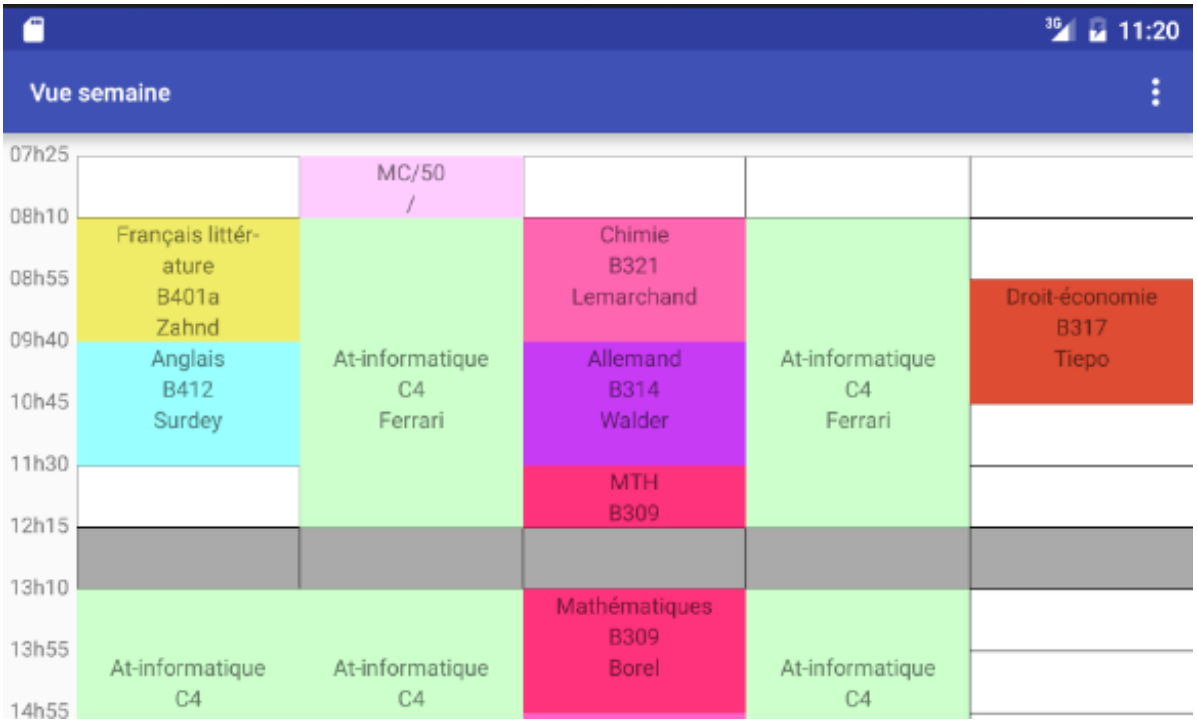
Hypercool Reader



Auteur	Murat Bayrakci
Date de début de projet	13.03.2017
Date de fin de projet	11.05.2017
Supérieur professionnel	Plinio Sacchetti
Experts	Principal : Nikles Jérôme Auditeur : Rota Renato
Lieu de travail	CPLN - B106B

Table des matières

1 Introduction

Le 13.03.2017 marque le début de ce projet. J'ai 2 mois pour réaliser ce qui m'est demandé, et répondre au cahier des charges. Je m'appelle Murat Bayrakci de la classe 3m3i2 et je suis très intéressé par la programmation. J'ai initialement demandé un sujet portant sur le développement. Durant notre cursus au Cpln nous avons appris, à travers les différents modules, les bases de la programmation en C#. Depuis cette année, nous avons commencé en atelier, à travailler avec du Java pour développer des applications Android. Les 2 langages sont assez proches et ça nous a permis de nous adapter facilement.

J'ai reçu un cahier des charges me demandant de réaliser une application mobile, orientée pour les appareils fonctionnant avec le système d'exploitation Android.

Le but principal de l'application est simple : Afficher l'horaire des classes depuis un téléphone mobile Android. Je dois faire attention à l'ergonomie de l'affichage car actuellement, la lisibilité de l'horaire disponible sur le site internet du Cpln, n'est pas satisfaisante pour les appareils mobiles.

Pour développer, je dispose des postes du Cpln ayant l'environnement de développement Android Studio. J'ai utilisé cet IDE depuis le début de l'année. Les données de l'horaire des classes me sont envoyées depuis un site crée par un professeur de l'école. J'ai également le droit d'amener mon appareil Android personnel pour pouvoir effectuer des tests.

Au-delà du fait de rendre un bon projet pour passer l'année et ainsi terminer ma formation, je suis motivé à produire une application de qualité, pour qu'elle puisse éventuellement être utilisée par les élèves des prochaines années.

2 Documentation de développement

2.1 Explications détaillées du projet

Pour ce projet, j'ai reçu une liste des fonctionnalités principales qui devaient être présentes dans l'application. J'ai également une deuxième liste de choses à faire si le temps le permet.

Comme introduit au chapitre précédent, le but principal de mon application est d'afficher l'horaire des classes présentes au Cpln. Deux vues différentes me sont demandées. Tout d'abord, lors de l'ouverture de l'application, l'horaire du jour d'une classe doit être affiché. La deuxième vue doit permettre de visualiser l'horaire de la semaine (du lundi au vendredi). J'ai reçu un exemple de design graphique pour la vue jour dans le cahier des charges. Je pouvais m'en inspirer ou bien créer quelque chose de totalement différent. Pour la vue semaine, je n'ai pas reçu d'exemple, et c'est moi qui devais proposer un design au professeur.

Au lancement de l'application, le jour actuel doit automatiquement être inséré dans le champ date. Ceci permet d'éviter à l'utilisateur de le taper manuellement à chaque ouverture. Egalement au lancement de l'application, le nom de la dernière classe que l'utilisateur a recherchée doit être insérée dans le champ classe. Ces deux étapes permettent de lancer la recherche et d'afficher l'horaire directement après l'ouverture de l'application.

Dans la vue jour, quatre boutons doivent être présents pour modifier la date. Les deux boutons situés aux extrémités permettront d'ajouter/diminuer la date d'une semaine, et les deux boutons du milieu, d'ajouter/diminuer un jour. Dans la seconde vue, seulement deux boutons seront présents, chacun modifiant d'une semaine la date.

L'utilisateur doit pouvoir garder un historique de toutes les recherches effectuées. A chaque appui sur le bouton rechercher, l'application doit enregistrer le nom de la classe pour pouvoir l'afficher par la suite. Il doit pouvoir accéder à cette liste à l'appui d'un bouton historique.

Un menu doit être présent pour pouvoir changer de vue et également effacer les recherches déjà effectuées.

Des contrôles de saisies doivent être effectués pour ne pas laisser à l'utilisateur la possibilité d'entrer des classes vides ou non existantes dans le champ classe. Pareil avec le champ date qui doit uniquement laisser les dates valides. En cas d'erreurs, des messages clairs et précis doivent signaler le problème à l'utilisateur.

L'application doit s'adapter à la taille de l'écran de l'utilisateur et proposer un affichage correct dans toutes les circonstances.

Les points suivants sont des objectifs secondaires présents dans le cahier des charges ou ajoutés par le professeur.

Pouvoir naviguer d'un jour à l'autre (vue jour) ou d'une semaine à l'autre (vue semaine) avec un balayage de l'écran de gauche à droite et inversement.

Gérer efficacement la rotation de l'écran. Les boutons doivent s'adapter à la nouvelle orientation (portrait/paysage) et l'affichage de l'horaire ne doit pas être de qualité inférieure.

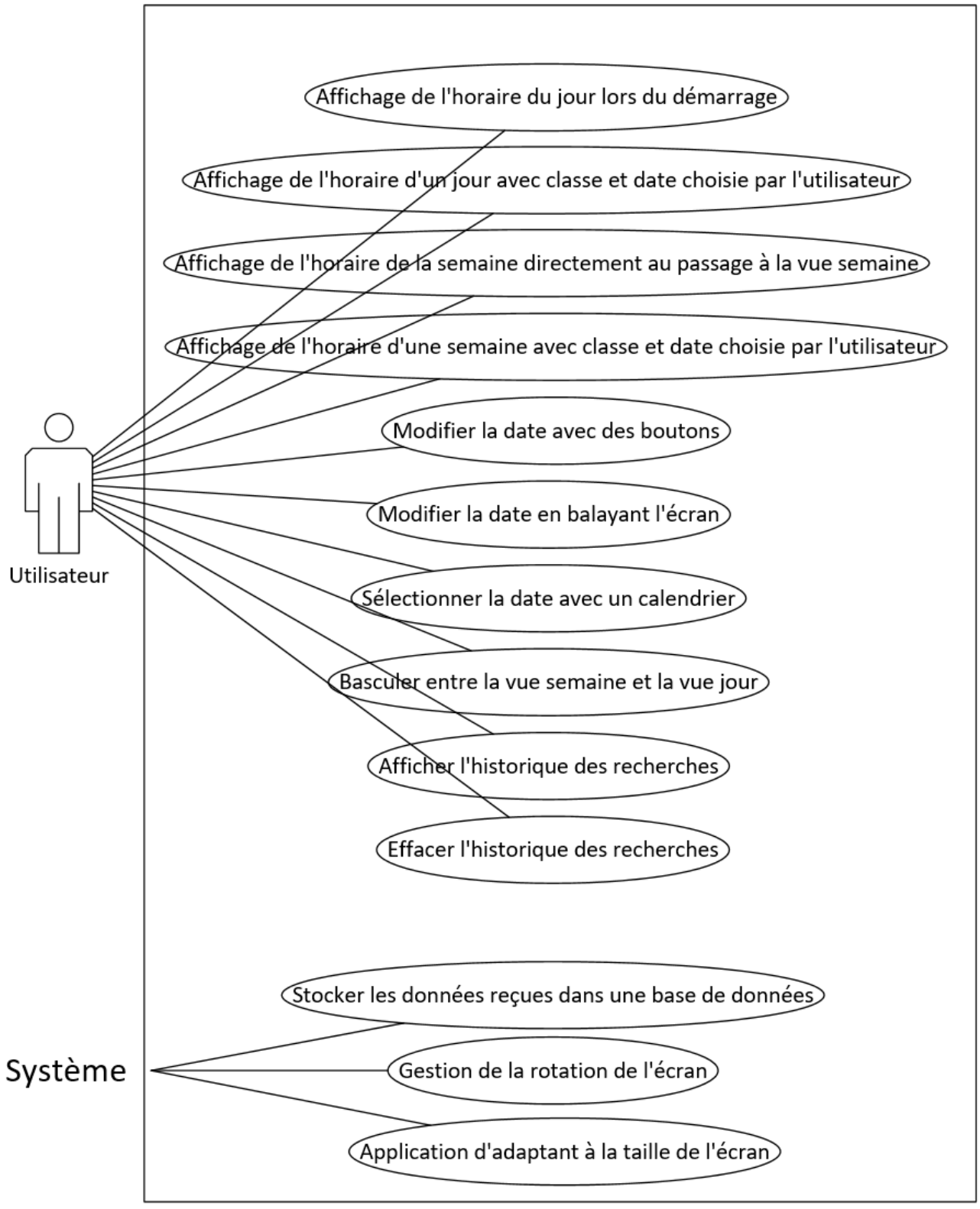
Stocker les données reçues dans une base de données. En cas d'inaccessibilité à internet, utiliser les données de cette base pour afficher tout de même l'horaire.

Proposer la liste de toutes les classes disponibles lorsque l'utilisateur est en train d'entrer une classe dans le champ classe. Convenu avec le professeur.

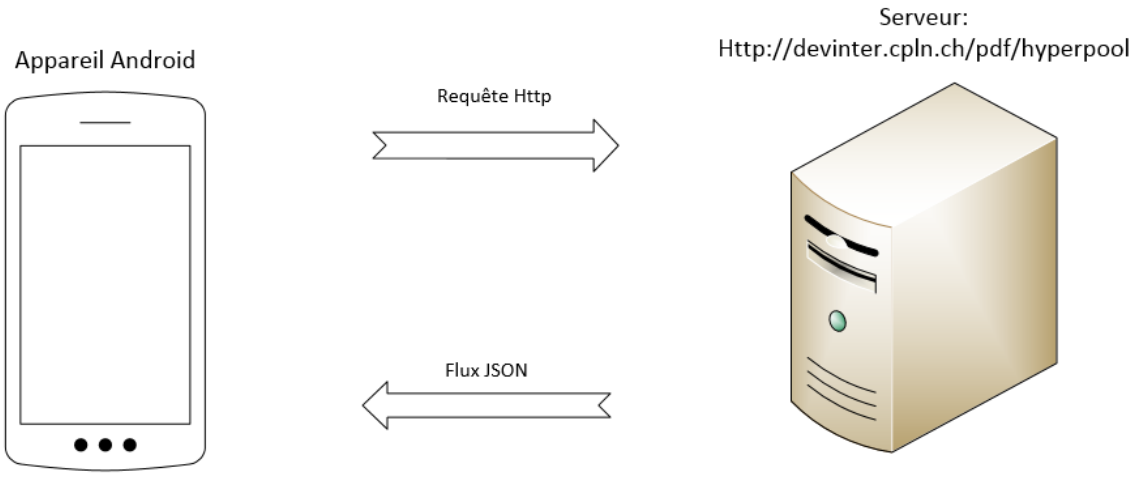
Prévoir un bouton qui va afficher un calendrier. L'utilisateur pourra alors choisir plus facilement la date désirée. Convenu avec le professeur.

La fonctionnalité suivante a été annulée au cours du projet. Le professeur m'avait demandé de lancer l'affichage de l'horaire, lorsque l'utilisateur quittait un des deux champs. Après une discussion avec lui et une partie de la classe, nous avons décidé de ne pas l'intégrer au produit final. Cela créerait plus de confusion, et des requêtes non voulues seraient lancées.

Diagramme des cas d'utilisation



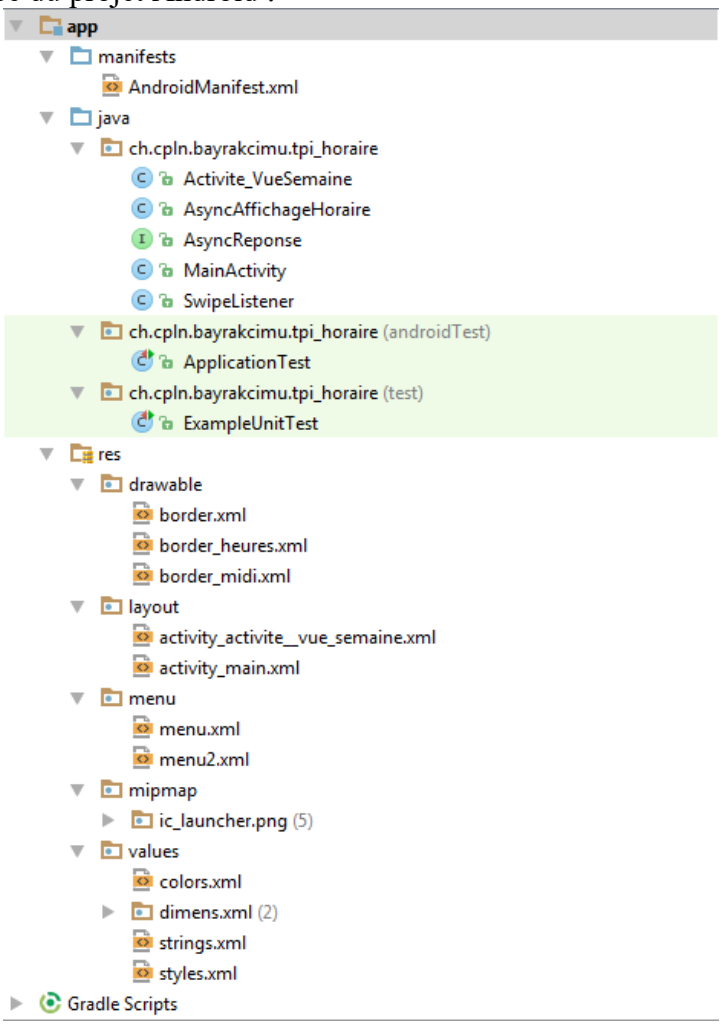
2.2 Architecture du système



1. L'application (appareil Android) va faire une requête au serveur avec des paramètres.
2. Le serveur va renvoyer les informations demandées sous format JSON.

2.3 Arborescence des fichiers

Voici l'arborescence du projet Android :



2.4 Définition *des conventions applicables*

Convention de nommage des variables :

Type	Convention
String	strExemple
Int	iExemple
Array	arrayExemple[]
Arraylist	AlistExemple
Float	flExemple
Boolean	bExemple
File	fExemple
Bouton	btnExemple
TextView	tvExemple
LinearLayout	llExemple
AutoCompleteTextView(Actv)	actvExemple
EditText	etExemple

Convention de nommage des activités :

Les noms des activités commencent par Activite_

Ex : Activite_VueSemaine

Convention de nommage des fonctions :

Première lettre des mots en majuscule.
Ex :

```
public void ExempleAfficher() {}
```

Si le premier mot est « get » ou « set », alors le premier mot est en minuscule, et les suivants en majuscules.
Ex:

```
public String getDerniereClasseRecherchee(){}
public void setDate(){}
```

2.5 Planning de livraison global

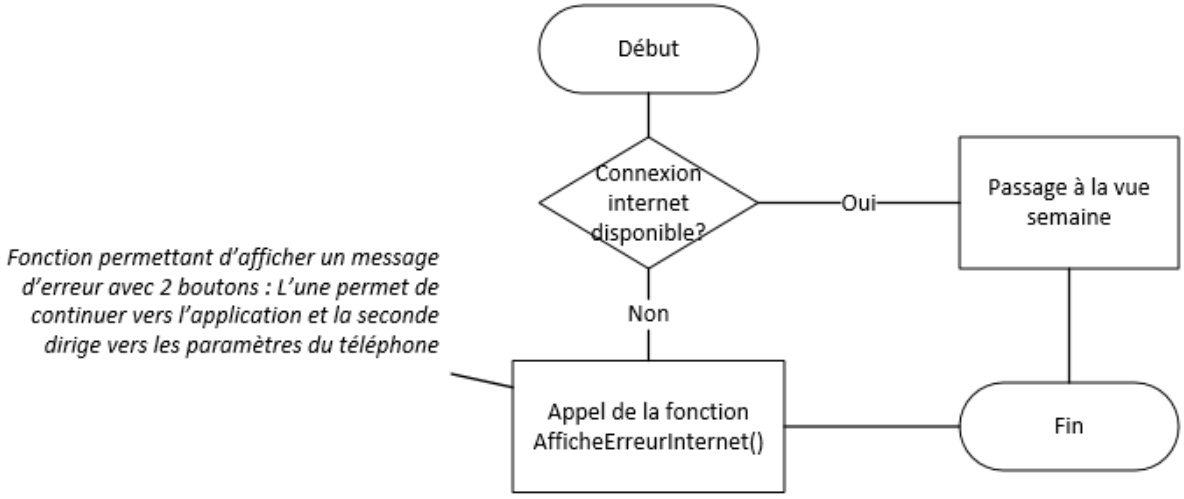
Planning initial et final en annexe.

2.6 Flux de Navigation

L'utilisateur peut librement naviguer entre les deux vues proposées.

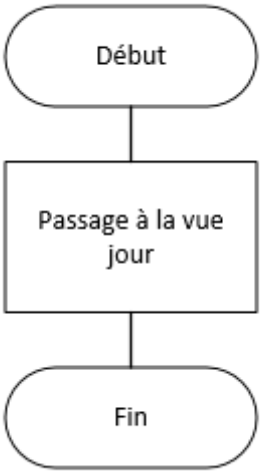
Vue Jour

Lors du clic sur le bouton « Vue Semaine » depuis la vue jour.



Vue Semaine

Lors du clic sur le bouton « Vue Jour » depuis la vue semaine.



2.7 Méthodologie

Pour pouvoir gérer le développement de mon application, j'ai utilisé le site GitHub. J'ai pu stocker mes données en ligne et faire du versioning.

Lien : https://github.com/MuratBayrakci/TPI_Horaire/tree/master

Le site Trello a également beaucoup été utilisé. Il permet de créer des listes de tâches à faire. Le professeur peut y accéder librement et ainsi voir l'avancement du projet.

Lien : https://trello.com/b/z8UjgyST/bayrakci-tpi

3 Réalisation des cas d'utilisation

Remarque : Certains cas d'utilisation sont exactement pareils dans les deux vues. Par exemple, le déroulement et l'explication de l'affichage du calendrier est similaire. Dans ces cas-là, les deux vues sont regroupées dans un seul cas d'utilisation.

3.1 Cas d'utilisation « Affichage de l'horaire du jour lors du démarrage »

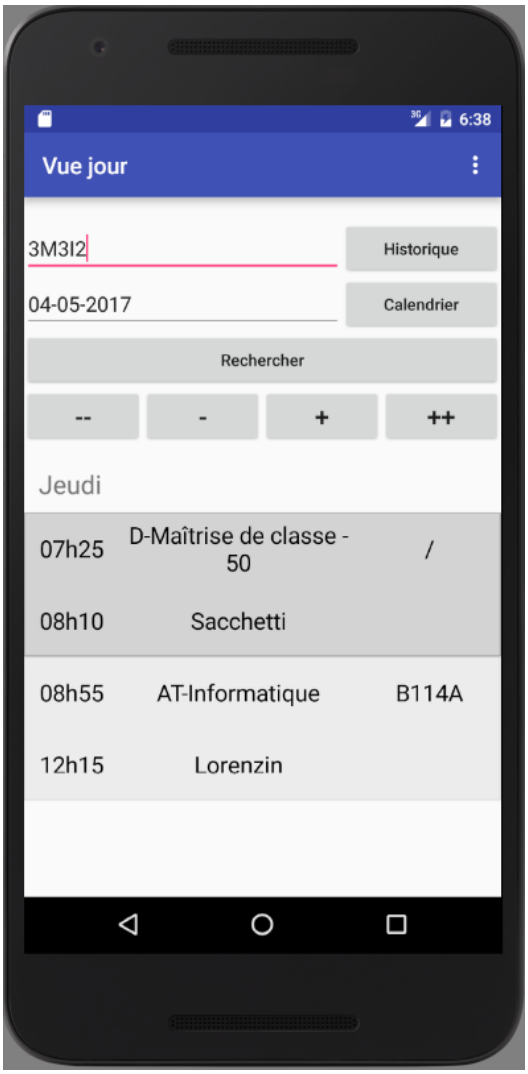
Ce cas d'utilisation est utilisé au démarrage de l'application. Il va servir à afficher l'horaire du jour actuel. La dernière classe recherchée va être insérée dans le champ classe. Ça permet à l'utilisateur de visualiser l'horaire utile à lui dès le démarrage de l'application.

3.1.1 Scénario

1. L'utilisateur ouvre l'application.
2. Le système vérifie la connexion internet.
3. Le système remplit le champ classe avec la dernière classe recherchée.
4. Le système remplit le champ date avec la date du jour actuelle.
5. Le système met dans des variables le contenu des champs, vérification des saisies.
6. Le système affiche le nom du jour de la semaine.
7. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toutes les classes).
8. Le système effectue des vérifications des données reçues.
9. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
10. Le système met le nom de la classe recherchée dans l'historique.
11. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
12. Le système met les données de l'horaire reçues dans des variables.
13. Le système tri les branches de manière chronologique à leurs heures de début.
14. Le système affiche les données.

3.1.2 Maquette

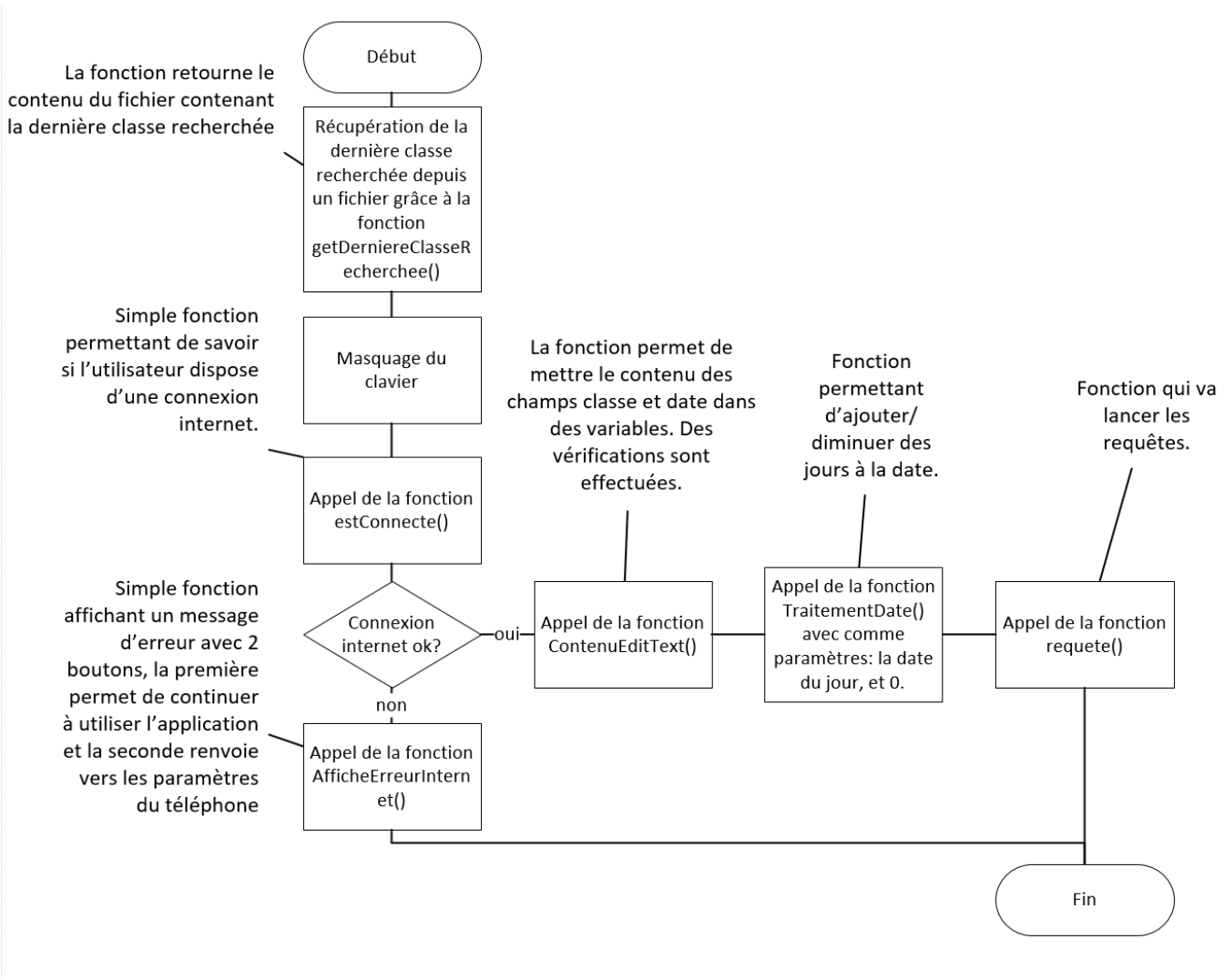
La maquette de la vue jour est très similaire à ce que j'avais imaginé, j'ai alors décidé de mettre directement le résultat final.



3.1.3 Analyse du scénario

3.1.3.1 Algorithme et explications

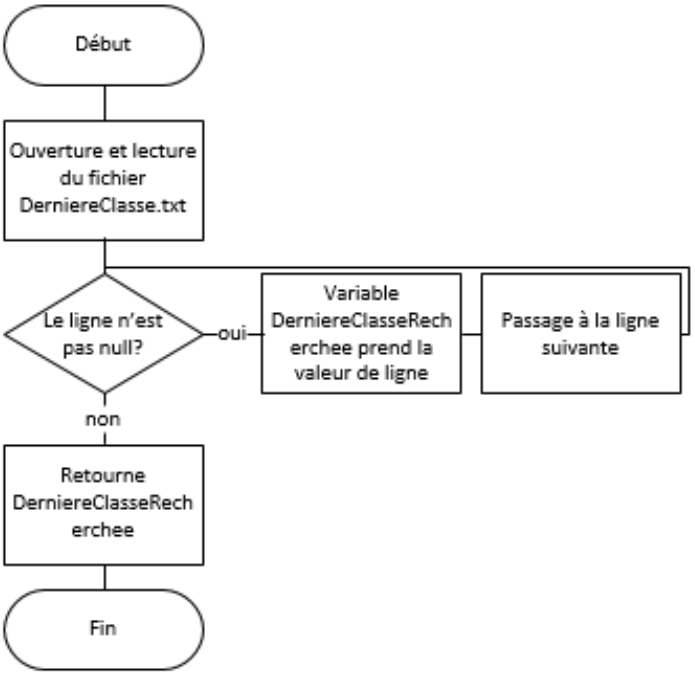
Lors de la première ouverture de l'application (onCreate()):



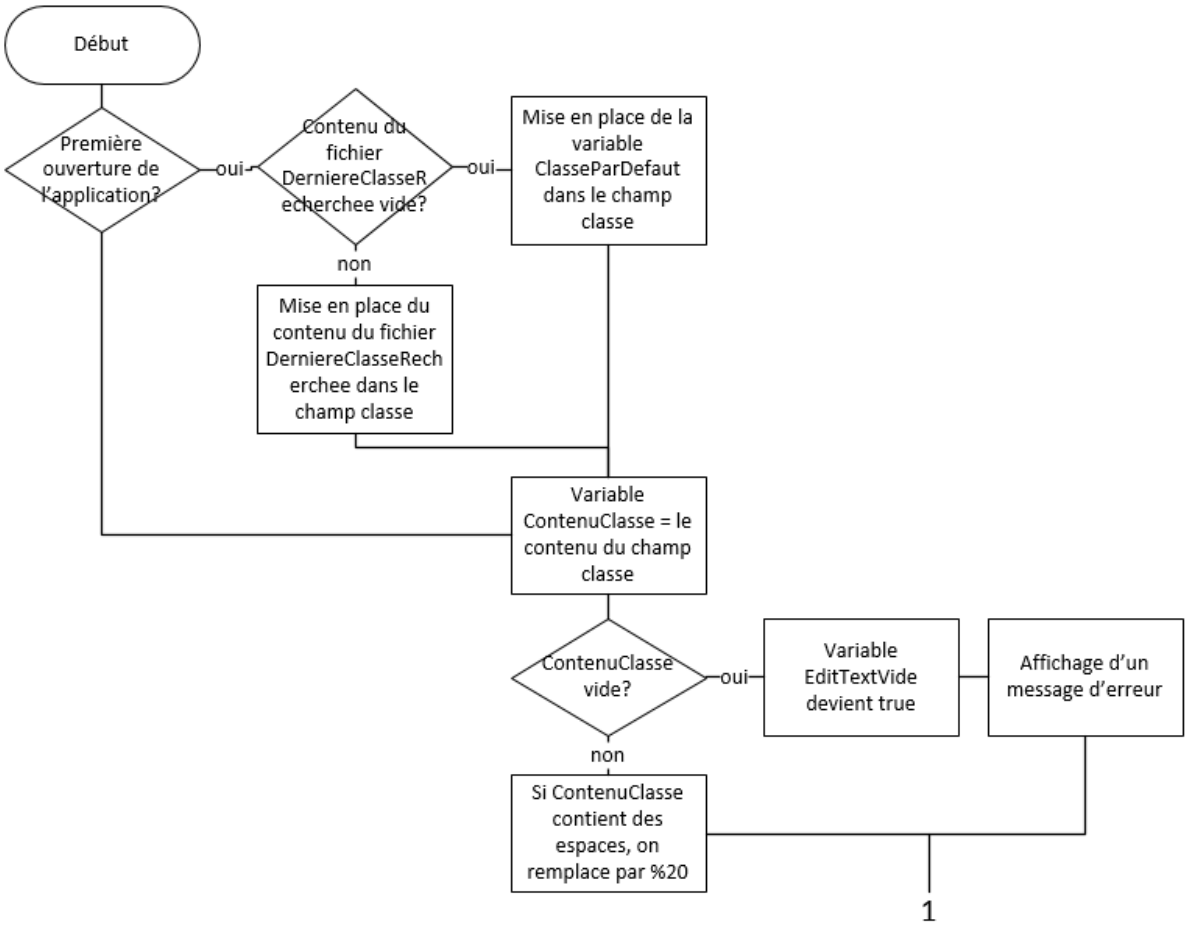
Au démarrage de l'application, ces actions vont être effectuées. Le nom de la dernière classe recherchée va être mis dans une variable afin d'être utilisé lors de la requête. Le clavier va être masqué pour ne pas gêner l'utilisateur. Je teste si la connexion internet est disponible et dans le cas contraire, j'avertis l'utilisateur.

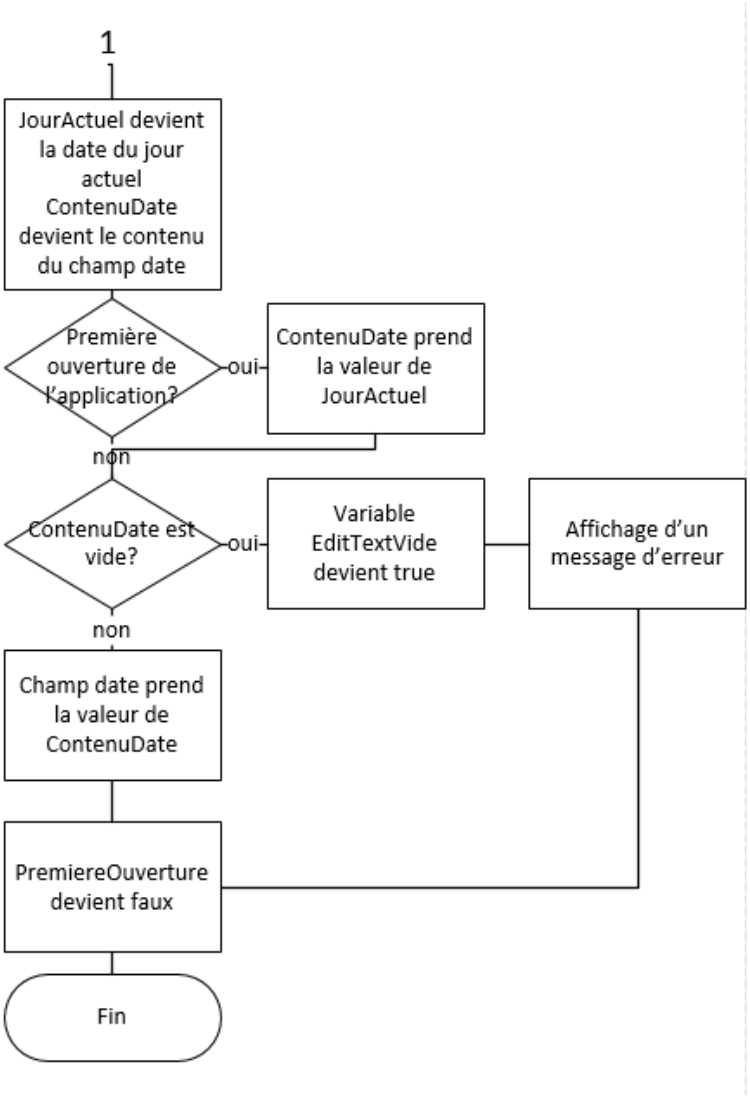
Si c'est ok, je fais appel à la fonction ContenuEditText() pour mettre dans des variables le contenu des champs. TraitementDate() est ensuite appelée. Elle va permettre d'informer l'utilisateur du nom du jour de la date (Lundi, Mardi…). Finalement les requêtes vont être lancées.

Fonction getDerniereClasseRecherchee() :



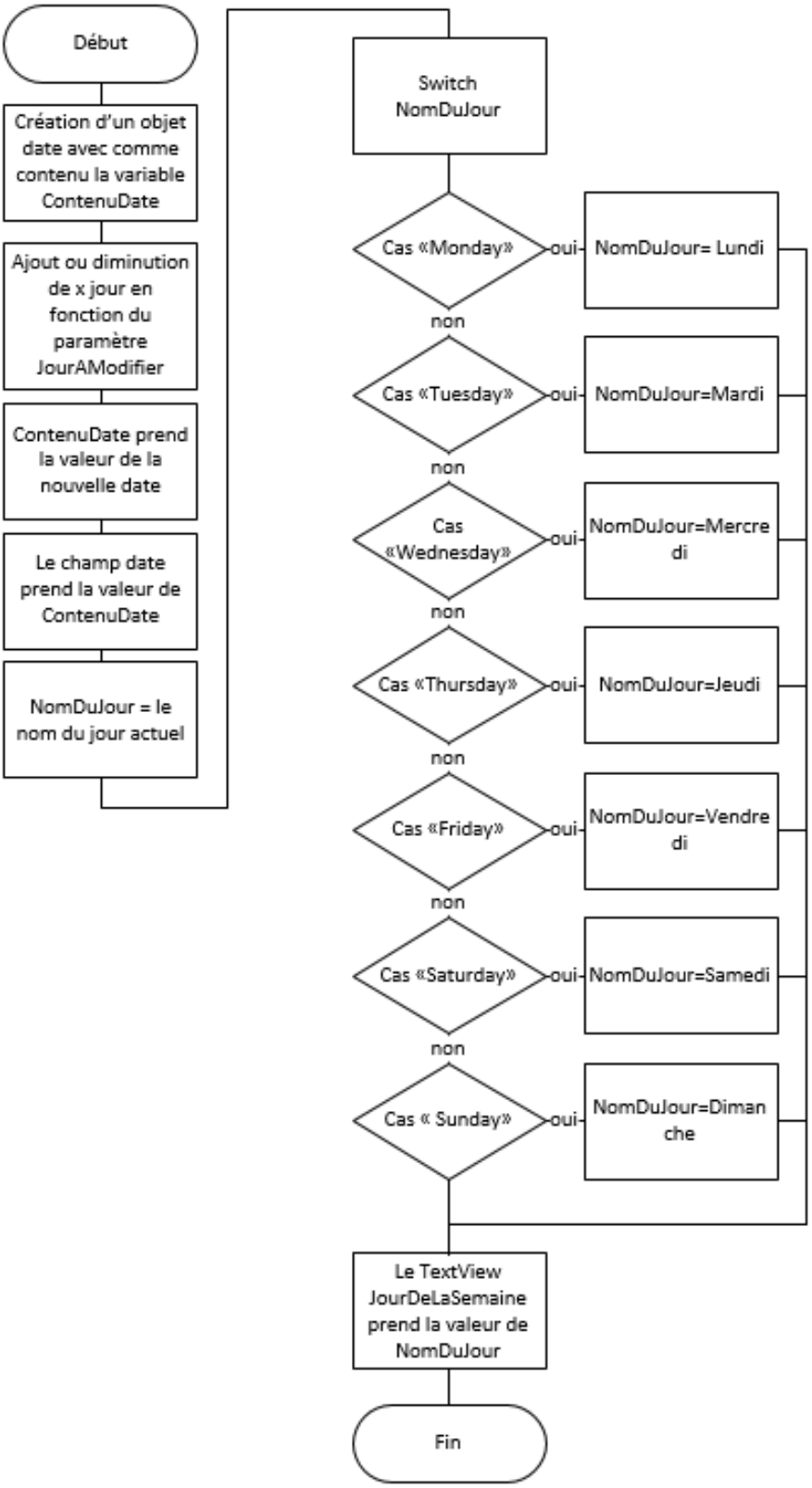
Fonction ContenuEditText() :





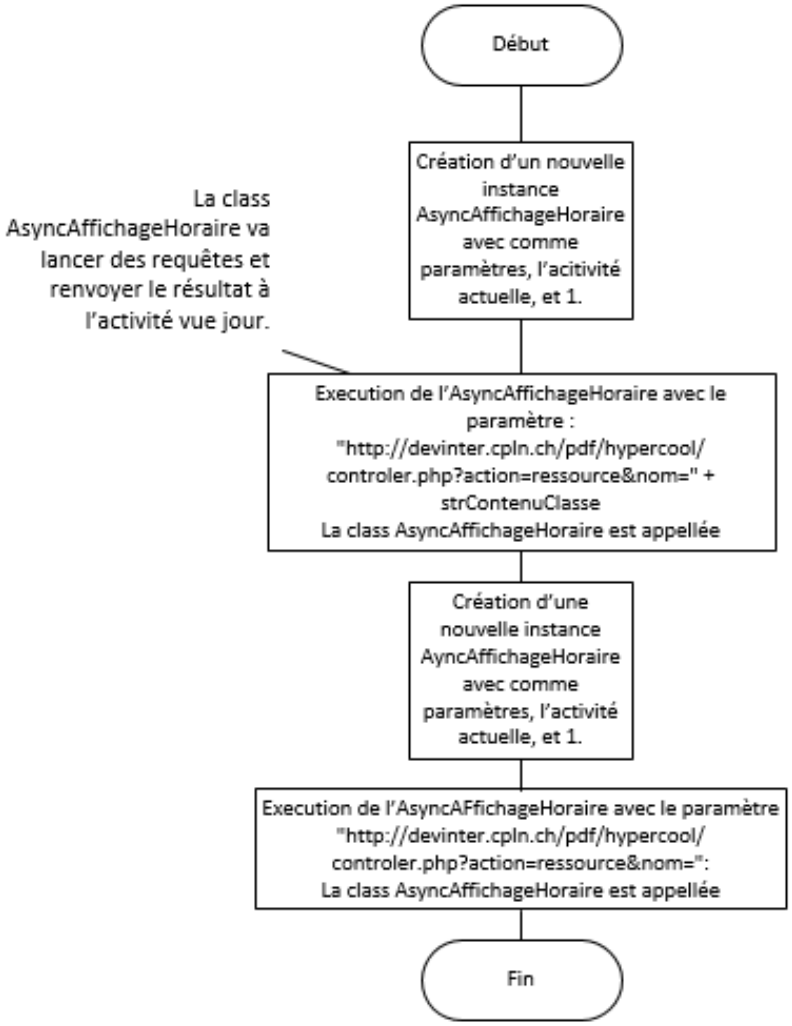
Cette fonction met le contenu des champs dans des variables. Lors du démarrage, la dernière classe recherchée et la date actuelle doivent être utilisées. Si le contenu de la dernière classe recherchée est vide, alors j'attribue une classe par défaut. Cette situation n'est pas censée se produire car le nom des classes vides ne sont pas mis dans le fichier. Néanmoins, ça permet d'éviter le crash de l'application si un bug survient. J'utilise la variable bPremiereOuverture pour savoir si cette fonction est appelée au démarrage ou ultérieurement. Cette variable passe à false à la fin de la fonction. Des tests sont effectués pour savoir si les champs sont vides, et dans ce cas bEditTextVide passe à true. Si le champ classe n'est pas vide, il faut remplacer tous les espaces du nom de la classe par des « %20 ». Certaines classes contiennent des espaces (ex : P-11 JET). Pour pouvoir faire les requêtes au serveur, il faut obligatoirement les remplacer par des %20.

Fonction TraitementDate() :



TraitementDate() sert à modifier la date de x jour(s). Le nombre de jour à modifier est envoyé en paramètre. Dans notre cas, j'envoie 0 et donc, je ne modifie pas la date. En revanche, le nom du jour de la semaine est mis dans un textview pour en informer l'utilisateur.

Fonction Requete() :



Avant d'expliquer le fonctionnement de cette fonction, je vais d'abord montrer comment s'effectue la manière pour récupérer l'horaire d'une classe. Pour pouvoir récupérer l'horaire d'une classe à une date précise, il faut tout d'abord obtenir son id.

http://devinter.cpln.ch/pdf/hypercool/controler.php?action=ressource&nom=NomClasse

Une requête de ce type doit être effectuée. NomClasse est remplacé par la variable contenant la classe.

Ce résultat va être reçu pour la classe 3m3i2 :


```
{"4354":{"nom":"3M3I2","prenom":"","code":"ET"}}
```

Après avoir obtenu l'id de la classe voulue, je dois faire une deuxième requête de ce type :

http://devinter.cpln.ch/pdf/hypercool/controler.php?action=horaire&ident=IdClasse&sub=date&date=DateVoulue

IdClasse est à remplacer avec l'id reçue précédemment, et DateVoulue, par la date souhaitée.

Voici le résultat de la requête pour la classe 3m3i2 à la date 04.05.2017 :

(Utilisation de http://jsonviewer.stack.hu/ pour faciliter la lecture des données JSON)


```
[
  {
    "codeMatiere": "MC\/50",
    "libelle": "D-Ma\u00eetrise de classe - 50",
    "indice": 1,
    "heureDebut": "07h25",
    "heureFin": "08h10",
    "classes": [
      "3INFC",
      "3INFD",
      "3M3I2"
    ],
    "professeur": [
      "Sacchetti Plinio [PSI]"
    ]
  },
  {
    "codeMatiere": "AT_SF",
    "libelle": "AT-Informatique",
    "indice": 1.43,
    "heureDebut": "08h55",
    "heureFin": "12h15",
    "classes": [
      "4M4I1C",
      "3M3I2",
      "3M3I3_Capocasale"
    ],
    "professeur": [
      "Lorenzin L\u00e9o [LL]"
    ],
    "salle": [
      "B114A-74431"
    ]
  }
]
```


Chaque branche est renvoyée avec différentes informations. Ces données vont être utilisées pour afficher l'horaire à l'utilisateur.

Pour revenir à la fonction : Je crée une nouvelle instance d'AsyncAffichageHoraire, avec en paramètre, le nom de l'activité, et 1. Le numéro envoyé sert à mettre en place ou non l'icône de chargement. J'exécute l'AsyncAffichageHoraire avec le lien pour récupérer l'id de la classe. La class AsyncAffichageHoraire est alors appelée. Une deuxième instance est créée avant d'être exécutée.
Le lien : http://devinter.cpln.ch/pdf/hypercool/controler.php?action=ressource&nom= permet de retourner toutes les id de toutes les ressources présentes sur le site (Classes, Professeurs, Salles…)

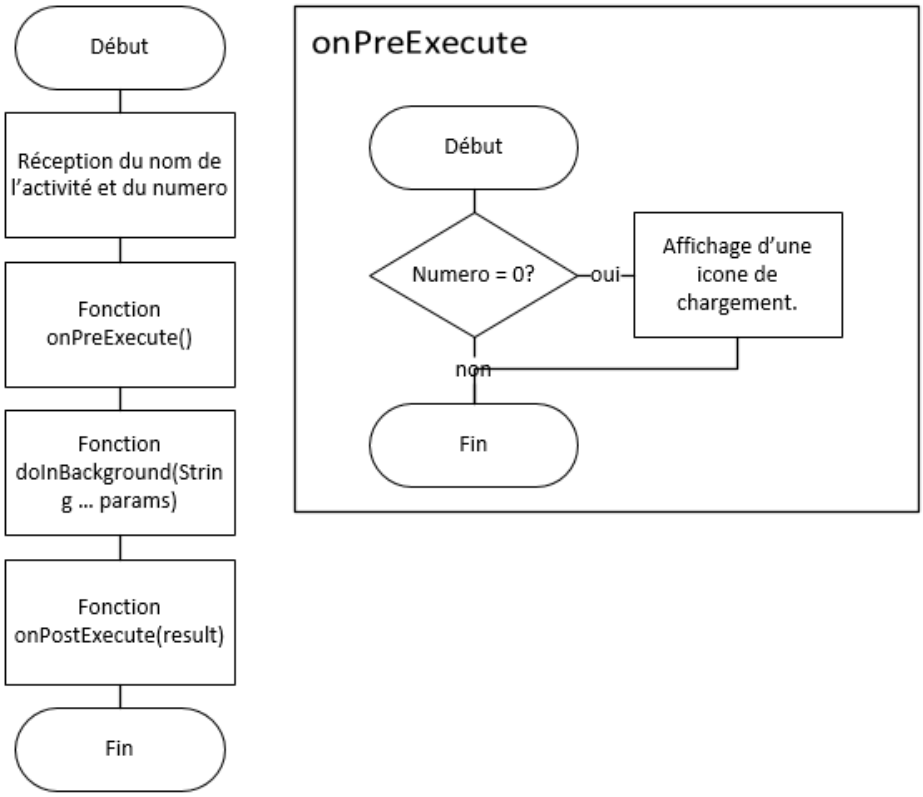
Une petite partie des données reçues :

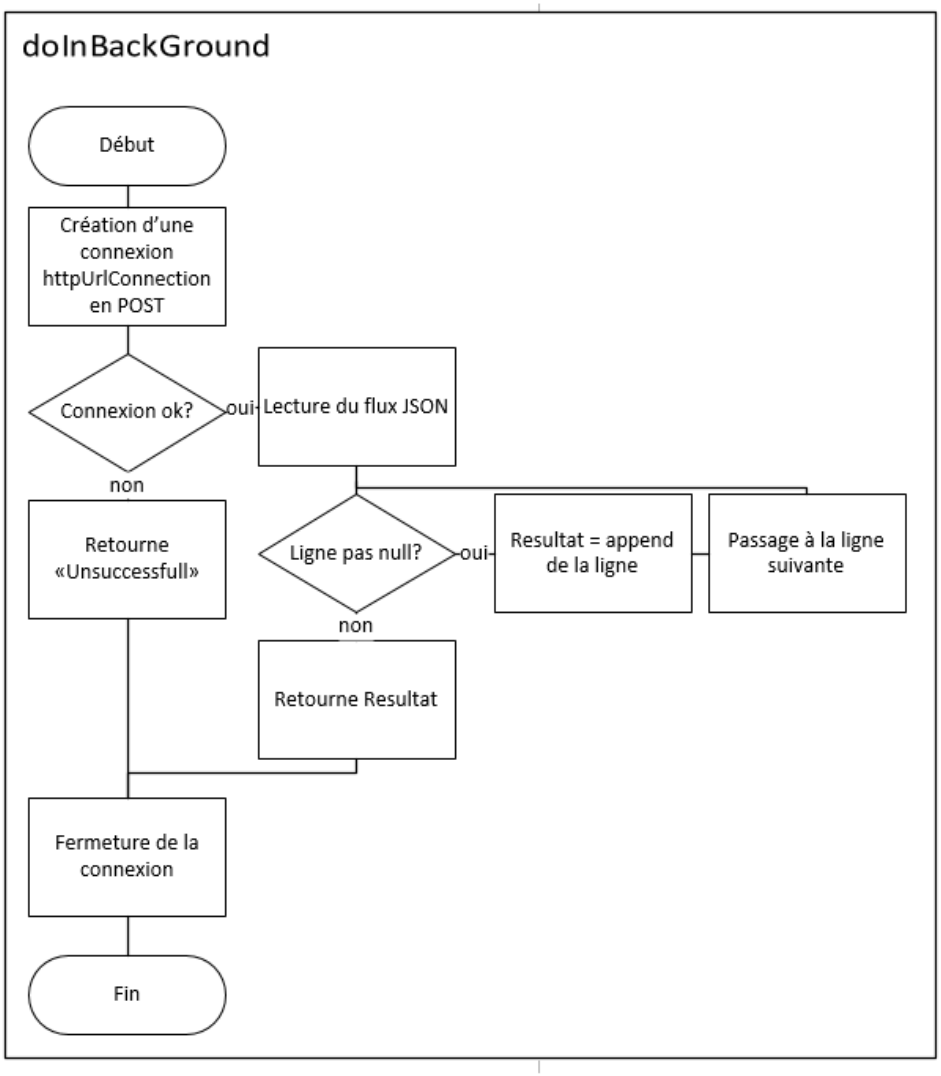

```
{
  "2922": {
    "nom": "1A-EX-A",
    "prenom": "",
    "code": "EAM"
  },
  "3910": {
    "nom": "1A-EX-B (virtuel)",
    "prenom": "",
    "code": "EAM"
  },
  "696": {
    "nom": "1ADA",
    "prenom": "",
    "code": "EPC"
  },
```

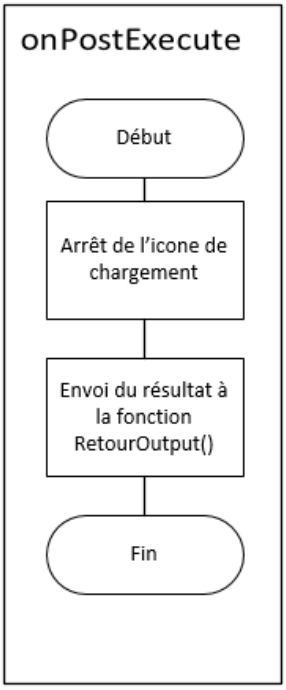

Cette requête va me permettre d'avoir le nom de toutes les classes disponibles. Ces noms seront stockés dans un tableau pour ensuite être suggérés à l'utilisateur.

Ces 2 requêtes sont exécutées simultanément.

Class AsyncAffichageHoraire() :




doInBackGround
Début
Création d'une connexion httpUrlConnection en POST
Connexion ok?
oui
Lecture du flux JSON
non
Retourne «Unsuccessfull»
Ligne pas null?
oui
Resultat = append de la ligne
Passage à la ligne suivante
non
Retourne Resultat
Fermeture de la connexion
Fin


onPostExecute
Début
Arrêt de l'icone de chargement
Envoi du résultat à la fonction RetourOutput()
Fin

Cette class va établir la connexion au serveur et ainsi exécuter les requêtes reçues en paramètre. Elle est utilisée par les 2 activités (Vue jour et Vue Semaine). Ce code a été repris de mon ancien projet qui consistait également à récupérer des flux JSON.

Cette class est de type asynchrone. Ça signifie que lors des requêtes, l'écran n'est pas bloqué et l'application ne « Freeze » pas.

Tout d'abord, je prends le numéro reçu en paramètre. Dans le onPreExecute(), j'affiche l'icône de chargement si le numéro est égal à 0. Dans les deux premières requêtes, ce numéro est égal à 1, donc l'icône n'est pas actionnée. Elle le sera lors de la troisième et dernière requête.

Remarque : Je voulais initialement afficher l'icône de chargement en même temps que la première requête. Malheureusement je n'ai pas réussi à mettre ceci en place. Soit l'icône tournait indéfiniment, soit elle ne s'affichait pas du tout. J'ai alors décidé de l'afficher uniquement lors de la dernière requête.

Dans le DoInBackGround(), on établit la connexion avec le serveur, et on envoie la requête. Le résultat reçu est envoyé à la fonction onPostExecute().

Dans cette fonction, l'icône de chargement (si elle existe) va être enlevée. Le résultat de la requête est renvoyé vers l'activité principale, dans la fonction RetourOutput().

Remarque importante : Pour pouvoir récupérer le résultat et l'utiliser dans l'activité principale, j'avais initialement fait d'une manière différente. Lors de l'exécution de la requête, je mettais un .get() pour récupérer le flux JSON reçu.

Ex :

```
Output =
asyncAffichageHoraire.execute("http://devinter.cpln.ch/pdf/hypercool/controler.php?
action=ressource&nom=" + strContenuClasse).get();
```

C'est une méthode simple que j'ai utilisée dans mes anciennes applications. A la moitié du projet, je me suis rendu compte (dans la vue semaine) que la tâche n'était plus asynchrone en utilisant cette manière. Durant la requête, l'application se figeait et c'était très désagréable pour l'utilisateur. J'ai alors dû trouver une nouvelle façon de transférer les données reçues de l'asyncAffichageHoraire vers l'activité principale. Après avoir cherché sur internet, un utilisateur proposait de créer une interface avec, à l'intérieur, la fonction qui allait recevoir le flux JSON. J'ai alors appliqué cette méthode et quelques modifications ont dues être faites.

Création de AsyncReponse.java

```
package ch.cpln.bayrakcimu.tpi_horaire;
public interface AsyncReponse {
    void RetourOutput(String output);
}
```

Ajout de cette ligne au début de la class AsyncAffichageHoraire

```
public AsyncReponse delegate = null;
```

Ajout de cette ligne dans le onPostExecute()

```
delegate.RetourOutput(result);
```

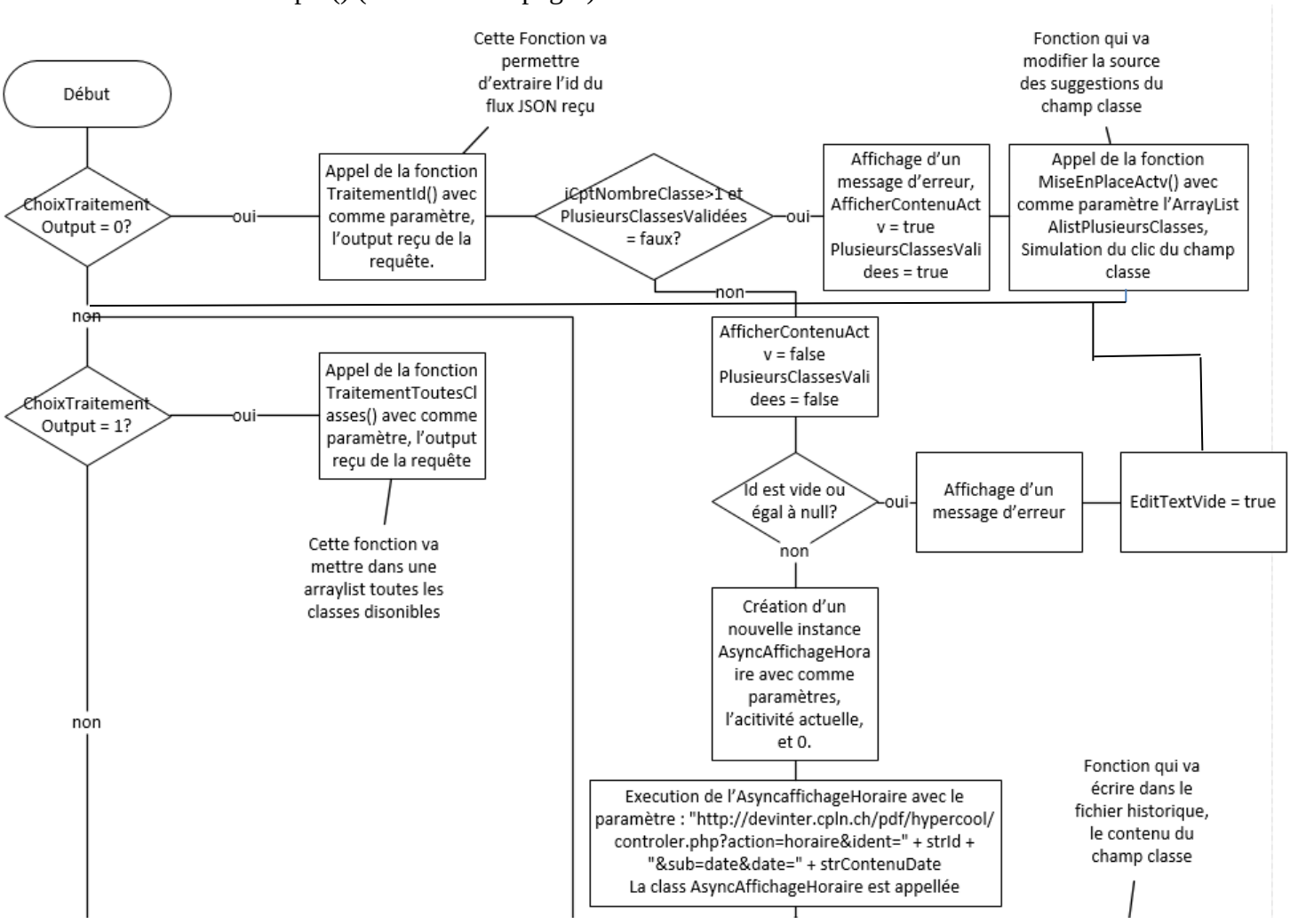
Avec l'utilisation du .get(), onPostExecute() n'était jamais utilisé.

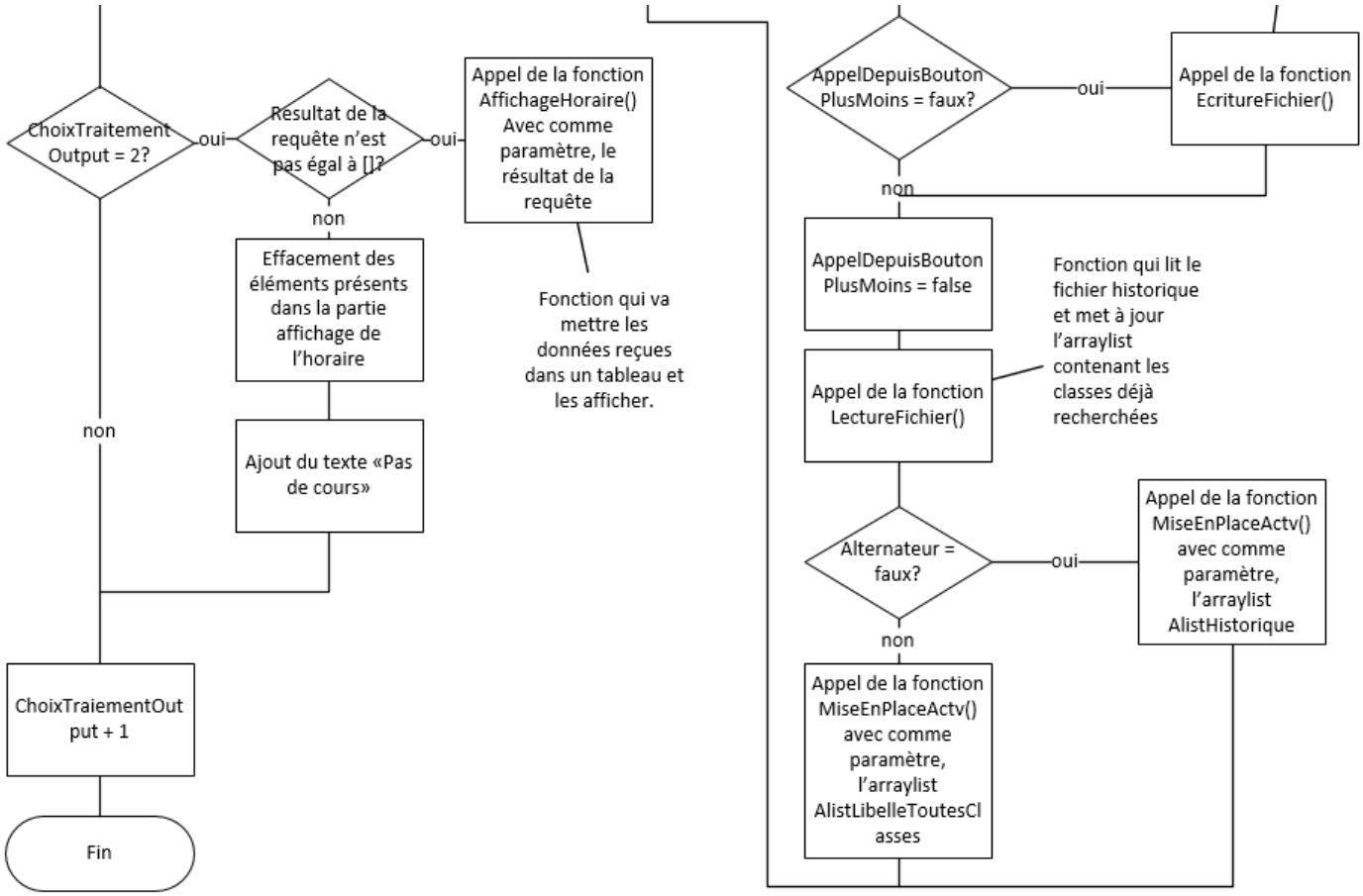
Nouvelle façon de lancer la requête.

```
        AsyncAffichageHoraire asyncAffichageHoraire0 = new
AsyncAffichageHoraire(MainActivity.this, 1);
        asyncAffichageHoraire0.delegate = (AsyncReponse) this;
asyncAffichageHoraire0.execute("http://devinter.cpln.ch/pdf/hypercool/controler.php
?action=ressource&nom=" + strContenuClasse);
```

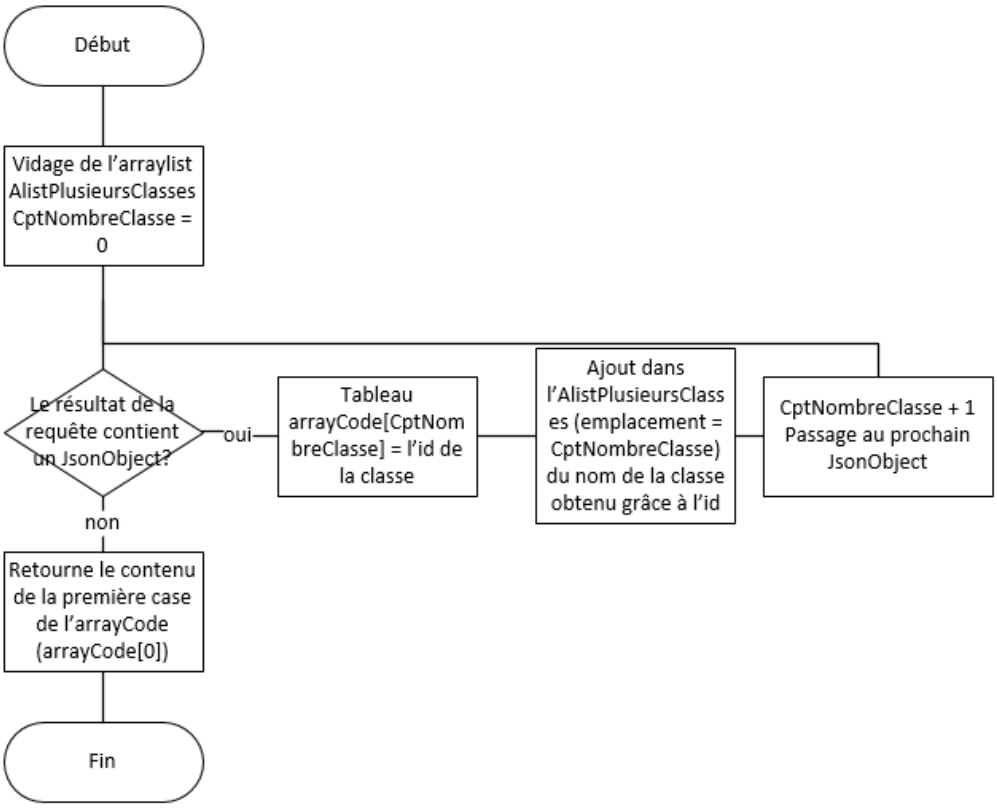
Et bien sûr, la création de la fonction RetourOutput().

Fonction RetourOutput() (schéma sur 2 pages) :

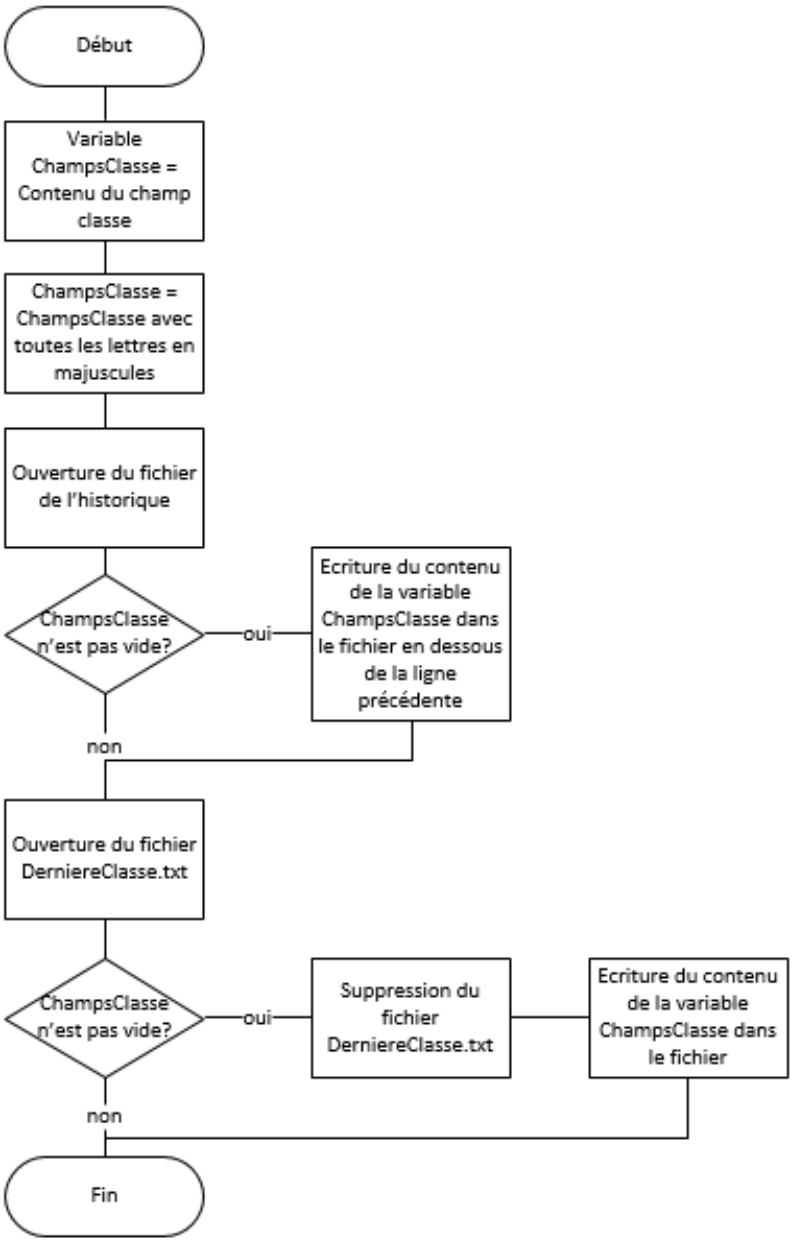




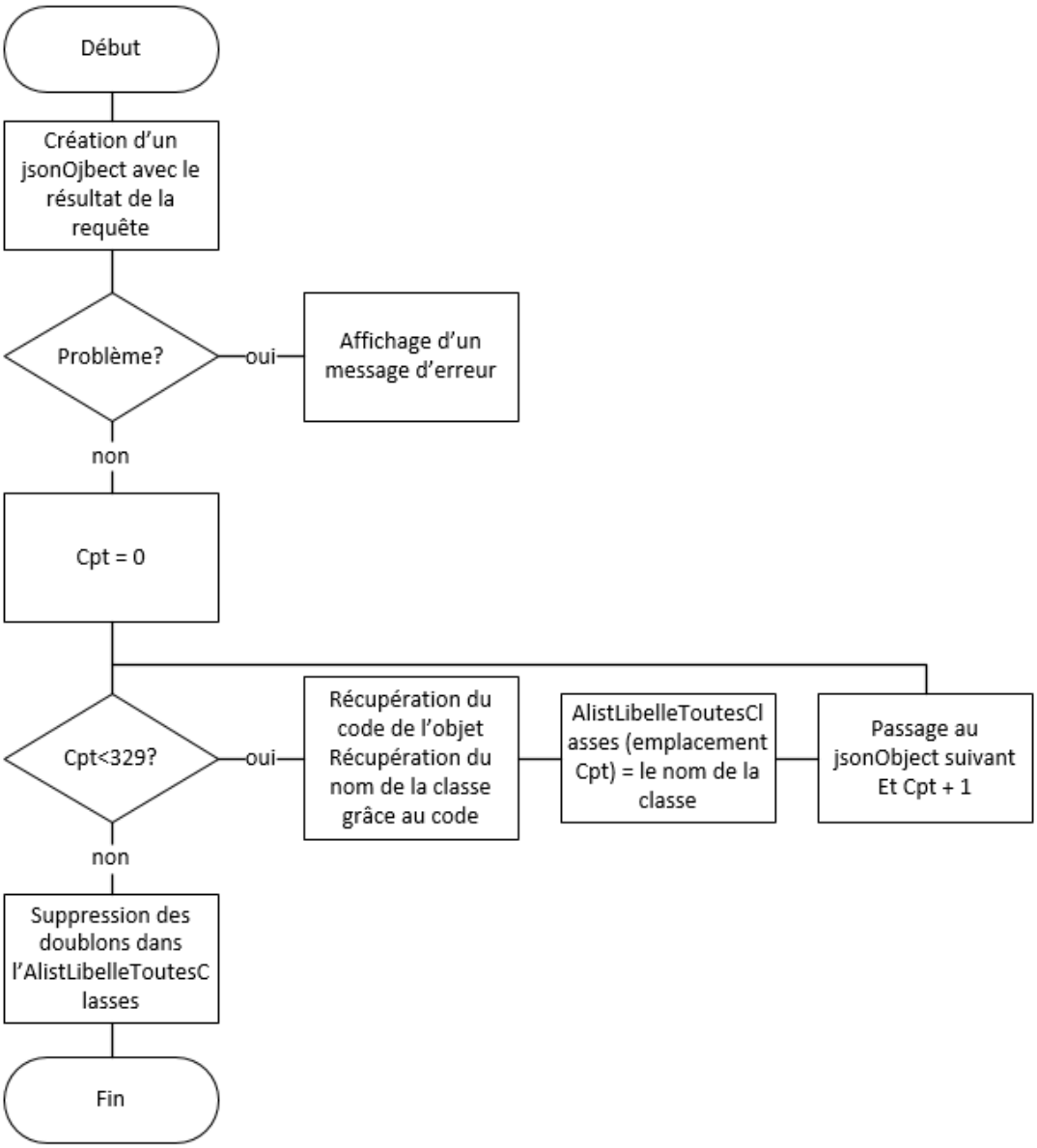
Fonction TraitementId() :



Fonction EcritureFichier() :



Fonction TraitementToutesClasses() :



Cette fonction (RetourOutput) reçoit les résultats des requêtes envoyées. Ces données reçues doivent être traitées de différentes façons. La variable iChoixTraitementOutput va aiguiller les résultats. Le principe est simple : lorsque les données de la première requête sont reçues, elles vont passer par la condition iChoixTraitementOutput =0 ? . Tout à la fin de la fonction, cette variable s'incrémente de 1. Comme ça, lorsque les données de la seconde requête arrivent, elles vont désormais passer par la condition iChoixTraitementOutput =1 ?. Même principe pour la troisième requête.

Comme expliqué auparavant, il faut l'id de la classe pour pouvoir demander l'horaire au serveur. La fonction TraitementId() extrait uniquement l'id du flux JSON reçu. Il y a cependant des exceptions. Lorsque l'on demande l'id de la classe 3m3i2, il n'y a pas de problème. Par contre, certaines classes renvoient plusieurs id.

Exemple avec la classe 3m3i1 :
Requête : http://devinter.cpln.ch/pdf/hypercool/controler.php?action=ressource&nom=3m3i1
Résultat reçu :


```
{
  "958": {
    "nom": "3M3I1",
    "prenom": "",
    "code": "ET"
  },
  "4594": {
    "nom": "3M3I1_Chopard",
    "prenom": "",
    "code": "ET"
  }
}
```


On voit que le serveur renvoie les informations de plusieurs classes. Il faut alors détecter ce cas, et demander à l'utilisateur de choisir la bonne classe parmi les classes reçues. TraitementId() va repérer le nombre de classes reçues avec la variable iCptNombreClasse. Un arraylist (AlistPlusieursClasses) va être rempli avec le nom des classes retournées. La fonction retourne alors l'id de la première classe.

Une fois l'id de la classe mis dans une variable, je teste si iCptNombreClasse est plus grand que 1 pour savoir si plusieurs classes ont été retournées. Dans la condition, il y a également un deuxième test. « && !bPlusieursClasseValidees ». Cette condition va permettre de savoir s'il faut avertir l'utilisateur que plusieurs classes ont été reçues et agir en conséquence. Car si effectivement, cette variable est égale à false, la dernière requête (celle pour avoir les informations de l'horaire) n'est pas lancée. A la place, le champ classe va suggérer les noms des classes reçues à l'aide de la fonction MiseEnPlaceActv().

La fonction permet, comme son nom l'indique, de modifier la source des suggestions du champ classe. Le clavier est alors désactivé et la liste des classes s'affiche pour que l'utilisateur choisisse la bonne. La variable bPlusieursClasseValidees est égale à true lors du démarrage de l'application. De cette manière, on ne va jamais passer par cette condition au début. Elle passe à false dans la condition suivante, et va permettre, lors des prochaines recherches, d'afficher la liste des classes retournées (si plusieurs retournées).
Comme je l'ai dit avant, dans certain cas, il ne faut pas proposer la liste des classes reçues à l'utilisateur. Par exemple, imaginons que l'utilisateur lance une recherche avec la classe 3m3i1 (après le démarrage de l'application). Il devra alors choisir la bonne entre les 2 reçues.
S'il appuie sur 3m3i1, puis sur le bouton rechercher, l'horaire de la classe 3m3i1 va s'afficher. La dernière classe recherchée est alors 3m3i1. S'il quitte l'application puis le redémarre, 3m3i1 va être automatiquement inséré dans le champ classe. Vu qu'au démarrage, l'utilisateur veut directement voir l'affichage de l'horaire de la classe 3m3i1. Il ne faut pas lui proposer la liste de toutes les classes reçues.

Dans notre cas (au démarrage), on passe par le chemin du « non » pour cette condition. bPlusieursClasseValidees et dorénavant égale à false pour afficher la liste lors des prochaines recherches.

Un test est effectué pour savoir si l'id est vide. Si c'est le cas, c'est que la classe n'existe pas et j'avertis l'utilisateur. bEditTexteVide passe à true. Si par contre, l'id est correcte, je peux enfin lancer la dernière requête. Cette requête contient l'id de la classe et la date voulue. Elle va également actionner l'icône de chargement grâce à son paramètre qui est à 1. La fonction

EcritureFichier() est appelée si la variable bAppelDepuisBoutonPlusMoins est égale à false. Cette variable sert à savoir si l'on doit écrire dans le fichier historique. Lors du démarrage, cette variable est false, alors on fait appel à la fonction EcritureFichier(). La fonction permet d'écrire le contenu du champ classe dans le fichier contenant l'historique et également dans le fichier contenant la dernière classe recherchée. Lors de l'écriture, les lettres sont écrites en majuscules. Cette opération est faite pour ne pas avoir de doublons dans l'historique (3m3i2 et 3M3I2). J'ai choisi cette méthode pour enregistrer les classes recherchées car elle était simple. Une base de données serait peut-être trop grosse pour l'utilisation que j'en fait.

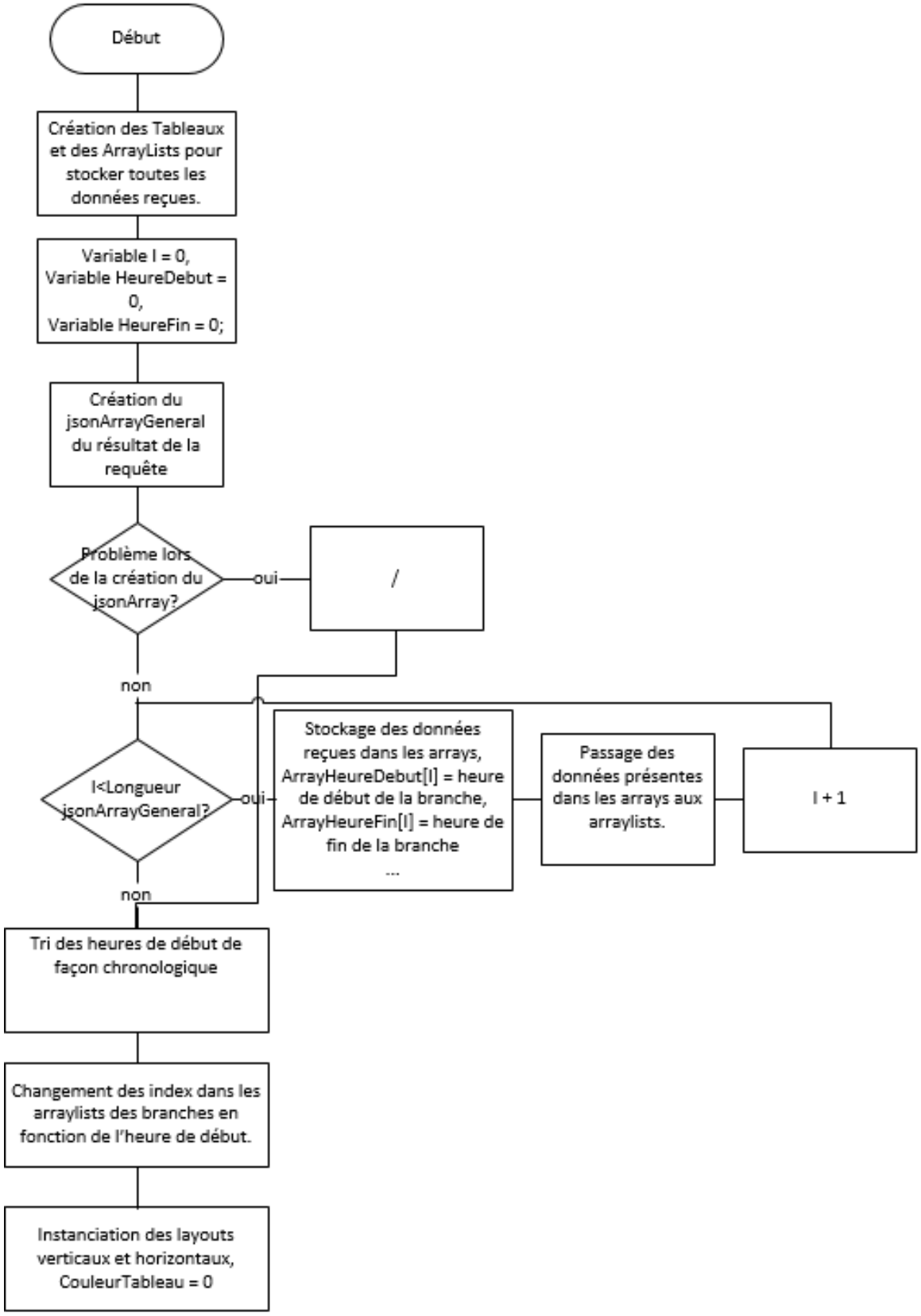
J'appelle ensuite la fonction LectureFichier(). Elle va lire le fichier historique et mettre tous les noms des classes dans l'arraylist AlistHistorique. Les doublons vont également être supprimés dans cette fonction.

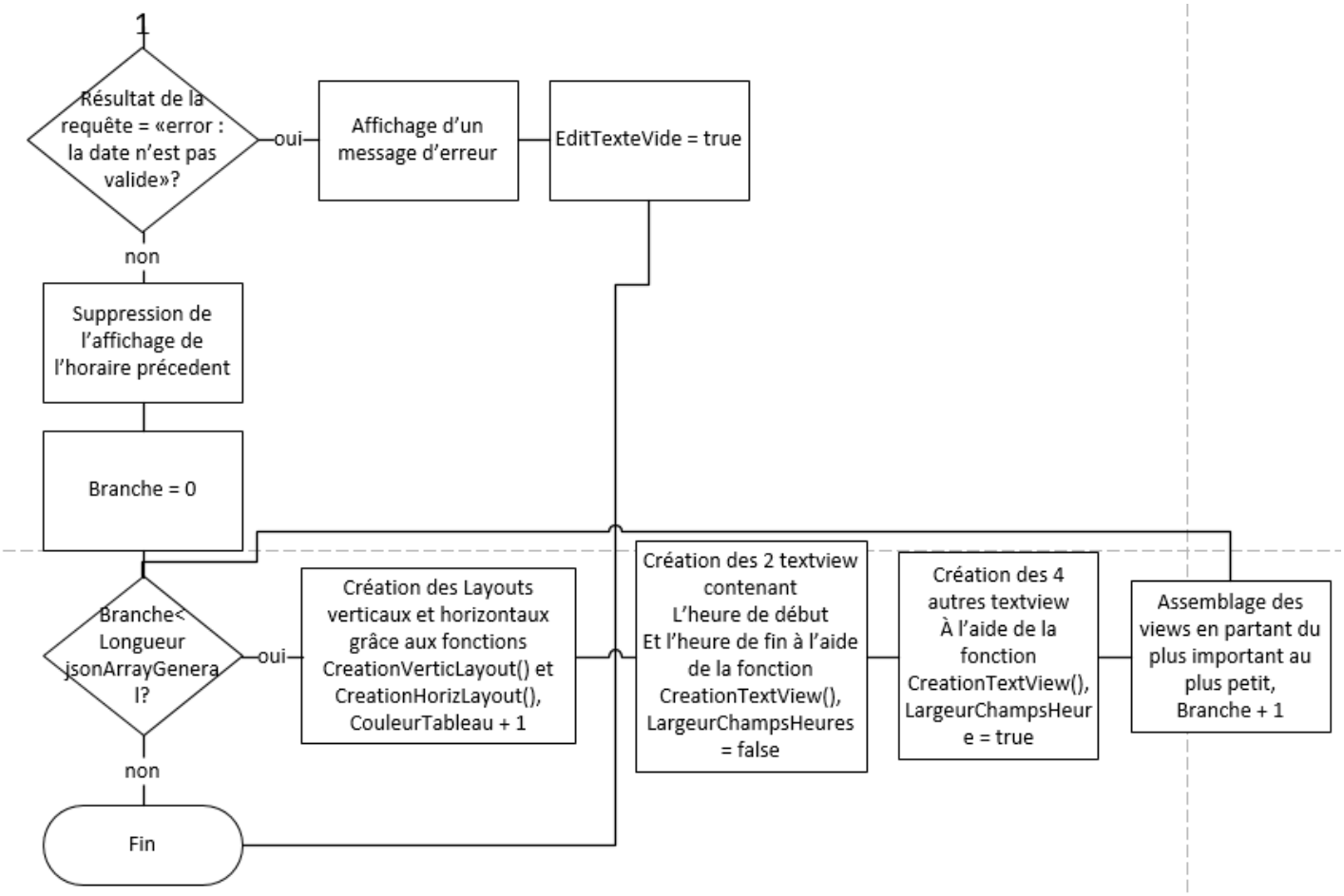
Un test s'effectue pour savoir avec quelle source il faut alimenter les suggestions du champ classe. Si bAlternateur est faux, alors les suggestions vont être l'historique des recherches, autrement, ça sera la liste de toutes les classes disponibles qui sera suggérée. Cette variable sert à savoir dans quel mode de suggestion on est (elle sera expliquée plus en détail dans le cas d'utilisation « Afficher l'historique des recherches »).

Toute cette partie que j'ai expliqué est effectuée lorsque iChoixTraitementOuput = 0, donc quand la requête renvoie l'id de la classe qui est reçue. Au même moment que cette requête-là, une deuxième requête, renvoyant les id de toutes les ressources était lancée. Les données reçues de cette requête passent par iChoixTraitementOutput = 1. Dans cette condition, la fonction TraitementToutesClasses() est appelée. Elle va remplir l'arraylist AlistLibelleToutesClasses avec le nom des 329 premiers jsonObject, ce qui représente le nom de toutes les classes. Cette requête reçoit toutes les ressources du site Hypercool (salles, classes, professeurs etc.). Pour extraire les données des classes uniquement, j'ai essayé divers tris. Par exemple, les classes n'ont pas de prénom, et j'ai essayé de trier les ressources en fonction de ça. Les noms des salles n'avaient également pas de prénom donc mon tri ne fonctionnait pas. Etant donné que les id des classes sont renvoyées avant les autres ressources, j'ai décidé de faire une boucle qui va prendre le nom des 329 (nombre de classes) premières id reçues. Ce n'est pas une très bonne méthode et il faudrait trouver un moyen d'améliorer cela, en proposant peut être un lien Hypercool qui renverrait uniquement les id des classes (un mail a été envoyée à M. Ferrari pour savoir si c'était possible, il m'a répondu que Hypercool était figé et qu'il ne pouvait pas le modifier dans l'immédiat).

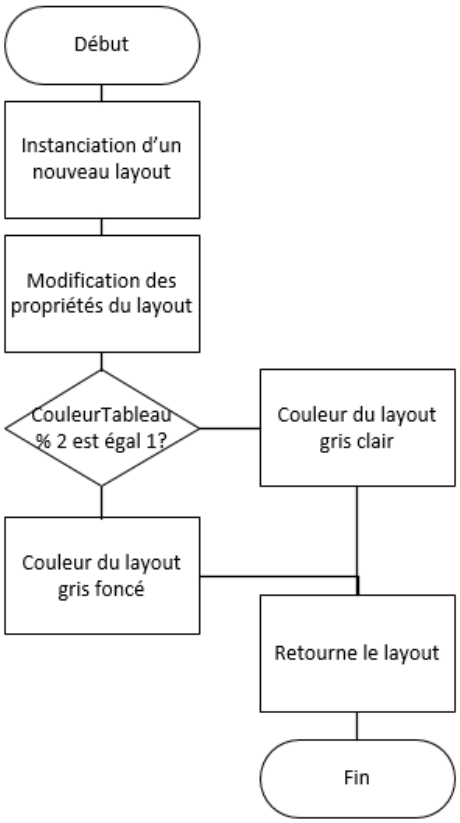
Une troisième requête pour avoir l'horaire de la classe a été lancée auparavant. Les données reçues passent alors dans iChoixTraitementOutput = 2 ? . Un test est effectué pour voir si l'on a bien reçu des données. Si le résultat est égal à « [] », alors ça signifie que la classe n'a pas cours et j'avertis l'utilisateur. Autrement, la dernière fonction AffichageHoraire() est appelée.

Fonction AffichageHoraire() :



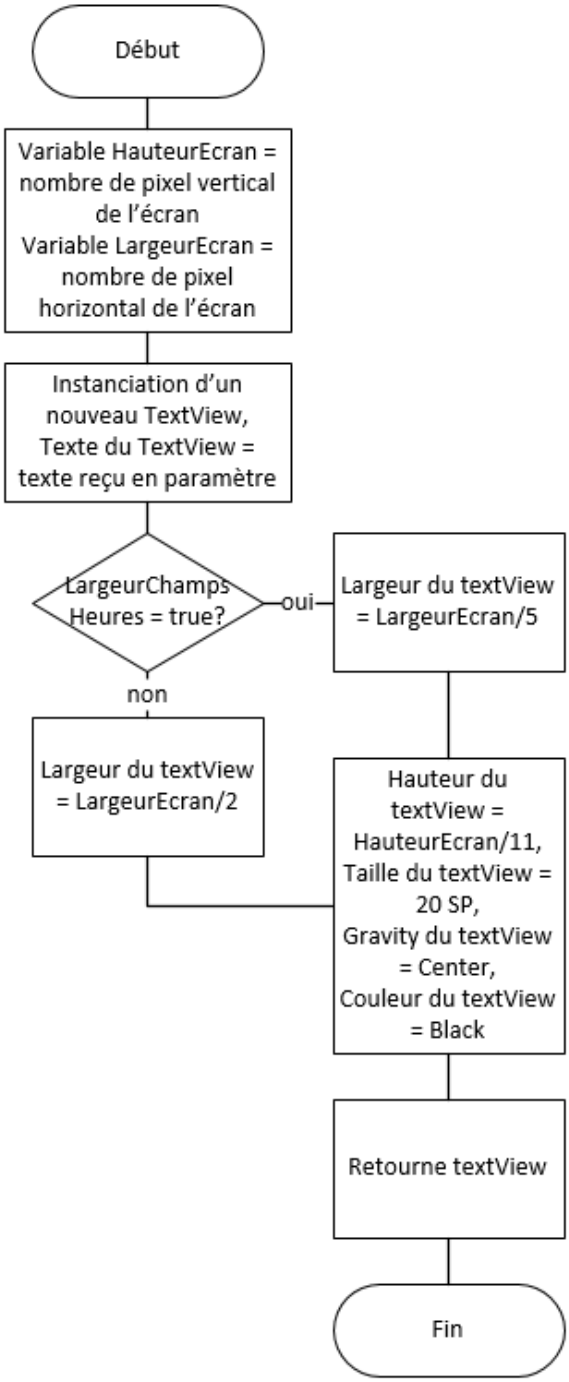


Fonction CreationVerticLayout() :



Fonction CreationHorizLayout() :

Fonction CreationTextView() :



Dans cette fonction, on va mettre toutes les données reçues dans des variables. Ces données seront triées et finalement affichées à l'utilisateur.

Je crée d'abord des arrays (tableaux) et des arraylists pour stocker les données.

Une partie des tableaux créés

```
        String[] ArrayLibelle = new String[100];
        String[] ArrayProfesseur = new String[100];
        String[] ArraySalle = new String[100];
        ArrayList<String> AlistLibelle = new ArrayList<String>();
```

```
        ArrayList<String> AlistProfesseur = new ArrayList<String>();
        ArrayList<String> AlistArraySalle = new ArrayList<String>();
```

Pour mieux comprendre mon développement, voici un exemple concret :
Requête :
http://devinter.cpln.ch/pdf/hypercool/controler.php?action=horaire&ident=4354&sub=date&date=05-05-2017

Résultat :


```
[
  {
    "codeMatiere": "FOP",
    "libelle": "AT-Informatique",
    "indice": 1.43,
    "heureDebut": "13h10",
    "heureFin": "16h25",
    "classes": [
      "3INFC",
      "3M3I2"
    ],
    "professeur": [
      "Sacchetti Plinio [PSI]"
    ],
    "salle": [
      "B106B-74487"
    ]
  },
  {
    "codeMatiere": "CHI",
    "libelle": "Chimie",
    "indice": 2.0833333333333,
    "heureDebut": "08h55",
    "heureFin": "10h45",
    "classes": [
      "3M3M",
      "3M3A",
      "3M3E",
      "3M3I2"
    ],
    "professeur": [
      "Lorimier Yvan [YL]"
    ],
    "salle": [
      "B319-74456"
    ]
  }
]
```


Une boucle de type for va faire le nombre de tour qu'il y a de branches reçues. Ici, 2. Pour connaitre ce chiffre, il suffit de faire jsonArray.length().
A l'intérieur de cette boucle, je vais mettre les données de la première branche dans les tableaux.

```
  JSONObject jsonObjectGeneral = jsonArrayGeneral.getJSONObject(i);
                ArrayHeureDebut[i] = jsonObjectGeneral.getString("heureDebut");
                ArrayHeureFin[i] = jsonObjectGeneral.getString("heureFin");
                ArrayLibelle[i] = jsonObjectGeneral.getString("libelle");
```

Etant donné que pour certaines branches (ex : Maitrise de classe), la salle ou le professeur ne sont parfois pas présents, des try/catch sont utilisés. Pour certaines branches, plusieurs salles sont renvoyées.

```
{
  "codeMatiere": "ALL",
  "libelle": "Allemand",
  "indice": 2.0833333333333,
  "heureDebut": "09h40",
  "heureFin": "11h30",
  "classes": [
    "3M3M",
    "3M3A",
    "3M3E",
    "3M3I2"
  ],
  "professeur": [
    "Ramseyer Manna G\u00e9raldine [GR]"
  ],
  "salle": [
    "B410-74466",
    "B408-74464 labolangues",
    "AB3-74419",
    "C6"
  ]
},
```


J'ai d'abord cru qu'il y avait une certaine logique, et que les salles actuelles étaient en premiers/derniers. Malheureusement, cette méthode fonctionnait pour certaines classes mais pas d'autres. J'ai alors demandé au professeur quelle salle je devais prendre et il m'a laissé 2 choix : soit je prends la première salle, ou bien je les prends toutes et je les colle à la suite. (Ex : B410-74466, B408- …). J'ai alors décidé de prendre uniquement la première salle, car sinon j'aurais une chaine de caractère trop longue et qui entraverait l'affichage.

Pour proposer un affichage plus agréable à l'utilisateur, les salles comportant des tirets sont découpées pour que seule la première partie soit gardée (ex : B410-74466 -> B410).

Les prénoms des professeurs sont également supprimés.

Une fois toutes les données récupérées, je les fais passer des arrays aux arraylists.

```
                AlistHeureDebutComplet.add(i, ArrayHeureDebut[i]);
                AlistHeureFinComplet.add(i, ArrayHeureDebut[i]);
                AlistProfesseur.add(i, ArrayProfesseur[i]);
                AlistArraySalle.add(i, ArraySalle[i]);
                AlistLibelle.add(i, ArrayLibelle[i]);
```

Je passe par cette étape car j'avais de gros problèmes pour trier et changer les index des arrays (étape suivante). En effectuant la même opération avec des arraylists, ça fonctionnait. Je ne sais toujours pas si je faisais une erreur humaine, ou que le programme m'interdisait de faire cette opération avec les arrays.

Comme on peut le voir avec mon premier exemple, les branches reçues ne sont pas forcément dans l'ordre. La chimie est en seconde place alors qu'elle commence avant l'autre branche. Pour pouvoir les placer dans l'ordre, j'ai décidé d'employer une méthode simple. J'additionne l'heure de début avec l'heure de fin.

Ex :
At- Informatique -> 13h10 à 16h25 -> 13 + 16 = 29
Chimie -> 08h55 à 10h45 -> 08 + 10 = 18

Dans le code :

```
ArrayCalculHeure[i] = iHeureDebut + iHeureFin;
AlistCalculHeure.add(i, ArrayCalculHeure[i]);
AListCalcul.add(i, iHeureDebut + iHeureFin);
```

Cette façon de faire a une limite. Si par exemple une branche commence à 08h10 et finit à 08h30, et une autre dure de 08h35 à 08h55, il n'est alors pas possible de savoir laquelle commence d'abord. Le calcul est égal à 16 dans les deux cas. Cette situation n'arrive jamais avec les branches du Cpln, et donc, je peux utiliser cette manière de faire.

Une fois toutes ces étapes effectuées, la boucle va refaire des tours, tant qu'il y a de branches.

Voici alors, l'état des arraylists :

AlistLibelle.get(0) = At-Informatique
AlistProf.get(0) = Sacchetti
AlistCalcul.get(0)= 29
AlistLibelle.get(1) = Chimie
AlistProf.get(1) = Lorimier
AlistCalcul.get(1)= 18

Maintenant, je dois trier du plus petit au plus grand l'alistCalcul.

```
Collections.sort(AListCalcul);
```

Après opération :
AlistCalcul.get(0) = 18
AlistCalcul.get(1) = 29

Il reste encore à basculer les index des données des autres arraylists, afin que les données de la branche qui commence soient en position 0 et ainsi de suite.

Après le basculement :

AlistLibelle.get(0) = Chimie
AlistProf.get(0) = Lorimier
AlistCalcul.get(0)= 18
AlistLibelle.get(1) = At-Informatique
AlistProf.get(1) = Sacchetti
AlistCalcul.get(1)= 29

Voici l'extrait de code permettant de faire cela. AlistCalculHeure contient l'addition de l'heure de début et l'heure de fin, non trié, Alors qu'AListCalcul est trié.

```
  for (int iBranche = 0; iBranche < jsonArrayGeneral.length(); iBranche++) {
                for (int i = 0; i < jsonArrayGeneral.length(); i++) {
                    if (AlistCalculHeure.get(i).equals(AListCalcul.get(iBranche)))
{
                        AlistLibelle.add(iBranche, ArrayLibelle[i]);
                        AlistHeureDebutComplet.add(iBranche,
ArrayHeureDebutComplet[i]);
                        AlistHeureFinComplet.add(iBranche,
ArrayHeureFinComplet[i]);
```

```
                        AlistProfesseur.add(iBranche, ArrayProfesseur[i]);
                        AlistArraySalle.add(iBranche, ArraySalle[i]);
                    }
                }
```

Après avoir trié et mis les données reçues dans des arraylists, je peux finalement passer à l'affichage.

Etant donné que j'allais afficher les données sous forme de liste, j'aurais pu utiliser une listView personnalisée. Je n'avais jamais utilisé ce type de view et je ne voulais pas perdre trop de temps avec la prise en main. J'ai alors choisi une autre méthode. J'ai décidé de créer dynamiquement la grille qui va contenir les données. Je me sers du layout « LlGeneral » présent dans le xml comme base. Les layouts crées dynamiquement vont se placer dessus. J'instancie tout d'abord 1 layout horizontal et 3 layout verticaux.

```
        LinearLayout llGeneral = (LinearLayout) findViewById(R.id.LlGeneral);
        LinearLayout llHoriz = new LinearLayout(getBaseContext());
        LinearLayout llVert1 = new LinearLayout(getBaseContext());
        LinearLayout llVert2 = new LinearLayout(getBaseContext());
        LinearLayout llVert3 = new LinearLayout(getBaseContext());
```

A ce point, un dernier test est effectué. Si le résultat de la requête est égal à « error : la date n'est pas valide » alors je n'affiche rien et je signale à l'utilisateur que la date n'est pas valide. bEditTexteVide passe à true. Dans le cas contraire, je peux passer à la mise en place des données. La première chose que je fais est que je supprime tout ce qui est présent dans le layout LlGeneral. Ceci permet de supprimer l'affichage des données présentes précédemment.

```
llGeneral.removeAllViews();
```

Une boucle de type for va de nouveau faire le nombre de tour qu'il y a de branches reçues.
A l'intérieur, je crée les layouts verticaux, horizontaux et les textviews grâce aux fonctions CreationLayoutHoriz(), CreationLayoutVertic() et CreationTextView().

```
                llHoriz = CreationLayoutHoriz();
                llVert1 = CreationLayoutVertic();
                llVert2 = CreationLayoutVertic();
                llVert3 = CreationLayoutVertic();
                iCouleurTableau++;
                TextView tv1 =
CreationTextView(AlistHeureDebutComplet.get(iBranche));
                TextView tv2 =
CreationTextView(AlistHeureFinComplet.get(iBranche));
                bLargeurChampsHeures = false;
                TextView tv3 = CreationTextView(AlistLibelle.get(iBranche));
                TextView tv4 = CreationTextView(AlistProfesseur.get(iBranche));
                TextView tv5 = CreationTextView(AlistArraySalle.get(iBranche));
                TextView tv6 = CreationTextView(" ");
                bLargeurChampsHeures = true;
```

La variable iCouleurTableau permet de modifier la couleur de la ligne du tableau. bLargeurChampsHeures permet d'attribuer une largeur de textview plus petite à certaines données (la colonne de gauche est plus petite que les autres).

Il ne me reste alors plus qu'à assembler le tout.

```
llGeneral.addView(llHoriz);
llHoriz.addView(llVert1);
llHoriz.addView(llVert2);
llHoriz.addView(llVert3);
llVert1.addView(tv1);
llVert1.addView(tv2);
llVert2.addView(tv3);
llVert2.addView(tv4);
llVert3.addView(tv5);
llVert3.addView(tv6);
```

3.2 Cas d'utilisation « Affichage de l'horaire d'un jour avec classe et date choisie par l'utilisateur »

Une fois l'application ouverte, l'horaire de la dernière classe recherchée avec la date actuelle est déjà affiché. Pour ce cas d'utilisation, l'utilisateur doit pouvoir modifier la classe et la date voulue. La liste de toutes les classes disponibles doit être suggérée lorsque l'utilisateur écrit dans le champ classe. Après avoir fait cela, il appuie sur le bouton Rechercher pour faire apparaitre l'horaire désiré.

Une grosse partie de la réalisation est similaire au cas d'utilisation précédent.

3.2.1 Scénario

1. L'utilisateur tape le nom de la classe voulue dans le champ classe.
2. Le système suggère la liste de toutes les classes disponibles.
3. L'utilisateur tape le nom de la date voulue dans le champ date.
4. L'utilisateur appuie sur le bouton rechercher.
5. Le système vérifie la connexion internet.
6. Le système met dans des variables le contenu des champs, vérification des saisies.
7. Le système affiche le nom du jour de la semaine.
8. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes).
9. Le système effectue des vérifications des données reçues.
10. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
11. Le système met le nom de la classe recherchée dans l'historique.
12. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
13. Le système met les données de l'horaire reçues dans des variables.
14. Le système tri les branches de manière chronologique à leurs heures de début.
15. Le système affiche les données.

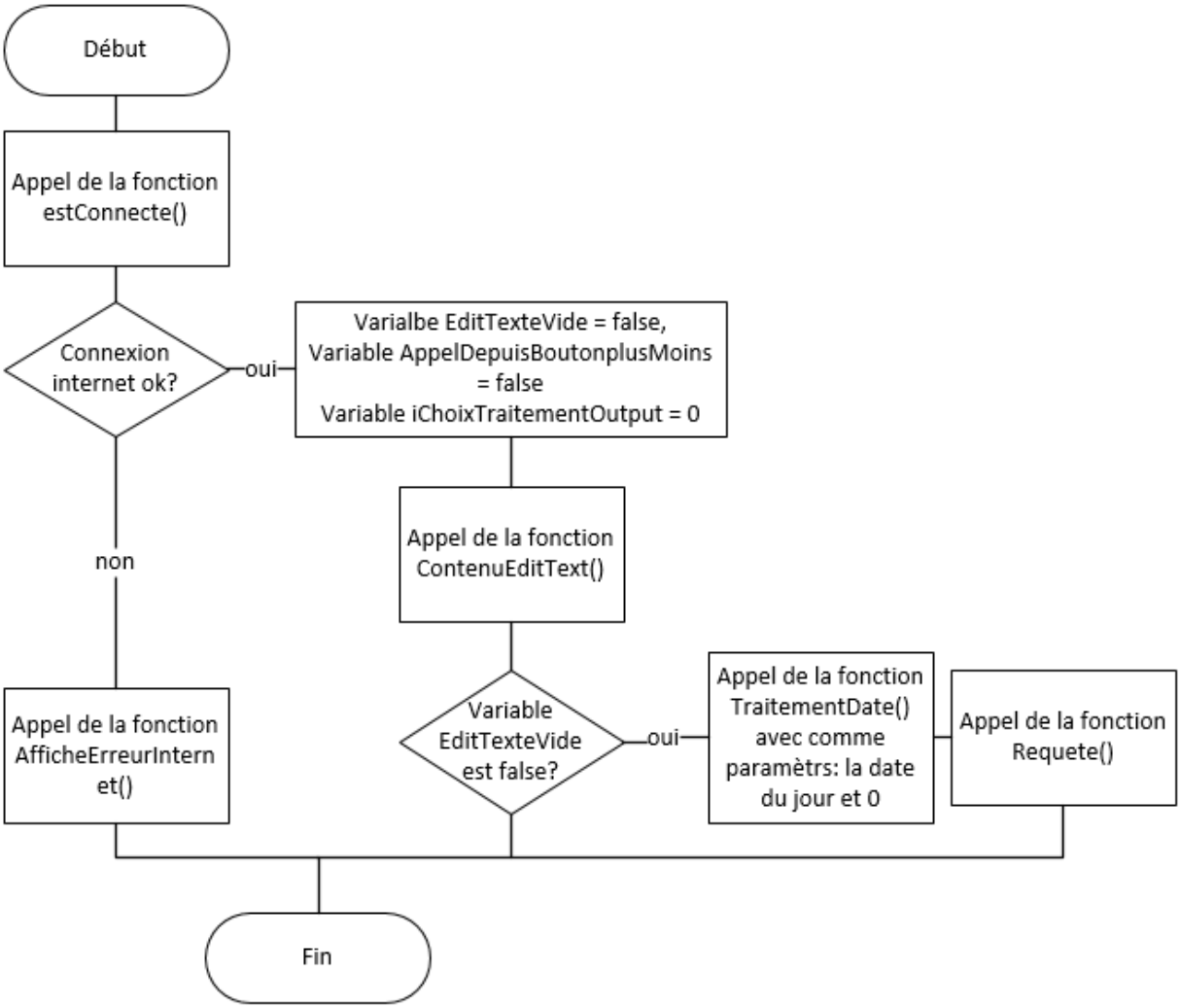
3.2.2 Maquettes

Même maquette que le cas précédent

3.2.3 Analyse du scénario

3.2.3.1 Algorithme

Lors de l'appui sur le bouton Rechercher :



3.2.3.2 Explications détaillées

Le développement est très similaire au cas précédent. Avant, la dernière classe recherchée et la date du jour étaient automatiquement placées dans les champs. Maintenant, c'est l'utilisateur qui choisit le contenu désiré. Lorsque l'utilisateur écrit quelque chose dans le champ classe, une liste de suggestions apparait en dessous. Cette liste contient le nom de toutes les classes disponibles.
Dans le ContenuEditText(), vu que la variable bPremiereOuverture est désormais false, les contenus des champs vont être placés dans les variables. Des tests pour savoir si elles sont vides sont à nouveau effectués. bAppelDepuisBoutonPlusMoins est égale à false. Elle permet d'enregistrer le nom de la classe recherchée dans le fichier historique. La variable bPlusieursClasseValidees est désormais false. Contrairement au démarrage de l'application, cette fois ci, si plusieurs classes sont retournées, l'application va alors agir et proposer la liste. iChoixTraitementOutput est remis à 0 pour pouvoir aiguiller les requêtes.

Toute la partie réception des données et de l'affichage, est le même que le cas d'utilisation précédent.

3.2.4 La phase de programmation

Ouverture de cette condition grâce au bPlusieursClasseValidees.

```
if (iCptNombreClasse > 1 && !bPlusieursClasseValidees) {
    Toast.makeText(MainActivity.this, iCptNombreClasse + " classes trouvées,
veuillez choisir la bonne classe", Toast.LENGTH_LONG).show();
    bAfficherContenuActv = true;
    MiseEnPlaceActv(AlistPlusieursClasses);
    bPlusieursClasseValidees = true;
    AutoCompleteTextView actvClasse = (AutoCompleteTextView)
findViewById(R.id.ActvClasse);
    actvClasse.performClick();

}
```

3.3 Cas d'utilisation « Affichage de l'horaire de la semaine directement au passage à la vue semaine »

Dès que l'utilisateur veut voir l'horaire de la semaine, il appuie sur le menu, puis « Vue Semaine ». L'horaire de la dernière classe recherchée doit directement s'afficher sans que l'utilisateur ait à taper quelque chose.

3.3.1 Scénario

1. L'utilisateur appuie sur le bouton « Vue Semaine » depuis l'activité vue jour.
2. Le système masque le clavier.
3. Le système passe l'application en orientation paysage.
4. Le système remplit le champ classe avec la dernière classe recherchée.
5. Le système remplit le champ date avec la date du jour actuelle.
6. Le système met dans des variables le contenu des champs, vérification des saisies.
7. Le système récupère les dates de la semaine.
8. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes).
9. Le système effectue des vérifications des données reçues.
10. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
11. Le système met le nom de la classe recherchée dans l'historique.
12. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
13. Le système met les données de l'horaire reçues dans des variables.
14. Le système tri les branches de manière chronologique à leurs heures de début.
15. Le système affiche l'horaire.

3.3.2 Maquettes

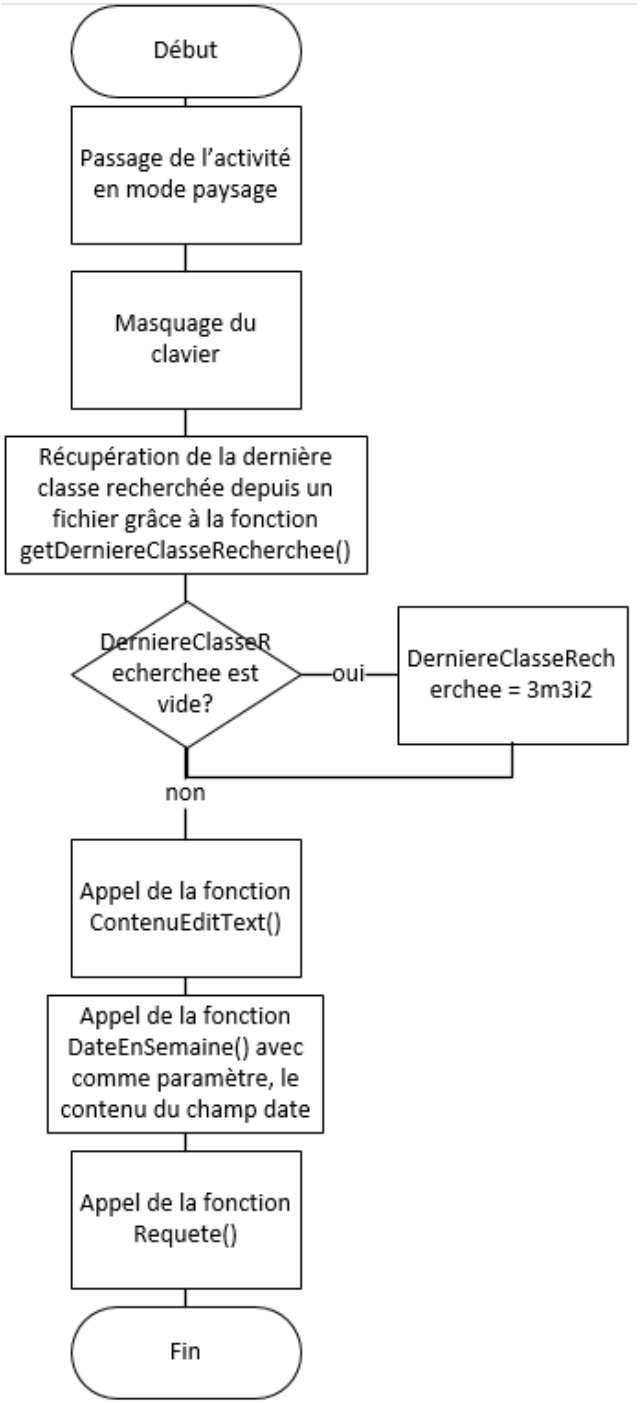
La maquette de la vue semaine est très similaire à ce que j'avais imaginé, j'ai alors décidé de mettre directement le résultat final.

3.3.3 Analyse du scénario

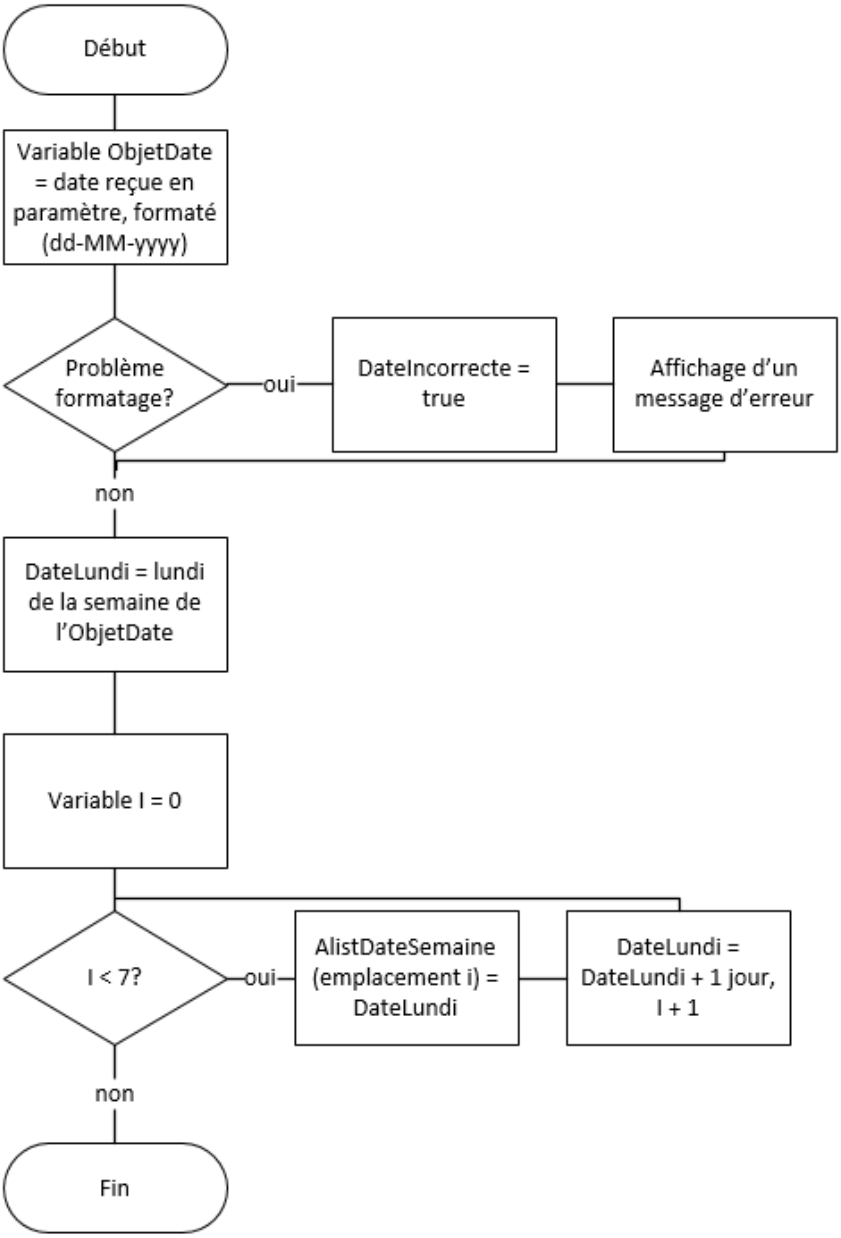
3.3.3.1 Algorithme et explications

Les algorithmes identiques à la vue jour ne sont présents ici, seuls les nouveaux ou ayant des différences sont affichés.

Lors de la création de l'activité :

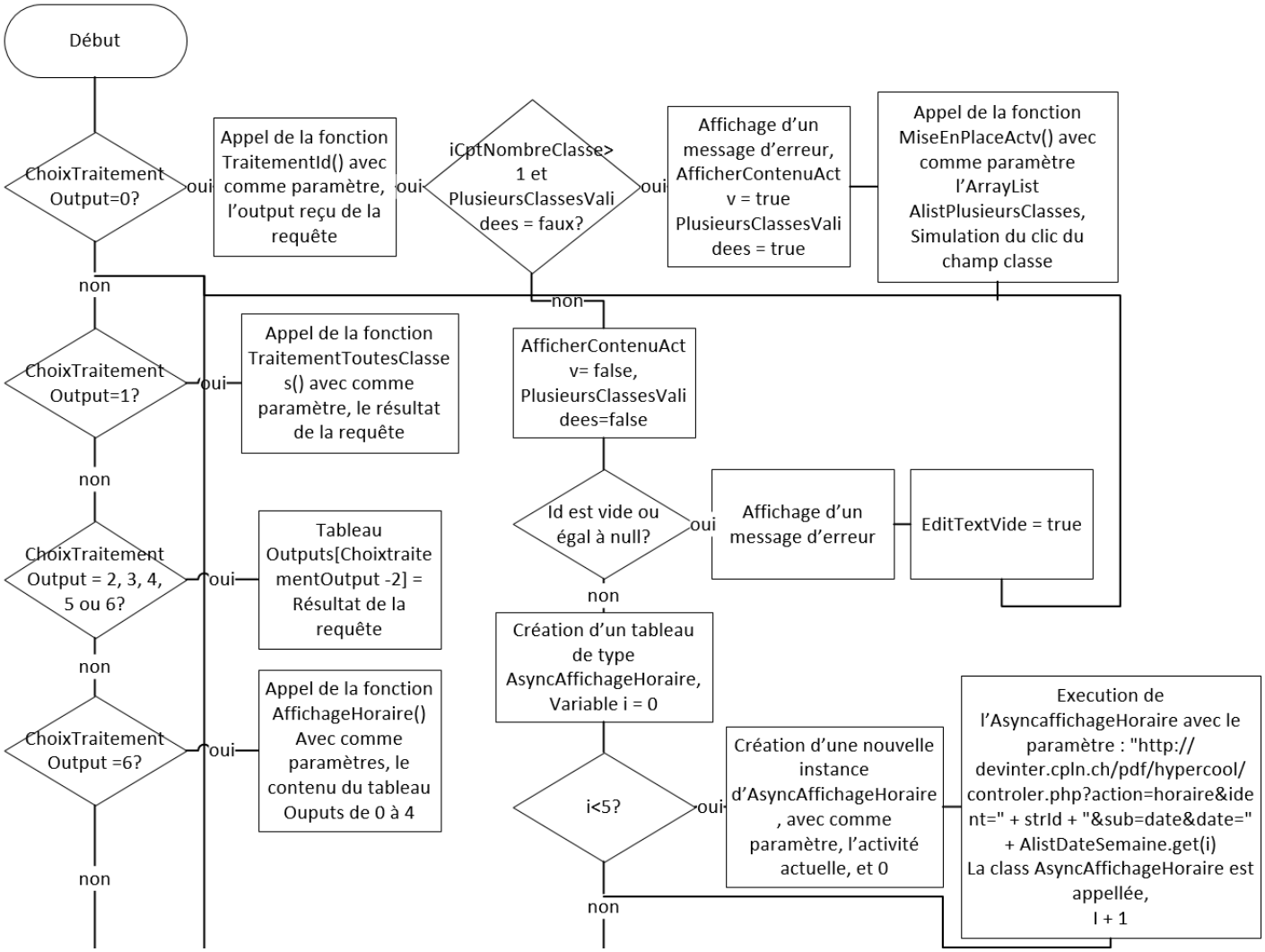

Début
Passage de l'activité en mode paysage
Masquage du clavier
Récupération de la dernière classe recherchée depuis un fichier grâce à la fonction getDerniereClasseRecherchee()
DerniereClasseRecherchee est vide?
oui
DerniereClasseRecherchee = 3m3i2
non
Appel de la fonction ContenuEditText()
Appel de la fonction DateEnSemaine() avec comme paramètre, le contenu du champ date
Appel de la fonction Requete()
Fin

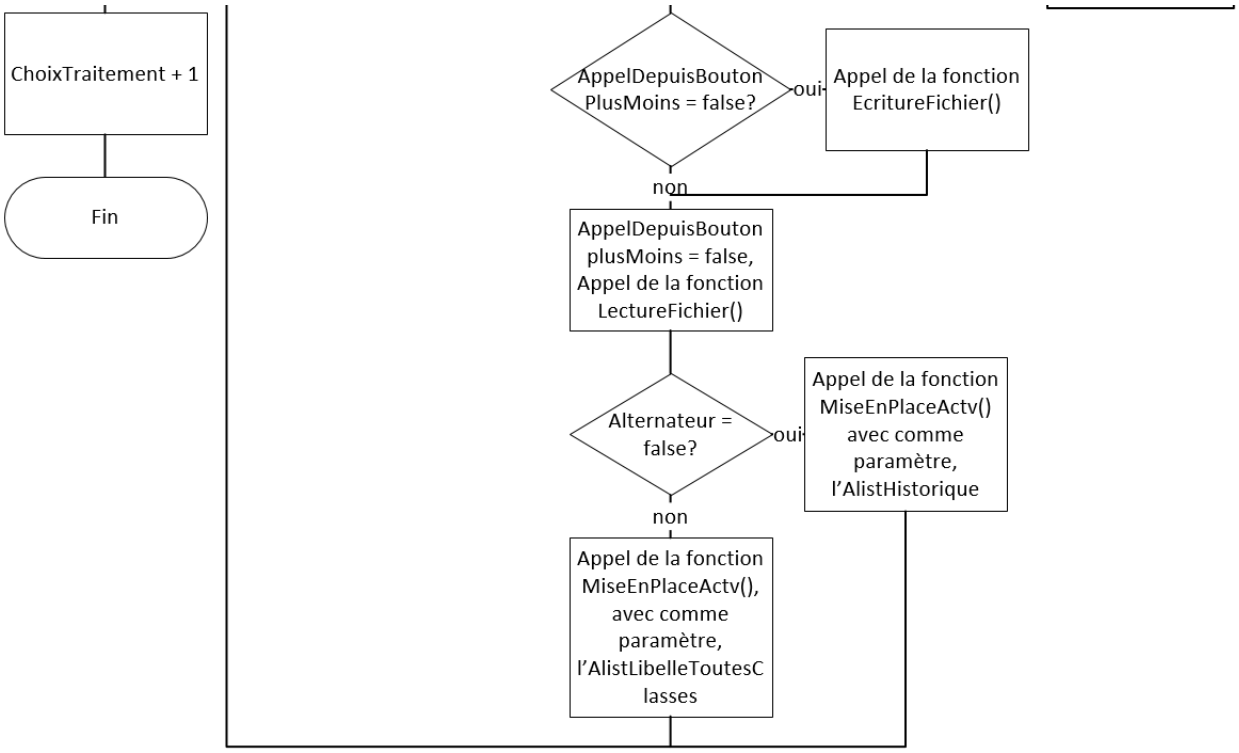
Fonction DateEnSemaine() :



Le déroulement de la vue semaine ressemble à la vue jour. Tout d'abord, l'application passe en mode paysage. Cette opération est effectuée pour gagner de la place pour pouvoir afficher les données. En mode portrait, l'espace n'est pas suffisant. Le clavier est à nouveau masqué. Je fais de nouveau appel à la fonction getDerniereClasseRecherchee(). Si la valeur reçue par le fichier est null, alors j'attribue en dur une valeur. Cette situation n'est pas censée arriver, mais ça permet de ne pas faire crash l'application si ça se produit. ContenuEditText() va mettre la dernière classe recherchée et la date du jour dans les champs. Je fais ensuite appel à la fonction DateEnSemaine(). Elle permet de mettre dans un tableau toutes les dates de la semaine en commençant depuis le lundi. Ces dates vont être utilisées lors des requêtes. Finalement, on appelle Requete(). Comme dans le cas d'utilisation précédent, elle exécute les mêmes requêtes et renvoie les résultats dans la fonction RetourOutput().

Fonction RetourOutput() (schéma sur 2 pages):





La fonction RetourOutput() ressemble beaucoup à celle de la vue jour. Une fois toutes les étapes passées, au lieu de faire une seule requête, on en fait 5. A chaque requête, la date change avec l'AlistDateSemaine.

```
AsyncAffichageHoraire[] asyncGeneral = new AsyncAffichageHoraire[8];
                    for (int i = 0; i < 5; i++) {
                        asyncGeneral[i] = new
AsyncAffichageHoraire(Activite_VueSemaine.this, 0);
                        asyncGeneral[i].delegate = (AsyncReponse) this;

asyncGeneral[i].execute("http://devinter.cpln.ch/pdf/hypercool/controler.php?action
=horaire&ident=" + strId + "&sub=date&date=" + AlistDateSemaine.get(i));
                    }
```

Les données reçues de ces 5 requêtes sont stockées dans le tableau Outputs[].

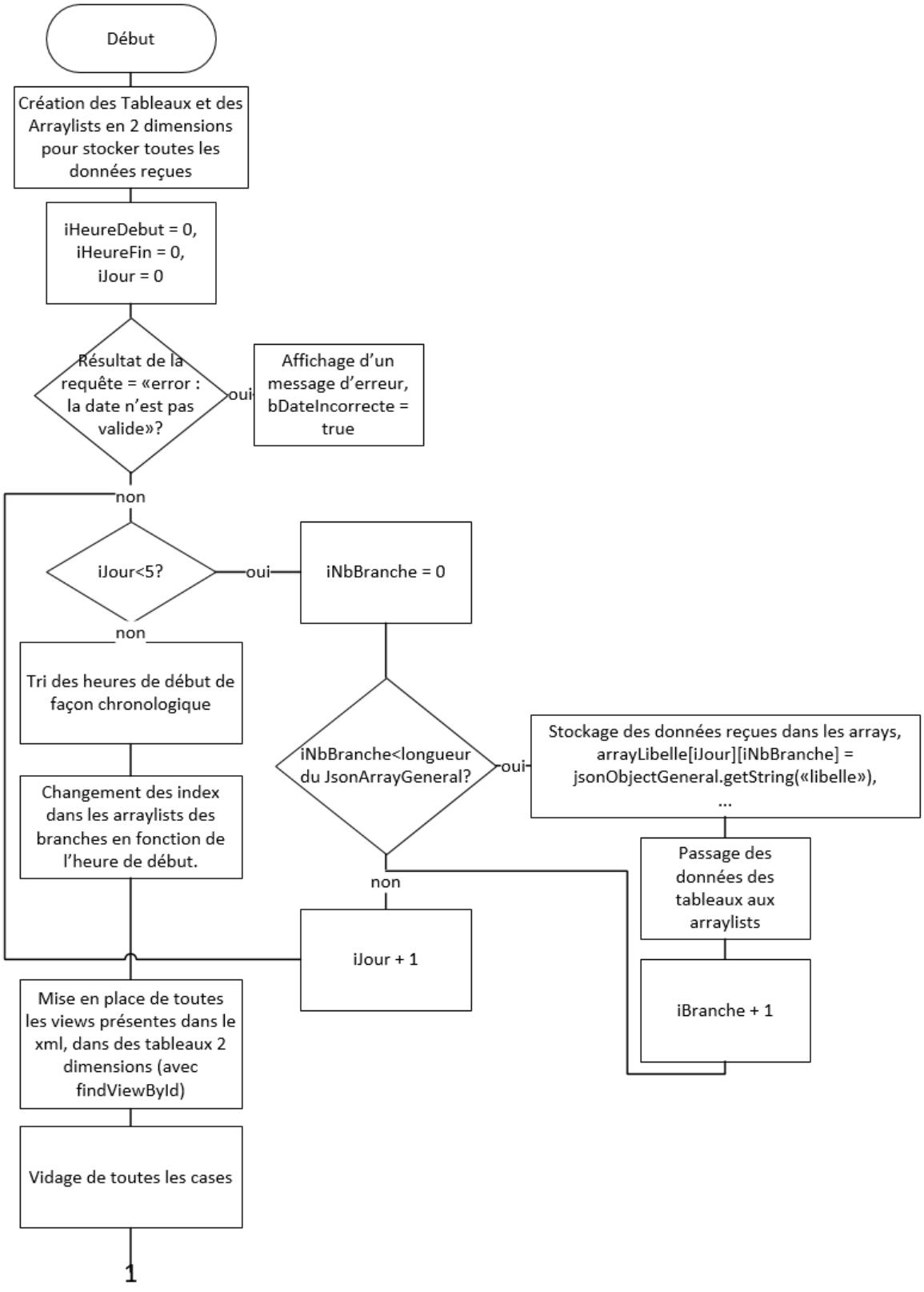
```
        if(iChoixTraitementOutput==2 ||iChoixTraitementOutput==3
||iChoixTraitementOutput==4 ||iChoixTraitementOutput==5
||iChoixTraitementOutput==6){
            Outputs[iChoixTraitementOutput-2] = strOutput;
        }
```

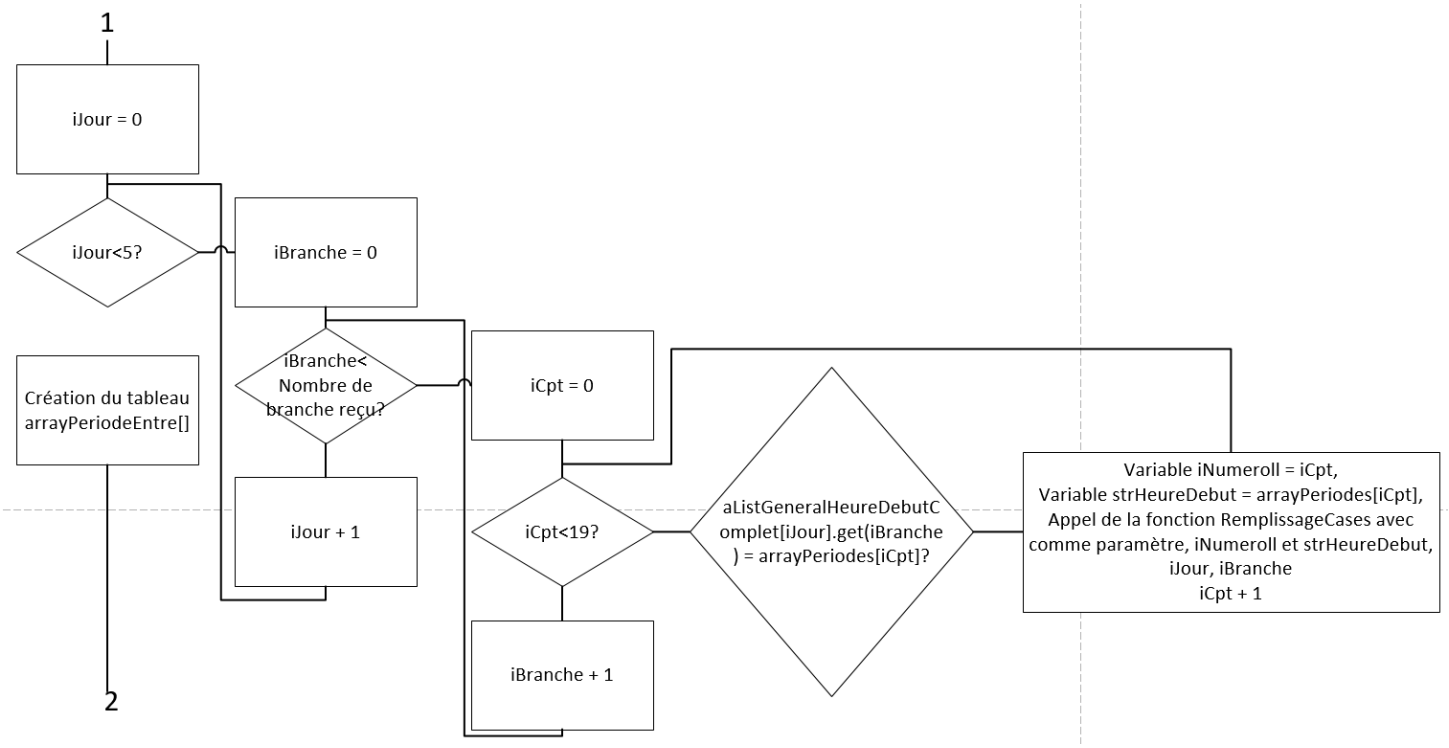
Une fois que les requêtes ont été effectuées, j'appelle la fonction AffichageHoraire() avec en paramètres, les 5 premières cases du tableau Outputs[].

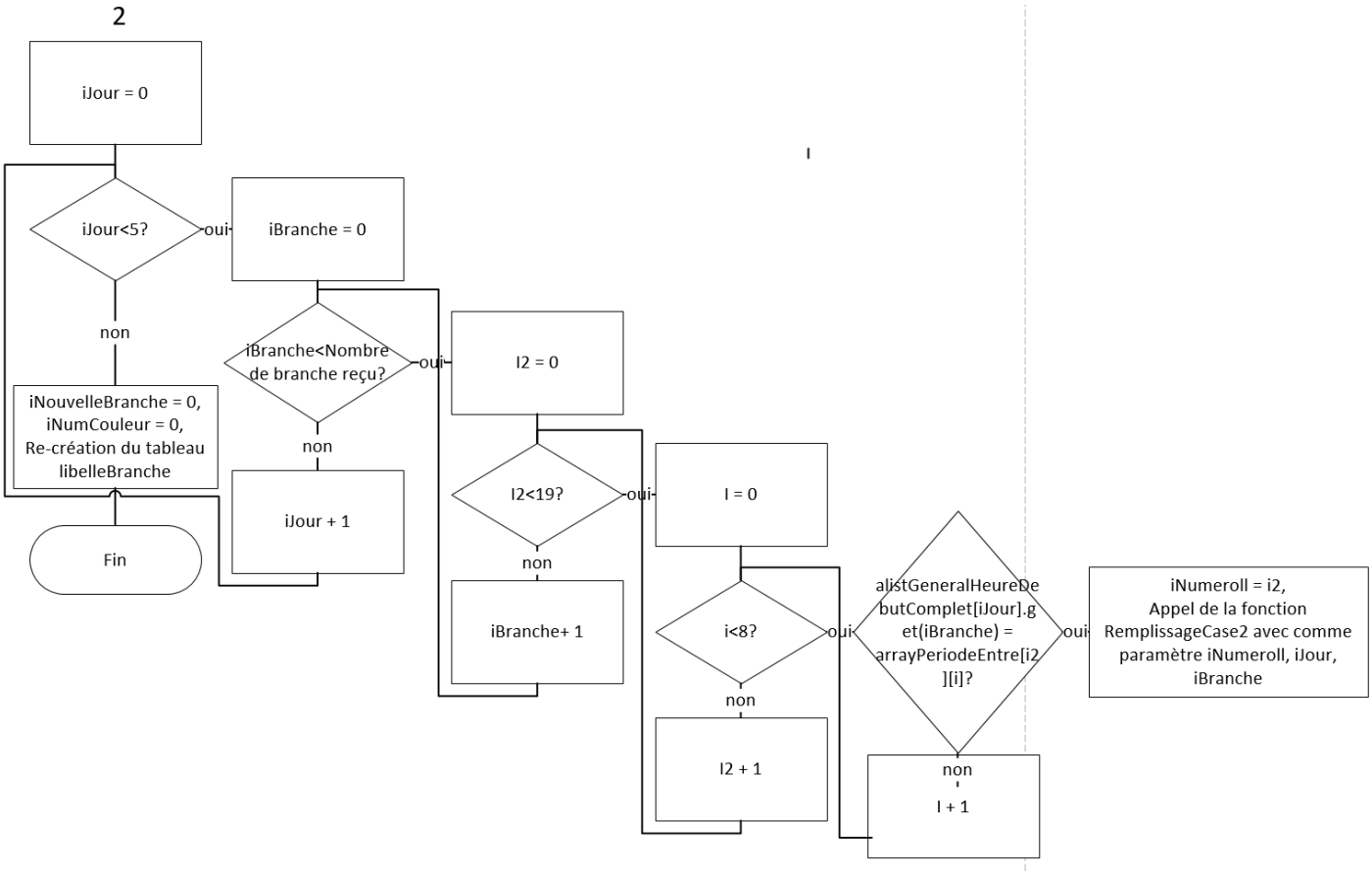
```
     if(iChoixTraitementOutput==6) {
            AffichageHoraire(Outputs[0], Outputs[1], Outputs[2], Outputs[3],
Outputs[4]);
            TextView tvJourDeLaSemaine2 =
(TextView)findViewById(R.id.TvJourDeLaSemaine2);
            if(!bDateIncorrecte) {
```

```
                tvJourDeLaSemaine2.setText("Lundi " + AlistDateSemaine.get(0) + "
au Vendredi " + AlistDateSemaine.get(6));
            }
        }
```

AffichageHoraire() :







Comme dans la vue jour, c'est dans cette fonction que les données sont placées dans des variables. Tout d'abord, je crée les tableaux et les arraylists. Cette fois-ci, ils doivent être en 2 dimensions pour contenir les données du jour et de la branche.

Exemple pour contenir la salle :

```
        ArrayList<String>[] alistGeneralSalle = new ArrayList[10];
         for(int i= 0;i<5;i++){
             alistGeneralSalle[i] = new ArrayList<>();
         }
```

```
        String[][] arraySalle = new String[10][10];
```

Les données sont réparties de cette manière :
arraySalle [numeroJour][numeroBranche]
Lundi = 0
Mardi = 1 …

2 boucles for sont utilisées cette fois pour mettre toutes les données dans les tableaux.

Voici une partie du code :

```
  for (int iJour = 0; iJour < 5; iJour++) {

                    JSONArray jsonArraygeneral2 = new JSONArray(output[iJour]);
                    for (int iNbBranche = 0; iNbBranche <
jsonArraygeneral2.length(); iNbBranche++) {

                        JSONArray jsonArrayGeneral = new JSONArray(output[iJour]);

                        JSONObject jsonObjectGeneral =
jsonArrayGeneral.getJSONObject(iNbBranche);
                        arrayLibelle[iJour][iNbBranche] =
jsonObjectGeneral.getString("libelle");
                        arrayLibelle[iJour][iNbBranche] =
arrayLibelle[iJour][iNbBranche].substring(0, 1).toUpperCase() +
arrayLibelle[iJour][iNbBranche].substring(1).toLowerCase();
                        arrayCodeMatiere[iJour][iNbBranche] =
jsonObjectGeneral.getString("codeMatiere");
                        arrayHeureDebut[iJour][iNbBranche] =
jsonObjectGeneral.getString("heureDebut");
                        arrayHeureFin[iJour][iNbBranche] =
jsonObjectGeneral.getString("heureFin");
```

Tout d'abord je remplis les branches du lundi. Le principe est la même que celui de la Vue jour. Sauf que cette fois, lorsque toutes les branches du lundi ont été traitées, la boucle passe au jour suivant. Les données sont à nouveau passées dans les arraylists pour pouvoir être traitées plus facilement.

```
                        alistGeneralProf[iJour].add(iNbBranche,
arrayProf[iJour][iNbBranche]);
                        alistGeneralSalle[iJour].add(iNbBranche,
arraySalle[iJour][iNbBranche]);
                        alistGeneralLibelle[iJour].add(iNbBranche,
arrayLibelle[iJour][iNbBranche]);
                        alistGeneralHeureDebutComplet[iJour].add(iNbBranche,
arrayHeureDebut[iJour][iNbBranche]);
                        alistGeneralHeureFinComplet[iJour].add(iNbBranche,
arrayHeureFinComplet[iJour][iNbBranche]);
                        alistCalcul[iJour].add(iNbBranche,
arrayCalcul[iJour][iNbBranche]);
                        alistCalculHeure[iJour].add(iNbBranche,
arrayCalcul[iJour][iNbBranche]);
```

```
                        alistGeneralCodeMatiere[iJour].add(iNbBranche,
arrayCodeMatiere[iJour][iNbBranche]);
```

Un nouveau tableau apparait : NombreInfoRecu[] contient le nombre de branche par jour.

Exemple : NombreInfoRecu[1] = 2 , ça signifie que le mardi contient 2 branches différentes.

Si aucune branche n'est reçue, alors je mets la valeur à 0.

```
                for (int i = 0; i < 5; i++) {
                    if (NombreInfoRecu[i] == null ||
NombreInfoRecu[i].equals("null")) {
                        NombreInfoRecu[i] = 0;
                    }
                }
```

Pour trier et mettre dans l'ordre les branches, le même principe que la vue jour est appliqué.

```
                for (int i = 0; i < 5; i++) {
                    Collections.sort(alistCalcul[i]);
                }
                for (int iJour = 0; iJour < 5; iJour++) {
                    for (int i2 = 0; i2 < NombreInfoRecu[iJour]; i2++) {
                        for (int i = 0; i < NombreInfoRecu[iJour]; i++) {

                            if
(alistCalculHeure[iJour].get(i).equals(alistCalcul[iJour].get(i2))) {
                                alistGeneralLibelle[iJour].add(i2,
arrayLibelle[iJour][i]);
                                alistGeneralHeureDebutComplet[iJour].add(i2,
arrayHeureDebutComplet[iJour][i]);
                                alistGeneralHeureFinComplet[iJour].add(i2,
arrayHeureFinComplet[iJour][i]);
                                alistGeneralProf[iJour].add(i2,
arrayProf[iJour][i]);
                                alistGeneralSalle[iJour].add(i2,
arraySalle[iJour][i]);
                                alistGeneralCodeMatiere[iJour].add(i2,
arrayCodeMatiere[iJour][i]);
                            }
                        }
                    }
                }
```

Après avoir effectué ces étapes, je peux passer à l'affichage des données.

Pour afficher toutes les données, plusieurs possibilités s'offraient à moi. J'ai fait des recherches pour voir si le gridView pouvait m'être utile, mais malheureusement, il est plus utilisé pour créer un tableau dynamique et sans fin. J'ai alors pensé à faire comme j'avais fait dans la vue jour. Créer dynamiquement la grille. J'ai longuement hésité et finalement, je n'ai pas fait de cette manière. En effet, lors de la création de ma grille dans la vue jour, j'ai eu énormément de mal à personnaliser le tableau. L'affichage est très important dans la vue semaine et je devais absolument pouvoir tout contrôler. Ma grille du tableau va alors être fixe et créée en xml. Je suis conscient que cette méthode n'est pas la plus optimisée (>2000 lignes xml), mais je voyais très bien comment je voulais faire. Si le temps à la fin le permettait, j'aurais essayé de créer le même affichage mais cette fois-ci dynamiquement.

Voici comment sont composées 2 cases du tableau :

```
<LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:background="@drawable/border"
                android:id="@+id/ll_Lundi_12h15">

                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"

                    android:id="@+id/tv_Lundi_12h15_libelle"
                    android:gravity="center"/>
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"

                    android:id="@+id/tv_Lundi_12h15_salle"
                    android:gravity="center"/>

            </LinearLayout>

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:background="@drawable/border"
                android:id="@+id/ll_Lundi_13h10">

                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"

                    android:id="@+id/tv_Lundi_13h10_libelle"
                    android:gravity="center"/>
                <TextView
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"

                    android:id="@+id/tv_Lundi_13h10_salle"
                    android:gravity="center"/>

            </LinearLayout>
```

Chaque case contient 1 layout vertical et 2 textview. Toutes les autres cases sont pareilles sauf les id, qui sont à chaque fois différentes.

Dans le code, je crée un tableau arrayPeriodes qui contient toutes les heures conventionnelles (07h25 ,08h10, 08h55…). Ce tableau me permettra de calculer le nombre de périodes que dure la branche, et ainsi colorier plusieurs cases visuellement. Le tableau arrayJourSemaine[] contient le nom des jours de la semaine du lundi au vendredi.

J'instancie un tableau à 2 dimensions (llGeneral) de type LinearLayout. Deux tableaux à 2 dimensions de type TextView (tvGeneralLibelle et tvGeneralSalle) sont également instanciés.

Grâce à des boucles, je vais mettre chaque views des cases (du xml) dans les tableaux correspondants. (Utilisation de findViewById). L'id des view ont tous la même structure ce qui me permet de les mettre facilement dans les tableaux.

Exemple :
llGeneral[0][1] correspond à la case de 08h10-08h55 du lundi
llGeneral[2][3] correspond à la case de 09h40-10h45 du mercredi

```
                for (int iJour = 0; iJour < 5; iJour++) {
                     for (int iPeriode = 0; iPeriode < 19; iPeriode++) {
                         String Idtv = "ll_" + arrayJourSemaine[iJour] + "_" +
arrayPeriodes[iPeriode];
                         int iResID = getResources().getIdentifier(Idtv, "id",
getPackageName());
                         llGeneral[iJour][iPeriode] = (LinearLayout)
findViewById(iResID);
                     }
                 }
```

Après cette étape, je vide toutes les cases de leurs contenus. Ça permet au tableau d'être vide avant de recevoir les nouvelles données. Les cases de midi sont colorées d'une autre couleur.

La fonction RemplissageCases() va mettre les données à afficher dans les textviews, permet d'ajouter une couleur aux branches, de couper les libelles trop long et de centrer le contenu des cases. Pour faire cela, elle a besoin d'avoir l'heure de début de la branche, et de savoir quelle case elle doit remplir.
Voici la boucle faisant l'appel de la fonction RemplissageCases().

```
                for (int iJour = 0; iJour < 5; iJour++) {
                    for (int iBranche = 0; iBranche < NombreInfoRecu[iJour];
iBranche++) {
                        for (int iCpt = 0; iCpt < 19; iCpt++) {
                            if
(alistGeneralHeureDebutComplet[iJour].get(iBranche).equals(arrayPeriodes[iCpt])) {
                                int iNumeroll = iCpt;
                                String strHeureDebut = arrayPeriodes[iCpt];
                                RemplissageCases(iJour, iBranche, iNumeroll,
strHeureDebut, llGeneral, arrayPeriodes, alistGeneralHeureFinComplet,
tvGeneralLibelle, tvGeneralSalle, alistGeneralLibelle, alistGeneralSalle,
alistGeneralProf, alistGeneralCodeMatiere, arrayCouleurs);
                            }
                        }
                    }
                }
```

Le principe est simple : je compare l'heure de début de la branche reçue avec le tableau contenant toutes les périodes. Si par exemple, la branche « allemand » débute à 08h55, alors après deux tours, la condition va être remplie (arrayPeriodes contient 07h25, 08h10 puis 08h55). Le numéro de la case à envoyer est égal au nombre du compteur, dans ce cas-là, 2. Si on regarde l'emplacement 2 du tableau LlGeneral, il correspond à la case de 08h55-09h40 dans le xml.

Cette opération est effectuée pour toutes les branches de tous les jours.

Un problème survient lorsque la branche débute à une heure non conventionnelle. Par exemple 08h30 au lieu de 08h10. La boucle actuelle n'arrive pas à détecter cette branche car 08h30 n'est pas présent dans le tableau arrayPeriodes. Pour essayer de résoudre ce problème, j'ai créé le tableau arrayPeriodesEntre. Dans ce tableau, j'ai entré manuellement les heures possibles d'une case.

Exemple :

```
                arrayPeriodeEntre[0][0] = "07h30";
                arrayPeriodeEntre[0][1] = "07h35";
                arrayPeriodeEntre[0][2] = "07h40";
                arrayPeriodeEntre[0][3] = "07h45";
                arrayPeriodeEntre[0][4] = "07h50";
                arrayPeriodeEntre[0][5] = "07h55";
                arrayPeriodeEntre[0][6] = "08h00";
                arrayPeriodeEntre[0][7] = "08h05";

                arrayPeriodeEntre[1][0] = "08h15";
                arrayPeriodeEntre[1][1] = "08h20";
                arrayPeriodeEntre[1][2] = "08h25";
                arrayPeriodeEntre[1][3] = "08h30";
                arrayPeriodeEntre[1][4] = "08h35";
                arrayPeriodeEntre[1][5] = "08h40";
                arrayPeriodeEntre[1][6] = "08h45";
                arrayPeriodeEntre[1][7] = "08h50";
```

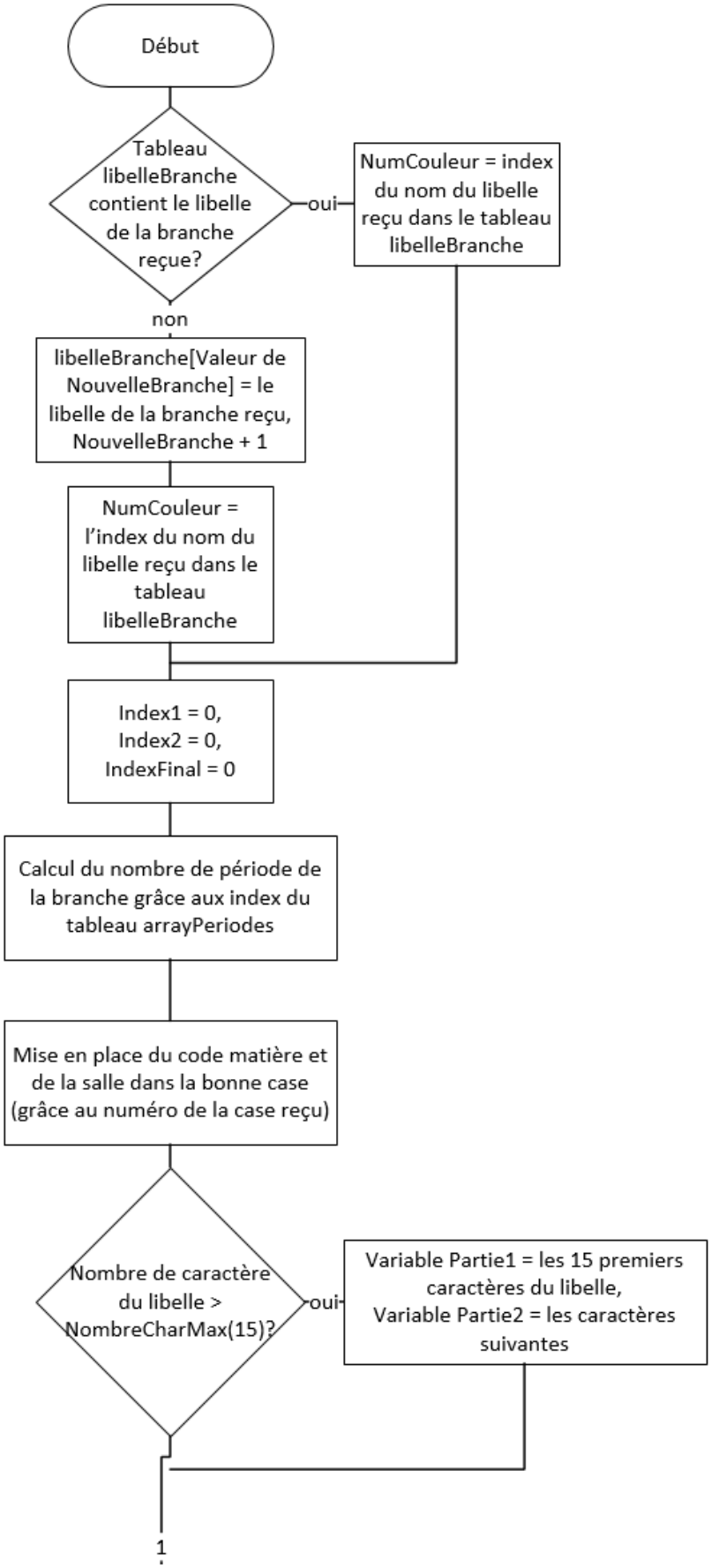
Avec ce tableau je peux cette fois tester si l'heure de début d'une branche correspond à une des heures présentes dans ce nouveau tableau.

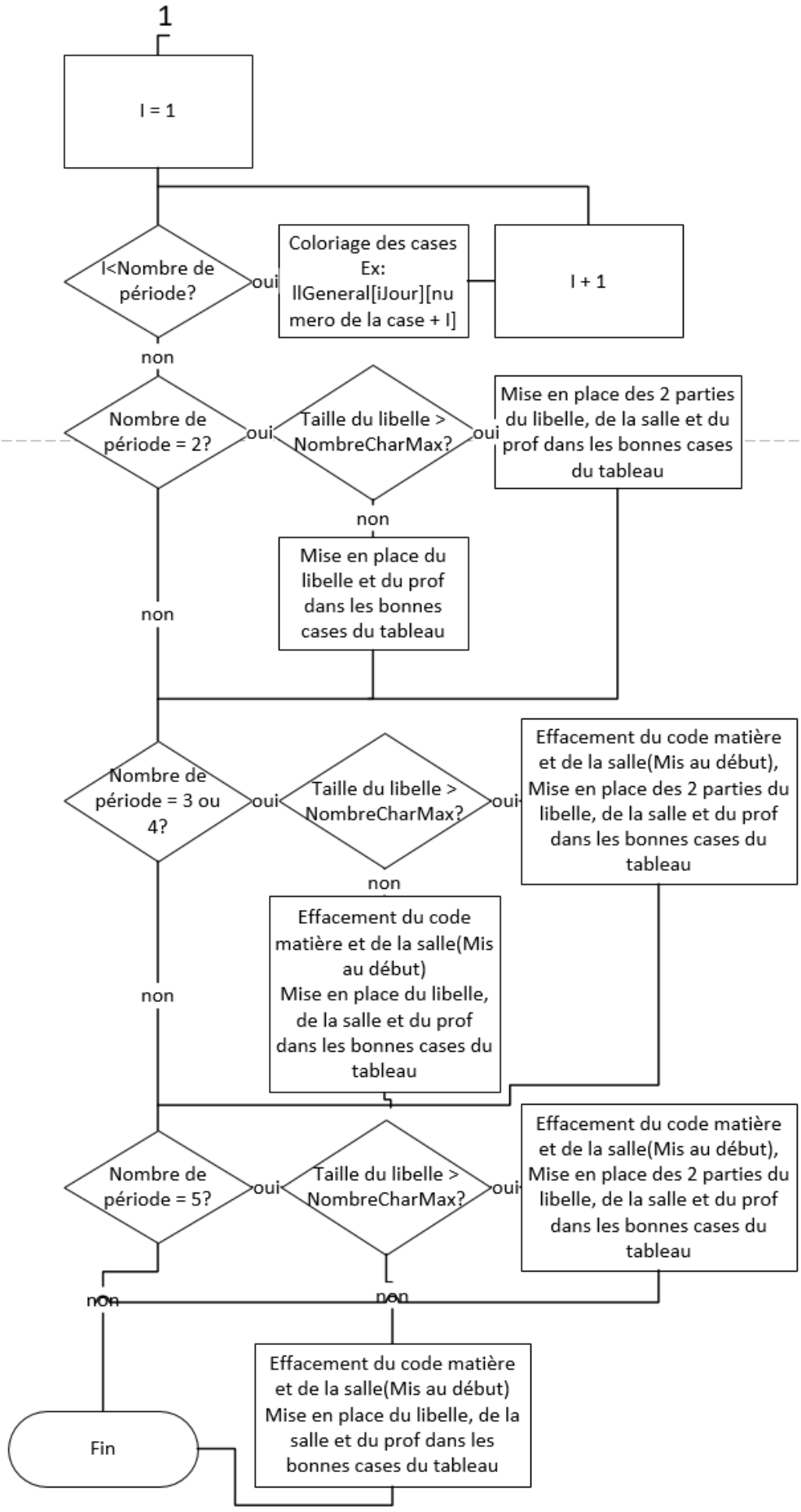
Si c'est le cas, le numéro de la dimension x du tableau est envoyé à la fonction RemplissageCases2().

```
                for(int iJour=0;iJour<5;iJour++) {
                    for (int iBranche = 0; iBranche < NombreInfoRecu[iJour];
iBranche++) {
                        for (int i2 = 0; i2 < 19; i2++) {   // le nombre de
périodes au total.
                            for (int i = 0; i < 8; i++) { // 8 = le nombre de la
seconde dimension du tableau.

if(alistGeneralHeureDebutComplet[iJour].get(iBranche).equals(arrayPeriodeEntre[i2][
i])){
                                    int iNumeroll = i2;
                                    RemplissageCases2(iJour, iBranche,
iNumeroll,tvGeneralLibelle,tvGeneralSalle,alistGeneralLibelle,alistGeneralSalle,ali
stGeneralCodeMatiere,arrayCouleurs);
                                }
                            }
                        }
                    }
                }
```

RemplissageCases() :




1
I = 1
I<Nombre de période?
oui
Coloriage des cases Ex: llGeneral[iJour][numero de la case + I]
I + 1
non
Nombre de période = 2?
oui
Taille du libelle > NombreCharMax?
oui
Mise en place des 2 parties du libelle, de la salle et du prof dans les bonnes cases du tableau
non
Mise en place du libelle et du prof dans les bonnes cases du tableau
non
Nombre de période = 3 ou 4?
oui
Taille du libelle > NombreCharMax?
oui
Effacement du code matière et de la salle(Mis au début), Mise en place des 2 parties du libelle, de la salle et du prof dans les bonnes cases du tableau
non
Effacement du code matière et de la salle(Mis au début) Mise en place du libelle, de la salle et du prof dans les bonnes cases du tableau
non
Nombre de période = 5?
oui
Taille du libelle > NombreCharMax?
oui
Effacement du code matière et de la salle(Mis au début), Mise en place des 2 parties du libelle, de la salle et du prof dans les bonnes cases du tableau
non
non
Effacement du code matière et de la salle(Mis au début) Mise en place du libelle, de la salle et du prof dans les bonnes cases du tableau
Fin

Première chose que la fonction fait : la gestion des couleurs. La première condition teste si le libellé reçu est déjà présent dans le tableau libelleBranche[]. Si c'est le cas, alors iNumCouleur = l'index du libelle dans le tableau. Autrement, j'ajoute le nouveau libelle dans le tableau, et je récupère à nouveau l'index que je place dans la variable iNumCouleur. Elle servira à savoir de quelle couleur il faut colorier la branche.

La fonction va ensuite mettre la couleur à la bonne case du tableau (iNumeroll). Ce numéro a été reçu en paramètre.

```
        llGeneral[iJour][iNumeroll].setBackgroundColor(arrayCouleurs[iNumCouleur]);
```

Il faut ensuite calculer le nombre de périodes de la branche reçue. Je me sers alors du tableau arrayPeriode[]. Je soustrais simplement l'index de l'heure de début à l'index de l'heure de fin.

Exemple :

Heure de début : 08h10
Heure de fin : 09h40

Index de 08h10 dans arrayPeriodes = 1
Index de 09h40 dans arrayPeriodes = 3

3-1 = 2 périodes.

Avant de m'occuper des branches ayant plusieurs périodes, je mets le code matière et la salle dans la case du tableau.

```
        tvLibelle[iJour][iNumeroll].setText(AlistCodeMatiere[iJour].get(iBranche));
        tvSalle[iJour][iNumeroll].setText(AlistSalle[iJour].get(iBranche));
```

Je compte le nombre de caractères du libelle, et s'il dépasse le nombre maximal (15), alors je le coupe en deux.

Si la branche a plusieurs périodes, une boucle for va venir colorier visuellement les cases.
iIndexFinal représente le nombre de période de la branche. Reprenons l'exemple de l'allemand qui commence à 08h10 et qui se termine à 09h40. La branche dure donc 2 périodes (iIndexFinal = 2). INumeroll représente la case de 08h10-08h55 dans le tableau xml. iNumeroll + 1 représente la case suivant, soit : 08h55-09h40. Vu que la boucle fait un tour, cette dernière case est coloriée. Visuellement l'espace entre 08h10 et 09h40 est de la même couleur.

```
        for (int i = 1; i < iIndexFinal; i++) {
            llGeneral[iJour][iNumeroll +
i].setBackgroundColor(arrayCouleurs[iNumCouleur]);
        }
```

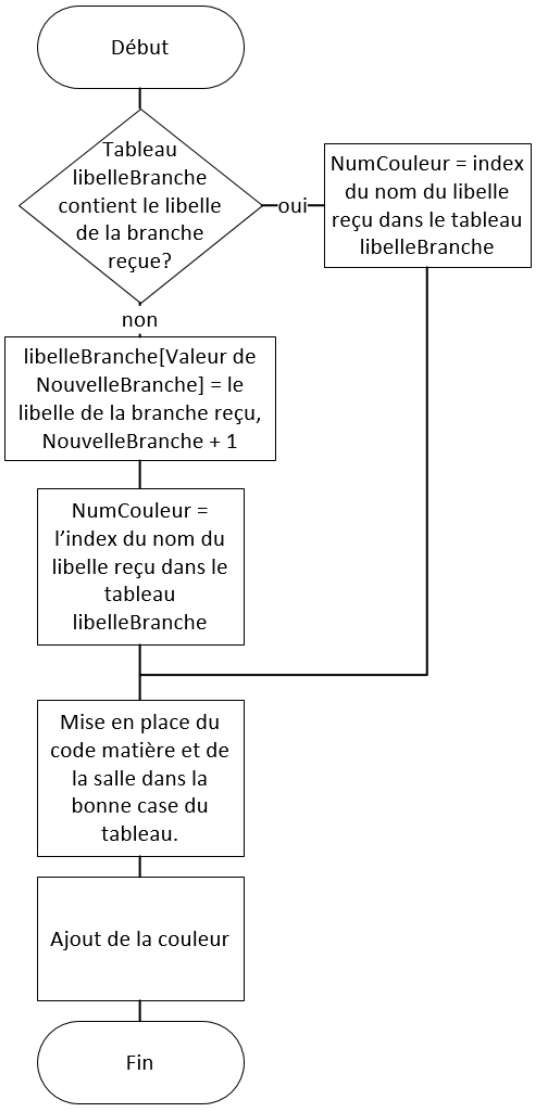
Maintenant, si le nombre de périodes est égal à 2, alors la fonction va afficher le libelle de la branche à la place du code matière. La salle et le professeur sont également affichés. Si le libellé a été coupé en 2 parties, alors chaque partie est mise dans un textview différent.

```
        if (iIndexFinal == 2) {
            if (iLength > iNombreCharMax) {
                tvLibelle[iJour][iNumeroll].setText(strPartie1 + "-");
                tvSalle[iJour][iNumeroll].setText(strPartie2);
                tvLibelle[iJour][iNumeroll +
1].setText(AlistSalle[iJour].get(iBranche));
                tvSalle[iJour][iNumeroll +
1].setText(AlistProf[iJour].get(iBranche));
            } else {

tvLibelle[iJour][iNumeroll].setText(AlistLibelle[iJour].get(iBranche));
                tvLibelle[iJour][iNumeroll +
1].setText(AlistProf[iJour].get(iBranche));
            }
        }
```

Le même principe s'applique lorsque le nombre de période est égal à 3, 4 ou 5.

RemplissageCases2() :



Comme avant, la fonction s'occupe de la couleur. Il n'y a pas de calcul de périodes, car je n'ai pas encore réussi à implémenter cette fonctionnalité avec les heures non conventionnelles. Je ne peux pas appliquer la même méthode que celle utilisée auparavant. Le code matière et la salle vont être mis dans la bonne case du tableau.

3.4 Cas d'utilisation « Affichage de l'horaire d'une semaine avec classe et date choisie par l'utilisateur »

Lors de l'ouverture de la vue semaine, la recherche est automatiquement exécutée.
L'utilisateur doit pouvoir choisir la classe et la date qu'il veut puis, à l'appui du bouton Rechercher, le programme doit afficher l'horaire de la classe correspondante.

3.4.1 Scénario

1. L'utilisateur tape le nom de la classe voulue dans le champ classe.
2. Le système suggère la liste de toutes les classes disponibles.
3. L'utilisateur tape le nom de la date voulue dans le champ date.
4. L'utilisateur appuie sur le bouton Rechercher.
5. Le système met dans des variables le contenu des champs, vérification des saisies.
6. Le système récupère les dates de la semaine.
7. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes).
8. Le système effectue des vérifications des données reçues.
9. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
10. Le système met le nom de la classe recherchée dans l'historique.
11. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
12. Le système met les données de l'horaire reçues dans des variables.
13. Le système tri les branches de manière chronologique à leurs heures de début.
14. Le système affiche l'horaire.

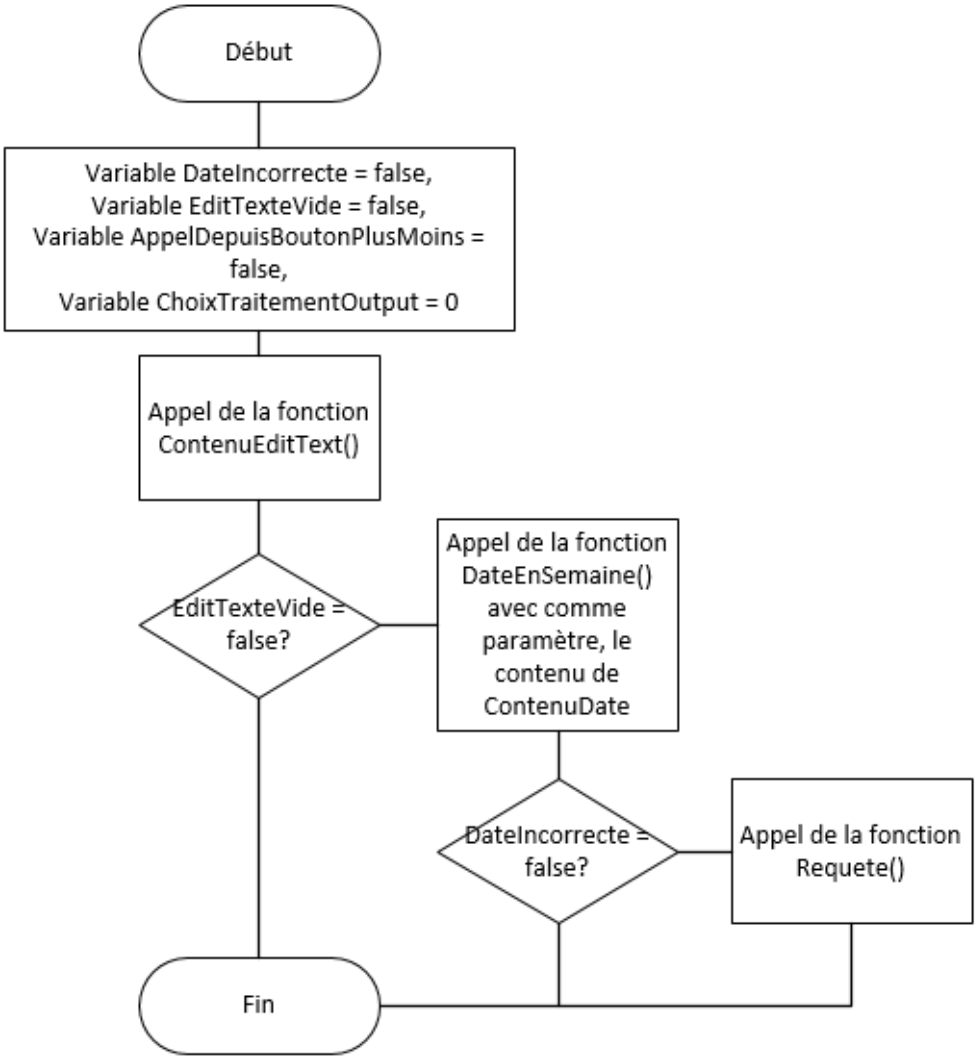
3.4.2 Maquette

Même maquette que le cas d'utilisation précédent.

3.4.3 Analyse du scénario

3.4.3.1 Algorithme

Lors du clic sur le bouton Rechercher :



3.4.3.2 Explications détaillées

Le développement est très similaire au cas d'utilisation précédent. Au lieu d'utiliser la dernière classe recherchée et la date du jour actuelle, le programme se sert du contenu des champs date et classe. Des vérifications sont effectuées pour détecter les champs vides et les dates incorrectes. Si plusieurs classes sont retournées, alors l'utilisateur devra choisir la bonne. Toute la partie requête et affichage des données est pareille que le cas d'utilisation précédent.

3.5 Cas d'utilisation « Modifier la date avec des boutons »

Pour pouvoir faciliter l'utilisation de mon application, l'utilisateur peut, en appuyant sur des boutons, changer la date du jour. La vue jour dispose de 4 boutons. Le bouton « -- » permet de diminuer la date du jour d'une semaine et de lancer la recherche. Pour diminuer d'un jour, l'utilisateur appuie sur le bouton « - ». Les boutons « ++ » et « + » augmente la date d'une semaine et d'un jour.

La vue semaine dispose uniquement de deux boutons. Ils permettent d'augmenter/diminuer la date d'une semaine entière (7 jours).

3.5.1 Scénario

Vue jour :

1. L'utilisateur appuie sur un des quatre boutons présents.
2. Le système modifie la date de x jours en fonction du bouton appuyé.
3. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toutes les classes).
4. Le système effectue des vérifications des données reçues.
5. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
6. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
7. Le système met les données de l'horaire reçues dans des variables.
8. Le système tri les branches de manière chronologique à leurs heures de début.
9. Le système affiche l'horaire.

Vue semaine :

1. L'utilisateur appuie sur un des deux boutons présents.
2. Le système modifie la date de x jours en fonction du bouton appuyé.
3. Le système récupère les dates de la semaine.
4. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes).
5. Le système effectue des vérifications des données reçues.
6. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
7. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
8. Le système met les données de l'horaire reçues dans des variables.
9. Le système tri les branches de manière chronologique à leurs heures de début.
10. Le système affiche l'horaire.

3.5.2 Maquette

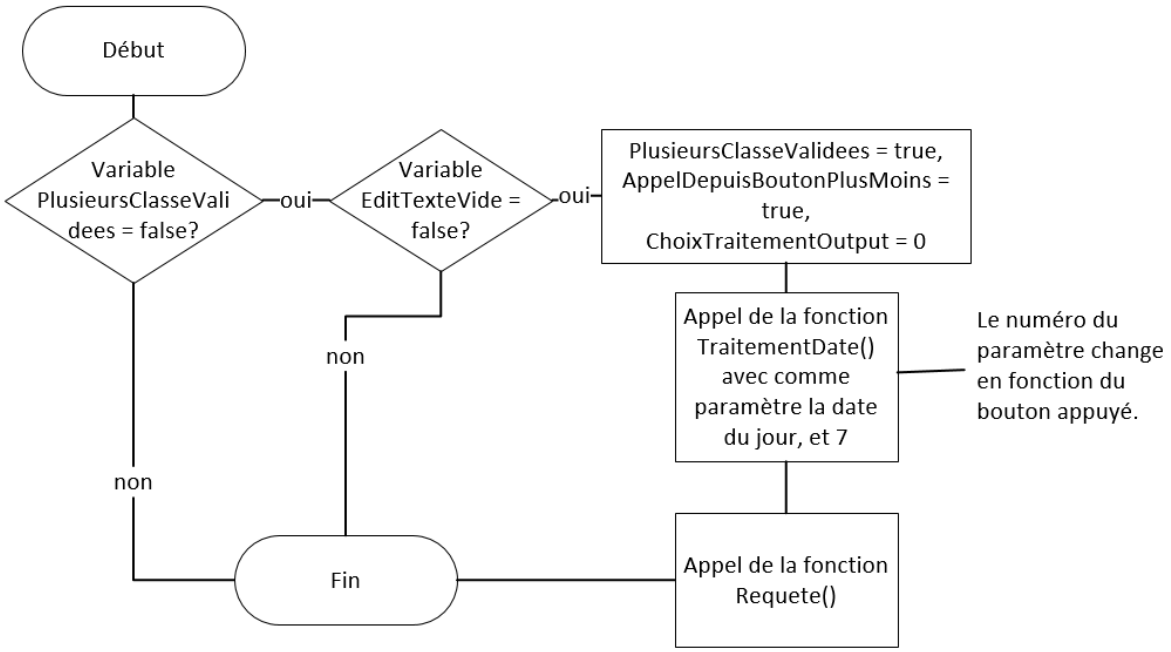
Même maquette que les cas d'utilisations précédents.

3.5.3 Analyse du scénario

3.5.3.1 Algorithme

Vue jour :

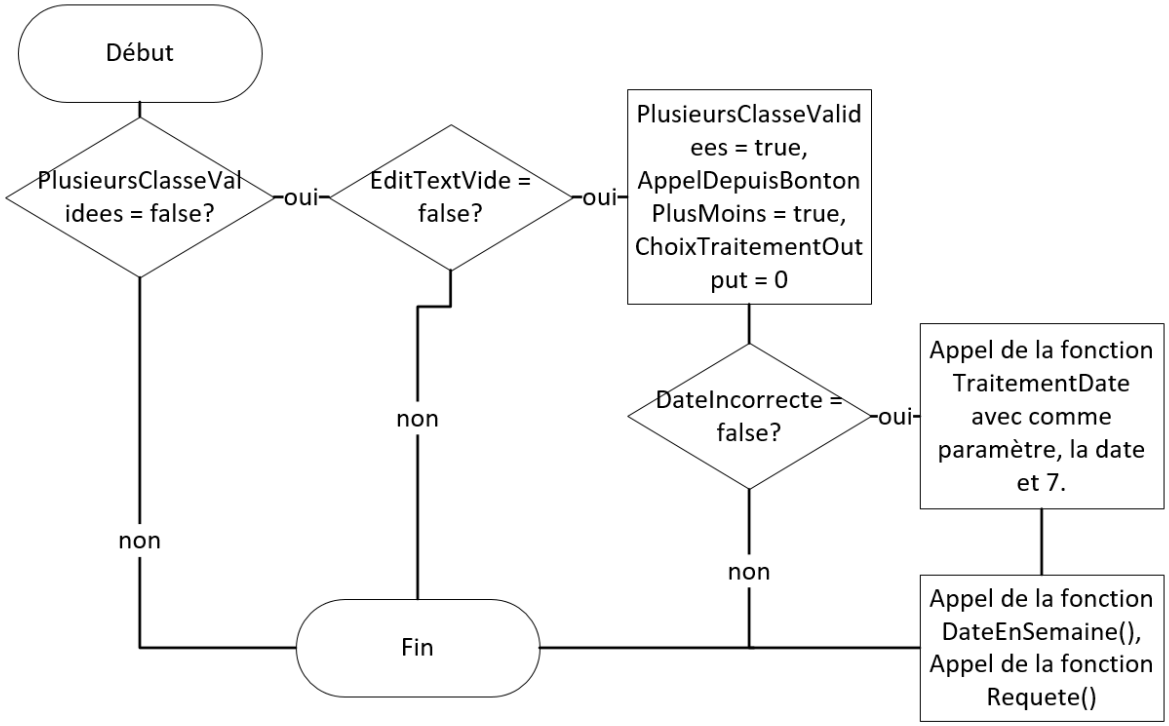
Lors de l’appui sur le bouton « ++ » :



Lors de l’appui sur le bouton « + » :
Même diagramme que le précédent avec cette fois ci 1 au lieu de 7 lors de l’appel de la fonction TraitementDate().
Lors de l’appui sur le bouton « - » :
Même diagramme que le précédent avec cette fois ci -1 au lieu de 7 lors de l’appel de la fonction TraitementDate().
Lors de l’appui sur le bouton « -- » :
Même diagramme que le précédent avec cette fois ci -7 au lieu de 7 lors de l’appel de la fonction TraitementDate().

Vue Semaine :

Lors de l’appui sur le bouton « ++ » :



Lors de l'appui sur le bouton « -- » :
Même diagramme que le précédent avec cette fois ci -7 au lieu de 7 lors de l'appel de la fonction TraitementDate().

3.5.3.2 Explications détaillées

Vue jour :

Tout d'abord, lorsque l'utilisateur appuie sur un bouton (par exemple augmenter d'une semaine), deux tests sont effectués. Si la variable bPlusieursClasseValidees = false, alors on continue. Ça va permettre de bloquer les 4 boutons lorsque l'utilisateur doit choisir parmi la liste des classes retournées. La deuxième vérification se fait avec la variable bEditTextVide. Si elle est égale à false, on continue. C'est comme avant, pour bloquer les boutons lorsque l'utilisateur a laissé un champ vide. Une fois ces 2 tests passés, je mets à true la variable bPlusieursClasseValidees et bAppelDepuisBoutonPlusMoins. La première variable va permettre de ne pas afficher la liste des classes, lorsque plusieurs classes sont reçues. Sans cela, si l'utilisateur tape « 3m3i1 » (retourne 3 classes), la liste des classes retournées va s'afficher à chaque appui sur un des boutons. La deuxième variable détermine si on doit enregistrer le nom de la classe dans le fichier historique. Cette étape a déjà été faite lors de l'appui sur le bouton Rechercher, donc ce n'est pas nécessaire d'enregistrer à nouveau le nom de la classe. IChoixTraitementOutput est remis à 0 pour permettre d'aiguiller les résultats des requêtes. La fonction iChoixTraitementDate() est appelée. Elle a déjà été utilisée lors de la création de l'activité mais avec en paramètre 0. Ce chiffre détermine de combien de jour il faut augmenter/diminuer la date. Dans notre cas, lors de l'appui sur le bouton « ++ », on envoie 7. La requête est finalement lancée. La suite se déroule de la même manière et a déjà été expliquée.

Vue Semaine :
Le raisonnement de la vue semaine ressemble beaucoup à celle de la vue jour. Les deux tests sont effectués et les 2 variables sont passées à true. Cette fois, on a un troisième test qui va passer à la suite uniquement si la date entrée est valide. La date est modifiée avec la fonction TraitementDate(). On remplit ensuite le tableau contenant les dates de la semaine actuelle à l'aide de DateEnSemaine(). La requête est lancée. La suite est pareille et a déjà été expliquée.

3.5.4 La phase de programmation

Vue jour :

Diminution d'un jour à la date

```
                if (!bPlusieursClasseValidees) {
                    if (!bEditTexteVide) {
                        bPlusieursClasseValidees = true;
                        bAppelDepuisBoutonPlusMoins = true;
                        iChoixTraitementOutput = 0;
                        TraitementDate(strContenuDate, -1);
                        Requete();
                    }
                }
```

Vue semaine :

Ajout d'une semaine à la date

```
   if(!bPlusieursClasseValidees) {
                    if (!bEditTexteVide) {
                        bPlusieursClasseValidees = true;
                        bAppelDepuisBoutonPlusMoins = true;
                        iChoixTraitementOutput = 0;
                        if (!bDateIncorrecte) {
                            TraitementDate(strContenuDate, 7);
                            DateEnSemaine(strContenuDate);
                            Requete();
                        }
                    }
                }
```

3.6 Cas d'utilisation « Modifier la date en balayant l'écran »

En plus de pouvoir modifier la date avec des boutons, l'utilisateur peut également le faire en balayant l'écran de gauche à droite et inversement. Cette action modifie la date de 1 jour dans la vue jour, et de 7 jours dans la vue semaine.

Une fois le mouvement de balayage détecté, le code permettant de changer la date et de lancer les requêtes est exactement le même que lors de l'appui sur les boutons « + » et moins « - ».

3.6.1 Scénario

Vue jour :

1. L'utilisateur balaye de gauche à droite ou inversement l'écran.
2. Le système modifie la date de x jours en fonction du bouton appuyé.
3. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes).
4. Le système effectue des vérifications des données reçues.
5. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.

6. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
7. Le système met les données de l'horaire reçues dans des variables.
8. Le système tri les branches de manière chronologique à leurs heures de début.
9. Le système affiche l'horaire.

Vue semaine :

1. L'utilisateur balaye de gauche à droite ou inversement l'écran.
2. Le système modifie la date de x jours en fonction du bouton appuyé.
3. Le système récupère les dates de la semaine.
4. Le système établit une connexion avec le serveur et lance les requêtes (Réception id, et liste de toute les classes)
5. Le système effectue des vérifications des données reçues.
6. Le système lance la requête pour avoir l'horaire de la classe avec l'id de la classe.
7. Le système lit l'historique pour mettre à jour le tableau contenant l'historique.
8. Le système met les données de l'horaire reçues dans des variables.
9. Le système tri les branches de manière chronologique à leurs heures de début.
10. Le système affiche l'horaire.

3.6.2 Maquette

Pas de maquette pour ce cas d'utilisation.

3.6.3 Analyse du scénario

3.6.3.1 Algorithme

Vue jour :

Lors du balayage de droite à gauche :

Même diagramme que lors de l'appui sur le bouton « + »

Lors du balayage de gauche à droite :

Même diagramme que lors de l'appui sur le bouton « - »

Vue Semaine :

Lors du balayage de droite à gauche :

Même diagramme que lors de l'appui sur le bouton « ++ »

Lors du balayage de gauche à droite :

Même diagramme que lors de l'appui sur le bouton « -- »

3.6.3.2 Explications détaillées

Vue jour :
Pour détecter les balayages, je me suis largement inspiré d'une réponse postée sur un forum (lien dans les sources). L'utilisateur conseillait de créer une class séparée qui détecte l'orientation du balayage. Une fois le sens détecté, j'ai mis le même code que celui des boutons augmentant et diminuant la date.

Vue Semaine :

Même explication que la vue jour.

3.6.4 La phase de programmation

SwipeListener.java

```
public class SwipeListener implements OnTouchListener {
    private final GestureDetector gestureDetector;
    public SwipeListener (Context ctx){
        gestureDetector = new GestureDetector(ctx, new GestureListener());
    }
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        return gestureDetector.onTouchEvent(event);
    }
    private final class GestureListener extends SimpleOnGestureListener {
        private static final int SWIPE_THRESHOLD = 100;
        private static final int SWIPE_VELOCITY_THRESHOLD = 100;
        @Override
        public boolean onDown(MotionEvent e) {
            return true;
        }
        @Override
        public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
float velocityY) {
            boolean result = false;
            try {
                float diffY = e2.getY() - e1.getY();
                float diffX = e2.getX() - e1.getX();
                if (Math.abs(diffX) > Math.abs(diffY)) {
                    if (Math.abs(diffX) > SWIPE_THRESHOLD && Math.abs(velocityX) >
SWIPE_VELOCITY_THRESHOLD) {
                        if (diffX > 0) {
                            onSwipeRight();
                        } else {
                            onSwipeLeft();
                        }
                        result = true;
                    }
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }
            return result;
        }
    }
    public void onSwipeRight() {
    }
    public void onSwipeLeft() {
    }
}
```

3.7 Cas d'utilisation « Sélectionner la date avec un calendrier »

En appuyant sur le bouton calendrier, l'application doit afficher un sélecteur de date pour que l'utilisateur puisse choisir plus facilement la date voulue. Une fois la date choisie, on la mettra dans le champ date. Ce bouton est présent dans les deux activités.

3.7.1 Scénario

Scénario Vue Jour :

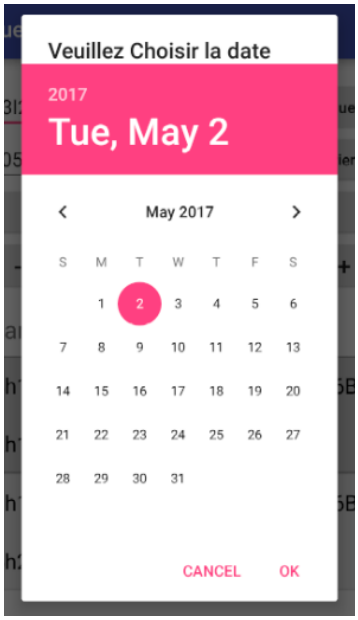
1. L'utilisateur appuie sur le bouton calendrier.
2. L'application affiche un sélecteur de date.
3. L'utilisateur choisit la date voulue et appuie sur « Ok ».
4. L'application met la date choisie dans le champ date.

Scénario Vue Semaine :

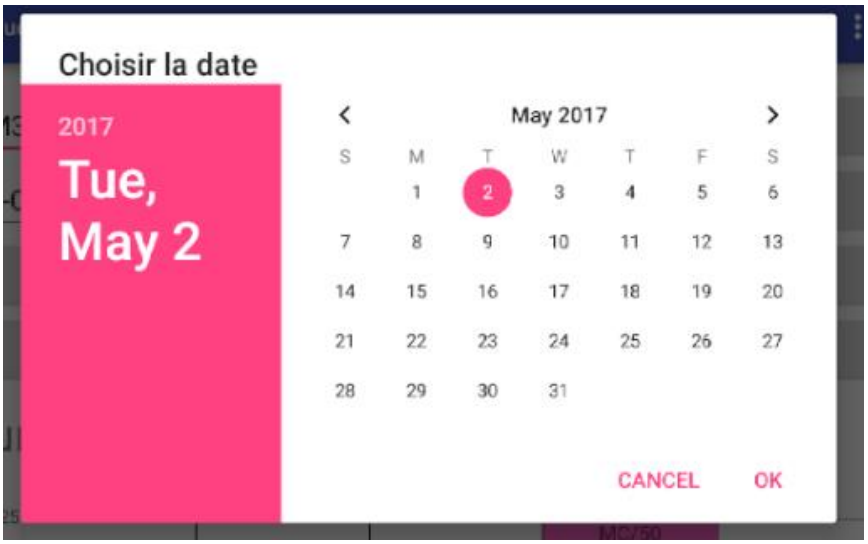
Même Scénario que la vue jour.

3.7.2 Maquette

Vue jour :



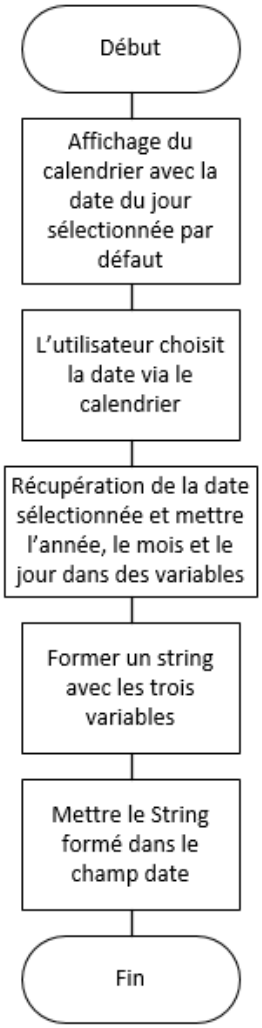
Vue Semaine :



3.7.3 Analyse du scénario

3.7.3.1 Algorithme

Vue Jour :
Lors du clic sur le bouton Calendrier :



Vue Semaine :

Même algorithme que la vue jour.

3.7.3.2 Explications détaillées

Activité jour :

Dans le onCreate() de l'activité, je mets un listener de clic sur le bouton Rechercher. Lors de l'appui sur le bouton, j'affiche un DatePickerDialog. La date du jour actuelle est sélectionnée par défaut. Une fois que l'utilisateur a choisi la date, je place la date choisie dans des variables. Avec ces variables Jour, Mois et Année, je peux créer un string formaté de la bonne manière (séparé par des tirets) que je placerais dans le champ date.

Activité semaine :

Même explication que la vue jour.

3.7.4 La phase de programmation

Vue jour :

Création du string qui va être placé dans le champ date. Le numéro du mois commence par 0, alors je dois ajouter 1.

```
String strDateComplete = String.valueOf(iJour) + "-" + String.valueOf(iMois + 1) +
"-" + String.valueOf(iAnnee);
```

Vue semaine :

Même Programmation que la vue jour.

3.8 Cas d'utilisation « Basculer entre la vue semaine et la vue jour »

L'utilisateur peut basculer de vue via le menu.

3.8.1 Scénario

Vue jour :

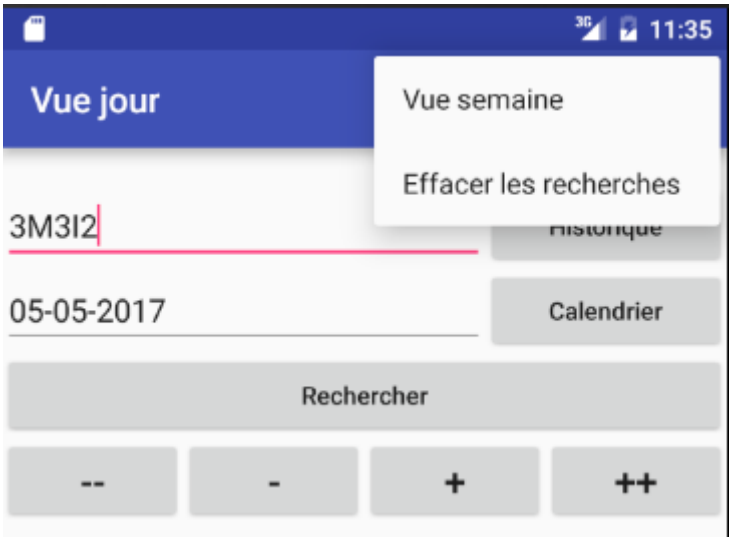
1. L'utilisateur appuie sur le bouton menu.
2. L'utilisateur appuie sur le bouton « Vue semaine ».
3. Le système vérifie si la connexion internet est disponible.
4. Le système passe à la vue semaine.

Vue semaine :

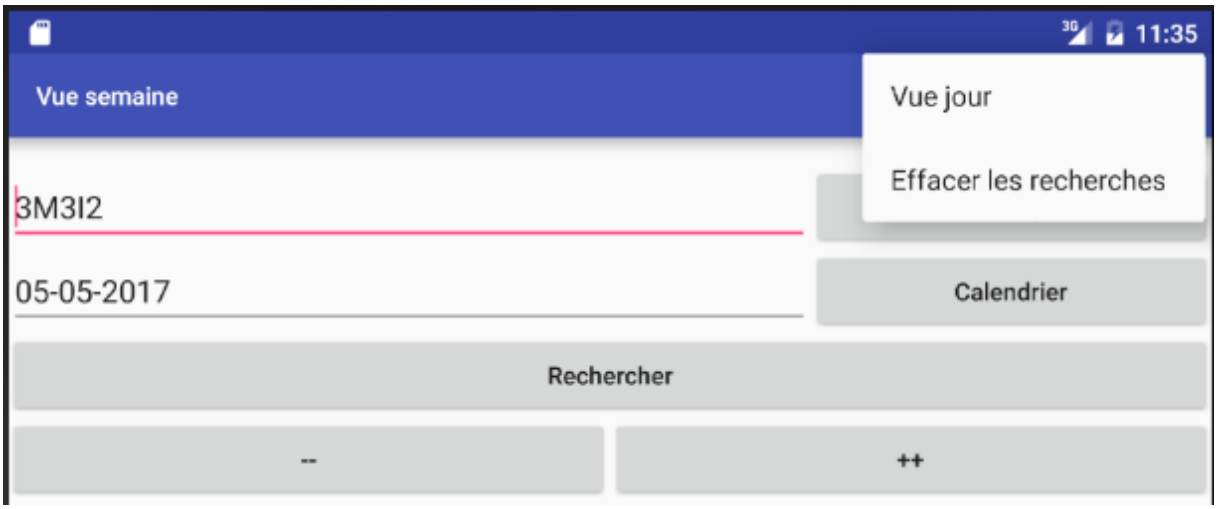
1. L'utilisateur appuie sur le bouton menu.
2. L'utilisateur appuie sur le bouton « Vue jour ».
3. Le système passe à la vue jour.

3.8.2 Maquettes

Vue jour :



Vue semaine :

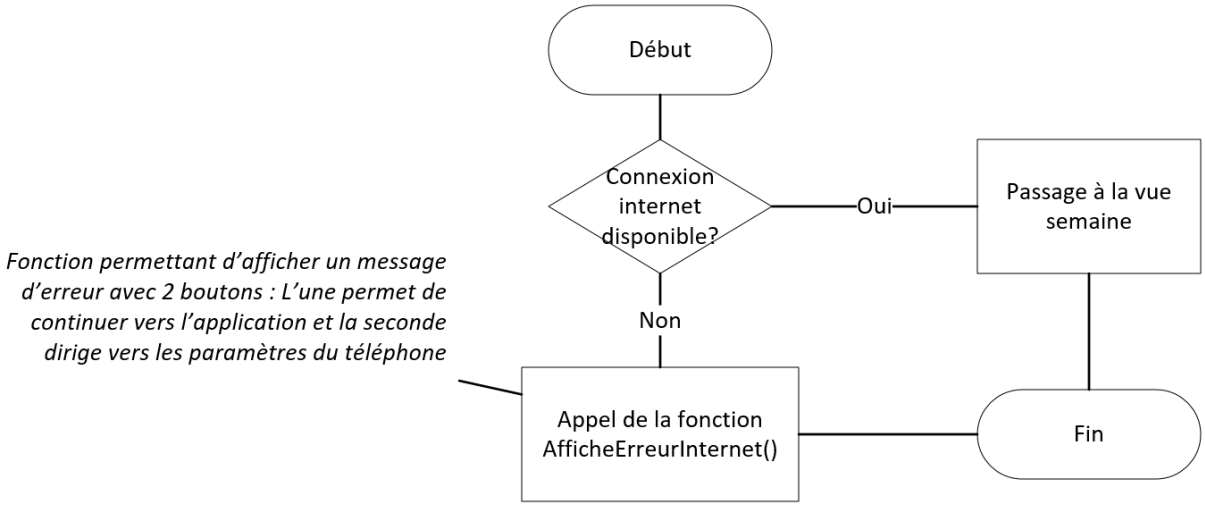


3.8.3 Analyse du scénario

3.8.3.1 Algorithme

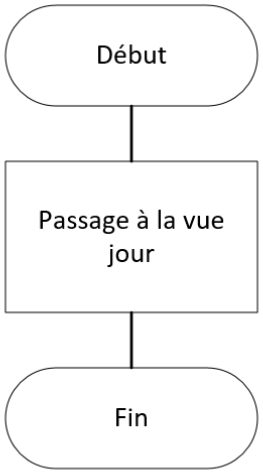
Vue jour :

Fonction VueSemaine()



Vue semaine :

Fonction VueJour() :



3.8.3.2 Explications détaillées

Vue jour :
Lorsque l'utilisateur appuie sur le bouton « Vue semaine » depuis le menu, on va vérifier que l'utilisateur ait bien une connexion internet. Si c'est le cas, alors on passe à la vue Semaine. Autrement un message d'erreur permettant d'accéder aux paramètres apparait. Le menu était très simple à mettre en place. Il fallait créer un fichier menu.xml, est l'utiliser dans l'activité. Lors de l'appui sur chacune des options, on appelle la fonction correspondante.

Vue Semaine :

La vérification internet n'est pas nécessaire dans cette vue et le bouton « Vue jour » permet directement de passer à la vue jour.

3.8.4 La phase de programmation

Vue jour :

Création du menu avec le fichier menu.xml

```
public boolean onCreateOptionsMenu(Menu menu){
        getMenuInflater().inflate(R.menu.menu, menu);
        return true;
    }
```

Menu.xml
Contiens-les 2 options possibles du menu, la deuxième option est documentée dans le cas d'utilisation « Effacer l'historique des recherches ».

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_changer_vue"
        android:title="Vue semaine"
        app:showAsAction="never" /
    <item
        android:id="@+id/action_effacer"
        android:title="Effacer les recherches"
        app:showAsAction="never" />
</menu>
```

Lancement de la fonction VueSemaine() lorsque que l'utilisateur appuie sur la bouton « Vue Semaine ».

```
public boolean onOptionsItemSelected(MenuItem item){
        switch (item.getItemId()){
            case R.id.action_changer_vue:
                VueSemaine();
                return true;
…
```

Vue Semaine :

Création du menu avec le fichier menu2.xml. Ce fichier-là affiche « Vue jour » dans le menu alors que menu.xml affiche « Vue Semaine ».

```
public boolean onCreateOptionsMenu(Menu menu){
        getMenuInflater().inflate(R.menu.menu2, menu);
        return true;
    }
```

3.9 Cas d'utilisation « Afficher l'historique des recherches »

L'utilisateur doit pouvoir accéder à une liste des classes déjà recherchées. A chaque requête valide que l'utilisateur fait depuis le bouton rechercher, le contenu du champ de la classe est inscrit dans un fichier. Les deux activités se partagent le fichier et peuvent y accéder.

3.9.1 Scénario

Vue jour :

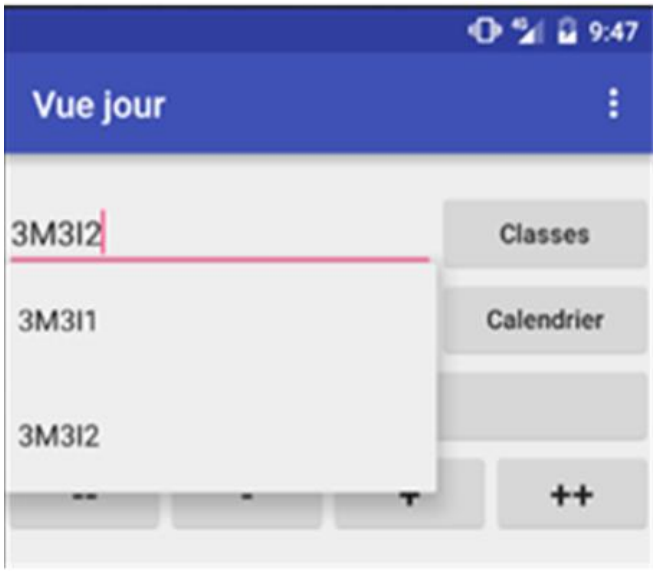
1. L'utilisateur appuie sur le bouton historique.
2. Le système met à jour l'AutoCompleteTextView (Actv) avec l'arraylist contenant le l'historique des recherches.
3. Le système affiche l'Actv.
4. Le système change le texte du bouton.

Vue semaine :

Même scénario que la vue jour.

3.9.2 Maquette

Vue jour :



Vue semaine :

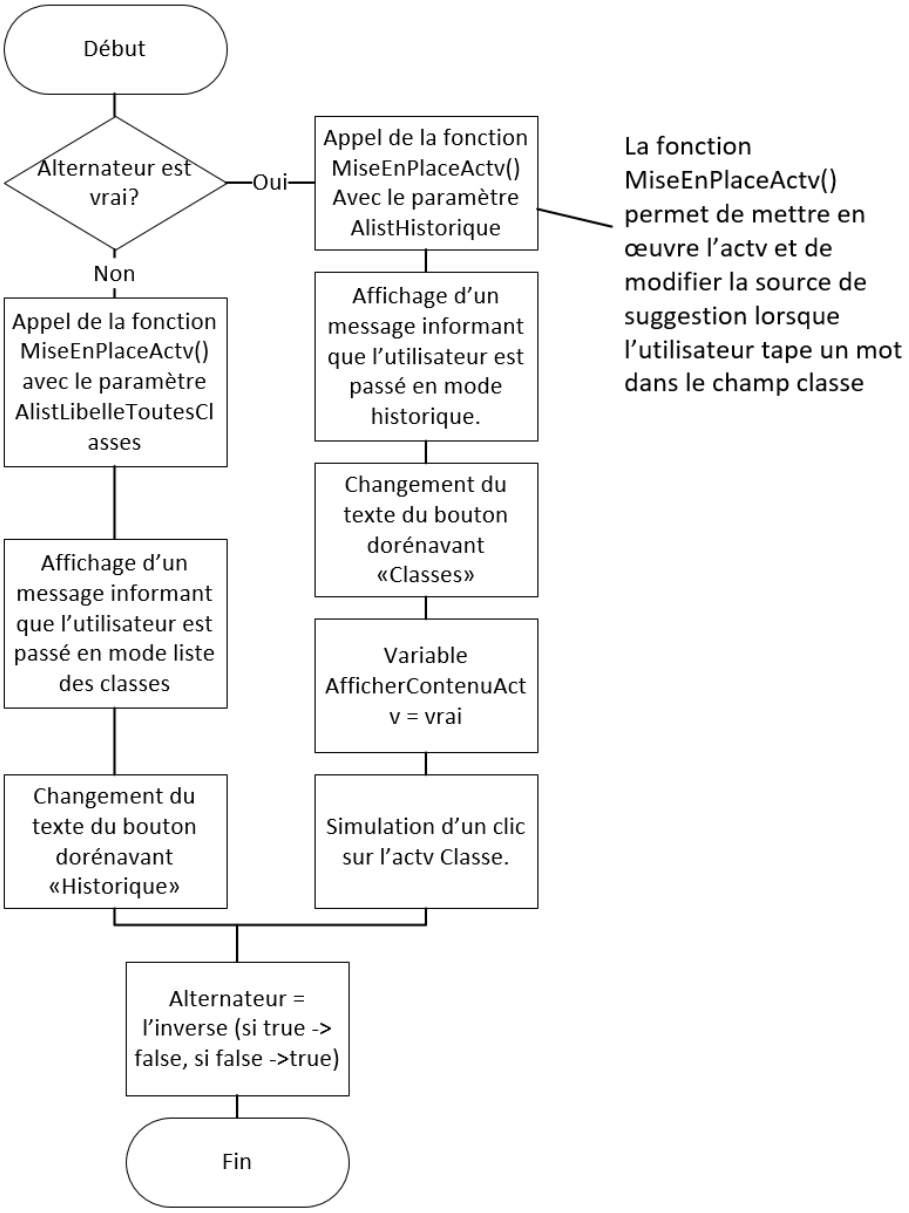
Même maquette que la vue jour.

3.9.3 Analyse du scénario

3.9.3.1 Algorithme

Vue jour :

Lors de l'appui sur le bouton « Historique/classes ».



Vue Semaine :

Même Algorithme que la vue Jour.

3.9.3.2 Explications détaillées

Tout d'abord, dès la création de l'activité, la variable bAlternateur est égale à true. Cette variable sert à déterminer quel Arraylist on va utiliser pour alimenter les suggestions du champ classe. Quand l'utilisateur va appuyer la première fois sur le bouton historique, la source de l'actv va être modifiée pour suggérer la liste des classes déjà recherchées. Un message d'information va apparaitre pour notifier l'utilisateur du changement de mode. Le texte du bouton va être modifié pour devenir « Classes ». L'utilisateur, en appuyant sur le bouton historique, s'attend à une liste des classes recherchées. Pour cela, la variable AfficherContenuActv va devenir true. Ça permet au clic du champ classe, d'afficher tout le contenu du tableau qui alimente l'actv. Il ne reste plus qu'à simuler un clic sur le champ pour ainsi afficher la liste.

Si au clic du bouton, bAlernateur est égale à false, alors ça signifie que l'utilisateur est actuellement en mode historique. Dans ce cas, l'actv va être alimenté par l'arraylist contenant la liste de toutes les classes disponibles. Comme avant, le bouton change de texte et un message informatif apparait.

Après toutes ces étapes, bAlternateur va devenir l'inverse de ce qu'il était. Ça permet, lors du prochain appui sur le bouton, de passer par l'autre condition, et d'alterner le mode d'affichage.

Vue Semaine :

Même explication que la vue jour.

3.10 Cas d'utilisation « Effacer l'historique des recherches »

L'utilisateur peut effacer la liste des classes déjà recherchées en appuyant sur le bouton « Effacer les recherches ». Après avoir appuyé sur ce bouton, les suggestions du champ classe affichent de nouveau toutes les classes disponibles.

3.10.1 Scénario

Vue jour :

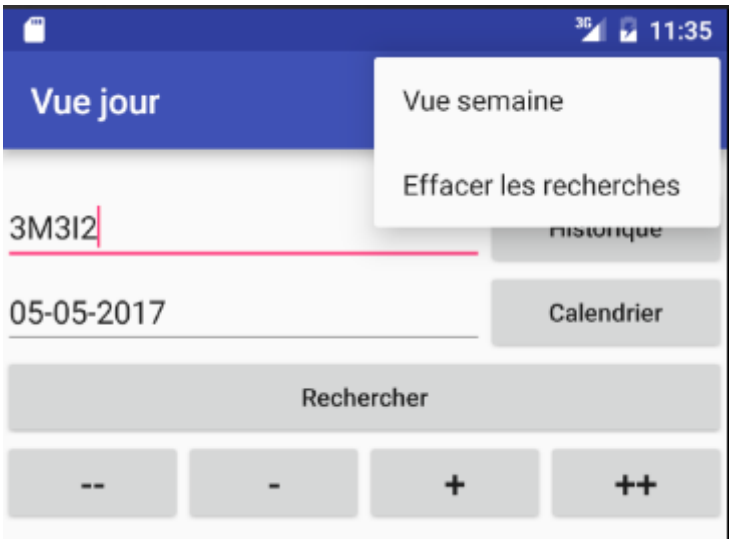
1. L'utilisateur appuie sur le bouton du menu.
2. L'utilisateur appuie sur « Effacer les recherches ».
3. Le système efface les recherches déjà effectuées.
4. Le système change la source des suggestions du champ classe.

Vue Semaine :

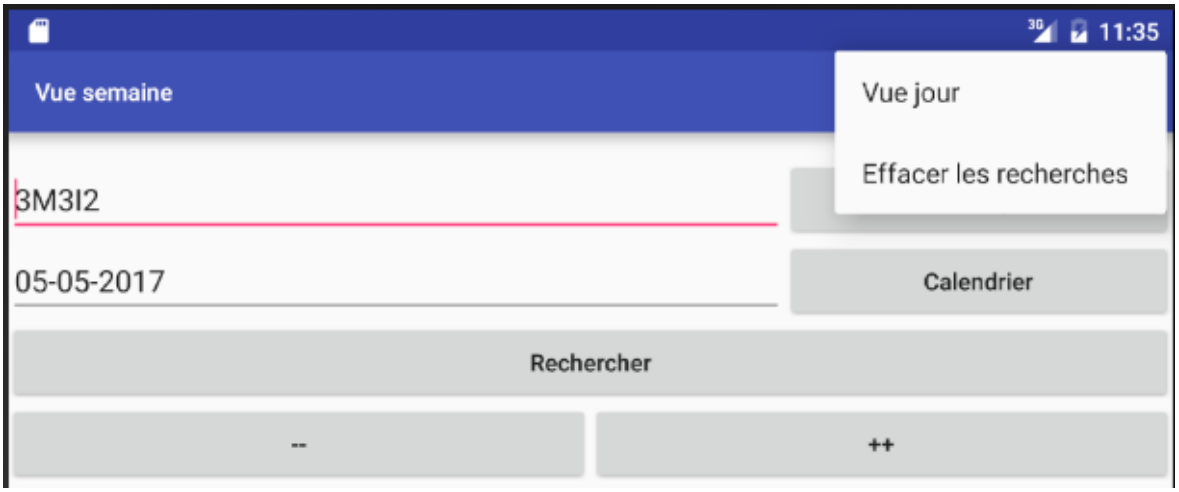
Même scénario que dans la vue jour.

3.10.2 Maquettes

Vue jour :



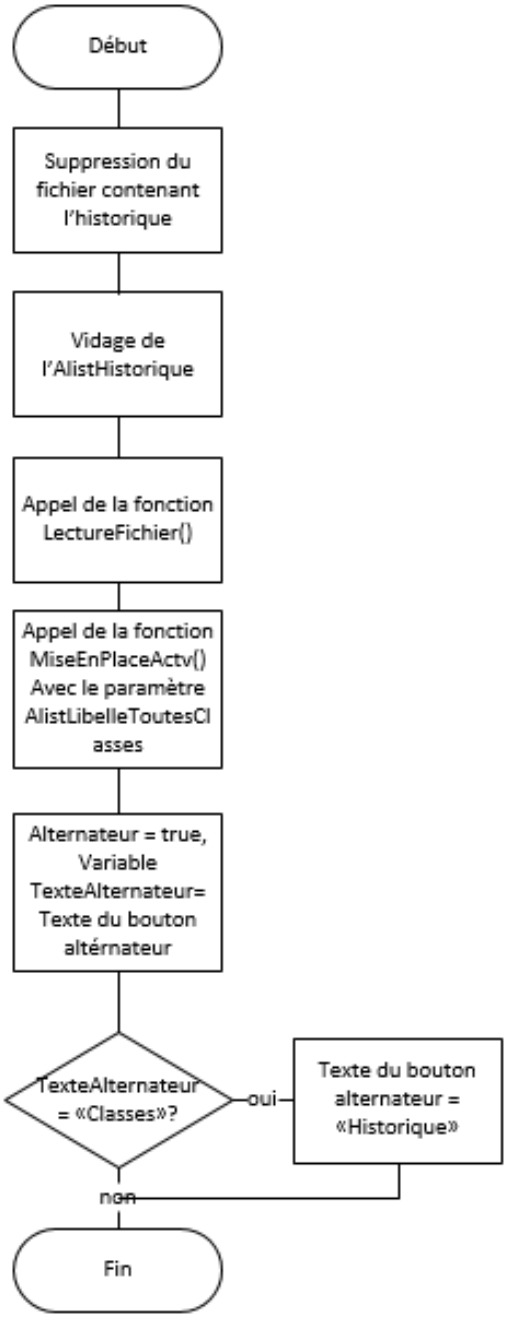
Vue semaine :



3.10.3 Analyse du scénario

3.10.3.1 Algorithme

Vue jour :



Vue semaine :

Même Diagramme que la vue jour.

3.10.3.2 Explications détaillées

Vue jour :

On efface simplement le fichier contenant les classes déjà recherchées. Il faut aussi vider AlistHistorique car il contient également cette liste. On fait appel à la fonction MiseEnPlaceActv() pour changer la source des suggestions. Etant donné que l'historique est dorénavant vide, ça ne sert à rien de continuer à suggérer les classes recherchées. La liste de toutes les classes disponibles est alors suggérée dans le champ classe. Il faut également

changer le texte du bouton pour qu'il affiche « Historique ». La variable bAlternateur passe à true. Ça permet de savoir si l'on est en mode suggestion historique ou liste des classes, et va être utilisée lors de l'appui sur le bouton Historique/Classes.

Vue semaine :

Même explication que la vue jour.

3.10.4 La phase de programmation

Suppression du fichier :

```
File fChemin = getBaseContext().getFilesDir();
File fFichier = new File(fChemin, "storage.txt");
try {
    PrintWriter pw = new PrintWriter(fFichier);
    pw.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

3.11 Cas d'utilisation « Application s'adaptant à la taille de l'écran »

L'application doit être adaptée pour toutes tailles de terminaux. Les boutons, les champs et surtout l'affichage de l'horaire doivent s'adapter à la taille pour être agréable à utiliser pour l'utilisateur.

3.11.1 Scénario

Vue jour :

1. L'utilisateur ouvre l'application.
2. Le système adapte les tailles des boutons, champs et de l'affichage des données.

Vue Semaine :

Même scénario que la vue jour.

3.11.2 Maquettes

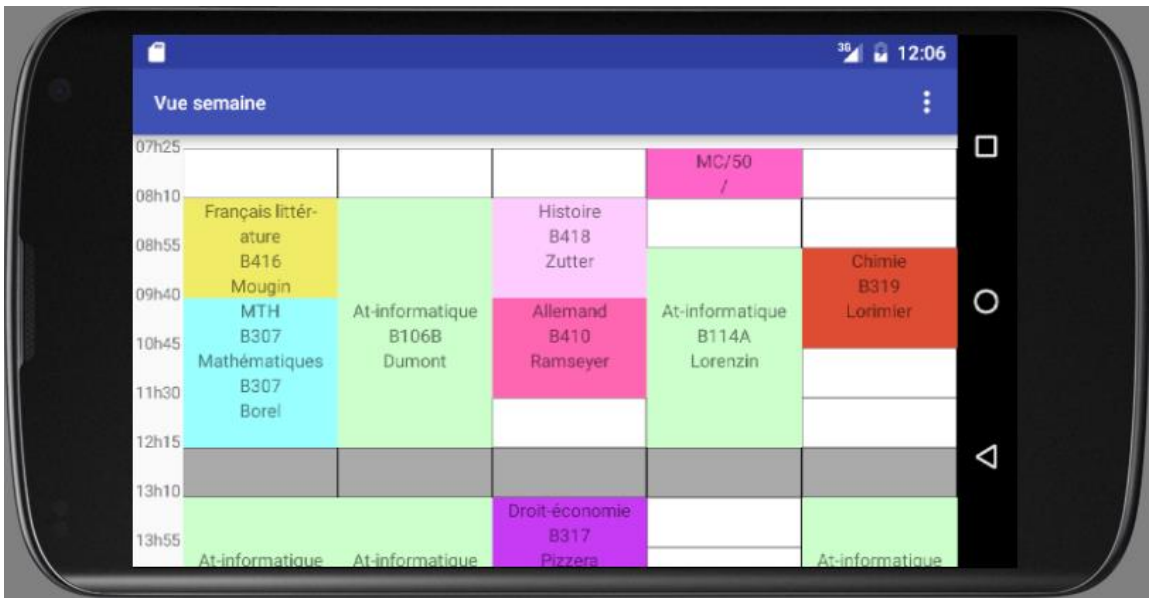
Vue jour :

Affichage Nexus_5X :


12:03
Vue jour
3m3i2
Classes
05-05-2017
Calendrier
Rechercher
--
-
+
++
Vendredi
08h55 Chimie B319
10h45 Lorimier
13h10 AT-Informatique B106B
16h25 Sacchetti


12:03
Vue semaine
07h25
08h10
08h55
09h40
10h45
11h30
12h15
13h10
13h55
14h55
Français littér-ature B416 Mougin
MTH B307 Mathématiques B307 Borel
At-informatique B106B Dumont
Histoire B418 Zutter
Allemand B410 Ramseyer
MC/50 /
At-informatique B114A Lorenzin
Chimie B319 Lorimier
Droit-économie B317 Pizzera
At-informatique B106B
At-informatique B106B
At-informatique B106B

Affichage Nexus 4 :



3.11.3 Analyse du scénario

3.11.3.1 Explications détaillées

Pour pouvoir adapter l'application aux différentes tailles, j'ai tout d'abord utilisé les unités DP. Ça permet d'avoir une taille de police proportionnelle à l'écran. Dans mes fichiers xml, mes views (boutons, textview…) n'ont pas une taille entrée en dur. A la place, j'attribue leurs tailles par rapport à l'espace disponible sur l'écran. Par exemple pour mettre en place les quatre boutons pour modifier la date, la largeur de l'écran est divisée en quatre et chaque bouton prend sa part. Ça permet d'adapter la taille des views à toute taille d'écran.

3.11.4 La phase de programmation

Exemple pour les 4 boutons :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="12">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/BtnMoinsSemaine"
        android:layout_weight="3"
        android:text="@string/dateMoinsMoins"
        android:textSize="22dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/BtnMoinsJour"
        android:layout_weight="3"
        android:text="@string/dateMoins"
        android:textSize="22dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/BtnPlusJour"
        android:layout_weight="3"
        android:text="@string/datePlus"
        android:textSize="22dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/BtnPlusSemaine"
        android:layout_weight="3"
        android:text="@string/datePlusPlus"
        android:textSize="22dp"/>
</LinearLayout>
```

3.12 Cas d'utilisation « Gestion de la rotation de l'écran »

Lorsque l'utilisateur décide de changer l'orientation, l'affichage des données doit s'adapter pour rester agréable à lire. Ce problème ne se pose pas dans la vue semaine car l'orientation est automatiquement bloquée en paysage. En revanche, dans la vue jour, l'utilisateur peut choisir le mode qu'il veut.

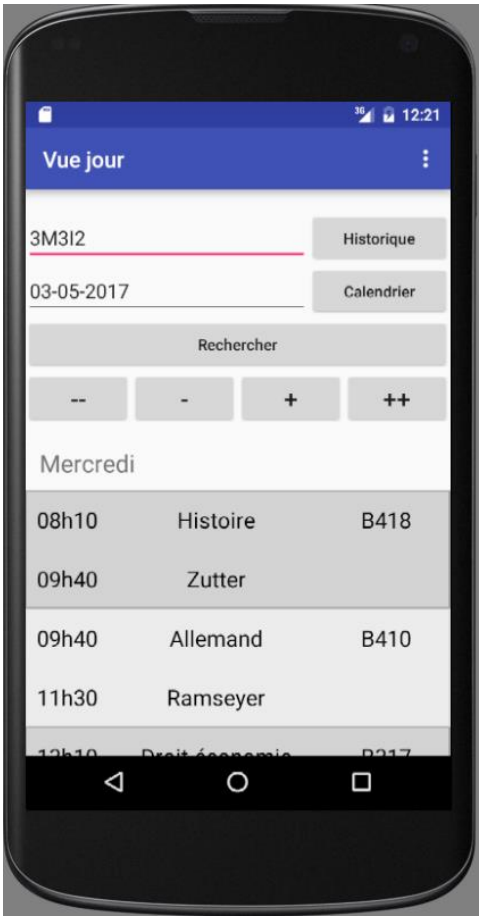
3.12.1 Scénario

Vue jour :

1. L'utilisateur passe de l'orientation portrait à l'orientation paysage.
2. Le système adapte l'affichage des données.

3.12.2 Maquettes

Vue jour :



3.12.3 Analyse du scénario

3.12.3.1 Explications détaillées

Lorsque j'ai commencé à effectuer des tests, j'ai remarqué que l'affichage s'adaptait lors de la rotation de l'écran. Le problème était que l'activité redémarrait à chaque changement d'orientation. C'était désagréable car la dernière classe recherchée et la date du jour actuelle se remettaient dans les deux champs. Après de multiples recherches, j'ai ajouté une ligne dans le fichier AndroidManifest.xml. Cela permettait de ne pas relancer l'activité. En contrepartie, un autre bug a été créé. L'affichage des données n'était plus correct.

Exemple :

C'est en fait normal car la taille des textview sont créée à partir de la largeur de l'écran. Et vu que l'activité n'est pas relancée, la taille reste la même. Si on appuie sur le bouton Rechercher, le bug disparait. J'ai essayé alors de détecter la rotation de l'écran, et ainsi changer la taille des textviews. Après de multiples essais et de nombreux crashs, je n'ai malheureusement pas réussi à mettre ça en place. Le temps passait et j'ai alors dû laisser cette fonctionnalité de côté.
Dans l'état actuel des choses, à chaque changement d'orientation, l'activité se relance. Vu que la dernière classe recherchée est mise dans le champ classe, ça ne pose pas de problème. En revanche le champ date reprend la valeur du jour actuel au lieu de garder la date précédente.

3.12.4 La phase de programmation

Ligne de code essayée pour ne pas relancer l'activité

```
<activity android:name=".MainActivity"
android:configChanges="orientation|keyboardHidden|screenSize"
android:label="@string/TitrevueJour">
```

3.13 Cas d'utilisation « Stocker les données reçues dans une base de données »

A chaque requête effectuée, l'horaire de la classe pour la date choisie doit être enregistré dans une base de données. Ceci permettra d'afficher l'horaire de la classe si l'utilisateur n'a pas de connexion internet. Par manque de temps je n'ai pas pu implémenter cette fonctionnalité.

3.13.1 Scénario

Vue jour :

1. L'utilisateur ouvre l'application.
2. L'utilisateur appuie sur le bouton Rechercher.
3. Le système enregistre les données reçues dans la base de données.

Lorsque pas de connexion :

1. L'utilisateur ouvre l'application.
2. Le système informe l'utilisateur qu'il n'a pas de connexion internet.
3. Le système va chercher si la combinaison classe/date existe dans la base de données.
4. Le système affiche l'horaire si celui-ci existe.

Vue semaine :

Même scénario que dans la vue jour.

3.13.2 Maquette

Pas de maquette pour ce cas d'utilisation.

3.13.3 Analyse du scénario

3.13.3.1 Explications détaillées

Voici comment je pourrais mettre en place ceci :

Une fois la recherche et l'affichage arrivée à terme, une fonction va être appelée. Elle permettra de mettre les données de l'horaire de la classe à la date choisie, dans la base de données. La combinaison classe/date sera utilisée pour identifier chaque recherche. Si des données existent déjà pour cette combinaison, alors elles seront mises à jour.

Lorsque l'utilisateur va ouvrir (ou lancer une recherche) l'application sans avoir de connexion internet, une fonction va chercher dans la base de données, si la combinaison classe/date existe. Si c'est le cas, alors les données vont être affichées. Autrement un message d'erreur va informer l'utilisateur qu'il n'est pas connecté, et que les données ne sont pas présentes dans la base.

La vue semaine se déroulera de la même façon, mais il faudra enregistrer dans la base les données des 5 jours reçues.

4 Points à améliorer

Une seule activité pour les deux vues :

Actuellement, mon programme a deux activités différentes. La vue jour et la vue semaine. Cela crée des problèmes de redondance. De nombreuses fonctions sont similaires et sont utilisées dans les deux vues. Je pourrais supprimer la deuxième activité et tout réunir dans le premier. Dans ce cas-là, l'appui sur « Vue Semaine » va uniquement changer l'affichage mais pas l'activité. Malheureusement, je suis parti trop rapidement sur l'idée d'avoir ces 2 vues séparées. Je pensais que de tout faire en une seule activité allait tout surcharger. Néanmoins, je vois très bien comment je pourrais mettre en place cela. Je devrais créer une variable booléenne (Ex : bAffichageJour). Cette variable sera true lorsque l'utilisateur est en mode vue jour, et en false quand il est en vue semaine. Elle va permettre de savoir combien de requêtes il faudra lancer (3 ou 7), d'aiguiller les données reçues dans le RetourOutput, et d'afficher l'horaire du jour ou de la semaine. Etant donné que la plupart des boutons ont les mêmes fonctions dans les deux activités, il n'y aurait pas de problème à les mettre en place. Il faudrait juste bloquer ou rendre inactif les boutons « + », et « - » dans la vue semaine. Le balayage de l'écran va également être influencé par cette variable. Si elle est à true, alors 1 jour sera augmenté ou diminué à chaque balayage. Autrement une semaine sera modifiée.

Traitement des heures de début non conventionnelles :

Dans l'état actuel du programme, les périodes commençant entre les cases du tableau sont affichées à moitié. Si la branche dure 1 période, alors il n'y a pas de problème. En revanche, lorsqu'il y a plusieurs périodes, je n'arrive pas à savoir le nombre. Un moyen de résoudre le problème serait de calculer le nombre de minutes entre l'heure début et l'heure fin, et en déduire le nombre de périodes. Il faudra également prendre en compte les pauses ce qui deviendrait assez compliqué.

Bordure dans le tableau :

Pour améliorer l'esthétique de la vue semaine, il faudrait ajouter des bordures entre les branches. J'ai laissé les améliorations esthétiques pour la fin du projet, mais malheureusement, je n'ai pas eu le temps de m'en charger.

Gestion de la rotation de l'écran :

Dans l'état actuel de l'application, à chaque changement d'orientation, l'activité est relancée. Vu que le champ classe est rempli par la dernière classe recherchée, elle reste la même que celle d'avant l'orientation. En revanche, le champ date prend la valeur de la date actuelle. Ça pose problème car si l'utilisateur visualisait l'horaire pour une date qui n'est pas la date actuelle, il devra relancer une requête avec la bonne date.

Centrer la salle dans la vue jour :

Pour afficher les données dans la vue jour, j'ai créé une grille 2x3. J'ai laissé une dernière case libre pour ajouter, au besoin, une information en plus. Actuellement 5 informations sont affichées (Heure début, Heure fin, Salle, Professeur, Libellé). Je n'avais finalement pas besoin

de cette dernière case. Pour centrer la salle, je devrais alors ne plus avoir 2 textviews dans la dernière colonne, mais plus qu'un seul. De cette façon la salle serait centrée. Comme je l'ai dit avant, j'ai laissé l'esthétique pour la fin du projet.

Tableau créé dynamiquement dans la vue semaine :

J'avais décidé de créer toutes les cases du tableau en dur dans le xml. Ce n'est pas une méthode optimisée mais au moins, je pouvais personnaliser exactement comme je voulais les cases. J'ai eu des difficultés à créer le rendu voulu dans la vue jour, et je ne savais pas si j'étais capable de refaire de cette manière pour la vue semaine. Je n'ai pas pris de risque et j'ai alors fait la méthode moins propre.

C'est certainement la plus grosse amélioration que je peux faire pour simplifier le xml.

Méthode de récupération de toutes les classes :

Cette requête envoie toutes les ressources (salles, professeurs, classes…) de Hypercool :
http://devinter.cpln.ch/pdf/hypercool/controler.php?action=ressource&nom=

Pour remplir mon arraylist avec le nom de toutes les classes disponibles, je devais prendre les id des classes uniquement. Je savais juste que les ressources de type classe sont avant toutes les autres. Je n'ai pas trouvé d'autres moyens que de compter le nombre de classes manuellement, équivalent à 329. Une boucle for fait alors 329 tours pour prendre le nom des 329 premiers id. Si un jour, le nombre de classes change, avec cette technique, je ne peux pas garantir d'avoir toutes les classes dans mon tableau.

L'idéal serait d'avoir un lien qui renvoie uniquement les id des classes, malheureusement il n'existe pas.

Temps de chargement entre les activités :

Lors du passage de la vue jour à la vue semaine, l'application se fige durant 3-4 secondes. Ceci est dû au changement d'orientation de la vue semaine. Si l'utilisateur est déjà en mode paysage dans la vue jour, et qu'il passe à la vue semaine, alors il n'y a pas de freeze. Peut-être que le problème est dû à l'émulateur. J'ai testé sur mon propre appareil, et le freeze est présent, mais dure deux fois moins longtemps.

5 Mode d'emploi utilisateur

5.1 Guide d'installation :

En annexe

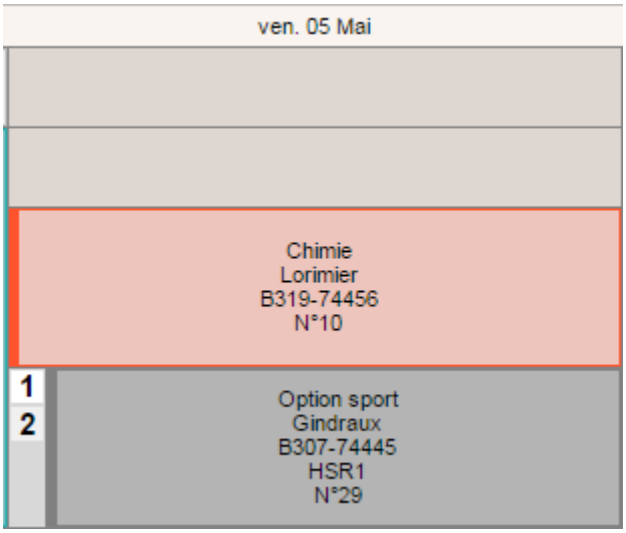
5.2 Guide d'utilisation :

En annexe

6 Problèmes rencontrés et solutions

- Branches pas affichées :
  Lorsqu'il y a plusieurs branches au même moment (souvent dû aux options), impossibilité d'afficher ces branches.

  Le site du Cpln propose 2 boutons pour choisir quelle branche on aimerait afficher.



  J'aurais pu essayer de reproduire cette méthode mais j'ai affaire à un gros problème. En effet, dans les données JSON reçues, les branches qui sont au même moment ne sont pas renvoyées. Il m'est donc impossible de savoir s'il y a cours à ces périodes-là.

  Solution : La seule solution serait de modifier le site Hypercool pour qu'il renvoie les branches ayant la même heure de début. Dans ce cas-là, je pourrais agir.

- Crash lorsque branche « maitrise de classe » :
  Mon programme crashait quand la branche « maitrise de classe » devait être affichée. Cette branche, contrairement aux autres, ne disposait pas de salle. Dans ma fonction, j'essayais de manipuler des données vides, ce qui entrainait un crash.

  Solution : Ajout de try/catch et si la salle est vide alors j'attribue « / » à la valeur de la salle.

- Conflit avec les Id :
  Lors de la création du xml de la seconde activité, j'ai bêtement copié/collé les éléments. Je me suis rendu compte que ça créait des conflits dans le code.

  Solution : Changer les id pour qu'ils soient uniques.

- Icone de chargement après le freeze :
  Dans la vue semaine, pendant que l'application lançait les requêtes, l'écran se bloquait (4-5 secondes) comme si l'application avait crash. Pour éviter cet effet, j'ai mis en place une icône de chargement. En testant, j'ai vu que l'icône s'affichait uniquement après que les requêtes soient terminées. L'effet freeze était toujours présent.

  Solution : Ne plus utiliser .get() pour faire passer le résultat de l'AsyncAffichageHoraire à l'activité vue semaine. A cause du get(), la tâche n'est plus asynchrone.

- Freeze de l'application :
  J'ai appris plus tard que c'était à cause du .get() lors de l'appel de la fonction AsyncAffichageHoraire. Ce .get permettait de prendre la valeur de retour de la requête, mais la tâche n'était plus asynchrone, ce qui provoquait ce blocage. Après plusieurs recherches sur internet, j'ai trouvé une manière pour faire passer les valeurs reçues de l'AsyncAffichageHoraire à l'activité principale.

  Solution : Création d'une interface contenant la fonction qui va recevoir le résultat, et modification du OnPostExecute() de la class AsyncAffichageHoraire.

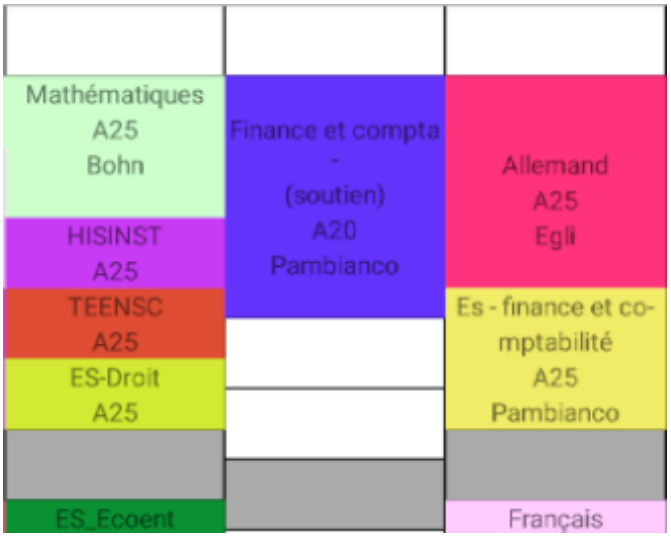
- Affichage de l'horaire après 2 clics :
  Après avoir mis en place mes requêtes de manière asynchrone, j'ai repéré un petit problème assez dérangeant. Il fallait cliquer 2 fois sur le bouton Rechercher pour que l'affichage ait lieu. J'ai perdu quelques périodes avant de trouver la solution. Avant, je lançais les 3 requêtes en même temps. La requête nous renvoyant les données de l'horaire d'une classe avait besoin de l'id de la classe pour fonctionner. Et vu que les requêtes étaient envoyées en même temps, l'id était par conséquent null.

  Solution : Lancer la requête (qui va renvoyer l'horaire de la classe) après la fin de la première requête (celle qui va obtenir l'id de la classe).

- Heure de début non conventionnelles :
  Certaines périodes commencent à par exemple 08h30 au lieu de 08h10. La méthode que j'ai utilisée ne fonctionne donc pas avec ces branches. J'ai alors dû créer une nouvelle fonction pour au moins afficher la branche, mais je ne peux toujours pas savoir le nombre de périodes.

  Solution : Création de RemplissageCases2()

- Libelles trop long dans la vue semaine :
  Quand un libellé d'une branche était trop long, le textview passait sur 2 lignes, ce qui entrainait un décalage de toutes les cases du tableau.



  Solution : Pour éviter cet effet, j'ai décidé d'utiliser le code matière au lieu du libelle lorsque la branche durait une seule période. Si par contre, elle durait plus d'une période, alors j'ai découpé le libelle en 2 parties pour être affiché dans 2 textviews différents. La taille des caractères a légèrement été diminuée et les lettres en majuscules après la première lettre sont passées en minuscules. Avec ces méthodes, le libelle ne passe plus sur 2 lignes et ne décale plus le tableau.

- Classes avec espaces :
  En effectuant des tests, je me suis rapidement rendu compte que les classes contenant un espace ne renvoyaient aucunes données. J'ai donc essayé d'accéder à l'horaire de ces classes directement depuis un navigateur internet. En recherchant l'id d'une classe avec espace, j'ai vu que celui-ci était remplacé par « %20 ».

  Solution : Tester le nom de la classe présent dans le champ classe, avant de le mettre dans une variable. S'il contient un espace, le remplacer par « %20 », sinon ne rien faire.

- Nom d'une classe vide/erroné dans l'historique :
  Quand l'utilisateur recherchait avec une classe erronée ou vide, le nom de cette classe était tout de même écrit dans le fichier historique. Le problème était que lors de l'appui sur le bouton Rechercher, j'écrivais automatiquement dans le fichier sans faire de vérification.

  Solution : Les fonctions EcritureFichier(), LectureFichier(), et MiseEnPlaceActv() se lancent désormais après la dernière requête. Lors du lancement de la requête, toutes les vérifications ont déjà été effectuées, ce qui nous garantit que la classe est juste et non vide.

- Le changement d'orientation redémarre l'activité :
  Lorsque l'on basculait d'orientation (portrait/paysage), l'activité se relançait. J'ai ajouté une ligne dans le fichier AndroidManifest.xml pour éviter cet effet. Malheureusement, l'affichage était alors bogué. J'ai alors essayé de détecter l'orientation et d'adapter la taille des textviews contenant les données, mais je n'ai pas réussi.

  Solution : Une méthode pourrait fonctionner : lorsque l'utilisateur change d'orientation, le détecter et ainsi adapter la taille des textviews (contenant les données) à la nouvelle taille d'écran.

- Boutons avancer/reculer date mettant n'importe quel nom dans l'historique :
  A l'appui des boutons pour changer la date d'une semaine/jour, le contenu du champ classe était écrit dans le fichier historique. Ça posait des problèmes car si l'utilisateur lançait une recherche, puis modifiait le contenu du champ classe (sans appuyer sur le bouton rechercher) et appuyait sur les boutons +. -, alors le contenu (pouvant être vide ou erroné) était stocké dans l'historique.

  Solution : Si la requête est effectuée depuis les boutons permettant de changer la date, je n'enregistre plus le contenu du champ dans le fichier. Le nom de la classe a déjà été stocké lors de l'appui sur le bouton Rechercher.

- Impossibilité de manipuler les données reçues avec des tableaux :
  Une fois les données de l'horaire reçues, il fallait les stocker dans des variables. J'ai tout d'abord essayé avec des tableaux. J'arrivais à stocker les données dans les tableaux, mais il m'était impossible de trier chronologiquement le contenu. Je ne sais toujours pas si c'est le programme qui n'autorise pas une telle opération ou bien une erreur de ma part. J'ai essayé la même opération avec des arraylists, et cette fois-ci tout marchait parfaitement.

  Solution : Stocker les données reçues dans des arraylists plutôt que dans des tableaux.

- Ecriture des noms à la suite dans le fichier historique :
  A la première version de l'historique, j'écrivais simplement les noms des classes dans le fichier. Je me suis rendu compte que de base, toutes les classes étaient écrites les unes à la suite des autres. Je devais alors, écrire le nom des classes ligne par ligne pour pouvoir les lire plus facilement.

  Solution : Changement de la fonction pour que l'écriture se fasse ligne par ligne.

- Crash lorsque la salle ne contenait pas « - » :
  Durant mes tests, certaines classes provoquaient un crash de l'application. Le point commun de ses classes étaient qu'elles disposaient d'une branche qui avait un nom de salle sans tiret (par exemple : « Maladière 4 »). Pour simplifier l'affichage des salles, une fonction supprime tout ce qui est après un tiret. (B106B-74487 devient B106B). Maladière 4 ne contenant pas de tiret, arrêtait la fonction mettant les données reçues

dans des arraylists. Les tableaux n'étaient donc pas totalement remplis ce qui provoquait ce crash lors de l'affichage.

Solution : Ajouter un try/catch et si la salle ne contient pas de tiret, alors ne rien faire.

- Réception de plusieurs salles pour une branche :
  En vérifiant les données JSON reçues, je me suis aperçu que certaines branches avaient plusieurs salles.


```
{
  "codeMatiere": "ALL",
  "libelle": "Allemand",
  "indice": 2.0833333333333,
  "heureDebut": "09h40",
  "heureFin": "11h30",
  "classes": [
    "3M3M",
    "3M3A",
    "3M3E",
    "3M3I2"
  ],
  "professeur": [
    "Ramseyer Manna G\u00e9raldine [GR]"
  ],
  "salle": [
    "B410-74466",
    "B408-74464 labolangues",
    "AB3-74419",
    "C6"
  ]
},
```


  Je ne savais donc pas quelle salle était la bonne. Au début j'ai cru que c'était par ordre chronologique, les salles de début d'année d'abord puis les salles actuelles. En vérifiant avec plusieurs classes, j'ai vu que ce n'était pas le cas. Après discussion avec le professeur, il m'a laissé le choix. Soit j'affiche toutes les salles reçues à la suite (ex : B410, B408, AB3, C6) ou je prends juste la première. J'ai décidé de choisir la deuxième option car sinon ça risquait de créer une chaîne de caractère trop long à afficher.

  Solution ; Dans l'état actuel, je ne peux pas choisir la bonne salle. Une solution serait de modifier Hypercool pour qu'il envoie uniquement la salle actuelle.

- AsyncAffichageHoraire crash à la deuxième requête.
  En exécutant mes requêtes cette fois-ci, de manière asynchrone, j'ai remarqué que l'application crashait lors de l'appui sur le bouton Rechercher. C'était bizarre car la requête dans le OnCreate() fonctionnait correctement. Après plusieurs recherches, j'ai trouvé qu'il n'était pas possible d'utiliser la même instance de l'AsyncAffichageHoraire plusieurs fois.

  Solution : Créer une nouvelle instance de l'AsyncAffichageHoraire avant de l'exécuter.

- Liste déroulante pour l'historique :
  J'avais initialement imaginé l'accès à l'historique de cette façon : l'utilisateur appuie sur le bouton historique, un spinner (liste déroulante) apparait, il choisit la classe, que je mets dans le champ classe. J'ai réussi à créer sans problème le spinner en ajoutant les données contenues dans le fichier historique. Les problèmes arrivaient lorsque je devais mettre à jour la liste. Quand l'utilisateur recherchait une classe, le nom était enregistré et venait s'ajouter dans la liste déroulante. S'il voulait choisir le nom de la classe via cette liste, il devait effectuer deux appuis pour que la classe choisie se mette dans le champ classe. Après avoir cherché le problème durant plusieurs périodes, j'ai décidé d'abandonner l'idée de la liste déroulante. La nouvelle méthode était plus simple à mettre en place. J'utilise dorénavant l'autoCompleteTextView du champ classe pour suggérer la liste des classes déjà recherchées. Lorsque l'utilisateur appuie sur le bouton Historique, la source des suggestions du champ classe change. Elle affiche désormais l'historique. Un deuxième appui sur le bouton historique (devenu « Classes ») affiche à nouveau la liste de toutes les classes disponibles.

  Solution : Changer de méthode pour afficher l'historique.

- Crash lors du test avec mon appareil mobile personnel :
  Durant les dernières périodes, j'ai dû créer le .apk. Après avoir fait ça, j'ai installé l'application sur mon propre appareil pour effectuer quelques tests. L'affichage est exactement pareil et l'application se comporte comme prévue. Malheureusement, l'application crashait parfois aléatoirement. Je ne sais pas du tout l'origine des crashs étant donné que tout fonctionne parfaitement avec l'émulateur. Je n'ai pas eu le temps de régler ce souci.

  Solution :

  Lorsque plusieurs classes sont retournées, ne pas autoriser d'entrer une classe renvoyant plusieurs classes :
  Actuellement quand l'utilisateur recherche avec une classe par exemple « 3m », le champ classe suggère la liste des classes retournées. Normalement, si l'utilisateur choisit une classe parmi la liste, il n'y a pas de problème. En revanche, s'il décide de tout de même appuyer sur le bouton rechercher sans avoir choisi dans la liste, la requête est lancée avec comme classe « 3m ». Dans ce cas, l'horaire du premier id reçu pour la classe 3m est affiché. Ça pose également problème car « 3m » est enregistré dans l'historique alors que ce n'est pas une classe. Pour régler ce problème, il faudrait vérifier à nouveau si la classe existe ou pas, pour que l'utilisateur ne puisse pas lancer une requête avec une classe inexistante.

  Solution : Vérifier l'existence de la classe et ne pas lancer la requête si elle n'existe pas.

7 Conclusions

Le déroulement du projet s'est bien passé. Les fonctionnalités principales ont été réalisées avec succès. Par manque de temps, je n'ai pas pu implémenter certains objectifs secondaires, mais j'ai pu proposer des solutions pour les réaliser. Durant le développement, le professeur m'a fait part de plusieurs idées que j'ai su efficacement intégrer dans mon programme.

Un point très important de l'application était d'avoir une bonne lisibilité de l'horaire. Actuellement l'affichage de l'horaire sur le site du Cpln n'est pas adapté aux appareils mobiles. Je pense avoir accompli cet objectif car l'affichage est agréable dans les deux vues. Les messages d'erreurs sont clairs et l'application est simple d'utilisation. Le design général peut néanmoins être amélioré.

Des grosses améliorations peuvent être effectuées au niveau de la propreté du code. Tout d'abord, utiliser une seule activité pour afficher les deux vues. Ça permettra premièrement d'alléger l'application (pas de doublons) mais aussi d'augmenter la vitesse de transition entre les deux vues. La deuxième amélioration serait de créer le tableau (dans la vue semaine) contenant les cases dynamiquement.

J'ai beaucoup appris grâce à ce projet. Je n'avais jamais développé une application aussi conséquente et j'ai pu tester mes capacités. J'ai mobilisé toutes les connaissances acquises durant ces 3 années de formation pour essayer de produire une application de qualité. Des erreurs de programmation ont été commises mais je tacherais de ne plus les reproduire dans mes futures applications.

D'un point de vue plus personnel, je me suis senti très autonome en classe. Les questions posées au professeur étaient toutes utiles et mon comportement a été correct. Les deux dernières semaines étaient par contre très éprouvantes, mais je suis fier de présenter mon projet.

8 Annexes

8.1 Journal de travail

8.2 Cahier des charges

8.3 Code source

8.4 Planning

8.5 Protocole de test

8.6 Guide d'installation

8.7 Guide d'utilisation

Remarque : Dans le cahier des charges, une « Documentation de développement permettant d'effectuer la maintenance du projet » m'est demandée. Pour mon projet, je ne considère pas ce dossier nécessaire. Il n'y a pas par exemple d'interface ou tout autre moyen de gérer l'application en tant qu'administrateur. Si par contre, le fonctionnement d'Hypercool est modifié, alors il faudrait revoir entièrement la partie programmation.

9 Références

J'ai pu réutiliser des parties de code de mes anciens projets, notamment pour la class AsyncAffichageHoraire.

Voici une liste de tous les liens des sites visités m'ayant été utiles.

JsonViewer :

http://jsonviewer.stack.hu/

Horaire :

https://horaires-cpln.s2.rpn.ch/etudiant
http://devinter.cpln.ch/pdf/hypercool/

Changement d'orientation :

http://stackoverflow.com/questions/5726657/how-to-detect-orientation-change-in-layout-in-android
http://stackoverflow.com/questions/4075540/android-application-restarts-on-orientation-change
http://stackoverflow.com/questions/456211/activity-restart-on-rotation-android

Affichage des données :

https://developer.android.com/reference/android/widget/GridView.html
http://www.mkyong.com/android/android-gridview-example/
http://stackoverflow.com/questions/26773113/android-table-like-user-interface
http://stackoverflow.com/questions/16271343/empty-gridlines-in-android-gridview

http://stackoverflow.com/questions/11307218/gridview-vs-gridlayout-in-android-apps
http://www.techotopia.com/index.php/Working_with_the_Android_GridLayout_in_XML_Layout_Resources
https://inducesmile.com/android/android-gridview-vs-gridlayout-example-tutorial/
http://androidexample.com/Table_Layout_-_Android_Example/index.php?view=article_discription&aid=74
https://developer.android.com/reference/android/view/View.html
http://stackoverflow.com/questions/6526874/call-removeview-on-the-childs-parent-first
http://stackoverflow.com/questions/7226168/android-tablelayout-programatically
http://stackoverflow.com/questions/4203506/how-can-i-add-a-textview-to-a-linearlayout-dynamically-in-android
http://stackoverflow.com/questions/10094392/displaying-multiple-data-in-a-list-view
http://www.vogella.com/tutorials/AndroidListView/article.html
http://techlovejump.com/android-multicolumn-listview/
http://stackoverflow.com/questions/43007932/creating-dynamic-linearlayout-with-textviews-on-it
http://stackoverflow.com/questions/19268926/clear-view-in-android
http://stackoverflow.com/questions/16884524/programmatically-add-border-to-linearlayout
http://stackoverflow.com/questions/3805599/add-delete-view-from-layout
http://stackoverflow.com/questions/6798867/android-how-to-programmatically-set-the-size-of-a-layout

Asynctask :

http://stackoverflow.com/questions/6373826/execute-asynctask-several-times
http://stackoverflow.com/questions/11752961/how-to-show-a-progress-spinner-in-android-when-doinbackground-is-being-execut
http://stackoverflow.com/questions/12300796/android-how-to-disable-controls-during-progress-bar-is-active
http://stackoverflow.com/questions/17549042/android-asynctask-passing-a-single-string
http://stackoverflow.com/questions/29782517/how-to-disable-button-while-asynctask-is-running-android
http://stackoverflow.com/questions/19624193/how-to-handle-return-value-from-asynctask
http://stackoverflow.com/questions/12575068/how-to-get-the-result-of-onpostexecute-to-main-activity-because-asynctask-is-a
http://stackoverflow.com/questions/6122812/using-wait-in-asynctask
http://stackoverflow.com/questions/29768116/how-to-make-a-dynamic-execute-of-an-asynctask
http://stackoverflow.com/questions/9458258/return-a-value-from-asynctask-in-android
http://stackoverflow.com/questions/15763464/asynctasks-are-too-slow-for-several-simultaneous-networking-operations

Historique :

http://stackoverflow.com/questions/4838992/android-open-spinner-from-button
http://stackoverflow.com/questions/2784081/android-create-spinner-programmatically-from-array
http://stackoverflow.com/questions/1947933/how-to-get-spinner-value
https://stackoverflow.com/questions/43275857/pop-up-a-dynamic-choice-list-when-pressing-a-button
http://stackoverflow.com/questions/14376807/how-to-read-write-string-from-a-file-in-android
http://stackoverflow.com/questions/4614227/how-to-add-a-new-line-of-text-to-an-existing-file-in-java
http://stackoverflow.com/questions/3554722/how-to-delete-internal-storage-file-in-android
https://developer.android.com/training/basics/data-storage/files.html
https://teamtreehouse.com/community/read-lines-of-a-text-file-into-an-array
https://developer.android.com/reference/android/widget/AutoCompleteTextView.html
http://stackoverflow.com/questions/1879700/how-to-remove-duplicate-value-from-arraylist-in-android

http://stackoverflow.com/questions/6994518/how-to-delete-the-content-of-text-file-without-deleting-itself

Traitement date :

http://stackoverflow.com/questions/17808373/popup-datepicker-for-edittext
http://stackoverflow.com/questions/6421874/how-to-get-the-date-from-the-datepicker-widget-in-android
http://stackoverflow.com/questions/9355747/how-can-i-create-a-weekly-calendar-view-for-an-android-honeycomb-application
http://stackoverflow.com/questions/16388268/getting-current-week-days-with-dates
http://stackoverflow.com/questions/7440719/how-can-i-get-all-the-days-of-the-current-week-in-android
http://stackoverflow.com/questions/11791513/converting-string-to-calendar-what-is-the-easiest-way
https://developer.android.com/reference/java/util/Calendar.html
http://stackoverflow.com/questions/428918/how-can-i-increment-a-date-by-one-day-in-java
http://stackoverflow.com/questions/18256521/android-calendar-get-current-day-of-week-as-string

Json :

http://stackoverflow.com/questions/27185629/how-to-parse-json-with-any-key-on-android/27188397#27188397
https://stackoverflow.com/questions/43255577/parse-json-with-unknown-key
http://www.androidhive.info/2012/01/android-json-parsing-tutorial/
http://stackoverflow.com/questions/29551168/how-to-parse-unnamed-json-array-in-android-app
http://stackoverflow.com/questions/9605913/how-to-parse-json-in-android
http://stackoverflow.com/questions/10164741/get-jsonarray-without-array-name
https://www.tutorialspoint.com/android/android_json_parser.htm

Menu :

http://www.javatpoint.com/android-popup-menu-example
http://tutos-android-france.com/menu-ajouter-des-actions-a-lactionbar/

Autres :

http://stackoverflow.com/questions/5985937/use-a-variable-in-a-url-object-in-android
http://stackoverflow.com/questions/6421507/how-to-display-double-quotes-symbol-in-android-text-view
http://stackoverflow.com/questions/2704956/strings-dont-seem-to-be-equal-in-java-on-android-even-though-they-print-the-sa
http://stackoverflow.com/questions/4313457/java-arraylist-index
http://stackoverflow.com/questions/9276493/how-to-delete-all-the-characters-after-one-character-in-the-string

http://stackoverflow.com/questions/12353252/copy-a-string-until-character-found-java
http://stackoverflow.com/questions/14316487/java-getting-a-substring-from-a-string-starting-after-a-particular-character
http://stackoverflow.com/questions/6835980/android-converting-string-to-int

http://stackoverflow.com/questions/31071283/how-to-sort-arraylist-of-string-having-integer-values-in-java
http://stackoverflow.com/questions/36595805/how-to-sort-array-list-integer-type
http://stackoverflow.com/questions/1514910/how-to-properly-compare-two-integers-in-java
http://stackoverflow.com/questions/6674341/how-to-use-scrollview-in-android
http://stackoverflow.com/questions/1016896/get-screen-dimensions-in-pixels
https://developer.android.com/reference/android/util/DisplayMetrics.html
http://stackoverflow.com/questions/16956720/how-to-create-an-2d-arraylist-in-java
http://stackoverflow.com/questions/2150287/force-an-android-activity-to-always-use-landscape-mode
http://stackoverflow.com/questions/35912558/how-to-find-array-index-value-in-android
http://stackoverflow.com/questions/4865244/android-using-findviewbyid-with-a-string-in-a-loop
http://stackoverflow.com/questions/26410811/how-to-make-an-array-of-colors-in-android-studio
http://stackoverflow.com/questions/6943588/can-java-use-string-as-an-index-array-key-ex-arraya-1
http://stackoverflow.com/questions/1109022/close-hide-the-android-soft-keyboard
http://stackoverflow.com/questions/7438612/how-to-remove-the-last-character-from-a-string
http://stackoverflow.com/questions/4139288/android-how-to-handle-right-to-left-swipe-gestures