

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 |
| 08/04/2016 | Jonathan Racaud | Toutes | Création du document |
| 26/04/2016 | Jonathan Racaud | 2 Projet  3.2.1 Introduction  3.2.2 AppBundle | Rédaction des sections et ajout des illustrations |
| 28/04/2016 | Jonathan Racaud | 3.2.3 MSWBundle | Rédaction de la section et ajout des illustrations |

Sommaire

[1. Introduction 1](#_Toc449452253)

[2. Le projet 2](#_Toc449452254)

[3. Les livrables 3](#_Toc449452255)

[3.1. Logiciel 3](#_Toc449452256)

[3.2. Serveur 3](#_Toc449452257)

[3.2.1. Introduction 3](#_Toc449452258)

[3.2.2. AppBundle 4](#_Toc449452259)

[3.2.3. MSWBundle 8](#_Toc449452260)

[3.3. Application Android 8](#_Toc449452261)

[3.4. Application iOS 8](#_Toc449452262)

[3.5. Application Windows Phone 8](#_Toc449452263)

[4. Gestion des sources et bug 8](#_Toc449452264)

[4.1. SVN 8](#_Toc449452265)

[4.2. Git 8](#_Toc449452266)

[4.3. Bugtracker 8](#_Toc449452267)

[5. Annexes 8](#_Toc449452268)

Liste des Tableaux

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

Liste des Figures

[Figure 1. Architecture de Music Sheet Writer 2](#_Toc449452247)

[Figure 2. Architecture serveur 3](#_Toc449452248)

[Figure 3. Architecture AppBundle 4](#_Toc449452249)

[Figure 4. Exemple fonctionnalités Login – définition 5](#_Toc449452250)

[Figure 5. Exemple fonctionnalité login – implémentation 6](#_Toc449452251)

[Figure 6. Exemple fonctionnalité login - implémentation service User 7](#_Toc449452252)

# Introduction

# Le projet

Le projet Music Sheet Writer est composé de plusieurs sous-projets ou livrables ayant tous un lien entre eux.

Le diagramme suivant présente les différents livrables et leurs relations.

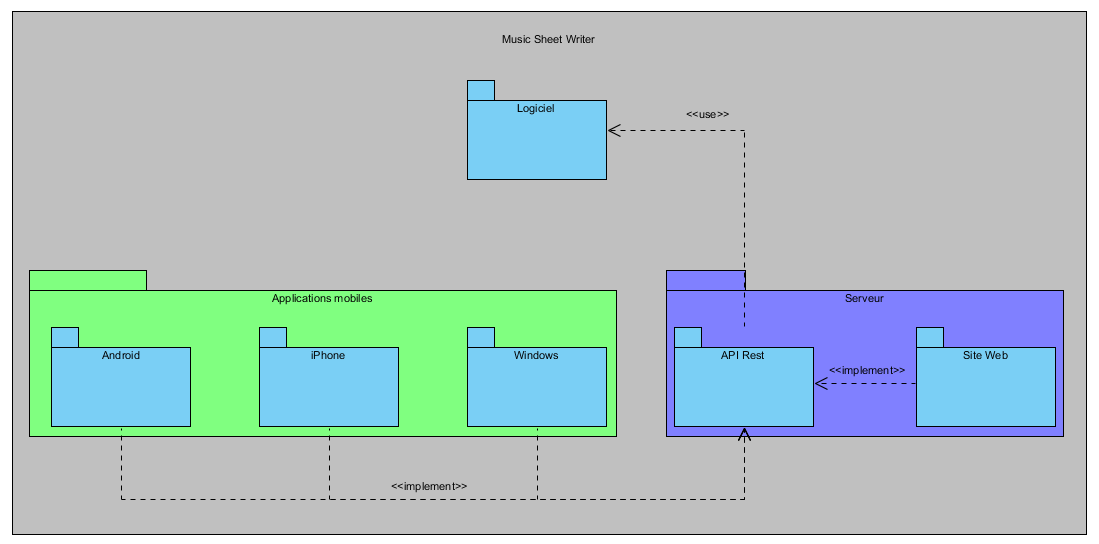


Figure 1. Architecture de Music Sheet Writer

Chacune des applications mobiles implémente les fonctionnalités données par le composant **API Rest** du serveur. Il en est de même pour le composant **Site Web** du serveur. De ce fait, toutes ces parties du projet sont dépendantes du composant **API Rest** qui est un des points les plus critiques du projet.

Bien que globalement le logiciel d’édition de partition soit indépendant des autres livrables, il existe une dépendance entre l’**API Rest** et ce dernier et concerne les partitions générées par le **Logiciel**. Cette dépendance est expliquée plus en détail dans la partie [**3.2.1 API Rest**](#_API_Rest).

# Les livrables

## Logiciel

## Serveur

### Introduction

La figure 1 de l’introduction représentait une version simplifiée de l’architecture actuellement en place au niveau du serveur. Vous trouverez ci-dessous l’architecture actuelle du serveur :

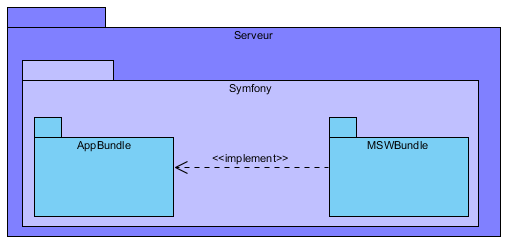


Figure 2. Architecture serveur

Le développement de la partie serveur du projet Music Sheet Writer est donc fait à l’aide du Framework PHP Symfony dans sa version 2.7. Toutes les procédures d’installation des différents outils de développement dont vous aurez besoin se trouvent dans le document prévu à cet effet : 2017\_ID1\_musicsheetwriter.pdf.

Un projet Symfony, dans sa version 2.7, est organisé avec l’arborescence suivante :

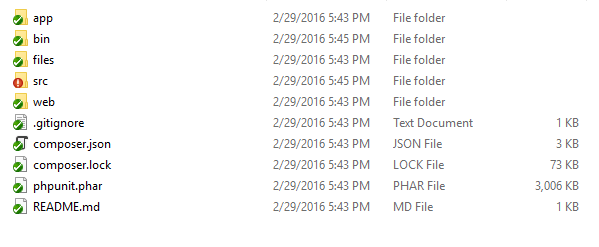


Figure 3. Arborescence projet Symfony

Le dossier **app** contient tous les fichiers en relation avec la configuration du projet et les logs. Le dossier **src** contient les sources du projet. C’est au sein de ce dernier que se trouvent les différents bundles du projet. Le dossier **web** est l’espace publique du projet. C’est au sein de celui-ci que se trouvent les ressources nécessaires à l’utilisateur pour l’utilisation du site internet, comme les images ou scripts JavaScripts. Nous ne rentrerons pas plus en détail dans la description d’un projet Symfony. Nous vous invitons à regarder la documentation officielle et tutoriels officiels de Symfony pour cela.

Une des notions principales de Symfony est le système de *bundle***.** Un bundle est un composant logique et indépendant dans un projet Symfony et ne devrait pas avoir de dépendance envers d’autres bundles. Dans notre projet, deux bundles ont été développés : le premier est le bundle **AppBundle** et est l’implémentation de notre API Rest définie dans le document 2017\_API\_musicsheetwriter.pdf ; le second est le bundle **MSWBundle**. Ce dernier contient toute la logique du site vitrine et de l’espace communautaire du projet et fait appel aux fonctionnalités de l’API Rest.

L’implémentation des bundles suit les bonnes pratiques définies dans la documentation officielle de Symfony :

* Documentation officielle : <https://symfony.com/doc/2.7/index.html>
* Bonnes pratiques : <https://symfony.com/doc/2.7/best_practices/index.html>

### AppBundle

Comme expliqué en introduction de cette partie, le bundle AppBundle est l’implémentation directe de l’API Rest définie dans le document 2017\_API\_musicsheetwriter.pdf. Cette dernière définies l’ensemble des routes pouvant être utilisé par l’utilisateur tout en définissant les différents rôles que doit avoir ce dernier pour pouvoir accéder à certaines routes et fonctions.

L’ensemble des routes et donc fonctionnalités définies dans le document de l’API se font à partir des routes commençant par **/api/**. Ceci est un choix dans l’implémentation de l’API Rest et n’est donc pas définit dans le document de définition.

Vous trouverez ci-dessous l’organisation de ce bundle telle qu’il est implémenté actuellement :

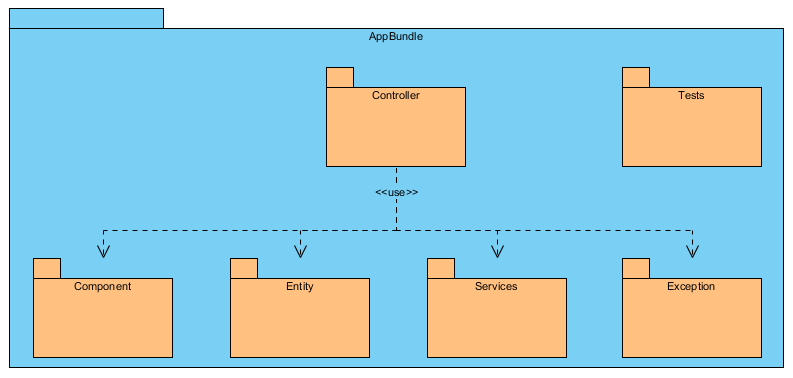


Figure 4. Architecture AppBundle

Vous l’aurez peut-être remarqué en parcourant l’arborescence du projet, mais le bundle contient plus de dossiers que ceux représenté dans la figure ci-dessus. Nous n’en parlerons pas car il s’agit de dossier contenant des fichiers de configuration générés lors de la création du bundle et qui n’ont pas besoin d’être modifié pour le projet.

Le bundle est donc composé de différents dossiers :

* **Controller** : Contient les fichiers des classes PHP implémentant les différentes routes définies par l’API Rest. Ces classes sont souvent appelées par le terme anglais *Controller.*
* **Component** : Contient les classes PHP permettant le formatage des réponses renvoyées par les différents Controllers au format spécifié par l’API Rest ainsi que d’autres classes pouvant être utile à l’implémentation de l’API Rest. Une des règles pour pouvoir ajouter une classe ou fonction aux classes déjà présentes est que cet ajout est indépendant de tous les autres composants du bundle. Cependant les autres composants peuvent être dépendant des informations contenues dans ces classes-là.
* **Entity** : Contient les classes PHP représentant les modèles gérées par l’API Rest (Users, Scores, etc). Aucune logique ne doit être implémentée dans ces classes-là.
* **Services** : Contient les classes PHP implémentant la logique de l’API Rest. C’est au sein de ces dernières que s’effectue le traitement des entités. Par exemple, le service **User**, s’occupe de l’implémentation des fonctionnalités liés aux utilisateurs et à la manipulation de l’entité **User**.
* **Exception** : Contient les classes PHP d’exception qui sont utilisées en cas d’erreur dans la manipulation des données de la part de l’utilisateur suite à une mauvaise utilisation de l’API ou dans le cas de problèmes liés au serveur en lui-même.
* **Tests** : Contient la définition des tests unitaires pour le bundle.

Afin de mieux comprendre comment marche cette organisation nous allons suivre l’exemple de l’implémentation de la fonctionnalité de login de l’API Rest.

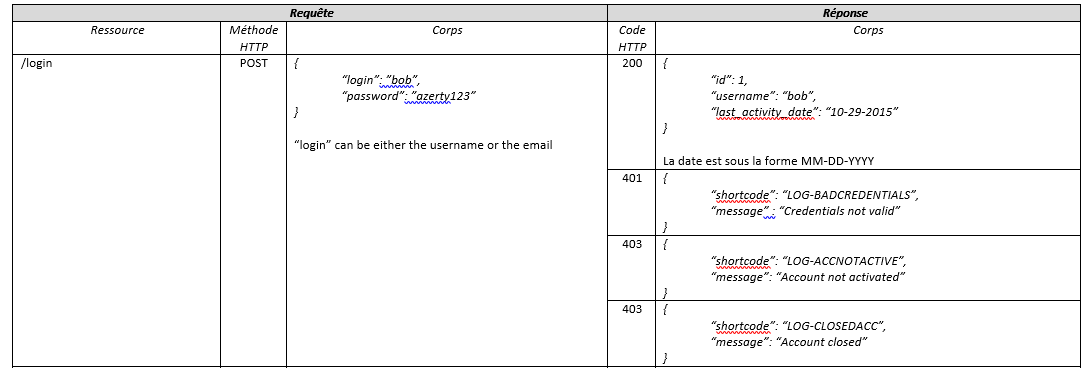


Figure 5. Exemple fonctionnalités Login – définition

Afin d’implémenter la fonctionnalité de login, nous devons implémenter un Controller pour la route **/api/login** acceptant la méthode POST et prenant dans le corps de la requête un objet JSON contenant le login de l’utilisateur et son mot de passe. Cette route peut renvoyer différents codes HTTP en fonction des informations qui lui ont été données : 200 en cas de succès 401 ou 403 en cas d’échec. Chacun des échecs contient un code définissant la raison de l’échec et un message expliquant ce dernier.

Cette route a été implémentée dans le Controlleur **ApiController.php** (c.f. : figure 5):

Plusieurs choses sont à noter : le bundle utilise le système de notation pour la gestion des routes (lignes ayants les mentions **@Route** et **@Method**). Si vous ne savez pas ce qu’est le système de notation je vous invites à regarder et lire la documentation officielle de Symfony.

Le corps du message étant sous forme de string JSON, nous utilisons la classe MSWSerializer et sa méthode *deserialize* pour transposer la chaine de caractère JSON en objet exploitable par PHP. Cette classe se situe dans le dossier Component

Vous pouvez lire qu’à plusieurs endroits nous faisons appels à la classe MSWResponse. Cette dernière s’occupe de retourner les bonnes réponses en fonctions du traitement des données. Elle se trouve elle aussi dans le dossier Component du bundle.

À la ligne 11 nous récupérons le service **User** afin de traiter les données. Nous appelons la méthode **login** du service en lui passant comme paramètre le login et mot de passe afin de savoir si l’utilisateur peut se connecter ou non.

### MSWBundle

Le bundle MSWBundle est le second bundle lié à la partie serveur du projet Music Sheet Writer. Il est responsable du site internet vitrine et de l’espace communautaire qui lui fait appels aux fonctionnalités de l’API Rest.

Voici son l’organisation interne du bundle :

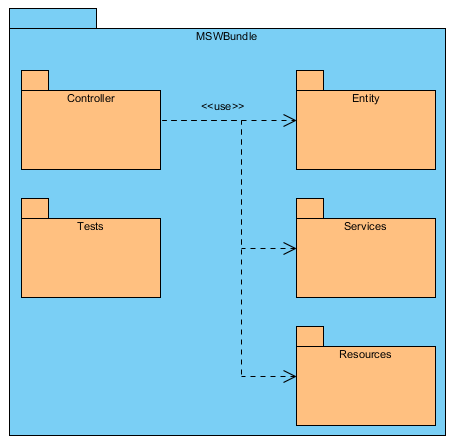


Figure 8. MSW Bundle – organisation

Comme pour le bundle AppBundle, nous ne décrivons ici que les dossiers réellement utilisé dans le développement de ce dernier. Les autres dossiers contiennent les fichiers de configuration liés au bundle.

Le dossier **Controller** contient les classes PHP s’occupant des routes pouvant être atteintes par l’utilisateur. Deux classes sont pour le moment implémentées : **ShowcaseController** pour toutes les routes liées au site vitrine et **CommunityController** pour les routes liées à l’espace communautaire.

Le dossier **Entiy** contient les classes PHP représentant les données manipulées par l’API Rest. Leurs définitions suivent celle de l’API Rest, mais est différent de celle du bundle AppBundle. Ici, les objets sont très proches de leur représentation dans les réponses renvoyées par l’API Rest.

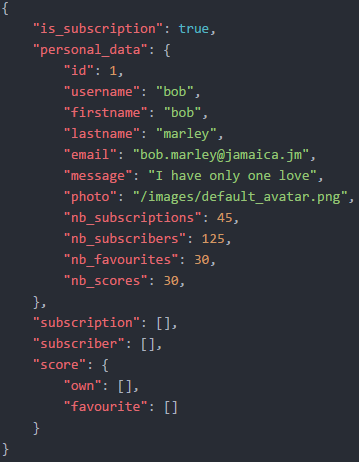
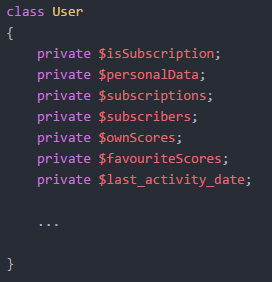


Figure 9. Implémentation User - MSWBundle

Figure 10. Réponse API Rest - User

Les classes qui se trouvent dans le dossier **Service** ont le même rôle que celles du dossier **Service** du bundle AppBundle. Ici, une seule classe a été définie : **EntityFactory.** Cette classe s’occupe de passer les entités des réponses JSON de l’API Rest en objets PHP exploitable pour l’espace communautaire.

Le dossier **Resources** contient l’ensemble des ressources utilisées pour le front-end du site internet. Vous trouverez dedans plusieurs dossiers :

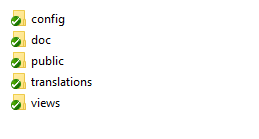


Figure 11. Arborescence dossier Resources – MSWBundle

* Config : Fichiers de configuration
* Doc : Fichiers de documentation
* Public : Dossier contenant les ressources utilisées sur les pages web tel que les images, vidéos ou autres scripts JavaScripts.
* Translations : Fichiers de traduction
* Views : Les différentes vues du projet. Ces vues peuvent être des pages web complètes ou des blocks de code HTML réutilisables. Ces vues utilisent le moteur de template Twig. Vous trouverez plus d’information sur ce moteur à l’adresse : <http://twig.sensiolabs.org/>.

Le dossier **Test** contient les différentes classes se chargeant d’effectuer les tests unitaires.

## Application Android

## Application iOS

## Application Windows Phone

# Gestion des sources et bug

## SVN

## Git

## Bugtracker

# Annexes

* API
* AA2
* TS
* Autres ressources