INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de Projet

Programmation avancée en JAVA & Conduite de projet

HAMDAN Célian Hilal, 4IR-SC
DA COSTA BENTO Marie-Line, 4IR-SC

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de Projet

Programmation avancée en JAVA & Conduite de projet

HAMDAN Célian Hilal, 4IR-SC
DA COSTA BENTO Marie-Line, 4IR-SC

Tables des Matières

INTRODUCTION

Ce rapport présente le projet Maven développé dans le cadre du cours de *Programmation Avancée en JAVA/Conduite de projet*. L'objectif principal de ce projet était de concevoir un *système de clavardage* permettant l'interaction entre différents utilisateurs. Ce système inclut des fonctionnalités comme la gestion des connexions, la personnalisation des surnoms, la gestion des contacts, et l'envoi/réception de messages.

Le projet a été développé en suivant une méthode agile, à l'aide de l'outil JIRA: https://pdlacp.atlassian.net/jira/software/projects/CHAT/boards/3. L'organisation du code a été facilitée par l'utilisation de Git, tandis qu'une intégration continue (DevOps) a été mise en place pour automatiser les tests et valider les fonctionnalités tout au long du processus de développement. Ce rapport détaillera ces différents aspects du travail réalisé ainsi que les outils et méthodes utilisés tout au long du projet.

Le code source du projet est disponible sur le repository GitHub suivant: https://github.com/insa-4ir-chatsystem/chatsystem-hamdan-da-costa-bento.

I - MÉTHODE AGILE

Pour la gestion de notre projet, nous avons appliqué une méthodologie agile basée sur le modèle SCRUM en utilisant JIRA. Voici les étapes principales :

A - Mise en place initiale

Nous avons structuré le projet autour de trois epics :

- **Contact Discovery** : Mise en place de la recherche et de la gestion des contacts.
- **Intermediate Evaluation** : Implémentation des mécanismes d'évaluation intermédiaire des fonctionnalités.
- **Implementation** : Réalisation des fonctionnalités principales telles que l'envoi de messages, les connexions et les nicknames.

Ces epics ont été déclinés en user stories selon le format : *As a {type of user}, I want {the goal} so that {the reason}; la première user story est la suivante par exemple: As a client, I want to be a user so that I can ask for help.*

B - Organisation des sprints

- **Sprint planning** : Nous avons estimé les efforts des tâches en attribuant des **story points** via la méthode **Planning Poker** et ajouté des sous-tâches au besoin.
- **Durée des sprints** : Chaque sprint a duré deux semaines pour permettre une progression cohérente.
- **Répartition des tâches** : Les tâches ont été assignées équitablement entre les membres de l'équipe, nous avons également décidé de fixer la capacité de chaque sprint entre 8 et 12 points, correspondant à entre 4 et 6 story points par personne, selon les sprints. Nous avons terminé le projet avec chacun 26 story points.
- **Adaptations** : Certaines tâches ont été ajoutées en cours de sprint pour combler des oublis ou ajuster les priorités.

C - Cycle des sprints

À la fin de chaque sprint, nous avons suivi le processus suivant :

1. Une **Sprint review & retrospective** pour évaluer le travail effectué et pour discuter des points à améliorer.
2. Un nouveau **Sprint planning** pour préparer le sprint suivant.

II - ORGANISATION DE GIT

A - Arborescence du projet

Nous avons conçu une arborescence claire et pratique pour une gestion efficace du projet :

- .github/workflows : Scripts pour l'intégration continue, garantissant des livraisons fiables.
- project/src/main/java : Code source principal, structuré par modules fonctionnels (database, request, review, ui, users) pour simplifier la maintenance.
- project/src/main/ressources : Contient les fichiers statiques nécessaires à l'application, comme les images ou les fichiers audio.
- project/src/test/java : Tests unitaires, organisés par module pour isoler et tracer facilement les tests.
- project/pom.xml : Fichier central pour configurer le build, gérer les dépendances et les plugins.
- .gitignore : Liste des fichiers à ignorer par Git pour éviter les fichiers inutiles dans le dépôt.
- README.md : Documentation pour présenter rapidement le projet.

B - Gestion du code avec Git

Pour la gestion du code source, nous avons utilisé **Git** avec une stratégie adaptée au travail à deux :

- **Suivi des modifications** : Chaque commit est soigneusement documenté, ce qui facilite le retour sur les changements réalisés.
- **Développement structuré** : Bien que nous n'ayons pas utilisé de branches pour structurer le développement, nous avons mis en place une discipline stricte dans la documentation et l'organisation des commits. Cette approche a permis de maintenir une base de code stable et compréhensible, tout en facilitant les révisions et le suivi des modifications. Les choix pris ont permis un développement rapide et efficace, adapté à la taille de l'équipe et aux objectifs du projet.

III - MISE EN PLACE DE L'INTÉGRATION CONTINUE (DEVOPS)

A - Gestion des dépendances avec Maven

Nous avons utilisé **Maven** pour gérer les dépendances, compiler le projet et exécuter les tests. Voici les dépendances principales définies dans pom.xml :

- **JUnit Jupiter:** Pour l'écriture et l'exécution des tests unitaires.

- **Log4j:** Pour la journalisation des événements du système.
- **SQLite JDBC:** Pour la connexion à une base de données SQLite.
- **JLayer:** Pour le traitement audio dans le projet.

Nous avons également ajouté les plugins suivants pour automatiser certaines tâches :

- **Surefire Plugin** : Exécution des tests unitaires.
- **Failsafe Plugin 3.1.2 :** Exécution des tests d'intégration.
- **Maven Jar Plugin** : Création de fichiers JAR exécutables.

B - Tests unitaires

Nous avons utilisé **JUnit Jupiter** pour valider les différentes fonctionnalités du projet. Voici les étapes principales :

1. **Structure des tests** : chaque module dispose de son propre fichier de test : databaseTest, requestTest, etc. Les tests suivent le format *AAA* : Arrange, Act, Assert.
2. **Exécution automatique** : Les tests sont déclenchés automatiquement grâce au plugin **Surefire,** configuré dans le fichier maven.yml.

C - Automatisation de la compilation, des tests et des packages avec GitHub Actions

Nous avons configuré des workflows GitHub Actions pour automatiser les étapes critiques :

1. **Compilation** : Vérification que le code compile correctement à chaque push.
2. **Tests** : Exécution des tests unitaires pour garantir la stabilité du projet.
3. **Packaging** : Génération d'un fichier JAR pour chaque version stable.
4. **Livraison continue** : Préparation automatique des artefacts prêts à être déployés.

CONCLUSION

Ce projet a permis de concevoir et de développer une application de clavardage favorisant les interactions en temps réel entre différents utilisateurs. L'application de la méthode agile, combinée à l'utilisation des outils JIRA, Git et Maven, a assuré une organisation structurée, une gestion optimale du code et une intégration continue efficace. Ces pratiques ont contribué à maintenir une qualité élevée tout au long du développement. En définitive, ce projet a répondu aux objectifs initiaux tout en consolidant nos compétences en gestion de projet, en méthode agile et en développement Java avancé.

TABLE DES ANNEXES

ANNEXE 1 : Exemple de sprint planning

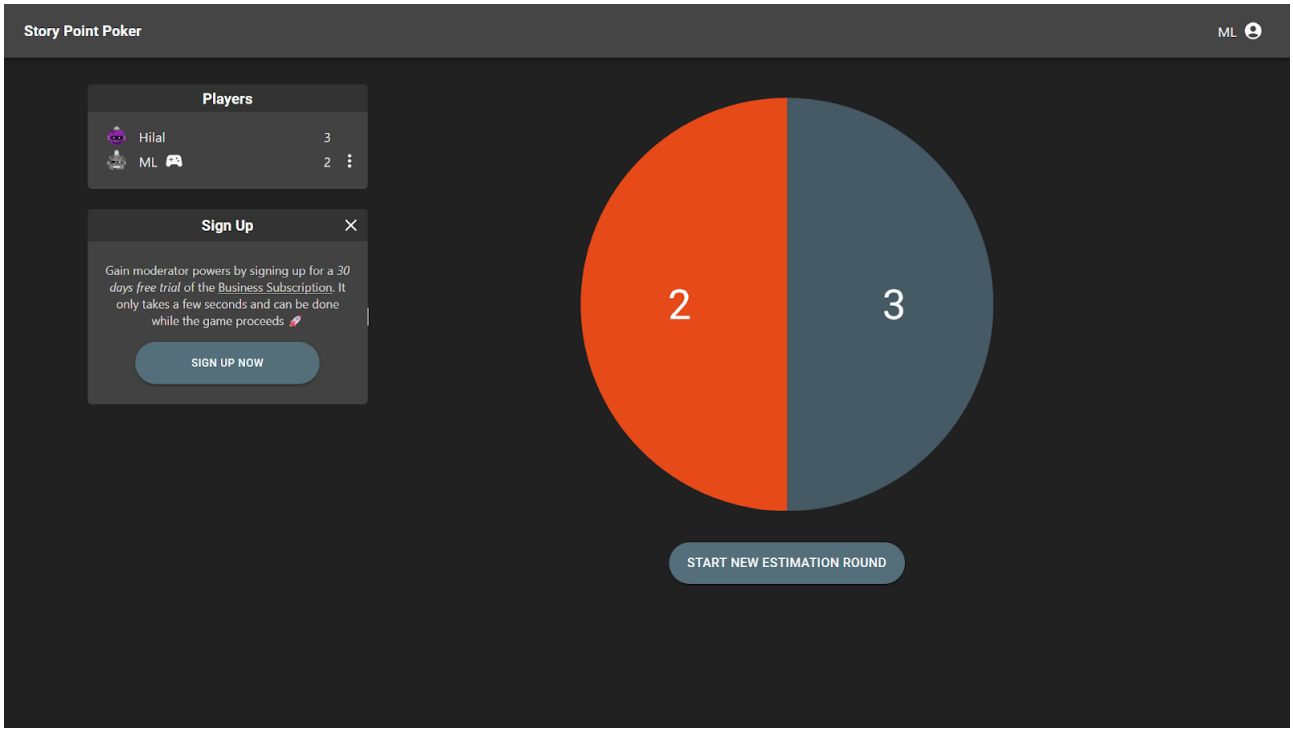

Figure 1 : Planning poker method pour la tâche CHAT-4 avec https://app.storypoint.poker

Type	Summary	Assignee	Story point estimate	
☑	Use case Diagram	CH Célian Hilal Hamd...	2	···
🔖	As a user, I want to have a unique nickname	MB Marie-Line Da Co...	2	···
🔖	As a user, I want to be able to notice new users	CH Célian Hilal Hamd...	2	···
🔖	As a user, I want to be able to notice any change of connexion	CH Célian Hilal Hamd...	2	···
🔖	As a user, I want to be able to access to the list of active users	CH Célian Hilal Hamd...	2	···
🔖	As a user, I want to be able to connect on the platform	MB Marie-Line Da Co...	2	···
+ Create				6 of 6

Figure 2 : Sprint 1 planning sur JIRA

ANNEXE 2 : Bilan du travail pour chaque étudiant

Key	T	Summary	Assignee	Story point estimate
CHAT-4		As a user, I want to be able to connect on the platform	Marie-Line Da Costa Bento	2
CHAT-5		As a user, I want to be able to access to the list of active users	Célian Hilal Hamdan	2
CHAT-6		As a user, I want to be able to notice any change of connexion	Célian Hilal Hamdan	2
CHAT-7		As a user, I want to be able to notice new users	Célian Hilal Hamdan	2
CHAT-8		As a user, I want to have a unique nickname	Marie-Line Da Costa Bento	2
CHAT-17		Use case Diagram	Célian Hilal Hamdan	2
CHAT-18		Sequence Diagram	Célian Hilal Hamdan	3
CHAT-21		Class Diagram	Marie-Line Da Costa Bento	2
CHAT-22		Adding some assets	Célian Hilal Hamdan	1
CHAT-23		Test each created classes	Marie-Line Da Costa Bento	2
CHAT-27		Clear actual code	Célian Hilal Hamdan	3
CHAT-30		MAJ ReadME	Célian Hilal Hamdan	1
CHAT-31		As a user, I want to be able to chat with at least one person	Marie-Line Da Costa Bento	5
CHAT-32		As a user, I want to be able to access history of my messages with my contacts	Célian Hilal Hamdan	3
CHAT-33		Modifications Contact Discovery	Marie-Line Da Costa Bento	5
CHAT-36		Unit Tests	Marie-Line Da Costa Bento	1
CHAT-37		Wireframe interface graphique	Marie-Line Da Costa Bento	2
CHAT-38		Update of UML diagrams for last implementations	Marie-Line Da Costa Bento	1
CHAT-43		Ending what's left for the intermediate evaluation	Célian Hilal Hamdan	2
CHAT-51		As a user, I want a chat system with integrated message storage in a database, support for special files, and a graphical interface, so that I can have a persistent communication experience.	Célian Hilal Hamdan	5
CHAT-52		As a user, I want the chat system to include a graphical interface, so that I can interact with the chat system in an intuitive and user-friendly way.	Marie-Line Da Costa Bento	4

Figure 3 : Liste de nos backlog sur JIRA avec leur story points associés (lien)