Fakher KARCHID BTS SIO - SLAM

Compte rendu d’activité 2

Evolution de l’application Android pour s’auto-former de Mediatek86

Table des matières

1 Rappel du contexte

InfoTech Services 86 (ITS 86), est une Entreprise de Services Numériques (ESN) spécialisée dans le développement informatique (applications de bureau, web, mobile), l'hébergement de site web, l'infogérance, la gestion de parc informatique et l'ingénierie système et réseau. Elle répond régulièrement à des appels d'offres en tant que société d'infogérance et prestataire de services informatiques.

ITS 86 est une équipe composée de 32 collaborateurs, administratifs, ingénieurs et techniciens dont les activités s'organisent autour de 2 pôles : le pôle développement et le pôle système et réseaux.

La principale activité du pôle Développement consiste à proposer des solutions d'hébergements sur des serveurs dédiés. Les développeurs possèdent également une expertise dans l'intégration de services, le développement logiciel et la gestion/création de bases de données.

Les équipes de développement accompagne les clients dans différentes étapes de leurs projets :

- Analyse de la problématique, rédaction du cahier des charges, assistance à maîtrise d'ouvrage (AMO) ;
- Développement d'applications de bureau en C# et en Java pour répondre à des besoins spécifiques en monoposte ou multipostes ;
- Développement d'applications mobiles (Android, iOS) ;
- Développement d'applications web : HTML, PHP, JavaScript, CSS, Ajax, Symfony, Angular JS. Les applications sont multi-plateformes (Windows, Linux, Mac) et multi-navigateurs (Edge, Mozilla, Firefox, Opera, Chrome, Safari) et répondent aux standards du W3C ;
- Administration de base de données : Intégration et interconnexion avec les systèmes existants, fourniture de l'ensemble des documentations liées à ses applications.

2 Rappel des missions à effectuer

2.1 Mission 1 : gérer le filtre des formations

Etape 1 :

Créer dans le contrôleur une méthode qui retourne la liste ne contenant que les formations dont le titre contient le filtre reçu en paramètre (penser à faire des comparaisons après avoir tout mis en majuscule pour ne pas tenir compte de la casse) ;

Etape 2 :

Créer dans la vue de la liste des formations, la méthode événementielle sur le clic du bouton "filtrer" qui doit vérifier si la zone de saisie est vide ou non pour savoir quelle méthode appeler dans le contrôleur, pour valoriser la liste de formations à utiliser pour l'affichage. Cela suppose que cette liste ne doit plus être valorisée dans la méthode CreerListe.

2.2 Mission 2 : gérer les favoris

Etape 1 :

Ajouter la classe technique pour gérer la BDD locale au format SQLite (récupérer la classe technique déjà utilisée dans les séances).

Etape 2 :

Créer la base de données qui permet de mémoriser les id des formations favorites.

Etape 3 :

Sur le clic d'un cœur gris dans la liste des formations, enregistrer l'id de la formation dans la base locale et réactualiser la liste pour avoir un cœur rouge.

Etape 4 :

Sur le clic d'un cœur rouge dans la liste des formations, supprimer l'id de la formation dans la base locale et réactualiser la liste pour avoir un cœur gris.

Etape 5 :

Dans la page d'accueil, sur le clic de la seconde option (liste des favoris), afficher la liste des formations favorites (donc uniquement les cœurs rouges) ;

Etape 6 :

Dans cette liste, sur le clic d'un cœur rouge, la formation disparait de cette liste. Sur le clic d'une formation, la suite est identique à la version d'origine (affichage de la page du détail d'une formation puis affichage de la vidéo) ;

Etape 7 :

Lors de la récupération des données distantes, contrôler si des formations enregistrées dans les favoris ont été supprimées. Dans ce cas, les supprimer aussi de la base locale.

2.3 Mission 3 : qualité, test et documentation technique

Etape 1 :

Contrôler la qualité du code avec SonarLint.

Etape 2 :

Créer les tests unitaires pour contrôler les conversions des dates.

Etape 3 :

Créer un scénario de tests pour contrôler les différentes manipulations.

Etape 4 :

Générer la documentation technique de l'application complète après avoir contrôlé l'insertion des commentaires normalisés.

3 Présentation des outils utilisés

IDE : Android studio 2021.1.1 Patch 1

SDK version : 32

SGBD : MariaDB 10.2

Lien BDD : API REST

Test unitaire : Junit

Qualité de code : SonarLint

Outils de versionning : Github

Logiciel pour générer la documentation : JavaDoc

Logiciel de suivi : Trello

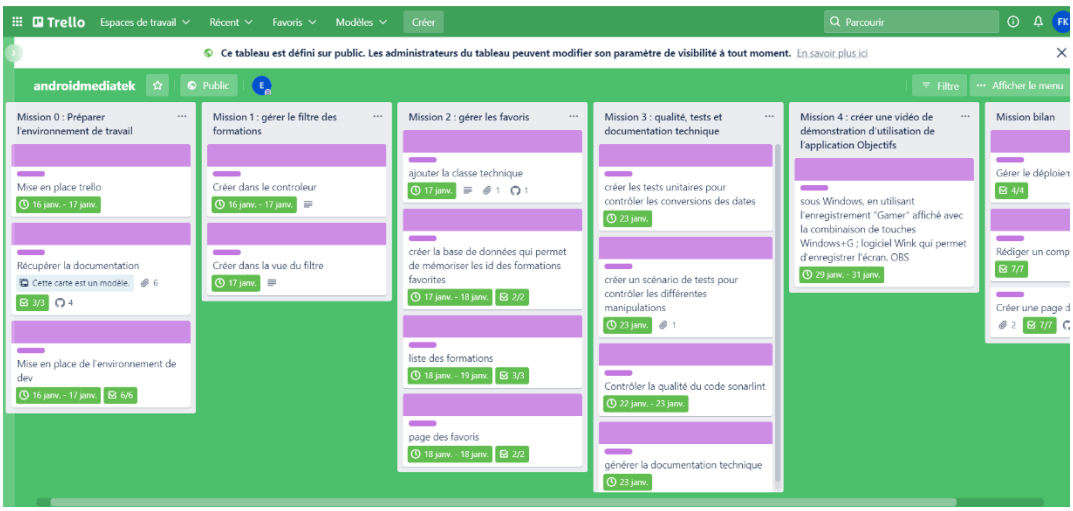
4 Mise en place du suivi avec Trello

Explication du travail réalisé :

La mise en place du suivi de projet a commencé par la création d'un tableau publique sur Trello du nom de . Ainsi, il est possible de constater qu'il comporte 6 listes dont les titres vont de la mission dite 0 à la mission 5 qui est aussi la mission de la mise en place du bilan. Chacune de ces listes contiennent des cartes qui correspondent à différentes étapes des missions à effectuer. Une fonction d'étiquette de couleur permet d'avoir un suivi visuel de ce qui est « terminé » en violet, « en cours » en orange et « en attente » en rouge. Il est a noté que la couleur violette a été choisi afin de pouvoir être distinguable du vert qui est automatiquement sélectionné pour le respect des dates et l'icône des check-lists remplies.

Enfin, un intervalle de temps en jours a été établi afin d'évaluer le temps de réalisation de chacune des tâches. De même, des check-lists par tache ont été mise en place afin d'obtenir un meilleur suivi. Aussi, afin d'accélérer la mise à jour des étiquettes, un bouton d'automatisation, « maj étiquette » à été mis en place, comme le montre la figure, ainsi en appuyant sur le bouton de la carte, lorsque l'étape est terminée l'étiquette passe de « En cours » à « Fini ». De plus, à la création d'une carte il est possible de la marquer comme carte modèle, ce qui permet de créer plus rapidement de nouvelles cartes.

Bilan du travail réalisé :



5 Réalisation des missions

5.1 Mission 1 : Gérer les filtres des formations

5.1.1 Etape 1 : Méthode pour gérer le filtre textuel

Explication du travail réalisé :

La méthode getLstFiltre attend en paramètre un filtre textuel et parcours la liste des formations pour trouver les correspondances textuelles puis retourne la liste des formations filtrées. La fonction attend en deuxième paramètre une liste car par la suite la fonction sera réutilisée par une autre liste.

Bilan du travail réalisé :

```java
/**
 *Filtre pour Favori et Formations
 * @param filtre
 * @param lesFormations
 * @return
 */
public ArrayList<Formation> getLstFiltres(String filtre, ArrayList<Formation> lesFormations)
{
    ArrayList<Formation> lstFiltres = new ArrayList<>();
    for(Formation uneFormation : lesFormations){
        if(uneFormation.getTitle().toUpperCase().contains(filtre.toUpperCase())){
            lstFiltres.add(uneFormation);
        }
    }
    return lstFiltres;
```

5.1.2 Etape 2 : Adapter la méthode évènementielle pour gérer le filtre

Explication du travail réalisé :

La méthode évènementielle sur la clique du bouton « filtrer » vérifie si la zone de saisie est vide pour savoir quelle méthode appeler dans le contrôleur, c'est à dire soit récupéré la liste complète soit la liste filtrée pour valoriser la liste de formations à utiliser pour la vue.

Bilan du travail réalisé :

```
/**
 * Procedure evenementielle sur le filtre
 */
private void ecouteFiltre(){
    btnFiltrer.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(txtFiltre.getText().toString() != "") {
                lesFormations = controle.getLesFormationFiltre(txtFiltre.getText().toString());
            }
            else{
                lesFormations = controle.getLesFormations();
            }
            creerListe();
        }

    });

}
```

5.2 Mission 2 : gérer les favoris

5.2.1 Etape 1 : Ajout de la classe MySQLiteOpenHelper

Explication du travail réalisé :

Cette classe permet la création d'une base de données locale MySQLite. Elle possède trois méthodes, dont son constructeur qui appelle la méthode onCreate() s'il ne trouve pas la base de donnée placée en paramètre name. Les deux autres paramètres sont le contexte de l'activity qui l'appelle et le numéro de version. Enfin, le mot clé « final » a été rajouté sur chacun des propriétés de la classe puisqu'ils restent immuables tout le long de l'activité de l'application.

Bilan du travail réalisé :

```
/**
 * Construction de l'accès à une base de données locale.
 *
 * @param context Context
 * @param name    String
 * @param version int
 */
public MySQLiteOpenHelper(final Context context, final String name, final int version) {
    super(context, name,  factory: null, version);
}
```

5.2.2 Etape 2 : Création de la BDD locale

Explication du travail réalisé :

Dans la classe MySQLiteOpenHelper j'ai ajouté la requête qui permet de créer la table favori avec comme unique champ l'identifiant de la formation (**figure 1**).

Ensuite il faut configurer la méthode qui nous permet de récupérer la liste des favoris. Pour cela on initialise un curseur et on boucle sur la méthode moveToNext() pour récuperer l'identifiant des formations favorites qu'on ajoute progressivement dans la liste des favoris , une fois qu'il n'y a plus de ligne , moveToNext() retourne « false » et tout les favoris ont été récupérés(**figure 2**) .

Enfin il faut créer les méthodes qui nous permettent d'ajouter et de supprimer une formation : dans le premier cas on appelle la fonction insert du curseur et dans le deuxième cas on appelle la fonction Delete (**figure 3 et 4**).

Il faut noter que la fonction getWritableDatabase() permet d'autoriser l'écriture dans la base de données.

Bilan du travail réalisé :

```
// propriété de création d'une table dans la base de données
private String creation="create table favori ( idformation INTEGER NOT NULL)";
```

Figure 1

```
/**
 * retourne une liste idformation de favori et met à jour la liste des favoris
 * @return dernier profil
 */
public void recupFavori(){
    ArrayList<Integer> listeFavori = new ArrayList<~>();
    bd = accesBD.getReadableDatabase();
    String req = "select * from favori";
    Cursor curseur = bd.rawQuery(req, selectionArgs: null);
    while(curseur.moveToNext()){
        Integer idformation = curseur.getInt( i: 0);
        listeFavori.add(idformation);
    }
    curseur.close();
    controle.setLesFavoris(listeFavori);
}
```

Figure 2

```
/**
 * ajout d'un formation dans la BDD favori
 * @param idformation
 */
public void ajout(Integer idformation){

    bd = accesBD.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("idformation", idformation);
    bd.insert( table: "favori", nullColumnHack: null, values);
    bd.close();
}
```

Figure 3

```
public void remove(Integer idformation){

    bd = accesBD.getWritableDatabase();
    bd.delete( table: "Favori",  whereClause: "idformation=?", new String[]{idformation.toString()});
}
```

Figure 4

5.2.3 Etape 3 & 4 : Ajout et suppression d'un favori

Explication du travail réalisé :

Premièrement il faut récupérer la liste des identifiants des formations favorites et faire le lien avec la liste des formations (**figure 1**).

La liste des favoris va ensuite être mise à jour lors du clique de l'utilisateur sur un cœur rouge ou sur un cœur gris. Pour cela les procédures évènementielles appelleront la méthode removeFavori et la méthode addFavori du contrôleur (**figure 2 et 3**). Chacune de ces méthodes mettent à jour la liste des favoris récupérées dans le contrôleur lors du démarrage de l'application et mettent à jour la base de données.

Bilan du travail réalisé :

```
public ArrayList<Formation> recupFavoris(){

    if(lstIdFavoris != null) {

        for (Formation uneFormation : lesFormations) {
            for (Integer idformation : lstIdFavoris) {

                if (idformation == uneFormation.getId()) {
                    lstFavoris.add(uneFormation);
                }
            }
        }

    }
    return lstFavoris;
}
```

Figure 1

```
/**
 * Supprime un favori de la liste est de la table
 * @param unFavori
 */

public void removeFavori(Formation unFavori)
{
    accesLocal.remove(unFavori.getId());
    removeLstFavoris(unFavori);

}
```

Figure 2

```java
/**
 * Ajoute un Favori à la table favori et à liste
 * @param unFavori
 */
public void addFavori(Formation unFavori)
{
    accesLocal.ajout(unFavori.getId());
    addLstFavoris(unFavori);

}
```

Figure 3

5.2.4 Etape 3 & 4 : Suite – Ajout des procédures évènementielles.

Explication du travail réalisé :

Premièrement, il faut créer une méthode dans le contrôleur pour savoir si une formation est un favori (**figure 1**).

Puis Il faut ajouter une méthode évènementielle sur le clic d'un cœur pour déterminer s'il faut ajouter ou supprimer une formation des favoris : On ajoute une condition qui permet de savoir si le cœur appartient à une formation favorite. Si c'est le cas la formation est supprimée des favoris et le cœur devient gris, sinon le cœur devient rouge et la formation est ajouté aux favoris (**figure 2**).

La fonction notifyDataSetChanged() est nécessaire pour que la vue soit actualisé, elle est appelé à la fin de la procédure évènementielle.

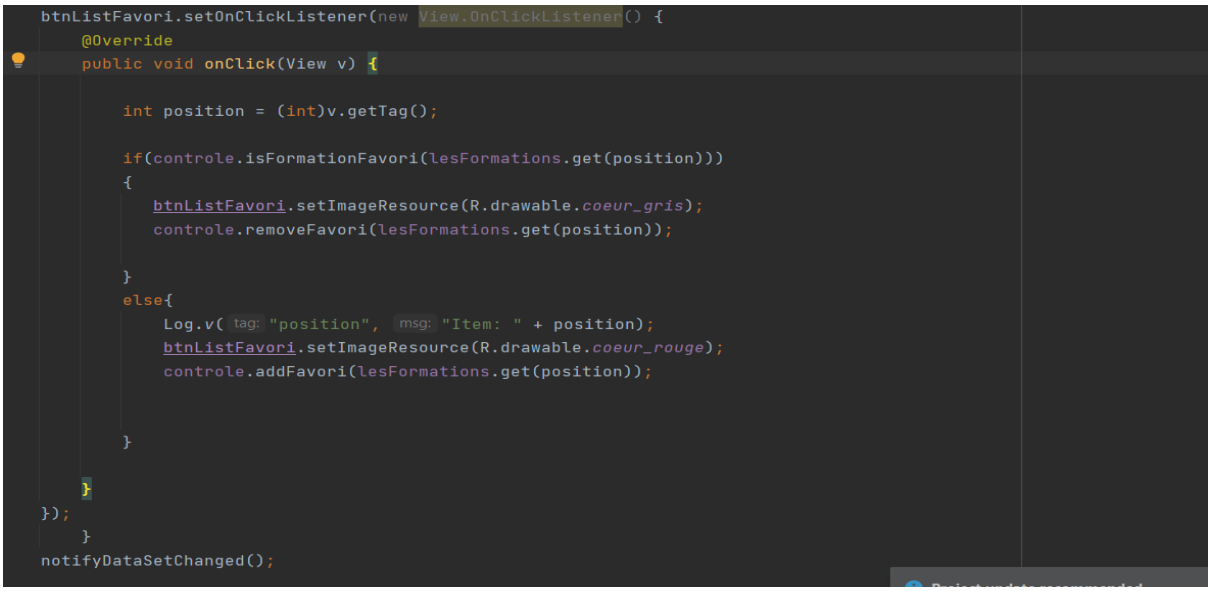
En amont il faut pouvoir identifier l'item pour cela on appelle la méthode setTag() sur celui-ci pour pouvoir faire le lien avec la méthode évènementielle(**figure 3**).

Au moment du premier chargement de la vue il faut pouvoir afficher les favoris, avec un cœur rouge qui ont déjà été stockés dans la base de données (**figure 4**).

Bilan du travail réalisé :

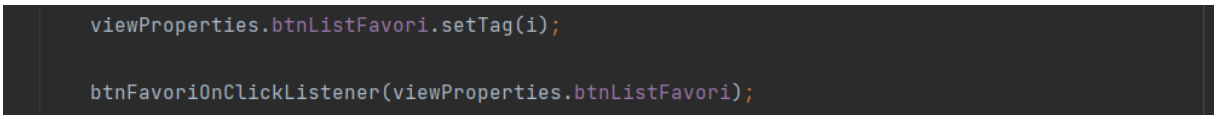
```java
    /**
     * retourne true si une Formation appartient à la table Favori
     * @param uneFormation
     * @return boolean
     */
    public boolean isFormationFavori(Formation uneFormation){
        if(lstFavoris != null)
        {
            for (Formation unFavori : lstFavoris) {
                if (unFavori == uneFormation) {
                    return true;
                }
            }
        }

        // retourne false si idformation n'est pas dans favori ou si favori est vide.
        return false;
    }
```

Figure 1



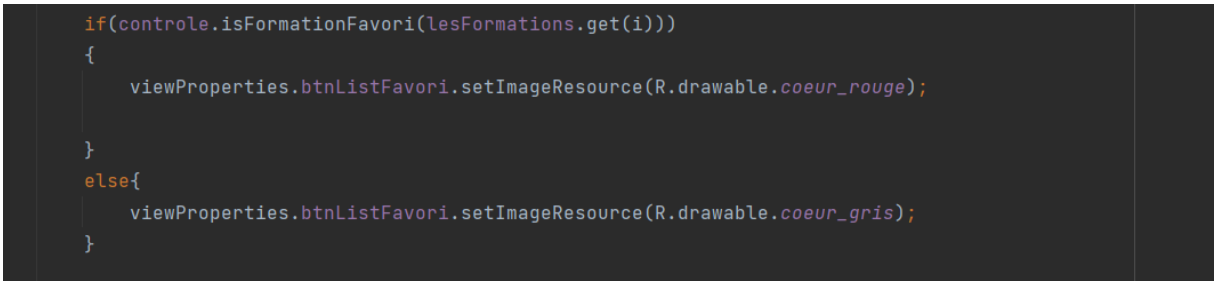
```
btnListFavori.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        int position = (int)v.getTag();

        if(controle.isFormationFavori(lesFormations.get(position)))
        {
           btnListFavori.setImageResource(R.drawable.coeur_gris);
           controle.removeFavori(lesFormations.get(position));

        }
        else{
            Log.v( tag: "position",  msg: "Item: " + position);
            btnListFavori.setImageResource(R.drawable.coeur_rouge);
            controle.addFavori(lesFormations.get(position));


        }

    }
});
    }
notifyDataSetChanged();
```

Figure 2



```
viewProperties.btnListFavori.setTag(i);

btnFavoriOnClickListener(viewProperties.btnListFavori);
```

Figure 3



```
if(controle.isFormationFavori(lesFormations.get(i)))
{
    viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);

}
else{
    viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
}
```

Figure 4

5.2.5 Etape 5 : Ajout d'une nouvelle Activity : listes des favoris

Explication du travail réalisé :

La nouvelle Activity est presque similaire à l'Activity qui affiche la liste des formations, mais ici l'Activity affiche exclusivement la liste des favoris.

L'Activity est donc presque identique mais au lieu de valoriser l'adapter avec la liste des formations, on valorise l'adapter avec la liste des favoris (**figure 1**).

La méthode pour filtrer les formations a été réadapté pour que les favoris puissent être filtrer également dans un souci d'optimisation.

En amont il faut créer un booléen dans le contrôleur pour pouvoir déterminer si nous sommes dans l'Activity qui affiche la liste des favoris ou dans celle qui affiche la liste des formations (**figure 2**).

Les 2 Activity utilisent le même adapter : il faut donc placer une condition dans la méthode évènementielle pour déterminer si l'utilisateur est dans l'Activity qui affiche la liste des favoris ou non et ceci grâce au booléen précédemment créée (**figure 3**).

Le reste est similaire à la première vue, il faut néanmoins rajouter dans le mainactivity un évènement sur le BtnFavori pour pouvoir accéder à cette Activity :

Enfin, il faut ajouter dans la première Activity (MainActivity) un événement sur le bouton Favori(btnFavori) pour pouvoir accéder à cette application (**figure 4**).

Bilan du travail réalisé :

```
// On récupere la liste des favoris pour valoriser l'adapter
lesFormations = controle.getLesFavoris();
controle.setFavoriWindow(true);
```

Figure 1

```
private boolean FavoriWindow = false;
```

Figure 2

```
private void btnFavoriOnClickListener(ImageButton btnListFavori){


if(controle.getFavoriWindow()) {
    btnListFavori.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = (int) v.getTag();
            controle.removeFavori(lesFormations.get(position));
            notifyDataSetChanged();

        }

    });
}else{
btnListFavori.setOnClickListener(new View.OnClickListener() {
```

Figure 3

```
private void creerMenu(){
    ecouteMenu((ImageButton)findViewById(R.id.btnFormations), FormationsActivity.class);
    ecouteMenu((ImageButton)findViewById(R.id.btnFavoris), FavoriActivity.class);
}
```

Figure 4

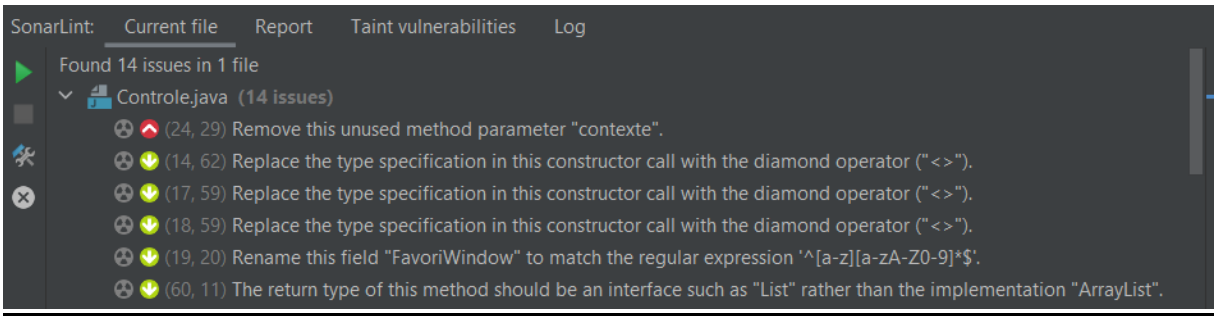
5.3 Mission 3 : qualité, test et documentation technique :

5.3.1 Etape 1 : Contrôle de la qualité du code avec SonarLint

Explication du travail réalisé :

La correction de la qualité du code s'effectue grâce aux informations affichées par l'extension SonarLint.

Bilan du travail réalisé :



5.3.2 Etape 2 : Création des tests unitaires

Explication du travail réalisé :

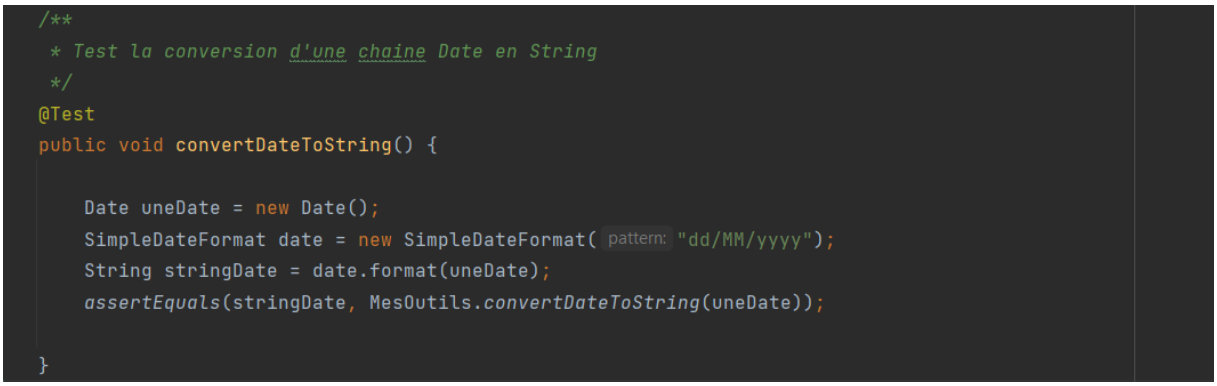
Pour créer un test unitaire de conversion d'une chaine vers une date il faut dans un premier temps convertir une date au format textuel puis la comparer à la date renvoyé par la fonction convertStringToDate() après lui avoir envoyé en paramètre la date au format textuelle (figure 1).

Le deuxième test unitaire conversion d'une date vers une chaine, le principe est similaire mais se fait beaucoup plus rapidement car il n'y a pas besoin de parser (figure 2).

Bilan du travail réalisé :

```
/**
 * Test la conversion d'une chaine Date en Date
 */
@Test
public void convertStringToDate() {
    SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyy-MM-dd hh:mm:ss");
    Date date = null;
    try {
        date = formatter.parse( source: "2020-12-28 22:00:29");
    } catch (ParseException e) {
        e.printStackTrace();
    }
    assertEquals(date, MesOutils.convertStringToDate( uneDate: "2020-12-28 22:00:29",  expectedPattern: "yyyy-MM-dd hh:mm:ss"));

}
```

Figure 1



```java
/**
 * Test la conversion d'une chaine Date en String
 */
@Test
public void convertDateToString() {

    Date uneDate = new Date();
    SimpleDateFormat date = new SimpleDateFormat( pattern: "dd/MM/yyyy");
    String stringDate = date.format(uneDate);
    assertEquals(stringDate, MesOutils.convertDateToString(uneDate));

}
```

Figure 2

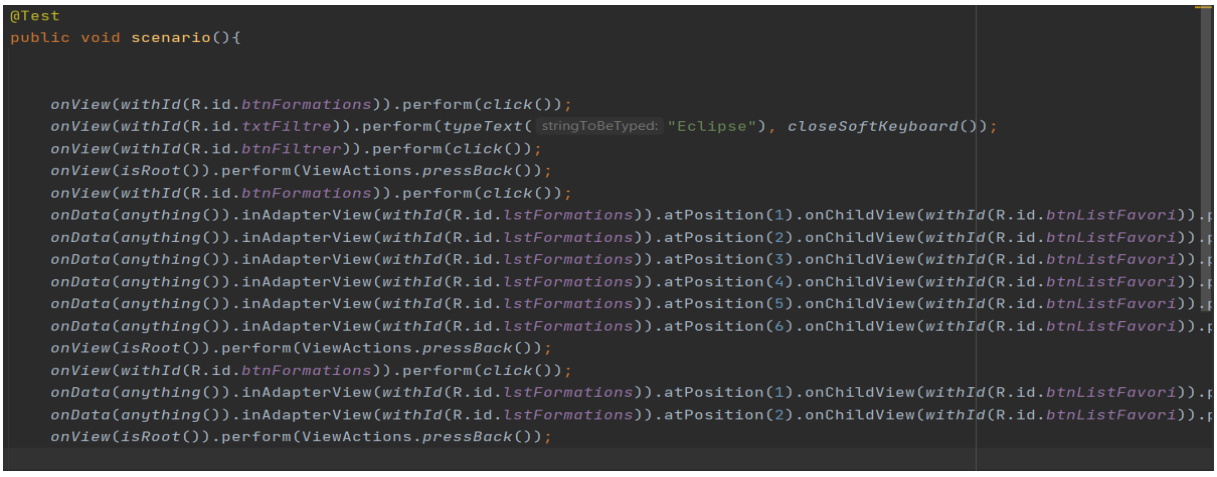
5.3.3 Etape 3 : Créations des scénarios de tests

Explication du travail réalisé :

Pour créer des scénarios de test, on test les différentes possibilités de navigation de l'utilisateur, ici on peut les diviser en 2 branches :

- Le premier scénario ou l'utilisateur clic sur le bouton formation en premier (figure 1)
- Le deuxième scénario ou l'utilisateur clic sur le bouton favori en premier (figure 2)

Bilan du travail réalisé :



```java
@Test
public void scenario(){


    onView(withId(R.id.btnFormations)).perform(click());
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "Eclipse"), closeSoftKeyboard());
    onView(withId(R.id.btnFiltrer)).perform(click());
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFormations)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(2).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(3).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(4).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(5).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(6).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFormations)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(2).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());
```

Figure 1

```java
@Test
public void scenariot(){

    onView(withId(R.id.btnFavoris)).perform(click());
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "Eclipse"), closeSoftKeyboard());
    onView(withId(R.id.btnFiltrer)).perform(click());
    onView(isRoot()).perform(ViewActions.pressBack());
    onView(withId(R.id.btnFavoris)).perform(click());
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).p
    onView(isRoot()).perform(ViewActions.pressBack());
```

Figure 2

5.3.4 Etape 4 : Génération de la documentation technique

Explication du travail réalisé :

Pour générer la documentation technique, il faut au préalable inserer les commentaires normalisées puis ensuite lé générer grâce à JavaDoc

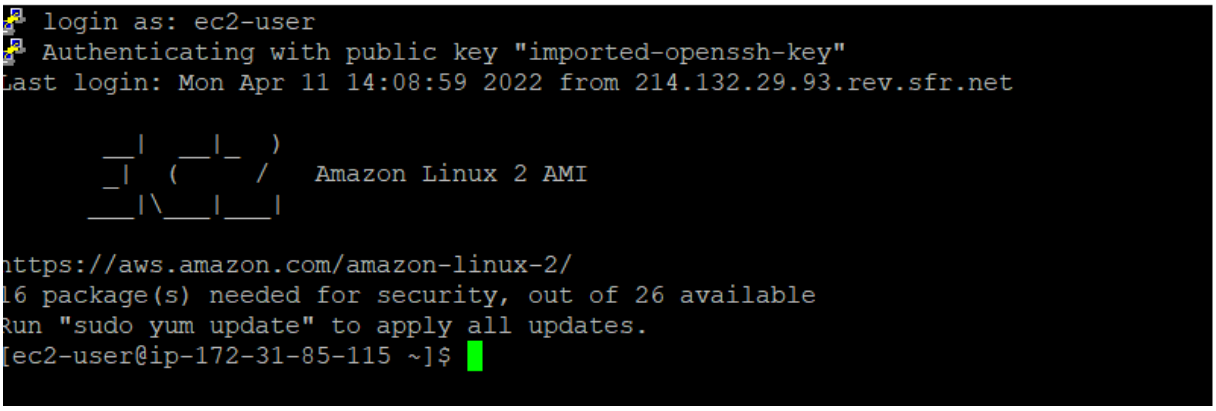
5.3.5 Etape 5 : Déploiement de la base de données et de l'API REST

Explication du travail réalisé :

Le déploiement a été fait sur une machine Ec2 proposé par AWS sous l'OS Amazon-Linux 2, il fallait au préalable installer LAMP sur cette machine. Pour autoriser les connexions les règles entrantes et sortantes ont été modifiés.

De plus il était nécessaire de reconfigurer les paramètres du serveur Apache afin d'autoriser les connexions entrantes.

Bilan du travail réalisé :



6 Bilan

Dans l'Activity qui affiche la liste des formations il est désormais possible d'ajouter ou de supprimer un favori, dans la vue cela se traduit par la présence d'un cœur rouge ou d'un cœur gris.

Par ailleurs une nouvelle Activity permet de consulter exclusivement les favoris, seul des cœurs rouges sont affichés au côté des différentes formations et lorsque que l'utilisateur clic sur un cœur rouge la formation est supprimée des favoris.

Dans ces 2 Activity il est désormais possible de filtrer les formations en cliquant sur le bouton correspondant.

7 Listes des compétences couvertes

7.1 B1 : Support et mise à disposition de services informatiques

7.1.1 B1.1

- ✓ Recenser et identifier les ressources numériques
- ✓ Mettre en place et vérifier les niveaux d'habilitation associés à un service
- ✓ Exploiter des référentiels, normes et standards adoptés par le prestataire informatique

7.1.2 B1.2

- ✓ Traiter des demandes concernant les applications
- ✓ Traiter des demandes concernant les services réseau et système, applicatifs
- ✓ Collecter, suivre et orienter des demandes

7.1.3 B1.3

- ✓ Participer à l'évolution d'un site Web exploitant les données de l'organisation

7.1.4 B1.4

- ✓ Analyser les objectifs et les modalités d'organisation d'un projet
- ✓ Planifier les activités

7.1.5 B1.5

- ✓ Déployer un service
- ✓ Réaliser les tests d'intégration et d'acceptation d'un service
- ✓ Accompagner les utilisateurs dans la mise en place d'un service
- ✓ Mettre en place et vérifier les niveaux d'habilitation associés à un service

7.2 B2 : BLOC 2 – SLAM – Conception et développement d'applications

7.2.1 B2.1

- ✓ Analyser un besoin exprimé et son contexte juridique.
- ✓ Modéliser une solution applicative.
- ✓ Participer à la conception de l'architecture d'une solution applicative.
- ✓ Rédiger des documentations techniques et d'utilisation d'une solution applicative.
- ✓ Identifier, développer, utiliser ou adapter des composants logiciels.
- ✓ Intégrer en continu les versions d'une solution applicative.

- ✓ Réaliser les tests nécessaires à la validation ou à la mise en production d'éléments adaptés ou développés.
- ✓ Exploiter les fonctionnalités d'un environnement de développement et de tests.
- ✓ Utiliser des composants d'accès aux données

7.2.2 B2.2

- ✓ Recueillir, analyser et mettre à jour les informations sur une version d'une solution applicative.
- ✓ Analyser et corriger un dysfonctionnement.
- ✓ Mettre à jour des documentations technique et d'utilisation d'une solution applicative.
- ✓ Évaluer la qualité d'une solution applicative.
- ✓ Élaborer et réaliser les tests des éléments mis à jour.

7.2.3 B2.3

- ✓ Exploiter des données à l'aide d'un langage de requêtes.
- ✓ Concevoir ou adapter une base de données.

7.3 B3 : SLAM - Cybersécurité des services informatiques – 2e année

7.3.1 B3.3

- ✓ Gérer les accès et les privilèges appropriés.
- ✓ Identifier les menaces et mettre en œuvre les défenses appropriées.
- ✓ Vérifier l'efficacité de la protection.

7.3.2 B3.5

- ✓ Participer à la vérification des éléments contribuant à la sûreté d'un développement informatique
- ✓ Prendre en compte la sécurité dans un projet de développement d'une solution applicative
- ✓ Mettre en œuvre et vérifier la conformité d'une solution applicative et de son développement à un référentiel, une norme ou un standard de sécurité
- ✓ Prévenir les attaques