

08/01/2019

Conception et réalisation d'un site web responsable Angular

Michel HUANG – Marwan ALWAN





Table des matières

Table des figures.....	4
1. Introduction générale :	7
2. Présentation du sujet :	7
2.1. Contexte	7
2.2. Les objectifs	7
3. Analyse et spécifications des besoins :.....	8
3.1. Introduction.....	8
3.2. Diagramme de cas d'utilisation :	8
3.3. Diagramme d'activité	10
3.3.1. Diagramme d'activité d'inscription	10
3.3.2. Diagramme d'activité d'un client	11
3.3.3. Diagramme d'activité d'un salarié.....	12
3.3.4. Diagramme d'activité d'un administrateur	13
3.4. Diagramme de séquence.....	14
3.4.1. Diagramme de séquence d'inscription.....	14
3.4.2. Diagramme de séquence d'un client pour un achat	15
3.4.3. Diagramme de séquence d'un client pour visualiser les commandes passées	16
3.4.4. Diagramme de séquence d'un client pour modifier ses informations personnelles	17
3.4.5. Diagramme de séquence d'un salarié	18
3.4.6. Diagramme de séquence d'un administrateur qui créer un utilisateur	19
3.5. Diagramme de séquence d'un administrateur qui gère les utilisateurs	19
3.6. Diagramme de collaboration.....	20
3.6.1. Diagramme de collaboration d'un client	20
3.6.2. Diagramme de collaboration d'un client qui visualise les commandes	21
3.6.3. Diagramme de collaboration d'un salarié qui ajoute un produit.....	22
3.6.4. Diagramme de collaboration d'un salarié qui modifie un produit.....	23
3.6.5. Diagramme de collaboration d'un salarié qui supprime un produit.....	24
3.7. Diagramme de classes	25
4. Spécification des techniques	29
4.1. SPRING	29
4.1.1. INTRODUCTION	29
4.1.2. LES MODULES DE SPRING.....	29
4.1.3. CORE SPRING CONTAINER.....	30
4.1.4. LE MODULE AOP	31

4.1.5.	LE MODULE DATA ACCESS AND INTEGRATION	31
4.1.6.	LE MODULE WEB	31
4.1.7.	MESSAGING	32
4.1.8.	TESTS.....	32
4.1.9.	CONFIGURATION DE L'ARCHITECTURE.....	33
4.2.	SPRING BOOT.....	34
4.2.1.	INTRODUCTION	34
4.2.2.	QU'EST-CE QUE MICRO SERVICE ?.....	34
4.2.3.	QU'EST-CE QUE SPRING BOOT ?.....	35
4.2.4.	SPRING BOOT STARTERS.....	36
4.3.	LE MODELE MVC 1 ET 2 (MODEL VIEW CONTROLLER)	38
4.3.1.	LE MODELE (MODEL).....	38
4.3.2.	LA VUE (VIEW)	38
4.3.3.	LE CONTROLEUR (CONTROLLER)	38
4.4.	SPRING SECURITY	40
4.4.1.	TYPES D'AUTHENTIFICATION PRIS EN CHARGE	40
4.4.2.	QUE SONT L'AUTHENTIFICATION ET L'AUTORISATION ?	41
4.4.3.	USERDETAILS ET USERDETAILSSERVICE.....	42
4.4.4.	SECURITY DEPENDANCES	42
4.5.	ANGULAR.....	43
4.5.1.	ARCHITECTURE ANGULARJS	43
4.5.2.	ARCHITECTURE D'ANGULAR.....	45
5.	Les outils utilisés.....	48
5.1.	Spring Tool Suite (STS).....	48
5.2.	WampServer	49
5.3.	NodeJs	51
5.4.	DBeaver	51
6.	Application des technologies d'hibernate.....	53
6.1.	Les annotations	53
6.2.	@Controller	53
6.3.	@RestController	54
6.4.	@RequestMapping.....	54
6.5.	@Service	54
6.6.	@Repository.....	54
6.7.	@Component	54
6.8.	@Autowired	54

6.9. @Configuration	54
7. Réalisations.....	55
7.1. Architecture du projet.....	55
7.2. Back-end.....	55
7.2.1. Structure du projet Spring.....	55
7.2.2. Structure du package src/main/java	56
7.2.3. POM.xml	66
7.3. Front-end.....	67
7.3.1. Structure du projet Angular.....	67
8. Problèmes rencontrés	118
9. Amélioration.....	118
10. Conclusion	119

Table des figures

Figure 1: Diagramme de cas d'utilisation	8
Figure 2:Diagramme d'activité d'inscription	10
Figure 3: Diagramme d'activité d'un client.....	11
Figure 4: Diagramme d'activité d'un salarié.....	12
Figure 5: Diagramme d'activité d'un administrateur	13
Figure 6: Diagramme de séquence d'inscription	14
Figure 7: Diagramme de séquence d'inscription avec fonction	15
Figure 8: Diagramme de séquence d'un client pour un achat	15
Figure 9: Diagramme de séquence d'un client pour un achat avec la fonction	16
Figure 10: Diagramme de séquence pour visualiser les commandes passées.....	16
Figure 11: Diagramme de séquence pour visualiser les commandes passées avec la fonction	17
Figure 12: Diagramme de séquence d'un client pour modifier ses informations personnelles	17
Figure 13:Diagramme de séquence d'un client pour modifier ses informations personnelles avec la fonction	18
Figure 14: Diagramme de séquence d'un salarié pour des fonctions CRUD	18
Figure 15: Diagramme de séquence d'un administrateur qui créer un utilisateur.....	19
Figure 16: Diagramme de séquence d'un administrateur qui gère les utilisateurs	19
Figure 17: Diagramme de collaboration d'un client	20
Figure 18: Diagramme de collaboration d'un client qui visualise les commandes	21
Figure 19:Diagramme de collaboration d'un salarié qui ajoute un produit.....	22
Figure 20: Diagramme de collaboration d'un salarié qui modifie un produit	23
Figure 21: Diagramme de collaboration d'un salarié qui supprime un produit	24
Figure 22: Diagramme de classe.....	25
Figure 23: Classement de Spring	30
Figure 24: Core Container	30
Figure 25: Module AOP	31

Figure 26: Data Access Integration.....	31
Figure 27: Module Web.....	32
Figure 28: Messaging.....	32
Figure 29: Tests.....	32
Figure 30: Design pattern de la programmation par template	33
Figure 31: Design pattern de l'inversion de contrôle	34
Figure 32: Schéma MVC.....	39
Figure 33: Flux de travail de traitement des requêtes du Dispatcher Servlet Spring Web MVC	39
Figure 34: Design pattern MVC	43
Figure 35: Structure du module	45
Figure 36: app.module.ts.....	45
Figure 37: Site pour le téléchargement de STS	48
Figure 38: Répertoire de STS	48
Figure 39: Architecture du projet.....	55
Figure 40: Structure d'un projet Spring	55
Figure 41: Structure du package src/main/java	56
Figure 42: Structure du package fr.utbm.ecommerce	56
Figure 43: Code source de ECommerceAppliation.java	56
Figure 44: Structure du package fr.utbm.ecommerce.config	57
Figure 45 : WebConfig.java :	57
Figure 46: Structure du package fr.utbm.ecommerce.controller	58
Figure 47: UserController.java.....	58
Figure 48: Structure du package fr.utbm.ecommerce.dto.....	59
Figure 49: Classe Payment.java	60
Figure 50: Structure du package fr.utbm.ecommerce.iservice	61
Figure 51: Interface IUserservice.java	62
Figure 52: Structure du package fr.utbm.ecommerce.repository	62
Figure 53: UserDao.java	63
Figure 54: Structure du package fr.utbm.ecommerce.service.....	64
Figure 55: UserService.java	65
Figure 56: Structure du package fr.utbm.ecommerce.util	66
Figure 57: pom.xml.....	66
Figure 58: Structure du projet Angular	67
Figure 59: Structure du dossier SRC	67
Figure 60: Structure du dossier app	68
Figure 61: Structure du dossier components	69
Figure 62: Code source HTML pour le composant category	70
Figure 63: Liste de catégories.....	70
Figure 64: Modifier une catégorie.....	71
Figure 65: Ajout d'une catégorie	71
Figure 66: TS du composant categorie	74
Figure 67: TS du composant footer	77
Figure 68: Page html footer.....	77
Figure 69: Code source page home	77
Figure 70: TS du composant home	78
Figure 71: Page html home	78
Figure 72: Code source de la page HTML liste des produits	79

Figure 73: TS du composant list-products.....	79
Figure 74: Page HTML liste des produits	80
Figure 75: Page html login.....	80
Figure 76: Code source de la page html login	81
Figure 77: TS du composant login	81
Figure 78: Page html du navbar	82
Figure 79: Code source de la page html navbar.....	82
Figure 80: TS du composant navbar	84
Figure 81: Page HTML de la liste des orders	85
Figure 82: Page html du détail de la commande.....	85
Figure 83: Code source de la page html orders.....	86
Figure 84: TS du composant orders.....	87
Figure 85: Page html détail d'un produit	87
Figure 86: TS du composant product	88
Figure 87: Code source de la page détail du produit	89
Figure 88: Page html pour un exemple de produit	89
Figure 89: TS du composant product-cart.....	90
Figure 90: Code source pour un produit	90
Figure 91: Page html d'une liste de produit	90
Figure 92: Formulaire d'ajout d'un produit.....	91
Figure 93: Formulaire d'édition d'un produit	91
Figure 94: TS du composant products.....	93
Figure 95: TS du composant products.....	95
Figure 96: Page html du profil d'un utilisateur.....	96
Figure 97: TS du composant profile.....	97
Figure 98: Code source de la page profile	100
Figure 99: Page html register	100
Figure 100: Code source de la page register	101
Figure 101: TS du composant register.....	102
Figure 102: Page html du panier	103
Figure 103: Code source de la page panier	103
Figure 104: TS du composant shoppingcart	106
Figure 105: Page html Liste de fournisseur	106
Figure 106: Page HTML ajout d'un fournisseur	107
Figure 107 : Page html modifier un fournisseur.....	107
Figure 108: TS du composant supplier	110
Figure 109: Code source de la page HTML suppliers.....	111
Figure 110: Page HTML liste des utilisateurs.....	112
Figure 111: Code source de la page HTML Liste des utilisateurs	114
Figure 112: TS du composant Users	116
Figure 113: Structure du dossier model	116
Figure 114: Model Category	116
Figure 115: Structure du dossier services	117
Figure 116: Code source du CategoryService.....	118

1. Introduction générale :

Dans le cadre de l'UV TO52 (Projet de développement), nous devons réaliser un projet sur la conception et la réalisation d'un site web responsable AngularJS.

Nous avons donc pensé de réaliser un site e-commerce pour le projet de développement.

2. Présentation du sujet :

2.1. Contexte

Nous allons réaliser une application d'e-commerce pour répondre au projet qu'on va réaliser.

Hypothèse :

- Pour percevoir ou vendre un produit, il faudrait concevoir une plateforme web pour que nous pouvons mettre des produits et la quantité.
- De s'inscrire pour devenir client pour avoir une historique d'achat, ou autre.
- Concevoir un site qui permet de rechercher des produits avec leurs informations comme le fournisseur, le prix, la description, l'image et autre.

2.2. Les objectifs

Les objectifs du site sont les suivantes :

- Possibilité de s'inscrire pour devenir client
- Ajout des produits dans le panier
- Percevoir le produit avec les détails et le prix
- Confirmation de la commande
- Le paiement en ligne
- Faire des recherches par filtre
- Ajout des produits dans le catalogue
- Possibilité de faire la mise à jour des produits (quantité, détails, image, ...)

3. Analyse et spécifications des besoins :

3.1. Introduction

Dans cette partie, nous allons analyser le système en définissant différents diagramme UML (Unified Modeling Language ou Langage de Modélisation Unifié en français est un langage de modélisation graphique normalisé pour concevoir un système).

3.2. Diagramme de cas d'utilisation :

Un diagramme de cas d'utilisation représente des possibilités d'interaction entre le système et les acteurs, c'est-à-dire les différentes fonctionnalités que doit fournir le système.

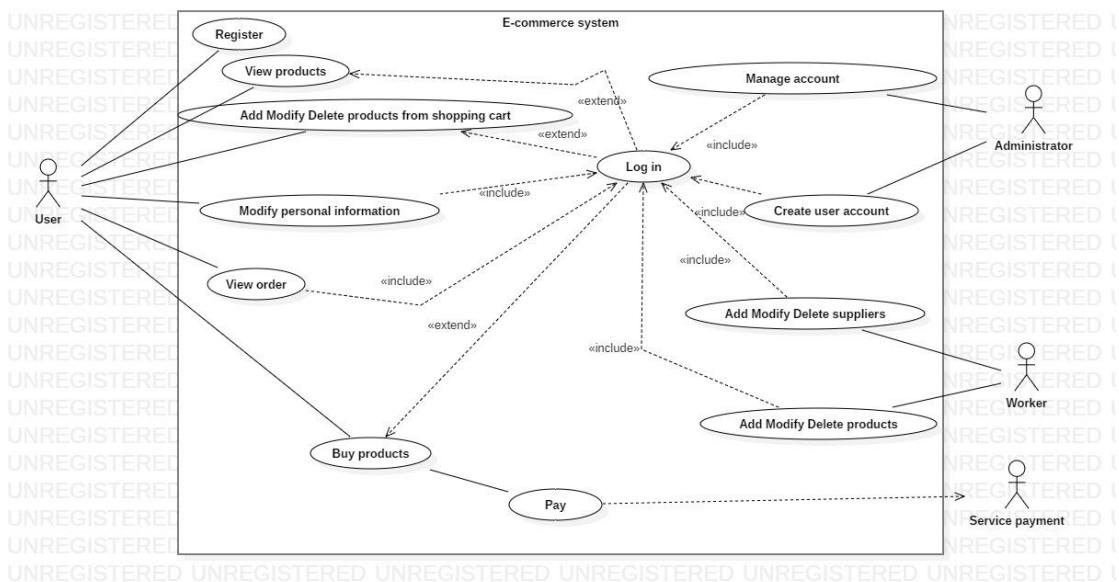


Figure 1: Diagramme de cas d'utilisation

Il y a 3 acteurs principaux pour le site d'e-commerce :

- Un utilisateur (User) : C'est un individu qui pourrait regarder le catalogue des produits, voir les prix ext ..., à ce stade il est pas encore un client.
- Un administrateur (Administrator) : C'est une personne qui permet d'assurer le dynamisme du site et de manager les utilisateurs du site.
- Un travailleur (Worker) : C'est une personne qui pourrait d'assurer et de faire les mises jours des produits, de leurs prix et de leurs disponibilités.

Un utilisateur à plusieurs scénarios :

- Se créer un compte (Register)

L'utilisateur pourra se créer un compte s'il n'y a pas encore créé.

- Visualiser le catalogue (View products)

L'utilisateur pourra fouiller dans le catalogue les différents produits par catégories sans avoir besoin de se connecter à son compte.

- Ajouter, modifier, supprimer des produits dans le panier (Add Modify Delete products from shopping cart)

L'utilisateur pourra ajouter, modifier ou supprimer des produits depuis son panier et de passer sa commande en tant en invité pour finaliser.

- Modifier ces données personnel (Modify personal information)

L'utilisateur pourra modifier ces données personnelles s'il y veut mais il faut d'abord se connecter à son compte pour modifier.

- Revoir les commandes (View order)

L'utilisateur pourra revoir ces commandes qui a passé ou les différents statuts de la commande mais il faudrait se connecter avant.

- Passer la commande (Buy products)

L'utilisateur pourra passer une commande sans se connecter mais il aura un statut invité.

L'administrateur à 2 scénarios :

- Créer un utilisateur (client ou salarié) (Create user account)

L'administrateur pourra créer des comptes pour des clients ou/et pour des salariés.

- Manager les comptes (Manage account)

L'administrateur pourra manager les comptes qui existent (modifier les informations, supprimer les comptes).

Le salarié à un scénario :

- Ajouter, modifier, supprimer des produits dans la base de données (Add Modify Delete products)

Le salarié pourra ajouter, modifier et supprimer des produits dans le catalogue.

- Ajouter, modifier, supprimer des fournisseurs.

Le salarié pourra ajouter, modifier et supprimer des fournisseurs dans la base de données.

3.3. Diagramme d'activité

Un diagramme d'activité représente le déroulement des actions, sans utiliser les objets. Il permet de représenter graphiquement le déroulement d'un cas d'utilisation.

3.3.1. Diagramme d'activité d'inscription

La phase d'inscription est de passer d'un simple visiteur du site qui n'a que le droit de consulter les produits et les détails alors qu'un client peut directement acheter ses produits et payer en ligne.

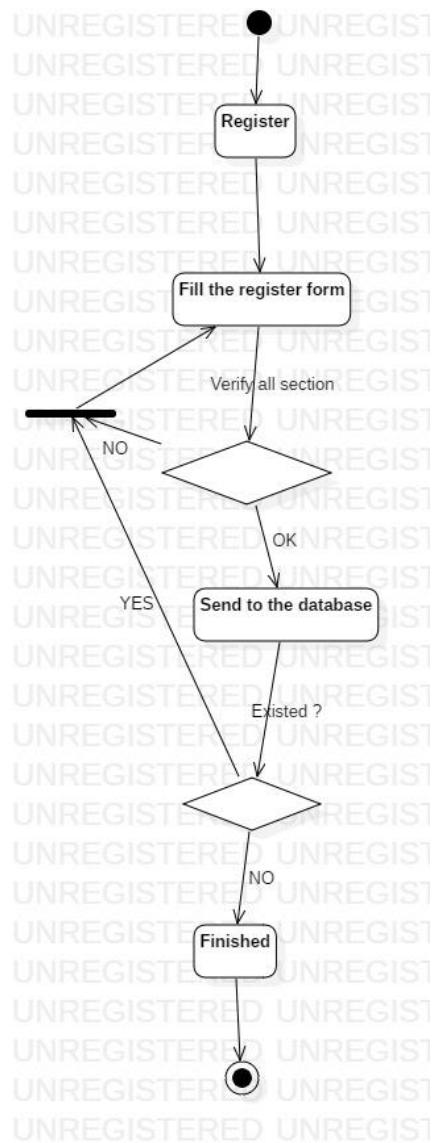


Figure 2: Diagramme d'activité d'inscription

3.3.2. Diagramme d'activité d'un client

L'activité d'un client se conduit à différents fonctionnements, d'abord il doit se connecter sinon il a que la possibilité de consulter le catalogue et les détails des produits mais bien sûr il peut aussi ajouter des produits dans le panier sans se connecter en tant que client.

Le client peut donc regarder modifier ses informations personnelles, visualiser ses commandes qui a passé, passer une commande et payer.

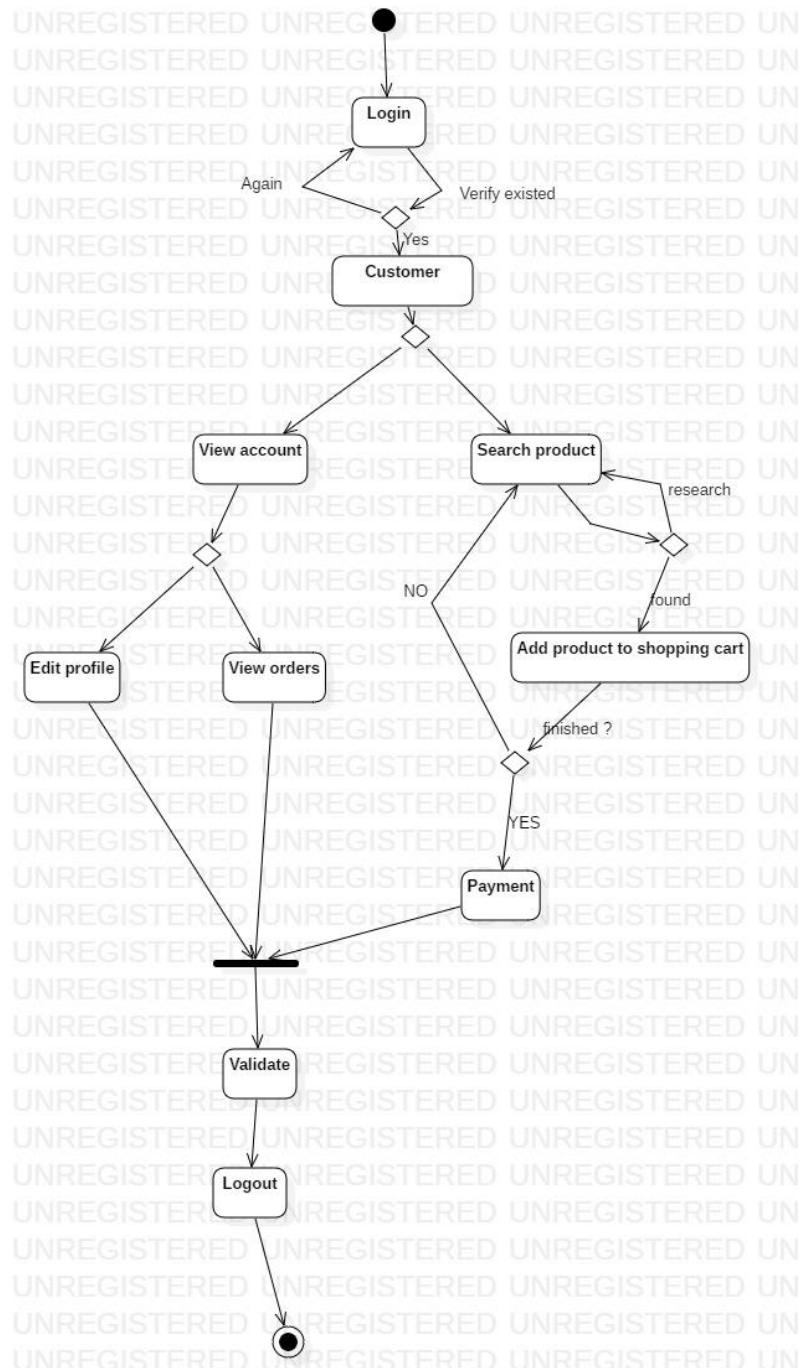


Figure 3: Diagramme d'activité d'un client

3.3.3. Diagramme d'activité d'un salarié

L'activité d'un salarié doit se connecter avant d'effectuer ce qu'il veut.

Il a la possibilité d'ajouter, modifier et supprimer des produits, des catégories de produits et de fournisseurs. Mais il a aussi la possibilité de visualiser le catalogue de produit.

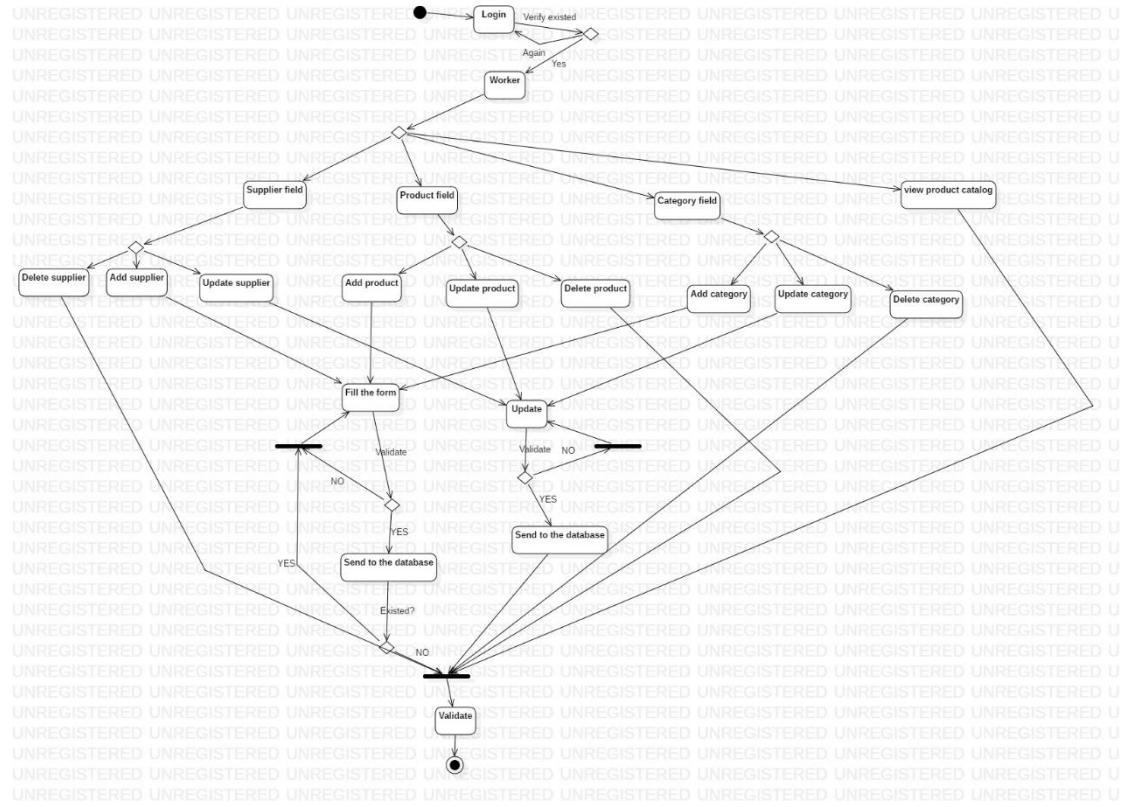


Figure 4: Diagramme d'activité d'un salarié

3.3.4. Diagramme d'activité d'un administrateur

L'activité d'un administrateur, il doit se connecter pour avoir accès à ces fonctionnalités. L'administrateur n'a que le droit de créer, modifier et supprimer des utilisateurs.

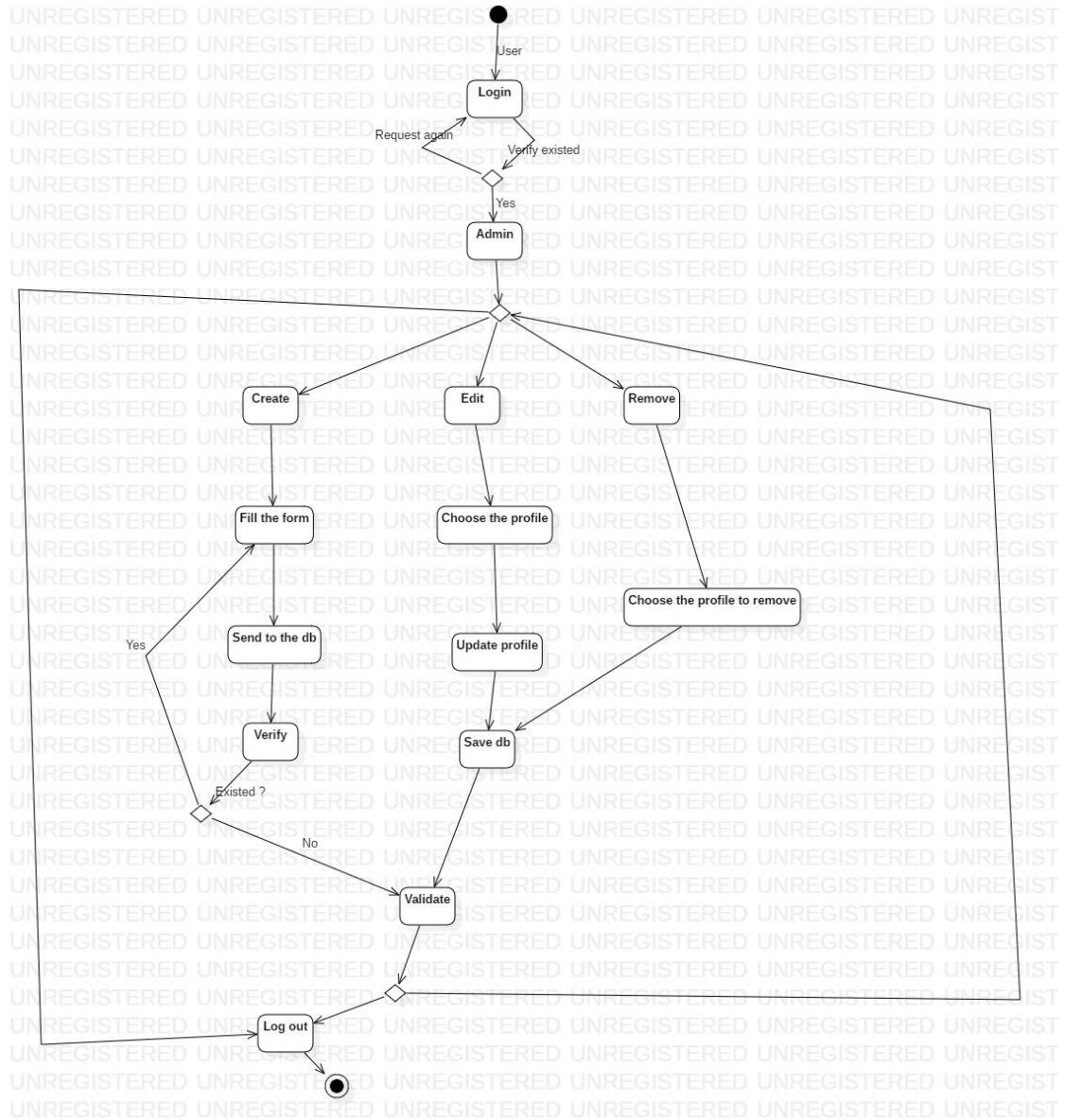


Figure 5: Diagramme d'activité d'un administrateur

3.4. Diagramme de séquence

Un diagramme de séquence est un diagramme d'interaction qui permet de décrire les différents scénarios d'utilisation du système.

3.4.1. Diagramme de séquence d'inscription

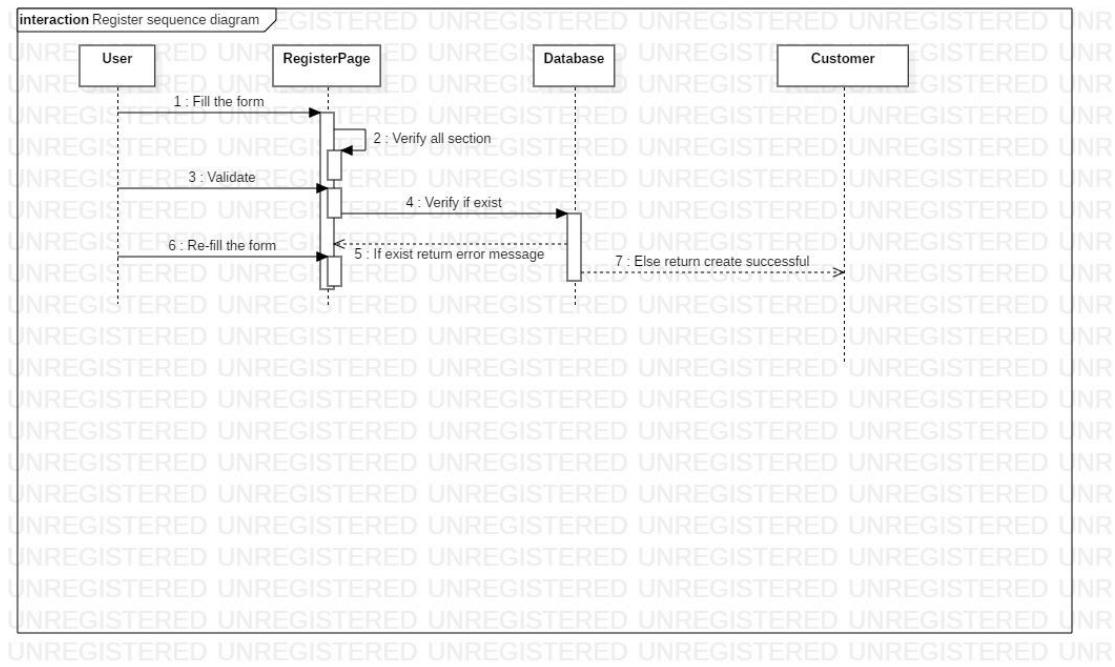


Figure 6: Diagramme de séquence d'inscription

Pour créer un compte l'utilisateur va remplir un formulaire et le système va vérifier si les données saisies par l'utilisateur sont-ils cohérents et vérifie s'il n'existe pas dans la base de données. Une fois cette vérification terminée avec succès l'utilisateur à créer son compte.

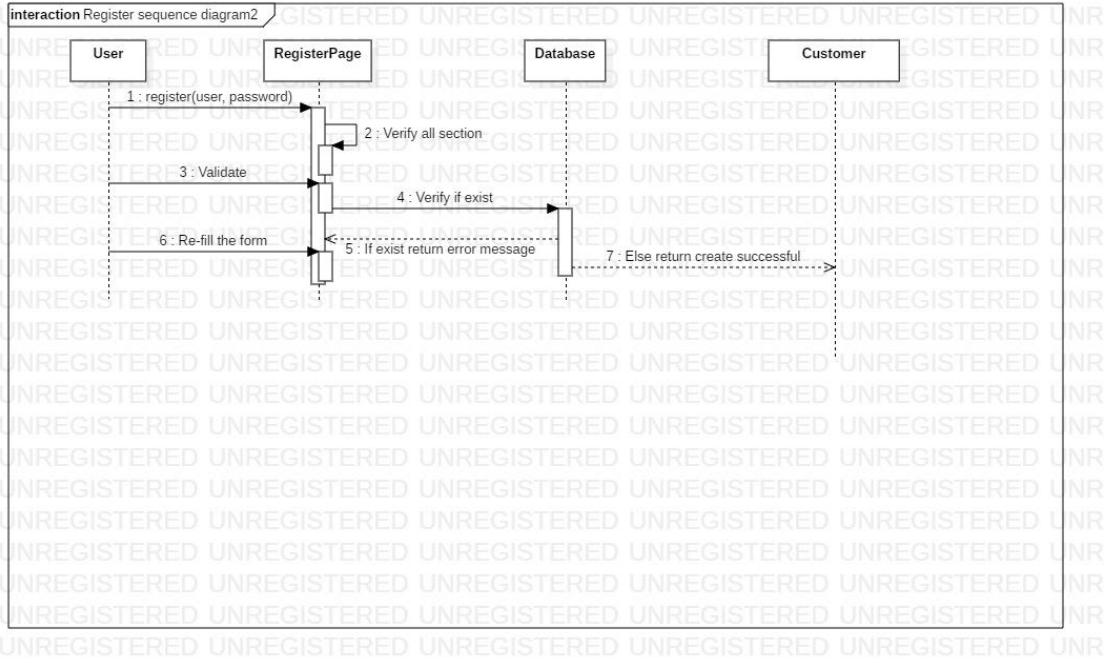


Figure 7: Diagramme de séquence d'inscription avec fonction

3.4.2. Diagramme de séquence d'un client pour un achat

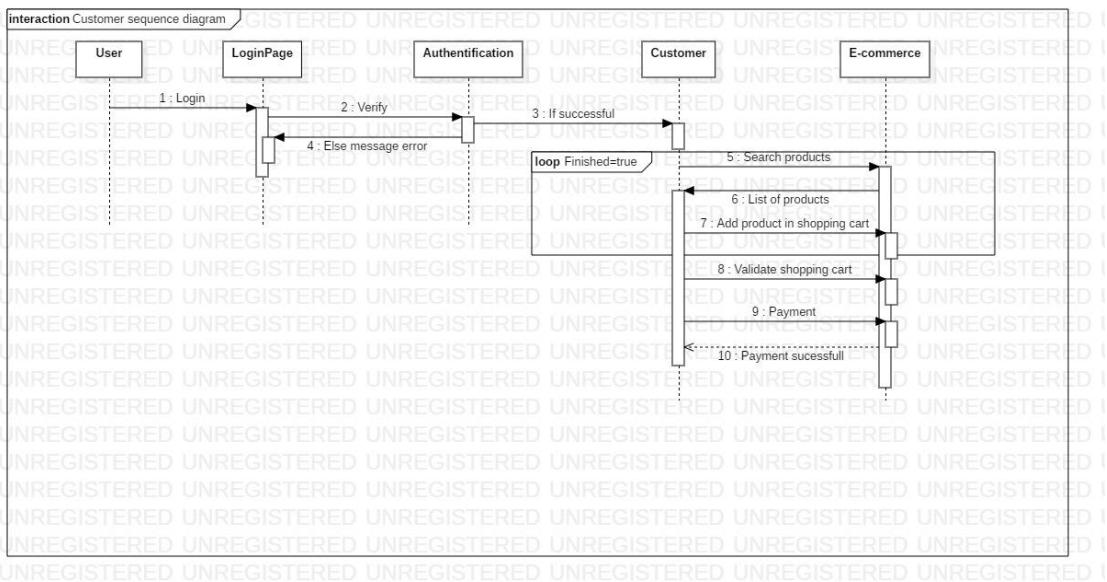


Figure 8: Diagramme de séquence d'un client pour un achat

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur à bien fournis des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode client et il pourra donc visualiser les catalogues de produits, choisir des produits et le mettre dans le panier. Une fois le panier terminer le client pourra payer.

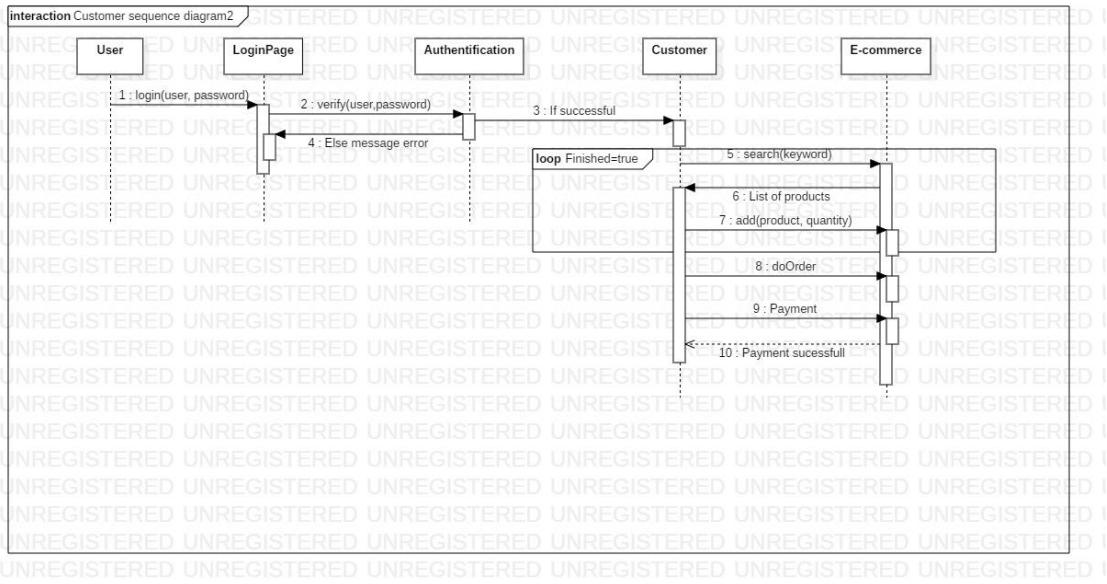


Figure 9: Diagramme de séquence d'un client pour un achat avec la fonction

3.4.3. Diagramme de séquence d'un client pour visualiser les commandes passées

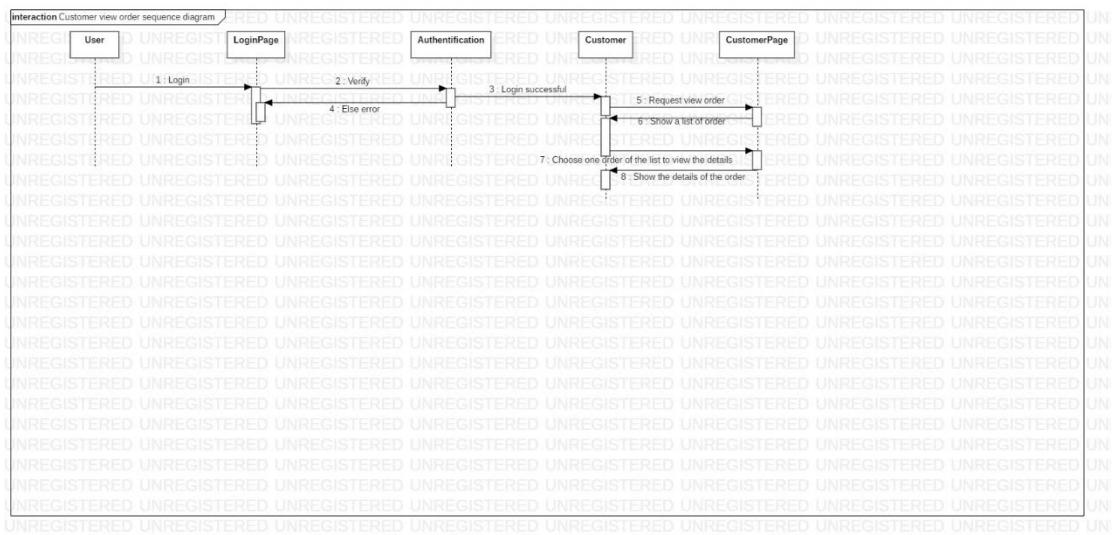


Figure 10: Diagramme de séquence pour visualiser les commandes passées

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur à bien fourni des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode client et il pourra visualiser ses commandes.

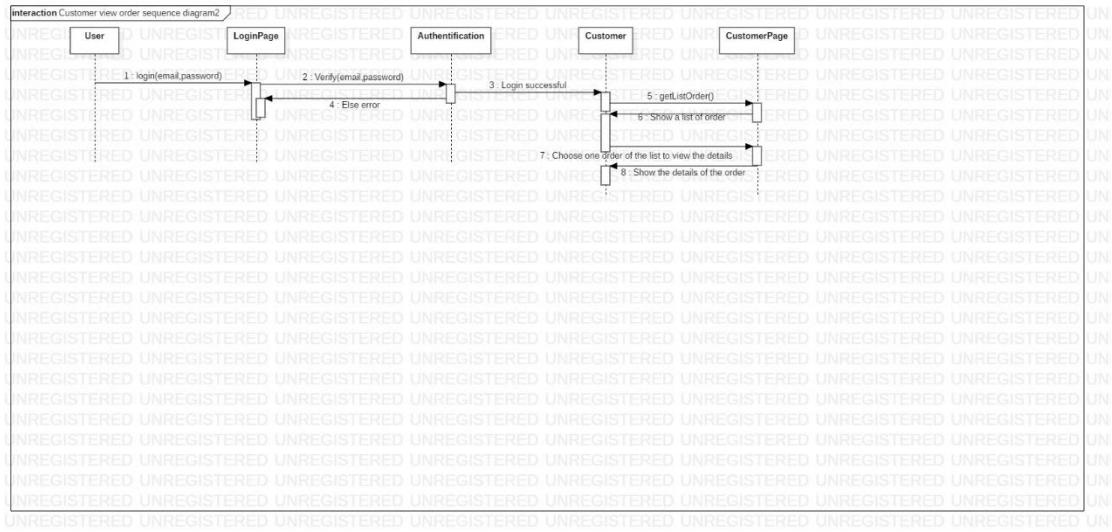


Figure 11: Diagramme de séquence pour visualiser les commandes passées avec la fonction

3.4.4. Diagramme de séquence d'un client pour modifier ses informations personnelles

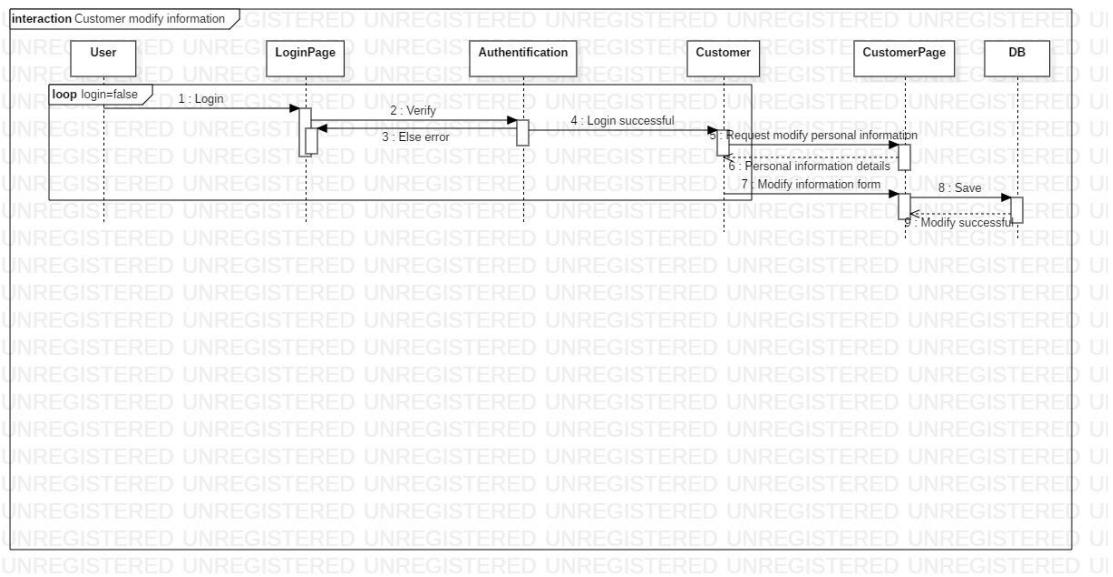


Figure 12: Diagramme de séquence d'un client pour modifier ses informations personnelles

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur à bien fournis des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode client et il pourra modifier ses données personnelles.

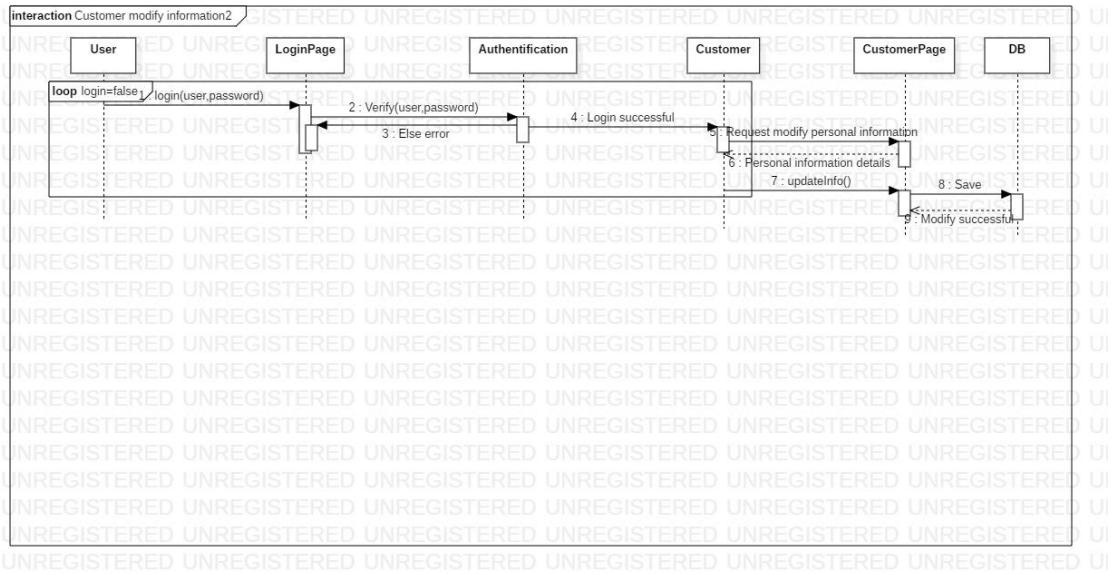


Figure 13: Diagramme de séquence d'un client pour modifier ses informations personnelles avec la fonction

3.4.5. Diagramme de séquence d'un salarié

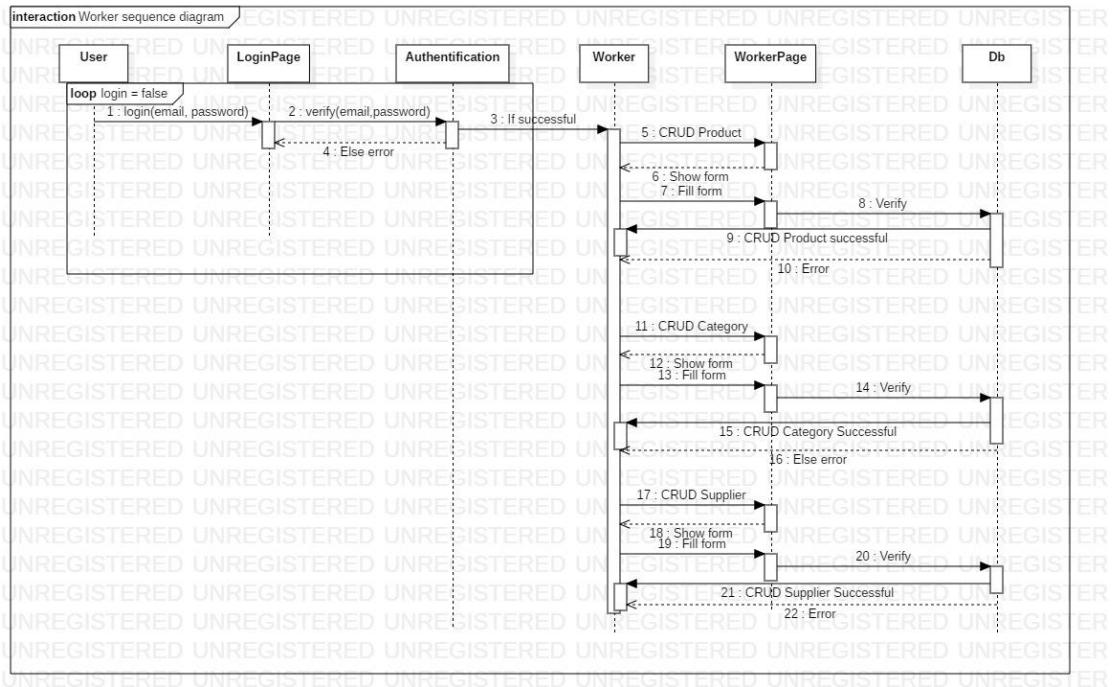


Figure 14: Diagramme de séquence d'un salarié pour des fonctions CRUD

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur à bien fournis des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode salarié et il pourra créer, lire, modifier et supprimer des fournisseurs, des produits, des catégories ext.

3.4.6. Diagramme de séquence d'un administrateur qui créer un utilisateur

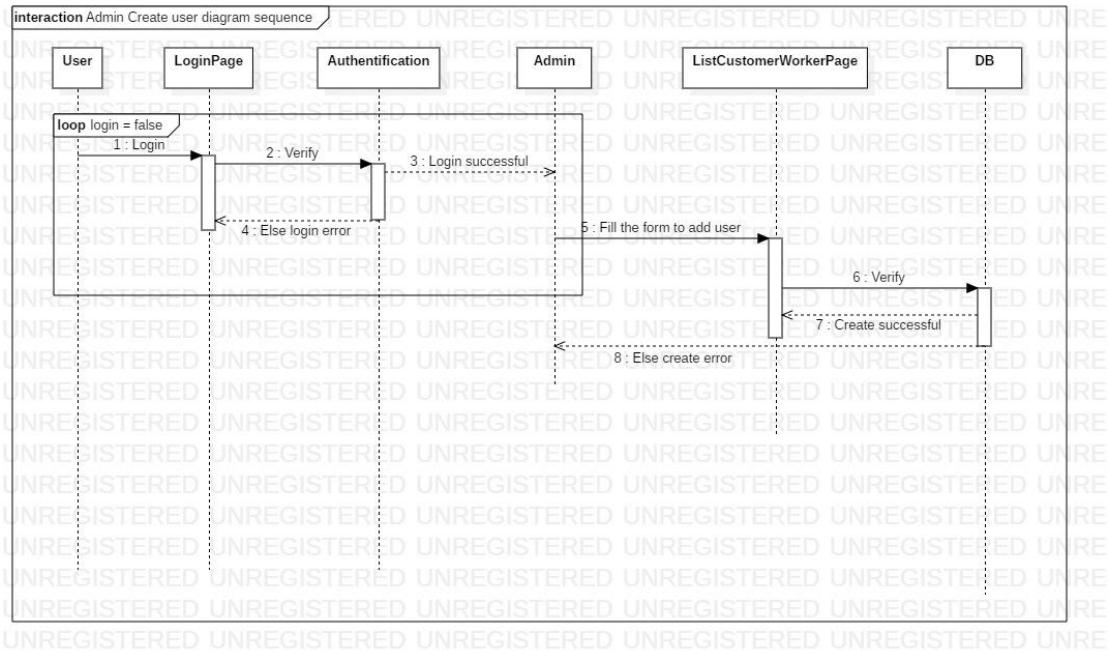


Figure 15: Diagramme de séquence d'un administrateur qui créer un utilisateur

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur à bien fournis des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode administrateur et il pourra donc créer des utilisateurs. Lors d'une création d'un utilisateur, l'administrateur doit remplir un formulaire, une fois ce formulaire a été rempli, une vérification si les données saisies n'ont pas été déjà créée.

3.5. Diagramme de séquence d'un administrateur qui gère les utilisateurs

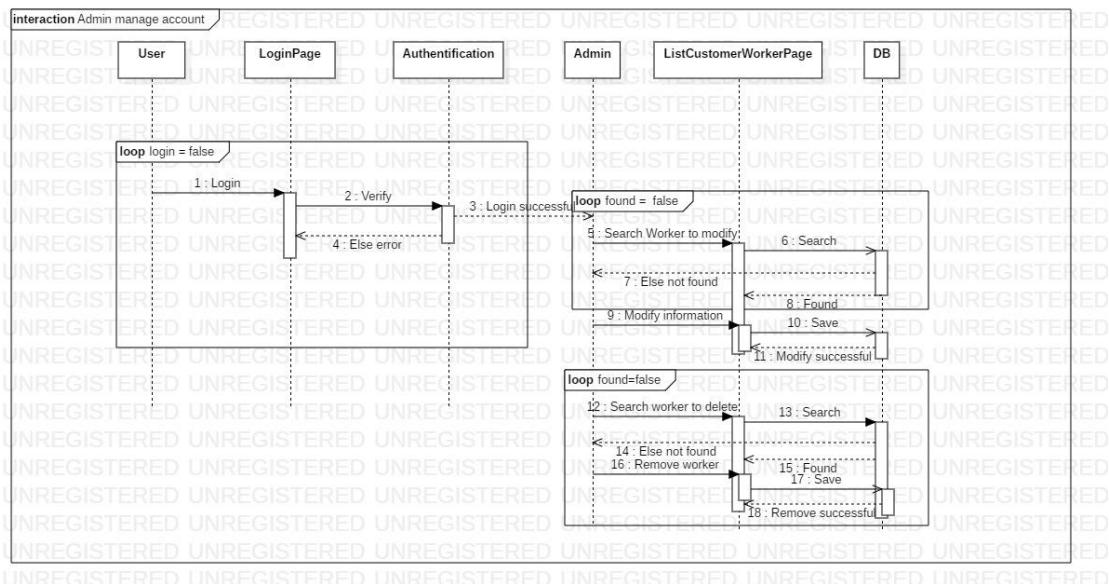


Figure 16: Diagramme de séquence d'un administrateur qui gère les utilisateurs

L'utilisateur va se connecter ensuite le système va vérifier si l'utilisateur a bien fourni des données correctes lors de la connexion sinon une erreur s'affiche. Une fois la vérification terminée avec succès, il s'est connecté en mode administrateur et il pourra donc gérer les comptes des utilisateurs comme modifier les données ou les supprimer.

3.6. Diagramme de collaboration

Un diagramme de collaboration ou diagramme de communication permet de mettre en évidence les échanges de messages entre objets. Cela nous aide à voir claire les actions qui sont nécessaires pour produire ces échanges de message et donc de compléter si besoin les diagrammes de séquence et de classe.

3.6.1. Diagramme de collaboration d'un client

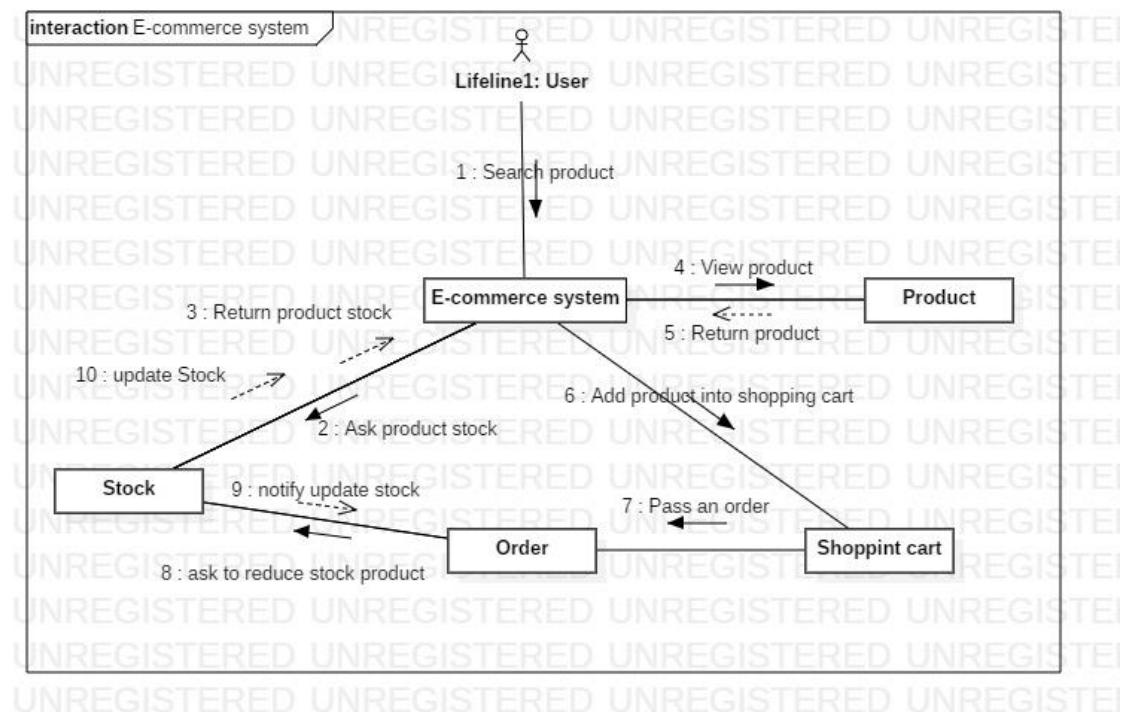


Figure 17: Diagramme de collaboration d'un client

Ce diagramme décrit la simulation d'un client qui fait la recherche d'un produit et ensuite l'ajoute dans le panier et valide la commande avec la mise à jour des quantités du produit à enlever commander dans la base de données.

3.6.2. Diagramme de collaboration d'un client qui visualise les commandes

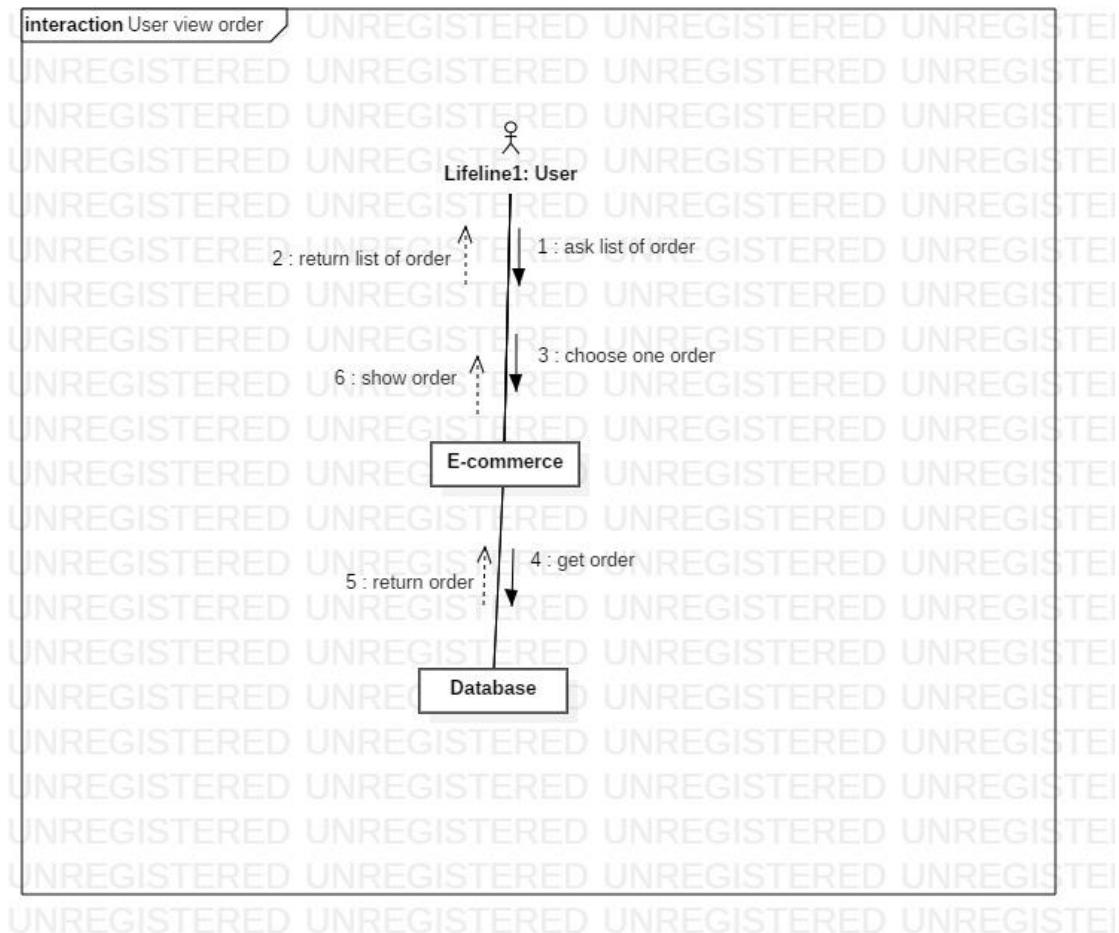


Figure 18: Diagramme de collaboration d'un client qui visualise les commandes

Ce diagramme décrit la simulation d'un client qui voudrait visualiser une commande qui à déjà été passé.

3.6.3. Diagramme de collaboration d'un salarié qui ajoute un produit

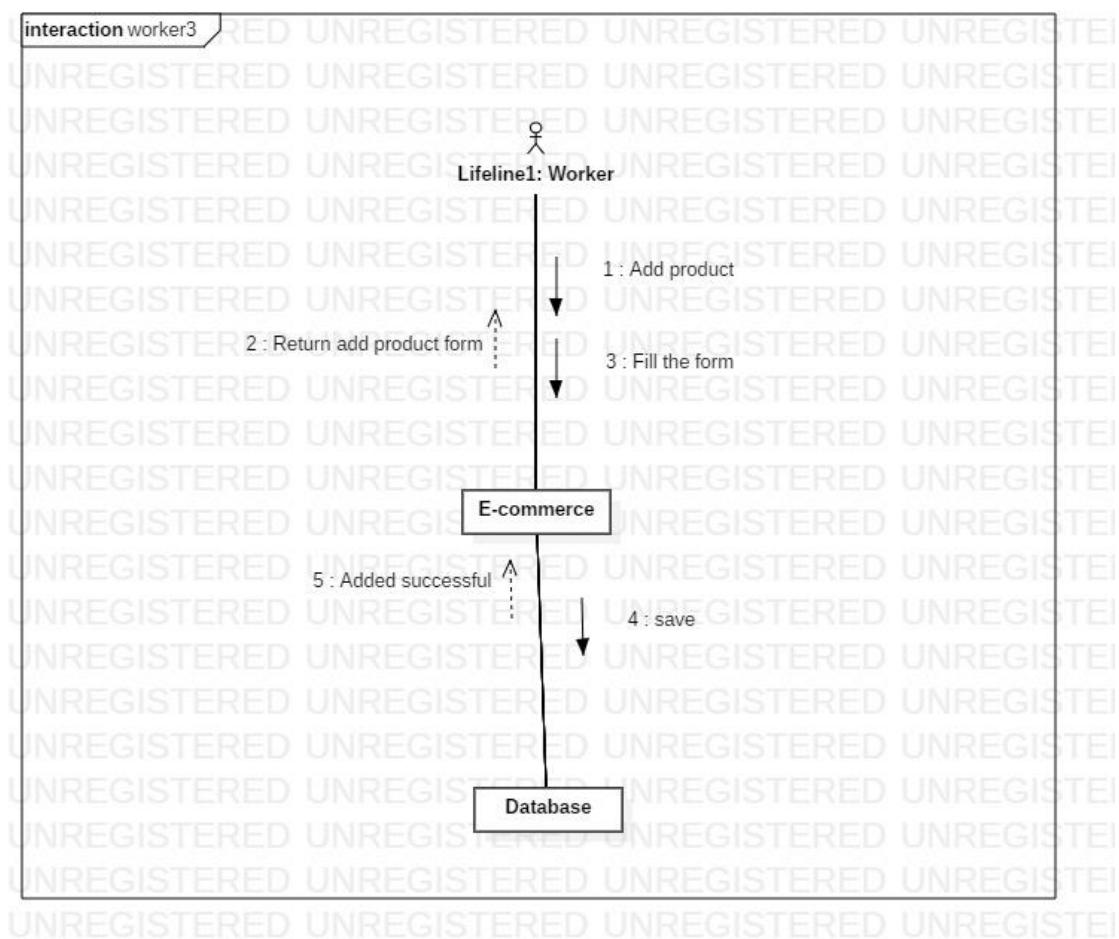


Figure 19: Diagramme de collaboration d'un salarié qui ajoute un produit

Ce diagramme décrit la simulation d'un salarié qui ajoute un nouveau produit dans le catalogue.

3.6.4. Diagramme de collaboration d'un salarié qui modifie un produit

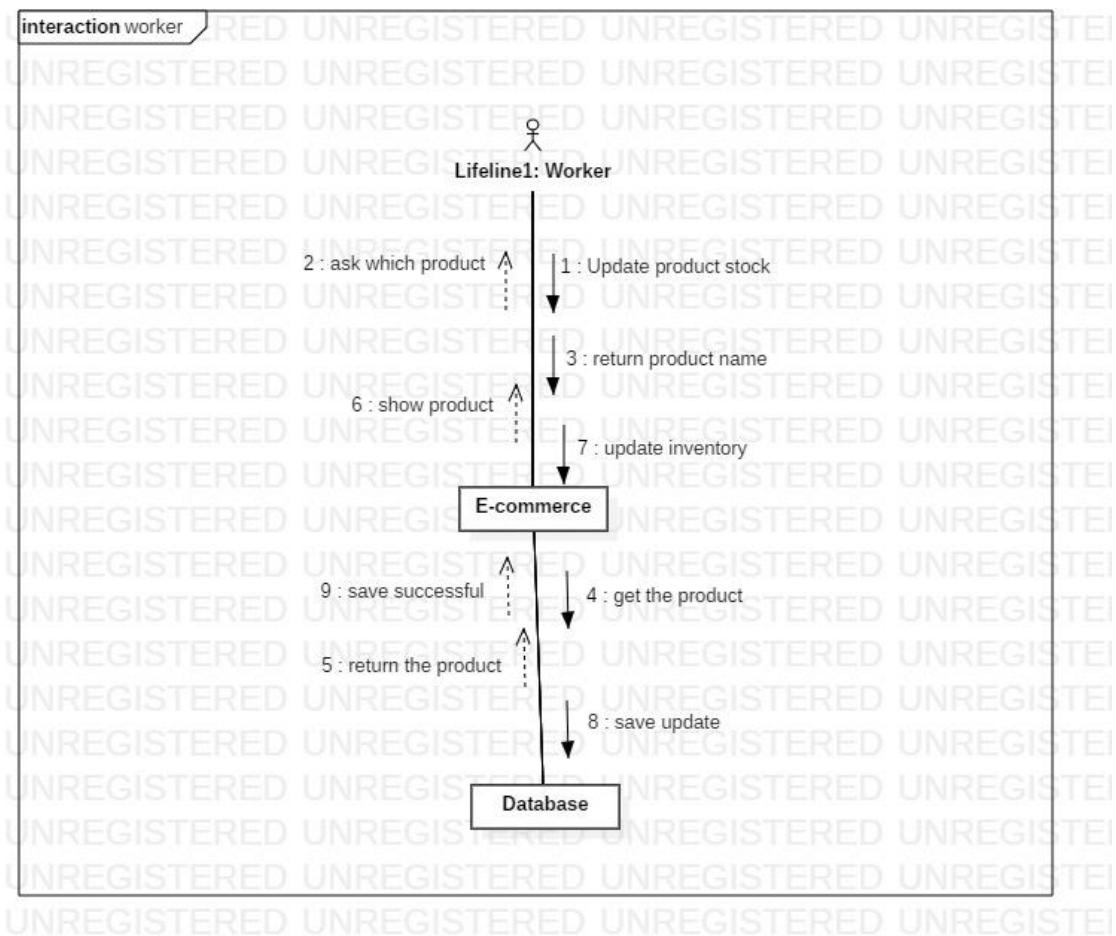


Figure 20: Diagramme de collaboration d'un salarié qui modifie un produit

Ce diagramme décrit la simulation d'un salarié qui fait la mise à jour d'un produit.

3.6.5. Diagramme de collaboration d'un salarié qui supprime un produit

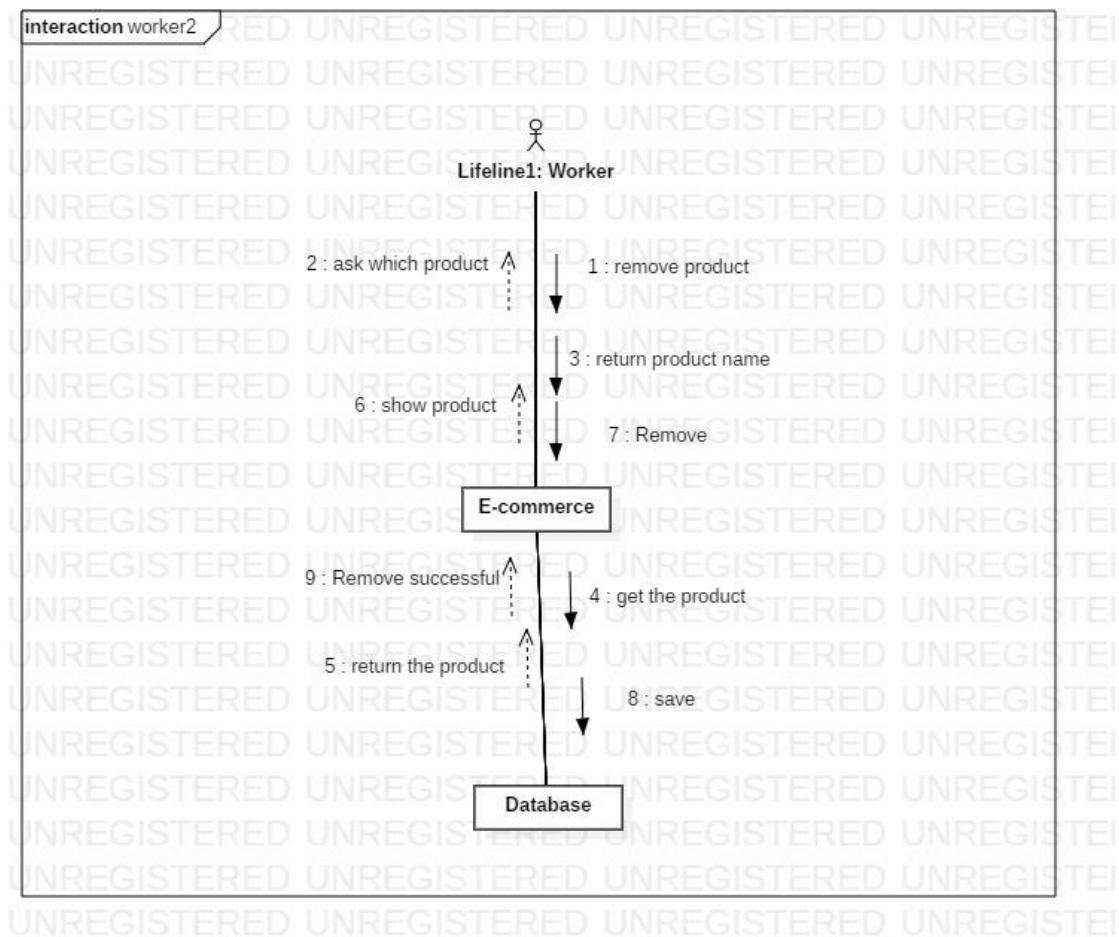


Figure 21: Diagramme de collaboration d'un salarié qui supprime un produit

Ce diagramme décrit la simulation d'un salarié qui supprime un produit de la base de données.

3.7. Diagramme de classes

Un diagramme de classe est un diagramme UML qui contient des classes, des interfaces, des packages et leurs relations, et qui fournit une vue logique de tout ou partie d'un système informatique.

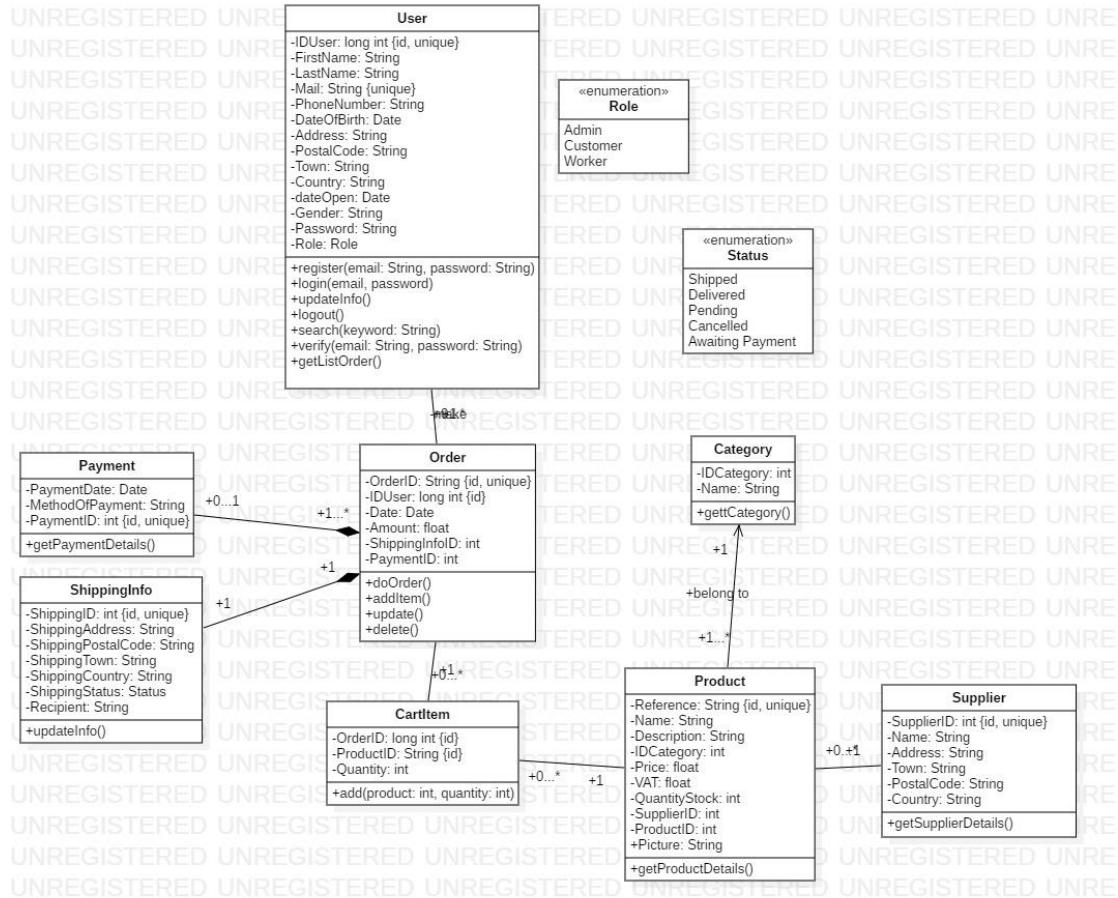


Figure 22: Diagramme de classe

L'application comporte les classes suivantes :

- User : Une classe qui contient les informations d'un utilisateur
- Order : Une classe qui contient les informations des commandes
- Payment : Une classe qui contient les informations du paiement
- ShippingInfo : Une classe qui contient les informations du destinataire de la commande
- CartItem : Cette classe contient juste la ligne de produit
- Product : Une classe qui contient les informations d'un produit
- Supplier : Toute informations du fournisseur
- Category : L'information de la catégorie du produit
- Status : Une énumération des statuts des commandes
- Role : Une énumération des statuts de l'utilisateur

Tous les classes ont des getters and setters dont ici nous n'avons pas détailler. Et aussi les fonction CRUD (Create, Read, Update, Delete).

User		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
IDUser	Long int	Identifiant unique de l'utilisateur
FirstName	String	Prénom de l'utilisateur
Lastname	String	Nom de l'utilisateur
Mail	String	Email de l'utilisateur
PhoneNumber	String	N° téléphone de l'utilisateur
DateOfBirth	Date	Date d'anniversaire de l'utilisateur
Address	String	L'adresse de l'utilisateur
PostalCode	String	Code Postal de l'utilisateur
Town	String	Ville de l'utilisateur
Country	String	Pays de l'utilisateur
DateOpen	Date	Date d'ouverture du compte
Gender	String	Sexe de l'utilisateur
Password	String	Mot de passe de l'utilisateur
Role	Role	Le statut de l'utilisateur
<i>Méthodes</i>		
Register(email : String, password : String)	Void	S'enregistrer un compte
Login(email : String, password : String)	Void	Se connecter
Search(keyword : String)	Void	Affiche la liste de produit
UpdateInfo()	Void	Mettre à jour les données de l'utilisateur
Logout()	Void	Se déconnecter
Verify(email : String, password, String)	Void	Vérifier s'y il existe dans la base de données
GetListOrder()	Void	Affiche la liste des commandes

Order		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
OrderId	Long int	Identifiant unique de la commande
IDUser	Long int	Identifiant de l'utilisateur
Date	Date	Date de la commande
Amount	Float	Le montant total de la commande

ShippingInfoID	Long int	Identifiant du destinataire
PaymentID	Long int	Identifiant du paiement
<i>Méthodes</i>		
makeOrder()	Void	Faire la commande
AddItem()	Void	Ajout un produit dans la commande
Update()	Void	Modifie la commande
Delete	Void	Supprimer la commande

Payment		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
PaymentID	Long int	Identifiant unique du paiement
PaymentDate	Date	Date de paiement
MethodOfPayment	String	Méthode de paiement
<i>Méthodes</i>		
GetPaymentDetails()	Void	Affiche les détails de paiement

ShippingInfo		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
ShippingID	Long int	Identifiant unique de la livraison
ShippingAddress	String	Adresse du destinataire
ShippingPostalCode	String	Code postal du destinataire
ShippingTown	String	Ville du destinataire
ShippingCountry	String	Pays du destinataire
ShippingStatus	Status	Status de la livraison
Recipient	String	Nom du destinataire
<i>Méthodes</i>		
UpdateInfo()	Void	Mettre mise à jour de la livraison

CartItem		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
OrderID	Long int	Identifiant de la commande
ProductID	Long int	Identifiant du produit
Quantity	Int	Quantité du produit
<i>Méthodes</i>		

Add(product : int, quantity : int)	Void	Ajout produit dans le panier
---	------	------------------------------

Product		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
ProductID	Long int	Identifiant unique du produit
Reference	String	Référence du produit
Name	String	Nom du produit
Description	String	Description du produit
IDCategory	Long int	Identifiant de la catégorie du produit
Price	Float	Prix du produit
VAT	Float	TVA du produit
QuantityStock	Int	Quantité de produit
SupplierID	Long int	Identifiant du fournisseur
Picture	String	Image du produit
<i>Méthodes</i>		
GetProductDetails()	Void	Affiche les détails du produit

Supplier		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
SupplierID	Long int	Identifiant unique du fournisseur
Name	String	Nom du fournisseur
Address	String	Adresse du fournisseur
Town	String	Ville du fournisseur
PostalCode	String	Code Postal du fournisseur
Country	String	Pays du fournisseur
<i>Méthodes</i>		
GetSupplierDetails()	Void	Affiche les détails du fournisseur

Category		
<i>Attributs</i>		
<i>Nom</i>	<i>Type</i>	<i>Description</i>
IDCategory	Long int	Identifiant unique de la catégorie du produit
Nom	String	Nom de la catégorie
<i>Méthodes</i>		
GetCategory()	Void	Affiche la catégorie du produit

4. Spécification des techniques

4.1. SPRING

4.1.1. INTRODUCTION

SPRING est effectivement un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application J2EE.

Il prend en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.

Le gros avantage par rapport aux serveurs d'application est qu'avec SPRING, vos classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le framework (au contraire des serveurs d'applications J2EE et des EJBs).

La décomposition du code en composants vise plusieurs objectifs :

- Favoriser la lisibilité du code,
- La documentation Simplifier la testabilité
- Mais surtout simplifier les maintenances futures et notamment : Les évolutions fonctionnelles

L'emploi d'un conteneur léger. Il s'agit d'un logiciel qui n'est pas intégré à la norme Java EE. Il va permettre :

- ✓ D'initialiser des composants
- ✓ Gérer le cycle de vie de ces composants
- ✓ Gérer les dépendances entre ces composants

4.1.2. LES MODULES DE SPRING

Le Spring Framework est composé de plusieurs modules distincts. Lorsque vous téléchargez et décompressez la distribution Spring Framework, vous trouverez 20 fichiers JAR différents.

Les 20 fichiers JAR qui composent Spring peuvent être classés dans l'une des six catégories différentes de fonctionnalités, comme l'illustre la figure ci-dessous.

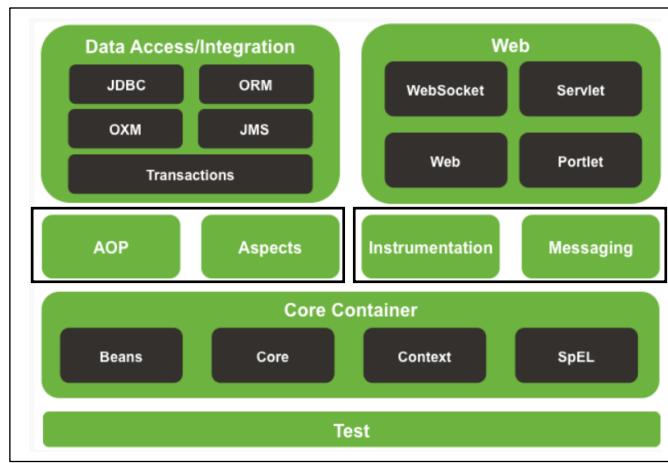


Figure 23: Classement de Spring

4.1.3. CORE SPRING CONTAINER

La pièce maîtresse du Spring Framework est un conteneur qui gère la façon dont les Beans sont créés, configurés et gérés dans une application Spring-Enabled. Dans ce module, vous trouverez :

- L'usine Spring **Bean**, qui est la partie du ressort qui fournit l'injection de dépendance. Basée sur l'usine Bean, vous trouverez plusieurs implémentations du contexte d'application de Spring, chacun d'entre eux fournissant une façon différente de configurer Spring.
- **Spring Core** : implémente notamment le concept d'inversion de contrôle (injection de dépendance). Il est également responsable de la gestion et de la configuration du conteneur.
- **Spring Context**: Ce module étend Spring Core. Il fournit une sorte de base de données d'objets, permet de charger des ressources (telles que des fichiers de configuration) ou encore la propagation d'évènements et la création de contexte comme par exemple le support de Spring dans un conteneur de Servlet.

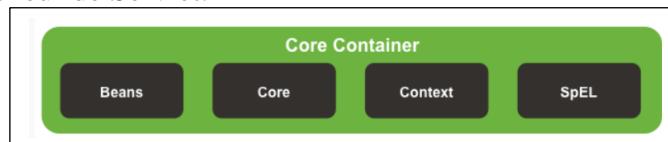


Figure 24: Core Container

4.1.4. LE MODULE AOP

Spring offre un support riche pour la programmation orientée aspect dans son module AOP.

Ce module sert de base au développement de vos propres aspects pour votre application Spring-Enabled.



Figure 25: Module AOP

4.1.5. LE MODULE DATA ACCESS AND INTEGRATION

- **Spring DAO** : Ce module permet d'abstraire les accès à la base de données, d'éliminer le code redondant et également d'abstraire les messages d'erreur spécifiques à chaque vendeur. Il fournit en outre une gestion des transactions.

Les DAO Spring ont la particularité de pouvoir facilement s'interfacer avec tout type de technologie permettant l'accès aux données :

- JDBC
- Hibernate,
- JDO,
- iBatis ou autre solution ORM logicielle JPA
- Etc....

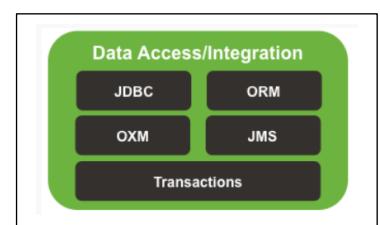


Figure 26: Data Access Integration

Pour chaque technologie, Spring propose une classe abstraite facilitant l'intégration des DAO, par exemple :

- DatasourceDaoSupport pour JDBC
- HibernateDaoSupport dans le cas de Hibernate
- JpaDaoSupport pour JPA
- **Spring ORM** : Cette partie permet d'intégrer des Framework de mapping Object/Relationnel tel que Hibernate, JDO ou iBatis avec Spring. La quantité de code économisé par ce package peut-être très impressionnante (ouverture, fermeture de session, gestion des erreurs)

4.1.6. LE MODULE WEB

- **Spring Web** : Ensemble d'utilitaires pour les applications web. Par exemple une servlet qui démarre le contexte (le conteneur) au démarrage d'une application web. Permet également d'utiliser des requêtes http. Ou une application à base de portlets pour le développement contre l'API portlet Java. Le paradigme Modèle-Vue-Contrôleur (MVC) est une approche

communément acceptée pour construire des applications Web de sorte que l'interface utilisateur soit séparée de la logique applicative.

- **WebSocket** : est un protocole de communication full-duplex superposé sur TCP. Il est généralement utilisé pour la communication interactive entre le navigateur d'un utilisateur et un serveur back-end.

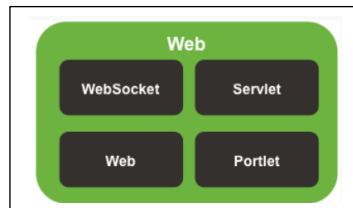


Figure 27: Module Web

4.1.7. MESSAGING

Spring Framework 4 inclut un module de Spring-Messaging avec des extraits clés du projet d'intégration Spring tels que **Message**, **MessageChannel**, **MessageHandler**, et d'autres pour servir de base aux applications basées sur la messagerie. Le module comprend également un ensemble d'annotations pour mapper les messages aux méthodes, similaire au modèle de programmation Spring MVC basé sur les annotations.



Figure 28: Messaging

4.1.8. TESTS

Reconnaissant l'importance des tests écrits par les développeurs, Spring propose un module dédié aux tests des applications Spring.

Dans ce module, vous trouverez une collection d'implémentations d'objets fantaisie pour écrire des tests unitaires contre le code qui fonctionne avec JNDI, les servlets et les portlets. Pour les tests au niveau de l'intégration, ce module permet de charger une collection de Beans dans un contexte d'application Spring et de travailler avec les Beans dans ce contexte.



Figure 29: Tests

4.1.9. CONFIGURATION DE L'ARCHITECTURE

Prenons un exemple si un client il va demander à chaque fois un service différent du service existant. Une solution est faire plusieurs implémentations à chaque nouveau service. Cette solution engendre des contraintes significatives, chaque nouvelle spécificité client demande :

- La modification du code existant
- La recompilation du code et le plus gênant, une montée de version ce qui implique généralement un redéploiement chez l'ensemble des clients Alors quelle est la solution ?

C'est ici que l'on introduit la notion de **configuration de l'architecture applicative**. Il va s'agir d'améliorer la solution précédente en procédant à l'instrumentation du code. Cette technique va permettre d'indiquer par simple configuration quelle implémentation doit être utilisée selon le contexte, Il ne sera plus nécessaire de modifier le code existant à chaque nouvelle implémentation.

4.1.9.1. NOTIONS D'ARCHITECTURE LOGICIELLE

4.1.9.1.1. PROGRAMMATION PAR TEMPLATE

L'objet de ce design pattern est de séparer l'invariant d'un procédé de sa partie variante. Dans Spring les Template sont très utilisés dans le cadre de l'accès aux données et de la gestion des transactions.

La partie invariante du code est placée dans la partie abstraite de la classe, ainsi toute classe qui héritera de cette classe abstraite n'aura qu'à implémenter la partie variante.

La partie variante est implémentée dans la méthode abstraite, en l'occurrence ce que vous voulez effectuer dans une transaction.

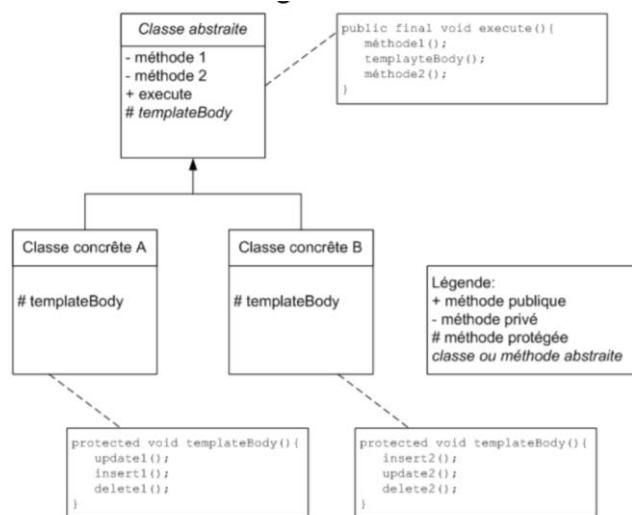


Figure 30: Design pattern de la programmation par template

4.1.9.1.2. L'INVERSION DE CONTROLE

Les conteneurs légers sont souvent appelés conteneurs d'inversion de contrôle, ou IoC (Inversion of Control). Il s'agit d'un "design pattern" qui a pour objectif de faciliter l'intégration de composants entre eux. Le concept est basé sur le fait d'inverser la façon dont sont créés les objets.

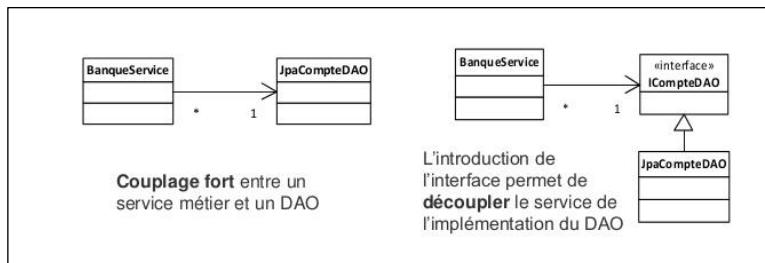


Figure 31: Design pattern de l'inversion de contrôle

4.2. SPRING BOOT

4.2.1. INTRODUCTION

Spring Boot est un Framework open source basé sur Java utilisé pour créer un micro service. Il est développé par Pivotal Team et est utilisé pour construire des applications en standalone et des applications Spring prêtes pour la production.

4.2.2. QU'EST-CE QUE MICRO SERVICE ?

Micro Service est une architecture qui permet aux développeurs de développer et de déployer des services de manière indépendante. Chaque service en cours d'exécution a son propre processus, ce qui permet d'obtenir le modèle léger pour prendre en charge les applications métier.

4.2.2.1. AVANTAGES

Micro services offre les avantages suivants à ses développeurs :

- Déploiement facile
- Évolutivité simple
- Compatible avec les conteneurs
- Configuration minimale
- Temps de production réduit

4.2.3. QU'EST-CE QUE SPRING BOOT ?

Spring Boot fournit une bonne plate-forme pour les développeurs Java pour développer une application de type standalone et production-grade(qualité-production). Que vous pouvez simplement exécuter. Vous pouvez commencer avec des configurations minimales sans avoir besoin d'une configuration complète de Spring.

4.2.3.1. AVANTAGES

Spring Boot offre les avantages suivants à ses développeurs :

- Facile à comprendre et à développer des applications de ressorts
- Augmente la productivité
- Réduit le temps de développement

4.2.3.2. OBJECTIFS

Spring Boot est conçu avec les objectifs suivants :

- Pour éviter une configuration XML complexe au Spring
- Développer des applications Spring prêtes pour la production d'une manière plus simple.
- Pour réduire le temps de développement et exécuter l'application de manière autonome
- Offrir un moyen plus facile de commencer à utiliser l'application

4.2.3.3. POURQUOI SPRING BOOT ?

Vous pouvez choisir Spring Boot en raison des caractéristiques et des avantages qu'il offre comme indiquer ici :

- Il offre un moyen flexible de configurer Java Beans, les configurations XML et les transactions de base de données.
- Il fournit un traitement par lots puissant et gère les terminaux REST.
- Dans Spring Boot, tout est configuré automatiquement ; aucune configuration manuelle n'est nécessaire.
- Il offre une application de Spring basée sur des annotations.
- Facilite la gestion des dépendances
- Il inclut le conteneur de servlet embarqué

4.2.4. SPRING BOOT STARTERS

La gestion des dépendances est une tâche difficile pour les grands projets. **Spring Boot** résout ce problème en fournissant un ensemble de dépendances pour la commodité des développeurs.

Par exemple, si vous voulez utiliser Spring et JPA pour l'accès aux bases de données, il suffit d'inclure la dépendance **spring-boot-starter-data-jpa** dans votre projet.

Notez que tous les Spring Boot starters suivent le même modèle de nom **spring-boot-starter-***, où * indique qu'il s'agit d'un type de l'application.

4.2.4.1. EXEMPLES

Regardez les Spring Boot starters expliqués ci-dessous pour une meilleure compréhension :

La dépendance de **Spring Boot Starter Security** est utilisée pour Spring Security. Son code est affiché ci-dessous :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

La dépendance Spring-Boot-Starter-web est utilisée pour écrire un Rest Endpoints. Son code est indiqué ci-dessous

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

4.2.4.2. SPRING BOOT APPLICATION

Le point de départ de l'application Spring Boot est la classe contient l'annotation `@SpringBootApplication`. Cette classe devrait avoir la méthode principale pour exécuter la commande Application de Spring Boot. L'annotation `@SpringBootApplication` inclut AutoConfiguration, Component Scan et Spring Boot Configuration.

Si vous avez ajouté l'annotation `@SpringBootApplication` à la classe, vous n'avez pas besoin d'ajouter la configuration automatique `@EnableAutoConfiguration`, `@ComponentScan` et `@SpringBootConfiguration` annotation. L'annotation `@SpringBootApplication` inclut toutes les autres annotations.

Observez le code suivant pour une meilleure compréhension :

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}

```

4.2.5. COMMAND LINE RUNNER

Command Line Runner est une interface. Il est utilisé pour exécuter le code après le démarrage de l'application Spring Boot. L'exemple ci-dessous montre comment implémenter l'interface Command Line Runner sur le fichier de classe principal. Par défaut, Spring Boot utilise le numéro de port 8080 pour démarrer le Tomcat.

```

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication implements CommandLineRunner {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Override
    public void run(String... arg0) throws Exception {
        System.out.println("Hello world from Command Line Runner");
    }
}

```

4.2.5.1. PROPERTIES FILE

Les fichiers de propriétés sont utilisés pour garder un nombre 'N' de propriétés dans un seul fichier pour exécuter l'application dans un environnement différent. Dans Spring Boot, les propriétés sont conservées dans le fichier **application.properties** sous le classpath.

Le fichier **application.properties** se trouve dans le répertoire **src/main/resources**

4.3. LE MODELE MVC 1 ET 2 (MODEL VIEW CONTROLLER)

L'un des objectifs les plus importants de Spring est la séparation des couches, la partie MVC et le concept d'un point de vue général semblent indispensables.

Pour réaliser la séparation entre les couches et les technologies, le paradigme se divise en trois parties :

4.3.1. LE MODELE (MODEL)

C'est la représentation des informations liées spécifiquement au domaine de l'application. C'est un autre nom pour désigner la couche métier.

La couche métier ou plutôt la partie qui représente l'information en respectant une structure liée au domaine d'activité, celle qui effectue des calculs ou des traitements amenant une plus valeur sur l'information brute. Par exemple le calcul du total des taxes sur un prix hors taxe ou encore des vérifications telles que : Y a-t-il encore des articles en stock avant d'autoriser une sortie de stock.

La couche d'accès aux données est ignorée ici parce que sous-jacente à la couche métier.

4.3.2. LA VUE (VIEW)

Cette couche ou ce module effectue le rendu de la couche métier dans une forme qui est compréhensible par l'homme, par exemple une page html dans notre cas ce angular.

4.3.3. LE CONTROLEUR (CONTROLLER)

Ce module organise la communication entre les deux premiers. Il invoquera une action dans la couche métier (par exemple récupérer les données d'un client) suite à une action de l'utilisateur et passera cette information à la vue (page html « angular » qui affiche les détails).

Cela vous permettra de pouvoir debugger et tester plus facilement le code en l'isolant de la partie affichage.

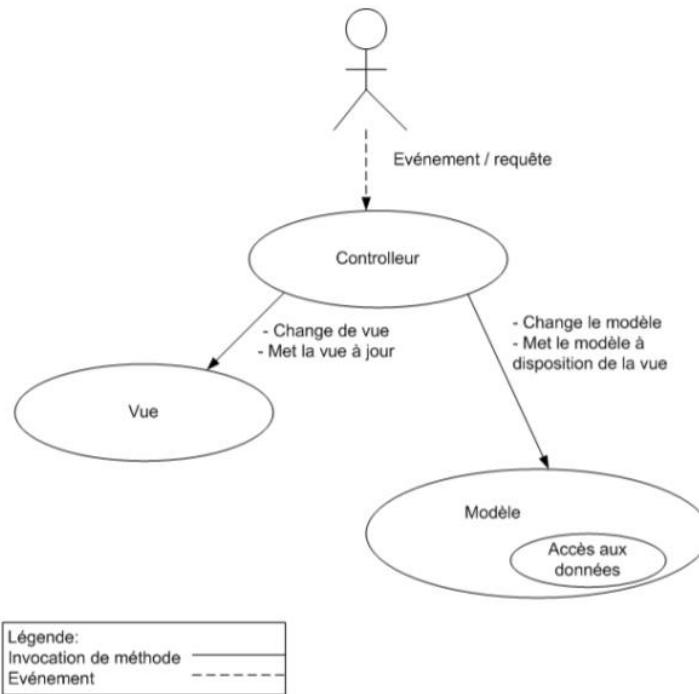


Figure 32: Schéma MVC

4.3.4. LE DISPATCHERSERVLET

Le Framework Spring Web Model-View-Controller (MVC) est conçu autour d'un Dispatcher Servlet qui traite toutes les requêtes et réponses HTTP. Le flux de travail de traitement des requêtes du Dispatcher Servlet Spring Web MVC est montré dans l'illustration suivante.

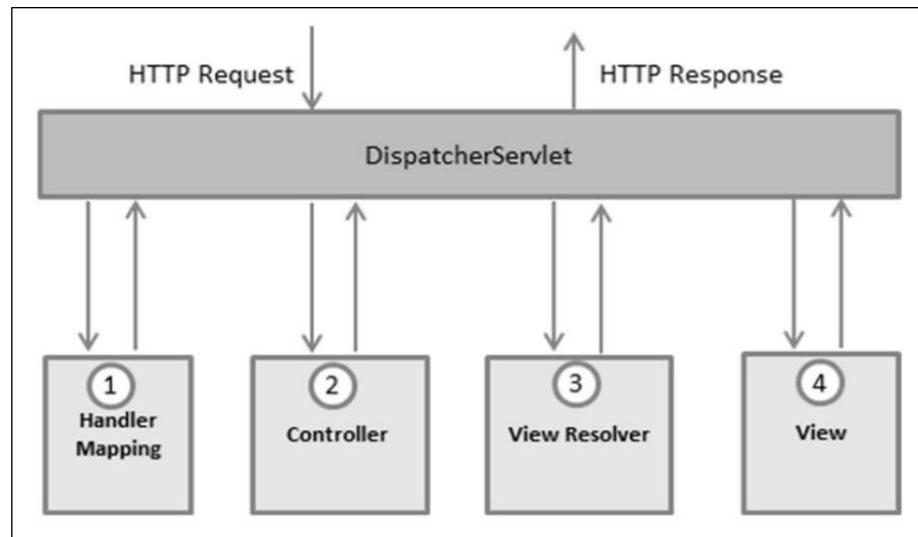


Figure 33: Flux de travail de traitement des requêtes du Dispatcher Servlet Spring Web MVC

Voici la séquence d'événements correspondant à une requête HTTP entrante vers Dispatcher Servlet :

- Après réception d'une requête HTTP, Dispatcher Servlet consulte le Handler Mapping pour appeler le Controller approprié.

- Le contrôleur prend la demande et appelle les méthodes de service appropriées basées sur la méthode GET ou POST utilisée. La méthode de service définira les données du modèle en fonction de la logique métier définie et renvoie le nom de la vue au Dispatcher Servlet.
- Le Dispatcher Servlet prendra l'aide de ViewResolver pour récupérer la vue définie pour la requête.
- Une fois la vue finalisée, le Dispatcher Servlet transmet les données du modèle à la vue, qui est finalement rendue sur le navigateur.

Tous les composants mentionnés ci-dessus, c'est-à-dire Handler Mapping, Controller et ViewResolver font partie de WebApplicationContext, qui est une extension de l'ApplicationContext simple avec quelques fonctionnalités supplémentaires nécessaires pour les applications web.

4.4. SPRING SECURITY

Spring Security est un Framework de sécurité léger qui fournit une authentification et un support d'autorisation afin de sécuriser les applications Spring. Il est livré avec des implémentations d'algorithmes de sécurité populaires.

But de la sécurité est d'empêcher un utilisateur d'accéder à une ressource à laquelle il n'a pas droit.

On a besoin de savoir deux choses :

Qui veut accéder ? et Est-ce qu'il a le droit d'accéder à cette ressource ?

Configuration de la sécurité :

Il y a deux façons de configurer la sécurité :

1. En utilisant des données statiques (en mémoire, dans un fichier)
2. En utilisant des données dynamiques provenant d'une base de données

Spring Security fournit des services d'authentification et d'autorisation au niveau de l'entreprise.

Spring Security fournit l'interface GrantedAuthority afin de s'authentifier à une application. En général, GrantedAuthority détient simplement le nom du rôle sous la forme d'une chaîne, comme "ROLE_USER", "ROLE_ADMIN", ou ce que vous voulez. Le préfixe "ROLE_" n'est pas vraiment nécessaire mais est considéré comme une bonne pratique.

L'autorisation est basée sur des listes de contrôle d'accès, telles que "Jean est autorisé à accéder à cet objet sécurisé particulier, mais pas à celui-ci".

4.4.1. TYPES D'AUTHENTIFICATION PRIS EN CHARGE

La méthode d'authentification la plus courante est le système de connexion simple basé sur un formulaire, un nom d'utilisateur et un mot de passe. Cependant, il supporte à peu près tout, y compris le certificat client X.509. Un autre outil intéressant est l'OpenID, qui vous permet d'utiliser quelque chose comme votre compte Google ou Facebook pour l'authentification.

- En-têtes d'authentification HTTP BASIC (une norme basée sur IEFT RFC)
- En-têtes d'authentification HTTP Digest (une norme basée sur IEFT RFC)
- Échange de certificat client HTTP X.509 (une norme basée sur IEFT RFC)

- LDAP (une approche très courante des besoins d'authentification multi-plateformes, en particulier dans les grands environnements)
- Authentification par formulaire (pour des besoins d'interface utilisateur simples)
- Authentification OpenID
- Authentification basée sur des en-têtes de requête préétablis (tels que Computer Associates Siteminder)
- JA-SIG Central Authentication Service (aussi connu sous le nom de CAS, qui est un système d'authentification unique open source populaire)
- Propagation transparente du contexte d'authentification pour Remote Method Invocation (RMI) et HttpInvoker (un protocole Spring remoting)
- Authentification automatique "remember-me" (vous pouvez donc cocher une case pour éviter la ré-authentification pendant une période de temps prédéterminée)
- Authentification anonyme (permettant à chaque appel d'assumer automatiquement une identité de sécurité particulière)
- Authentification Run-as (ce qui est utile si un appel doit passer avec une identité de sécurité différente)
- Service d'authentification et d'autorisation Java (JAAS)

4.4.2. QUE SONT L'AUTHENTIFICATION ET L'AUTORISATION ?

4.4.2.1. L'AUTHENTIFICATION

Pour expliquer l'authentification, pensez à un simple nom d'utilisateur et mot de passe. Basé sur la fourniture d'un nom d'utilisateur et d'un mot de passe corrects, un mécanisme quelconque permet à l'utilisateur d'effectuer ou d'empêcher l'utilisateur d'effectuer certaines tâches. Par exemple, un nom d'utilisateur authentifié avec un rôle d'utilisateur normal ne sera normalement pas autorisé à supprimer d'autres comptes utilisateurs.

4.4.2.2. L'AUTORISATION

Par contre l'autorisation est l'équivalent d'une liste de contrôle d'accès. Spring Security fournit l'interface `AccessDecisionManager` et trois implémentations pour permettre l'autorisation `AffirmativeBased`, `ConsensusBased` et `UnanimousBased` pour un objet sécurisé.

4.4.2.3. LE CONTEXTE D'AUTHENTIFICATION ET DE SECURITE

- Authentication représente l'utilisateur (personne se connectant à l'application)
- GrantedAuthority – quelles sont les permissions dont dispose l'utilisateur ?

Un objet d'authentification représente la personne qui se connecte à votre application. A l'intérieur de l'objet `Authentication` se trouve une collection d'objets `GrantedAuthority`, qui spécifie les permissions dont dispose l'utilisateur. L'utilisateur peut avoir plusieurs autorités.

- SecurityContext supports l'authentification : l'interface de SecurityContext et son implémentation contient à son tour l'objet Authentication.
- SecurityContextHolder permet d'accéder à l'application SecurityContext. Enfin, la classe SecurityContextHolder associe le SecurityContext au thread courant.

4.4.3. USERDETAILS ET USERDETAILSSERVICE

L'interface UserDetails définit les informations nécessaires pour créer un objet d'authentification que Spring Security peut utiliser. Il définit environ 7 méthodes pour obtenir le nom d'utilisateur et le mot de passe, pour obtenir la collection d'objets GrantedAuthority, et pour déterminer si le compte est verrouillé, expiré, activé, etc.

L'interface UserDetailsService définit la méthode loadUserByUsername.

4.4.3.1. CONTROLE D'ACCES BASE SUR LES EXPRESSIONS

Explications :

- hasRole() - évalue à true si le principal courant a un rôle représenté par un argument String.
- hasAnyRole() - évalue à true si le principal courant a un rôle représenté par une liste de chaînes délimitée par des virgules
- isAuthenticated() - true si l'utilisateur n'est PAS anonyme
- isFullyAuthenticated() - true si l'utilisateur n'est PAS anonyme ou un utilisateur remember-me
- permitAll() - toujours vrai

4.4.4. SECURITY DEPENDANCES

Ajout les dépendances suivantes à pom.xml :

- spring-security-core
- spring-security-web
- spring-security-config

4.5. ANGULAR

4.5.1. ARCHITECTURE ANGULARJS

4.5.1.1. INTRODUCTION

AngularJS est un framework JavaScript, créé en octobre 2010 par des développeurs de chez Google et sous la licence MIT.

Il propose une architecture basée sur le pattern MVC (Model View Controller) pour des SPA (Single Page Application), autrement dit des applications web où la navigation se fait sur une même et unique page (exemple : Gmail, Dropbox, iCloud, etc.).

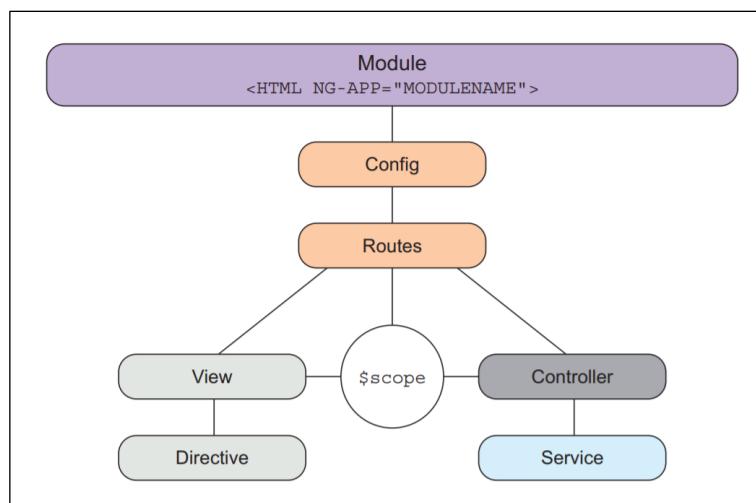


Figure 34: Design pattern MVC

4.5.1.2. CONFIG

Le bloc config d'une application AngularJS permet d'appliquer la configuration avant l'exécution effective de l'application. Ceci est utile pour configurer les routes, configurer dynamiquement les services, etc.

4.5.1.3. ROUTES

Routes vous permet de définir des façons de naviguer vers des états spécifiques dans votre application. Ils vous permettent également de définir des options de configuration pour chaque route spécifique, telles que le modèle et le contrôleur à utiliser.

4.5.1.4. VUES

La vue dans AngularJS est ce qui existe après qu'AngularJS ait compilé et rendu le DOM avec tout le câblage JavaScript en place.

4.5.1.5. \$SCOPE

\$scope est essentiellement la colle entre la vue et le contrôleur dans une application AngularJS. Avec l'introduction du contrôleur comme syntaxe, le besoin d'utiliser explicitement \$scope a été considérablement réduit.

4.5.1.6. LE CONTROLEUR

Le contrôleur est responsable de la définition des méthodes et des propriétés auxquelles la vue peut se lier et interagir. En tant que meilleure pratique, les contrôleurs devraient être légers et se concentrer uniquement sur la vue qu'ils contrôlent à l'aide de l'attribut ng-controller.

4.5.1.7. DATA BINDING

Pour insérer dynamiquement des données de l'applications dans les vues des composants, Angular définit des techniques pour assurer la liaison des données. Il y a deux types de data binding :

1.1.1. One way data binding (Liaison de données unidirectionnelle)

- Interpolation (Composante > Template)
 - En utilisant {{ }} nous affichons les valeurs du composant dans le modèle
- Liaison de propriétés (Composant > Template)
 - En utilisant [], Propriété de l'élément, propriété du composant, propriété de la directive. Exemple [attr.aria-label] pour les attributs,[class.class1] pour les classes,[style.color] pour css.
- Événements biding(Template > Component)
 - En utilisant (), événements comme clicked, mouse enter, changed

1.1.2. Two way data binding (Liaison de données bidirectionnelle)

Envoie les valeurs d'un composant à l'autre et renvoie les valeurs modifiées d'un composant à l'autre. En utilisant [()].

4.5.2. ARCHITECTURE D'ANGULAR

Angular est un Framework pour créer la partie Front End des applications web en utilisant HTML et JavaScript ou Type Script compilé en JavaScript.

Une application Angular se compose de :

- Un à plusieurs modules dont un est principal.
- Chaque module peut inclure :
 - Des composants web : La partie visible de l'application Web (IHM)
 - Des services pour la logique applicative.
 - Les composants peuvent utiliser les services via le principe de l'injection des dépendances.
 - Les directives : un composant peut utiliser des directives
 - Les pipes : utilisés pour formater l'affichage des données dans les composants.

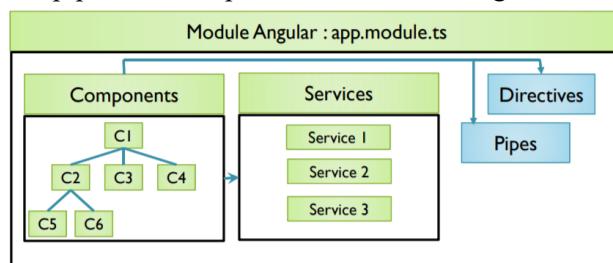


Figure 35: Structure du module

4.5.2.1. LES MODULES

Les modules servent de conteneurs pour vous aider à organiser le code dans votre application AngularJS. Les modules peuvent contenir des sous-modules, ce qui facilite la composition des fonctionnalités selon les besoins. Le module racine, appelé classiquement **AppModule**, Un module angulaire est une classe avec un décorateur **@NgModule**

```

src/app/app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports: [ BrowserModule ],
  providers: [ Logger ],
  declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Figure 36: app.module.ts

- Déclarations : la classe représentant le module. Angular a trois types de classes de modules
- Composants, directives et Pipes.

- exports - Pour exporter des classes qui peuvent être utilisées dans d'autres modules.
- imports - Pour importer d'autres modules.
- Provider - Pour enregistrer les usines de services.
- bootstrap - Pour déclarer le composant racine du module. Seul le module racine doit définir cette propriété.

4.5.2.2. DIRECTIVE

Une directive est une extension d'une vue dans AngularJS en ce sens qu'elle vous permet de créer des éléments personnalisés et réutilisables qui encapsulent le comportement. Vous pouvez considérer les directives comme des composants ou des décorateurs pour votre HTML. Les directives sont utilisées pour étendre les vues et pour rendre ces extensions disponibles pour une utilisation à plus d'un endroit.

Dans Angular 2.0, il y aura trois types de directives :

- **Directives sur les composants** : Ces directives créeront des composants réutilisables en encapsuler la logique en JavaScript, HTML ou une feuille de style CSS optionnelle.
- **Directives sur les décorateurs** : Ces directives seront utilisées pour décorer (par exemple en affichant/masquant des éléments à l'aide de la commande ng-show/ng-hide).
- **Directives sur les modèles** : Elles transformeront le HTML en un modèle réutilisable. Le l'instanciation du template et son insertion dans le DOM(Document Object Model) peut être entièrement contrôlé par l'auteur de la directive. Exemples : ng-if et ng-repeat.

4.5.2.3. LES SERVICES

Les services fournissent des fonctionnalités communes à une application AngularJS. Par exemple, si vous avez des données dont plus d'un contrôleur a besoin, vous pouvez promouvoir ces données à un service et les mettre ensuite à la disposition des contrôleurs via le service. Les services étendent les contrôleurs et les rendent plus accessibles globalement. Ils permettent d'éviter une redondance du code. Ce sont des singltons. AngularJS en propose plusieurs par défaut (routing, i18n, ajax, scope, filter, etc.).

4.5.2.4. COMPOSANTS

Les composants sont des éléments importants dans Angular. L'application est formée par un ensemble de composants. Chaque composant peut imbriquer d'autres composants définissant ainsi une structure hiérarchique. Le composant racine s'appelle Root Component.

Chaque composant se compose principalement des éléments suivants :

- HTML Template : représentant sa vue
- Une classe représentant sa logique métier
- Une feuille de style CSS
- Un fichier spec sont des tests unitaires Ils sont exécutés à l'aide du framework de test javascript Jasmine via le programme de tâches Karma lorsque vous utilisez la commande 'ng test'.

Les composants sont faciles à mettre à jour et à échanger entre les différentes parties des applications.

Pour créer facilement des composants Angular, on peut utiliser la commande ng comme suit : ng generate component NomComposant ou ng g c NomComposant.

4.5.2.5. DÉMARRAGE DE L'APPLICATION

Le module racine est démarré dans le fichier main.ts, Par défaut le module racine s'appelle AppModule. On peut utiliser la commande ng comme suit ng serve .

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

5. Les outils utilisés

Dans ce document les outils de développement utilisés sont les suivants :

- Spring Tool Suite (STS) pour le serveur Spring, pour la partie back-end.
- Webstorm, un éditeur pour la partie front-end.
- WampServer, pour la gestion de la base de données MySQL 5.

5.1. Spring Tool Suite (STS)

- Installation de Spring Tool Suite depuis le site : <https://spring.io/tools3/sts/all> .

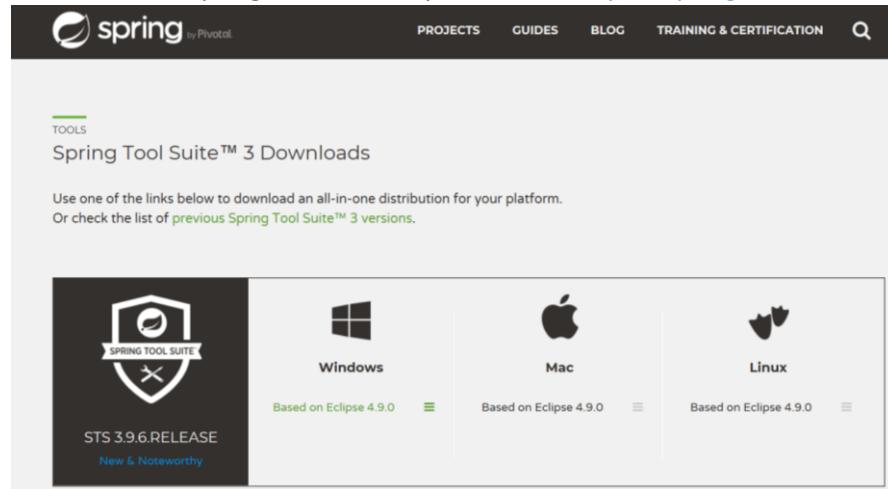


Figure 37: Site pour le téléchargement de STS

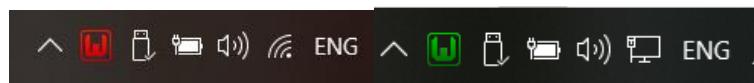
- Télécharger la dernière version de STS.

C > Downloads > spring-tool-suite-4-4.0.0.RELEASE-e4.9.0-win32.win32.x86_64 > sts-4.0.0.RELEASE			
Name	Date modified	Type	Size
configuration	12/2/2018 3:00 PM	File folder	
features	11/26/2018 5:10 PM	File folder	
META-INF	10/5/2018 6:26 PM	File folder	
p2	10/11/2018 7:07 PM	File folder	
plugins	11/26/2018 5:10 PM	File folder	
readme	10/5/2018 6:45 PM	File folder	
.eclipseproduct	10/5/2018 6:25 PM	ECLIPSEPRODUCT ...	1 KB
artifacts.xml	11/26/2018 5:10 PM	XML Document	246 KB
derby.log	11/1/2018 10:38 PM	Text Document	2 KB
eclipsec.exe	10/5/2018 6:25 PM	Application	120 KB
license.txt	10/5/2018 6:25 PM	TXT File	12 KB
SpringToolSuite4.exe	10/5/2018 6:25 PM	Application	408 KB
SpringToolSuite4.ini	11/26/2018 5:10 PM	Configuration setti...	1 KB

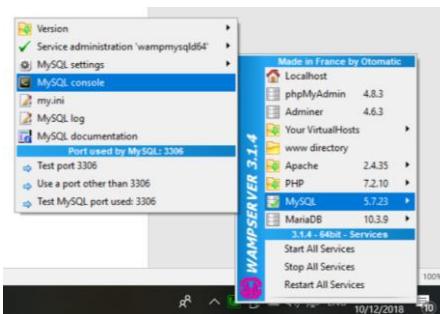
Figure 38: Répertoire de STS

5.2. WampServer

- Télécharger depuis le site : <https://sourceforge.net/projects/wampserver/d>.
- Lancer le fichier télécharger wapser64.exe pour lancer l'installation.
- Pendant l'installation, il va vous demander de choisir un navigateur par défaut.
- Après l'installation, il faut configurer un utilisateur et un mot de passe. Lancer WampServer une icône va apparaître comme ci-dessous, puis attendez jusqu'à que l'icône soit verte.

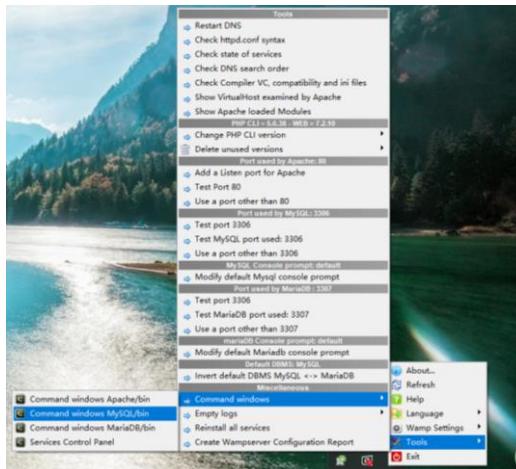


- Puis faites comme ci-dessous :



WampServer > MySQL>MySQL console

Si vous n'avez pas la même, faites ci-dessous sinon passez au suivant :



WampServer > Tools > Command window > Command windows MySql\bin

Ensuite pour exécuter mysql.exe, il faut taper la commande : mysql.exe -u root -p

Ensuite entrez.

- Création d'un utilisateur avec son mot de passe en exécutant la commande suivante :
SET PASSWORD for 'root'@'localhost' = password('mypass');
- Puis aller dans le répertoire : C:\wamp64\apps\phpmyadmin4.8.3 et chercher le fichier « config.inc.php », ouvrez avec Notepad++ ou autre éditeur. Et modifier comme ci-dessous :

```

<?php

/* Servers configuration */
$i = 0;

$cfg['blowfish_secret'] = 'h]C+{nqW$omNoTIkCwC$%z-LTcy%p6_j$|$Wv[mwngi~|e'; //What you
want

//Checking Active DBMS Servers
$wampConf = @parse_ini_file('../wampmanager.conf');
//Check if MySQL and MariaDB with MariaDB on default port
$mariaFirst = ($wampConf['SupportMySQL'] == 'on' && $wampConf['SupportMariaDB'] == 'on'
&& $wampConf['mariaPortUsed'] == $wampConf['mysqlDefaultPort']) ? true : false;
if($wampConf['SupportMySQL'] == 'on') {
/* Server: localhost [1] */
    $i++;
    if($mariaFirst) $i++;
    $cfg['Servers'][$i]['verbose'] = 'MySQL';
    $cfg['Servers'][$i]['host'] = '127.0.0.1';
    $cfg['Servers'][$i]['port'] = $wampConf['mysqlPortUsed'];
    $cfg['Servers'][$i]['extension'] = 'mysqli';
    $cfg['Servers'][$i]['auth_type'] = 'cookie';
    $cfg['Servers'][$i]['user'] = 'root';
    $cfg['Servers'][$i]['password'] = 'root';

    // Hidden databases in PhpMyAdmin left panel
    //$cfg['Servers'][$i]['hide_db'] =
    '(information_schema|mysql|performance_schema|sys)';

    // Allow connection without password
    $cfg['Servers'][$i]['AllowNoPassword'] = false;
}
/* Server: localhost [2] */
if($wampConf['SupportMariaDB'] == 'on') {
    $i++;

```

- Pour aller sur MyPhpAdmin, taper sur la barre URL : localhost et l'username : root et le mot de passe : mypass. Comme ci-dessous.

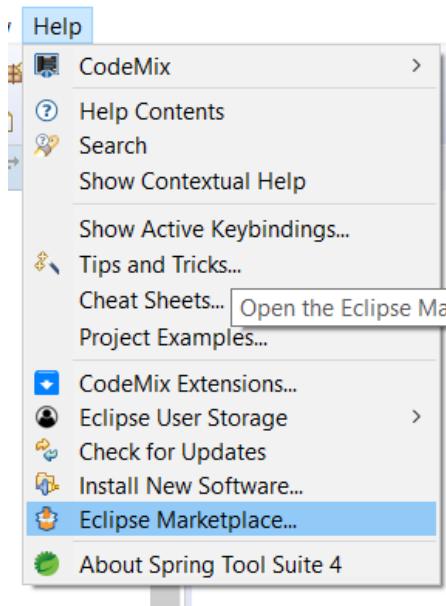


5.3. NodeJs

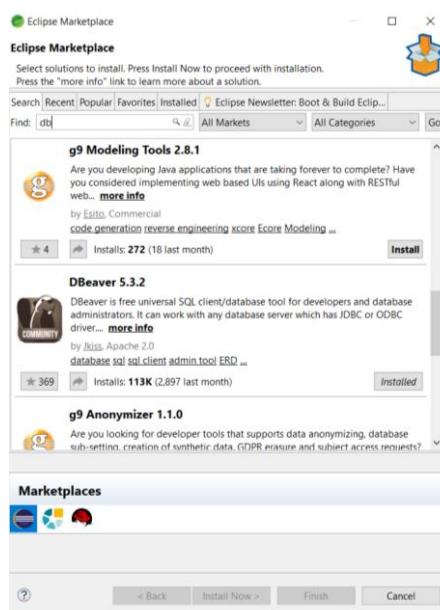
Installation de NodeJS depuis le site : <https://nodejs.org/en/download/>

5.4. DBeaver

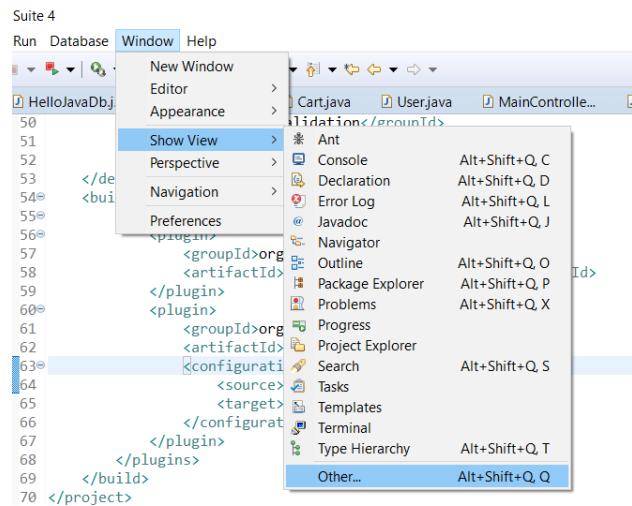
- Installation de DBeaver sur STS depuis Eclipse Marketplace depuis le menu Help > Eclipse Marketplace



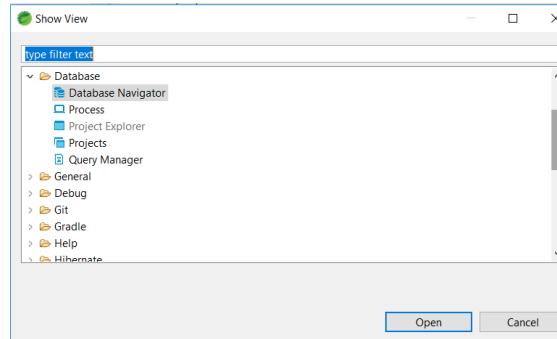
- Ensuite, chercher DBeaver dans la barre de recherche



- Puis redémarrer STS, et dans le menu Window > Show view puis chercher Database



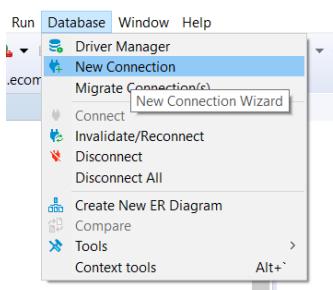
- Chercher dans la catégorie Database puis choisir Database Navigator



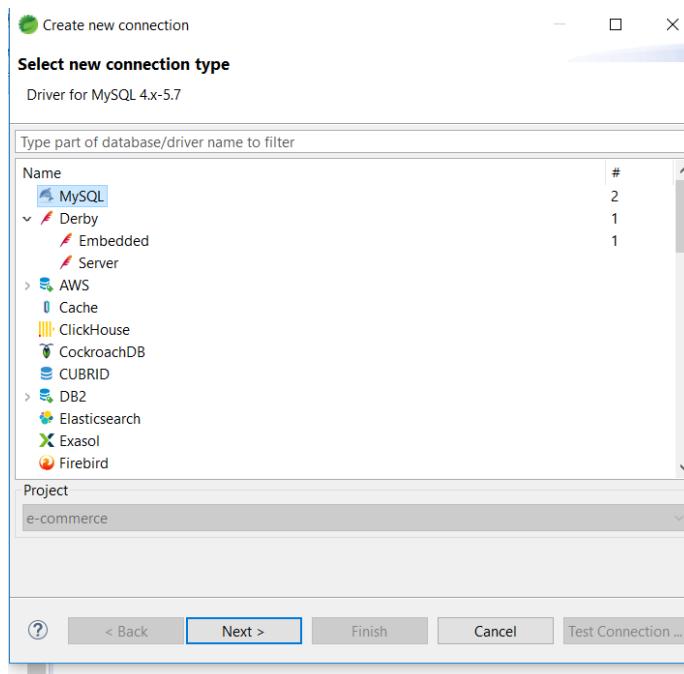
- Le résultat est le suivant :



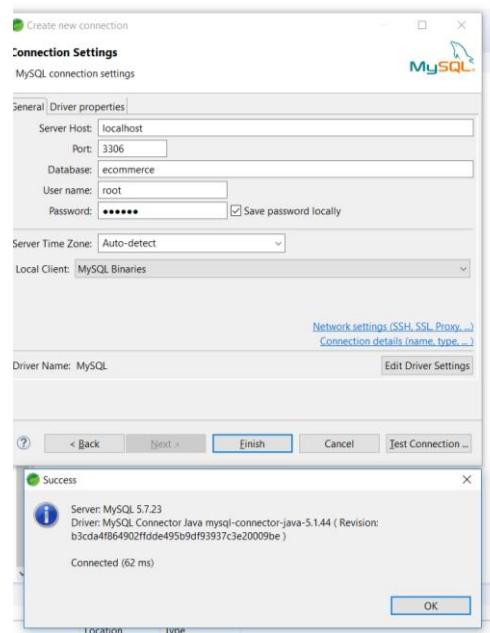
- Dans le menu Database > New Connection



- Choisissiez MySQL



- Et remplissez comme ci -dessous : User name : root et mot de passe : mypass



6. Application des technologies d'hibernate

6.1. Les annotations

Dans cette partie nous expliquant les annotations principales utilise dans notre application les autre sera dans des commentaire dans le code, que nous avons utilisé dans notre application.

6.2. @Controller

Annotation indiquant que la classe est une classe contrôleur.

6.3. `@RestController`

Une annotation de commodité qui est elle-même annotée avec `@Controller` et `@ResponseBody`. Utilisé dans les contrôleurs qui se comporteront comme des ressources RESTful.

6.4. `@RequestMapping`

Annotation à utiliser sur les méthodes des classes `@RestController`. Vous pouvez fournir une URI à servir comme service RESTful.

6.5. `@Service`

Annotation stéréotypée pour la couche service.

6.6. `@Repository`

Annotation stéréotypée pour la couche de persistance.

6.7. `@Component`

Annotation stéréotypée générique utilisée pour indiquer à Spring de créer une instance de l'objet dans le contexte de l'application. Il est possible de définir n'importe quel nom pour l'instance, la valeur par défaut est le nom de la classe.

6.8. `@Autowired`

Annotation utilisée pour injecter des objets de plusieurs façons possibles, telles que : variable d'instance, constructeur et méthodes. Il ne repose pas sur le nom comme `@Resource`, donc, pour des implémentations concrètes multiples, l'annotation `@Qualifier` doit être utilisée avec lui.

6.9. `@Configuration`

Annotation utilisée pour fournir les configurations.

```
@Entity
@Table
@Column
@Id
@Temporal
@Index
@ManyToOne
```

7. Réalisations

7.1. Architecture du projet

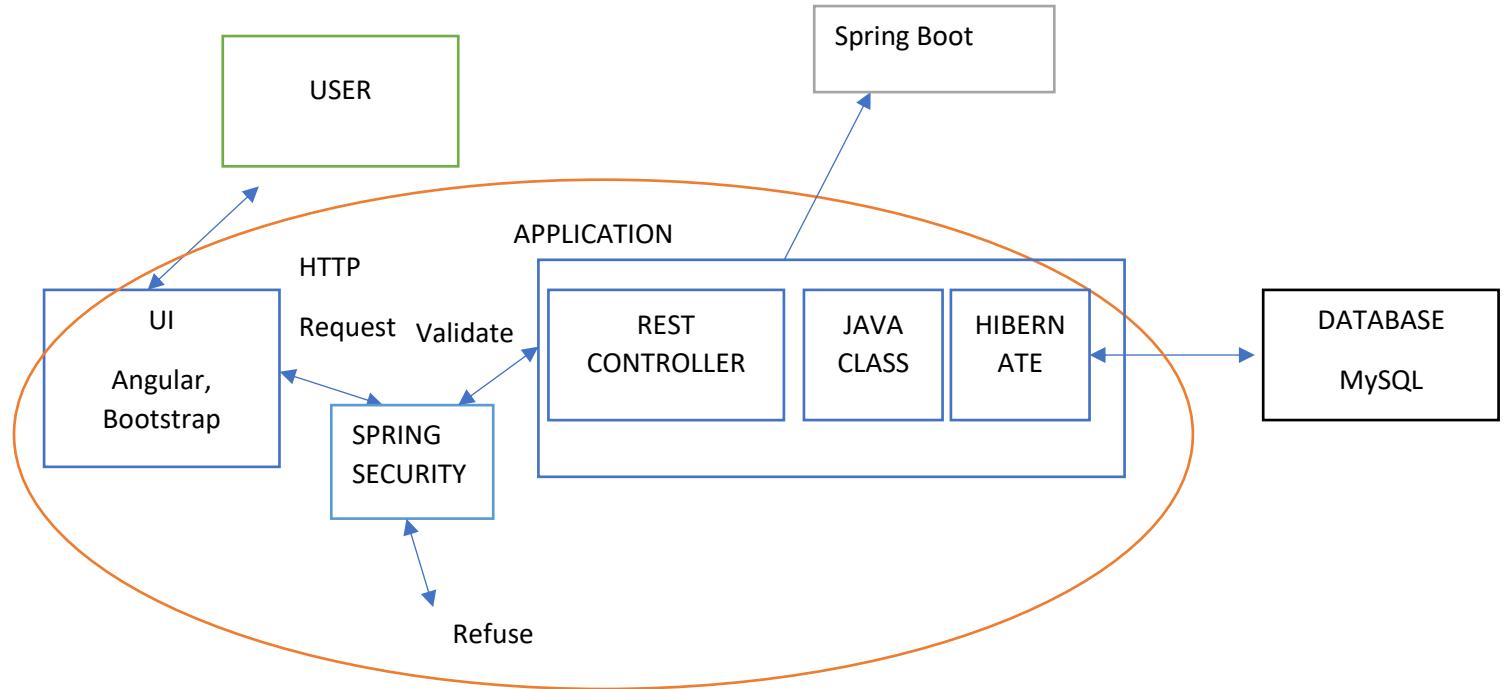


Figure 39: Architecture du projet

7.2. Back-end

7.2.1. Structure du projet Spring

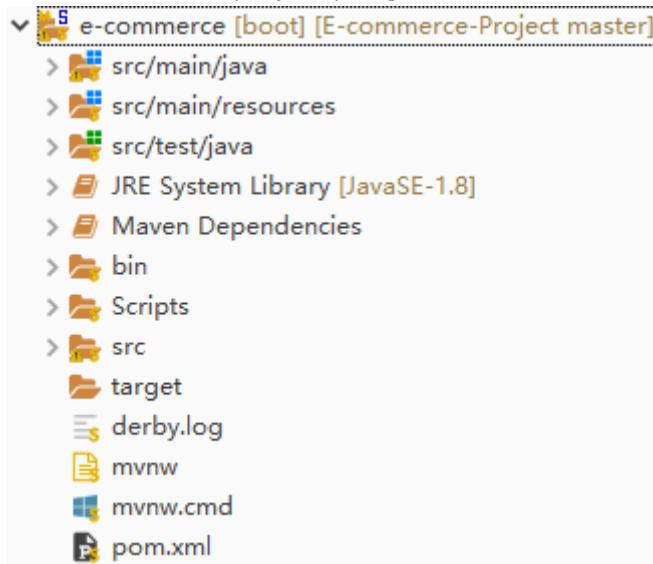


Figure 40: Structure d'un projet Spring

- Le package src/main/java contient toutes les classes de couche métier, DAO, contrôleur, service et
- Le package src/test/resources contient le fichier de configuration de l'application
- Le package src/test/java contient les tests unitaires

- Le dossier Scripts contient le script sql pour la création de la base de données et aussi des données des tables
 - Le fichier pom.xml contient les dépendances utilisées de notre application

7.2.2. Structure du package src/main/java

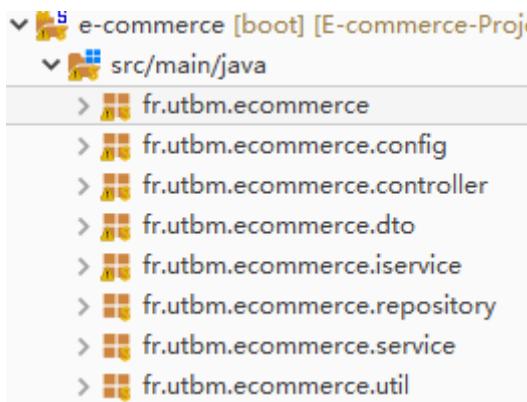


Figure 41: Structure du package src/main/java

7.2.2.1. Structure du package `fr.utbm.ecommerce`

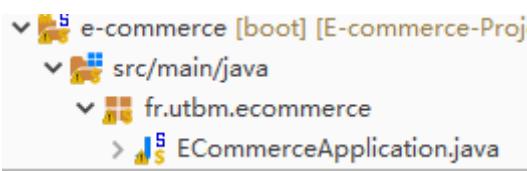


Figure 42: Structure du package `fr.utbm.ecommerce`

Ce package contient que l'application spring boot.

```
package fr.utbm.ecommerce;

import java.text.SimpleDateFormat;

@SpringBootApplication
//({scanBasePackages={"fr.utbm.ecommerce", "fr.utbm.ecommerce.repository", "fr.utbm.ecommerce.service", "fr.utbm.ecommerce.dto", "fr.utbm.ecommerce.config"})
public class ECommerceApplication implements CommandLineRunner {
    @Autowired
    CategoryService cs;
    @Autowired
    UserService us;

    public static void main(String[] args) {
        SpringApplication.run(ECommerceApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
    }
}
```

Figure 43: Code source de ECommerceApplication.java

7.2.2.2. Structure du package `fr.utbm.ecommerce.config`

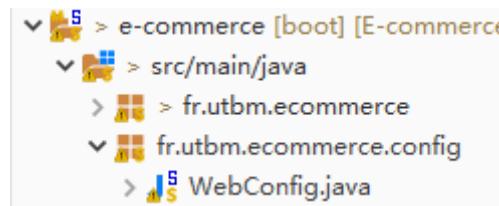


Figure 44: Structure du package `fr.utbm.ecommerce.config`

Ce package contient le fichier de configuration de Spring Sécurité.

```
// this configuration allow the client app to access the this api
// all the domain that consume this api must be included in the allowed o'rings
@SuppressWarnings("deprecation")
@Bean
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurerAdapter() {
        @Override
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**").allowedOrigins("http://localhost:4200");
        }
    };
}

// This method is used for override HttpSecurity of the web Application.
// We can specify our authorization criteria inside this method.
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and()
        // starts authorizing configurations
        .authorizeRequests()
        // ignoring the guest's urls"
        .antMatchers("/account/register", "/account/login", "/logout", "/account/save", "/account/users",
                    "/account/delete", "/category/categories", "/category/create", "/category/update",
                    "/category/delete", "/category/{id}", "/supplier/suppliers", "/supplier/create",
                    "/supplier/update", "/supplier/delete", "/product/products", "/product/create",
                    "/product/update", "/product/delete", "/product/fetch/{id}", "/order/create",
                    "/order/update", "/order/delete", "/order/{orderid}", "/cartitem/create",
                    "/cartitem/update", "/cartitem/delete", "/cartitem/order/items/{orderid}").permitAll()
        // authenticate all remaining URLs
        .anyRequest().fullyAuthenticated().and()
        /*
         * "/logout" will log the user out by invalidating the HTTP Session, cleaning up
         * any {link rememberMe()} authentication that was configured,
         */
        .logout().permitAll().logoutRequestMatcher(new AntPathRequestMatcher("/logout", "POST")).and()
        // enabling the basic authentication
        .httpBasic().and()
        // configuring the session on the server
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED).and()
        // disabling the CSRF - Cross Site Request Forgery
        .csrf().disable();
}
```

Figure 45 : `WebConfig.java` :

Ce fichier contient les fonctions qui permet d'autoriser le client à accéder à notre application. Et aussi de définir la liste des urls pour l'autorisation d'accès au server Spring Boot.

7.2.2.3. Structure du package `fr.utbm.ecommerce.controller`

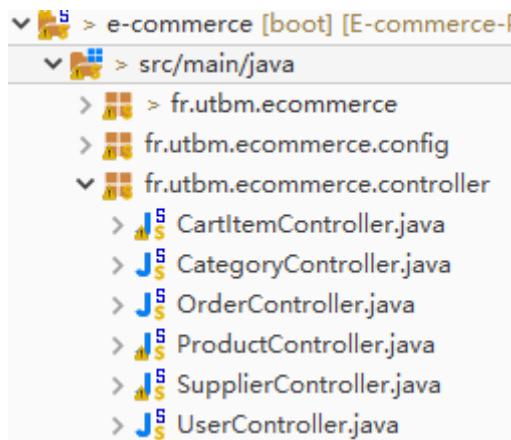


Figure 46: Structure du package `fr.utbm.ecommerce.controller`

Ce package contient tous les controller (REST Controller) qui sont responsable à la communication entre la partie Back end et Front end.

Ci-dessous un exemple d'UserController, les autres contrôleur sont pareils.

```
package fr.utbm.ecommerce.controller;

import java.security.Principal;

@RestController
@RequestMapping("account")
public class UserController {
    public static final Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserService userService;
    // this is the login api/service
    @CrossOrigin
    @RequestMapping(value="/login", method=RequestMethod.GET)
    public Principal user(Principal principal) {
        logger.info("user logged " + principal);
        return principal;
    }
    @CrossOrigin
    @RequestMapping(value="/users",method = RequestMethod.GET)
    public List<User> getUsers(Principal principal) {

        return userService.getUsers("ADMIN","WORKER");
    }
}
```

Figure 47: UserController.java

Le code est simple. Pour chaque URI « /account/users », avec la méthode d'appel (GET), on invoque la fonction `getUsers()` de la classe `UserController`. Cette fonction retourne la liste des utilisations (que les rôles avec Worker et Admin).

7.2.2.4. Structure du package `fr.utbm.ecommerce.dto`

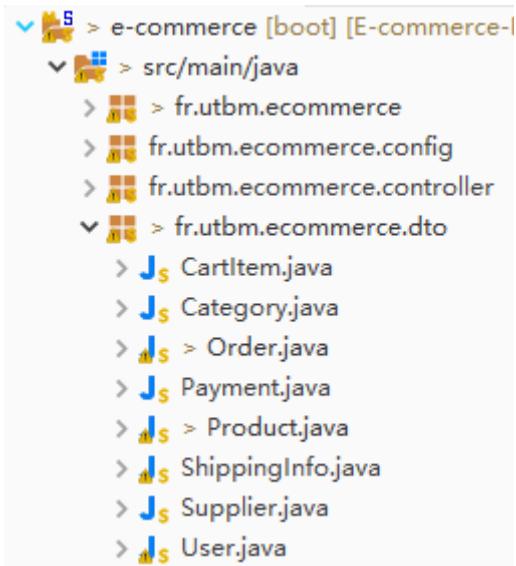


Figure 48: Structure du package `fr.utbm.ecommerce.dto`

Cette package contient toutes les classes de notre application ci-dessous un exemple de classe `Payment`, les autres classes ont les mêmes formes :

```

package fr.utbm.ecommerce.dto;

import java.io.Serializable;
@Entity
@Table(name="payment")
public class Payment implements Serializable{

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="PAYMENTID")
    private int Paymentid;
    @Temporal(TemporalType.DATE)
    @Column(name="PAYMENTDATE")
    private Date PaymentDate;
    @Column(name="METHODOFPAYMENT")
    private String MethodOfPayment;

    /**Getters and Setters**/

    public int getPaymentid() {
        return Paymentid;
    }
    public void setPaymentid(int paymentid) {
        Paymentid = paymentid;
    }
    public Date getPaymentDate() {
        return PaymentDate;
    }
    public void setPaymentDate(Date paymentDate) {
        PaymentDate = paymentDate;
    }
    public String getMethodOfPayment() {
        return MethodOfPayment;
    }
    public void setMethodOfPayment(String methodOfPayment) {
        MethodOfPayment = methodOfPayment;
    }
    @Override
    public String toString() {
        return "Payment [Paymentid=" + Paymentid +
               ", PaymentDate=" + PaymentDate +
               ", MethodOfPayment=" +
               + MethodOfPayment + "]";
    }
    public Payment(Date paymentDate, String methodOfPayment) {
        super();
        PaymentDate = paymentDate;
        MethodOfPayment = methodOfPayment;
    }

    public Payment() {
        super();
    }
}

```

Figure 49: Classe Payment.java

Toutes les classes contient des attributs et de getters et setters avec un constructeur par défaut et un constructeur et avec la méthode ToString().

7.2.2.5. Structure du package `fr.utbm.ecommerce.iservice`

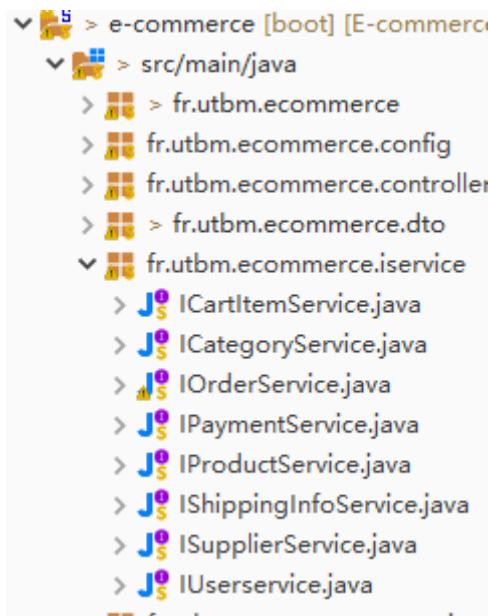
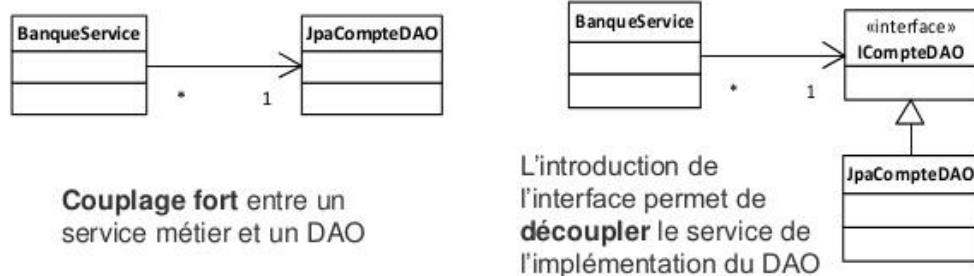


Figure 50: Structure du package `fr.utbm.ecommerce.iservice`

Ce package contient tous les interfaces que on a implémenté dans le package service. On utilise le concept d'inversion de contrôle (IOC).



Ci-dessous, un exemple de l'interface `IUserService`, les autres interfaces ont les mêmes formes. Chaque interface contient les méthodes : `addNomClass()`, `deleteNomClass()`, `updateNomClass()` et `getNomClassByExample()`.

```

package fr.utbm.ecommerce.iservice;

import java.util.Date;

public interface IUserservice {

    //update a user
    int updateUser(String firstname,String lastname,
                  String password,Date dateofbith,String gender,
                  String address,String country,String town,
                  String postalcode,String phonenumer,
                  String role,String mail);

    //add a user
    User addUser(User user);

    //delete a user
    void deleteUser(String mail);

    //get user by mail
    User getUserByMail(String mail);

    //get all users who has the role admin and worker
    List<User> getUsers(String admin,String worker);

    //get all users
    List<User> getAllUsers();
}

```

Figure 51: Interface IUserservice.java

7.2.2.6. Structure du package *fr.utbm.ecommerce.repository*

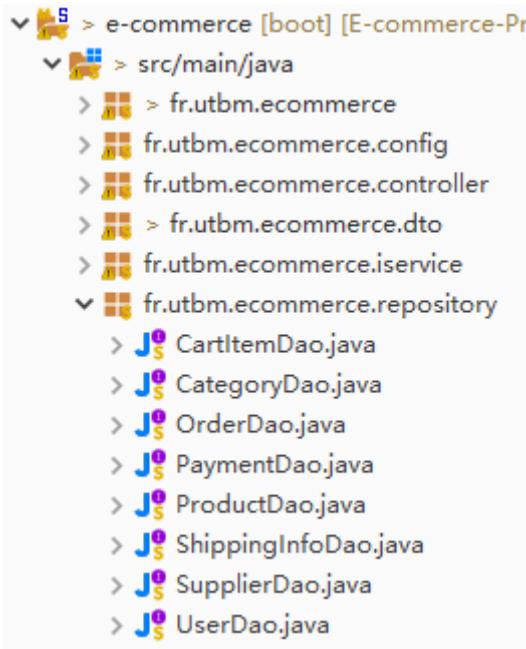


Figure 52: Structure du package fr.utbm.ecommerce.repository

Ce package contient toutes les interfaces DAO (Data Access Object) responsable à l'accès à la base de données.

Un exemple sur l'interface UserDao, les autres interfaces ont les mêmes formes.

```

package fr.utbm.ecommerce.repository;

import java.util.Date;[]

@Repository("UserDao")
@Transactional
public interface UserDao extends CrudRepository<User, Integer> {

    //get user by mail
    @Query("SELECT u FROM User u WHERE LOWER(u.Mail) LIKE LOWER(:mail) ")
    User getUserByMail(@Param("mail") String mail);

    //delete user by mail
    @Modifying
    @Query("DELETE FROM User WHERE LOWER(Mail) LIKE LOWER(:mail) ")
    int deleteUserByMail(@Param("mail") String mail);

    //get users who has role admin and worker
    @Query("SELECT u FROM User u WHERE LOWER(u.Role) LIKE"
           + " LOWER(:admin) OR LOWER(u.Role) LIKE LOWER(:worker) ")
    List<User> getUsers(@Param("admin") String admin,@Param("worker") String worker);

    //update user by mail
    @Modifying
    @Query("UPDATE User u set u.FirstName=:firstname,u.LastName=:lastname"
           + ",u.Password=:password,u.DateOfBirth=:dateofbirth,u.Gender=:gender"
           + ",u.Address=:address,u.Country=:country,u.Town=:town,u.PostalCode=:postalcode"
           + ",u.PhoneNumber=:phonenumber,u.Role=:role WHERE LOWER(u.Mail) LIKE LOWER(:mail) ")
    int updateUserByMail(@Param("firstname") String firstname,@Param("lastname") String lastname
                         ,@Param("password") String password,@Param("dateofbirth") Date dateofbirth
                         ,@Param("gender") String gender,@Param("address") String address
                         ,@Param("country") String country,@Param("town") String town
                         ,@Param("postalcode") String postalcode
                         ,@Param("phonenumber") String phonenumber
                         ,@Param("role") String role,@Param("mail") String mail);
}

```

Figure 53: UserDao.java

Dans notre projet les interfaces DAO sont héritées de la classe CrudRepository, qui contient les fonction par défaut(save(), Delete(), FindByID() etc...). Pour faire de CRUD méthodes avec le base de donne. Spring est responsable d'implémenter les méthodes si on a envie d'ajouter une méthode personnaliser, c'est tout simple on écrit la requête dans un annotation @Query et après on passe les variables dans les fonctions.

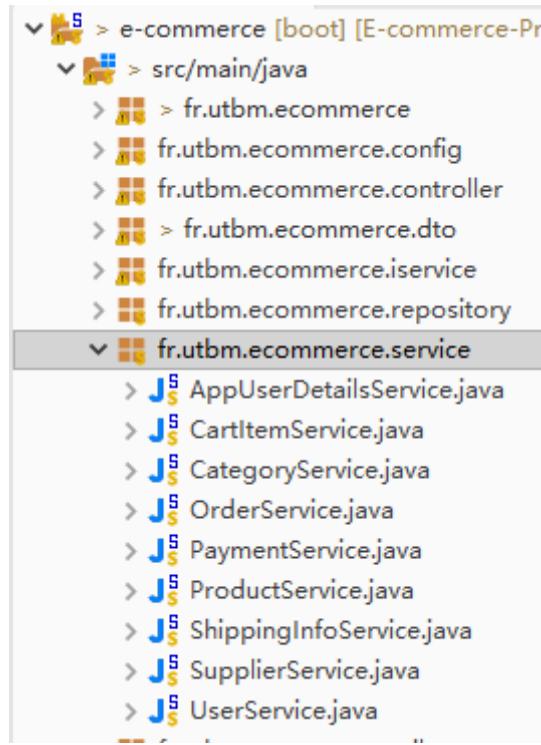
7.2.2.7. *Structure du package fr.utbm.ecommerce.service*

Figure 54: Structure du package fr.utbm.ecommerce.service

Ce package contient toutes les classes qui sont implémentées qu'on a définies dans le package iservice.

Ci-dessous un exemple de la classe UserService, les autres classes ont les mêmes formes.

```

package fr.utbm.ecommerce.service;

import java.util.Date;

@Service("UserService")
@Transactional
public class UserService implements IUserservice {
    @Autowired
    private UserDao userDao;

    //add a new user
    public User addUser(User user) {
        return userDao.save(user);
    }

    //update the user
    public int updateUser(String firstname,String lastname
        ,String password,Date dateofbith, String gender
        ,String address, String country, String town
        ,String postalcode, String phonenumerber,
        String role, String mail) {
        return userDao.updateUserByMail(firstname, lastname
            ,password, dateofbith, gender, address, country,
            town, postalcode, phonenumerber,role, mail);
    }

    //delete a user
    public void deleteUser(String mail) {
        userDao.deleteUserByMail(mail);
    }

    //get the user by mail
    public User getUserByMail(String mail) {
        // return userDao.findUserByMail(mail);
        User user = userDao.getUserByMail("%" + mail + "%");
        return user;
    }

    //get all users
    public List<User> getAllUsers() {
        return (List<User>) userDao.findAll();
    }

    //get users who has the role admin and worker
    @Override
    public List<User> getUsers(String admin, String worker) {
        return userDao.getUsers(admin,worker);
    }
}

```

Figure 55: UserService.java

7.2.2.8. Structure du package `fr.utbm.ecommerce.util`

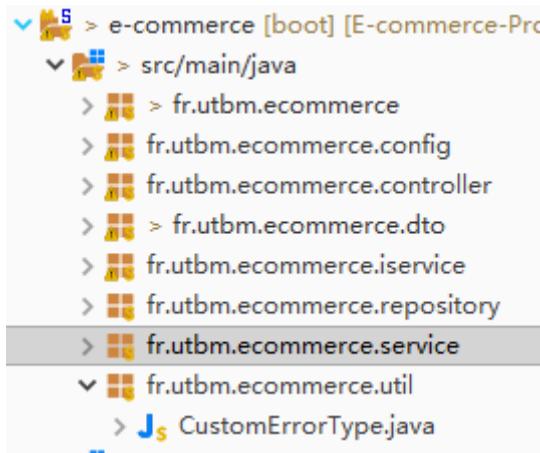


Figure 56: Structure du package `fr.utbm.ecommerce.util`

7.2.3. POM.xml

```
<!-- SPRING BOOT -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<!-- Spring Data jpa -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!-- Spring security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- Spring ORM -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
</dependency>
<!-- Hibernate -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.1.Final</version>
</dependency>
```

Figure 57: pom.xml

7.3. Front-end

7.3.1. Structure du projet Angular

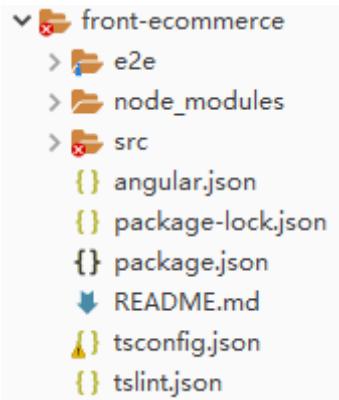


Figure 58: Structure du projet Angular

- Le dossier E2e comporte les fichiers de configuration du projet
- Le dossier node_module comporte les dépendances externes du projet
- Le dossier src comporte toute les codes sources de du projet
- Les autres fichiers correspondent aux fichiers de configuration.

7.3.2. Le dossier SRC

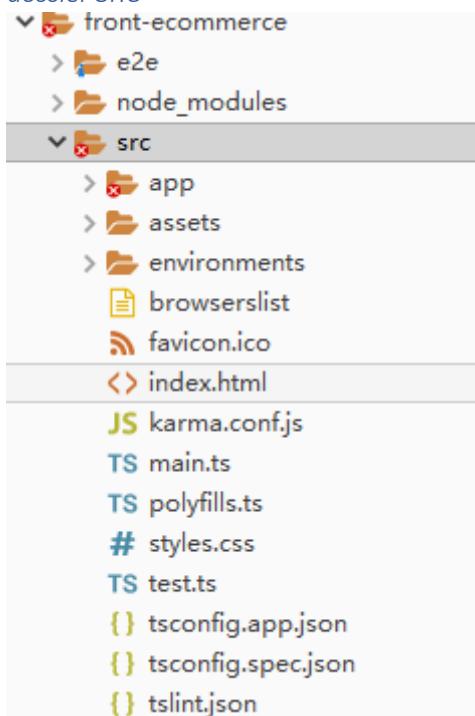


Figure 59: Structure du dossier SRC

- Le dossier app contient les fichiers sources et la logique métier du projet
- Le dossier assets contient toutes les fichiers ressources tel que les images, polices, ext...
- Le dossier environnements, on retrouve différents fichiers de configuration spécifiques aux environnements d'exécution.

- Les autres fichiers, ce sont des fichiers de paramétrage.

Nous s'intéressons que le dossier app. Car les autres dossiers nous avons pas fait des modifications.

7.3.3. Le dossier app

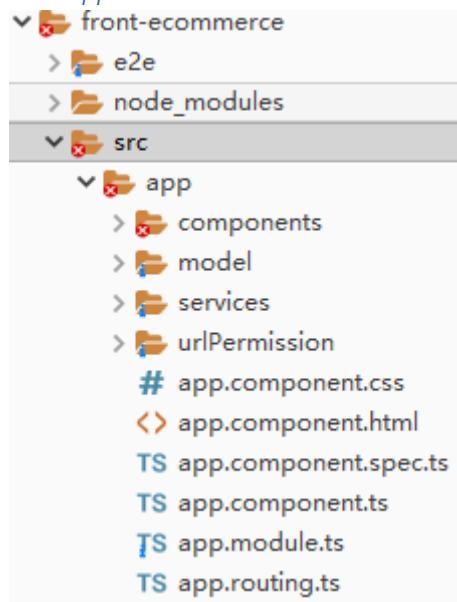


Figure 60: Structure du dossier app

- Le dossier components contient toutes les parties webs de l'application (c'est-à-dire partie IHM)
- Le dossier model contient les classes comme user, order, ... toutes les entités qu'on a créé dans le back-end
- Le dossier services

7.3.4. Le dossier components

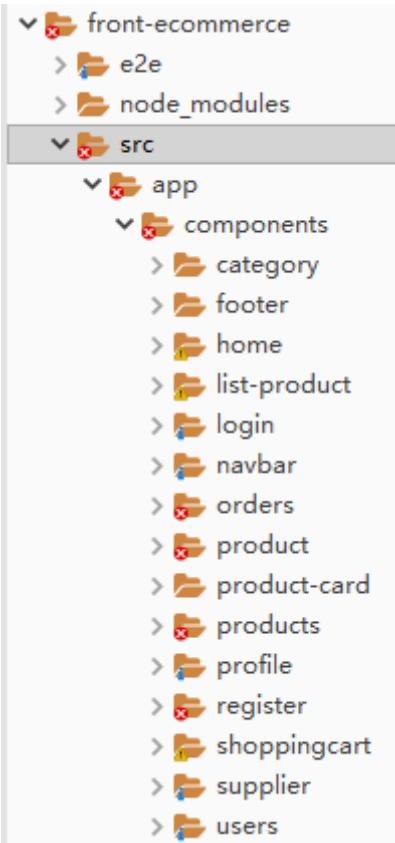


Figure 61: Structure du dossier components

Un component ou composant en français angular est de la forme de 4 fichiers : un fichier css, html, ts et spec.ts.

- Les fichiers spec sont des tests unitaires pour vos fichiers source. La convention pour les applications Angular2 est d'avoir un fichier .spec.ts pour chaque fichier .ts. Ils sont exécutés à l'aide du Framework de test javascript Jasmine via le programme de tâches Karma lorsque vous utilisez la commande ‘ng test’.
- Les fichiers .html contenant de code html ou bien la vue dans angular projet.
- Les fichiers .ts contient de code typescript ou bien le contrôleur dans angular projet.
- Chaque composant nous avons créé nous devons exporter et puis ajouter dans le fichier app.moule.ts. si on a créé le composant avec la commande ng g c nom_composant tous ces étapes précédent ils font automatiquement.

7.3.4.1. Category

```
<> category.component.html </>
1 <app-navbar></app-navbar>
2
3 <mdb-card style=" margin-top:100px;margin-bottom: 100px;" ngIf="currentCategory" class="text-center">
4   <mdb-card-header class="info-color white-text text-center py-4">
5     <h5>
6       <strong>Category Details</strong>
7     </h5>
8   </mdb-card-header>
9   <form style="color: #757575;">
10    <div class="md-form mt-3">
11      <input type="text" name="name" class="form-control" [(ngModel)]="currentCategory.name" mdbInputDirective required>
12      <label for="name">Category name : </label>
13    </div>
14
15    <div class="md-form">
16      <input type="text" name="description" class="form-control" [(ngModel)]="currentCategory.description" mdbInputDirective required>
17      <label for="description">Description : </label>
18    </div>
19  </form>
20  <mdb-card-footer class="justify-content-center">
21    <button type="button" mdbBtn color="secondary" class="waves-light aria-label="Close" (click)="update()" mdbWavesEffect>Update</button>
22    <button type="button" mdbBtn color="primary" class="relative waves-light" mdbWavesEffect (click)="delete(currentCategory)">Delete</button>
23  </mdb-card-footer>
24 </mdb-card>
25
```

Page modifier une catégorie

Code source ci-dessus, correspond à la page modifier une catégorie.

App-navbar correspond au composant navbar.

```
<h2 style="margin-top:100px;">List of Categories</h2>
<button type="button" mdbBtn gradient="blue" rounded="true" class="relative waves-light" (click)="basicModal.show()" mdbWavesEffect> Add </button><button type="button" mdbModal #basicModal="mdbModal" class="modal top" tabindex="-1" role="dialog" aria-labelledby="myBasicModalLabel" aria-hidden="true">
<div class="modal-dialog modal-lg modal-full-height modal-top" role="document">
  <div class="modal-content">
    <div class="modal-header">
      <button type="button" class="close pull-right" aria-label="Close" (click)="basicModal.hide()"></button>
      <span>Add new Category</span>
    </div>
    <div class="modal-body">
      <form class="text-center" style="color: #757575;">
        <div class="md-form mt-3">
          <input type="text" name="name" class="form-control" [(ngModel)]="newCategory.name" mdbInputDirective required>
          <label for="name">Category name : </label>
        </div>
        <div class="md-form">
          <input type="text" name="description" class="form-control" [(ngModel)]="newCategory.description" mdbInputDirective required>
          <label for="description">Description : </label>
        </div>
      </form>
      <div class="modal-footer justify-content-center">
        <button type="button" mdbBtn color="secondary" class="waves-light" aria-label="Close" (click)="basicModal.hide()" mdbWavesEffect>Close</button>
        <button type="button" mdbBtn color="primary" class="relative waves-light" mdbWavesEffect (click)="addCategory(newCategory);basicModal.hide()">OK!</button>
      </div>
    </div>
  </div>
</div>
<div class="table-responsive table-bordered" cellspacing="0">
  <table mdbTable #table class="table table-hover table-fixed table-striped">
    <!-- table head-->
    <thead>
      <tr>
        <th *ngFor="let head of headElements; let i = index" scope="col" [mdbTableSort]="'categories'" [sortBy]="headElements[i]"><(head)> <span>{{head}}</span> </th>
      </tr>
    <!--table head-->
  </thead>
  <!--table body-->
  <tbody>
    <tr class="table" ngFor="let category of categories; let i = index" [class.selected]="category === currentCategory">
      <th scope="row">{{category.categoryID}}</th>
      <td>{{category.name}}</td>
      <td>{{category.description}}</td>
      <td><a (click)="onEdit(category, i)" class="clickable">Edit </a></td>
      <td><a (click)="onDelete(category, i)" class="clickable">Delete </a></td>
    </tr>
  </tbody>
<!--table body-->
```

Page ajout d'une catégorie

Liste de catégorie

Figure 62: Code source HTML pour le composant category



List of Categories

ID	Name	Description	Edit	Delete
1	Monitor	Monitor	Edit	Delete
2	Motherboard	Motherboard	Edit	Delete
3	CPU	Microprocessor	Edit	Delete
4	Motherboard	Motherboard	Edit	Delete
5	Expansion cards	Expansion cards	Edit	Delete
6	Power supply unit	Power supply unit	Edit	Delete

Figure 63: Liste de catégories

The screenshot shows a 'Category Details' page with a blue header bar containing the TOB2 logo and navigation links for Home, Products, Shopping cart, and My Account. A search bar is also present. The main content area has a light blue header 'Category Details'. Below it, there's a table with one row containing 'Category name: Monitor' and 'Description: Monitor'. At the bottom are two buttons: 'UPDATE' (purple) and 'DELETE' (blue).

List of Categories

The screenshot shows a 'List of Categories' page with a blue header bar containing the TOB2 logo and navigation links for Home, Products, Shopping cart, and My Account. A search bar is also present. The main content area has a table with one row. The columns are labeled 'ID', 'Name', and 'Description'. The row contains the value '1' under 'ID', 'Monitor' under 'Name', and 'Monitor' under 'Description'. There are 'Edit' and 'Delete' buttons at the end of the row.

ID	Name	Description	Edit	Delete
1	Monitor	Monitor	Edit	Delete

Figure 64: Modifier une catégorie

Formulaire de modifier une catégorie.

The screenshot shows an 'Add new Category' form with a blue header bar containing the TOB2 logo and navigation links for Home, Products, Shopping cart, and My Account. A search bar is also present. The main content area has two input fields: 'Category name:' and 'Description:'. At the bottom are two buttons: 'CLOSE' (purple) and 'OK!' (blue).

Figure 65: Ajout d'une catégorie

Formulaire d'ajout d'une catégorie.

```
1 import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';
2 import { Category } from '../../../../../model/category.model';
3 import { CategoryService } from '../../../../../services/category.service';
4 import { Http } from '@angular/http';
5 import { Router } from '@angular/router';
6
7 @Component({
8   selector: 'app-category',
9   templateUrl: './category.component.html',
10  styleUrls: ['./category.component.css']
11})
12 export class CategoryComponent implements OnInit {
13
14   //list of categories
15   // 4 references
16   categories : any
17
18   // 4 references
19   currentCategory : Category
20
21   // 1 reference
22   newCategory : Category = new Category()
23
24   //error message
25   // 5 references
26   error : String
27
28   // 2 references
29   currentIndex : number;
30
31   //get element of the HTML page
32   // 0 references
33   @ViewChild('table') table : ElementRef
34
35   // 0 references | 0 references | 0 references | 4 references
36   constructor(public router : Router, public http: Http, public categoryService: CategoryService) {
37 }
38
39   // 2 references
40   ngOnInit() {
41     this.getCategories();
42   }
43 }
```

```
//get all categories
1 reference
getCategories(){
    this.categoryService.getCategories()
        .subscribe(data=>{
            //console.log(data);
            this.categories = data;
        },
        err=>{
            this.error = err;
            console.log(err);
        })
}

0 references
onEdit(category : Category, i : number){
    this.currentIndex = i;
    this.currentCategory = category
}

//delete a category
0 references
delete(category : Category, i : number){
    this.categoryService.deleteCategory(category)
        .subscribe(data=>{
            console.log(data);
            if(data==null){
                this.categories.splice(i,1);
                alert("Category deleted successful");
            }
        },
        err =>{
            this.error = err;
            alert(this.error);
        })
}

//add a category
0 references
addCategory(category : Category){
    this.categoryService.addCategory(category)
        .subscribe(data=>{
            console.log(data);
            this.categories.push(data);
            this.newCategory = new Category();
        },
        err=>{
            this.error = err;
            console.log(err);
        })
}
```

```
//update a category
0 references
update(){
  console.log(this.currentCategory)
  this.categoryService.updateCategory(this.currentCategory)
  .subscribe(data=>{
    console.log(data);
    this.categories[this.currentIndex] = this.currentCategory;
  },
  err=>{
    this.error = err;
    console.log(err);
  })
}
```

Figure 66: TS du composant categorie

7.3.4.2. Footer

```
<!-- Footer -->
<footer class="page-footer font-small blue-grey lighten-5">

    <div style="background-color: #21d192;">
        <div class="container">

            <!-- Grid row-->
            <div class="row py-4 d-flex align-items-center">

                <!-- Grid column -->
                <div class="col-md-6 col-lg-5 text-center text-md-left mb-4 mb-md-0">
                    <h6 class="mb-0">Get connected with us on social networks!</h6>
                </div>
                <!-- Grid column -->

                <!-- Grid column -->
                <div class="col-md-6 col-lg-7 text-center text-md-right">

                    <!-- Facebook -->
                    <a class="fb-ic">
                        <mdb-icon icon="facebook" class="white-text mr-4"></mdb-icon>
                    </a>
                    <!-- Twitter -->
                    <a class="tw-ic">
                        <mdb-icon icon="twitter" class="white-text mr-4"></mdb-icon>
                    </a>
                    <!-- Google -->
                    <a class="gplus-ic">
                        <mdb-icon icon="google-plus" class="white-text mr-4"></mdb-icon>
                    </a>
                    <!--Linkedin -->
                    <a class="li-ic">
                        <mdb-icon icon="linkedin" class="white-text mr-4"></mdb-icon>
                    </a>
                    <!--Instagram-->
                    <a class="ins-ic">
                        <mdb-icon icon="instagram" class="white-text mr-4"></mdb-icon>
                    </a>

                </div>
                <!-- Grid column -->

            </div>
            <!-- Grid row-->

        </div>
    </div>
```

```
<!-- Grid row -->
<div class="row mt-3 dark-grey-text">

  <!-- Grid column -->
  <div class="col-md-3 col-lg-4 col-xl-3 mb-4">

    <!-- Content -->
    <h6 class="text-uppercase font-weight-bold">T052</h6>
    <hr class="teal accent-3 mb-4 mt-0 d-inline-block mx-auto" style="width: 60px;">
    <p>Welcome !</p>
    </p>

  </div>
  <!-- Grid column -->

  <!-- Grid column -->
  <div class="col-md-2 col-lg-2 col-xl-2 mx-auto mb-4">

    <!-- Links -->
    <h6 class="text-uppercase font-weight-bold">Products</h6>
    <hr class="teal accent-3 mb-4 mt-0 d-inline-block mx-auto" style="width: 60px;">
    <p>
      <a class="dark-grey-text" href="#">MDBootstrap</a>
    </p>
    <p>
      <a class="dark-grey-text" href="#">MDWordPress</a>
    </p>
    <p>
      <a class="dark-grey-text" href="#">BrandFlow</a>
    </p>
    <p>
      <a class="dark-grey-text" href="#">Bootstrap Angular</a>
    </p>

  </div>
  <!-- Grid column -->

  <!-- Grid column -->
  <div class="col-md-3 col-lg-2 col-xl-2 mx-auto mb-4">

    <!-- Links -->
    <h6 class="text-uppercase font-weight-bold">Useful links</h6>
    <hr class="teal accent-3 mb-4 mt-0 d-inline-block mx-auto" style="width: 60px;">
    <p>
      <a class="dark-grey-text" [routerLink]=["'/profile'" routerLinkActive="router-link-active">Your Account</a>
    </p>
    <p>
      <a class="dark-grey-text" href="#">Help</a>
    </p>

  </div>
  <!-- Grid column -->
```

```

<!-- Grid column -->
<div class="col-md-4 col-lg-3 col-xl-3 mx-auto mb-md-0 mb-4">

    <!-- Links -->
    <h6 class="text-uppercase font-weight-bold">Contact</h6>
    <hr class="teal accent-3 mb-4 mt-0 d-inline-block mx-auto" style="width: 60px;">
    <p>
        <mdb-icon icon="home" class="mr-3"></mdb-icon> BELFORT
    </p>
    <p>
        <mdb-icon icon="envelope" class="mr-3"></mdb-icon> to52@example.com
    </p>
    <p>
        <mdb-icon icon="phone" class="mr-3"></mdb-icon> + 01 234 567 88
    </p>
    <p>
        <mdb-icon icon="print" class="mr-3"></mdb-icon> + 01 234 567 88
    </p>

    </div>
    <!-- Grid column -->

</div>
<!-- Grid row -->

</div>
<!-- Footer Links -->

</footer>
<!-- Footer -->

```

Figure 67: TS du composant footer

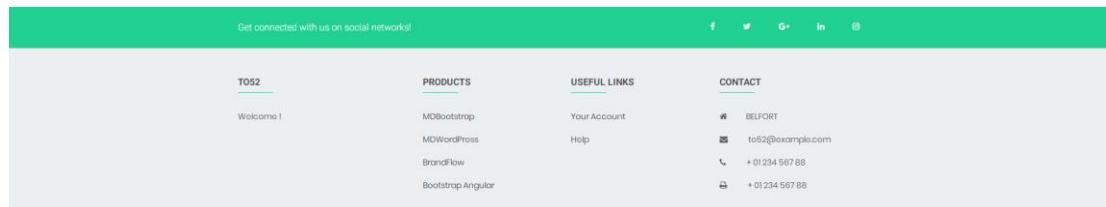


Figure 68: Page html footer

Le pied page qui va presque dans toute les autre pages sauf les pages pour les salariées et admin.

7.3.4.3. Home

```

<app-navbar></app-navbar>
<div class="container" style="margin-top:100px; margin-bottom: 100px;"></div>
<section class="text-center my-5">
<
    <!-- Section heading -->
    <h2 class="h1-responsive font-weight-bold text-center my-5">Our bestsellers</h2>
    <!-- Section description -->
    <p class="grey-text text-center w-responsive mx-auto mb-5">Lorem ipsum dolor sit amet, consectetur
    adipisciing elit. Fugit, error amet numquam iure provident voluptate esse quasi, veritatis totam voluptas
    nostrum quisquam eum porro a pariatur veniam.</p>
    <div class="container" style="margin-bottom: 10px; margin-right: 10px; margin-left: 10px; margin-top: 10px;">
    <div class="row">
        <div class="col-3 pb-4" *ngFor="let product of products">
            <a (click)=navigate(product.productID)>
                <app-product-card [product]="product"></app-product-card>
            </a>
        </div>
    </div>
    </div>
    <
    </section>
    <!-- Section: Products v.5 -->
    <
    <app-footer></app-footer>

```

Figure 69: Code source page home

```

1 import { Component, OnInit } from '@angular/core';
2 import {ProductService} from '../../services/product.service';
3 import {Http} from '@angular/http';
4 import {Router} from '@angular/router';
5
6 @Component({
7   selector: 'app-home',
8   templateUrl: './home.component.html',
9   styleUrls: ['./home.component.css']
10 })
11 export class HomeComponent implements OnInit {
12
13   //list of products
14   products :any;
15   error : String
16   constructor(public router : Router, public http : Http, public productService : ProductService) { }
17
18   ngOnInit() {
19     this.getProducts();
20   }
21
22   //get all products
23   getProducts(){
24     this.productService.getProducts()
25       .subscribe(data=>{
26         this.products = data;
27         console.log(data);
28       },
29       err=>{
30         this.error = err;
31         console.log(err);
32     })
33   }
34
35   //navigate to the product details page
36   navigate(id: number){
37     this.router.navigate(['/product/'+id]);
38   }
39 }
40

```

Figure 70: TS du composant home

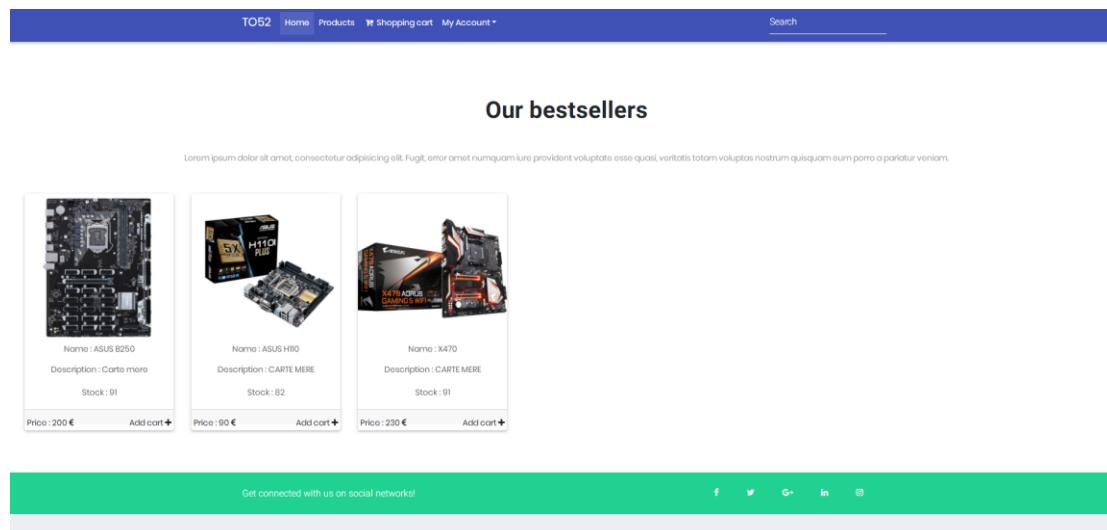


Figure 71: Page html home

Une page d'accueil qui permet d'afficher les meilleures ventes des produits.

7.3.4.4. List-product

```

1<app-navbar></app-navbar>
2<div class="container test-center" style="margin-bottom: 10px; margin-right: 10px; margin-left: 10px; margin-top: 10px;">
3<div class="row">
4    <div class="col-3 pb-4" *ngFor="let product of products">
5        <a (click)=navigate(product.productID)>
6            <app-product-card [product]= "product"></app-product-card>
7        </a>
8    </div>
9</div>
10</div>
11<app-footer></app-footer>

```

Figure 72: Code source de la page HTML liste des produits

```

1import { Component, OnInit } from '@angular/core';
2import { ProductService } from '../../../../../services/product.service';
3import { Http } from '@angular/http';
4import { Router } from '@angular/router';
5
6@Component({
7    selector: 'app-list-product',
8    templateUrl: './list-product.component.html',
9    styleUrls: ['./list-product.component.css']
10})
10 references
11export class ListProductComponent implements OnInit {
12
13    //list of products
13    1 reference
14    products :any;
14    1 reference
15    error : String
15    0 references | 1 reference | 0 references | 1 reference
16    constructor(public router : Router, public http : Http, public productService : ProductService) { }
17
17    2 references
18    ngOnInit() {
19        this.getProducts();
20    }
21
22    //get all products
22    1 reference
23    getProducts(){
24        this.productService.getProducts()
25            .subscribe(data=>{
26                this.products = data;
27                console.log(data);
28            },
29            err=>{
30                this.error = err;
31                console.log(err);
32            })
33    }
34
35    //navigate to the product details
35    0 references
36    navigate(id: number){
37        this.router.navigate(['/product/'+id]);
38    }
39}
40

```

Figure 73: TS du composant list-products

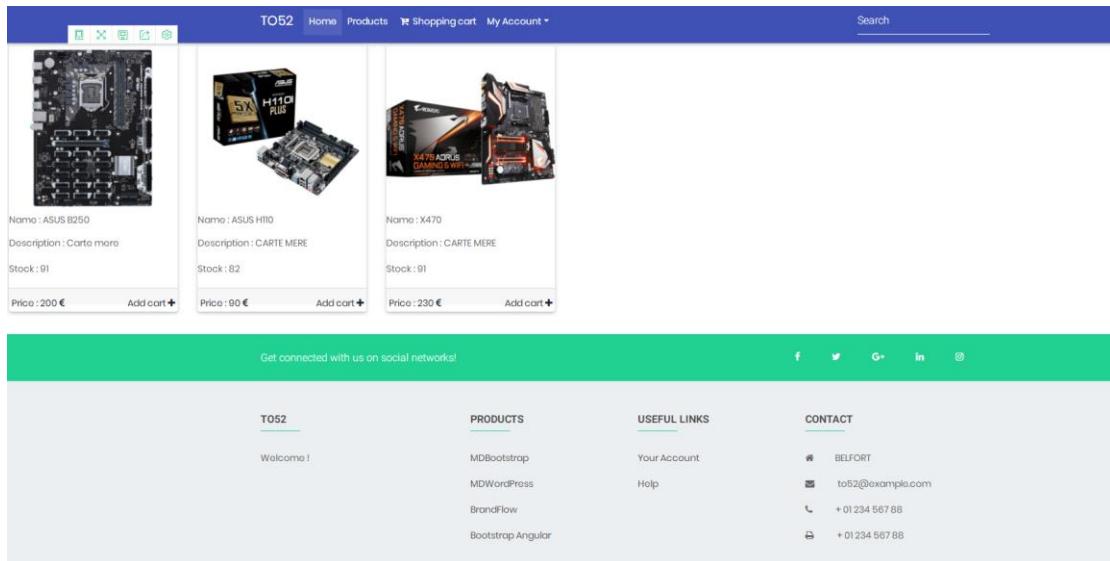


Figure 74: Page HTML liste des produits

Une page qui permet d'avoir toutes les produits qui existe.

7.3.4.5. Login

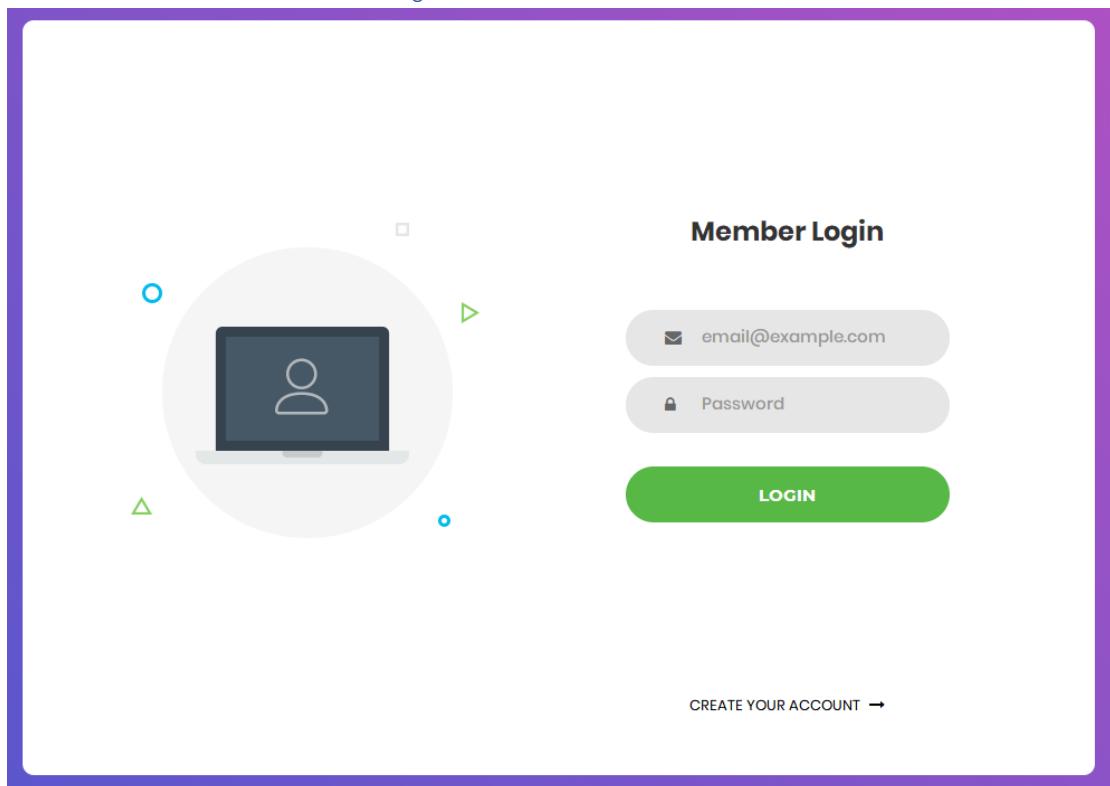


Figure 75: Page html login

```

1 <app-navbar></app-navbar>
2
3 <div class="limiter" >
4   <div class="container-login100" >
5     <div class="wrap-login100" >
6       <div class="login100-pic js-tilt" data-tilt>
7         
8       </div>
9
10      <form class="login100-form validate-form" (ngSubmit)="f.form.valid && login()" #f="ngForm" novalidate>
11        <span class="login100-form-title">
12          Member Login
13        </span>
14
15        <div class="wrap-input100 validate-input" data-validate = "Valid email is required: ex@abc.xyz">
16          <input class="input100" type="email" id="username" name="username" placeholder="email@example.com" pattern="[^ @]*@[^ @]*$ [(ngModel)]="user.mail" #username="ngModel" email required>
17          <span class="focus-input100"></span>
18          <span class="symbol-input100">
19            <i class="fa fa-envelope" aria-hidden="true"></i>
20          </span>
21        </div>
22        <div *ngIf="f.submitted && username.valid" class="wrap-input100 validate-input">Valid email is required: ex@abc.xyz</div>
23
24        <div class="wrap-input100 validate-input" data-validate = "Password is required">
25          <input class="input100" type="password" placeholder="Password" id="password" name="password" [(ngModel)]="user.password" #password="ngModel" required>
26          <span class="focus-input100"></span>
27          <span class="symbol-input100">
28            <i class="fa fa-lock" aria-hidden="true"></i>
29          </span>
30        </div>
31        <div *ngIf="f.submitted && password.valid" class="help-block">Password is required</div>
32
33        <div class="container-login100-form-btn" >
34          <button class="login100-form-btn" type="submit">
35            Login
36          </button>
37        </div>
38
39        <div class="text-center p-t-136" >
40          <a [routerLink]="/register" class="btn btn-link" >
41            Create your Account
42            <i class="fa fa-long-arrow-right m-1-5" aria-hidden="true"></i>
43          </a>
44        </div>
45      </form>
46    </div>
47  </div>
48</div>
49</div>
50

```

Figure 76: Code source de la page html login

```

1 import { Component, OnInit, ViewEncapsulation } from '@angular/core';
2 import { User } from '../../../../../model/user.model';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-login',
8   templateUrl: './login.component.html',
9   styleUrls: ['./login.component.css'],
10  encapsulation:ViewEncapsulation.None
11 })
12 export class LoginComponent implements OnInit {
13   user: User = new User();
14
15   errorMessage: string;
16   constructor(private authService: AuthService, private router: Router) { }
17
18   ngOnInit() {
19   }
20
21   //login
22   login(){
23     this.authService.logIn(this.user)
24       .subscribe(data=>{
25         this.router.navigate(['/home']);
26       },err=>{
27         this.errorMessage="error : Username or password is incorrect";
28       }
29     )
30   }
31
32 }
33

```

Figure 77: TS du composant login

Une page login qui permet de se logger.

7.3.4.6. Navbar

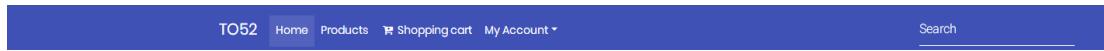


Figure 78: Page html du navbar

```

1<header class="header" style="--webkit-margin-top-collapse:collapse;">
2  <mdb-navbar SideClass="navbar navbar-expand-lg navbar-dark indigo fixed-top scrolling-navbar">
3    <mdb-navbar-brand>
4      <a class="navbar-brand" href="#"><strong>TO52</strong></a>
5    </mdb-navbar-brand>
6
7    <!-- Collapsible content -->
8    <links>
9
10   <!-- Links -->
11   <ul class="nav navbar-nav mr-auto">
12     <li class="nav-item active">
13       <a class="nav-link waves-light" [routerLink]="/home" routerLinkActive="router-link-active" mdbWavesEffect>Home<span class="sr-only">(current)</span></a>
14     </li>
15     <li class="nav-item">
16       <a class="nav-link waves-light" mdbWavesEffect [routerLink]="/product" routerLinkActive="router-link-active">Products</a>
17     </li>
18     <li class="nav-item">
19       <a class="nav-link waves-light" mdbWavesEffect [routerLink]="/shoppingcart" routerLinkActive="router-link-active" ><mdb-icon icon="cart-plus"></mdb-icon> Shopping cart</a>
20     </li>
21     <li #register class="nav-item">
22       <a class="nav-link waves-light" [routerLink]="/register" routerLinkActive="router-link-active" mdbWavesEffect>Register</a>
23     </li>
24     <li #login class="nav-item">
25       <a class="nav-link waves-light" [routerLink]="/login" routerLinkActive="router-link-active" mdbWavesEffect>
26         <span class="glyphicon glyphicon-log-in"></span> Log in</a>
27     </li>
28
29   <!-- Dropdown -->
30   <li #userMenu class="nav-item dropdown float-right" dropdown>
31     <a dropdownToggle mdbWavesEffect type="button" class="nav-link dropdown-toggle waves-light">
32       My Account<span class="caret"></span></a>
33     <div #dropdownMenu class="dropdown-menu dropdown dropdown-primary" role="menu">
34       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/profile" routerLinkActive="router-link-active">Profile</a>
35       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/orders" routerLinkActive="router-link-active">My orders</a>
36       <div class="divider dropdown-divider"></div>
37       <a class="dropdown-item waves-light" mdbWavesEffect (click)="logOut()">Log out</a>
38     </div>
39   </li>
40
41
42   <li #workerMenu class="nav-item dropdown float-right" dropdown>
43     <a dropdownToggle mdbWavesEffect type="button" class="nav-link dropdown-toggle waves-light">
44       My Account<span class="caret"></span></a>
45     <div #dropdownMenu class="dropdown-menu dropdown dropdown-primary" role="menu">
46       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/profile" routerLinkActive="router-link-active">Profile</a>
47       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/orders" routerLinkActive="router-link-active">My orders</a>
48       <div class="divider dropdown-divider"></div>
49       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/products" routerLinkActive="router-link-active">Add product</a>
50       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/categories" routerLinkActive="router-link-active">Add category</a>
51       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/suppliers" routerLinkActive="router-link-active">Add suppliers</a>
52       <div class="divider dropdown-divider"></div>
53       <a class="dropdown-item waves-light" mdbWavesEffect (click)="logOut()">Log out</a>
54     </div>
55   </li>
56
57
58   <li #adminMenu class="nav-item dropdown float-right" dropdown>
59     <a dropdownToggle mdbWavesEffect type="button" class="nav-link dropdown-toggle waves-light">
60       My Account<span class="caret"></span></a>
61     <div #dropdownMenu class="dropdown-menu dropdown dropdown-primary" role="menu">
62       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/profile" routerLinkActive="router-link-active">Profile</a>
63       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/orders" routerLinkActive="router-link-active">My orders</a>
64       <div class="divider dropdown-divider"></div>
65       <a class="dropdown-item waves-light" mdbWavesEffect [routerLink]="/users" routerLinkActive="router-link-active">Add user/worker</a>
66       <div class="divider dropdown-divider"></div>
67       <a class="dropdown-item waves-light" mdbWavesEffect (click)="logOut()">Log out</a>
68     </div>
69   </li>
70
71   <ul>
72     <!-- Links -->
73
74   <!-- Search form -->
75   <form class="form-inline waves-light" mdbWavesEffect>
76     <div class="md-form my-0">
77       <input class="form-control mr-sm-2" type="text" placeholder="Search">
78     </div>
79   </form>
80
81   </links>
82   <!-- Collapsible content -->
83 </mdb-navbar>
```

Figure 79: Code source de la page html navbar

```
1 import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';
2 import { User } from '../../../../../model/user.model';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-navbar',
8   templateUrl: './navbar.component.html',
9   styleUrls: ['./navbar.component.css']
10 })
11 0 references
12 export class NavbarComponent implements OnInit{
13   4 references
14   currentUser: User;
15
16   3 references
17   @ViewChild('register') idRegister: ElementRef
18
19   6 references
20   @ViewChild('login') idLogin : ElementRef
21
22   7 references
23   @ViewChild('userMenu') idMenuUser : ElementRef
24   5 references
25   @ViewChild('workerMenu') idMenuWorker : ElementRef
26   5 references
27   @ViewChild('adminMenu') idMenuAdmin : ElementRef
28
29   0 references | 1 reference | 1 reference
30   constructor(public authService: AuthService, public router: Router) {
31     }
32
33   4 references
34   ngOnInit() {
35     this.currentUser = JSON.parse(localStorage.getItem('currentUser'));
36     console.log(this.currentUser);
37
38     //to know if the user is worker, user or admin
39     if(this.currentUser === null){
40       if(this.idMenuUser.nativeElement.style.display === "" || this.idMenuUser.nativeElement.style.display === "inline-block"){
41         this.idMenuUser.nativeElement.style.display = "none";
42         this.idMenuWorker.nativeElement.style.display = "none";
43         this.idMenuAdmin.nativeElement.style.display = "none";
44
45       }
46       if(this.idLogin.nativeElement.style.display === "none"){
47         this.idLogin.nativeElement.style.display= "inline-block";
48         this.idRegister.nativeElement.style.display = "inline-block";
49       }
50     }else{
51      .. ...
52    }
53  }
```

```

42     }else{
43         switch(this.currentUser.role){
44             case "ADMIN":
45                 console.log("admin");
46                 this.idMenuWorker.nativeElement.style.display = "none";
47                 this.idMenuUser.nativeElement.style.display = "none";
48                 this.idMenuAdmin.nativeElement.style.display = "inline-block";
49             break;
50             case "WORKER":
51                 this.idMenuWorker.nativeElement.style.display = "inline-block";
52                 this.idMenuUser.nativeElement.style.display = "none";
53                 this.idMenuAdmin.nativeElement.style.display = "none";
54             break;
55             default:
56                 this.idMenuWorker.nativeElement.style.display = "none";
57                 this.idMenuUser.nativeElement.style.display = "inline-block";
58                 this.idMenuAdmin.nativeElement.style.display = "none";
59             }
60         }
61         if(this.idLogin.nativeElement.style.display==""){
62             this.idLogin.nativeElement.style.display="none";
63             this.idRegister.nativeElement.style.display="none";
64         }
65     }
66 }
67
68 // log out from the app
69 0 references
70 logOut() {
71     this.authService.logOut()
72     .subscribe(
73         data => {
74             this.idMenuWorker.nativeElement.style.display = "none";
75                 this.idMenuUser.nativeElement.style.display = "none";
76                 this.idMenuAdmin.nativeElement.style.display = "none";
77             if(this.idLogin.nativeElement.style.display=="none"){
78                 this.idLogin.nativeElement.style.display= "inline-block";
79                 this.idRegister.nativeElement.style.display = "inline-block";
80             }
81             this.router.navigate(['/home']);
82         },
83         error => {
84             console.log("error : "+ error);
85         });
86
87 }
88

```

Figure 80: TS du composant navbar

Un composant de navigateur qui va être dans tous les pages.

7.3.4.7. Orders

The screenshot shows a web page titled 'List of order'. At the top, there is a navigation bar with links for 'Home', 'Products', 'Shopping cart', and 'My Account'. A search bar is also present. Below the navigation, the page title 'List of order' is displayed. A table follows, with one row containing the data: Order reference (1), Date (2019-01-06), Amount (100€), and a 'View' button. Below the table, a green banner says 'Get connected with us on social networks!' with icons for Facebook, Twitter, Google+, LinkedIn, and Email.

Order reference	Date	Amount	
1	2019-01-06	100€	View

Figure 81: Page HTML de la liste des orders

En cliquant sur le bouton view, le détail de la commande s'affiche.

The screenshot shows a modal window titled 'Reference order : 1'. It contains a table with one row, showing details of an order item: Article line (1), Product ID (2), Product Reference (B250), Quantity (1), and Price (200€). At the bottom right of the modal is a purple 'CLOSE' button.

Article line	Product ID	Product Reference	Quantity	Price
1	2	B250	1	200€

Figure 82: Page html du détail de la commande

```

1 <app-navbar></app-navbar>
2 <div class="container">
3 <div mdbModal #basicModal="mdbModal" class="modal top" tabindex="-1" role="dialog" aria-labelledby="myBasicModalLabel"
4 aria-hidden="true">
5   <div class="modal-dialog modal-lg modal-full-height modal-top" role="document">
6     <div class="modal-content">
7       <div class="modal-header">
8         <button type="button" class="close pull-right" aria-label="Close" (click)="basicModal.hide()">
9           <mdb-icon icon="close"></mdb-icon>
10        </button>
11        <h4 class="modal-title w-100" id="myModalLabel">Reference order : {{reference}}</h4>
12      </div>
13      <div class="modal-body">
14        <div class="table-responsive table-bordered" cellspacing="0">
15          <tbl_struct version="1">
16            <tbl_header version="1">
17              <tbl_header version="1">
18                <tbl_header version="1">
19                  <tbl_header version="1">
20                    <tbl_header version="1">
21                      <tbl_header version="1">
22                        <tbl_header version="1">
23                          <tbl_header version="1">
24                            <tbl_header version="1">
25                              <tbl_header version="1">
26                                <tbl_header version="1">
27                                  <tbl_header version="1">
28                                    <tbl_header version="1">
29                                      <tbl_header version="1">
30                                        <tbl_header version="1">
31                                          <tbl_header version="1">
32                                            <tbl_header version="1">
33                                              <tbl_header version="1">
34                                                <tbl_header version="1">
35                                                  <tbl_header version="1">
36                                                    <tbl_header version="1">
37                                                      <tbl_header version="1">
38                                                        <tbl_header version="1">
39                                                          <tbl_header version="1">
40                                                            <tbl_header version="1">
41                                                              <tbl_header version="1">
42                                                                <tbl_header version="1">
43                                                                  <tbl_header version="1">
44                                                                    <tbl_header version="1">
45                                                                      <tbl_header version="1">
46                                                                        <tbl_header version="1">
47                                                                          <tbl_header version="1">
48                                                                            <tbl_header version="1">
49                                                                              <tbl_header version="1">
50                                                                                <tbl_header version="1">
51                                                                                  <tbl_header version="1">
52                                                                                    <tbl_header version="1">
53                                                                                      <tbl_header version="1">
54                        </tbl_header>
55                      </tbl_header>
56                    </tbl_header>
57                  </tbl_header>
58                </tbl_header>
59              </tbl_header>
60            </tbl_header>
61          </tbl_header>
62        </tbl_struct>
63        <table mdbTable #table class="table table-hover table-fixed table-striped">
64          <thead>
65            <tr>
66              <th>Article line</th>
67              <th>Product ID</th>
68              <th>Product Reference</th>
69              <th>Quantity</th>
70              <th>Price</th>
71            </tr>
72          </thead>
73          <tbody>
74            <tr *ngFor="let cartitem of listcartitem; let i = index">
75              <th>{{i+1}}</th>
76              <th scope="row">{{cartitem.product.productID}}</th>
77              <td>{{cartitem.product.reference}}</td>
78              <td>{{cartitem.quantity}}</td>
79              <td>{{cartitem.quantity * cartitem.product.price}}<span style="color: blue; font-weight: bold;">{{cartitem.product.icon}}

```

```

50 <h1><strong>List of order</strong></h1>
51 <div class="table-responsive table-bordered" cellspacing="0">
52   <table mdbTable #table class="table table-hover table-fixed table-striped">
53     <!-- table head-->
54     <thead>
55       <tr>
56         <th>Order reference</th>
57         <th>Date</th>
58         <th>Amount</th>
59       </tr>
60     <!--table head-->
61   </thead>
62
63   <!--table body-->
64   <tbody>
65     <tr class="table" *ngFor="let order of listOrder; let i = index">
66       <th>{{order.orderID}}</th>
67       <th scope="row">{{order.date}}</th>
68       <td >{{order.amount}} <mdb-icon icon="euro"></mdb-icon></td>
69       <td ><a (click)="view(order.orderID);basicModal.show()" class="clickable">View</a></td>
70     </tr>
71   </tbody>
72 <!--table body-->
73
74 </table>
75 </div>
76 <app-footer></app-footer>
```

Figure 83: Code source de la page html orders

```

1 import { Component, OnInit } from '@angular/core';
2 import {OrderService} from '../../../../../services/order.service';
3 import {User} from '../../../../../model/user.model';
4 import {Order} from '../../../../../model/order.model';
5 import {Http} from '@angular/http';
6 import {CartitemService} from '../../../../../services/cartitem.service';
7
8 @Component({
9   selector: 'app-orders',
10  templateUrl: './orders.component.html',
11  styleUrls: ['./orders.component.css']
12 })
13 export class OrdersComponent implements OnInit {
14
15   0 references | 0 references | 1 reference | 1 reference
16   constructor(public http : Http, public orderService : OrderService, public cartitemService : CartitemService) { }
17
18   2 references
19   user : User;
20
21   //list of orders
22   1 reference
23   listOrder : any;
24   1 reference
25   listcartitem : any;
26   1 reference
27   reference : any;
28
29   1 reference
30   ngOnInit() {
31     this.user = JSON.parse(localStorage.getItem("currentUser"));
32     this.getOrders();
33   }
34
35   //get all orders of the user
36   1 reference
37   getOrders(){
38     this.orderService.getOrderByUser(parseInt(this.user.userID)).subscribe(data=>{
39       this.listOrder = data;
40       console.log(data);
41     });
42   }
43 }
```

```

37  view(id : number){
38    this.reference = id;
39    this.cartitemService.getcartitemByOrder(id).subscribe(data=>{
40      this.listcartitem = data;
41      console.log(data);
42    });
43  }
44
45 }
46

```

Figure 84: TS du composant orders

Une page qui affiche toutes les commandes passées.

7.3.4.8. Product

Product Details	
Details of the ASUS H110	
Image	
Reference:	H110
Product description :	CARTE MERE
Supplier :	ASUS
Category :	MotherboardT
Price :	90€
VAT:	10%
Stock Available:	82
Add to cart:	ADD +

Figure 85: Page html détail d'un produit

Une page qui permet d'avoir les détails du produit.

```

1 import { Component, OnInit } from '@angular/core';
2 import {Product} from '../../../../../model/product.model';
3 import {ProductService} from '../../../../../services/product.service';
4 import {Http} from '@angular/http';
5 import {Router, ActivatedRoute} from '@angular/router';
6
7 @Component({
8   selector: 'app-product',
9   templateUrl: './product.component.html',
10  styleUrls: ['./product.component.css']
11})
12 export class ProductComponent implements OnInit {
13
14   1 reference
15   error : String;
16
17   1 reference
18   product : Product;
19
20   2 references
21   id : String
22   5 references
23   listCart : any;
24
25   0 references | 1 reference | 0 references | 0 references | 1 reference
26 constructor(public route : ActivatedRoute, public router : Router, public http : Http, public productService : ProductService) { }
27
28   2 references
29   ngOnInit() {
30     this.id = this.route.snapshot.paramMap.get('id');
31     this.getProduct(this.id);
32   }
33
34   0 references
35   //add a new product into cart
36   0 references
37   onAdd(product: Product){
38     var item = {id : product.productID, quantity : 1, ref: product.reference, price : product.price};
39     console.log(JSON.parse(localStorage.getItem("cart")));
40     this.listCart = JSON.parse(localStorage.getItem("cart"));
41     if(this.listCart==null){
42       this.listCart = [];
43     }
44     this.listCart.push(item);
45     localStorage.setItem("cart", JSON.stringify(this.listCart));
46   }
47
48 }
49
50 }
51
52 }
53
54 
```

Figure 86: TS du composant product

```

1<app-navbar></app-navbar>
2
3<div class="container" style="margin-top:100px; margin-bottom: 100px;">
4<h2>Product Details</h2>
5<p>Details of the {{product.name}}</p>
6<table class="table table-bordered">
7   <tbody>
8     <tr>
9       <td>Image</td>
10      <td></td>
11    </tr>
12    <tr>
13      <td>Reference :</td>
14      <td>{{product.reference}}</td>
15    </tr>
16    <tr>
17      <td>Product description :</td>
18      <td>{{product.description}}</td>
19    </tr>
20    <tr>
21      <td>Supplier :</td>
22      <td>{{product.supplier.firstName}}</td>
23    </tr>
24    <tr>
25      <td>Category :</td>
26      <td>{{product.category.name}}</td>
27    </tr>
28    <tr>
29      <td>Price :</td>
30      <td>{{product.price}} <span><img icon="euro"/></span></td>
31    </tr>
32    <tr>
33      <td>VAT :</td>
34      <td>{{product.vat}} <span><img icon="percent"/></span></td>
35    </tr>
36    <tr>
37      <td>Stock Available :</td>
38      <td>{{product.quantityStock}}</td>
39    </tr>
40  </tbody>
41</table>
42
43 
```

```
41      <tr>
42          <td>Add to cart :</td>
43          <td ><a (click)="onAdd(product)" class="clickable">ADD <mdb-icon icon="plus"></mdb-icon></a></td>
44      </tr>
45  </tbody>
46 </table>
47 </div>
48 <app-footer></app-footer>
```

Figure 87: Code source de la page détail du produit

7.3.4.9. Product-card



Figure 88: Page html pour un exemple de produit

Un composant qui permet d'avoir juste la présentation d'un produit.

```

1 import { Component, OnInit, Input } from '@angular/core';
2 import { Product } from '../../../../../model/product.model';
3
4 @Component({
5   selector: 'app-product-card',
6   templateUrl: './product-card.component.html',
7   styleUrls: ['./product-card.component.css']
8 })
9 export class ProductCardComponent implements OnInit {
10
11   @Input() product : Product;
12   listCart :any;
13
14   constructor() { }
15
16   ngOnInit() {
17
18   }
19
20   // add a product to the cart
21   onAdd(product: Product){
22     var item = {id : product.productID, quantity : 1, ref: product.reference, price : product.price};
23     console.log(JSON.parse(localStorage.getItem("cart")));
24     this.listCart = JSON.parse(localStorage.getItem("cart"));
25     if(this.listCart==null){
26       this.listCart = [];
27     }
28     this.listCart.push(item);
29     localStorage.setItem("cart", JSON.stringify(this.listCart));
30   }
31 }
32

```

Figure 89: TS du composant product-cart

```

1 <div class="card" *ngIf="product">
2   
3   <div class="card-block">
4     <p class="card-title">
5       Name : {{product.name}}
6     </p>
7     <p class="card-text">
8       Description : {{product.description}}
9     </p>
10    <p class="card-number">
11      Stock : {{product.quantityStock}}
12    </p>
13    <!-- Card footer -->
14    <div class="card-footer px-1">
15      <span class="float-left">Price : {{product.price}} <span class="float-right"><span> <span>
16      <a (click)="onAdd(product)" style="float:right" class="clickable">Add cart <span> <span>
17    </div>
18  </div>
19</div>

```

Figure 90: Code source pour un produit

7.3.4.10. Products



.list of Products

ID	Reference	Name	Description	Price	VAT	Stock	Supplier	Category	Edit	Delete
2	B250	ASUS B250	Carte mère	200 €	10 %	91	ASUS	MotherboardT	Edit	Delete
3	H110	ASUS H110	CARTE MERE	90€	10 %	82	ASUS	MotherboardT	Edit	Delete
4	X470T	X470	CARTE MERE	230 €	10 %	91	Gigabyte	Motherboard	Edit	Delete

Figure 91: Page html d'une liste de produit

Cette page est accessible que pour les utilisateurs worker.

Sur cette page on peut ajouter, modifier et supprimer des produits.

Add new Product X

Reference :

Name :

Description :

Price :

VAT :

Stock :

Color :

Size :

Picture name :

Category :

Supplier :

CLOSE OK!

Figure 92: Formulaire d'ajout d'un produit

Product Details

Reference : B250

Name : ASUS B250

Description : Carte mère

Price : 200

VAT : 10

Stock : 91

Color : 1

Size : 1

Picture name : 2

Category :

Supplier :

UPDATE DELETE

Figure 93: Formulaire d'édition d'un produit

```

1 import { Component, OnInit } from '@angular/core';
2 import { Product } from '../../../../../model/product.model';
3 import { ProductService } from '../../../../../services/product.service';
4 import { Http } from '@angular/http';
5 import { Router } from '@angular/router';
6 import { CategoryService } from '../../../../../services/category.service';
7 import { SupplierService } from '../../../../../services/supplier.service';
8
9 @Component({
10   selector: 'app-products',
11   templateUrl: './products.component.html',
12   styleUrls: ['./products.component.css']
13 })
14 export class ProductsComponent implements OnInit {
15
16   5 references
17   currentProduct : Product;
18   3 references
19   newProduct : Product = new Product();
20
21   4 references
22   products : any;
23   6 references
24   error : String;
25
26   1 reference
27   listCategories : any;
28
29   0 references
30   headElements = ["ID", "Reference", "Name", "Description", "Price", "VAT", "Stock", "Supplier", "Category"];
31
32   4 references
33   currentIndex : number;
34
35   0 references | 0 references | 0 references | 4 references | 1 reference | 1 reference
36   constructor(public router : Router, public http : Http, public productService : ProductService, public categoryService : CategoryService, public supplierService: SupplierService) {
37 }
38
39   2 references
40   ngOnInit() {
41     this.getProducts();
42     this.getProductCategories();
43     this.getSuppliers();
44   }
45
46   //get all products
47   1 reference
48   getProducts(){
49     this.productService.getProducts()
50       .subscribe(data=>{
51         this.products = data;
52         //console.log(data);
53       },
54       err=>{
55         this.error = err;
56         console.log(err);
57       })
58   }
59
60   //delete a product
61   0 references
62   deleteProduct(product : Product){
63     this.productService.deleteProduct(product)
64       .subscribe(data=>{
65         //console.log(data);
66         this.products.splice(this.currentIndex, 1);
67         alert("deleted");
68       },
69       err=>{
70         this.error = err;
71         console.log(err);
72       })
73   }
74
75   //add a new product
76   0 references
77   addProduct(){
78     console.log(this.newProduct);
79     this.productService.addProduct(this.newProduct)
80       .subscribe(data=>{
81         console.log(data);
82         this.products.push(data);
83       },
84
85       err=>{
86         this.error = err;
87         console.log(err);
88       })
89   }
90
91 }

```

```

83     0 references
83  onEdit(product : Product, index: number){
84      this.currentProduct = product;
85      this.currentIndex = index;
86      console.log(this.currentProduct);
87  }
88
89     0 references
89  onDelete(product : Product, index : number){
90      this.currentIndex = index;
91      this.currentProduct = product;
92  }
93
94     0 references
94  onNew(product : Product){
95      this新产品 = product;
96  }
97
98 //get all categories
99    1 reference
99  getCategories(){
100     this.categoryService.getCategories()
101        .subscribe(data=>{
102            this.listCategories = data;
103            //console.log(this.listCategories);
104        },
105        err=>{
106            this.error = err;
107            console.log(err);
108        })
109    }
110
111 //get all suppliers
112    1 reference
112  getSuppliers(){
113     this.supplierService.getSuppliers()
114        .subscribe(data=>{
115            this.listSuppliers = data;
116            //console.log(this.listSuppliers);
117        },
118        err=>{
119            this.error = err;
120            console.log(err);
121        })
122    }
123
124
125 //update a product
125    0 references
126  updateProduct(){
127      this.productService.updateProduct(this.currentProduct)
128        .subscribe(data=>{
129            this.products[this.currentIndex] = this.currentProduct;
130        },
131
132        err=>{
133            this.error = err;
134            console.log(err);
135        })
136    }
137
138 }
139

```

Figure 94: TS du composant products

```

1 navbar></app-navbar>
2
3 card style=" margin-top:100px;margin-bottom: 100px;" ngIf="currentProduct" class="text-center">
4 mdb-card-header class="info-color white-text text-center py-4">
5 h5
6 <strong>Product Details</strong>
7 /h5
8 db-card-header>
9 r= style="color: #757575;">
0   <div class="md-form mt-3">
1     <input type="text" name="reference" class="form-control" [(ngModel)]="currentProduct.reference" mdbInputDirective required>
2     <label for="reference">Reference : </label>
3   </div>
4   <div class="md-form">
5     <input type="text" name="name" class="form-control" [(ngModel)]="currentProduct.name" mdbInputDirective required>
6     <label for="name">Name : </label>
7   </div>
8   <div class="md-form">
9     <input type="text" name="description" class="form-control" [(ngModel)]="currentProduct.description" mdbInputDirective required>
10    <label for="description">Description : </label>
11  </div>
12  <div class="md-form">
13    <input type="number" name="price" class="form-control" [(ngModel)]="currentProduct.price" mdbInputDirective required>
14    <label for="price">Price : </label>
15  </div>
16  <div class="md-form">
17    <input type="number" name="vat" class="form-control" [(ngModel)]="currentProduct.vat" mdbInputDirective required>
18    <label for="vat">VAT : </label>
19  </div>
20  <div class="md-form">
21    <input type="number" name="stock" class="form-control" [(ngModel)]="currentProduct.quantityStock" mdbInputDirective required>
22    <label for="stock">Stock : </label>
23  </div>
24  <div class="md-form">
25    <input type="text" name="color" class="form-control" [(ngModel)]="currentProduct.color" mdbInputDirective required>
26    <label for="color">Color : </label>
27  </div>
28  <div class="md-form">
29    <input type="number" name="size" class="form-control" [(ngModel)]="currentProduct.size" mdbInputDirective required>
30    <label for="size">Size : </label>
31  </div>
32  <div class="md-form">
33    <input type="text" name="picture" class="form-control" [(ngModel)]="currentProduct.picture" mdbInputDirective required>
34    <label for="picture">Picture name : </label>
35  </div>
36  <div class="md-form">
37    <div class="row">
38      <div class="col-md-6">Category : </div>
39      <div class="col-md-6">
40        <select class="form-control" name="category" [(ngModel)]="currentProduct.category.categoryID" placeholder="Choose your category" required mdbInputDirective>
41          <option ngFor="let category of listCategories" [value]=>category.categoryID [attr.selected]=[&currentProduct.category.categoryID==category.categoryID? true: null]>{category.name}
42        </select>
43      </div>
44    </div>
45  </div>
46  <div class="md-form">
47    <div class="row">
48      <div class="col-md-6">Supplier : </div>
49      <div class="col-md-6">
50        <select class="form-control" name="supplier" [(ngModel)]="currentProduct.supplier.supplierID" placeholder="Choose your supplier" required>
51          <option ngFor="let supplier of listSuppliers" [value]=>supplier.supplierID [attr.selected]=[&currentProduct.supplier.supplierID==supplier.supplierID? true: null]>{supplier.name}
52        </select>
53      </div>
54    </div>
55  </div>
56 </div>
57 </div>
58 </div>
59 </div>
60 </div>
61 </div>
62 </div>
63 </div>
64 </div>
65 </div>
66 </div>
67 </div>
68 </div>
69 </div>
70 </div>
71 </div>
72 </div>
73 </div>
74 <div style="margin-top:100px;">List of Products</h2>
75 on type="button" mdbBtn gradient="blue" rounded="true" class="relative waves-light" aria-label="Close" (click)="basicModal.show()" mdbWavesEffect Add <mdb-icon icon="plus-square"></mdb-icon></button>
76 mdbModal #basicModal="mdbModal" class="modal top" tabindex="-1" role="dialog" aria-labelledby="myBasicModalLabel"
77 in-hidden="true"
78 div class="modal-dialog modal-lg modal-full-height modal-top" role="document"
79 <div class="modal-content">
80   <div class="modal-header">
81     <button type="button" class="close pull-right" aria-label="Close" (click)="basicModal.hide()>
82       <mdb-icon icon="close"></mdb-icon>
83     </button>
84     <h4 class="modal-title w-100" id="myBasicModalLabel">Add new Product</h4>
85   </div>
86   <div class="modal-body">
87     <form class="text-center" style="color: #757575;">
88       <div class="md-form mt-3">
89         <input type="text" name="reference" class="form-control" [(ngModel)]="newProduct.reference" mdbInputDirective required>
90         <label for="reference">Reference : </label>
91       </div>
92       <div class="md-form">
93         <input type="text" name="name" class="form-control" [(ngModel)]="newProduct.name" mdbInputDirective required>
94         <label for="name">Name : </label>
95       </div>
96       <div class="md-form">
97         <input type="text" name="description" class="form-control" [(ngModel)]="newProduct.description" mdbInputDirective required>
98         <label for="description">Description : </label>
99       </div>
100      <div class="md-form">
101        <input type="number" name="price" class="form-control" [(ngModel)]="newProduct.price" mdbInputDirective required>
102        <label for="price">Price : </label>
103      </div>
104      <div class="md-form">
105        <input type="number" name="vat" class="form-control" [(ngModel)]="newProduct.vat" mdbInputDirective required>
106        <label for="vat">VAT : </label>
107      </div>
108    </div>
109    <div>
110      <button type="button" mdbBtn color="secondary" class="waves-light" aria-label="Close" (click)="updateProduct()" mdbWavesEffect>Update</button>
111      <button type="button" mdbBtn color="primary" class="relative waves-light" mdbWavesEffect (click)="onDelete(currentProduct)">Delete</button>
112    </div>
113  </div>
114 </div>

```

```

111         <input type="number" name="stock" class="form-control" [(ngModel)]="newProduct.quantityStock" mdbInputDirective required>
112     </div>
113     <div class="md-form">
114         <input type="text" name="color" class="form-control" [(ngModel)]="newProduct.color" mdbInputDirective required>
115         <label for="color">Color : </label>
116     </div>
117     <div class="md-form">
118         <input type="number" name="size" class="form-control" [(ngModel)]="newProduct.size" mdbInputDirective required>
119         <label for="size">Size : </label>
120     </div>
121     <div class="md-form">
122         <input type="text" name="picture" class="form-control" [(ngModel)]="newProduct.picture" mdbInputDirective required>
123         <label for="picture">Picture name : </label>
124     </div>
125     <div class="md-form">
126         <div class="row">
127             <div class="col-md-6">Category : </div>
128         </div>
129         <div class="col-md-6">
130             <select class="form-control" name="category" [(ngModel)]="newProduct.categoryID" placeholder="Choose your category" required>
131                 <option *ngFor="let category of listCategories; let i = index" [attr.selected]="i==0?true:null" [value]={{category.name}}>{{category.name}}</option>
132             </select>
133         </div>
134     </div>
135     <div class="md-form">
136         <div class="row">
137             <div class="col-md-6">Supplier : </div>
138         </div>
139         <div class="col-md-6">
140             <select class="form-control" name="supplier" [(ngModel)]="newProduct.supplierID" placeholder="Choose your supplier" required>
141                 <option *ngFor="let supplier of listSuppliers; let j = index" [attr.selected]="j==0?true:null" [value]={{supplier.firstName}}>{{supplier.firstName}}</option>
142             </select>
143         </div>
144     </div>
145     </div>
146 </div>
147 </form>
148 </div>
149 </div>
150 </div>
151 <div class="table-responsive table-bordered" cellspacing="0">
152 <table mdbTable #table class="table table-hover table-fixed table-striped">
153     <!-- table head-->
154     <thead>
155         <tr>
156             <th *ngFor="let head of headElements; let i = index" scope="col" [mdbTableSort]="products" [sortBy]={{headElements[i]}}>{{head}} <mdb-icon icon="sort"></mdb-icon>
157         </th>
158     </tr>
159     <!-- table head-->
160     </thead>
161     <tbody>
162         <tr>
163             <th *ngFor="let head of headElements; let i = index" scope="col" [mdbTableSort]="products" [sortBy]={{headElements[i]}}>{{head}} <mdb-icon icon="sort"></mdb-icon>
164             </th>
165         </tr>
166     </tbody>
167     <!-- table body-->
168     <tbody>
169         <tr class="table" *ngFor="let Product of products; let i = index" [class.selected]={{Product === currentProduct}}>
170             <th scope="row">{{product.productID}}</th>
171             <td >{{product.reference}}</td>
172             <td >{{product.name}}</td>
173             <td >{{product.description}}</td>
174             <td >{{product.price}} <mdb-icon icon="euro"></mdb-icon></td>
175             <td >{{product.vat}} <mdb-icon icon="percent"></mdb-icon></td>
176             <td >{{product.quantityStock}}</td>
177             <td >{{product.supplier.firstName}}</td>
178             <td >{{product.category.name}}</td>
179             <td ><a (click)="onEdit(Product, i)" class="clickable">Edit <mdb-icon icon="edit"></mdb-icon></a></td>
180             <td ><a (click)="onDelete(Product,i)" class="clickable">Delete <mdb-icon icon="trash"></mdb-icon></a></td>
181         </tr>
182     </tbody>
183     <!-- table body-->
184 </tbody>
185     <!-- table body-->
186 </table>
187 
```

Figure 95: TS du composant products

7.3.4.11. Profile

The screenshot shows a user profile page titled "Profile of WORKER WORKER". The page is divided into two main sections: "Personal Information" and "Address".

Personal Information:

- First Name: WORKER
- Last Name: WORKER
- Mail: WORKER@WORKER.FR
- Password: WORKER
- Gender: (dropdown menu)
- Date of Birth: 年/月/日 (Year/Month/Day)

Address:

- Rue: Address
- Country: Country
- Town: Town
- Postal Code: Postal Code
- Phone Number: Phone Number

A green "SAVE" button is located at the bottom right of the form.

Figure 96: Page html du profil d'un utilisateur

La page qui permet d'avoir les informations de l'utilisateur.

```

1 import { Component, OnInit } from '@angular/core';
2 import { User } from '../../../../../model/user.model';
3 import { AuthService } from '../../../../../services/auth.service';
4 import { Router } from '@angular/router';
5 import { Http } from '@angular/http';
6 import { UpdateUserService } from '../../../../../services/updateUser.service';
7
8 @Component({
9   selector: 'app-profile',
10  templateUrl: './profile.component.html',
11  styleUrls: ['./profile.component.css']
12 })
13 export class ProfileComponent implements OnInit {
14   currentUser: User;
15   updateUser: User=new User();
16
17
18
19
20   constructor(public updateUserService:UpdateUserService,private authService: AuthService, public router: Router,public http: Http) {
21     this.currentUser = JSON.parse(localStorage.getItem('currentUser'));
22     this.updateUser.firstName=this.currentUser.firstName;
23     this.updateUser.lastName=this.currentUser.lastName;
24     this.updateUser.gender=this.currentUser.gender;
25     this.updateUser.password=this.currentUser.password;
26     this.updateUser.mail=this.currentUser.mail;
27     this.updateUser.address=this.currentUser.address;
28     this.updateUser.country=this.currentUser.country;
29     this.updateUser.town=this.currentUser.town;
30     this.updateUser.postalCode=this.currentUser.postalCode;
31     this.updateUser.phoneNumber=this.currentUser.phoneNumber;
32     this.updateUser.dateOfBirth=this.currentUser.dateOfBirth;
33
34 }
35
36   ngOnInit() {
37 }
38 // login out from the app
39 /*logout() {
40   this.authService.logout()
41   .subscribe(
42     data => {
43       this.router.navigate(['/login']);
44     },
45     error => {
46   });
47 });
48 }*/.
49   saveUser(){
50
51     this.updateUser.mail=this.currentUser.mail;
52     /*this.updateUser.password=this.currentUser.password;*/
53     this.updateUserService.saveUser(this.updateUser)
54     .subscribe(data=>{
55       console.log(data)
56     },err=>console.log(err));
57
58     this.authService.logIn(this.updateUser)
59     .subscribe(data=>{
60       this.router.navigate(['/profile']);
61     },err=>console.log(err));
62   }
63 }
64

```

Figure 97: TS du composant profile

```
5 <app-navbar>/<app-navbar>
6
7 <div class="container">
8   <div class="limiter">
9     <div class="container-fluid justify-content-md-between">
10
11   <header class="container my-4">
12     <h1 class="h1-responsive">
13       <strong>Profile of {{updateUser.firstName}} {{updateUser.lastName}} </strong> </h1>
14     <div class="row">
15   </div>
16 </header>
17
18 </div>
19 <div class="container"><br></div>
20
21
22 <form name="form-profile" >
23 <div class="container table-responsive-sm align-content-center align-content-md-center align-items-center">
24   <table class="table">
25     <tr><td colspan="2" ><h3>Personal Information</h3></td></tr>
26     <tr>
27       <td>
28         <label>
29           First Name
30         </label>
31       </td>
32
33       <td>
34         <input type="text" class="input100" placeholder="First name" [(ngModel)]="updateUser.firstName" id="firstname" name="firstname" >
35       </td>
36     <td>
37       <label>
38         Last Name
39       </label>
40     </td>
41
42       <td>
43         <input type="text" class="input100" placeholder="Last name" id="lastname" name="lastname" [ngModelOptions]="{standalone:true}" [(ngModel)]="updateUser.lastName" #lastName="ngModel" >
44       </td>
45     </tr>
46     <tr>
47       <td>
48         <label>
49           Mail
50         </label>
51       </td>
52
53       <td>
54         <input type="email" class="input100" placeholder="Email" value="{{currentUser.mail}}" readonly id="mail" name="mail" >
55       </td>
```

```

47      <td>
48        <label>
49          Mail
50        </label>
51      </td>
52
53      <td>
54        <input type="email" class="input100"    placeholder="Email" value="{{currentUser.mail}}"  readonly id="mail" name="mail" >
55      </td>
56      <td>
57        <label>
58          Password
59        </label>
60      </td>
61
62      <td>
63        <input type="text" class="input100"    placeholder="Passowrd" id="password" name="password" [(ngModel)]="updateUser.password" >
64      </td>
65    </tr>
66    <tr>
67      <td>
68        <label>
69          Gender
70        </label>
71      </td>
72
73      <td>
74        <label>
75          <select ng-model="selectedCar" class="input100" [(ngModel)]="updateUser.gender" [ngModelOptions] "{standalone:
76 true}" #gender="ngModel" >
77            <option value="male" class="input100" >Male</option>
78            <option value="female" class="input100" >Female</option>
79          </select>
80        </label>
81      </td>
82      <td>
83        <label>
84          Date of Birth
85        </label>
86      </td>
87
88      <td>
89        <input type="date" class="input100"    placeholder="JJ/MM/AAAA"  [ngModelOptions] "{standalone:
90 true}" [(ngModel)]="updateUser.dateOfBirth" #dateOfBirth="ngModel" id="dateofbirth" name="dateofbirth" >
91
92      </td>
93    </tr>
94    <tr><td colspan="2" ><h3>Address</h3></td></tr>
95    <tr>
96      <td>
97        <label>
98          Rue
99        </label>
100       </td>
101
102      <td>
103        <input type="text" class="input100"    placeholder="Address" [(ngModel)]="updateUser.address" [ngModelOptions] "{standalone:
104 true}" #address="ngModel" id="address" name="address" >
105      </td>
106
107      <td>
108        <label>
109          Country
110        </label>
111      </td>
112      <input type="text" class="input100"    placeholder="Country" [(ngModel)]="updateUser.country" [ngModelOptions] "{standalone:
113 true}" #country="ngModel" id="country" name="country" >
114
115    </tr>
116    <tr>
117      <td>
118        <label>
119          Town
120        </label>
121      </td>
122
123      <td>
124        <input type="text" class="input100"    placeholder="Town" [(ngModel)]="updateUser.town" #town="ngModel" [ngModelOptions] "{standalone:
125 true}" id="town" name="town" >
126
127      <td>
128        <label>
129          Postal Code
130        </label>
131      </td>
132
133      <td>
134        <input type="text" class="input100"    placeholder="Postal Code" [(ngModel)]="updateUser.postalCode" [ngModelOptions] "{standalone:
135 true}" #postalcode="ngModel" id="postalcode" name="postalcode" >
136
137    </tr>
138    <tr>
139      <td>
140        <label>
141          Phone Number
142        </label>
143      </td>
144
145      <td>
146        <input type="tel" class="input100"    placeholder="Phone Number" value="{{currentUser.phoneNumber}}" [(ngModel)]="updateUser.phoneNumber" [ngModelOptions] "{standalone:
147 true}" #phonenumber="ngModel" id="phonenumber" name="phonenumber" >
148
149    </td>
150
151      <td>
152        </label>
153      </td>

```

▲ CodeMix V

```

145   <td>
146     <input type="tel" class="input100" placeholder="Phone Number" value="{{currentUser.phoneNumber}}" [(ngModel)]="updateUser.phoneNumber" [ngModelOptions]={standalone:
147       true}> #phoneNumber="ngModel" id="phonenumber" name="phonenumber" >
148   </td>
149   <td>
150     <label>
151       </label>
152     </td>
153   </td>
154   <td>
155     <div class="container-login100-form-btn">
156       <button class="login100-form-btn" (click)="saveUser()">
157         Save
158       </button>
159     </div>
160     <!-- <button class="btn btn-primary btn-block" (click)="saveUser()" >Save</button>-->
161   </td>
162 </tr>
163
164
165 </table>
166 </div>
167 </div>
168 </form>
169 </div></div>
170

```

Figure 98: Code source de la page profile

7.3.4.12. Register

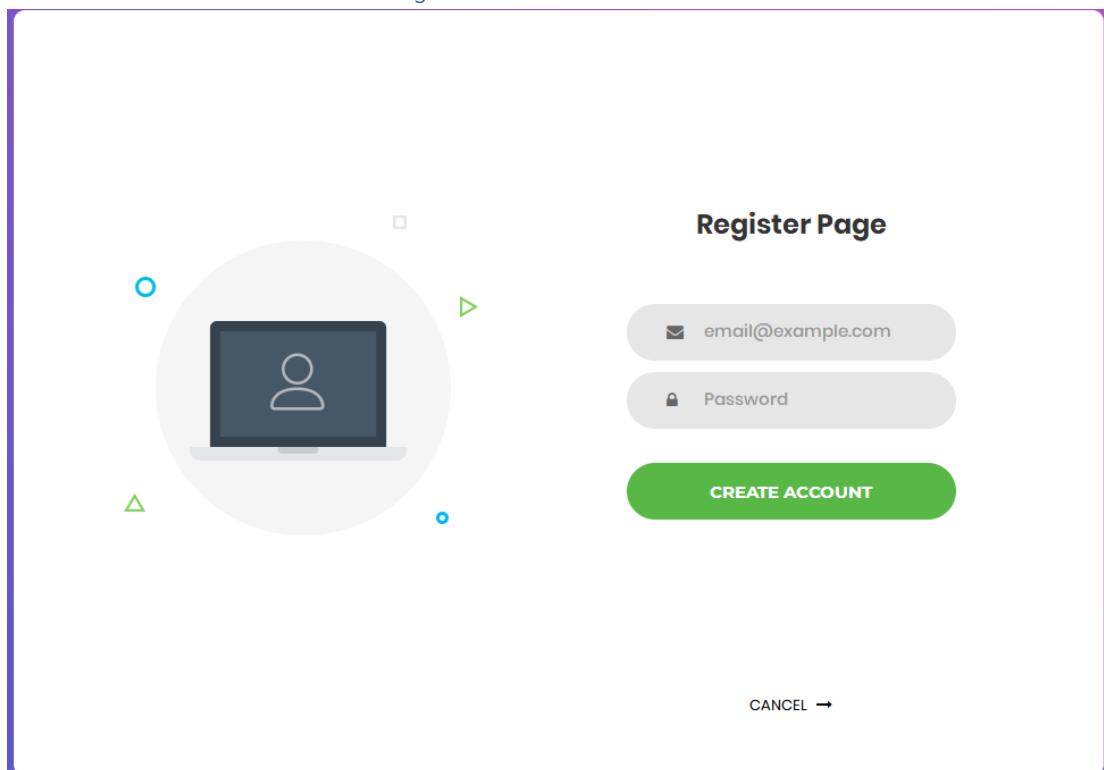


Figure 99: Page html register

Une page qui permet de se créer un compte.

```

1 <app-navbar></app-navbar>
2 <div class="limiter" >
3   <div class="container-login100">
4     <div class="wrap-login100">
5       <div class="login100-pic js-tilt" data-tilt>
6         
7       </div>
8
9       <form class="login100-form validate-form" (ngSubmit)="f.form.valid && register()" #f="ngForm" novalidate>
10      <span class="login100-form-title">
11        Register Page
12      </span>
13
14      <div class="wrap-input100 validate-input" data-validate = "Valid email is required: ex@abc.xyz">
15        <input class="input100" type="email" id="username" name="username" placeholder="email@example.com" pattern="[^@]+@[^\n]*" [(ngModel)]="user.mail" #username="ngModel" email required>
16        <span class="focus-input100"></span>
17        <span class="symbol-input100">
18          <i class="fa fa-envelope" aria-hidden="true"></i>
19        </span>
20      </div>
21      <div *ngIf="f.submitted && !username.valid" class="help-block">
22        <div *ngIf="username.errors.required">
23          Email is required.
24        </div>
25        <div *ngIf="username.errors.pattern">
26          A valid email address is required
27        </div>
28      </div>
29
30      <div class="wrap-input100 validate-input" data-validate = "Password is required">
31        <input class="input100" type="password" placeholder="Password" id="password" name="password" [(ngModel)]="user.password" #password="ngModel" required>
32        <span class="focus-input100"></span>
33        <span class="symbol-input100">
34          <i class="fa fa-lock" aria-hidden="true"></i>
35        </span>
36      </div>
37      <div *ngIf="f.submitted && !password.valid" class="help-block">
38        <div *ngIf="password.errors.required">
39          Password is required.
40        </div>
41        <div *ngIf="password.errors.minLength">
42          Password must be at least 4 characters long.
43        </div>
44      </div>
45
46      <div class="container-login100-form-btn">
47        <button class="login100-form-btn" type="submit">
48          Create Account
49        </button>
50      </div>
51    </div>
52
53    <div class="text-center p-t-136">
54      <a [routerLink]="/login" class="btn btn-link">Cancel
55        <i class="fa fa-long-arrow-right m-l-5" aria-hidden="true"></i>
56      </a>
57    </div>
58  </div>
59 </form>
60 </div>
61 </div>
62 </div>
63
64

```

Figure 100: Code source de la page register

```
1 import { Component, OnInit } from '@angular/core';
2 import { User } from '../../../../../model/user.model';
3 import { AccountService } from '../../../../../services/account.service';
4 import { Router } from '@angular/router';
5
6 @Component({
7   selector: 'app-register',
8   templateUrl: './register.component.html',
9   styleUrls: ['./register.component.css']
0 })
0 references
1 export class RegisterComponent implements OnInit {
2   references
2   user: User = new User();
1   reference
3   errorMessage: string;
0 references | 1 reference | 1 reference
4   constructor(public accountService: AccountService, public router: Router) { }
5
2 references
6   ngOnInit() {
7   }
0 references
8   register() {
9
0   //this.user.role="USER";
1
2     console.log(this.user);
3     this.accountService.createAccount(this.user).subscribe(data => {
4       console.log(data);
5       this.router.navigate(['/login']);
6     }, err => {
7       console.log(err);
8       this.errorMessage = "username already exist";
9     }
0   )
1 }
2 }
3
```

Figure 101: TS du composant register

The screenshot shows a shopping cart page with a table of items. The table has columns for Article line, Product ID, Product Reference, Quantity, and Price. There are also '+' and '-' buttons to change the quantity, and a trash icon to remove items. A 'Total' row shows a total price of 810 €. A blue 'DO ORDER' button is located at the bottom right of the table. Below the table, there's a green footer bar with social media links (f, t, G+, in, e) and a 'Get connected with us on social networks!' message. At the very bottom, there's a grey footer with links for 'TO52', 'PRODUCTS', 'USEFUL LINKS', and 'CONTACT'.

Figure 102: Page html du panier

Une page panier qui permet d'avoir les produits qui veut acheter et d'ajouter, diminuer ou enlever un produit.

```

1 <app-navbar></app-navbar>
2 <div class="container">
3   <h1>Shopping Cart</h1>
4   <div class="table-responsive table-bordered" cellspacing="0">
5     <table mdbTable #table class="table table-hover table-fixed table-striped">
6       <!-- table head-->
7       <thead>
8         <tr>
9           <th>Article line</th>
10          <th>Product ID</th>
11          <th>Product Reference</th>
12          <th>Quantity</th>
13          <th>Price</th>
14        </tr>
15        <!--table head-->
16      </thead>
17
18      <!--table body-->
19      <tbody>
20        <tr class="table" *ngFor="let Product of listCart; let i = index">
21          <th>{{i+1}}</th>
22          <th scope="row">{{Product.id}}</th>
23          <td>{{Product.ref}}</td>
24          <td>{{Product.quantity}}</td>
25          <td>{{Product.quantity*Product.price}} <mdb-icon icon="euro"></mdb-icon></td>
26          <td><a (click)="plus(i)" class="clickable"><mdb-icon icon="plus"></mdb-icon></a></td>
27          <td><a (click)="minus(i)" class="clickable"><mdb-icon icon="minus"></mdb-icon></a></td>
28          <td><a (click)="remove(i)" class="clickable"><mdb-icon icon="trash"></mdb-icon></a></td>
29        </tr>
30      </tbody>
31      <tfoot>
32        <tr>
33          <td> <strong>Total</strong></td>
34          <td></td>
35          <td></td>
36          <td></td>
37          <td> <strong> {{total}} </strong><strong> {{total}} </strong></td>
38          <td><button class="btn btn-primary" type="button" (click)=doOrder()>
39            Do Order
40          </button></td>
41        </tr>
42      </tfoot>
43      <!--table body-->
44
45    </table>
46  </div>
47 <app-footer></app-footer>
```

Figure 103: Code source de la page panier

```

1 import { Component, OnInit } from '@angular/core';
2 import { ProductService } from '../../../../../services/product.service';
3 import { Http } from '@angular/http';
4 import { Router } from '@angular/router';
5 import { ShoppingCartService } from '../../../../../services/shoppingcart.service';
6 import { OrderService } from '../../../../../services/order.service';
7 import { Product } from '../../../../../model/product.model';
8 import { Order } from '../../../../../model/order.model';
9 import { User } from '../../../../../model/user.model';
10 import { Payment } from '../../../../../model/payment.model';
11 import { ShippingInfo } from '../../../../../model/shippinginfo.model';
12 import { CartItem } from '../../../../../model/cartitem.model';
13
14 @Component({
15   selector: 'app-shoppingcart',
16   templateUrl: './shoppingcart.component.html',
17   styleUrls: ['./shoppingcart.component.css']
18 })
19 export class ShoppingCartComponent implements OnInit {
20
21   listCart : any;
22   total : any;
23   error : String;
24   listProduct : any;
25   product : Product;
26   order : Order = new Order();
27
28   payment : Payment;
29
30   user : User;
31
32   shippingInfo : Shippinginfo;
33
34   cartitem : Cartitem;
35
36   constructor(public router : Router, public shoppingCartService : ShoppingCartService, public productService: ProductService, public http : Http, public orderService : OrderService) { }
37
38   ngOnInit() {
39     this.listCart = JSON.parse(localStorage.getItem("cart"));
40     this.calculTotal();
41     this.productService.getProducts()
42       .subscribe(data =>{
43         this.listProduct = data;
44       });
45   }
46

```

```
0 references
47 plus(i: number){
48     this.listCart[i].quantity += 1;
49     this.calculTotal();
50     localStorage.setItem("cart", JSON.stringify(this.listCart));
51 }
52
0 references
53 minus(i : number){
54     if(this.listCart[i].quantity<=1){
55         this.remove(i);
56     }else{
57         this.listCart[i].quantity -= 1;
58         this.calculTotal();
59         localStorage.setItem("cart", JSON.stringify(this.listCart));
60     }
61 }
62
1 reference
63 remove(i: number){
64     this.listCart.splice(i, 1);
65     this.calculTotal();
66     localStorage.setItem("cart", JSON.stringify(this.listCart));
67 }
68
4 references
69 calculTotal(){
70     this.total = 0;
71     if(this.listCart !==null){
72         this.listCart.forEach(element=> {
73             this.total += element.quantity*element.price;
74         });
75     }
76 }
77
1 reference
78 find(id){
79     this.product = null;
80     this.listProduct.forEach(element=>{
81         if(element.productID == id){
82             this.product= element;
83         }
84     })
85 }
```

```

86     0 references
87     doOrder(){
88         if(this.listCart.length>0){
89             var products = [];
90             this.error = "";
91             this.listCart.forEach(element=> {
92                 this.find(element.id);
93                 if(this.product.quantityStock>=element.quantity && this.product.quantityStock>0){
94                     this.product.quantityStock-= 1;
95                     products.push(this.product);
96                 }else{
97                     this.error += element.ref + " not enough quantity, "
98                 }
99             });
100            if(this.error === ""){
101                products.forEach(element =>{
102                    this.productService.updateProduct(element)
103                    .subscribe(data=>{
104                        console.log("updated");
105                    });
106                });
107            }
108
109            this.user = JSON.parse(localStorage.getItem("currentUser"));
110            this.order.user = this.user.userID;
111            this.order.amount = this.total;
112            this.order.date = new Date();
113            this.order.payment = "1";
114            this.order.shippingInfo = "1";
115
116            console.log(this.order);
117            this.orderService.addorder(this.order).subscribe(data=>{
118                console.log(data);
119                this.order = data;
120            });
121
122            this.listCart.forEach(element =>{
123                this.cartitem.quantity = element.quantity;
124                this.cartitem.order = this.order.id;
125                this.cartitem.product = element.id;
126                this.shoppingCartService.addcartitem(this.cartitem)
127                .subscribe(data=>{
128                    console.log(data);
129                })
130            });
131            localStorage.removeItem("cart");
132        }else{
133            console.log(this.error);
134        }
135    }
136 }
137 }

```

Figure 104: TS du composant shoppingcart

7.3.4.14. Supplier

List of Suppliers

ID	First name	Last name	Mail	Phone number	Town		
3	ASUS					Edit	Doloto
4	Gigabyte					Edit	Doloto

Figure 105: Page html Liste de fournisseur

Une page qui permet d'avoir une liste de fournisseur.

Add new Supplier X

First name :

Last name :

Mail :

Phone number :

Date :

Address :

ZIP Code :

Town :

Country :

CLOSE OK!

Figure 106: Page HTML ajout d'un fournisseur

Supplier Details

First name :

Last name :

Mail :

Phone number :

Date :

Address :

ZIP Code :

Town :

Country :

UPDATE DELETE

Figure 107 : Page html modifier un fournisseur

Cette liste est accessible que pour le salarié.

```
1 import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';
2 import { Supplier } from '../../../../../model/supplier.model';
3 import { Router } from '@angular/router';
4 import { Http } from '@angular/http';
5 import { SupplierService } from '../../../../../services/supplier.service';
6
7 @Component({
8   selector: 'app-supplier',
9   templateUrl: './supplier.component.html',
10  styleUrls: ['./supplier.component.css']
11})
12 export class SupplierComponent implements OnInit {
13
14   //list of suppliers
15   suppliers : any
16   currentSupplier : Supplier
17   newSupplier : Supplier = new Supplier()
18
19   //error message
20   error : String
21
22   currentIndex : number;
23
24   @ViewChild('table') table : ElementRef
25
26   headElements = ["ID", "First name", "Last name", "Mail", "Phone number", "Town"];
27
28 constructor(public router: Router, public http : Http, public supplierService : SupplierService) { }
29
30   ngOnInit() {
31     this.getSuppliers();
32   }
--
```

```
//get all suppliers
1 reference
getSuppliers(){
    this.supplierService.getSuppliers()
    .subscribe(data=>{
        //console.log(data);
        this.suppliers = data;
    },
    err =>{
        this.error = err;
        console.log(err);
    })
}

//edit
0 references
onEdit(supplier : Supplier, i : number){
    this.currentIndex = i;
    this.currentSupplier = supplier;
}

//delete
0 references
onDelete(supplier : Supplier, i : number){
    this.supplierService.deleteSupplier(supplier)
    .subscribe(data=>{
        //console.log(data);
        if(data==null){
            this.suppliers.splice(i,1);
            alert("supplier deleted successful");
        }
    },
    err =>{
        this.error = err;
        alert(this.error);
    })
}
```

```
//add
0 references
addSupplier(supplier : Supplier){
    this.supplierService.addSupplier(supplier)
    .subscribe(data=>{
        //console.log(data);
        this.suppliers.push(data);
        this.newSupplier = new Supplier();
    },
    err=>{
        this.error = err;
        console.log(err);
    })
}

//update
0 references
update(i: number){
    this.supplierService.updateSupplier(this.currentSupplier)
    .subscribe(data=>{
        this.suppliers[i] = this.currentSupplier;
        //console.log(data);
    },
    err=>{
        this.error = err;
        console.log(err);
    })
}
```

Figure 108: TS du composant supplier

```
1<app-navbar></app-navbar>
2
3<mdb-card style="margin-top:100px;margin-bottom: 100px;" *ngIf="currentSupplier" class="text-center">
4    <mdb-card-header class="info-color white-text text-center py-4">
5        <h5>
6            <strong>Supplier Details</strong>
7        </h5>
8    </mdb-card-header>
9    <form style="color: #757575;">
10        <div class="md-form mt-3">
11            <input type="text" name="firstname" class="form-control" [(ngModel)]="currentSupplier.firstName" mdbInputDirective required>
12            <label for="name">First name : </label>
13        </div>
14
15        <div class="md-form">
16            <input type="text" name="lastname" class="form-control" [(ngModel)]="currentSupplier.lastName" mdbInputDirective required>
17            <label for="lastname">Last name : </label>
18        </div>
19        <div class="md-form">
20            <input type="email" name="mail" class="form-control" [(ngModel)]="currentSupplier.mail" mdbInputDirective required>
21            <label for="mail">Mail : </label>
22        </div>
23        <div class="md-form">
24            <input type="tel" name="tel" class="form-control" [(ngModel)]="currentSupplier.phoneNumber" mdbInputDirective required>
25            <label for="tel">Phone number : </label>
26        </div>
27        <div class="md-form">
28            <input type="date" name="date" class="form-control" [(ngModel)]="currentSupplier.dateOfBirth" mdbInputDirective required>
29            <label for="date">Date : </label>
30        </div>
31        <div class="md-form">
32            <input type="text" name="address" class="form-control" [(ngModel)]="currentSupplier.address" mdbInputDirective required>
33            <label for="address">Address : </label>
34        </div>
35        <div class="md-form">
36            <input type="text" name="postalcode" class="form-control" [(ngModel)]="currentSupplier.postalCode" mdbInputDirective required>
37            <label for="postalcode">ZIP Code : </label>
38        </div>
39        <div class="md-form">
40            <input type="text" name="town" class="form-control" [(ngModel)]="currentSupplier.town" mdbInputDirective required>
41            <label for="town">Town : </label>
42        </div>
43        <div class="md-form">
44            <input type="text" name="country" class="form-control" [(ngModel)]="currentSupplier.country" mdbInputDirective required>
45            <label for="country">Country : </label>
46        </div>
47    </form>
```

```

1 48  <mdb-card-footer class="justify-content-center">
1 49    <button type="button" mdbBtn gradient="blue" rounded="true" class="waves-light" aria-label="Close" (click)="update()" mdbWavesEffect>Update</button>
1 50    <button type="button" mdbBtn color="primary" class="relative waves-light" mdbWavesEffect (click)="onDelete(currentSupplier)">Delete</button>
1 51 </mdb-card-footer>
1 52 </mdb-card>
1 53
1 54 ch2 style="margin-top:100px;">List of Suppliers</h2>
1 55 <div mdbModal #basicModal="mdbModal" class="modal top" tabindex="-1" role="dialog" aria-labelledby="myBasicModalLabel">
1 56   <div class="modal-dialog modal-lg modal-full-height modal-top" role="document">
1 57     <div class="modal-content">
1 58       <div class="modal-header">
1 59         <button type="button" class="close pull-right" aria-label="Close" (click)=>basicModal.hide()</button>
1 60         <span><img alt="close icon" /></span>
1 61       </div>
1 62       <h4 class="modal-title w-100" id="myModalLabel">Add new Supplier</h4>
1 63     </div>
1 64     <div class="modal-body">
1 65       <form class="text-center" style="color: #757575;">
1 66
1 67         <div class="md-form mt-3">
1 68           <input type="text" name="firstname" class="form-control" [(ngModel)]="newSupplier.firstName" mdbInputDirective required>
1 69           <label for="name">First name : </label>
1 70         </div>
1 71
1 72         <div class="md-form">
1 73           <input type="text" name="lastname" class="form-control" [(ngModel)]="newSupplier.lastName" mdbInputDirective required>
1 74           <label for="lastname">Last name : </label>
1 75         </div>
1 76
1 77         <div class="md-form">
1 78           <input type="email" name="mail" class="form-control" [(ngModel)]="newSupplier.mail" mdbInputDirective required>
1 79           <label for="mail">Mail : </label>
1 80         </div>
1 81
1 82         <div class="md-form">
1 83           <input type="tel" name="tel" class="form-control" [(ngModel)]="newSupplier.phoneNumber" mdbInputDirective required>
1 84           <label for="tel">Phone number : </label>
1 85         </div>
1 86
1 87         <div class="md-form">
1 88           <input type="date" name="date" class="form-control" [(ngModel)]="newSupplier.dateOfBirth" mdbInputDirective required>
1 89           <label for="date">Date : </label>
1 90         </div>
1 91
1 92         <div class="md-form">
1 93           <input type="text" name="address" class="form-control" [(ngModel)]="newSupplier.address" mdbInputDirective required>
1 94           <label for="address">Address : </label>
1 95         </div>
1 96
1 97         <div class="md-form">
1 98           <input type="text" name="address" class="form-control" [(ngModel)]="newSupplier.address" mdbInputDirective required>
1 99           <label for="address">Address : </label>
1 100
1 101         <div class="md-form">
1 102           <input type="text" name="postalcode" class="form-control" [(ngModel)]="newSupplier.postalCode" mdbInputDirective required>
1 103           <label for="postalcode">ZIP Code : </label>
1 104         </div>
1 105
1 106         <div class="md-form">
1 107           <input type="text" name="town" class="form-control" [(ngModel)]="newSupplier.town" mdbInputDirective required>
1 108           <label for="town">Town : </label>
1 109         </div>
1 110
1 111         <div class="md-form">
1 112           <input type="text" name="country" class="form-control" [(ngModel)]="newSupplier.country" mdbInputDirective required>
1 113           <label for="country">Country : </label>
1 114         </div>
1 115
1 116 </div class="table-responsive table-bordered">
1 117   <table mdbTable #table class="table table-hover table-fixed table-striped">
1 118     <!-- table head-->
1 119     <thead>
1 120       <tr>
1 121         <th *ngFor="let head of headElements; let i = index" scope="col" [mdbTableSort]="suppliers" [sortBy]="#headElements[i]">{{head}} <span><img alt="sort icon" /></span></th>
1 122       </tr>
1 123     </thead>
1 124     <!-- table head-->
1 125   </thead>
1 126
1 127   <!-- table body-->
1 128   <tbody>
1 129     <tr class="table" *ngFor="let supplier of suppliers; let i = index" [class.selected]="Supplier === currentSupplier">
1 130       <th scope="row" [mdbTableSort]="supplier.supplierID"></th>
1 131       <td >{{Supplier.firstName}}</td>
1 132       <td >{{Supplier.lastName}}</td>
1 133       <td >{{Supplier.mail}}</td>
1 134       <td >{{Supplier.phoneNumber}}</td>
1 135       <td >{{Supplier.town}}</td>
1 136       <td >a (click)="onEdit(supplier, i)" class="clickable">Edit <span><img alt="edit icon" /></span></a></td>
1 137       <td >a (click)="onDelete(supplier, i)" class="clickable">Delete <span><img alt="trash icon" /></span></a></td>
1 138     </tr>
1 139
1 140   </tbody>
1 141   <!-- table body-->
1 142 </div>

```

Figure 109: Code source de la page HTML suppliers

7.3.4.15. Users

Personal Information

The screenshot shows a user management interface. At the top, there is a form for creating a new user with fields for First Name, Last Name, Mail, Password, Role of User (with a dropdown menu), and Date of Birth (with a date input field). Below the form are two green buttons: 'UPDATE' and 'CREATE'. Underneath these buttons is a table displaying a list of users. The table has columns for First Name, Last Name, Mail, Password, Role, Date of Birth, Edit, and Delete. Two rows are visible: one for 'ADMIN' and one for 'WORKER'.

First Name	Last Name	Mail	Password	Role	Date of Birth	Edit	Delete
ADMIN	ADMIN	ADMIN@ADMIN.FR	ADMIN	ADMIN		Edit	Delete
WORKER	WORKER	WORKER@WORKER.FR	WORKER	WORKER		Edit	Delete

Figure 110: Page HTML liste des utilisateurs

Une liste qui permet d'avoir la liste des utilisateurs des rôles admin et worker

```

3 <div class="limiter" >
4   <div class="container-fluid justify-content-md-between" >
5
6   <app-navbar></app-navbar>
7
8
9   <form name="form-users" class="" #f="ngForm" >
10  <!-- <div class="container table-responsive-sm align-content-center align-content-md-center align-items-center" >-->
11   <div class="container" >
12     <table class="table" >
13       <tr>
14         <td>
15           <table class="table table-responsive-lg" >
16             <tr><td colspan="2" ><h3>Personal Information</h3></td></tr>
17             <tr>
18               <td>
19                 <label>
20                   First Name
21                 </label>
22               </td>
23
24               <td>
25                 <input type="text" class="input100" placeholder="First name" [(ngModel)]="newUser.firstName" id="firstname" name="firstname" >
26               </td>
27             <td>
28               <label>
29                 Last Name
30               </label>
31             </td>
32
33             <td>
34               <input type="text" class="input100" placeholder="Last name" id="lastname" name="lastname" [(ngModel)]="newUser.lastName" >
35             </td>
36       </tr>

```

```
37      <tr>
38        <td>
39          <label>
40            Mail
41          </label>
42        </td>
43
44        <td>
45          <input type="email" class="input100" placeholder="Email" [(ngModel)]="newUser.mail" id="mail" name="mail" required>
46        </td>
47
48        <td>
49          <label>
50            Password
51          </label>
52        </td>
53
54        <td>
55          <input type="text" class="input100" placeholder="Passowrd" id="password" name="password" [(ngModel)]="newUser.password" required>
56        </td>
57
58      <tr>
59        <td>
60          <label>
61            Role of User
62          </label>
63        </td>
64
65        <td>
66          <select ng-model="selectedRole" class="input100" [(ngModel)]="newUser.role" [ngModelOptions]={`${standalone:
67 true}` #role="ngModel" required>
68            <option value="ADMIN" class="input100" >ADMIN</option>
69            <option value="WORKER" class="input100" >WORKER</option>
70          </select>
71        </td>
72      </tr>
73
74      <td>
75        <label>
76          Date of Birth
77        </label>
78      </td>
79
80      <td>
81        <input type="date" class="input100" placeholder="JJ/MM/AAAA" [ngModelOptions]={`${standalone:
82 true}` [(ngModel)]="newUser.dateOfBirth" #dateOfBirth="ngModel" id="dateofbirth" name="dateofbirth" >
83
84      </td>
85
86      <tr>
87        <td></td><td></td>
88        <td>
89          <div class="container-login100-form-btn">
90            <button class="login100-form-btn" (click)="updateUser()" >
91              Update
92            </button>
93          </div>
94        <td>
95          <div class="container-login100-form-btn">
96            <button class="login100-form-btn" (click)="saveUser()" >
97              Create
98            </button>
99          </div>
100         <!--<button class="btn btn-primary btn-block" (click)="updateUser()" >Update</button>
101       </td>
102
103       <td>
104         <button class="btn btn-primary btn-block" (click)="saveUser()" >Save</button>-->
105       </td>
106     </tr>
107   </table>
108
```

```

105      </td>
106    </tr>
107  <tr>
108    <td>
109      <!--<div class="container table-responsive-sm align-content-center align-content-md-center align-items-center">-->
110      <table class="display-1 table dataTable table-responsive-lg dark-grey-text">
111        <tr>
112          <th class="th-lg">First Name</th>
113          <th class="th-lg">Last Name</th>
114          <th class="th-lg">Mail</th>
115          <th class="th-lg">Password</th>
116          <th class="th-lg">Role</th>
117          <th class="th-lg">Date of Birth</th>
118          <th class="th-lg">Edit</th>
119          <th class="th-lg">Delete</th>
120        </tr>
121      </table>
122    </td>
123  </tr>
124  <tr *ngFor="let u of usersPage" class="table-info dark-grey-text">
125    <td>
126      {{u.firstName}}
127    </td>
128    <td>
129      {{u.lastName}}
130    </td>
131    <td>
132      {{u.mail}}
133    </td> <td>
134      {{u.password}}
135    </td>
136    <td>
137      {{u.role}}
138    </td>
139    <td>
140      {{u.dateOfBirth}}
141    </td>
142    <td><a (click)="onEditUser(u)" class="clickable">Edit</a></td>
143    <td><a (click)="DeleteUser(u)" class="clickable">Delete</a></td>

```

Figure 111: Code source de la page HTML Liste des utilisateurs

```

1 import { Component, OnInit } from '@angular/core';
2 import {User} from '../../../../../model/user.model';
3 import {UpdateUserService} from '../../../../../services/updateUser.service';
4 import {AuthService} from '../../../../../services/auth.service';
5 import {Router} from '@angular/router';
6 import {Http} from '@angular/http';
7 import {AccountService} from '../../../../../services/account.service';
8 import {UsersService} from '../../../../../services/users.service';
9
10 @Component({
11   selector: 'app-users',
12   templateUrl: './users.component.html',
13   styleUrls: ['./users.component.css']
14 })
15 export class UsersComponent implements OnInit {
16   currentUser: User;
17   usersPage: any;
18   newUser: User=new User();
19   userDelete: User=new User();
20   userModified: User=new User();
21   errorMessage: string;
22   constructor(public updateUserService:UpdateUserService,public usersService:UsersService,public accountService: AccountService, public router: Router,public http: Http) {
23
24 }
25
26   ngOnInit() {
27     /*this.getUsers();
28   }
29 }

```

```
 0 references
30 saveUser() {
31   console.log(this.newUser);
32   this.accountService.createAccount(this.newUser)
33     .subscribe(data => {
34     this.router.navigate(['/users']);
35     console.log(data);
36   }, err => {
37     console.log(err);
38     this.errorMessage = "username already exist";console.log(this.errorMessage);
39   }
40 )
41 }
0 references
42 onEditUser(user:User){
43   this.newUser=user;
44 }
0 references
45 updateUser(){
46
47   this.userModified.firstName=this.newUser.firstName;
48   this.userModified.lastName=this.newUser.lastName;
49   this.userModified.gender=this.newUser.gender;
50   this.userModified.password=this.newUser.password;
51   this.userModified.mail=this.newUser.mail;
52   this.userModified.address=this.newUser.address;
53   this.userModified.country=this.newUser.country;
54   this.userModified.town=this.newUser.town;
55   this.userModified.postalCode=this.newUser.postalCode;
56   this.userModified.phoneNumber=this.newUser.phoneNumber;
57   this.userModified.dateOfBirth=this.newUser.dateOfBirth;
58   this.userModified.role=this.newUser.role;
59 //this.updateUser.mail=this.currentUser.mail;
60 /*this.updateUser.password=this.currentUser.password;*/
61 this.updateUserService.saveUser(this.userModified)
62   .subscribe(data=>{
63     console.log(data);
64   },
65
66   err=>console.log(err));
67
68 /*this.authService.logIn(this.userModified)
69   .subscribe(data=>{
70     this.router.navigate(['/profile']);
71   },err=>console.log(err));*/
72 }
0 references
73 DeleteUser(user:User){
74
75 //this.userDelete.id=user.id;
76 this.userDelete.mail=user.mail;
77 console.log(this.userDelete);
78 this.updateUserService.deleteUser(this.userDelete)
79   .subscribe(data=>{
80     console.log(data);
81   },
82   err=>console.log(err));
83 }
```

```

1 reference
getUsers(){
    this.userService.getUsers()
        .subscribe(data=>{
            this.usersPage=data;
            console.log(data);
        },
        err=>{
            console.log(err);
        })
}

```

Figure 112: TS du composant Users

7.3.5. Le dossier model

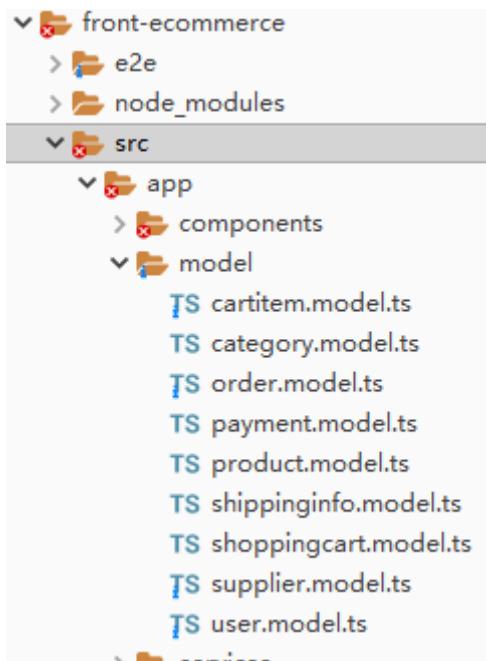


Figure 113: Structure du dossier model

Ce package contient que les entités qu'on a dans le backend.

Pour créer un model : ng generate class -type=model

Voici un exemple :

```

export class Category {
  id : string;
  name : string = "";
  description : string = "";
}

```

Figure 114: Model Category

7.3.6. Le dossier services

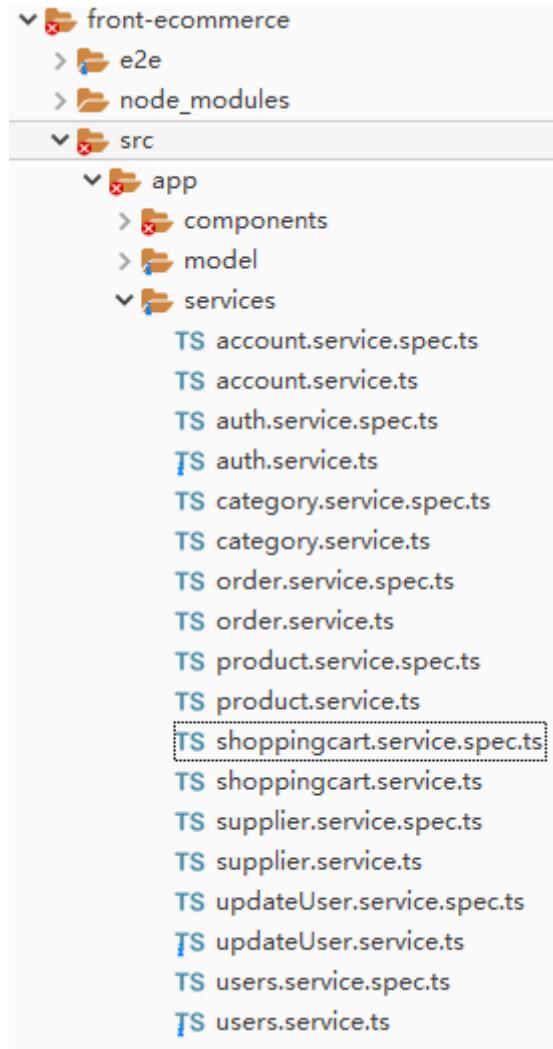


Figure 115: Structure du dossier services

Dans ce package contient donc toutes les services web (REST Controller) qui permet donc d'interagir avec le serveur.

Un exemple de service, toutes les autres ont de la même forme :

```

1 import { Injectable } from '@angular/core';
2 import {Http} from '@angular/http';
3 import {Category} from '../model/category.model';
4 import {AppComponent} from '../app.component';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class CategoryService {
10
11   constructor(public http: Http) { }
12
13   public getCategories(){
14     return this.http.get(AppComponent.API_URL+"/category/categories")
15     .map(response=> response.json());
16   }
17
18   public deleteCategory(category : Category){
19     return this.http.post(AppComponent.API_URL+"/category/delete", category)
20     .map(response=>response.json());
21   }
22
23   public updateCategory(category:Category){
24     return this.http.put(AppComponent.API_URL+"/category/update", category)
25     .map(response=>response.json());
26   }
27
28   public addCategory(category:Category){
29     return this.http.post(AppComponent.API_URL+"/category/create", category)
30     .map(response=>response.json());
31   }
32 }

```

Figure 116: Code source du CategoryService

Chaque service ont un add, delete, update et un get qu'on appelle CRUD (Create, Read, Update and Delete).

8. Problèmes rencontrés

De nombreuses problèmes qu'on a rencontré pendant le projet.

Le premier est lié au Spring sécurité, quelque bug persiste lorsqu'on veut faire des manipulations.

Le deuxième est lié au Hibernate, par exemple on a toujours pas réussi à résoudre le problème quand on veut faire une commande c'est-à-dire la commande ne se valide pas.

Le troisième problème est plutôt lié au design, l'utilisateur du html. On n'est pas tellement à l'aise dans le design.

9. Amélioration

De nombreuse amélioration pourrait être à faire : comme le design de certaine page, faire la page de paiement, ext....

10. Conclusion

Un projet intéressant qui nous a permis d'apprendre pas mal de technologie. Et de mettre en avant nos compétences qu'on a appris.

Le projet est réalisé en quelque mois, tout d'abord nous avons fait la conception de l'application ensuite une étude approfondie des technologies qu'on devait utilisées et pour finir la réalisation de l'application.

Pendant la réalisation, des difficultés qu'on a rencontrées mais certains problèmes ont été résolus et certain non.

Pour finir, nous avons réalisé 90% du projet, mais il reste quand même 10% à finir. C'est 10% correspond qu'aux bugs qui ont pas été corriger pour la partie commander et la partie de paiement.