Buchmuller Nassim
El Kihal Ewan
Loeb Dorian
Rizzante–Madonna Alexandre
Turmo Baptiste

3ème année Groupe A1

R6.A.06 Maintenance: Compte rendu

Etat des lieux

Observations:
Pas de Programmation Orienté Objet => Programmation procédurale
Pas d'architecture logicielle claire: les tâches ne sont pas séparées (interfaces, requêtes BDD, …)
Structure du projet confuse: beaucoup de fonctions sont dans un seul fichier (mauvaise séparation des fonctions)
Normes de langages non respectées (mot-clefs PHP et noms des balises HTML mal écrites)
Code non documenté, peu de commentaires
Pas de jeux de tests (tests unitaires, …)
Utilisation de caractères spéciaux
Utilisation de variables globales
Problèmes de sécurité conséquent.
BDD sans clefs étrangères
Code non factorisé: certaines fonctions font plusieurs actions => fonction avec beaucoup de code, complexe à comprendre
API Mysql obsolète. exemple (`mysql_connect` dépréciative, supprimé avec PHP 7)

Conclusion: Le code en l'état n'est pas maintenable car il est trop compliqué à comprendre à cause du manque de structure du projet et l'absence de documentation technique.

Choix techniques

Type de technologie	Technologie initiale	Technologie choisie	Pourquoi ?
Langage de programmation back-end	PHP 5.x	PHP 8.5	Sécurité, Syntaxe, POO avancé (interfaces, …)
Framework back-end	Aucun (PHP natif)	Symfony 7.2	Maintenable, modèle MVC, simple à apprendre, bon ORM, performances optimales, compatibilité avec **Behat => pas de Symfony 8** tant que Behat ne soit pas à jour
Gestionnaire packages back-end	Aucun	Composer 2.2.6	Permet de gérer les paquets PHP facilement, est pris en compte avec Symfony
Base de données	MySQL sans InnoDB	PostgreSQL 16.11 hébergé sur alwaysdata	Maintenable, largement soutenu par la communauté, solution moderne avec grande mise à l'échelle alwaysdata est gratuit
Framework front-end	Aucun	React 18.3 et Vite 7.3	Maintenable et extensible, support étendu de la communauté, solutions de mise en page modernes
Gestionnaire packages front-end	Aucun	npm 11.5.1 avec environnement Node JS 22.14.0	Permet de gérer les paquets JS facilement, s'intègre bien avec React
Intégration continue	Aucune	Jenkins 2.541.1	Permet d'automatiser le déploiement du serveur, facile, plugin GitHub pour DevOps
Amélioration continue	Aucune	Sonarqube 26.1.0	S'assure de la qualité de code + sécurité, DevOps (intégration continue), différents indicateurs

Déploiement de l'intégration continue	Aucun	VPS Oracle avec Docker	Déploiement personnalisé pour pouvoir avoir l'intégration continue gratuitement
Système de gestion de version	Aucun	GitHub	Facile la collaboration, permet de centraliser sur un dépôt distant commun, gestion des branches et pull request (ainsi que des conflits)
Qualité de code	Aucun	PHP Code Standards Fixer 3.93	Linters qui permet de respecter les standards du langage PHP, facile à installer et à utiliser depuis le CLI

Chaînes de production et de tests

Type de technologie	**Technologie initiale**	**Technologie choisie**	**Pourquoi ?**
Tests unitaires	Aucun	PHPUnit 13.0	Tests unitaires, Utilisation facile avec terminal
Tests de comportement	Aucun	Behat 3.29, implémentation PHP de Cucumber	Tests fonctionnelles, facile à utiliser (description facile des comportements)
Tests finaux	Aucun	Cypress 15.0.0	Partie gratuite et open-source, permet de tester en temps réel l'application du point de vue de l'interface

Workflow GitHub

Nous avons ajouté un workflow sur le dépôt GitHub selon un modèle pour un projet Symfony. Cela permet de vérifier que l'application fonctionne et d'éviter des pull request faux. C'est un script qui se lance pour vérifier la pull request.
Nous avons organisé les branches du dépôt de cette façon:

- `main`: branche principale de l'application une fois que les fonctionnalités à ajouter sont validés
- `dev`: branche de développement auquel nous fusionnons les nouvelles fonctionnalités
- `configuration`: branche pour la configuration initiale du projet
- `feature/%fonctionnalité%`: branche pour ajouter une fonctionnalité.

Nous avons aussi ajouté une règle sur la branche main qui impose la revue par un tiers du contenu de la pull request et force la pull request.

Métriques utilisées

Nous avons utilisé une métrique pour vérifier la qualité de code:

- Index de Maintenabilité (MI): Elle indique la maintenabilité du code. Plus code est facile à maintenir, c'est-à-dire corriger le code, ajouter des fonctionnalités précisément, elle précise que si les, vérifier la qualité de code et observer les points où le code est trop complexe et doit être simplifié.

Pour réaliser les calculs de ces métriques, nous avons utilisé un script utilisant Php Metric un logiciel de tableur comme Microsoft Excel ou LibreOffice exporté sous le format CSV.

il suffit de se positionner dans le fichier backend et de faire la commande:

php metrics_report.php

Maintenant vous avez un rapport en csv de l'Index de Maintenabilité.

Test PHPUnit :

Objectifs de la stratégie de test

L'objectif était de valider l'intégrité des données chargées depuis les fixtures dans la base de données PostgreSQL et de garantir le bon fonctionnement de l'API REST.

Nous avons voulu nous assurer que les relations entre entités (Sports, Championnats, Compétitions, Épreuves) sont correctement établies et que l'API Platform retourne les données attendues au format JSON-LD.

PHPUnit avec API Platform Test a été choisi pour sa capacité à tester les endpoints de manière unitaire et à valider les réponses JSON sans nécessiter de serveur externe.

Test finaux : assurance de qualité + tests auto (Cypress)

1. Objectifs de la stratégie de test

L'objectif était d'assurer la stabilité de l'application dans le temps et d'éviter les régressions lors des évolutions. Nous avons également voulu valider le bon fonctionnement de l'application du point de vue utilisateur à travers des tests end-to-end.

Cypress a été choisi pour sa rapidité d'exécution, sa simplicité d'utilisation et sa capacité à simuler des interactions réelles avec l'interface.

2. Périmètre

Nous avons mis en place une suite de 17 scénarios de test organisée autour de quatre axes principaux.

Tests de parcours utilisateur (E2E)
On vérifie ici le fonctionnement complet de l'application, de la page d'accueil jusqu'au détail d'un championnat.

Nous avons vérifié que les données issues de l'API Symfony étaient correctement intégrées et affichées dans l'interface React. Les tests couvrent les parcours classiques (Happy Paths) ainsi que certains cas limites (Edge Cases).

Robustesse de l'API
Grâce à `cy.intercept()`, nous avons mocké les réponses de l'API afin de tester différents scénarios sans dépendre du backend réel.

Nous avons simulé :

- des temps de latence (vérification des loaders),
- des erreurs serveur 500
- des erreurs 404
- des cas sans données (on affiche des empty states).

Validation responsive
Des tests ont été exécutés sur différents viewports (mobile, tablette, desktop) afin de garantir la cohérence et l'utilisabilité de l'interface sur tous les supports.

Tests d'intégration des données
Un script de fixtures PHP permet de préparer la base de données avant les tests, assurant un environnement stable et maîtrisé.

3. Résultats

Les 17 scénarios sont validés. Un script `cypress:run` a été ajouté afin de faciliter l'intégration des tests dans un pipeline CI/CD.

Cette approche permet de sécuriser les évolutions futures, d'assurer la non-régression et de détecter rapidement les anomalies d'affichage, notamment sur mobile, avant mise en production.