Dépendances et packages

Pour ce projet, nous avons voulu améliorer la qualité de code et la maintenabilité. Pour ce faire, certains packages sont disponibles sur internet, voici ceux que nous avons sélectionnés :

- JUnit 5
- REST Assured
- Checkstyle
- Adminer
- JaCoCo
- PiTest
- Swagger
- SonarCloud
- Docker

JUnit 5

JUnit est un framework de test unitaire pour les projets Java. Il permet de créer et d'exécuter des tests automatisés pour vérifier le bon fonctionnement des différentes parties de notre code. Nous l'avons utilisé pour :

- Écrire des tests unitaires pour nos classes et méthodes
- Automatiser l'exécution de ces tests
- Vérifier que notre code produit les résultats attendus
- Détecter rapidement les erreurs et les bugs pendant le développement

Pour lancer tous les tests (Besoin d'une base de donnée UP) :

```
mvn test
```

Pour lancer tous les tests unitaires :

```
mvn test -Dtest=*UnitTest
```

Pour lancer tous les tests End-To-End (Besoin d'une base de donnée UP) :

```
mvn test -Dtest=*E2ETest
```

REST Assured

REST Assured est une librairie disponible pour Java et Ruby permettant d'effectuer des requêtes REST simplement et prenant la forme de tests similaires à des tests unitaires. Son utilisation est pratique lorsque

l'on souhaite effectuer des tests End to End sur son application. Nous l'avons utilisé pour :

- Ecrire des tests e2e
- Automatiser ces tests
- Vérifier que le code produit les comportements attendus par les différents scénarios du sujet
- Tester le CRUD des différents model
- Détecter les erreurs de comportements non volontaires
- Détecter les codes HTTP qui ne correspondaient pas ou n'étaient pas clairs

La syntaxe de REST Assured respecte la notation Gherkin, elle permet d'être assez clair pour le lecteur, à condition de ne pas avoir trop de tests dans la même classe :

```
given().
        param("x", "y").
        header("z", "w").
when().
Method().
Then().
        statusCode(XXX).
        body("x, ”y", equalTo("z"));
```

Checkstyle

Checkstyle est un outil permettant la vérification des styles pour un développeur. Il vérifie les bonnes pratiques et l'utilisation de méthodes et fonctions non dépréciés par exemple.

A la fin de son travail, il génère un rapport permettant de comprendre ce qui ne va pas.

Pour l'utiliser :

```
mvn checkstyle:check
```

Adminer

Adminer est une interface de gestion de base de données très simplifiée et légère en PHP qui permet de visualiser et éditer sa base de manière très facile.

JaCoCo

Il s'agit d'un outil de couverture de code pour Java, il est utilisé pour mesurer la quantité de code qui a été exécutée lors de l'exécution de tests automatisés. Cela nous permet de trouver et d'éliminer les morceaux de code qui n'ont pas été testés. Dans notre cas, nous l'avons utilisé pour :

- L'évaluation de la qualité des tests
- Détection des zones non testées
- Intégration continue : Il s'agit d'une partie nous permettant de vérifier cette couverture avant chaque pull request ou push sur notre branche de développement et principale.

- Rapports : Il permet également de créer des rapports visible sous la forme d'un site html, permettant d'avoir un visuel sur ce qui va et ne va pas.

Il est lancé automatiquement lors de l'exécution de test.

PiTest

Pitest est un outil de couverture de mutation pour les projets Java. Il mesure la qualité de nos tests en générant des mutations dans le code source et vérifie si nos tests automatisés détectent ces mutations. Dans notre cas, il est complémentaire et ajoute une couche à JaCoCo. Nous l'avons utilisé pour :

- Évaluer la qualité de nos tests : Il permet de modifier notre code afin de vérifier la solidité de notre code en cas d'oublis de certains éléments. Exemple : Suppression de ligne, Suppression d'un paramètre, etc...
- Détecter les zones non testées
- Intégration continue : Comme pour JaCoCo, il nous est possible de vérifier tout cela dans notre Intégration continue et d'ajouter un trigger permettant de vérifier le taux de couverture et de mutation.
- Créer des rapports visuels pour une meilleure compréhension de la couverture de mutation

Pour lancer un test de mutation, il faut avoir tout les tests au vert :

```
mvn org.pitest:pitest-maven:mutationCoverage
```

Swagger

Swagger est un outil qui permet de documenter et de tester les API. Il offre une manière de décrire les points d'entrée d'une API, les opérations qu'ils supportent, les paramètres nécessaires et les réponses possibles. Nous l'avons utilisé pour :

- Documenter clairement notre API, ce qui la rend facilement compréhensible pour les développeurs et les utilisateurs.
- Générer automatiquement une interface web pour tester notre API.
- Valider automatiquement les requêtes et les réponses de l'API.

SonarCloud

SonarCloud est une plateforme d'analyse de code qui vise à améliorer la qualité du code. Il offre des fonctionnalités de détection des erreurs, de mesure de la qualité, de suivi des problèmes de sécurité et de maintenabilité du code. Nous l'avons utilisé pour :

- Identifier et signaler les problèmes vulnérabilités de sécurité et les violations des normes de codage.
- Encourager les meilleures pratiques de développement en mettant en évidence les zones qui nécessitent une amélioration.

Tout est automatiquement lancé grâce à SonarCloud.

Docker

Docker nous permet de créer des images de l'application, de la base de données et d'adminer afin d'avoir un environnement similaire à toutes les exécutions. Si le conteneur fonctionne pour quelqu'un, il fonctionnera forcément pour tout le monde

API REST

Nous avons fait le choix d'utiliser Swagger afin de fournir à l'utilisateur une interface web affichant et décrivant notre API.

Celle-ci est dite "RESTFUL" puisque les méthodes suivent la norme RFC2616 GET permettent d'obtenir, POST de créer, PUT de modifier et DELETE de supprimer.

Afin de pouvoir comprendre comment l'api fonctionne, le descriptif des routes est dispnibles dans Routes HTTP et egalement sur :

- La documentation de l'API REST est disponible via : https://sondage.hirokx.dev/v3/api-docs
- La documentation de l'API REST lisible est disponible via Swagger : https://sondage.hirokx.dev/swagger-ui/index.html

Routes HTTP

Sondage

- **GET** /api/sondage/{id} - Obtenez un sondage
- **PUT** /api/sondage/{id} - Mettez à jour un sondage
- **DELETE** /api/sondage/{id} - Supprimez un sondage
- **GET** /api/sondage/ - Obtenez tous les sondages
- **POST** /api/sondage/ - Créez un nouveau sondage

Participant

- **GET** /api/participant/{id} - Obtenez un participant
- **GET** /api/participant/ - Obtenez tous les participants
- **POST** /api/participant/ - Créez un nouveau participant
- **PUT** /api/participant/{id} - Mettez à jour un participant
- **DELETE** /api/participant/{id} - Supprimez un participant

Commentaire

- **PUT** /api/commentaire/{id} - Mettez à jour un commentaire
- **DELETE** /api/commentaire/{id} - Supprimez un commentaire
- **GET** /api/sondage/{id}/commentaires - Obtenez les commentaires d'un sondage
- **POST** /api/sondage/{id}/commentaires - Créez un commentaire pour un sondage

Dates

- **GET** /api/sondage/{id}/dates - Obtenez les dates disponibles d'un sondage
- **POST** /api/sondage/{id}/dates - Créez une date pour un sondage

- **GET** /api/sondage/{id}/maybe - Obtenez les possibles meilleures dates pour un sondage en fonction des disponibilités
- **GET** /api/sondage/{id}/best - Obtenez les meilleures dates pour un sondage en fonction des disponibilités
- **DELETE** /api/date/{id} - Supprimez une date

Participation

- **POST** /api/date/{id}/participer - Votez pour une date

GitHub Actions

UnitAndIntegration.yml

Cette pipeline GitHub est utilisée pour tester les rapports du projet à l'aide de Maven.

Elle inclut des étapes pour la construction du projet, la gestion de cache pour améliorer les performances, la configuration du JDK (Java Development Kit), l'exécution de tests unitaires, la génération d'un badge JaCoCo (pour la couverture de code), et l'exécution des tests de mutation.

Elle nous sert à chaque fois qu'il y a une pull request sur develop ou sur notre branche principale. Evitant ainsi de merge du code qui ne compile pas ou ne remplissant pas les conditions de test.

Testing.yml

Cette pipeline GitHub est utilisée pour tester et déployer les rapports du projet à l'aide de Maven.

Elle inclut des étapes pour la construction du projet, la gestion de cache pour améliorer les performances, la configuration du JDK (Java Development Kit), l'exécution de tests unitaires, la génération d'un badge JaCoCo (pour la couverture de code), et l'exécution des tests de mutation. A la fin, elle execute un script permettant de déployer une GitHub pages qui déploie les rapports JaCoCo et PiTest.

Elle nous sert à chaque fois qu'il y a un push sur develop ou sur notre branche principale. Evitant ainsi de push du code qui ne compile pas ou ne remplissant pas les conditions de test. Déployant donc les rapports JaCoCo et PiTest.

E2E.yml:

Cette pipeline GitHub est utilisée pour lancer les tests End-To-End du projet à l'aide de Maven et de Docker.

Elle inclut des étapes pour la construction de la base de donnée, la construction du projet avec Docker, la gestion de cache pour améliorer les performances, la configuration du JDK (Java Development Kit), l'exécution de tests End-To-End, la génération d'un badge JaCoCo (pour la couverture de code), et l'exécution des tests de mutation. A la fin, elle execute un script permettant de déployer une GitHub pages qui déploie les rapports JaCoCo et PiTest.

Elle nous sert à chaque fois qu'il y a un push/pull request sur develop ou sur notre branche principale. Evitant ainsi de push/merge du code qui ne compile pas ou ne remplissant pas les conditions des test End-To-End.

CD.yml:

Cette pipeline GitHub est utilisée pour déployer notre projet sur un serveur à distance, le rendant ainsi accessible à tout le monde.

Elle inclut des étapes pour la connexion au serveur à distance en ssh, la récupération du dépot distant et le déploiement grâce à Docker.

Elle nous sert à chaque fois qu'une pull request sur develop ou sur notre branche principale est acceptée.

Checkstyle.yml:

Cette pipeline Github utilisée pour vérifier les erreurs sur notre projet. Ici, nous suivons les normes qui sont présentes ici.

Elle inclut des étapes pour la construction du projet et la vérification statiques des styles.