

Dossier de projet

Application Web Dressing

Léa DELANNAY
29/07/2019

Table des matières

I.	Liste des compétences du référentiel couvertes par le projet	2
A.	Développer la partie front-end d'une application web ou web mobile	2
B.	Développer la partie back-end d'une application web ou web mobile.....	2
II.	Résumé du projet	2
III.	Cahier des charges	3
A.	Présentation du projet :	3
1.	Les objectifs de l'application.....	3
2.	La cible	3
3.	Les objectifs quantitatifs.....	3
B.	Graphisme et ergonomie.....	4
1.	La charte graphique	4
2.	Wireframe et Maquettage	4
C.	Spécificités et livrables	4
1.	Le contenu de l'application	4
2.	Contraintes techniques.....	5
3.	Les livrables.....	5
4.	Le Planning.....	6
D.	Glossaire	6
IV.	User Stories et spécifications techniques	7
A.	Enregistrement.....	7
V.	Base de données	9
A.	Modélisation	9
B.	Modèle Conceptuel des Données et Modèle Logique des Données.....	9
C.	Code SQL	9
D.	Mise en place	9
VI.	Réalisations	10
A.	Création d'un compte	10
B.	Connexion à l'application.....	12
C.	Création d'un vêtement	15
D.	Modification d'un vêtement	21
VII.	Jeu d'essai de la fonctionnalité la plus représentative	24
VIII.	Veille sur les vulnérabilités de sécurité	29
IX.	Recherche sur un site anglophone.....	31
X.	Extrait du site anglophone et traduction en français.	34

I. Liste des compétences du référentiel couvertes par le projet

A. 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

B. 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

II. 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 enregistrés dans l'application, et de lui permettre de les gérer. Dans sa seconde version, l'application permettra de générer aléatoirement des tenues en fonction de certains critères saisis par l'utilisateur tel que la couleur ou la marque.

L'utilisateur aura accès aux fonctionnalités suivantes :

- créer un compte,
- se connecter,
- créer un vêtement,
- visualiser la liste de ses vêtements,
- visualiser le détail d'un vêtement,
- le modifier,
- le supprimer,
- 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 sept pages et deux modales. Elle est responsive afin s'adapter aux différentes tailles d'écrans (téléphone, tablette, ordinateur...).

III. Cahier des charges

Vous trouverez le cahier des charges complet en annexe A01.

A. Présentation du projet :

1. Les objectifs de l'application

Dans la version 1, il s'agit de proposer une application permettant à un utilisateur d'avoir de la visibilité sur tous les vêtements qu'il aura enregistrés. L'utilisateur doit pouvoir créer un compte, s'y connecter, ajouter des vêtements à son dressing virtuel, les visualiser, les modifier et les supprimer. Il doit également pouvoir voir la météo du jour.

Dans la version 2 du projet, l'utilisateur doit également pouvoir générer aléatoirement des tenues selon certains critères tels que les couleurs, les caractéristiques, les occasions, et sauvegarder ces tenues, en créer de nouvelles, les visualiser, les modifier, et les supprimer.

Dans la version 3 du projet, la tenue sera composée directement en fonction de la météo. Du machine learning sera ajouté pour apprendre au programme les goûts de l'utilisateur en fonction des tenues précédemment créées manuellement. De plus, un partenariat avec des sites qui vendent des vêtements pourra être mis en place (espaces publicitaires dédiés).

2. La cible

La cible est grand public, des particuliers hommes et femmes qui souhaitent déléguer le choix de leur tenue. Les personnes qui seront le plus intéressées sont les hommes et femmes actifs de 18 à 35 ans.

Personna en annexe A01.1.1.

3. Les objectifs quantitatifs

Dans les premières versions de l'application, le public visé se compose de 5 bêta-testeurs.

Dans la version finale comportant toutes les fonctionnalités, le public sera plus large car l'application sera accessible sur internet. Elle sera hébergée chez OVH, qui propose des solutions scalables et évolutives, sous le nom de domaine monddressing.net. Le nombre d'utilisateurs visé est environ 1000 personnes par jour.

Le site dans sa première version va contenir 8 pages et 2 modales : la page d'accueil, la page de création du compte, la page de connexion, la page de création d'un vêtement, la page de visualisation de tous les vêtements, la modale de visualisation d'un vêtement en particulier, la

page de modification d'un vêtement, la modale de suppression d'un vêtement, la page de météo.

B. Graphisme et ergonomie

1. La charte graphique

Pour une question de lisibilité, le fond principal est blanc. Etant donné que le public ciblé est large et pour tous les genres, nous éviterons les couleurs discriminantes (ex : rose).

- La couleur principale choisie est le RGB (77, 232, 190) ou l'hexadécimal #4DE8BE. Cette couleur doit être appliquée à la barre de navigation et à la bordure extérieure des boutons de validation et d'enregistrement.
- La police des menus est en noir (#000) dans la barre de navigation.
- La police est en noir (#000) lorsque le fond est blanc.
- La police des titres est la police Cutive Mono (Google Font, libre de droits).
- La police des textes est la police Roboto (Google Font, libre de droits).
- La taille par défaut du texte est de 20px.
- Les boutons ont tous les bords arrondis de 0.25rem en taille standard, 0.2rem en petite taille.
- Le logo est constitué des caractères : MonDressing écrits en blanc (#FFF) sur fond bleu thème principal (#4DE8BE) et écriture Srisakdi (Google Font, libre de droits). Il n'apparaît que dans la barre de navigation.
- Toutes les pages sont responsives.

2. Wireframe et Maquettage

a) Moodboard :

Moodboard en annexe A01.2.1.

b) Maquettes :

- Page d'enregistrement en annexe A01.3.1.
- Page de connexion en annexe A01.3.2.
- Page d'accueil en annexe A01.3.3.
- Page de création d'un vêtement en annexe A01.3.4.
- Page de visualisation des vêtements en annexe A01.3.5.
- Modale de visualisation d'un vêtement spécifique en annexe A01.3.6.
- Page de modification d'un vêtement en annexe A01.3.7.
- Modale de suppression d'un vêtement en annexe A01.3.8.
- Page de météo en annexe A01.3.14.
- Détail barre de navigation en annexe A01.3.15.

C. Spécificités et livrables

1. Le contenu de l'application

Pour la première version de l'application : il faut créer, 7 pages et 2 modales :

- la page d'accueil,
- la page de création du compte,

- la page de connexion,
- la page de création d'un vêtement,
- la page de visualisation de tous les vêtements,
- la modale de visualisation d'un vêtement,
- la page de modification d'un vêtement,
- la modale de suppression d'un vêtement,
- la page de météo.

Concernant la page de météo, un module existant sera importé.

Il faut également créer la logique qui permet d'enregistrer puis de récupérer les informations entrées par l'utilisateur depuis la base de données.

Les données dynamiques concernent :

- Les noms et ID de vêtements/tenues
- Les caractéristiques
- Les catégories
- Les marques
- Les couleurs
- Les occasions
- Les notes
- Les descriptions
- Les images.

Les contenus proposés sont essentiellement du texte (caractéristiques des vêtements/tenues) et des images (image par défaut ou image enregistrée par l'utilisateur).

2. Contraintes techniques

L'hébergement du site sera en local dans un premier temps.

Le site doit être responsive, car il doit pouvoir être utilisé sur des terminaux différents (téléphone, tablette, ordinateur...).

Les technologies à utiliser sont le framework Angular 7 côté front, le serveur NodeJs/Express, la base de données MySQL.

3. Les livrables

Les prestations attendues sont la conception, les développements, les tests, le déploiement et la maintenance de l'application.

Les livrables attendus sont la base de données, l'application Angular (front) et le serveur NodeJs/Express (back).

L'application Angular doit permettre à un utilisateur de créer, modifier, supprimer et visualiser des vêtements.

La partie NodeJs/Express doit recevoir et envoyer à Angular les données de la base de données.

4. Le Planning

Tâche	Durée en jours
Cahier des charges, Mockup, Persona	1,5
Dictionnaire des données	0,5
User Stories détaillées	2
Base de données	2
Front-end - Angular (13 composants)	13
Back-end - NodeJs	8
Tests Unitaires et refacto	4
Veille technologique	1
Dossier professionnel	2
Dossier projet	2
Total	36

D. Glossaire

Caractéristique	Caractéristiques d'un vêtement : moulant, ample, léger, chaud ... - un vêtement peut avoir plusieurs caractéristiques. L'utilisateur peut ajouter des caractéristiques.
Catégorie	Catégorie à laquelle appartient un vêtement : pantalon, jupe, veste, t-shirt, blouse... - un vêtement appartient à une seule catégorie. L'utilisateur peut ajouter des catégories.
Couleur	Couleur d'un vêtement - un vêtement peut avoir plusieurs couleurs. L'utilisateur peut ajouter des couleurs.
Création	Contient les formulaires permettant de créer un vêtement et de créer une tenue.
Dressing	Liste tous les vêtements et toutes les tenues.
Marque	Marque d'un vêtement - un vêtement a une seule marque. L'utilisateur peut ajouter des marques.
Météo	Permet de saisir un code postal, un pays, et ainsi d'obtenir la météo liée au lieu renseigné.
Note	Note d'un vêtement : l'utilisateur peut noter le vêtement de 1 à 5 en fonction de ses préférences. 1 est la moins bonne note, 5 est la meilleure note. L'utilisateur ne peut pas ajouter de note.
Occasion	Occasions lors desquelles le vêtement peut être porté : Soirée, Décontractée, Travail, Sport ... - un vêtement peut avoir plusieurs occasions. L'utilisateur peut ajouter des occasions.
Tenue	Une tenue est composée de deux vêtements. Elle possède un libellé.
Utilisateur	User en base de données - personne qui utilise l'application (et donc qui s'est créé un compte).

Vêtement	L'utilisateur peut enregistrer des vêtements en renseignant leurs noms, images, descriptions, caractéristiques, catégories, couleurs, marques, notes, occasions. Il pourra ensuite visualiser la liste de ses vêtements, le détail d'un vêtement, le modifier et le supprimer.
----------	--

IV. 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

A. Enregistrement

User story A.1 : Une personne souhaite s'enregistrer pour utiliser l'application
DOR : Le site MonDressing existe, le projet a été initialisé
DOD : J'accède à la page de création d'un compte
<ul style="list-style-type: none"> • En tant que personne n'ayant pas encore de compte. • J'accède au site MonDressing. • Je clique sur le bouton « S'enregistrer » dans la barre de navigation. Au clic, la couleur de fond du bouton « S'enregistrer » devient plus claire afin d'indiquer quel bouton vient d'être cliqué. • J'accède au formulaire permettant de créer un compte. • La page correspond à la maquette 1 en annexe.
Spécifications techniques
<ul style="list-style-type: none"> • Le front est fait en Angular 7. • Le back est fait en NodeJs et Express. • Le site a été préalablement initialisé avec l'outil Angular CLI. • L'URL de la page est http://localhost:4200/register. • La partie « register » est un composant Angular du module Angular « account ».

User story A.2 : Contrôle sur l'enregistrement d'une personne : Vérifier que tous les champs obligatoires sont correctement remplis
DOR : Le formulaire permettant de s'enregistrer existe
DOD : Je gère le cas des champs obligatoires sur le formulaire
<ul style="list-style-type: none"> • En tant que personne n'ayant pas encore de compte. • Lorsque je complète le formulaire d'enregistrement. • Si je clique sur le bouton « S'ENREGISTRER » alors que je n'ai pas rempli les trois champs obligatoires, celui-ci est désactivé. • Le champ « Adresse email » nécessite un @. Si j'envoie le formulaire sans @, la phrase « Une erreur est survenue. Merci de réessayer » apparaît en haut du formulaire. • Lorsque je saisis un mot de passe, le champ « Mot de passe » n'affiche pas le mot de passe en clair. • Mon mot de passe doit contenir des chiffres, des lettres, et contenir au minimum 6 caractères. Si ce n'est pas le cas, le message « Votre mot de passe doit contenir des chiffres, des lettres, et contenir au minimum 6 caractères » apparaît en dessous du champ concerné. Le bouton « S'ENREGISTRER » est désactivé tant que tout n'est pas conforme. • Si je clique en dehors de n'importe quel champ du formulaire, après avoir cliqué

dedans, et que le champ est vide, le message « Vous n'avez pas rempli ce champ » apparaît en dessous du champ concerné, surligné en rouge clair (#F8d7da). Le message reste affiché tant que les champs n'ont pas été remplis.

- Si je saisis des espaces avant ou après mon pseudo, mon login ou mon mot de passe, et/ou si je saisis un « ; », ceux-ci sont supprimés lors de l'envoi du formulaire avant d'être transmis au serveur.
- Si tout est correctement complété, l'appel au service concernant l'enregistrement des données se fait.

Spécifications techniques

- Les vérifications sont faites avec les directives structurales Angular.
- L'URL à appeler pour soumettre l'enregistrement de l'utilisateur est : <http://localhost:3000/api/users>.
- On doit soumettre les informations en POST au format JSON
- L'objet JSON est structuré ainsi :
PSEUDO_USER : string, LOGIN_USER : string, MDP_USER : string
Ces données sont obligatoires.
- Pour que les données qui transitent sur le réseau soient chiffrées, un certificat https doit être installé sur le serveur afin que le front et le back doivent communiquer via le protocole https.

User story A.3 : Contrôle sur l'enregistrement d'une personne : Vérifier que l'inscription en base a bien été effectuée

DOR : Les US A.1 et A.2 ont bien été effectuées

DOD : Je deviens un utilisateur enregistré en me servant du formulaire d'enregistrement pour créer un compte

- En tant que personne n'ayant pas encore de compte.
- J'ai complété le formulaire d'enregistrement.
- Je clique sur le bouton « S'ENREGISTRER ».
- Si une erreur survient lors de l'envoi des données pour l'enregistrement en base, la phrase « Une erreur est survenue. Merci de réessayer » apparaît en haut du formulaire.
- Lorsque l'enregistrement s'est correctement déroulé, la phrase « Votre compte a bien été créé. Merci de vous connecter » apparaît en haut du formulaire, puis je suis redirigé vers la page de connexion au bout de trois secondes.

Spécifications techniques

- Le serveur a été initialisé avec NodeJs et Express.
- La base de données MySQL nommée « dressing » a été créée et possède la table *user* contenant les champs dont le détail figure dans le dictionnaire des données ci-joint.
- La dépendance MySQL en version 2.17.1 est utilisée pour la connexion entre le serveur et la base de données.
- Le serveur renvoie le code http 400 « Bad Request » s'il y a une erreur dans la requête formulée, le code http 500 « Internal Server Error » s'il y a un problème côté serveur, et le code http 200 « OK » si tout s'est correctement déroulé.
- Des requêtes préparées doivent être utilisées afin de se protéger des injections SQL.
- La route /api/users doit être créée afin que le serveur puisse récupérer et traiter les informations que le front lui envoie.

V. Base de données

A. 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 (voir MCD MLD chapitre suivant).

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

B. Modèle Conceptuel des Données et Modèle Logique des Données

J'ai réalisé le MCD et le MLD avec le logiciel Power AMC à l'aide de la méthode MERISE apprise en cours. Vous les trouverez en annexe A03.1 et A03.2.

C. Code SQL

J'ai ensuite généré le code SQL avec Power AMC. Je l'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, et 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.

```
/* Nom de SGBD : MySQL 5.0
/* Date de creation : 31/03/2019 15:49:37
/*=====

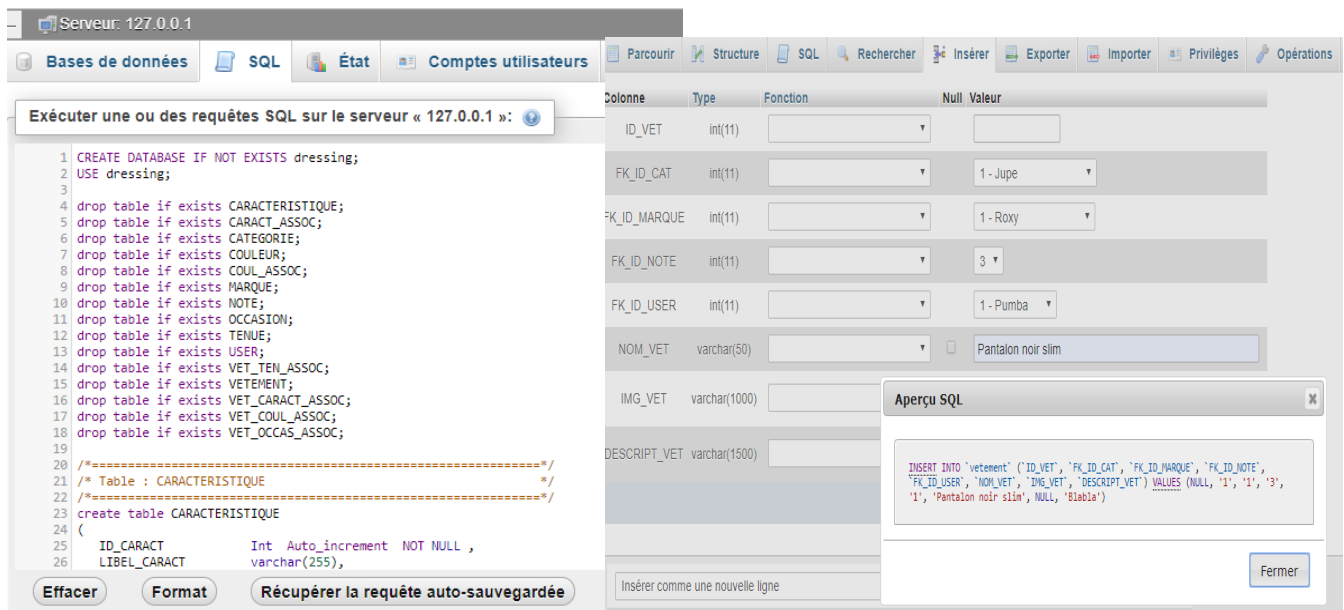
CREATE DATABASE IF NOT EXISTS dressing;
USE dressing;

drop table if exists CARACTERISTIQUE;
drop table if exists CARACT_ASSOC;
drop table if exists CATEGORIE;
drop table if exists COULEUR;
drop table if exists COUL_ASSOC;
drop table if exists MARQUE;
drop table if exists NOTE;
drop table if exists OCCASION;
drop table if exists TENUE;
drop table if exists USER;
drop table if exists VET_TEN_ASSOC;
drop table if exists VETEMENT;
drop table if exists VET_CARACT_ASSOC;
drop table if exists VET_COUL_ASSOC;
drop table if exists VET_OCCAS_ASSOC;

/*=====
/* Table : CARACTERISTIQUE
/*=====
create table CARACTERISTIQUE
(
  ID_CARACT          Int Auto_increment NOT NULL ,
  LIBEL_CARACT       varchar(255),
  primary key (ID_CARACT)
);
```

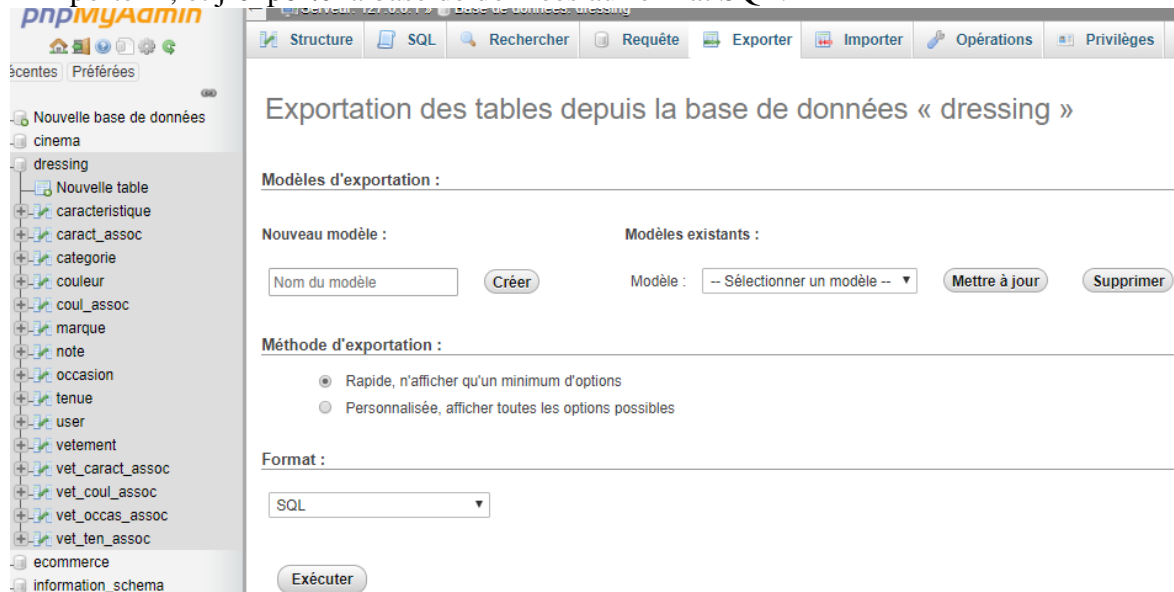
D. Mise en place

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.



J'ai réalisé une sauvegarde de la base de données contenant tous les jeux de données. Une sauvegarde mensuelle est prévue.

Pour cela, je vais dans phpMyAdmin, dans la base de données « Dressing », sous l'onglet « Exporter », et j'exporte la base de données au format SQL.



VI. Réalisations

A. 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 ».

```
// récupère la valeur saisie dans l'input pour ensuite le passer à la bdd via une fonction
newPseudo: string = "";
newLogin: string = "";
newMdp: string = "";
```

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

```
<input type="email" name="email" [(ngModel)]="newLogin" #email="ngModel"
```

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

```
onRegister(form: NgForm) {  
  if (form.valid) {  
    if (form.value["email"].search('@') === -1) {  
      this.registerNotOk = true;  
    } else {  
      let user = new Account; //création d'un objet Json contenant les données attendues par le serveur  
      user.PSEUDO_USER = form.value["pseudo"].trim().replace(/;/g, "");  
      user.LOGIN_USER = form.value["email"].trim().replace(/;/g, "");  
      user.MDP_USER = form.value["password"].trim().replace(/;/g, "");  
  
      this.service.addNewUser(user).subscribe(response => { //envoi le tableau au back  
        this.codeHttp = response.status;  
        console.log("Le user a bien été enregistré");  
        setTimeout(() => this.router.navigate(['login']), 3000);  
      },  
      error => {  
        this.codeHttp = error.status; //Récupère la réponse du serveur (codeHttp) et l'insère dans codeHttp  
        console.log(error); //Affiche le retour du serveur  
        console.log(this.codeHttp); //Affiche la variable codeHttp qui a été injectée par error.status  
        console.log(" Les requêtes n'ont pas été enregistrées / erreur lors de l'appel au service account.service - register -- " + error);  
      });  
    }  
  }  
}
```

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). Rest signifie Representational State Transfer.

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.

```
//S'enregistrer  
public addNewUser(user: Account): Observable<HttpResponse<Account[]>> {  
  return this.http.post<Account[]>(`${this.baseUrl}/users`, user, { observe: 'response' });  
}
```

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

```
var usersRouter = require('./routes/users');  
app.use('/api/users', usersRouter);
```

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.

```
//CREATE USER
router.post('/', function (req, res, next) {
  console.log("req.body ----> " + req.body);
  var contenuRequReq = JSON.stringify(req.body);
  console.log("req.body json.stringify ----> " + contenuRequReq);
  if (!req.body.LOGIN_USER || !req.body.MDP_USER) {
    res.sendStatus(400);
  }
  dbacc.createUser(req.body, function (err, data) {
    if (err) {
      res.sendStatus(500);
    }
    res.send(data);
  });
});
```

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.

```
//CREATION D'UN USER EN BASE DE DONNEES
module.exports.createUser = function (obj, fct) {

  var sql = "INSERT INTO user (PSEUDO_USER, LOGIN_USER, MDP_USER) VALUES(?, ?, SHA2(?, 256))";
  var inserts = [obj.PSEUDO_USER, obj.LOGIN_USER, obj.MDP_USER];

  // création du vêtement en base de données
  connection.query(mysql.format(sql, inserts), (err, results) => {
    if (err) {
      console.error(err);
      fct(err, null);
      return;
    }
    fct(null, results);
  });
};
```

B. 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

```

onLogin(form: NgForm) {
  let user = new Account; //création d'un objet Json contenant les données attendues par le serveur
  user.LOGIN_USER = form.value["email"].trim().replace(/;/g, "");
  user.MDP_USER = form.value["password"].trim().replace(/;/g, "");

  if (form.valid) {
    this.authService.sendUserToCompare(user).subscribe( response =>{
      console.log(response.body.token);
      this.authService.sendToken(response.body.token);
      this.router.navigate(["homepage"]);
    },
    error => {
      this.codeHttp = error.status;
      console.log(error); //Affiche le retour du serveur
      console.log(" Les requêtes n'ont pas été enregistrées / erreur lors de l'appel au service account.service - login -- " + error);
    });
  }
}

```

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.

```

//Se connecter
public sendUserToCompare(user: Account): Observable<HttpResponse<JwtToken>> {
  return this.http.post<JwtToken>(`${this.baseUrl}/users/login`, user,{ observe: 'response' });
}

```

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.readUser et un code http 500 « Internal Server Error », 403 « Forbidden » 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 ».

```

//READ USER
router.post('/login/', function (req, res, next) {
  console.log("----> call read user");
  if (!req.body.LOGIN_USER || !req.body.MDP_USER) {
    res.sendStatus(400);
    return;
  }
  dbacc.readUser(req.body, function (err, data) {
    if (err) {
      res.sendStatus(500);
      return;
    }
    console.log(data);
    if (data == 0 || data == null || data == undefined || data == "") {
      res.sendStatus(403); //forbidden
      return;
    }
    //JWT
    //déclare la charge utile = le corps de la requete
    let payload = {
      login: req.body.LOGIN_USER
    };
    //Encode la charge utile avec la secret key
    let token = jwt.encode(payload, secretKeyForJwt);
    console.log(token);
    res.status(200).send({
      token: token
    }); //renvoie 200 ok + token
  });
});

```

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.

La 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 hashés en SHA256, 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 hashes 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 une fois hashé d'une part, au mot de passe enregistré en base de données d'autre part. 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.

```
//COMPARAISON USER MDP
module.exports.readUser = function (obj, fct) {
  var mdpFromBdd = '';
  var idFromBdd = '';
  var mdpFromClt = '';

  var sql = "SELECT LOGIN_USER FROM user WHERE LOGIN_USER = ?";
  var inserts = [obj.LOGIN_USER];
  connection.query(mysql.format(sql, inserts), (err, results) => {
    if (err) {
      console.error(err);
      fct(err, null);
      connection.end();
      return;
    }

    var sql2 = "SELECT ID_USER, MDP_USER FROM user WHERE LOGIN_USER = ?";
    var inserts2 = [obj.LOGIN_USER];
    connection.query(mysql.format(sql2, inserts2), (err, results) => {
      if (err) {
        console.error(err);
        fct(err, null);
        connection.end();
        return;
      }
      // fct(null, results);
      results.forEach(element => {
        mdpFromBdd = element.MDP_USER;
        idFromBdd = element.ID_USER;
      });

      var sql3 = "SELECT SHA2(?, 256) as mdp FROM DUAL ";
      var inserts3 = [obj.MDP_USER];
      connection.query(mysql.format(sql3, inserts3), (err, results) => {
        if (err) {
          console.error(err);
          fct(err, null);
          connection.end();
          return;
        }
        results.forEach(element => {
          mdpFromClt = element.mdp;
        });

        if (mdpFromClt === mdpFromBdd) {
          console.log(idFromBdd);
          fct(null, idFromBdd);
        } else {
          fct(null, '');
          console.log(results);
        }
      });
    });
  });
}
```


C. 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.

```
//variables permettant de passer au html le contenu des listes déroulantes et cb
brands: any[] = [];
categories: any[] = [];
colors: any[] = [];
features: any[] = [];
notes: any[] = [];
occasions: any[] = [];
```

Je passe les données au front grâce aux directives structurales Angular et au binding bidirectionnel. Des valeurs par défaut sont saisies dans la base de données lors de la première connexion de l'utilisateur.

```
<select class="form-control" id="clotheBrand" name="clotheBrand" ngModel #clotheBrand="ngModel" required>
  <option value="" selected> ----- Marque ----- </option>
  <option *ngFor="let brand of this.brands" value="{{brand.ID_MARQUE}}"> {{brand.NOM_MARQUE}} </option>
</select>
```

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.

```
ngOnInit() {
  this.service.getAllBrands().subscribe(response => {
    this.brands = response.body;
    // console.log(JSON.stringify(brands));
    this.erreur = response.status;
  },
  error => {
    this.erreur = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans erreur
    console.log("Erreur lors de l'appel au service clothes.service - brands -- " + error);
  });
}
```

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.

```
//récupère la liste de toutes les marques en base de données
//on ajoute l'option {observe:'response'} afin de retourner toute la http response et pas juste le body
//pour cela, on ajoute HttpResponse à l'observable : l'observable est typé HttpResponse qui contient lui même un tableau de any
public getAllBrands(): Observable<HttpResponse<any[]>> {
  return this.http.get<any[]>(`${this.baseUrl}/brands`, { observe: 'response' });
}
```

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

```
var brandsRouter = require('./routes/brands');
var categoriesRouter = require('./routes/categories');
var clothesRouter = require('./routes/clothes');
var colorsRouter = require('./routes/colors');
var featuresRouter = require('./routes/features');
var notesRouter = require('./routes/notes');
var occasionsRouter = require('./routes/occasions');

app.use('/', indexRouter); //route quand on ne demande aucune page
app.use('/api/brands', brandsRouter);
app.use('/api/categories', categoriesRouter);
app.use('/api/clothes', clothesRouter);
app.use('/api/colors', colorsRouter);
app.use('/api/features', featuresRouter);
app.use('/api/notes', notesRouter);
app.use('/api/occasions', occasionsRouter);
```

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 :


```

<div class="form-group col-10 offset-1 col-md-4 offset-md-1 p1" *ngIf="brandExists == false">
  <div class="row">
    <div class="col-12">
      <label for="brandName">Marque&nbsp;  : </label><span> *</span>
    </div>
  </div>
  <div class="row">
    <div class="col-10 col-xl-11">
      <input type="text" class="form-control" id="brandName" placeholder="Nom de la marque" name="brandName" ngModel
        #brandName="ngModel" required [(ngModel)]="newBrand" (keyup)="fctBrandNameExists()">
    </div>
  </div>

```

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

```

onClickBrand() {
  this.brandExists = !this.brandExists;
}

```

Lorsque l'utilisateur souhaite ajouter une marque, couleur, catégorie, caractéristique ou occasion qui n'existe pas en base, cela lance une fonction (dans cet exemple la fonction on SubmitBrand() pour ajouter une marque)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

```

onSubmitBrand() {
  console.log(this.newBrand);
  let brandArray = new Clothe;
  brandArray.NOM_MARQUE = this.newBrand;
  //ajoute la marque en BDD
  this.service.addNewBrand(brandArray).subscribe(response => { //envoie le tableau au back
    this.erreur = response.status;
    console.log("La requête a bien été enregistrée");
    this.service.getAllBrands().subscribe(response => {
      this.brands = response.body;
      this.erreur = response.status;
    },
    error => {
      this.erreur = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans erreur
      console.log("Erreur lors de l'appel au service clothes.service - brands -- " + error);
    });
  },
  error => {
    this.erreur = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans erreur
    console.log(error); //Affiche le retour du serveur
    console.log(this.erreur); //Affiche la variable erreur qui a été injectée par error.status
    console.log("La requête n'a pas été enregistrée / Erreur lors de l'appel au service clothes.service - brand -- " + error);
  });
  //recharge les éléments du select

  this.brandExists = true;
}

```

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.

```

//Création d'une marque - permet d'envoyer les valeurs à enregistrer en base de données
public addNewBrand(brand: any): Observable<HttpResponse<any[]>> {
  return this.http.post<any[]>(`${this.baseUrl}/brands`, brand, { observe: 'response' });
}

```

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 (par exemple 200 « ok »).

```
//CREATE BRAND
router.post('/', function (req, res, next) {
  console.log("req.body ----> " + req.body);
  if (!req.body.NOM_MARQUE) {
    res.sendStatus(400);
    return;
  }
  dbacc.createBrand(req.body, function (err, data) {
    if (err) {
      res.sendStatus(500);
      return;
    }
    res.send(data);
  });
});
//CREATION D'UNE MARQUE EN BASE DE DONNEES
module.exports.createBrand = function (obj, fct) {

  var sql = "INSERT INTO marque (NOM_MARQUE) VALUES(?)";
  var inserts = [obj.NOM_MARQUE];

  // création du vêtement en base de données
  connection.query(mysql.format(sql, inserts), (err, results) => {
    if (err) {
      console.error(err);
      fct(err, null);
      return;
    }
    fct(null, results);
  });
});
```

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.

```
<input type="text" class="form-control" id="clotheName" placeholder="Nom du vêtement" name="clotheName"
  [(ngModel)]="newClothe" #clotheName="ngModel" required (keyup)="fctClotheNameExists()">
```

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.

```
colors: any[] = [];
features: any[] = [];
notes: any[] = [];
occasions: any[] = [];

categories: any[] = [];
colors: any[] = [];
features: any[] = [];

get selectedColors() {
  return this.colors
    .filter(color => color.checked)
    .map(color => color.ID_COUL);
}
```

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

```
<div *ngFor="let color of this.colors" class="col-4 col-lg-3">
  <div class="row">
    <label class="labelSize col-6 p-0 text-right" for="{{color.LIBEL_COUL}}">{{color.LIBEL_COUL}}&nbsp;  :
  </label>
    <input type="checkbox" class="form-control col-6 p-0" id="{{color.LIBEL_COUL}}" name="{{color.LIBEL_COUL}}"
      [(ngModel)]="color.checked" value="{{color.ID_COUL}}">
```

Au clic sur le bouton « Enregistrer » du formulaire, la fonction `onSubmit` 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
- Je vérifie que le champ contenant 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

```
onSubmit(form: NgForm) {
  if (form.valid) { //Si tous les champs du formulaire sont remplis
    let clotheArray = new Clothe; //création d'un tableau Json contenant les données attendues par le serveur
    clotheArray.NOM_VET = form.value["clotheName"]; //met le nom du vetement saisi dans le formulaire, dans le tableau clotheArray
    clotheArray.FK_ID_CAT = form.value["clotheCategory"];
    clotheArray.FK_ID_MARQUE = form.value["clotheBrand"];
    clotheArray.FK_ID_NOTE = form.value["clotheNote"];
    clotheArray.DESCRIP_VET = form.value["clotheDescr"];
    clotheArray.ID_CARACT = this.selectedFeatures;
    clotheArray.ID_COUL = this.selectedColors;
    clotheArray.ID_OCCAS = this.selectedOccasions;
    if (this.clotheImg == "" || this.clotheImg == undefined) {
      clotheArray.IMG_VET = null;
    } else {
      // let nomFichier = this.clotheImg.match('[a-zA-Z0-9_-]*\.\.*$');
      let nomFichier = this.clotheImg.replace(/^(\\|\/|:)/, '');
      clotheArray.IMG_VET = this.URL + '/' + nomFichier;
    }
    // clotheArray.FK_ID_USER = form.value["idUser"]; // a implémenter en fonction d'une session
    // console.log(clotheArray.FK_ID_USER); // a implémenter en fonction d'une session
    clotheArray.FK_ID_USER = 1; // y mettre le JWT que le back décryptera - en attendant par défaut : 1

    this.service.addNewClothe(clotheArray).subscribe(response => { //envoie le tableau au back
      this.erreur = response.status;
      console.log("Les requêtes ont bien été enregistrées");
      this.router.navigate(['homepage']);
    },
    error => {
      this.erreur = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans erreur
      console.log(error); //Affiche le retour du serveur
      console.log(this.erreur); //Affiche la variable erreur qui a été injectée par error.status
      console.log(" Les requêtes n'ont pas été enregistrées / Erreur lors de l'appel au service clothes.service - clothes -- " + error);
    });
  }
}
```

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, l'id n'est pas fourni, 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).

```
//Création d'un vêtement - permet d'envoyer les valeurs à enregistrer en base de données
public addNewClothe(clothe: Clothe): Observable<HttpResponse<any[]>> {
  return this.http.post<any[]>(`${this.baseUrl}/clothes`, clothe, { observe: 'response' });
}
```

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.

```
//CREATE CLOTHE
router.post('/', function (req, res, next) {
  console.log("req.body ----> " + req.body);
  var contenuRecuReq = JSON.stringify(req.body);
  console.log("req.body json.stringify ----> " + contenuRecuReq);
  if (!req.body.NOM_VET) {
    res.sendStatus(400);
    return;
  }
  dbacc.createClothe(req.body, function (err, data) {
    if (err) {
      res.sendStatus(500);
      return;
    }
    res.send(data);
  })
});
```

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.DESCRPT_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);
});
}

```

D. 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 ».

```

<input type="text" class="form-control" id="clotheName" placeholder="Nom du vêtement" name="clotheName"
| [(ngModel)]="clotheDetail.NOM_VET" #clotheName="ngModel" required (keyup)="fctClotheNameExists()">
clotheDetail: Clothe = new Clothe(); //récupère le contenu renvoyé par le back

```

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

```

this.service.getSpecificClothe(this.idClothe).subscribe(response => {
  this.clotheDetail = response.body;
  console.log(this.clotheDetail);
  // console.log(JSON.stringify(specificFeatureFromService));
  this.statusCode = response.status;

  //récupère le chemin de l'image
  this.urlImage = this.clotheDetail.IMG_VET;
},
error => {
  this.statusCode = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans statusCode
  console.log("Erreur lors de l'appel au service clothes.service - specificFilterOpt -- " + error);
});

```

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.

```

onSubmit(form: NgForm) {
  if (form.valid == true) { //Si tous les champs du formulaire sont remplis
    this.service.updateClothe(this.clotheDetail).subscribe(response => { //envoie le tableau au back
      this.statusCode = response.status;
      console.log("Les requêtes ont bien été enregistrées");
      this.router.navigate(['clothes-list']);
    },
    error => {
      this.statusCode = error.status; //Récupère la réponse du serveur (erreur) et l'insère dans erreur
      console.log(error); //Affiche le retour du serveur
      console.log(this.statusCode); //Affiche la variable erreur qui a été injectée par error.status
      console.log(" Les requêtes n'ont pas été enregistrées / Erreur lors de l'appel au service clothes.service - clothes -- " + error);
    });
  }
}

```

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.

```

//modification
//modification d'un vêtement + tables associatives en base de données
public updateClothe(clothe: Clothe): Observable<HttpResponse<Clothe>> {
  return this.http.put<Clothe>(`${this.baseUrl}/clothes`, clothe, { observe: 'response' });
}

```

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 », 403 « Forbidden » ou 204 « No Content » est renvoyé au front.

```
//UPDATE CLOTHE
router.put('/', function (req, res, next) {
  console.log("----> call update one clothe : " + JSON.stringify(req.body));
  if (!req.body.ID_VET) {
    res.sendStatus(400);
    return;
  }
  dbacc.updateClothe(req.body, function (err, data) {
    if (err) {
      res.sendStatus(500);
      return;
    }
    if (data !== true) {
      res.sendStatus(401);
      return;
    }
    res.sendStatus(204); //renvoie 204 car la requête ne renvoie rien. 204 est pour le front.
    return;
  })
});
```

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.

```
//MODIFICATION D'UN VETEMENT EN BASE DE DONNEES
//d'abord, modification du vêtement concerné
module.exports.updateClothe = function (obj, fct) {
  var sql1 = "UPDATE vetement SET FK_ID_CAT = ?, FK_ID_MARQUE = ?, FK_ID_NOTE = ?, NOM_VET = ?, IMG_VET = ?, DESCRIPT_VET = ? WHERE vetement.ID_VET = ?";
  var insert1 = [obj.FK_ID_CAT, obj.FK_ID_MARQUE, obj.FK_ID_NOTE, obj.NOM_VET, obj.IMG_VET, obj.DESCRPT_VET, obj.ID_VET];
  connection.query(mysql.format(sql1, insert1), (err, results) => {
    if (err) {
      console.error(err);
      fct(err, null);
      return;
    }
    console.log("Nb de lignes affectées pour la modification du vêtement : " + results.affectedRows);
    var idVet = obj.ID_VET;
  });
}
```

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

```
//si ça s'est bien passé, supprimer puis recréer les éléments dans les tables associatives
//dans la table associative : vet caract assoc
//suppression
var sql2 = "DELETE FROM vet_caract_assoc WHERE vet_caract_assoc.ID_VET = ?";
var insert2 = [idVet];
connection.query(mysql.format(sql2, insert2), (err, results) => {
  if (err) {
    console.error(err);
    fct(err, null);
    return;
  }
  console.log("Nb de caractéristiques supprimées : " + results.affectedRows);
});
//ajout
var featureArray = obj.idCaracteristiques.split(",");
featureArray.forEach(function (item) {
  var sql3 = "INSERT INTO vet_caract_assoc (ID_VET, ID_CARACT) VALUES(" + idVet + ", ?)";
  var inserts3 = [item];
  console.log("CARACTERISTIQUE =>" + inserts3);
  connection.query(mysql.format(sql3, inserts3), (err) => {
    if (err) {
      console.error(err);
      fct(err, null);
      return;
    }
  });
});
});
```


VII. 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 :

ID_VET	FK_ID_CAT	FK_ID_MARQUE	FK_ID_NOTE	FK_ID_USER	NOM_VET	IMG_VET	DESCRIPT_VET
52	5	1	5	1	Pull gris chiné femme à capuche	http://localhost:3000/api/upload/pull_roxy.png	Pull Roxy gris chiné à capuche, manches longues, s...

Dans la table *vet_caract_assoc* :

	ID_VET	ID_CARACT
primer	52	6
primer	52	4

Dans la table *vet_coul_assoc* :

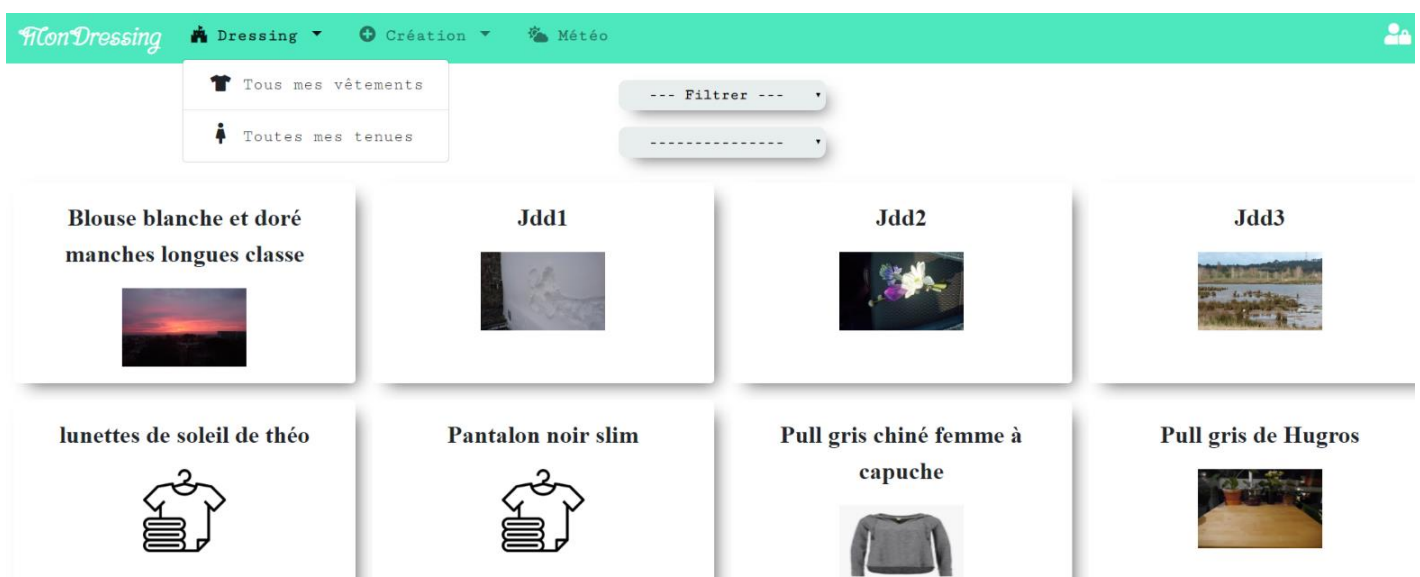
	ID_VET	ID_COUL
mer	52	12
mer	52	10
mer	52	8

Dans la table *vet_occas_assoc* :

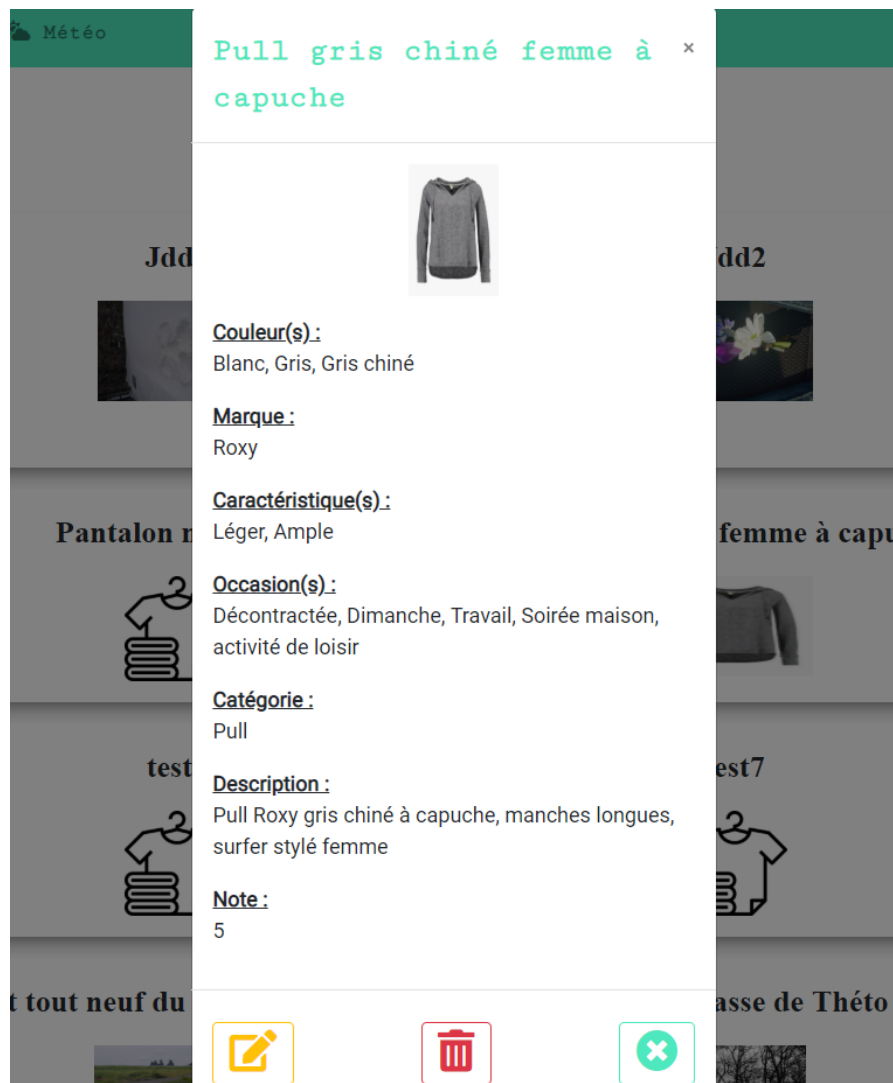
	ID_VET	ID_OCCAS
Options ← T →	52	8
Éditer Copier Supprimer	52	7
Éditer Copier Supprimer	52	4
Éditer Copier Supprimer	52	3
Éditer Copier Supprimer	52	1

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.

Modifier un vêtement

Nom : *

Marque : *

Caractéristiques : *
 Ample : ☒ Chaud : ☐ Large : ☐ Léger : ☒
 Moulant : ☐ Regular : ☐

Description : *

Catégorie : *

Couleurs : *
 Blanc : ☒ Bleu : ☐ Bleu chiné : ☐ Bleu roi : ☐
 Doré : ☐ Gris : ☒ Gris chiné : ☒ Indigo : ☐
 Jaune : ☐ Noir : ☐ Orange : ☐ Rouge : ☐
 Vert : ☐ Violet : ☐

Occasions : *
 activité de loisir : ☒ Décontractée : ☒ Dimanche : ☒
 Grandes Occasions : ☐ Soirée : ☐ Soirée maison : ☒
 Sport : ☐ Travail : ☒

Note : *

Image :

[Choisir un fichier](#) [Aucun fichier choisi](#)

[Télécharger l'image](#)



* Tous ces champs sont obligatoires. Vous pourrez les modifier ultérieurement.

[Enregistrer](#)

[Annuler](#)

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

Nom : *

Vous n'avez pas rempli ce champ

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

Nom : *

Ce nom de vêtement existe déjà

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.

Couleurs : *

Blanc : ☒ Bleu : ☒ Bleu roi : ☐ Gris : ☒
 Gris chiné : ☐ Indigo : ☐ Jaune : ☐ Noir : ☐
 Orange : ☐ Rouge : ☐ Vert : ☐ Violet : ☐

[-> Couleur inexistante dans la liste](#)

Couleur : *

Nom de la couleur +

[--> Revenir à la liste des couleurs](#)

Couleurs : *

Blanc : ☐ Bleu : ☐ Bleu chiné : ☐ Bleu roi : ☐
 Gris : ☐ Gris chiné : ☐ Indigo : ☐ Jaune : ☐
 Noir : ☐ Orange : ☐ Rouge : ☐ Vert : ☐
 Violet : ☐

-> Couleur inexistante dans la liste

L'utilisateur modifie son vêtement comme suit :

Dressing

+ Création

Météo

Modifier un vêtement

Nom : *

Sweat bleu gris à capuche

Marque : *

Roxy

-> Marque inexistant dans la liste

Caractéristiques : *

Ample : ☒

Chaud : ☐

Large : ☐

Léger : ☒

Moulant : ☐

Regular : ☐

-> Caractéristique inexistant dans la liste

Description : *

Pull bleu gris à capuche

Catégorie : *

Sweat

-> Catégorie inexistant dans la liste

Couleurs : *

Blanc : ☒

Bleu : ☒

Bleu roi : ☐

Gris : ☒

Gris chiné : ☐

Indigo : ☐

Jaune : ☐

Noir : ☐

Orange : ☐

Rouge : ☐

Vert : ☐

Violet : ☐

-> Couleur inexistant dans la liste

Occasions : *

activité de loisir : ☒

Décontractée : ☒

Dimanche : ☒

Grandes Occasions : ☐

Soirée : ☐

Soirée maison : ☒

Sport : ☐

Travail : ☐

-> Occasion inexistant dans la liste

Note : *


4

Image :

Choisir un fichier

Aucun fichier choisi

Télécharger l'image



* Tous ces champs sont obligatoires. Vous pourrez les modifier ultérieurement.

Enregistrer

Annuler

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

Sweat bleu gris à capuche



Le vêtement a bien modifié :

- Dans l'application.

Sweat bleu gris à
capuche

×



Couleur(s) :

Bleu, Blanc, Gris

Marque :

Roxy

Caractéristique(s) :

Léger, Ample

Occasion(s) :

Décontractée, Dimanche, Soirée maison, activité de loisir

Catégorie :

Sweat

Description :

Pull bleu gris à capuche

Note :

4



- En base de données.

ID_VET	FK_ID_CAT	FK_ID_MARQUE	FK_ID_NOTE	FK_ID_USER	NOM_VET	IMG_VET	DESCRIPT_VET
52	18	1	4	1	Sweat bleu gris à capuche	http://localhost:3000/api/upload/pull_roxy.png	Pull bleu gris à capuche

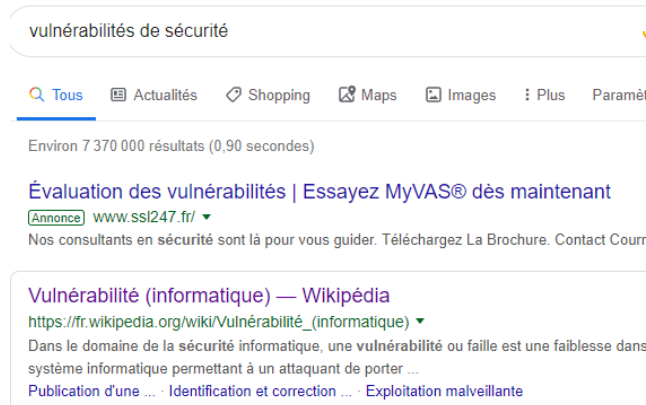
ID_VET	ID_CARACT
52	6
52	4

ID_VET	ID_COUL
52	10
52	8
52	5

ID_VET	ID_OCCAS
52	8
52	7
52	3
52	1

VIII. Veille sur les vulnérabilités de sécurité

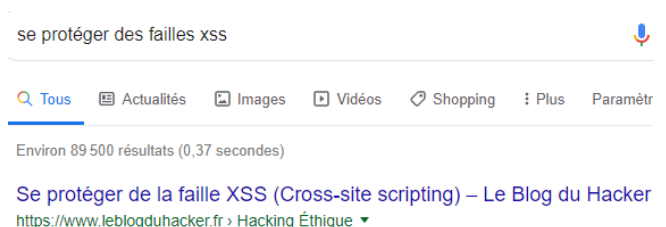
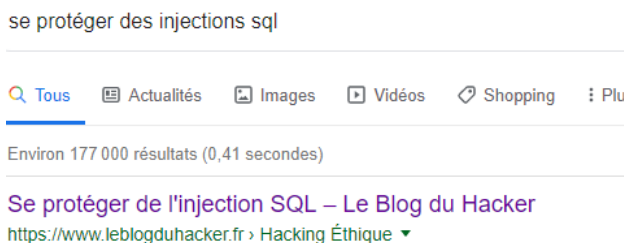
J'ai, dans un premier temps, cherché les mots clés « vulnérabilités de sécurité » sur le moteur de recherche Google.



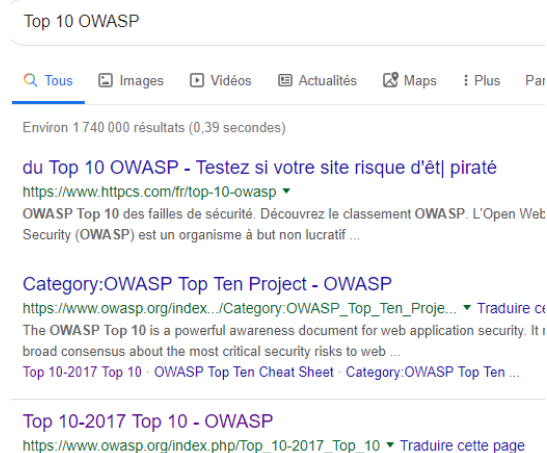
La première page était Wikipédia, j'ai commencé à m'informer grâce à cette page.
[https://fr.wikipedia.org/wiki/Vuln%C3%A9rabilit%C3%A9_\(informatique\)](https://fr.wikipedia.org/wiki/Vuln%C3%A9rabilit%C3%A9_(informatique))

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 (plus officiellement appelée Cross-Site Scripting), 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.

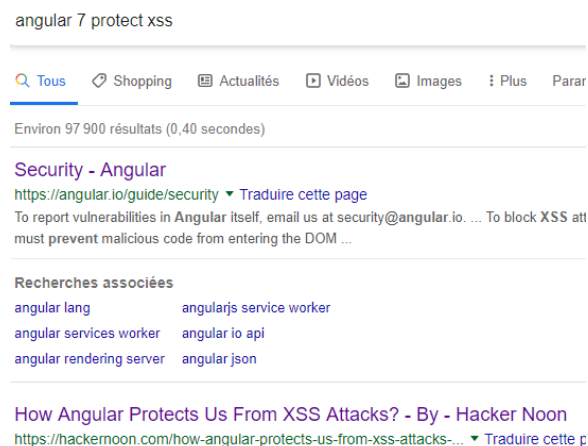


J'ai choisi le 3^e site qui est le site officiel, et qui liste le top 10 des risques de sécurité.

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 l'injection de requêtes 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 échappant 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 grâce au framework Express sur Nodejs.

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 passe). Des expressions régulières seront mises en place pour contrôler ce qui est transmis du formulaire vers le serveur.

J'ai choisi d'aller sur le premier site, stackoverflow.com, car il s'agit d'un site connu dans le monde du développement comme étant 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 sélectionnées.

```
Here's a simple way using ngModel (final Angular 2)

99 <!-- my.component.html -->
<div class="form-group">
  <label for="options">Options:</label>
  <div *ngFor="let option of options">
    <label>
      <input type="checkbox"
        name="options"
        value="{{option.value}}"
        [(ngModel)]="option.checked"/>
      {{option.name}}
    </label>
  </div>
</div>

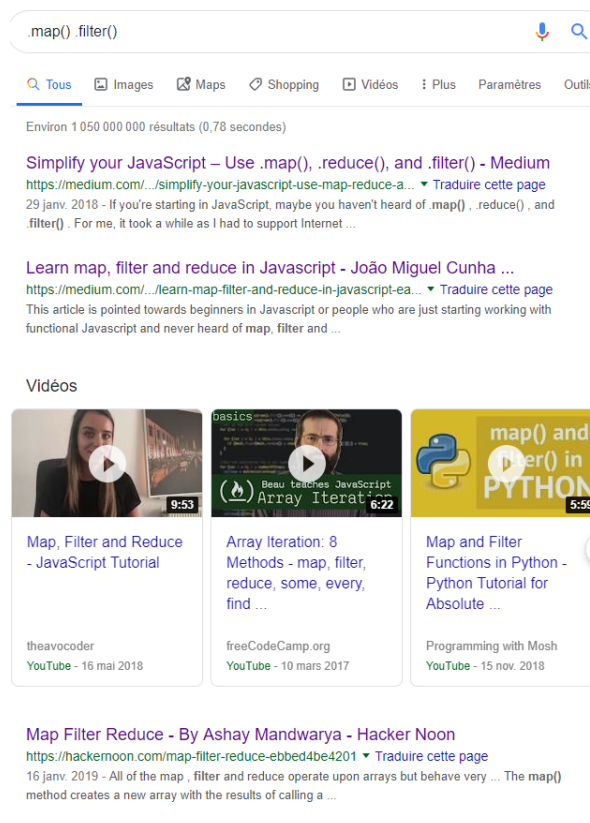
// my.component.ts

@Component({ moduleId:module.id, templateUrl:'my.component.html'})

export class MyComponent {
  options = [
    {name:'OptionA', value:'1', checked:true},
    {name:'OptionB', value:'2', checked:false},
    {name:'OptionC', value:'3', checked:true}
  ]

  get selectedOptions() { // right now: ['1','3']
    return this.options
      .filter(opt => opt.checked)
      .map(opt => opt.value)
  }
}
```

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 je le connais pour ses nombreux articles sur le monde de l'informatique notamment.

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

- Fichier Typescript :

```
// permet de récupérer les valeurs des checkboxes
// ngModel ajoute le checked sur la couleur sélectionnée
//.filter ne renvoie que les id qui sont à true (.checked)
//.map indique que la réponse ne sera constituée que des id
get selectedColors() {
    return this.colors
        .filter(color => color.checked)
        .map(color => color.ID_COUL);
}
get selectedFeatures() {
    return this.features
        .filter(feature => feature.checked)
        .map(feature => feature.ID_CARACT);
}
get selectedOccasions() {
    return this.occasions
        .filter(occasion => occasion.checked)
        .map(occasion => occasion.ID_OCCAS);
}
```


- Fichier html

```
<input type="checkbox" class="form-control col-6 p-0" id="{{color.LIBEL_COUL}}" name="{{color.LIBEL_COUL}}"
[(ngModel)]="color.checked" value="{{color.ID_COUL}}">
```


X. Extrait du site anglophone et traduction en français.

<https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d>

Code The Verge - Technol... Projet https://disneyhd.tk/... Jasminelle GIT - Ligne de com... Codes http

 | poka-techblog

Simplify your JavaScript – Use `.map()`, `.reduce()`, and `.filter()`

 Etienne Talbot in poka-techblog [Follow](#)
Jan 29, 2018 · 6 min read

If you're starting in JavaScript, maybe you haven't heard of `.map()`, `.reduce()`, and `.filter()`. For me, it took a while as I had to support Internet Explorer 8 until a couple years ago. But if you don't need to be compatible with this very old browser, you have to become familiar with those methods.

Take note that this article most likely applies to whatever other programming language you might be using, as these are concepts that exist in many other languages.

`.map()`

Let me explain how it works with a simple example. Say you have received an array containing multiple objects – each one representing a person. The thing you really need in the end, though, is an array containing only the id of each person.

```
// What you have
var officers = [
  { id: 20, name: 'Captain Piett' },
```

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 autre 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 un tableau contenant seulement l'identifiant de chaque personne.