|  |
| --- |
| M2 STRI |
| Interaction homme-machine |
| Livrable 3 : Document de spécification de la conception |

|  |
| --- |
| BADENS Florian  GUENAT Vincent  12-12-2016 |

# Analyse des composants graphiques Android

Dans notre projet nous avons utilisés les composants ci-dessous :

* Panneau latéral (navigation drawer)
* Option Menu (icônes en haut à droite)
* Boutons
* Listes
* Champs de texte (Edit Text)
* Date Picker
* Rating Bar

Et les vues et layout :

* TextView
* ImageView
* ListView
* LinearLayout
* RelativeLayout
* CoodinatorLayout

Ces composants peuvent être classés en plusieurs catégories selon leur fonction. Nous avons ainsi les éléments qui :

* permettent à l’utilisateur de nous donner une information
* nous permettent d’afficher du contenu à l’utilisateur
* permettent de passer d’une page à une autre.

## Obtention d’une information

Les champs de textes sont assez rares dans notre application. Ils sont utilisés dans la page de Connexion/inscription pour demander les informations d’authentification à l’utilisateur, dans la page Mon profil pour lui demander des informations sur lui et dans la page de recherche pour chercher le contenu qu’il souhaite voir.

Le Date Picker est un élément permettant de choisir une date. Nous l’utilisons dans la page de recherche pour définir la période dans laquelle nous cherchons un concert. Nous avons choisi ce composant pour la sélection de date parce qu’il nous semble être le plus adapté. Il est à la fois simple à utiliser et permet de bien faire comprendre le type d’information attendu de la part de l’utilisateur.

Les Rating bar sont utilisés pour que l’utilisateur donne son avis sur un concert. Tout comme pour le Date Picker, le Rating Bar est un élément qui doit permettre à l’utilisateur de très rapidement comprendre le type d’information que nous attendons de lui, tout en étant très simple à utiliser.

## Affichage de contenu à l’utilisateur

Les listes sont l’un des éléments les plus utilisé dans notre application et servent à montrer un certain nombre d’éléments d’un même type sur une même page. Ainsi l’utilisateur pourra voir tous les concerts de son historique sur la même page, ou tous les résultats de sa recherche sur la même page.

## Passage d’une page à une autre

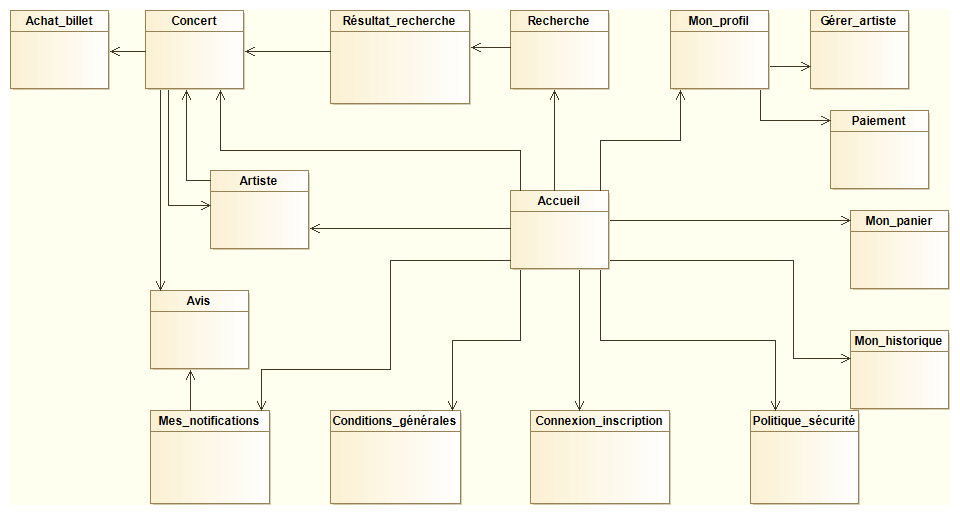
Nous avons utilisé un panneau latéral (Navigation drawer) parce qu’il s’agit de la plus simple méthode pour créer un menu. Nous avons ensuite repris ce menu dans toutes les autres activités à l’exception de la page de Connexion/inscription, puisque cela ne nous semblait pas nécessaire dans ce cas.

Tout comme le menu latéral gauche, le menu d’option situé en haut à droite permet de passer d’une page à l’autre en n’affichant qu’une très faible sélection des pages les plus visités par les utilisateurs. Il s’agit de la page de Connexion/inscription quand l’utilisateur n’est pas connecté et des pages Mon profil et Recherche quand il l’est.

Les boutons sont dans notre application l’une des méthodes utilisées afin de passer d’une page à une autre. La seule exception ici est dans le cas de la gestion d’artistes où le bouton permet de cesser de suivre un artiste mais sans changer de page.

Nous pouvons également noter que les listes font également partie de cette catégorie de composant puisque nous pouvons appuyer sur un élément d’une liste et être redirigé vers une autre page de l’application.

# Spécification de l’architecture logicielle de notre application



Nous voyons ici que la page d’accueil est le centre de notre application. En réalité, la plupart de ces liens sont accessibles par le panneau latéral gauche, qui est repris dans presque toutes les pages. Nous avons cependant choisi de ne pas représenter tous les liens en découlant afin de ne pas surcharger inutilement le diagramme.

Par souci de simplicité nous avons également omis de mettre les méthodes dans les classes. Il s’agit seulement de méthodes permettant de prendre en charge les évènements.

# Spécification de l’interface du noyau fonctionnel

Dans notre application, nous n’avons pour le moment réalisé que l’interface entre l’utilisateur est le système. Cela est la partie qui représente comment nous affichons des informations à l’utilisateur et comment nous récupérons des données de sa part (avec son consentement). Pour le moment, nous n’avons donc pas de partie de notre application qui montre comment nous traitons et stockons ces données ou comment nous réagissons aux évènements et aux entrées de l’utilisateur. C’est ce que nous détaillons dans cette partie du document.

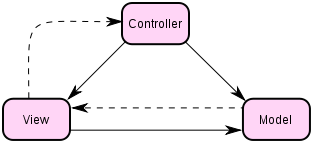
## Le langage

La totalité du noyau fonctionnel est également programmé avec le langage java. D’autres langages orientés-objets auraient pu être employés comme par exemple le C++, mais dans un souci de compatibilité et surtout de simplicité, nous restons sur l’utilisation du même langage à travers notre application.

## Le modèle d’architecture choisi : le MVC

La méthode qui nous semble la plus simple pour penser notre application est l’utilisation de l’architecture Modèle-Vue-Contrôleur ou MVC. Comme nous pouvons le voir sur le schéma suivant, ce modèle se compose de 3 parties : le(s) Modèle(s), les Vues et le(s) Contrôleur(s). Leur rôle peut être défini ainsi :

* Modèle : accède aux données et représente l’état de l’application.
* Vue : représente ce que l’utilisateur voit et avec lequel il interagit. Les vues s’inscrivent en observateur du/des modèle(s).
* Contrôleur : gère les évènements envoyés par les vues lors d’interactions avec l’utilisateur.



Source de l’image : [un très bon tutoriel](http://www.therealjoshua.com/wp-content/uploads/2011/11/mvc_diagram.png)

Nous pouvons donc voir que lorsque l’utilisateur effectue une action, l’évènement associé est repéré par la vue et envoyé au contrôleur qui le gère. Si besoin est, ce dernier va mettre à jour l’état du modèle. Cette modification est alors notifiée aux vues concernées (qui sont inscrites en observateur du modèle).

## La partie Modèle

En ce qui concerne les données, nous utilisons le concept de Data Access Object (DAO) afin de séparer le code permettant de récupérer les données sur la solution de persistance choisie (notre base de données). Les informations importantes relatives à l’état de l’application sont gardées en mémoire dans des attributs du modèle. Dans notre application c’est ici que nous gardons un booléen pour indiquer si l’utilisateur actuel est ou non connecté à l’application, ce qui se traduit par des permissions différentes pour l’accès aux fonctions personnalisées (profil, historique, notifications, panier et suivi d’artiste).

Il est important de noter que les DAOs sont des classes séparées de la classe principale du modèle, mais accessibles seulement par lui.

Nous devons donc créer une classe modèle qui va conserver l’état de l’application et une classe d’observation qui va suivre cette classe modèle pour être notifiée des changements et mettre à jour la vue associée.

## La partie Contrôleur

Le code du contrôleur est déjà en partie réalisé puisqu’il s’agit des méthodes permettant la gestion des évènements. Ces méthodes sont par exemple *onItemSelectedMenu*. Nous pouvons donc récupérer le code de nos activités pour le contrôleur et éventuellement en mutualiser une partie en réalisant un système d’héritage.

## La partie Vue

Nous avons déjà créé en partie la partie Vue de notre application. Cependant, pour effectivement rejoindre l’architecture du MVC, nous devons séparer les vues du code du contrôleur. Pour cela, nous créons des classes héritant des layouts utilisés avec par exemple :

**public** **class** ExampleView **extends** RelativeLayout

## Les framework et librairies Android utilisés

Pour la réalisation de cette application, nous n’utilisons pas de framework particulier. En ce qui concerne les librairies Android utilisées, nous nous servons de la SDK Android version 24. L’ajout d’un noyau fonctionnel pourrait modifier cette partie, mais nous sommes trop peu avancés dans la conception de celui-ci pour pouvoir l’affirmer ou l’infirmer.

## Partie du noyau fonctionnel déjà réalisée

Dans l’état actuel du code, la seule partie du noyau fonctionnel est réalisée dans les contrôleurs et sert à la gestion d’évènements pour assurer la transition entre les pages.

# Librairies externes utilisées

Dans l’état actuel de notre application, nous n’utilisons pas de librairies externes à la SDK officielle Android version 24. De plus, nous avons créé notre application pour la version 6.0 d’Android et n’avons donc pas de compatibilité garantie avec les versions antérieures.