**Projet 3 parcours Android**

**Mémorisez votre humeur du jour**

**Nom de l’application: MoodTracker**

Utiliser le lien <https://github.com/fponroy/MoodTracker> pour des ressources utiles au projet, images etc...

Les utilisateurs notent leur humeur dans l’application chaque jour.

Ils peuvent ajouter des commentaires expliquant la raison de leur humeur.

Il faut ajouter les dossiers Drawable permettant à l’application de s’adapter à différents types d’écrans aux formet xxx\_hdpi xx\_hdpi x\_hdpi, hdpi et mdpi. Il faut aller dans le dossier ressources, clique droit et faire new / Android ressource directory. On précise le type choix Ressource Type dans en mettant drawable, le nom de dossier sera affecté automatiquement. Dans la section Available qualifiers, on sélectionne Density, on clique sur le boutton permettant de switcher à droite et on obtien la capacité de sélectionner un dossier qui correspondra à un seul format, pour chaque format il faut créer un dossier Ressource Type drawable et y mettre les images relatives.

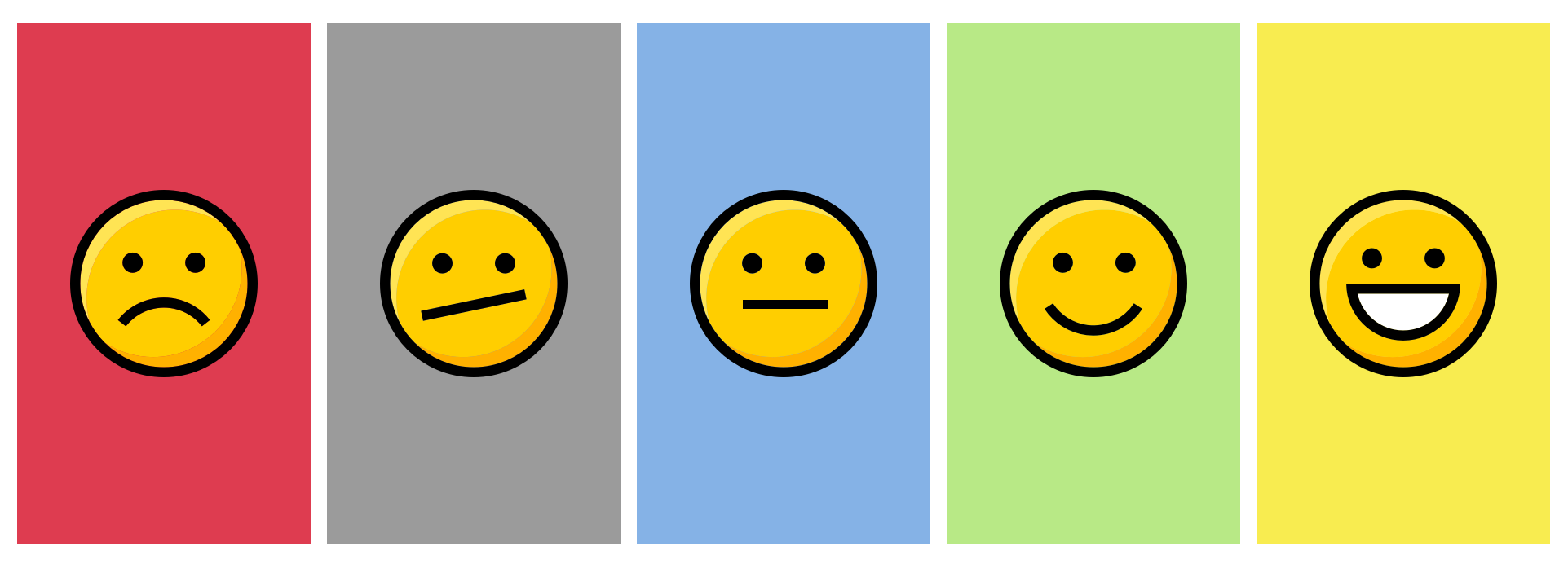
→ Enregistrer l’historique de l’humeur sur 7 jours maximum.

Il faut utiliser du code permettant d’interagir avec le temps

→ Il y’aura 5 humeurs possibles, correspondant chaqune à une image

1) très mauvaise humeur – 2) Mauvaise humeur - 3) Humeur normale

4) Bonne humeur - 5) Super bonne humeur



* téléchargement des smileys

→ code permettant d’enregistré les humeur en base de données

* → Il faudra inclure les couleurs correspondant dans les background de chacune ds images.

|  |
| --- |
| **"faded\_red">#ffde3c50** **Rouge**  **"warm\_grey">#ff9b9b9b** **Gris**  **"cornflower\_blue\_65">#a5468ad9** **Bleu**  **"light\_sage">#ffb8e986** **Vert**  **"banana\_yellow">#fff9ec4f Jaune** |

Les composants images doivent avoir chacune avoir un conteneur parent FrameLayout qui prend la hauteur de l’écran et la largeur de l’écran,

c’est les couleurs de background de ces FrameLayout qui doivent être modifier

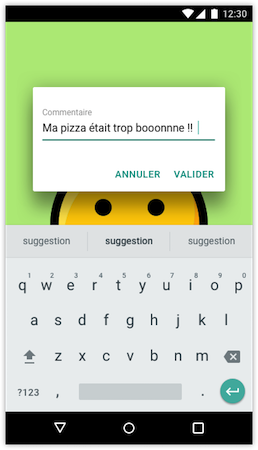
android:background=«#ff0000» est l’attribut servant à indiquer la couleur de fond d’un composant

En slidant vers le haut on passe progressivement au humeur les plus joyeuses

en slidant vers le bas on passe progressivement au humeur les plus tristes

* → L’humeur est mémorisé pour la journée complète et peut être changer à tout moment. A minuit les dernière humeur sélectionné sera celle enregistré définitivement pour ce jour.

Au début de chaque jour, l’humeur joyeuse est affiché par défaut, la variable enregistrant cette humeur devra donc avoir bonne humeur par défaut, si on ne modifie rien dans la journée elle sera enregistrée définitivement.

* L’écran de choix d'humeur contient deux boutons :
* • En bas à gauche, un bouton permet d’ajouter un commentaire. Lorsque vous appuyez dessus, une popup s’affiche, avec une zone de texte et un clavier vous permettant de saisir du texte.
* • En bas à droite, un bouton permet d’accéder à l’historique de vos humeurs. 
* → il y’aura 5 classes dont 4 avec leur activité respectives, la première qui affiche le smiley actuel et aussi là ou on slide les humeurs et nommé MainAtivity et la seconde qui affiche l’historique des humeurs nommé MhistoryActivity. La 3 ème sera une activité qui affiche une bôite de dialogue personnalité nommé DiaogComment pour commenter l’humeur. La classe MoodManager sans activité layout qui devra manager la séréalisation des données avant leur enregistrement de données et la déséréalisation de celle-ci avant leur lecture et leur affichage ou leur usage. La dernière classe sera la classe PiechartHistory qui associé a son layout, affichera des statistiques de l’humeur des 7 dernies jours en forme de camenbert.
* → S’en suit le téléchargement des fichiers correspondant aux icones des bouttons permettant de créer le commentaire et d’afficher l’historique
* Concernant les Imagebuttons il peut y avoir un fond gris en arrière plan à ces bouttons, si ça arrive, pour leur enlever il suffit de copier la valeur situé dans l’attribut srcCompat et de la copier dans l’attribut background. Cette duplication règle le problème.

**Construction de MainActivity**

* → J’ajoute deux composants ImagesButton et je fais le lien avec les images.J’ajoute aussi la ScrollView pour slider verticalement. Les images ont été ajoutée et défilent verticalement les unes au dessus des autres

Arborescence : ScrollView > LinearLayout > ImageView

→ Dans le ImagesButton servant à ajouter un commentaire il faudra générer un fenêtre pop up permettant de saisir du texte

* → les méthodes et variables

setContentView(R.layout.*activity\_main*) sera utilisé pour déterminé quel layout utilisé

La méthode findViewById() servira à récuperer via la classe des ressources R les boutons pour code leur fonctionnements en les affectant d’abord a des variable du même type.

La méthode setOnClickListener possède en paramètre la méthode appeler View.OnClickListener() qui est utiliser lorsque des évenements arrive

La méthode OnClick est appeleé chaque fois qu’un utilisateur clique sur le bouton.

Pour pouvoir permette au bouton d’afficher l’historique situé sur l’autre page, il faut construire un objet Intent en lui mettant en premier paramètre la page où il se situe, c’est à dire MainActivity puis en deuxième paramètre la page qu’on souhaite affiche.

Après la construction de l’Intent, on utilise la méthode startActivity() en mettant l’Intent en paramètre pour démarrer la nouvelle activité.

* → Les images doivent devenir des ImageButton pour que l’utilisateur puisse cliquer sur l’humeur et que lorsqu’on cloque dessus un evenement se déclenche et qu’on puisse exécuter du code. Après J’affiche une fenêtre dialogue qui s’ouvre pour confirmer si l’utilisateur souhaite afficher l’humeur choisi ou pas.

Le constructeur des boite de dialogue est

AlertDialog.Builder alertMood = new AlertDialog.Builder(MainActivity.this);

* Je précise ensuite avec des méthodes le nom des messages, chaque boite de dialogue sera nommé <<Votre Humeur>> et la question sera adapté à chaque humeur «Etes vous sur d’être de très bonne humeur ? Etes vous sur d’être de très mauvaise humeur>> etc.
* Le commentaire sur l’humeur doit être écris avant d’enregistrer les autres données de l’humeur, c’est lorsque que l’on clique sur un bouton d’humeur et en cliquant sur validerque le commentaire sera enregistrer avec les autres données
* l’ utilisateur confirme par la suite si il veut enregistré l’humeur ou pas. Méthodes utilisées setMessage() setTitle setPositiveButton() et SetNegativeButton() de l’objet Builder. Toast.makeText() est la méthode utilisé dans les méthodes onClick pour afficher les messages.
* → Après les avoir codé pour chaque bouton je remarque la répétition de code similaire. Je crée donc une méthode qui prendra tout ce code et sen seul paramètre la phrase demandent si l ’utilisateur est sur de valider l’humeur sélectionné et le bouton relatif à l’humeur, ils seront les 2 paramètres. La méthode s’appelle public

void alterDialogMood( ImageButton valeur, String valeur2). J’économise 102 lignes de code.

* → Jouer une petite note de musique différente lorsque vous sélectionnez une humeur .

Concernant les fichiers son, il faut créer un dossier raw dans le dossier de ressource.

Dans le MainActivity, j’implémente la classe MediaPlayer qui sert à lire les musique.

Je créer un objet de type MediaPlayer nommé playMusic

sans l’initaliser au départ. L’inialisation se fera dans un switch situé dans le code de la méthode qui gère les bouttons. Le Switch doit être placé après la méthode onClick(). Le constructeur appelé est MediaPlayer.*create*(getApplicationContext(),R.raw.*sound\_happy*) et la méthode permettant de lancer la musique est start playMusic.start().

Une icône est affichée si l’humeur contient un commentaire. Si une humeur contient un commentaire et que vous cliquez dessus, le commentaire est affichée brièvement en bas de l’écran, via un message de type Toast.

* → Pour mémoriser les données j’utiliserais la classe SharedPreferences dans la MainActivty et aussi dans MhistoryMood pour récupérer les données et les affichés dans l’historique. elle récupère les données que vous lui passez et les stocke dans un fichier JSON.

SharedPreferences mood\_Record = getSharedPreferences("mood\_data\_file", *MODE\_PRIVATE*);

* Je crée la variable objet de cette classe pour gérer l'entrée de date dans le fichier en utilisant la méthode d'un objet SharedPreferences, je mets d'abord le nom du fichier xml et mettre MOVE\_PRIVATE pour nous assurer que les données ne sont accessibles que par l'application MoodTracker et pas en dehors. L'objet Editor de cette classe sera utilisé pour écrire des données dans le fichier, le premier paramètre est la clé et le second est la valeur.
* Il faut au préalable télécharger la libraire JSON et la mettre dans le dossier app/libb et l’ajouter en tant que libraire,

SharedPreferences mood\_gson = getSharedPreferences("mood\_data\_file\_gson", MODE\_PRIVATE);

SharedPreferences.Editor mood\_gson\_Editor = mood\_gson.edit();

Gson gson\_manager = new Gson();

String mood\_data\_gson = gson\_manager.toJson(mood\_recorder.mood\_list\_data\_gson\_string());

mood\_gson\_Editor.putString("mood\_data\_gson", mood\_data\_gson).apply();

Partager votre humeur du jour avec un ami, en lui envoyant par SMS ou par email

Il faut déclarer au préalable les permissions d’usage des SMS et d’internet dans le fichier de manifestes, il faut quand même garder l’instruction de permission pour internent dans le manifest car seul les appareils ayant Android Marshmallow ou verions superieur ont cette permission par défaut, par précaution je la mets pour les version inférieur à Marshmallow

<uses-permission android:name="android.permission.SEND\_SMS"/>

<uses-permission android:name="android.permission.INTERNET"/>

le code suivant est à inclure pour avoir la fonctionnalité de partage, on utilise une variable objet de type Intent, la méthode putExtra() permet de mettre le message à envoyer:

Intent sendIntent = new Intent();

sendIntent.setAction(Intent.*ACTION\_SEND*);

sendIntent.putExtra(Intent.*EXTRA\_TEXT*,mood\_recorder.mood\_list\_data\_gson\_string());

sendIntent.setType("text/plain");

startActivity(sendIntent);

* La mise à jour de l’affichage des données d’historique doit se faire dans le MainActivity en mettant une condition permettant de vérifier si la date de l’humeur la plus récente est égale à la date d’aujord’hui on ne fait rien, seul le premier rectangle sinon si la date est différente on switch les valeurs de chaque jour relative aux rectangle au dessus.

Il faut commencer par alterner les données les plus vieilles 6=X à 7=E .. 7=X , 5=O a 6=X… 6=O, 4=6 a 5=O…. 5=6, si on comme par les plus recente seule la donné la l’humeur la plus récente sera affiché dans l’historique et les autres rectangles ne seront pas mis à jour et leur valeure seront perdues. 1=D 2=R 3=I 4=L

1 = D a 2=R… 2=D, 2=D a 3=I… 3=D , 3=D a 4=L…. 4=D.

Pour ne pas perdre en cohérence, lorsque l’utilisateur appuie pour valider, l’affection de la nouvelle donné à l’humeur la plus recente doit se faire après le test de la condition de comparaison de date, sinon l’ancienne donnée sera perdu.

Concernant la date il ne faut pas comparé directement des chaine variable string formaté mais utilisé la méthode contentEquals() de l’une et mettre l’autre date à comparé en paramètre sinon leur valeur de ne pourront pas être vraiment comparé.

* Pour faire l’implémentation du code de valeur par défaut et du code adapté au premier lancement de l'applicationDans la MainActivity pour vérifier si c'est le premier lancement de l'application, je fais une condition qui verifié si la clé 1 existe dans le fichier de sauvegarde.

Si elle n'existe pas, la fonction enregistrant les données prend en paramètres dans le nom de couleur d'une variable objet "first\_launch\_application" qui sera la valeur attribuer sur les 7 clé.

Dans le code MHhistoryActivity, quand les conditions voient cette valeur alors ils n'éxecutent pas le code pour leur rectangle respectif.

Si la clé existe alors sa signifie que l'appliation n'est pas utilisé la première fois et la condition ne sera plsu jamis executé étant donné que la clé 1 existe forcément. Chaque jour qui passera les variables possèdant la valeur first\_launch\_application changeront cette valeur les unes après les autres et executeront les codes de changement de couleur des rectangle et d'affichage du texte montrant de combien de temps date l'humeur.

* Pour que les valeur par défaut lorsque l'utilisateur n''appuie sur aucun bouton dans l'application, il faut affecter les valeur mood\_color et mood\_sentence par Super Bonne humeur et jaune en hexadécimal dans uen condition if juste en dessous des appel à la méthode permettant de valider les boutons alertDialogMood.

Dans la condition on précise de vérifier si les variables mood\_name et mood\_Color sont nulles, si la condition est vrai cela signifie que l'utilisateur n'a choisie aucune humeur. Il faut aussi appeler la méthode qui mets à jour les données sauvegardé dans la condition après avoir affecté les valeur par défaut au variables.

**Construction de MHistoryActivity**

* Si j’ai besoin d’utiliser un layout pour l’agencement, j’utiliserais le LinearLayout pour agencer l’historique des humeurs les une au dessus des autres.
* La couleur gris clair en android:background doit être définie

→ L’affichage des éléments d’historique doit être crée dynamiquement. La taille de chaque conteneur rectangulaire varie en fonction de l’humeur choisie, plus l’utilisateur est heureux, plus le rectangle sera grand. Je dois voir si je peux modifier les propriétés de largeur des rectangles dans le code.

→ Je dois agencer les ButtonImage situés dans les rectangles qui permettront d’afficher les commentaire en le positionnant à droite avec l’attribut android:layout-gravity en mettant la valeur right. Pour fonctionner les bouttons doivent être mises en tant que composants enfant du Scrollviews

* → Le texte indiquant depuis quand une phrase disant de combien de temps date l’humeur doit être agencer en haut à gauche du rectangle

→ 7 humeurs, 7 jours, 7 rectangles avec leur couleur relatives. Pour changer les couleurs des rectangles, j’utilise la méthode de la classe Color parseColor en y mettant une valeur hexadécimal.

Le site <http://www.code-couleur.com/> en fournis, parseColor, qui prend en paramètre une couleur au format hexadécimal, convertie cette couleur en int car la méthode de modification de couleur setBackgroundColor utilisé dans MhistoryMoodn appartenant au TextView que prend que des couleur représenté par des valeur de type int en paramètre:

int theGreen = Color.*parseColor*("#65D164");t\_yesterday.setBackgroundColor(theGreen);

* Sur l’écran d’historique, les sept dernières humeurs enregistrées sont affichées verticalement, de la plus ancienne à la plus récente.
* → Pour associer un élément de vue / composant d’un entité layout à sa classe relative, il faut dans la classe relative utiliser la méthode et on affecte l’objet retourné à une variable qui doit être déclarér du même type que le composant au préalable. Une fois cela fait on peut manipuler les propriétes de l’objet.

TextView t\_yesterday = findViewById(R.id.*mood\_View*),

Pour accéder aux données, je code dans l’activité d’historique MHActivityHistory, la méthode getSharedPreferences est utilisée pour lire les données dans le fichier xml et donner leurs valeurs aux composants du layout de l'historique. Je mets le nom du fichier que je veux des données et je mets la clé nécessaire.

Gson gson = new Gson();

String json\_file = getSharedPreferences("mood\_data\_file\_gson",MODE\_PRIVATE).getString("mood\_data\_gson", "");

String many\_gson = gson.fromJson(json\_file, String.class);

* Les boutons présents dans la page d'historique apparaissent si les données d'humeur auxquels ils sont liés appraissent et n'apparaissent pas si les commentaires sont vides.

On modifie la visibilité du boutton avec la méthode setVisibility() qui prend en paramètre soit View.INVISIBLE soit View.Visible.

Les messages toast sont implémenté à l'interieur des conditions confirmant l'existance de commentaires.

**Construction de DialogComment**

DialogComment devra contenir les variables Button yes et no en propriété ainsi que la variable EdiTexte permettant de récupérer le commentaire, avec chacune le méthode get() respectives

**Construction de MoodManager**

* Les objet des types MoodManager possèdent 4 variables, le nom de l’humeur, la phrase qui correspond au commentaire et qui peut être vide, la couleur de l’humeur et la date uniquement au format année moi et jour. Les méthodes set et get pour chaque propriété, un constructeur par défaut vide et une méthode permettant de regrouper toutes les méthodes set pour les ajouter dans une collection de type Set<String> car la méthode permettant d’enregister une colletion de donné de l’objet Editor ne prend que ce type de collection uniquement.

Pour la date je crée une variable de type Date et une autre de type SimpleDateFormat(«ddmmyy»)

en précisant dans les paramètres l’année, le mois et le jour uniquement. Puis j’applique la méthode format() de l’objet SimpleDateFormat en y mettant la variable Date pour obtenir le jour au format String.

La méthode appelé pour l’insertion des données agence les données en retournant un string pour être enregistrer au format JSON de la manière suivante en y mettant les accolades, les guilletemets et les virgulets:

String mood\_list = "{";

mood\_list += " \"mood\_name\" : \""+ mood\_name+"\",";

mood\_list += " \"mood\_sentence\" : \""+ mood\_sentence+"\",";

mood\_list += " \"mood\_sentence\" : \""+ mood\_sentence+"\",";

mood\_list += " \"mood\_date\" : \""+ mood\_date+"\"}";

MoodManager contient une méthode qui permet de rendre les données exploitable lorsque l’on souhaite les lire dans l’historique au lieu de les affiches avec les accolades guillement et virgules inutiles pour l’utilisateur

//Curly bracket deleted

the\_string = the\_string.replaceAll("\\{|\\}","");

//quote

the\_string = the\_string.replaceAll("\"","");

// i replace all the comma by ":" to make this character the split parameter to create an array

the\_string = the\_string.replaceAll(",",":");

//I split all the data and put them in an array

String[] the\_array\_string = the\_string.split(":");

****

**Construction de DialogComment**

* Déclarer DialogComment dans le manifest après avoir créer la classe et le layout respectif
* j'importe la classe Activity pour utiliser un objet d'activité en tant que paramètre

cause je vais utiliser l'activité qui contient le composant nécessaire pour un dialogue personnalisé.

J'appelle le super constructeur parce que je devrai utiliser le contexte et le second paramètre qui sera le genre de style de dialogue utilisé dans la ressource R.style.Theme\_Appcomat\_Day\_Night\_Dialog. Il faut bien choisir le thème car certains sont incompatibles et créent des bug, par exemple mon texteView ne s’est pas affiché avec un autre thème

* avec la méthode setContentView() je dois mettre le layout à la boîte de dialogue

Je crée les methode permettant de retourné les bouttons personnalisé car les méthodes setPostivieButton et setNegativeButton sont inutile pour montrer des boutons personnalisé

**Construction de PiechartHistory**

* Pour utiliser des fonctionnalités graphiques de statistiques j’utilise la librairie MPAndroidChart en ajoutant la ligne de code maven { url 'https://jitpack.io' } dans le fichier build.gradle section repositories. Ensuite j’ajoute la ligne de code implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0-alpha' dans app/build.gradle.

Déclarer ensuite un objet de type PieChart et vie l’une de c’est méthode je mets le mode % activé piechart\_history.setUsePercentValues(true); Je récupère les données en les déséréalisant puis j’utilise des variables de types float pour chaque couleur. Chaque fois qu’un couleur est utilisé dans le fichier de sauvegarde j’incrémente sa variable relative. Le résultat est ensuite enregistré dans des variable de type float.

Pour obtenir le pourcentage on multiple le de fois que la couleur est trouvé dans le fichier par 100 puis on la divise par le nombre total de couleur trouver dans le fichier.

Je crée une list d’objet PieEntry, ceci prennent en paramètre une variable de type float qui recoit les résultat des clacul et en second paramètre le label qui sera affiché et décrivant le nom du pourcentage Super bonne humeur 60% par exemple. Un objet PieDataSet prend en paramètre la liste PieEntry et en second paramètre le nom de la liste en string. Puis un objet PieData prend cet objet PieDataSet en paramètre.

Pour permettre d’avoir des parties colorés dans le camenbert de pourcentage j’utilsie la méthode mood\_data\_set.setColors(ColorTemplate.COLORFUL\_COLORS).

Pour afficher le graphique j’utilise la méthode piechart\_history.setData(mood\_pie\_data) dans un objet PieChart qui a reçu le layout affichant les données graphiques en tant que valeure.PieChart piechart\_history = findViewById(R.id.piechart\_drawing);

* Indentez le code

• La longueur de vos classes est idéalement inférieure à 200 lignes. Le maximum toléré est de 500 lignes.

• La longueur de vos contrôleurs (Fragments et/ou Activités) ne doit pas excéder 300 lignes (les imports ne seront pas inclus).

• La longueur de vos méthodes est idéalement inférieure à 30 lignes. Le maximum toléré est de 50 lignes.

* • Votre code est hébergé sur Github avec des commits les plus fins possibles et un historique cohérent ;

• Vous respectez les bonnes pratiques de développement indiquées plus haut ;

* • Votre code est entièrement écrit en anglais (commentaires, variables, méthodes, etc) ;

• Le projet compile correctement et ne contient aucun message d’avertissement ou d’erreur ;

• L'application doit s'afficher correctement sur toutes les tailles de téléphone Android en mode portrait ;

* L'application doit supporter Android 4.4 et les versions supérieures. Une capture d’écran a été prise pour montrer le choix d’Android 4.4 lors de la création du projet

Fournir

* Le projet Android Studio
* Une documentation fonctionnelle de l’application finale, fonctionnalité complémentaire inclue, au format PDF.