



Rapport de stage  
Concepteur développeur d'applications  
AFPA Amiens

*Thème*

**Création d'une Api REST et développement d'une application  
Desktop et mobile pour le suivi des stagiaires.**

Réalisé par :

**Mr : BENCHIKH Hamid**

**Tuteur du stage : Mr. SIPIERE Germain**

Session 2019/2020

## *Remerciements*

*Je tiens à remercier dans un premier temps mes parents, mes frères ma sœur se trouvant à l'autre rive de la méditerranée pour leurs aides et leurs encouragements durant ma formation.*

*A ma femme est mes deux petits anges qui ont su être sage, patient et encourageant.*

*Je tiens à exprimer ma profonde reconnaissance et mes remerciements à :*

*Mr Germain Sepiere formateur et maître de mon stage pour le suivi, ces conseils, sa présence et tout son aide précieux durant la formation et durant le stage.*

*A Isabele, Cedric et à François Regis pour leurs conseils et leurs présences durant toute la formation.*

*Bien sûr, sans oublié mes collègues du groupe CDA 19123 Tristan, Jason avec qui j'ai beaucoup appris et pris du plaisir à travailler ensemble.*

## **Listes des abréviations**

Afpa : Agence nationale pour la formation professionnelle des adultes.

SGDBR : système de gestion de base de données relationnel.

Api : Application Programming Interface.

JDBC : Java Database Connectivity.

DOM : Document Object Model .

REST : REpresentational State Transfer

URL : Uniform Resource Locator

URI : Uniform Resource Identifier précédemment connue sous le terme UDI

PDO : PHP Data Objects.

SGBD : système de gestion de base de données

Dal : data access logique

Gui : Graphical User Interfaces

JWT : JSON Web Token.

IHM : Interfaces Homme Machine

Css : Cascading Style Sheet

XML : Extensible Markup Language

Fxml : un format de données textuelles, dérivé du format XML,

HTML : Hypertext Markup Language

PHP : Hypertext Preprocessor

SQL : Structured Query Language

# Sommaire

Remerciements	
Liste des abréviations	
Introduction .....	1
➤ Chapitre I : Présentation de l'entreprise d'accueil	
I.1/ AFPA .....	2
I.2 / Quelques chiffres .....	2
➤ Chapitre II : Déroulement et missions du stage	
II. Outils et langages.....	3
II. Création de L'api avec PHP .....	3
II.1. mise en place et environnement de développement .....	4
II.1.1. la base de donnée .....	4
II.1.2. l'API ncode .....	6
II.1.2.1.Config .....	6
II.1.2.2.Object.....	7
II.1.2.3.Offre .....	11
II.1.2.3.Offre .....	9
II.1.2.4.Recherche d'utilisateurs .....	15
II.1.3. Sécurisation de l'api.....	16
II.1.3.1 Utilisation de HTTPS .....	16
II.1.3.2 Authentification .....	16
II.2. L'application DESKTOP .....	17
II.2.1 Cr�ation du projet JavaFX .....	17
II.2.2 L'interface graphique .....	20
II.2.2.3 Evolution du projet .....	22
II.2.3 Retrofite et persistance des donn�es .....	24
II.2.4 Configuration et mise en place .....	24
II.2.5. R�cup�rer et afficher le r�sultat dans les applications.....	27

II.2.5.1 Gérer les codes d'erreur .....	29
II.2.5.2 Liste des offres.....	29
II.2.5.3 Le détail de l'offre.....	30
II.5. Recherche par nom ou prénom .....	32
II.3. L'application Android.....	33
II.3.1 Mise en place .....	33
II.3.2 Création de la fenêtre d'accueil le splash screen .....	35
II.3.3 Animer le splash screen .....	37
II.3.4 Authentification .....	39
II.3.5 Le planning .....	41
II.3.6 Recherche par nom ou prénom .....	42
II.3.7 Le design .....	44
II.4. Axes d'améliorations .....	45
Conclusion .....	45

## **Introduction**

J'ai effectué mon stage à l'AFPA Amiens, ce projet est né d'un besoin exprimé par la direction, avoir un accès facile depuis leurs postes aux différentes informations liées aux stagiaires et aux offres proposées par l'AFPA.

Dans ce présent rapport, je décrirai dans un premier temps la mission qui m'a été confiée et je présenterai l'AFPA, et enfin je détaillerai le déroulement de la mission et les moyens mis en place.

Permettez-moi à présent de me présenter et de vous souhaiter une bonne lecture de ce rapport.

Je m'appelle Hamid BENCHIKH, j'ai 35 ans papa de Lina et Aylan.

Bac +5 dans l'agroalimentaire j'ai suivi un master qualité et sécurité des aliments, une formation que j'ai effectué à l'université Paris Sud Créteil, l'AgroParisTech et l'école vétérinaire d'Alfort, j'ai travaillé comme responsable qualité, auditeur mais aussi dans des laboratoires d'analyse de produits alimentaires.

Ma dernière expérience avant de commencer ma formation était manager boucherie, produits frais chez Carrefour.

L'envie de faire du l'informatique ne m'a jamais quitté depuis jeune, le développement en particulier. Avant de faire Concepteur Développeur d'Applications, j'ai suivi une formation de deux mois sur les techniques de base du développement d'applications.

## Chapitre I : Présentation de l'organisme d'accueil

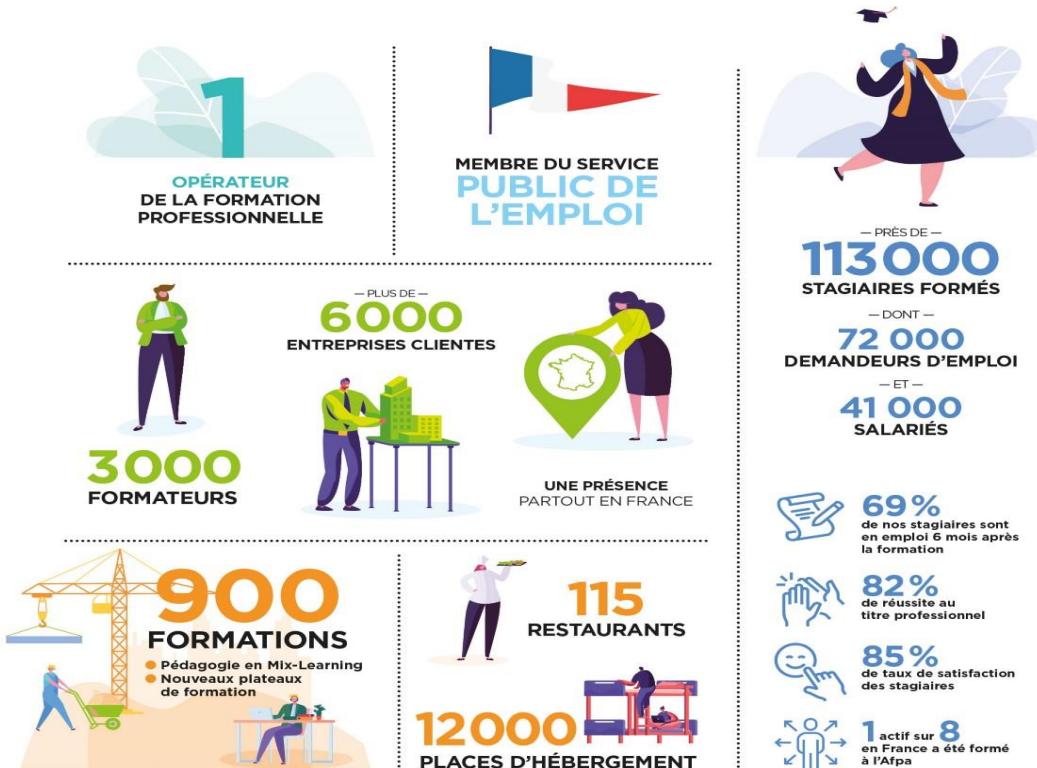
### L'AFPA :

L'Agence nationale pour la formation professionnelle des adultes (Afpa) est un organisme français de formation professionnelle, au service des Régions, de l'État, des branches professionnelles et des entreprises. Membre du Service public de l'emploi, l'Afpa, constituée en association avant de devenir en 2017 un Établissement public à caractère industriel et commercial (EPIC), Cette transformation a pour conséquence de modifier « association » par « agence », elle propose des formations professionnelles qualifiantes, sanctionnées par un titre professionnel du ministère du Travail.

L'Afpa a été créée le 11 janvier 1949 sous l'appellation Association nationale interprofessionnelle pour la formation rationnelle de la main-d'œuvre (ANIFRMO). Son rôle consistait alors à former rapidement les adultes pour les amener à un premier niveau de qualification dans le bâtiment et la métallurgie et répondre aux besoins de la France en pleine reconstruction,. En 1966 l'organisme change de nom pour devenir l'Afpa.

### Quelques chiffres : chiffre 2019

#### L'AFPA EN CHIFFRES



2200558 - Afpa direction de la communication © Ds-vocoda & Invincible bulldog - Stock-Afpa service communication.

80 pole régionaux, 5000 stagiaires en situation d'handicap, 44 000 femmes stagiaires objectif 100000 cette année 2020. 70% des stagiaires trouvent un emploi dans les mois suivant la formation...

## **Chapitre II : Déroulement et missions du stage**

Afin de mener à bien ma mission, j'ai dû utilisé ces langages et Framework.

Conception : UML. Partie fil rouge

Langages : HTML, XML, CSS, JAVASCRIPT, PHP, JAVA (JavaFX et Android), SQL

Framework, Librairies : VUEJS, JQUERY, BOOTSTRAP, MATERIALIZE, CODEIGNITER.

BASES DE DONNÉES : MYSQL.

Outils de développement : ANDROID STUDIO, IntelliJ, NETBEANS, Visual Studio Code, PHPMYADMIN, WAMP, LARAGON, SceneBuilder, Postman, Insomnia.

Systèmes : Windows, Android.

Intégration : Git, WinSCP.

Serveur distance : dev.amorce.

Test API : Postman, Insomnia.

Moyen de communication : emails, réunions, Microsoft Teams, Discord.

### **II. Création de L'API avec PHP**

J'ai créé une API en PHP pour récupérer les données et les informations dont on a besoin dans nos applications desktop et mobile.

Tout d'abord API signifie “Application Programming Interface”. En d'autres termes c'est une interface qui permet le transfert de données d'une application à une (des) autre(s).

En informatique, une interface de programmation d'application ou interface de programmation applicative souvent désignée par le terme API est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels

- Les API permettent de communiquer des données.
- Elles permettent la communication entre différents composants de votre application et entre votre application et d'autres développeurs, par l'utilisation de requêtes et de réponses.
- Elles donnent un moyen d'accès aux données de façon réutilisable et standardisée.

Une API REST. REST (pour REpresentational State Transfer) est un style d'architecture basé sur le protocole HTTP et qui permet de manipuler des ressources via un URI. Pour manipuler ces ressources, une API REST utilise les méthodes HTTP suivantes :

- **GET** : Récupération d'une ressource ;
- **POST** : Ajout d'une ressource ;
- **PUT** : Mise à jour complète d'une ressource ;
- **PUT** : Mise à jour partielle d'une ressource ;
- **DELETE** : Suppression d'une ressource ;
- **HEAD** : Similaire à GET, mais permet uniquement de récupérer les en-têtes HTTP.

## II.1) Mise en place et environnement de développement

### II.1.1. La base de donnée :

La base de données NCODE m'a été fournie par Germain SIPIERE, mon maître de stage.

J'ai bien pris en main la base de données fournie en ciblant toutes les clés primaires, secondaires et les liens entre les tables.

Le besoin est plus focalisé sur la table *Offre* qui prend l'ensemble des formations proposée par l'AFPA avec les groupes de formation. La table *Utilisateur* quant à elle prend toutes les informations et enregistrements de stagiaires et des administrateurs (formateurs). Les deux tables sont reliées avec la table *Suit*.

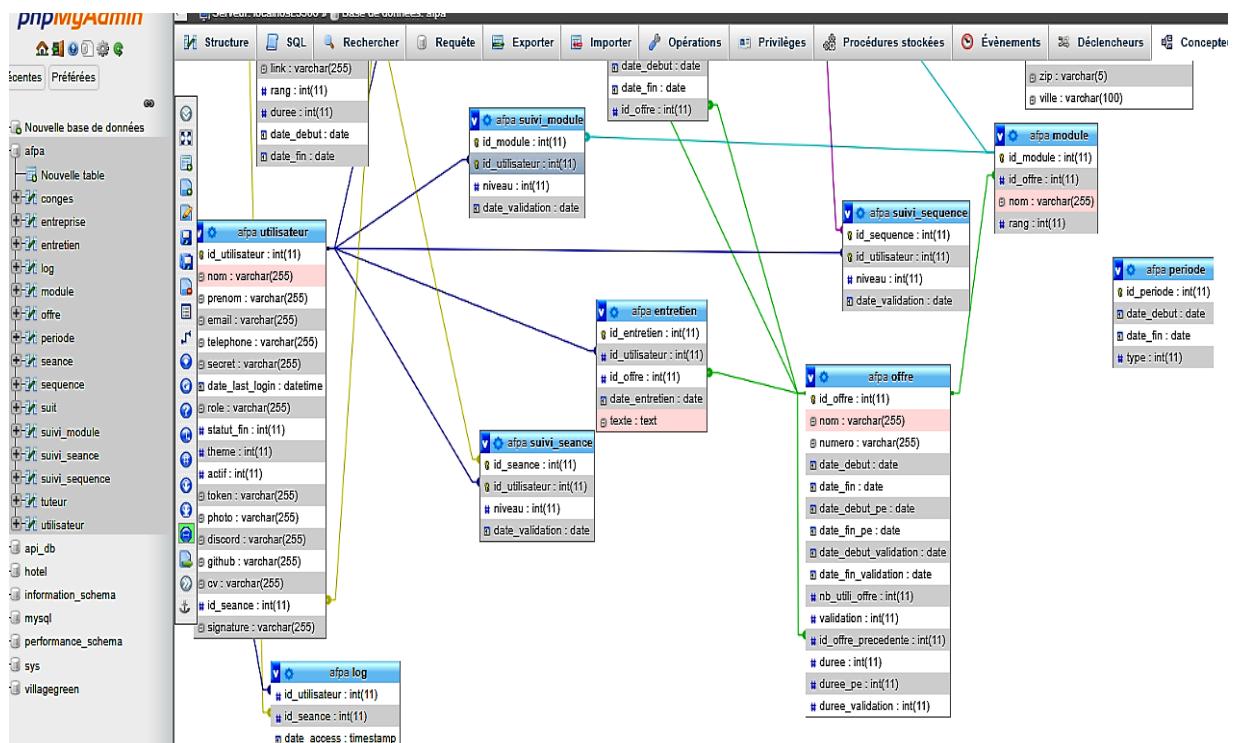
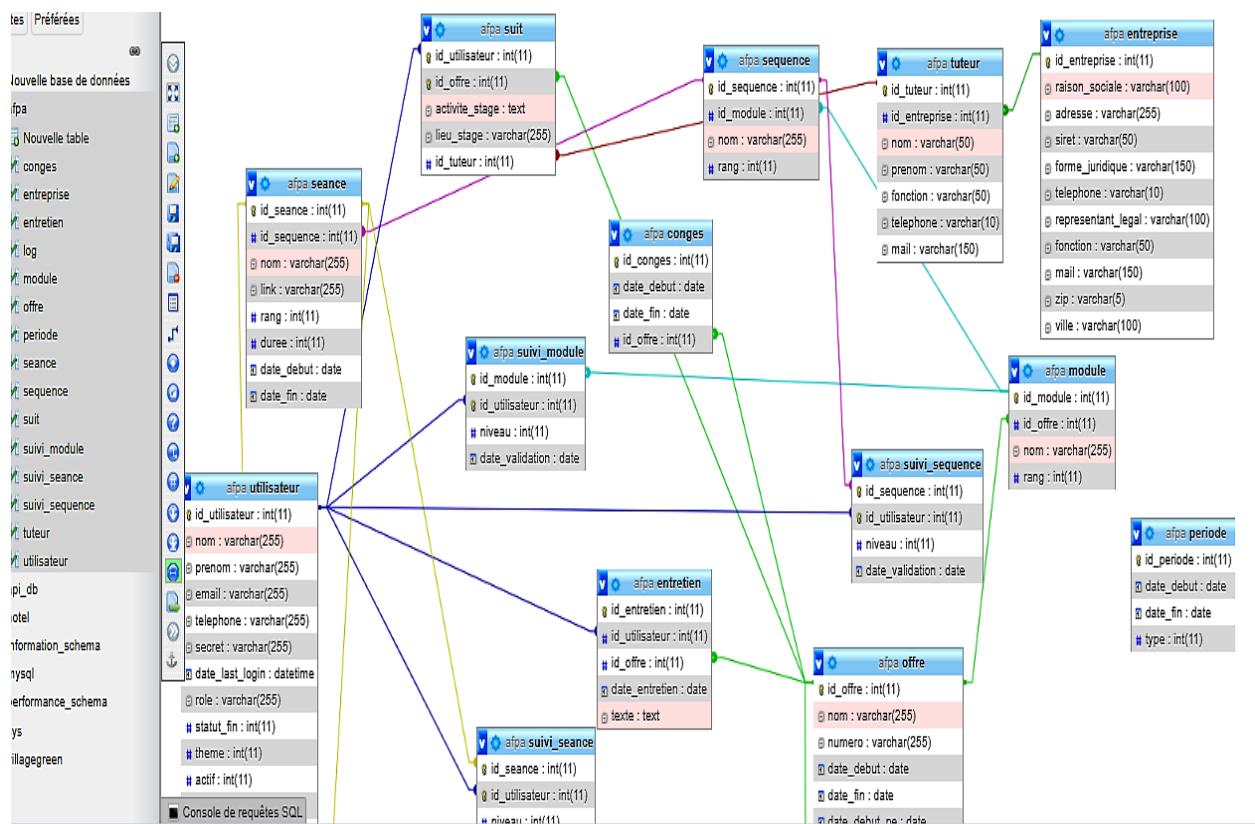


Figure 1 : Vue relationnelle de la base



**Figure 2 : Vue relationnelle de la base**

Beaucoup de tests de requêtes ont été fait avant de commencer à développer :

Table	Action	Lignes	Type	Interclassement	Taille	Perte
auth	Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	16,0 kio	-
conges	Parcourir Structure Rechercher Insérer Vider Supprimer	10	InnoDB	utf8_general_ci	32,0 kio	-
entreprise	Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8_general_ci	16,0 kio	-
entretien	Parcourir Structure Rechercher Insérer Vider Supprimer	308	InnoDB	utf8_general_ci	224,0 kio	-
log	Parcourir Structure Rechercher Insérer Vider Supprimer	~54 749	InnoDB	utf8_general_ci	7,5 Mio	-
module	Parcourir Structure Rechercher Insérer Vider Supprimer	163	InnoDB	utf8_general_ci	32,0 kio	-
offre	Parcourir Structure Rechercher Insérer Vider Supprimer	58	InnoDB	utf8_general_ci	32,0 kio	-
periode	Parcourir Structure Rechercher Insérer Vider Supprimer	33	InnoDB	utf8_general_ci	16,0 kio	-
seance	Parcourir Structure Rechercher Insérer Vider Supprimer	1 380	InnoDB	utf8_general_ci	176,0 kio	-
sequence	Parcourir Structure Rechercher Insérer Vider Supprimer	422	InnoDB	utf8_general_ci	64,0 kio	-
suit	Parcourir Structure Rechercher Insérer Vider Supprimer	244	InnoDB	utf8_general_ci	64,0 kio	-
suivi_module	Parcourir Structure Rechercher Insérer Vider Supprimer	608	InnoDB	utf8_general_ci	80,0 kio	-
suivi_seance	Parcourir Structure Rechercher Insérer Vider Supprimer	5 917	InnoDB	utf8_general_ci	480,0 kio	-
suivi_sequence	Parcourir Structure Rechercher Insérer Vider Supprimer	1 516	InnoDB	utf8_general_ci	176,0 kio	-
tuteur	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8_general_ci	32,0 kio	-
utilisateur	Parcourir Structure Rechercher Insérer Vider Supprimer	197	InnoDB	utf8_general_ci	80,0 kio	-

**Figure 3 Structure de la base de donnée NCode**

## II.1.2. L'API Ncode

Pour mettre en place l'API, les informations et données à récupérer ont été fixée lors des réunions de travail avec la direction en accord avec mon maître de stage. Les tests des résultats de l'API j'ai utilisé Postman et Insomnia, qui sont des plateformes de collaboration pour le développement d'API. Leurs fonctionnalités simplifient chaque étape de la création d'une API et rationalisent la collaboration afin de créer de meilleures API plus rapidement.

L'API a été développée sur NetBeans et Visual Studio Code et j'ai utilisé Laragon comme serveur Apache en local qui est un environnement de développement universel, et la base de données amorceoveffoad de l'AFPA en MySQL et PHPMyAdmin.

Notre API est organisée comme suit :

```
─ config
  └── database.php
─ objects
  ├── offre.php
  └── utilisateur.php
─ offre
  ├── read.php
  ├── readAll.php
  ├── readMonth.php
  ├── readOne.php
  └── readSum.php
─ user
  ├── readAllUser.php
  ├── userNom.php
  └── userPrenom.php
```

### II.1.2.1 Config :

Le dossier *config* contient la classe *Database* qui établit la connexion à la base de données amorceoveffoad. Elle instancie un PDO et puis elle l'exécute.

PDO : l'interface de connexion à une base de données en PHP.

```
class Database {
```

Connexion à la base, On spécifie les coordonnées de connexion

```
// On spécifie les coordonnées de connexion
private $host = "localhost";
private $db_name = "amorceoveffoad"; // nom de la base amorceoveffoad
private $dsn; // concaténé les info de connex à la base
private $username = "root";
private $password = "";
public $conn; // objet de connexion
```

Méthode de connexion à la base : getConnection()

```
public function getConnection() {
    $this->conn = null; // initialiser la conn a null
    $this->dsn = "mysql:host=" . $this->host . ";dbname=" . $this-
>db_name;
    try {
        $this->conn = new PDO($this->dsn, $this->username, $this-
>password);
        $this->conn->exec("set names utf8");//instancier un PDO +executee
    } catch (PDOException $exception) {
        echo "Conneciton error: " . $exception->getMessage();
    }
    return $this->conn;
}
```

### II.1.2.2 Objects

Le dossier objets contient nos classe objets *Offre* et *Utilisateur* qui eux contiennent les informations de nos tables *Offre* et *Utilisateur* dans notre base de données, mais aussi contiennent des méthodes qui vont nous permettent de préparer et exécuter les requêtes dont on aura besoin.

#### Offres :

On m'a demandé de récupérer ces informations dans le cahier des charges : *Nom\_offre*, *Numero\_offre*, *Date\_debut*, *Date\_fin*, *Nom\_stagiaire*, *Prenom\_stagiaire*, *Email\_stagiaire* classé par date début.

Je déclare mes attributs de la classe nécessaire à l'utilisation de mon objet et un constructeur.

```
class Offre {
    //initialisé la connexion a la base
    private $db;
    // connexion à la base
    private $conn;
    private $table_name = "offre";
    private $table_name2 = "utilisateur";
    // membres de l'objet (les champs de la table offre)
    public $id_offre;
    public $nom;
    public $numero;
    public $date_debut;
    public $date_fin;
    public $date_debut_pe;
    public $date_fin_pe;
    public $date_debut_validation;
    public $date_fin_validation;
```

```

    public $nb_utili_offre;
    public $validation;
    public $id_offre precedente;
    public $duree;
    public $duree_pe;
    public $duree_validation;
    public $resultats;

// je crée un constructeur pour instancier ma class et lui attribuer une conn a la base
    public function __construct($db) {
        $this->conn = $db;
    }
....}

```

```

    public function read() {
        // requête select* de la table offre (tt les groupes)
        $query = "SELECT o.id_offre, o.nom AS 'nom_offre', o.numero AS 'numero_offre', o.date_debut AS 'date_debut', o.date_fin AS 'date_fin', o.nb_utili_offre AS 'nb_stagiaire' FROM `offre` o ORDER BY offre.date_debut";
    }

```

On prépare la requête :

```
$stmt = $this->conn->prepare($query);
```

On exécute la requête :

```

        // on execute la requête
        $stmt->execute();
        return $stmt;
    }

```

PDOStatement::execute — Exécute une requête préparée

```

public function readAll() {
    // requête select* de la table offre (tt les groupes)
    $query = "SELECT * FROM "
        . $this->table_name;
    // on prépare la requête
    $stmt = $this->conn->prepare($query);
    // on execute la requête
    $stmt->execute();
    return $stmt;
}

```

La fonction readAll récupère toutes les valeurs de notre table *Offre* et affiche donc toutes les informations des groupes de stagiaires.

Pour lire une seule offre en passant son numéro ou son id en paramètre : requête pour lire un seul enregistrement.

Requête pour lire 1 enregistrement

Afin de récupérer une information ciblée en passant un id ou autre en paramètre il faut utiliser une requête avec une condition *WHERE* ;

La commande *WHERE* dans une requête SQL permet d'extraire un jeu de résultats d'une base de données qui respecte une condition. Cela permet d'obtenir uniquement les informations désirées.

```
public function readOne() {
```

```
    $query = "SELECT offre.id_offre, offre.nom AS 'nom_offre', offre.numero AS 'numero', offre.date_debut AS 'date_debut', offre.date_fin AS 'date_fin', offre.nb_utili_offre AS 'nb_stagiaire', utilisateur.nom AS 'nom', utilisateur.prenom, utilisateur.email, utilisateur.telephone, utilisateur.role FROM `offre` JOIN suit ON suit.id_offre = offre.id_offre JOIN utilisateur ON utilisateur.id_utilisateur=suit.id_utilisateur WHERE offre.numero = ?";
```

On prépare la requête

```
$stmt = $this->conn->prepare($query);
```

On met l'id à sa place

```
$stmt->bindParam(1, $this->numero);
```

Le bindParam c'est une fonctionnalité de PDO

PDOStatement::bindParam — Lie un paramètre à un nom de variable spécifique. Je l'ai préféré à bindValue qui lui

PDOStatement::bindValue — Associe une valeur à un paramètre.

On exécute et on le retourne

```
$stmt->execute();
return $stmt;
```

D'autres fonctions et requêtes que je ne vais pas détailler ont été faite comme readSUM qui récupère la somme des stagiaires par offre et qui ont la même date de fin de formation, readMonth lui récupère la liste des stagiaires qui finissent mois par mois en choisissant le mois en paramètre exemple janvier = mois 1, et ainsi de suite.

Ces méthodes ont été créées par anticipation d'un futur besoin ou une évolution de nos applications.

D'autres méthodes aussi créées et abandonnées au fil du projet.

### II.1.2.3 Utilisateur :

Comme avec la classe *Offre*, ici aussi je déclare mes attributs et je crée notre constructeur

```
// connexion à la base
private $conn;
private $table_name = "utilisateur";
// membres de l'objet
public $id_utilisateur;
public $nom;
public $prenom;
public $email;
public $telephone;
```

```

public $secret;
public $date_last_login;
public $role;
public $statut_fin;
public $theme;
public $actif;
public $token;
public $photo;
public $discord;
public $github;
public $cv;
public $id_seance;
public $signature;

// un constructeur avec $db comme connexion à la base
public function __construct($db) {
    $this->conn = $db;
}

```

On veut permettre à l'utilisateur de trouver un stagiaire en tapant juste quelques lettres de son nom ou de son prénom d'où le *WHERE u.nom LIKE ? OR u.prenom LIKE ?*.

```

public function chercheNom() {
$query = "SELECT u.nom, u.prenom , u.email, u.telephone, u.role, o.nom AS 'nom
_offre',"
        . "o.numero AS 'numero_offre', o.date_debut AS 'date_debut', "
        . "o.date_fin AS 'date_fin' FROM `utilisateur` u JOIN suit ON "
        . "u.id_utilisateur = suit.id_utilisateur JOIN offre o "
        . "ON o.id_offre=suit.id_offre WHERE u.nom LIKE ? OR u.prenom LIKE ? "
;

```

*On prépare la requête*

```
$stmt = $this->conn->prepare($query);
```

Je crée une variable intermédiaire que j'ai appelé question

```
$question = "%" . $this->nom . "%";
```

Envoie, ici, deux paramètre que je stock dans une variable intermédiaire \$question, le 1 est le nom le 2 le prénom. Les pourcentages nous permettent de chercher une chaîne de caractère soit au début ou au milieu ou à la fin du mot. En résumé si notre nom ou prénom contient cette chaîne de caractère.

```
$stmt->bindParam(1, $question);
$stmt->bindParam(2, $question);
```

*On exécute la requête et on retourne le statement (jeu de résultat de ma requête).*

```

$stmt->execute();
return $stmt;
}
```

J'ai aussi une fonction readAll qui lit tous les utilisateurs de ma table.

```

public function readAllUser() {
    // requête select* de la table utilisateur
    $query = "SELECT * FROM `utilisateur` WHERE role = 'stagiaire'";
    // on prépare la requête
    $stmt = $this->conn->prepare($query);
    // on execute la requête
    $stmt->execute();
    return $stmt;
}

```

- PDOStatement::execute — Exécute une requête préparée

### II.1.2.3 Dossier offre :

Nos portes d'accès à notre API, sont ces méthodes read, readAll, readOne, etc., qui récupèrent les résultats des requêtes et qui les affiche au format JSON. Elles vont lire nos listes d'objet de nos classes *Offre* et *Utilisateur*, mais pour cela il nous faut tout d'abord des headers.

JSON : JavaScript Object Notation, JSON est un format de données textuelles dérivé de la notation objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

JSON est utilisé pour transporter de l'information. C'est un support d'échange de données entre le serveur et le navigateur.

#### L'action Read :

Elle nous affiche tous les groupes et leurs utilisateurs, c'est la porte d'entrée de notre API avec *Offre* et sa méthode read().

Les headers dont nous avons besoin

```
header("Access-Control-Allow-Origin: *");
```

Qui nous autorise l'accès de n'importe quel navigateur (public tout site).

```
header("Content-Type: application/json; charset=UTF-8");
```

Renvoyer du JSON, même si l'URL est en PHP avec cet header, il reconnaît qu'il renvoie du JSON

```
header("Access-Control-Allow-Methods: GET");
```

Limite l'accès qu'aux méthodes GET puisque, dans mon cas, je ne vais pas utiliser de méthode POST, delete ou autre pour modifier des informations dans ma base, j'ai donc besoin de ne récupérer que des informations et non les modifier.

```
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-  
Headers, Authorization, X-Requested-With"); //header autorisé
```

Pour commencer, on vérifie si la méthode utilisée est bien GET

```
if($_SERVER['REQUEST_METHOD'] == 'GET'){ si oui on affiche la suite}  
On gère l'erreur url (get) et on renvoie l'erreur 405.
```

```
else{  
    // On gère l'erreur url (get)  
    http_response_code(405);  
    echo json_encode(["message" => "La méthode n'est pas autorisée"]);  
}
```

Une fois la méthode vérifiée, on va faire les imports et les inclusions de notre connexion à la base et des classe *Objects*.

```
// faire nos imports, inclusion de database et offre  
include_once '../config/database.php';  
include_once '../objects/offre.php';
```

On instancie la connexion à la base et on génère une connexion avec :

```
// instantiation de database et de offre et generation d'une connexion  
$database = new Database();  
$db = $database->getConnection();
```

On initialise notre classe objet *Offre* et lui envoie la connexion

```
// On initialise offre  
$offre = new Offre($db);
```

On récupère les données en utilisant la méthode read de notre classe *Offre*.

```
// On récupère les données avec la methode read de offre  
$stmt = $offre->read();
```

Si on a des données ou notre stm nous renvoi un résultat alors on récupère nos résultats de la requête dans un tableau associatif que j'appelle ici : \$offres\_arr = [];

Pour récupérer, ligne par ligne, on va utiliser un fetch\_ASSOC de PDO :

```
if ($stmt->rowCount() > 0) {  
    // tableau de groupe(offre) On initialise un tableau associatif  
    $offres_arr = [];  
    // on récupère le contenu de la table ligne par ligne (au lieu de faire un  
    fetchAll)  
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {  
        extract($row);  
        $offre_item = array(  
            "id_offre"=> $id_offre,  
            "nom_offre" => $nom_offre,  
            "numero_offre" => $numero_offre,  
            "date_debut" => $date_debut,  
            "date_fin" => $date_fin,  
            "nb_stagiaire" =>$nb_stagiaire,
```

```
    );
    array_push($offres_arr, $offre_item);}
```

Explication :

PDOStatement::rowCount — Retourne le nombre de lignes affectées par le dernier appel à la fonction PDOStatement::execute()

- PDOStatement::fetch — Récupère la ligne suivante d'un jeu de résultats PDO
- le fetch\_assoc :Récupère une ligne de résultat sous forme de tableau associatif
- PDOStatement::fetchAll — Retourne un tableau contenant toutes les lignes du jeu d'enregistrements
- PDOStatement::fetchObject — Récupère la prochaine ligne et la retourne en tant qu'objet

On envoie la réponse HTTP à 200 OK, requête bien exécutée

```
http_response_code(200);
```

Et on affiche la réponse (tableau d'objet) en JSON. On encode en JSON.

```
echo json_encode($offres_arr);
```

Si on a aucun résultat de \$stm ou s'il y a un problème de connexion à la base, on retourne l'erreur (404)

```
else {// on renvoie le code 404 Not found
    http_response_code(404);
```

On avertit l'utilisateur en affichant un message

```
echo json_encode(array("message" => "Aucun groupes ni utilisateur trouvés."));
```

On aura ce résultat en JSON en utilisant URL suivant sur Postman:

[https://dev.amorce.org/APIcode/API\\_Ncode/offre/read.php](https://dev.amorce.org/APIcode/API_Ncode/offre/read.php)

```
[
    {
        "id_offre": "69",
        "nom_offre": "DWWM",
        "numero_offre": "20054",
        "date_debut": "2020-09-28",
        "date_fin": "2021-05-28",
        "nb_stagiaire": "6"
    },
    {
        "id_offre": "66",
        "nom_offre": "CDA",
        "numero_offre": "20052",
        "date_debut": "2020-08-31",
        "date_fin": "2021-06-04",
        "nb_stagiaire": "6"
    },
    .......]
```

L'action ReadOne :

Avec notre porte d'accès readOne, on va utiliser pour envoyer le paramètre (numéro) en GET.

En résumé, cette action nous permet la connexion et la récupération d'un seul groupe de stagiaire, en récupérant le numéro de l'offre en paramètre en GET dans l'URL, on lance la requête avec la méthode readOne(). On récupère dans un tableau et on renvoie en JSON

```
// Récupérons le numéro de l'objet à lire en get
```

```
$offre->numero = isset($_GET['numero']) ? $_GET['numero'] : die();
```

Si ma valeur existe (numéro en GET) sinon j'annule mon lancement de requête.

Pour le reste, on fera comme avec read, mais cette fois on va utiliser la méthode readOne de la classe objet *Offre*.

Je ne vais pas détailler toutes les requêtes et les porte d'entrées de l'API mais, dans la logique, on suit les mêmes exemples. Notre API nous renvoie bien les résultats recherchés et les informations voulus.

Si on tape le lien: [http://localhost/API\\_Ncode/offre/readOne.php?numero=19123](http://localhost/API_Ncode/offre/readOne.php?numero=19123)

Pour chercher une offre on aura un tableau d'objet JSON de ce type sur Postman ou Insomnia

[

```
{
    "id_offre": "37",
    "nom_offre": "CDA",
    "numero": "19123",
    "nb_stagiaire": "4",
    "date_debut": "2019-11-18",
    "date_fin": "2020-07-24",
    "nom": "Sagez",
    "prenom": "Jason",
    "email": "jsagez1998@gmail.com",
    "telephone": "0628696332",
    "role": "stagiaire"
},
{
    "id_offre": "37",
    "nom_offre": "CDA",
    "numero": "19123",
    "nb_stagiaire": "4",
    "date_debut": "2019-11-18",
    "date_fin": "2020-07-24",
    "nom": "Benchickh",
    "prenom": "hamid",
    "email": "hamid.benchikh@gmail.com",
    "telephone": "0659883304",
    "role": "stagiaire"
},
{
    "id_offre": "37",
    "nom_offre": "CDA",
    "numero": "19123",
    "nb_stagiaire": "4",
    "date_debut": "2019-11-18",
    "date_fin": "2020-07-24",
    "nom": "Ory",
    "prenom": "Tristan",
    "email": "t.ory@laposte.net",
    "telephone": "0682682822",
    "role": "stagiaire"
}
```

]

#### II.1.2.4 Recherche d'utilisateurs :

Pour la recherche du nom ou prénom, on va utiliser la porte d'entrée user/readNom et la méthode rechercheNom() de l'objet utilisateur.

```
// Les headers dont nous avons besoin
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: GET");
header("Access-Control-Allow-Credentials: true");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-
Headers, Authorization, X-Requested-With");
// On vérifie que la méthode utilisée est correcte
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    //faire les imports
    // inclusion de database et product
    include_once '../config/database.php';
    include_once '../objects/offre.php';
    include_once '../objects/utilisateur.php';
    // instanciation de database et de product et génération d'une connexion
    $database = new Database();
    $db = $database->getConnection();
    // On initialise offre et utilisateur
    $utilisateur = new Utilisateur($db);
    // Récupérons le nom de l'objet à lire en get
    $utilisateur->nom = isset($_GET['nom']) ? $_GET['nom'] : die();
    // le query
    $stmt = $utilisateur->chercheNom();
    // on regarde si on a un résultat
    if ($stmt->rowCount() > 0) {
        $result = [];
        // on récupère le contenu de la table et on associe les valeurs
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            extract($row);
            $result_item = array(
                "nom_offre" => $nom_offre,
                "numero_offre" => $numero_offre,
                "date_debut" => $date_debut,
                "date_fin" => $date_fin,
                "nom" => $nom,
                "prenom" => $prenom,
                "email" => $email,
                "telephone" => $telephone,
                "role" => $role
            );
            array_push($result, $result_item);
        }
        // on envoie la réponse http à 200 OK si requête bien exécutée
        http_response_code(200);
        // on renvoie et on affiche la réponse (tableau) en Json
        echo json_encode($result);
    }
}
```

[http://localhost/API\\_Ncode/user/userNom.php?nom=hami](http://localhost/API_Ncode/user/userNom.php?nom=hami)

```
[  
  {  
    "nom_offre": "TB",  
    "numero_offre": "19117",  
    "date_debut": "2019-09-23",  
    "date_fin": "2019-11-15",  
    "nom": "Benchickh",  
    "prenom": "hamid",  
    "email": "hamid.benchikh@gmail.com",  
    "telephone": "0659883304",  
    "role": "stagiaire"  
  },  
  {  
    "nom_offre": "CDA",  
    "numero_offre": "19123",  
    "date_debut": "2019-11-18",  
    "date_fin": "2020-07-24",  
    "nom": "Benchickh",  
    "prenom": "hamid",  
    "email": "hamid.benchikh@gmail.com",  
    "telephone": "0659883304",  
    "role": "stagiaire"  
  }  
]
```

La porte d'entrée readAll pour récupérer toutes les informations sur les utilisateurs je ne détaillerai pas plus ici.

### II.1.3 Sécurisation de l'API :

Maintenant que l'on a créé notre API REST, voyons comment sécuriser les échanges entre le client et celle-ci.

#### II.1.3.1 Utilisation de HTTPS:

la première étape de la sécurisation d'une API REST est l'utilisation du protocole HTTPS. Cela permettra de chiffrer les données transmises et reçues empêchant ainsi leur lecture.

[https://dev.amorce.org/APIcode/API\\_Ncode/offre/read.php](https://dev.amorce.org/APIcode/API_Ncode/offre/read.php).

#### II.1.3.2 Authentification:

Il faut ensuite un mécanisme authentification des échanges entre le client et l'API REST. Une API REST est sans état (stateless) c'est-à-dire qu'il n'y a pas de session côté serveur pour l'authentification de l'utilisateur. De ce fait, chaque requête doit contenir les informations nécessaires à l'authentification.

On pourrait très bien utiliser la méthode “Basic” de l'authentification, mais cela implique de transmettre pour chaque requête, le couple login/mot de passe ce qui n'est franchement pas top.

J'ai décidé d'utiliser une clé API qui est la plus pratique dans mon cas que d'utiliser un JWT(JSON Web Token).

Cette étape n'est pas encore terminée (en cours de réalisation).

## II.2. L'application Desktop :

Maintenant que notre API est prête je crée le projet NCode sur IntelliJ, le projet sera créé sous Gradle et non avec Maven car j'ai eu des problèmes de Retrofit avec Maven.

### II.2.1 Création du projet JavaFX avec Gradle :

On ajoute les plugins et les dépendances nécessaires à la compilation:

```
plugins {  
    id 'java'  
    id 'application'  
    id 'org.openjfx/javafxplugin' version '0.0.8'  
}
```

Ce sera grâce aux plugins application et org.openjfx.javafxplugin qui vont nous permettre de développer sous JavaFX ainsi que la déclaration des modules nécessaires au fonctionnement de JavaFX avec Gradle.

Il faut aussi ajouter ces lignes pour pouvoir faire du CSS et encoder en UTF-8 et apporter les modification à build.gradle pour que Gradle prenne en charge le FXML.

```
sourceSets.main.resources.srcDirs("src/main/java").includes.addAll(["**/*.fxml", "**/*.css"])  
sourceSets.main.resources.srcDirs("src/main/resources").includes.addAll(["**/*.*"])  
compileJava.options.encoding = 'UTF-8'
```

Et enfin il faut déclarer la classe principale dans le build.gradle en ajoutant :

```
mainClassName = 'org.afpa.App'
```

C'est la classe qui appelle notre vue principale avec Gradle car il est déprécié de déclarer « extend Application ».

Quelques problèmes sont survenus lors du changement de PC et le projet ne s'exécutait plus, le problème venait de Gradle qui prenait plus les anciennes versions qui était en 6.1.1 qui ne prend pas en compte le JDK 14 je suis donc passé à la version 6.3.

```
gradle.version=6.3  
distributionUrl=https://services.gradle.org/distributions/gradle-6.3-all.zip.
```

Il faut aussi vérifier la version du SDK et la java langue pour le bon fonctionnement du projet. Notre *build.gradle* ressemblera désormais à ça :

The screenshot shows a Java project structure in an IDE. The project root is named 'nnode'. It contains a 'gradle' folder with subfolders like '6.3', 'buildOutputCleanup', 'checksums', and 'vcs-l'. A 'build' folder is selected, containing 'gradle/wrapper' (with 'gradle-wrapper.jar' and 'gradle-wrapper.properties') and 'src/main/java/org.afpa/dal' (containing 'GlobalData.java', 'GroupeData.java', 'NnodeAPI.java', 'RetrofitInstance.java', and 'UserData.java'). It also includes 'gui/main' (with 'main.fxml' and 'Main.java'), 'icon' (with 'afpa.jpg' and 'images.jpg'), 'style' (with 'main\_style.css'), and 'App.java'. The 'src/test' folder contains 'java' and 'resources' subfolders. The 'build.gradle' file is open in the code editor, defining the build configuration with plugins, repositories, source sets, and dependencies.

```

1 plugins {
2     id 'java'
3     id 'application'
4     id 'org.openjfx.javafxplugin' version '0.0.8'
5 }
6 group 'org.afpa'
7 version '1.0-SNAPSHOT'
8 repositories {
9     mavenCentral()
10 }
11
12 sourceSets.main.resources.srcDirs("src/main/java").includes.addAll(["**/*.fxml", "**/*.css"])
13 sourceSets.main.resources.srcDirs("src/main/resources").includes.addAll(["**/*.*"])
14 compileJava.options.encoding = 'UTF-8'
15
16 dependencies {
17     testCompile group: 'junit', name: 'junit', version: '4.12'
18     compile 'com.squareup.retrofit2:retrofit:2.8.1'
19     compile 'com.squareup.retrofit2:converter-gson:2.8.1'
20     compile 'org.kordamp.ikonli:ikonli-javafx:11.4.0'
21     compile 'org.kordamp.ikonli:ikonli-fontawesomeawesome-pack:11.4.0'
22 }
23 javafx {
24     version = "14"
25     modules = ["javafx.fxml", "javafx.base", "javafx.controls", "javafx.graphics", "javafx.web"]
26 }
27
28 mainClassName = 'org.afpa.App'
29

```

Comme montré sur l'image notre projet est organisé comme suit :

```

src
└── main
    ├── java
    │   └── org
    │       └── afpa
    │           └── dal
    │               ├── GlobalData.java
    │               ├── GroupeData.java
    │               ├── NnodeAPI.java
    │               ├── RetrofitInstance.java
    │               └── UserData.java
    └── gui
        └── main
            ├── main.fxml
            ├── Main.java
            └── MainController.java
    └── icon
        ├── afpa.jpg
        └── images.jpg
    └── style
        └── main_style.css
    └── App.java
└── resources
└── test
    ├── java
    └── resources

```

Explication :

Org.afpa : c'est notre groupe ID. domaine de notre compagny.

Dal : data acces logique, c'est le dossier qui va contenir nos classe objet et notre interface API.

Gui : graphique user interface, c'est le dossier qui va contenir notre class main qui entend application et aussi notre interface graphique fxml et son contrôleur.

Icon : dossier pour stocker nos icons, images...

Style : contient nos class css.

App: c'est la class qui appelle notre main lors de l'exécution du projet, expliquée précédemment.

```
package org.afpa;

import org.afpa.gui.main.Main;

public class App {

    public static void main(String[] args) {
        //lancer le main
        Main.main(args);
    }
}
```

Notre main :

```
package org.afpa.gui.main;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("main.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

## II.2.2 L'interface graphique :

Dans cette partie, j'aborderai les interfaces graphiques (on parle aussi d'IHM pour Interfaces Homme Machine ou de GUI pour Graphical User Interfaces) et, par extension, la programmation événementielle. Notre programme ne réagira plus à des saisies au clavier mais à des appels d'événements provenant d'un composant graphique : un bouton, une liste, un menu...

SceneBuilder va nous être de grande utilité sur cette partie.

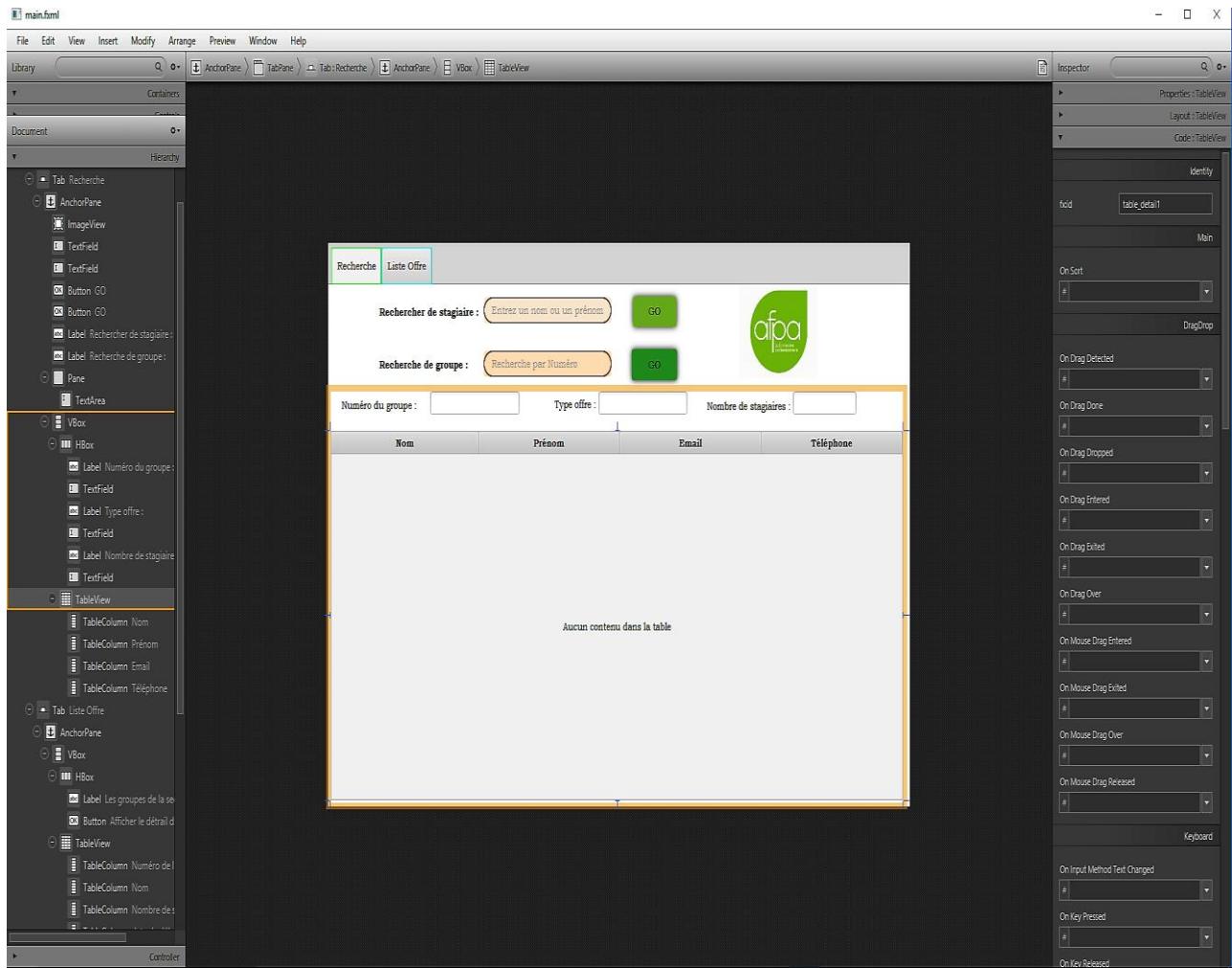


Figure 4 Scene builder (Interface)

```

<AnchorPane prefHeight="611.0" prefWidth="912.0" stylesheets="@/../style/main_style.css" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="org.afpa.gui.main.MainController">
    <children>
        <TabPane layoutX="10.0" layoutY="10.0" prefHeight="568.0" prefWidth="829.0" tabClosingPolicy="UNAVAILABLE" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0">
            <tabs>
                <tab style="-fx-padding: 8; -fx-border-color: #16F324;" text="Recherche">
                    <content>
                        <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="357.0" prefWidth="912.0">
                            <children>
                                <Image fitHeight="102.0" fitWidth="135.0" layoutX="645.0" pickOnBounds="true" preserveRatio="true">
                                    <image>
                                        <Image url="@/icon/afpa.jpg" />
                                    </image>
                                </Image>
                                <Textfield fx:id="text_nom" layoutX="244.0" layoutY="14.0" prefHeight="28.0" prefWidth="200.0" promptText="Entrez un nom ou un prénom" style="-fx-background-color: #FAEACD" />
                                <Textfield fx:id="text_numero" layoutX="244.0" layoutY="72.0" prefHeight="28.0" prefWidth="200.0" promptText="Recherche par Numéro" style="-fx-background-color: #FFE4C4" />
                                <Button layoutX="476.0" layoutY="78.0" mnemonicParsing="false" onAction="#chercherNumero" prefHeight="33.0" prefWidth="71.0" style="-fx-background-color: #48A243;" />
                                <Button layoutX="478.0" layoutY="13.0" mnemonicParsing="false" onAction="#rechercheNom" prefHeight="32.0" prefWidth="68.0" style="-fx-background-color: #88BA3F;" />
                                <Label layoutX="88.0" layoutY="18.0" prefHeight="23.0" prefWidth="159.0" style="-fx-font-weight: bold;" text="Rechercher de stagiaire :" textAlignment="CENTER" />
                                <Label layoutX="88.0" layoutY="78.0" nodeOrientation="LEFT_TO_RIGHT" prefHeight="34.0" prefWidth="154.0" style="-fx-font-weight: bold;" text="Recherche de groupe :" />
                                <Pane fx:id="paneDetailNom" layoutX="154.0" prefHeight="414.0" prefWidth="912.0" visible="false">
                                    <children>
                                        <TextArea fx:id="text_result" layoutX="110.0" layoutY="2.0" prefHeight="411.0" prefWidth="549.0" />
                                    </children>
                                </Pane>
                            </children>
                        </AnchorPane>
                    </content>
                </tab>
            </tabs>
        </TabPane>
    </children>

```

L'outil **SceneBuilder** permet de manipuler et d'éditer les fichiers FXML.

- La hiérarchie d'une application Java FX est relativement simple : un `Stage` qui contient une `Scene` qui contient des composants.

On attribuera des ID (`fx:id`) à nos champs de textes à nos inputs, les tables, colonnes... et des méthodes à nos boutons qu'on appellera dans notre contrôleur pour pouvoir faire nos évènements contrôler nos saisies, récupérer nos valeurs afficher nos données ...

Sans oublié de relier notre fichier à notre contrôleur `MainController.java` et à notre feuille de style `main_style.css`, faire aussi tout les imports nécessaires.

```

<AnchorPane prefHeight="611.0" prefWidth="912.0" stylesheets="@/../style/main_style.css"
    xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="org.afpa.gui.main.MainController">
```

Il reste plus qu'à déclarer nos composant `fx:id` dans notre contrôleur et faire les imports.

```

public class MainController implements Initializable {
    @FXML
    public TableView<GlobalData> list_offre;
    @FXML
    public TableColumn col_numero, col_nom, col_nombre_stagiaire, col_date_debut, col_date_fin, col_id_offre;
    ObservableList<GlobalData> observableList = FXCollections.observableArrayList();
    //detail offre
    @FXML
    public VBox paneDetail;
    @FXML
    public TextField text_num_group, text_nom_group;
```

```

@FXML
public TableView table_detail;
@FXML
public TableColumn colnom, colprenom, colmail, coltel;
//recherche par numero d'offre
public VBox paneDetailnum;
public TextField text_numero;
public TextField text_num_group1, text_nom_group1, text_nb_stagiaire;
public TableView table_detail1;
public TableColumn colnom1, colprenom1, colmail1, coltel1;
//recherche par nom
public Pane paneDetailNom;
public TextArea text_result;
//tab recherche
@FXML
public TextField text_nom;
ObservableList<GroupeData> detail = FXCollections.observableArrayList();

```

On remarqué que tous les champs que nous avons renommé dans SceneBuilder sont présent ici et annoté avec @FXML . Cette annotation permet de spécifier à quoi le fichier FXML pourra accéder. Ensuite nous déclarons un constructeur par défaut, convention oblige puis nous voyons une méthode initialize() . Cette méthode est appelée automatiquement après le chargement du fichier FXML, donc après le chargement de l'interface graphique.

```

@Override
public void initialize(URL location, ResourceBundle resources) {
    chargeTableau();
}

```

### II.2.2.3 Evolution du projet :

Il m'a été demandé, au début, de faire une WebView qui donne accès au planning des formations à partir de notre application Desktop, cette étape a besoin d'une authentification. On voulait passer un header et une clé lors du chargement du lien.

```

@FXML
private WebView webView;
private WebEngine engine;
public void chargerWebview(){
    engine = webView.getEngine();
    engine.load("https://ncode.amorce.org/offre/planning/\\");
}

```

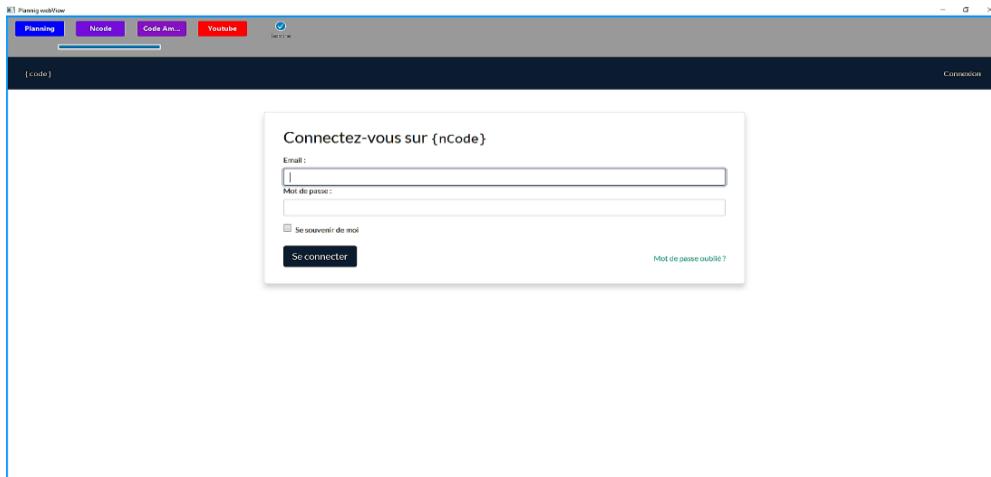


Figure 5 vue sur le planning (webView)

J'ai essayé de passer plus de paramètres que le lien en ajoutant les header et clés mais c'était impossible. Germain SIPIERE, mon maître de stage, a lui aussi essayé et après plusieurs jours. Au final, nous avons décidé de le retirer du projet, pour se concentrer sur le reste et le faire en Android ou JavaScript.

J'ai proposé, par la suite, quelques améliorations pour notre interface graphique, voici quelques prise d'écran. Au final, nous avons opté pour une version plus simple pour faciliter son utilisation désormais l'application possède deux tabpane un pour la recherche d'utilisateurs et l'autre pour l'affichage des offres et leurs détails.

Figure 6 vues interfaces

### **II.2.3 Retrofit et persistance des données**

#### **Retrofit :**

C'est l'une des bibliothèques de réseaux largement utilisées dans Android. De par sa conception et sa facilité d'utilisation, de nature très flexible et offre une large gamme de fonctionnalités telles que le support de divers JSON Parser tels que GSON, Jackson, Moshi, le support de Rx-Java, ...etc.

Retrofit facilite la récupération et le téléchargement de JSON (ou d'autres données structurées) via un service Web basé sur REST. Retrofit utilise la bibliothèque OkHttp pour les requêtes HTTP.

Cette bibliothèque facilite le téléchargement de données JSON ou XML depuis une API Web. Une fois les données téléchargées, elles sont analysées dans un objet POJO (Plain Old Java Object) défini pour chaque requête à l'aide de n'importe quel adaptateur / analyseur répertorié, Retrofit est une librairie qui simplifie la consommation d'API, On peut la trouver sur [GitHub](#).

J'ai utilisé la librairie [retrofit2](#) pour récupérer les informations et des données de la base de données devamorce via l'API (NCode) que j'ai créé.

### **II.2.4 Configuration et Mise en place :**

Pour travailler avec Retrofit, nous avons essentiellement besoin des trois classes suivantes :

- Classe de modèle utilisée comme modèle JSON
- Interfaces qui définissent les opérations HTTP possibles
- Classe Retrofit.Builder - Instance qui utilise l'interface et l'API Builder pour permettre de définir le point de terminaison URL pour les opérations HTTP.

Pour incorporer retrofit à notre projet, j'ai ajouté à mon projet Gradle dans le build.gradle (Module: App) à nos dépendances ainsi qu'un convertisseur qui transformera nos objets JSON en objets Java.

```
dependencies {  
    compile 'com.squareup.retrofit2:retrofit:2.8.1'  
    compile 'com.squareup.retrofit2:converter-gson:2.8.1'  
}
```

Créer le bon POJO Plain Old Java Object (modèle) en fonction de notre réponse Json : En d'autre terme on va créer la correspondance entre le json et un objet java.

IntelliJ nous facilite la création de POJO en ajoutant un plugin par exemple Json2Pojo ou autre, puis créer des getters. En copiant notre réponse en JSON récupérer dans Postman ou Insomnia et generate new pojo avec IntelliJ.

[https://dev.amorce.org/APIcode/API\\_Ncode/offre/read.php](https://dev.amorce.org/APIcode/API_Ncode/offre/read.php)

```
{  
    "id_offre": "69",  
    "nom_offre": "DWWM",  
    "numero_offre": "20054",  
    "date_debut": "2020-09-28",  
    "date_fin": "2021-05-28",  
    "nb_stagiaire": "6"  
},
```

Les tableaux représentés par [] ; Les objets représenté par {}.

Ce qui vous aidera à créer notre modèle (class objet).

```
package org.afpa.dal;  
import com.google.gson.annotations.SerializedName;  
public class GlobalData{  
    @SerializedName("nom_offre")  
    private String nomOffre;  
    @SerializedName("date_debut")  
    private String dateDebut;  
    @SerializedName("date_fin")  
    private String dateFin;  
    @SerializedName("id_offre")  
    private String idOffre;  
    @SerializedName("numero_offre")  
    private String numeroOffre;  
    @SerializedName("nb_stagiaire")  
    private String nbStagiaire;  
    public String getNomOffre() {  
        return nomOffre;  
    }  
    public String getDateDebut() {  
        return dateDebut;  
    }  
    public String getDateFin() {  
        return dateFin;  
    }  
    public String getIdOffre() {  
        return idOffre;  
    }  
    public String getNumeroOffre() {  
        return numeroOffre;  
    }  
    public String getNbStagiaire() {  
        return nbStagiaire;  
    }  
}
```

Je fais pareil pour les autres classes tel que : GroupeData, UserData...

Les annotations @SerializedName proviennent de la bibliothèque GSON et nous permettent de serialize et deserialize cette classe en JSON utilisant le nom sérialisé comme clé. En d'autre termes il nous permet de renommer nos objets tout en faisant le lien avec l'objet JSON renvoyé.

J'ai créé en tout 3 classes objet récupérés de nos objets JSON : la *GlobalData.java* qui représente toutes les offres proposées par l'AFPA et le nombre de stagiaire par offre.

*GroupeData.java* : liste de stagiaire par offre.*et UserData.java* : information de chaque stagiaire.

Ensuite, nous avons besoin d'une instance de Retrofit qui joue le rôle de contrôleur pour toutes les requêtes et réponses.

Remarque : Nous préférons créer ce contrôleur en tant que singleton, ce qui est très utile si on souhaite définir des propriétés supplémentaires du client.

Le singleton est un patron de conception (*design pattern*) dont l'objectif est de restreindre l'instanciation d'une classe

```
package org.afpa.dal;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
public class RetrofitInstance {
    private static Retrofit retrofit;
    private static final String BASE_URL = "https://dev.amorce.org/APIcode/API_Ncode/";
    public static Retrofit getRetrofitInstance(){
        if(retrofit == null){
            retrofit = new retrofit2.Retrofit.Builder()
                .baseUrl(BASE_URL)//base url du site de l'API
                .addConverterFactory(GsonConverterFactory.create()) //convertor en gson
                .build(); //le construire
        }
        return retrofit;
    }
}
```

Base URL du site de l'API, on créer le convertor en GSON, puis construire.

Ensuite, on crée une classe d'interface pour définir tous les appels d'API avec la requête, le type de réponse et les paramètres de requête pour chaque appel (nous devons créer une interface pour gérer les appels d'URL tels que GET, POST ...etc.)

```
import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;
import java.util.List;
public interface NcodeAPI {
    @GET("offre/read.php")
    Call<List<GlobalData>> getGlobalData();
    //La requête pour avoir uniquement l'offre qui nous intéresse
```

```

    @GET("offre/readOne.php/")
    Call<List<GroupeData>> getGroupeData(@Query("numero") int numero);
    //recherche par nom
    @GET("user/userNom.php/")
    Call<List<UserData>> getUserDataNom(@Query("nom") String nom);
}

```

Cette interface décrit un WebService contenant ces méthodes :

- `getGlobalData` affiche la liste des offres.
- `getGroupeData` recherche des offres en fonction de leurs numéro
- `getUserDataNom` recherche des utilisateurs en fonction de leurs nom ou prénom.

Chaque méthode d'une interface représente un appel d'API possible. Il doit avoir une annotation HTTP (GET, POST, etc.) pour spécifier le type de demande et l'URL relative. La valeur de retour encapsule la réponse dans un `Call` avec le type du résultat attendu.

Exemple 1 : passer une requête pour sélectionner toutes les listes de groupe

```

    @GET("offre/read.php")
    Call<List<GlobalData>> getGlobalData();

```

Exemple 2 : pour chercher la liste de stagiaire et passer le nom ou le prénom en paramètre avec `@Query`

```

    @GET("offre/readOne.php/")
    Call<List<GroupeData>> getGroupeData(@Query("numero") int numero);

```

Exemple 3 : recherche offre en passant le numéro de l'offre en paramètre

```

    @GET("user/userNom.php/")
    Call<List<UserData>> getUserDataNom(@Query("nom") String nom);

```

Le `Call` de retrofit c'est un appel aux class objet pour récupérer les objets JSON.

J'ai utilisé la méthode `@Query` au lieu `@Path` pour envoyer les paramètres de nos requêtes parce que nos paramètres sont saisis soit directement par l'utilisateur ou sélectionné dans nos données récupérées ce que la méthode `@Path` ne permet pas. Cette méthode oblige le passage du paramètre dans l'url dès le départ sans laisser le choix ou d'interaction avec l'application.

### **II.2.5. Récupérer et afficher le résultat dans les applications :**

Retrofit facilite la lecture des données renvoyées par le WebService. Pour cela, il retournera directement le résultat dans des objets Java.

Par exemple, dans le cas de `getGlobalData`, vous avez pu remarquer que j'ai écrit `Call<List<GlobalData>>` avant le nom de la méthode. Ici, Retrofit va automatiquement transformer le résultat sous forme d'une `ArrayList` de `GlobalData`.

On fait appel à notre class retrofit, en lui demandant de créer l'instance de retrofit (créer instance d'objet grâce à notre interface)

Il nous faut maintenant fournir notre interface à un objet nommé Retrofit.Builder. C'est lui qui va nous retourner une implémentation de notre WebService

```
//on instancie retrofit  
NcodeAPI ncodeAPI = RetrofitInstance.getRetrofitInstance().create(NcodeAPI.class);
```

Maintenant on récupérer les données grâce à nos méthodes *getGlobalData*, *getGroupeData*, *getUserDataNom...*, de notre interface.

Nous pouvons ensuite réaliser nos appels du WebService en utilisant l'objet Call créé par le Retrofit, pour cela il nous offre plusieurs possibilités : synchrone ou asynchrone. Dans notre cas j'utilise des appels asynchrones.

On utilise *enqueue()* mise en queue avec un Callback qui appelle nos classe objet.

Le retour du WebService s'effectue par un callback, un objet dont les méthodes seront appelées suite à la réception de données du WebService.

La méthode *enqueue()* doit implémenter deux méthodes: *onResponse* et *onFailure* (réponse ou échec). De plus, une réponse peut aussi être un code erreur d'où *leif(response.isSuccessful()){...}*.

```
Call<List<GroupeData>> call = ncodeAPI.getGroupeData(num);  
call.enqueue(new Callback<List<GroupeData>>() {  
    @Override  
    public void onResponse(Call<List<GroupeData>> call, Response<List<GroupeData>> response)  
    {  
        if (!response.isSuccessful()) {.....}  
        detail = FXCollections.observableArrayList(response.body());  
        text_nom_group.setText(response.body().get(0).getNomOffre());  
        text_num_group.setText(response.body().get(0).getNumero());  
        colnom.setCellValueFactory(new PropertyValueFactory<>("nom"));  
        colprenom.setCellValueFactory(new PropertyValueFactory<>("prenom"));  
        colmail.setCellValueFactory(new PropertyValueFactory<>("email"));  
        coltel.setCellValueFactory(new PropertyValueFactory<>("telephone"));  
        table_detail.setItems(detail);  
        paneDetail.setVisible(true);  
    }  
    @Override  
    public void onFailure(Call<List<GroupeData>> call, Throwable t) {  
        paneDetail.setVisible(false);  
        Platform.runLater(() -> {  
            Alert alert = new Alert(Alert.AlertType.WARNING);  
            alert.setContentText(t.getMessage());  
            alert.showAndWait();  
        });  
    }  
});
```

## II.2.6. Gérer les codes d'erreur :

Le WebService fournit souvent un code HTTP en cas d'erreur de requête, par exemple 404 Not found. Cette erreur est interceptée avec :

```
if (!response.isSuccessful()) {
    paneDetail.setVisible(false);
    Platform.runLater(() -> {
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setHeaderText(null);
        alert.setContentText("aucune données trouvées pour le groupe ");
        alert.showAndWait();
    });
    return;
}
```

Ici, je crée un message d'alerte que j'affiche à l'utilisateur et je rends l'affichage du résultat invisible.

Maintenant qu'on a appris à utiliser retrofit et a créé notre interface et nos objets, affichant nos résultats dans nos vues.

## II.5.2 Liste des offres :

Pour afficher la liste des offres dans un tableau j'ai utilisé la TableView de JavaFX *public TableView<GlobalData> list\_offre;*

Et puis on affiche ligne par ligne en récupérant la réponse retrofit dans une observablelist :

```
ObservableList<GlobalData> observableList = FXCollections.observableArrayList();
```

Il ne reste plus qu'à attribuer les valeurs au bonne colonnes de la TableView (list\_offre) avec

```
observableList = FXCollections.observableArrayList(response.body());
col_numero.setCellValueFactory(new PropertyValueFactory<>("numeroOffre"));
col_nom.setCellValueFactory(new PropertyValueFactory<>("nomOffre"));
col_nombre_stagiaire.setCellValueFactory(new PropertyValueFactory<>("nbStagiaire"));
col_date_debut.setCellValueFactory(new PropertyValueFactory<>("dateDebut"));
col_date_fin.setCellValueFactory(new PropertyValueFactory<>("dateFin"));
col_id_offre.setCellValueFactory(new PropertyValueFactory<>("idOffre"));
list_offre.setItems(observableList);
```

Numéro de l'offre	Nom	Nombre de stagiaires	date de début	date fin	identifiant
20054	DWWM	6	2020-09-28	2021-05-28	69
20052	CDA	6	2020-08-31	2021-06-04	66
20053	DWWM	6	2020-08-31	2021-04-30	68
20050	DWWM	6	2020-07-20	2021-02-12	65
20049	CDA	6	2020-07-20	2021-04-09	64
20051	TB	15	2020-07-13	2020-09-04	67
20071	DWWM Hélène BOUT...	1	2020-07-06	2021-03-26	70
20048	TB	15	2020-06-15	2020-08-07	60
20047	TB	15	2020-05-18	2020-07-17	57
20046	CDA	12	2020-05-11	2021-01-22	61

Figure 7 Affichage de la liste des offres

### II.2.5.3 Le détail de l'offre :

Une fois le tableau des offre affiché, l'utilisateur sélectionne une offre dans le tableau qui affiche la liste des offres on récupère le numéro de l'offre pour la ligne sélectionnée

```
//on recuperer l'id et on stock
int num = Integer.parseInt(list_offre.getSelectionModel().getSelectedItem().getNumeroOffre());
```

Et en l'envoie en paramètre pour la requête retrofit

```
//on instancie retrofit
NcodeAPI ncodeAPI = RetrofitInstance.getRetrofitInstance().create(NcodeAPI.class);
Call<List<GroupeData>> call = ncodeAPI.getGroupeData(num);
```

Ça nous donne La requête pour avoir uniquement l'offre qui nous intéresse

```
@GET("offre/readOne.php/")
Call<List<GroupeData>> getGroupeData(@Query("numero") int numero);
```

Le liens http générer ressemble à :

[https://localhost/API\\_Ncode/offre/readOne.php?numero=19123](https://localhost/API_Ncode/offre/readOne.php?numero=19123)

On affiche le résultat récupéré (la réponse) dans notre pane (paneDetail)

```
detail = FXCollections.observableArrayList(response.body());
text_nom_group.setText(response.body().get(0).getNomOffre());
text_num_group.setText(response.body().get(0).getNumero());
colnom.setCellValueFactory(new PropertyValueFactory<>("nom"));
colprenom.setCellValueFactory(new PropertyValueFactory<>("prenom"));
colmail.setCellValueFactory(new PropertyValueFactory<>("email"));
coltel.setCellValueFactory(new PropertyValueFactory<>("telephone"));
table_detail.setItems(detail);
paneDetail.setVisible(true);
```

La recherche de groupe ou d'offre quant à elle suit le même chemin et appelle la même requête et la même class objet GroupeData, sauf que cette fois on contrôle la saisie utilisateur avec des expressions régulières puis on parse la valeur récupérée en entier.

```
//controle de saisie utilisateur (regex)
Pattern r_num = Pattern.compile("[0-9]{5}");// ou "\d"
String num = this.text_numero.getText();
//on recupere le numero offre saisi on l'envoie en get
if(r_num.matcher(num).matches()){
    int numero = Integer.parseInt(text_numero.getText());...
```

The screenshot shows a web-based application interface. At the top, there are two tabs: "Recherche" (highlighted in green) and "Liste Offre". Below the tabs, there is a search bar labeled "Rechercher les groupes de la section informatique" and a button "Afficher le détail du groupe". The main content area displays a table of offers with columns: Numéro de l'offre, Nom, Nombre de stagiaires, date de début, date fin, and a small icon. A scroll bar is visible on the right side of the table. At the bottom of the table, there are filters: "Numéro du groupe" (set to 19094), "Type offre" (set to TB), and a "Rechercher" button. Below the table, there is another table showing details for each offer, with columns: Nom, Prénom, and Email. The data includes names like Stevens, Axil, Alexis, Nicolas, Mouhamadou, Catherine, etc., along with their respective emails.

Figure 8 Détail des offres

The screenshot shows a search interface with two search fields: "Rechercher de stagiaire : ben" and "Rechercher de groupe : 19123", each with a "GO" button next to it. To the right of these fields is the AFPA logo. Below the search fields, there are three input fields: "Numéro du groupe : 19123", "Type offre : CDA", and "Nombre de stagiaires : 4". The main area displays a table with columns: Nom, Prénom, Email, and Téléphone. The data includes Sagez (Jason), Afriad (Ghizlane), Benchickh (hamid), and Ory (Tristan).

Figure 9 Recherche par numéro d'offre

## Recherche par nom ou le prénom :

Comme détaillé au chapitre de l'API, on va passer en paramètre le nom ou le prénom du stagiaire pour effectuer une recherche.

L'utilisateur va saisir le nom ou le prénom dans un champ de texte. La saisie est contrôlée, si elle correspond aux critères de recherche elle sera envoyée en paramètre de la requête retrofit comme avec le numéro du groupe expliqué précédemment.

On appellera la fonction rechercheNom avec l'évènement Click :

```
public void rechercheNom(ActionEvent actionEvent) {...}
```

On contrôle la saisie de l'utilisateur avant de la récupérer :

```
//contrôle de saisie utilisateur (regex)
Pattern reg_nom = Pattern.compile("^[^@+=|><\\]\\\\[{}]+$");
String nom = this.text_nom.getText();
```

Si la saisie dans le champ text\_nom est correcte :

```
if(reg_nom.matcher(nom).matches()){
    String nom2 = text_nom.getText();
```

On stock la valeur et on l'envoie pour notre requête retrofit.

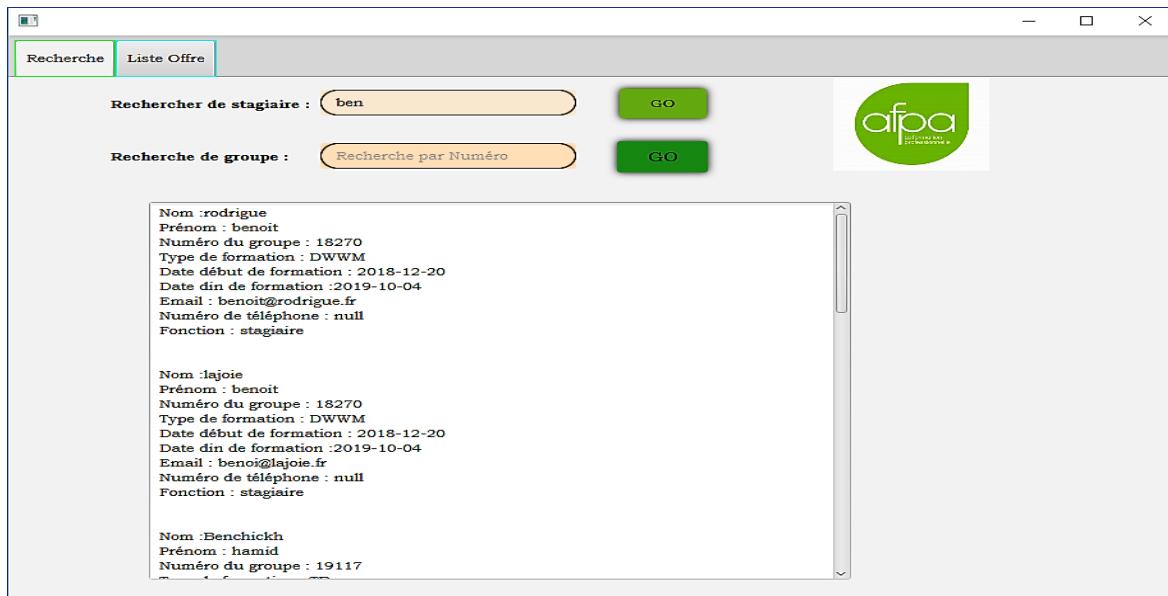
```
NcodeAPI ncodeAPI = RetrofitInstance.getRetrofitInstance().create(NcodeAPI.class);
Call<List<UserData>> call = ncodeAPI.getUserDataNom(nom2);
```

@GET("user/userNom.php")Call<List<UserData>>getUserDataNom(@Query("nom") String nom)); ce qui nous donnera un exemple de lien comme suit:

[https://dev.amorce.org/APIcode/API\\_Ncode/user/userNom.php?nom=jas](https://dev.amorce.org/APIcode/API_Ncode/user/userNom.php?nom=jas)

Le résultat est un tableau associatif (clé valeur) afficher dans un textArea

```
List<UserData> posts = response.body();
for (UserData post : posts) {
    String content = "";
    content += "Nom :" + post.getNom() + "\n";
    content += "Prénom : " + post.getPrenom() + "\n";
    content += "Numéro du groupe : " + post.getNumeroOffre() + "\n";
    content += "Type de formation : " + post.getNomOffre() + "\n";
    content += "Date début de formation : " + post.getDateDebut() + "\n";
    content += "Date fin de formation : " + post.getDateFin() + "\n";
    content += "Email : " + post.getEmail() + "\n";
    content += "Numéro de téléphone : " + post.getTelephone() + "\n";
    content += "Fonction : " + post.getRole() + "\n\n";
    text_result.appendText(content);
}
```



*Figure 10 recherche la nom, prénom*

### **II.3. L’application Android :**

L’application Android est née de l’idée d’avoir accès au planning de formation NCode pour le personnel administratif de l’AFPA, puis des améliorations sont apportées au fil du temps comme de créer une vue connexion (login) et puis faire un onglet de recherche de stagiaire.

L’application est directement installée sur les téléphones du personnel concerné sans avoir à passer par l’étape de distribution ou publication.

Mon API REST réalisé pour l’application desktop API\_ncode va encore nous servir pour notre application mobile. Je travaillerai sur Android Studio, j’ai utilisé la bibliothèque Retrofit2.

#### **II.3.1 Mise en place :**

##### **Création du projet :**

On va développer une application mobile sur la plateforme Android. Nous pouvons développer de l’Android dans plusieurs langages (Java, Kotlin, Dart via Flutter...), j’ai choisi Java comme langage et une compatibilité de notre applications avec le parc des mobiles (API 24 :Android 7.0(Nougat) , compatible avec 73,7% du parc.

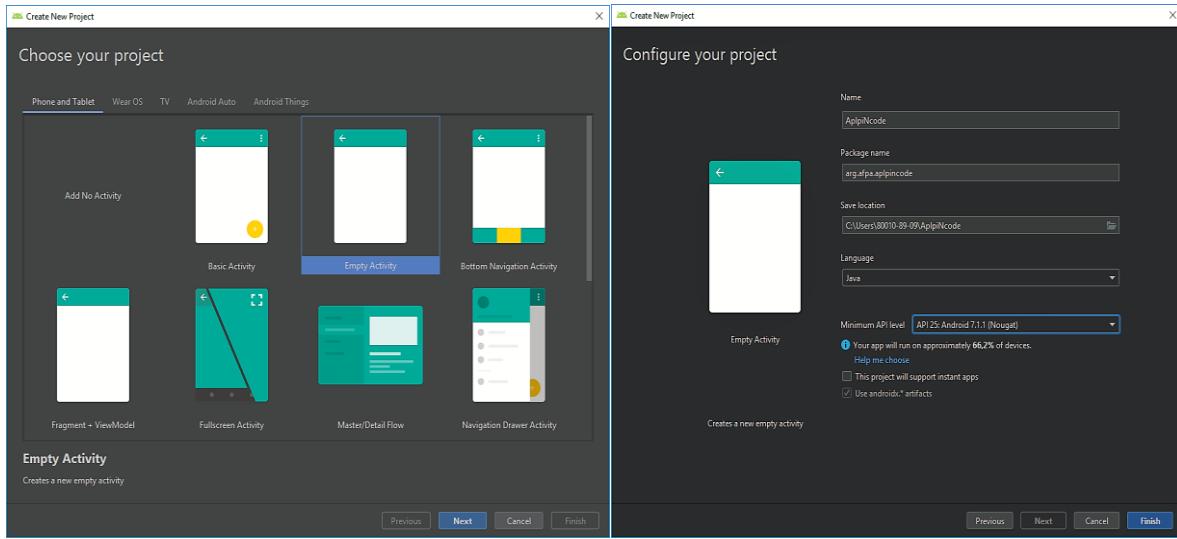
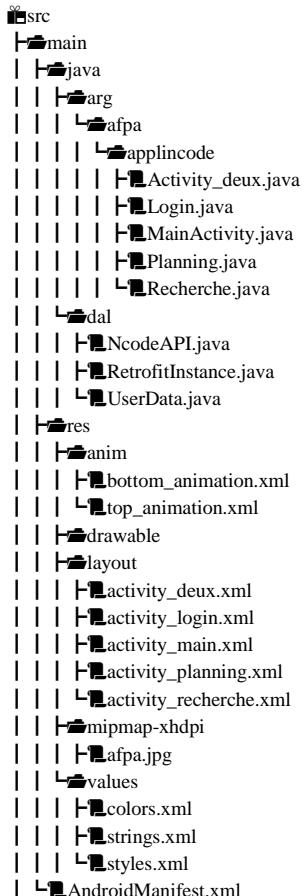


Figure 11 création de nouveau projet Android



## Organisation du projet

Dans le fichier `build.gradle` (`Module: app`), On ajoute les dépendances :

```

dependencies {
    ...
    implementation 'com.squareup.retrofit2:retrofit:2.4.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
    ...
}

```

Dans le manifest on ajoute les permissions d'accès à internet, pour accéder à internet. Le fichier AndroidManifest.xml déclare l'ensemble des éléments de l'application.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Dans la dernière version de Google Play, Google supprimait le besoin de demander la permission pour Internet car "la plupart des applications en ont besoin de toute façon". Cependant, pour les utilisateurs ayant des versions plus anciennes, il est toujours recommandé de laisser le code ci-dessus dans notre manifeste.



Refaire l'icône de l'appli : Manifest.xml  
Pour faire une jolie icone bien parlante

```
        android:roundIcon="@mipmap/afpa"
```

Un moyen de revenir en arrière en allant sur le planning et l'onglet recherche de stagiaires.

```
<activity
    android:name=".Planning"
    android:label="Planning"
    android:parentActivityName=".Activity_deux" />
<activity
    android:name=".Recherche"
    android:label="Recherche"
    android:parentActivityName=".Activity_deux" />
<activity android:name=".Activity_deux" />
<activity android:name=".MainActivity">
```

On indique par quelle activity démarré l'application à l'exécution.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

### II.3.2 Création et animation de la fenêtre d'accueil Le Splash Screen :

Au démarrage d'une application, il est souvent agréable ou plus visuel d'avoir une animation. D'où mon choix de démarrer avec le SPLASH Screen.

onCreate() / onDestroy() : permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans onCreate() afin d'afficher l'activité le plus rapidement possible.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

```

Layaout :activity\_main notre vue contient un TextView et une ImageView et un background de la carte de France région Picardie en vert. Le logo et le text de bienvenue se glisse et la page animation reste sur écran d'accueil 5sec.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/afpa_carte"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="348dp"
        android:layout_height="57dp"
        android:text="Bonjour et bienvenu sur l'application Ncode de l'AFPA"
        android:textAlignment="center"
        android:textColor="#010101"
        android:textSize="20dp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.403" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="160dp"
        android:layout_height="154dp"
        android:layout_marginStart="40dp"
        android:layout_marginTop="396dp"
        android:layout_marginEnd="40dp"
        android:src="@drawable/afpa"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Mais tout d'abord on retire la barre d'informations, dans styles.xml

```

<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">

```

Et dans le mainActivity pour que notre animation prend toute la largeur de l'écran :

```

getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.
LayoutParams.FLAG_FULLSCREEN);

```

### II.3.3 Animer le Splash Screen :

On crée un package anim avec deux fichiers top\_animation.xml et bottom\_animation.xml  
Dans le fichier top\_animation (animation qui descend du haut)

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:fromXDelta="0%"
        android:fromYDelta="-100%"
        android:duration="2000"/>
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="1500"/>
</set>
```

La méthode translate prend deux paramètres de départ de l'animation en X et en Y, et une durée. Ici l'élément garde le même X (horizontal) mais va partir à -100% à partir du haut. Cette animation durera 2 seconde.

L'alpha contrôle la transparence. Ici nous partons avec un coefficient de 0,1 pour arriver à 1 (visible) en 1,5 secondes.

On fait de même pour le fichier bottom\_animation

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:fromXDelta="0%"
        android:fromYDelta="100%"
        android:duration="1500"/>
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="1500"/>
</set>
```

On va gérer tout ça dans notre MainActivity (contrôleur) : on déclare nos animations puis nous ouvrons notre application sur la page login dans une durée de 6 seconde

```
private static int SPLASH_SCREEN = 6000;// durée 6sec
Animation topAnim, bottomAnim;
ImageView logo;
TextView desc;

//Animations
topAnim = AnimationUtils.loadAnimation(this,R.anim.top_animation);
bottomAnim = AnimationUtils.loadAnimation(this,R.anim.bottom_animation);
```

```
// ensuite les éléments à animer:  
logo = findViewById(R.id.imageView);  
desc = findViewById(R.id.textView);  
//Assignment des animation  
logo.setAnimation(topAnim);  
desc.setAnimation(bottomAnim);  
  
new Handler().postDelayed(new Runnable() {  
    @Override  
    public void run() {  
        Intent intent = new Intent(MainActivity.this,Login.class);  
        startActivity(intent);  
        finish();  
    }  
},SPLASH_SCREEN);  
}
```



Figure 12 Animation entrée application

### II.3.4 Authentification :

Sur cette partie j'aurai pu travailler avec Retrofit2 ou volley pour aller chercher les valeurs de la base de données et les comparer aux valeurs entrées par l'utilisateur, mais puisque que le personnel pour qui l'application est destinée ne sont pas enregistré sur la base ; donc j'ai préféré créer des codes d'accès enregistrés en interne et leurs donner les codes d'accès.

**Layout :** notre layout login contient : un TextView identification, deux EditText pour la saisie du mot de passe et de l'identifiant, un autre TextView pour afficher un message d'erreur en cas de saisie erronée et un Button "GO" avec l'action onClick qui déclenche l'action de vérification et de redirection.

Une image en background faite avec Figma est mise en arrière-plan pour avoir un design plus correct et pouvoir reprendre les couleurs de l'AFPA plus facilement.

Un quantifieur d'erreur est ajouté pour désactiver le bouton onClick si on dépasse 4 essais.

#### Contrôleur Login:

Comme précisé précédemment, j'ai mis un identifiant et un mot de passe en local

```
public class Login extends AppCompatActivity {
    //compteur pour le nombre de saisie erronée
    int count =4;
    //textview pour afficher un message d'erreur
    TextView text_erreur;
    Button btn_login;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
    }
    public void login(View view) {
        EditText username = (EditText)findViewById(R.id.username);
        EditText password = (EditText)findViewById(R.id.password);
        text_erreur = (TextView)findViewById(R.id.text_erreur);
        btn_login = (Button)findViewById(R.id.btn_login);

        if (username.getText().toString().equals("afpa") && password.getText().toString().equals("APIcode")){
            Toast.makeText(this, "Vous êtes bien authentifié", Toast.LENGTH_LONG).show();
            Intent intent = new Intent(this, Activity_deux.class);
            startActivity(intent);
        }
        else
        {
            Toast.makeText(this, "Login ou mot de passe incorrect", Toast.LENGTH_LONG).show();
            //rendre le text-erreur visible
        }
    }
}
```



```

text_erreur.setVisibility(View.VISIBLE);
text_erreur.setTextColor(Color.RED);
count--;/incrementation
text_erreur.setText("il vous reste ".concat(Integer.toString(count)).concat("tentatives"));
    //désactiver de bouton si tentative épuisée
if (count == 0) { btn_login.setEnabled(false); }}}
```

Explications :

On déclare nos composants d'interface et initialiser notre compteur d'erreur.

Si mot de passe et identifiant saisie correspondent on redirige vers la prochaine activité (activity\_deux) ; on récupère les valeurs saisie avec getText().toString() parce que dans un editText on peut pas récupérer directement avec getText().

le Toast.makeText(this, "Vous êtes bien authentifié", Toast.LENGTH\_LONG).show(); c'est pour afficher un message pour informer l'utilisateur qui lui disparaît au bout d'un moment.

Si saisie erronée le textView text\_erreur est rendu visible contenant un message de couleur rouge avec le setTextColor(color.RED); le quantifieur d'erreur est décrémenté avec count; au bout de 4 erreur le bouton "GO" est désactivé.

```
if (count == 0) { btn_login.setEnabled(false); }
```

```

//rendre le text-erreur visible
text_erreur.setVisibility(View.VISIBLE);
text_erreur.setTextColor(Color.RED);
count--;/incrementation
text_erreur.setText("il vous reste ".concat(Integer.toString(count)).concat(" tentatives"))
;
```

Tout en affichant un message dans notre text\_erreur en le rendant visible en cas d'erreur en lui donnant le nombre de tentatives restantes.

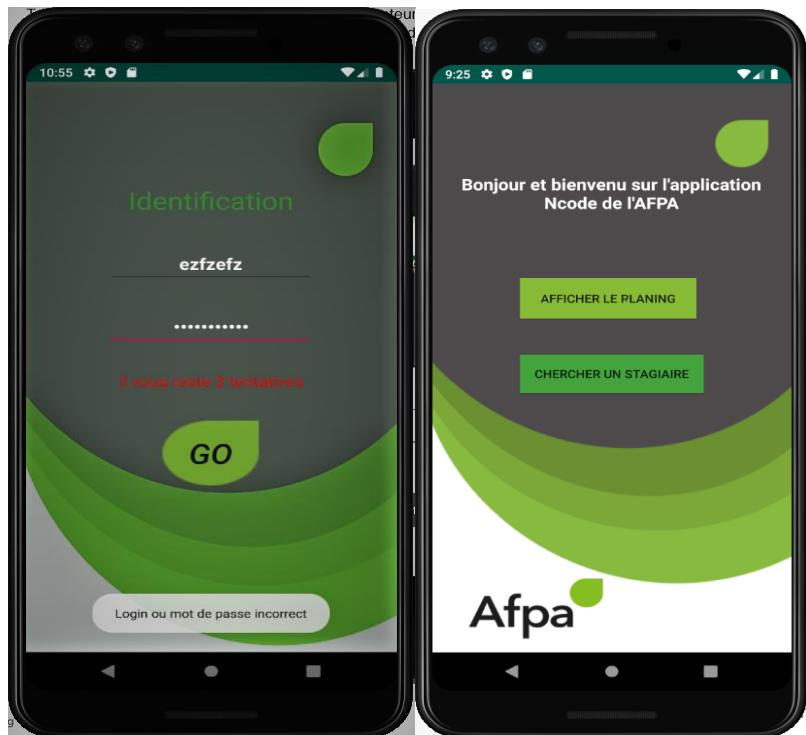


Figure 13 échec et succès d'authentification

L'authentification réussie nous envoie à la prochaine activity que j'ai appelé activity\_deux qui contient deux boutons qui redirigent vers l'affichage du planning et la page de recherche de stagiaires je passe rapidement sur cette partie vu que j'ai déjà détaillé les redirections avant avec intent.

```
public class Activity_deux extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_deux);
    }
    public void lanchRecherche(View view) {
        Intent intent1 = new Intent(this, Recherche.class);
        startActivity(intent1);
    }
    public void lanchPlanning(View view) {
        Intent intent = new Intent(this, Planning.class);
        startActivity(intent);
    }
}
```

### II.3.5 Le planning :

Cette partie est demandée par la direction et vu que ce n'était pas réalisable en JavaFX en desktop je l'ai fait en Android. Dans le layout j'ai juste déclaré une WebView qui prend la largeur de la fenêtre.

Le but de la WebView est d'afficher le planning sans passer par l'étape d'authentification à NCode. Cette action est possible en passant un header et une clé de connexion avec Android chose qui n'était pas possible avec JavaFX.

Nous, au préalable, déclaré la permission d'accès à internet dans le manifest ;

Dans le contrôleur on déclare notre WebView et l'URL à charger et une valeur fixe de notre URL :

```
private WebView webView;
final String url = "https://ncode.amorce.org/offre/planning//";
```

Premièrement : on crée une méthode **getHeaders()** qui sera le retour de nos entêtes que l'on souhaite ajoutées, à savoir le header et la clé.

```
// d'abord créer une méthode, qui sera le retour de vos en-
têtes que vous souhaitez ajouter à la demande:
private Map<String, String> getHeaders()
{
    Map<String, String> headers = new HashMap<>();
    headers.put("AFPA-TOKEN-API-
AUTH", "pcS50ha7RIOUAM3vM3P1Lib00cZm090ua4qi8hNVtAL0cpMapWGSgAXZEYHTJ8ZZLxtJl070DoSu10nC");
    return headers;
}
```

Deuxièmement : on crée une notre WebClient() qu'on ajoute à notre WebView.

```

//Deuxièmement, vous devez créer WebViewClient:
private WebViewClient getWebViewClient()
{
    return new WebViewClient()
    {
        @Override
        @TargetAPI(Build.VERSION_CODES.LOLLIPOP)
        public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request)
        {
            view.loadUrl(request.getUrl().toString(), getHeaders());
            return true;
        }
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url)
        {
            view.loadUrl(url, getHeaders());
            return true;
        }
    };
}

```

Il ne nous reste plus qu'à appeler tout ça dans notre méthode onCreate pour charger l'URL avec les headers.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_planning);
    webView = (WebView) findViewById(R.id.webview);
    webView.setWebViewClient(getWebViewClient());
    webView.loadUrl(url, getHeaders());}

```

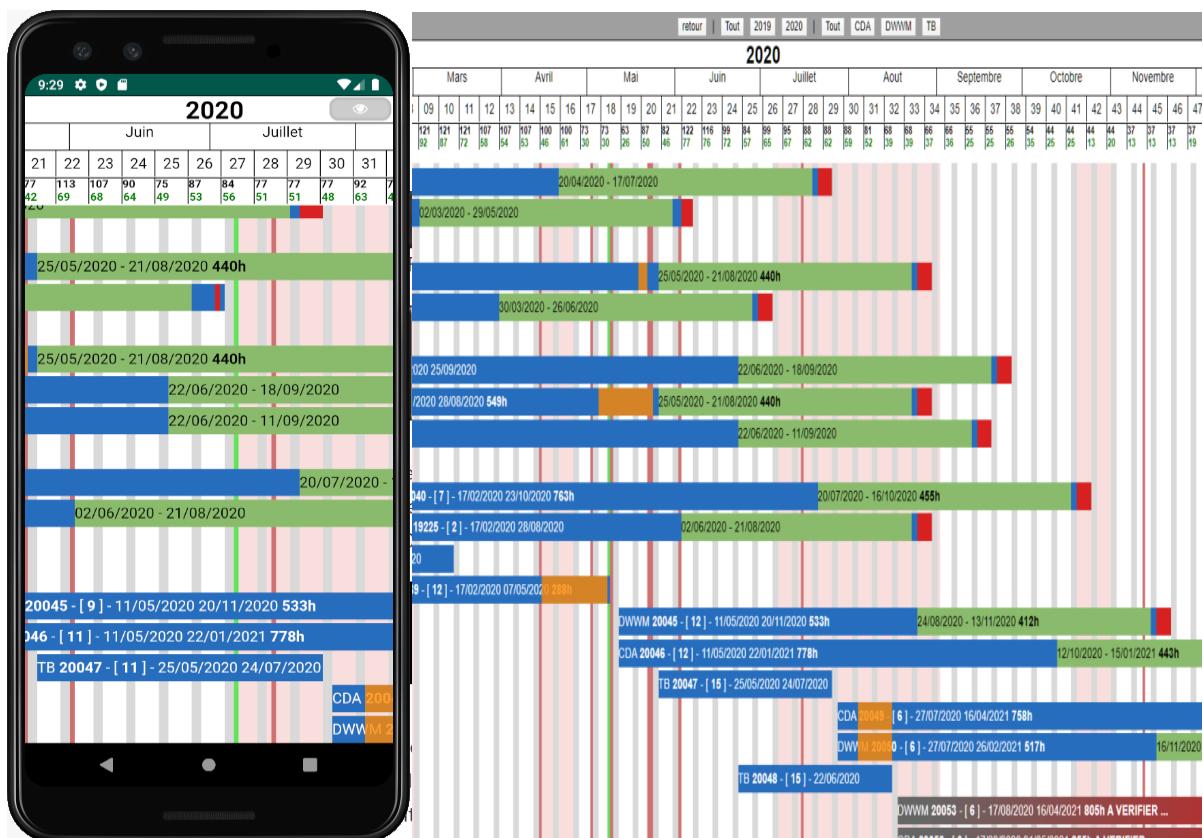


Figure 14 vue sur le planning

### II.3.6 Recherche par nom ou prénom :

Comme avec l'application Desktop j'ai réutilisé Retrofit2 pour la consommation de mon API NCode créée précédemment

Une interface NcodeAPI, une class instance de retrofit et une class objet qui contient les objets récupérés en JSON. Le code ressemble sensiblement à celui JavaFX

Juste la syntaxe et l'appel dans champs qui diffèrent.

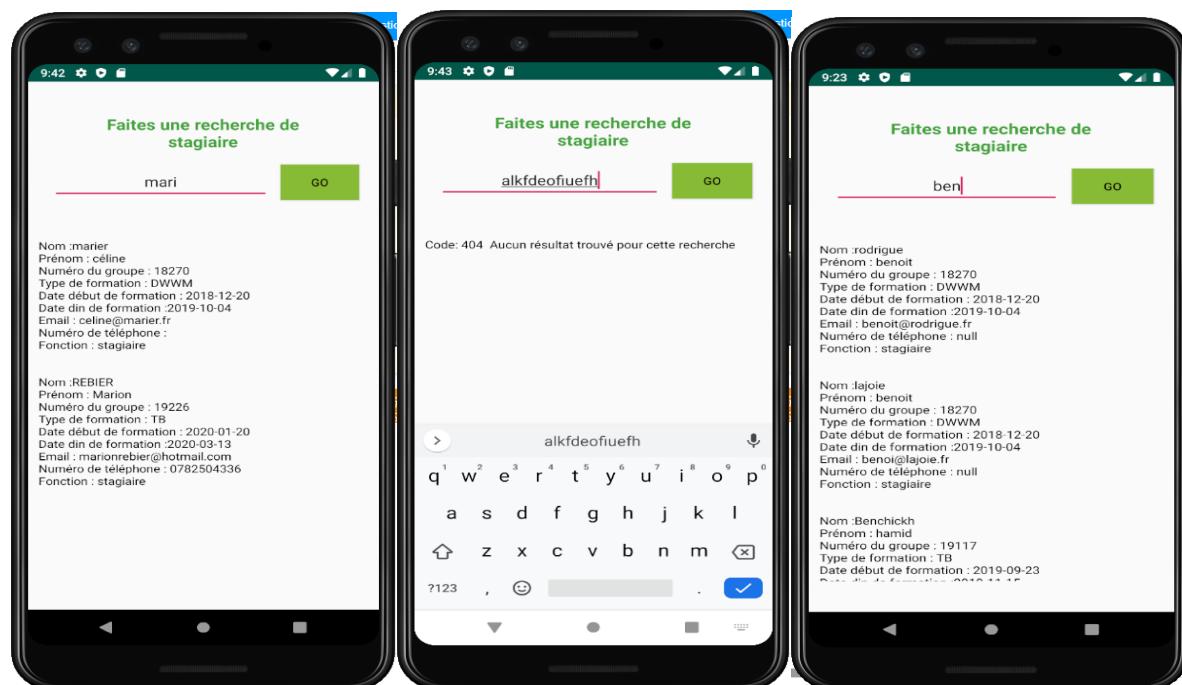


Figure 15 recherche de nom ou prénom

```
public class Recherche extends AppCompatActivity {

    private TextView textResult;
    private EditText text_nom;
    private String nom;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recherche);
    }
    public void rechercheNom(View view) {
        text_nom = findViewById(R.id.text_nom);
        textResult = findViewById(R.id.textResult);
        textResult.setText(""); //pour vider la recherche précédente
        nom = String.valueOf(text_nom.getText());
        //dans un editetext on peut pas recuperer la valeur du string d'où String.valueOf
        //appeler l'instance retrofit
        NcodeAPI ncodeAPI = RetrofitInstance.getRetrofitInstance().create(NcodeAPI.class);

        Call<List<UserData>> call = ncodeAPI.getUserDataNom(nom);

        call.enqueue(new Callback<List<UserData>>() {
            @Override
```

```

public void onResponse(Call<List<UserData>> call, Response<List<UserData>> response) {
    //qlq groupe mal lié dans la base (j'affiche une alete pour l'erreur) ERREUR 404 jason
    if (!response.isSuccessful()) {
        textResult.setText("Code : " + response.code() + " Aucun résultat trouvé pour cette recherche");
        return;
    }

    List<UserData> posts = response.body();
    for (UserData post : posts) {
        String content = "";
        content += "Nom : " + post.getNom() + "\n";
        content += "Prénom : " + post.getPrenom() + "\n";
        content += "Numéro du groupe : " + post.getNumeroOffre() + "\n";
        content += "Type de formation : " + post.getNomOffre() + "\n";
        content += "Date début de formation : " + post.getDateDebut() + "\n";
        content += "Date fin de formation : " + post.getDateFin() + "\n";
        content += "Email : " + post.getEmail() + "\n";
        content += "Numéro de téléphone : " + post.getTelephone() + "\n";
        content += "Fonction : " + post.getRole() + "\n\n\n";
        textResult.append(content);
    }
}
@Override
public void onFailure(Call<List<UserData>> call, Throwable t) {
    textResult.setText(t.getMessage());
}
});
}
}

```

### II.3.7 Le design :

J'ai utilisé quelques photos notamment le logo AFPA, et un background créé avec Figma, mais je me suis vite rendu compte que pour faire des radius en Android fallait que je crée moi-même le design.

Pour se faire, je me suis servi du ShapeDrawable pour créer des formes et des design. ShapeDrawable qui est une sous classe de Drawable je vous mets un exemple dans ce rapport.

Voici le code du rectangle arrondi autour du bouton d'accès :

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <corners
        android:radius="20dp" />
    <solid
        android:color="#87bb34" />
    <stroke
        android:color="#FFFFFF"
        android:width="2dp" />
    <size
        android:width="165dp"
        android:height="40dp" />
</shape>

```

Il suffit de l'appeler par la suite en background à l'endroit où on veut l'utiliser.

## **II.4. Axes d'améliorations**

De belles avancées ont été réalisées et tout est fonctionnelle. Des changements seront apportés à nos applications, le code sera mieux organisé, l'accès à l'API sera sécurisé. L'étape authentification de l'application mobile sera amené elle aussi à disparaître.

D'autres portes d'entrées sont aussi préparées pour notre API pour anticipation d'un futur besoin.

## **Conclusion**

Afin de réaliser cette mission j'ai, en premier lieu, analyser le besoin, discuter avec les différentes parties du projet et identifier les ressources dont j'aurai besoin.

Plusieurs versions proposées, on est arrivé au final à un projet plus pratique et simple d'utilisation, j'ai réalisé donc une API REST qui récupère des données de notre base de données et qui les renvoie en JSON et réaliser deux application Desktop et Mobile pour notre client, fonctionnelles et pratique avec utilisation de Retrofit2 une bibliothèque de consommation d'API.

Pour conclure, j'ai effectué mon stage de fin de formation ici même à l'AFPA. Lors de ce stage de 3 mois, que j'ai effectué en partie en confinement et en télétravail, j'ai pu mettre en pratique beaucoup connaissances théoriques acquises durant ma formation et être confronté aux difficultés du monde du travail et la réalisation de projet et la recherche continue de solution dans le secteur du développement et de la conception de projet informatique.

Ce stage a été un vrai challenge pour moi, d'être confronter à un projet qui démarre de la base, il m'a fallu donc être force de proposition et de créativité, bien sûr je remercie germain mon maître de stage pour tous ces conseils et à François régis aussi d'être toujours aussi présent et je les remercie particulièrement de m'avoir fait confiance et de m'avoir confié cette mission.

Je garde du stage un excellent souvenir, il constitue désormais une expérience professionnelle valorisante et encourageante pour mon avenir.

Je pense que cette expérience m'a offert une bonne préparation à mon insertion professionnelle car elle fut pour moi une expérience enrichissante et complète qui conforte mon désir d'exercer mon futur métier de « concepteur développeur d'applications ».

Enfin, je tiens à exprimer ma satisfaction d'avoir pu travailler dans d'excellentes conditions matérielles et dans un environnement très agréable.