|  |
| --- |
|  |
| 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](#_Toc13923377)

[A. Développer la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité 3](#_Toc13923378)

[B. Développer la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité 3](#_Toc13923379)

[II. Résumé du projet 4](#_Toc13923380)

[III. Cahier des charges 4](#_Toc13923381)

[IV. User Stories et spécifications techniques 4](#_Toc13923382)

[V. Base de données 5](#_Toc13923383)

[A. Modélisation 5](#_Toc13923384)

[1. Modèle Conceptuel des Données 5](#_Toc13923385)

[2. Modèle Logique des Données 5](#_Toc13923386)

[3. Code SQL 5](#_Toc13923387)

[4. Mise en place 5](#_Toc13923388)

[VI. Réalisations 5](#_Toc13923389)

[A. Création d’un compte 5](#_Toc13923390)

[B. Connexion à l’application 5](#_Toc13923391)

[C. Création d’un vêtement 5](#_Toc13923392)

[D. Visualisation et filtres 5](#_Toc13923393)

[E. Modification d‘un vêtement 5](#_Toc13923394)

[F. Suppression d’un vêtement 6](#_Toc13923395)

[VII. Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues) 6](#_Toc13923396)

[VIII. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité 6](#_Toc13923397)

[IX. Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone 6](#_Toc13923398)

[X. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes). 6](#_Toc13923399)

# 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 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 6 et Bootstrap 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.

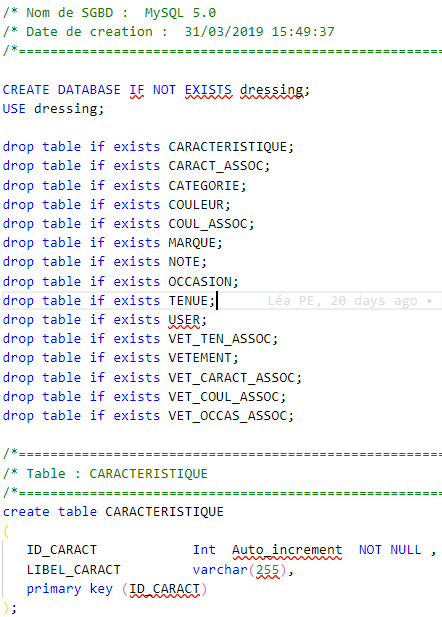
### Modèle Logique des Données

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

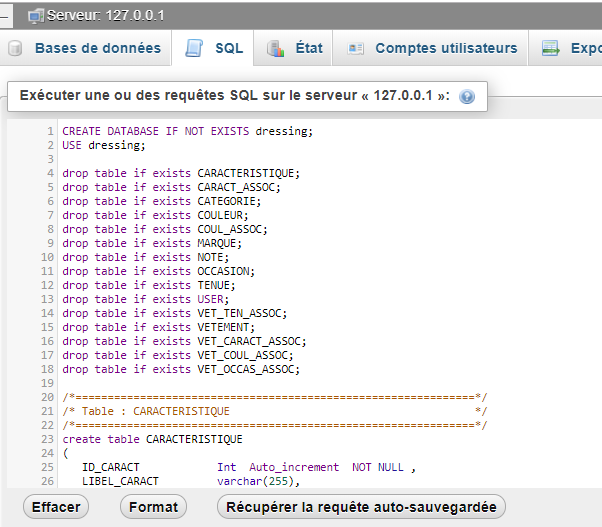
J’ai ensuite généré le code SQL avec Power AMC, que j’ai retravaillé afin que les noms des clés étrangères ne soient pas les mêmes que ceux des clés primaires.

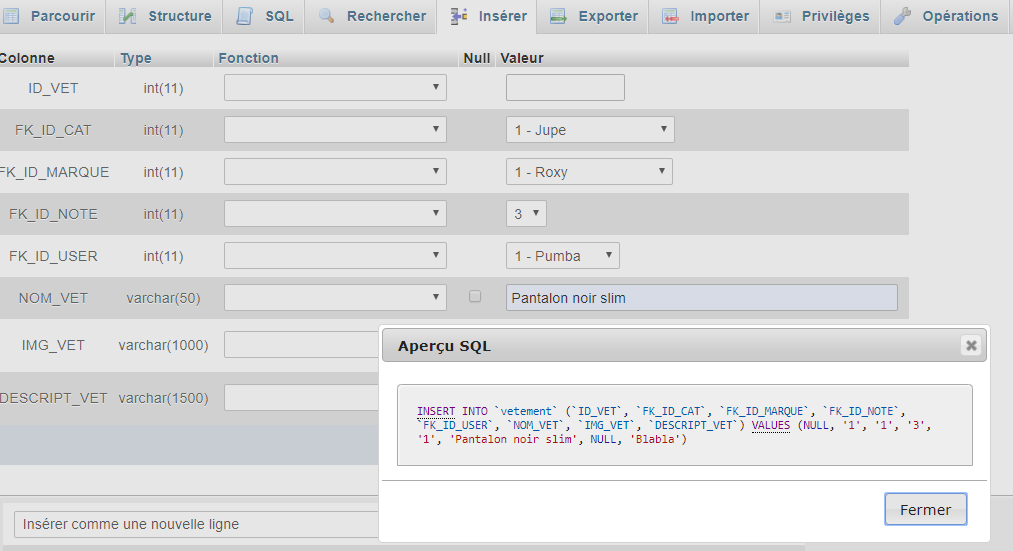
Pour des questions de re jouabilité, j’ai ajouté en début de fichier les instructions SQL permettant de créer la base de données, d’utiliser la base de données, et supprimer les tables si elles existent avant de créer toutes les tables.



## 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.



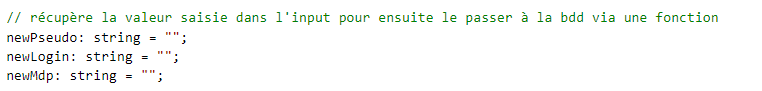


# Réalisations

Pour réaliser cette application, j’ai utilisé le framework Angular 7 (typescript, HTML5, CSS3), le framework Bootstrap4, le serveur NodeJs (javascript), le framework Express, et la base de données Oracle Mysql. J’ai également utilisé l’IDE Visual Studio Code.

## Création d’un compte

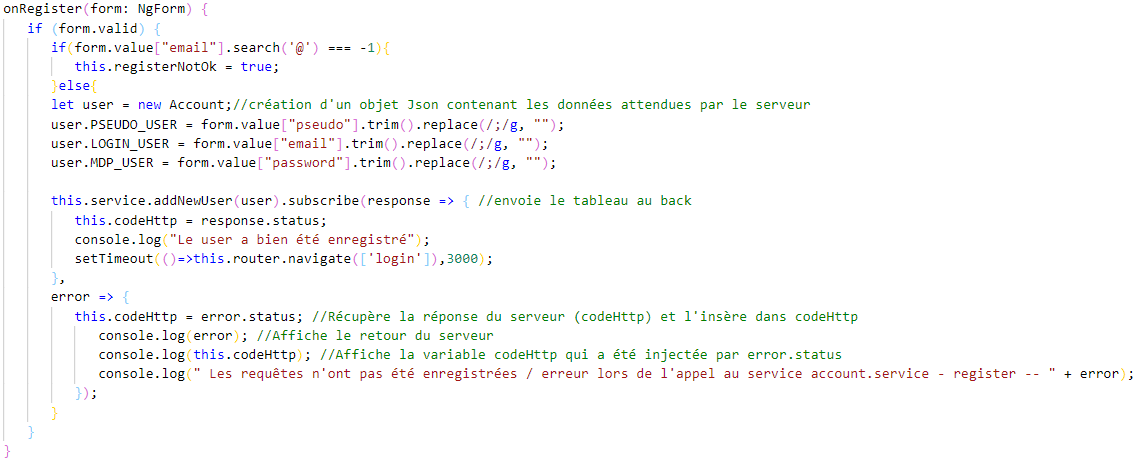
Pour créer un compte, je récupère les données saisies par l’utilisateur dans le formulaire au niveau du composant Angular « register ».



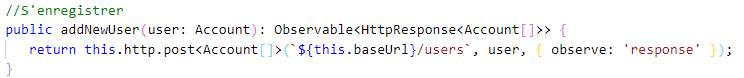
La récupération des informations est faite grâce à la directive Angular « ngModel » qui permet de faire du binding bidirectionnel des données.



Au clic sur le bouton « S’enregistrer » du formulaire, la fonction onRegister définie dans le fichier typescript du component 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.

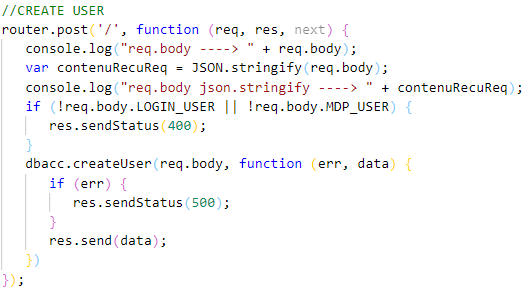
Je mets en place la route qui crée le point d’entrée sur le serveur.



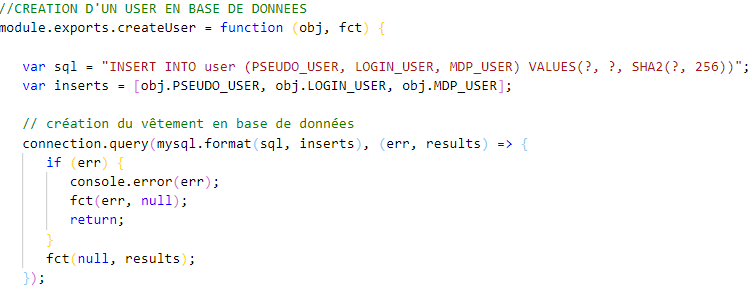


Les données du front sont donc envoyées à la ressource « users » sur le serveur, qui joue la fonction correspondant à la requête http faite par le front. Dans ce cas, la fonction jouée est router.post(‘/’, …). Celle-ci vérifie que les données attendues sont bien présentes dans l’objet Json transmis par le front (sinon renvoie un code http 400), 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 ou 200 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.

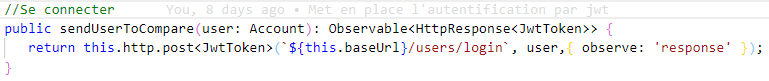
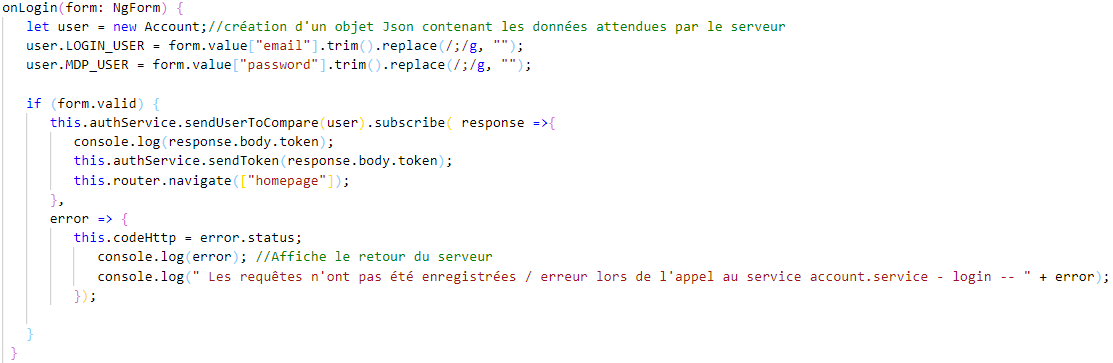
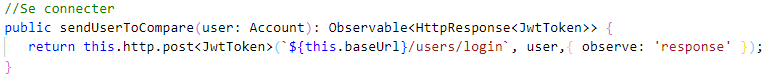


## Connexion à l’application

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 component 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 souscris au retour de l’observable contenu dans le 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

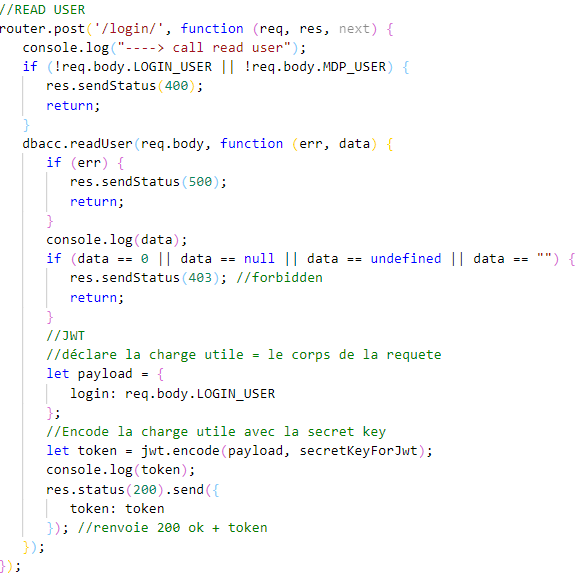
Le service fait appel à la ressource mise à disposition par le serveur. La requête http est en post car je dois envoyer les informations au serveur pour qu’il puisse les comparer aux informations présentes en base de données.

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), 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 ou 200 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.



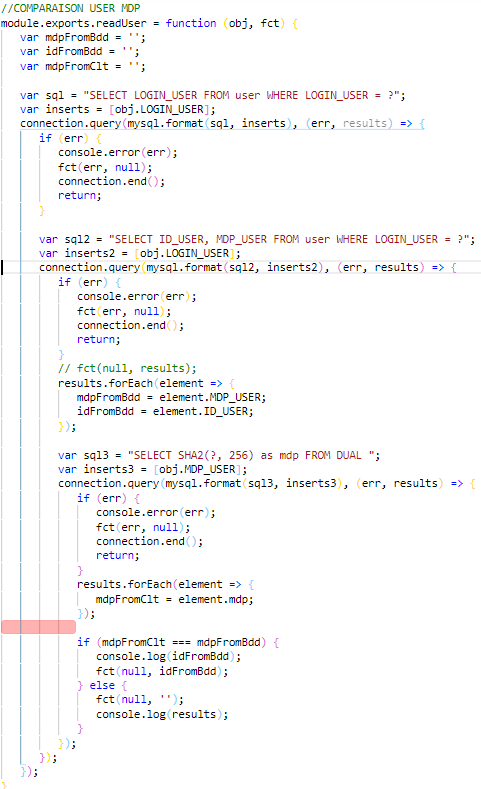
La fonction dans index.js appelée par la ressource « users » permet de faire les requêtes SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une.

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.

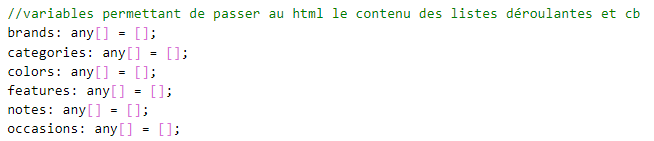
La troisième requête permet de comparer le mot de passe transmis par le front et hashé, au mot de passe enregistré en base de données. S’il n’y a pas d’erreur, alors je renvoie l’id à la fonction précédente pour qu’il soit encrypté, sinon, je renvoie vide (‘’).

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

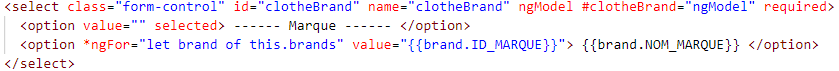


## Création d’un vêtement

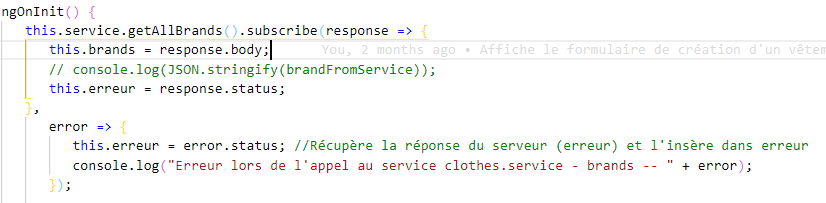
Au chargement du formulaire de création d’un vêtement, j’initialise les variables qui vont me permettre de transmettre au html le contenu des listes déroulantes et des cases à cocher.

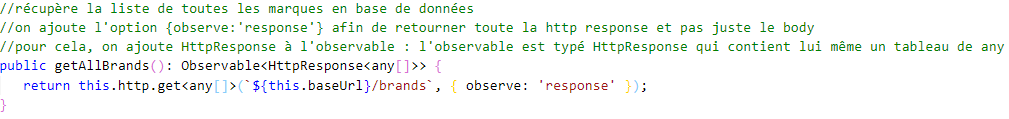


Je passe les données au front grâce aux directives structurelles Angular et au binding bidirectionnel.

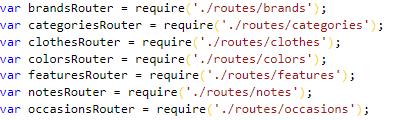


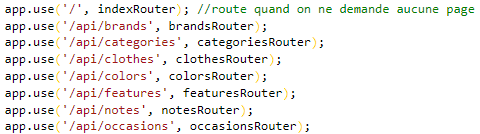
J’initialise également les appels au service permettant de récupérer la liste des marques, catégories, couleurs, caractéristiques, notes et occasions, et lorsque j’ai un retour du service, j’ajoute le contenu à la variable brands précédemment initialisée.



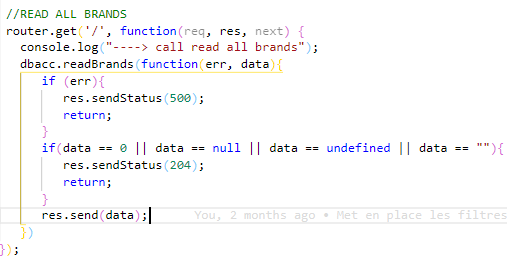
Le service fait appel aux ressources mises à disposition par le serveur. La requête http est en get car il s’agit d’une récupération de données du point de vue REST.

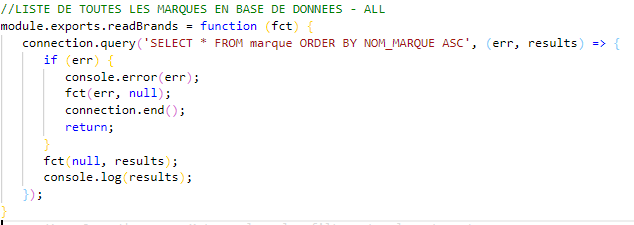
Je mets en place les routes qui créent les points d’entrée sur le serveur.



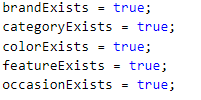


La requête http faite en get depuis le front vient chercher les informations de la ressource concernée (brands.js, categories.js, features.js, colors.js, notes.js, occasions.js). La ressource fait appel à une fonction readBrands qui gère l’appel à la base de données (dans index.js) et retourne la liste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



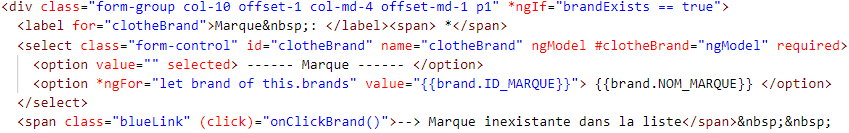
Cette fonction située dans index.js appelée par la ressource permet de faire la requête SQL à la base de données et de retourner le résultat ou l’erreur s’il y en a une, à la fonction ci-dessus contenue dans la ressource. Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL.

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.

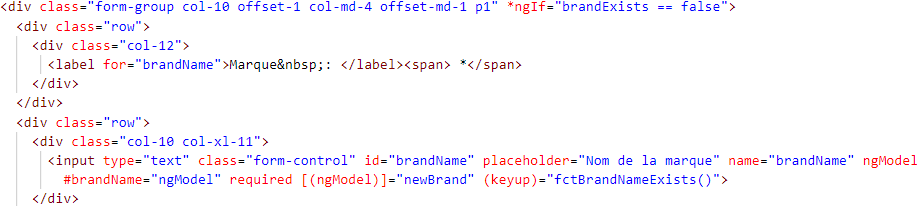


J’utilise ensuite cette variable pour afficher un contenu différent si elle est true ou false.

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



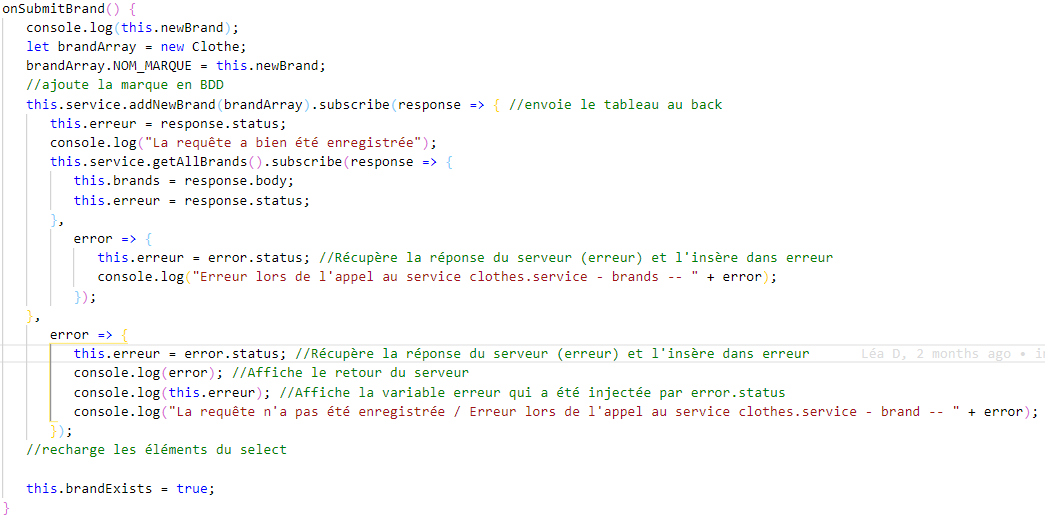
Si elle est « false », j’affiche un input de type texte permettant de renseigner un nouvel élément.

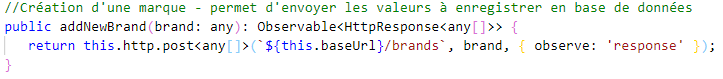
Le changement d’état est géré au clic sur le lien « Marque inexistante dans la liste ».



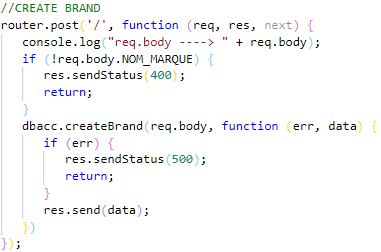
Lorsque l’utilisateur soumet l’ajout du nom de la marque, couleur, catégorie, caractéristique ou occasion qui n’existe pas en base, cela lance une fonction (dans cet exemple la fonction on SubmitBrand() )qui :

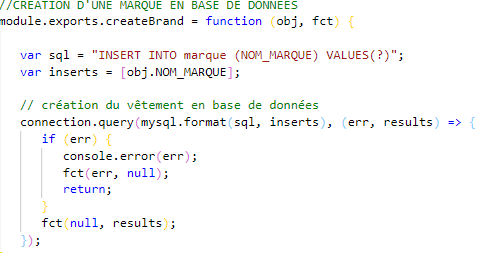
* Initialise un objet Json dans lequel j’ajoute le contenu du champ texte saisi par l’utilisateur dans le formulaire.
* Fait l’appel au service en lui passant l’objet Json créé, et souscrit au retour de l’observable contenu dans le service
* Lors du retour du service, rappelle la fonction permettant 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.



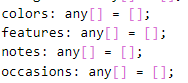


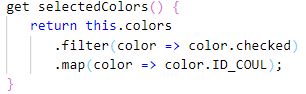
Pour que l’utilisateur puisse ajouter un vêtement, je récupère les données saisies par l’utilisateur dans le formulaire au niveau du composant Angular « clothe-create ».

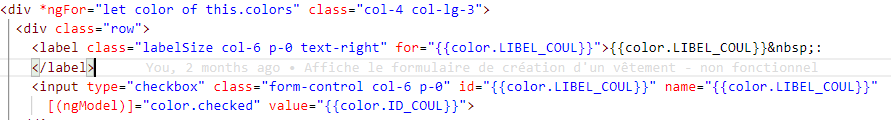
Pour cela, je récupère les informations grâce à l’attribut name des champs du formulaire pour les données simples.



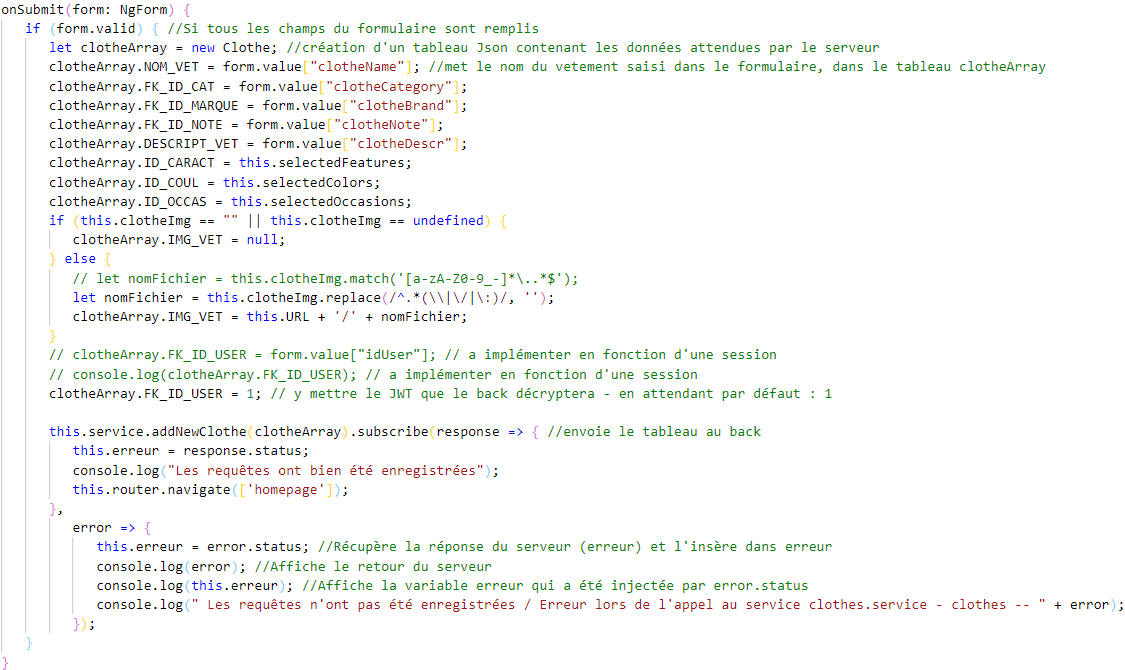
Concernant les données multiples issues des cases à cocher, je les récupère grâce à l’initialisation de variables et à une fonction permettant de trier les informations récupérées.

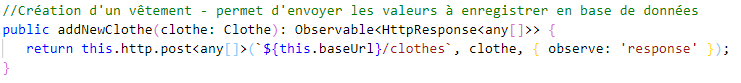




Je lie cela au html grâce à la directive structurelle Angular ngFor et grâce à ngModel.

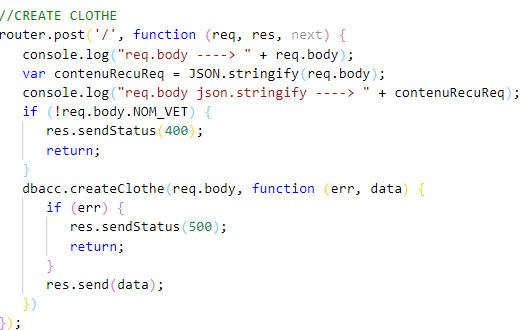
Au clic sur le bouton « Enregistrer » du formulaire, la fonction onSubmit définie dans le fichier typescript du component se lance :

* 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).

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), 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 ou 200 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*.

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.

Pour cela, j’ai ajouté ……..

## Visualisation et filtres

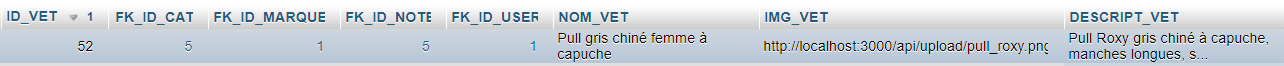
## Suppression d’un vêtement

# Jeu d’essai : formulaire de modification d’un vêtement Présentation du jeu d’essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

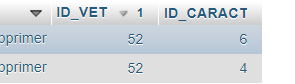
La fonctionnalité la plus représentative est celle permettant la modification d’un vêtement.

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.

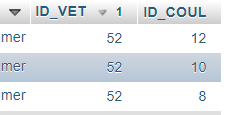
Le vêtement est modifié en base de données et s’affiche correctement sur l’application.

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

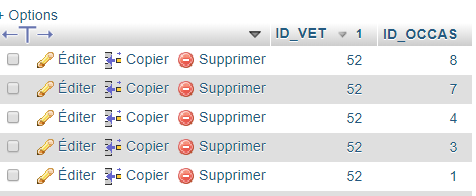
Dans la table vet\_caract\_assoc :



Dans la table vet\_coul\_assoc :

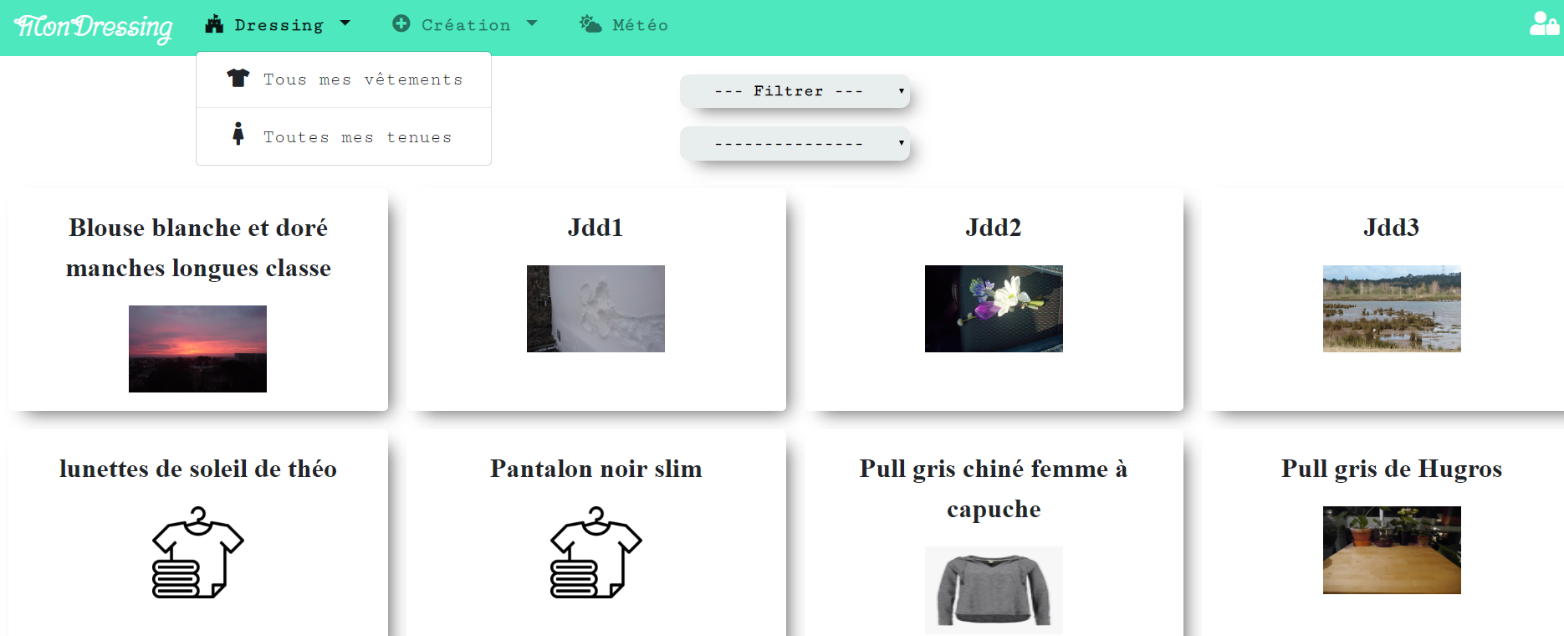


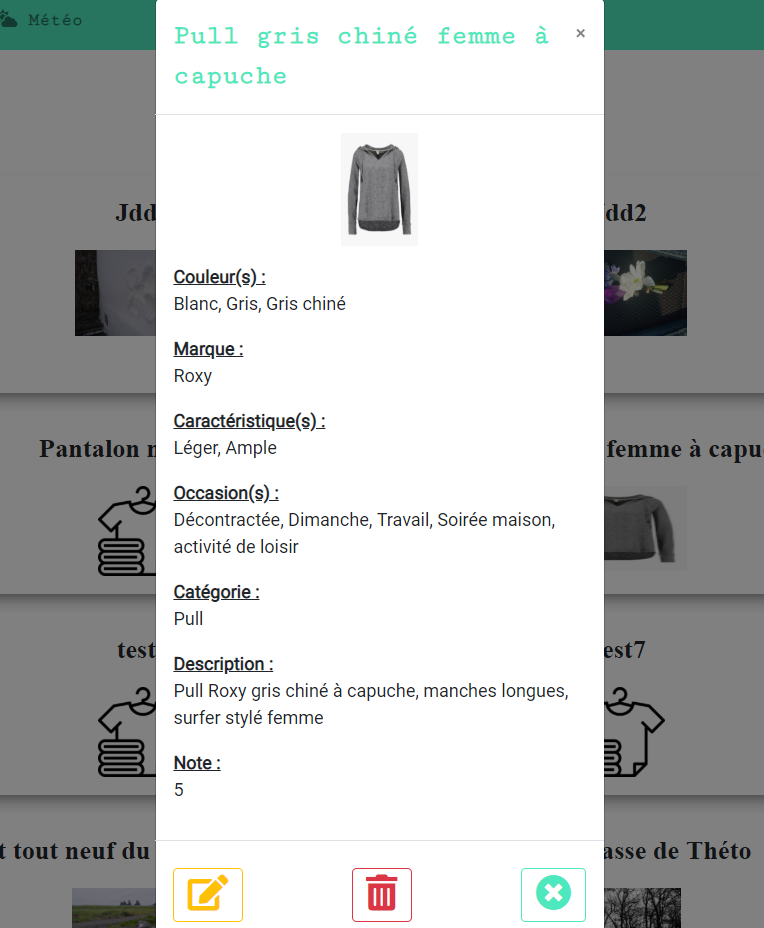
Dans la table vet\_occas\_assoc :

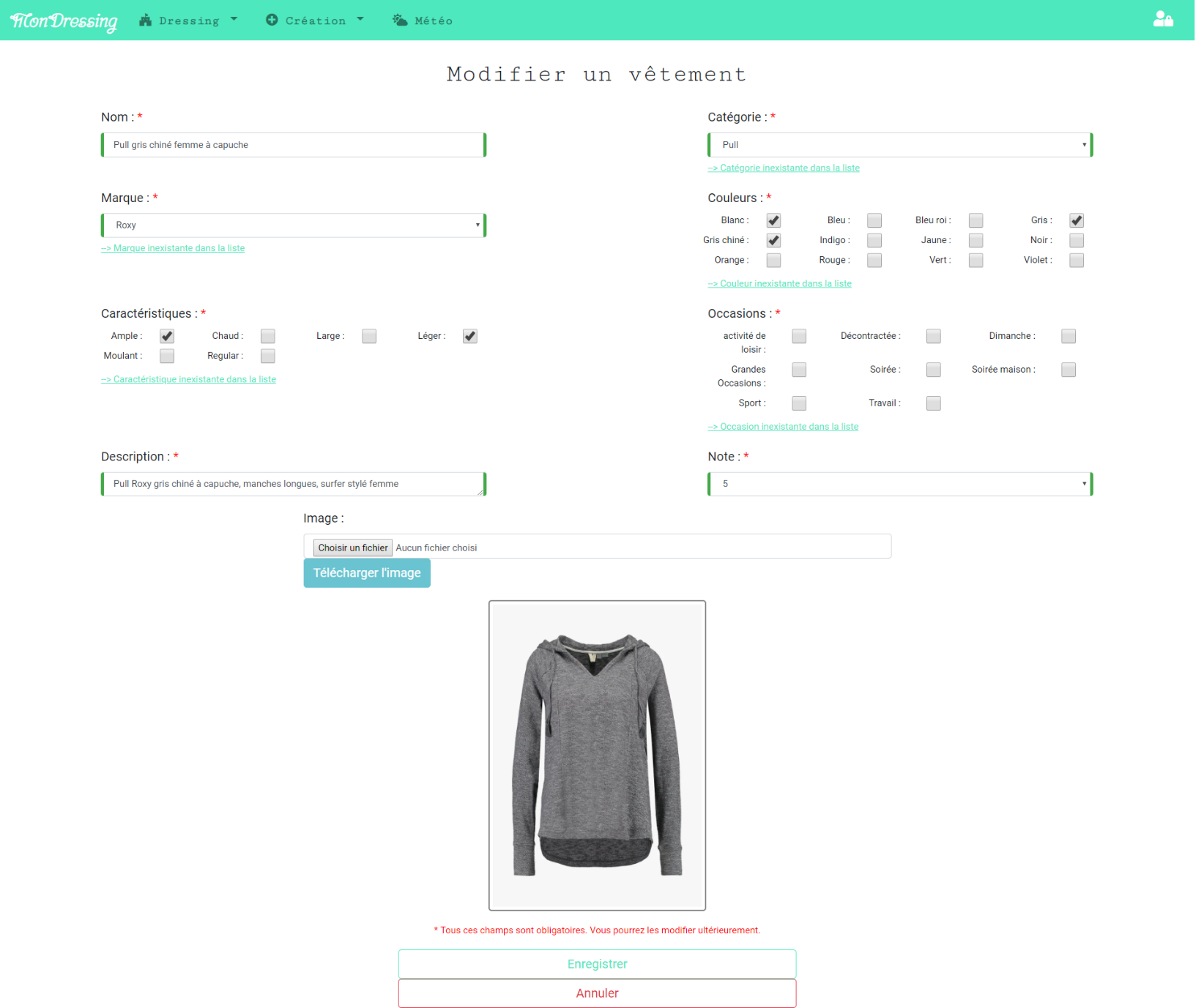


L’utilisateur souhaite modifier ce vêtement.

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.

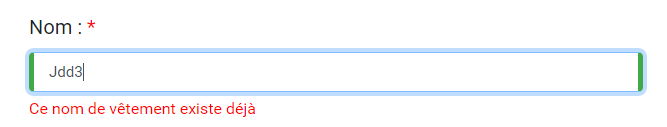


L’utilisateur clique sur le dessin indiquant la modification en bas à gauche de la modale, et il est redirigé vers la page de modification d’un vêtement qui est préremplie avec le détail du vêtement.

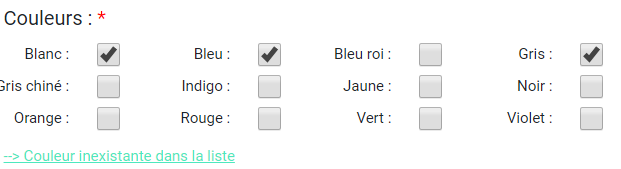
Lorsqu’il efface le contenu d’un champ du formulaire et qu’il clique ailleurs, un message d’erreur apparaît.



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

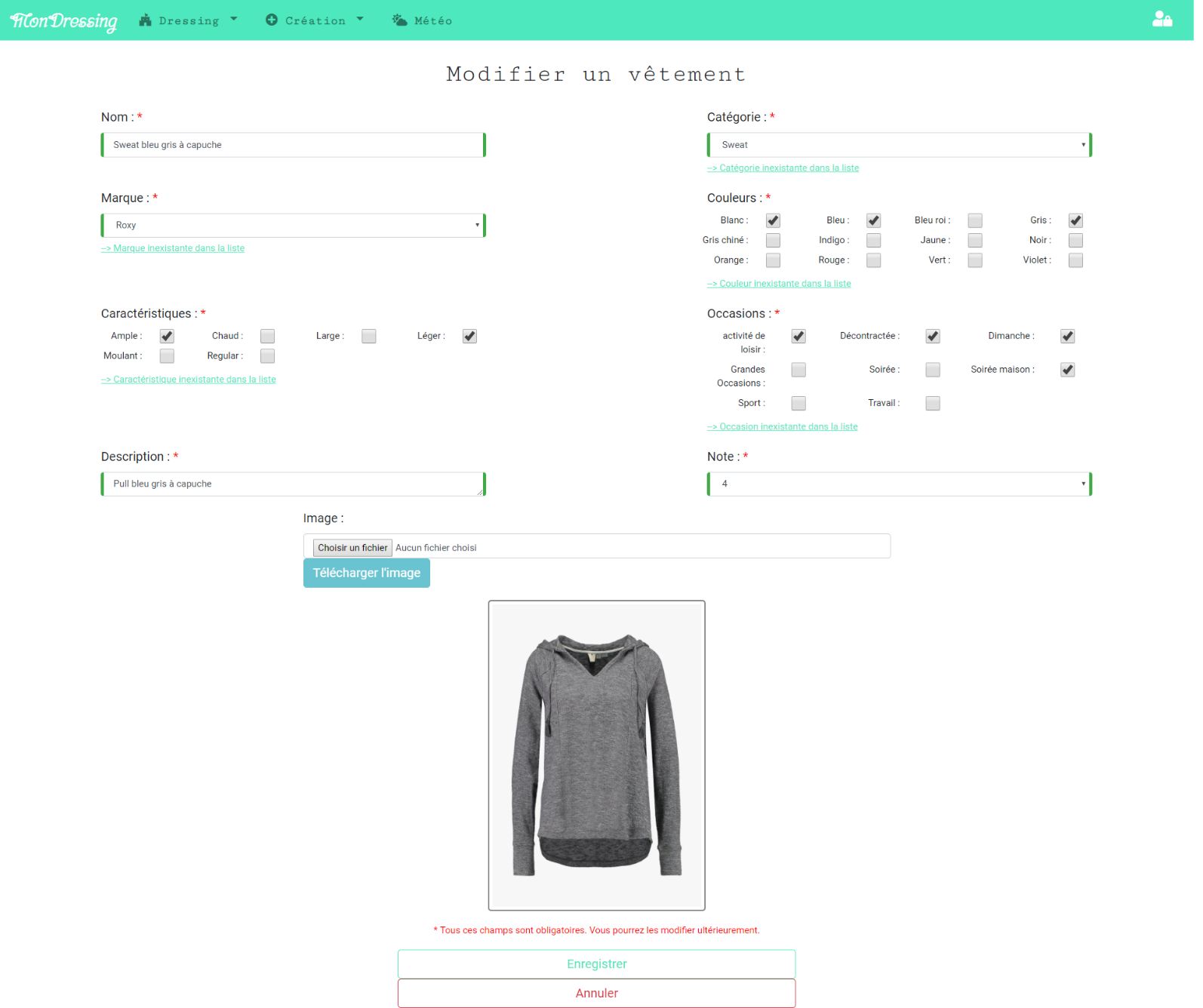


Si un élément n’existe pas dans la liste proposée, il clique sur le lien en dessous et un champ texte apparaît à la place de la liste.







L’utilisateur modifie son vêtement comme suit : 

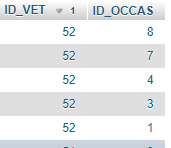
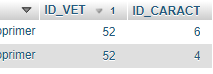
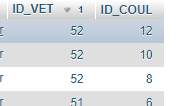
Lorsqu’il clique sur enregistrer, l’utilisateur est redirigé vers la liste de tous les vêtements mise à jour.



Le vêtement a bien modifié :

Dans l’application.



En base de données.

# Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

La description de la veille sur les vulnérabilités de sécurité est liée à l’une des deux compétences « Développer une interface utilisateur web dynamique » ou « Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce » dans le cadre du projet en entreprise.

Dans le cas de la veille sur les vulnérabilités, le candidat indique comment il a effectué la veille : les sites et les mots clés utilisés. Il indique les vulnérabilités trouvées et éventuellement les failles potentielles corrigées.

# Description d’une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone

La description de la situation de travail, ayant nécessité une recherche basée sur un ou des sites anglophones, concerne un problème technique ou une nouvelle fonctionnalité à mettre en oeuvre, dans le cadre du projet en entreprise. Elle est liée à l’une des compétences du titre professionnel.

Dans le cas de la recherche de solution, basée sur un ou des sites anglophones, à un problème technique ou une nouvelle fonctionnalité à mettre en oeuvre, le candidat décrit le besoin d’information, et indique comment il a effectué la recherche : les mots clés de recherche utilisés et la liste des sites retournés. Il précise les critères de sélection du (ou des) site(s). Il indique la solution trouvée et si elle a pu être mise en oeuvre.

# Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes).