

Développeur web et web mobile en FAD

Session mars 2021 – janvier 2022

**MÉMOIRE PROFESSIONNEL :**

**Projet de stage**

**Najet EL BOUBKARI**

**SOMMAIRE**

**1. Cahier des charges p 4**

1.1. Introduction p 4

1.2. Contexte p 5

1.3. Besoins p 6

1.4. Fonctionnalités p 7

1.5. Contraintes p 8

1.6. Déroulement p 9

1.7. Authentification p 10

**2. Spécifications fonctionnelles p 11**

2.1. Cas d’utilisation p 11

2.2. Diagrammes d’activités p 12

2.3. Diagrammes de séquences p 17

2.4. Diagrammes de classes p 22

2.5. Maquettage p 24

**3. Conception p 30**

3.1. Modèle conceptuel des données p 30

3.2. Modèles logique des données p 31

3.3. Requêtes SQL p 32

**4. Arborescence p 33**

**5. Outils techniques p 34**

**6. Extrait de code Java p 35**

6.1. Se connecter p 35

6.2. Créer un nouveau client p 39

**7. Recherche à partir d’un site anglophone p 45**

**8. Conclusion p 48**

**Cahier des charges :** introduction

Occ’Fermetures est une entreprise spécialisée en menuiseries, localisée à Bruguières, près de Toulouse.   
Son effectif est de 3 salariés.   
  
SIRET : 49496201200014   
Statut : SARL coopérative ouvrière de production (SCOP)   
Secteur d’activité : travaux de menuiserie bois et pvc

Intervenants du projet :

Frédéric VERDELET,  
responsable communication

David FREDE & Gilles CONORT,  
dirigeants de l’entreprise

Najet EL BOUBKARI,  
stagiaire en développement web/mobile

**Cahier des charges :** contexte

Situation actuelle :

- site web existant   
- application non existante

Afin d’apporter une amélioration dans la productivité de l’entreprise et de faciliter les activités au quotidien des employés, les responsables ont pensé au développement d’une application mobile interne et propre à leurs besoins.

Objectifs :

- réduire le temps consacré au chiffrage d’un projet client   
- accéder à tout moment aux données clients   
- optimiser la gestion d’un projet client

**Cahier des charges :** besoins

|  |  |
| --- | --- |
| PENDANT LE PROJET | APRÈS LE PROJET |
| > plateforme Java   > outils pour :   - la conception   - les spécifications fonctionnelles   - le maquettage | > hébergement   > maintenance |

**Cahier des charges :** fonctionnalités

|  |  |  |
| --- | --- | --- |
| CLIENT |  | PROJET |
| - Rechercher  - Créer  - Modifier  - Supprimer |  | - Rechercher - Créer  - Ajouter un élément  - Modifier un élément  - Supprimer un élément ou le projet intégral |
|  |  |  |
| OPTION |  | COMPTE |
| - Ajouter |  | - Ajouter  - Supprimer |

**Cahier des charges :** contraintes

Dateline : 26/11/2021

Budget : 5000€

Techniques :   
- Support Android   
- Hébergement   
- Maintenance   
- Sécurité des données

Règlementaires : RGPD, CNIL, mentions légales

**Cahier des charges :** déroulement

Octobre :

|  |  |  |  |
| --- | --- | --- | --- |
| Semaine 1 | Semaine 2 | Semaine 3 | Semaine 4 |
| Analyse, conception et  modélisation du projet | |  |  |
|  | Mise en place de la BDD | |  |
|  |  |  | Déploiement |

Novembre :

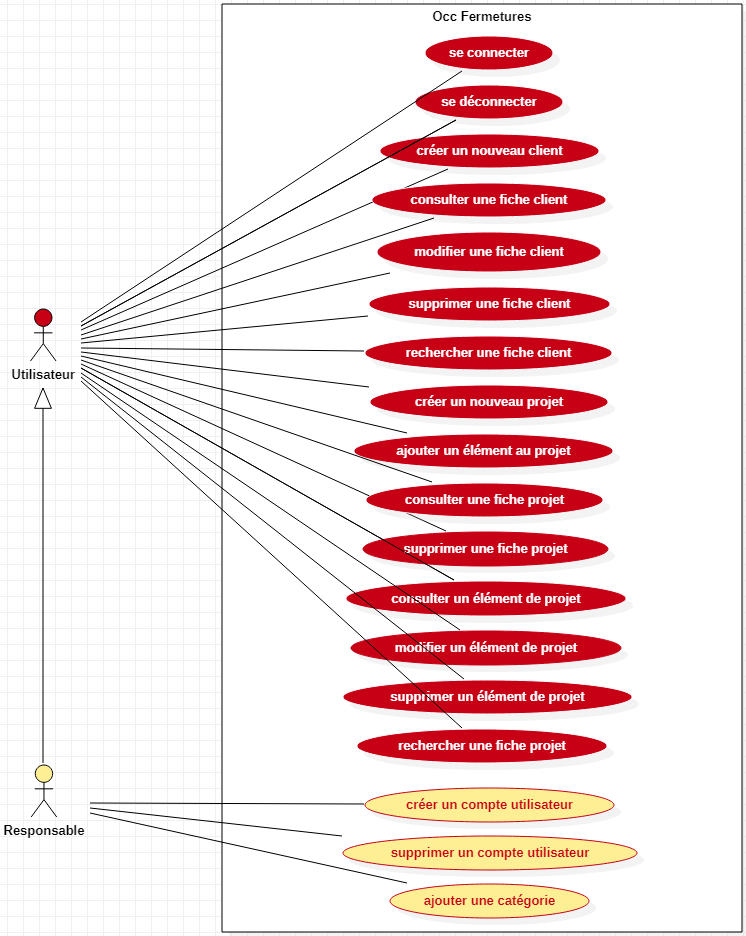
|  |  |  |  |
| --- | --- | --- | --- |
| Semaine 1 | Semaine 2 | Semaine 3 | Semaine 4 |
| Déploiement | | | Test |

**Cahier des charges :** authentification

Signature Occ’Fermetures Signature Pitxucom

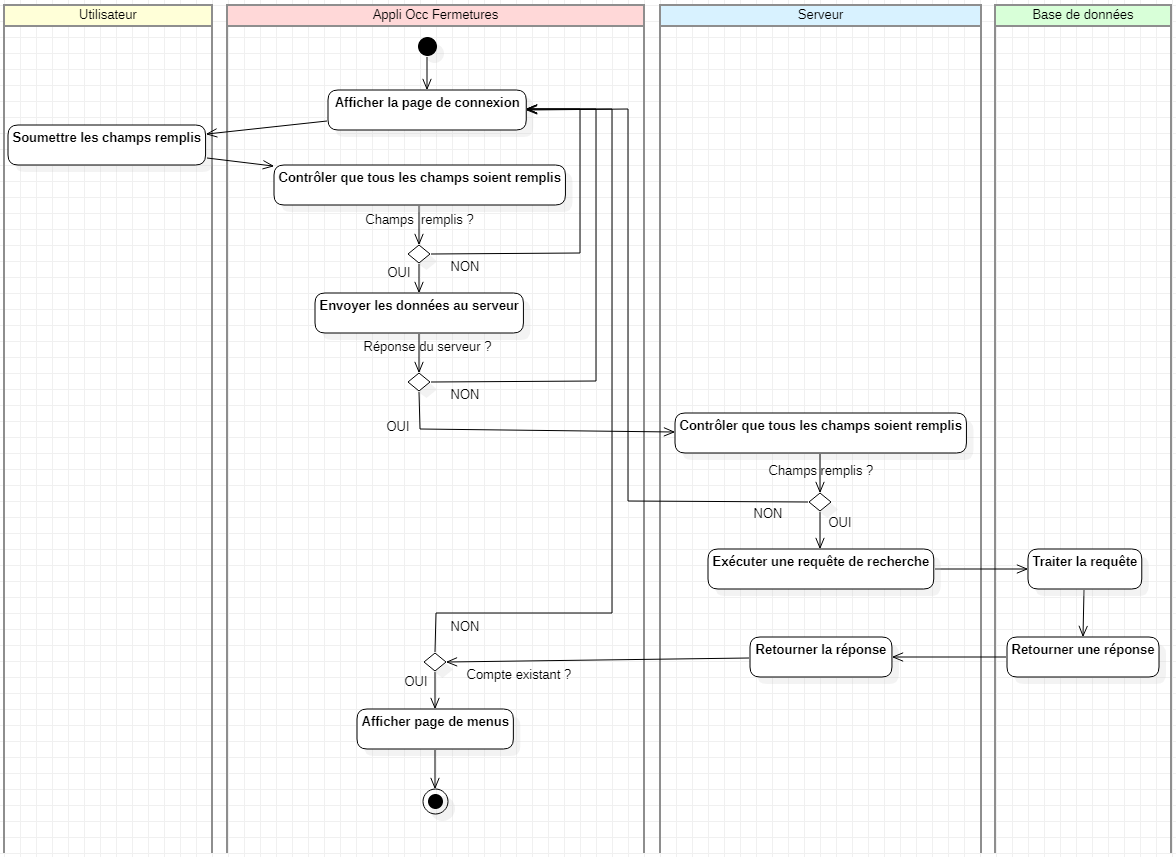
précédée de la mention précédée de la mention  
“Bon pour accord” “Bon pour accord”

**Spécifications fonctionnelles :** cas d’utilisation



**Spécifications fonctionnelles :** diagramme d’activités

en réponse au cas d’utilisation « se connecter » :



**Spécifications fonctionnelles :** diagramme d’activités

Suite à un NON, retour à la page de connexion avec un message qui s'affiche à l'écran selon le cas :

* Champs non remplis :

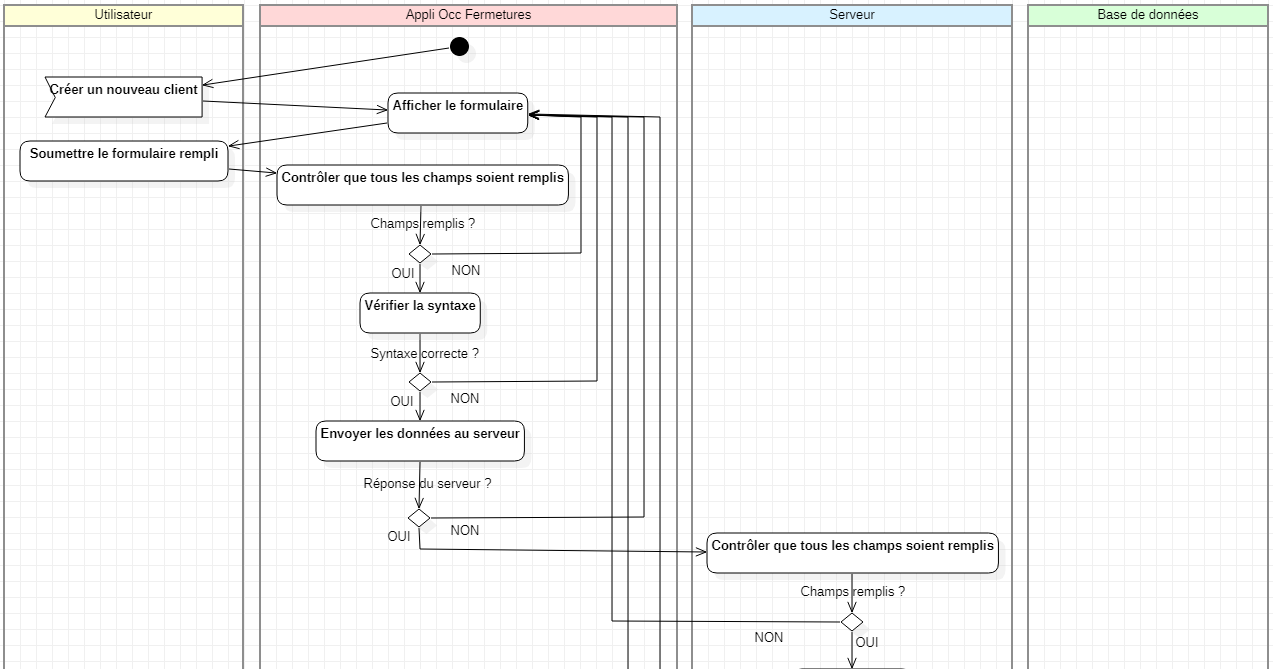
{côté client} « Veuillez remplir tous les champs. »

{côté serveur} « Aucun *(nom du champ)* saisi. »

* Absence de réponse du serveur : « Problème d’accès au serveur. »
* Compte non existant :  « Login incorrect. »

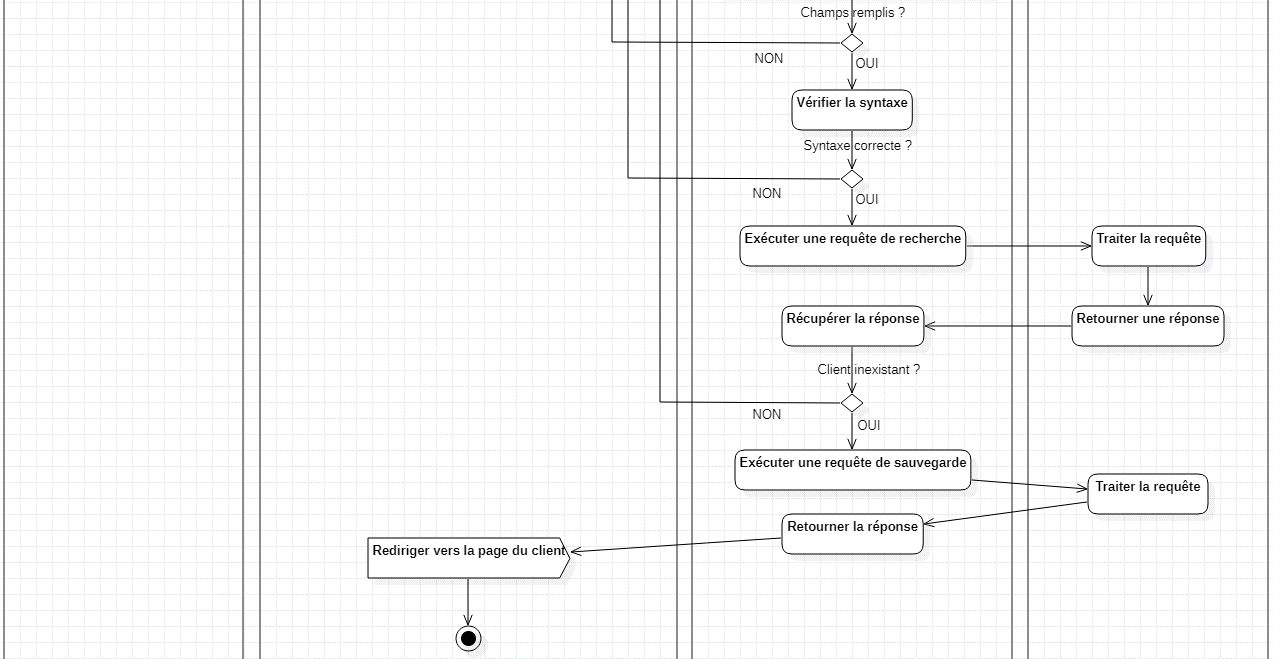
**Spécifications fonctionnelles :** diagramme d’activités

en réponse au cas d’utilisation « créer un nouveau client » :



**Spécifications fonctionnelles :** diagramme d’activités

en réponse au cas d’utilisation « créer un nouveau client » :



**Spécifications fonctionnelles :** diagramme d’activités

Suite à un NON, retour au formulaire avec un message qui s'affiche à l'écran selon le cas :

* Champs non remplis :

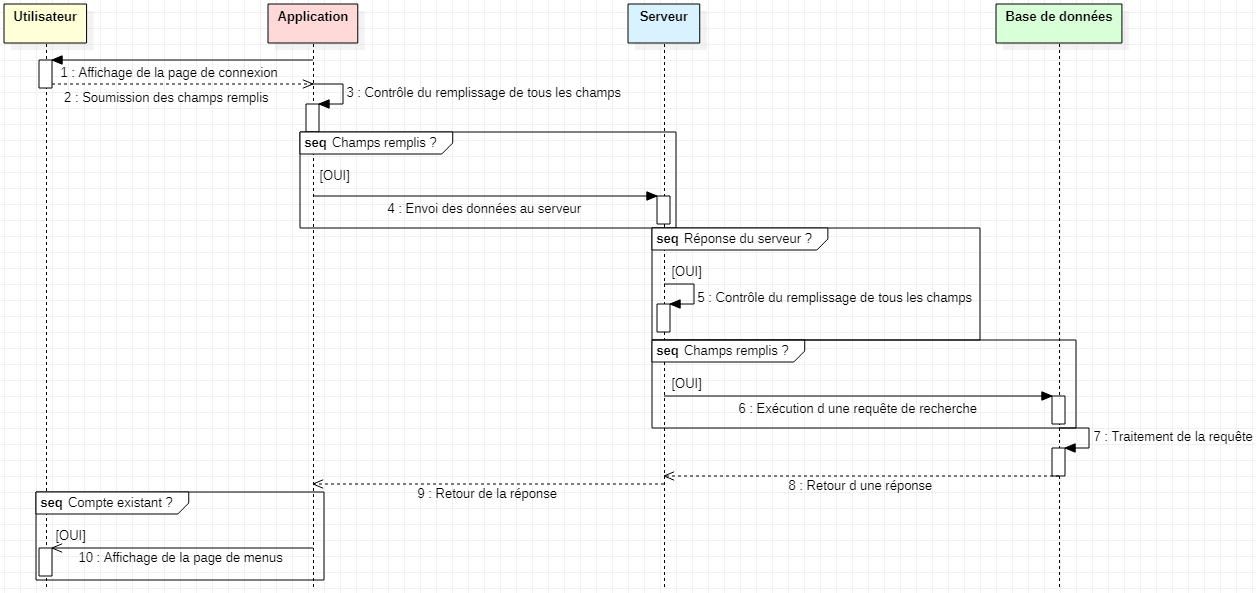
{côté client} « Veuillez remplir tous les champs. »

{côté serveur} « Aucun *(nom du champ)* saisi. »

* Syntaxe incorrecte : « Format du *(nom du champ)* incorrect. »
* Absence de réponse du serveur : « Problème d’accès au serveur. »
* Client existant :  « L’adresse mail *(email du client)* existe déjà. »

**Spécifications fonctionnelles :** diagramme de séquence

en réponse au cas d’utilisation « se connecter » :



**Spécifications fonctionnelles :** diagramme de séquence

Suite à un NON, retour au formulaire avec un message d’erreur qui s'affiche à l'écran selon le cas :

* Après l’étape 3 : Champs remplis ?

Si NON, message : « Veuillez remplir tous les champs. »

* Après l’étape 4 : Réponse serveur ?

Si NON, message : « Problème d’accès au serveur. »

* Après l’étape 5  :  Champs remplis ?

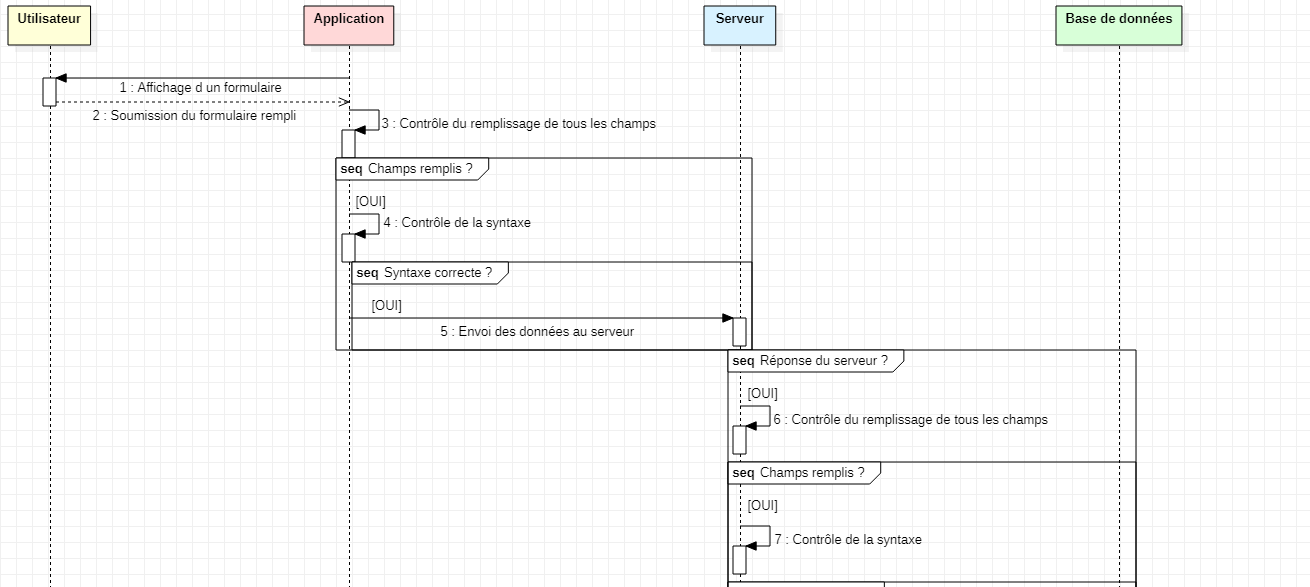
Si NON, message : « Aucun *(nom du champ)* saisi. »

* Après l’étape 9 : Compte existant ?

Si NON, message : « Login incorrect. »

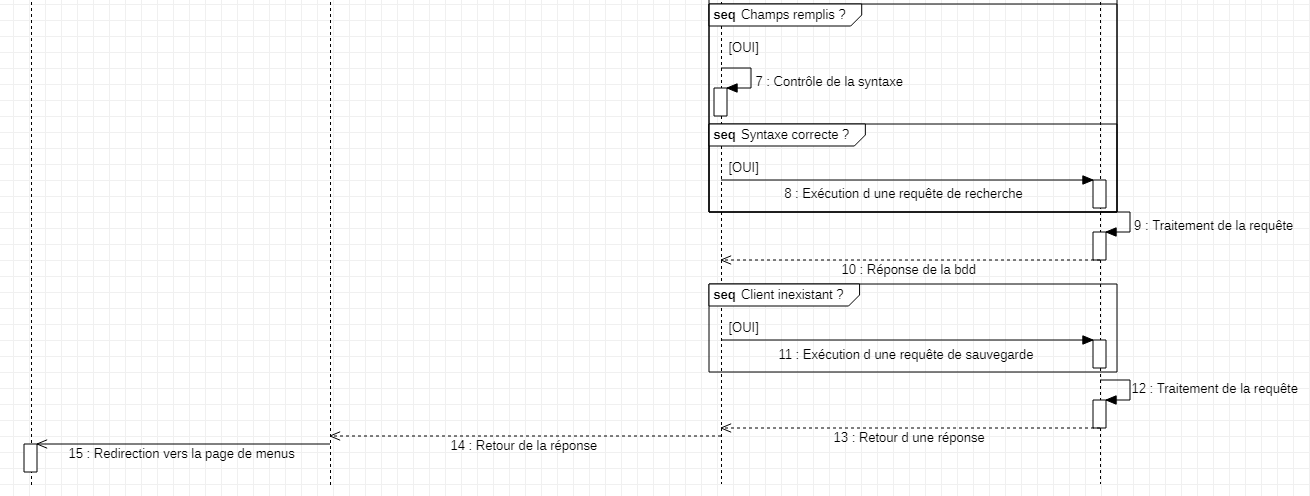
**Spécifications fonctionnelles :** diagramme de séquence

en réponse au cas d’utilisation « créer un nouveau client » :



**Spécifications fonctionnelles :** diagramme de séquence

en réponse au cas d’utilisation « créer un nouveau client » :



**Spécifications fonctionnelles :** diagramme de séquence

Suite à un NON, retour au formulaire avec un message d’erreur qui s'affiche à l'écran selon le cas :

* Après l’étape 3 : Champs remplis ?

Si NON, message : « Veuillez remplir tous les champs. »

* Après l’étape 4 et 7 : Syntaxe correcte ?

Si NON, message : « Format du *(nom du champ)* incorrect. »

* Après l’étape 5 : Réponse serveur ?

Si NON, message : « Problème d’accès au serveur. »

* Après l’étape 6  :  Champs remplis ?

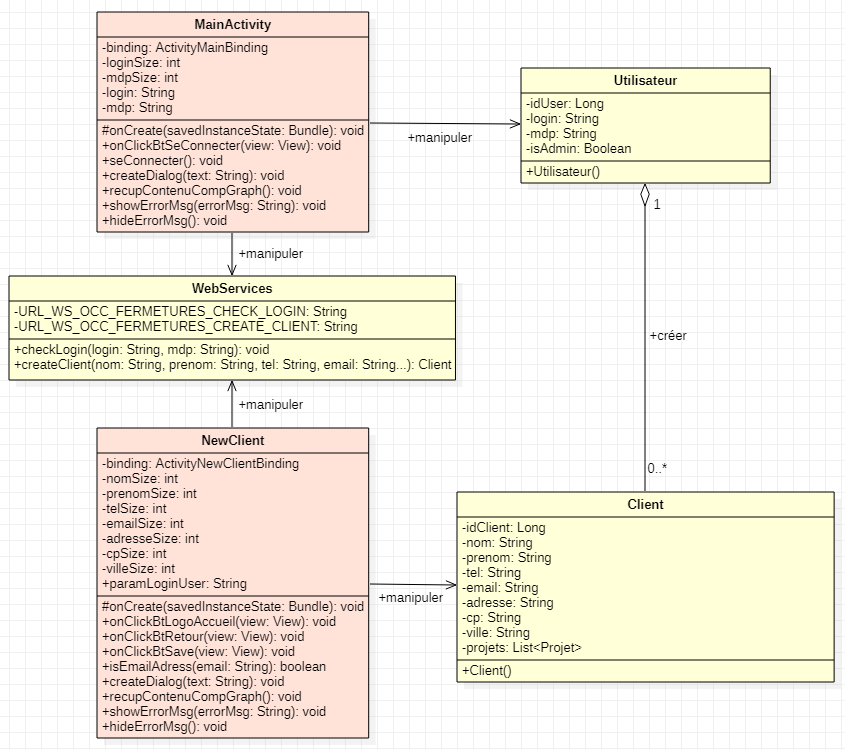
Si NON, message : « Aucun *(nom du champ)* saisi. »

* Après l’étape 10 : Client inexistant ?

Si NON, message : « L’adresse mail *(email du client)* existe déjà. »

**Spécifications fonctionnelles :** diagramme de classes

**Côté client :**

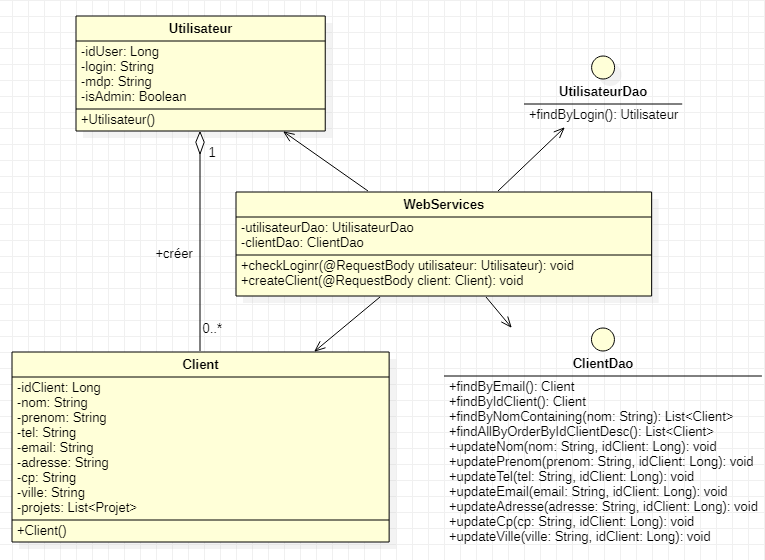


Par soucis de visibilité :

* je n’ai présenté que les classes nécessaires aux fonctionnalités « se connecter » et « créer un nouveau client ».
* je n’ai pas noté les constructeurs avec paramètres et les getters/setters dans les classes Utilisateur et Client.
* j’ai noté que les attributs et méthodes nécessaires aux fonctionnalités étudiées dans la classe WebServices.

**Spécifications fonctionnelles :** diagramme de classes

**Côté serveur :**



Par soucis de visibilité :

* je n’ai présenté que les classes nécessaires aux fonctionnalités « se connecter » et « créer un nouveau client ».
* je n’ai pas noté les constructeurs avec paramètres et les getters/setters dans les classes Utilisateur et Client.
* j’ai noté que les attributs et méthodes nécessaires aux fonctionnalités étudiées dans la classe WebServices.

**Spécifications fonctionnelles :** maquettage

**Zoning** page de connexion :



**Spécifications fonctionnelles :** maquettage

**Wireframe** page de connexion :



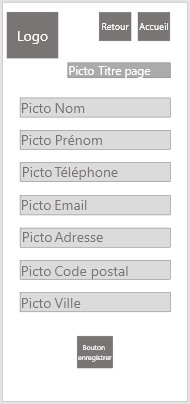
**Spécifications fonctionnelles :** maquettage

**Mockup** page de connexion :



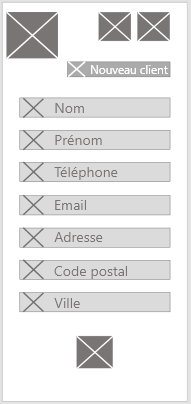
**Spécifications fonctionnelles :** maquettage

**Zoning** page de création de client :



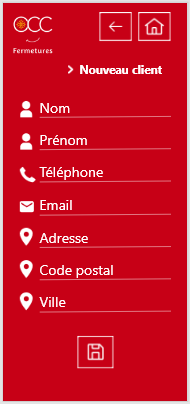
**Spécifications fonctionnelles :** maquettage

**Wireframe** page de création de client :



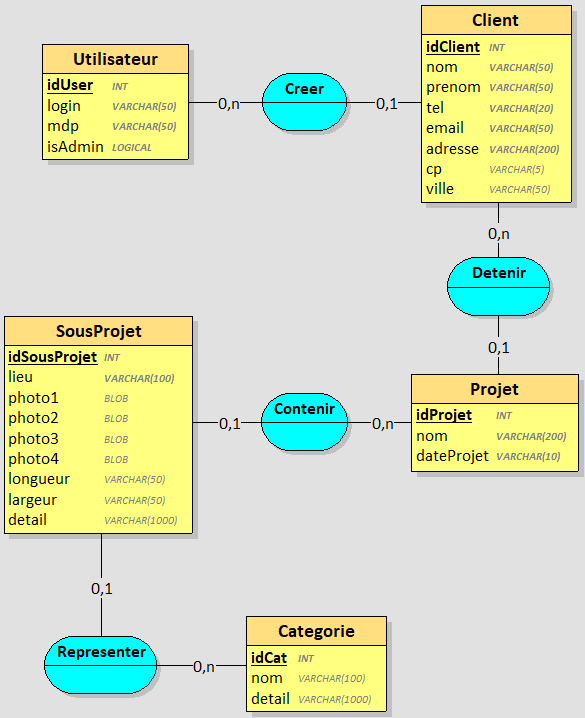
**Spécifications fonctionnelles :** maquettage

**Mockup** page de création de client :



**Conception :** modèle conceptuel des données

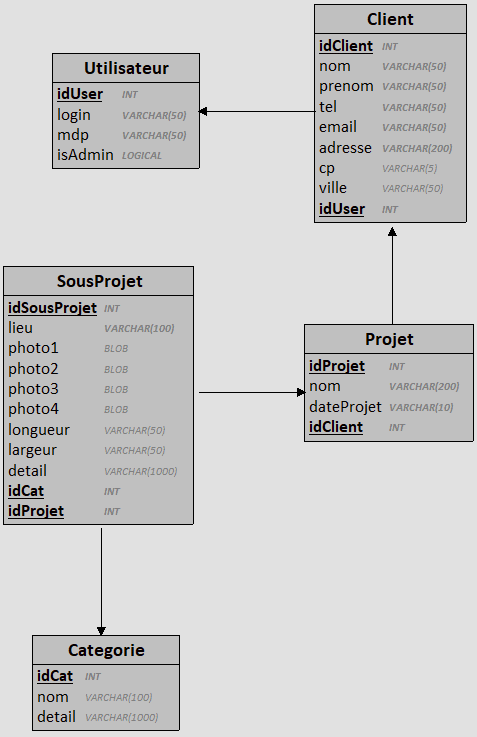
Voici la représentation des données de mon programme sous forme d’entités. Les entités sont liées par une association de type 1 avec de chaque côté une cardinalité (0,n) et (0,1).



**Conception :** modèle logique des données

Les entités de mon MCD deviennent des tables dans le MLD et intègrent une clé étrangère.

Dans le cas d’une association de type n (cardinalité (0,n) et (0,n)), on aurait une table d’association qui intègrerait deux clés étrangères qui formeraient sa clé primaire.

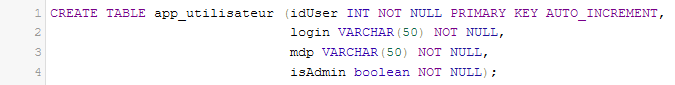


**Conception :** requêtes SQL

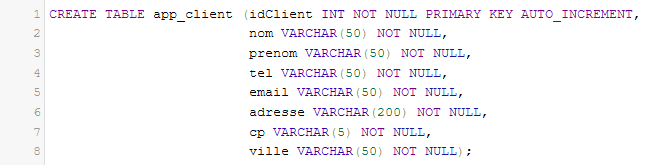
Création de ma base de données :



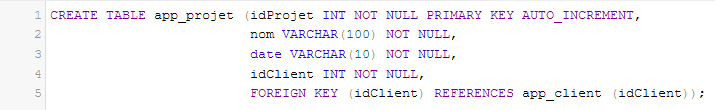
Création de la table utilisateur :



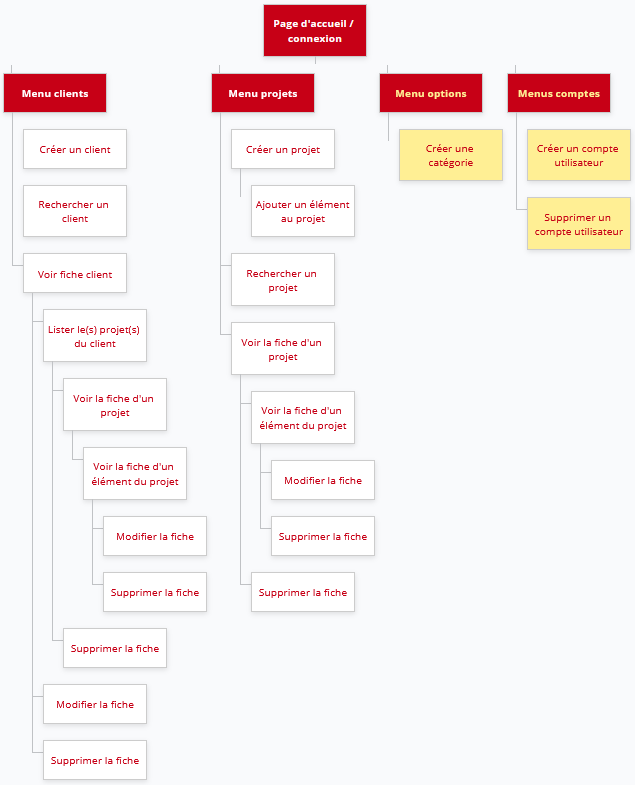
Création de la table client :



Création de la table projet :



**Arborescence :**

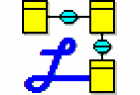
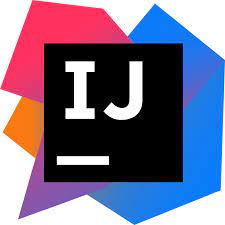


**Outils techniques :**

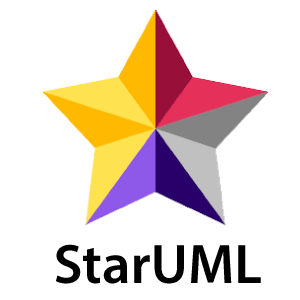
**Côté client :**



**Côté serveur :**

**Côté client et serveur :**



**Extrait de code :** pourse connecter (côté client)

Cette méthode est appelée lors d’un clic sur le bouton SE CONNECTER.

Elle vérifie que les champs soient bien remplis avant d’exécuter la requête qui contrôle les données en accédant au webservice.

public void seConnecter() {

*//on commence par cacher les éventuels précédents msg d’erreur* hideErrorMsg();  
*//méthode qui stocke la longueur des données saisies par l’utilisateur et leur contenu* recupContenuCompGraph();  
*//si tous les champs sont remplis* if (loginSize > 0 && mdpSize > 0) {  
 new Thread() {  
 @Override  
 public void run() {  
 try {

*//on exécute la requête qui contrôle les données saisies* WSUtils.*checkLogin*(login, mdp);  
 Intent intent = new Intent(MainActivity.this, MenuActivity.class);  
*//on passe un paramètre à envoyer dans MenuActivity* intent.putExtra("sendLoginUser", login);  
 startActivity(intent);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showErrorMsg(e.getMessage());  
 }  
 }  
 }.start();  
 } else {  
 createDialog("Veuillez saisir votre login et votre mot de passe.");  
 }  
}

**Extrait de code :** pourse connecter (côté client)

J’ai déporté les méthodes suivantes de la méthode seConnecter pour l’alléger.

*//ces méthodes utilisent un runOnUiThread car font appel à des composants graphiques*

*//rend le champ d’erreur invisible*public void hideErrorMsg() {  
 runOnUiThread(new Runnable() {  
 @Override  
 public void run() {  
 binding.tvError.setVisibility(View.*INVISIBLE*);  
 }  
 });  
}

*//stocke la longueur des données saisies et leur contenu*

public void recupContenuCompGraph() {  
 runOnUiThread(new Runnable() {  
 @Override  
 public void run() {loginSize = binding.etLogin.getText().length();  
 mdpSize = binding.etMdp.getText().length();  
 login = binding.etLogin.getText().toString();  
 mdp = binding.etMdp.getText().toString();  
 }  
 });  
}

*//rend le champ d’erreur visible et affiche le texte saisi en paramètre*public void showErrorMsg(String errorMsg) {  
 runOnUiThread(new Runnable() {  
 @Override  
 public void run() {  
 binding.tvError.setVisibility(View.*VISIBLE*);  
 binding.tvError.setText(errorMsg);  
 }  
 });  
}

**Extrait de code :** pour se connecter (côté client)

Cette méthode permet d’accéder au webservice. J’y crée un nouvel objet Utilisateur avec le login et le mot de passe saisis.

Je transforme cet utilisateur en JSON pour l’utiliser lors de la requête qui parse en POST.

public static void checkLogin(String login, String mdp) throws Exception {  
*//on crée un nouvel utilisateur avec les paramètres reçus* Utilisateur utilisateur = new Utilisateur(login, mdp);

*//on transforme notre utilisateur en json* String json = *myGson*.toJson(utilisateur);

*//on effectue la requête de parsing* OkHttpUtils.*sendPostOkHttpRequest*(*URL\_WS\_OCC\_FERMETURES\_CHECK\_LOGIN*, json);  
}

**Extrait de code :** pour se connecter (côté serveur)

Cette méthode permet d’effectuer la requête en POST. J’y fais une recherche d’utilisateur par le login pour vérifier si l’utilisateur existe dans la base de données.

@PostMapping("/checkLogin")  
public void checkLogin(@RequestBody Utilisateur utilisateur) throws Exception {  
 System.*out*.println("/checkLogin");  
*//on stocke l’utilisateur que nous retourne la requête de recherche par le login reçu en paramètre* Utilisateur userBdd = utilisateurDao.findByLogin(utilisateur.getLogin());

*//si userBdd est null, le login saisi est incorrect* if(userBdd==null){  
 throw new Exception("Login incorrect");  
 }

*// on récupère le mot de passe de la bdd* String mdpBdd = userBdd.getMdp();

*// on récupère le mot de passe saisi par l’utilisateur*

String mdpUser = utilisateur.getMdp();

*// on compare le mot de passe saisi à celui haché et stocké en bdd* if (!BCrypt.*checkpw*(mdpUser, mdpBdd)){  
 throw new Exception("Mot de passe incorrect");  
 }  
}

Dans mon projet, je passe par un ORM. Grâce à JPA, j’utilise la méthode findByLogin(utilisateur.getLogin()) qui génère une requête SQL :

**SELECT \* FROM app\_utilisateur WHERE login = utilisateur.getLogin() ;**

**Extrait de code :** pour créer un client (côté client)

Cette méthode est appelée lors d’un clic sur le bouton sauvegarder.

Elle vérifie que les champs soient bien remplis avant d’exécuter la requête qui crée le client en accédant au webservice.

public void onClickBtSave(View view) {  
*//on commence par cacher les éventuels précédents msg d’erreur* hideErrorMsg();  
*//méthode qui stocke la longueur des données saisies par l’utilisateur et leur contenu* recupContenuCompGraph();  
*//si tous les champs sont remplis* if (nomSize > 0 && prenomSize > 0 && telSize > 0 && emailSize > 0 && adresseSize > 0 && cpSize > 0 && villeSize > 0) {

*//si le format du numéro de tel est correct* if (isTelephoneValid(tel)) {

*//si le format de l’email est correct* if (isEmailAdress(email)) {

*//si le format du code postal est correct* if (isCPValid(cp)) {  
 new Thread() {  
 @Override  
 public void run() {  
 try {

*////on exécute la requête qui crée le client en le stockant pour réutiliser ses données* Client createdClient = WSUtils.*createClient*(nom, prenom, tel, email, adresse, cp, ville);  
*//redirection vers FicheClientActivity* Intent intent = new Intent(NewClientActivity.this, FicheClientActivity.class);

**Extrait de code :** pour créer un client (côté client)

*//passage de paramètres à envoyer dans FicheClientActivity* intent.putExtra( "sendIdClient", createdClient.getIdClient());

intent.putExtra( "sendLoginUser", paramLoginUser);  
 startActivity(intent);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showErrorMsg( e.getMessage());  
 }  
 }  
 }.start();

} else {  
 createDialog("Format du code postal incorrect.");  
 }  
 } else {  
 createDialog("Format de l'email incorrect.");  
 }  
 } else {  
 createDialog("Format du numéro de téléphone incorrect.");  
 }  
 } else {  
 createDialog("Veuillez remplir tous les champs.");  
 }  
}

**Extrait de code :** pourcréer un client (côté client)

Cette méthode permet d’accéder au webservice. J’y crée un nouvel objet Client avec les données saisies (nom, prénom…).

Je transforme ce client en JSON pour l’utiliser lors de la requête qui parse en POST. Je récupère un JSON de type Client pour le retourner.

public static Client createClient(String nom, String prenom, String tel, String email, String adresse, String cp, String ville) throws Exception {  
*//on crée un nouveau client avec les paramètres qui seront saisis lors de l’appel de la méthode* Client client = new Client(nom, prenom, tel, email, adresse, cp, ville);

*//on transforme notre client en json* String json = *myGson*.toJson(client);

*//on effectue la requête de parsing* String myJson = OkHttpUtils.*sendPostOkHttpRequest*(*URL\_WS\_OCC\_FERMETURES\_CREATE\_CLIENT*, json);

*//on récupère un json de type Client* Client clientInJson = *myGson*.fromJson(myJson, Client.class);

*//on retourne ce json* return clientInJson;  
}

**Extrait de code :** pourcréer un client (côté serveur)

Cette méthode permet d’effectuer la requête en POST. J’y fais tous les contrôles du remplissage des champs et de la bonne syntaxe. J’y fais ensuite une recherche de client par l’email pour vérifier si le client existe dans la base de données. Si il n’existe pas je l’enregistre dans la base de données.

@PostMapping("/createClient")  
public Client createClient(@RequestBody Client client) throws Exception {  
 System.*out*.println("/createClient");

*//contrôle des données saisies avant de les enregistrer* if(client.getNom()==null || client.getNom().trim().length()==0){  
 throw new Exception("Aucun nom saisi");  
 }  
 if(client.getPrenom()==null || client.getPrenom().trim().length()==0){  
 throw new Exception("Aucun prénom saisi");  
 }  
 if(client.getTel()==null || client.getTel().trim().length()==0){  
 throw new Exception("Aucun numéro de téléphone saisi");  
 }  
 if(client.getEmail()==null || client.getEmail().trim().length()==0){  
 throw new Exception("Aucun email saisi");  
 }  
 if(client.getAdresse()==null || client.getAdresse().trim().length()==0){  
 throw new Exception("Aucune adresse saisie");  
 }  
 if(client.getCp()==null || client.getCp().trim().length()==0){  
 throw new Exception("Aucun code postal saisi");

**Extrait de code :** créer un nouveau client (côté serveur)

}  
 if(client.getVille()==null ||

client.getVille().trim().length()==0){  
 throw new Exception("Aucune ville saisie");  
 }  
 if(!isTelephoneValid(client.getTel())){

throw new Exception("Format du numéro de téléphone incorrect");  
 }if(!isEmailAdress(client.getEmail())){  
 throw new Exception("Format de l'email incorrect");  
 }  
 if(!isCPValid(client.getCp())){  
 throw new Exception("Format du code postal incorrect");  
 }

*//pas de contrôle sur la longueur du nom, prénom, adresse et ville : Mr O existe (secrétaire d'Etat chargé du numérique), la ville Y existe (commune française située dans le département de la Somme, en région Hauts-de-France) !!  
  
//on stocke le client que nous retourne la requête de recherche par l’email reçu en paramètre* Client clientBdd = clientDao.findByEmail(client.getEmail());

*//on contrôle si l’email client est reconnu par la bdd* if(clientBdd.getEmail().equals(client.getEmail())) {  
 throw new Exception("L'adresse " + clientBdd.getEmail() + " existe déjà");  
 }  
  
*//si tout est ok, on sauvegarde les données saisies* clientDao.save(client);

*//on retourne le client sauvegardé*  
 return client;  
}

**Extrait de code :** créer un nouveau client (côté serveur)

Dans mon projet, je passe par un ORM. Grâce à JPA :

* j’utilise la méthode findByEmail(client.getEmail()) qui génère une requête SQL :

**SELECT \* FROM app\_client WHERE email = email.getEmail() ;**

* j’utilise la méthode save(client) qui génère une requête SQL :

**INSERT INTO app\_client (nom, prenom, tel, email, adresse, cp, ville) VALUES (client.getNom(), client.getPrenom(), client.getTel(), client.getEmail(), client.getAdresse(), client.getCp(), client.getVille()) ;**

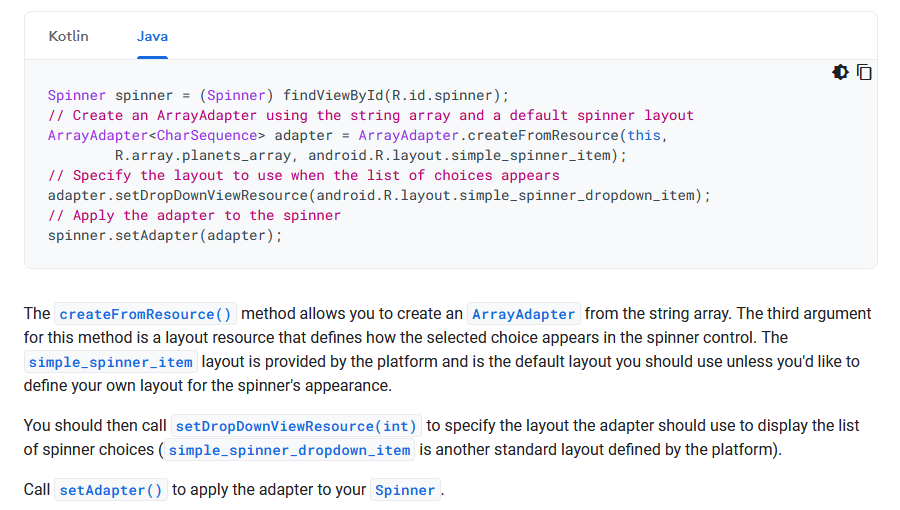
**Recherche à partir d’un site anglophone :**

In my application, to create a project, the user has to select a category (window, door, shutter...). For this, I had to set up a drop-down list, "spinner", which is displayed when clicking on an arrow.

Having never used this object, I followed a documentation on a site for Android application developers.

My source : <https://developer.android.com/guide/topics/ui/controls/spinner>

Extract from the site :



**Recherche à partir d’un site anglophone :**

J’ai donc interprété le code et traduit les explications comme ceci :

Étapes, du code en Java, pour mettre en place un spinner :

* récupérer le composant graphique par la méthode findViewById(R.id.spinner).
* instancier un ArrayAdapter de type CharSequence à qui on applique la méthode createFromResource() prenant en paramètres le contexte, le chemin pour accéder au tableau et la mise en page pour l’apparence du spinner, ici celle par défaut sera utilisée.
* spécifier la mise en page à utiliser lorsque la liste de choix apparaît.
* appliquer l’adapteur à notre spinner.

Il est précisé que :

* la méthode createFromResource() nous permet de créer un ArrayAdapter à partir du tableau de String (chaînes de caractères).
* le troisième argument pour cette méthode est une mise en page de ressource qui définit comment l’élément sélectionné apparait dans le spinner.
* la mise en page simple\_spinner\_item est fournie par la plateforme et est celle par défaut que nous devons utilisée sauf si l’on souhaite définir notre propre mise en page pour l’apparence du spinner.
* nous devrions ensuite appeler la méthode setDropDownViewResource(int) pour spécifier la mise en page que l’adapter devrait utiliser pour afficher la liste de choix du

**Recherche à partir d’un site anglophone :**

spinner (simple\_spinner\_dropdown\_item est un autre standard de mise en page définie par la plateforme).

* nous devons appeler la méthode setAdapter() pour appliquer l’adapter à notre spinner.

**Conclusion :**

Ce projet d’application mobile pour l’entreprise Occ’Fermetures m’a permis de mettre en application les compétences acquises durant ma formation.

J’ai ainsi pu travailler en autonomie, avec quelques galères bien sûr ! Pour y remédier, j’ai beaucoup suivi de tutoriels, de documentations officielles et surtout appris une bonne méthode de travail par étapes.

Cette expérience a été très positive et enrichissante ; elle m’a conforté dans mon projet professionnel en m’apportant satisfaction face à ma débrouillardise et au résultat final.

Résultat final qui tend à être améliorer sur le point de la sécurité, de l’optimisation du code et des fonctionnalités à proposer à l’utilisateur.

A l’issue de mon stage, j’ai mené à terme les menus Client et Projet. En ce qui concerne les menus Option et Compte elles restent à être poursuivies pour y apporter plus de fonctionnalités.

Je reviendrai, par la suite, vers mon maitre de stage pour connaître la suite qui a été donnée à mon projet.