 Application de gestion de notes scolaire de l’ETML

André Gomes / Grégory Poget – Cin2a

EMTL

100 périodes

Gilbert Gruaz – Laurent Deschamps

Table des matières

[1 Spécifications 3](#_Toc485802899)

[1.1 Titre 3](#_Toc485802900)

[1.2 Description 3](#_Toc485802901)

[1.3 Matériel et logiciels à disposition 3](#_Toc485802902)

[1.4 Prérequis 3](#_Toc485802903)

[1.5 Cahier des charges 3](#_Toc485802904)

[1.5.1 Objectifs et portée du projet (objectifs SMART) 3](#_Toc485802905)

[1.5.2 Fonctionnalités requises (du point de vue de l’utilisateur) 3](#_Toc485802906)

[1.5.3 Contraintes 4](#_Toc485802907)

[1.5.4 Si le temps le permet … 4](#_Toc485802908)

[1.5.5 Méthodes de validation des solutions 4](#_Toc485802909)

[1.6 Les points suivants seront évalués 4](#_Toc485802910)

[1.7 Validation et conditions de réussite 4](#_Toc485802911)

[2 Planification Initiale 4](#_Toc485802912)

[3 Analyse 4](#_Toc485802913)

[3.1 Opportunités 4](#_Toc485802914)

[3.2 Document d’analyse et conception 5](#_Toc485802915)

[3.3 Conception des tests 7](#_Toc485802916)

[3.4 Planification détaillée 7](#_Toc485802917)

[4 Réalisation 7](#_Toc485802918)

[4.1 Dossier de Réalisation 7](#_Toc485802919)

[4.1.1 Base de donnée 7](#_Toc485802920)

[4.1.2 Option 9](#_Toc485802921)

[4.1.3 Login 12](#_Toc485802922)

[4.1.4 Accueil 12](#_Toc485802923)

[4.1.5 Branche 13](#_Toc485802924)

[4.1.6 Note 14](#_Toc485802925)

[4.1.7 Moyenne 18](#_Toc485802926)

[5 Tests 20](#_Toc485802927)

[5.1 Dossier des tests 20](#_Toc485802928)

[6 Conclusion 20](#_Toc485802929)

[6.1 Bilan des fonctionnalités demandées 20](#_Toc485802930)

[6.2 Bilan de la planification 21](#_Toc485802931)

[6.3 Bilan personnel 21](#_Toc485802932)

[7 Divers 21](#_Toc485802933)

[7.1 Journal de travail 21](#_Toc485802934)

[7.2 Webographie 21](#_Toc485802935)

[8 Annexes 21](#_Toc485802936)

# Spécifications

## Titre

Gestionnaire de notes ETML

## Description

Pendant le projet de bureautique de première année, les élèves doivent confectionner un fichier Excel avec un tableau qui permet de regrouper les notes scolaires durant l’apprentissage.

Nous avons décidé de ré-implémenté ce fichier au format Android

## Matériel et logiciels à disposition

* PC Etml.
* Téléphone Android.
* Android studio.
* SQLite.
* Outil de Mokeup

## Prérequis

* Modules de programmation
* Module 214

Module 120

## Cahier des charges

### Objectifs et portée du projet (objectifs SMART)

* Journal de travail.
* Planification et répartition équilibrée des taches.
* Respect des normes ETML.
* Livrer un produit de fonctionnel et de qualité.
* Utilisation du modèle MVC.
* Versioning GitHub.

### Fonctionnalités requises (du point de vue de l’utilisateur)

* Création de la grille de notes
* Mise en forme de base (CFC, Maturité, Formation accélérée)
* Insérer des modules
* Insérer des notes (pondérées)
* Calcule automatique des moyennes
* Affichage des résultats finaux
* Stockage des données en local
* Choix du thème par l’utilisateur (Couleur du fond)
* Initialisation du mot de passe utilisateur

### Contraintes

* Langage Java pas étudié dans les modules
* Prendre en main l’interface Android

### Si le temps le permet …

* Stockage des données en Cloud

### Méthodes de validation des solutions

## Les points suivants seront évalués

* Le rapport
* Les planifications (initiale et détaillée)
* Le journal de travail
* Le code et les commentaires
* Les documentations de mise en œuvre et d’utilisation

## Validation et conditions de réussite

* Compréhension du travail
* Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
* Etat de fonctionnement du produit livré

# Planification Initiale

Le projet commence le Lundi 29 mai 2017 le dernier jour pour le rendre avant les présentations est le Lundi 26 Juin 2017

Les périodes de travails sont réparties comme suis :

* Lundi : 8 périodes
* Mardi : 4 périodes
* Mercredi : 9 périodes
* Jeudi : Théorie
* Vendredi : 9 périodes
* Total : 30 périodes par semaines

Il y a un seul jour de congé pendant ce projet, le lundi de pentecôte le 5 juin 2017

Et une semaine du au camp de sport.

# Analyse

## Opportunités

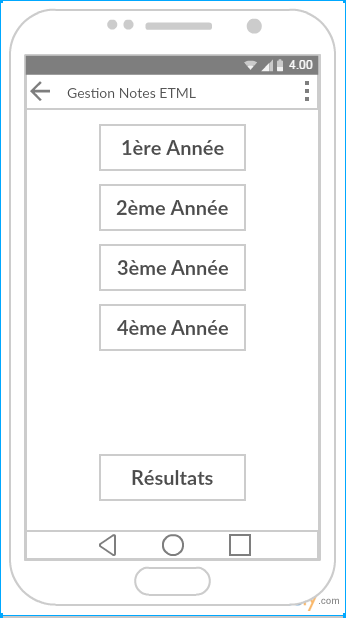
Le projet comporte plein d’opportunité, nous avons la chance de pouvoir découvrir le développement d’application sous Android et un nouveau langage de programmation « Java ». On va aussi pouvoir utiliser comme il le faut GitKraken pour le versionning dans un vrai projet de programmation complet.

Au niveau des difficultés on aura pas mal du fait que nous devons découvrir entièrement le nouveau langage en autonome ce qui risque surement de nous ralentir lors du développement dans des tâches relativement difficiles à mettre en place comme par exemple l’affichage dynamique que nous avons prévus de faire.

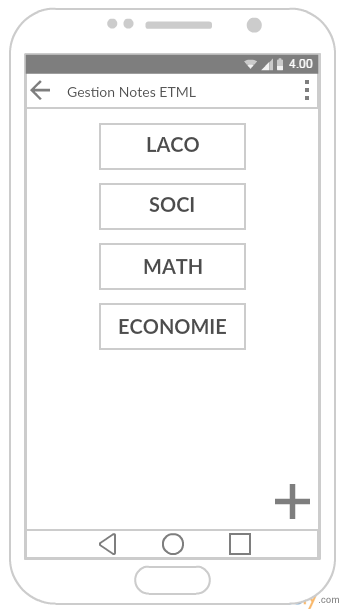
## Document d’analyse et conception

Nous avons décidé de séparer l’application en 4 activités pour pouvoir faire notre application.

Page d’accueil avec les 4 années déjà créé automatiquement avec un bouton qui nous permet d’afficher les résultats finaux de chaque année.



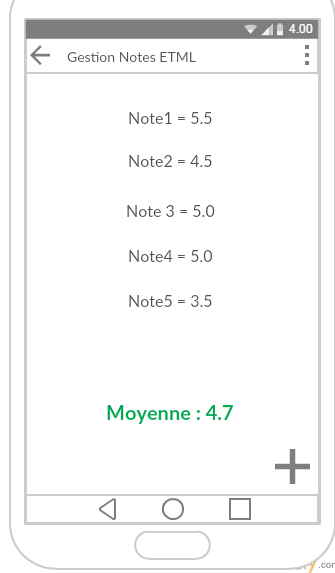
Page qui affiche les branches de chaque année avec dans le menu en bas à droite un petit + qui va permettre d’ajouter des branches.



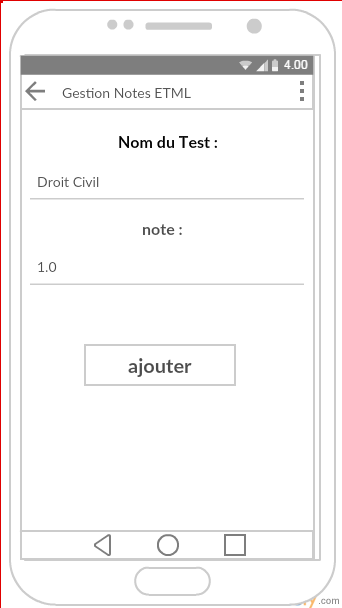
Après avoir cliqué pour ajouter une branche nous aurons un dialogue qui s’ouvre pour nous permettre de créer une branche supplémentaire.



Ensuite nous avons l’affichages des notes de la branche avec la moyenne de la branche tout en bas et dans le menu nous avons la possibilité d’ajouter une note classique et une note pondérée.



Pour l’ajout des notes classiques nous auront un affichage de ce genre-là.



Et voici avec le pourcentage en plus pour les notes pondérées.

Nous pensons aussi créer une activité pour pouvoir modifier certain paramètre dans l’application tel que le thème et le mot de passe de connexion.



## Conception des tests

Pour la conception des tests vu que nous utilisons Android Studio comme programme nous avons un émulateur de téléphone Android intégrée ce qui nous permet dès l’ajout d’une ligne de code de pouvoir test si le programme ne plante pas et que la ligne fait bien ce que l’on demande. Nous allons aussi faire des tests sur nos téléphones respectifs pour voir si l’affichage et les fonctionnalités ne pose pas de problème.

Le tableau de test sera un tableau avec un titre, représentant le plus possible ce qui a été fait pour valider le test. Une colonne test pour savoir si le test est une réussite ou un échec et finalement un commentaire ajoutant une précision sur pour est-ce que le test est réussi ou pas

## Planification détaillée

Fichier en annexes « Modele-JDT.xlsm » sous l’onglet Diagramme.

# Réalisation

## Dossier de Réalisation

### Base de donnée

Pour la nécessité du projet nous avons dû créé une base de donnée comportant deux tables. Une pour sauvegarder les informations des boutons créés et une autre pour garder en mémoire les notes de l’utilisateur qu’il rentrera.

Code pour la table Branch :

private String createBranchTable = "CREATE TABLE " + *TABLE\_BRANCH* + "("  
 + *KEY\_ID\_BRANCH* + " INTEGER PRIMARY KEY," + *KEY\_TEXT\_BRANCH* + " TEXT,"  
 + *KEY\_YEAR\_BRANCH*+ " TEXT" + ")";

Code pour la table Note :

private String createMarkTable = "CREATE TABLE " + *TABLE\_MARK* + "("  
 + *KEY\_ID\_MARK* + " INTEGER PRIMARY KEY," + *KEY\_NAME\_NOTE* + " TEXT,"  
 + *KEY\_NOTE\_NOTE* + " TEXT," + *KEY\_YEAR\_NOTE* + " TEXT," + *KEY\_IDBRANCH\_NOTE* + " TEXT" + ")";

Pour pouvoir travailler on a créé plusieurs méthodes pour pouvoir ajouter supprimer et modifier les données ajouter dans la base.

Cette méthode est utile pour pouvoir ajouter les notes dans la bdd

void addMark(MarkClass markClass){  
 SQLiteDatabase db = this.getWritableDatabase();  
  
 ContentValues values = new ContentValues();  
 values.put(*KEY\_NAME\_NOTE*, markClass.getMarName());  
 values.put(*KEY\_NOTE\_NOTE*, markClass.getMarNote());  
 values.put(*KEY\_YEAR\_NOTE*, markClass.getMarYear());  
 values.put(*KEY\_ID\_BRANCH*, markClass.getIdBranch());  
  
 db.insert(*TABLE\_MARK*, null, values);  
 db.close();  
}

En premier on ouvre la base en mode écriture puis on va récupérer les valeurs entrées dans le constructeur de la classe MarkClass pour pouvoir les entrées dans la variable « values » qui va stocker les informations grâce à la méthode insert puis en referme la base.

void addBranch(BranchClass branchClass){  
 SQLiteDatabase db = this.getWritableDatabase();  
  
 ContentValues values = new ContentValues();  
 values.put(*KEY\_ID\_BRANCH*, branchClass.getIdBranch());  
 values.put(*KEY\_TEXT\_BRANCH*, branchClass.getBraText());  
 values.put(*KEY\_YEAR\_BRANCH*, branchClass.getBraYear());  
  
 db.insert(*TABLE\_BRANCH*, null, values);  
 db.close();  
}

Cette méthode fait la même chose que celle présenter au-dessus mais pour la classe Branche qui stock les informations des boutons.

Pour pouvoir afficher les notes ou les branches nous avons dû créé une méthode qui doit récupérer grâce à une requête SQL qui parcours toute la base de donnée et qui stock dans une liste les informations qu’il va récupérer.

public List<MarkClass> getAllMark(String year, String branch) {  
 List<MarkClass> markList = new ArrayList<>();  
  
 String selectQuery = "SELECT \* FROM " + *TABLE\_MARK* + " WHERE " + *KEY\_YEAR\_NOTE* + " = " + year + " AND " + *KEY\_IDBRANCH\_NOTE* + " = " + branch;  
  
 SQLiteDatabase db = this.getWritableDatabase();  
 Cursor cursor = db.rawQuery(selectQuery, null);  
  
 if(cursor.moveToFirst()) {  
 do {  
 MarkClass markClass = new MarkClass();  
 markClass.setIdMark(Integer.*parseInt*(cursor.getString(0)));  
 markClass.setMarName(cursor.getString(1));  
 markClass.setMarNote(cursor.getString(2));  
 markClass.setMarYear(cursor.getString(3));  
 markClass.setIdBranch(cursor.getString(4));  
  
 markList.add(markClass);  
 }while (cursor.moveToNext());  
 }  
 return markList;  
}

Dans notre cas nous avons fait en sorte que la requête SQL trie selon les l’année et l’id de la branche que nous récupérons quand nous passons d’activité en activité pour pouvoir seulement sélectionner les notes que nous voulons.

Pour pouvoir gérer les notes nous avons fait une méthode qui permet de supprimer une note comme ça on peut supprimer des notes ou des boutons si l’utilisateur se trompe lors de l’insertion.

public void deleteMark(MarkClass markClass){  
 SQLiteDatabase db = this.getWritableDatabase();  
 db.delete(*TABLE\_MARK*,*KEY\_ID\_MARK* + "=?", new String[] {String.*valueOf*(markClass.getIdMark())});  
  
 db.close();  
}

Le code récupère juste l’intégralité de l’entité et va supprimer l’entité en se basant sur son ID.

### Option

Pour les options il faut que le thème change selon la couleur choisie par l’utilisateur, la modification du mot de passe et choisir si on est Matu, CFC, ou Formation Accélérée (Abandonné au cours du projet)

Tout d’abord il y a la liste déroulante qui s’appellera ici un spinner. Pour afficher son contenu on a créé un tableau de string contenant les couleurs qu’on a ensuite lié au spinner grâce à un adapter à qui on donne le tableau

//Initialisation tableau de string pour le spinner  
private String[] arraySpinner;

//Ajout du contenu dans le spinner  
this.arraySpinner = new String[] {  
 "Rouge", "Bleu", "Vert", "Orange", "Jaune", "Rose", "Bleu Clair", "Brun", "Gris",  
};  
//Création d'un adapter, pour des spinner item, et on donne le tableau créé précedemment  
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.*simple\_spinner\_item*, arraySpinner);  
//Initialisation du spinner  
Spinner s = (Spinner) findViewById(R.id.*spinner*);  
//Ajout de l'adapter sur le spinner  
s.setAdapter(adapter);

Maintenant notre liste déroulante faite, il faut pouvoir récupérer ce qui est a été sélectionné quand on clique sur le bouton Envoyer pour pouvoir changer la couleur du thème. On a fait ceci en utilisant un bouton qui, quand il est cliqué appelle une méthode qui récupère le choix de la couleur et qui envoi cette variable sur un switch.

//Ràcupére le contenu du spinner  
Spinner s = (Spinner) findViewById(R.id.*spinner*);  
String themePicked = s.getSelectedItem().toString();  
  
//Change le themeColor et themeTextColor selon le contenu du spinner  
switch (themePicked)  
{  
 case "Rouge":  
 *themeColor* = "redTheme";  
 *themeTextColor* = "#FFFFFF";  
 break;  
 case "Bleu":  
 *themeColor* = "blueTheme";  
 *themeTextColor* ="#FFFFFF";  
 break;

...

La variable s’appelle donc themePicked et selon ce qu’il y a dedans ça va modifier 2 variables : themeColor et themeTextColor. La première étant pour la couleur de fond et la deuxième pour la couleur du texte, qui parfois ne va pas avec le fond. Ici on y stocke un nom de style qui se trouve dans un fichier annexes et un hexadécimal qu’on utilise pour mettre la couleur au texte plus tard. Exemple pour le rouge et le bleu :

<style name="redTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
 <item name="colorPrimary">#f44336</item>  
 <item name="colorPrimaryDark">#ba000d</item>  
</style>  
  
<style name="blueTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
 <item name="colorPrimary">#3f51b5</item>  
 <item name="colorPrimaryDark">#002984</item>  
</style>

La variable themeColor contient le même texte que le nom du style, on modifie ensuite 2 variable android qui permettent de modifier la couleur du thème qui sont colorPrimary et colorPrimary dark, la première étant la couleur de l’en-tête de l’application et la deuxième celle de l’android. Celle de l’android sera un peu plus sombre pour donner du relief

Maintenant il va falloir appliquer le thème qu’on obtient dans la variable, pour se faire on a utilisé la méthode setTheme()

int styleId = getResources().getIdentifier(*themeColor*, "style", getPackageName());  
setTheme(styleId);

Ici on stocke dans une seule variable le fichier de style et on lui dit de mettre la variable themeColor, on met ensuite le tout dans la méthode setTheme(), à la base je voulais faire setTheme(R.style.themeColor) mais themeColor étant une variable String lui attendait un identifier, d’où l’utilisation de la ligne de code juste au-dessus.

Pour le texte voici le code que j’ai utilisé :

//Applique la couleur du texte sur le titre  
if(storageThemeTextColor == "#000000") {  
 getSupportActionBar().setTitle(Html.*fromHtml*("<font color='#000000'>Gestionnaire de notes ETML</font>"));  
}  
else  
{  
 getSupportActionBar().setTitle(Html.*fromHtml*("<font color='#FFFFFF'>Gestionnaire de notes ETML</font>"));  
}

Ayant décidé de n’avoir que noir (#FFFFFF) ou blanc (#000000) comme couleurs de texte, on a fait un if pour vérifier la couleur de la variable themeTextColor. Ensuite on a un peu triché, n’ayant pas trouvé comment modifier la couleur de l’en-tête sans modifier son aspect on a décidé d’écrire le titre de l’application à la main et de la modifier avec le tag HTML <font color=’’> et d’y insérer donc la couleur voulue.

Le code pour changer le thème et la couleur du titre ont été mise dans une méthode qu’on a nommée « applyTheme », il suffit donc de l’appeler juste comme ça avant que l’activité charge le layout pour changer le thème :

applyTheme();

Maintenant que le changement de thème fonctionne bien on a encore un problème, au redémarrage de l’application le thème s’en va, car stocké dans une variable locale. Ayant d’abord pensé à ajouter une table à la base de donnée, ce serait bête de faire tout ça pour une simple variable, on va donc à la recherche d’un autre moyen qui sera finalement les SharedPreferences, pour faire simple c’est une mini base de donnée sans table ni colonne mais où chaque information est stockée avec une clé. C’est beaucoup plus simple qu’une base de donnée mais très limité. Pour ce qu’on fera avec nos deux variables ça ira

//Créé une SharedPreferences et y stocke une donnée qui est reliée à une clé pour l'identifier  
//SharedPreferences agit un peu comme une base de donnée où chaque table (clé) contient une seule donnée  
//ça permet dans notre cas d'éviter de faire une table dans la base de donnée pour deux variable seulement  
SharedPreferences themeStorage = getSharedPreferences("Data", Context.*MODE\_PRIVATE*);  
SharedPreferences.Editor editor = themeStorage.edit();  
 /\*Clé Donnée\*/  
editor.putString("themeColor",*themeColor*);  
editor.putString("themeTextColor",*themeTextColor*);  
//Commit pour sauvegarder ce qui a été enregistré dans les preferences  
editor.commit();

Ce bout de code sert à créer une instance de SharedPreferences qu’on appelle Data, on lui met le mode privé pour que les données soient lisibles que par l’application elle-même, il y a 2 autres mode permettant de lire les SharedPreferences via une autre application et une autre qui permet d’écrire depuis une autre application. La variable créée s’appellera themeStorage. On crée un éditeur pour themeStorage pour pouvoir modifier les données. Ensuite on appelle cet éditeur et on lui fait un putString avec une clé et la donnée, dans ce cas on a utilisé la clé (string) « themeColor », et on a utilisé la donnée (variable) themeColor, pareil pour la clé themeTextColor on a utilisé la variable themeTextColor, ensuite on doit faire un commit pour sauvegarder la mini base de données. Voyons maintenant comment on a récupéré les données stockées.

//Récupere les préférences utilisateurs  
SharedPreferences themeStorage = getSharedPreferences("Data", Context.*MODE\_PRIVATE*);  
String storageThemeColor = themeStorage.getString("themeColor", "blueTheme");  
String storageThemeTextColor = themeStorage.getString("themeTextColor", "#FFFFFF");

Encore une fois on crée une variable de SharedPreferences en privé, il est important d’utiliser le même nom de SharedPreferences, ici « Data » si on le remplace par autre chose on retrouvera pas informations qui ont été stockées dans « Data ».

Ensuite il faut créer une variable pour chaque donnée à veux récupérer et on utilise la méthode getString sur le themeStorage en insérant la clé dont on veut la donnée. Le 2ème string qu’on récupère est une « Donnée de base » c’est-à-dire que s’il ne trouve rien dans themeColor car l’utilisateur n’a encore rien changé dessus, il utilisera le blueTheme et la couleur de texte blanc. On reprend donc notre méthode applyTheme() qu’on peut modifier pour mettre ces lignes au tout début. Il suffira d’utiliser storageThemeColor au lieu de themeColor. On teste ensuite si le thème reste après fermeture de l’application.

Dans les options il y a aussi le changement du mot de passe. Une zone de texte est prévue à cet effet. Ce mot de passe est lui aussi stocké dans les SharedPreferences, avec la clé « passwrodLogin »

//Instancie l'EditText  
EditText txtPasswordLogin = (EditText) findViewById(R.id.*txtPasswordEdit*);  
//Création des SharedPreferences  
SharedPreferences passwordChange = getSharedPreferences("Data", Context.*MODE\_PRIVATE*);  
SharedPreferences.Editor editor = passwordChange.edit();  
//récupération du login de l'utilisateur  
*passwordLogin* = txtPasswordLogin.getText().toString();  
//Ajoute dans les SharedPreferences le mot de passe  
editor.putString("passwordLogin",*passwordLogin*);  
//Commit pour sauvegarder ce qui a été enregistré dans les preferences  
editor.commit();

Ensuite on a créé une méthode getPassword permettant de le récupérer depuis une autre activité le mot de passe que contient la clé password login et la retourner pour pouvoir l’utiliser dans le login notamment. Le mot de passe de base est. Etml- Comme indiqué dans le manuel d’utilisation

public String getPassword(){  
 //Création du Shared Preferences  
 SharedPreferences passwordChange = getSharedPreferences("Data", Context.*MODE\_PRIVATE*);  
 //Récupération dans une variable le mot de passe  
 //Mot de passe de base : .Etml-  
 String storageThemeColor = passwordChange.getString("passwordLogin", ".Etml-");  
 //Return le mot de passe  
 return storageThemeColor;}

### Login

Le login a été très vite fait c’est la partie de l’application avec le moins de code. On démarre l’application sur cette activité et on a un bouton et une zone de texte qui sont instanciés.

bLogin = (Button)findViewById(R.id.*bConnexion*);  
txtLogin = (EditText) findViewById(R.id.*txtLogin*);

Quand le bouton est appuyé il y a un appel à la méthode getPassword des options qui permet de vérifier si le mot de passe est identique à celui configuré.

if(txtLogin.getText().toString().equals(getPassword())) {  
 Toast.*makeText*(getApplicationContext(),  
 "Redirection...",Toast.*LENGTH\_SHORT*).show();  
  
 //Redirige vers l'activité princpal  
 Intent intent = new Intent(getApplicationContext(), MainActivity.class);  
 startActivity(intent);  
}else{  
 //Affiche si le mot de passe est eronné  
 Toast.*makeText*(getApplicationContext(), "Mot de passe erroné",Toast.*LENGTH\_SHORT*).show();  
 txtLogin.setVisibility(View.*VISIBLE*);  
}

Si le mot de passe est correct, un message qui dit « Redirection… » apparait puis on est redirigé vers l’accueil, sinon un message nous informe que le mot de passe est erroné. Je rappelle que le mot de passe de base est « .Etml-»

### Accueil

Étant la première chose qu’on voit dans l’application après le login on doit savoir quoi faire on a donc pris une interface très simple avec 5 boutons alignés en colonne. Un bouton par année et un bouton pour les résultats.

Chacun de ces boutons contient une action quand il est appuyé

//Initialisation des différents boutons  
Button bFirstYear = (Button) findViewById(R.id.*bFirstYear*);  
bFirstYear.setOnClickListener(branchFirstYear);  
  
Button bSecondYear = (Button) findViewById(R.id.*bSecondYear*);  
bSecondYear.setOnClickListener(branchSecondYear);

//Si le bouton de 1ère année est appuyé  
private View.OnClickListener branchFirstYear = new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 //Création d'une activité avec l'extra de l'année du bouton  
 Intent i = new Intent(getApplicationContext(), BranchActivity.class);  
 i.putExtra("year", "1");  
 startActivity(i);  
 }  
};

Chaque boutons amène à l’activité BranchActivity sauf le bouton résultats, mais ils sont différents du fait qu’ils envoient quelque chose à la prochaine activité qui sera récupérable, c’est un Extra qui s’appelle « year » et qui est égal à l’année selon le bouton, il sert à récupérer l’année sur laquelle l’utilisateur à cliquer pour savoir quelles branches doivent être affichées (1ère années, 2ème années…)

Quant au bouton résultats il amène juste à une autre activité qui est l’activité résultats avec les moyenne de chaque année

Intent i = new Intent(getApplicationContext(), ResultActivity.class);

### Branche

Cette activité et la première réelle grosse difficulté rencontrée dans ce projet. Afficher dynamiquement des boutons et les stocker en mémoire a été une tâche plus compliquée que prévu mais le rendu de la chose est plutôt propre.

D’abord il y a la méthode pour ajouter un bouton « addButtonDB » avec comme paramètre « name »

//Récuperation des extras fournis dans l'activité précédente  
Bundle extras = getIntent().getExtras();  
String years = extras.getString("year");  
//Création du bouton selon l'année et son nom  
button = new Button(this);  
db.addBranch(new BranchClass(i ,name , years));  
button.setText(name);  
button.setId(i++);  
button.setHeight(200);  
button.setWidth(330);  
layout.addView(button);  
button.setOnClickListener(goToNote);

D’abord on récupère les extras venant de l’accueil, puis on crée un new button qu’on ajoute à la Base de donnée avec un ID qui est la variable i instanciée au tout début du programme et qui s’incrémente à chaque bouton, on y met aussi le nom du bouton pour l’afficher avec un nom quand on ouvre l’application et son année pour savoir sur quelle année il doit afficher le bouton.

Ensuite on lui donne un nom et un id qu’on incrémente. On lui set sa taille en hauteur à 200 et 330 en largeur et on l’ajoute ensuite au layout et on lui donne quelque chose à faire quand il est appuyé.

On a aussi la méthode showButton pour afficher les boutons.

private void showButton(String name, int id){  
 //Affichage du bouton selon son nom et id  
 button = new Button(this);  
 button.setText(name);  
 button.setId(id);  
 button.setHeight(200);  
 button.setWidth(330);  
 layout.addView(button);  
 button.setOnClickListener(goToNote);  
 // i = id;  
}

Elle est quasiment pareil, sauf qu’on lui donne l’id du bouton en paramètre car le bouton a déjà été ajouté à la base de donnée il faut juste l’afficher, le but de ces 2 méthodes est de pouvoir afficher les boutons au démarrage de l’activité grâce à showbutton() et une boucle et que de nouveau boutons s’ajoutent dynamiquement qu’on on les ajoute grâce à addButtonBD()

Quand on clique sur les 3 petits ronds en haut à droite et qu’on ajoute une branche ça appelle la méthode onCreateDialog qui affiche un dialogue pour créer une branche

public void onCreateDialog() {  
 //Nouveau dialogue selon le layout de "layout\_addbutton"  
 AlertDialog.Builder mBuilder = new AlertDialog.Builder(BranchActivity.this);  
 final View mView = getLayoutInflater().inflate(R.layout.*layout\_addbutton*, null);  
 //Initialisation du boutton d'envoi  
 Button bSend = (Button) mView.findViewById(R.id.*bSend*);  
 //Quand le bouton Envoyer est cliqué  
 bSend.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 //Initialisation d'un editText  
 final EditText txtName = (EditText) mView.findViewById(R.id.*txtName*);  
 //Appel de la méthode addButtonBD avec le contenu de l'editText en paramètre  
 addButtonBD(txtName.getText().toString());  
 }  
 });

Ici on a un dialogue qui est créé selon le layout de « layout\_addbutton » et qui appelle la méthode « addButtonBD » en prenant le nom qu’on a donné à la branche dans le dialogue.

Maintenant si on veut afficher les boutons à l’ouverture de l’activité il nous suffit de faire une boucle for sur la base de données pour récupérer chaque branche et y appeler la méthode showButton pour chacun d’eux

//Pour chaque branches  
List<BranchClass> branchClasses = db.getAllBranch(years);  
for (BranchClass c : branchClasses) {  
 //Appele la méthode showButton  
 showButton(c.getBraText(),c.getIdBranch());  
}

### Note

Les notes sont affichées via la méthode displayNote() appelée de cette manière :

//Pour chaque note  
List<MarkClass> markClassList = db.getAllMark(years,branchName);  
for (MarkClass c : markClassList) {  
  
 //appel de la méthode displayNote  
 displayNote(c.getMarName(), c.getMarNote(), c.getMarPourcent());

private void displayNote(String name, String mark, String pond) {  
 //Initialisation de layout et des textView  
 LayoutInflater inflater = LayoutInflater.*from*(this);  
 int id = R.layout.*layout\_shownote*;  
 LinearLayout relativeLayout = (LinearLayout) inflater.inflate(id, null, false);  
 txtOne = (TextView) relativeLayout.findViewById(R.id.*txtViewOne*);  
 txtTwo = (TextView) relativeLayout.findViewById(R.id.*txtViewTwo*);  
  
 //Si la pondération n'est pas null  
 if(pond != null) {  
 //Affiche un texte dans le premier textView  
 //Exemple du texte : "Site PHP (Pondération 70%)"  
 txtOne.setText(name + " (Pondération : " + pond + "%)");  
 }  
 //sinon  
 else  
 {  
 //Affiche seulement le nom de la note  
 txtOne.setText(name);  
 }  
 //affiche la note dans le 2ème TextView  
 txtTwo.setText(mark);  
 //Ajoute le textView au layout  
 layout.addView(relativeLayout);  
}

Tout d’abord on accède à la base de donnée et pour chaque notes on appelle la méthode displayNote() en lui donnant comme paramètre le nom de la note, la note, et la pondération

Ensuite, les premières lignes de la méthode sont purement initialisation de variable, de widgets et layout puis on a une if() else qui sert à vérifier si la pondération est null ou pas. S’il y a une pondération on change le texte du premier textView pour y mettre le nom de la note et la pondération. Dans l’application ça donne quelque chose comme : « Site PHP (Pondération 70%) », s’il n’y a pas de pondération on met uniquement le nom de la note. Ensuite dans la 2ème zone de texte on met la valeur de la note, ensuite on les ajoute au relative layout.

L’ajout de note se fait via le menu en haut à droite qui crée un dialogue selon si la note veut être pondérée ou pas (La note pondérée ajoute seulement une zone de texte pour ajouter la pondération)

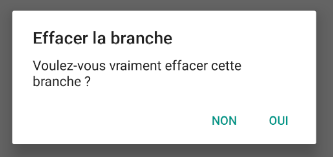
//Méthode créant un dialogue pour ajouter une note  
public void onCreateDialog()  
{  
 //Nouveau dialogue selon le layout de "layout\_addnote"  
 AlertDialog.Builder mBuilder = new AlertDialog.Builder(NoteActivity.this);  
 final View mView = getLayoutInflater().inflate(R.layout.*layout\_addnote*, null);  
  
 //Initialisation du boutton d'envoi  
 Button bSend = (Button) mView.findViewById(R.id.*bSend*);  
 //Quand le bouton Envoyer est cliqué  
 bSend.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 //Initialisation de deux EditText  
 final EditText txtName = (EditText) mView.findViewById(R.id.*txtName*);  
 final EditText txtNote = (EditText) mView.findViewById(R.id.*txtNote*);  
 // //Appel de la méthode addNoteDB avec le contenu des editText en paramètre  
 addNoteDB(txtName.getText().toString(), txtNote.getText().toString());  
 }  
 });  
 //Affichage du dialogue  
 mBuilder.setView(mView);  
 AlertDialog dialog = mBuilder.create();  
 dialog.show();  
}

Ici on crée le dialogue grâce à la ligne AlertDialog.Builder[…] et on l’affichage avec les 3 dernière ligne en bas, puis on dit que quand l’utilisateur appuie sur le bouton pour ajouter la note ça initialise les editText modifié par l’utilisateur avec le nom de la note et la valeur de la note. Puis on appelle la méthode addNoteDB vu précédemment dans la création de la base de données et on y met le nom et la note qui sont alors ajouté à la base de donnée (méthode addNotePondDB si c’est le dialogue pour note pondérée, dans ce cas on ajoute un 3ème paramètre avec la pondération.

Etant toujours dans les menus, il y a une méthode pour supprimer la branche dans laquelle on est actuellement :

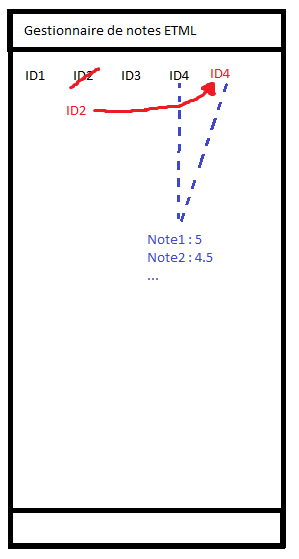
private void alertDialogBranch(final String years, final String branchName)  
{  
 //Crée un dialogue  
 AlertDialog.Builder builder = new AlertDialog.Builder(this);  
 //Titre du dialogue et texte  
 builder.setTitle("Effacer la branche");  
 builder.setMessage("Voulez-vous vraiment effacer cette branche ?");  
 //Bouton oui  
 builder.setPositiveButton("OUI", new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int which) {  
 //Pour chaque note  
 List<MarkClass> markClassList = db.getAllMark(years,branchName);  
 for (MarkClass c : markClassList) {  
 //La retire de la base de donnée  
 db.deleteMark(c);  
 }  
 //Appel de la méthode pour effacer la branche  
 db.deleteBranch(db.getBranch(Integer.*parseInt*(branchName)));  
 //Ferme le dialogue  
 dialog.dismiss();  
 finish();  
 }  
  
 });  
 //Bouton non  
 builder.setNegativeButton("NON", new DialogInterface.OnClickListener() {  
 @Override  
 public void onClick(DialogInterface dialog, int which) {  
 // Ferme le dialogue  
 dialog.dismiss();  
 }  
 });  
 //Affiche le dialogue  
 AlertDialog alert = builder.create();  
 alert.show();  
}

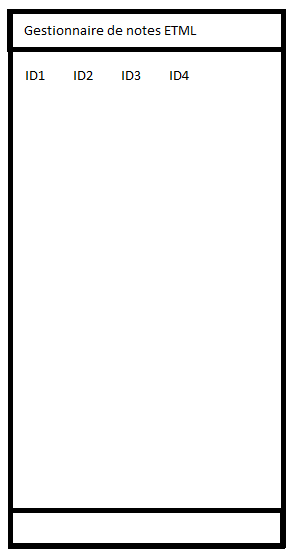
Ici on a un dialogue un peu plus complexe qui ressemble à cela :



Au cas où on clique sur le bouton de suppression par hasard on demande ici une confirmation pour la suppression de la branche, dans le code on donne un titre qui est affiché en gras et un message qui demande s’il veut vraiment supprimer la branche. Ensuite on crée un positiveButton et un NegativeButton qui ont chacun un onClick, Si on appuie sur le non on ferme le dialog, rien d’autre ne se passe.

Mais si on appuie sur le bouton oui, on récupère chaque note, qu’on efface pour ne plus rien avoir dans la base de données puis une fois cela fait on supprime la branche. Les méthodes utilisées sont celles de la base de données présentée précédemment dans le 4.1.1

Malheureusement ce procédé interfère avec la création des branches, j’ai fait un schéma pour l’expliquer plus facilement :

Ici nous avons la liste des branches représentées par leurs IDs, il y en a 4.

Mais à la suppression de l’ID2 il nous en reste 3, à la création de la prochaine branche il verra 3 branche et donnera donc l’ID4 ce qui liera 2 branches aux mêmes notes dû à leurs ID

Ce problème peut être réglé en modifiant l’algorithme, mais pour l’instant on a d’autre préoccupations. Si vous voulez utiliser la fonction de suppression il faut supprimer uniquement la dernière branche créée ou alors toutes les branches après celle supprimée pour régler le problème car si dans notre exemple on supprime la 4, il y aura 3 ID et ça va créer la 4ème, ou alors on supprime la 2,3,4 et ça les recréera sans problème.

Ensuite il reste le calcul de moyenne, pour ne pas faire un copié/collé de l’algorithme qui est vraiment long, il est dans l’activité « NoteActivity » à partir de la ligne 57 et commenté. Je vais quand même en faire un petit résumé sans mettre trop de code pour avoir l’essentiel de cet algorithme.

Pour le calcul de moyenne on a créé quelques variables

private int nbNote =0;  
private double calculmoy;  
private int totalPond;  
private boolean isPond;

* Le nombre total de note dans la branche
* Variable pour le calcul de la moyenne
* Le total de pourcentage de pondération
* Booléen pour savoir si la branche a des notes pondérées

Pour le moment, on est parti du principe que l’utilisateur ne mélangera pas notes normales et pondérées et rentrera ses notes correctement. Pour calculer la moyenne on récupère chaque note. Pour chacune d’elles on vérifie d’abord si elle est pondérée, si oui on y ajoute sa pondération à la variable totalPond et la variable calculmoy on y ajoute la pondération \* la valeur de la note puis le booléen isPond passe à true car juste après on sort de la base de donnée et on aura toujours besoin de savoir si la branche est pondérée ou pas.

totalPond = totalPond + Integer.*parseInt*(c.getMarPourcent());  
calculmoy = calculmoy + Double.*parseDouble*(c.getMarPourcent()) \* Double.*parseDouble*(c.getMarNote());

Ensuite On vérifie si la branche est pondérée avec le booléen isPond et on calcul la moyenne en la divisant pour la pondération totale.

calculmoy = calculmoy / totalPond;

Par contre si la branche n’est pas pondérée, pour chaque note on incrément le nombre de note et on ajoute la valeur de la note dans la variable de calcul moyenne.

nbNote++;  
calculmoy = calculmoy + Double.*parseDouble*(c.getMarNote());

Ensuite on calcul la moyenne en la divisant par le nombre de note

calculmoy = calculmoy / nbNote;

Puis dernière étape en vérifie si la moyenne est supérieure ou égale à 4, ou bien inférieur à 4 et on stocke la moyenne dans une variable string pour l’ajouter au texte

moyenne.setTextColor(Color.*parseColor*("#4caf50"));

moyenne.setText(stringMoy);

J’ai aussi intégré à l’ajout d’une note le calcul de la moyenne pour que celle-ci soit calculée dynamiquement à l’ajout d’une note et pas seulement au lancement de l’activité, mais cela pose un problème : Si c’est la première note qui est ajoutée à la branche le calcul de la moyenne ne se fait pas.

Après plusieurs recherches j’ai trouvé que c’était cette ligne qui posait problème

calculmoy = calculmoy / nbNote;

Vu que Calcul moyenne est égal à 0 car c’est la première note qu’on ajoute, s’il y a une division par 0 qui est stockée dans calculmoy ça ne stocke pas un chiffre et la variable devient inutilisable, j’ajoute donc une condition aux endroits avec une division pour minimiser les risques et cela fonctionne parfaitement

### Moyenne

Pour le calcul des moyennes des années on a passé un moment à réfléchir comment stocker les différentes moyennes en les mettant dans les 4 textView dynamiquement, on a essayé des tableaux de string mais on a fini par trouver qu’on pouvait faire un tableau de Widget, on a donc fait un tableau de TextView

//Tableau de TextView  
TextView []tv=new TextView[4];  
tv[0]=(TextView)findViewById(R.id.*moy1*);  
tv[1]=(TextView)findViewById(R.id.*moy2*);  
tv[2]=(TextView)findViewById(R.id.*moy3*);  
tv[3]=(TextView)findViewById(R.id.*moy4*);

L’algorithme est long et ressemble beaucoup à la précédente du fait qu’on calcul toutes les branches puis la moyenne des branches de l’années. Je décrirai donc surtout les changements

private double yearMoy;  
private double branchMoy;  
private int totalPond;  
private boolean isPond;  
private int nbNote;  
private int nbMoy;  
private int year =1;

* isPond qui sert à savoir si une branche est pondérée ou pas
* year qui permet de savoir à quelle année on est (pour la base de données)

Variables qui se réinitialise à chaque année :

* YearMoy qui est utilisé pour calculer la moyenne de l’année.
* nbMoy qui sert à savoir combien de moyenne (donc branches) il y a chaque année

Variables qui se réinitialise à chaque branche :

- BranchMoy qui stocke la moyenne d’une branche.

- totalPond qui stocke encore une fois la pondération totale d’une branche

- nbNote qui stocke le nombre de note dans la branche

for (int i=0; i < 4; i++) {

On fait une boucle for qui tourne 4 fois, pour les 4 années et dans chacune on prend toutes les branches comme vu précédemment et pour chacune on calcul la moyenne de la branche qu’on additionne dans yearMoy ensuite on la divise par le nombre de moyenne qu’on y a additionné (nbMoy) et on l’ajoute de cette manière au textView :

tv[i].setText(String.*valueOf*(yearMoy));

Le i correspond à la boucle for, pour la premiere année il sera à 0 et il ira donc dans la première position du tableau de textView qui sera la première année, pour l’année 2 il ira dans le tableau position 1 qui est le textView de la 2ème année et ainsi de suite…

Ensuite on refait une vérification de la moyenne pour changer la couleur en vert ou en rouge et le tout s’affiche parfaitement



# Tests

## Dossier des tests

|  |  |  |
| --- | --- | --- |
| Type de test | Réussi ? | Commentaire |
| Lancement de l’application | OUI | Le programme se lance parfois avec une erreur de compilation de l’émulateur |
| Création de l’Apk | NON | Problème lors de l’installation de l’apk signée |
| Stocker des infos dans les SharedPreferences | OUI | Il y a juste besoin d’une clé pour stocker une valeur et de la récupérer via cette même clé |
| Affichage de boutons dynamique qui s’adaptent avec l’orientation de l’écran (portait, paysage) | NON | Notre application sera uniquement au format portrait c’est un bon début pour une application Android déjà |
| Création de l’Apk | OUI | Le smartphone Android doit être mis à jour (Minimum version 7.0) |
| Interface sur téléphones physiques | OUI | Aucun problème distingué entre l’émulateur et le téléphone, en terme de fonctionnalité et de design |

# Conclusion

## Bilan des fonctionnalités demandées

* Création de la grille de notes Fonctionnel
* Mise en forme de base (CFC, Maturité, Formation accélérée) Abandonnée
* Insérer des modules Fonctionnel
* Insérer des notes (pondérées) Fonctionnel
* Calcule automatique des moyennes Fonctionnel
* Affichage des résultats finaux Fonctionnel
* Stockage des données en local Fonctionnel ou en Cloud si le temps le permet Abandonnée
* Choix du thème par l’utilisateur (Couleur du fond) Fonctionnel
* Initialisation du Mot de passe utilisateur Fonctionnel

## Bilan de la planification

Selon la planification on a fini juste à temps malgré les petits bug et problèmes qu’on aurait voulu résoudre. Grégory a pu en résoudre un peu car il n’était pas au camp de sport.

La plupart des fonctionnalités ont été implémentées plus rapidement que prévu mais on a pris du retard sur quelques points notamment : Création de boutons dynamique/Création APK/Choix du thème par l’utilisateur

Pour les points qui n’ont pas été totalement fini ou abandonné :

* Le cloud est pas disponible à cause de la complexité de celui-ci par rapport à ce que l’on a appris sur android pendant ce projet, le cloud Google étant le plus simple est faisable pour nous va déjà trop loin comparé à nos connaissances
* Le choix CFC/Matu/FIN a été annulé du fait que l’utilisateur crée toutes les branches lui-même incluant celles de bases.
* Une fonctionnalité ajoutée en plus quand on avait quasiment fini le projet : la suppression des branches fonctionne malgré un bug non résolu qui a été documenté au point [4.1.6](#_Note)

## Bilan personnel

Grégory Poget :

Si c’était à refaire, je n’aurais pas à apprendre toutes les bases car elles sont déjà apprises, j’aurai une semaine en plus pour travailler directement sur le projet même si finalement il était pas si compliqué on a surtout été heurtés à des problèmes de débutant qui, une fois avoir compris comment le régler ne se reproduit plus, je le finirai bien plus vite ce qui me permettrait d’aller mettre des options supplémentaires et d’optimiser le code pour éviter les problèmes, que l’application soit vraiment utilisable par des élèves de l’ETML.

Pendant le projet de GestProj2 nous n’avions pas pu utiliser Git dans une situation adéquate dû aux 4 périodes par semaine de projet, nous l’avons donc utilisé sur ce projet avec GitKraken sur un dépôt GitHub et à part quelques problème, ce serait quelque chose à réutiliser dans un prochain projet de groupe je pense. Pendant mon projet j’ai eû un gros problème avec GitKraken je ne pouvais plus ouvrir ou cloner le repository, je faisais mes commit via GitHub ce qui entrainait des problèmes sur certains fichiers que j’ai commit qui n’étaient pas là pour André sur GitKraken alors que pour moi ils y étaient sur le GitHub. Malgré Ce problème qui s’est réglé 1 semaine plus tard sans rien toucher, Git est un outil de versionning qui s’avère très intéressant dans des travaux de groupes en situation réelle comme nous l’avons utilisé

# Divers

## Journal de travail

Les journaux de chacun sont présents sur le dépôt github du projet à l’adresse <https://github.com/Wayne121/AGS-GPT-DemoMot>

## Webographie

<https://www.youtube.com/>

<http://jmdoudoux.developpez.com/cours/developpons/java/>

<https://www.stackoverflow.com/>

<https://www.developer.android.com/>

# Annexes

[H:\DemoMot\AGS-GPT-DemoMot\Modele-JDT.xlsm](file:///H:\DemoMot\AGS-GPT-DemoMot\Modele-JDT.xlsm)