Rapport du projet CAPI

Sommaire

1. Introduction

Le projet capi est un permet de simuler le déroulement réel d'une séance de planning pocker.

a. Péparation

Chaque participant reçoit un jeu de cartes Planning Poker

Les cartes contiennent généralement les valeurs 1, 2, 3, 5, 8, 13, 20, 40 et 100 (séquence de Fibonacci modifiée)

Ces valeurs représentent des story points, jours idéaux ou autres unités d'estimation

Certains jeux incluent aussi une carte "infini" pour les tâches très complexes et une carte "?" pour les tâches dont la complexité est difficile à estimer

b. Présentation du scénario

Le Product Owner ou client présente une user story ou fonctionnalité à estimer

Il explique en détail les attentes, l'objectif et l'impact du scénario

Un temps de questions/réponses est accordé pour que tous aient le même niveau d'information

c. Discussion et estimation

Les participants discutent de la fonctionnalité et posent des questions au Product Owner

Chacun réfléchit individuellement à son estimation

Une fois la discussion terminée, chaque estimateur choisit secrètement une carte représentant son estimation

d. Recherche du consensus

Si les estimations divergent, les participants avec les scores les plus extrêmes expliquent leur raisonnement

Une brève discussion (1-2 minutes) a lieu

Un nouveau vote est effectué si nécessaire

Le processus est répété jusqu'à l'obtention d'un consensus

2. Choix technique

Python a été choisi comme langage de backend pour sa lisibilité, sa riche bibliothèque standard, et son excellent support pour le développement web asynchrone. Les règles de validation (strict, majorité absolue, etc.) ont été implémentées comme des fonctions distinctes pour permettre une flexibilité maximale et faciliter les tests unitaires.

Nous avons choisi FastAPI comme framework backend pour plusieurs raisons :

- Performance élevée grâce à l'utilisation d'ASGI
- Génération automatique de documentation API avec Swagger UI
- Typage statique avec Pydantic, réduisant les erreurs potentielles

3. Intégration continue

Nous avons mis en place GitHub Actions pour automatiser nos tests et assurer la qualité du code. Notre workflow (***.github/workflows/python-app.yml***) effectue les actions suivantes à chaque push ou pull request :

- Configure l'environnement Python
- Installe les dépendances
- Exécute les tests unitaires avec pytest

Cette approche nous permet de détecter rapidement les régressions et maintenir une base de code stable.

4. Tests unitaires

Les tests unitaires sont une composante cruciale du développement logiciel, particulièrement dans un projet Agile comme CAPI. Voici les détails sur l'approche des tests unitaires dans ce projet :

a. Approche de test (pytest)

Le projet utilise pytest comme framework de test. Pytest a été choisi pour plusieurs raisons :

- Syntaxe simple et intuitive
- Fonctionnalités avancées comme le paramétrage des tests
- Bonne intégration avec d'autres outils Python

Les tests sont organisés dans le dossier backend/tests, reflétant la structure du code source dans backend/app.

b. Couverture des tests

Bien que le taux de couverture ne soit pas explicitement mentionné, l'objectif est de couvrir les fonctionnalités critiques du système, notamment :

- Les règles de validation (strict, majorité absolue, etc.)
- Les opérations sur le backlog
- Le processus de vote

Il est important de noter que viser un taux de couverture arbitraire (comme 80% ou 90%) n'est pas toujours pertinent. L'accent est mis sur la qualité et la pertinence des tests plutôt que sur un pourcentage brut.

c. Exemple de test unitaire commenté

Voici un exemple de test unitaire pour la règle de validation stricte :

```
def test_strict_rule():
    assert strict_rule([1, 1, 1]) == {"validated": True, "estimate": 1}
    assert strict_rule([1, 2, 1]) == {"validated": False, "estimate": None}
```

Ce test vérifie deux scénarios :

- Quand tous les votes sont identiques (1, 1, 1), la règle stricte doit valider l'estimation.
- Quand les votes diffèrent (1, 2, 1), la règle stricte ne doit pas valider l'estimation.

Ces tests assurent que la fonction strict_rule() fonctionne correctement dans différents cas d'utilisation, contribuant ainsi à la fiabilité globale du système de Planning Poker