Master 2 Développement Logiciel

Rédigé avec LaTeX
Version du 12 novembre 2015

Documentation de BandStorm

Intégration, Vérification, Validation, Qualification

Université Toulouse III – Paul Sabatier

— Julian Bironneau
— Antoine de Roquemaurel
— Steve Magras
— Dylan Roletto
— Zaccaria Zyat

12 novembre 2015

Table des matières

1 Le projet

1.1 Le projet Bandstorm

Bandstorm est un projet qui a pour but de réaliser un réseau social pour des amateurs de musique. Celui-ci a pour objectif de mettre en relations des utilisateurs par le biais de communautés. Ces communautés sont constitués en fonction de préférences musicales.

1.2 Les liens utiles. . .

1.3 Fonctionnalités

Notre application possède de multiples fonctionnalités qui vont être détaillées ci-dessous. Chaque utilisateur de BandStorm possède un compte personnel sécurisé par un système de login.

— Les utilisateurs ont la possibilité de suivre d'autres utilisateurs.
— Les utilisateurs peuvent créer, rejoindre et visualiser un groupe.
— Les utilisateurs peuvent créer et visualiser un évênement.
— Les utilisateurs ont la possibilité de rechercher des entités via une barre de recherche (utilisateurs, groupes, évênements).
— Les utilisateurs peuvent poster des statuts.
— Les utilisateurs peuvent visualiser les statuts de leurs abonnements.
— Les utilisateurs ont la possibilité de "lighter" (aimer) un statut.
— Les utilisateurs ont la possibilité de changer leurs informations personnelles.

2 Le *workflow* de développement

Afin de faciliter le travail collaboratif et l'intégration de nos développements respectifs, nous avons utilisé Git et Github. Via un workflow en branches par fonctionnalités, nous avons pu développer en parallèles nos fonctionnalités. Lorsqu'un développeur choisi de répondre à une issue, il va créer une nouvelle branche, nommée suivant la convention suivante : sprintN0/noIssue-recapitulatifIssue

Lorsque le développeur estime avoir terminé sa fonctionnalité, il doit ouvrir une Pull Request, celle-ci nous permet d'éffectuer de la revue de code. La fonctionnalité ne sera fusionnée avec la branche de niveau inférieur que lorsque l'équipe estime avoir terminé l'issue.

Plus de détails notre workflow est disponible ici : https ://github.com/BandStormTeam/BandStormProject/b
feature.md

2.1 Tests et couverture

Lors du développeemnt d'un projet complexe, il est indispensable de tester correctement le projet afin de pouvoir le vérifier et le valider. Ainsi, dans le cadre de BandStorm nous avons définis des pourcentages de couverture de tester à respecter impérativement ; Le bon respect de ces chiffres contribuent à effectuer un projet de qualité.

Cette couverture doit être de 80%,
- 100% pour les classes du domaine.
- 80% pour les contrôleurs.
- 80% pour les services.

Les classes du domaine, services et contrôleurs devront être couverts par des tests unitaires. Les services de type DAO devront être couverts par des tests d'intégrations.

Afin d'avoir des tests les plus exhaustifs possibles, nous avons également mis en place des tests fonctionnels en utilisant Geb. Ces tests nous permettent de vérifier le bon fonctionnement global de notre application.

2.2 Intégration continue

Nous avons utilisés Travis CI afin d'intégrer et de vérifier cette intégration de manière continue. Notre serveur d'intégration effectue plusieurs actions : Compilation du projet Lancement des tests unitaires Lancement des tests d'intégration Calcul de la couverture de code Éventuel déploiement sur heroku (uniquement master) Le build ne peut être correcte que si l'ensemble de ces actions se

sont bien effectuées. À noter que Travis CI ne lance pas les tests fonctionnels. En effet, les tests fonctionnels nécessitent un serveur graphique ainsi que le démarrage d'un navigateur, la version gratuite de Travis-CI ne semble pas répondre à nos attentes. Il pourrait être intéressant d'installer un serveur d'intégration continue sur une machine personnelle tel que Jenkins ou Gitlab-CI.

Les tests fonctionnels étant les tests les plus proches d'un utilisateur permettrait ainsi de pouvoir effectuer de la non-régression de façon systématique et automatique.

2.3 Déploiement continue

La vérification de la bonne intégration de notre application ne suffit pas à avoir un processus de mise en production fiable. Il est indispensable de vérifier le bon déploiement de l'application sur un serveur autre que celui de production avant. C'est dans ce but que nous utilisons Heroku. La La branche master de l'application est déployée de façon continue et automatique sur heroku.

2.4 Analyse statique de code

our avoir un code de qualité, il est faut également prendre en compte la maintenance que celui-ci va couter. Cette maintenance peut être réduite en développant avec des conventions claires précises et mesurables. Ainsi, nous avons utilisé codenarc qui permet de vérifier les règles de nos conventions d'écritures.

SonarQube quant-à lui permet d'une part de centraliser toutes les informations nécessaires, tout en conservant un historique de celles-ci, mais il nous permet également d'avoir de nouvelles métriques tel que le nombre de ligne de code, le pourcentage de commentaire ou la complexité cyclomatique de notre projet.

3 Agilité

3.1 Sprints

Pendant toutes la durée du projet, l'équipe à suivi les principes du processus agile Scrum. La durée du projet a été découpée en trois sprints d'une durée de deux semaines chacun. Un sprint 0 a été réalisé afin de mettre en place l'architecture, le processus de développement et la première version du Backlog.

Il a été décidé que la répartition des roles de Product owner et de Scrum Master changent à chaque Sprint afin que tout le monde puisse essayer chacun des roles.

Une revue et une retrospective ont été effectuées à la fin de chaque Sprint. Le tableau ci-dessous rescence les différents sprints associés aux roles et aux date de revues et retrospectives.

3.2 Planning Poker et vélocité

Pour attribuer un nombre de points aux issues, l'équipe Scrum à utilisé le planning poker. Celui-ci a été effectué grace à un outil disponibile gratuitement sur internet, Pointing Poker (https ://www.pointingpoke

Après avoir terminé un sprint, il nous était donc possible de calculer la vélocité de celui-ci. Le plugin ZenHub nous a permis de représenter facilement et de manière graphique la courbe Burn-Down de notre vélocité.

Ci-dessous, le burn-down représentant le premier sprint :

3.3 Objectifs des sprints

3.4 Définition de fini

3.5 Releases

4 Architecture