ABBES Mohamed
LAFON Victor
MARTINEZ Loïc
RATSIMBAZAFY Randy

Documentation projet IVVQ : ChessMind

Le but de ce projet est de développer une application Grails, avec pour objectif principal de mettre en place le workflow IVVQ (Intégration, Vérification, Validation, Qualification). Pour ce projet, nous avons constitué une équipe de quatre développeurs afin de réaliser un site web de type réseau social, ayant pour thème la résolution de parties de jeu d'échecs.

Informations techniques

URL du projet Github : https://github.com/Metabolic24/ChessMind
URL de l'intégration continue Travis CI : https://travis-ci.org/Metabolic24/ChessMind
URL du déploiement Heroku : http://chessmind.herokuapp.com

Identifiants des contributeurs :

- ABBES Mohamed : Abbess ;
- LAFON Victor : Metabolic24 ;
- MARTINEZ Loïc : gentuxx ;
- RATSIMBAZAFY Randy : rrandy .

Exposition méthode

Contexte et Outils

Les contraintes techniques pour le projet ont été d'utiliser le framework Grails 2.3.X, pour lequel nous avons choisi la version 2.3.11 qui est la plus récente, et de développer un projet Open Source utilisant notamment des outils comme Git, Github et des outils de tests et de déploiement.

Le framework de tests unitaires utilisé est le framework Spock, un framework qui peut tester le langage Groovy, et qui s'intègre idéalement avec Grails. Il est exécuté par un JUnit runner, et par rapport à JUnit, il permet un découpage logique du scénario de test avec des annotations comme "given, when, then", qui séparent l'état initial, de l'évènement et du résultat attendu. Il apporte donc une meilleure visibilité pour être lu par une personne.

Le site devait intégrer les principales fonctionnalités d'un site web orienté réseaux sociaux, comme la gestion des membres de la communauté, la gestion des items liant la communauté, ou la gestion des participations à la communauté au contenu du site. Pour cela, nous avons choisi comme thème les jeux d'échecs, en développent un site où les membres pourraient proposer des problèmes d'échecs, et les membres de la communauté pourraient proposer des solutions et interagir avec le site.

Méthode de développement

Pour organiser le développement du projet, nous avons choisi d'utiliser une méthode orientée agile, basée sur la méthode Scrum, qui s'oriente principalement sur un découpage du travail en Sprints, et des réunions régulières pour vérifier l'avancement du projet et proposer des améliorations.

Nous avons adapté l'organisation en fonction des contraintes auxquelles nous étions soumises. Ainsi, nous n'avons pas fait de mêlée quotidienne car nous ne pouvions nous voir directement chaque jour d'un sprint. En revanche, nous avons mis en place un groupe Facebook et un Google Drive pour faciliter la communication et l'échange de données, hors code source.

Au début du projet, nous avons effectué un sprint 0, d'une durée d'environ une semaine et demie, où nous avons établi le backlog produit et préparé l'environnement de développement et de gestion de projet. Par la suite, nous avons réalisé des sprints d'une durée fixe d'une semaine. Ce choix de délai assez court est venu de la nécessité de travailler efficacement vu le temps imparti pour la réalisation du projet, et cela a permis de faire remonter plus rapidement les problèmes et donc de les corriger également plus vite.

Chaque sprint a été précédé d'une réunion de préparation du sprint, où nous avons décidé des users stories à intégrer dans le backlog de sprint. De plus, chacun s'est terminé par une revue de sprint pour faire le point sur l'avancement du projet et une rétrospective de sprint, pour identifier les méthodes à mettre en place pour améliorer notre démarche de travail.

Gestion du code source et des tâches

Concernant le code source, nous utilisons Git comme gestionnaire de versions, conjointement avec la forge en ligne Github. Cette combinaison nous a permis de collaborer facilement sur le développement du code via l'outil Git, tout en ayant via Github un système d'issues de gérer les user stories, tâches et bugs associés à notre projet. Github fournit également la structure de type milestone, correspondant à un sprint, et propose la visualisation de l'avancement et la gestion des délais.

Au début de chaque sprint, nous avons convenu de sélectionner les tâches que devaient effectuer chaque membre du projet, et nous avons ainsi chacun développé sur l'ensemble du

projet plusieurs aspects, comme la gestion des membres, la gestion des scores ou la gestion des problèmes proposés.

Concernant les branches au sein de Git, nous avons choisi une structure simple en deux branches : master, correspondant au dernier état stable du projet, et dev, correspondant à son état le plus récent. Tous les développements se sont effectués sur la branche dev, et afin de répercuter les modifications de dev vers master, nous avons, à chaque fin de sprint, fusionné, mergé ces branches, permettant ainsi une intégration par Travis et un déploiement sur Heroku en prévision de la revue de sprint.

Outre ces deux branches, nous avons parfois utilisé d'autres branches, dans le cas du développement de fonctionnalités importantes pouvant affecter le fonctionnement d'autres éléments du logiciel. Dans ce cas, une branche locale et remote ont été créées et le merge s'est fait dès que possible vers dev, avec au final suppression de la branche créée.

Stratégies de tests et qualité

Afin de contrôler la qualité de notre logiciel, nous nous sommes basés sur plusieurs critères. Tout d'abord, les tests unitaires réalisés par le biais du framework Spock et dont le pourcentage minimum à atteindre a été fixé à 80%, afin de couvrir la majorité du projet tout en ayant une marge pour certains tests problématiques. Afin d'évaluer ce taux, nous avons exploité le potentiel de Cobertura, qui fournit des rapports assez précis sur l'avancement et la couverture des tests.

Notre stratégie de tests a été assez classique car nous avons essayé de développer et faire passer les tests au fur et à mesure du code développé. Malgré certains blocages dûs à la présence importante de certains plugins dans certaines classes, nous avons réussi à garder un rythme assez constant concernant la mise en place des tests unitaires, nous permettant d'approximer de manière quasi-constante 70% de couverture.

Concernant la qualité du code, nous utilisons l'outil CodeNarc qui propose une analyse de la syntaxe du code et fournit des rapports en résultant.
L'utilisation de l'outil SonarQube permet de mesurer la qualité globale du projet, en réutilisant notamment les outils Cobertura et CodeNarc avec une visualisation unifiée.

Automatisation et déploiement

Afin d'automatiser le flow de builds, tests et déploiement, nous avons utilisé l'outil Travis CI, un serveur d'intégration continue qui repose sur des projets hébergés sur Github. Cet outil a permis, via le fichier .travis.yml, de définir des environnements de compilation pour ensuite permettre, à chaque commit, de construire et tester l'ensemble du projet. Nous avons limité ces opérations à la branche master, afin d'éviter un trop grand nombre d'échecs pendant les développements sur dev, et pour mieux identifier les sources d'erreurs.

Pour l'automatisation du workflow, nous avons défini plusieurs jobs Travis CI. Tout d'abord nous avons défini le build du projet sur plusieurs JDKs; les JDKs oraclejdk7 et oraclejdk8 car ce sont les deux dernières versions officielles d'Oracle, ainsi que l'openjdk7 pour garantir une compatibilité avec une autre implémentation du JDK, notamment l'implémentation libre.

Ensuite, nous avons rendu exécutable le script grailsw pour pouvoir exécuter les commandes Grails, et nous avons défini l'exécution du rafraîchissement des dépendances pour s'assurer que les librairies nécessaires étaient bien présentes, ainsi qu'exécuté les tests Grails via la commande test-app.

Pour finir, nous avons défini le déploiement sur Heroku via la clé API fournie par le site. Heroku est une PaaS, Platform as a Service, qui permet de le déploiement rapide d'un site sur internet via l'outil Git. De plus, nous avons spécifié que la branche master de Github devait être déployée, car c'est la dernière version en état de fonctionnement.

Architecture technique applicative

Le site web est développé via Grails, un framework complet pour le développement d'applications web, alliant le langage Groovy, et d'autres outils comme Spring et Hibernate.

Le langage Groovy est utilisé sur l'ensemble du projet, que ce soit au niveau du fonctionnement de l'application que des tests. L'application suit le modèle MVC Modèle Vue Controlêur, et est divisé ainsi via les dossiers grails-app/domain, grails-app/controllers, et grails-app/views.

Les vues utilisent principalement le langage GSP (Groovy Server Page), qui est un langage qui permet d'interfacer le Groovy pour afficher une page web, et donc les vues utilisent également les langages HTML, CSS et javascript.

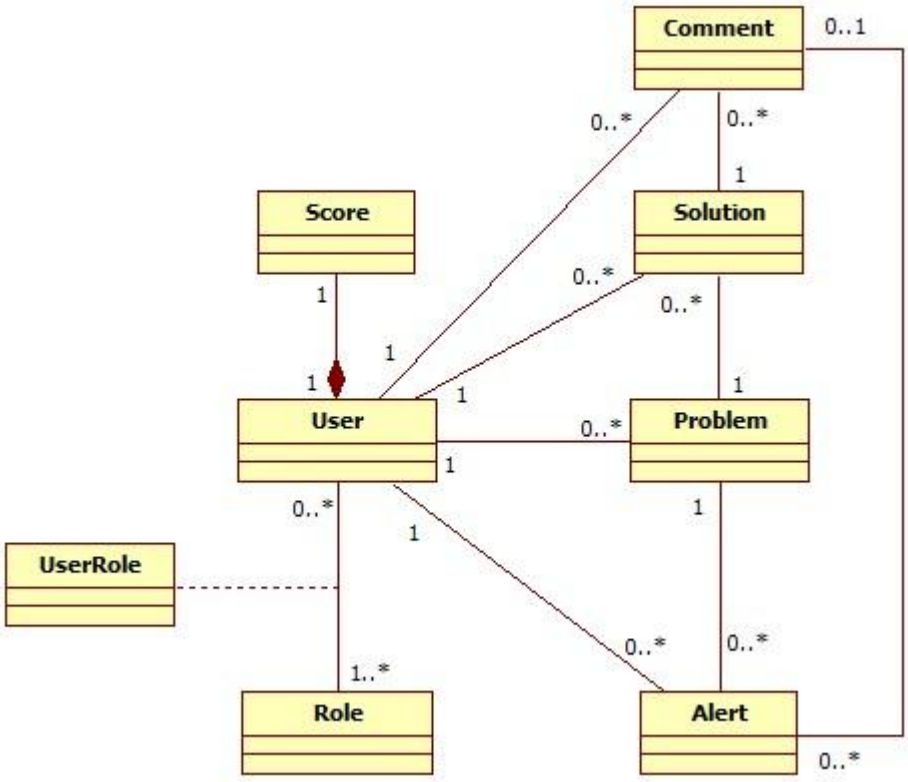
La gestion des utilisateurs et l'authentification du site est gérée par le plugin Spring Security. C'est un plugin qui permet de créer des utilisateurs avec des rôles différents, pour pouvoir gérer leur droits d'accès au niveau des ressources sur l'ensemble du site.

Les classes principales de l'application sont définies dans le dossier grails-app.
Les classes du domaine permettant de définir quelles sont les entités définies sont réparties dans les dossiers suivants :

- domain
    - alert
        - Alert : une alerte de non respect des règles pour les commentaires ou les problèmes
    - problems
        - Comment : un commentaire d'une solution
        - Problem : un problème de jeu d'échecs à résoudre

        - Solution : une solution proposée par les membres
    - score
        - Score : un score gérant les points gagnées par un membre
    - users
        - Role : le rôle d'un utilisateur du site
        - User : un utilisateur du site
        - UserRole : l'association entre en utilisateur et un rôle

Le diagramme suivant décrit les interactions entre les différentes classes du domaines :



Chaque classe du domaine a un contrôleur correspondant dans le dossier controllers, qui définit les opérations faisables par les entités et les interactions avec les vues.
De même, chaque vue pour les domaines est définie dans le dossier views. En plus des vues permettant d'afficher et de gérer chaque classe du domaine, des fichiers supplémentaires gèrent l'affichage du site, la gestion des connexions, etc.

En dehors du dossier grails-app, le dossier test contient tous les tests du projet, et le dossier doc contient la documentation projet ainsi que les rapports de qualités.

Particularité technique importante

Ce projet présente plus particulièrement une interface permettant de proposer des problèmes pour des jeux d'échecs. Le site se situe entre un site de réseau social, et un site de jeu ou de plateforme collaborative. Chaque membre peut proposer une configuration de jeu d'échecs qui pose un "problème" de résolution. Les autres membres peuvent alors commenter le problème, répondre au problème en proposant une solution, et suivant les autres solutions existantes, "aimer" ou commenter les solutions.

Le fait que les membres aiment une solution rajoute des points à celle-ci, ce qui augmente le score du membre l'ayant proposée. Le joueur ayant proposé le problème peut finalement choisir la solution qui lui semble être la meilleure, et ceci entraîne également un gain de point.

Un affichage des scores est disponible, et rend compte de l'avancement du score de chaque membre, avec un classement des points suivant un des deux critères (nombre de "j'aime" des solutions, ou nombre de meilleures solutions choisies).

Il existe trois différents quatre de profils, définis avec SpringSecurity :

- l'administrateur, propriétaire du site, qui a le maximum de droits, dont la gestion des utilisateurs et des problèmes
- les modérateurs, désignés par l'administrateur, qui ont plus de droits que les membres normaux, notamment en ce qui concerne la gestion et la validation des problèmes
- les membres, qui peuvent proposer un problème, proposer une solution, commenter un problème et une solution, et aimer un commentaire
- les visiteurs anonymes, qui peuvent simplement visiter certaines pages

Suivant leurs rôles, les utilisateurs peuvent soit avoir accès ou non à certaines pages, ou bien à l'intérieur même d'une page avoir accès à différentes fonctionnalités.

L'interaction entre les problèmes et les solutions est la suivante, un problème est soumis par les membre sous forme d'une image qui représente la partie d'échec à un moment donné, et de diverses informations comme une description, un endroit et un tournois, ainsi que noms des joueurs blancs et noirs. Les solutions quant à elles sont formatées suivant un schéma précis utilisant des expressions régulières, sous forme d'une chaîne de caractère qui peut spécifier quels sont les pions et leurs places sur l'échiquier.

Un système de validation de problème a été mis en place, pour pouvoir filtrer et accepter les problèmes suivant leur conformité aux règles du site. A la création d'un problème par un membre, celui-ci est En attente de validation. Ce sont les modérateurs et l'administrateur qui ont la responsabilité de choisir si un problème est valide ou non. C'est seulement une fois que le problème est validé qu'il peut être soumis à la communauté, que les autres membres peuvent le commenter, ou proposer une solution à ce problème.

De plus, dans le cas d'un commentaire irrespectueux ou d'un problème qui contiendrait des erreurs ou qui pourrait être amélioré, un système d'alertes a été mis en place. Pour chaque problème et chaque commentaire, il est possible d'émettre une alerte en précisant quel est le motif par une description, et ces alertes seront relayées aux modérateurs et à l'administrateur pour prendre les mesures nécessaires.

Rapports qualités

Les rapports qualités générés ont été placé dans le dossier doc du projet Github. Ces rapports ont été générés pour chaque sprint, suivant la disponibilité en xml ou en html. Ils ont été générés via les outils suivants : Cobertura, CodeNarc et SonarQube.