Université de Caen Basse-Normandie

Rapport de stage

1 avril 2015 - 30 septembre 2015

Open orchestra

Auteurs :
Amaury Lavieille

Enseignants :
François Rioult

12 août 2015

UNICAEN
université de Caen
Basse-Normandie

Remerciements

Table des matières

Introduction

Dans le cadre de ma deuxième année du master DNR2I (Master document numérique en réseau, ingénierie de l'Internet), j'ai effectué un stage chez Business & Decision au sein de l'agence Interakting à Caen.

Pour une durée de six mois, du 1er Avril au 30 Septembre 2015, j'ai été intégré au sein de l'équipe de développement du projet Open Orchestra.

Dans un premier temps, je vais vous présenter succinctement l'entreprise. Ensuite, je vous décrirais le projet Open Orchestra, son origine, ces particularités, etc. Enfin, je vous parlerais de l'organisation et des méthodes mises en place pour la réalisation du projet.

Première partie

L'entreprise

Chapitre 1

Présentation de l'entreprise

1.1 Business & Decision

Business & Decision est un groupe international spécialisé dans trois grand domaines, la Business Intelligence [1] (BI), la gestion de la relation client et enfin le e-business.

Le groupe a était créé en 1992 par Patrick Bensabat, aujourd'hui il est présent dans 15 pays et emplois plus de 2500 personnes.

Événement majeurs et évolution du groupe depuis 1992 :

1992 : Création de Business & Decision par Patrick Bensabat autour de projets de Business Intelligence

1993-1996 : Mise en place des premiers Datawarehouses [2] et applications de prévisions financières

1999-2000 : Création de la division CRM (Gestion de la relation client)

2002-2003 : Acquisition en Grande-Bretagne et au Benelux. Création d'agence régionales en France

2004 : Acquisition en Suisse et aux Pays-bas et implantation en Tunisie

2005 : Implantation aux Etats-Unis et acquisition de Metaphora [3]

2007-2008 : Création de la marque Interkating

2011 : Déploiement d'offres Cloud et Mobilité. Inauguration du Datacenter éco-responsable d'Eolas

2013 : Lancement de Datalyse, programme de recherche en Big Data. Implantation au Pérou

2014 : Création de Herewecan, agence de communication digitale, Lancement des pôles d'expertise Big Data, Transformation et Hub Mobile)

1.2 Interakting

Interakting est l'agence web du groupe B&D composé de 340 employés répartis principalement entre Caen et Paris, les équipes de l'agence interviennent sur des projets web de grande envergure (Canal +, Bnp Paribas, Inra, Moët Hennessy, PSA Peugeot Citroën, ...).

1. La Business Intelligence ou l'informatique décisionnelle représente les différentes solutions informatiques qui permettent l'exploitation des données du entreprise dans le but de faciliter la prise de décision

2. Les Datawarehouses ou entrepôt de données désigne une base de données utilisée pour stocker et structurer des données de production d'une entreprise afin de fournir des informations stratégiques pour la prise de décision

3. Métaphora est une société de service spécialisée dans la conduite du changement. Elle intervient dans toutes les étapes d'un projet pour faciliter l'appropriation du futur système d'informations par les utilisateurs finaux.

Les différents projets de l'agence sont développés avec différent outils : eZ publish (système de gestion de contenu créé par l'entreprise eZ Systems As), PHP Factory (plateforme réalisé par conjointement par Interakting et Zend Technologies) et de plus en plus avec Symfony2 (framework php écrit par SensionLabs) avec notamment Open Orchestra.

Deuxième partie

Open Orchestra

Chapitre 2

Orchestra

2.1 Présentation

Open Orchestra est un CMS (Content Management System) open source en développement depuis un peu plus d'un an. Il est actuellement en version bêta, la sortie de la première version est prévus pour le mois de septembre.

Il est basé sur le framework PHP MVC Symfony 2 et MongoDB. Open Orchestra offre bien-sûr les fonctionnalités attendues par tous CMS : gestion de contenu, médiathèque, contribution de page, gestion de version, utilisateurs, worklow, rôles, multi-site, multi- langue, multi-device , etc.

La grande particularité d'Open Orchestra est son faible couplage au niveau du code, mais aussi au niveau des solutions techniques, c'est-à-dire que l'on peut facilement remplacer ou étendre un de ses composants. Je reviendrais plus en détail sur ce point dans une section ultérieure.

2.2 Architecture

Avec Open Orchestra le « Back-office » et le « Front office » peuvent être dissociés dans deux applications Symfony différente.

Ce découpage a deux nombreux avantages, tous d'abord un seul « Back-office » peux gérer plusieurs sites (« Front-office »), de plus cela permet de mettre « Back-office » et le « Front-office » sur des serveurs différents, l'unique condition est qu'ils doivent tous utiliser la même base de données.

Pour finir, Open Orchestra utilise une API RESTFull [1] pour accéder, gérer ces différentes entités (pages, utilisateurs, contenus, etc).

Le schéma 2.1 présente l'architecture d'Open Orchestra avec un « Back-office » qui gère plusieurs « Fronts ».

1. REST (Representational State Transfer) est un style d'architecture qui définie différentes réglés (ressource identifié unitairement, utilisation des verbes HTTP POST, DELETE, PUT, GET) pour accéder et manipuler des ressources

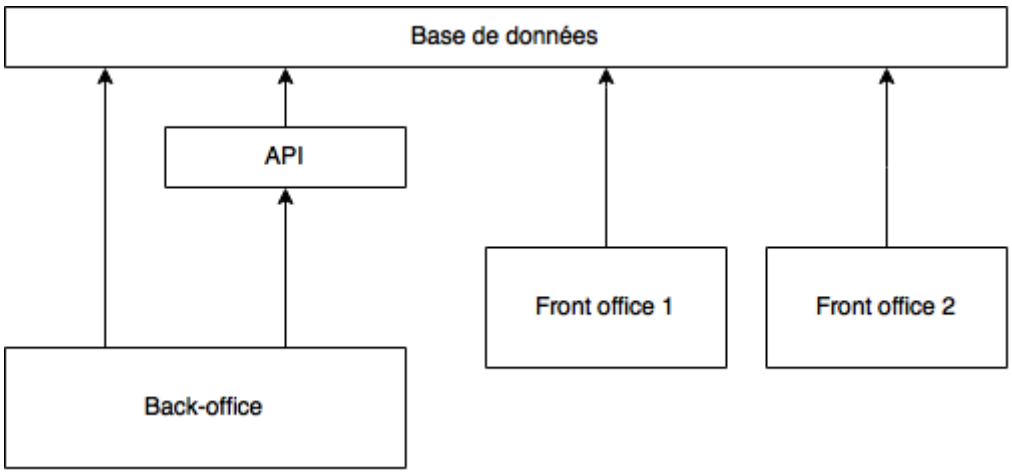

FIGURE 2.1 – Exemple d'architecture d'utilisation d'Open Orchestra

2.3 Fonctionnalités

Dans ce chapitre je vais vous présenter les différentes fonctionnalités d'Open Orchestra. Bien sûr, je ne vais pas tous les détailler, mais uniquement les points clés qui permettent une bonne compréhension du projet.

2.3.1 Blocs

Dans Open Orchestra tous les éléments visibles en front sont représentés par des blocs. Un bloc est simplement une entité avec des attributs qui varient selon le type de bloc. Chaque type de bloc sont indépendant, c'est-à-dire que eux seul connaissent leurs attributs, leurs façon de s'afficher en front ou en back-office ou encore la façon dont ils peuvent être contribué en back-office.

Pour permettre cette indépendance, Open Orchestra utilise le design pattern stratégie. Le pattern stratégie permet de rendre une famille d'algorithmes interchangeables et ainsi la possibilité d'exécuter un traitement spécifique selon le contexte.

Comme le présente le diagramme de classe 2.2 chaque bloc possède une classe qui indique comment il doit s'afficher pour cela la classe a deux méthodes, la première qui indique le bloc qu'elle supporte (`support`) et la méthode `show` qui fourni le rendu, d'un autre coté il y a une classe que appelé `manager` qui est la seule a connaitre toutes les stratégies des différents blocs.

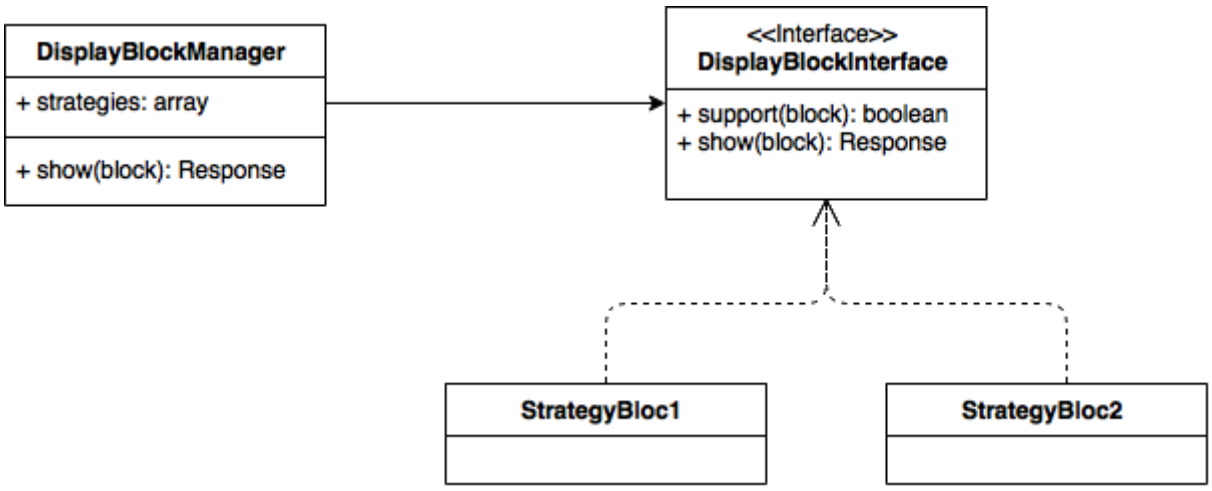

FIGURE 2.2 – Utilisation du pattern stratégie pour l'affichage des bloc en front

Ainsi, lorsque que l'on désire afficher un bloc, il suffit de demander au manager d'afficher ce bloc, ce dernier va chercher parmi les stratégies qu'il connait laquelle supporte (méthode `support`) le bloc et s'il en trouve une alors il exécute la méthode (`show`) de la stratégie.

Par défaut, le CMS propose de nombreux types blocs comme par exemple un bloc pour lister un type de contenu, afficher un texte formaté, afficher un carrousel ou un média, un menu, une carte, un bloc de contact, etc.

Bien-sûr, il possible pour un intégrateur [2], de créer son propre type de bloc, il lui suffit de développer les différentes stratégies (affichage en front et back-office, formulaire en back-office, etc) nécessaire à un bloc.

2.3.2 Nodes

Un des points central d'un CMS est les pages. Sur Open Orchestra les pages sont identifiées comme des noeuds (« nodes »). Les nodes sont simplement des conteneurs qui contiennent des zones. Les zones permettent d'organiser la page, elles contiennent des sous-zones ou des blocs ce qui permet un découpage fin de la page pour simplifier sont organisation.

Il existe trois types de nodes :

— Les nodes qui représentent les pages visibles en « front ».

— Les « node tranverse » qui contiennent les blocs transverses, c'est-à-dire les blocs qui sont communs à plusieurs pages comme le bloc « menu » ou encore le bloc « footer ».

— Le dernier type de nodes sont ceux qui permettent de contribuer les pages d'erreurs (404, 503) d'un site.

2. Dans le cadre de ce rapport un intégrateur indique une personne ou une équipe qui utilise Open Orchestra afin de l'étendre pour des besoins particuliers

Le schéma 2.3 montre un exemple typique de page avec trois zones, le header qui contient un bloc menu, le footer un bloc footer et une zone principale découpé elle-même en deux zones avec un bloc de contact pour la zone de droite et un bloc de texte à gauche.

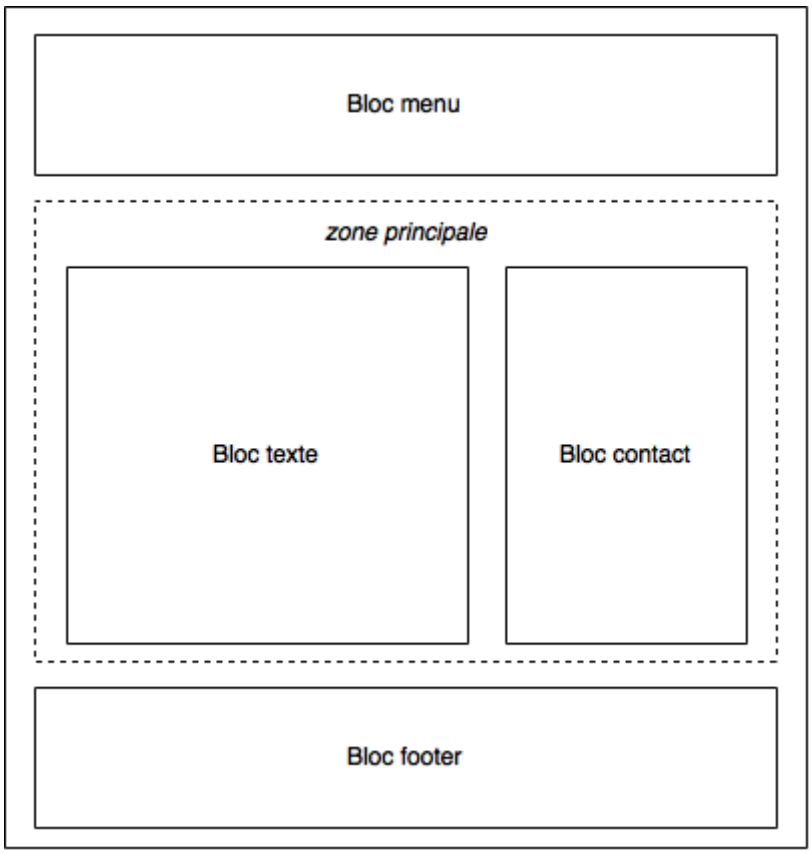

FIGURE 2.3 – Exemple de page

2.3.3 Content type

Le second point important d'un CMS est les contenus avec Open Orchestra il est facile de créer des types de contenus (actualité, client, etc, ...).

En effet, Open Orchestra propose une approche graphique grâce à un formulaire pour créer ou éditer des types de contenus comme l'illustre la figure. Un type de contenu est composé de différent champs avec différente option, par exemple pour un champ de type texte il peut y avoir une option pour limiter le nombre de caractère ou encore une valeur par défaut.

Par défaut, Open Orchestra offre une liste de types de champs (date, texte, monnaie, média, email, entier, zone de texte riche, etc) qui comme pour tous les composants d'Open Orchestra peut être étendu par d'autre type de champ personnalisé.

2.4 Caractéristiques

2.4.1 Performance

Open Orchestra a été développé pour supporter une charge de trafic importante. Pour cela le CMS exploite au niveau du front l'ESI (Edge Side Includes) couplé à un reverse proxy [3].

L'ESI est un langage de balisage HTML qui permet de diviser une page en différent éléments dont les rendus sont faits dans différentes requêtes par le serveur web. Ce qui permet d'avoir un cache HTTP sur les différents éléments et ainsi rafraichir seulement les éléments obsolètes de la page et non toute la page.

Comme nous l'avons vu dans la section **??** sur Open Orchestra les pages sont déjà découpées en différent blocs ainsi l'utilisation de l'ESI est adapté et permet une amélioration significative des performances.

Si nous reprenons l'exemple de notre page (schéma 2.3), le schéma 2.4 présente le processus effectué par le serveur dans le cas où les blocs `header` et `footer` sont déjà en cache lorsqu'un utilisateur demande la page.

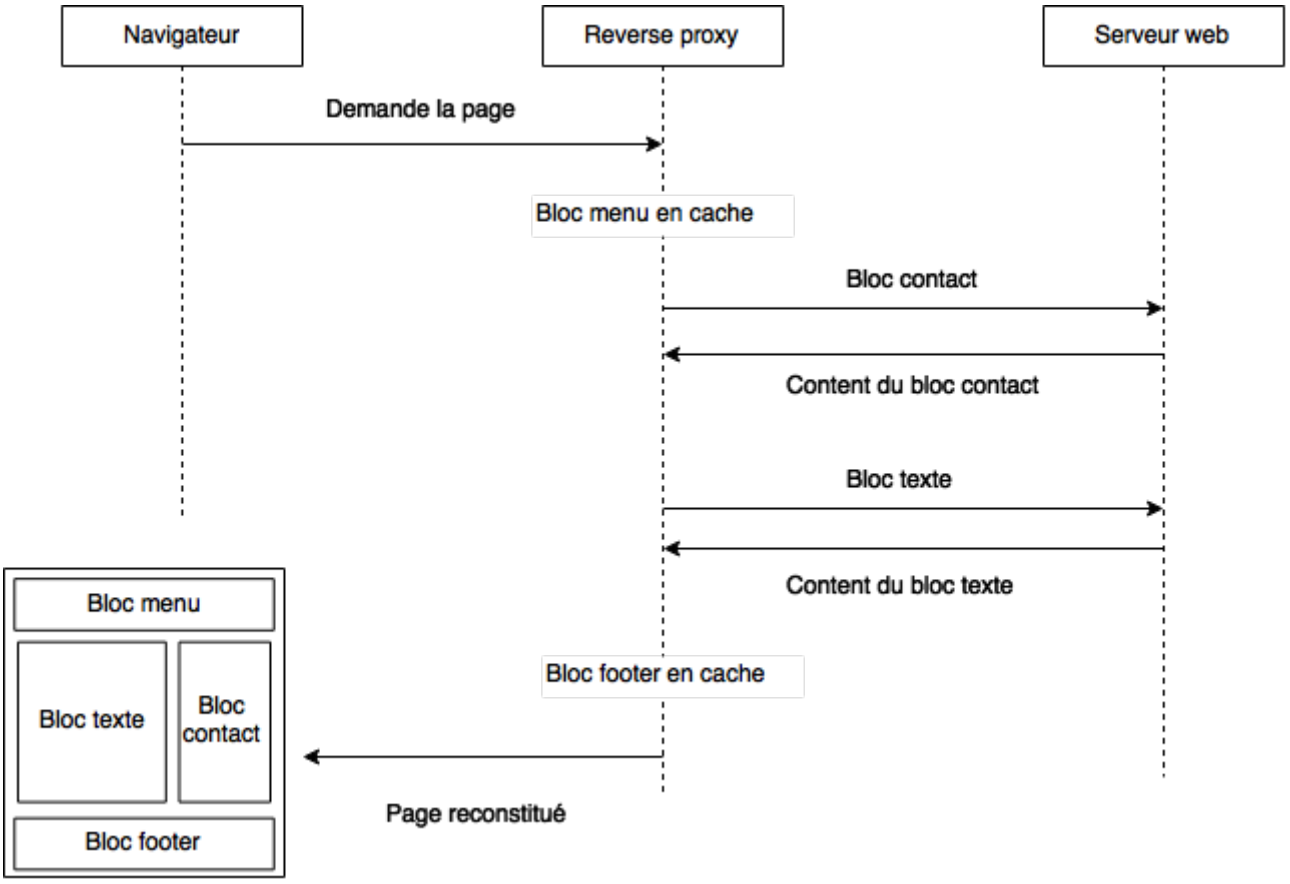

FIGURE 2.4 – Processus d'affichage d'une page qui utile l'ESI

3. Un reverse proxy est un serveur qui traite les requêtes en amont du serveur web. L'intérêt d'un reverse proxy est multiple gestion de cache, chiffrement, répartition de charge

2.4.2 Modularités

Comme je vous l'expliquais en introduction la principale particularité d'Open Orchestra est sa modularité. Pour cela, les différents composants du CMS sont répartis dans différents bundles [4]

— **open-orchestra-base-bundle** : Classes communes au back-office et front-office
— **open-orchestra-base-api-bundle** : Api d'Open Orchestra
— **open-orchestra-base-api-mongo-model-bundle** : Implémentation des models nécessaires à l'api pour `MongoDB`
— **open-orchestra-cms-bundle** : Logique du back-office
— **open-orchestra-front-bundle** : Logique du front-office
— **open-orchestra-display-bundle** : Logique d'affichage des blocs en Front-office
— **open-orchestra-media-bundle** : Médiathèque d'Open Orchestra
— **open-orchestra-media-admin-bundle** : Administration de la médiathèque d'Open Orhcestra
— **open-orchestra-model-interface** : Interfaces des models utilisés par les autres bundles
— **open-orchestra-model-bundle** : Implémentation des models pour `MongoDB`
— **open-orchestra-user-bundle** : Gestion des utilisateurs
— **open-orchestra-workflow-function-bundle** : Système de workflow pour le back-office

Une application front-office n'a aucun intérêt à charger toute la logique du back-office ainsi grâce à ce découpage chaque application (front-office ou back-office) charge uniquement les composant qui lui est nécessaire.

De plus, cela permet de désactiver facilement une fonctionnalités. Par exemple, si un intégrateur n'a pas bessoin de gérer des médias il peut alors désactiver la médiathèque en n'utilisant pas les bundles (`open-orchestra-media-bundle` et `open-orchestra-media-admin-bundle`).

Un autre avantage est la facilité de remplacement d'un composants. Par défaut Open Orchestra utilise `MongoDB` comme système de gestion de base de données, mais si un intégrateur veut utiliser un autre système comme `Mysql` alors il lui suffit d'écrire son propre `open-orchestra-model-bundle` qui utilise bien-sûr les différentes interfaces de `open-orchestra-model-interface` adapté a `Mysql`.

4. Sous Symfony2 un bundle est un ensemble de fichiers structurés (contrôleur, entité commande, listener, formulaire) qui permettent d'implémenter une ou des fonctionnalités qui dans l'idéale peuvent être réutilisés dans différent projets

Troisième partie

Organisation du développement

Chapitre 3

Organisation du développement

Introduction, pas eu de tâche particulière intégration au sein de l'équipe avec diff tâches (ajout fonctionnalités, correction bug, Documentation en aglais, refacto, optimisation, formation

3.1 Gestion de projet agile

3.1.1 Méthodes agiles

Le projet a été mené selon une approche agile. La gestion de projet agile repose sur un développement itératif qui consiste à découper un projet en plusieurs itérations appelé sprint. Les méthodes agiles définis plusieurs valeurs fondamentales :

- La communication au sein de l'équipe
- La collaboration avec le client, celui-ci doit être impliqué tous au long du développement.
- L'adaptation au changement, c'est-à-dire que le planification initiale doit être flexible pour permettre l'évolution de la demande.

Il existe différentes type de méthode agile (Scrum, XP, RAD...) qui possèdent leurs propres caractéristiques.

Pour le développement d'Open Orchestra, c'est la méthode Scrum qui est utilisé. Scrum définis trois rôles au sein d'une équipe :

- Le « Product Owner (PO) » qui porte la vision du produit
- Le « Scrum Master » est le responsable de la mise en œuvre de la méthode, il doit s'assurer que cette dernière est correctement appliqué
- L'équipe de développement qui est chargé de transformer les besoins exprimés par le Product Owner.

La réalisation d'un projet utilisant la méthode est rythmée par différents évènements :

Tous d'abord, le sprint qui est représente une période de courte durée, de une à quatre semaines, durant laquelle l'équipe effectue un nombre de tâches définis à l'avance.

La réunion de planification où l'équipe répond à deux questions, « Quoi ? » c'est-à-dire les tâches du backlog [1] qu'elle réalisera au prochain sprint et « Comment ? » c'est a ce moment que l'équipe estime les tâches choisis au « Quoi »

1. Le backlog est un ensemble de fonctionnalités ou de tâches nécessaire pour la réalisation satisfaisante d'un projet

Le « daily scrum » qui est une réunion réalisé quotidiennement. Durant cette réunion chaque membre de l'équipe indique les tâches qu'il a réalisé depuis le dernier daily et celles qu'il va effectuer jusqu'au prochaine daily et pour finir les difficultés qu'il a ou pense rencontrer. Cette réunion permet à tous les membres de connaître les tâches de chacun afin de s'entraider et d'anticiper plus facilement les obstacles.

Pour finir à la fin du sprint, les membres de l'équipe se réunisse pour la revue de sprint durant laquelle les différentes tâches réalisés durant le sprint sont présentés et validés. Puis pour la rétrospective qui permet de mettre en évidence les points positif et les actions a mettre en place pour améliorer le prochain sprint.

SCHEMA (schéma sprint)

3.1.2 Scrum avec Open Orchestra

Au sein du projet Open Orchestra, la méthode Scrum est appliqué.. Ainsi des sprint de une semaine avec un daily scrum tous les jours a midi. De plus, tous les mercredis ou mardis selon les disponibilités une « cérémonie » qui comprend la réunion de planification et la rétrospective. Concernant, la revue de sprint (validation des taches du sprint) celle-ci n'est pas faite durant une réunion définis mais tout au long du sprint par le product ownver.

Pour faciliter la mise en place de la méthode Scrum, l'équipe utilise différent outils. Tous d'abord Skype et Slack [2] pour la communication.

Et enfin Trello pour organiser les tâches, en effet l'équipe possédé un `board` avec différentes colonnes pour organiser les tâches :

- **Backlog** : le backlog qui contient les différentes tâches a effectué pour le projet, le tâches du backlog sont organisées en quatre colonnes (Nice to have, Good, Great, Must have) selon la priorité de la tâche.
- **Support** : Les demandes effectué par les différentes équipes d'intégrateur d'Open Orchestra au sein d'Interakting.
- **Proposition** : Les différentes propositions d'amélioration ou de re- factorisation du code faites par les membres de l'équipe.
- **Bugs** : Les différents bug rencontrés durant le développement
- **Todo** : Les taches a effectuer durant le sprint en cours.
- **Doing** : Les taches actuellement en développement.
- **Blocked** : Les taches bloqué pour différentes raison (manque de précision de la tache, bug empêchant la réalisation de la taches)
- **ToDeploy** : Taches réalisés qui sont prêtes a être déployé sur le serveur d'intégration.

2. Slack est une plateforme de communication réalisé en 2014 qui permet d'intégrer facilement différents outils de service en ligne tel que github, trello, dropbox, google drive, ...)

— **ToValidate** : Taches du sprint en attente de validation par le product owner.

— **Failed** : Taches du sprint e non validé par le product owner

— **Done** : Taches du sprint validé par le product owner

Les colonnes `ToDeploy`, `Done` sont unique a un sprint, il y a donc une colonne différent`ToDeploy`, `Done` pour chaque sprint.

Par exemple, une tâches suit un processus, présenté par le schéma, bien particulier entre l'intrégration de la tache dans la sprint et ca validation par le product ownver. Ce processus permet de suivre facilement l'avancement ou encore les différents problèmes rencontrés sur les différentes tâches par tous les membres de l'équipe.

3.2 Intégration continue

Pour le développement d'Open Orchestra les principes de l'intégration continue sont utilisés.

> L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

@TODO source wikipedia

L'utilisation de ces pratiques apporte de nombreux avantages, tous d'abord comme il est indiqué dans la citation cela permet de minimiser les régréssions mais aussi d'avoir a tous moment un produit (logiciel, site, etc) utilisable.

La mise en place de cette technique nécessite différentes pré-requis :

— Dépôt unique de code source versionné
— Automatisation des tests
— Un environnement similaire a celui de production
— Automatiser le déploiement

3.2.1 Git

Dans le cadre d'Open Orchestra, le code source est versionné avec git [3] et utilise GitHub qui est un service web permettant d'héberger un dépôt git. Comme je l'expliquais lors des présentations des caractéristiques d'Open Orchestra, ce dernier est découpé en plusieurs bundles et donc chaque bundles possède son propre dépôt sur GitHub.

Open orchestra étant toujours en version bêta, il n'y a pas de realease a proprement parlé. Ainsi le worklow de développement sur github est simplifié car il n'y a qu'une seule branche stable, la branche master. Les autres branches sont des branches temporaires d'ajout de fonctionnalités ou de correction de bugs. Ce type de workflow est appelé `GitHub flow` et est représenté par le schéma @TODO schéma git.

3. Git est un logiciel de versionning décentralisé, c'est-à-dire qu'il n'existe pas un dépôt unique mais un dépôt local pour chaque développeurs.

Ainsi avec ce type de workflow, la modification du code source que ce soit pour corriger un bug ou ajouter une fonctionnalités s'effectue toujours de la même manière :

1. Mettre à jour sa branche master en local
2. Créer une nouvelle branche en local à partir de master
3. Effectuer les modifications dans cette branche
4. Pousser sa branche sur GitHub (origin)
5. Vérifier que les tests fonctionnne, je reviendrais plus en détail sur ce point par la suite
6. Ouvrir une pull-request, sur GitHub une pull-request consiste à faire une proposition de modification, à partir de cette pull-request les différents membres peuvent commenter, valider ou non la modification.

Une fois la pull-request validé, elle peut être mergé avec master, c'est-à-dire inclure la modification dans la branche master.

Comme je l'ai précisé en début de section pour le moment Open Orchestra ne possédé pas de branche realease. Toutefois le projet Open Orchestra est déjà utilisé par des intégrateurs pour réaliser des sites a distanciation de différent clients, il ne peuvent donc pas utiliser directement la branche master, ils leurs fait des points d'arrêts. Pour cela, sauf cas particulier (bug bloquant, besoins particulier) la branche master est `taggé` toute les deux semaines environs ainsi cela permet à l'équipe d'Open Orchestra de continuer le développement du projet et aux intégrateurs d'avoir un version fixe de la branche master qu'il mettre à jour au moment le plus opportunt pour eux. Lorsque Open Orchestra ne sera plus en bêta cela changera la version 1 aura sa propre branche dans la quelle aucune nouveautés sera intégré or mis les corrections de bug.

@TODO Explication des tag github.

3.2.2 Travis CI

Lorsqu'une branche est poussé sur l'un des dépôts GitHub du projet, les tests sont automatiquement lancé. Pour cela, les différents dépôts du projet intégre un outil `Travis CI`. Travais CI est un service qui est liée a GitHub, il permet de lancer à déclencher un certains nombre de tâches lors d'événement sur le dépôt. Dans le cas d'Open Orchestra Travis CI permet de lancer les tests unitaires après chaque `push` sur un dépôt. Un des avantages de travis est qu'il se configure très facilement grâce un fichier yaml @TODO footnote yaml.

Par exemple le fichier de configuration travis d'un des dépôts d'Open Orchestra

```
language: php

php:
      - 5.4

install:
      - composer install --prefer-dist --no-progress

script: ./bin/phpunit
```

Indique que le langage utilisé est php et que les tests doivent être exécuté avec la version 5.4 de php et enfin avant d'exécuter `phpunit` pour lancer les test il est demandé a travis de faire un `composer install` pour récupérer les dépendances du projet. Ensuite à partir de ce fichier

de configuration Travis CI est capable de créer l'environnement pour effectuer les tests, il est bien sûr possible d'aller plus loin dans personnalisation de l'environnement en allant cherchant d'autre dépendances avec `apt-get install` par exemple ou encore de demander a être notifié par email si les tests ne passe pas. @TODO Schéma travis

3.2.3 Déploiement

Le déploiement sur un serveur de production n'est pas toujours une tâche facile(uploader le code, mettre à jour les dépendances, lancer les migrations de la base de données, vider le cache, etc) or pour permettre une intégration continue il est nécessaire de déployer régulièrement son application, il semble donc nécessaire d'automatiser cette taches pour cela il existe de nombreux outils.

En ce qui concerne Open Orchestra nous utilisons `Capistrano`. `Capistrano` est un outil écrit en Ruby qui permet d'exécuter des scripts sur un ou plusieurs serveur. Il est principalement utilisé pour faire du déploiement mais il permet aussi de créer ces propres par exemple pour Open Orchestra nous l'utilisons aussi pour lancer des tests de charge, avec Jmeter, directement sur le serveur d'intégration.

Un autre avantage `Capistrano` est qu'il permet de revenir facilement sur une version précédente de votre application grâce a une architecture de dossier que met en place `Capistrano` dans le dossier de votre application sur serveur :

```
/app/
      releases/ Contient un dossier pour chaque déploiement estampillé
      current lien symbolique pointant vers un dossier de release spécifique
      shared/ Contient les données partagées entre chaque déploiement
```

Ainsi, si un déploiement provoque des erreurs, il suffit de faire pointé le lient symbolique `current` vers une autre version.

@TODO Schéma resumé de capistrano

3.3 Qualité

L'équipe d'Open Orchestra essaye de fournir un produit de qualité que ce soit au niveau du code, du couplage ou encore du fonctionnel. Pour cela différente pratique, outils sont mise en place.

3.3.1 Qualité du code

Tous d'abord au niveau du code Open Orchestra utilise le standard PSR-2. PSR-2 est un ensemble de recommandation sur le style et l'organisation du code dont voici les plus importantes :

- La tag de fermeture \>. doit être omis de tous les fichiers contenant uniquement du PHP
- Tous les fichiers PHP doivent se terminer par une ligne vide.
- La visibilité doit être déclarée sur toutes les propriétés et méthodes.
- L'ouverture des accolades pour les classes, les méthodes doit figurer sur la ligne suivante, les accolades de fermeture DOIVENT figurer sur la ligne suivante après le corps de la classe.

— Il doit y avoir un espace après la structure clé de contrôle et pas d'espace après la parenthèse ouvrante et avant la parenthèse fermante
— Il doit y avoir un espace entre la parenthèse fermante d'une structure de contrôle et de l'accolade ouvrante
— L'accolade fermante d'une structure de contrôle doit être sur la ligne suivante après le corps

Utiliser cette recommandations permet une harmonisation du code et ainsi il est plus simple pour toute l'équipe de relire et reprendre le code d'un autre membre mais aussi pour tous les développeurs qui utilise Symfony 2 puisque ce dernier utilise aussi les recommandations PSR-2

Lors de revue de code des pull-request, il n'est pas toujours aisé de détecter l'utilisation des bonnes pratiques ou non. Il existe des outils qui permettent d'analyser le code automatiquement et de détecter les oublis lors de la revue.

Pour le projet Open Orchestra deux outils sont utilisés :

Tous d'abord, Code Climate s'interface avec les dépôts c'est à dire que lors d'un push le code est analysé. Code Climate permet de détecter la duplication de code, la couverture des tests, la qualité globale du projet. De plus, il fournis une note indique la qualité globale du dépôt. Le second outil est SensioInsignt fonctionne de la même façon que Code Climate mais en supplément des vérifications sur les bonne pratiques de Symfony 2.

3.3.2 Test

Le plus grand intérêt des tests est qu'ils permettent de réduire les régressions lorsque l'on applique des modifications a une applications. En effet, lorsqu'une application devient assez conséquente il n'est pas toujours aisé d'anticiper toutes les répercutions lorsque l'on modifie une portion de code. Les tests permettent donc de tester l'ensemble de l'application et ainsi vérifier que cette dernière fonctionne toujours correctement après les modifications effectué. De plus, ils sont en générale gage de qualité

Il existe deux grandes familles de test, Tous d'abord les tests unitaires qui permettent de vérifier que chaque méthode et fonction fonctionne correctement, les test doit être indépendant au maximum les uns des autres pour cela il existe des oulits (@TODO les MOCKS) . D'autre part les tests fonctionnels qui eux contrairement aux tests unitaires les fonctionnalités de bout en bout. c'est-à-dire qu'ils toutes les couches de l'application (routing, modèle, action, controlleurs, template).

Avec Symfony, il est assez facile de créer des tests puisqu'il intègre la bibliothèque PHPUnit (@TODO explication PHPunit) qui fournis un framework de test complet, de plus pour les tests fonctionnels Symfony propose une class WebTestCase qui permet d'initier le noyau de Symfont, elle fournis notamment un objet `client` qui permet d'effectuer des requêtes HTTP sur son application et un crawler qui permet de vérifier si le résultat de la requêtes est correct .

Avec ces deux types de tests toutes les fonctionnalités d'une application ne sont pas encore couverte, il reste encore l'IHM (Interface Homme Machine) qui est a mon avis une des parties

les plus dur a tester puisqu'elle peut intégrer notamment du JavaScript, ce qui rend les tests côté serveur que nous avons vue précédemment inutile.

Test unitraite, Test fonctionel, (Mise en place de test pour tous les codes metiers) Behat (mise en place de test behat)

3.3.3 Qualité fonctionnel

Fonctionnel