|  |
| --- |
|  |
| Dossier de projet |
| Application Web Dressing |

|  |
| --- |
| Léa DELANNAY  29/07/2019 |

Table des matières

[I. Liste des compétences du référentiel couvertes par le projet 3](#_Toc14035636)

[A. Développer la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité 3](#_Toc14035637)

[B. Développer la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité 3](#_Toc14035638)

[II. Résumé du projet 3](#_Toc14035639)

[III. Cahier des charges 4](#_Toc14035640)

[IV. User Stories et spécifications techniques 4](#_Toc14035641)

[V. Base de données 4](#_Toc14035642)

[A. Modélisation 4](#_Toc14035643)

[1. Modèle Conceptuel des Données 4](#_Toc14035644)

[2. Modèle Logique des Données 4](#_Toc14035645)

[B. Code SQL 4](#_Toc14035646)

[C. Mise en place 5](#_Toc14035647)

[VI. Réalisations 6](#_Toc14035648)

[A. Création d’un compte 6](#_Toc14035649)

[B. Connexion à l’application 8](#_Toc14035650)

[C. Création d’un vêtement 11](#_Toc14035651)

[D. Modification d’un vêtement 19](#_Toc14035652)

[E. Visualisation et filtres 22](#_Toc14035653)

[F. Suppression d’un vêtement 22](#_Toc14035654)

[VII. Jeu d’essai de la fonctionnalité la plus représentative 22](#_Toc14035655)

[VIII. Veille sur les vulnérabilités de sécurité 28](#_Toc14035656)

[IX. Recherche sur un site anglophone 30](#_Toc14035657)

[X. Extrait du site anglophone et traduction en français. 33](#_Toc14035658)

# Liste des compétences du référentiel couvertes par le projet

Les deux activités types du référentiel sont :

* Développer la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité
* Développer la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité

## Développer la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité

Les compétences couvertes sur cette partie sont :

* Maquetter une application
* Réaliser une interface utilisateur web statique et adaptable
* Développer une interface utilisateur web dynamique

## Développer la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité

Les compétences couvertes sur cette partie sont :

* Créer une base de données
* Développer les composants d’accès aux données
* Développer la partie back-end d’une application web ou web mobile

# Résumé du projet

Dans le cadre de ma préparation du titre de Développeur Web et Web Mobile de niveau III à l’IFPA de Mérignac, j’ai réalisé une application personnelle que je vais vous présenter.

Il s’agit d’une application de gestion de vêtements très originalement nommée « MonDressing ».

Le but est, dans une première version, de proposer à un utilisateur d’avoir de la visibilité sur tous les vêtements qu’il aura enregistré dans l’application, et de lui permettre les gérer.

Dans sa seconde version, l’application permettra de générer aléatoirement des tenues en fonction de certains critères saisis par l’utilisateur.

L’utilisateur peut créer un compte, se connecter, créer un vêtement, visualiser la liste de ses vêtements, visualiser le détail d’un vêtement, le modifier, le supprimer, et afficher la météo (module externe).

Lors de la création du vêtement, l’utilisateur saisit toutes les caractéristiques, et a la possibilité d’ajouter une image.

Lors de la visualisation, cette image apparaît accompagnée du nom du vêtement.

Des filtres ont été mis en place sur la page de visualisation de tous les vêtements afin de les trier selon divers critères.

Pour développer cette application, j’ai utilisé les frameworks Angular 7 et Bootstrap 4 pour le front-end, NodeJs avec le framework Express pour le back-end, et le système de gestion de base de données MySQL.

J’ai utilisé l’IDE Visual Studio Code et je me suis appuyée sur Xampp comme plate-forme locale de développement.

L’application comporte actuellement 7 pages et deux modales, et est responsive afin s’adapter aux différentes tailles d’écrans (téléphone, tablette, ordinateur…).

# Cahier des charges

Vous trouverez le cahier des charges en annexe A01.

# User Stories et spécifications techniques

Vous trouverez les user stories et spécifications techniques en annexe A02.

Vous trouverez également le dictionnaire des données en annexe A02.1

# Base de données

## Modélisation

Dans un premier temps, j’ai effectué une étape de réflexion et d’analyse des processus de l’application souhaitée sur papier avant de reporter le schéma dans Power AMC.

J’ai identifié les objets et les relations entre les objets.

J’ai fait le choix de détailler dans des tables distinctes les caractéristiques des vêtements car l’application sera amenée à évoluer.

### Modèle Conceptuel des Données

J’ai réalisé le MCD avec le logiciel Power AMC à l’aide de la méthode MERISE apprise en cours. Vous le trouverez en annexe A03.1.

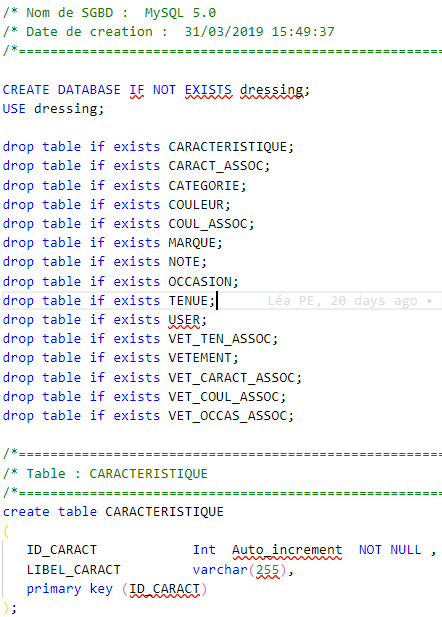
### Modèle Logique des Données

J’ai réalisé le MLD avec le logiciel Power AMC à l’aide de la méthode MERISE apprise en cours. Vous le trouverez en annexe A03.2.

## Code SQL

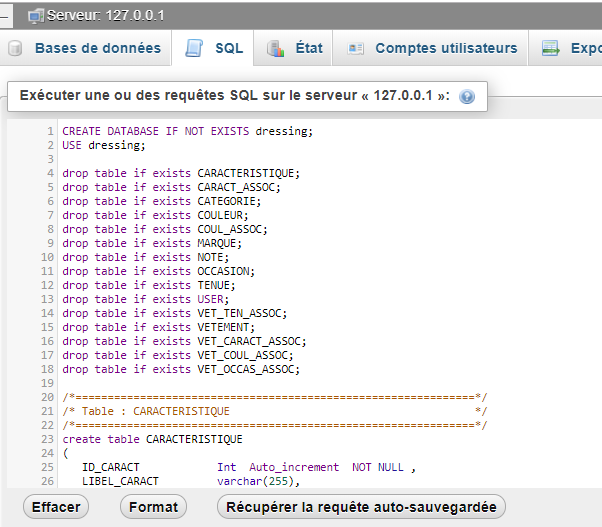
J’ai ensuite généré le code SQL avec Power AMC, que j’ai retravaillé pour plus de lisibilité, notamment afin que les noms des clés étrangères ne soient pas les mêmes que ceux des clés primaires.

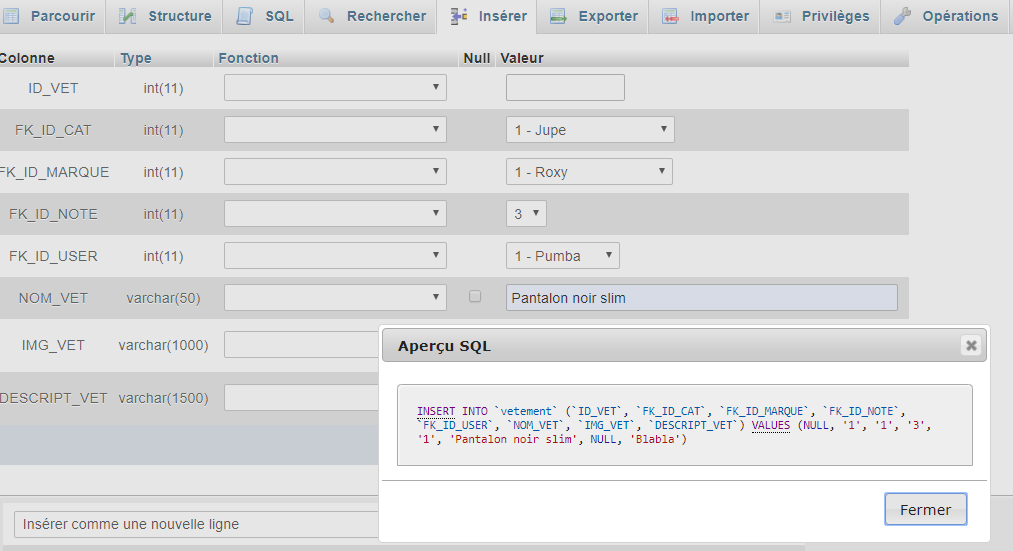
Pour des questions de re jouabilité, j’ai ajouté en début de fichier les instructions SQL permettant de créer la base de données si cette dernière n’existe pas, d’utiliser cette base de données. Le script SQL permet aussi de supprimer les tables si elles existent avant de les créer afin d’éviter d’éventuelles incohérences dans les données.



## Mise en place

J’ai enfin intégré le code SQL dans l’onglet SQL de phpMyAdmin, et j’ai ajouté quelques jeux de données aux tables afin de pouvoir tester l’application au fur et à mesure de son développement.

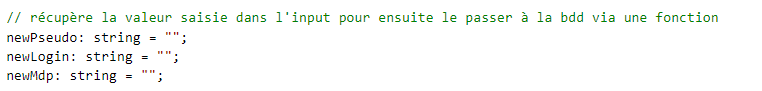




# Réalisations

## Création d’un compte

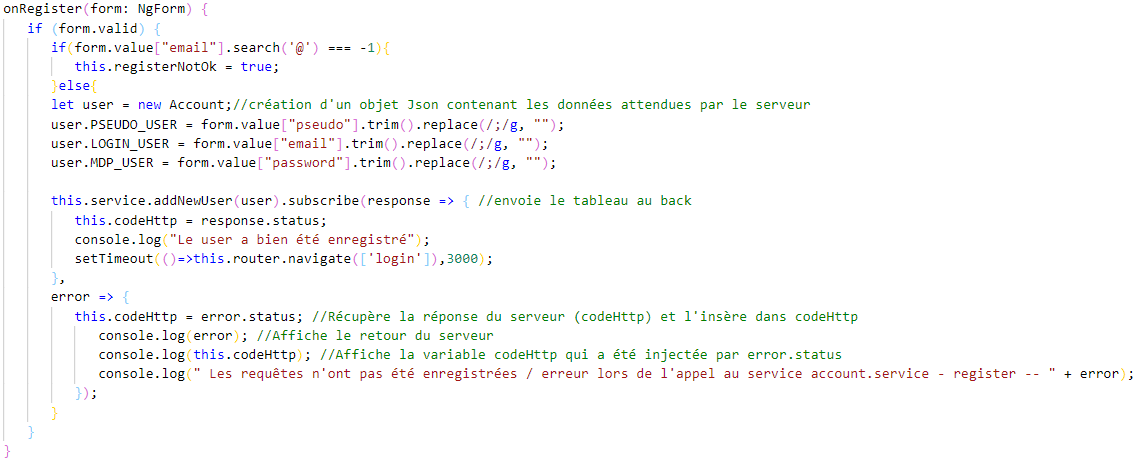
Pour créer un compte, je récupère les données saisies par l’utilisateur dans le formulaire au niveau du composant Angular « register ».



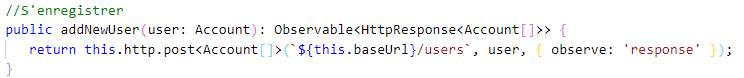
La récupération des informations est faite grâce à la directive Angular « ngModel » qui permet de faire du binding bidirectionnel des données.



Au clic sur le bouton « S’enregistrer » du formulaire, la fonction onRegister définie dans le fichier typescript du composant se lance :

* Je vérifie que le champ email contient bien un « @ »
* Je crée un objet Json dans lequel j’ajoute les données récupérées du formulaire et attendues par le serveur soit le pseudo, le login et le mot de passe
* Je supprime les espaces avant et après ainsi que les points-virgules, sur les valeurs saisies dans les champs concernant le pseudo, le login et le mot de passe
* Je fais l’appel au service en lui passant l’objet Json créé, et je souscris au retour de l’observable contenu dans le service
* Lors du retour du service, je récupère le code http retourné par le serveur pour afficher un message dans le html et je renvoie l’utilisateur vers le formulaire de connexion

Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en post car il s’agit d’une création et que j’applique autant que possible les principes REST(c’est-à-dire que je considère les éléments sur mon serveur comme des ressources).

Afin que les données qui transitent entre le front et le serveur soient chiffrées, il faut installer sur le serveur le certificat https permettant d’utiliser le protocole https, puis le lier au site web.

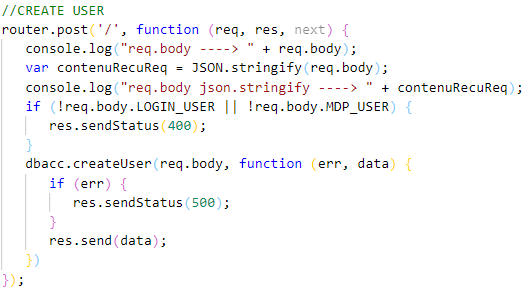
Je mets en place la route qui crée le point d’entrée sur le serveur.



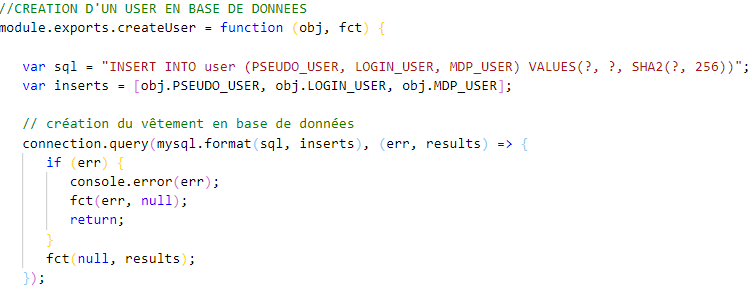


Les données du front sont donc envoyées à la ressource « users » sur le serveur, qui joue la fonction correspondant à la requête http faite par le front. Dans ce cas, la fonction jouée est router.post(‘/’, …). Celle-ci vérifie que les données attendues sont bien présentes dans l’objet Json transmis par le front (sinon renvoie un code http 400 « Bad Request »), puis fait appel à une autre fonction qui gère l’appel à la base de données.

Le résultat de la requête effectuée dans l’autre fonction est récupéré et traité par la fonction dbacc.createUser et un code http 500 « Internal Server Error » ou 200 « OK » est renvoyé au front.



La fonction dans index.js appelée par la ressource « users » permet de faire la requête SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une, à la fonction ci-dessus contenue dans la ressource « users ». Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

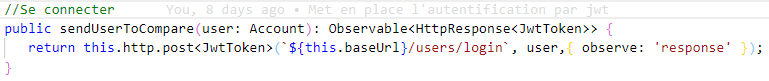
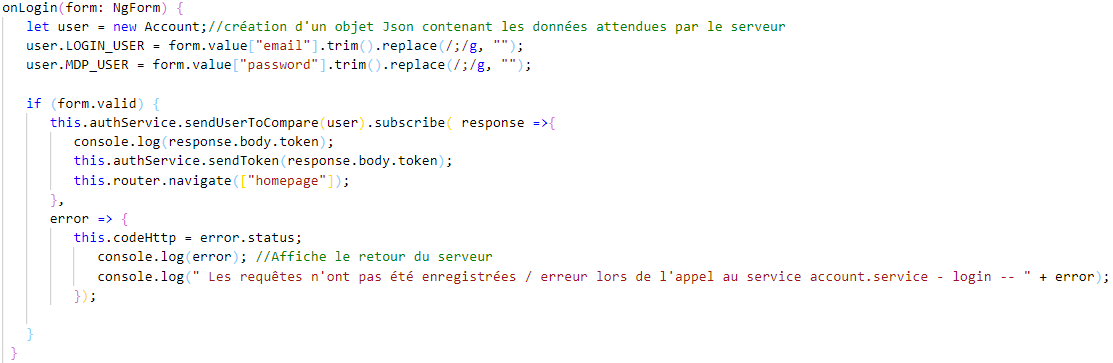
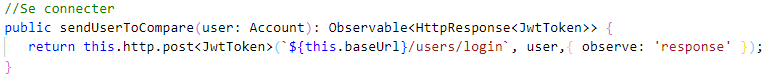


## Connexion à l’application

Pour connecter l’utilisateur à l’application, je récupère les données saisies par l’utilisateur dans le formulaire au niveau du composant Angular « login », grâce à la directive Angular « ngModel », comme dans le formulaire précédent « register ».

Au clic sur le bouton « Se connecter » du formulaire, la fonction onLogin définie dans le fichier typescript du composant se lance :

* Je crée un objet Json dans lequel j’ajoute les données récupérées du formulaire et attendues par le serveur soit le login et le mot de passe
* Je supprime les espaces avant et après ainsi que les points-virgules, sur les valeurs saisies dans les champs concernant le login et le mot de passe
* Je fais l’appel au service en lui passant l’objet Json créé, et je m’abonne au retour du service
* Lors du retour du service, je récupère le token généré par le back et je l’envoie dans le local storage ou il sera stocké jusqu’à ce que l’utilisateur se déconnecte
* Je récupère le code http retourné par le serveur pour afficher un message d’erreur (s’il y en a une) dans le html et je renvoie l’utilisateur vers la page d’accueil

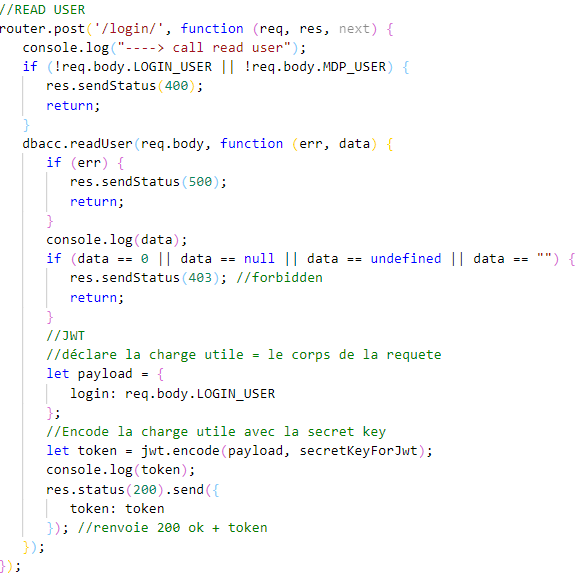
Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en post car je dois envoyer les informations au serveur pour qu’il puisse les comparer aux informations présentes en base de données.

Les données du front sont envoyées à la ressource « users » sur le serveur, qui joue la fonction correspondant à la requête http faite par le front. Dans ce cas, la fonction jouée est router.post(‘/login/’…). Celle-ci vérifie que les données attendues sont bien présentes dans l’objet Json transmis par le front (sinon renvoie un code http 400 « Bad Request »), puis fait appel à une autre fonction qui gère l’appel à la base de données.

Le résultat de la requête effectuée dans l’autre fonction est récupéré et traité par la fonction dbacc.createUser et un code http 500 « Internal Server Error » ou 200 « OK » est renvoyé au front.

De plus, l’id de l’utilisateur est récupéré et encodé sous la forme d’un token Json Web Token à l’aide d’une librairie NodeJs (JWT).

Ce token est transmis au front en plus du code http 200 « OK ».



La fonction dans index.js appelée par la ressource « users » permet de faire les requêtes SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une.

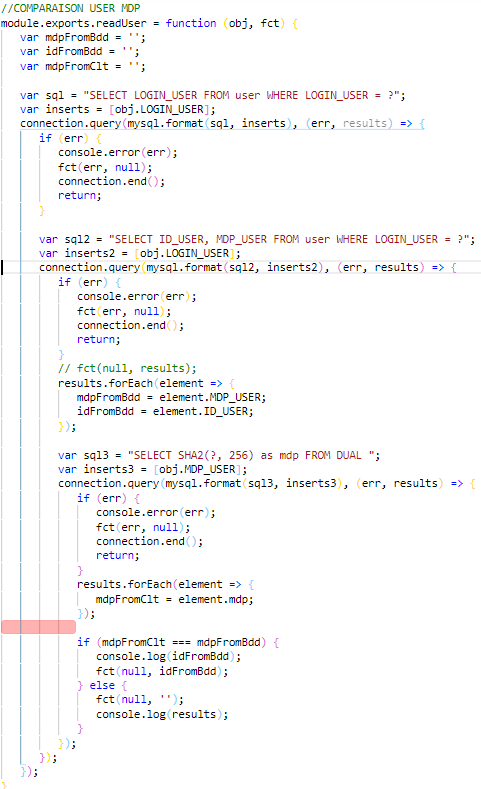
La première requête vérifie que le login saisi par l’utilisateur existe bien en base de données.

Le seconde requête permet de récupérer depuis la base de données l’id et le mot de passe de l’utilisateur correspondant au login envoyé. L’id est récupéré pour être encrypté et le mot de passe pour être comparé.

Les mots de passes sont stockés hashé, c’est à dire qu’ils ne sont pas en clair et qu’il est impossible de le retrouver à partir de ce qu’il y a en base. On ne fait que hasher de la même manière ce que l’utilisateur saisit quand il s’enregistre et ce qu’il saisit quand il se connecte, et si les hash sont identiques, c’est que le mot de passe tapé par l’utilisateur est le bon.

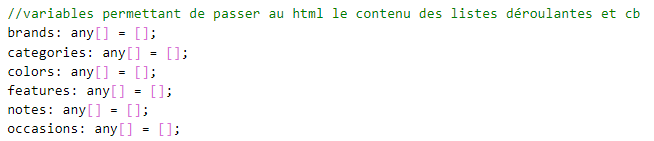
La troisième requête permet de comparer le mot de passe transmis par le front et hashé, au mot de passe enregistré en base de données. S’il n’y a pas d’erreur, alors je renvoie l’id à la fonction précédente pour qu’il soit utilisé pour créer le token JWT, sinon, je renvoie vide (‘’).

Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

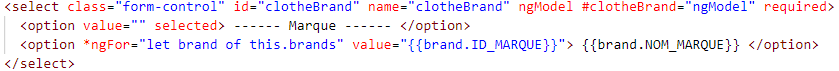


## Création d’un vêtement

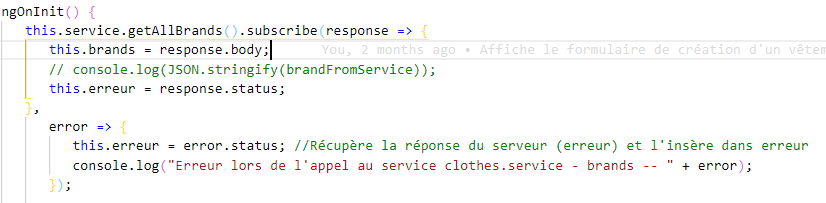
Au chargement du formulaire de création d’un vêtement, j’initialise les variables qui vont me permettre de transmettre au html le contenu des listes déroulantes et des cases à cocher.

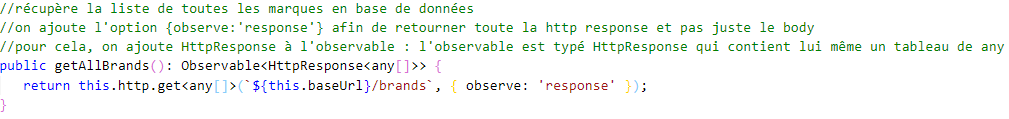


Je passe les données au front grâce aux directives structurelles Angular et au binding bidirectionnel.

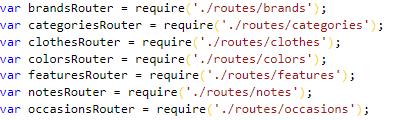


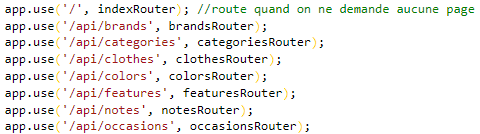
J’initialise également les appels au service permettant de récupérer la liste des marques, catégories, couleurs, caractéristiques, notes et occasions, et lorsque j’ai un retour du service, j’ajoute le contenu à la variable brands précédemment initialisée.



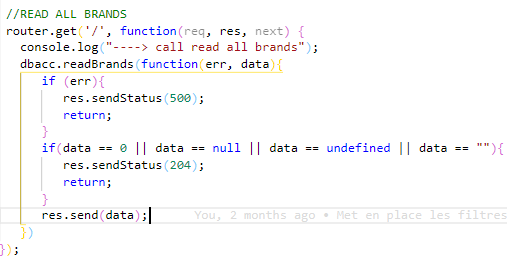
Le service fait appel aux ressources mises à disposition par le serveur. La requête http est en get car il s’agit d’une récupération de données du point de vue REST.

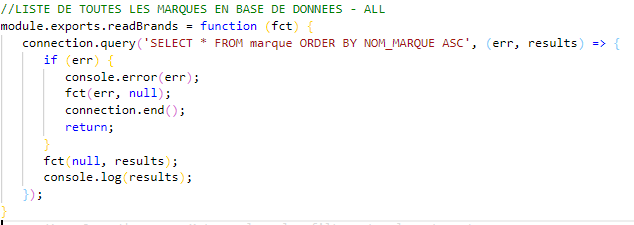
Je mets en place les routes qui créent les points d’entrée sur le serveur.



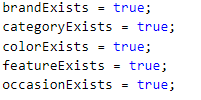


La requête http faite en get depuis le front vient chercher les informations de la ressource concernée (brands.js, categories.js, features.js, colors.js, notes.js, occasions.js). La ressource fait appel à une fonction qui gère l’appel à la base de données (dans index.js) et retourne la liste des éléments demandés. Voici l’exemple pour la liste des marques :



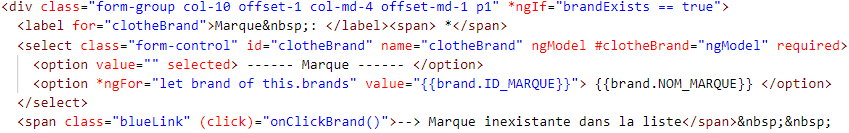
Cette fonction située dans index.js appelée par la ressource permet de faire la requête SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une, à la fonction ci-dessus contenue dans la ressource. Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

Pour que l’utilisateur puisse ajouter une marque, couleur, catégorie, caractéristique ou occasion qui n’existe pas en base, j’initialise des variables de type booléen.

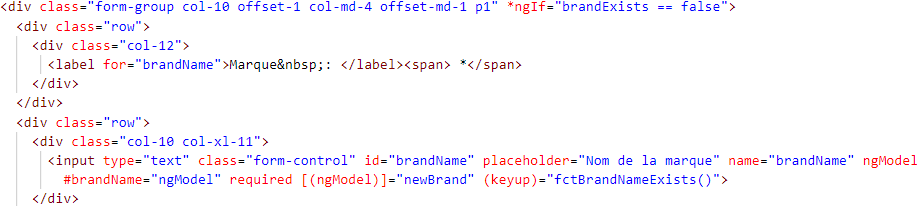


J’utilise ensuite cette variable pour afficher un contenu différent si elle est true ou false.

Si elle est « true », j’affiche la liste déroulante.



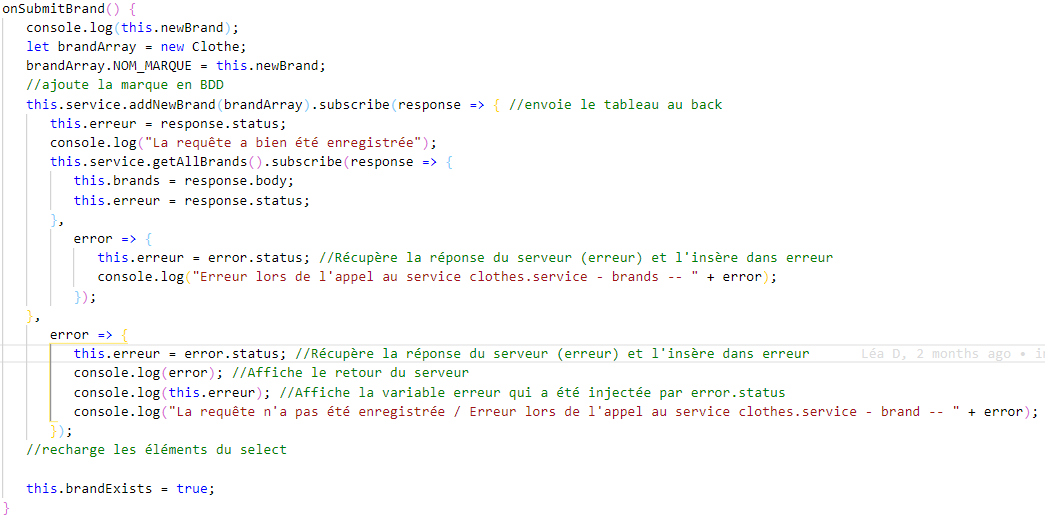
Si elle est « false », j’affiche un input de type texte permettant de renseigner un nouvel élément.

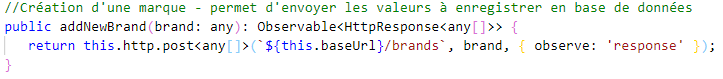
Le changement d’état est géré au clic sur le lien « Marque inexistante dans la liste ».



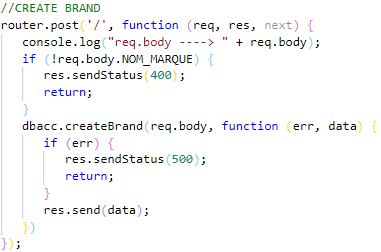
Lorsque l’utilisateur soumet l’ajout du nom de la marque, couleur, catégorie, caractéristique ou occasion qui n’existe pas en base, cela lance une fonction (dans cet exemple la fonction on SubmitBrand() )qui :

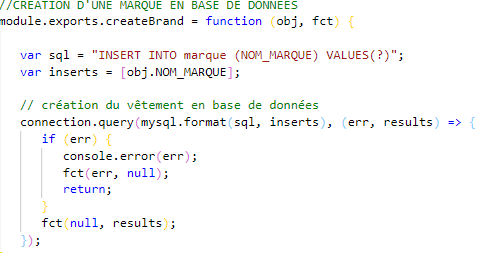
* Initialise un objet Json dans lequel j’ajoute le contenu du champ texte saisi par l’utilisateur dans le formulaire.
* Fait l’appel au service en lui passant l’objet Json créé, et souscrit au retour de l’observable contenu dans le service
* Lors du retour du service, rappelle la fonction permettant de charger la liste de toutes les marques (dans cet exemple) afin de la mettre à jour avec l’élément qui vient d’être enregistré

Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en post car il s’agit d’une création.



Le fonctionnement de la ressource et du requêtage sur le serveur est le même que pour les cas précédents. La ressource appelle une fonction qui fait la requête en base puis retourne un résultat, et renvoie un code http au front.



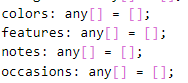


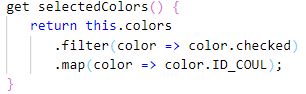
Pour que l’utilisateur puisse ajouter un vêtement, je récupère les données saisies par l’utilisateur dans le formulaire au niveau du composant Angular « clothe-create ».

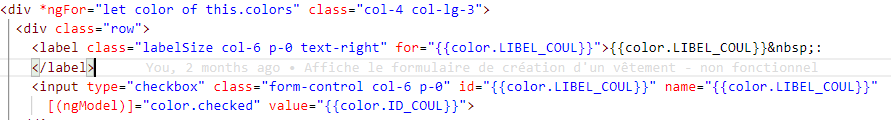
Pour cela, je récupère les informations grâce à l’attribut name des champs du formulaire pour les données simples.



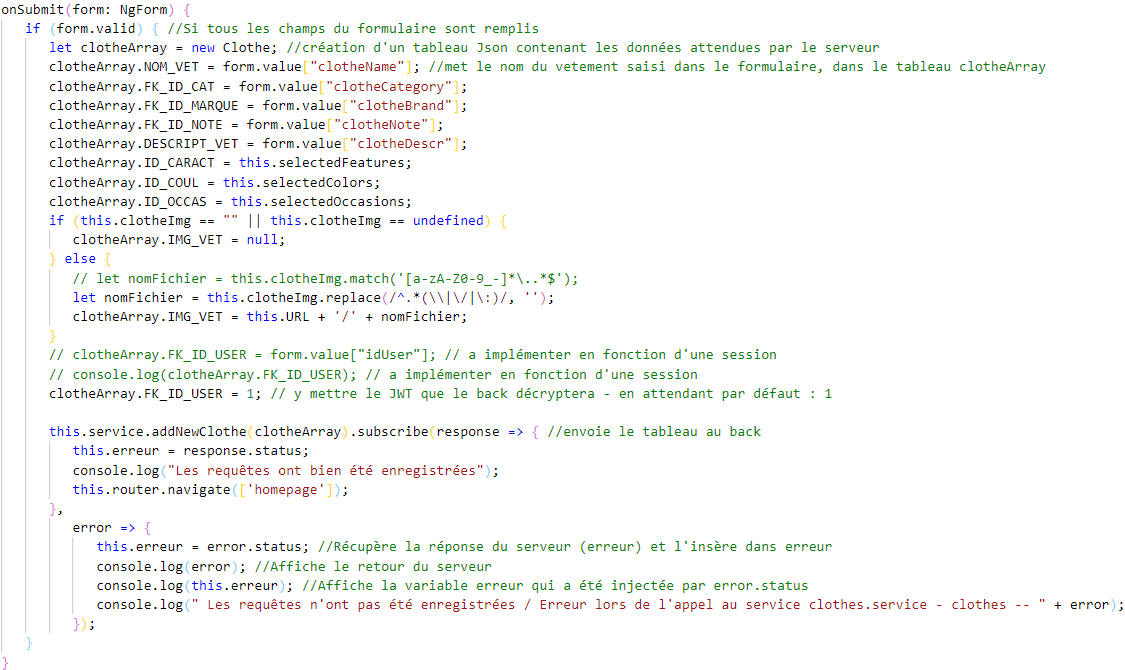
Concernant les données multiples issues des cases à cocher, je les récupère grâce à l’initialisation de variables et à une fonction permettant de trier les informations récupérées.

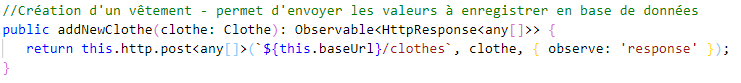




Je lie cela au html grâce à la directive structurelle Angular ngFor et grâce à ngModel.

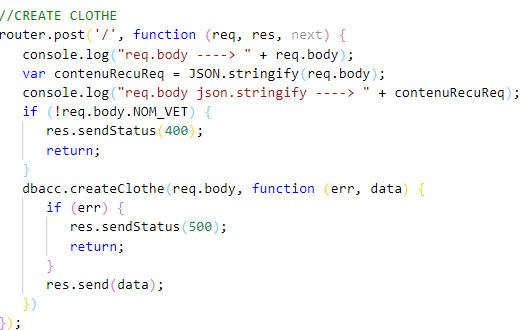
Au clic sur le bouton « Enregistrer » du formulaire, la fonction onSubmit définie dans le fichier typescript du component se lance :

* Je crée un objet Json dans lequel j’ajoute les données récupérées du formulaire et attendues par le serveur
* Je vérifie que le champ concernant l’image n’est pas vide. S’il est vide ou indéfini, j’ajoute la valeur null afin qu’une image par défaut soit affichée, sinon, j’ajoute le chemin vers l’image.
* Je fais l’appel au service en lui passant l’objet Json créé, et je souscris au retour de l’observable contenu dans le service
* Lors du retour du service, je récupère le code http retourné par le serveur pour afficher un message dans le html et je renvoie l’utilisateur vers la page d’accueil

Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en post car il s’agit d’une création et que j’applique autant que possible les principes REST(c’est-à-dire que je considère les éléments sur mon serveur comme des ressources et m’appuie sur le protocole http pour interagir avec lui).

Les données du front sont donc envoyées à la ressource « clothes » sur le serveur, qui joue la fonction correspondant à la requête http faite par le front. Dans ce cas, la fonction jouée est router.post(‘/’, …). Celle-ci vérifie que les données attendues sont bien présentes dans l’objet Json transmis par le front (sinon renvoie un code http 400 « Bad Request »), puis fait appel à une autre fonction qui gère l’appel à la base de données.

Le résultat de la requête effectuée dans l’autre fonction est récupéré et traité par la fonction dbacc.createClothe et un code http 500 « Internal Server Error » ou 200 « OK » est renvoyé au front.



La fonction dans index.js appelée par la ressource « clothes » permet de faire la requête SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une à la fonction ci-dessus contenue dans la ressource « clothes ».

La première requête insère les éléments dans la table *vetement* puis récupère l’id du vêtement créé.

Les trois requêtes suivantes permettent d’insérer les éléments dans les tables associatives *vet\_caract\_assoc*, *vet\_coul\_assoc* et *vet\_occas\_assoc*. Pour cela, elles utilisent l’id du vêtement créé à l’étape précédente.

Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

//CREATION D'UN VETEMENT EN BASE DE DONNEES

module.exports.createClothe = function (obj, fct) {

var idVet;

var sql1 = "INSERT INTO vetement (FK\_ID\_CAT, FK\_ID\_MARQUE, FK\_ID\_NOTE, FK\_ID\_USER, NOM\_VET, IMG\_VET, DESCRIPT\_VET) VALUES(?, ?, ?, ?, ?, ?, ?)";

var inserts1 = [obj.FK\_ID\_CAT, obj.FK\_ID\_MARQUE, obj.FK\_ID\_NOTE, obj.FK\_ID\_USER, obj.NOM\_VET, obj.IMG\_VET, obj.DESCRIPT\_VET];

// création du vêtement en base de données

connection.query(mysql.format(sql1, inserts1), (err, results) => {

if (err) {

console.error(err);

fct(err, null);

return;

}

//récupération de l'id du vêtement créé

idVet = results.insertId;

//création des associations

var featureArray = obj.ID\_CARACT;

featureArray.forEach(function (item) {

var sql2 = "INSERT INTO vet\_caract\_assoc (ID\_VET, ID\_CARACT) VALUES(" + idVet + ", ?)";

var inserts2 = [item];

console.log("CARACTERISTIQUE" + inserts2);

connection.query(mysql.format(sql2, inserts2), (err) => {

if (err) {

console.error(err);

fct(err, null);

return;

}

})

});

var colorArray = obj.ID\_COUL;

colorArray.forEach(function (item) {

var sql3 = "INSERT INTO vet\_coul\_assoc (ID\_VET, ID\_COUL) VALUES(" + idVet + ", ?)";

var inserts3 = [item];

console.log("COULEUR" + inserts3);

connection.query(mysql.format(sql3, inserts3), (err) => {

if (err) {

console.error(err);

fct(err, null);

return;

}

})

});

var occasArray = obj.ID\_OCCAS;

occasArray.forEach(function (item) {

var sql4 = "INSERT INTO vet\_occas\_assoc (ID\_VET, ID\_OCCAS) VALUES(" + idVet + ", ?)";

var inserts4 = [item];

console.log("OCCASION" + inserts4);

connection.query(mysql.format(sql4, inserts4), (err) => {

if (err) {

console.error(err);

fct(err, null);

return;

}

})

});

fct(null, results);

});

}

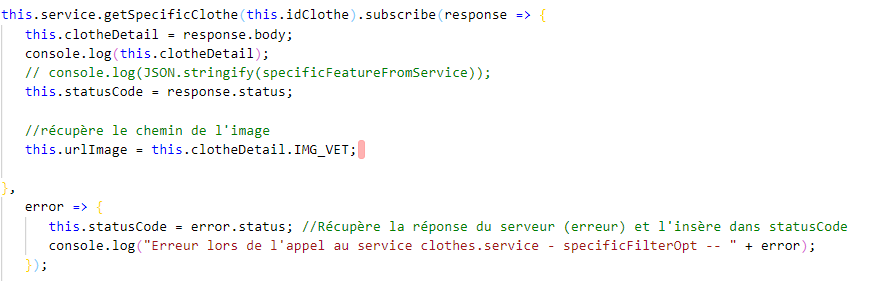
## Modification d’un vêtement

La gestion du formulaire de modification d’un vêtement est la même que celle du formulaire de création d’un vêtement, sauf qu’il est déjà prérempli avec les données provenant de la base de données.

Cela est fait grâce à la directive Angular « ngModel » au niveau du composant Angular « clothe-update ».



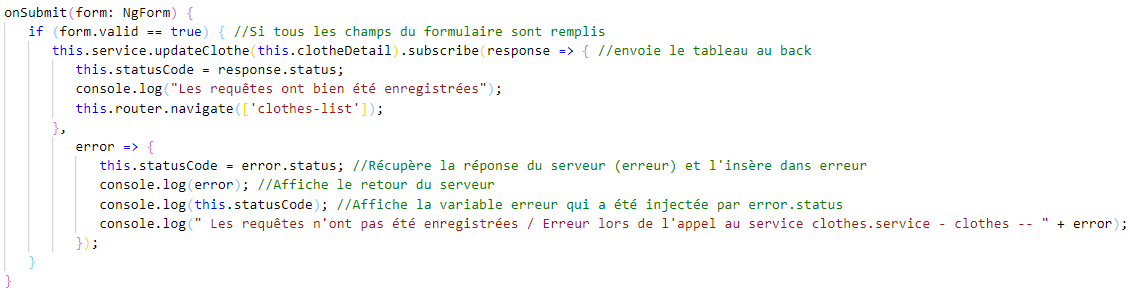
A l’initialisation de la page, je récupère le détail d’un vêtement en faisant appel au service.

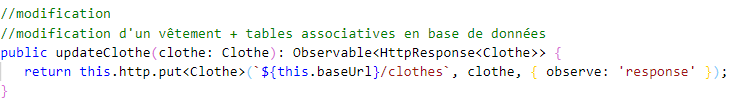


Le service fait appel à la ressource « clothes » en get de la même manière que les fonctions décrites précédemment.

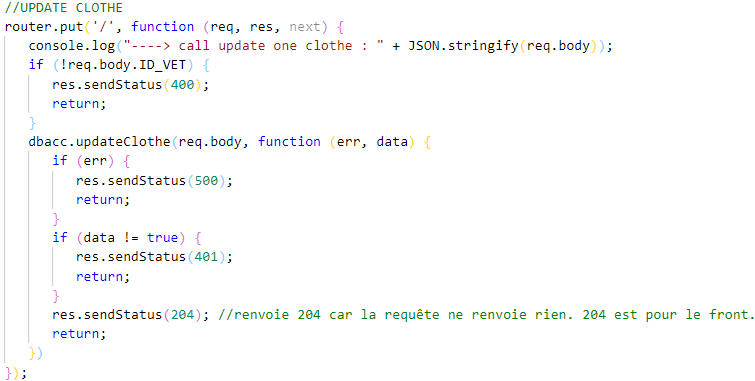
Au fur et à mesure que les champs sont modifiés dans le formulaire, la variable clotheDetail initialisée au départ est mise à jour grâce au binding bidirectionnel et à la directive NgModel.

Au clic sur le bouton « Enregistrer » du formulaire, la fonction onSubmit définie dans le fichier typescript du composant se lance :

* Je fais l’appel au service en lui passant l’objet Json contenu dans la variable clotheDetail, et je souscris au retour de l’observable contenu dans le service
* Lors du retour du service, je récupère le code http retourné par le serveur je dirige aussitôt l’utilisateur vers la liste de tous les vêtements mise à jour.

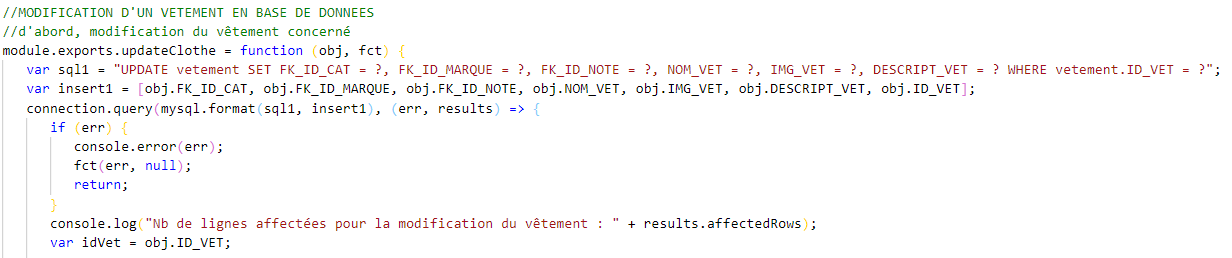
Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en put car il s’agit d’une modification et que j’applique autant que possible les principes REST(c’est-à-dire que je considère les éléments sur mon serveur comme des ressources).

Les données du front sont donc envoyées à la ressource « clothes » sur le serveur, qui joue la fonction correspondant à la requête http faite par le front. Dans ce cas, la fonction jouée est router.put(‘/’, …). Celle-ci vérifie que l’ID\_USER est bien présent dans l’objet Json transmis par le front (sinon renvoie un code http 400 « Bad Request »), puis fait appel à une autre fonction qui gère l’appel à la base de données.

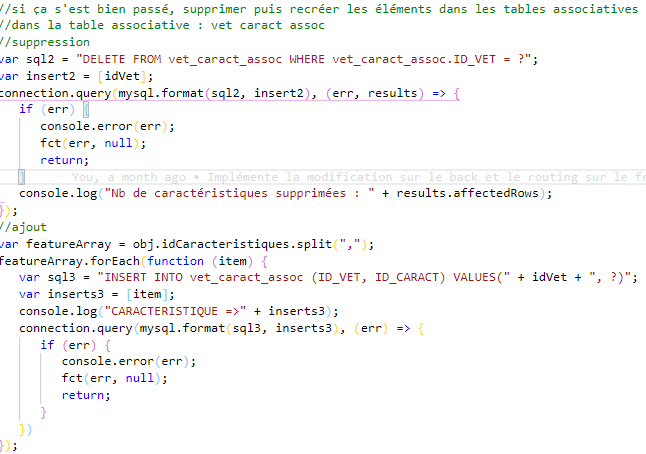
Le résultat de la requête effectuée dans l’autre fonction est récupéré et traité par la fonction dbacc.updateClothe et un code http 500 « Internal Server Error », 401 « Unauthorized » ou 204 « No Content» est renvoyé au front.

La fonction dans index.js appelée par la ressource « clothes » permet de faire les requêtes SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une.

Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

La première requête met à jour la table *vetement* de la base de données.

Les trois parties suivantes permettent de supprimer les données dans les tables associatives et d’y réinsérer les nouvelles données.



## Visualisation et filtres

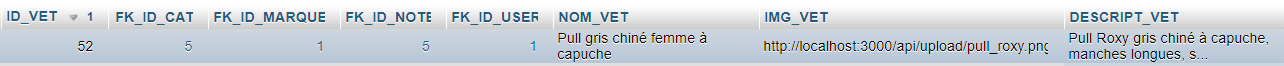
## Suppression d’un vêtement

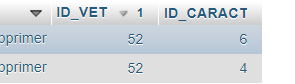
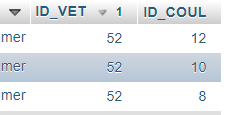
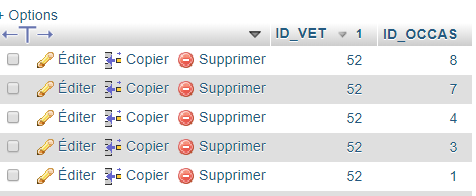
# Jeu d’essai de la fonctionnalité la plus représentative

La fonctionnalité la plus représentative est celle permettant la modification d’un vêtement.

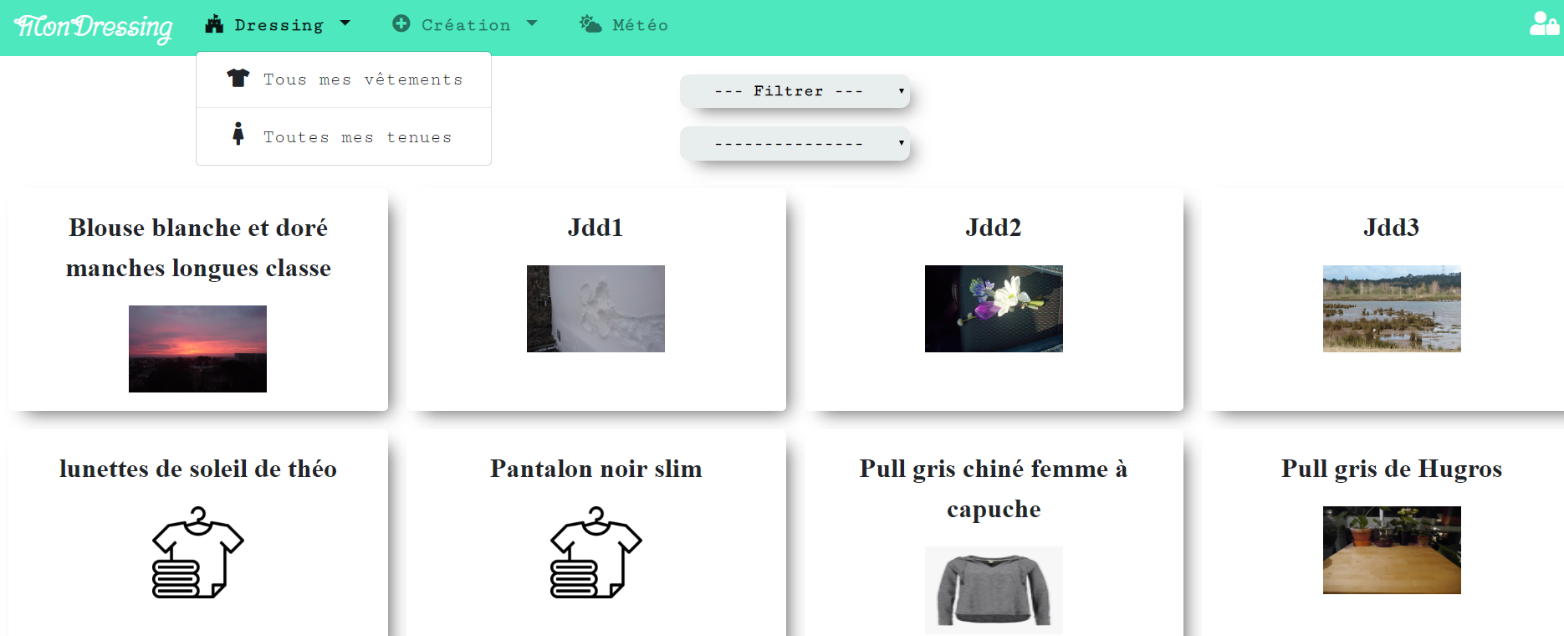
L’utilisateur a ajouté un vêtement et souhaite le modifier. Il se rend sur le formulaire de modification du vêtement, effectue la modification, et enregistre.

Le vêtement est modifié en base de données et s’affiche correctement sur l’application.

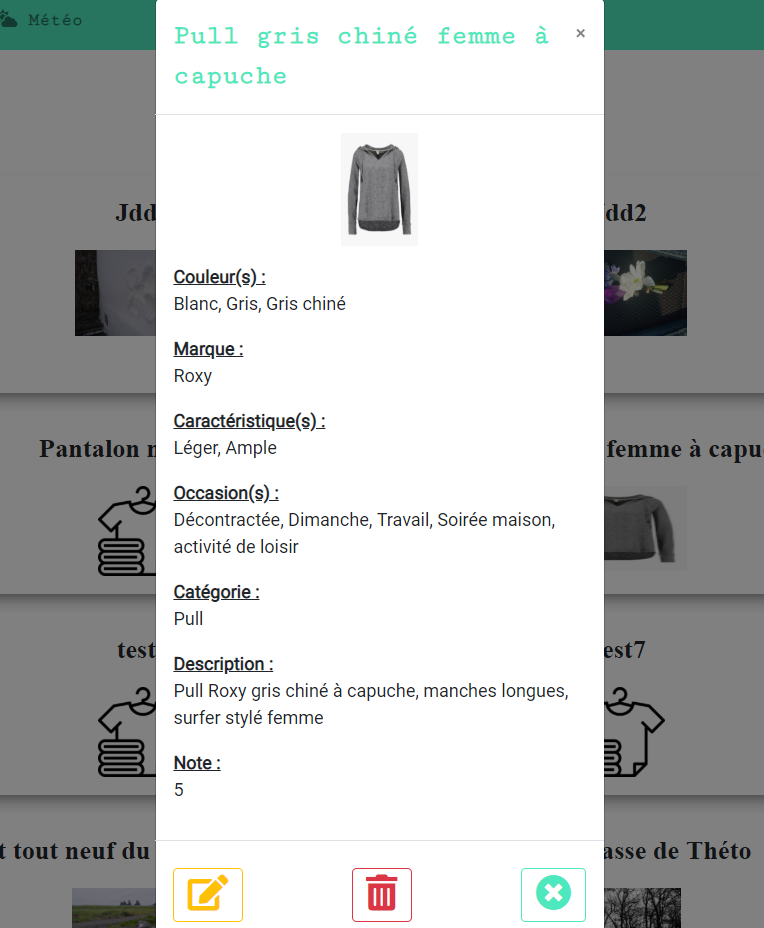
Un vêtement existe en base de données dans la table *vetement*, possédant l’id 52 :

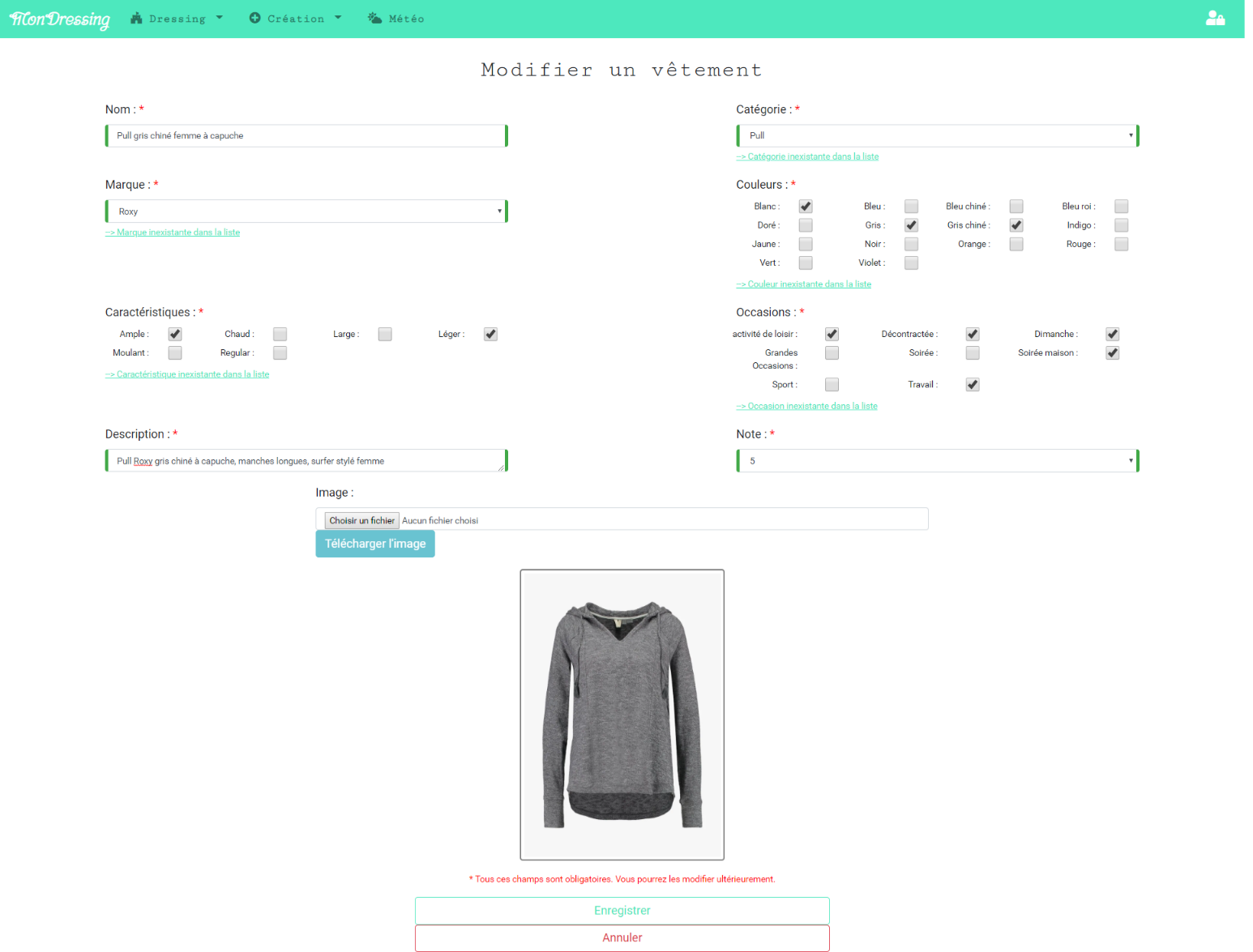
Dans la table vet\_caract\_assoc Dans la table vet\_coul\_assoc Dans la table vet\_occas\_assoc

L’utilisateur souhaite modifier ce vêtement.

Il se rend tout d’abord sur la page listant tous les vêtements en cliquant dans la barre de navigation sur « Dressing » puis sur « Tous mes vêtements ».

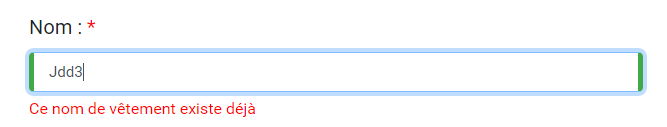
Il clique sur le vêtement qu’il souhaite modifier. Une modale de détail s’affiche.



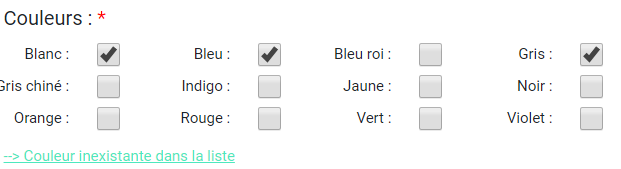
L’utilisateur clique sur le dessin indiquant la modification en bas à gauche de la modale, et il est redirigé vers la page de modification d’un vêtement qui est préremplie avec le détail du vêtement.

Lorsqu’il efface le contenu d’un champ du formulaire et qu’il clique ailleurs, un message d’erreur apparaît.

Lorsqu’il saisit un nom de vêtement, couleur, catégorie, caractéristique, marque ou occasion qui existe déjà, un message d’erreur apparaît.



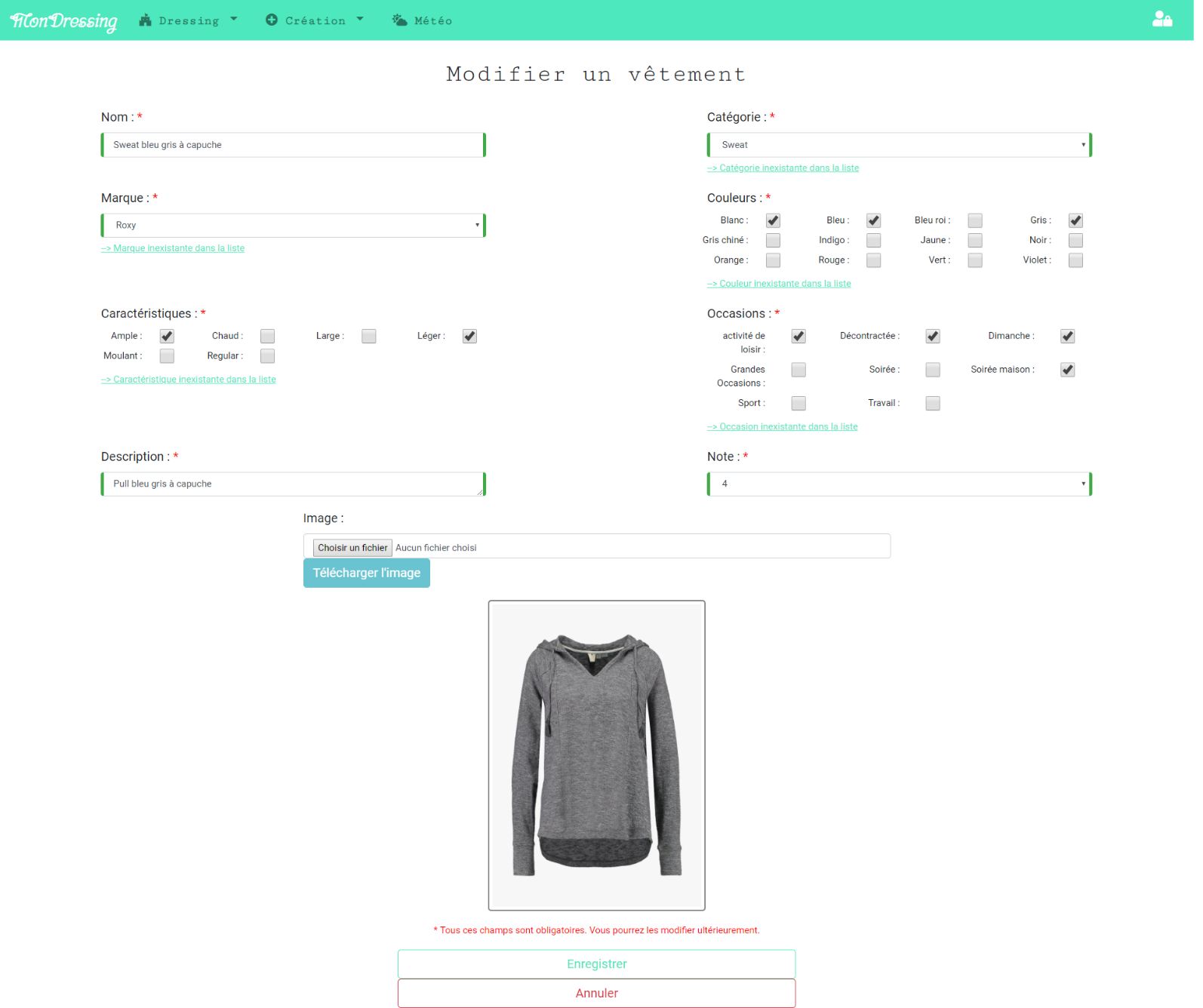
Si un élément n’existe pas dans la liste proposée, il clique sur le lien en dessous et un champ texte apparaît à la place de la liste.







L’utilisateur modifie son vêtement comme suit :

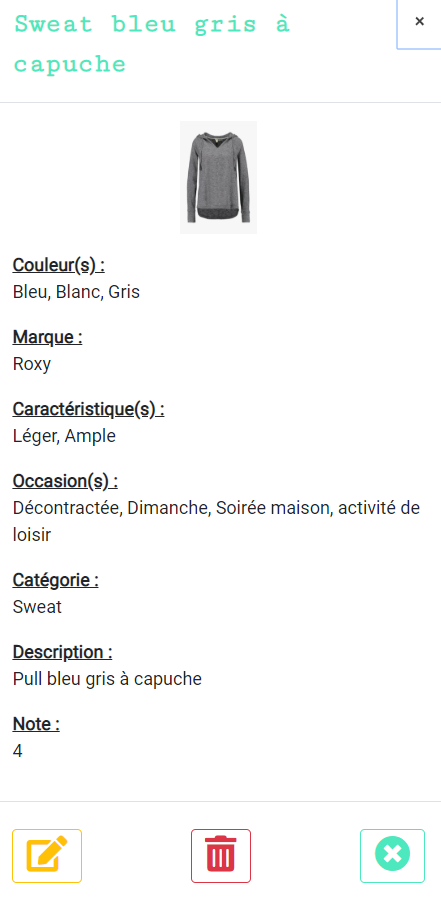


Lorsqu’il clique sur enregistrer, l’utilisateur est redirigé vers la liste de tous les vêtements mise à jour.

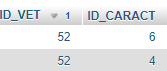
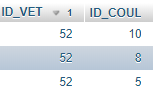
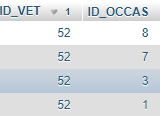


Le vêtement a bien modifié :

Dans l’application.

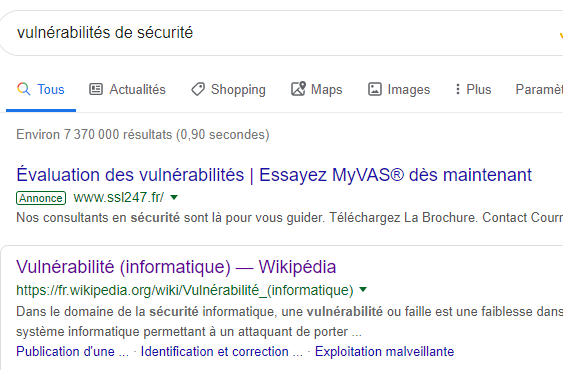


En base de données.



# Veille sur les vulnérabilités de sécurité

J’ai, dans un premier temps, cherché les mots clés « vulnérabilités de sécurité » sur le moteur de recherche Google.

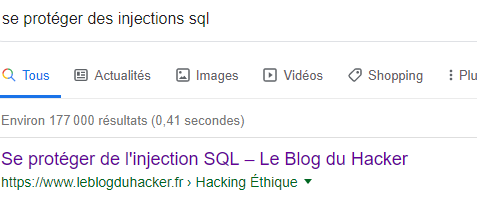


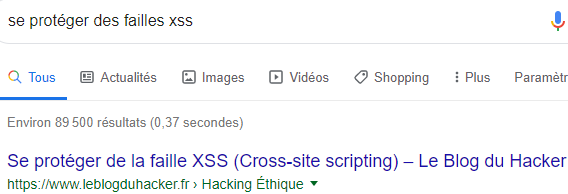
La première page était Wikipédia, j’ai commencé à m’informer grâce à cette page.

<https://fr.wikipedia.org/wiki/Vuln%C3%A9rabilit%C3%A9_(informatique)>

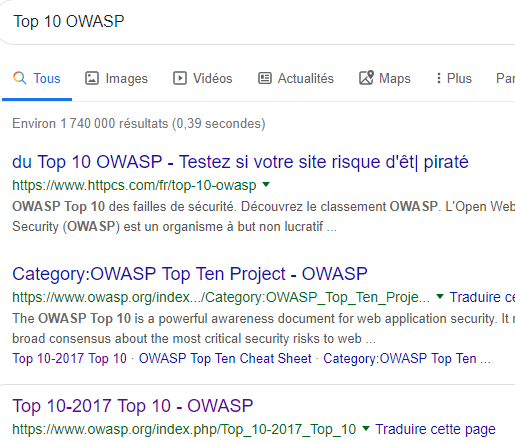
Une vulnérabilité ou faille est une faiblesse dans un système informatique permettant à un attaquant de porter atteinte à l'intégrité de ce système, c'est-à-dire à son fonctionnement normal, à la confidentialité ou à l'intégrité des données qu'il contient.

Cela m’a amenée à me renseigner sur les injections SQL et les failles XSS, qui sont les vulnérabilités les plus répandues.

J’ai donc saisi ces mots clés sur le moteur de recherche Google.



Ce site m’a amenée à me renseigner sur d’autres types de failles et sur le top 10 OWASP.

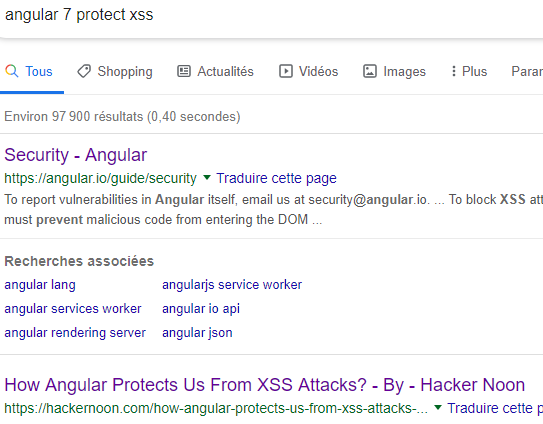


J’ai choisi le 3e site qui est le site officiel, et qui liste le top 10 des risques sécurité.

En premier, vient l’injection SQL. Ensuite, viennent les authentifications brisées (lorsqu’elles n’ont pas été correctement développées), l’exposition des données sensibles, les anciens processeurs XML mal configurés, le contrôle d’accès brisé (lorsque ça n’a pas été correctement développé), une mauvaise configuration de la sécurité, des failles XSS, la désérialisation non sécurisée, l’utilisation de composants ou librairie contenant des failles connues (car les composants ont les mêmes privilèges que l’application), et enfin un suivi insuffisant des incidents sur le site.

Certaines vulnérabilités surviennent lorsque la saisie d'un utilisateur n'est pas contrôlée, permettant l'exécution de commandes ou de requêtes SQL (connues sous le nom d'[injection SQL](https://fr.wikipedia.org/wiki/Injection_SQL)). Afin de s’en prémunir, il est recommandé d’échapper/supprimer les caractères spéciaux transmis via les formulaires, de préparer ses requêtes, et d’utiliser des expressions régulières pour vérifier les données.

La faille XSS ou Cross-Site Scripting consiste à injecter un script dans une page pour provoquer une action malveillante. Les autres utilisateurs exécutent ensuite ce script sans s’en rendre compte dès l’ouverture de la page. Cela permet notamment de récupérer les cookies de quelqu’un pour se faire passer pour cet utilisateur sans avoir de droits.



Angular permet de se prémunir des failles XSS en nettoyant et en utilisant des méthodes telle que bypassSecurityTrustX. Cela lui permet d’échapper toutes les valeurs ajoutées au DOM qu’il considère comme non fiables (par défaut, toutes les valeurs).

Pour vérifier cela, j’ai ajouté ce script <script>alert('Il y a une faille XSS')</script> à l’un de mes formulaires que j’ai envoyé. Le script n’a pas été lu par Angular, qui me protège bien des failles XSS.

Afin de protéger mon application des injections SQL, j’ai préparé toutes mes requêtes et j’ai intégré une fonction permettant de supprimer les espaces et les points-virgules saisis dans les formulaires.

Pour protéger les accès aux ressources de mon site, j’ai également mis en place des guards Angular (qui interdisent l’accès à une page tant que l’utilisateur n’est pas connecté avec un token JWT valide).

Afin de protéger les données utilisateur et notamment les mots de passe, ces derniers sont hashés en SHA256 avant d’être stockés en base de données.

De plus, un certificat https sera installé sur le serveur et lié à l’application afin de chiffrer les données qui transitent sur le réseau (notamment identifiants et mots de passes). Des expressions régulières seront mises en place pour contrôler ce qui est transmis du formulaire vers le serveur.

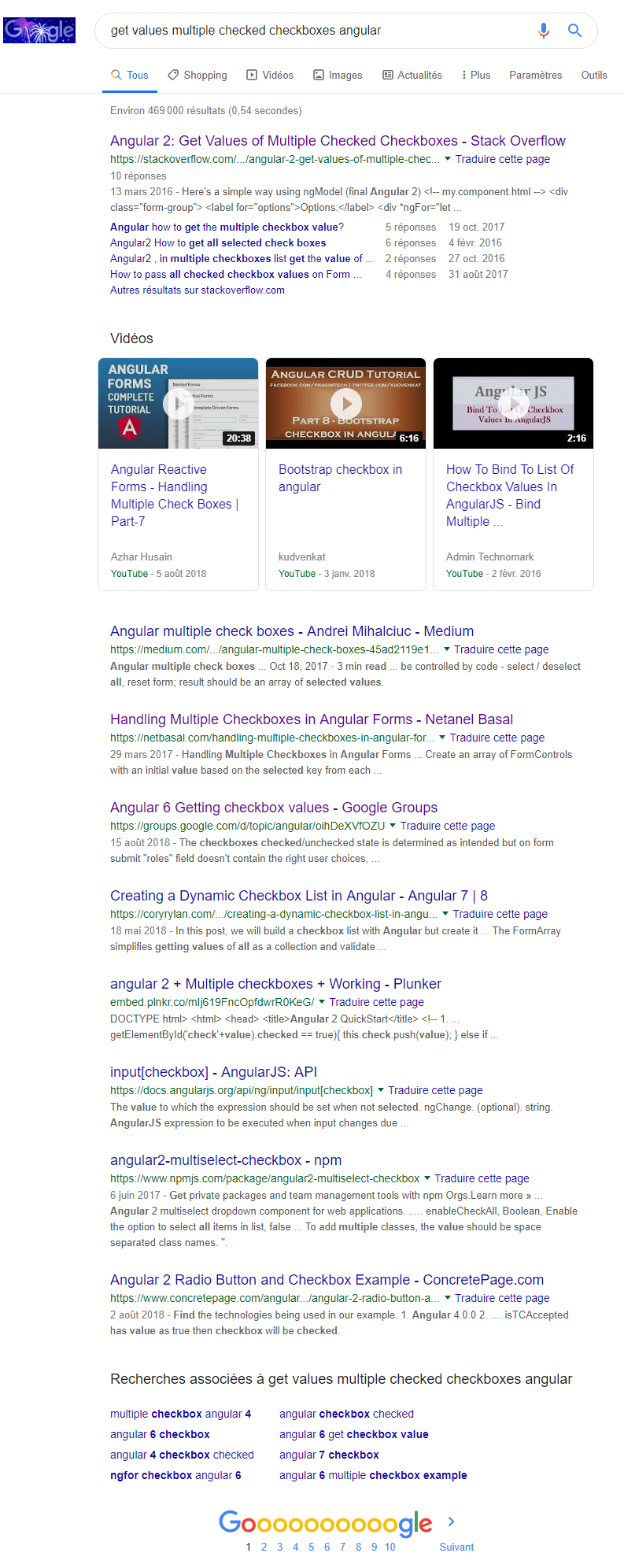
# Recherche sur un site anglophone

Lors du développement de l’application MonDressing, j’ai dû faire des recherches en anglais et notamment sur la façon de récupérer les valeurs de plusieurs cases à cocher cochées par un utilisateur.

En effet, le formulaire de création d’un vêtement permet de renseigner plusieurs couleurs, caractéristiques et occasions pour un vêtement.

Cependant, la récupération des valeurs de plusieurs cases à cocher ne se fait pas de la même manière que la récupération des autres données plus simples saisies dans le formulaire.

Pour effectuer la recherche, j’ai tapé les mots clés « get values multiple checked checkboxes angular » sur le moteur de recherche Google.



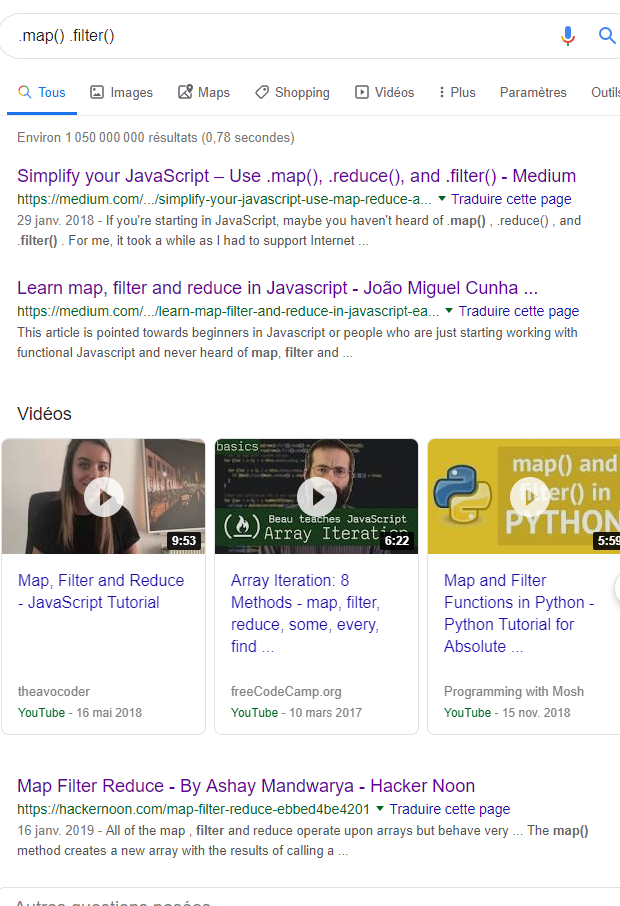
J’ai choisi d’aller sur le premier site, stackoverflow.com, car il s’agit d’un site connu dans le monde du développement pour être une ressource sur laquelle on trouve la majorité de ses réponses.

J’y ai trouvé, parmi les nombreuses réponses, la méthode que j’ai finalement utilisée me permettant de récupérer les valeurs des cases à cocher cochées.



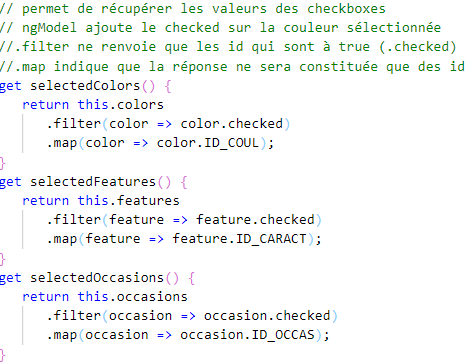
J’ai ensuite cherché sur internet comment fonctionnent les méthodes « .map() » et « .filter() ».

J’ai pour cela utilisé les mots clés « .map() .filter() » sur le moteur de recherche Google.



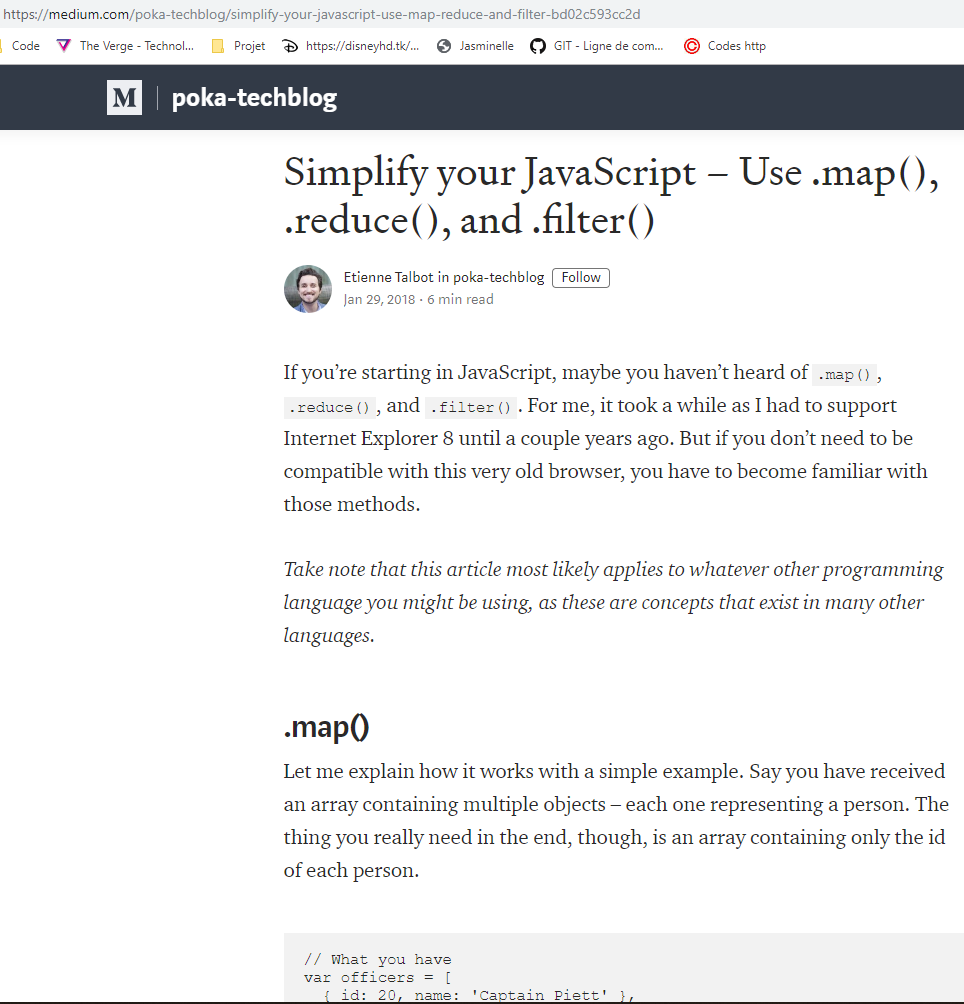
J’ai choisi le site medium.com car il est connu pour ses nombreux articles sur le monde de l’informatique notamment.

Voici mon code dans lequel j’ai implémenté la méthode trouvée :





# Extrait du site anglophone et traduction en français.



**Simplifiez votre JavaScript – utilisez .map(), .reduce(), et .filter().**

Si vous débutez en Javascript, peut être que vous n’avez pas entendu parler de .map(), .reduce() et .filter(). Pour moi, cela a pris un certain temps car mon code devait être compatible avec Internet Explorer 8 jusqu’à il y a quelques années. Mais si votre code n’a pas besoin d’être compatible avec ce très vieux navigateur, vous devez vous familiariser avec ces méthodes.

Notez que cet article s’applique très probablement à n’importe quel langage de programmation que vous pourriez utiliser, car il s’agit de concepts qui existent dans de nombreux autres langages.

**.map()**

Laissez-moi vous expliquer comment cela fonctionne avec un exemple simple. Supposons que vous ayez reçu un tableau qui contient plusieurs objets – chacun représentant une personne. Cependant, ce dont vous avez réellement besoin à la fin, c’est d’un tableau contenant seulement l’identifiant de chaque personne.