01/03/2015

Android Avancé

Simon Jacquemin & Olivia Ronot

# [RAPPORT – MEDIA VIEW]

Ce rapport pour l'application MediaView développé dans le cadre du cours d'Android Avancé est un post mortem des différentes phases de développement. De la conception au résultat final.

# Contenu

# Introduction

Lors de la réalisation de ce projet d'Android Avancé, nous avons travaillé la plupart du temps en autonomie. Cela nous a permis de réfléchir à une conception qui nous était propre. Mais nous avons eu un problème au niveau du cahier des charges.

En effet nous avons mal compris les spécifications initiales. Nous sommes donc partis sur une conception biaisée de l'application puisque nous n'avions pas compris les contraintes du client.

Toutes les explications qui vont être présentées dans ce document doivent donc être lues en gardant ce fait en tête. Nous avons essayé de rectifier les choses lors des derniers jours du projet. Toutes les modifications qui ont été faites dans ce sens sont dans la partie "Mauvaise compréhension de la consigne".

# Schémas de l'application

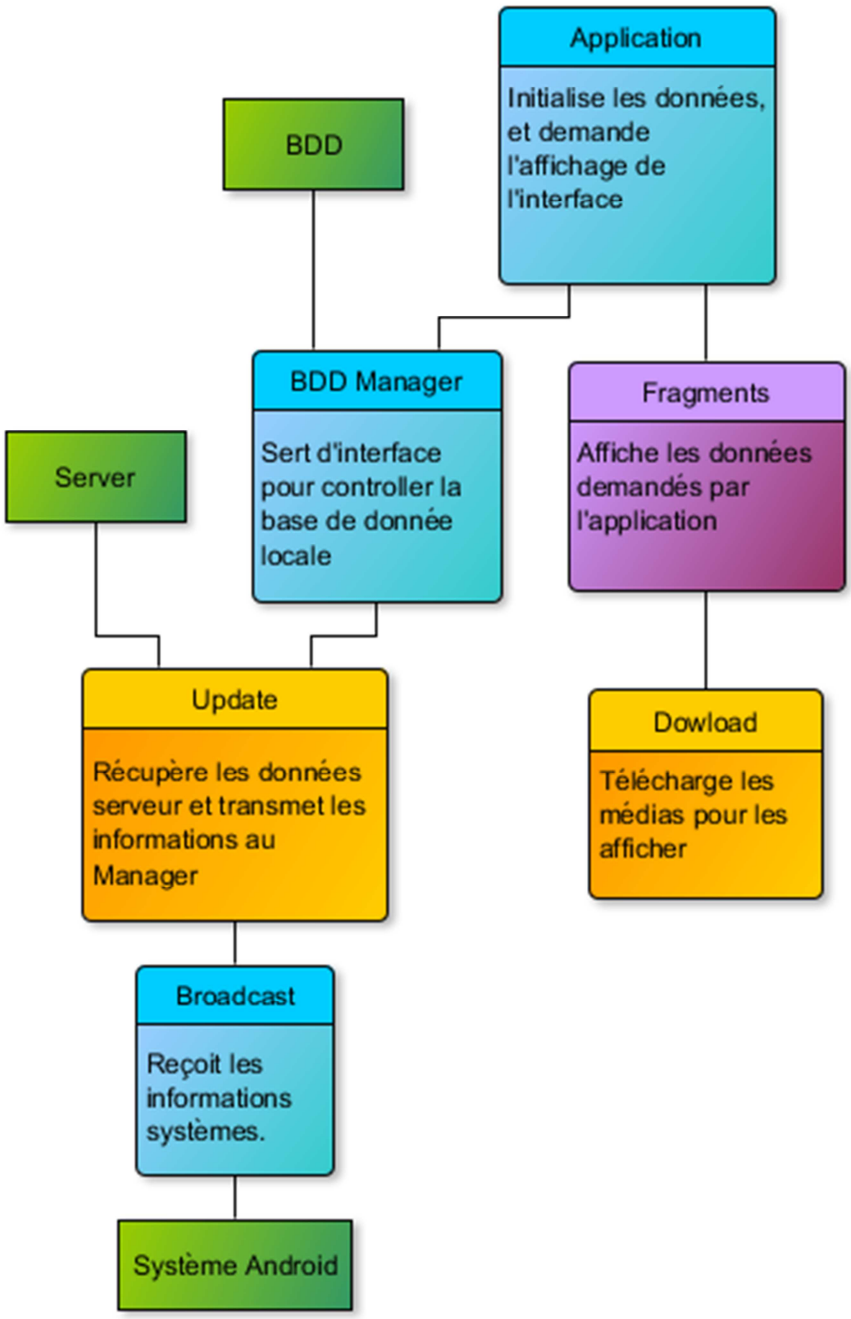
Le schéma de conception et celui de la réalisation finale sont assez proches dans les grandes lignes.

En effet, nous avons gardé le système avec un objet qui gère la base de donnée (BDD Manager) qui lui-même est utilisé dans l'affichage par l'intermédiaire du cœur de l'application elle-même. Elle est mise à jour grâce à un système de broadcast et service (Voir la partie Compréhension des services) afin d'avoir une gestion non-asynchrone des évènements. Enfin, toute l'interface est gérée grâce à un système de fragment (Voir la partie Fragments), qui donne plus de souplesse à l'application et à son utilisation.

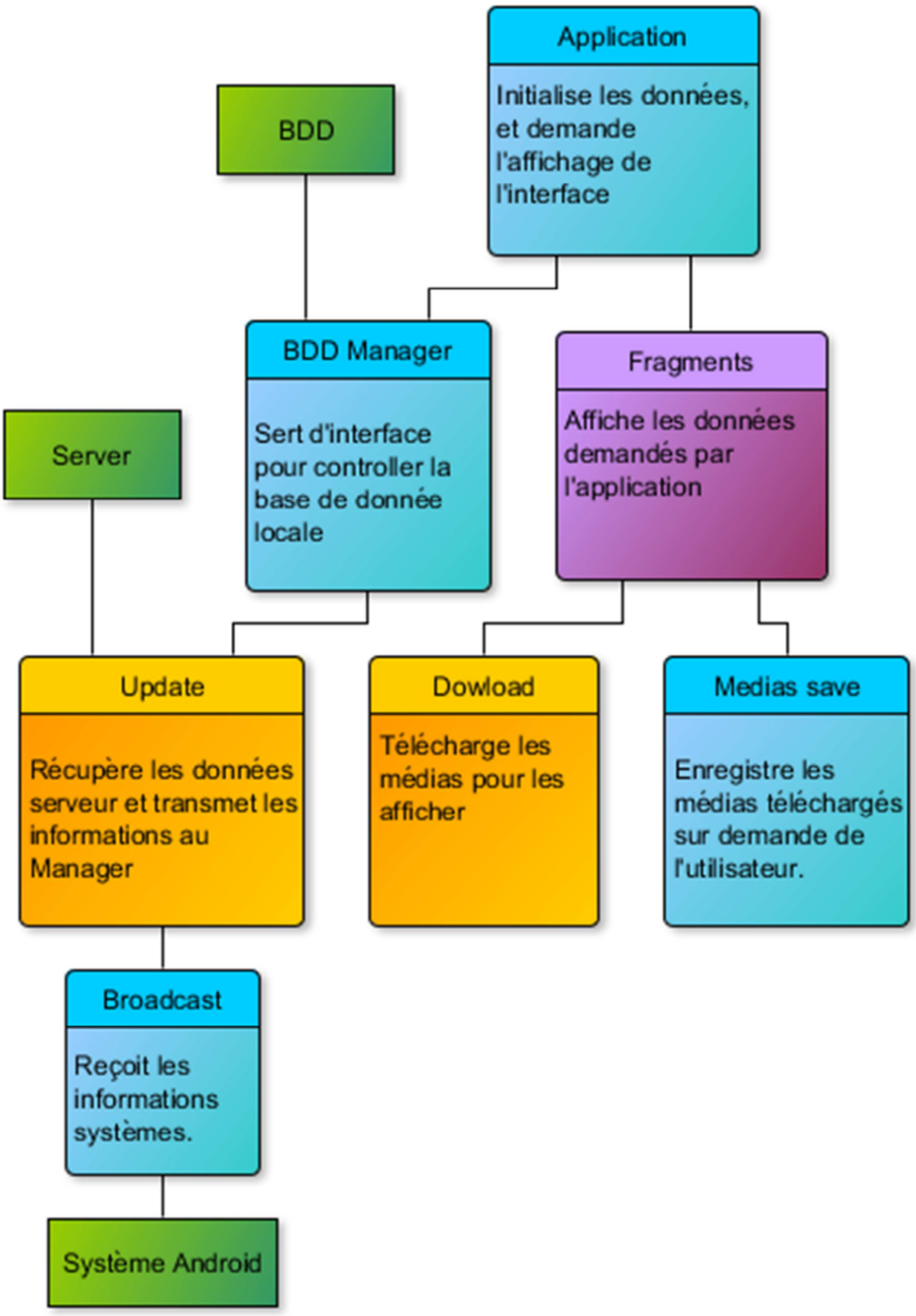
Voici la légende des différents schémas :

- **Cyan** : Modules de gestion des données
- **Vert** : Services externes, que çà soit à l'intérieur du téléphone ou sur Internet
- **Jaune** : Modules des fonctionnalités reliées au réseau
- **Violet** : Interfaces et tout ce qui est de la gestion de l'affichage

## Schéma de conception



## Schéma de l'application finale



# Développement

Dans cette partie, nous allons parler de tous les éléments qui nous ont semblé importants durant la phase de développement de l'application. Que ce soit des problèmes et leurs solutions apportées ou bien des éléments nouveaux et librairies que nous avons utilisés. Les différentes parties ne sont pas classées par ordre chronologique mais plutôt en fonction les affinités de chacun.

## Multi-support

La contrainte du multi-support était de trouver un système qui aurait permis à l'application de s'afficher différemment sur chaque type d'écran. Avoir un affichage double sur les grands écrans de tablettes et un affichage simple avec changement d'interface en fonction de la sélection du média sur petit écran.

Pour résoudre cette contrainte, plusieurs solutions ont été abordées et nous sommes passés par plusieurs tests. Mais ce que nous avons retenu a été un système utilisant la gestion natif d'Android des fichiers en fonction de différents paramètres.

En effet, Android permet de gérer les différentes configurations grâce aux noms de fichiers présents dans l'arborescence du projet. Nous avons donc tout simplement fait un fichier de configuration qui contient un booléen et une fois ce fichier placé dans chaque dossier cela nous a permis de savoir sur quel type de plate-forme on se trouve puisque d'Android va naturellement chercher le bon répertoire. Et donc grâce à cette information nous avons pu afficher différemment les informations.

Nous avons utilisé les arguments suivant pour différencier les types de périphériques :

- Small
- Normal
- Large
- Xlarge

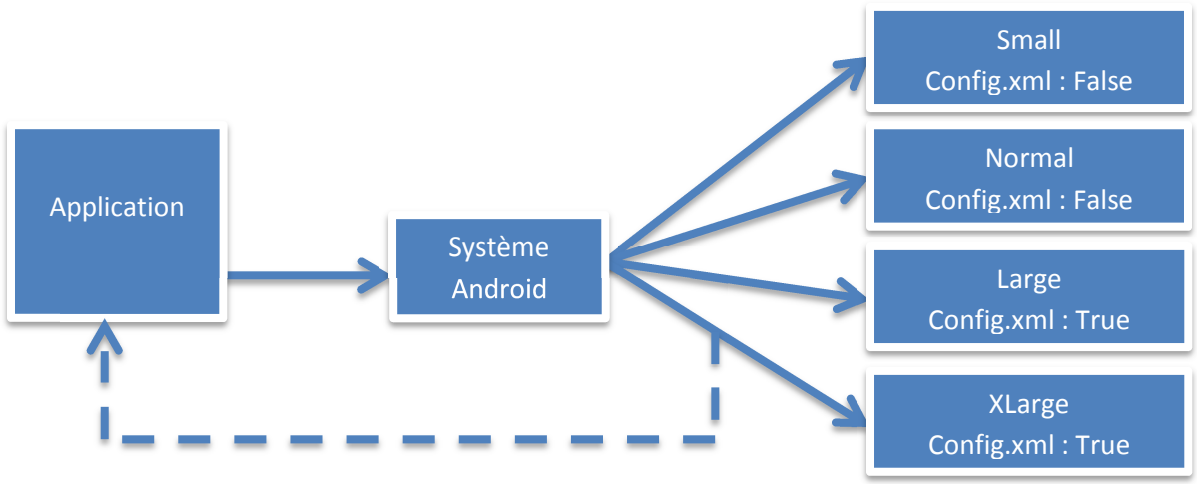



*Schéma du système de configuration natif par Android*

## Fragments

CLa gestion d'une interface grâce aux fragments a été une vraie découverte pour un des membres du binôme. Nous avons été amenés à cette utilisation grâce à la consigne initiale qui a voulu que l'on utilise un Naviguation Drawer, composant natif d'Android pour faire des barres latérales de menu.

Nous sommes donc tout naturellement partis sur une architecture avec une activité principale qui charge et décharge des fragments en fonction des besoins de l'utilisateur. Cette partie a été très en lien avec la gestion du multi-support puisque l'activité a dut être pensée pour afficher le bon nombre de fragments et surtout le bon type.

Grâce au fragment, une partie de l'interface peut être déportée dans l'activité afin de ne pas dupliquer des éléments pour rien. Les interactions entre l'activité et les différents fragments se font grâce à un système de remontée d'information à travers des évènements écoutés par le conteneur.


*Force des fragments : Les faires s'adapter à la taille de l'écran*

## Gestion de la BDD

*SQLiteOpenHelper*

L'application nécessitait l'utilisation d'une base de données pour gérer les différentes informations des médias. Android possédait déjà un outil très efficace : SQLiteOpenHelper. Nous avons donc décidé de l'utiliser pour notre base de données.

Cette librairie utilitaire permet l'utilisation simple et rapide d'une base de données. Elle donne la possibilité de gérer la connexion, la création, et l'évolution d'une base de données locale.

Grâce à ces fonctionnalités, nous avons pu créer et avoir une base de données fonctionnelle très rapidement. Ce qui ne nous a pas coincés dans le développement des autres fonctionnalités de l'application.

Nous avons choisi cette librairie parce qu'Olivia la connaissait puisqu'elle l'a utilisé dans son travail à de nombreuses reprises. Ce qui nous a permis d'aller très vite sur le déploiement des fonctions. De plus, elle est disponible nativement dans Android, nous n'avons donc pas dû rechercher une documentation sur un site externe. Tout était disponible sur le site officiel Android Developer.

## Gestions des médias

Dans les contraintes de développement de l'application, nous avions plusieurs types de médias à gérer. Nous avons utilisé des techniques particulières afin de gérer les différents types de médias.

### Texte

Le texte a été récupéré de manière "traditionnelle". C'est à dire que nous avons récupéré le flux du fichier et l'avons affiché directement dans le TextView.

### Son & Vidéo

Le son, comme la vidéo, devait être géré en streaming. La fonctionnalité a été implémentée mais nous avons manqué de support pour les tests. Nous ne sommes donc pas allés au bout de ce que nous voulions faire et nous avons préféré concentrer nos efforts sur d'autres fonctionnalités.

### Image

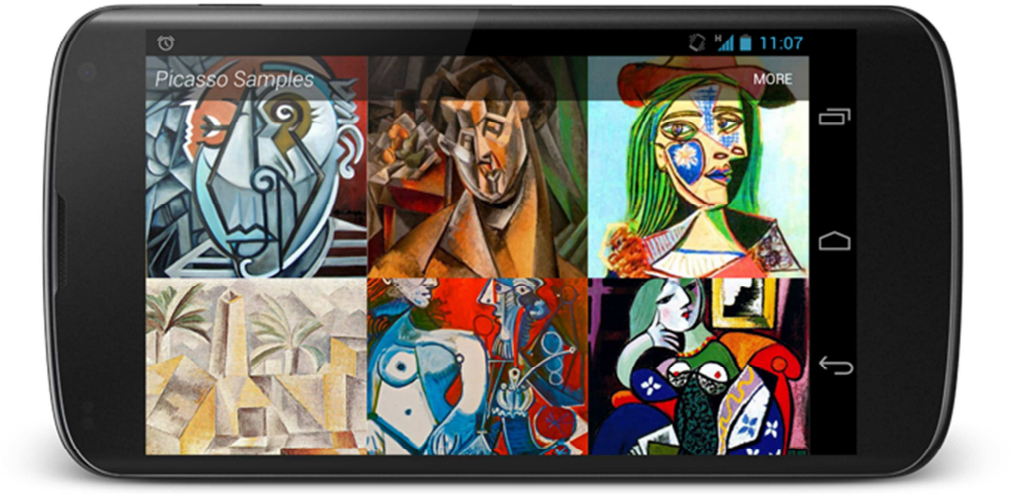
Enfin, pour les images nous avons décidé d'utiliser une librairie : Picasso

C'est une librairie utilisée pour le chargement et l'affichage des images. Elle gère le cycle de recyclage des ImageView ainsi que la mémoire et le cache de l'appareil. Son plus gros point fort est sa simplicité d'utilisation. En 2 ou 3 lignes, il est facile de gérer le chargement d'une image en asynchrone avec une gestion du cache avec une image de chargement.

Elle permet aussi de charger des images directement depuis un fichier présent sur l'appareil, dans les ressources de l'application ou depuis le réseau. C'est pour cette raison que nous l'avons choisie. De plus, Olivia l'ayant souvent employée dans son entreprise, son utilisation en a été facilitée.

Dans la compréhension initiale du cahier des charges, il ne fallait pas télécharger et enregistrer localement les images avant de les afficher. Cette librairie semblait donc idéale dans notre cas.

Malgré tous, nous avons décidé de la garder puisque nous avons trouvé un moyen de conserver les images en local depuis une ImageView (Voir la partie "Mauvaise compréhension de la consigne").




*Interface de base d'une galerie utilisant Picasso*

## Compréhension des services

Les Broadcasts et services nous ont un peu posé problème pour comprendre leur fonctionnement. Nous avons eu du mal à saisir le fait que le Broadcast pouvait recevoir des événements systèmes, sans que l'application soit lancée et même si son processus a été arrêté.

En pratique, le système envoie des événements aux Broadcasts s'étant enregistrés auprès de lui. Ceux-ci déclarent un ou plusieurs "intent-filter" de manière à ne recevoir que les types d'événement désirés. Ensuite, le Broadcast exécute les actions nécessaires. Dans le cas de notre application, un service gérant le téléchargement des informations relatives aux médias.

Une fois que nous avons assimilé le fonctionnement, nous avons su en tirer parti afin d'avoir une mise à jour des données invisible pour l'utilisateur. C'est un outil très puissant que nous remettrons sûrement en place dans nos applications futures.

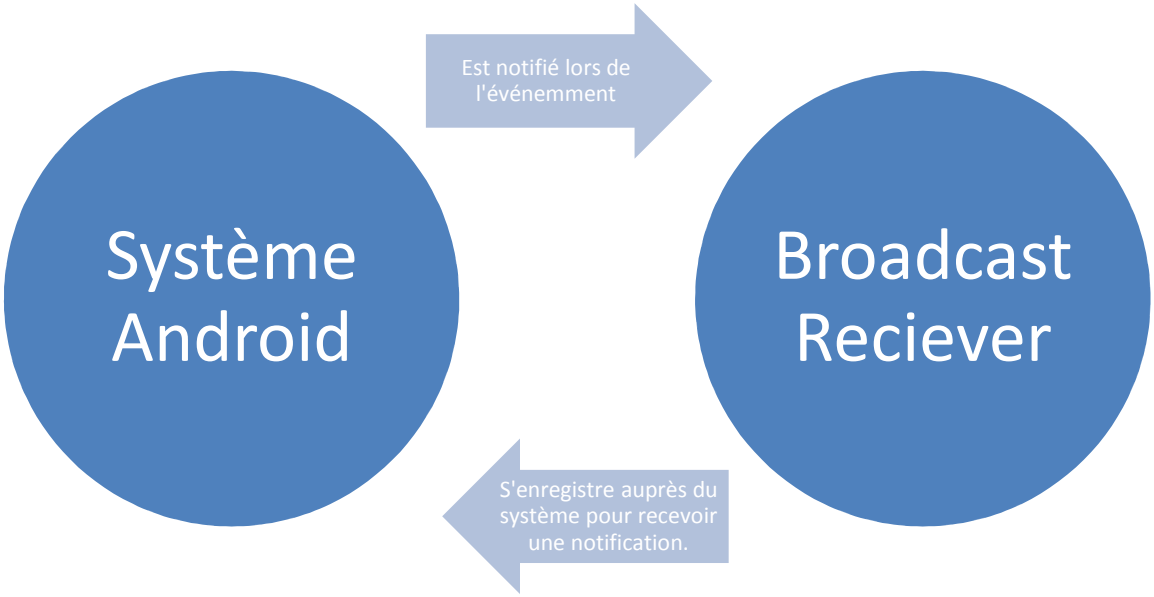


*Schéma de fonctionnement des broadcast/services*

## Mauvaise compréhension de la consigne

### Sujets compris

Le sujet était de mettre en place une application permettant le téléchargement d'un certain nombre de médias, par demande de l'utilisateur, puis de les afficher. Nous avons mis en place une application qui permet de visualiser les médias sans passer par le téléchargement intentionnel de l'utilisateur.

Cela implique que le téléchargement se faisait sans que l'utilisateur le demande, et que le média n'était pas non plus conservé sur l'appareil. Cela excluait donc toute utilisation offline de l'application. Mais cela permettait à l'utilisateur de visualiser une bibliothèque de fichiers stockés sur un serveur, comme le ferait tout service de visualisation de bibliothèque de fichiers stockés sur le Cloud.

### Solution pour rectifier

La solution que nous avons mise en place consiste à mêler les deux visions de l'application : la nôtre et celle du sujet initial. Bien sûr cela ne nous fait pas rentrer dans les clous du cahier des charges, nous avons tendu à nous rapprocher le plus possible vers les demandes clients.

Nous avons mis en place la possibilité de télécharger le média après l'avoir visualisé.

Pour cela nous avons développé des techniques spécifiques à chaque média qui nous ont permis de simuler l'effet de téléchargement en local :

- Le texte a été enregistré en l'écrivant dans un fichier.
- L'image étant téléchargée par une librairie externe (Picasso) qui ne propose pas de conserver le fichier, il était difficile de la récupérer de cette manière. Nous avons donc eu recours à l'ImageView, le composant qui permet d'afficher l'image. Grâce à cela nous avons extrait l'image affichée et nous l'avons conservé dans un fichier local.
- L'audio comme la vidéo n'ont pas été traités par manque de temps.

# Résultat final

Globalement, nous avons plutôt réalisé une application fonctionnelle. Toutes les fonctionnalités que nous avions imaginés sont présentes et fonctionnent bien. Sur ce point, nous sommes donc plutôt contents.

Malgré tout, nous n'avons effectivement pas respecté le cahier des charges initial et nous n'avons donc pas réalisé les objectifs de la mission et pour cela l'exercice a été un échec au strict point de vue du projet.

Nous n'en tirons pas que de mauvaises choses et tout le travail fournit pour en arriver là ne peut pas rester juste comme étant un échec. En effet nous avons appris ou révisé de nombreux éléments d'Android et cela ne peut être qu'être bénéfique pour notre travail futur.

La leçon du projet serait de bien vérifier ce qui est demandé, et de faire une conception nette avec le client. Ne pas partir tout seul dans son coin. Même si nous le savons très bien, cela nous a permis de

nous rappeler que même si nous connaissons bien la technologie, il ne faut pas faire ce que l'on veut mais ce que le client veut faire. Et il y a une grande différence.

## Amélioration possible

### Nettoyage du code

Lors de nos modifications pour rendre le projet conforme au cahier des charges, nous avons fait beaucoup de changements de dernière minute dans le code. Or, ces changements ne sont pas toujours bien intégrés au reste, ne collent pas à la conception de base ou n'utilisent pas le design que nous avions décidé d'utiliser.

Si nous avions du temps et de l'énergie à remettre dans ce projet, nous pourrions facilement corriger beaucoup d'éléments qui auraient permis au moins d'avoir un code propre. Parce que nous avons mis beaucoup d'énergie à se fixer des règles afin d'avoir quelque chose qui, même si les fonctionnalités ne sont pas au rendez-vous, soit facile à reprendre par quelqu'un d'autre.

### Gestion de tous les type de médias

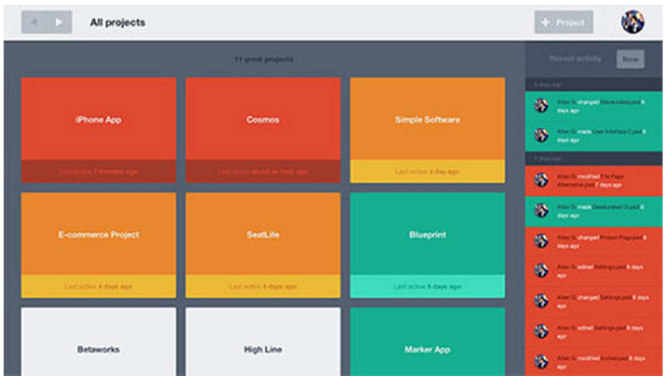
La partie de la gestion des médias est arrivée assez tard dans le développement et malgré tous nos efforts, nous n'avons pas réussi à gérer tous les types prévus dans le cahier des charges (Voir la partie Gestion des médias).

Nous avons appris à la fin du projet qu'un bout de code permettait de gérer tous les types assez facilement et automatiquement. Malheureusement, avec tous les autres problèmes qui nous sont tombés dessus à la fin, nous n'avons pas eu le temps d'implémenter cette méthode. Cela aurait pourtant été très intéressant pour un investissement peu coûteux en temps et en efforts.

### Nouveau design

Nous nous réservions la partie design pour la fin puisque cette partie semblait être la cerise sur le gâteau et nous voulions tous les deux la faire. Nous nous sommes dit que nous allions attendre d'avoir fini nos parties respectives.

Malheureusement ce moment n'est jamais arrivé et nous n'avons donc pas eu le temps de designer une interface propre et sympathique pour l'utilisateur. Cela est pour nous deux, un grand regret car nous avions tous les deux des idées sur les interfaces utilisateurs que nous voulions mettre en place. Nous voulions établir une charte graphique et nous y tenir afin d'avoir une réelle identité visuelle.




*Exemple d'identité visuelle forte pensé pour l'application MediaView : Le flat design*