



*Projet Nooky*

Soutenance  
Concepteur Développeur d'Applications

04/06/2024



# Introduction

# Introduction

- Humanities background
- Web dev diploma - august 2023
- Enrolled at Simplon - september 2023

# Plan

- 1) Le contexte
- 2) La conception graphique
- 3) Les données
- 4) Les traitements
- 5) L'architecture
- 6) L'implémentation
- 7) Les tests
- 8) Veille sécurité
- 9) Démo



# Partie 1 : le contexte

# Présentation de l'application

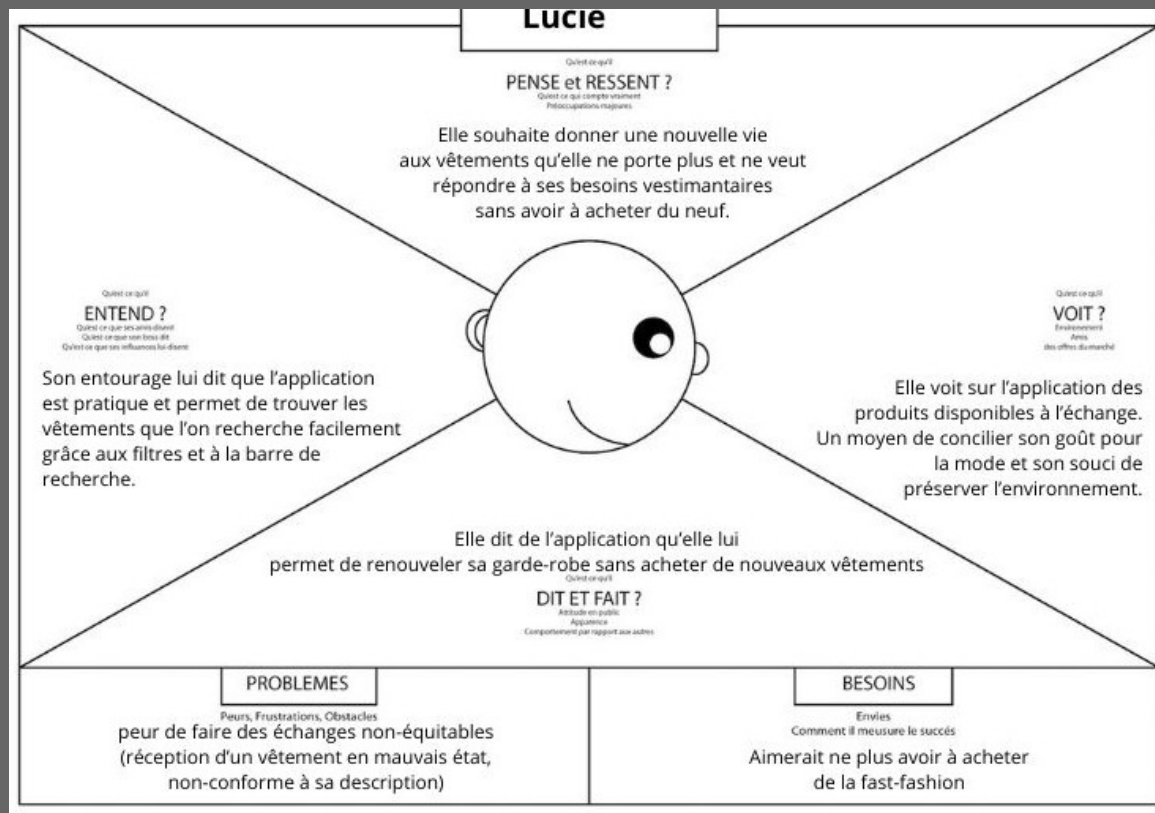
Le pitch

# Les utilisateurs cibles

La technique des personas :

- Lucie, passionnée de mode et éco-responsable
- Maxime, l'étudiant à petit-budget
- Sophie, la maman organisée

# Persona 1





# Les principales fonctionnalités

- Compte utilisateur
- Produit
- Recherche
- Échanges

# L'expression des besoins

- La méthode des User stories
- Construites selon un modèle précis
- Classées par thème (connexion, produit, etc.)

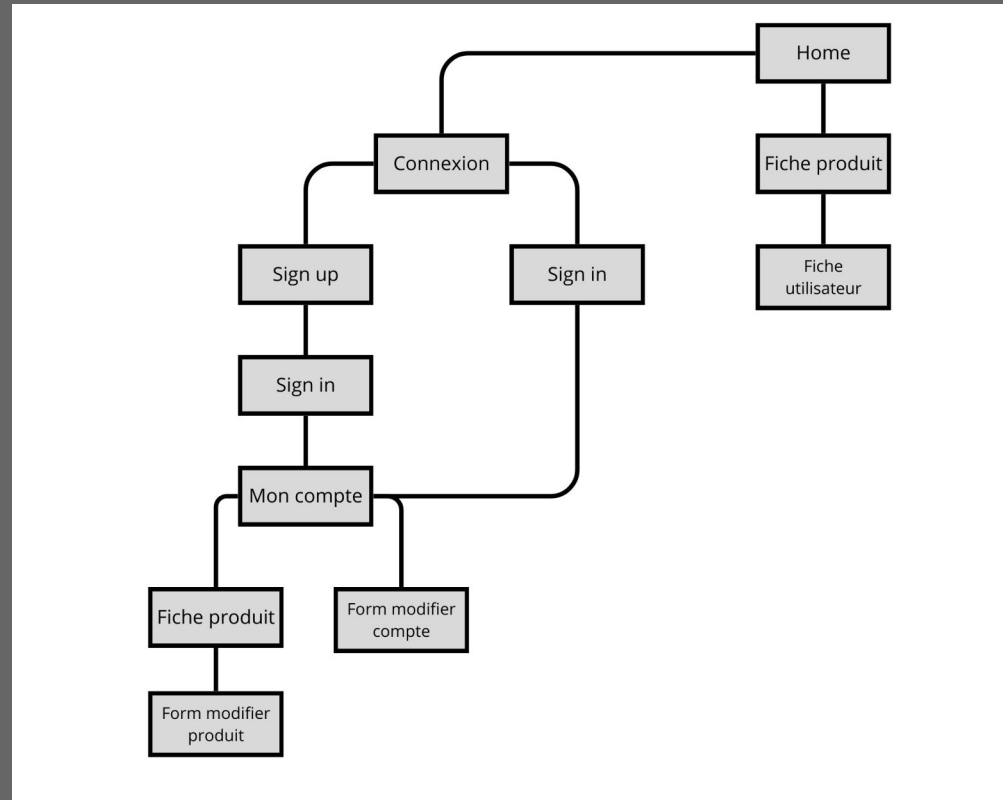
## Exemple de User storie

*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.*

# Les objectifs de qualité

- Performance
- Eco-conception
- Test
- Responsive

# L'arborescence du site



# La gestion du projet

- Agilité : Kanban
- Outil de gestion de projet : [Trello](#)

# DOD

Afin d'être considérée comme terminée, la fonctionnalité doit :

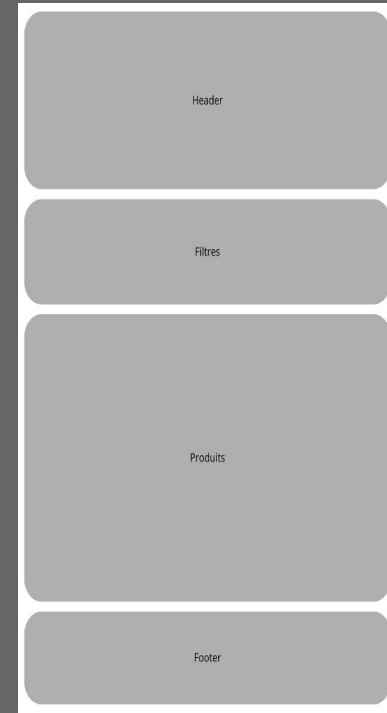
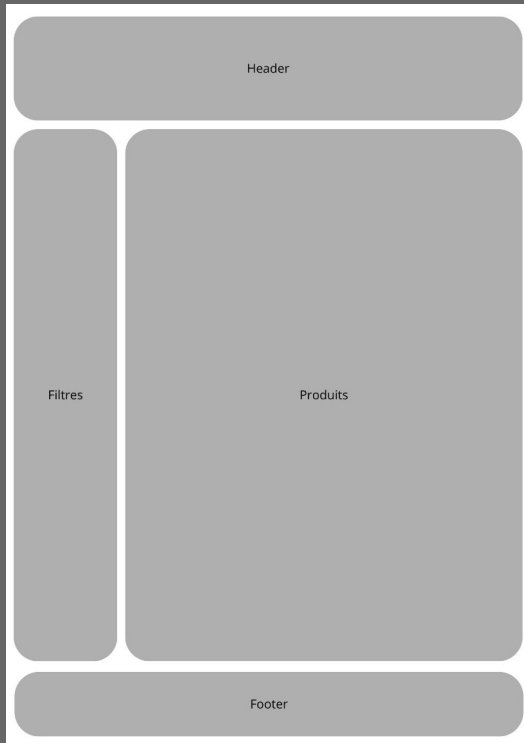
- avoir été conçue avec un souci de performance
- avoir été conçue avec un souci d'éco-conception
- avoir été testée
- avoir des écrans responsives
- être conforme aux spécifications fonctionnelles initiales



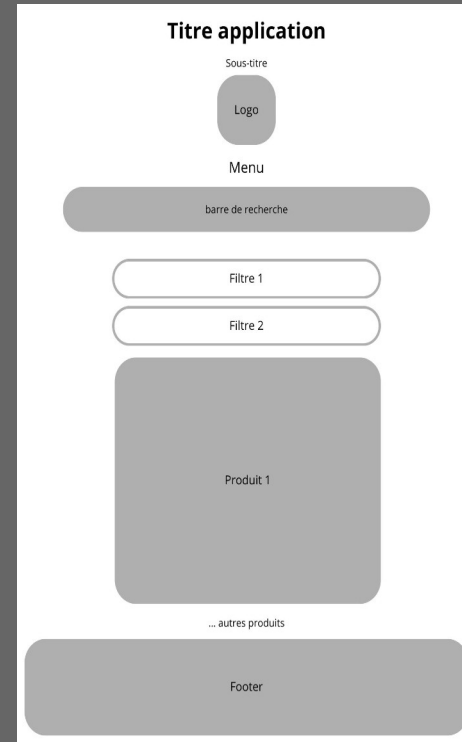
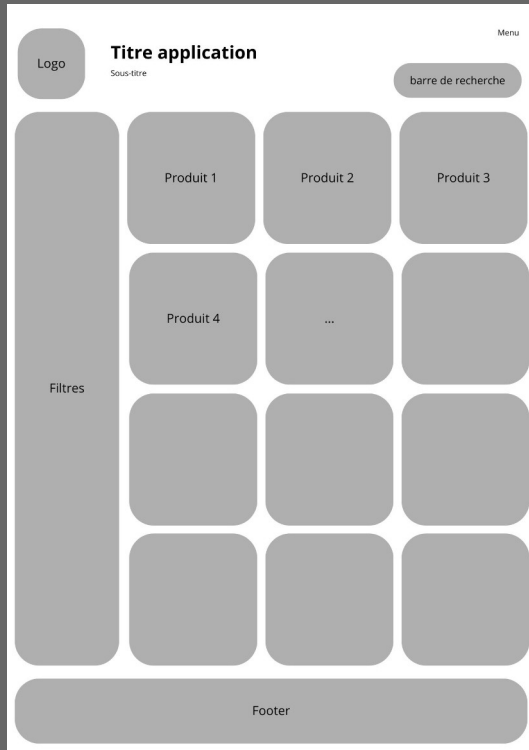
## Partie 2 : la conception graphique



# Le zoning



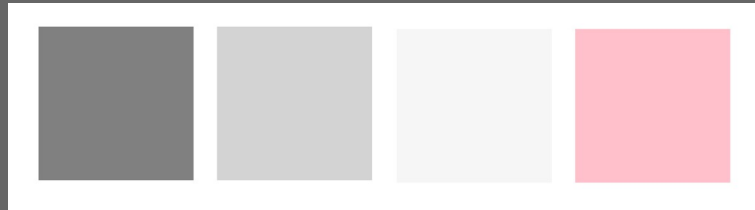
# Le wireframe



# La charte graphique

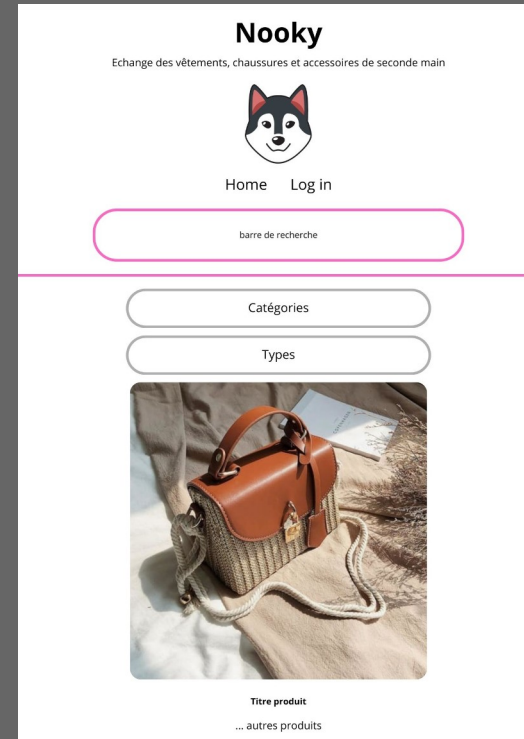
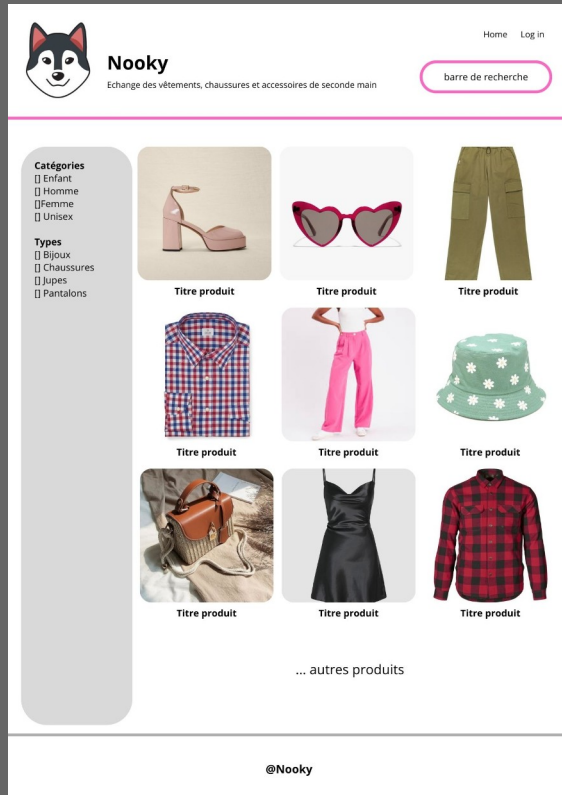
- Polices d'écriture
- Couleurs
- Une mascotte

Montserrat



**Nooky**

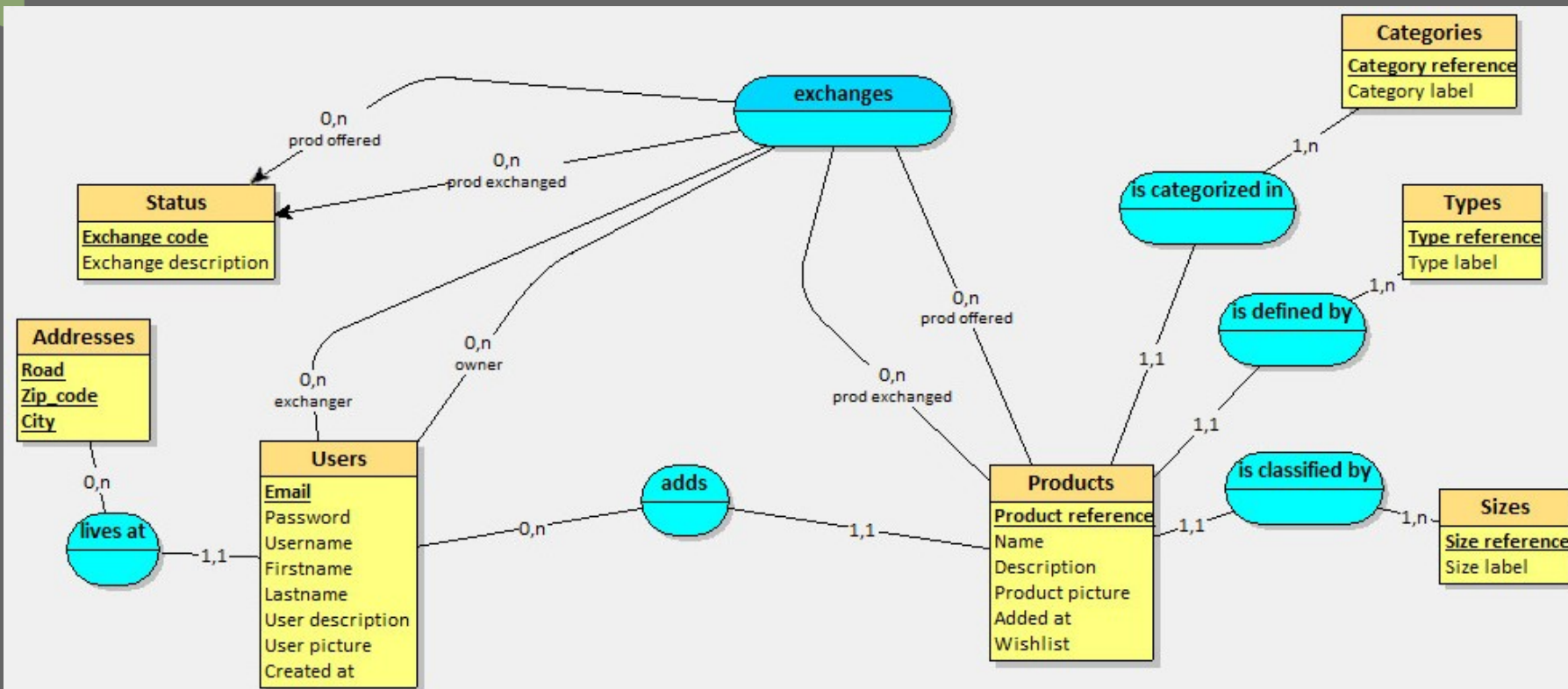
# La maquette statique



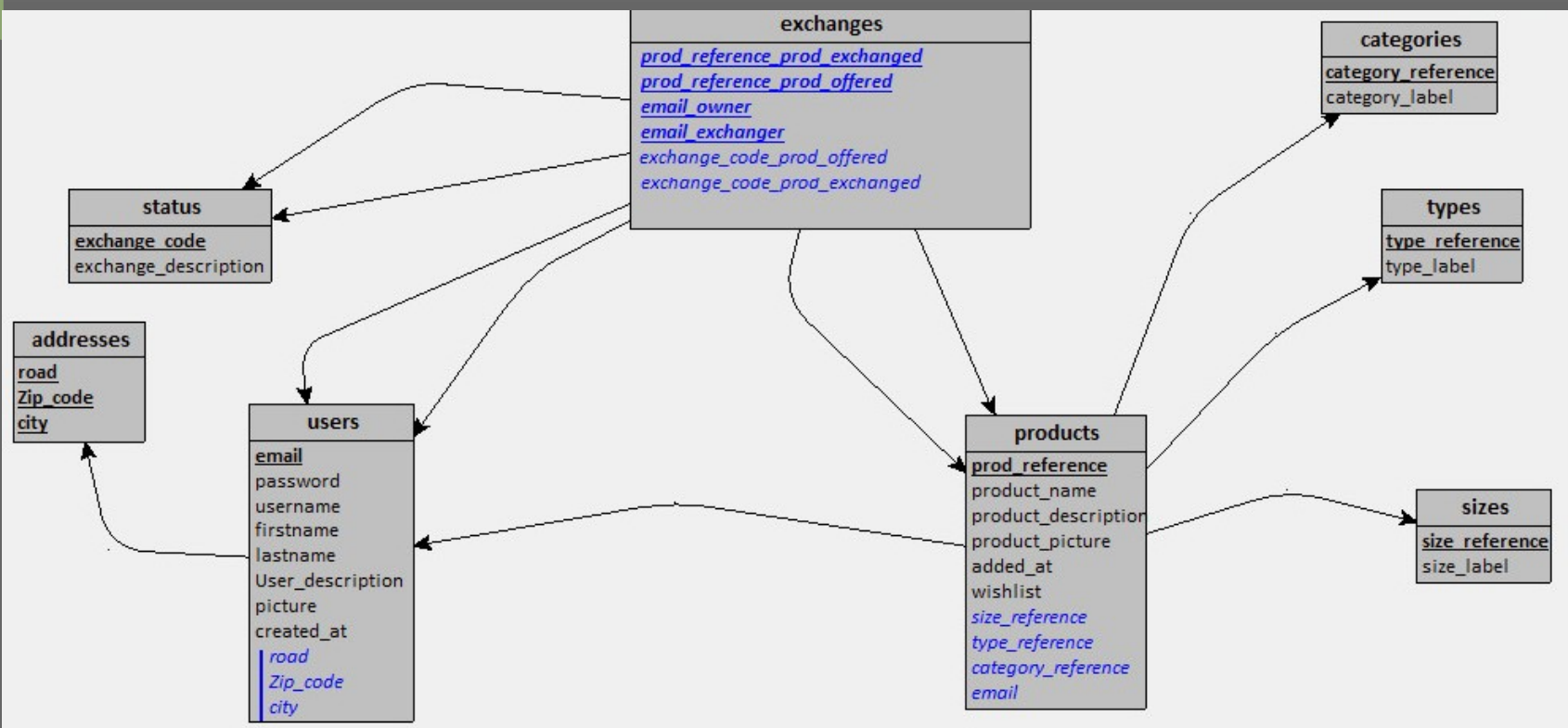


## Partie 3 : les données

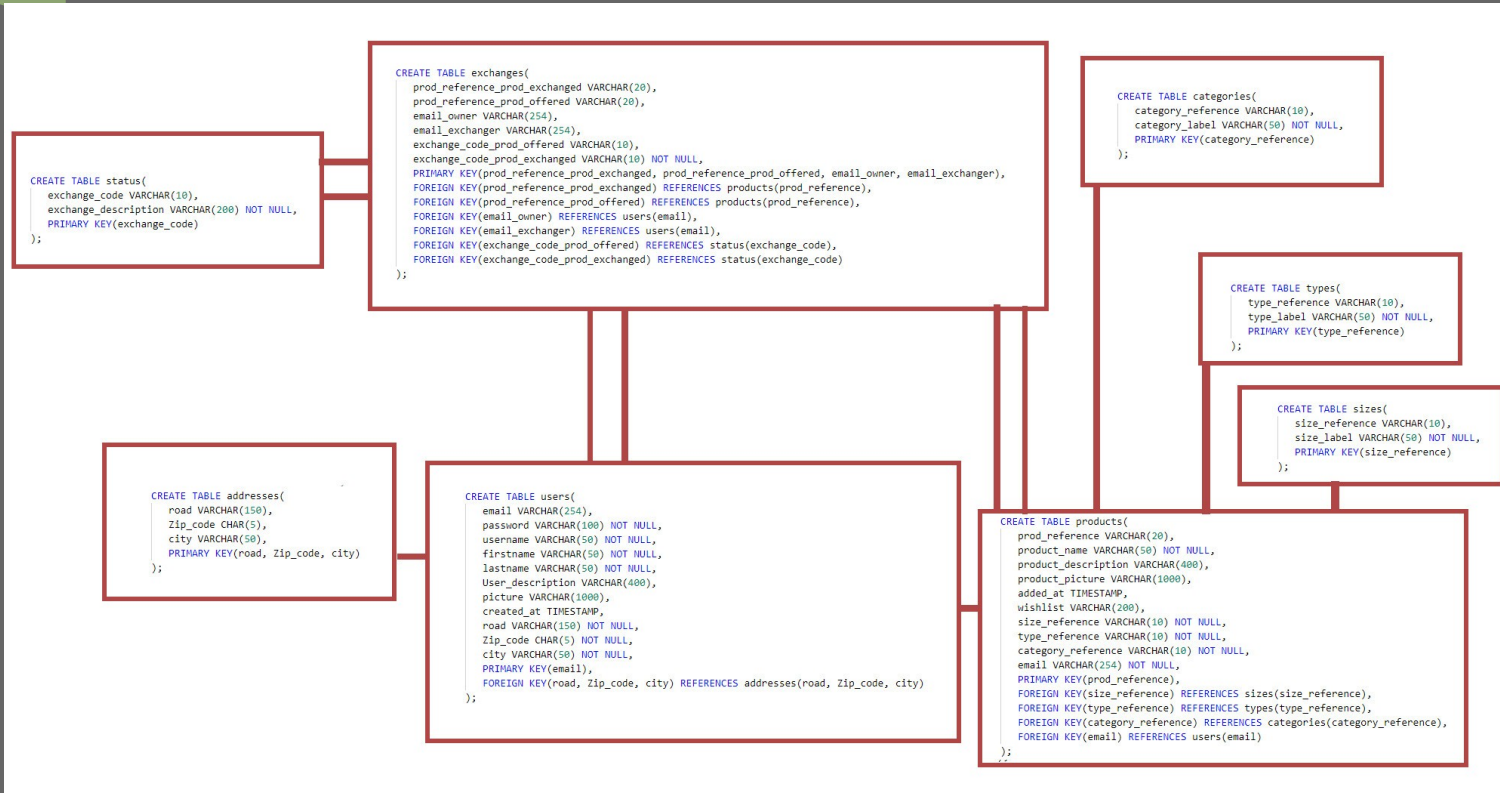
# Le MCD



# Le MLD



# Le MPD

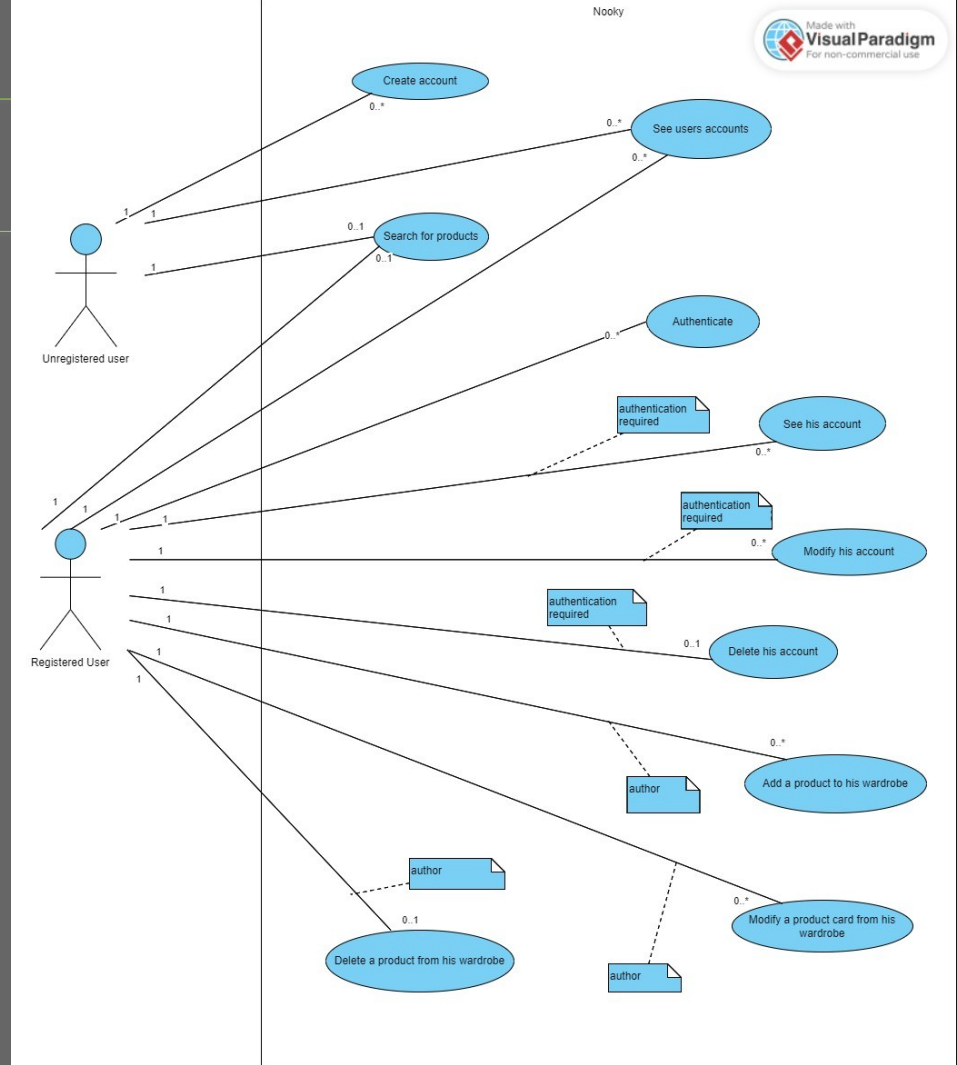




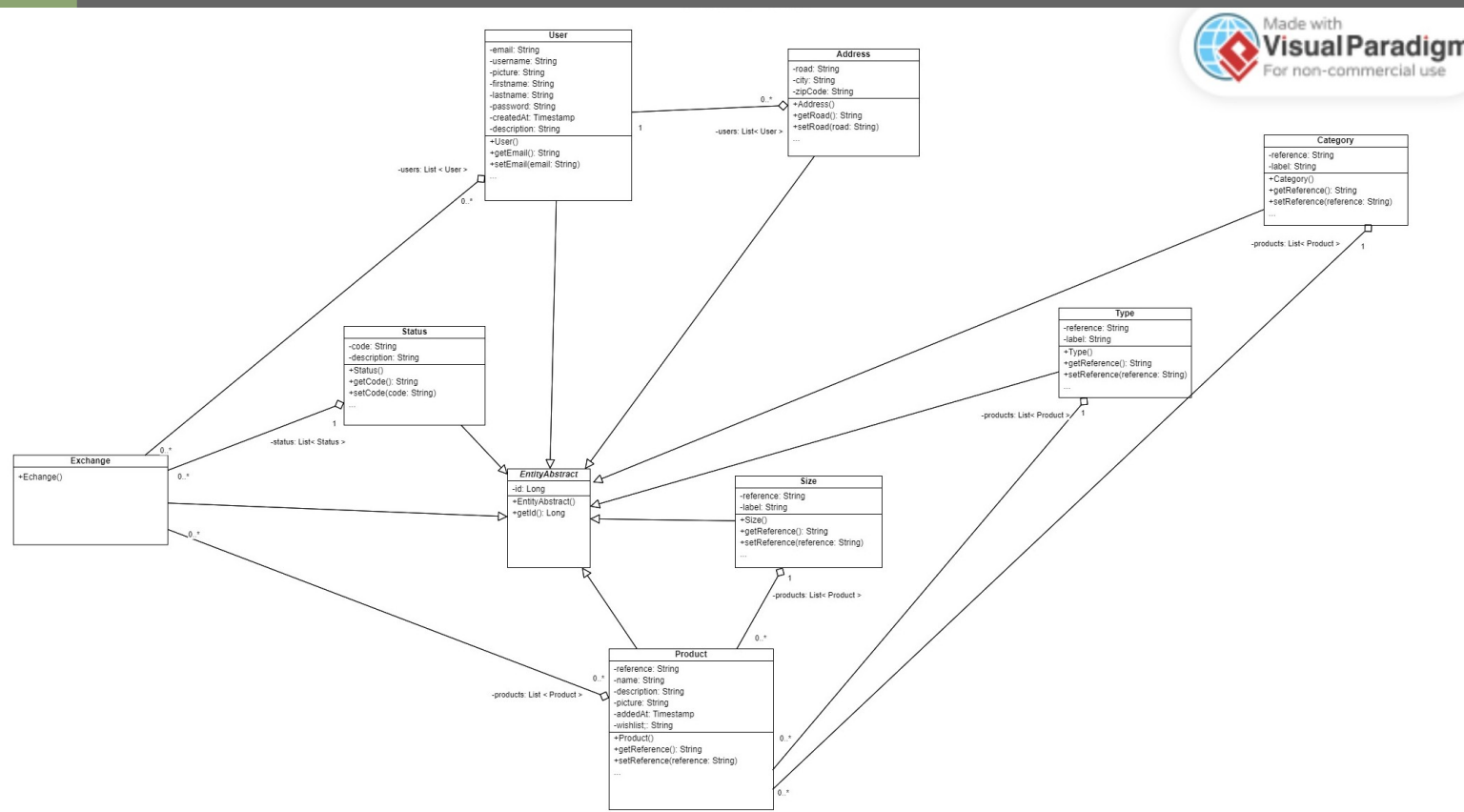


## Partie 4 : les traitements

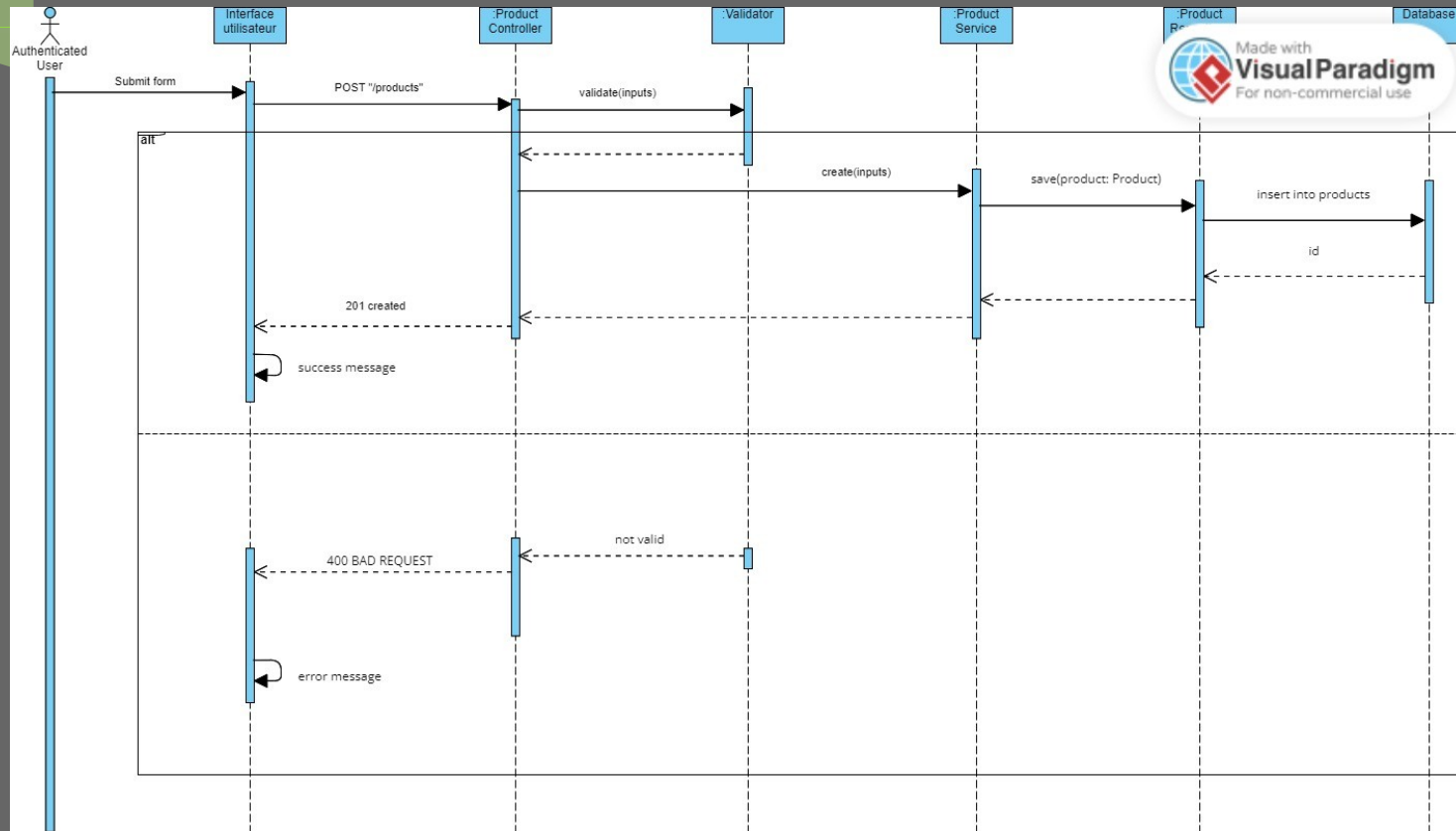
# Use case



# Diagramme de classes



# Diagramme de séquences





## Partie 5 : l'architecture

# Une séparation en trois couches

Une séparation en trois couches :

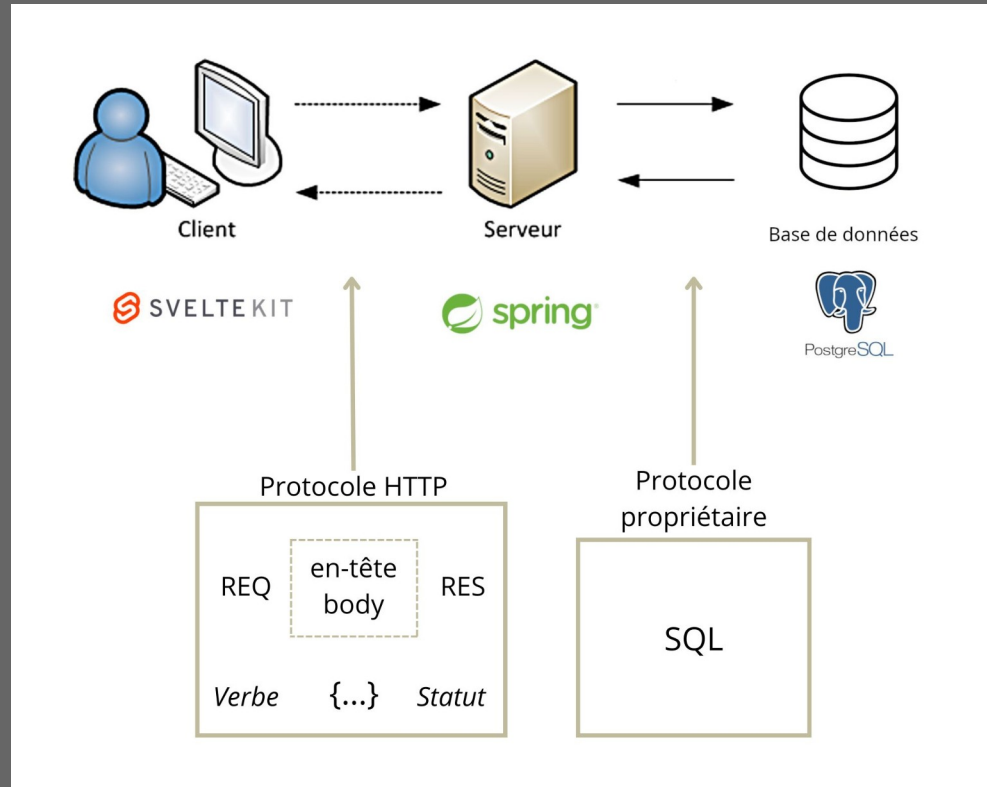
- Données
- Métier
- Client

# Une séparation en trois couches

La stack :

- Données : PostgreSQL
- Métier : Java - Spring
- Client : Sveltekit TS

# Une séparation en trois couches





# Avantages de cette architecture

- Séparation des responsabilités
- Facilité la maintenance et l'évolution du code
- Permet un déploiement couche par couche



## Partie 6 : l'implémentation

# Client – structure du projet

- assets
- components
- lib
- routes

```
▼ src
  > assets
  > components
  > lib
  > routes
  TS app.d.ts
  <> app.html
```

# Client - SPA

## Point d'entrée de mon application – app.html

```
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>
```

# Client – les routes

```
▼ routes
  > account
  > addproduct
  > connexion
  > error
  > modifyaccount
  > modifyproduct
  > myaccount
  > product
  > signin
  > signup
  Ⓢ +page.svelte
```

```
▼ 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
```

# Client – composants graphiques

- ▼ components
  - > Card
  - > Filters
  - > Footer
  - > Header
  - > InformationBloc
  - > ProductsBloc
  - > Searchbar

## Client – lib

✓ lib

> API

> Objects

TS utils.ts

# Client – lib – requêtes à l'API

```
✓ lib
  ✓ API
    > deleteToAPI
    > getFromAPI
    > patchToAPI
    > postToAPI
    > Objects
  TS utils.ts
```



# Client – lib – objects et utils

```
✓ lib
  > API
  ✓ Objects
    TS address.ts
    TS category.ts
    TS product.ts
    TS productCard.ts
    TS size.ts
    TS type.ts
    TS user.ts
    TS utils.ts
```

# Client – fonctionnalité « ajouter un produit »

```
72 <form on:submit={handleSubmit}>
73   <input bind:value={inputOneUser} type="text" placeholder="Entrez le nom du produit">
74   <input bind:value={inputTwoUser} type="text" placeholder="Entrez la description du produit">
75   <input bind:value={inputThreeUser} type="text" placeholder="Entrez l'url de l'image">
76   <input bind:value={inputFourUser} type="text" placeholder="Produits souhaités en échange">
77
78   <select name="category" bind:value={selectedCategory}>
79     {#each categoryList as item (item.id)}
80     <option value={item.id}>{item.label}</option>
81   {/each}
82 </select>
83 <select name="size" bind:value={selectedSize}>
84   {#each sizeList as item (item.id)}
85   <option value={item.id}>{item.label}</option>
86 {/each}
87 </select>
88 <select name="type" bind:value={selectedType}>
89   {#each typeList as item (item.id)}
90   <option value={item.id}>{item.label}</option>
91 {/each}
92 </select>
93 <button class="add" type="submit">Valider</button>
94 <p><a href="/account/{userId}">Retour</a></p>
95 </form>
```

## Client – fonctionnalité « ajouter un produit »

```
38     function handleSubmit(event: Event) {  
39         event.preventDefault();  
40  
41         productData.name = inputOneUser;  
42         productData.description = inputTwoUser;  
43         productData.picture = inputThreeUser;  
44         productData.wishlist = inputFourUser;  
45         productData.userId = userId;  
46         productData.typeId = selectedType;  
47         productData.sizeId = selectedSize;  
48         productData.categoryId = selectedCategory;  
49         postProduct(productData);  
50  
51         goto("/account/"+userId);  
52     }
```

# Client – fonctionnalité « ajouter un produit »

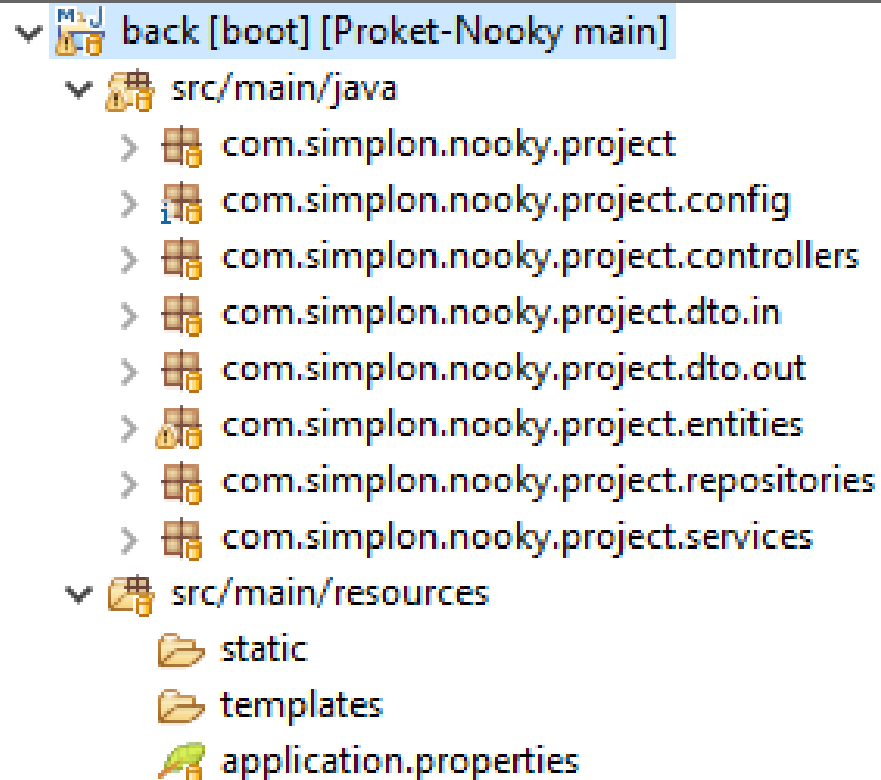
## Fonction postProduct()

```
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 }
```

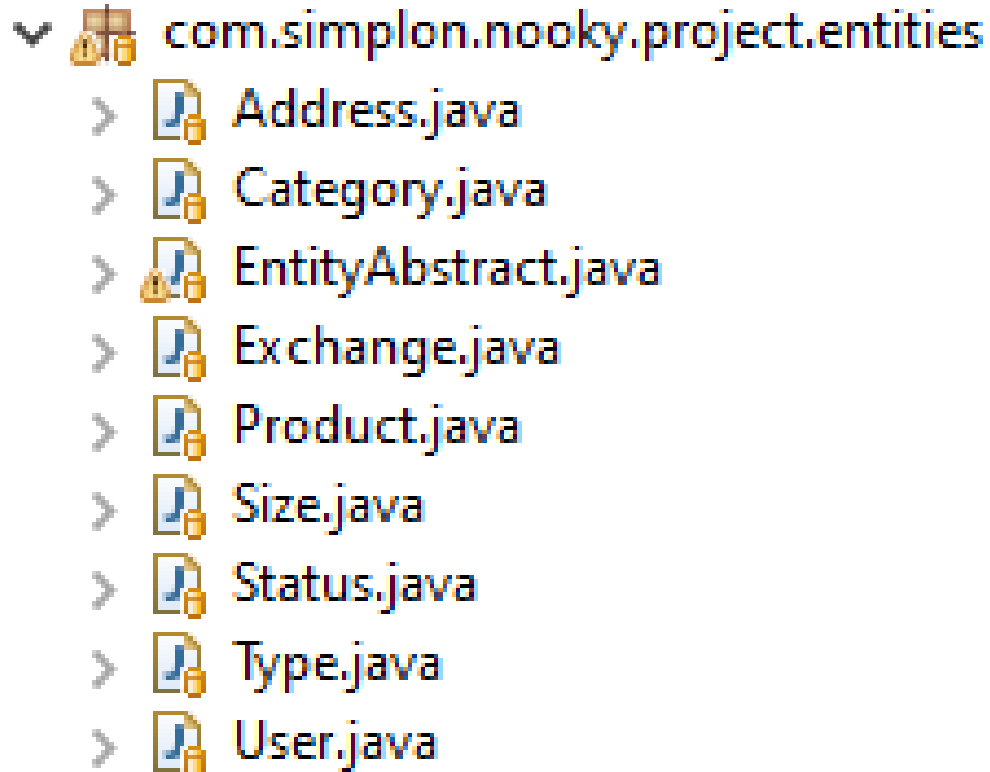
## Extrait de la fonction postData()

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

# Métier – structure du projet










## Métier – entities



```
▼ com.simplon.nooky.project.entities
  > Address.java
  > Category.java
  > EntityAbstract.java
  > Exchange.java
  > Product.java
  > Size.java
  > Status.java
  > Type.java
  > User.java
```

## Métier – DTO in

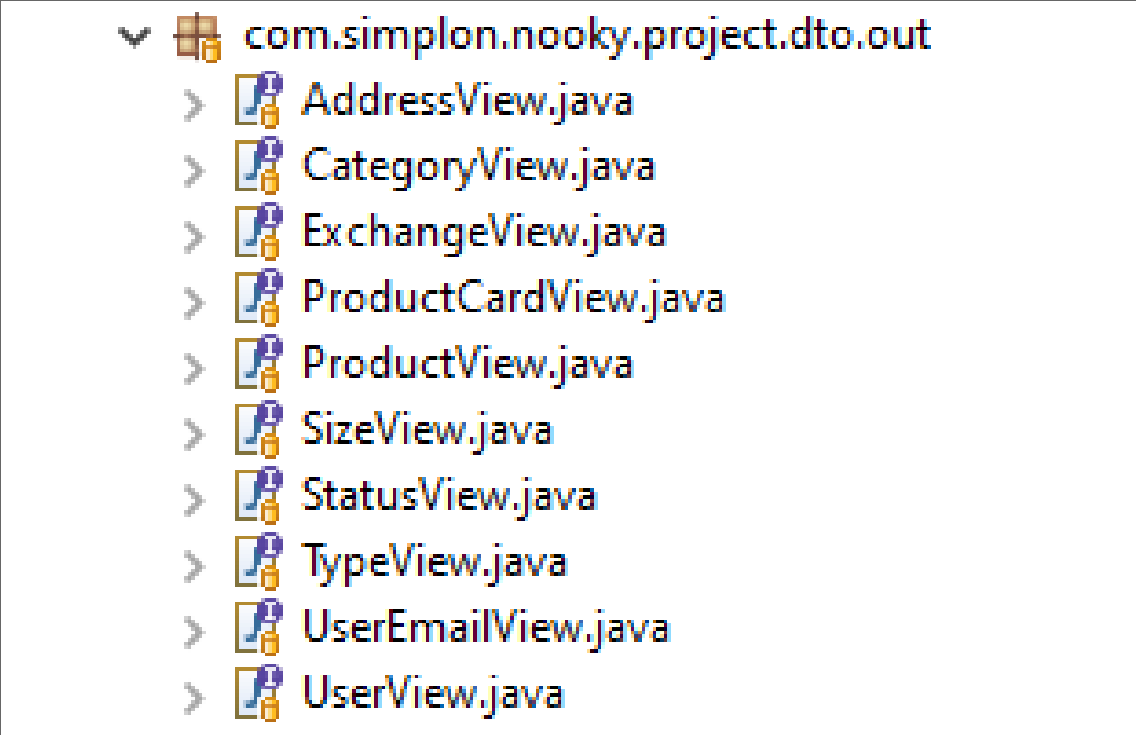
- ▼  com.simplon.nooky.project.dto.in
  - >  AuthUser.java
  - >  CreateExchange.java
  - >  CreateProduct.java
  - >  CreateUser.java
  - >  ModifyProduct.java
  - >  ModifyUser.java

## Métier - DTO CreateProduct.java

```
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 typeId;
```



## Métier – DTO out



A screenshot of an IDE's package explorer showing a package named `com.simplon.nooky.project.dto.out`. The package is expanded, revealing a list of Java files. Each file icon includes a small blue padlock, indicating that the files are locked. The files listed are:

- > `AddressView.java`
- > `CategoryView.java`
- > `ExchangeView.java`
- > `ProductCardView.java`
- > `ProductView.java`
- > `SizeView.java`
- > `StatusView.java`
- > `TypeView.java`
- > `UserEmailView.java`
- > `UserView.java`










## Métier – DTO out – ProductCardView.java

```
3  public interface ProductCardView {  
4      Long getId();  
5      String getName();  
6      String getDescription();  
7      String getPicture();  
8      Long getCategoryId();  
9      Long getTypeId();  
10     Long getSizeId();  
11 }
```

## Métier – DTO out – ProductView.java

```
5  public interface ProductView {  
6      Long getId();  
7      String getName();  
8      String getDescription();  
9      String getCategoryLabel();  
10     String getSizeLabel();  
11     String getTypeLabel();  
12     String getPicture();  
13     Timestamp getAddedAt();  
14     String getWishlist();  
15     Long getUserId();  
16     String getUserUsername();  
17 }
```










# Métiers - controllers

- ▼  com.simplon.nooky.project.controllers
  - >  AddressController.java
  - >  CategoryController.java
  - >  ExchangeController.java
  - >  ProductController.java
  - >  SizeController.java
  - >  StatusController.java
  - >  TypeController.java
  - >  UserController.java

# Métier – ProductController.java

```
1 package com.simplon.nooky.project.controllers;
2
3+ import java.util.List;
21
22 @RestController
23 @RequestMapping("/products")
24 public class ProductController {
25     private final ProductService service;
26
27-     public ProductController(ProductService service) {
28         this.service = service;
29     }
30
31-     @PostMapping
32     @ResponseStatus(HttpStatus.CREATED)
33     public void createProduct(@RequestBody CreateProduct product) {
34         service.createProduct(product);
35     }
```

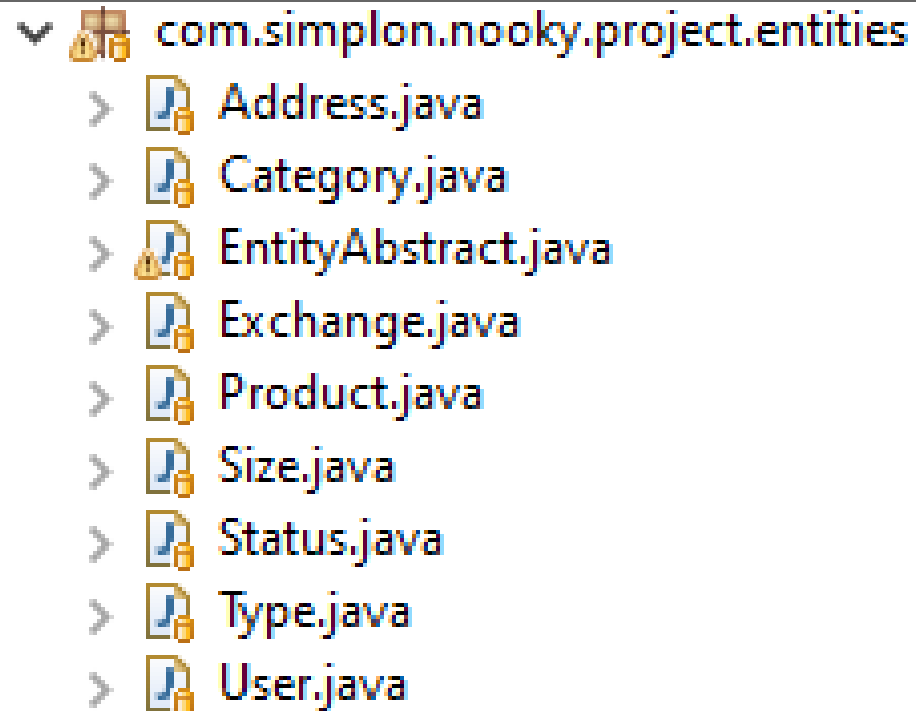
## Métier - services

- ▼  com.simplon.nooky.project.services
  - >  AddressService.java
  - >  CategoryService.java
  - >  ExchangeService.java
  - >  ProductService.java
  - >  SizeService.java
  - >  StatusService.java
  - >  TypeService.java
  - >  UserService.java

# Métier – services - createProduct()

```
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 }  
65
```

# Métier – entities - ORM



```
▼ com.simplon.nooky.project.entities
  > Address.java
  > Category.java
  > EntityAbstract.java
  > Exchange.java
  > Product.java
  > Size.java
  > Status.java
  > Type.java
  > User.java
```

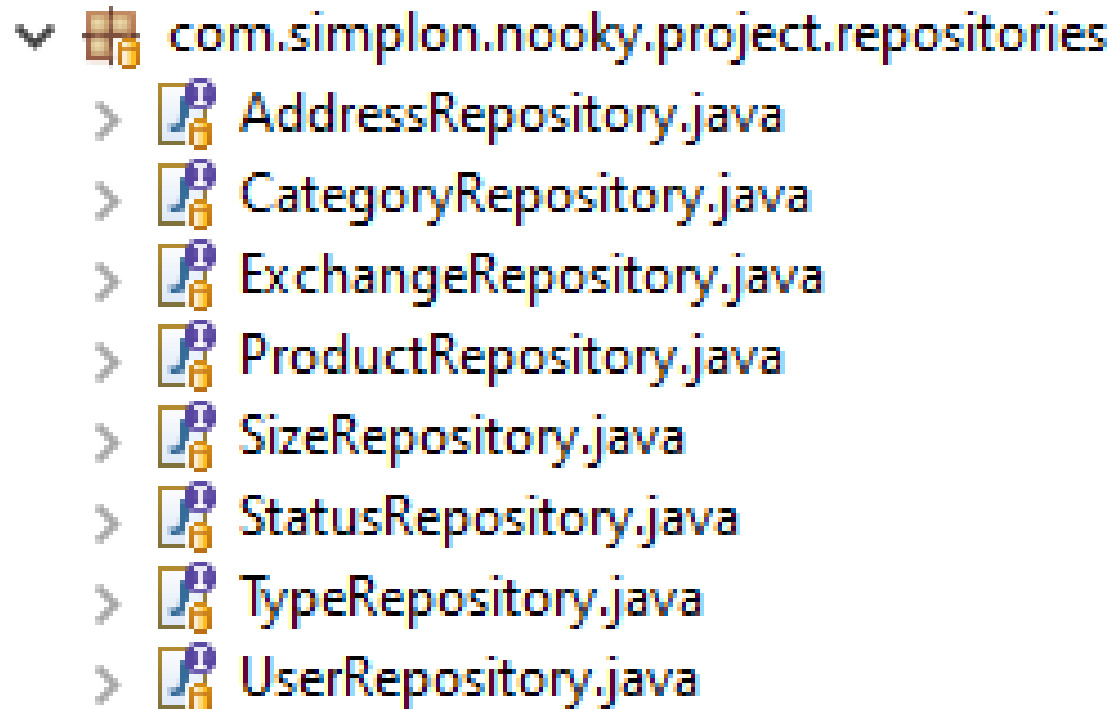


# Métier – ORM – Classe Product

```
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;
```

```
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
56     public void setReference(String reference) {
```

# Métier - repositories



The screenshot shows a file explorer in an IDE. The package `com.simplon.nooky.project.repositories` is expanded, revealing a list of Java files. Each file icon includes a blue 'I' (Interface) and a yellow lock icon, indicating that these are interfaces. The files listed are:

- > `AddressRepository.java`
- > `CategoryRepository.java`
- > `ExchangeRepository.java`
- > `ProductRepository.java`
- > `SizeRepository.java`
- > `StatusRepository.java`
- > `TypeRepository.java`
- > `UserRepository.java`

# Métier - Sécurisation des routes

- Autoriser les requêtes "GET", "POST", "PATCH", "PUT", "DELETE" dans le fichier CorsConfig
- Sécurisation de chaque endpoint en fonction des besoins (permitAll(), anonymous(), fullyAuthenticated()) dans le fichier SpringSecurityConfig

# Métier - JWT




- Génération d'un token au moment de l'authentification
- Ce token est découpé en trois parties : en-tête, charge utile et signature
- Envoi du token vers le client

# Métier - BCrypt

- Lors de l'inscription de l'utilisateur, mot de passe haché avant d'être stocké
- A chaque connexion, le mot de passe envoyé vers le back est haché avant d'être comparé à celui stocké en BDD

# Données – 3 scripts SQL

- Script de création des tables
- Script d'insertion des données de référence
- Script d'insertion des données de tests

 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

# Données – script de création des tables

```
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 );
```

## Données – script d'insertion des données de référence

```
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');
```



# Données – script données de tests

```
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'),
```



# Partie 7 : les tests

# Tests d'intégration back-end

Postman

# Tests d'intégration back-end

The screenshot shows a REST client interface with the following details:

- URL:** localhost:8080/products
- Method:** POST
- Body:** A JSON object representing a product:

```
1 {
2   "name": "Chemise à carreaux",
3   "description": "Chemise à carreaux rouge et bleue, tissu 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 }
```
- Format:** JSON
- Status:** 201 Created
- Time:** 224 ms
- Size:** 387 B

# Tests d'intégration back-end

The screenshot shows a REST client interface with the following details:

- URL:** localhost:8080/products/37
- Method:** GET
- Response Status:** 200 OK
- Response Time:** 81 ms
- Response Size:** 887 B
- Response Format:** JSON
- Response Body (JSON):**

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

# Tests UAT

## Les tests UAT sur Chrome et Mozilla



**Ajouter un produit**

Entrez le nom du produit

Entrez la description du produit

Entrez l'url de l'image

Produits souhaités en échange

▼

▼

▼

Valider

Retour



## Partie 8 : veille sécurité

# Les injections SQL

Pourquoi les injections SQL ?

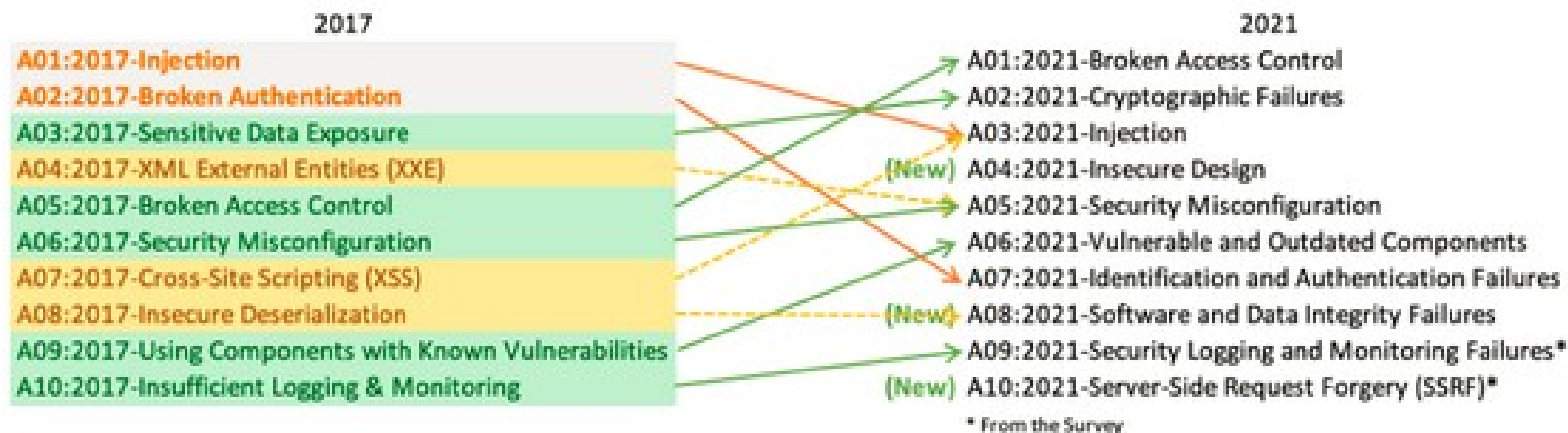
- Fréquent et critique
- Récupération de données sensibles
- Contrôle du serveur



# Les injections SQL

## 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.



# Les types d'injection SQL

- Blind-based
- Error-based
- Union-based
- Stacked-queries

# Adapter son code

## DTO

- Entités non-exposées
- Jakarta Bean Validation

## JPA

- ORM

```
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 typeId;
```

# Adapter son code

## Requêtes paramétrées avec Hibernate :

- Automatisation des requêtes

```
60 product.setType(typeRepository.getReferenceBy  
61 product.setUser(userRepository.getReferenceBy  
62  
63 productRepository.save(product);  
64 }  
65
```

- Paramètres passés séparément

```
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=?
```

# Sources

- 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>



## Partie 9 : démonstration



# Conclusion

# Conclusion

- Ce que m'a apporté ce projet
- Ce que je ferais autrement si je le pouvais
- L'après formation





Merci de votre attention