

Compte rendu application Android CCI-Promotion

Sommaire

1. Introduction.....
2. Préparation de l'environnement de travail.....
3. Mission 1 : Pages liste des promotions et détail d'une promotion.....
4. Mission 2 : Page messages.....
5. Mission 3 : Page favoris.....
6. Déploiement.....
7. Conclusion.....
8. Compétences validées.....

Introduction

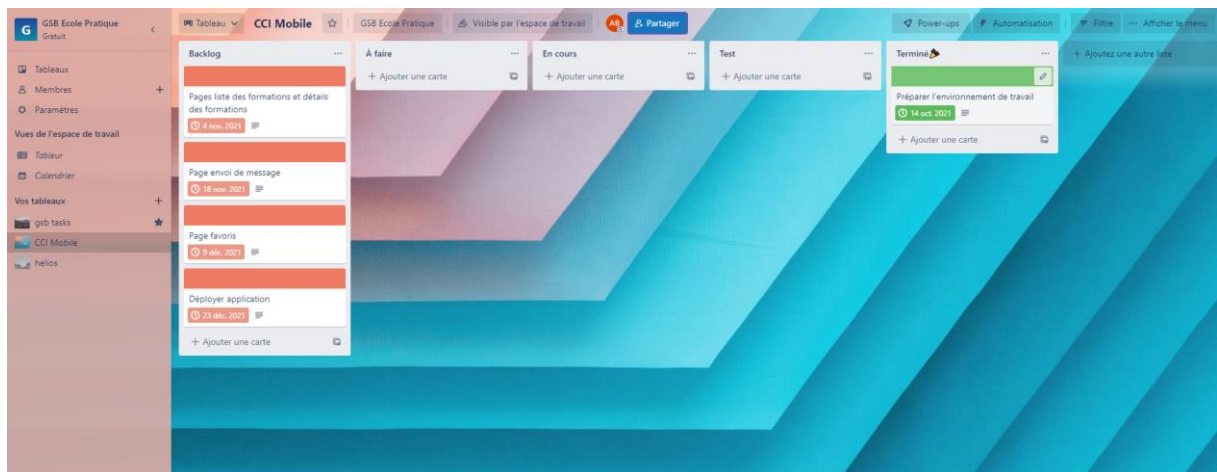
Badenia Tech vient de remporter le marché de développement d'une application mobile Android qui doit participer à la promotion de la chambre de commerce et de l'industrie, particulièrement, de leurs formations proposées. En tant que technicien développeur junior pour l'ESN, ma mission est faire évoluer l'application existante qui est basique et comporte une seule IHM, la page d'accueil.

Préparation de l'environnement de travail

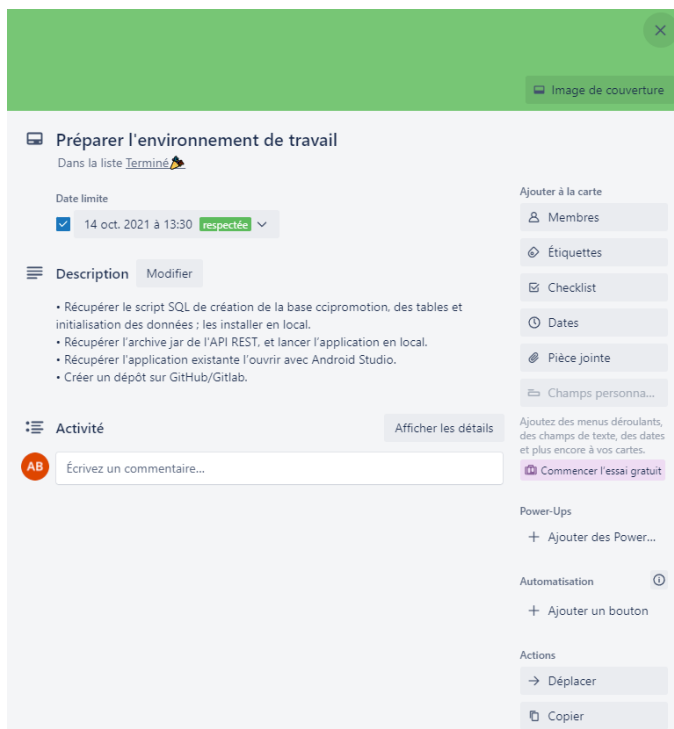
Avant de commencer la réalisation des besoins. Il est important de mettre en place la gestion de ces derniers. Celle-ci se fera grâce à l'outil Trello (voir notice en annexe).

Trello permettra de connaître clairement les besoins et de les suivre tout au long des réalisations. Toutes les tâches à effectuer sont listées dans le Backlog et ensuite attribuées.

Voici un aperçu du tableau après avoir ajouté les tâches et préparé l'environnement de travail :



Il est possible de voir la date limite des tâches et quand on clique dessus on voit ceci :



C'est le détail de la tâche, permettant, au développeur en charge, de savoir quels outils à utiliser ou procédures à suivre.

Il est maintenant possible de commencer l'installation du projet et des outils.

En premier temps, j'ai téléchargé le fichier contenant le code SQL pour initialiser la base de données.

La base de données « ccipromotion » est au format mysql.

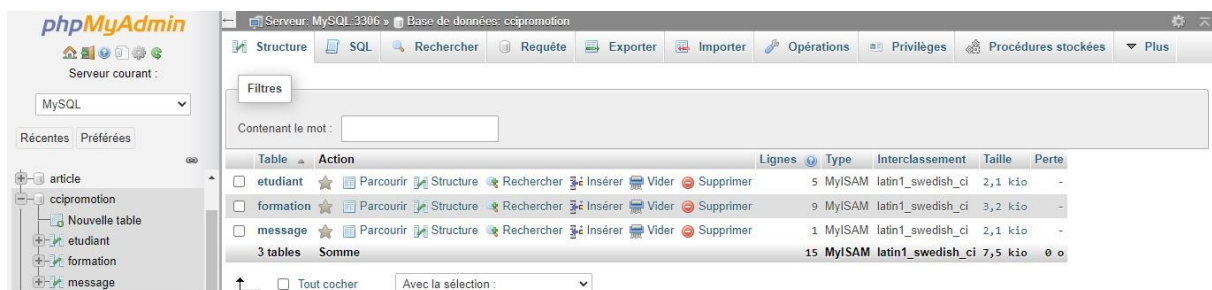
Elle contient deux tables :

Table « formation »

```
CREATE TABLE IF NOT EXISTS `formation` (  
  `acronyme` varchar(20) NOT NULL,  
  `adresse_image` varchar(255) DEFAULT NULL,  
  `date_debut` date DEFAULT NULL,  
  `description` text,  
  `duree_mois` int(11) NOT NULL,  
  `intitule` varchar(255) DEFAULT NULL,  
  `link` varchar(255) DEFAULT NULL,  
  `video_url` varchar(255) DEFAULT NULL,  
  PRIMARY KEY (`acronyme`)  
)
```

Table « message »

```
CREATE TABLE IF NOT EXISTS `message` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `email` varchar(255) DEFAULT NULL,  
  `message` varchar(255) DEFAULT NULL,  
  `nom` varchar(255) DEFAULT NULL,  
  `prenom` varchar(255) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
)
```



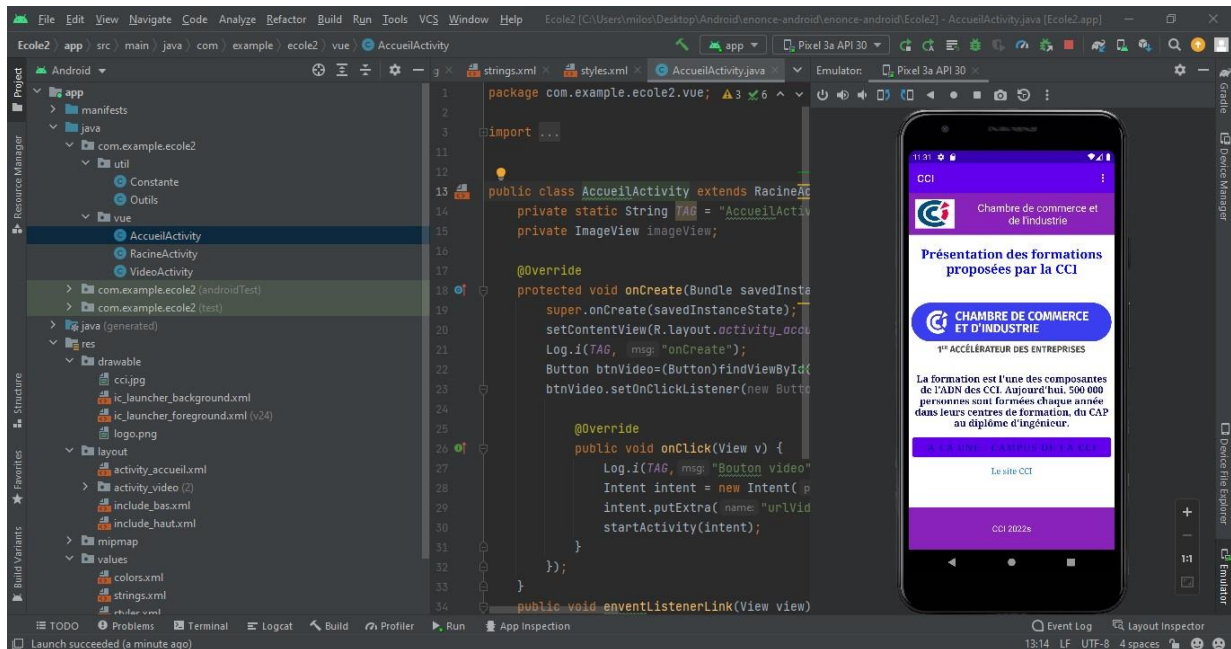
Ensuite j'ai téléchargé le fichier jar permettant de lancer l'API Rest. Pour cela il est requis de posséder la suite WAMP (pour Windows), je l'avais déjà installée auparavant. Pour lancer l'API, il suffit d'entrer la commande suivante dans l'invite de commande :

- java -jar cci5.jar

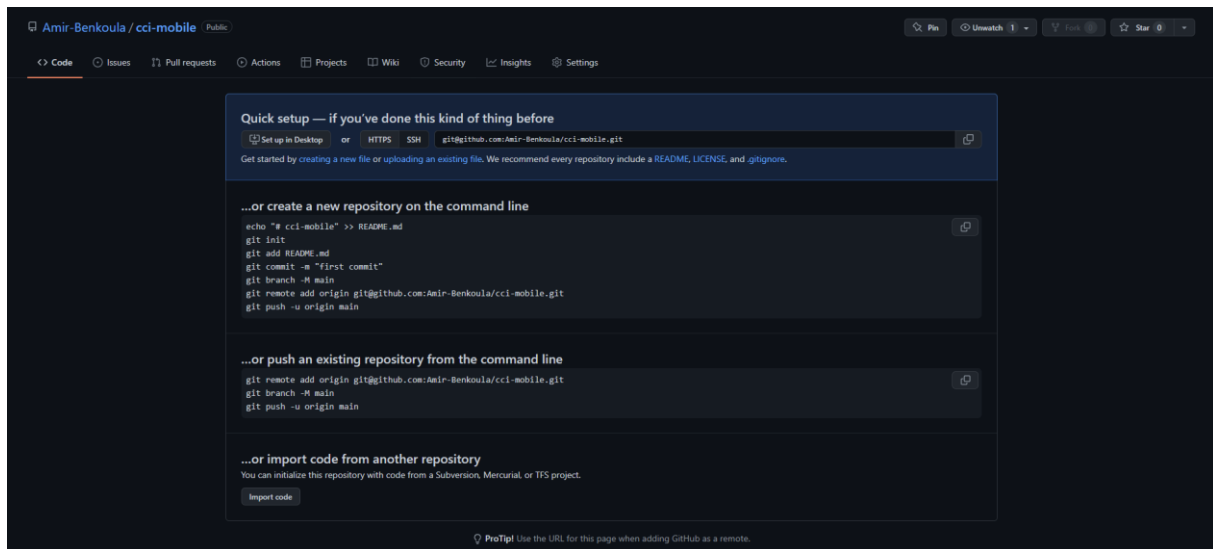
Le serveur est donc disponible à utilisation :

[illegible]

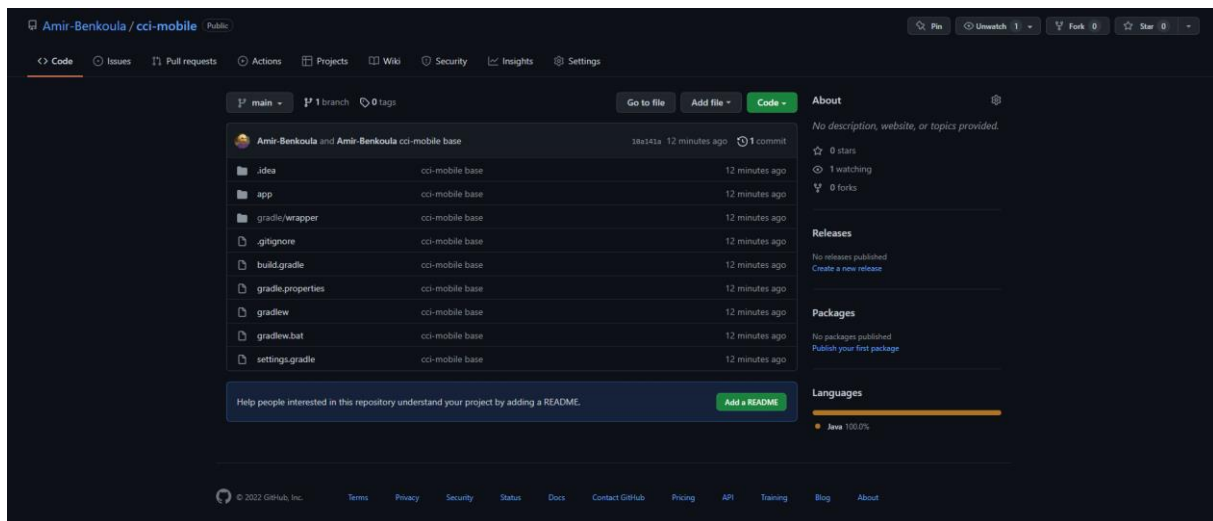
Tout est prêt pour utiliser l'application. J'ai donc testé sur Android Studio qu'elle fonctionnait bien :



Enfin, il ne reste plus qu'à créer un dépôt distant afin d'enregistrer la progression de l'application. J'ai donc utilisé l'outil GitHub (voir notice en annexe). J'ai créé le répertoire cci-mobile :



Et enregistré le projet de base sur ce répertoire avec les commandes suivantes recommandées par le site. Voici le répertoire après le premier « commit » :



Mission 1 : Pages liste des formations et détails des formations

La première mission consiste à créer une page contenant la liste des formations du cci, quand on clique sur une formation de la liste, une autre page doit s'ouvrir avec le détail de la formation. Voici la tâche sur Trello :



C'est la base de données qui contient toutes les formations. Afin d'y accéder l'API fournit des endpoint accessible via des requête GET. Voici le détail des services exposés par l'API :

L'adresse suivante permet d'accéder aux informations des différentes formations proposées par la CCI :

[http://\[host\]:8089/ecole/formation](http://[host]:8089/ecole/formation)

[host] : adresse IP de l'hôte hébergeant l'application.

Ecole : nom de l'application

Exemple :

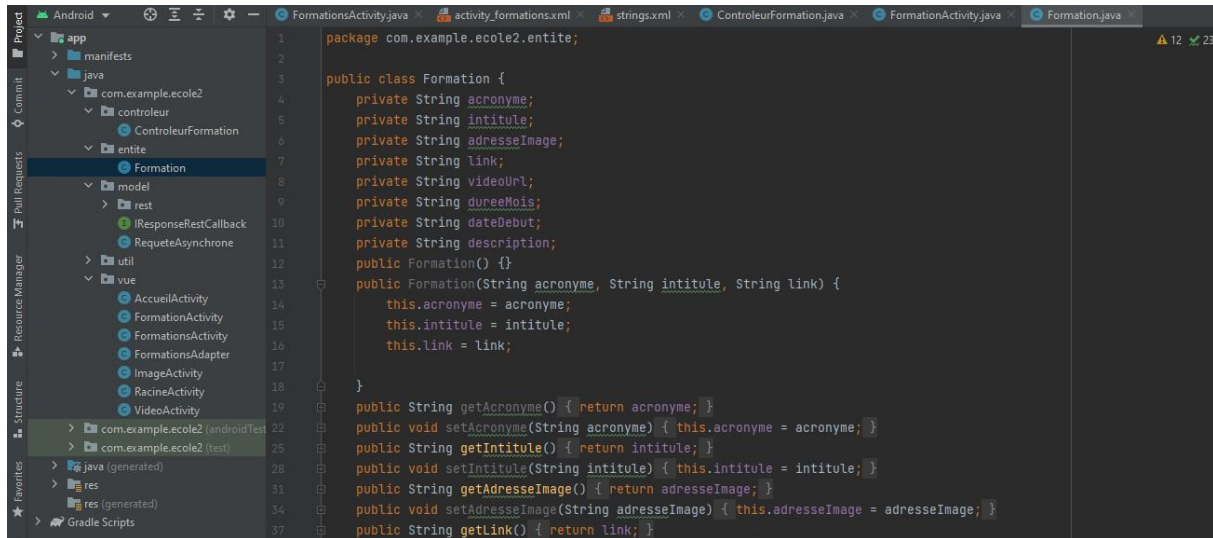
Requête : GET - [http://\[host\]:8089/ecole/formation](http://[host]:8089/ecole/formation)

Réponse :

```
[
  {
    "acronyme": "ASC22",
    "intitule": "Assistant(e) Commercial(e)",
    "adresseImage": "6CVnKuw.png",
    "link": "https://www.campuscci.fr/liste-formations/liste-formations-bac2/assistant-commercial&ar=formations-diplomantes",
    "videoUrl": "UDph-yE1LZQ",
    "dureeMois": 16,
    "dateDebut": "2022-08-01",
    "description": "L'assistant (e) commercial assure le suivi ..."
  },
  {
    "acronyme": "DISII22",
    "intitule": "..."
  }
]
```

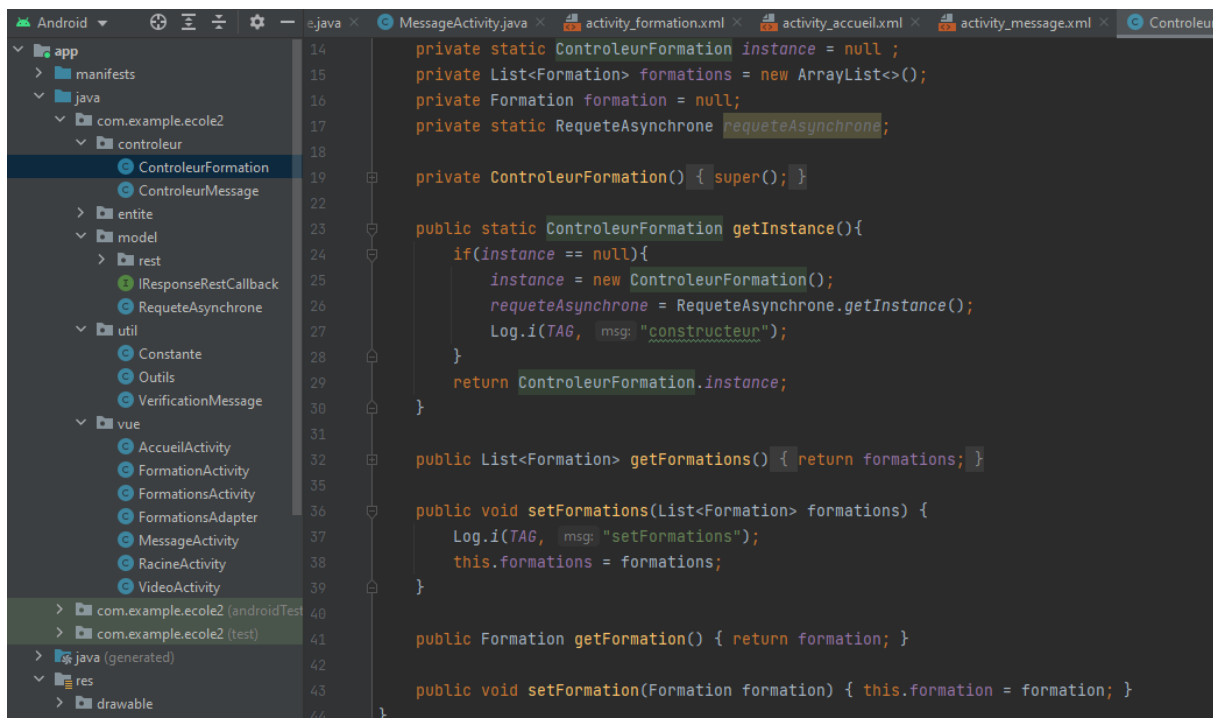

Voici le code de la fonctionnalité :

L'application suit l'architecture MVC, avec d'abord, l'entité Formation :



Elle contient les attributs de la formation et des méthodes de base pour récupérer et définir ces attributs.

Ensuite le Contrôleur :

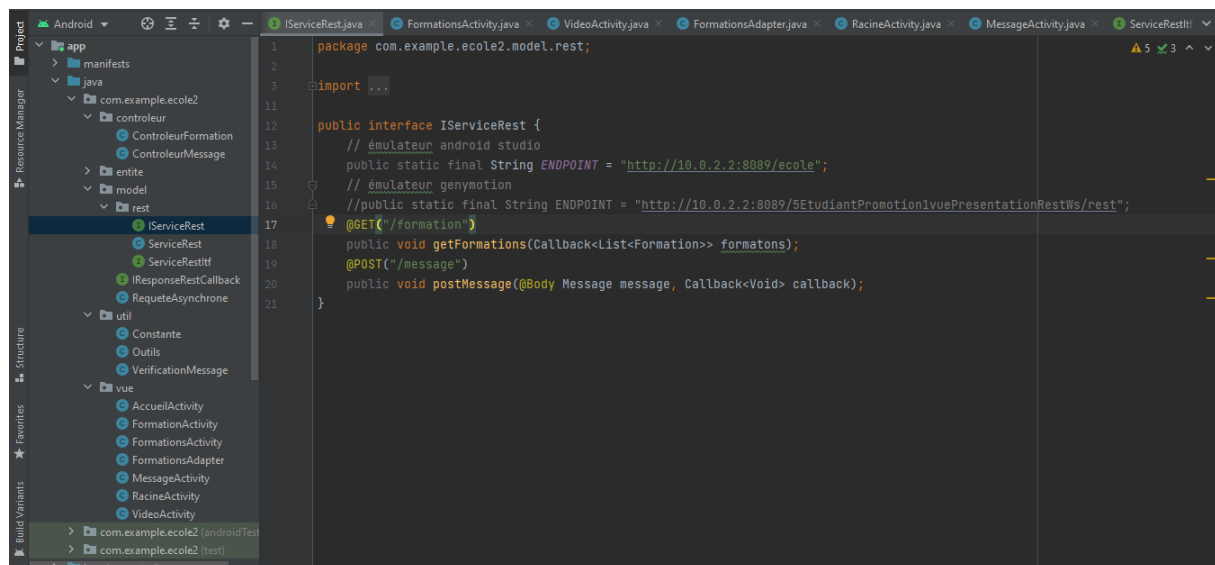


Il va servir à initialiser les données dans l'application à son instantiation grâce à la méthode getInstance.

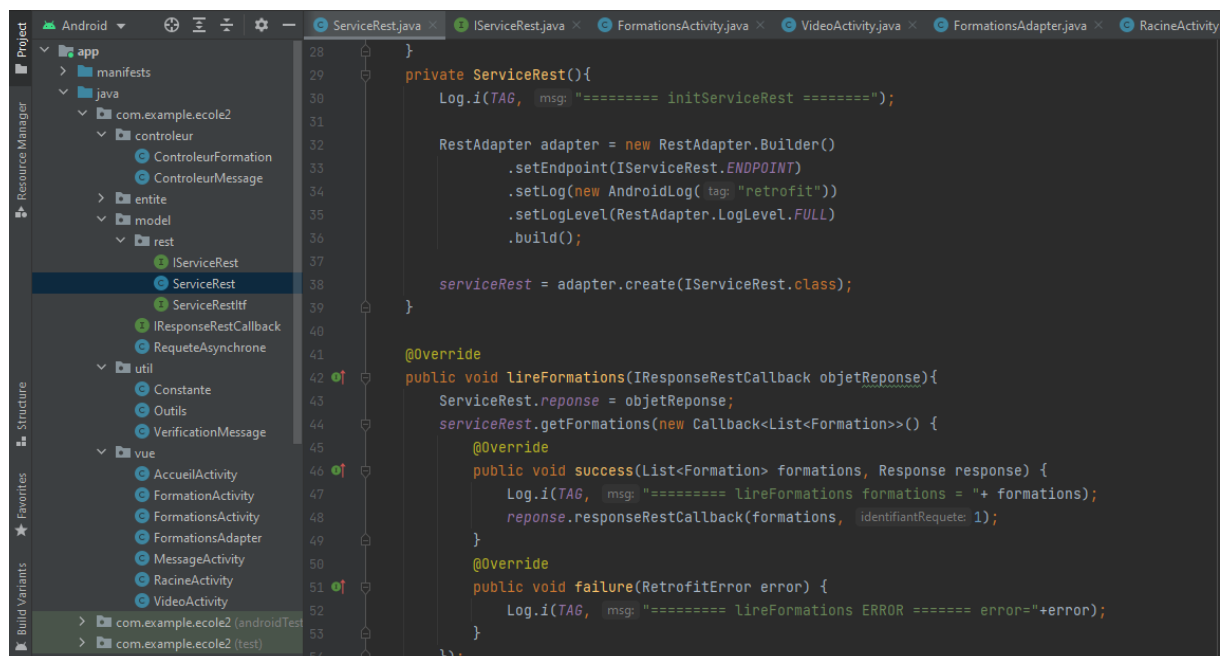
On peut voir que s'il n'y a pas d'instance du contrôleur, donc pas de données récupérées, la méthode va appeler une autre méthode de la classe RequeteAsynchrone, qui est en fait le model qui va effectuer l'appel sur l'API. Le model est la couche qui va interagir avec la bdd.

Voici la couche Model :

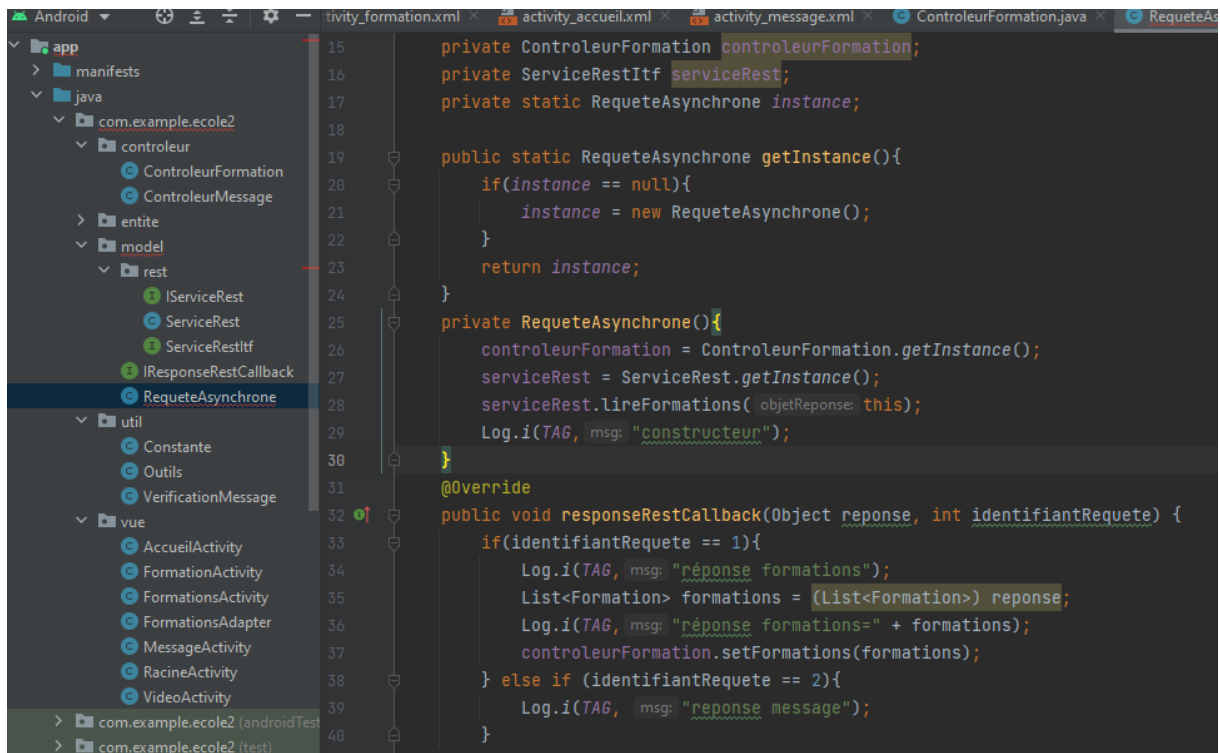
Ici, l'interface qui configure les endpoint a appeler avec quel type de requête http, on passe un objet Callback a getFormations, pour récupérer la liste des formations.



Dans ServiceRest, on initialise le service et on a une methode lireFormations qui va faire la requête GET sur l'endpoint /formation. Si la requête échoue, cela nous renvoie une erreur en console, sinon on instancie objetReponse avec la liste des formations du callback.



On appelle ensuite lireFormations dans RequeteAsynchrone, qui va initialiser le service rest et le contrôleur. Et ensuite dans responseRestCallback on va définir la liste des formations avec la méthode setFormations du contrôleur.



```
15 private ControleurFormation controleurFormation;
16 private ServiceRestItf serviceRest;
17 private static RequeteAsynchrone instance;
18
19 public static RequeteAsynchrone getInstance(){
20     if(instance == null){
21         instance = new RequeteAsynchrone();
22     }
23     return instance;
24 }
25 private RequeteAsynchrone(){
26     controleurFormation = ControleurFormation.getInstance();
27     serviceRest = ServiceRest.getInstance();
28     serviceRest.lireFormations( objetReponse: this);
29     Log.i(TAG, msg: "constructeur");
30 }
31
32 @Override
33 public void responseRestCallback(Object reponse, int identifiantRequete) {
34     if(identifiantRequete == 1){
35         Log.i(TAG, msg: "réponse formations");
36         List<Formation> formations = (List<Formation>) reponse;
37         Log.i(TAG, msg: "réponse formations=" + formations);
38         controleurFormation.setFormations(formations);
39     } else if (identifiantRequete == 2){
40         Log.i(TAG, msg: "reponse message");
41     }
42 }
```

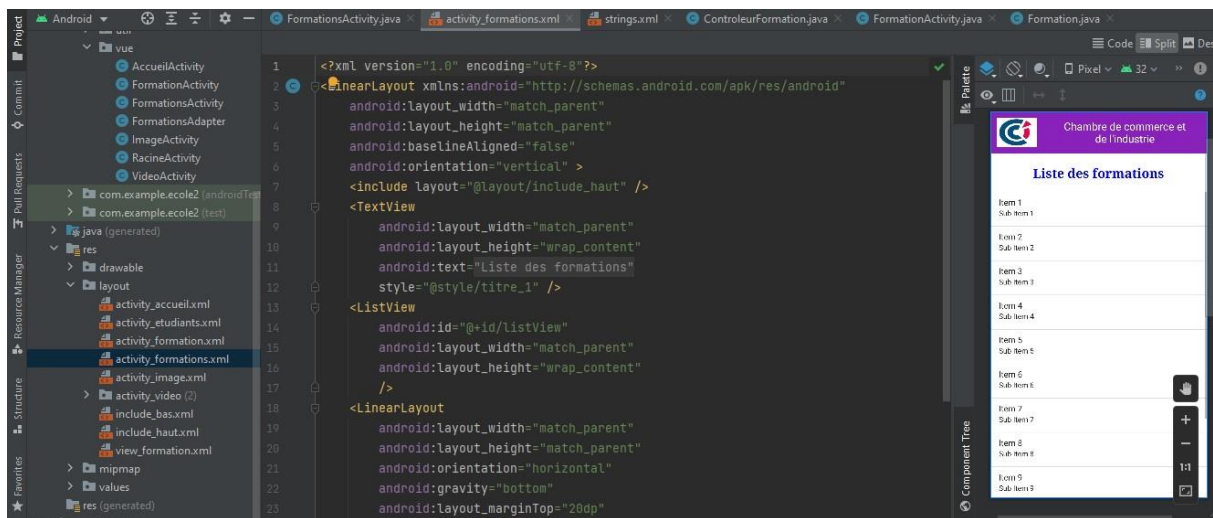
Nous avons donc maintenant réussi à récupérer la liste des formations et à la définir dans l'application.

Il ne reste donc plus qu'à coder la Vue :

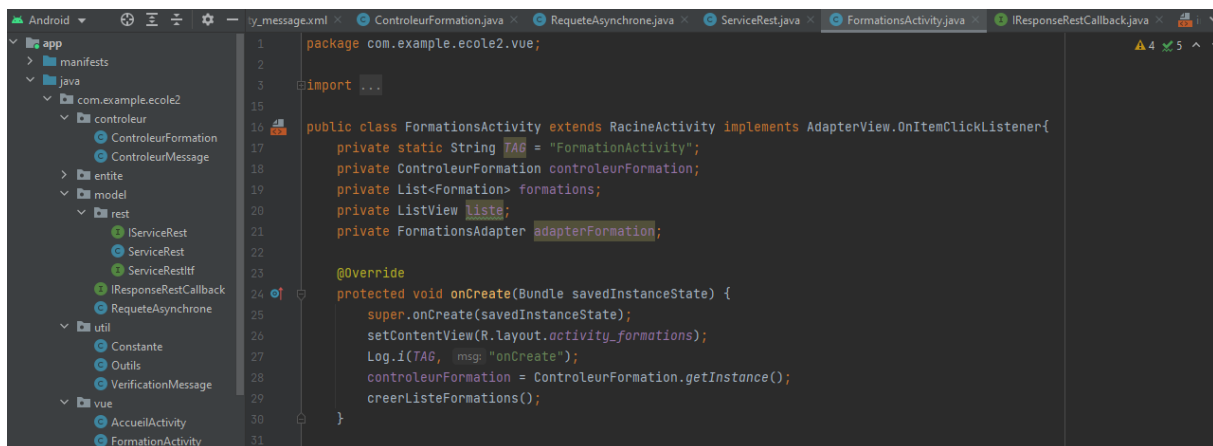
C'est la partie qui s'occupe de l'affichage, ce que va voir l'utilisateur. La Vue d'une application Android est composée d'un fichier activité en Java et d'un autre en xml.

Le xml permet la mise en forme de la page, et le Java permet de charger le contenu en fonction du xml. Par exemple, pour la vue d'une seule formation, on a le xml qui suit :

Le xml est en langage de balisage, comme le HTML, l'interface se crée grâce a des balises. Ici on a une balise TextView, pour afficher le texte Liste des formations, et une balise ListView, pour la liste.

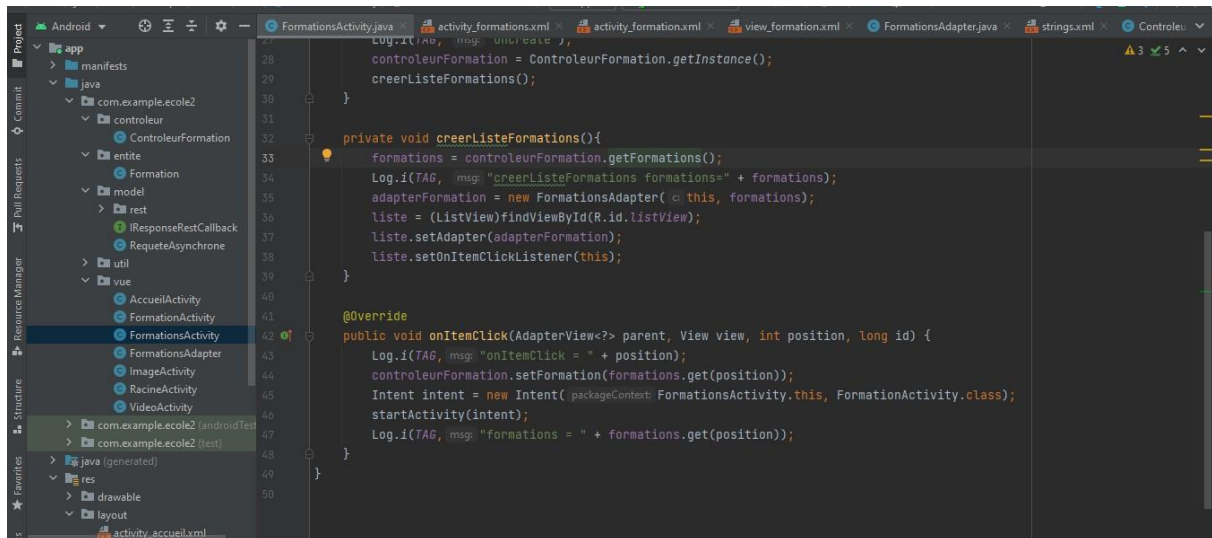


Voici le code Java lié à cette interface :



La classe FormationsActivity hérite de RacineActivity et qui contient la méthode onCreate. Cette dernière est redéfinie dans chaque activité pour le chargement des vues.

setContentview, va utiliser le layout en xml, ensuite on va instancier le contrôleur des formations afin de pouvoir ensuite appeler sa méthode getFormations dans creeListeFormations ci-dessous :



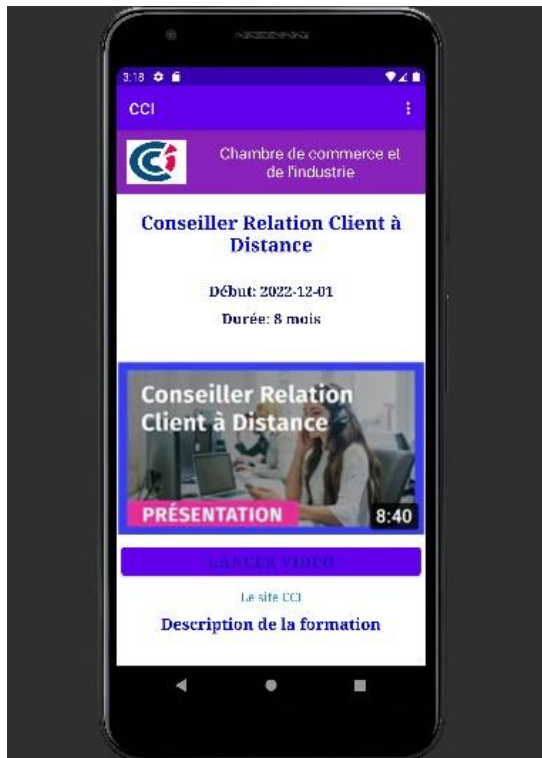
Ici on récupère la liste des formations et on va utiliser un adapteur. L'adapteur est un objet contenant les données qu'on a récupéré et qui va être assigné, ici, à la balise ListView, pour générer la liste des formations.

Afin d'avoir accès au détail des formations, on rend les éléments de la liste cliquables avec `setOnItemClickListener`. Et on va redéfinir la méthode `onItemClick` provenant de `AdapterView.OnItemClickListener`. La méthode va créer une intention. L'intention permet de charger une vue en mentionnant celle actuelle. Donc ici, on récupère la position d'une formation dans la liste et on l'affecte au contrôleur, ensuite on crée et on lance l'intention. Ça va charger la vue et on va donc obtenir le détail de la formation sélectionnée dans la liste.

La vue de détail d'une formation suit le même fonctionnement que celui décrit précédemment pour la liste, c'est-à-dire, une interface en xml, une activité en Java, et des méthodes permettant de gérer des événements en fonction des interactions de l'utilisateur :

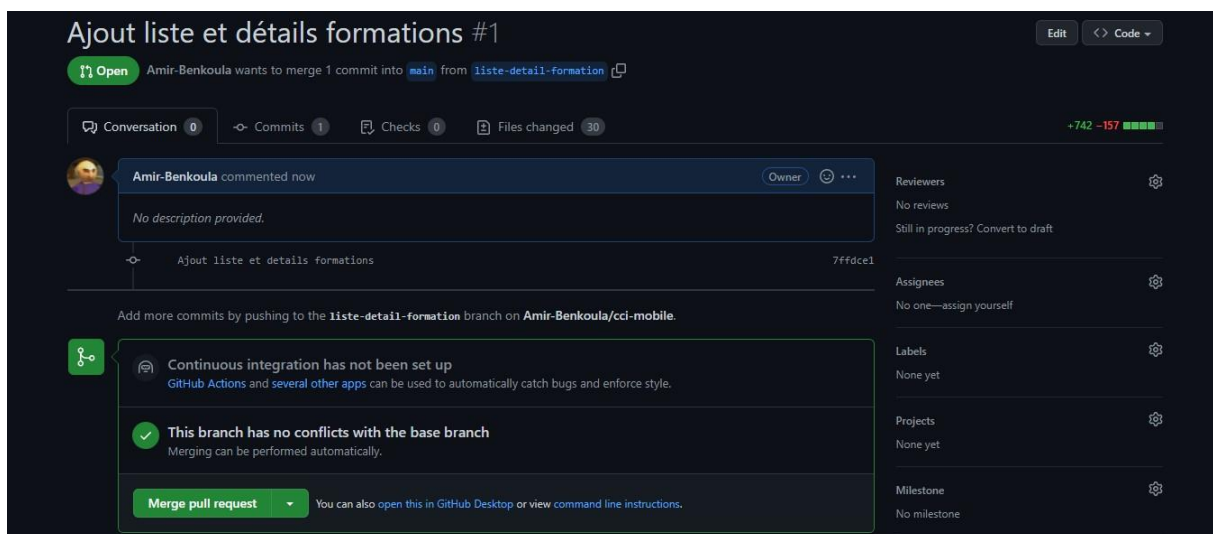
- Charger une image
- Lire une vidéo
- Accéder au site de la formation

Voici à quoi ressemble la page de détail d'une formation :



Après avoir finalisé la tâche, j'envoie le code sur GitHub, sûr une branche distante de la principale. Ce qui va générer une Pull Request, une demande de fusion avec la branche principale.

La PR contient le détail des changements, le code, et les commits. Pour effectuer la fusion il faut que les deux branches ne rencontrent pas de conflits, quand tout est bon, on peut confirmer, et la branche principale est mise à jour. Cette opération va être répétée pour chaque ajout de fonctionnalité.



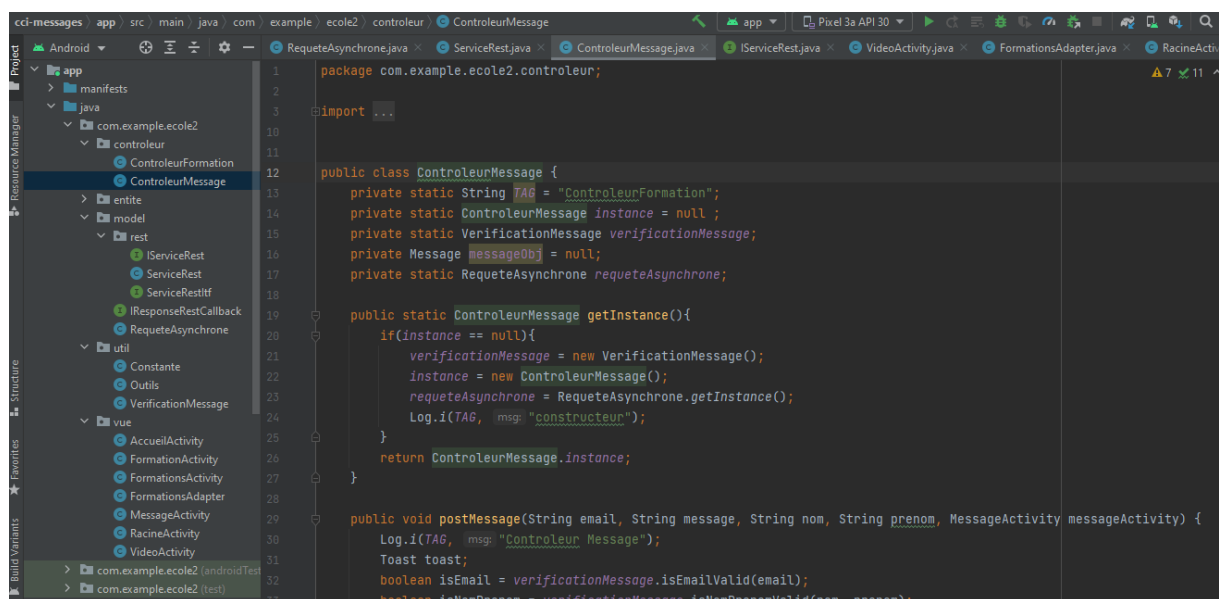
Mission 2 : Page messages

Une page « envoyer un message » est requise pour que les utilisateurs puissent contacter l'administration afin de donner leur avis ou de poser des questions.

Pour se faire, la base de données possède une table message, qui va contenir les champs : email, nom, prénom et message.

Dans le code de l'application, ça se traduit par ajouter une requête POST sur l'endpoint <http://localhost:8089/ecole/message>.

On continue de suivre le modèle MVC donc, il y a un contrôleur :



```
1 package com.example.ecole2.controller;
2
3 import ...
4
5
6
7
8
9
10
11
12 public class ControllerMessage {
13     private static String TAG = "ControllerFormation";
14     private static ControllerMessage instance = null ;
15     private static VerificationMessage verificationMessage;
16     private Message messageObj = null;
17     private static RequeteAsynchrone requeteAsynchrone;
18
19     public static ControllerMessage getInstance(){
20         if(instance == null){
21             verificationMessage = new VerificationMessage();
22             instance = new ControllerMessage();
23             requeteAsynchrone = RequeteAsynchrone.getInstance();
24             Log.i(TAG, "constructeur");
25         }
26         return ControllerMessage.instance;
27     }
28
29     public void postMessage(String email, String message, String nom, String prenom, MessageActivity messageActivity) {
30         Log.i(TAG, "msg: \"Contrôleur Message\"");
31         Toast toast;
32         boolean isEmail = verificationMessage.isEmailValid(email);
33         boolean isNonPrenom = verificationMessage.isNonPrenomValid(nom, prenom);
```

Cette fois le contrôleur possède une requête postMessage qui d'abord faire une vérification du formulaire, puis envoyer le message avec cette methode du model :



```
32 @Override
33 public void RequeteAsynchroneMessage(Message message){
34     serviceRest = ServiceRest.getInstance();
35     serviceRest.postMessage(message, objetReponse: this);
36     Log.i(TAG, "msg: \"requete message\"");
37 }
38
```

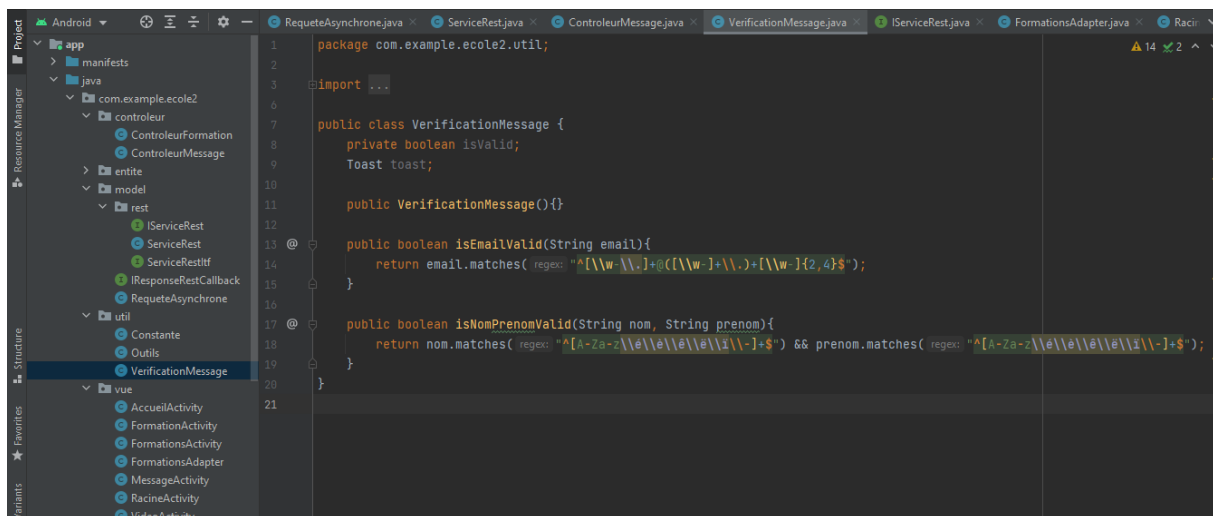

Qui appelle postMessage du service rest :



On utilise aussi l'objet Callback ici, pour gérer la réponse de la requête.

Pour la validation, j'ai utilisé des expressions régulières. En informatique, une expression régulière ou expression rationnelle ou expression normale ou motif est une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles. Les expressions régulières sont également appelées regex.

Le code :



Afin de vérifier que la validation fonctionne bien, j'ai rédigé une série de tests unitaires avec des entrées qui ne devraient pas passer. Un test unitaire consiste à appeler une fonction de la couche service ou contrôleur afin de tester son fonctionnement.

Voici les tests :

```

package com.example.ecole2.util;

import junit.framework.TestCase;

public class VerificationMessageTest extends TestCase {
    VerificationMessage verificationMessage = new VerificationMessage();

    public void testIsEmailValid1() {
        assertEquals( expected: false, verificationMessage.isEmailValid("test.com"));
    }

    public void testIsEmailValid2() {
        assertEquals( expected: false, verificationMessage.isEmailValid("test@test"));
    }

    public void testIsEmailValid3() {
        assertEquals( expected: false, verificationMessage.isEmailValid("test@test.a"));
    }

    public void testIsEmailValid4() {
        assertEquals( expected: false, verificationMessage.isEmailValid("test@test.ca"));
    }

    public void testIsEmailValid5() {
        assertEquals( expected: false, verificationMessage.isEmailValid("test@test.com"));
    }
}

```

Pour la vue, je dois récupérer les entrées du formulaire et les passer la méthode du contrôleur :

```

EditText editMail;
EditText editMessage;
EditText editNom;
EditText editPrenom;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_message);
    controleurFormation = ControleurFormation.getInstance();
    controleurMessage = ControleurMessage.getInstance();
    Log.i(TAG, msg: "onCreate");
    Button btnMsg=(Button)findViewById(R.id.envoyerMessage);
    btnMsg.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View v) {
            Log.i(TAG, msg: "Bouton message");
            editMail = (EditText) findViewById(R.id.editEmail);
            editMessage = (EditText) findViewById(R.id.editMessage);
            editNom = (EditText) findViewById(R.id.editNom);
            editPrenom = (EditText) findViewById(R.id.editPrenom);
            controleurMessage.postMessage(editMail.getText().toString(), editMessage.getText().toString(), editNom.getText().toString(), editPrenom.getText().toString());
        }
    });
}

```


Au moment du clic sur le bouton d'envoi, on assigne les entrées a des objets EditText avec findViewById(), qui sont convertis en chaîne de caractère String avec toString() dans la méthode postMessage. Ainsi la méthode va valider les entrées et envoyer le message.

En base de données on peut bien voir que la requête a fonctionné :

+ Options

	id	email	formation	message	nom	prenom
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	1	hugo@victor.com	Assistant(e) Commercial(e)	Je voudrais avoir des informations complémentaires...	Hugo	Victor

Et voilà à quoi ressemble la page :



The screenshot displays a mobile application interface for the CCI (Chambre de commerce et de l'industrie). The top status bar shows the time 7:01 and various icons. The app's header is blue with the CCI logo and the text 'CCI' and 'Chambre de commerce et de l'industrie'. Below the header, the title 'Envoyer un message' is centered. The form consists of four input fields: 'Email', 'Nom', 'Prenom', and 'Message'. At the bottom of the form is a blue button labeled 'ENVOYER'. The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

En cas d'erreur, un message apparait sur l'écran. Grâce à l'objet Toast

Mission 3 : Page Favoris

La page « favoris » contient les formations favorites par l'utilisateur de l'application.

Pour mettre en place cette page, il a été choisi d'utiliser la base de données locale de l'appareil mobile. Cela se fera à l'aide de SQLite.

D'abord, il faut initialiser la bdd avec la classe DatabaseOpenHelper :

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {

    public final static String TABLE_NAME = "favoris";
    public final static String FORMATION_NAME = "formation_name";
    public final static String _ID = "_id";
    public final static String[] columns = { _ID, FORMATION_NAME };

    final private static String CREATE_CMD =

        "CREATE TABLE favoris (" + _ID
        + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + FORMATION_NAME + " TEXT NOT NULL)";

    final private static String NAME = "favoris_db";
    final private static Integer VERSION = 1;
    final private Context mContext;

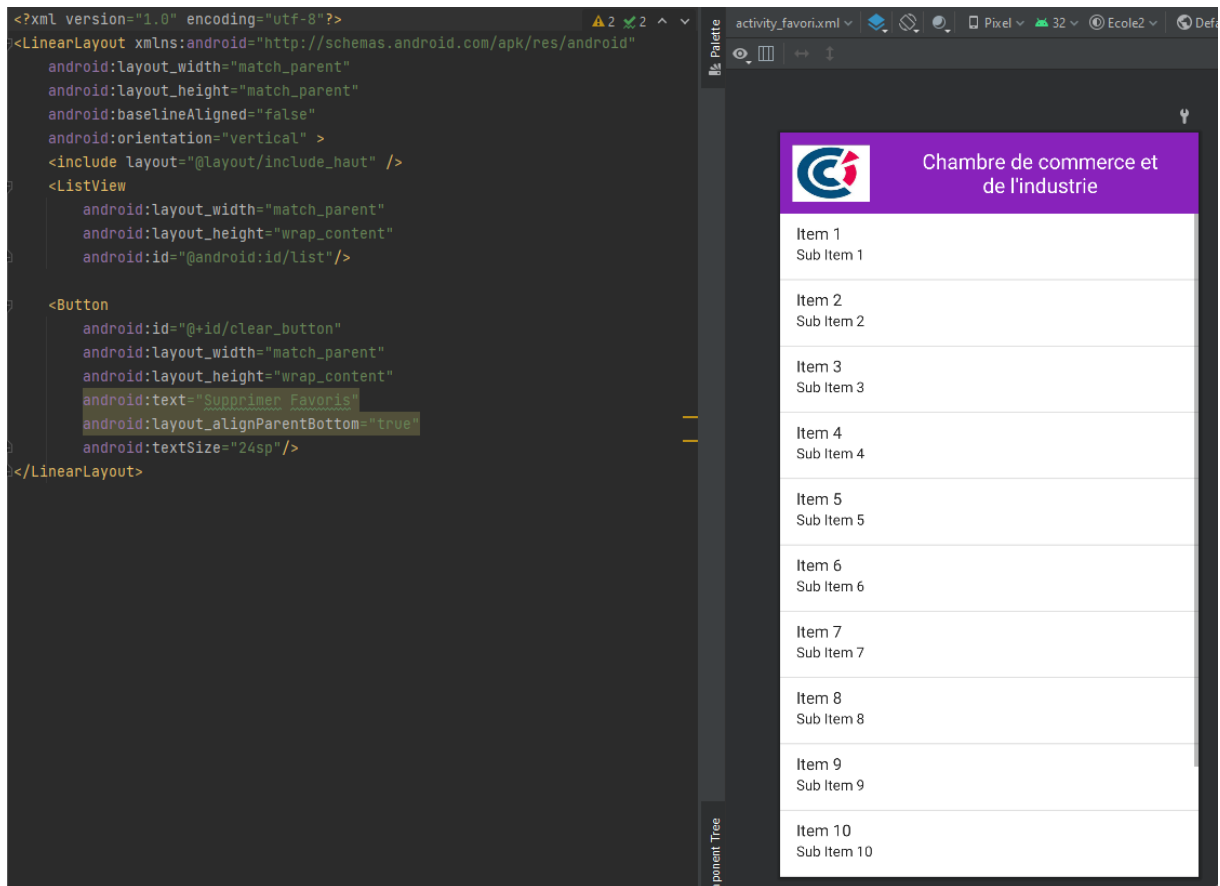
    public DatabaseOpenHelper(Context context) {
        super(context, NAME, factory: null, VERSION);
        this.mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) { db.execSQL(CREATE_CMD); }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // N/A
    }

    public void deleteDatabase() { mContext.deleteDatabase(NAME); }
}
```

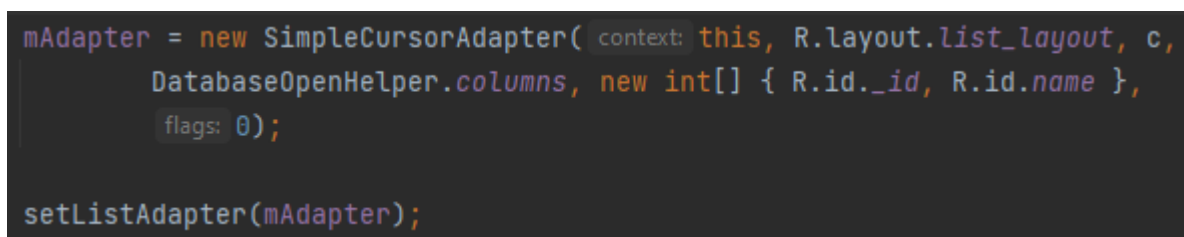
Ensuite, créer une activité qui va contenir la liste des favoris et un bouton pour les supprimer selon le layout suivant :



Dans l'activité, on utilise un curseur, c'est l'objet qui va contenir tous les éléments de la table favoris :



L'intérêt est d'utiliser un adaptateur qui va créer la liste selon les données du curseur choisi :



Enfin, pour la suppression :

```
Button clearButton = (Button) findViewById(R.id.clear_button);
clearButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        // execute database operations
        clearAll();

        // Redisplay data
        mAdapter.setCursor();
        mAdapter.notifyDataSetChanged();
    }
});
```

```
// Delete all records
private void clearAll() {
    mDbHelper.getWritableDatabase().delete(DatabaseOpenHelper.TABLE_NAME, whereClause: null, whereArgs: null);
}
```

Pour ajouter des formations, un bouton a été ajouté à l'activité FormationActivity :

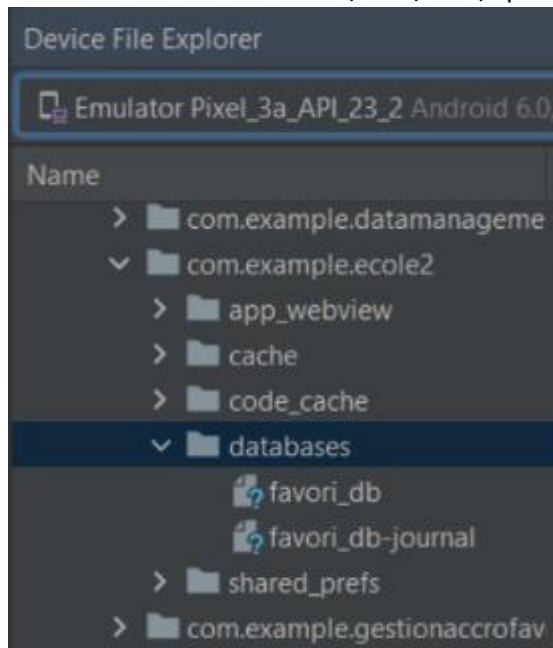


Quand on clique sur ce bouton, cela va ajouter la formation a la base de données :

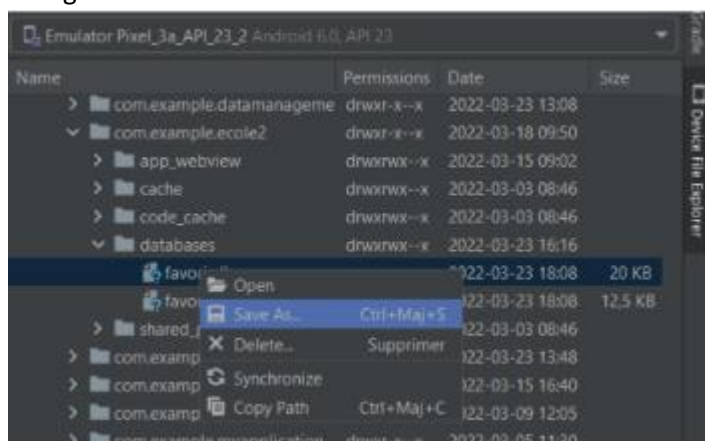
```
ImageView imgFavori= (ImageView) findViewById(R.id.formationFavoriId);
imgFavori.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.i(TAG, msg: "Bouton favori");
        ContentValues values = new ContentValues();
        values.put(DatabaseOpenHelper.FORMATION_NAME, formation.getTitre());
        dbHelper.getWritableDatabase().insert(DatabaseOpenHelper.TABLE_NAME, nullColumnHack: null, values);
        Toast.makeText(getApplicationContext(), text: "Formation ajoutée aux favoris", Toast.LENGTH_LONG).show();
    }
});
}
```

Il est possible de contrôler que la base de données est bien enregistrée dans le téléphone :

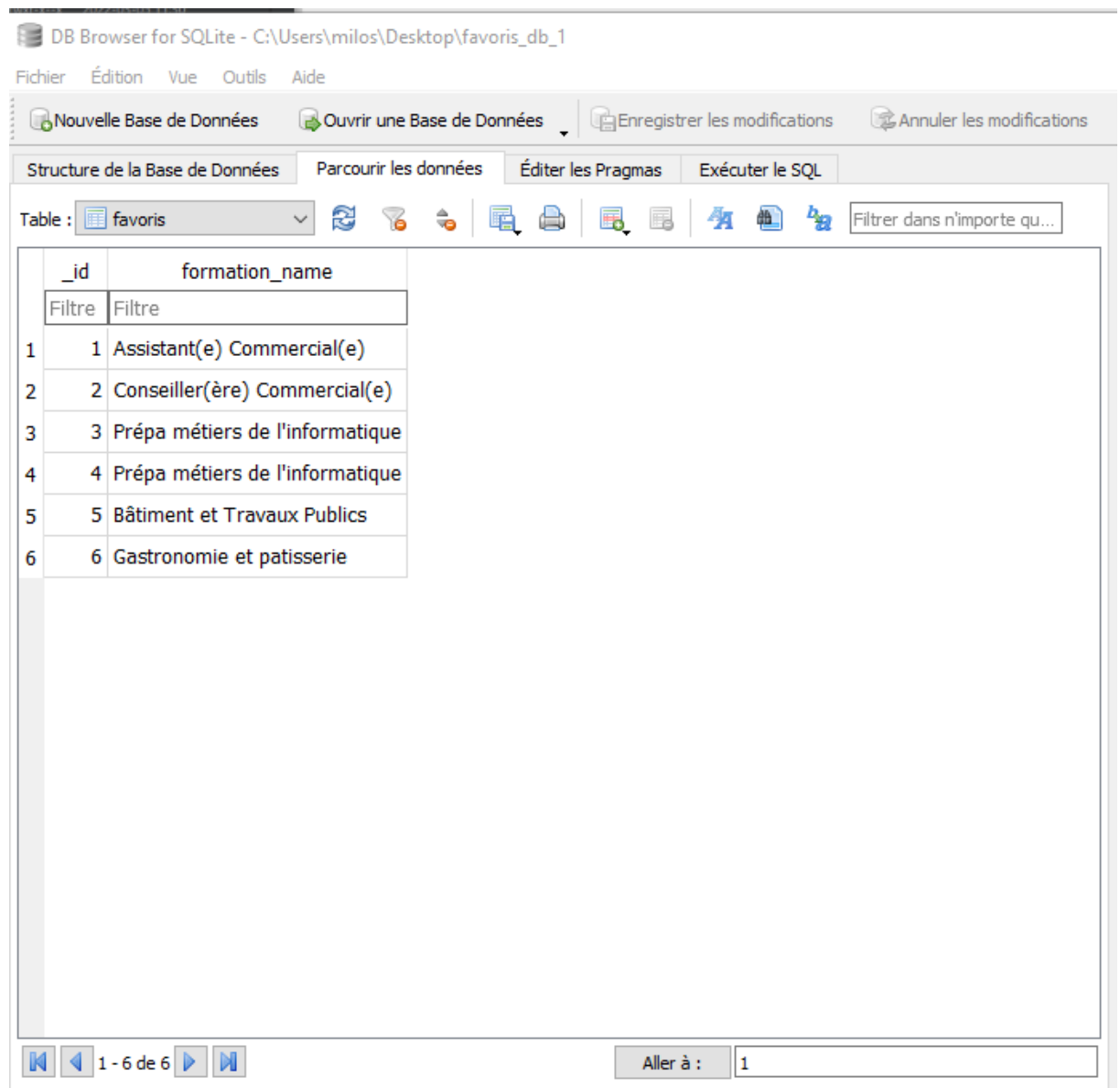
- Sous Android Studio, cliquez à droite sur "Device File Explorer"
- Base de données stockée en /data/data/<package name>/databases/



- Enregistrer la bdd



- Et avec l'outil DB browser for SQLite, lire la table « favoris » :



Déploiement

Afin de nous former au déploiement d'application sur un serveur distant, il nous a été proposé de déployer l'api et la bdd sur Ubuntu.

Toutes les informations sont en annexe.

Conclusion

Pour conclure, je pense avoir beaucoup appris avec ce projet. C'est la première fois que je code avec Android Studio et je trouve ça assez plaisant. Les cours précédents m'ont permis de ne pas trop avoir de soucis pour comprendre le code et l'architecture MVC aussi. Au départ j'ai eu un peu de mal avec la couche model, pour interagir avec l'API, mais avec l'aide du professeur et de quelques recherches j'ai réussi à comprendre son fonctionnement. Pour le reste, j'ai l'habitude d'utiliser GitHub et des outils de gestion de projet en entreprise donc je n'ai pas eu de problème avec ça. Finalement, cette application mobile, même si elle n'est pas complète, m'a permis d'acquérir beaucoup de connaissances en développement d'application Android. Je pense donc maintenant être capable d'en créer une moi-même, avec du temps et des recherches. J'espère qu'à terme de mes années d'études, j'en aurai développé une ou plusieurs, avec de meilleurs visuels et plus de fluidité et, pourquoi pas, aussi avec d'autres Framework ou d'autre langages.

Compétences validées

Développer la présence en ligne de l'organisation	Travailler en mode projet	Mettre à disposition des utilisateurs un service informatique
<ul style="list-style-type: none"> ▸ Participer à la valorisation de l'image de l'organisation sur les médias numériques en tenant compte du cadre juridique et des enjeux économiques ▸ Référencer les services en ligne de l'organisation et mesurer leur visibilité. ▸ Participer à l'évolution d'un site Web exploitant les données de l'organisation. 	<ul style="list-style-type: none"> ▸ Analyser les objectifs et les modalités d'organisation d'un projet ▸ Planifier les activités ▸ Évaluer les indicateurs de suivi d'un projet et analyser les écarts 	<ul style="list-style-type: none"> ▸ Réaliser les tests d'intégration et d'acceptation d'un service ▸ Déployer un service ▸ Accompagner les utilisateurs dans la mise en place d'un service
X	X	X