

Jonathan Racaud

Music Sheet Writer

Music Sheet Writer

Documentation Technique

Objectifs du document

Résumé

Glossaire

***– M –***

Mot : Définition

***– S –***

Signification de l’Acronyme (SDA) : Définition

Description du document

|  |  |
| --- | --- |
| Titre | Music Sheet Writer : Documentation Technique |
| Date de création | 08/04/2016 |
| Date de publication | 08/04/2016 |
| Auteur | Jonathan Racaud |
| Responsable | Jonathan Racaud |
| E-mail | musicsheetwriter\_2017@labeip.epitech.eu |
| Sujet | Documentation Technique |
| Version du modèle | 1.0 |

Tableau des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Auteur | Section(s) | Commentaire |
| 04/08/2016 | Jonathan Racaud | Toutes | Création du document |
| 19/04/16 | Antoine Simon | 3.4 Application iOS | Rédaction |
| 20/04/16 | Antoine Simon | 3.4 Application IOS | Correction erreur en tout genre |
| 21/04/16 | Antoine Simon | 3.4 Application IOS | Rédaction sur les xassets |

Sommaire

[1. Logiciel 1](#_Toc447872840)

[2. Site Web 1](#_Toc447872841)

[2.1. API Rest 1](#_Toc447872842)

[2.2. Site vitrine et espace communautaire 1](#_Toc447872843)

[3. Application Android 1](#_Toc447872844)

[4. Application iOS 1](#_Toc447872845)

[5. Application Windows Phone 1](#_Toc447872846)

Liste des Tableaux

**Aucune entrée de table d'illustration n'a été trouvée.**

Liste des Figures

**Aucune entrée de table d'illustration n'a été trouvée.**

# Introduction

# Le projet

Diagrammes de haut niveau, les différents livrables (UML)

# Les livrables

## Logiciel

### Norme de codage

### Implémentation

#### Gestion de projet

#### Lecture de partition

#### Edition de partition

### Tests

### Historique des modifications

### Problèmes existant

## Site Web

### API Rest

### Site vitrine et espace communautaire

## Application Android

## Application iOS

### Organisation du projet

Le projet est organisé selon un système simple, tous les fichiers « .m » et « .h » communs sont regroupés dans un dossier portant leur nom ou fonction par rapport à l'application, par exemple test.m et test.h seront dans le dossier test. Tous les dossiers sont à la racine du projet. Il peut y avoir certaines fois plusieurs fichiers dans le même dossier si jamais ils sont relativement semblable ou si jamais ils fonctionnent ensemble comme par exemple le dossier test connexion contient une classe pour l’affichage de l’erreur et une autre pour les tests de connexion. La langue utilisée dans les noms des dossiers ou des fichiers n'est pas fixe, la seule règle est qu'ils doivent être clairs et précis. L’application est composé de plusieurs composant donc, qui sont approfondi dans le AA2.

Chaque méthode est formatée selon le prototype suivant : VoiciUneMethodeDeClasse. Cette règle de nomination est générale et est utilisée par Apple. Tous les commentaires visant à expliquer une méthode doivent être placés au-dessus de la méthode et l'explication d'une ligne de code doit être au-dessus de cette dernière.

De plus, aucune mise en place de l’application sur l’App Store ne doit contenir de fonction écrivant sur une sortie, donc aucun NSLOG(), et aucune fonction désuète (noté : deprecated).

Un autre point clef de ce projet est la gestion des assets ; chaque image, logo ou tout autre chose visuel extérieur venant à être intégrée au projet devront être placée dans le dossier Xcassets directement par glissé-déposé au travers de XCode. Ce dossier permet une classification claire des images et par la même occasion permet en cas de changement de ne le faire qu’à cette endroit-là et non à chaque fois que l’image est affiché. Il viendra aussi gérer les différentes tailles selon l’appareil.

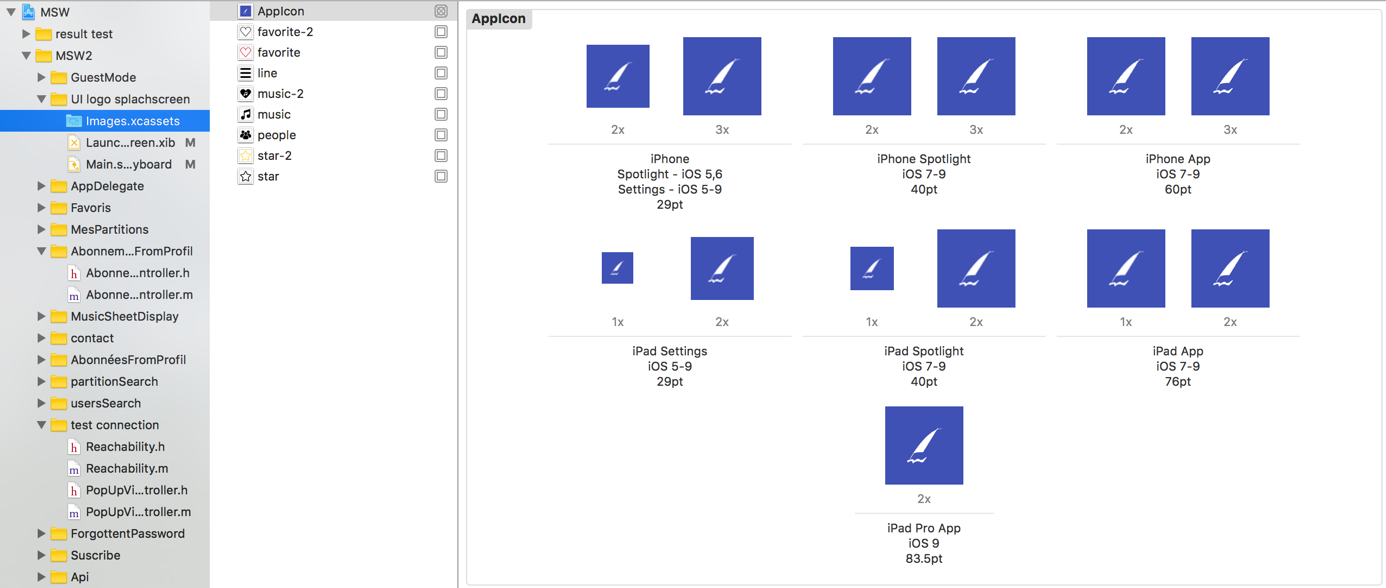


Figure - Dossier xcasset

### Installation et démarrage du projet

Pour pouvoir démarrer le projet il faut installer XCode. Seul ce composant suffit et le restant est directement dans le dossier IPhone du dépôt SVN. Pour le processus d’installation de l’application vous pouvez vous référer au document ID1.

Une fois XCode installé, il suffit donc de lancer le fichier «.workspace » et à partir de là il est possible de le lancer sur n’importe quel émulateur.



En cliquant sur le label situé après MSW 2, vous pourrez changer le type de simulateur disponible. Ensuite le projet se lance en cliquant sur RUN.

### Démarrage et gestion des tests

Grâce à la librairie KIF, des tests de l'interface peuvent être faits, mais Xcode intègre déjà un système pour tester le reste(Voir le TS). Pour pouvoir gérer, modifier les tests il suffit de se rendre dans le dossier MSW2Test, ensuite les tests UI sont dans le dossier testUI et les tests du code dans le dossier test logique. Pour lancer les tests il suffit d'aller dans le « test navigator » et de clique sur le petit symbole « play » à côté du nom du fichier pour lancer tous les tests ou sinon de cliquer sur le symbole devant la fonction de test pour n'en lancer qu'une.

Attention, si jamais un seul est lancé, vérifier qu'il ne requiert aucun lancement avant comme par exemple le test de déconnexion ne pourra marcher si le test de connexion n'est pas lancé, il faut donc faire attention au dépendances de certains tests.

### Compréhension général des vues

L’application IOS est gérée dans un système de workspace, il y a donc plusieurs projets en un, exactement deux. Pour la mise en place des tests, l’outil CocoaPods a été installé. Il est donc possible aujourd’hui en modifiant le Podfile d’ajouter n’importe quelle librairie au projet. Pour cela il suffit de remplir le fichier Podfile, pour mieux comprendre il suffit de suivre la procédure d’installation d’une libraire via Podfile. <https://guides.cocoapods.org/syntax/podfile.html>

Une fois le workspace lancé, le projet est donc accessible. Le cœur du projet se trouve dans le fichier MainStoryBoard qui permet la gestion de toutes les vues mais aussi des liens entre elles. Grâce à ce fichier, on peut modifier toutes les informations de l’application.

Chaque vue est donc liée à une classe, soit native à IOS soit coder dans le projet, cela permet donc de pouvoir faire hériter une classe afin de pouvoir en changer son comportement pour certaines spécifications. Pour pouvoir donc lier une classe personnelle à une vue il suffit de rentrer son nom dans « Custom class ». De plus, certains liens entre les vues permettent l’échange de données entre les classes, c’est pour cela que certains liens ont des identifiants.

Certaines vues ont aussi des identifiants dans accesibility Label, ce genre de label est utilisé pour que des éléments extérieurs puissent trouver un élément de l’interface comme par exemple KIF lors des tests.

Le projet a été développé pour pouvoir fonctionner sur plusieurs IPhone, il faut donc, dans la création de vue place des layouts et les tester pour être sur du bon fonctionnement sur le reste des appareils.

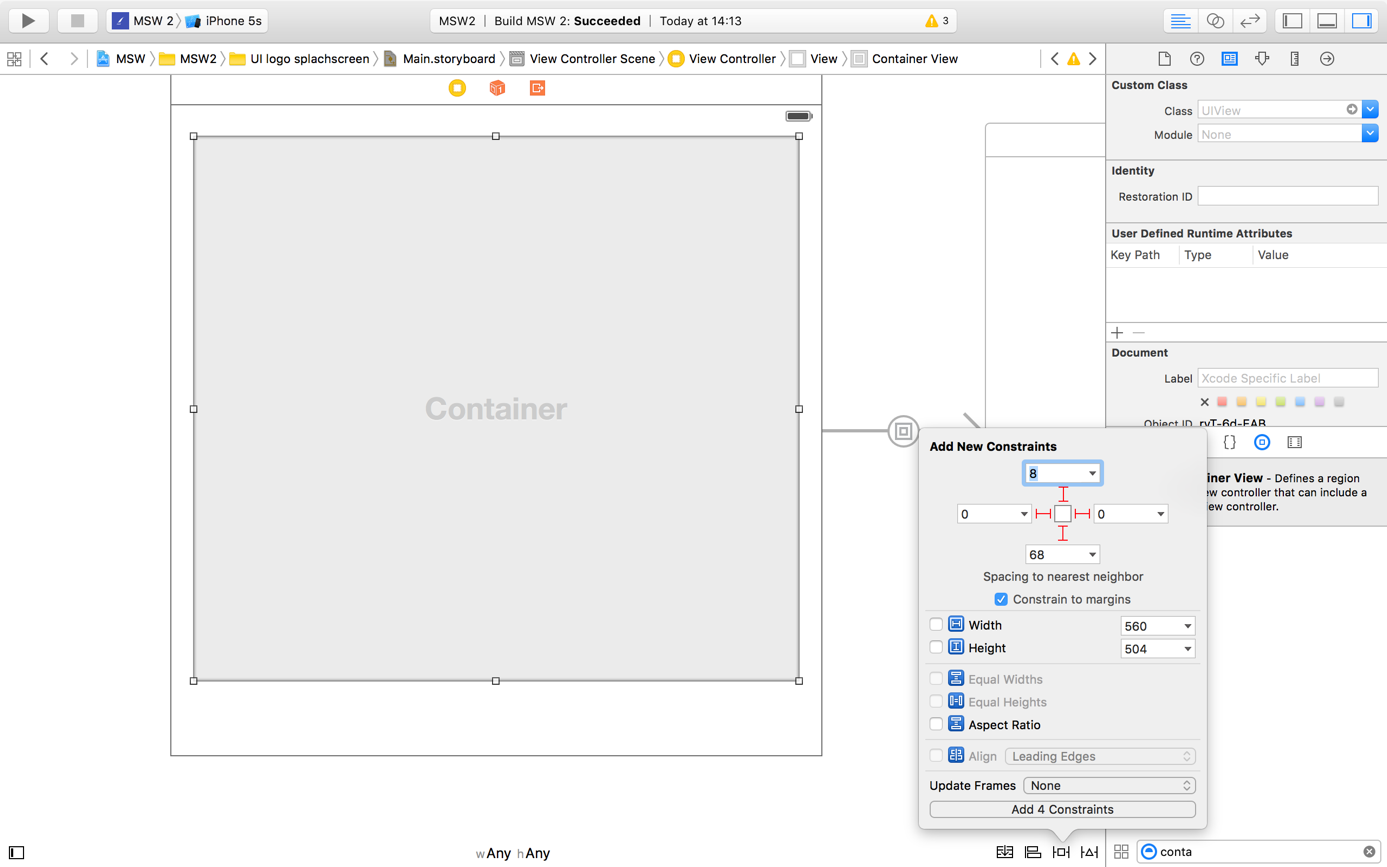
### Création de vue

Les vues entant le cœur du projet c’est aussi un point complexe mais surtout sensible par le fait que toute erreur peut avoir un large impact et qu’énormément de chose sont possible via l’interface d’XCode.

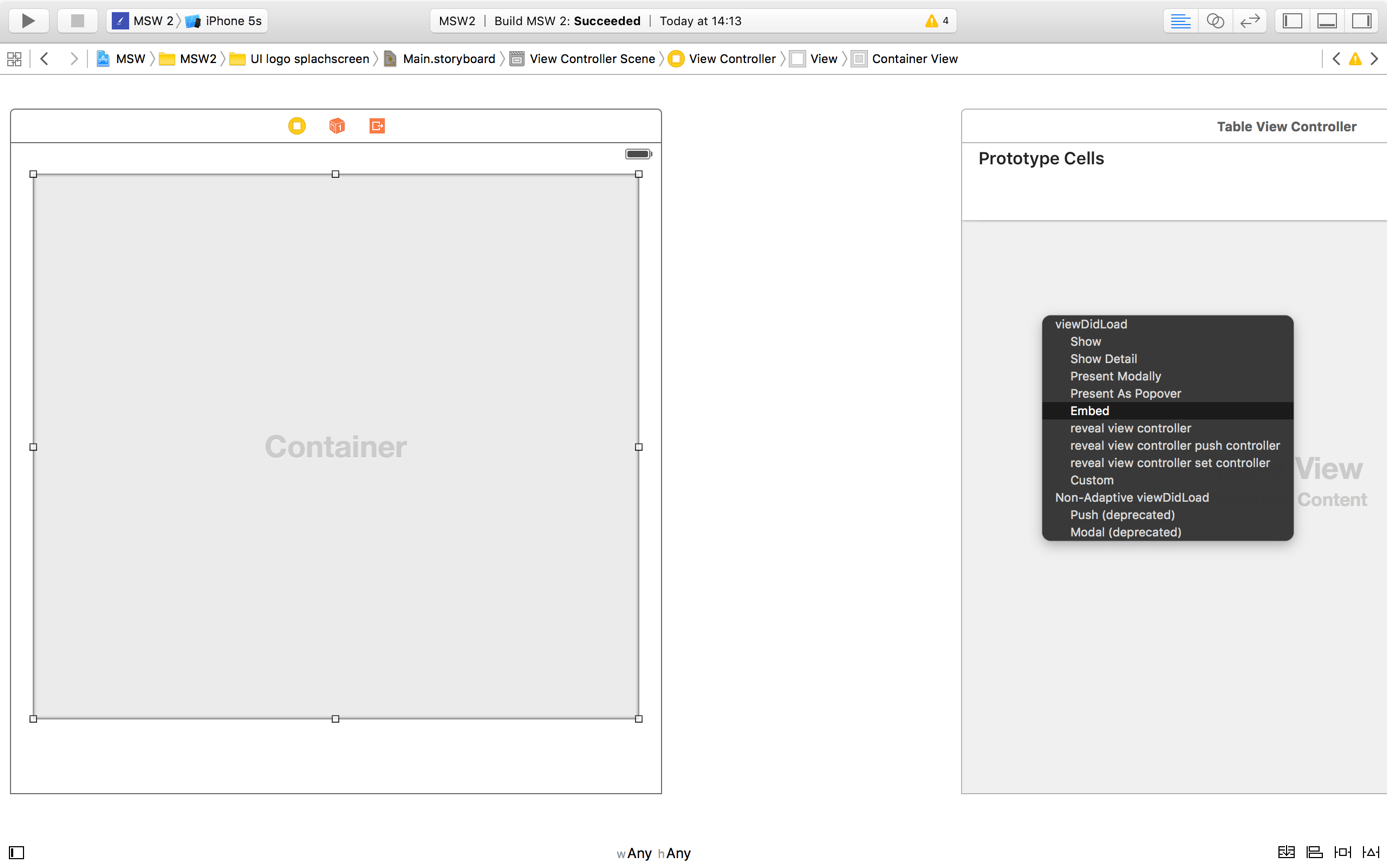
Voici dans un exemple pour une vue simple avec un embed de tableViewController.

Un embed est un lien qui permet l’affichage d’un view controller dans un view controller et non un view dans un view controller.

Tout d’abord il faut placer la vue, une fois placé, vous pouvez placer à l’intérieur un container view.



Vous pouvez ensuite créer le layout pour éviter que le container ne se place n’importe où dans la vue. Un view controller a été créer, supprimez-le pour cet exemple.



Il faut ensuite placer un table view controller et créer un lien (ctrl + clique gauche) entre le container et le table view, il faut ensuite choisir embed.

À partir de là, une vue vient d’être créé avec à l’intérieur une table view. L’intérêt de faire une embed est que on peut quand même créer un table view CONTROLLER alors que si l’on place un simple table view dans la vue il pourrait y avoir des problèmes si la gestion devient complexe. Donc grâce à un lien embed on peut gérer un view controller dans un autre view controller, le point important est donc d’avoir ces controller et non une simple vue.

Vous pouvez ensuite lié une classe à la vue, un moyen simple de lier les composants de l’interface à la classe est d’utiliser l’assistant.

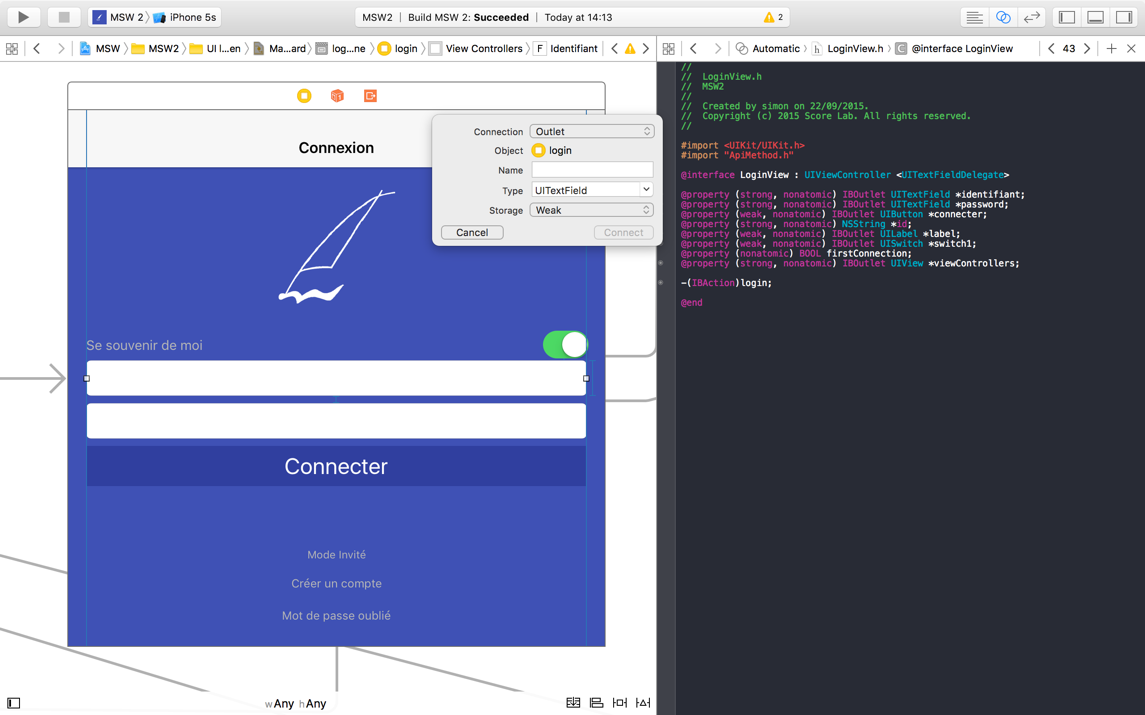


Figure - Mode assistant

Une nouvelle fois en utilisant ctrl et clique gauche des liens vont pouvoir être créer entre le .h et l’interface.

Cette procédure a été utilisé pour toutes les créations de vue dans le projet.

### Problèmes et axes d’amélioration

Un de problème pouvant être rencontrer et le bloque lors d’un requête HTTPS. L’application utilise la library AFNetworking qui parfois sans raison apparent ne réagis plus à une requête puis bloque pendant plusieurs secondes. Un moyen de régler ce genre de problème serait de remplacer le code utilisant Afnetworking dans le fichier ApiMethod.m par les méthodes natives a IOS. Pour avoir un exemple de comment elles fonctionnent, il faut reprendre les méthodes utilisées pour l’envoi de photo de profil.

Un des axes d’amélioration pourrait être de toujours améliorer l’expériences utilisateurs en rajoutant par exemple des chargements s’il en manque et aussi d’améliorer la vitesse sur la vue MonCompte en allégeant le code.

# WP

## Application Windows Phone

# Gestion des sources et bug

## SVN

## Git

## Bugtracker

# Annexes

* API
* AA2
* TS
* Autres ressources