Projet Intégration continue - Création de groupes d'utilisateurs.

Tom Avenel

2022 / 2023

Résumé

Votre équipe de développement a reçu une nouvelle commande de la part d'un client souhaitant développer une application Web pour permettre à ses utilisateurs de créer des groupes de travail.

Table des matières

Présentation du projet **1**
- Administrateur . . . 1
- Utilisateur . . . 1
- Vérifications . . . 2

Limitations **3**

Améliorations **5**

Outils d'intégration continue **7**
- Le gestionnaire de versions . . . 7
- Les tests unitaires et de comportement . . . 7
- La qualité du code . . . 8
- Un pipeline intégré . . . 8

Travail à réaliser **9**

Rendus attendus **11**

Exemple (non exhaustif) de rapport de votre étude : **13**
- Partie Intégration Continue : . . . 13
- Partie tests : . . . 13

Legal **15**

Présentation du projet

Le projet consiste à développer une application Web permettant à ses utilisateurs de créer des groupes de travail.

Le détail du cahier des charges est le suivant - en cas de question sur les spécifications, le formateur jouera le rôle du client final pour préciser celles-ci.

Administrateur

Le compte administrateur crée la config des groupes, c'est-à-dire qu'il définit :

- Le nombre d'utilisateurs
- Le nombre de groupes
- La configuration du dernier groupe : `LAST_MIN` vs `LAST_MAX` :
  - Si le nombre d'utilisateurs est un multiple du nombre de groupes (ex : 20 utilisateurs et 5 groupes => 5 groupes de 4), ce paramètre n'a pas d'incidence.
  - Si la configuration vaut `LAST_MIN`, le dernier groupe a moins d'utilisateurs que les autres (ex : 19 utilisateurs et 5 groupes => 4 groupes de 4 et 1 groupe de 3)
  - Si la configuration vaut `LAST_MAX`, le dernier groupe a plus d'utilisateurs que les autres (ex : 19 utilisateurs et 5 groupes => 5 groupes de 3 et 1 groupe de 4)

Utilisateur

Les utilisateurs doivent pouvoir réaliser les actions suivantes :

- Lister les utilisateurs sans groupe
- Demander à créer un nouveau groupe => Création d'un lien d'invitation à partager. Un groupe ne sera réellement créé que lorsqu'au moins une personne accepte l'invitation (jamais de groupe avec un seul utilisateur).
- Rejoindre un groupe spécifique déjà créé mais pas encore complet en utilisant le lien d'invitation
- Demander à être placé automatiquement dans un groupe aléatoire
- Membre d'un groupe : afficher son groupe et les membres du groupe, se retirer du groupe

Vérifications

- Pas trop de groupes => lorsque tous les groupes sont créés, il faut obligatoirement rejoindre un groupe
- Remplissage du dernier groupe => un utilisateur peut être contraint à rejoindre un groupe
- Placement automatique => en cas de placement automatique, l'utilisateur est soit placé dans un groupe non complet, sinon un nouveau groupe est créé si tous les groupes sont complets
- Un groupe ne peut pas avoir moins de 2 utilisateurs sinon il doit être supprimé
- Vérifier la cohérence du modèle => le nombre de groupes est-il correct ?

Limitations

A ce stade du projet, on pourra se limiter aux développements suivants :

- Il n'est pas demandé de gérer les comptes des utilisateurs : on pourra se limiter à un simple identifiant de connexion sans mot de passe.
- Les groupes peuvent être simplement numérotés groupe 1, groupe 2, ...
- Le compte administrateur sera un identifiant `admin` stocké en dur dans le code

Améliorations

Une fois les fonctionnalités principales implémentées, on pourra ajouter les suivantes :

- Interface d'administration : l'administrateur possède une vue d'administration sur le programme lui permettant de voir la liste des groupes constitués.
- Modification des groupes : l'administrateur peut assigner ou retirer des utilisateurs d'un groupe, détruire un groupe existant, créer un groupe et le remplir avec ses utilisateurs
- Ajout ou retrait dynamique d'utilisateurs pendant le fonctionnement du programme :
  - Le nombre total d'utilisateurs change
  - Il faut supprimer l'utilisateur d'un groupe s'il en faisait partie
  - Le nombre total de groupes change
- Exceptions de groupe : l'administrateur peut changer le nombre d'utilisateurs dans un groupe, ou forcer le nombre de groupes durant l'exécution du programme
- Contraintes de liens : ajouter des métadonnées sur les utilisateurs pour : interdire des utilisateurs d'être dans le même groupe, obliger des utilisateurs à être dans le même groupe.

- Création de comptes utilisateurs : les utilisateurs disposent d'un véritable compte, avec un login par identifiant / mot de passe

Après étude des besoins du client, vous décidez de développer cette application suivant un processus d'intégration continue, afin d'accélérer le développement du projet et pour garantir la qualité des fonctionnalités implémentées.

Outils d'intégration continue

Le gestionnaire de versions

Le gestionnaire de versions a 2 objectifs principaux dans une intégration continue :

- Pouvoir partager les modifications de code entre les développeurs du projet de manière sûre.
- Avoir une référence de code stable pour tester la qualité.

En intégration continue, il est important d'intégrer le plus régulièrement possible ses modifications (dans un commit) afin de limiter les changements à tester / valider.

- Utiliser le gestionnaire de versions `git` pour intégrer et partager les modifications du projet au sein du binôme. On pourra utiliser une version hébergée (`Github`, `Bitbucket`, `Gitlab`, ...)
- Configurer votre environnement de développement (IDE) pour associer le versionning Git au projet.

Les tests unitaires et de comportement

La métrique principale dans un pipeline d'intégration continue est souvent le rapport de test.

Celui-ci permet de valider les modifications d'un commit et d'éviter les régressions.

Intégrer un framework de tests unitaires au projet - par exemple en `Java`, on pourra utiliser le framework `Junit`, en `Python` le framework `Pyunit`, ...

Il est très compliqué de définir une couverture de test nécessaire et suffisante, car cela dépend énormement du code à tester : on privilégiera donc toujours la qualité du test sur les statistiques de sa couverture de code.

Les classes ayant des dépendances externes sont en général difficiles à tester. Au contraire, les classes contenant majoritairement du code métier sont les plus critiques et celles à tester le plus en profondeur dans les tests unitaires.

- On activera la couverture de test dans l'IDE afin de vérifier que les classes sont bien testées.
- On ajoutera les tests unitaires et/ou de comportement nécessaires.
- On modifiera le code source de l'application pour corriger les bugs trouvés au fur et à mesure du développement.
- On pourra également lancer l'application et réaliser des tests de fonctionalité à la main.

La qualité du code

La qualité du code ne se limite pas aux tests !

De nombreuses métriques peuvent être mises en place pour limiter les bugs et faciliter la maintenance du code.

Par exemple, pour simplifier la compréhension du code on limitera la profondeur d'heritage dans les classes Java du projet à 2.

- Utiliser les outils d'analyse de l'IDE pour détecter d'éventuels problèmes dans l'application et améliorer la qualité de celle-ci.
- En plus de la profondeur d'héritage, on choisira quelles analyses paraissent pertinentes. Par exemple : $<$ 15 lignes de code dupliquées

Un pipeline intégré

Les outils précédents, bien que très efficaces, nous ont obligé à configurer notre IDE de manière adéquate.

Cela peut poser problème lorsque le projet devient conséquent, lors de l'arrivée d'un nouveau membre dans l'équipe, ou en cas de changement d'IDE : les critères de qualité risquent de diverger au sein de l'équipe.

Pour éviter cela, on peut essayer de partager au maximum ces critères de 2 manières :

- En intégrant les outils de vérification de la qualité dans les sources du projet
- En déployant un serveur d'intégration dédié

Ces 2 méthodes ne sont pas incompatibles.

- Intégrer les outils précédents dans un serveur d'intégration continue. On pourra également utiliser un outil de build pour faire le lien entre l'environnement de développement et l'environnement d'intégration continue : `Gradle`, ...

Travail à réaliser

Dans le respect du cahier des charges, réaliser :

- La définition d'un pipeline d'intégration continue pour le développement d'une application respectant ce cahier des charges (tests unitaires et d'intégration, analyse de code source, création des binaires, publication des rapports, ...). Il n'est pas demandé de réaliser un déploiement continue (ni génération d'artéfacts de production, ni livraison, ni déploiement).
- La mise en place d'un serveur Jenkins correctement administré et réalisant le pipeline d'intégration continue défini précédemment. Les différents jobs configurés dans Jenkins devront s'interfacer avec les outils d'intégration continue.
- L'application sera testée dans plusieurs environnements, par exemple :
  - Firefox
  - Google Chrome
  - Microsoft Edge
- Le développement de l'application en suivant ce processus d'intégration. Le choix des technologies de développement est laissé libre.
- La mise en place d'outils de l'intégration continue sur la machine du développeur (tests unitaires, analyse statique, formatage du code, ...) pour les technologies choisies lors du développement. **Ces outils devront être documentés dans le rapport du projet**.
- L'intégration des rapports des tests automatisés dans Jenkins.

Le projet étant réalisé en intégration continue, on sera particulièrement vigilant :

- à utiliser un outil de gestion des versions du code source (`git`) et à intégrer ses changements le plus souvent possible.
- à travailler sur des branches git dédiées avant d'intégrer ses changements dans la branche commune (`master`, `main`) et à réaliser des revues de code avant d'intégrer les changements.
- à penser aux différents tests dès le début du projet, voir à écrire les tests avant l'implémentation du code (TDD, BDD).
- un soin tout particulier sera apporté aux tests unitaires qui devront être nombreux.

Rendus attendus

On fournira donc :

- Un rapport sur la réalisation de l'étude de cas, notamment sur l'intégration continue mise en place. Voir la partie suivante : *rapport de votre étude* ;
- Un ensemble de dépôts de code source référencés dans le rapport et contenant les différentes parties du projet. Ces dépôts de code devront contenir le code de production ainsi que le code de test ;
- Les captures d'écran de revues de code effectuées pour l'intégration de changements dans le projet ;
- Les captures d'écran de la configuration de Jenkins et des différents jobs et pipelines créés.

Le barême est le suivant :

- Partie Intégration Continue
  - 5 points pour le rapport ;
  - 5 points pour le schéma du pipeline d'intégration ;
  - 5 points pour le développement itératif (git, branches, revues de code) ;
  - 5 points pour le déploiement de Jenkins et la configuration des jobs ;
- Partie Automatisation des tests
  - 2 points pour le rapport ;
  - 5 points pour la mise en place du framework et son intégration dans les différents outils (environnement de développement, Jenkins) ;
  - 7 points pour la couverture de tests (nombre de tests, choix des scénarii, ...) ;
  - 6 points pour la qualité des tests (design patterns utilisés, lisibilité du code de test, ...).

Un bonus de 3 points sera accordé pour chacune des parties pour la qualité de l'application développée

Exemple (non exhaustif) de rapport de votre étude :

L'équipe projet (noms, prénoms) :

Partie Intégration Continue :

1. Quelle(s) technologie(s) avez-vous choisi d'utiliser pour le développement de l'application ?
2. Quels outils d'intégration continue avez-vous conseillé à l'équipe de développement d'installer sur leur machine personnelle ? Justifiez votre réponse.
3. Quels sont les différents dépôts de code source utilisés pour l'ensemble du projet ? Pourquoi avoir fait ce choix de découpage ?
4. Quel(s) pipeline(s) d'intégration continue avez-vous défini(s) pour gérer votre projet ?

- *Insérez un schéma des grandes étapes du processus complet d'intégration continue que vous avez définies, depuis l'environnement du développeur à la validation finale des changements.*
- *Il n'est pas demandé de réaliser le déploiement continu de l'application, ni de générer des binaires de production.*

5. Quels jobs et/ou pipelines avez-vous définis dans Jenkins ? Justifier ces choix.
6. Pour chaque job de Jenkins, fournir un screenshot des rapports affichés sur la page principale du job.

Partie tests :

1. Quel(s) framework(s) de tests avez-vous choisi d'utiliser ? Pourquoi ?
2. Avez-vous choisi de tester certaines fonctionnalités ou zones de code plus que d'autres ? Pourquoi ?
3. Avez-vous réalisé des tests manuels ? Si oui, lesquels ?
4. Description des plans de tests :

Legal

Ce document a été généré grâce à l'outil pandoc.

Les sources au format Markdown de ce document sont disponibles sur le site web.