



# Compte rendu d'activité

Application mobile d'accès aux auto-formations de  
MediaTek 86

24/01/2022

## Table des matières

Compte rendu d'activité.....	1
Contexte.....	4
InfoTech Services 86.....	4
Contexte InfoTech Services 86.....	4
MediaTek86.....	4
Contexte MediaTek86.....	4
Projet DigimediaTek86.....	4
Contexte projet DigimediaTek86.....	4
Application mobile MediaTek86.....	5
Contexte et objectifs de l'application mobile MediaTek86.....	5
Rappel de l'existant.....	5
La base de données.....	5
L'API.....	6
Les fonctionnalités existantes.....	6
Les classes existantes.....	7
Package contrôleur.....	7
Package modele.....	8
Package outils.....	8
Package vue.....	8
Expressions des besoins.....	9
Fonctionnalité de filtrage des formations.....	9
Fonctionnalité de gestion des favoris.....	9
Tests unitaires et scénario de test.....	9
Informations techniques et outils utilisés.....	10
Spécificités IDE et émulateurs.....	10
Spécificités outils et plugins.....	10
Tests unitaires et scénario.....	11
Documentation technique.....	11
Suivi de projet.....	11
Réalisation.....	12
Préparation de l'environnement de travail.....	12
Établir le plan de suivi sur la plateforme Trello.....	12
Récupération de la documentations.....	15
Mise en place de l'environnement de développement.....	15
Configuration de WampServer.....	15
Importation de la base de donnée.....	17
Mise en place de l'IDE Android ArcticFox.....	19
Récupération du code source et configuration de l'émulateur.....	20
Création du dépôt sur Github.....	23
Gestion des filtres.....	24
Suivi de projet.....	24
Création dans le controleur.....	25
getLesFormationsFiltre().....	25
Modifications dans la vue.....	26
ecouteFiltre().....	26
creeListe().....	27
init().....	27

Dépot github.....	28
Gestion des favoris.....	29
Suivi de projet.....	29
Ajout de la classe technique MySQLiteOpenHelper.....	30
Ajout de la classe AccesLocal.....	32
recup().....	33
ajoutFavoris().....	34
removeFavoris().....	34
Modifications dans le contrôleur.....	35
getInstance().....	35
checklesFavoris().....	36
ajoutFavoris().....	37
removeFav().....	37
Gestion de l'Activity FormationsActivity.....	38
Gestion des clics sur btnFavoris.....	38
Mise en place de la page des favoris.....	43
Dans la vue MainActivity.....	43
Dans le controleur.....	44
Dépot github.....	46
Qualité du code.....	47
Suivi de projet.....	47
Tests unitaires.....	48
FormationTest.....	48
MesOutilsTest.....	50
Scénario.....	52
Contrôler la qualité du code avec SonarLint et SonarQube.....	53
SonarLint.....	53
Mise en place de SonarQube.....	54
Doctequine.....	58
Déploiement.....	58
Heroku.....	58
Mise en place composer.....	59
ClearDB MySQL.....	60
Heroku mode sommeil.....	63
Bilan sur les objectifs atteints.....	64
Finalités.....	64
Liste des compétences couvertes (B1, B2, B3).....	65
B1 Gérer le patrimoine informatique.....	65
B1 Répondre aux incidents et aux demandes d'assistance et d'évolution.....	65
B1 Travailler en mode projet.....	65
B1 Mettre à disposition des utilisateurs un service informatique.....	65
B1 Organiser son développement professionnel.....	65
B2 Concevoir et développer une solution applicative.....	65
B2 Assurer la maintenance corrective ou évolutive d'une solution applicative.....	65
B2 Gérer les données.....	66
B3 Assurer la cybersécurité d'une solution applicative et de son développement.....	66

# Contexte

## InfoTech Services 86

### Contexte InfoTech Services 86

InfoTech Services 86 (**ITS 86**) est une Entreprise de Services Numériques (ESN) spécialisée dans le développement informatique d'applications bureau, web ou mobile, dans 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 grâce à son pôle Développement qui permet de proposer des solutions d'hébergements ainsi qu'une expertise d'intégration de services, de développement de logiciel ou encore de gestion de bases de données.

## MediaTek86

### Contexte MediaTek86

Parmi les clients d'**ITS 86** se trouve le réseau de médiathèques de la Vienne, MediaTek86 dont la gestion du parc informatique et la numérisation des activités internes du réseau dépendent des services d'**ITS 86**. Dans l'optique d'accroître l'attractivité de son réseau de médiathèques, **MediaTek86** veut développer et enrichir ses services notamment par l'intermédiaire d'un accès à des cours et d'auto-formations en ligne. Ces projets numériques sont pilotés par un chef de projet numérique chargé de la maîtrise d'ouvrage (MOA) auprès des ESN auxquelles **MediaTek86** a fait appel.

## Projet DigimediaTek86

### Contexte projet DigimediaTek86

Le projet DigimediaTek86 fait parti de cette liste de projet à mettre en place. L'objectif de ce projet s'étend sur deux axes, l'un interne puisqu'il implique la mise à disposition de services numériques pour ses employés, le second orienté vers ses usagers. Ici nous nous intéresseront à cette seconde partie puisque ce rapport détaille de la mise en place des nouvelles fonctionnalités dans l'application mobile de MediaTek86 dont la mise en place s'inscrit dans la volonté de fournir des formations au numérique en ligne.

**Il est à noter qu'un document détaillant davantage les différents éléments du présent contexte est accessible à ce lien sous le nom de [Contexte\\_MediaTek86.pdf](#)**

# Application mobile MediaTek86

## Contexte et objectifs de l'application mobile MediaTek86

L'application mobile de MediaTek86 est une plateforme mobile sous Android écrite en Java. Son objectif est de permettre à ses utilisateurs d'avoir accès à des vidéos d'auto-formation mise en ligne via la plateforme **Youtube**. L'application permet ainsi de rassembler ces formations via un même portail mobile. De plus, l'utilisateur doit pouvoir les trier selon le titre des formations afin d'accéder plus rapidement à la vidéo désirée. Dans cette même optique, un bouton à côté de chaque formation doit permettre de marquer comme favoris ces vidéos. Enfin, un bref descriptif contenant les différentes étapes(ou chapitres) de la vidéo doit être mis à disposition après avoir sélectionné une formation. Les données de ces formations doivent être stockées dans une base de données mise en ligne et une base de données locale doit permettre de mémoriser les formations marquées comme favorites afin de pouvoir les récupérer à l'ouverture de l'application. Enfin, la base de données distante doit être accessible via une API REST.

## Rappel de l'existant

### La base de données

Pour accéder à la base de données, il faut récupérer le script de création de la base de données puis l'importer dans un logiciel de gestion de base de données sous **MySQL** permettant ainsi une gestion en local avec **phpMyAdmin**. Cette base de données contient deux tables, la première « **doctrine\_migration\_version** » possédant les données de migration de version pour doctrine dont nous n'auront pas l'utilité ici et la seconde, « **formations** » détenant les informations sur les formations qui sont :

- **id** : l'id de la formation en auto incrémentation qui est ici définie comme clé primaire
- **published\_at** : la date de publication
- **title** : le titre de formation
- **description** : la description de la vidéo
- **miniature** : l'adresse internet de l'image de présentation de la vidéo au format 120\*90
- **picture**: l'adresse internet de l'image de présentation de la vidéo au format 640\*480
- **video\_id** : l'id de la vidéo sur Youtube.

De plus, il est à noter que l'id est au format int, la date de parution au format Date non nulle et les autres sont de chaînes de type String. Le script de création de cette base de donnée est accessible via le lien suivant sous le nom [script\\_sql\\_mediatek86formations.zip](#)

## L'API

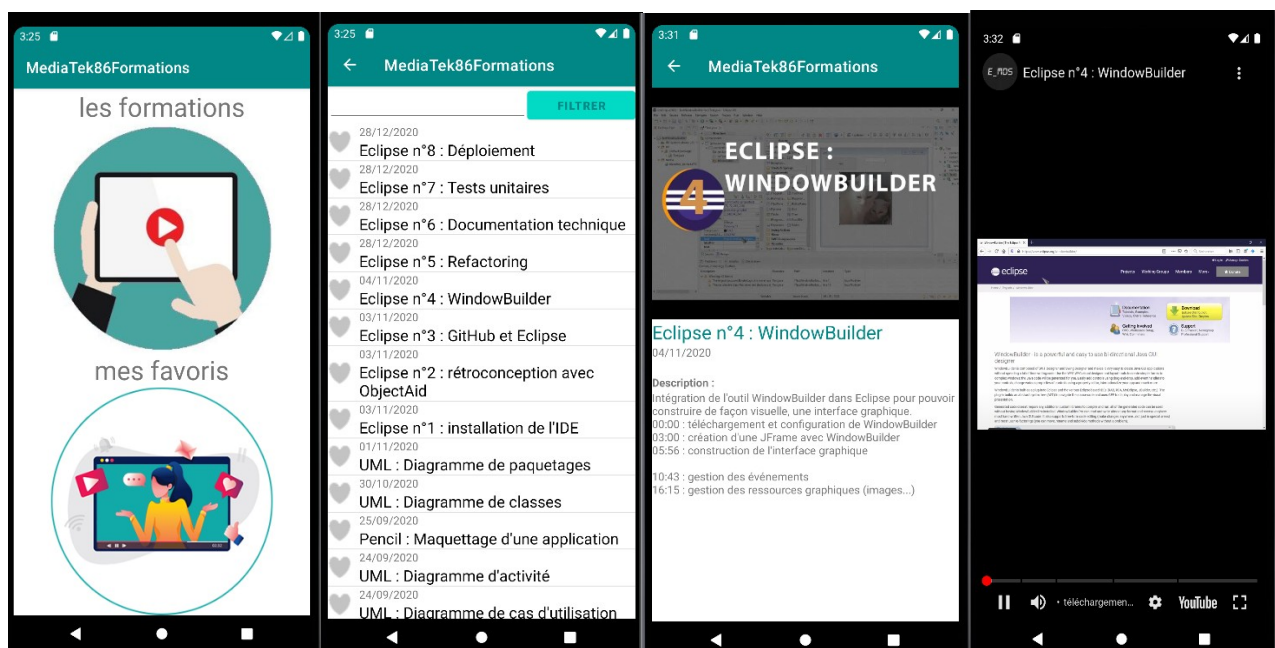
L'API REST est déjà fournis et écrite en PHP et est composée de 5 fichiers :

- **.htaccess** : fichier qui contient la route définie pour récupérer les informations du serveurs
- **mediatek86formation.php** : fichier d'entrée dans l'API permettant ainsi de récupérer les variables en GET et d'appeler les méthodes de Controle.php.
- **Controle.php** : fichier qui sollicite AccessBDD pour exécuter les demandes de récupération de données et retourne ses informations au format JSON.
- **AccesBDD.php** : fichier qui prépare les requêtes puis demande à ConnexionPDO.php de les exécuter.
- **ConnexionPDO.php** : fichier qui se connecte à la base de donnée puis exécute les requetes .

Bien que l'application n'utilise que le verbe « GET », l'API permet de gérer les 4 verbes http standards. L'API fournis est trouvable dans ce dossier sous le nom de [rest\\_mediatek86formations.zip](#)

## Les fonctionnalités existantes

A l'ouverture, l'application accède via l'API à la base de donnée distante permettant alors de consulter la liste des titres des formations mises à dispositions. La page d'accueil permet de choisir entre deux boutons, le premier permet d'accéder à la liste de toutes les formations, le second qui n'est pas fonctionnel permettra d'accéder à la liste des formations désignées comme favorites. Dans l'activité des formations, en sélectionnant un titre dans la liste, une nouvelle activité s'ouvre offrant alors davantage de détails sur la dite formation. Un nouveau clic sur l'image permet ensuite d'accéder à la page d'ouverture de la vidéo depuis son adresse dont l'affichage peut être pivoté à l'horizontale pour un meilleur confort de lecture. Il y a donc 4 pages d'activités dont le design en layout est déjà mis en place et chacune dotée un bouton permet la navigation de retour d'une activité fille à une autre. Étant en développement Android, la liste des formations s'effectue par l'intermédiaire de listes interactives. Aussi, le bouton en forme de coeur pour marquer en favoris bien qu'affiché cette fonctionnalité n'est pas non plus implémentés. Finalement, nous obtenons les figures qui sont les images des différentes pages existantes :



Page 1 accueil

Page 2 les  
formations

Page 3 une formation

Page 4 une vidéo

Enfin, les différents documents (contextes détaillés, script SQL, API, documentations) sont mis à dispositions à l'adresse suivante : [documentations](#). De même, le code source de l'application d'origine est accessible ici : [code source](#).

## Les classes existantes

Le code source fourni contient 4 packages qui sont les suivants : **contrôleur**, **modele**, **outils** et **vue**.

### Package contrôleur

Le package **contrôleur** contient la classe **Contrôle.java** qui est la classe centrale de l'application. Elle permet de créer une instance unique de la classe **Contrôle** ainsi que d'appeler la Classe **AccesDistant.java** du package **modele** permettant de récupérer les données de la base distante dans un ArrayList.

Son contenu à l'état d'origine est visible [ici](#)

### Package modele

Ce package contient deux classes **AccesDistant.java** et **Formation.java**. La première implémente la classe **AsyncResponse** qui permet de mettre en place un thread où la classe récupère les informations de la base de donnée de manière indépendante vis à vis de l'application. La classe **AccesDistant** possède deux méthodes la première pour envoyer la requête à l'**API** et la seconde pour en récupérer la réponse. Aussi, elle contient l'adresse du serveur distant. Concernant la classe **Formation**, il s'agit d'une classe métier permettant de créer un objet Formation avec l'ensemble de ses propriétés privées qui sont valorisées avec les données envoyées depuis la base de données par l'API. Cette classe implémente la classe Comparable permettant un tri des formations sur la date de publication.

Le code source de ces deux classes sont retrouvables dans les liens suivants : [Formation](#) et [AccesDistant](#)

### Package outils

Ce package contient deux interfaces nommées **MesOutils** et **AsyncResponse**, ainsi qu'une classe **AccesREST** qui hérite d'**AsyncTask**. La première interface **MesOutils** comporte un ensemble de méthodes réutilisables permettant de convertir un **objet** de type Date en String selon un pattern précis et inversement. Aussi, cette interface contient la méthode **loadMapPreview()** gérant l'affichage d'images issues d'une adresse internet. L'interface **AsyncTask** permet à la classe **AccesDistant** de récupérer la méthode **processFinish()** permettant de récupérer la réponse du serveur distant. Enfin, **AccesRest.java** permet de créer la requête envoyée à l'API dans un thread indépendant.

Les différents codes sources sont visibles vers les liens suivants : [AsyncResponse](#), [AccesREST](#) et [MesOutils](#).

### Package vue

Ce package contient les cinq classes déterminants les différentes vues de l'application. La première **MainActivity** est la classe d'accueil, elle permet de choisir entre deux options le clic sur les formations ou sur les favoris, bien que non implémentés. La vue **FormationsActivity** est la classe permettant d'afficher la liste des formations récupérées par la classe **Controle**. Elle fonctionne en collaboration avec la classe **FormationListAdapter** puisqu'elle nécessite l'usage d'une liste adaptative. Au clic d'une de ces formations, la classe **uneFormationActivity** est appelée permettant alors l'affichage du détail d'une formation. Enfin, au clic sur l'image de cette vue, la classe **VideoActivity** est appelée permettant alors de visionner la vidéo de formation.



Les différents codes sources de ces classe sont accessibles depuis les liens suivants : [MainActivity](#), [FormationsActivity](#), [FormationListAdapter](#), [UneFormationActivity](#) et [VideoActivity](#)

## Expressions des besoins

### Fonctionnalité de filtrage des formations

Afin de permettre un accès rapide et aisé à la formation désirée aux utilisateurs, une nouvelle fonctionnalité permettant de filtrer sur le titre de la formation doit être implémentée, ainsi seules les formations contenant le mot ou le morceau de mot inséré sans être non plus sensible à la case. A cette effet, un bouton « **filtrer** » avait déjà été mis en place bien qu'il soit inactif. Si la zone de texte pour le filtre est vide lorsqu'un clic sur le bouton filtrer alors l'ensemble des formations sont affichées.

### Fonctionnalité de gestion des favoris

Cette fonctionnalité a pour intérêt de permettre d'accéder plus rapidement à un ensemble de formations choisis par l'utilisateur. Lorsqu'un utilisateur clic sur un coeur gris dans la liste de formations, celui-ci doit devenir rouge le marquant alors comme favoris. A l'inverse, un clic sur un coeur rouge le fera devenir gris le supprimant donc des favoris. Par conséquent, les formations considérées comme favorites seront listées dans la liste des favoris qui sera accessible depuis la page d'accueil en cliquant sur le bouton favoris. Seules les formations avec un coeur rouge, apparaîtront dans cette liste. Néanmoins, il sera toujours possible de supprimer de cette liste ces formations en cliquant sur le coeur rouge. Enfin, cliquer sur l'une des formation présente dans cette liste aura le même effet que dans la liste de toute les formations : une nouvelle page s'ouvre afficher le détail de la vidéo et un clic sur l'image de la vidéo permet de la lancer.

Afin d'assurer la sauvegarde de cette liste de favoris, une base de donnée locale sous SQLite doit être mise en place. La dite base de données ne doit contenir que l'id de la formation.

### Tests unitaires et scénario de test

Afin d'assurer la non régression dans le développement de l'application actuelle et future et l'usage convenable de l'application, des tests unitaires sur la conversion des dates de parutions doivent être mises en place. De même, un scénario de test des différentes fonctionnalités de l'application doivent être établies pour les mêmes raisons.

## Informations techniques et outils utilisés

### Spécificités IDE et émulateurs

L'évolution de l'application récupérée a été effectuée avec la version ArcticFox 2020.3.1 de l'IDE d'Android Studio et est associé à l'outil de compilation **Gradle** dans sa version 7.0.4. Concernant le SDK sélectionné, il s'agit de la version Android 11.0 R et la version de l'émulateur Android visée est l'API 30 afin d'être testé sur émulateur Pixel 3a API 30. Aussi, l'application a été testé sur un appareil Android réel de la marque Samsung Galaxy.

### Spécificités outils et plugins

La revue du code produit a été effectuée par l'intermédiaire de **SonarLint** qui est un plugin permettant d'analyser du code dans l'IDE. Ainsi, il permet de mettre en évidence les éventuelles vulnérabilités et bogues lors de l'écriture du code comme le montre la [figure 1](#).

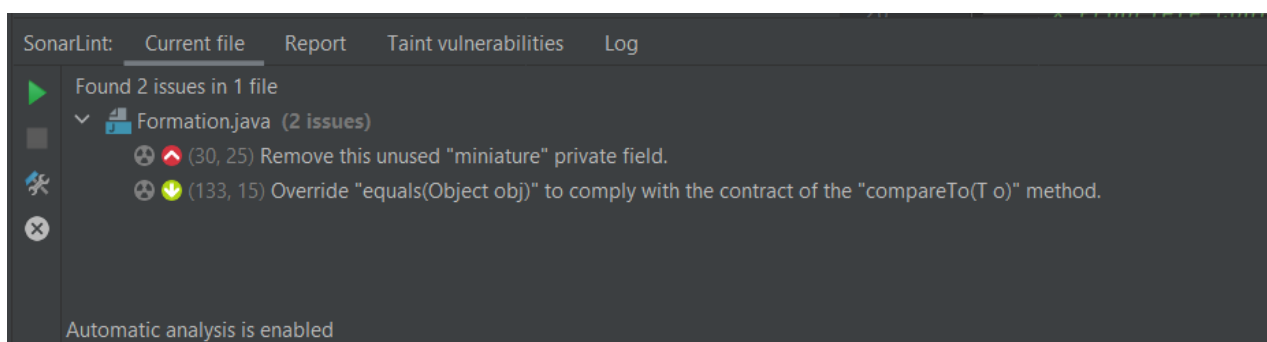


Figure 1: Exemple de contre rendu d'analyse de code avec SonarLint

De même, le logiciel [SonarQube](#) a été intégré au projet afin de mesurer la qualité du code. Il est capable de recenser plusieurs niveaux de bogues ou d'erreurs tels que le niveau de documentation ou couverture de test, de repérer les duplications de code et le non respect de règles de programmation ainsi que la complexité du code produit. Enfin, il permet d'évaluer sous plusieurs notes comme le montre le [figure 2](#) ci-dessous :



Figure 2: Exemple de résultat d'évaluation par SonarQube

Enfin, la clarté de l'indentation du code a été révisé avec le plugin [CheckStyle](#) qui est un logiciel de contrôle de code permet de vérifier le style d'un code produit dont certaines erreurs de style ne sont pas prises en compte par l'auto-indentation de l'IDE, comme la

présence de commentaires Javadoc. Ici cette analyse s'est effectuée selon les règles « Sun Checks », comme le montre la *figure 3*.

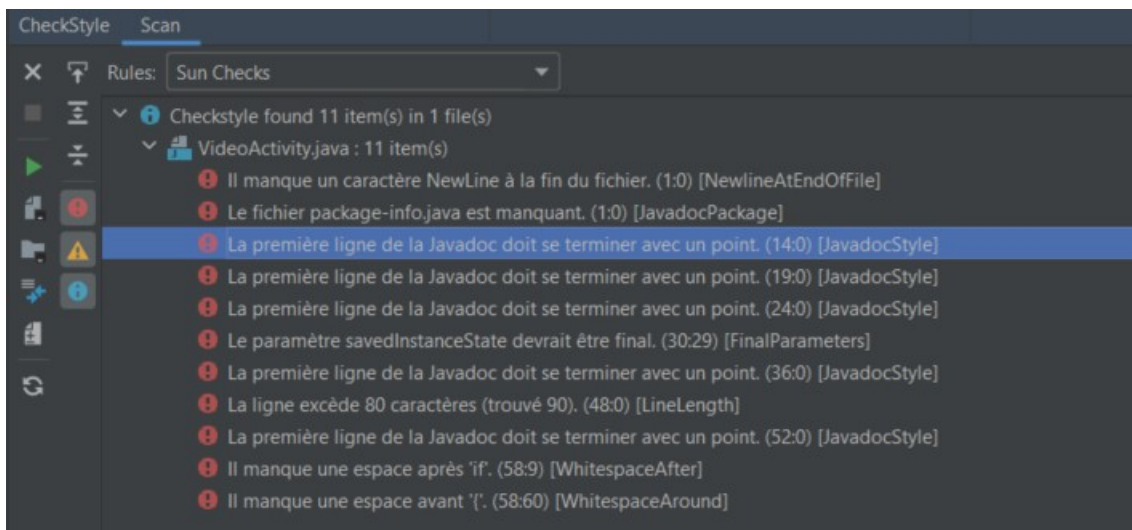


Figure 3: Exemple d'alerte de style de code par CheckStyle

### Tests unitaires et scénario

Les tests et le scénario ont été établis avec Junit 4 ainsi qu'avec les dépendances implémentées d'Android espresso pour les tests automatisés de l'interface, puisque ce dernier permet de simuler les interactions utilisateurs.

### Documentation technique

La documentation technique des diverses classes, propriétés et méthodes établis lors de la poursuite du développement de l'application mobile a été établie avec [Doxygen](#) qui est un logiciel sous libre licence permettant de générer une documentation technique standard. Ce dernier a été choisi pour ses fonctionnalités de personnalisations. Cette documentation est accessible en ligne à l'adresse suivante : [Documentation technique de l'application mobile](#).

### Suivi de projet

Le suivi de projet a été établi avec la plateforme [Trello](#) qui permet d'établir un plan d'évolution et de suivi de projet. Enfin, l'intégralité des codes sources produits ont été stockés sur la plateforme [Github](#) au nom de projet [Android\\_MediaTek86Formations](#) enfin le Portfolio présentant l'ensemble de ce projet est accessible [ici](#).

# Réalisation

## Préparation de l'environnement de travail

### Établir le plan de suivi sur la plateforme Trello

La mise en place du suivi de projet a commencé par la création d'un tableau public sur Trello du nom d'[androidmediatek](#). 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 à noter que la couleur violet a été choisie afin de pouvoir être distinguable du vert qui est automatiquement sélectionné pour le respect des dates et l'icône des check-lists remplies.

La [figure 4](#) illustre un exemple de contenu d'une de ces listes et la [figure 5](#) montre les différents menus de création.

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 tâche 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** » a 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.

Enfin, le dépôt sur GitHub a été lié à ce suivi de projet permettant d'obtenir en tant que PowerUp, qui sont des plugins permettant d'ajouter des fonctionnalités supplémentaires au tableau. Concernant le lien entre le tableau et le dépôt GitHub, il permet d'abord d'établir un lien constant vers le projet, mais aussi de faire apparaître le message du

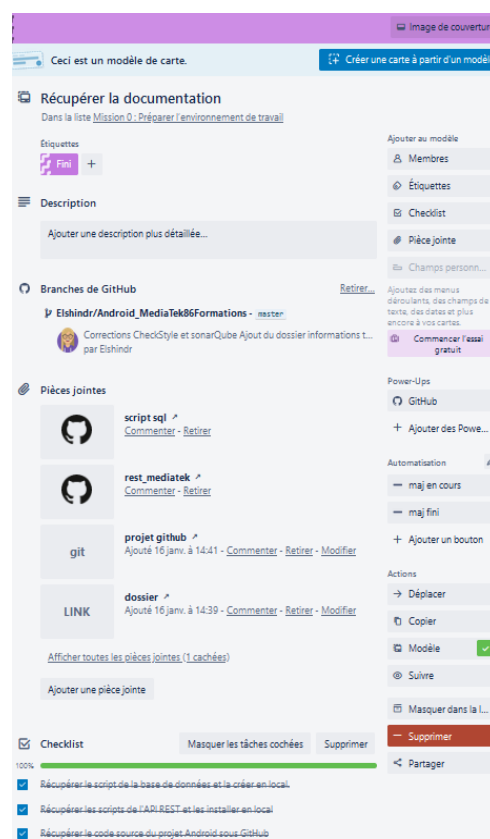


Figure 4: Exemples de carte de suivi avec étiquette, dates de fin et check-list

dernier commit, renseignant ainsi de l'avancé du projet. Pour finir, ce tableau Trello étant publique, il est consultable à [l'adresse suivante](#).

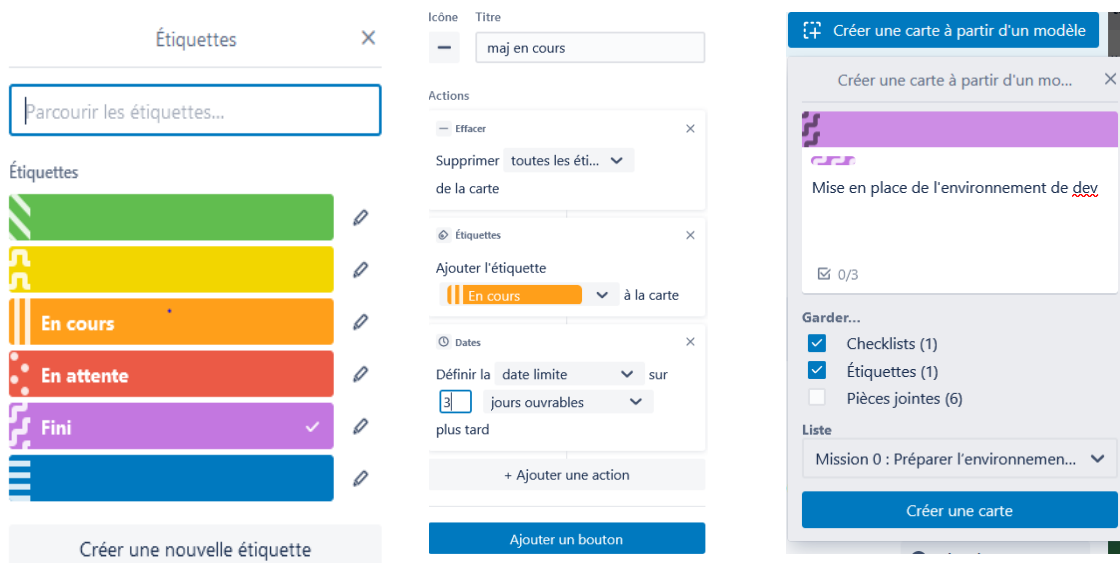


Figure 5: Menus de création d'étiquettes et des boutons d'automatisation

Il est ainsi possible d'énumérer les différentes missions :

- **Mission 0 : Préparer l'environnement de travail – temps estimé 2h**

- Mise en place du tableau sur Trello
- Récupération de la documentation
  - Script SQL de la base de données
  - L'API REST
  - Le code source de l'application à faire évoluer
  - Le dossier de l'existant et des missions à effectuer
  - Le dossier des informations techniques.
- Mise en place de l'environnement de développement
  - Installation de WampServer et configuration de l'API
  - Création de la base de donnée distantes dans PhpMyAdmin
  - Installation de l'IDE Android Arctic Fox
  - Tester le code source avec l'émulateur et un appareil physique
  - Création du dépôt sur Github

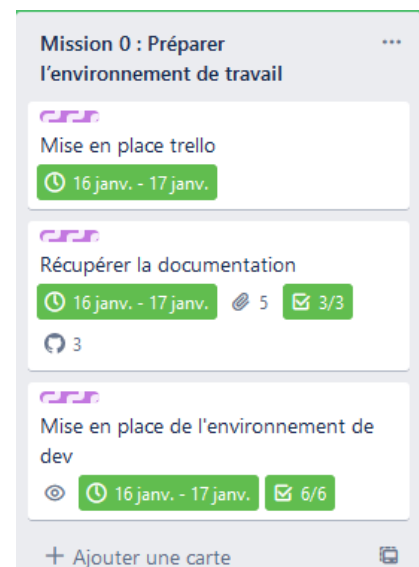


Figure 6: Liste des cartes de la mission 0 préparer l'environnement de travail

- **Mission 1 : Gérer les filtres des formations – temps estimé 2h**
  - **Dans la Classe Controle**
    - Création d'une méthode qui retourne la liste ne contenant que les formations dont le titre contient le filtre reçu en paramètre
    - Ne pas tenir compte de la case
  - **Dans la vue qui contient le filtre**
    - Création d'une méthode événementielle sur le clic du bouton "filtrer" qui vérifie si la zone de saisie est vide ou non pour savoir quelle méthode appeler dans le controleur, pour valoriser la liste de formations à utiliser pour l'affichage.
- **Mission 2 : Gérer les favoris – temps estimé 10h**
  - **Gestion de la base de données locale**
    - Ajouter les classes technique AccesLocal et MySQLiteOpenHelper fournies
    - Création de la base de donnée locale qui mémorise les id des formations marquées comme favorites
    - Vérifier que la suppression d'une formation dans la base de données distante se répercute aussi dans la base de donnée locale.
  - **Gestion de l'Activity FormationsActivity**
    - Gestion des clics sur le bouton des favoris dans l'activity FormationsActivity.
    - Gestions de l'ajout et de la suppression des id dans la base de données de données locale au clic du bouton des favoris
  - **Création de la page des favoris**
    - Gestion du clic sur le bouton des favoris pour supprimer la formation concernée de la liste et la réactualiser
    - Mise en place des mêmes fonctionnalités que pour la page des formations
- **Mission 3: Qualité, tests et documentation technique – temps estimé 4h**
  - Contrôler la qualité du code avec SonarLint
  - Création des tests unitaires pour la conversion des dates
  - Création d'un scénario de tests pour les différentes manipulations de l'interface
  - Création de la documentation technique de l'application
- **Mission 4: Création d'une vidéo de démonstration via OBS – temps estimé 1h**
- **Mission Bilan**
  - Déployer l'application mobile sous apk et la mettre à disposition sur la plateforme de dépôt.
  - Mise en ligne de la base de donnée distante et de l'API
  - Rédaction du contre rendu d'activité.
  - Mise en place du portfolio de l'activité.

La **figure 7** suivante illustre ce tableau de suivi à la fin de son élaboration, soit à la Mission 0.



Figure 7: Tableau de suivi de projet Trello, début de la mission 0

## Récupération de la documentations

L'ensemble de la documentation a été télécharger pour être stockée en local, puis mise à disposition sur le suivi de projet Trello permettant alors de pouvoir y accéder avec facilité. Aussi, elle est récupérable sur le dépôt [git du projet](#), quand au code source de l'application d'origine, il est accessible ici [code source](#).

## Mise en place de l'environnement de développement

### Configuration de WampServer

La première étape dans cette mise en place a été de placer le contenu du dossier `rest_mediatek86formations`, comportant l'API REST, dans le dossier `C://wamp64/www/rest_mediatek86formations`, cf **figure 8**, les fichiers `htaccess`, `mediatek86formation.php`, `Controle.php`, `AccesBDD.php` et `ConnexionPDO.php`.

wamp64 > www > rest_mediatek86formations				
Rechercher dans : rest_mediatek86formations				
	Nom	Modifié le	Type	Taille
	.htaccess	09/12/2021 15:20	Fichier HTACCESS	
	AccessBDD.php	09/12/2021 15:17	Fichier PHP	
	ConnexionPDO.php	30/10/2021 15:57	Fichier PHP	
	Controle.php	16/01/2022 17:30	Fichier PHP	
	mediatek86formations.php	09/12/2021 15:20	Fichier PHP	

Figure 8: dossier de mise en place de l'api en serveur en local



Ce dossier est l'emplacement des hôtes virtuels de la plateforme de développement web WampServer (Windows, Apache, MySQL et PHP). La version de PHP utilisée est la 7.3.21, la version de MySQL 5.7.31 et le port d'écoute est celui par défaut 3006. Aussi, il est possible de noter que ce poste utilise le système d'exploitation Windows 10 et qu'ici c'est la version [wampserver](#) 64bits qui est utilisée.

D'ailleurs ce logiciel étant déjà installé sur le poste de travail, sa mise en route sera détaillé brièvement. Le téléchargement de son exécutable permet une installation rapide de ce logiciel. A son exécution, plusieurs fenêtres d'invite de commande apparaissent brièvement permettant l'ouverture du serveur local et de phpMyadmin. Il suffit de cliquer sur l'onglet des icônes à droite dans la barre des tâches Windows, [figure 9](#), pour avoir apparaître l'icône verte de **WampServer** indiquant ainsi que le serveur s'est correctement ouvert. Il faut alors aller dans « Vos VirtualHosts » puis « Gestion VirtualHost ». La [figure 10](#) s'ouvre alors dans le navigateur par défaut, dans notre cas il s'agit de **Modzilla Firefox**. Il s'agit d'un formulaire de création de virtual Host, il convient alors de renseigner un nom de VirtualHost, ici « mediatek86formations », et l'adresse du dossier à virtualiser, soit « C://wamp64/www/rest\_mediatek86formations ». Une fenêtre de validation s'ouvre ensuite indiquant que l'hôte virtuel a bien été créée mais qu'il faut redémarrer les services. Pour cela il faut à nouveau cliquer sur l'icône dans wampserver dans les icônes de la barre des taches puis sélectionner « redémarrer les services ».

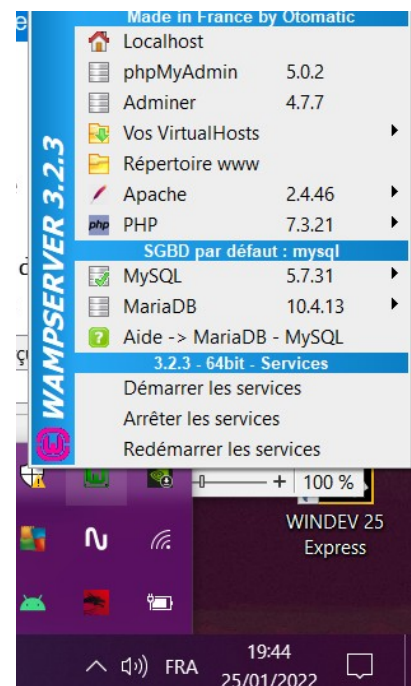


Figure 9 Onglet d'options WampServer

Nom du  Pas d'espace - Pas de tiret bas (\_) Requis

---

Chemin complet absolu du  VirtualHost - Exemples : C:/wamp/www/projet/ ou E:/www/site1/ Requis

---

Figure 10: Formulaire de création de virtual host



### Importation de la base de donnée

De retour à la barre des tâches de Windows, il suffit d'ouvrir à nouveau le menu de WampServer. Il faut alors cliquer sur **phpMyAdmin**. A cet instant, une nouvelle page du navigateur s'ouvre pour afficher un nouveau formulaire, cf [figure 11](#). Il s'agit de la page de connexion à phpMyAdmin, il faut alors remplir en tant qu'utilisateur : « root » et laisser vide pour le mot de passe, à moins que ce dernier ait été configuré.

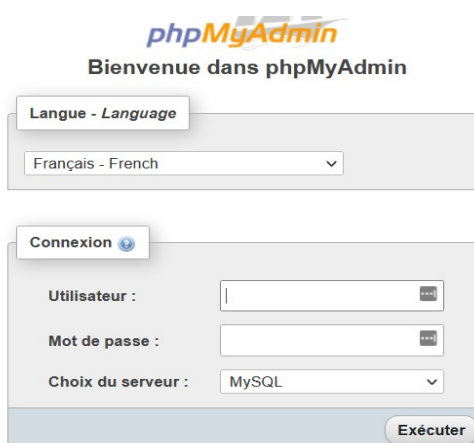


Figure 11: Fenêtre de connexion à phpMyAdmin

En arrivant dans le menu de phpMyAdmin, il faut alors aller dans l'onglet « importer » qui affiche alors la [figure 12](#).

### Importation dans le serveur courant

#### Fichier à importer :

Le fichier peut être compressé (gzip, bzip2, zip) ou non.

Le nom du fichier compressé doit se terminer par **.[format].[compression]**. Exemple : **.sql.zip**

Parcourir les fichiers :  mediatekformations.sql (Taille maximale : 128Mio)

Il est également possible de glisser-déposer un fichier sur n'importe quelle page.

Jeu de caractères du fichier :

Figure 12: Menu d'importation de script SQL de phpMyAdmin

Il convient ainsi de renseigner le script de création de la base de données, soit **mediatekformations.sql**, puis en bas de page cliquer sur « Executer ». La page finit par s'actualiser pour afficher plusieurs messages indiquant que l'importation a réussi, comme l'illustre la [figure 13](#).



Figure 13: Exemples de messages confirmant la bonne importation du script de création de la base de données

Il est alors possible de voir la base de donnée **mediatekformation** apparaître dans la liste des bases disponibles, en cliquant dessus les tables créées apparaissent, soient « **doctrine\_migration\_version** » et « **formation** ». Ici puisqu'il ne s'agit pas d'une application symfony, seul la base **formation** nous intéresse. En allant dans la table formation, la liste des formations avec ses différents éléments apparaissent sous forme de liste comme le montre la **figure 14**. Il existe en tout 239 entrées.

























			(OC2Type datetime_immutable)							
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	1	2020-12-28 22:00:29	Eclipse n°8 : Déploiement	Exécution de l'application en dehors de l'IDE, en ...	<a href="https://i.ytimg.com/vi/Z4yTTXka958/default.jpg">https://i.ytimg.com/vi/Z4yTTXka958/default.jpg</a>	<a href="https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg">https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg</a>	Z4yTTXka958
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	2	2020-12-28 21:55:06	Eclipse n°7 : Tests unitaires	Intégration de JUnit dans l'application et créatio...	<a href="https://i.ytimg.com/vi/-nw42Xq6cYE/default.jpg">https://i.ytimg.com/vi/-nw42Xq6cYE/default.jpg</a>	<a href="https://i.ytimg.com/vi/-nw42Xq6cYE/sddefault.jpg">https://i.ytimg.com/vi/-nw42Xq6cYE/sddefault.jpg</a>	-nw42Xq6cYE
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	3	2020-12-28 21:49:27	Eclipse n°6 : Documentation technique	Intégration des commentaires normalisés et générat...	<a href="https://i.ytimg.com/vi/PrK_P3TKc00/default.jpg">https://i.ytimg.com/vi/PrK_P3TKc00/default.jpg</a>	<a href="https://i.ytimg.com/vi/PrK_P3TKc00/sddefault.jpg">https://i.ytimg.com/vi/PrK_P3TKc00/sddefault.jpg</a>	PrK_P3TKc00
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	4	2020-12-28 21:37:57	Eclipse n°5 : Refactoring	Utilisation des outils de refactoring et de généra...	<a href="https://i.ytimg.com/vi/1p_mKDDSMnQ/default.jpg">https://i.ytimg.com/vi/1p_mKDDSMnQ/default.jpg</a>	<a href="https://i.ytimg.com/vi/1p_mKDDSMnQ/sddefault.jpg">https://i.ytimg.com/vi/1p_mKDDSMnQ/sddefault.jpg</a>	1p_mKDDSMnQ
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	5	2020-11-04 14:06:07	Eclipse n°4 : WindowBuilder	Intégration de l'outil WindowBuilder dans Eclipse ...	<a href="https://i.ytimg.com/vi/pQfbr3hbw04/default.jpg">https://i.ytimg.com/vi/pQfbr3hbw04/default.jpg</a>	<a href="https://i.ytimg.com/vi/pQfbr3hbw04/sddefault.jpg">https://i.ytimg.com/vi/pQfbr3hbw04/sddefault.jpg</a>	pQfbr3hbw04
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	6	2020-11-03 17:39:37	Eclipse n°3 : GitHub et Eclipse	Créer un compte sur le site GitHub (site offrant u...	<a href="https://i.ytimg.com/vi/miN7VvZkXIM/default.jpg">https://i.ytimg.com/vi/miN7VvZkXIM/default.jpg</a>	<a href="https://i.ytimg.com/vi/miN7VvZkXIM/sddefault.jpg">https://i.ytimg.com/vi/miN7VvZkXIM/sddefault.jpg</a>	miN7VvZkXIM
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	7	2020-11-03 17:19:30	Eclipse n°2 : rétroconception avec ObjectAid	Utilisation de l'outil ObjectAid sous Eclipse pour...	<a href="https://i.ytimg.com/vi/9UBVxHsnNk/default.jpg">https://i.ytimg.com/vi/9UBVxHsnNk/default.jpg</a>	<a href="https://i.ytimg.com/vi/9UBVxHsnNk/sddefault.jpg">https://i.ytimg.com/vi/9UBVxHsnNk/sddefault.jpg</a>	9UBVxHsnNk
<input type="checkbox"/>	 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...	<a href="https://i.ytimg.com/vi/EBzTRPgbqdc/default.jpg">https://i.ytimg.com/vi/EBzTRPgbqdc/default.jpg</a>	<a href="https://i.ytimg.com/vi/EBzTRPgbqdc/sddefault.jpg">https://i.ytimg.com/vi/EBzTRPgbqdc/sddefault.jpg</a>	EBzTRPgbqdc

Figure 14: Exemple d'entrées de formations dans la table formation de la base de donnée distante

### Mise en place de l'IDE Android ArcticFox

ArcticFox est un IDE permettant le développement d'application mobile Android en Kotlin, ou en Java, et c'est ce dernier langage qui a été sélectionné. L'IDE est téléchargeable à [ce lien](#), cependant la version pourrait changer selon les évolutions de l'IDE. Après l'installation de cet IDE avec les options par défaut, il convient de le démarrer avec le mode d'administrateur afin de simplifier les différents démarrages le logiciel a été configuré pour s'ouvrir automatiquement dans ce mode. Pour cela, il convient d'effectuer un clic droit sur l'icône de lancement d'Android Studio, d'aller dans « Propriétés », puis à l'ouverture de la fenêtre des propriétés, de cliquer sur « Avancé » puis de cocher « exécuter en tant qu'administrateur » comme le montre la [figure 15](#). Ensuite, il est nécessaire de configurer l'AVD qui est l'émulateur permettant de simuler l'existence d'un mobile Android afin de tester.

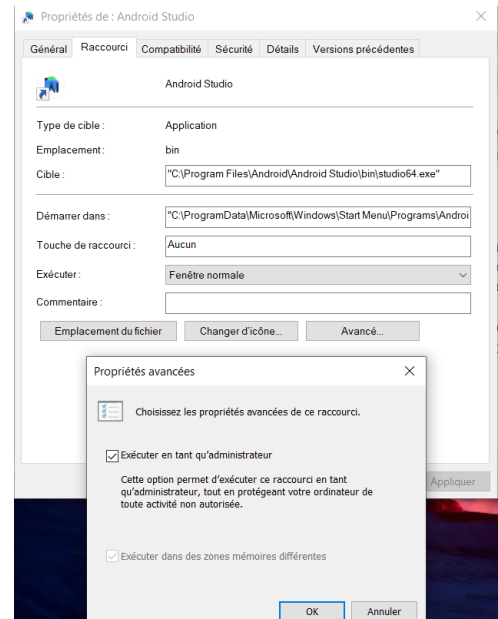


Figure 15: Fenêtre des propriétés de l'icône de lancement de l'IDE

Pour configurer le SDK, il suffit de se rendre dans le SDK Manager, ce dernier est accessible depuis le raccourci en haut à droite dans la barre des icônes, ou bien depuis l'onglet « **File > Settings, Appearance & Behavior > System Settings > Android SDK** ». La fenêtre de la [figure 16](#) apparaît, il suffit alors de sélectionner le SDK qui convient à l'aide des checkboxes puis de le télécharger en acceptant éventuellement les conditions d'utilisation. Dans notre cas, c'est Android 11.0 R qui a été sélectionné, niveau d'API 30.

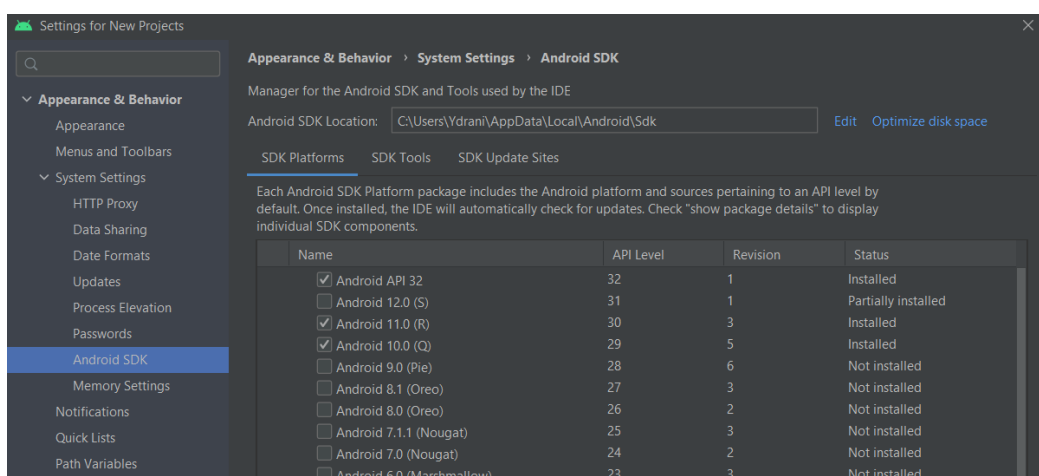


Figure 16: Fenêtre du SDK manager

### Récupération du code source et configuration de l'émulateur

La récupération du code source se fait sur la plateforme GitHub à cette [adresse](#) en cliquant sur le bouton vert à droite avec inscrit « Code » puis de sélectionner « **Download zip** ». Après téléchargement, avoir dézipper le dossier et avoir placé le projet dans le dossier de traitement, ici dans « %user > **AndroidStudioProjects** », il faut aller dans « **Android Studio, File > New > Import projet** » puis sélectionner la racine du projet normalement reconnu par l'IDE en tant que projet Android tel qu'il est possible de voir sur la [figure 17](#). Au clic sur « **OK** », une nouvelle fenêtre demande comment il faut gérer le projet importer, il faut ainsi sélectionner « **Trust project** ». Par la suite, le projet s'importe et **Gradle** se synchronise avec lui.

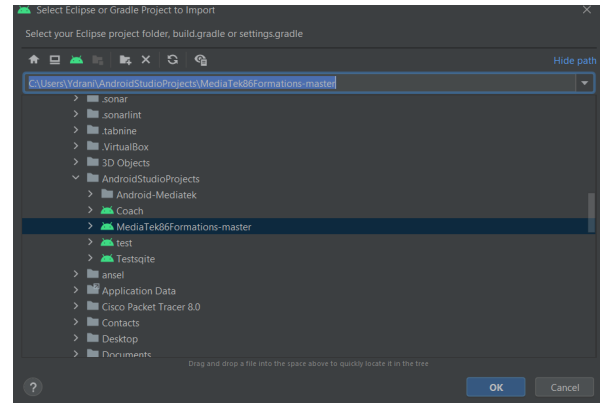


Figure 17: Fenêtre de création de projet

Pour finir, il faut configurer l'AVD, à savoir l'émulateur. Ainsi, il convient d'ouvrir l'AVD Manager dont le raccourci est en haut à droite dans la barre d'icônes, à gauche de l'icône du SDK. Au clic sur cet icône, la fenêtre de la [figure 18](#) s'ouvre.

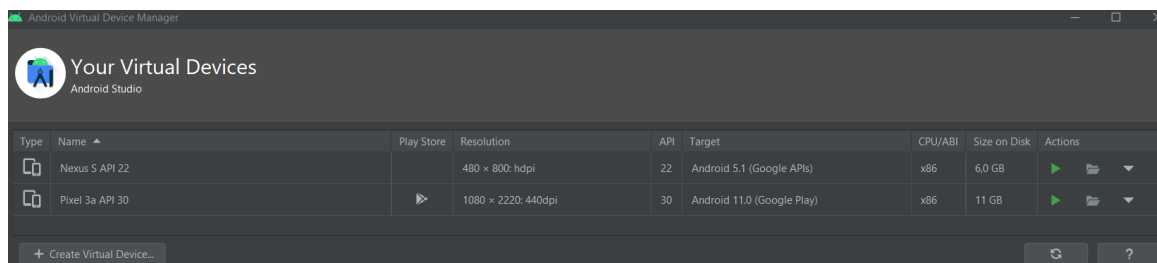


Figure 18: Gestionnaire de l'AVD Manager

Si l'émulateur Pixel 3a n'est pas dans la liste, contrairement à la [figure 18](#), il faut cliquer sur « **+ create Virtual Device** », puis dans la nouvelle fenêtre illustrée par la [figure 19](#), aller dans « **Phone** » puis sélectionner « **Pixel 3a** » qui figure dans la liste puis cliquer sur « **Next** ».

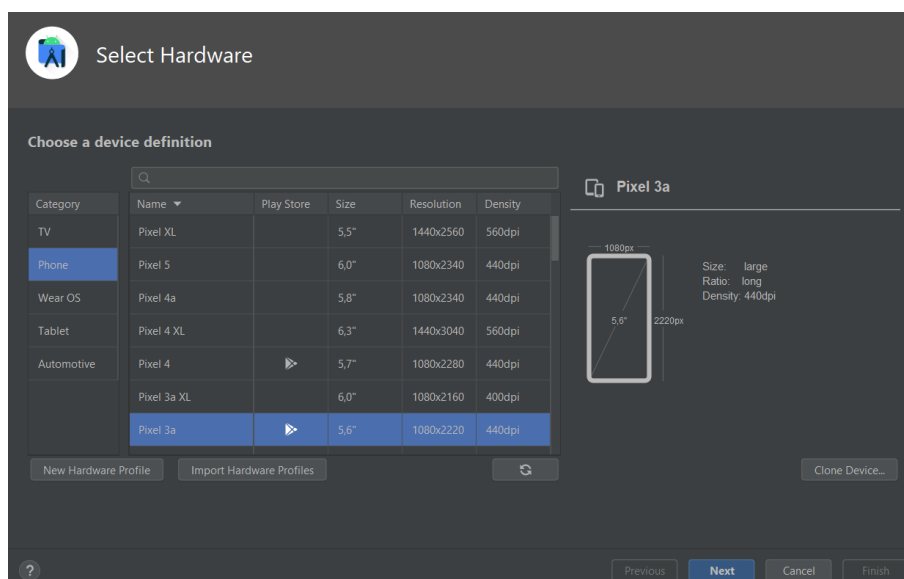


Figure 19: AVD manager sélection du type d'émulateur

Après installation de ce dernier, la fenêtre suivante, [figure 20](#) demande de sélectionner un système d'image qui est ici « **R** », s'il n'est pas déjà téléchargé, il est nécessaire de cliquer sur « **Download** » et d'attendre que le téléchargement se fasse, puis de cliquer sur « **Next** ».

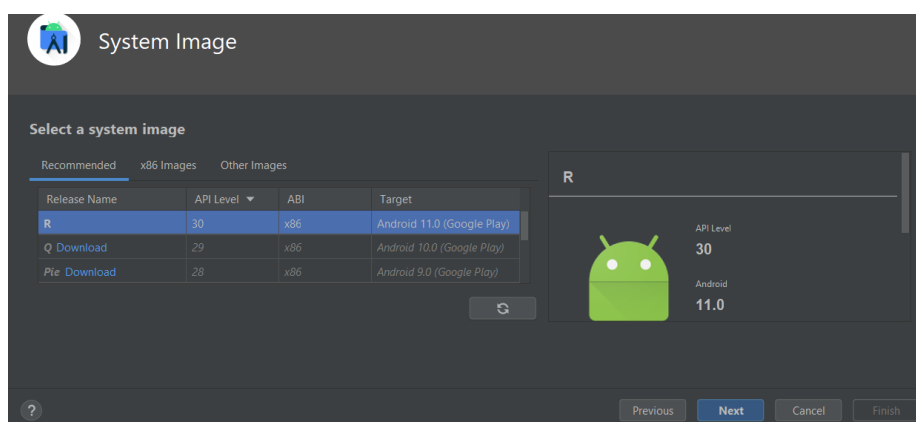


Figure 20: AVD manager sélection du système d'image

Une dernière fenêtre, [figure 21](#), s'ouvre demandant de donner un nom à l'émulateur créé ou de laisser celui par défaut, cette fenêtre présente un récapitulatif des options sélectionnées comme il est possible d'observer.

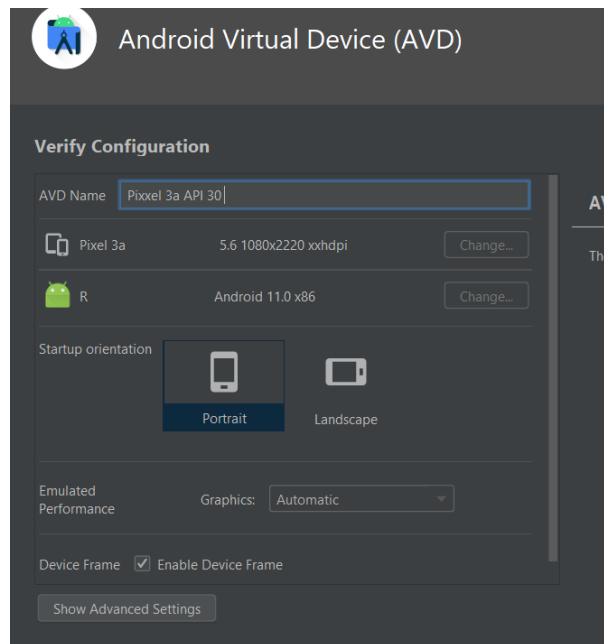


Figure 21: AVD manager vérification de configuration

A cette dernière étape de configuration de l'appareil virtuel, il convient d'appuyer sur « **Show Advanced Settings** », d'aller dans « **Boot options** » puis de cocher l'option « **Cold boot** ». Pour en finir avec la configuration, à l'allumage de l'émulateur, il faut permettre à l'appareil de pouvoir passer du mode vertical à horizontal, ainsi il convient de faire glisser le menu déroulant de haut en bas puis l'activer l'option « **auto-rotate** ».

Avant de tester l'application, une dernière manipulation reste à faire, il faut ouvrir la classe **AccesDistant.java** et modifier l'adresse du serveur afin qu'elle coïncide avec son adresse IP locale. Pour connaître cette adresse IP, il suffit d'ouvrir l'invite de commande Windows, de taper la commande : `ipconfig` puis de chercher la valeur de l'adresse IPv4, dans notre cas nous avons la valeur : **192.168.1.30**. La valeur d'origine du code source de l'adresse du serveur, soit **SERVERADDR**, était "[http://192.168.0.10/rest\\_mediatek86formations/](http://192.168.0.10/rest_mediatek86formations/)". Il faut donc la remplacer par la valeur suivante "[http://192.168.1.30/rest\\_mediatek86formations/](http://192.168.1.30/rest_mediatek86formations/)" comme le montre la *figure 22*. Il est donc possible de tester l'application avec l'appareil virtuel, où les fonctionnalités

```

/**
 * Classe d'accès à la base de données distante qui implémente AsyncResponse.
 */
public class AccesDistant implements AsyncResponse {

    /**
     * Propriété contenant la chaîne de l'adresse du serveur.
     */
    private static final String SERVERADDR = "http://192.168.1.30./rest_mediatek86formations/";

```

Figure 22: Classe AccesDistant.java valorisation de la propriété SERVERADDR

D'ailleurs, il est possible de tester l'application source depuis un appareil physique Android en le branchant au poste de travail, l'IDE est censé le reconnaître automatiquement si les options de développement ainsi que le mode « Débogage USB » sont activés dans les paramètres de l'appareil. Cependant, si les différentes pages s'ouvrent et qu'il sera possible de tester les passages de vertical à l'horizontal, il ne sera pas possible de récupérer le retour de l'API puisque le serveur et la base de données sont en local.

### Création du dépôt sur Github

Afin de pouvoir directement gérer le dépôt GitHub depuis Android Studio, il convient d'aller dans l'onglet « **VCS > Enable Version Control Integration** » puis de choisir « **git** » dans la fenêtre qui apparaît. A cet instant, les codes sources non intégrés au dépôt s'affiche en rouge. Pour créer le commit d'origine, il faut se rendre dans l'onglet « **Git** » qui alors remplacer celui de « **VCS** », puis « **GitHub > Share project on Github** », comme la [figure 24](#) suivante le démontre, il est alors possible de renseigner un nom de dépôt et éventuellement une description. En appuyant sur « **Share** », une nouvelle fenêtre s'ouvre demandant de choisir les fichiers à mettre en ligne ainsi que de laisser un message de commit, cf [figure 23](#).

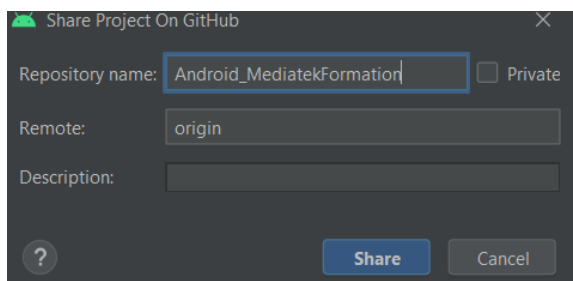


Figure 24: Creation de dépôt GitHub depuis Android Studio

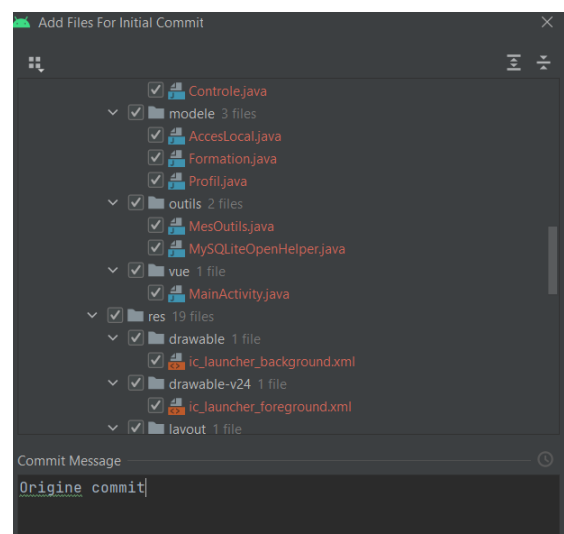


Figure 23: Creation du premier commit du projet et sélection des fichiers à déposer.

Au premier commit avec l'IDE, une authentification peut être demandée, dans notre cas cette dernière s'est effectuée par la génération d'un Token lié au compte suivant [Elshindr](#), quand au dépôt, il est accessible à [cette adresse](#), ainsi que le lien du premier [commit](#)

## Gestion des filtres

### Suivi de projet

Selon le tableau de suivi établi dans Trello illustré par la [figure 25](#), nous nous situons dans la mission 1 qui consiste à gérer les fonctionnalités de filtre sur le titre dans la page des formations. Il est possible de constater que les check-list de la mission 0 ont bien été rempli et les étiquettes sont passées de jaune à violet, indiquant ainsi leur fin de traitement.

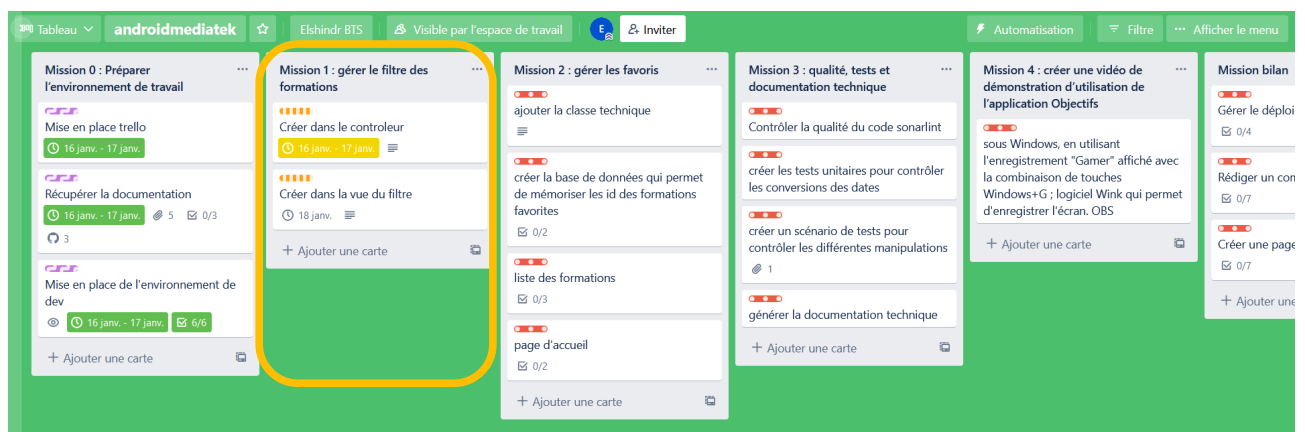
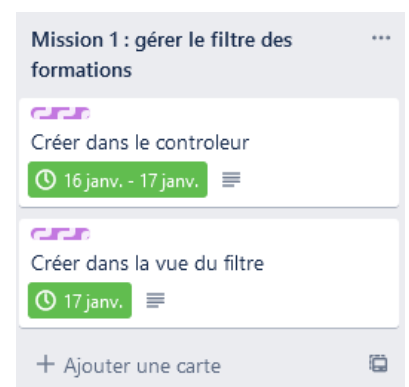


Figure 25: Tableau de suivi début de mission 1

Pour cette mission, deux cartes ont été créées, la première relatant des actions à effectuer dans le contrôleur, la seconde dans la vue, [cf figure 26](#).

- **Dans la Classe Controle**
  - Création d'une méthode qui retourne la liste ne contenant que les formations dont le titre contient le filtre reçu en paramètre.
  - Ne pas tenir compte de la case.
- **Dans la vue qui contient le filtre**
  - Création d'une méthode événementielle sur le clic du bouton "filtrer" qui vérifie si la zone de saisie est vide



Application mobile , Figure 26: Liste des cartes de la mission 1 gérer le bouton filtrer



ou non pour savoir quelle méthode appeler dans le controleur, pour valoriser la liste de formations à utiliser pour l'affichage.

Pour la mission 1, deux cartes ont été créé, chacune représentant un axe particulier à cette mission à savoir le développement dans le package du controleur puis dans celui de la vue.

## Création dans le controleur

### getLesFormationsFiltre()

Une seule modification a été effectuée dans la classe **Controle.java**, l'ajout de la fonction **getLesFormationsFiltre()** qui est visible sur la [figure 27](#).

```
/**
 * Methode qui retourne la liste des formations avec le filtre.
 *
 * @param filtre String
 * @return lstFiltre List<Formation>
 */
public List<Formation> getLesFormationFiltre(String filtre) {
    List<Formation> lstFiltre = new ArrayList<>();
    if (lesFormationsChoix != null) {
        for (Formation uneFormation : lesFormationsChoix) {
            if (uneFormation.getTitle().toUpperCase().contains(filtre.toUpperCase())) {
                lstFiltre.add(uneFormation);
            }
        }
    }
    return lstFiltre;
}
```

Figure 27: Methode getLesFormationsFiltre de la classe Controle.java

Sur cette figure, il est possible de constater que la méthode récupère en paramètre une variable de type String nommée **filtre** et qu'elle retourne une liste d'objets de type Formation. Ainsi **lstFiltre** est créée afin de récupérer les formations concernées par le filtre et sera la variable retournée par la méthode.

Une condition if vérifie que la liste de formations utilisée, **lesFormationsChoix()**, n'est pas vide, puis génère une boucle de type **foreach** afin d'itérer sur chaque formations de la liste fournis. Une seconde condition **if** vérifie si le titre de la formation à l'instant t de la boucle contient ou non la chaîne **filtre**. Afin d'éviter que le filtre soit sensible à la case, la chaîne et le titre ont été mis en majuscule avec la méthode **toUpperCase()**. Ainsi, si le titre de la

formation contient le filtre alors la dite formation est ajoutée à la liste **lstFiltre** qui est finalement retournée.

## Modifications dans la vue

### ecouteFiltre()

Puisque l'activity concernée est celui de **FormationsActivity**, c'est dans cette classe que va s'effectuer l'implémentation de la méthode d'écoute du filtre. La méthode événementielle **ecouteFiltre()** a ainsi été créée. Comme le montre la *figure 28*, elle ne retourne rien et ne possède pas de paramètres. Cependant, elle fait appel aux objets graphiques **txtFiltre** et **btnFiltrer**. Sur ce dernier composant, on appelle la méthode **setOnClickListener()** qui aura pour paramètre un objet instancié avec **View.OnClickListener()** sur lequel il est possible de réécrire la méthode **onClick()**. Cependant, ici on utilise une méthode lambda afin de clarifier l'écriture.

```
/**
 * Methode événementielle sur le clic du bouton filtrer. Permet de filtrer les formations sur le titre.
 */
private void ecouteFiltre() {
    txtFiltre = findViewById(R.id.txtFiltre);
    btnFiltrer.setOnClickListener(v -> {
        if (!txtFiltre.getText().toString().equals("")) {
            List<Formation> lstFormationFiltre = new ArrayList<>(controle.getLesFormationFiltre(txtFiltre.getText().toString()));
            creerListe(lstFormationFiltre);
        } else {
            lesFormationsChoix = controle.getLesFormationsChoix();
            creerListe(lesFormationsChoix);
        }
    });
}
```

Figure 28: Méthode événement au clic du bouton Filtrer

Dans cette méthode lambda, une condition est mise en place : soit le texte de **txtFiltre** est vide, et dans ce cas on affiche toutes les formations, soit le texte de **txtFiltre** n'est pas vide et dans ce cas on appelle la méthode écrite précédemment dans le contrôleur, à savoir **getLesFormationsFiltre()** dans laquelle on injecte le **txtFiltre** en tant que paramètre. En parallèle, on récupère le retour de cette méthode dans une liste **lstFormationFiltre** de type **Formation**, puis on appelle la méthode **creerListe()** avec cette liste en paramètre. D'ailleurs cette méthode est aussi appelée dans le cas où le filtre est vide, néanmoins dans ce cas il retourne la liste **lesFormationsChoix** dont le contenu dépend du type d'activité choisi, à savoir si on cherche à avoir toutes les formations ou juste les favoris.

### creerListe()

Cette méthode était déjà présente dans le code source d'origine, néanmoins quelques aménagements ont été effectués afin de se coordonner avec la méthode **ecouteFiltre()**. Bien que cette méthode ne retourne toujours rien, elle contient maintenant le paramètre **lesFormations** qui est une liste d'objet de type **Formation**. Elle est appelée par la méthode **ecouteFiltre()** mais aussi par la méthode **init()** qui se lance à la création de l'activity. La [figure 29](#) ci-dessous détaille cette méthode.

```
/**
 * Méthode de création de la liste adapter.
 *
 * @param lesFormations List<Formation>
 */
private void creerListe(List<Formation> lesFormations) {
    if (lesFormations != null) {
        Collections.sort(lesFormations, Collections.reverseOrder());
        ListView listView = findViewById(R.id.lstFormations);
        FormationListAdapter adapter = new FormationListAdapter(lesFormations, lesFavoris, context: FormationsActivity.this);
        listView.setAdapter(adapter);
    }
}
```

Figure 29: Methode creerListe permet de créer la liste de formation dans la vue

Globalement, cette fonction vérifie que la liste envoyée n'est pas nulle et dans ce cas, elle appelle la méthode **sort** de l'objet **Collections** permettant de trier la liste **lstFormations** dans l'ordre décroissant. Par la suite elle crée et instancie **listView** un objet de type **ListView** avec l'id du composant graphique **lstFormations** qui est une liste adaptable. C'est pourquoi on a créé un second objet **adapter** de type **FormationListAdapter** qui est instancié avec trois paramètres : la liste des formations **lesFormations**, une liste contenant les id des formations favorites nommée **lesFavoris** et avec le contexte de l'activité en cours soit **FormationsActivity**.

### init()

Représentée par la [figure 30](#), cette méthode est appelée à la création de l'activity, elle récupère l'instance du contrôleur puis initialise les composants graphiques **btnFiltrer** et **txtFiltre**. Aussi, elle valorise désormais deux nouvelles propriétés globales de type liste **lesFormationsChoix<Formation>** et **lesFavoris<Integer>** en appelant des getters situés dans le contrôleur. Enfin, si l'appel de **creerListe()** est toujours présent, elle envoie désormais en paramètre **lesFormationsChoix**.

```
/**
 * Methode qui initialise les objets graphiques et récupère l'instance du controleur.
 */
private void init() {
    btnFiltrer = findViewById(R.id.btnFiltrer);
    txtFiltre = findViewById(R.id.txtFiltre);

    controle = Controle.getInstance(this);

    lesFormationsChoix = controle.getLesFormationsChoix();
    lesFavoris = Controle.getLesFavoris();
    creerListe(lesFormationsChoix);
    ecouteFiltre();
}
```

Figure 30: Méthode init qui est lancée à la création de l'activity et initialise les diverses propriétés graphiques et appelle le controleur pour valoriser les propriétés globales.

## Dépôt github

A l'issue de ce code, un second [commit](#) a été effectué sur la plateforme GitHub dans le dépôt du projet.

## Gestion des favoris

### Suivi de projet

Selon le suivi de projet, nous nous situons au début de la mission 2 : gérer les favoris, cf *figure 31*. Dans cette intention, la liste est composée de 4 cartes chacune correspondant à une étape différente, à savoir Ajoutez la classe technique de **MySqlLiteHelper**, créer la base de données qui permettra de mémoriser les id des formations favorites, dans la page des formations mettre en place la fonctionnalité du clic sur le bouton en forme de coeur qui doit passer du gris au rouge et réciproquement, et ajouter ou supprimer de la base de donnée local la formation concernée, puis implémenter la fonctionnalité de la page des favoris qui fonctionne comme la page des formations, à la différence qu'elle ne contiendra que les formations marquées comme favorite. Les étiquettes des cartes de la mission 0 sont passées de jaune à violet indiquant ainsi la fin de ces étapes. De même, les différentes check-lists ont été mises à jours.

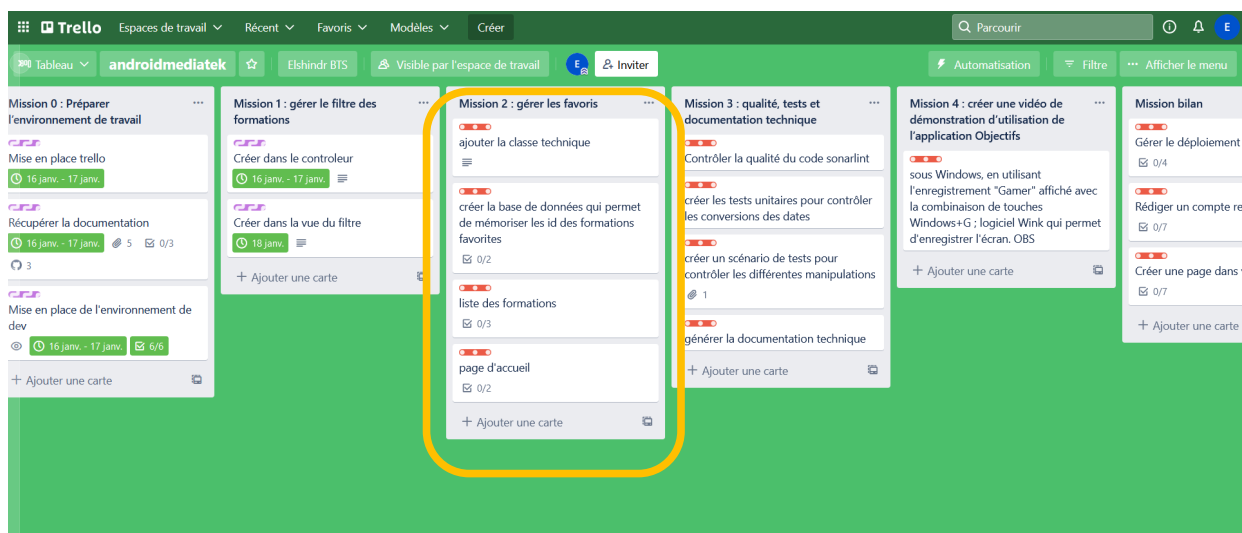


Figure 31: Suivi de mission Début de l'étape 2 gérer la fonctionnalité des favoris

Concernant de à nouveau la mission 1, chaque carte comportent une check-list avec ses différentes étapes.

- **Gestion de la base de données locale**
  - Ajouter les classes techniques AccesLocal et MySQLLiteOpenHelper fournies
  - Création de la base de donnée locale qui mémorise les id des formations marquées comme favorites
  - Vérifier que la suppression d'une formation dans la base de données distante se répercute aussi dans la base de donnée locale.
- **Gestion de l'Activity FormationsActivity**

- Gestion des clics sur le bouton des favoris dans l'activity **FormationsActivity**.
- Gestions de l'ajout et de la suppression des id dans la base de données de données locale au clic du bouton des favoris
- **Création de la page des favoris**
  - Gestion du clic sur le bouton des favoris pour supprimer la formation concernée de la liste et la réactualiser
  - Mise en place des mêmes fonctionnalités que pour la page des formations

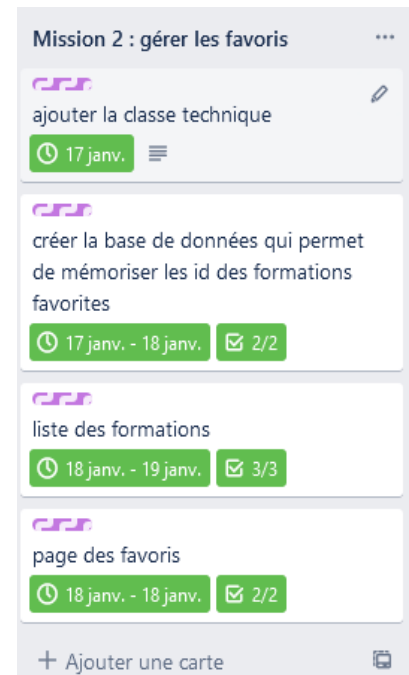


Figure 32: Liste des étapes de la mission 2

### Ajout de la classe technique **MySQLiteOpenHelper**

Afin de la mettre en place dans le projet, il convient de récupérer la dite classe fournie par les cours et de la copier dans le package outils du projet. Ensuite, il faut vérifier que l'importation s'est correctement effectuée, c'est à dire en vérifiant que le namespace du package soit : « **package com.example.mediatek86formations.outils;** ». Cette classe représentée par la [figure 33](#), 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.

```
/**
 * 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);
}
```

Figure 33: Constructeur de la classe **MySQLiteOpenHelper**

La méthode public **onCreate()**, cf [Figure 34](#), ne retourne rien mais reçoit en paramètre l'instance de la base de donnée puis elle exécute une demande de création en SQL de la table de la base de donnée avec la propriété globale **CREATION**, visible à la [figure 35](#).

```
/**
 * Méthode qui redéfinie onCreate. Elle est automatiquement appelée par le constructeur
 * si celui-ci repère que la base n'existe pas encore
 */
@param db SQLiteDatabase
@Override
public void onCreate(final SQLiteDatabase db) { db.execSQL(CREATION); }
```

Figure 34: Methode onCreate appelée automatiquement si la base de donnée locale n'est pas trouvée, permet de créer la base donnée locale

Cette propriété contient la requête de création de la table qui ici demande de créer dans la base locale de créer la table favoris qui a pour unique entrées un **Integer** nommé **idformation** qui ne doit pas être nul.

```
/**
 * Propriété contenant la requete de creation de la table
 */
private static final String CREATION = "create table favoris (idformation INTEGER NOT NULL);";
```

Figure 35 Propriété globale de la classe MySQLiteOpenHelper, contient la requête SQL de création de la base de donnée locale

La dernière méthode est **onUpgrade()** est appelée en cas de changement de version dans la base. Or pour ce projet, elle ne sera pas utilisée. Enfin le code source de cette classe [est visible ici](#).



## Ajout de la classe **AccesLocal**

Afin d'utiliser la classe **MySQLiteOpenHelper**, une nouvelle classe a été créée dans le package **modele**. Nommée **AccesLocal**, son constructeur permet d'instancier la classe **MySQLiteOpenHelper** en stockant son instance dans une propriété globale à la classe, **accesBD**.

```
/**
 * Constructeur public qui valorise la propriété d'accès à la base de donnée locale.
 *
 * @param context Context
 */
public AccesLocal(final Context context) {
    accesBD = new MySQLiteOpenHelper(context, NOMBASE, VERSIONBASE);
}
```

Figure 36: Constructeur de la classe **AccesLocal**

Cette classe possède en propriétés privées l'ensemble des paramètres nécessaires à l'instanciation de **MySQLiteOpenHelper** qui sont visibles sur la figure 37. **NOMBASE** est une chaîne contenant le nom de la base de donnée locale dont la valeur est : « **bdMediatek.sqlite** », **VERSIONBASE** est un **Integer** contenant le numéro de version de la base, **accesBd** va contenir l'instance de et la propriété **bd** est une instance de **SQLiteDatabase** qui servira lors de l'exécution des requêtes vers la base locale.

```
/**
 * Propriété contenant la chaîne du nom de la base de donnée locale.
 */
private static final String NOMBASE = "bdMediatek.sqlite";
/**
 * Propriété contenant la valeur de la version de la base de donnée.
 */
private static final Integer VERSIONBASE = 1;
/**
 * Propriété contenant l'instance de la classe technique gestion de la base de donnée SQLite.
 */
private final MySQLiteOpenHelper accesBD;
/**
 * Propriété contenant l'instance de la base de donnée locale SQLite.
 */
private SQLiteDatabase bd;
```

Figure 37: Ensemble des propriétés de la classe **AccesLocal**



Cette classe contient trois méthodes, une de récupération des données, une pour ajouter des données, quand à la dernière elle permet la suppression de ces dernières.

### recup()

La méthode **recup()**, illustrée par la *figure 38*, permet de récupérer les id contenu dans la base locale sous forme de liste qu'elle initialise et instancie au nom de **favoris**. Puis la variable de type **MySQLiteDatabase** **bd** récupère le contenu de la méthode **getReadableDatabase()** sur la propriété **accesBD**. Ainsi **bd** contient le contenant de la base qui est donc en lecture. Cette lecture s'effectue par l'intermédiaire d'un curseur, nommé curseur qui récupère les lignes contenues dans la variable **bd** après avoir lancé une requête SQL de type **select** dans la table des favoris avec la méthode **rawQuery()**.

Le curseur pouvant être placé à différents endroits dans la liste des lignes, il faut le placer à la première ligne de données en utilisant la méthode **moveToFirst()** sur la variable curseur. Ensuite, puisque notre objectif est de récupérer toute les lignes de la base, il convient de boucler avec le mot clé **while** tant que le curseur est sur une ligne de la base, ainsi la condition d'arrêt de la boucle est lorsque le curseur est après la dernière ligne, ce qui peut être traduit avec la négation sur la propriété curseur associée à la méthode **isAfterLast()**.

Dans cette boucle, on stocke à l'instant t dans la variable de type **Integer** **idFormation** la valeur du curseur dans la colonne 0, qui correspond à l'emplacement de notre entrée. Puisqu'il s'agit d'un int, on appelle la méthode **getInt()** sur le curseur. Par la suite, on ajoute la valeur **idFormation** à la liste favoris, puis on déplace le curseur à la ligne suivante avec la méthode **moveToNext()**

```
/**
 * Methode qui retourne la liste des id contenu dans la base de données locale.
 *
 * @return favoris List<Integer>
 */
public List<Integer> recup() {
    List<Integer> favoris = new ArrayList<>();
    bd = accesBD.getReadableDatabase();
    String req = "select * from favoris";
    Cursor curseur = bd.rawQuery(req, selectionArgs: null);
    curseur.moveToFirst();

    while (!curseur.isAfterLast()) {
        Integer idformation = curseur.getInt(0);

        favoris.add(idformation);
        curseur.moveToNext();
    }
    Log.d( tag: "recupfavoris", msg: "id favoris: " + favoris.toString());

    curseur.close();

    return favoris;
}
```

Figure 38: Méthode recup() permet de récupérer les données de la base locale

A la fin de la boucle, il faut fermer le curseur avec la méthode **close()**, puis on retourne la liste **favoris**.

### ajoutFavoris()

Cette méthode visible sur la [figure 39](#) permet d'insérer une nouvelle ligne de données dans la base locale. Elle ne retourne rien mais reçoit en paramètre une variable **id** de type **int**. La variable globale **bd** récupère l'instance de **getWritableDatabase()** sur **accesBD**, permettant ainsi de pouvoir écrire dans la base locale. Ici afin de limiter les éventuelles injections SQL la requête SQL n'est pas directement écrite. Il faut pour cela créer une instance nommée **values** de type **ContentValues**, sur laquelle il est possible d'utiliser la méthode **put()** qui reçoit deux paramètres, le nom de la colonne et la valeur à insérer, soit ici « **idformation** » et **id**. Enfin, on utilise **insert()** sur la variable **bd** et comme paramètres le nom de la table soit **favoris**, null et **values**. Puis on ferme la base avec la méthode **close()**.

```
/**
 * Methode qui ajoute dans la base de donnée locale la valeur du parametre fourni.
 *
 * @param id int
 */
public void ajoutFavoris(final int id) {
    bd = accesBD.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("idformation", id);
    bd.insert( table: "favoris", nullColumnHack: null, values);
    bd.close();
}
```

Figure 39: Méthode d'ajout de donnée dans la base locale

### removeFavoris()

Visible sur la [figure 40](#), cette méthode est similaire à **ajoutFavoris()**, elle ne retourne rien et reçoit la variable **notFavId** de type **int**. La variable locale **bd** récupère l'instance de la base de donnée grâce à l'appel de **getWritableDatabase()** sur **accesBD**, puis il faut utiliser sur cette variable il faut utiliser la méthode **delete()** qui reçoit 3 paramètres, la table concernée, soit **favoris**, la clause de localisation « **idformation= ?** », puis un array chaîné contenant la valeur à supprimer de la base locale. D'ailleurs, puisque la variable est de

type int alors qu'on attend une chaîne, la méthode **String.valueOf()** est appliquée sur la variable envoyée en paramètre.

```
/**
 * Methode qui supprime l'id fourni de la base de donnée locale.
 *
 * @param notFavId int
 */
public void removeFavoris(final int notFavId) {
    bd = accesBD.getWritableDatabase();
    bd.delete( table: "favoris", whereClause: "idformation=?", new String[]{String.valueOf(notFavId)});
    Log.d( tag: "suppr", msg: "id: " + notFavId);
}
}
```

Figure 40: Méthode de suppression de donnée dans la base locale

## Modifications dans le contrôleur

### getInstance()

Afin que l'appel de la base de donnée locale se fasse à l'ouverture de l'application, l'appel de la classe **AccesLocal** se fait au même niveau que celui pour l'**AccesDistant**, c'est à dire dans la méthode **getInstance()**. Une nouvelle propriété globale **accesLocal** de type **AccesLocal** récupère l'instance de **AccesLocal** avec pour propriété une variable de type **Contexte** accessible via le nouveau paramètre de la méthode **getInstance()**. Il est a noter que désormais l'ensemble des appels à la méthode **contrôle.getInstance()** envoie ce paramètre ,comme le désigne la [figure 41](#).

```
/**
 * Methode de récupération de l'instance unique du Controleur.
 *
 * @param context Context
 * @return instance
 */
public static Controle getInstance(Context context) {
    if (Controle.instance == null) {
        Controle.instance = new Controle();

        Handler handler = new Handler(Looper.getMainLooper());
        handler.postDelayed(() -> {
            accesDistant = new AccesDistant();
            accesDistant.envoi( operation: "tous", lesDonneesJSON: null);
        }, delayMillis: 2000); //2 secondes

        accesLocal = new AccesLocal(context);
        lesFavoris = accesLocal.recup();
    }

    return Controle.instance;
}
```

Figure 41: Methode getInstance de la classe controle

Enfin, un second paramètre global a été ajouté, **lesFavoris** qui est une liste de type **Integer** qui récupère le retour de la méthode **recup()** de la classe **AccesLocal**. De plus, les méthodes respectives de getter et setter ont été générées par l'IDE pour cette nouvelle variable.

### checklesFavoris()

Afin de ne pas avoir de formations dans les favoris qui auraient été supprimées de la base de donnée distante, il est nécessaire de mettre en place la méthode **checkLesFavoris()** permettant de supprimer de la liste des favoris toute formations n'étant plus dans la liste de toutes les formations.

Cette méthode visible à la [figure 42](#), ne retourne rien et n'a pas de paramètres puisqu'elle récupère diverses variables globales à la classe **Controle**. Cette méthode instancie une liste de type **Integer** nommée **testIdFavoris** qui va récupérer toutes les **id** des formations existantes via la variable globale **lesFormationsAll**.

```
/**
 * Methode qui permet de nettoyer la base de donnée locale si une formation est supprimée.
 */
public static void checkLesFavoris() {
    List<Integer> testIdFavoris = new ArrayList<>();
    for (Formation uneFormation : lesFormationsAll) {
        testIdFavoris.add(uneFormation.getId());
    }

    for (int i = 0; i < lesFavoris.size(); i++) {
        if (!testIdFavoris.contains(lesFavoris.get(i))) {
            Log.d( tag: "checkFav", msg: "Suppr favoris: " + lesFavoris.get(i).toString());
            accesLocal.removeFavoris(lesFavoris.get(i));
            lesFavoris.remove(lesFavoris.get(i));
        }
    }
}
```

Figure 42: Méthode permettant de nettoyer la base locale de toute formation n'existant plus

Ensuite, dans une nouvelle boucle for qui va incrémenter une variable **i** de type **int** jusqu'à atteindre la valeur de la taille de la liste **lesFavoris**. Dans chacune de ces boucles, on teste si la valeur de **lesFavoris** à l'instant **i** est contenue dans la liste **testIdFavoris**. Si elle n'existe pas dans la liste **testIdFavoris** alors la valeur de **lesFavoris** à cet instant est supprimée de la base local via l'appel de la méthode **removeFavoris()** puis et supprimer de la liste **lesFavoris**.

Enfin cette méthode est appelée dans la méthode **processFinish()** de la classe **AccesDistant**, une fois que la liste de **Formations** **lesFormationsAll** soit valorisée.

### ajoutFavoris()

Visible à sur la [figure 43](#), cette méthode permet d'appeler la méthode **ajoutFavoris()** de la classe **AccesLocal**. Elle ne retourne rien mais reçoit en paramètre **favFormation** de type **Formation**. Cette méthode vérifie que l'**Id** de la formation fournie n'est pas déjà présente dans la liste **lesFavoris**, ainsi cette valeur est ajoutée dans la liste **lesFavoris** et on appelle la méthode **ajoutFavoris** d'**AccesLocal** en injectant en paramètre l'**id** de la formation. Puis, on ajoute la formation dans la variable globale **lesFormationsFavorites** qui est une liste de type **Formation**.

```
/**
 * Methode qui ajoute un objet Formation à la liste lesFavoris.
 *
 * @param favFormation Formation
 */
public void ajoutFavoris(Formation favFormation) {
    if (!lesFavoris.contains(favFormation.getId())) {
        lesFavoris.add(favFormation.getId());
        accesLocal.ajoutFavoris(favFormation.getId());
        lesFormationsFavorites.add(favFormation);
    }
}
```

Figure 43: Methode ajoutFavoris de la classe Controle

### removeFav()

Visible à sur la [figure 44](#), cette méthode permet d'appeler la méthode **removeFavoris()** de la classe **AccesLocal**. Elle ne retourne rien mais reçoit v en paramètre **notfavFormation** de type **Formation**. Cette méthode vérifie que l'**Id** de la formation fournie est bien présente dans la liste **lesFavoris**, ainsi cette valeur est supprimée dans la liste **lesFavoris**, puis on appelle la méthode **removeFavoris()** d'**AccesLocal** en injectant en paramètre l'**id** de la formation. Puis, on supprime la formation dans la variable globale **lesFormationsFavorites**.

```

/**
 * Methode qui supprime un objet Formation à la liste lesFavoris.
 *
 * @param notfavFormation Formation
 */
public void removeFav(Formation notfavFormation) {
    if(lesFavoris.contains(notfavFormation.getId())){
        accesLocal.removeFavoris(notfavFormation.getId());
        lesFavoris.remove((Integer) notfavFormation.getId());
        lesFormationsFavorites.remove(notfavFormation);
    }
}

```

Figure 44: Méthode qui supprime une formation des favoris en appelant la méthode removeFavoris de la classe AccesDistant

## Gestion de l'Activity FormationsActivity

### Gestion des clics sur btnFavoris

La gestion de ce bouton est liée à la liste adaptable, **listView**, initialisée dans la méthode **creerListe()** de la classe **FormationsActivity** visible à la [figure 45](#). Dans le fichier layout **activity\_ formations**, cette liste est liée au composant **lstFormations** décrivant cette liste adaptable. C'est dans cette méthode qu'est instanciée la variable **adapter** de type **FormationListAdapter** qui va ainsi appeler le constructeur de la classe **FormationListAdapter** où va se dérouler l'ajout de la fonctionnalité du clic sur le bouton des favoris puisque c'est dans cette classe que sont gérés les objets graphiques qui sont liées à **lstFormations**.

```

/**
 * Methode de création de la liste adapter.
 *
 * @param lesFormations List<Formation>
 */
private void creerListe(final List<Formation> lesFormations) {
    if (lesFormations != null) {
        Collections.sort(lesFormations, Collections.reverseOrder());
        ListView listView = findViewById(R.id.lstFormations);
        FormationListAdapter adapter = new FormationListAdapter(lesFormations, lesFavoris, context, FormationsActivity.this);
        listView.setAdapter(adapter);
    }
}

```

Figure 45: Methode creerListe permet la création des listes adapter

La classe **FormationListAdapter** hérite de **BaseAdapter** qui met ainsi à disposition plusieurs méthodes permettant d'avoir des getters sur plusieurs propriétés telles que **getCount()** qui retourne la taille de la liste **lesFormations**, **getItem()** qui renvoie la formation de la liste **lesFormations** à un instant **i** puis **getItemId()** qui renvoie la valeur **i** à



l'instant donné. Enfin, cette classe réécrit la méthode **getView()** qui sera détaillée juste après.

**FormationListAdapter** possède aussi une classe interne, **ViewProperties** illustré par la [figure 46](#), décrivant les objets graphiques interactives de l'objet **IstFormations**. Ainsi cette classe interne possède deux objets de type **TextView**, à savoir **txtListPublishedAt** qui va afficher la date de publication et **txtListeTitle** qui va recevoir le titre de la formation. Le dernier objet graphique est **btnListFavoris** qui est de type **ImageButton**.

```
/**
 * Classe privée interne possédant les propriétés des lignes de la liste adapter.
 */
private static class ViewProperties {
    /**
     * Propriété graphique représentant le bouton des favoris.
     */
    ImageButton btnListFavori;
    /**
     * Propriété graphique de la zone de text de la date de publication.
     */
    TextView txtListPublishedAt;
    /**
     * Propriété graphique de la zone de texte du titre.
     */
    TextView txtListeTitle;
}
```

Figure 46: Classe interne ViewProperties

Le constructeur de la classe **FormationListAdapter** valorise ses propriétés globales privées. Parmi ces propriétés il est possible d'énumérer la liste **lesFormations** qui reçoit une objet de type **Formation**, une seconde liste **lesFavoris** de type **Integer**, **context** de type **Context**, contrôle une instance du controleur et **inflater** qui reçoit comme valeur le retour de la méthode **LayoutInflater.from(Context)**.

```
/**
 * Constructeur de la liste graphique adaptable. Valorise l'ensemble de ses propriétés privées.
 *
 * @param lesFormations List<Formation>
 * @param lesFavoris List<Integer>
 * @param context Context
 */
public FormationListAdapter(List<Formation> lesFormations, List<Integer> lesFavoris, Context context) {
    this.lesFormations = lesFormations;
    this.lesFavoris = lesFavoris;
    this.controle = Controle.getInstance(context);
    this.context = context;
    this.inflater = LayoutInflater.from(context);
}
```

Figure 47 : Constructeur de la liste adapter

Quand à la méthode **getView()**, elle permet la construction de la vue d'une ligne et de ses différents événements, ainsi elle sera lancée pour chaque objet contenu dans la liste **lesFormations**, c'est pourquoi cette méthode reçoit le paramètre **i** de type **int** qui sera incrémenté à chaque nouvelle ligne à créer, un objet **view** de type **View** qui définit la vue actuelle et l'objet **viewGroup** de type **viewGroup** qui définit l'ensemble des lignes, néanmoins ce dernier paramètre ne sera pas utilisé.

Cette méthode commence par initialiser une variable locale **viewProperties** du type de la classe interne **ViewProperties**, puis vérifie si **view** est nulle. Dans ce cas, elle instancie **viewProperties** et relie ses propriétés à l'identifiant défini dans le design d'activité **layout\_ formations** par l'intermédiaire de la méthode **findViewById()**. Ainsi la propriété **txtListeTitle** de la classe interne **viewProperties** est reliée à l'identifiant de la vue **view** à **txtListTitle**. Il en va de même pour les deux autres propriétés, la propriété de la classe interne **ViewProperties** **txtListPublishedAt** est valorisée par l'identifiant de la vue **txtListPublishedAt** et la propriété **btnListFavoris** à **btnListFavori**.

Enfin, il faut créer une étiquette avec la méthode **setTag()** sur **view** avec en **viewProperties** paramètre. Néanmoins, dans le cas où la vue **view** n'est pas nulle, il faut récupérer l'étiquette dans **viewProperties** avec la méthode **getTag()**. La [figure 48](#) suivante détaille le début de cette méthode.



```

/**
 * Methode de construction des lignes de la liste. Appellée pour chaque item.
 *
 * @param i      int
 * @param view   View
 * @param viewGroup ViewGroup
 * @return view View
 */
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    ViewProperties viewProperties;
    if (view == null) {
        viewProperties = new ViewProperties();
        view = inflater.inflate(R.layout.layout_liste_formation, root: null);
        viewProperties.txtListeTitle = view.findViewById(R.id.txtListTitle);
        viewProperties.txtListPublishedAt = view.findViewById(R.id.txtListPublishedAt);
        viewProperties.btnListFavori = view.findViewById(R.id.btnListFavori);
        view.setTag(viewProperties);
    } else {
        viewProperties = (ViewProperties) view.getTag();
    }
}

```

Figure 48: Methode getView récupération des objets graphiques

Par la suite, comme le montre la [figure 49](#) la méthode **getView()** valorise les différentes propriétés de la classe interne avec les propriétés correspondantes de la liste **lesFormations**, soit avec la valeur du titre pour **txtListeTitle** et la valeur de la date de publication pour **txtListPublishedAt**. Enfin sur ces deux composants graphiques une méthode d'écoute **setOnClickListener()** est appliquée appelant alors la méthode privée **ouvrirUneFormationActivity()** qui reçoit en paramètre la vue **view**. Cette méthode déjà présente dans le code source d'origine permet simplement d'ouvrir une nouvelle activity, tout en fermant celle actuelle, **UneFormationActivity** en fonction du tag en cours.

```

viewProperties.txtListeTitle.setTag(i);
viewProperties.txtListeTitle.setText(lesFormations.get(i).getTitle());
viewProperties.txtListeTitle.setOnClickListener(this::ouvrirUneFormationActivity);

viewProperties.txtListPublishedAt.setTag(i);
viewProperties.txtListPublishedAt.setText(lesFormations.get(i).getPublishedAtToString());
viewProperties.txtListPublishedAt.setOnClickListener(this::ouvrirUneFormationActivity);
|

```

Figure 49: Suite getView qui valorise les composants graphiques de viewProperties puis met en place une methode d'écoute pour ouvrir la prochaine Activity.

Concernant la propriété **btnListFavori** qui permet d'identifier une formation comme favoris ou non, elle est valorisée avec le contenu de la méthode **setImageResource()** qui appelle la méthode privée **selectFavBtnColor()** recevant la valeur de l'indice **i**, visible sur la [figure 50](#). Cette dernière méthode valorise **btnListFavoris** lors de la création de la liste, soit avec la ressource nommée **coeur\_rouge** si l'id de la formation donnée à l'instant **i** est

contenue dans la liste **lesFavoris** de type **Integer**, sinon la ressource nommée **coeur\_gris**.

```
/**
 * Methode événementielle qui permet de gérer si le bouton favoris d'une formation est marquée comme favoris ou non
 *
 * @param id int
 * @return int Ressource
 */
private int selectFavBtnColor(int id) {
    if (lesFavoris.contains(lesFormations.get(id).getId())) {
        return R.drawable.coeur_rouge;
    } else {
        return R.drawable.coeur_gris;
    }
}
```

Figure 50: Méthode `selectFavBtnColor` permet de valoriser `btnListFavoris` avec la bonne ressource

Enfin, cette propriété **btnListFavoris** est liée à une écoute sur le clic avec la méthode **setOnClickListener()**. Cette écoute est liée à un objet **v** de type **ViewListener** qui dans une méthode lambda vérifie que l'**id** de la formation à l'instant donnée **i** de la liste **lesFormations** soit présent dans la liste des favoris, ou non. Dans le premier cas cela signifie que la formation en cours était marquée en tant que favoris, ainsi il faut changer la ressource de l'image en mettant la valeur sur **coeur\_gris**, puis supprimer de la liste des favoris l'**id** de cette formation. Pour cela, il faut appeler l'instance du controleur afin d'appeler la méthode **removeFav()** vu plus tôt qui permettra de supprimer de la liste des favoris, **lesFavoris**, l'id de la formation concernée ainsi que dans la base de donnée locale.

```
viewProperties.btnListFavori.setImageResource(selectFavBtnColor(i));
viewProperties.btnListFavori.setOnClickListener(v -> {
    if (lesFavoris.contains(lesFormations.get(i).getId())) {
        viewProperties.btnListFavori.setImageResource(R.drawable.coeur_gris);
        controle.removeFav(lesFormations.get(i));
    } else {
        viewProperties.btnListFavori.setImageResource(R.drawable.coeur_rouge);
        controle.ajoutFavoris(lesFormations.get(i));
    }
    lesFavoris = Controle.getLesFavoris();
    notifyDataSetChanged();
});

return view;
}
```

Figure 51: Suite de la classe `getView()`, gestion des favoris

A l'inverse, si l'identifiant n'est pas contenu dans la liste des favoris, on change la ressource image de **btnListFavori** sur **coeur\_rouge** puis on appelle la méthode **ajoutFavoris** de la classe **Controle** afin d'insérer cette nouvelle valeur dans la liste des favoris **lesFavoris** et de la base de donnée locale. Pour finir, il convient d'appeler la méthode **notifyDataSetChanged()** afin de mettre à jour la liste adapter.

## Mise en place de la page des favoris

### Dans la vue MainActivity

La page des favoris fonctionnant de la même manière que la page des formations, il a été décidé d'utiliser les mêmes activity des pages pour toutes les formations pour créer celles des favoris, cela permettant de ne pas avoir à dupliquer l'activity **FormationsActivity** avec une éventuelle nouvelle activity **FavorisActivity** pour la page des favoris, puisque la seule différence provient du contenu de la liste de formations envoyée.

Cette mise en place commence par l'activity **MainActivity** dans laquelle la méthode **creerMenu()** a été modifiée afin d'avoir deux appels sur la méthode **ecouteMenu()** qui reçoit désormais deux paramètres. Le premier est de type **ImageButton** qui reçoit le bouton sur lequel le clic s'est effectué, soit celui pour accéder à toutes les formations, soit celui des favoris, le second est une chaîne indiquant le **choix** d'activité. Soit sa valeur est « all », soit « favoris ». Cette méthode est visible sur la [figure 52](#).

```
/**
 * Méthode qui appelle les procédures événementielles gérant le menu.
 */
private void creerMenu() {
    ecouteMenu(findViewById(R.id.btnFormations), choix: "all");
    ecouteMenu(findViewById(R.id.btnFavoris), choix: "favoris");
}

/**
 * Méthode événementielle sur le clic d'une image du menu.
 *
 * @param btn ImageButton
 * @param choix String
 */
private void ecouteMenu(ImageButton btn, String choix) {
    btn.setOnClickListener(v -> {
        Activity activity = MainActivity.this;
        Intent intent = new Intent(activity, FormationsActivity.class);
        activity.startActivity(intent);
        Controle.setChoix(choix);
    });
}
```

Figure 52: Méthodes d'écoute sur les boutons de la page d'accueil

La méthode **ecouteMenu()** met en place un événement sur le bouton envoyé en paramètre avec **setOnClickListener()** qui dans une fonction lambda instancie **activity** de type **Activity** et la valorise avec l'instance de l'activité actuelle **MainActivity**. Puis il faut instancier un objet **intent** de type **Intent** qui va permettre de passer de l'activité actuelle à une autre. Dans notre cas, il faudra passer systématiquement vers **FormationsActivity**. Ensuite, l'activité se lance avec la méthode **startActivity()** sur la variable **activity** en passant l'intention **intent** en paramètre. Enfin, on appelle la classe **Controle** dans laquelle on définit la variable globale **choix** de cette classe avec le setter **setChoix()** en passant en paramètre la valeur appropriée : soit « all » si c'est le bouton **btnFormations** qui a été sélectionné ou « favoris » si c'est **btnFavoris**.

### Dans le controleur

Parmi les variables globales de la classe **Controle**, il est désormais possible d'énumérer les propriétés suivantes :

- **lesFormationsAll**:qui est une liste de Formations qui reçoit l'intégralité des formations existantes dans la base de données locales.
- **LesFormationsFavorites** qui est une liste de Formations qui reçoit les formations marquées comme favorites
- **LesFormationsChoix** qui est une liste de Formations qui reçoit soit la liste **lesFormationsAll** soit la liste **lesFormationsFavorites**, selon le contenu de la propriété choix.
- **LesFavoris** qui une liste **d'Integer** qui reçoit l'id des formations contenues dans **lesFormationsFavorites**.

```
public class Controle {
    /**
     * Propriété contenant la liste de toutes les formations.
     */
    private static List<Formation> lesFormationsAll = new ArrayList<>();
    /**
     * Propriété contenant la liste des formations marquées comme favorites.
     */
    private final List<Formation> lesFormationsFavorites = new ArrayList<>();
    /**
     * Propriété contenant la liste des formations en cours d'utilisation.
     */
    private List<Formation> lesFormationsChoix = new ArrayList<>();
    /**
     * Propriété contenant la liste des id des formations favorites.
     */
    private static List<Integer> lesFavoris = new ArrayList<>();
    /**
     * Propriété contenant le choix de l'activité.
     */
    private static String choix = "";
```

Figure 53: Liste des propriétés qui sont liées à la gestion du choix entre afficher toutes les formations ou celles inscrites comme favorites.

Afin de récupérer la bonne liste, une méthode **getLesFormationsChoix()** illustrée par la [figure 54](#), a été mise en place. Elle permet de vérifier le contenu de la variable choix si elle vaut « all » ou « favoris », ainsi la propriété **lesFormationsChoix** est valorisée en conséquence avec respectivement **lesFormationsAll** ou **lesFormationsFavorites**.

```
/**
 * Getter sur la liste des formations choisies.
 *
 * @return lesFormationsChoix List<Formation>
 */
public List<Formation> getLesFormationsChoix() {
    if (choix.equals("favoris")) {
        lesFormationsChoix = getLesFormationsFavorites();
    }
    if (choix.equals("all")) {
        lesFormationsChoix = getLesFormationsAll();
    }
    return lesFormationsChoix;
}
```

Figure 54 : Méthode qui permet de valoriser lesFormationsChoix selon l'activité sélectionné

La liste **LesFormationsFavorites** est établie par la méthode **getLesFormationsFavorites()**, qui dans une boucle foreach fait une itération sur chaque formations de la liste **lesFormationsAll**, puis vérifie que l'**id** de la formation actuelle existe dans la liste **lesFavoris** et qu'elle n'y soit pas déjà présente. Dans ce cas, la formation est ajoutée à la liste **lesFormationsFavorites**, cf [figure 55](#).

```
public List<Formation> getLesFormationsFavorites() {
    for (Formation uneFormation : lesFormationsAll) {
        if (lesFavoris.contains(uneFormation.getId()) && !lesFormationsFavorites.contains(uneFormation)) {
            lesFormationsFavorites.add(uneFormation);
        }
    }
    return lesFormationsFavorites;
}
```

Figure 55: Methode getLesFormationsFavorites valorise la liste lesFormationsFavorites

Pour rappel, la liste **lesFormationsAll** est valorisée par appel de la procédure **ProcessFinish()** de la classe **AccesDistant**.

### Dépot github

A l'issue de ce code, un troisième [commit](#) a été effectué sur la plateforme GitHub dans le dépôt du projet.



## Qualité du code

### Suivi de projet

La mission 2 terminée, ses étiquettes dans ce tableau modifiées pour être mises en violet pour signifier leurs clôtures, nous obtenons alors le tableau de suivi visible sur la *figure 56*. La mission suivante a pour objectif de mettre en place la qualité du code produit, ainsi une liste de 4 quatre ont été établis.

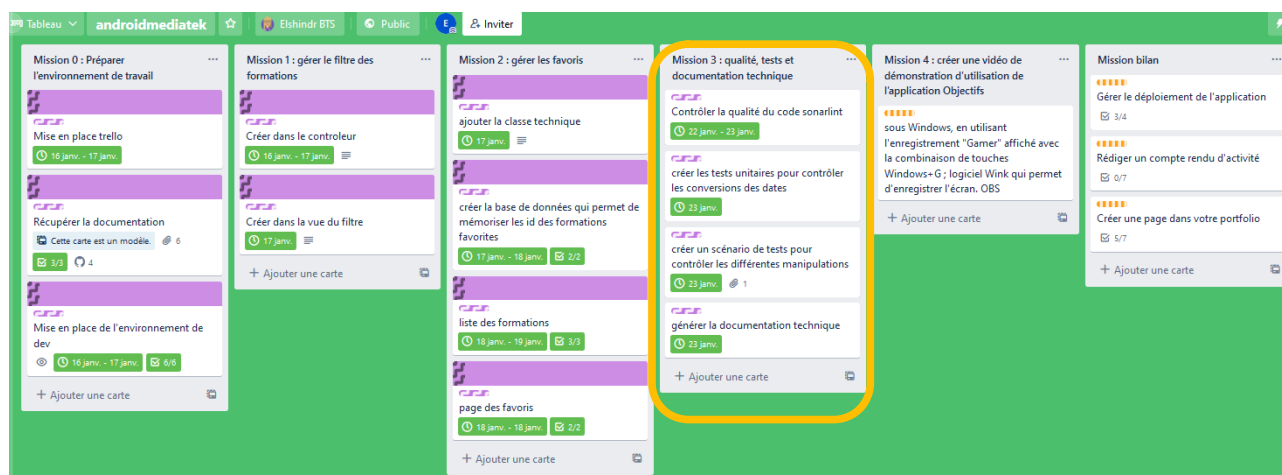


Figure 56: Tableau de suivi début de la mission 3 gérer la qualité du code

Ainsi les différentes étapes établies sont les suivantes :

- Contrôler la qualité du code avec SonarLint
- Création des tests unitaires pour la conversion des dates
- Création d'un scénario de tests pour les différentes manipulations de l'interface
- Création de la documentation technique de l'application



Figure 57: Liste des cartes de la mission 3



## Tests unitaires

Les tests unitaires demandés concernaient les conversions de String vers Date, et inversement pour la propriété **publishedAt** de la classe Formation. En effet la classe Formation récupère la valeur de cette propriété au format Date, alors que son utilisation dans l'application nécessite de la formater selon le pattern jour/mois/année. Pour cela l'interface **MesOutils** fourni par les cours donne accès à différentes méthodes dont **convertDateToString()** et **convertStringToDate()** qui permettent de convertir le contenu de **publishedAt** de type Date en String avec le format adapté et inversement.

Dans ce but, la classe MesOutilsTest a été créé avec le menu dédié à cet effet en allant dans l'IDE « **Navigate>Test** ». Suite à cela la fenêtre à la [figure 58](#) suivante s'ouvre :

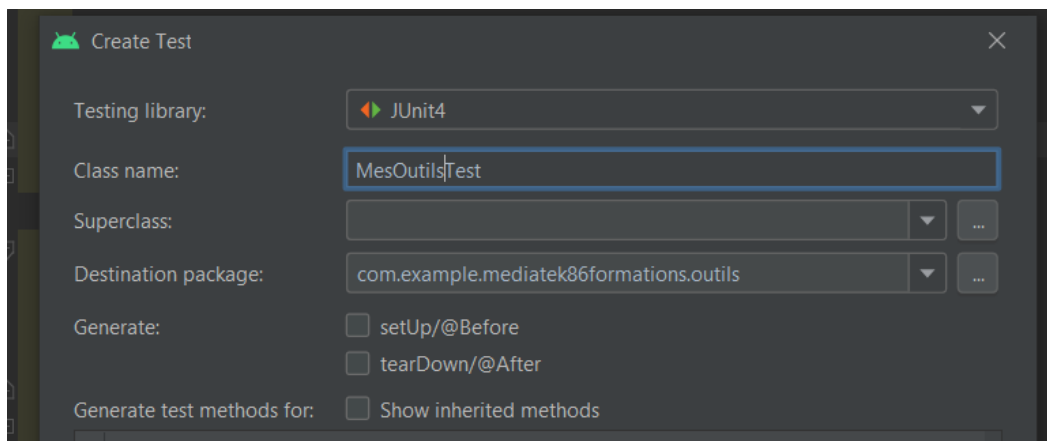


Figure 58: Fenêtre de création de tests unitaires

Il faut ensuite sélectionner les méthodes à tester par l'intermédiaire de checkbox, puis une nouvelle fenêtre s'ouvre : la classe des tests unitaires dédiés à la classe sélectionnée. Dans notre cas deux classes de tests ont été créé **MesOutilsTest** et **FormationTest**.

### FormationTest

Cette première classe de test permet de tester les valeurs des propriétés **publishedAt** qui retourne une **Date** et la méthode **publishedAtToString()** de la classe **Formation**. Cinq propriétés ont été établie afin de mettre en place la variable de test date de type Date.

- **StrDate** de type **String** qui porte comme valeur une chaine contenant une date : "2020-08-07 05:59:10"
- **format** de type **String** qui représente le format de la date attendu
- **formatter** de type **SimpleDateFormat** qui instancie un objet SimpleDateFormat

- **date** de type **Date** qui reçoit la valeur de **formatter.parse(strDate)** qui permet ainsi de convertir la chaîne **strDate** en **Date** au format attendu.

Enfin un objet **uneFormation** de type **Formation** qui initialise une nouvelle formation avec la variable **date** comme valeur pour **publishedAt**

```
/**
 * Classe de tests unitaire pour la classe Formation
 */
public class FormationTest extends TestCase {
    private final String strDate = "2020-08-07 05:59:18";
    private final String format = "dd/MM/yyyy";
    private final SimpleDateFormat formatter = new SimpleDateFormat(format);
    private Date date = new Date();
    private final Formation uneFormation = new Formation( id: 13, date, titre: "titre", description: "description", miniature: "", picture: "", videoid: "");

    {
        try {
            date = formatter.parse(strDate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

Figure 59: Initialisation des variables pour tester **publishedAt**

L'IDE permet de générer automatiquement les différentes méthodes de tests qui ont été sélectionnées lors de la création de la classe de test, ainsi il est possible d'obtenir la figure suivante.

```
/**
 * Test unitaire permettant de vérifier le getter de publishedAt
 */
public void testGetPublishedAt() { assertEquals(date, uneFormation.getPublishedAt()); }

/**
 * Test unitaire permettant de vérifier la conversion en String de publishedAt
 */
public void testGetPublishedAtToString() {
    assertEquals(formatter.format(date), uneFormation.getPublishedAtToString());
}
```

Figure 60: Méthodes de tests unitaire de la classe **FormationTest**

La première **testGetPublishedAt()** contient la méthode **assertEquals()** qui permet de tester deux valeurs, ainsi on peut comparer **date** à la valeur de **publishedAt** de l'objet **uneFormation** en appelant son getter **getPublishedAt()**.

La seconde **testGetPublishedAtToString()** contient la même méthode et permet de tester la valeur renvoyée par **getPublishedAtToString()**. Ainsi, on compare la chaîne renvoyée par **formatter.format(date)** qui ici doit renvoyer **date** donnée sous forme de chaîne au format « **dd/MM/yyyy** », il en va de même pour la méthode **getPublishedAtToString()**. Enfin, les résultats de ces tests sont visibles à la [figure 61](#) ci-dessous.

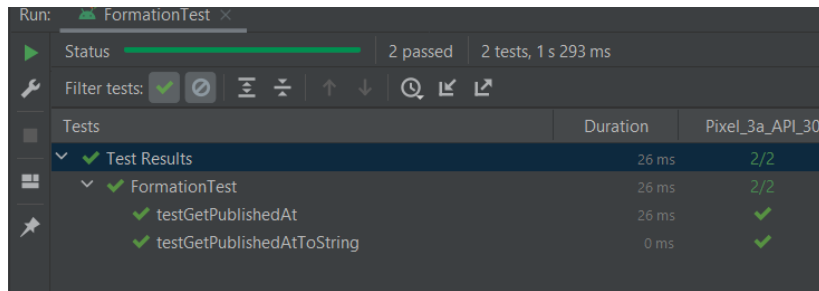


Figure 61: Résultats des tests unitaires de la classe FormationTest

### MesOutilsTest

Cette seconde classe de test permet de vérifier les résultats des conversions de **publishedAt** du type **Date** vers le type **String**, et inversement. Comme le montre la [figure 61](#), les propriétés à initialiser sont les mêmes que pour la classe de test **FormationTest**, à la différence que la propriété chaînée **format** est la suivante : « **yyyy-MM-dd HH:mm:ss** ».

```
/**
 * Classe de tests unitaires de la classe MesOutils
 */
public class MesOutilsTest extends TestCase {
    private final String strDate = "2020-08-07 05:59:10";
    private final String format = "yyyy-MM-dd HH:mm:ss";
    private final SimpleDateFormat formatter = new SimpleDateFormat(format);
    private Date date = new Date();

    {
        try {
            date = formatter.parse(strDate);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }

    private final Formation uneFormation = new Formation(13, date, "titre", "description", "miniature", "picture", "videoid");
}
```

Figure 62: Propriétés initialisées pour les tests la classe MesOutilsTest

On retrouve alors deux méthodes à tester **testConvertStringToDate()** et **testConvertDateToString()**. La première permet de tester si la chaîne fournie se convertit bien en **Date** au format fourni. Pour cela, une chaîne **leFormat** définit le format de la date puis **dataFormat** de type **SimpleDateFormat** est instancié avec **leFormat** en paramètre. Enfin, la méthode **assertEqual()** est utilisée afin de comparer la propriété globale **date** avec le résultat retourné par la méthode **convertStringToDate** avec pour paramètre la chaîne formatée **dateFormat.format(date)** et la chaîne du format attendu.

```
/**
 * Test unitaire vérifiant qu'un String est converti en Date au format donné
 */
public void testConvertStringToDate() {
    String leFormat = "yyyy-MM-dd HH:mm:ss";
    SimpleDateFormat dateFormat = new SimpleDateFormat(leFormat);

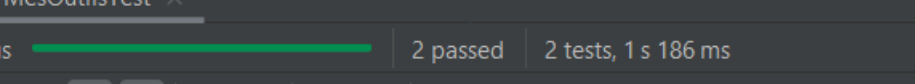
    assertEquals(date, MesUtils.convertStringToDate(dateFormat.format(date), leFormat));
}

/**
 * Test unitaire vérifiant qu'une Date est convertie en String au format donné
 */
public void testConvertDateToString() {
    String leFormat = "dd/MM/yyyy";
    SimpleDateFormat dateFormat = new SimpleDateFormat(leFormat);

    assertEquals(dateFormat.format(date), MesUtils.convertDateToString(date));
    assertEquals(dateFormat.format(date), MesUtils.convertDateToString(uneFormation.getPublishedAt()));
}
```

Figure 63: Tests unitaires de la classe MesOutilsTest

Il en va de même pour la seconde méthode de test **testConvertDateToString()**, une chaîne de format « **dd/MM/yyyy** » et un autre objet instance de type **SimpleDateFormat**. Néanmoins, la méthode **assertEquals()** est utilisée deux fois. Une première permettant de tester la comparaison des résultats entre la chaîne **dateFormat.format(date)** et le retour de la méthode **convertDateToString()** avec l'objet **date** en paramètre. La seconde est la même, à la différence que l'objet envoyé en paramètre à la méthode **convertDateToString()** est la chaîne renvoyée par **uneFormation.getPublishedAt()**. Il est alors possible de retrouver les résultats de test de la [figure 64](#) suivante.



The screenshot shows the Android Studio interface with the Run tab selected. The top bar indicates the test suite is 'MesUtilsTest'. Below this, a green progress bar shows the test status, with '2 passed' and '2 tests, 1 s 186 ms' displayed. A 'Filter tests' section contains several icons for filtering and sorting. The main area displays a table of test results:

Tests	Duration	Pixel_3a_API_30
✓ Test Results	26 ms	2/2
✓ MesUtilsTest	26 ms	2/2
✓ testConvertDateToString	26 ms	✓
✓ testConvertStringToDate	0 ms	✓

Figure 64: Résultats des tests unitaires de la classe MesOutilsTest

## Scénario

Un scénario de test du comportement de l'application a été établi avec **Junit 4** ainsi qu'avec les dépendances implémentés d'**Android espresso** pour les tests automatisés de l'interface, puisque ce dernier permet de simuler les interactions utilisateurs.

```

/**
 * Scenario testant les fonctionnalités de l'application
 */
@Test
public void scenario() {
    /// Vers les toutes les formations
    // MainActivity
    sleep( ms: 5000); // Permet de donner le temps au serveur Heroku de s'activer et à l'application de se connecter à la base de donnée distante

    onView(withId(R.id.btnFormations)).perform(click());
    assertEquals( expected: "all", Controle.getChoix());

    // FormationsActivity
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "4", closeSoftKeyboard());
    onView(withId(R.id.btnFiltre)).perform(click());
    for (int i = 0; i < 7; i++) {
        onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(1).onChildView(withId(R.id.btnListFavori)).perform(click());
    }
    onView(withId(R.id.txtFiltre)).perform(typeText( stringToBeTyped: "5", closeSoftKeyboard());
    onView(withId(R.id.btnFiltre)).perform(click());
    onView(withId(R.id.txtFiltre)).perform(clearText(), closeSoftKeyboard());
    onView(withId(R.id.btnFiltre)).perform(click());

    // UneFormationActivity
    onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(0).onChildView(withId(R.id.txtListPublishedAt)).perform(click());
    onView(withId(R.id.btnPicture)).perform(click());

    // VideoActivity
    onView(withId(R.id.wbvYoutube)).perform(click());
    onView(isRoot()).perform(ViewActions.pressBack());
}

```

Figure 65: Exemple du scénario de test établi

Ce dernier démarre sur la page d'accueil après avoir attendu un délais de 5 secondes afin que l'application reçoive bien le retour du serveur avec la méthode, un clic automatisé se déclenche sur le bouton pour aller dans les formations avec la méthode :

```
onView(withId(R .id.btnFormation)).perform(click()) ;
```

La page des formations s'ouvre alors permettant de tester la fonction de filtre avec la méthode :

```
onView(withId(R .id.btnFiltre)).perform(typeText(«4 »), closeSoftKeyboard()) ;
```

puis actionne le bouton du filtre. Une formation de la liste peut être sélectionnée avec la méthode suivante :

```
onData(anything()).inAdapterView(withId(R.id.lstFormations)).atPosition(i).onChildView(withId(R.id.txtListPublishedAt)).perform(click())
```

Où la variable i désigne la ligne de la liste à sélectionner.

Avec ces différentes méthodes il est possible de passer d'Activity en Activity et d'interagir avec les différents menu. Différents tests sur les filtres sont effectués ainsi que de sélection de favoris parfois en « spammant ». D'ailleurs, une boucle incrémentée a été mise en place afin de sélectionner plusieurs fois des formations où l'id est choisit via un nombre au hasard avec la méthode **random()**. Le même ensemble de test est effectué autant pour tester la page des formations que celle des favoris. Enfin, le résultat de ce scénario est visible sur la [figure 66](#).

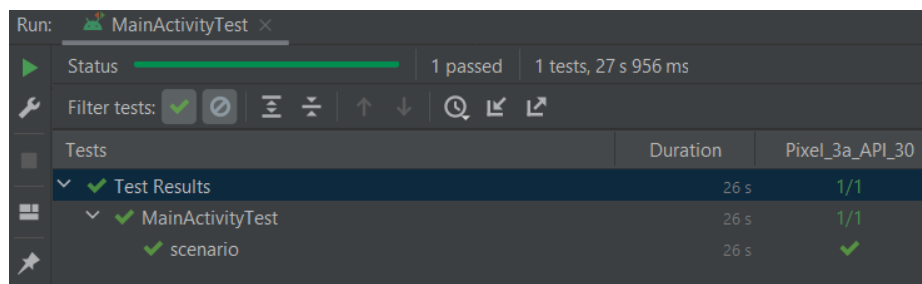


Figure 66: Résultat du scénario de test

## Contrôler la qualité du code avec SonarLint et SonarQube

### SonarLint

**SonarLint** étant un plugin pouvant être intégré sur AndroidStudio, la grande majorité des morceaux de codes mal écrits ou d'erreurs de sécurité peuvent être rapportés directement dans un console dédiée.

Ainsi, plusieurs alertes comme celle illustrée en [figure 67](#) on était directement gérer. Sur cette figure, l'alerte signale que la méthode d'écoute sur un bouton avec **setOnClickListener()** doit être mise sous forme de fonction lambda.

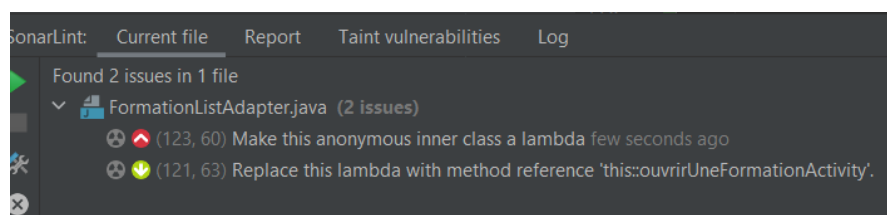


Figure 67: Exemples d'alertes traitées avec le plugin SonarLint

Aussi, pour certaines de ces fonctions lambda, la seconde alerte demande de modifier certaines fonctions lambda par un appel de méthode référencé. C'est pourquoi l'ensemble des méthodes appelant **setOnClickListener()** ont été mise soit sous cette dernière forme, soit sous forme de fonction lambda. Ceci est principalement visible dans la vue **FormationListAdapter**.

Néanmoins SonarQube permet d'apporter une plus grande précision et éventuellement de relever davantage d'erreurs de code, notamment certaines relevant de la sécurité.

### Mise en place de SonarQube

Pour utiliser cette plateforme, il convient de télécharger l'application puis de l'installer dans le dossier racine **C** : du poste de travail. Ensuite, il suffit de lancer le fichier **StartSonar.bat** afin de lancer le serveur SonarQube local, de se rendre à l'adresse **localhost:9000** dans un navigateur, puis de se connecter avec login et mot de passe admin. Par la suite, il faut créer un nouveau projet ici : **MediaTek86Formations** et de le créer avec les options en local. Puis, il faut générer un token, sélectionner le type de projet, ici gradle puis d'exécuter le script affiché dans la console de l'IDE.

Néanmoins, une petite modification doit être effectuée dans le script puisque le poste de travail actuel est windows10. Celui fournit est le suivant :

```
./gradlew sonarqube \
-Dsonar.projectKey= MediaTek86Formations \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.login={{montoken}}
```

Alors qu'il est nécessaire d'avoir :

```
.\gradlew sonarqube -Dsonar.projectKey=MediaTek86Formations -
Dsonar.host.url=http://localhost:9000 -Dsonar.login= {{montoken}}
```

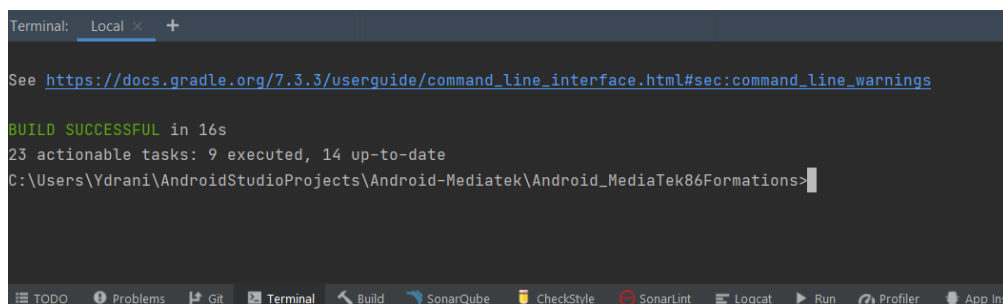


Figure 68: Résultat de la console intégrée par AndroidStudio après exécution d'une analyse SonarQube



Cette commande lance une première analyse et permet d'avoir la fenêtre suivante, *figure 69*.

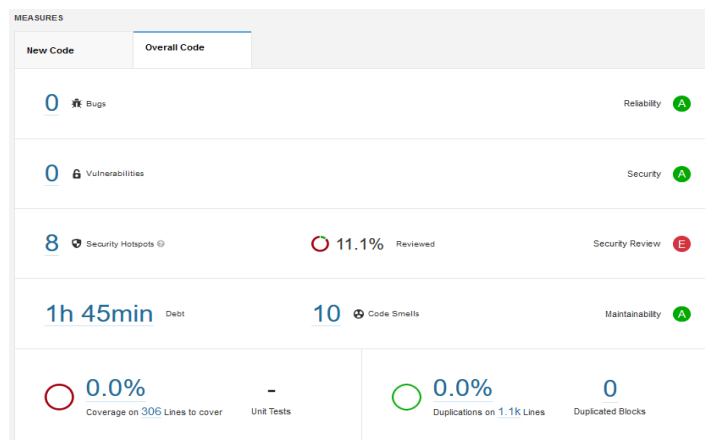


Figure 69: SonarQube première analyse du code du projet

Il est alors possible de constater de nouvelles alertes dont 8 notées comme alertes de sécurité et 10 de mauvaises utilisation du code. Néanmoins, la majorité de ces erreurs peuvent être considérées comme des faux positifs puisqu'elles relèvent de obsolescence de la classe **AsyncTask** et de ses méthodes.



Figure 70: Exemples d'alertes relevés par SonarQube.

La résolution de ces alertes se fait individuellement, en cliquant sur le message de l'erreur. Aussi, il est possible d'avoir une explication de l'alerte avant d'estimer s'il faut la réparer ou non. Ce choix se fait par l'intermédiaire du bouton Open visible sur chacun des messages de la figure 69. Il est possible de constater qu'une erreur relevant de l'utilisation de l'extension Comparable est détectée. Celle-ci demande de réécrire aussi la méthode **equals()** pour respecter la mise en place avec **CompareTo()**. N'ayant ici besoin que de **compareTo()** cette alerte est notée comme « à ne pas réparer » puisque seule la comparaison des valeurs nous intéresse dans l'usage de cette méthode.

Concernant les alertes de sécurité, la plus importante relevée concerne l'activation du langage **Javascript** dans la vue **VideoActivity** permettant ainsi d'avoir accès directement aux vidéos présentes sur la plateforme Youtube. Cette alerte informe que cela peut présenter une faille de sécurité de type **XXS** permettant d'injecter du contenu dans une page web, cf *figure 71*.

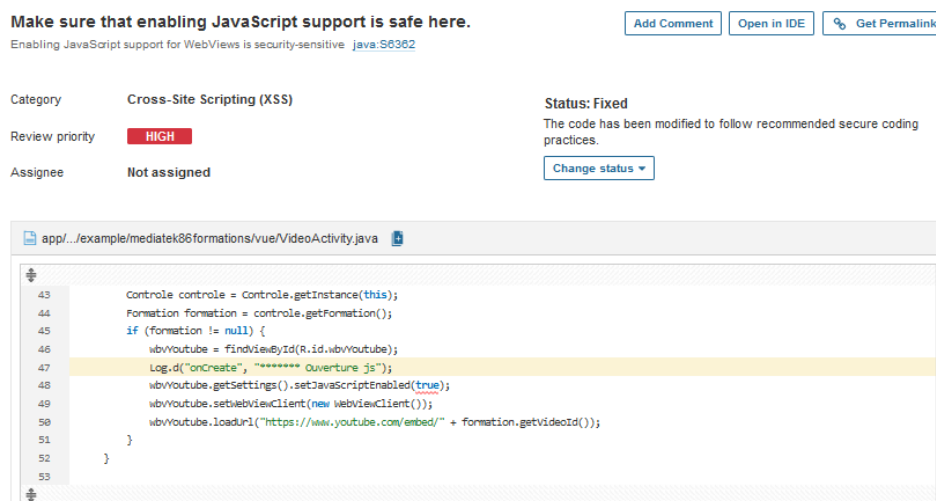


Figure 71: Exemple d'alerte de SonarQube pour la faille de sécurité XSS

La ligne relevée est la suivante :

```
wbvYoutube.getSettings().setJavaScriptEnabled(true);
```

Cette ligne étant exécutée lors de la méthode `init()` de la classe `VideoActivity` appelé par `onCreate()`, la correction de cette alerte a demandé l'écriture d'une nouvelle méthode `onDestroy()` qui va alors s'exécuter à la fermeture de l'activity. Cette dernière demande alors la désactivation de l'utilisation de Javascript en mettant la valeur `setJavaScriptEnabled` sur `false`, cf *figure 72*.

```

/**
 * Methode appelée à la fermeture de l'activité
 */
@Override
protected void onDestroy() {
    super.onDestroy();
    if (wbvYoutube.getSettings().getJavaScriptEnabled()) {
        wbvYoutube.getSettings().setJavaScriptEnabled(false);
        Log.d( tag: "onDestroy", msg: "***** Fermeture js");
    }
}

```

Figure 72: Méthode onDestroy() de la classe VideoActivity

Enfin, la dernière alerte notable est celle concernant la valeur true pour la propriété usesCleartextTraffic dans le fichier **AndroidManifest.xml** de l'application. En effet, cette propriété ayant été volontairement ajouté avec cette valeur, l'alerte d'encryptage de données sensibles relevé sera considérée comme sûre, d'autant plus qu'il n'y a pas vraiment de traitement de données sensibles dans cette application.

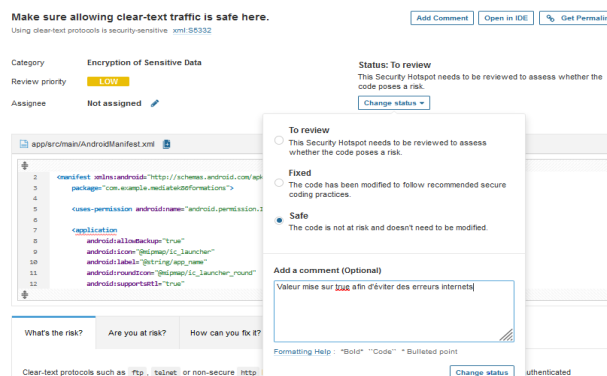


Figure 73: Alerte de sécurité sur l'encryptage de données sensibles

Finalement, il a été possible d'obtenir les notes suivantes :

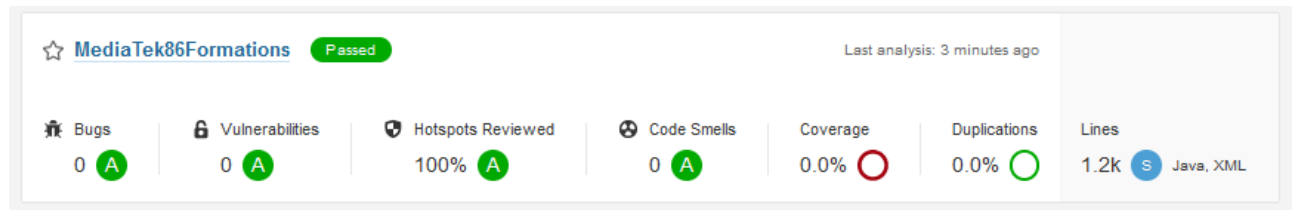


Figure 74: Mesures des différents points d'alerte de SonarQube

## Doctechnique

La documentation technique générée avec Doxyfile est visible à l'adresse [suivante](#)

## Déploiement

### Heroku

La plateforme de déploiement choisie est [Heroku](#) qui permet la mise en ligne de site web et d'applications web. Pour mettre l'API REST en ligne, il suffit de créer gratuitement un compte, de créer une nouvelle application et de le lier au compte du dépôt GitHub.

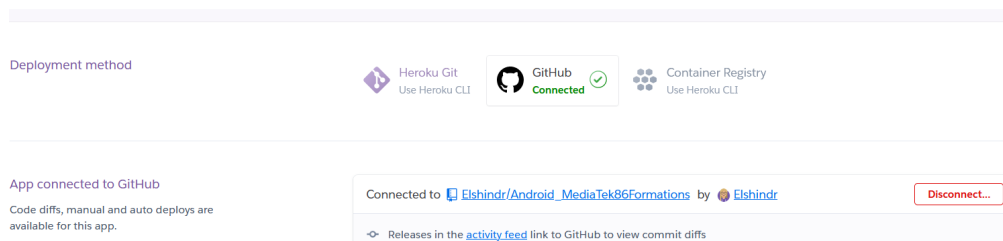




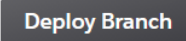
Figure 75: Fenêtre de liaison au dépôt GitHub pour le déploiement sur Heroku

Dans cette même fenêtre, il est possible d'avoir accès au bouton de déploiement manuel et automatique, c'est à dire qu'à chaque nouveau commit sur le dépôt du projet GitHub lié, il est possible de mettre en place un déploiement automatique ou bien de le faire manuellement en appuyant sur « Deploy Branch », cf figure 76.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more.](#)

Choose a branch to deploy

 master  

Receive code from GitHub 

Build **master** 7fd43e32 

Release phase 

Deploy to Heroku 

Your app was successfully deployed.

 [View](#)

Figure 76: Fenêtre de déploiement de la branche

### Mise en place composer

Afin que ce déploiement fonctionne, plusieurs manipulations sont à effectuer. Puisque l'API est écrite en PHP, alors il est nécessaire de créer un fichier composer.json dans le dossier racine du projet afin qu'il soit reconnu par Heroku comme projet PHP. Il est aussi possible d'utiliser composer par l'intermédiaire de la console si ce dernier est installé sur le poste. Ce fichier composer doit contenir les lignes suivantes :

```
{
  "require": {
    "symfony/apache-pack": "^1.0"
  }
}
```

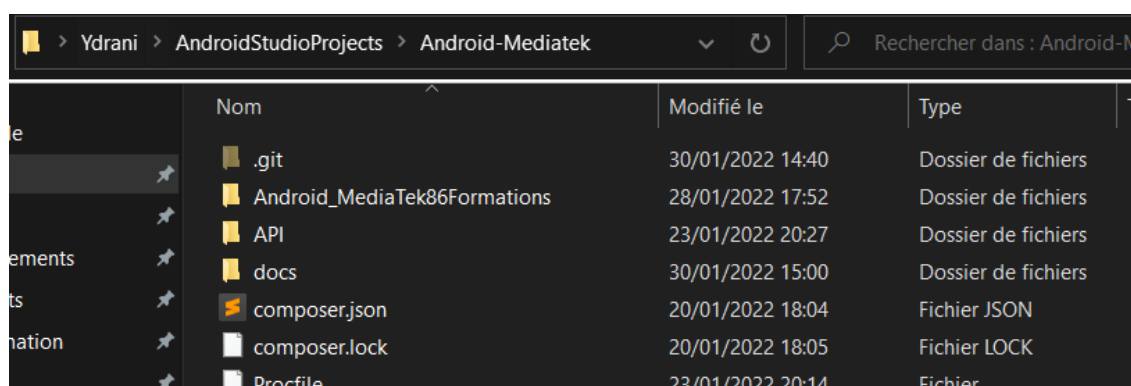


Figure 77: Fenêtre du dossier contenant le projet de l'application et l'API

Le second fichier a créé est le fichier Profile sans extension qui va permettre à Heroku de s'exécuter et où, ici dans le dossier API qui contient le code de l'API fournie. Ce fichier contient donc la ligne suivante :

```
web: vendor/bin/heroku-php-apache2 API/
```

## ClearDB MySQL

Enfin, une dernière modification dans l'application est effectuée dans la classe AccesDistant, il faut modifier l'adresse du serveur qui est désormais :

```
https://api-mediatekformations.herokuapp.com
```

```
/**
 * Propriété contenant la chaîne de l'adresse du serveur.
 */
private static final String SERVERADDR = "https://api-mediatekformations.herokuapp.com/";
```

Figure 78: Propriété SERVERADDR de la classe AccesDistant.java

La dernière étape consiste à mettre en place la base de donnée en ligne, pour cela **Heroku** dispose de plusieurs add-ons dont **MySQLClearDB** qui sera utilisé à cet intention. Pour cela, il suffit d'aller dans la liste des add-ons afin de trouver l'add-on, puis de choisir la formule concordant aux besoins de l'application, ici « **Ignite** » qui est gratuite et suffisante pour les besoins actuels de l'application.

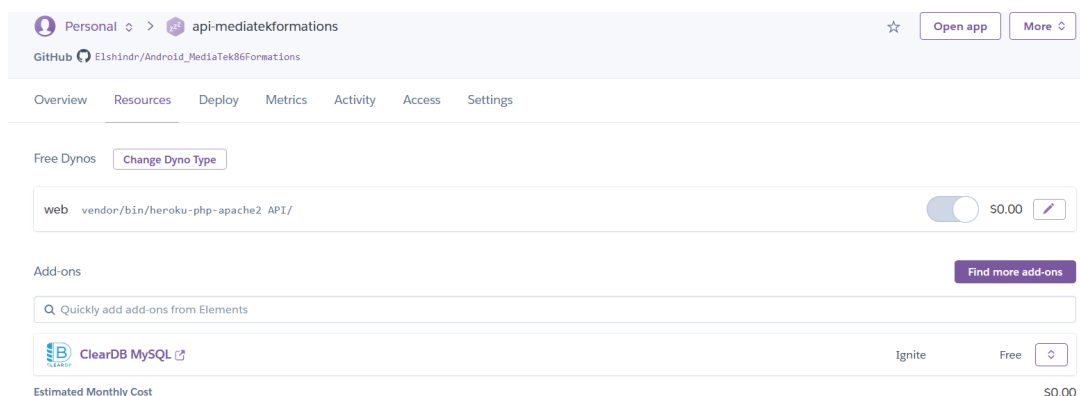


Figure 79: Fenêtre Heroku des addons

En cliquant sur « **ClearDB** », une nouvelle s'ouvre dans laquelle il est possible de trouver les différentes informations de connexion à la base de donnée en cliquant sur le nom de la base mise à disposition, cf *figure 80*.

Community Edition						
Dev & Production Edition						
Security						
How to Connect						
Name	Cloud	Region	Status	Type	Last Updated	Action
heroku_677e15f58da576c	Other	EU-West	Online	ignite	Sun Jan 23 2022 17:43:27 UTC	N/A

Figure 80: Fenêtre de gestion de base de donnée ClearDB

En cliquant sur le nom de la base, puis dans « **System Information** », il est possible de trouver les identifiants de connexion à la base, figure 81. Ces informations sont à remplacer avec celles qui définissaient l'accès local dans **AccesDistant.php**, cf *figure 82*.

Dashboard
Performance
Backups & Jobs
System Information
Support Cases

heroku\_677e15f58da576c: System Information

Below are your access credentials that you need to use in order to access your database. If you need assistance in determining your database URL, please check your Herod environment for details.

**Access Credentials**

Username: b723e57fc203e0  
Password: 5dd47000 (Reset)

Figure 81: Fenêtre de ClearDB contenant les informations de connexion

```

/**
 * Propriété qui contient la valeur du login pour la base de données ClearDB
 * @var String $login
 */
public $login = "b723e57fc203e0";

/**
 * Propriété qui contient la valeur du mdp pour la base de données ClearDB
 * @var String $mdp
 */
public $mdp = "5dd47000";

/**
 * Propriété qui contient la valeur du nom de base pour la base de données ClearDB
 * @var String $bd
 */
public $bd = "heroku_677e15f58da576c";

/**
 * Propriété qui contient la valeur du serveur de base pour la base de données ClearDB
 * @var String $serveur
 */
public $serveur = "eu-cdbr-west-02.cleardb.net";

/**
 * Propriété qui contient la valeur du port de base pour la base de données ClearDB
 * @var String $port
 */
public $port = "3306";

```

Figure 82: Classe AccesDistant.php informations de connexion à la base de donnée distante



Enfin, il faut remplir la nouvelle base de donnée. Afin d'y accéder avec phpMyAdmin il est nécessaire de modifier son fichier config.inc.php, figure 83, puis d'aller en bas du fichier afin de rajouter les données de connexion afin que le nouveau serveur soit disponible lors de la connexion à phpMyAdmin

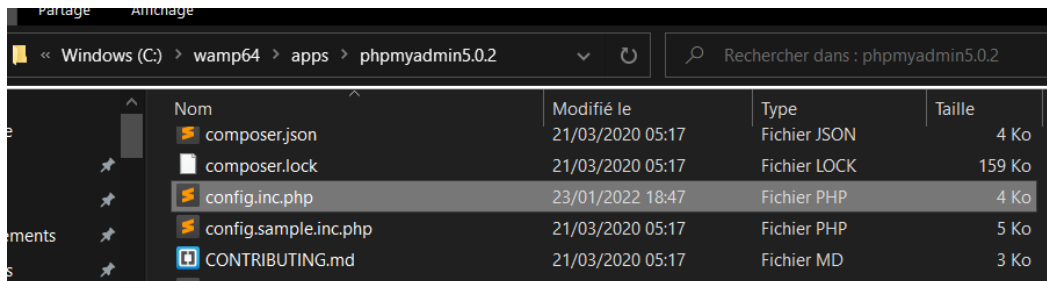


Figure 83: Localisation du fichier config.inc.php

Ainsi, il faut rajouter le contenu suivant :

```
$i++;
$config['Servers'][$i]['host'] = 'eu-cdbbr-west-02.cleardb.net'; //provide hostname
and port if other than default
$config['Servers'][$i]['user'] = 'b723e57fc203e0'; //user name for your remote
server
$config['Servers'][$i]['password'] = '5dd47000'; //password
$config['Servers'][$i]['auth_type'] = 'config'; // keep it as config
```

Après avoir sauvegardé ce fichier, il est alors possible d'accéder au serveur de la base de donnée via l'interface de connexion de phpMyAdmin, figure 84. Enfin, la base de donnée peut être rempli comme précédemment via le menu d'importation de script SQL.

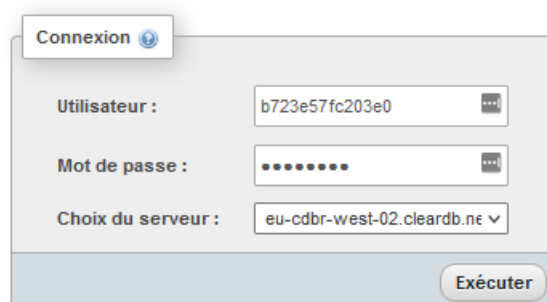


Figure 84: Interface de connexion à phpMyAdmin sur le nouveau serveur

## Heroku mode sommeil

L'un des inconvénient d'Heroku en mode d'hébergement gratuit est que au bout de quelques heures, l'application se met en mode sommeil, à savoir que le serveur mis en ligne est désactivée. Néanmoins, il suffit de rouvrir le lien du serveur pour qu'il se réactive, cela va juste demander quelques secondes d'attente. Ceci est détaillé dans la vidéo de présentation de l'application.

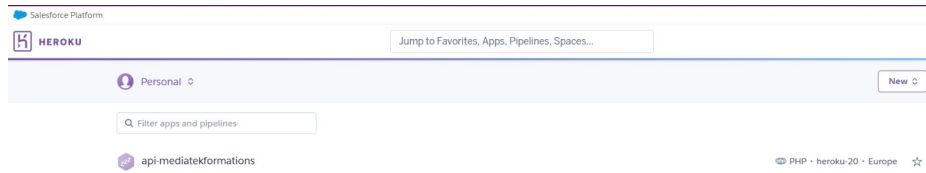


Figure 85: Projet sur Heroku en sommeil

Cela peut poser problème dans notre application puisque le délais de réponse du serveur est plus long, ainsi cela peut retourner une erreur du serveur alors que ce dernier fonctionne parfaitement. Afin de contourner ce problème une méthode **Handler()** a été mis en place dans la méthode **getInstance()** de la classe **Controle.java**.

Ce dernier permet de donner un délais supplémentaire à l'exécution du code présent dans cette méthode **Handler()** quelques secondes afin que le serveur retourne correctement sa réponse.

```
* @param context Context
* @return instance
*/
public static Controle getInstance(Context context) {
    if (Controle.instance == null) {
        Controle.instance = new Controle();

        Handler handler = new Handler(Looper.getMainLooper());
        handler.postDelayed(() -> {
            accesDistant = new AccesDistant();
            accesDistant.envoi(operation: "tous", lesDomaines50% null);
        }, delayMin: 2000); // 2 secondes

        accesLocal = new AccesLocal(context);
        lesFavoris = accesLocal.recup();
    }

    return Controle.instance;
}
```

Figure 86: Méthode getInstance() de la classe Controle.java

# Bilan sur les objectifs atteints

## Finalités

Pour conclure, l'ensemble des missions à effectuer ont été accompli, la fonctionnalité de filtre est fonctionnelle aussi bien dans la page regroupant toutes les formations que celle des favoris. La gestion des favoris est aussi fonctionnelle et permet d'ailleurs de vérifier que les formations marquées comme favorites sont bien présentes dans la base de données distantes, dans le cas contraire la formation concernée est supprimée des favoris. Aussi dans les deux pages il est possible d'accéder à la page de description de la formation ainsi qu'à sa vidéo. De plus, ces pages peuvent être visitées en changeant l'orientation du téléphone de verticale à horizontale.

Cependant, deux détails sont à prendre en note. Ainsi lors du passage de la page de description d'une formation à celle de sa vidéo, il n'est pas possible de passer de l'une à l'autre par le mode horizontale. En effet, il n'y a qu'un écran grisâtre qui apparaît, néanmoins la vidéo peut quand même être visualisée à l'horizontale, à condition d'arriver sur la page de la vidéo en mode verticale puis de passer à l'horizontale.

Enfin, lorsque le serveur met trop de temps à répondre, il est tout de même possible d'accéder aux pages de formations et de favoris dont les listes seront vides. Il suffit de faire un retour arrière puis d'attendre que l'application ait reçue la réponse serveur. Bien que cela puisse être désagréable le délais d'attente n'est que de l'ordre de 5 secondes.

## Liste des compétences couvertes (B1, B2, B3)

### B1 Gérer le patrimoine informatique

- Recenser et identifier les ressources numériques
- Exploiter des référentiels, normes et standards adoptés par le prestataire informatique
- Vérifier les conditions de la continuité d'un service informatique

### B1 Répondre aux incidents et aux demandes d'assistance et d'évolution

- Traiter des demandes concernant les services réseau et système, applicatifs
- Traiter des demandes concernant les applications

### B1 Travailler en mode projet

- 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

### B1 Mettre à disposition des utilisateurs un service informatique

- Réaliser les tests d'intégration et d'acceptation d'un service
- Déployer un service
- Accompagner les utilisateurs dans la mise en place d'un serv

### B1 Organiser son développement professionnel

- Mettre en place son environnement d'apprentissage personnel
- Mettre en œuvre des outils et stratégies de veille informationnelle
- Gérer son identité professionnelle
- Développer son projet professionnel

### B2 Concevoir et développer une solution applicative

- Participer à la conception de l'architecture d'une solution applicative
- Modéliser une solution applicative
- Exploiter les ressources du cadre applicatif (framework)
- Identifier, développer, utiliser ou adapter des composants logiciels
- Exploiter les technologies Web pour mettre en œuvre les échanges entre applications, y compris de mobilité
- Utiliser des composants d'accès aux données
- 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
- Rédiger des documentations technique et d'utilisation d'une solution applicative
- Exploiter les fonctionnalités d'un environnement de développement et de tests

### B2 Assurer la maintenance corrective ou évolutive d'une solution applicative

- Recueillir, analyser et mettre à jour les informations sur une version d'une solution applicative

- Évaluer la qualité d'une solution applicative
- Analyser et corriger un dysfonctionnement
- Mettre à jour des documentations technique et d'utilisation de solution applicative

## **B2 Gérer les données**

- Développer des fonctionnalités applicatives au sein d'un système de gestion de base de données (relationnel ou non)
- Concevoir ou adapter une base de données
- Administrer et déployer une base de données

## **B3 Assurer la cybersécurité d'une solution applicative et de son développement**

- Participer à la vérification des éléments contribuant à la qualité 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