalog/product/cache/e3613eb285f3e
47  (SELECT id FROM categories WHERE reference = 'CAT002'),
48  (SELECT id FROM sizes WHERE reference = 'SIZ025'),
49  (SELECT id FROM types WHERE reference = 'TYP034'),
50  (SELECT id FROM users WHERE email = 'user2@example.com')),
51  ('PROD004', 'Montre en cuir', 'Montre-bracelet en cuir marron', 'https://charlie-paris.com/cdn/shop/products/bracelet-cuir-mick-t
52  (SELECT id FROM categories WHERE reference = 'CAT004'),
53  (SELECT id FROM sizes WHERE reference = 'SIZ040'),
54  (SELECT id FROM types WHERE reference = 'TYP003'),
```

5 Les tests

5.1 Tests d'intégration back-end

Les tests d'intégration back-end visent à vérifier que les différents modules et services de l'application fonctionnent ensemble comme prévu.

Endpoint par endpoint, j'ai vérifié que chaque route fonctionnait avec différents types d'input et j'ai fait attention à ce que les outputs renvoyés correspondent aux types de données attendues en retour.

Ci-dessous, une capture d'écran de l'envoi d'une requête POST sur Postman, route 'products'. Ceci correspond au end-point dédié à la création de produit. On peut voir que le code http renvoyé est '201 created' ce qui indique que la requête a bien fonctionné et qu'un nouveau produit a été ajouté à la base de données.

Ensuite, afin de vérifié que les données enregistrées correspondent au format attendu, je teste avec la méthode GET la products/[id], avec l'id du produit qui vient d'être créé.

Je peux alors constater que l'output est conforme.

The screenshot shows a Postman interface with the following details:

- Request URL:** `localhost:8080/products`
- Method:** `POST`
- Body:** `raw` (JSON)
- Body Content:**

```
1 {
2     "name": "Chemise à carreaux",
3     "description": "Chemise à carreaux rouge et bleue, tissus de qualité",
4     "userId": 10,
5     "picture": "https://www.cottonsociety.com/173158-thickbox_default/chemise-homme-gratte-carreaux-rouge-et-bleu.jpg",
6     "wishlist": "Veste noire en taille 40, pantalon noir en taille 40 ou chemises hommes en taille L",
7     "categoryId": 2,
8     "sizeId": 3,
9     "typeId": 14
10 }
```
- Response Status:** `201 Created`
- Response Time:** `224 ms`
- Response Size:** `387 B`

```

1 {
2   "name": "Chemise à carreaux",
3   "id": 37,
4   "description": "Chemise à carreaux rouge et bleue, tissus de qualité",
5   "picture": "https://www.cottonsociety.com/173158-thickbox_default/chemise-homme-gratte-carreaux-rouge-et-bleu.jpg",
6   "wishlist": "Veste noire en taille 40, pantalon noir en taille 40 ou chemises hommes en taille L",
7   "userId": 10,
8   "addedAt": "2024-05-10T14:23:04.202+00:00",
9   "sizeLabel": "S",
10  "userUsername": "ManonParis",
11  "typeLabel": "Chemise",
12  "categoryLabel": "Femme"
13 }

```

5.2 Tests de non-régression

Les tests de non-régression sont conçus pour s'assurer que les modifications ou les ajouts récents au code n'ont pas introduit de nouveaux bugs ou régressions dans les fonctionnalités existantes. Ces tests sont effectués après chaque modification importante du code pour vérifier que les fonctionnalités précédemment testées et validées continuent de fonctionner correctement.

Ici, à chaque fois qu'une fonctionnalité est sur le point d'être terminée ou que le code est refactoré, je vérifie, à l'aide d'une série de tests manuels, que tout ce qui a déjà été réalisé est toujours fonctionnel.

5.3 Tests des composants graphiques

Afin de vérifier que les composants graphiques sont stables, j'ai effectué quelques tests avec Playwright côté front. Cela m'a permis de valider que l'interface utilisateur fonctionne correctement et offre une expérience utilisateur cohérente.

Plus spécifiquement, j'ai testé les composants ‘Header’, ‘Card’ et ‘InformationBloc’ qui sont utilisés à plusieurs endroits de l'application. Pour le composant ‘Header’, j'ai vérifié que le logo, le titre et la description sont affichés correctement et que les liens de navigation fonctionnent comme prévu. Pour les composants ‘Card’ et ‘InformationBloc’, j'ai vérifié leur apparence visuelle, ainsi

que leur comportement lors de différentes interactions utilisateur, telles que le survol de la souris ou les clics.

```
PS C:\Users\Utilisateur\Documents\Projet-Nooky\projet-code\front> npx playwright test  
Running 12 tests using 2 workers  
12 passed (19.5s)
```

Test Results		
✓ card.test.ts		
✓ Product component renders correctly	chromium	957ms
card.test.ts:3		
✓ Product component renders correctly	firefox	3.6s
card.test.ts:3		
✓ Product component renders correctly	webkit	960ms
card.test.ts:3		
✓ header.test.ts		
✓ Header Component › should display the logo and title correctly	chromium	993ms
header.test.ts:11		
✓ Header Component › should display the navigation links correctly	chromium	800ms
header.test.ts:25		
✓ Header Component › should display the logo and title correctly	firefox	6.0s
header.test.ts:11		
✓ Header Component › should display the navigation links correctly	firefox	4.1s
header.test.ts:25		
✓ Header Component › should display the logo and title correctly	webkit	1.2s
header.test.ts:11		
✓ Header Component › should display the navigation links correctly	webkit	1.2s
header.test.ts:25		
✓ informationBloc.test.ts		
✓ InfoBloc component renders correctly	chromium	945ms
informationBloc.test.ts:3		
✓ InfoBloc component renders correctly	firefox	1.4s
informationBloc.test.ts:3		
✓ InfoBloc component renders correctly	webkit	993ms
informationBloc.test.ts:3		

5.4 Tests UAT sur Chrome et Mozilla

Les tests UAT (User Acceptance Testing), ou tests d'acceptation utilisateur, sont effectués pour vérifier que l'application répond aux exigences fonctionnelles et aux attentes des utilisateurs finaux.

Dans le cadre de ce projet, j'ai effectué ces tests sur Chrome et Mozilla pour m'assurer que l'application fonctionne correctement sur ces navigateurs populaires.

6 La veille sécurité – Injection SQL

6.1 Définition

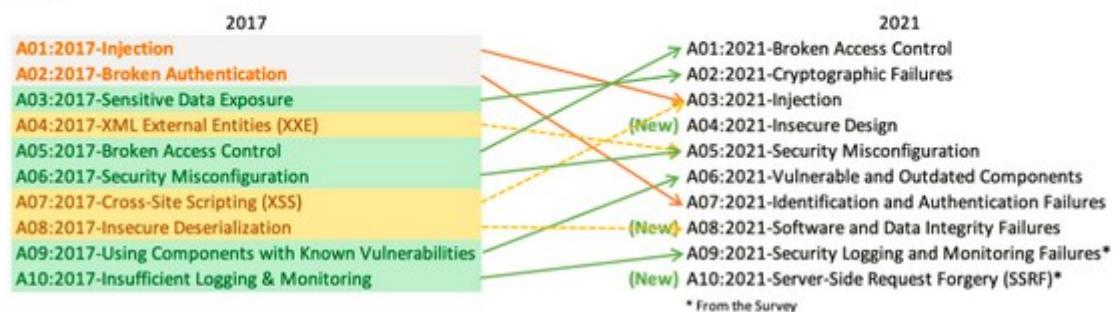
« La faille SQLi, abréviation de SQL Injection, soit injection SQL en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système et pouvant en compromettre la sécurité. »³

Les injections SQL font partie des attaques les plus fréquentes et les plus critiques. Dans le dernier rapport OWASP⁴ présentant les 10 risques les plus préoccupants concernant les applications web, les injections arrivent troisième⁵. Il s'agit de tout types d'injections. Les injections côté client, comme les injections XSS, et les injections côté serveur, comme les injections SQL.

Ci-dessous un extrait du dernier rapport OWASP (2021).

Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.



Elles visent généralement à récupérer des données personnelles en database comme des emails, mots de passe ou mots de passe hachés ou à prendre le contrôle du serveur.

6.2 Types d'injection SQL

Comment est-ce que les injections SQL sont envoyées au serveur de données ?

Les injections SQL peuvent être envoyées au serveur de données via différents vecteurs, le plus courant étant les champs de formulaire d'une interface utilisateur (IHM).

³ Wikipédia, article sur les injections SQL

⁴ OWASP (The Open Worldwide Application Security Project)

⁵ Le dernier rapport a été publié en 2021. Le prochain rapport est annoncé pour septembre 2024.

Par exemple, dans le formulaire de connexion ‘sign in’, on pourrait imaginer qu’un utilisateur malveillant pourrait inscrire la requête ‘SELECT * FROM users’ et lors de la soumission, envoyer cette requête en base de données.

Les techniques d’injection SQL les plus fréquentes sont les plus suivantes :

- **blind-based**

Cette technique consiste à envoyer des requêtes SQL qui ne divulguent pas directement les données dans la réponse de l’application, mais permettent à l’attaquant de déduire des informations en fonction des comportements de l’application. Par exemple, en envoyant des requêtes qui causent des modifications de comportement basées sur des conditions vraies ou fausses, l’attaquant peut progressivement extraire des données sensibles.

- **error-based**

Cette méthode est basée sur le détournement des messages d’erreur généré par le SGBD dans le but de récupérer des valeurs en BDD.

- **boolean-based**

Une technique similaire à la blind-based, où l’attaquant envoie des requêtes SQL qui évaluent des conditions booléennes. Selon que la condition est vraie ou fausse, l’application se comportera différemment, permettant ainsi à l’attaquant de déduire des informations sensibles par un processus de déduction logique.

- **time-based**

Cette technique implique l’envoi de requêtes SQL qui provoquent des délais dans la réponse de l’application. Par exemple, une requête SQL qui attend un certain nombre de secondes avant de répondre si une condition est vraie permet à l’attaquant de déterminer des informations en mesurant les temps de réponse.

- **union-based**

Cette méthode profite d’une requête SQL en cours en ajoutant des morceaux de code afin de récupérer un volume de données.

- **stacked queries**

Cette méthode est considérée comme particulièrement dangereuse car elle permet d’envoyer toute une série de requête à la base de données. Elle peut aussi bien de récupérer des données que de les modifier. Elle exploite une mauvaise configuration du serveur de la base de données.

Face à la menace des injections SQL et à leur potentiel destructeur, il est impératif de mettre en place des mesures de protection adéquates pour sécuriser les applications web contre ces attaques.

6.3 Adaptation du code

Prendre conscience du danger des injections SQL m'a amenée à adapter mon code et à être attentive à certains aspects clés pour renforcer la sécurité.

6.3.1 DTO

Ne pas exposer directement les entités de la base de données et utiliser une couche DTO lors de la récupération des données côté front est une bonne pratique de sécurité.

Les DTO permettent de contrôler les données transférées entre les couches de l'application et d'appliquer des contraintes de validation.

L'utilisation de Jakarta Bean Validation permet d'ajouter des contraintes de validation directement sur les DTO, garantissant que seules des données valides et sécurisées sont traitées par l'application.

```
7  public class CreateProduct {
8
9  @NotNull
10 @Size(max= 50)
11 private String name;
12
13 @Size(max= 20)
14 private String reference;
15
16 @Size(max= 400)
17 private String description;
18
19 @Size(max= 1000)
20 private String picture;
21
22 @Size(max= 200)
23 private String wishlist;
24
25 @NotNull
26 @Positive
27 private Long categoryId;
28
29 @NotNull
30 @Positive
31 private Long sizeId;
32
33 @NotNull
34 private Long typeTd;
```

6.3.2 JPA

L'utilisation de JPA pour l'accès aux données permet de bénéficier de la sécurité intégrée fournie par les ORM (Object-Relational Mapping).

JPA utilise des entités pour représenter les données de la base et offre des fonctionnalités pour gérer les relations et les requêtes de manière sécurisée.

6.3.3 Requêtes paramétrées

L'utilisation de requêtes paramétrées est une méthode efficace pour prévenir les injections SQL. Au lieu de concaténer des chaînes de requêtes SQL, les paramètres sont passés séparément, empêchant ainsi l'injection de code malveillant.

Les ORM, comme Hibernate, offrent des mécanismes pour protéger contre les injections SQL. Ils utilisent des requêtes paramétrées et automatisent la génération de SQL sécurisé. Les ‘?’ représentent les paramètres passés séparément.

Exemple ci-dessous avec une requête de création d'un produit.

```

--  

56     product.setAddedAt(timestamp);  

57  

58     product.setCategory(categoryRepository.getReferer());  

59     product.setSize(sizeRepository.getReferenceById(id));  

60     product.setType(typeRepository.getReferenceById(id));  

61     product.setUser(userRepository.getReferenceById(id));  

62  

63     productRepository.save(product);  

64 }  

65

```

Ci-dessous la façon dont Hibernate traduit la requête en SQL.

```
Hibernate: select p1_0.id,p1_0.name,p1_0.description,p1_0.picture,p1_0.category_id,p1_0.type_id,p1_0.size_id from products p1_0 where p1_0.user_id=?
```

6.4 Les sources

Afin de comprendre ce que sont les injections SQL et adapter mon code à la prévention de ces attaques, j'ai consulté un certain nombre de ressources.

- Injections SQL – articles Wikipédia en anglais et en français
- OWASP – Rapport OWASP 2021
- OpenClassrooms, cours mis à jour le 30 novembre 2023 – Réalisez un test d'intrusion web – Chapitre : Identifiez les vulnérabilités de l'application web
- Article Baeldung mis à jour le 8 janvier 2024 : <https://www.baeldung.com/sql-injection>

Conclusion

Le projet Nooky a représenté une expérience enrichissante et formatrice, consolidant mes compétences en développement d'applications informatiques. J'ai eu l'opportunité de mener à bien toutes les étapes clés, depuis la définition des besoins jusqu'à la phase de tests, en passant par la conception et l'implémentation.

La méthodologie Agile et l'utilisation de l'outil Trello ont permis une gestion efficace du projet, assurant une organisation et une flexibilité optimales. La conception du graphisme, des données, du traitement sur ces données, ainsi que le choix des technologies appropriées pour chaque couche de l'application, ont été des étapes cruciales qui ont facilité l'implémentation des fonctionnalités.

La mise en œuvre de la sécurité, notamment à travers l'utilisation de JWT et bcrypt, ainsi que les précautions prises contre les injections SQL, ont renforcé la fiabilité de l'application. Les tests rigoureux, tant pour le back-end que pour le front-end, ont garanti la robustesse et la qualité des fonctionnalités développées.

Tout cela a été l'occasion d'approfondir mes connaissances et de développer de nouvelles compétences, notamment en conception d'architecture logicielle, en gestion de base de données relationnelle et en développement d'interfaces utilisateur. Je suis désormais impatiente d'appliquer ces acquis et de continuer à évoluer dans ma future vie professionnelle.

Bibliographie

Cyril Gruau (2006), « **Conception d'une base de données : support de cours** ». (44 pages).

Nanci D., B. Espinasse avec la collaboration de B. Cohen, J.C. Asselborn et H. Heckenroth (2001), « **Ingénierie des systèmes d'information : Merise deuxième génération** », Vuibert éditions, Paris. ISBN : 2-7117-8674-9 (416 pages).

Ivar Jacobson, Ian Spence et Brian Kerr (2016), « **Use-Case 2.0, The hub of software development** », Development. (30 pages).

Kathy Sierra, Bert Bates and Trisha Gee (2022), « **Head First Java: A Brain-Friendly Guide** », O'Reilly Media, 3e édition. ISBN :978-1491910771 (688 pages).

Eric T Morrison and Elisabeth Robson (2014), « **Head First JavaScript Programming** », O'Reilly. ISBN : 978-1449340131 (704 pages).

« **Injections SQL** » – articles Wikipédia en anglais et en français

OWASP (2021), « **Rapport OWASP Top 10** ». <https://owasp.org/Top10/fr/>

Thibaut Bonnetain et Étienne Capgras (2023), « **Réalisez un test d'intrusion web – Chapitre : Identifiez les vulnérabilités de l'application web** », OpenClassrooms.
<https://openclassrooms.com/fr/courses/7727176-realisez-un-test-dintrusion-web/>

Philippe Sevestre et Michal Aibin (2024), « **SQL Injection and How to Prevent It?** », Baeldung.
<https://www.baeldung.com/sql-injection>

MDN Web docs : <https://developer.mozilla.org/en-US/>

ANSSI : <https://cyber.gouv.fr/>

Remerciements

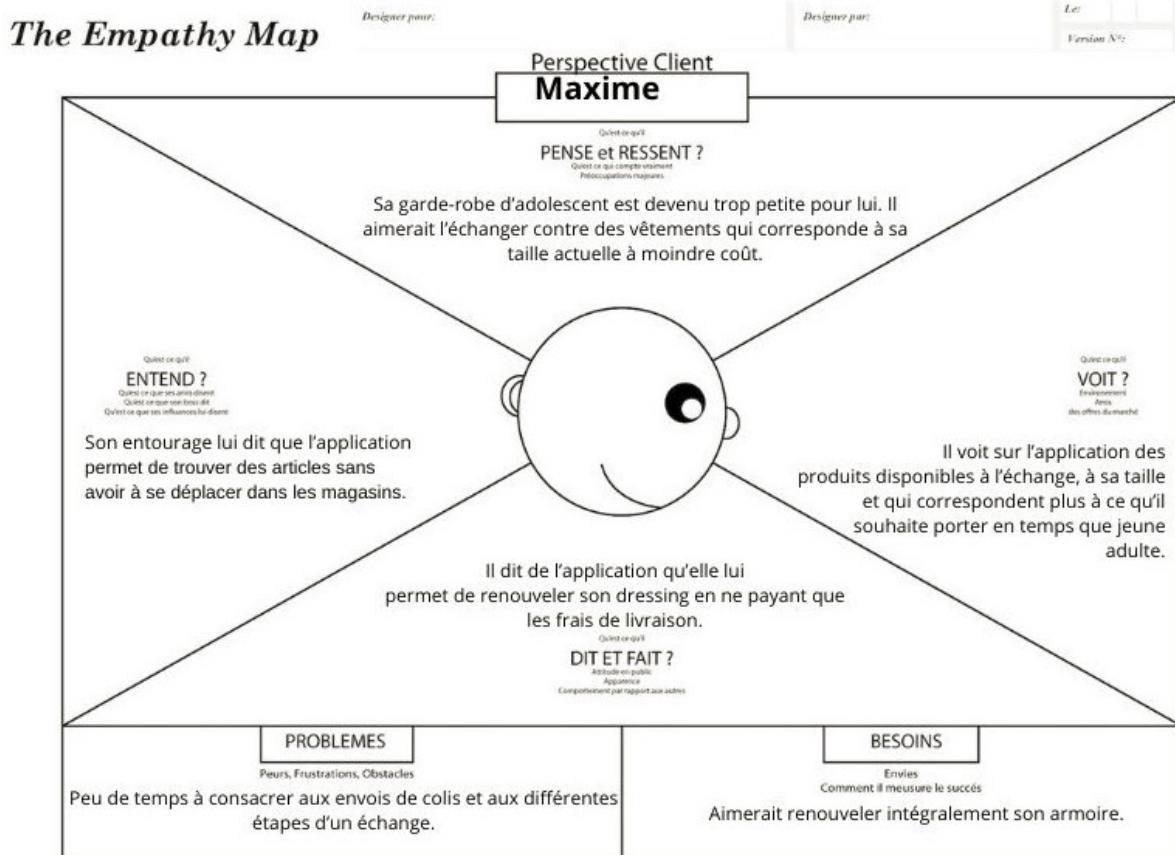
Je tiens à exprimer ma reconnaissance à l'entreprise DSI pour m'avoir offert cette opportunité de formation, et plus spécifiquement à Xavier Barrois, mon manager, pour son accompagnement tout au long de cette expérience. Un immense merci également à toute l'équipe pédagogique de l'école Simplon, et en particulier à Frank Marshall et Hugo Arru, nos formateurs, ainsi qu'à Célia Dulac, chargée de formation, pour leur engagement et leur expertise qui ont été déterminants dans mon apprentissage. Enfin, je remercie chaleureusement mes collègues, partenaires de programmation, auprès de qui j'ai pu partager des moments de collaboration enrichissants, et qui ont été une source d'inspiration et de motivation tout au long de cette aventure.

Annexes

Persona numéro 2	p. 58
Persona numéro 3	p. 59
Zoning de tous les écrans de l'application	p. 60
Wireframe de tous les écrans de l'application	p. 62
Maquette statique de tous les écrans de l'application	p. 66
MPD	p. 71
Diagramme de classes	p. 72
Script SQL de création des tables	p. 73

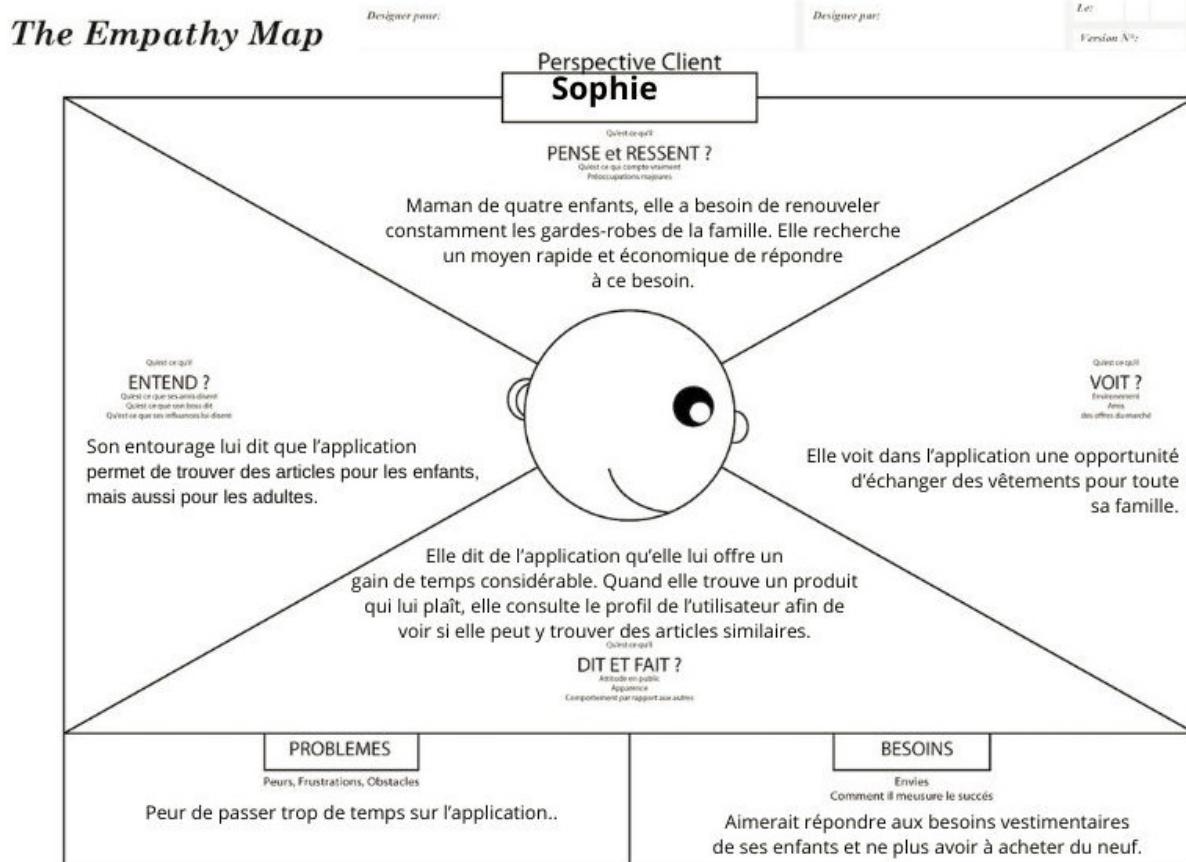
Persona numéro 2

Maxime, l'étudiant à petit budget



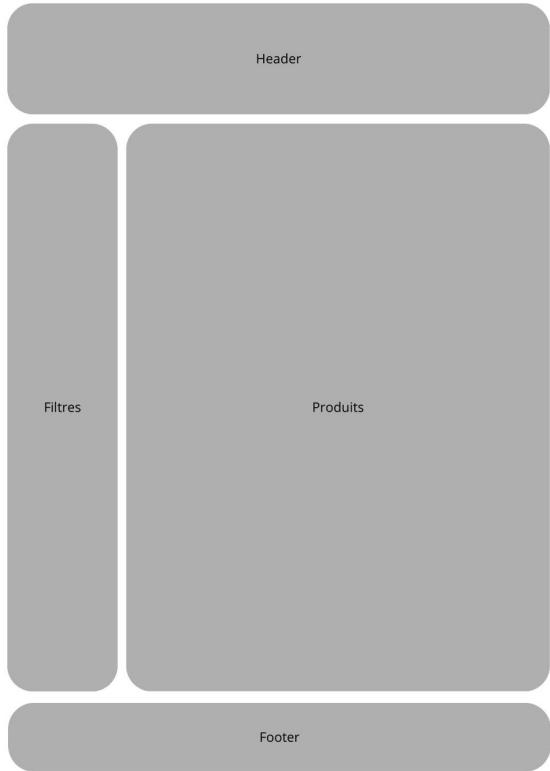
Persona 3

Sophie, la maman organisées

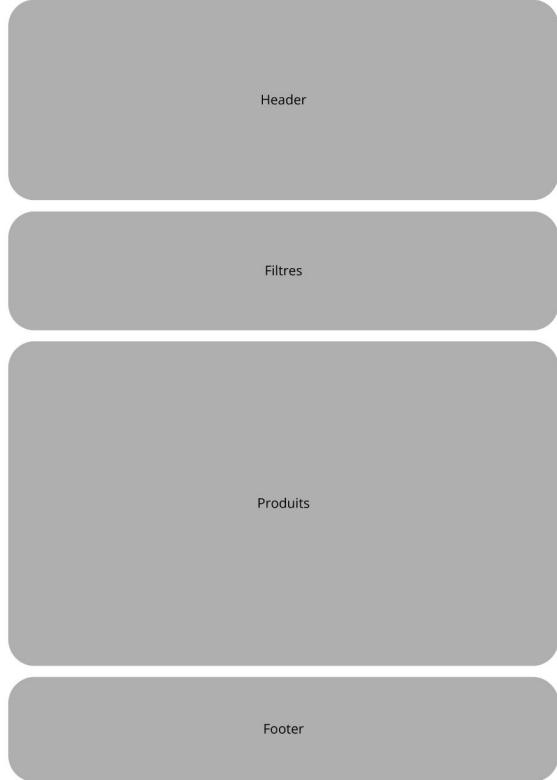


Zoning de tous les écrans de l'application

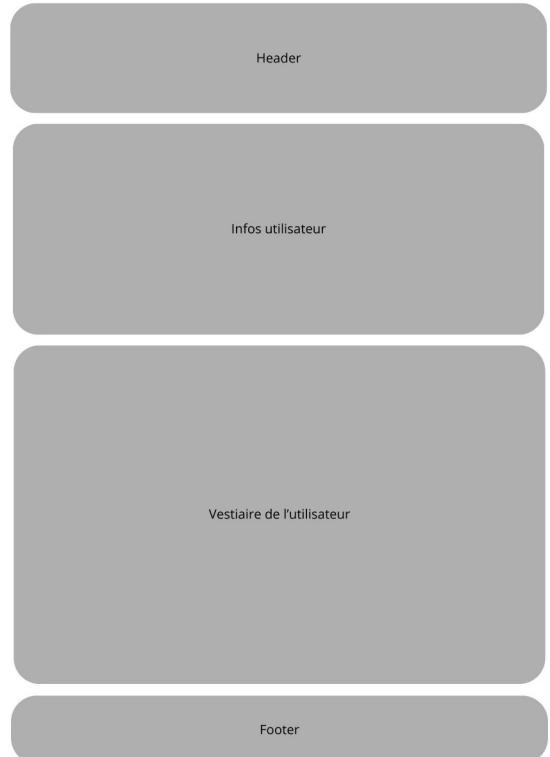
Home desktop



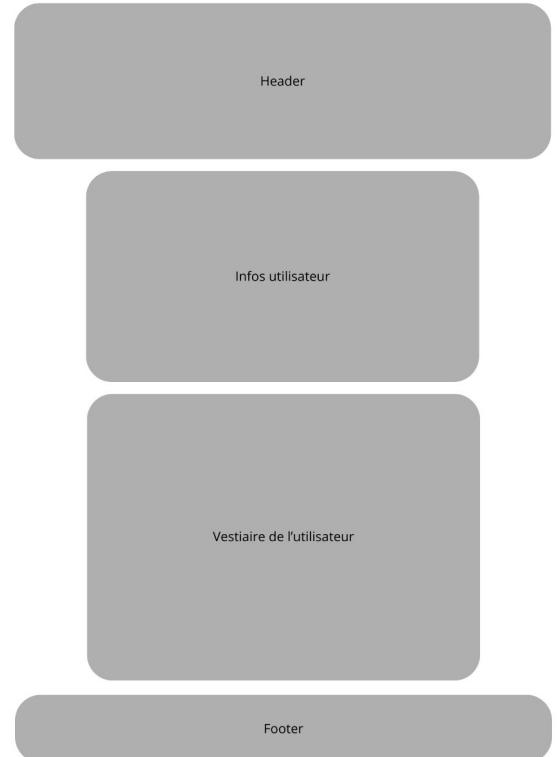
Home mobile



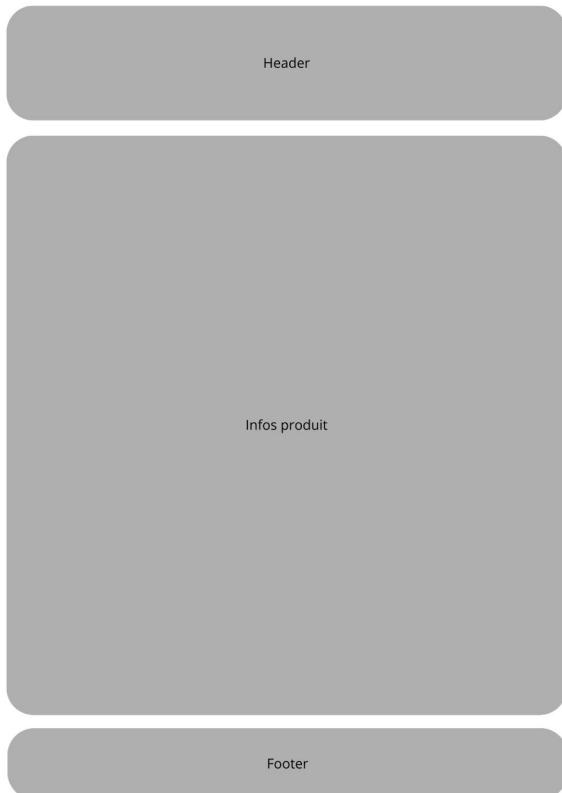
Compte utilisateur desktop



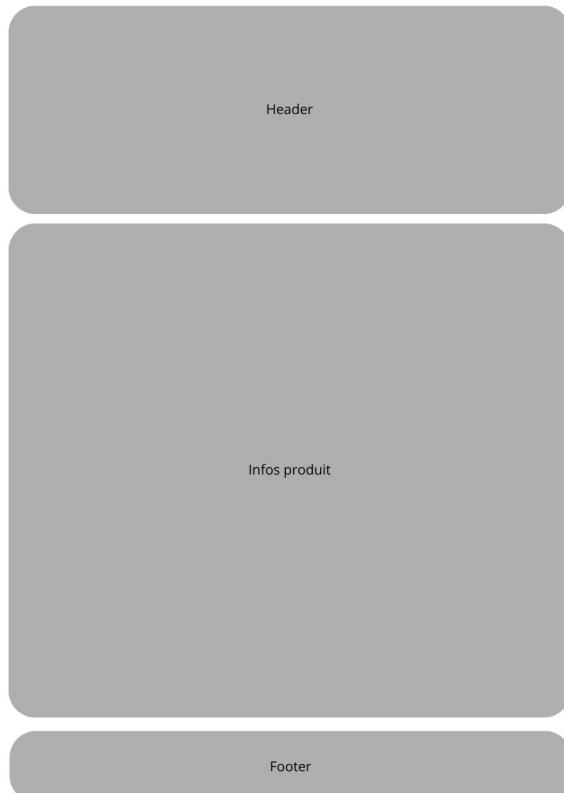
Compte utilisateur mobile



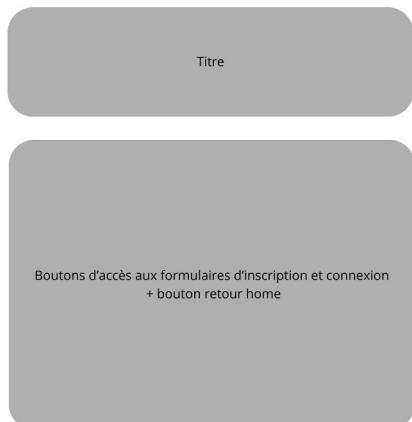
Fiche produit desktop



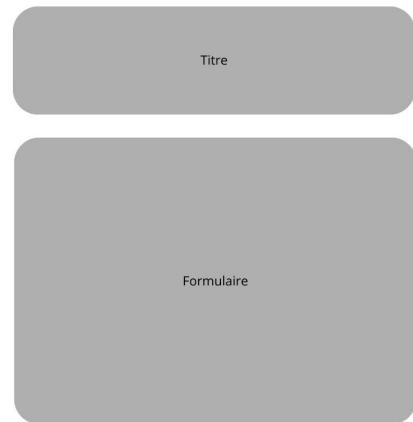
Fiche produit mobile



Formulaire d'inscription ou connexion desktop et mobile

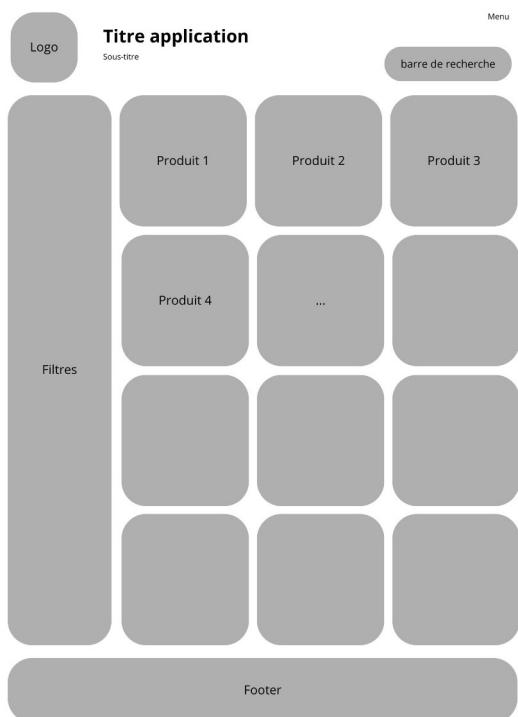


Formulaire d'ajout ou de modification d'un produit desktop ou mobile

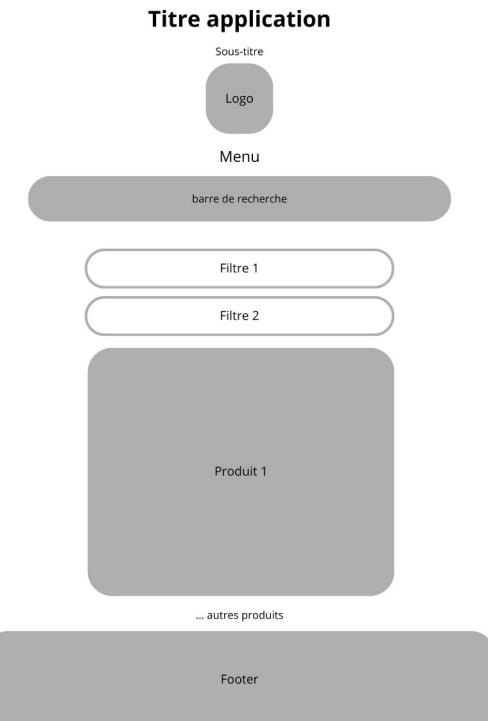


Wireframe de tous les écrans de l'application

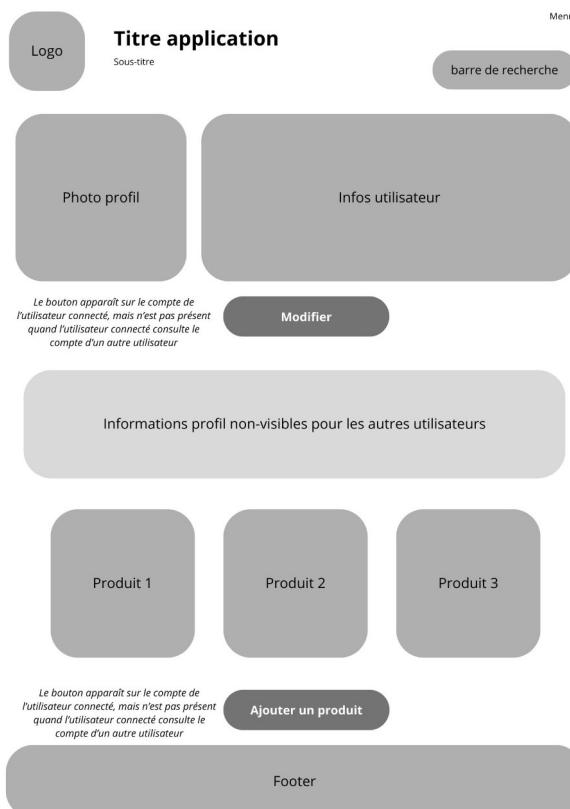
Home desktop



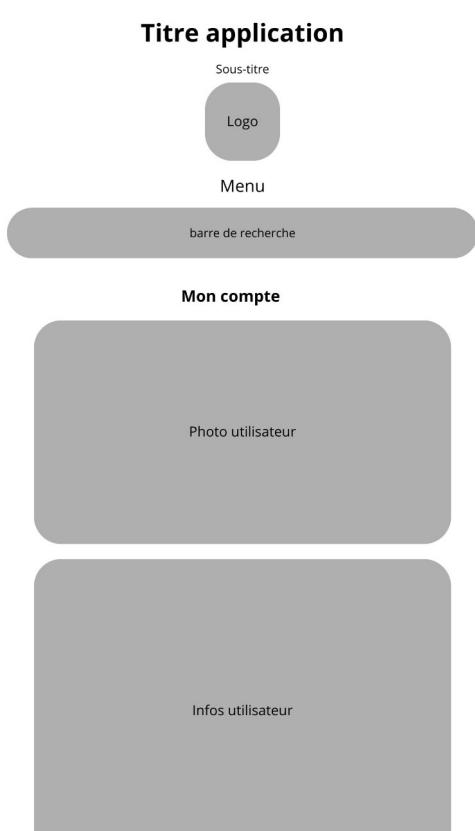
Home mobile



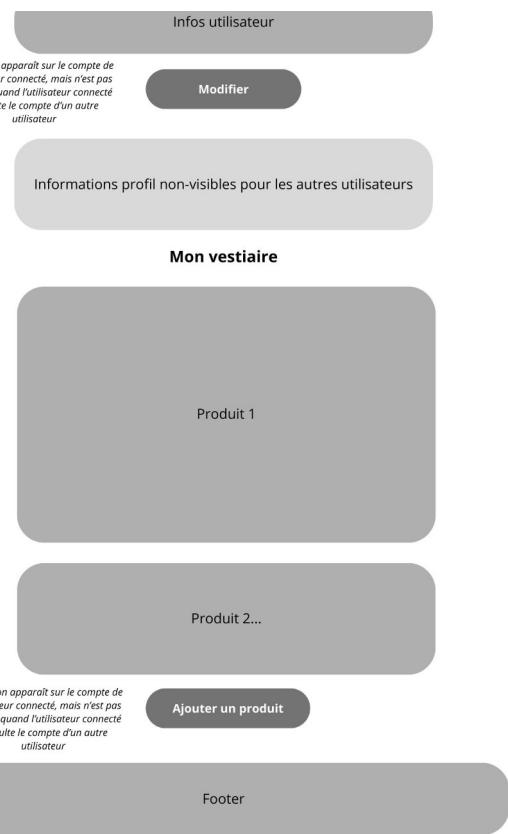
Compte utilisateur desktop



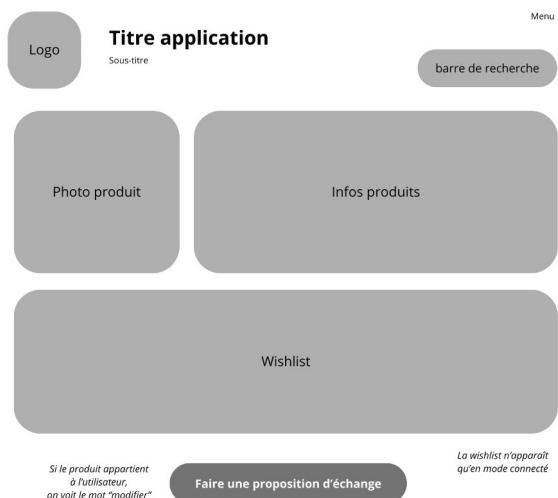
Compte utilisateur mobile - top



Compte utilisateur - bottom

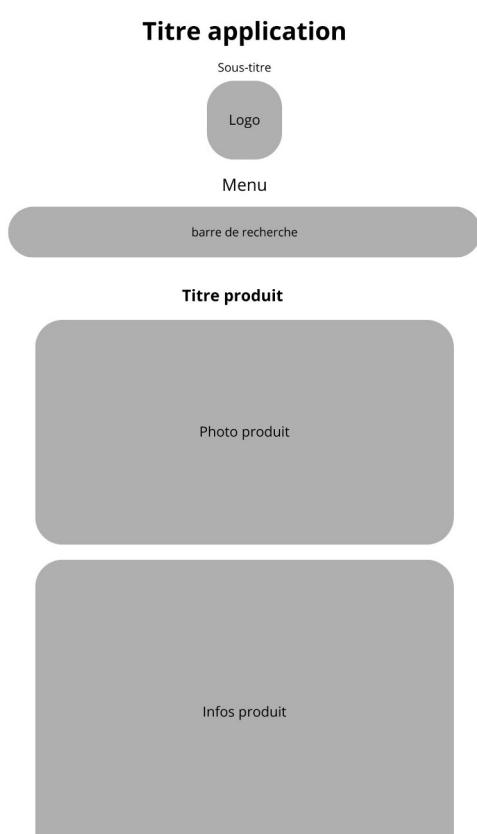


Fiche produit desktop

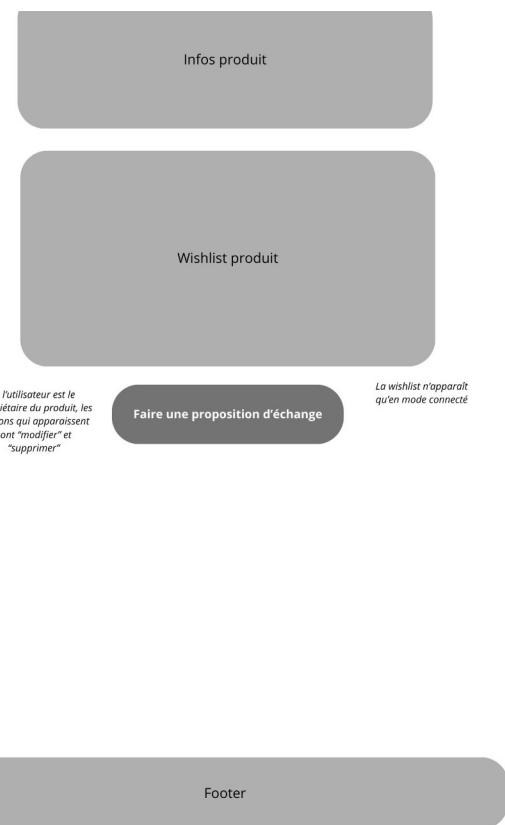


Footer

Fiche produit mobile - top



Fiche produit mobile - bottom



Formulaire d'inscription ou connexion desktop et mobile

Titre

Description bouton
Bouton Sign up

Description bouton
Bouton Sign in

Retour home

Formulaire d'inscription desktop et mobile

Titre formulaire inscription

Input nom d'utilisateur

Input email

Input mot de passe

Input prénom

Input nom de famille

Input numéro et rue

Input code postal

Input ville

Submit

[Retour vers page d'accès aux formulaires d'inscription et connexion](#)

Formulaire de connexion desktop et mobile

Titre du formulaire

Input email

Input mot de passe

Submit

[Retour vers page d'accès aux formulaires d'inscription et connexion](#)

Formulaire d'ajout et modification de produit desktop et mobile

Titre du formulaire

Input produit

Input description du produit

Input url de l'image

Input Wishlist

Select catégorie >

Select type >

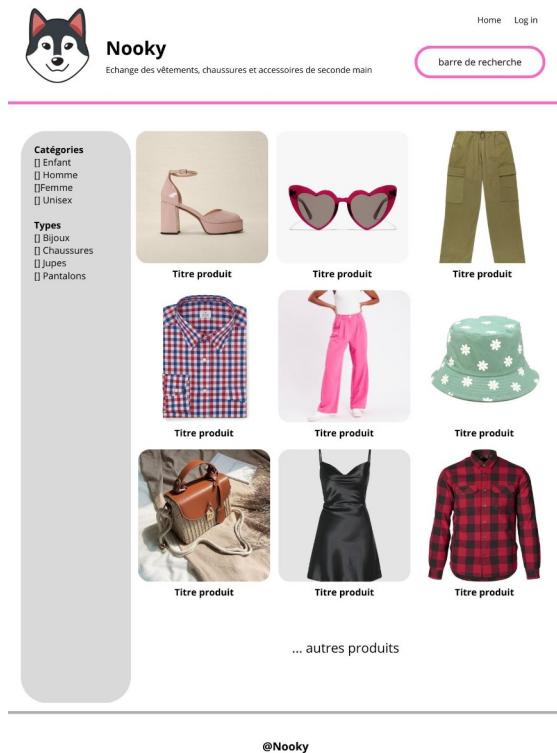
Select size >

Submit

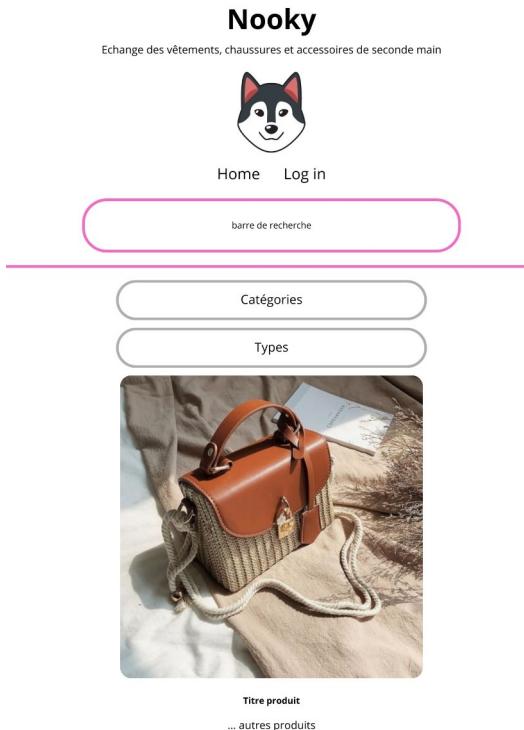
[Retour vers le compte de l'utilisateur](#)

Maquettes statiques

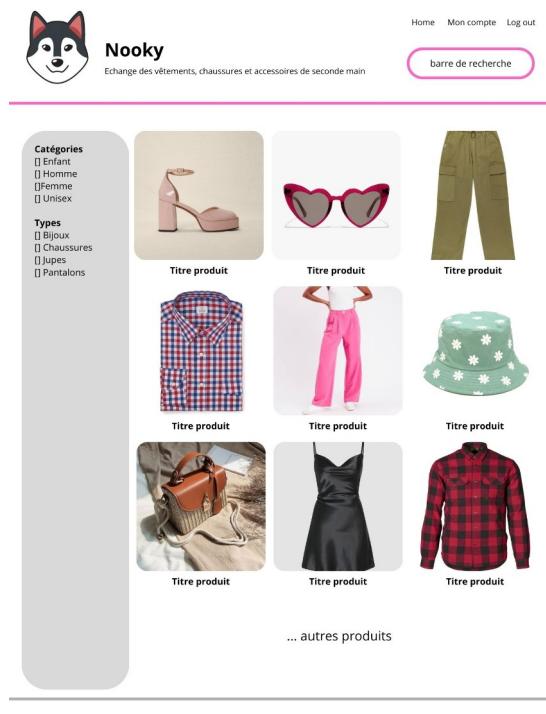
Home desktop – user non-connecté



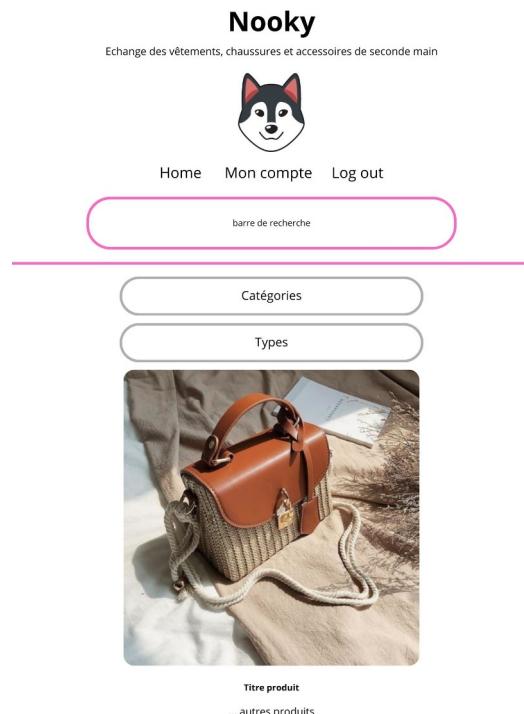
Home mobile – user non-connecté



Home desktop – user connecté



Home mobile – user connecté



Compte utilisateur desktop

Nooky
Echange des vêtements, chaussures et accessoires de seconde main

Home Mon compte Log out

barre de recherche

Mon compte

 Descriptif profil

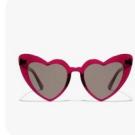
Le bouton apparaît sur le compte de l'utilisateur connecté, mais n'est pas présent quand l'utilisateur connecté consulte le compte d'un autre utilisateur

Modifier

 Informations profil non-visibles pour les autres utilisateurs

Mon vestiaire

 Titre produit

 Titre produit

 Titre produit

Le bouton apparaît sur le compte de l'utilisateur connecté, mais n'est pas présent quand l'utilisateur connecté consulte le compte d'un autre utilisateur

Ajouter un produit

@Nooky

Compte utilisateur mobile - top

Nooky
Echange des vêtements, chaussures et accessoires de seconde main

Home Mon compte Log out

barre de recherche

Mon compte

 #115378

Descriptif profil

Compte utilisateur mobile - bottom

Descriptif profil

Le bouton apparaît sur le compte de l'utilisateur connecté, mais n'est pas présent quand l'utilisateur connecté consulte le compte d'un autre utilisateur

Mon vestiaire

 Titre produit

Le bouton apparaît sur le compte de l'utilisateur connecté, mais n'est pas présent quand l'utilisateur connecté consulte le compte d'un autre utilisateur

Ajouter un produit

@Nooky

Fiche produit desktop

The screenshot shows a desktop product page for a red and blue checkered shirt. At the top, there's a navigation bar with the Nooky logo, a search bar, and links for Home, Mon compte, and Log out. Below the navigation is a title section labeled "Titre produit". To the left of the title is a small image of the shirt, and to the right is a large gray box labeled "Descriptif produit". Below the title is a button labeled "Wishlist échange". A callout box points to this button with the text: "Si l'utilisateur est le propriétaire du produit, les boutons qui apparaissent sont 'modifier' et 'supprimer'". Another callout box to the right says: "La wishlist n'apparaît qu'en mode connecté". At the bottom of the page is a footer with the text "@Nooky".

Fiche produit mobile - top

Nooky

Echange des vêtements, chaussures et accessoires de seconde main



Home Mon compte Log out

barre de recherche

Titre produit



Descriptif produit

Fiche produit mobile - bottom

Nooky

Echange des vêtements, chaussures et accessoires de seconde main



Home Mon compte Log out

barre de recherche

... descriptif produit

Wishlist échange

Si l'utilisateur est le propriétaire du produit, les boutons qui apparaissent sont "modifier" ou "supprimer"

Faire une proposition d'échange

La wishlist n'apparaît qu'en mode connecté

@Nooky

Formulaire d'inscription ou connexion desktop et mobile

Bienvenue dans l'application Nooky

Vous souhaitez vous inscrire ?

[Sign up](#)

Déjà inscrit, vous souhaitez vous connecter ?

[Sign in](#)

[Retour](#)

Formulaire d'inscription desktop et mobile

Sign up

Entrez votre nom d'utilisateur

Entrez votre email

Entrez votre mot de passe

Entrez votre prénom

Entrez votre nom de famille

Entrez votre numéro et rue

Entrez votre code postal

Entrez votre ville

[Valider](#)

[Retour](#)

Sign up

Entrez votre nom d'utilisateur

Entrez votre email

Entrez votre mot de passe

Entrez votre prénom

Entrez votre nom de famille

Entrez votre numéro et rue

Entrez votre code postal

Entrez votre ville

[Valider](#)

Code postal invalide

La validation n'a pas fonctionnée, le compte n'a pas été créé

Un compte utilisateur avec cette adresse email existe déjà, veuillez vous connecter sur la page sign in

[Sign in](#)

[Retour](#)

Formulaire de connexion desktop et mobile

Sign in

Entrez votre email

Entrez votre mot de passe

Valider

Retour

Sign in

Entrez votre email

Entrez votre mot de passe

Identifiant et mot de passe erronés

Valider

Retour

Formulaire d'ajout de produit desktop et mobile

Ajouter un produit

Entrez le nom du produit

Entrez la description du produit

Entrez l'url de l'image

Produits souhaités en échange

Catégorie >

Type >

Size >

Valider

Retour

Formulaire de modification d'un produit desktop et mobile

Modifier un produit

Jean bleu

Jean bleu en très bon état marque Levis

http://

Une veste en jean taille 34-36

Femme

Manteaux et veste

S

Valider

Retour

Le MPD

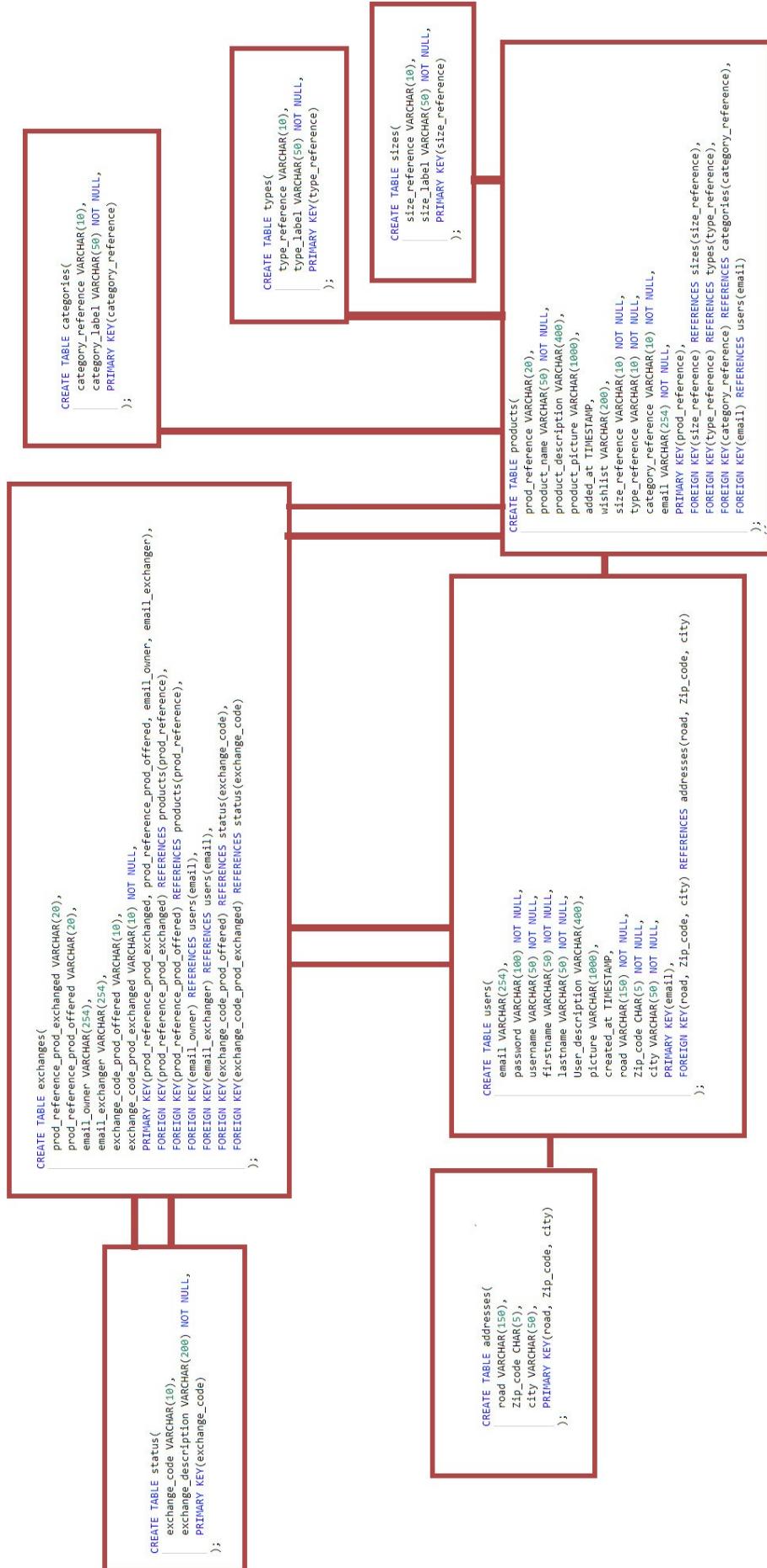
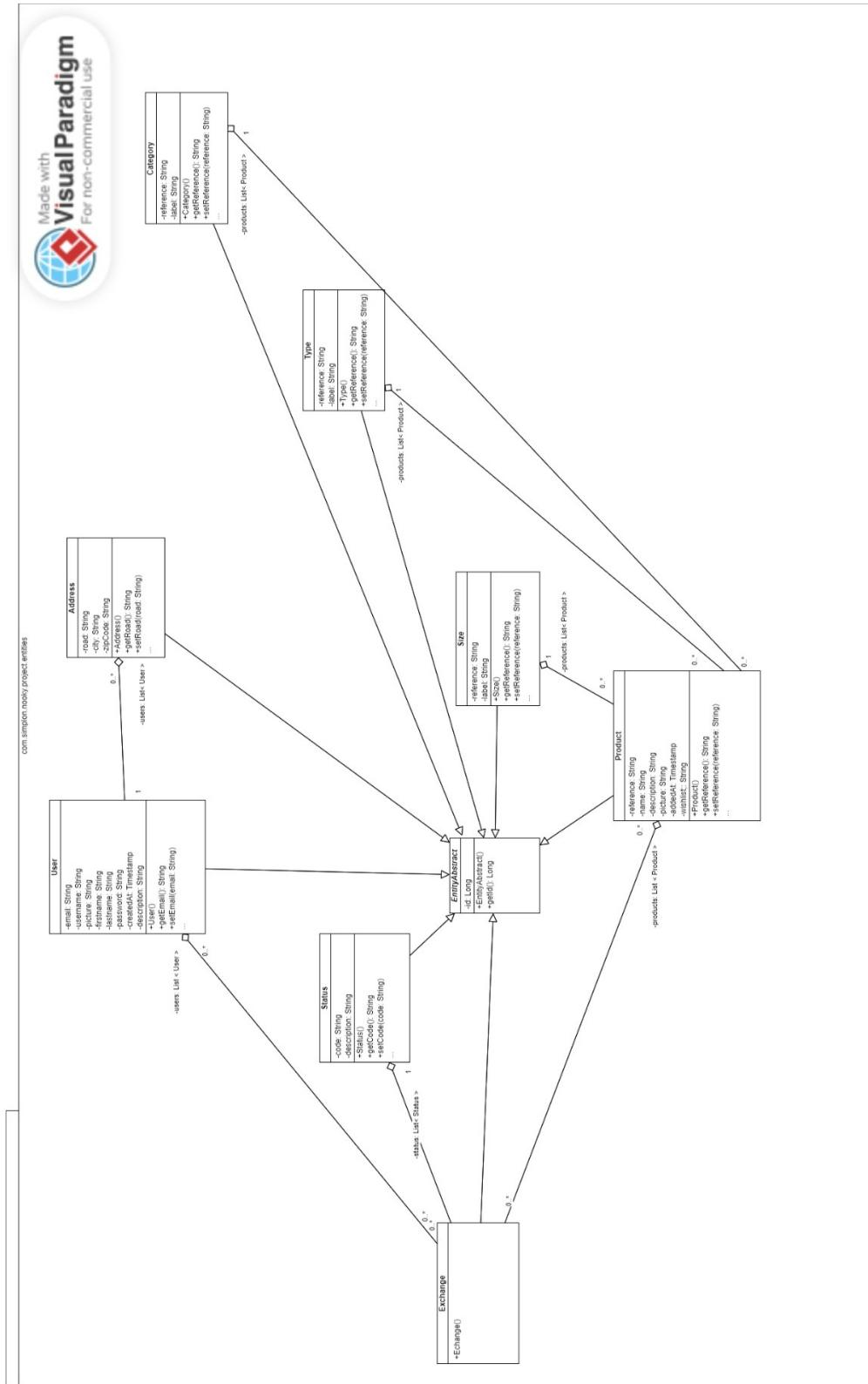


Diagramme de classes



Script SQL

```
C: > Users > Utilisateur > Documents > Proket-Nooky > projet-code > database > schema1-ddl.sql
1  CREATE TABLE "categories" (
2    id SERIAL PRIMARY KEY,
3    reference VARCHAR(10) UNIQUE NOT NULL,
4    label VARCHAR(50) NOT NULL
5  );
6
7  CREATE TABLE "sizes" (
8    id SERIAL PRIMARY KEY,
9    reference VARCHAR(10) UNIQUE NOT NULL,
10   label VARCHAR(50) NOT NULL
11 );
12
13 CREATE TABLE "types" (
14   id SERIAL PRIMARY KEY,
15   reference VARCHAR(10) UNIQUE NOT NULL,
16   label VARCHAR(50) NOT NULL
17 );
18
19 CREATE TABLE "status" (
20   id SERIAL PRIMARY KEY,
21   code VARCHAR(10) UNIQUE NOT NULL,
22   description VARCHAR(200) NOT NULL
23 );
24
25 CREATE TABLE "addresses" (
26   id SERIAL PRIMARY KEY,
27   road VARCHAR(150) NOT NULL,
28   zip_code CHAR(5) NOT NULL,
29   city VARCHAR(50) NOT NULL,
30   CONSTRAINT addresses_ukey UNIQUE (road, zip_code, city)
31 );
32
```

```

33 CREATE TABLE "users" (
34     id SERIAL PRIMARY KEY,
35     email VARCHAR(254) UNIQUE NOT NULL,
36     password VARCHAR(100) NOT NULL,
37     username VARCHAR(50) NOT NULL,
38     description VARCHAR(400),
39     picture VARCHAR(1000),
40     firstname VARCHAR(50) NOT NULL,
41     lastname VARCHAR(50) NOT NULL,
42     created_at TIMESTAMP,
43     address_id INT NOT NULL,
44     FOREIGN KEY (address_id) REFERENCES addresses(id)
45 );
46
47 CREATE TABLE "products" (
48     id SERIAL PRIMARY KEY,
49     reference VARCHAR(1000) UNIQUE NOT NULL,
50     name VARCHAR(50) NOT NULL,
51     description VARCHAR(400),
52     picture VARCHAR(1000),
53     added_at TIMESTAMP,
54     wishlist VARCHAR(200),
55     category_id INT NOT NULL,
56     size_id INT NOT NULL,
57     type_id INT NOT NULL,
58     user_id INT NOT NULL,
59     FOREIGN KEY (category_id) REFERENCES categories(id),
60     FOREIGN KEY (size_id) REFERENCES sizes(id),
61     FOREIGN KEY (type_id) REFERENCES types(id),
62     FOREIGN KEY (user_id) REFERENCES users(id)
63 );
64
65 CREATE TABLE "exchanges" (
66     id SERIAL PRIMARY KEY,
67     product_offered_id INT NOT NULL,
68     product_exchanged_id INT NOT NULL,
69     owner_id INT NOT NULL,
70     exchanger_id INT NOT NULL,
71     status_prod_offered_id INT NOT NULL,
72     status_prod_exchanged_id INT NOT NULL,
73     CONSTRAINT exchanges_ukey UNIQUE (product_offered_id, product_exchanged_id, owner_id, exchanger_id),
74     FOREIGN KEY(product_offered_id) REFERENCES products(id),
75     FOREIGN KEY(product_exchanged_id) REFERENCES products(id),
76     FOREIGN KEY(owner_id) REFERENCES users(id),
77     FOREIGN KEY(exchanger_id) REFERENCES users(id),
78     FOREIGN KEY(status_prod_offered_id) REFERENCES status(id),
79     FOREIGN KEY(status_prod_exchanged_id) REFERENCES status(id)
80 );
81
82
83
84

```