Atelier de professionnalisation

Romain MARTIN

Mediatek86 Formations
- Android

Table des matières

Introduction

Contexte

Il s'agissait de faire évoluer une application mobile (Android) exploitant une base de données relationnelle (MySQL) à travers une API REST.

Le but du projet était de faire évoluer une application mobile présentant des formations afin de gérer les favoris.

Au départ, l'application se composait d'une page d'accueil, d'une page référençant toutes les formations, d'une page contenant le détail d'une formation et d'une page permettant de lire la vidéo.

Figure 1-Page d'accueil

Figure 2-Liste des formations

Figure 3-Détails d'une formation

Figure 4-Vidéo Formation

Outils utilisés

- ✓ **Langage de programmation** : Java
- ✓ **SGBD** : MySQL, Base de donnée locale SQLite
- ✓ **Environnement de travail collaboratif** : Trello, Github
- ✓ **Environnement de développement** : Android Studio (version 2021.1.1.23)
- ✓ **Gestion de versions** : Github
- ✓ **Tests des comportements anormaux** : tests unitaires, tests fonctionnels
- ✓ **Documentation technique** : Javadoc
- ✓ **Contrôle de la qualité du code** : SonarLint

Mission 0 : Démarrage du projet

Mise en place d'un suivi de projet : Trello

J'ai, au début du projet, mis en place un suivi du projet que j'ai tenu à jour tout au long de la réalisation.

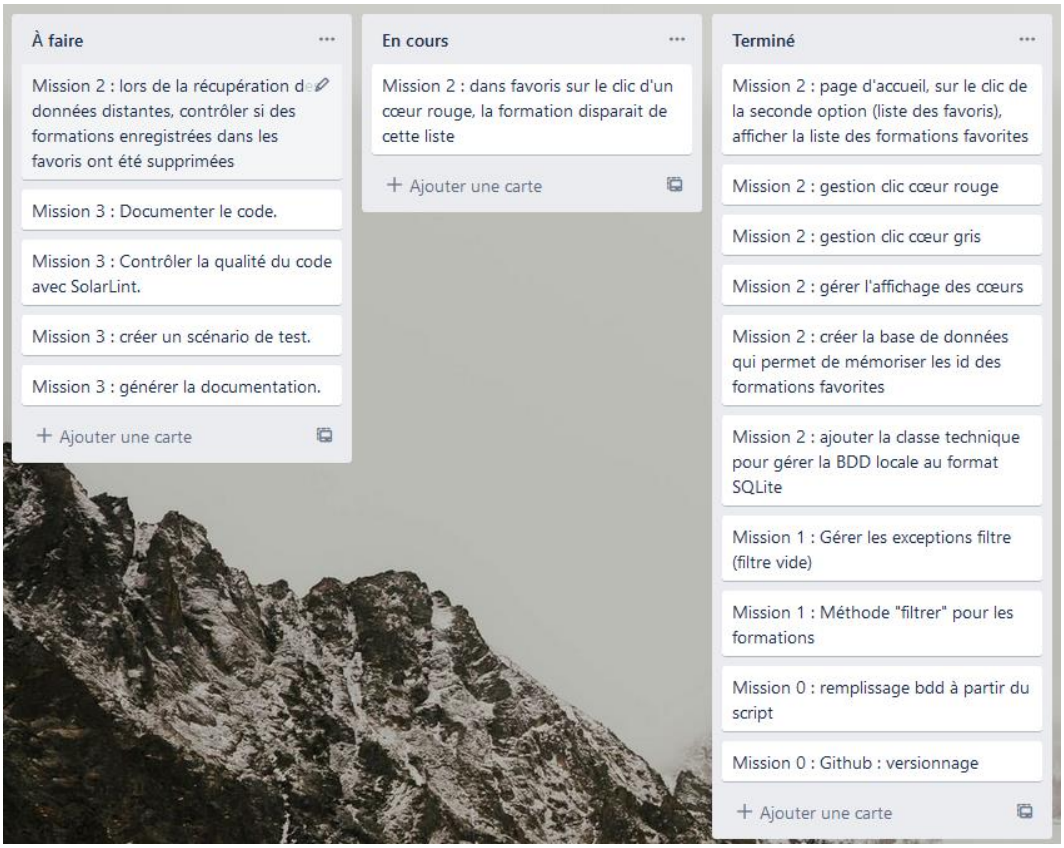


Figure 5-Suivi du projet sous Trello

Gestion du versionnage : GitHub

Tout au long du projet, j'ai utilisé GitHub afin de versionner le projet :
https://github.com/RomainMartin1/Mediatek86-Apk

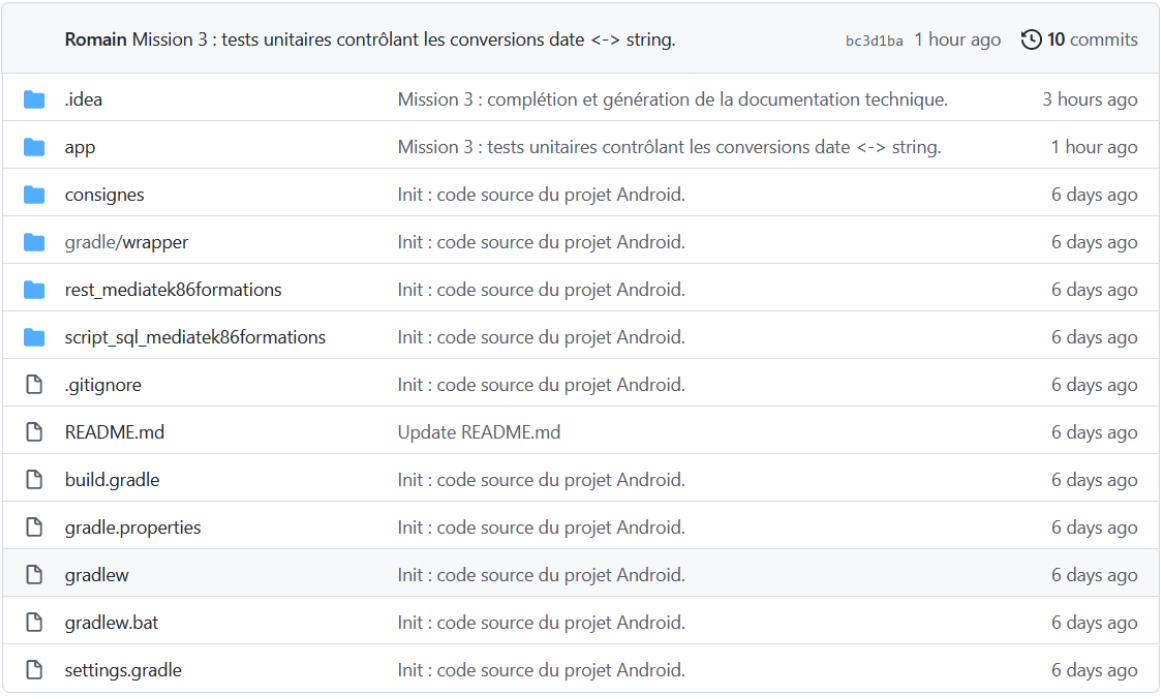
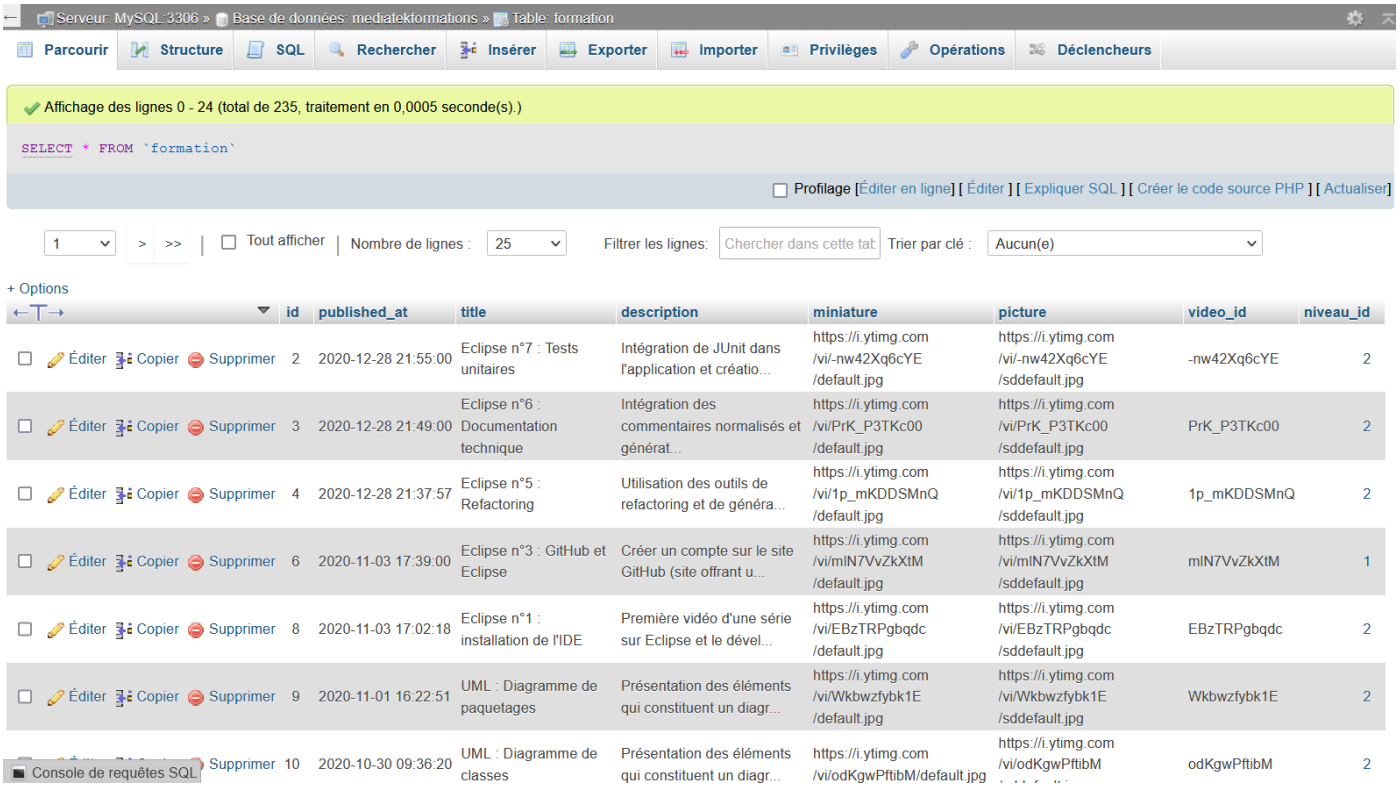


Figure 6-Projet sur GitHub

Base de données mediatekformations

J'ai importé la base de données dans phpmyadmin à l'aide du script fourni dans l'énoncé.



	id	published_at	title	description	miniature	picture	video_id	niveau_id
Éditer Copier Supprimer	2	2020-12-28 21:55:00	Eclipse n°7 : Tests unitaires	Intégration de JUnit dans l'application et créatio...	https://i.ytimg.com/vi/-nw42Xq6cYE/default.jpg	https://i.ytimg.com/vi/-nw42Xq6cYE/sddefault.jpg	-nw42Xq6cYE	2
Éditer Copier Supprimer	3	2020-12-28 21:49:00	Eclipse n°6 : Documentation technique	Intégration des commentaires normalisés et générat...	https://i.ytimg.com/vi/PrK_P3TKc00/default.jpg	https://i.ytimg.com/vi/PrK_P3TKc00/sddefault.jpg	PrK_P3TKc00	2
Éditer Copier Supprimer	4	2020-12-28 21:37:57	Eclipse n°5 : Refactoring	Utilisation des outils de refactoring et de généra...	https://i.ytimg.com/vi/1p_mKDDSMnQ/default.jpg	https://i.ytimg.com/vi/1p_mKDDSMnQ/sddefault.jpg	1p_mKDDSMnQ	2
Éditer Copier Supprimer	6	2020-11-03 17:39:00	Eclipse n°3 : GitHub et Eclipse	Créer un compte sur le site GitHub (site offrant u...	https://i.ytimg.com/vi/mlN7VvZkXtM/default.jpg	https://i.ytimg.com/vi/mlN7VvZkXtM/sddefault.jpg	mlN7VvZkXtM	1
Éditer Copier Supprimer	8	2020-11-03 17:02:18	Eclipse n°1 : installation de l'IDE	Première vidéo d'une série sur Eclipse et le dével...	https://i.ytimg.com/vi/EBzTRPgbqdc/default.jpg	https://i.ytimg.com/vi/EBzTRPgbqdc/sddefault.jpg	EBzTRPgbqdc	2
Éditer Copier Supprimer	9	2020-11-01 16:22:51	UML : Diagramme de paquetages	Présentation des éléments qui constituent un diagr...	https://i.ytimg.com/vi/Wkbwzfybk1E/default.jpg	https://i.ytimg.com/vi/Wkbwzfybk1E/sddefault.jpg	Wkbwzfybk1E	2
Supprimer	10	2020-10-30 09:36:20	UML : Diagramme de classes	Présentation des éléments qui constituent un diagr...	https://i.ytimg.com/vi/odKgwPftibM/default.jpg	https://i.ytimg.com/vi/odKgwPftibM	odKgwPftibM	2

Figure 7-Table Formation Base de Données mediatekformations

Mission 1 : Gestion du filtre des formations

Méthode filtrer (Contrôleur)

La méthode getLesFormationsFiltre permettant de filtrer les formations à partir d'une chaîne de caractères et renvoyant le tableau de formations filtré.

```
    public List<Formation> getLesFormationFiltre(String filtre){
        List<Formation> lesFormationsFiltre = new ArrayList<>();
        for(Formation uneFormation : lesFormations){
            if(uneFormation.getTitle().toUpperCase().contains(filtre.toUpperCase())){
                lesFormationsFiltre.add(uneFormation);
            }
        }
        return lesFormationsFiltre;
    }
```

Création de la méthode événementielle sur le clic du bouton "filtrer" (Vue)

But : 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.

Dans le code de base, il existait déjà un tableau :

```
private ArrayList<Formation> lesFormations
```

Ce tableau était rempli par la liste de toutes les formations et c'est le tableau qui est affiché dans la liste des formations. J'ai décidé de créer un deuxième tableau :

```
private ArrayList<Formation> lesFormationsCopie
```

Ce tableau me permettant de ne pas solliciter tout le temps ma base de données et d'avoir en permanence un tableau de consultation permettant de récupérer la liste de toutes formations si besoin.

J'ai ensuite créé la méthode ecouteFiltrer. Cette méthode permet de récupérer le texte saisi par l'utilisateur. Si le texte saisi est une chaîne de caractère vide, c'est-à-dire si sa taille fait 0, alors on affecte au tableau de formation toutes les formations (lesFormationsCopie). Sinon, on recherche le texte saisi à l'aide de la méthode getLesFormationsFiltre. Il suffit ensuite de rafraîchir l'affichage à l'aide de la méthode déjà fournie creerListe.

```
    private void ecouteFiltrer() {
        btnFiltrer.setOnClickListener(view ->  {
            String txt = txtFiltre.getText().toString();
            if(txt.length() > 0) {
                controle.setLesFormations(controle.getLesFormationFiltre(txt));
            } else {
                controle.setLesFormations(controle.getLesFormationsCopie());
            }
            creerListe();
        });
    }
```

Au niveau du rendu :

Figure 8-Filtre vide

Figure 9-Filtre existant

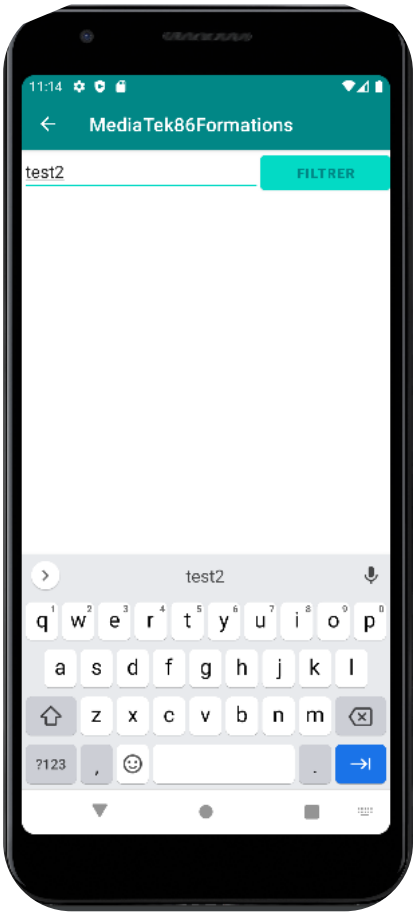


Figure 10-Filtre n'existant pas

Mission 2 : Gestion des favoris

Base de données locale – SQLite

But : ajouter la classe technique pour gérer la BDD locale au format SQLite.

La première tâche de cette deuxième mission était d'ajouter la classe technique AccesLocal (Modèle) permettant de gérer la base de données locale au format SQLite.

```
public class AccesLocal {
    private String nomBase="bdFavoris.sqlite";
    private Integer versionBase=1;
    private MySQLiteOpenHelper accesBD;
    private SQLiteDatabase bd;

    /**
     * Constructeur de la classe AccesLocal.
     * @param context
     */
    public AccesLocal(Context context) {
        accesBD = new MySQLiteOpenHelper(context, nomBase, versionBase);
    }
}
```

Cette classe hérite de la classe MySQLiteOpenHelper :

```
public class MySQLiteOpenHelper extends SQLiteOpenHelper {
      private String creation="create table FormationsFavorites ("+ "id INTEGER PRIMARY KEY);";

    /**
     * Construction de l'accès à une base de données locale.
     * @param context
     * @param name
     * @param version
     */
      public MySQLiteOpenHelper(Context context, String name, int version) {
            super(context, name, null, version);
            // TODO Auto-generated constructor stub
      }

    /**
     * Méthode redéfinie appelée automatiquement par le constructeur
     * uniquement si celui-ci repère que la base n'existe pas encore.
     * @param db
     */
      @Override
      public void onCreate(SQLiteDatabase db) {
            // TODO Auto-generated method stub
            db.execSQL(creation);
      }

    /**
     * Méthode redéfinie appelée automatiquement s'il y a changement de version de la base.
     * @param db
     * @param oldVersion
     * @param newVersion
     */
      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            // TODO Auto-generated method stub
      }
}
```

Dans la classe Contrôle, j'ai ajouté une donnée membre privée accesLocal de type AccesLocal permettant l'accès à la base de données locale.

Création de la base de données FormationsFavorites (Modèle)

But : créer la base de données qui permet de mémoriser les id des formations favorites.

La base de données FormationsFavorites est une base de données contenant uniquement les id des formations favorites. J'ai créé la classe correspondant à cette base de données comme suit :

```
public class FormationsFavorites implements Comparable<FormationsFavorites> {
    private Integer id;

    public FormationsFavorites(Integer id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    @Override
    public int compareTo(FormationsFavorites formationsFavorites) {
        return id.compareTo(formationsFavorites.getId());
    }
}
```

Gestion de l'affichage des cœurs

But : afficher un cœur rouge dans la liste des formations si la formation est une formation favorite, un cœur gris sinon.

AccesLocal

J'ai ajouté la méthode exists dans la classe AccesLocal. Cette méthode prend l'id de la formation en paramètre et renvoie true si une formation est une formation favorite, false sinon.

```
public boolean exists(Integer id) {
        bd = accesBD.getReadableDatabase();
        Cursor curseur = bd.query("FormationsFavorites", null, "id =?", new
String[]{String.valueOf(id)}, null, null, null);
        boolean favoriExiste = !curseur.isAfterLast();
        curseur.close();
        return favoriExiste;
    }
```

Controle

Puis dans Controle, j'ai ajouté la méthode isFavori, qui est appelle la méthode exists de AccesLocal.

```
public boolean isFavori(Integer formationId) {
        return accesLocal.exists(formationId);
    }
```

FormationListAdapter

Enfin, dans la méthode getView de FormationListAdapter, si une formation est une formation favorite, on affiche le cœur en rouge, sinon on l'affiche en gris.

```
        if (controle.isFavori(lesFormations.get(i).getId())) {
            viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);
        } else {
            viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
        }
```

Ajout et suppression d'un favori

But : en cas de clic sur 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. Inversement, en cas de clic sur 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.

AccesLocal

Pour l'ajout, j'ai créé la méthode add, qui ajoute une formation à la base de données FormationsFavorites.

```
    public void add(Formation formation) {
        bd = accesBD.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("id", formation.getId());
        bd.insert("FormationsFavorites", null, values);
        bd.close();
    }
```

Pour la suppression, j'ai crée la méthode remove, qui supprime une formation de la base de données FormationsFavorites à partir d'un id passé en paramètre.

```
    public void remove(Integer id) {
        bd = accesBD.getWritableDatabase();
        String req = "delete from FormationsFavorites where id=" + id;
        bd.execSQL(req);
        bd.close();
    }
```

Controle

La méthode addFavori appelle la méthode add de AccesLocal.

```
public void addFavori(Formation formation) {
        accesLocal.add(formation);
    }
```

La méthode removeFavori appelle la méthode remove de AccesLocal.

```
public void removeFavori(int formationId) {
        accesLocal.remove(formationId);
    }
```

FormationListAdapter

Enfin, dans la méthode getView de FormationListAdapter, si une formation est une formation favorite et qu'on clique sur le cœur, alors on change la couleur du cœur et on appelle la méthode remove du Contrôleur. Si ce n'était pas une formation favorite, on change toujours la couleur du cœur, mais cette fois-ci on appelle la méthode add du Contrôleur.

```
            viewProperties.btnListFavori.setTag(i);
            viewProperties.btnListFavori.setOnClickListener(view1 ->  {
                int position = (int) view1.getTag();
                if (controle.isFavori(lesFormations.get(position).getId())) {
                    viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
                    controle.removeFavori(lesFormations.get(position).getId());
                    notifyDataSetChanged();
                } else {
                    viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);
                    controle.addFavori(lesFormations.get(position));
                    notifyDataSetChanged();
                }
            });
```

Affichage des formations favorites

But : en cas de clic sur 'Mes Favoris', afficher la liste des formations favorites.

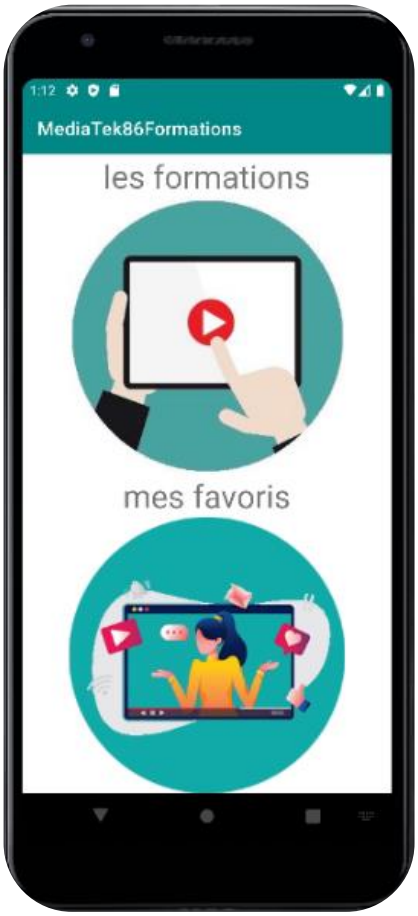


Figure 11-Page d'accueil

Figure 12-Affichage des formations favorites

AccesLocal

J'ai créé la méthode getFavorisId, qui renvoie une liste composée des id de toutes les formations favorites.

```
    public List<Integer> getFavorisId() {
        bd = accesBD.getReadableDatabase();
        String req = "select * from FormationsFavorites";
        List<Integer> favoris = new ArrayList<>();
        Cursor curseur = bd.rawQuery(req, null);
        curseur.moveToFirst();

        while(!curseur.isAfterLast()){
            favoris.add(curseur.getInt(0));
            curseur.moveToNext();
        }

        curseur.close();
        bd.close();
        return favoris;
    }
```

Controle

Dans la classe Contrôle, j'ai ajouté une donné membre privée booléenne 'favori'. Si cette variable est à true, on affiche uniquement les favoris, si elle est à false, on affiche toutes les formations.

J'ai créé la méthode getFavoris, permettant de récupérer la liste de toutes les formations favorites, à l'aide de la méthode getFavorisId d'AccesLocal.

```
    public List<Formation> getFavoris() {
        List<Formation> lesFavoris = new ArrayList<>();
        List<Integer> lesIdsFavoris = accesLocal.getFavorisId();
        for(Formation formations : lesFormations){
            if(lesIdsFavoris.contains(formations.getId())){
                lesFavoris.add(formations);
            }
        }
        return lesFavoris;
    }
```

Main Activity

J'ai modifié la procédure événementielle ecouteMenu en lui passant un booléen en paramètre.

```
    private void ecouteMenu(ImageButton btn, boolean isFavori){
        btn.setOnClickListener(view ->  {
            Controle.getInstance(MainActivity.this).setFavori(isFavori);
            Activity activity = MainActivity.this;
            Intent intent = new Intent(activity, FormationsActivity.class);
            activity.startActivity(intent);
        });
    }
```

FormationsActivity

Dans la méthode init, si on veut afficher les favoris, alors on affiche la liste composé des formations favorites, sinon on affiche toutes les formations.

```
        if(!controle.getFavori()) {
            controle.setLesFormations(controle.getLesFormationsCopie());
        } else {
            controle.setLesFormations(controle.getFavoris());
        }
```

Suppression d'un favori sur la page des favoris

But : Dans la liste des formations favorites, en cas de clic sur cœur rouge, la formation disparaît de cette liste. En cas de clic sur une formation, on affiche les détails d'une formation (exactement comme pour la liste de toutes les formations).

FormationListAdapter

Il faut différencier le cas où on affiche les formations favorites et celui où on affiche toutes les formations. On vérifie donc à l'aide du getter getFavori si favori est à true ou s'il est à false. S'il est à false, on laisse le code comme il était, puisque c'est le cas où toutes les formations sont affichées. Si, en revanche, il est à true et qu'on clique sur un cœur rouge, alors on retire l'id de la formation de la base de données FormationsFavorites comme précédemment, et on enlève également la formation de la liste de formations. Puis on met à jour le tableau de formations favorites pour qu'il corresponde aux nouvelles données.

```
    if(!controle.getFavori()) {
           viewProperties.btnListFavori.setTag(i);
           viewProperties.btnListFavori.setOnClickListener(view1 ->  {
               int position = (int) view1.getTag();
               if (controle.isFavori(lesFormations.get(position).getId())) {
                   viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
                   controle.removeFavori(lesFormations.get(position).getId());
                   notifyDataSetChanged();
               } else {
                   viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);
                   controle.addFavori(lesFormations.get(position));
                   notifyDataSetChanged();
               }
           });
       } else {
           viewProperties.btnListFavori.setTag(i);
           viewProperties.btnListFavori.setOnClickListener(view1 ->  {
               int position = (int) view1.getTag();
               controle.removeFavori(lesFormations.get(position).getId());
               lesFormations.remove(lesFormations.get(position));
               controle.setLesFormationsFavorites(controle.getFavoris());
               notifyDataSetChanged();
           });
       }
```

Mission 3 : Qualité, tests et documentation technique

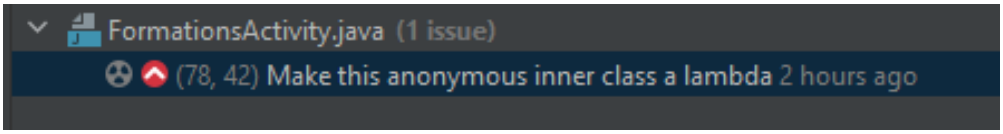
Qualité : SonarLint

But : contrôler la qualité du code avec SonarLint.

Parmi tous les problèmes signalés par SonarLint, il y en a deux qui sont ressortis majoritairement :

1. La transformation des ArrayList en List

2. La transformation en lambda d'une classe anonyme



Par exemple :

Avant

```
private void ecouteFiltrer() {
        btnFiltrer.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                String txt = txtFiltre.getText().toString();
                if(txt.length() > 0) {
                    controle.setLesFormations(controle.getLesFormationFiltre(txt));
                } else {
                    if(!controle.getFavori()) {
                        controle.setLesFormations(controle.getLesFormationsCopie());
                    } else {
                        controle.setLesFormations(controle.getLesFormationsFavorites());
                    }
                }
                creerListe();
            }
        });
}
```

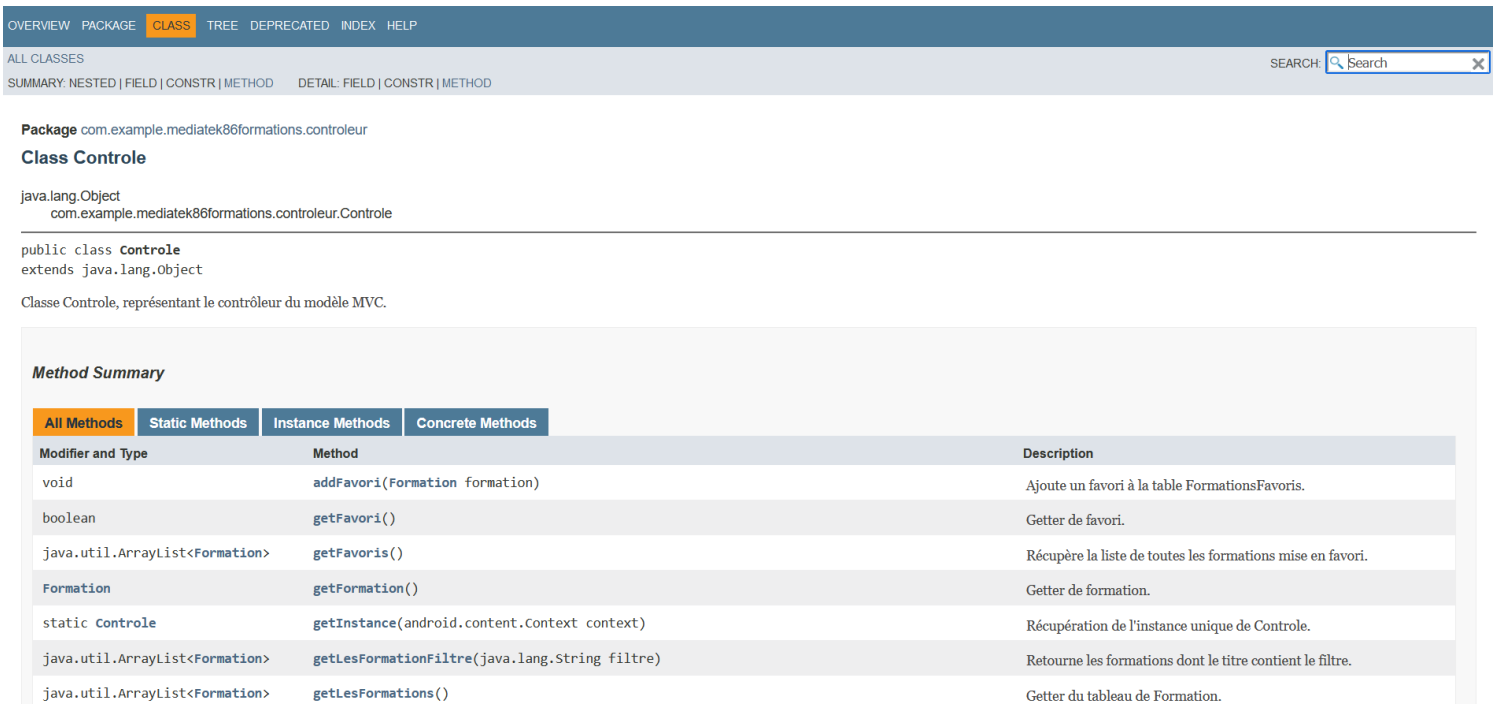
Après

```
    private void ecouteFiltrer() {
        btnFiltrer.setOnClickListener(view ->  {
            String txt = txtFiltre.getText().toString();
            if(txt.length() > 0) {
                controle.setLesFormations(controle.getLesFormationFiltre(txt));
            } else {
                if(!controle.getFavori()) {
                    controle.setLesFormations(controle.getLesFormationsCopie());
                } else {
                    controle.setLesFormations(controle.getLesFormationsFavorites());
                }
            }
            creerListe();
        });
    }
```

Documentation technique

But : générer la documentation technique de l'application.

Tout au long de mon code, j'ai utilisé des commentaires normalisés, ce qui m'a permis ensuite de générer la documentation technique à l'aide de Javadoc.



OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package com.example.mediatek86formations.controleur

Class Controle

java.lang.Object
com.example.mediatek86formations.controleur.Controle

public class **Controle**
extends java.lang.Object

Classe Controle, représentant le contrôleur du modèle MVC.

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	addFavori(Formation formation)	Ajoute un favori à la table FormationsFavoris.
boolean	getFavori()	Getter de favori.
java.util.ArrayList<Formation>	getFavoris()	Récupère la liste de toutes les formations mise en favori.
Formation	getFormation()	Getter de formation.
static Controle	getInstance(android.content.Context context)	Récupération de l'instance unique de Controle.
java.util.ArrayList<Formation>	getLesFormationFiltre(java.lang.String filtre)	Retourne les formations dont le titre contient le filtre.
java.util.ArrayList<Formation>	getLesFormations()	Getter du tableau de Formation.

Figure 13-Exemple documentation technique

Tests unitaires

But : créer les tests unitaires pour contrôler les conversions des dates.

Les conversions de Date en String (et de String en Date) se font dans la classe MesOutils.

J'ai donc généré une classe Test à partir de cette classe. Cette classe teste les trois méthodes de conversion, avec à chaque fois une date qui correspond (assertEquals) et une date qui ne correspond pas (assertNotEquals).

```
public class MesOutilsTest extends TestCase {
    /**
     * Teste la méthode convertStringToDate prenant en paramètre une date au format String et
     * le pattern attendu au format String.
     */
    public void testConvertStringToDate() {
        String date = "10-05-2018 12:15:23";
        String expectedPattern = "dd-MM-yyyy HH:mm:ss";
        Date testDate = MesOutils.convertStringToDate(date, expectedPattern);

        //TEST OK
        Calendar calendar = new GregorianCalendar(2018, Calendar.MAY, 10, 12, 15, 23);
        Date dateTest2 = calendar.getTime();
        Assert.assertEquals(0, testDate.compareTo(dateTest2));

        //TEST PAS OK
        Calendar calendar2 = new GregorianCalendar(2015, Calendar.AUGUST, 10, 12, 15, 23);
        Date dateTest3 = calendar2.getTime();
        Assert.assertNotEquals(0, testDate.compareTo(dateTest3));
    }

    /**
     * Teste la méthode convertStringToDate prenant en paramètre une date au format String
     * (sous la forme "EEE MMM dd hh:mm:ss 'GMT+00:00' yyyy").
     */
    public void testTestConvertStringToDate() {
        String date = "Dim. Mai 08 03:34:45 GMT+00:00 2015";
        Date testDate = MesOutils.convertStringToDate(date);

        //TEST OK
        Calendar calendar = new GregorianCalendar(2015, Calendar.MAY, 8, 3, 34, 45);
        Date dateTest2 = calendar.getTime();
        Assert.assertEquals(0, testDate.compareTo(dateTest2));

        //TEST PAS OK
        Calendar calendar2 = new GregorianCalendar(2015, Calendar.AUGUST, 10, 12, 15, 23);
        Date dateTest3 = calendar2.getTime();
        Assert.assertNotEquals(0, testDate.compareTo(dateTest3));
    }

    /**
     * Teste la méthode convertDateToString.
     */
    public void testConvertDateToString() {
        Calendar calendar = new GregorianCalendar(2015, Calendar.MAY, 15, 3, 34, 45);
        String date = MesOutils.convertDateToString(calendar.getTime());

        Assert.assertEquals("15/05/2015", date);

        Assert.assertNotEquals("25/12/1995", date);
    }
}
```

Tests fonctionnels

But : créer un scénario de tests pour contrôler les différentes manipulations.

Bilan final

Ce projet m'a permis de développer mes compétences de gestion et d'organisation.

Compétences acquises

- ✓ B1.2 – Répondre aux incidents et aux demandes d'assistance et d'évolution
  - ⇨ **Traiter des demandes concernant les applications** : respecter les consignes du cahier de charges

- ✓ B1.4 – Travailler en mode projet
  - ⇨ **Outils de gestion de projet** : versionnage du projet sur GitHub
  - ⇨ **Planifier les activités** : suivi du Projet sur Trello

- ✓ B1.5 – Mettre à disposition des utilisateurs un service informatique
  - ⇨ **Réaliser les tests d'intégration et d'acceptation d'un service** : tests unitaires et fonctionnels
  - ⇨ **Déployer un service** : déploiement de l'API Rest et de la base de données
  - ⇨ **Accompagner les utilisateurs dans la mise en place d'un service** : Guide d'utilisation de l'application

Table des illustrations