# Présentation Orale projet MonDressing

## Présentation de l’entreprise et/ou du service et contexte du projet (cahier des charges, environnement humain et technique)

* Léa Delannay
* Fais partie de la promotion Développeur Web et Web Mobile janvier juillet 2019
* Présentation d’un projet personnel car contenu de mon stage ne permettait pas de couvrir toutes les compétences requises pour le titre
* Projet nommé MonDressing
* Choix de réaliser ce projet car c’est quelque chose que je souhaite faire évoluer dans le temps, durant mes études (intégration d’intelligence artificielle notamment – machine learning sur les habitudes et les gouts des utilisateurs)
* But :
  + 1e version : permettre à utilisateur d’avoir visibilité sur vêtements enregistrés, et les gérer (ajouter modifier supprimer visualiser)
  + 2e version : générer aléatoirement des tenues en fonction de certains critères définis par utilisateur tels que couleur ou occasion -> pas encore fait
* Cible : hommes et femmes pressés mais soucieux de leur image qui souhaitent déléguer le choix de leur tenue
* Obj quantitatifs :
  + 5 bêta-testeurs dans un premier temps. En local
  + Pas destinée à être vendue.
  + Ensuite à terme en ligne sur OVH car scalable et adaptable.
  + Environs 1000 utilisateurs.
  + 7 pages et 2 modales.
* Contenu V1 -> 7 pages et 2 modales :
  + page d’accueil,
  + page de création du compte,
  + page de connexion,
  + page de création d’un vêtement,
  + page de visualisation de tous les vêtements,
  + modale de visualisation d’un vêtement en particulier,
  + page de modification d’un vêtement,
  + modale de suppression d’un vêtement,
  + page de météo(module ext).
* Logique entre front back et bdd
* Contraintes techniques : hébergement en local dans un premier temps, site responsive.
* Fait en Angular 7, Bootstrap 4, NodeJs, Express, MySQL - VSCode, Xampp
* Livrables : conception, développements, tests, déploiement et maintenance.
* User stories et spécifications techniques correspondantes en annexe. Précisent selon le point de vue de l’utilisateur, comment l’application doit se comporter.
* Si ça vous intéresse, charte graphique, moodboard et maquettes graphiques en annexe
* Moodboard fait sur photoshop
* Maquettes graphiques faites avec Adobe XD

## Conception et codage des composants front-end et des composants back-end

* Base de données
  + Réflexion et analyse des processus de l’application souhaitée sur papier, établir objets et relations, éviter redondance des informations
  + MCD MLD en annexe, faits sur Power AMC
  + SQL généré, puis modifié ex : clés étrangères pas les mêmes que clés primaires, ajout des instructions SQL Create database if not exists, use dressing, drop table if exists caracteristiques etc pour éviter éventuelles incohérences
  + Intégré le code SQL dans l’onglet SQL de phpMyAdmin
  + Ajout de jeux de données pour tester l’appli : via phpMyAdmin et via le code SQL
* Front et back
  + Création d’une maquette html/css
* Page d’accueil
* Utilisation de bootstrap pour le côté pratique et responsive. Pas le plus joli, mais clairement le plus pratique.
* J’ai pris la maquette graphique préalablement faite
* J’ai découpé en div, ensuite j’ai modifié les div par les balises adéquates (ex = h2, button)
* Ensuite j’ai ajouté le css et les classes bootstrap
* Puis intégration de la maquette dans angular
  + Création d’un compte :
* Récupération des données saisies par utilisateur: data binding dans le composant Angular, variable dans .ts, directive ngModel dans html
* Au clic sur le bouton s’enregistrer, lance une fonction OnRegister qui vérifie @, crée objet JSON contenant données formulaire, appel au service, souscrit au retour du service, puis lors du retour du service contenant le code http retourné par le serveur, data binding et directive structurelle ngIf avec html pour affichage d’un message (erreur ou ok) à destination de l’utilisateur
* Service fait appel à ressource mise à dispo par serveur. Requete http en post car création, id inexistant au préalable, et je souhaite appliquer les principes REST (representational state transfer, basé sur http (hypertext transfer protocol), je considère les éléments sur mon serveur comme des ressources).
* Afin que données qui transitent entre le front et serveur chiffrées, installer sur serveur certificat https permettant d’utiliser protocole https, puis lier au site web.
* Sur le serveur : création de la route / point d’entrée
* Données du front envoyées à la ressource users sur serveur. Ressource joue la fonction router.post /. Fonction vérifie que données attendues sont bien dans l’objet Json envoyé par front (Si tout est ok, renvoie code http 200 OK, si NOK, envoie code 400 bad request.), puis fait appel à fonction createUser qui gère l’appel à la BDD. Si ok, renvoie code http 200 OK, si NOK, envoie code 500 Internal Server Error.
* La fonction createUser dans dbaccess/index.js fait la requete insert into sous la forme d’une requête préparée pour éviter injections sql. Si ok, renvoie code http 200 ok, si NOK, envoie code 500 Internal Server Error. Le mdp est stoché hashé en sha256.
  + Connexion :
* Récupération des données comme précédemment
* Au clic sur bouton se connecter, fonction onLogin se lange, crée objet JSON dans lequel je mets les données récupérées du formulaire pour envoi au serveur
* Appel au service en lui passant l’objet créé, abonnement au retour du service
* Lors du retour du service, récupère le token JWT généré par le back que je stocke dans le localstorage jusqu’à ce que l’utilisateur se déco
* Récupère code http retourné par serveur et renvoie utilisateur vers page accueil si code 200ok, affiche un message d’erreur si code http 500 Internal Server error ou 403 Forbidden
* Service appel a la ressource en post car je dois envoyer les infos de l’utilisateur au serveur pour pouvoir les comparer aux infos présentes en bdd
* Données envoyées à ressource users, route définie, sur router.post /login/. Cette fonction renvoie 500 internal server error si erreur, si ok vérifie que que données attendues sont bien dans l’objet (sinon 400 Bad Request), puis fait appel a fonction dans index.js readUser qui joue la requete sql
  + La 1e requete vérifie que login saisi existe bien en bdd
  + 2e requete récupère id et mdp en bdd
  + 3e requete permet de comparer le mdp transmis avec le mdp en base.
* Toutes les requêtes SQL sont préparées afin d’éviter les injections SQL
* Si NOK, renvoi code http 403 forbidden
* Si ok, serveur crypte l’id dans un token JWT (grâce à librairie JWT), renvoie code http 200ok + token.
  + Création d’un vêtement :
* Fait de la même manière que le user mais le service appelle la ressource clothes
* Possibilité de créer des couleurs (ressource colors), catégories (ressource catégories), marques (ressource brands), caractéristiques (ressource features), occasions (ressource occasions)
* Concernant les couleurs, occasions et caractéristiques relatives à un vêtement, le traitement est un peu spécial car un vêtement peut en avoir plusieurs. Dans le ts, je récupère les couleurs, caractéristiques, occasions grâce à un get sur lequel je filtre uniquement les éléments cochés et pour lesquels je ne récupère que les ID. Côté back, pour insérer ces tableaux d’éléments en base, j’itère dessus grâce à une boucle forEach.
  + Modification d’un vêtement :
* La gestion du formulaire et récupération est la même que précédemment
* Ce qui change :
* A l’affichage du formulaire, il est rempli avec les données provenant de bdd grâce au binding bidirectionnel directive ngModel. A l’initialisation de la page j’appelle le service qui fait lui-même appel à la ressource clothes en get (car c’est juste de la lecture, n’envoie pas d’objet). La ressource appelle la fonction readSpecificClothes avec l’id en paramètre. Cette fonction fait un select dans la bdd pour récupérer toutes les informations concernant le vêtement concerné. Toutes ces informations sont ensuite transmises au front dans la variable clotheDetail, liée au html grâce à ngModel
* Etant donné le binding bidirectionnel entre html et ts sur variable clotheDetail, toute modification est prise en compte dans la variable, et c’est cette même variable qui est renvoyée au back lors de la soumission du formulaire.
* Côté back, cela joue la fonction router.put car il s’agit d’une modification selon les principes REST. Cette fonction jour la fonction updateClothe qui va mettre à jour la table vetement, et supprimer le contenu des tables associatives vet\_caract\_assoc, vet\_coul\_assoc et vet\_occas\_assoc pour les reremplir avec les données transmises par le front
  + Visualisation et filtres :
* Dans le composant clothe-list, concernant les filtres,
* Je fais les appels au service pour récupérer la liste des couleurs, occasions, caractéristiques, catégories, marques, notes
* J’initialise des getters et des setters qui récupèrent grâce au binding bidirectionnel et à la directive Angular ngModel des éléments dans le html
* Le data binding fait appel à des gets et sets existants définis par Angular. Mais si je les écris, alors il fait appel aux méthodes que moi j’ai écrites.
* Ici je les ai modifiées car je souhaite que mon filtre fasse appel au service a chaque changement.
* Le premier filtre est lié au ts grâce à la directive ngModel et aux get et set selectedFilter. Les values contenues dans le select correspondent aux noms des ressources présentes sur le serveur, écrites en dur.
* Lorsqu’une valeur est choisie dans le select, je récupère le nom de la ressource (value du select) que je stocke dans la variable tempSelectedFilter
* Dans le html, j’utilise la directive ngModel pour récupérer la value du filterSelect. En fonction du filtre sélectionné, j’affiche dans un second filtre correspondant aux options, la liste des couleurs, occasions, caractéristiques, catégories, marques, ou notes. Si filterSelect.value = notes alors j’affiche les notes
* Dans le second filtre, la value correspond à l’id du filtre (ex= note.ID\_NOTE) bindé avec le fichier TS qui récupère la liste des notes. Lorsque je choisis cela, je récupère la value donc l’id, que je stocke dans la variable tempSelectedOption.
* Le get et le set de selectedOption est bindé au html. Lorsque je fais un changement, cela fait appel au service la fonction getSpecificFilterOpt en lui passant en paramètre le contenu de la variable tempSelectedFilter (nom de la ressource) et le contenu de la variable tempSelectedOption (id de l’option choisie)
* Le service se charge de transmettre la demande en get au serveur avec dans l’url la ressource sélectionnée et l’id sélectionné.

**return** this.http.get<**any**[]>(`${this.baseUrl}/${selectedFilter}/${selectedOption}`, { observe: 'response' });

* Le back reçoit la demande, joue la requête, et renvoie les données de la base comme précédemment. S’il n’y a pas de données concernant la requête, il renvoie un code http 204 No Content, que le front utilise pour afficher un message précisant qu’il n’y a pas de vêtement correspondant à la demande, sinon il affiche la liste des vêtements correspondant au filtre sélectionné
  + Suppression d’un vêtement :
* Dans la modale de détail d’un vêtement, se trouve un bouton permettant à l’utilisateur de supprimer le vêtement.
* Au clic sur ce bouton, j’affiche une modale de confirmation demandant à l’utilisateur s’il est sur de vouloir supprimer le vêtement.
* La modale est faite grâce au module ng-bootstrap d’angular
* Lorsqu’il confirme, je fais appel au service via la fonction deleteClothe() et je m’abonne au retour du service. Dès que j’ai un retour, la modale se ferme et je retourne sur la page de liste des vêtements
* Le service fait appel au back en lui passant l’id du vêtement dans l’url, avec le verbe http delete car il s’agit d’une suppression.
* Le back recoit la demande, joue la fonction deleteClothe qui va faire les requetes DELETE from nomtable where id = ?
* Le back supprime d’abord le contenu des tables associatives concernant les couleurs, caractéristiques et occasions pour ensuite pouvoir supprimer le vêtement et renvoie un code 204 quand tout s’est bien passé.

## Présentation des éléments les plus significatifs de l’interface de l’application

* Créer un compte
* Se connecter
* Ajouter un vêtement
* Liste des vêtements
* Modale détail
* Suppression
* Modification
* Vêtement modifié

## Présentation du jeu d’essai de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues) et analyse des écarts éventuels

* Présenter la modification d’un vêtement
  + Cas passant sur postman code 204 no content normal
* Présenter cas non passant sur postman :
  + Enlever la ligne ID\_VET dans postman et test postman : err 400 Bad Request
  + Remettre l’id
  + Modifier dans index.js fonction updateClothe : UPDATE vet SET … : erreur 500 Internal Server Error
  + Remettre nom de table vetement correctement

## Présentation d’un exemple de recherche effectuée à partir de site anglophone

* Je ne savais pas comment récupérer la valeur de cases à cocher sélectionnées
* Tapé mots clés « get values multiple checked checkboxes angular » sur le moteur de recherche Google
* premier site choisi, 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
* trouvé les méthodes .filter.map me permettant de faire ce que je voulais
* cherché sur google comment fonctionnent les méthodes « .map() » et « .filter() »
* .filter() permet de créer et retourner un nouveau tableau contenant uniquement les cases checkées
* .map() permet de créer un tableau qui ne contient que les id correspondant à chaque case checkée

## Sécurité

* Requêtes préparées
* Certificat https prévu sur le serveur
* Guards angular (qui interdisent l’accès à une page tant que l’utilisateur n’est pas connecté avec un token JWT valide).
* Authentification
* Cryptage mdp sha256
* Token jwt
* Angular gère failles XSS

## Synthèse et conclusion (satisfactions et difficultés rencontrées)

* Difficultés
  + Manque de temps pour tout réaliser entre les cours à bosser et l’application à réaliser en parallèle + la partie administrative qui n’est pas la plus intéressante mais néanmoins nécessaire
  + Au début, fautes d’inattention très frustrant, ça va mieux maintenant car rigueur dans la façon de développer
  + Frustration de chercher et mettre longtemps à implémenter des solutions, qui à d’autres plus expérimentés, prennent très peu de temps. Car manque de pratique et manque de compétences. Apprendre la patience, ça va venir.
  + Intégrer et adapter les méthodes vues en cours prends du temps.
  + Récupérer les valeurs des checkboxes, trouver comment fonctionnent les directives sur les formulaires Angular, l’upload d’images, back mise en place pour connexion.query(mysql…) car on avait fait avec postgres mais c’est différent, mise en place des modales en leur passant l’id, requetes SQL complexe permettant de récupérer un vêtement (la partie récupérant plusieurs couleurs, occasions, caractéristiques) avec group concat
* Satisfactions
  + Plaisir de faire une application fonctionnelle, entièrement seule. Pas dingue, prend du temps, mais fierté d’avoir créé quelque chose qui fonctionne.
  + Voir mon évolution, au départ je galérais à apprendre les boucles quand j’ai commencé avec les cours en ligne, et maintenant je bute sur des choses plus difficiles. Et plus j’avance, plus j’apprends des choses et plus j’arrive à les appliquer facilement, c’est très gratifiant.
  + Chercher longtemps et finir par trouver la solution, l’implémenter et tout fonctionne, très satisfaisant
  + Niveau pro, satisfaction d’avoir intégré une super équipe qui m’a accompagnée et dans laquelle je m’épanouis, pouvoir travailler sur un projet commun dans le but de la satisfaction client, selon méthodes agiles/scrum, au top
  + Partager ce qu’on sait : mon équipe m’apprend beaucoup, mais il y a aussi des petites choses que je sais et que je peux partager avec eux (par ex le provideIn :root d’Angular sur les récents, dans le component, qui évite d’avoir à provider manuellement le fichier dans le module)
  + Plaisir de résoudre des problèmes au quotidien, d’aider l’équipe à avancer, constater les progrès en temps réel