Daphné Avias
Léo Garcia
Jalil Kanboui
Cyril Tamine
Thomas Tran

R6.A06 - Maintenance
Feuille de route

Initialisation du projet

Le projet a été structuré dès le départ avec une organisation claire des branches Git :

- `main` : version stable
- `dev` : branche d'intégration
- branches `feature/*` : développement des fonctionnalités

Cette organisation permet une meilleure traçabilité des évolutions et facilite la maintenance.

Le projet a été initialisé à partir d'un squelette Symfony 7, garantissant une architecture claire basée sur le modèle MVC.

La logique métier est centralisée dans des services dédiés afin d'éviter toute surcharge des contrôleurs.

Cette structuration prépare l'application des principes :

- SOLID
- Loi de Demeter
- KISS

Application des principes de conception

La logique métier est répartie dans des services spécifiques :

- `CompetitionService`
- `ChampionnatService`
- `InscriptionService`

Les contrôleurs sont limités à la gestion des requêtes HTTP et à la délégation vers les services.

Les entités ne contiennent pas de logique métier complexe.

Cette séparation des responsabilités favorise :

- le respect du principe de responsabilité unique (SRP),
- une meilleure testabilité,
- une architecture maintenable et évolutive.

Qualité logicielle et tests automatisés

Le projet intègre une stratégie complète de qualité logicielle basée sur plusieurs niveaux de tests :

- **Tests unitaires (PHPUnit)** : validation des services métier indépendamment de l'infrastructure.
- **Tests d'intégration** : vérification du fonctionnement des repositories et de l'interaction avec la base de données.
- **Tests comportementaux (Behat)** : validation des scénarios utilisateurs via des fichiers `.feature`.
- **Tests fonctionnels finaux (Symfony Panther)** : simulation d'un navigateur réel permettant de tester les interactions complètes avec l'interface web.

L'intégration de Panther permet d'approcher un niveau professionnel de tests end-to-end, garantissant que l'application fonctionne correctement du point de vue utilisateur.
Pour que les tests Behat et Panther fonctionnent :

- La **base de données de test** doit être initialisée (`doctrine:database:create --env=test`) et migrée (`doctrine:migrations:migrate --env=test`).
- Les variables d'environnement (`DEFAULT_URI`, `DATABASE_URL`) doivent être définies dans `.env.test`.
- Les scénarios Behat utilisent maintenant un **FeatureContext adapté à Panther**, permettant de démarrer un navigateur ou un client HTTP simulé pour tester les routes.

Analyse statique et contrôle qualité

L'outil **PHPStan** est utilisé pour détecter les incohérences de typage et les problèmes potentiels avant exécution.
Certaines alertes ont permis d'améliorer le typage des entités (notamment l'initialisation correcte des identifiants Doctrine).

PHP CS Fixer assure la conformité aux standards PSR-12 et aux conventions Symfony, garantissant un code homogène et maintenable.

L'ensemble de ces outils permet de détecter précocement les erreurs, d'améliorer la qualité du code et de faciliter sa maintenance.

Environnement technique

Le projet repose sur des technologies modernes et pérennes :

- Symfony 7.4 (LTS)
- PHP 8.3
- Doctrine ORM & Migrations
- MySQL
- PHPUnit
- PHP CS Fixer
- PHPStan

- GitHub + Pull Requests
- Behat / Symfony Panther
- GitHub + Pull Requests

Ce choix garantit :

- compatibilité long terme
- maintenabilité
- standard professionnel

Intégration Continue (CI)

Un **workflow GitHub Actions** automatise :

- l'installation des dépendances
- la vérification du style de code
- l'analyse statique (PHPStan)
- l'exécution des tests unitaires et d'intégration
- l'exécution des tests comportementaux et fonctionnels

Cette automatisation permet :

- détection précoce des régressions
- sécurisation des merges
- reproductibilité de l'environnement
- professionnalisation de la démarche

Apports en maintenance

La mise en place de ce workflow permet :

- une meilleure homogénéité du code
- une détection précoce des anomalies
- une traçabilité des modifications
- une professionnalisation de la démarche

La maintenance est abordée non seulement comme une correction d'anomalies, mais comme une amélioration continue du projet.

Version des systèmes utilisés

- **Symfony 7.4.5 LTS** : architecture MVC/3-tiers, support jusqu'en 2028
- **PHP 8.3.6** : performances optimisées et compatibilité Symfony
- **PHPUnit 11.5.51** : tests unitaires et d'intégration
- **PHP CS Fixer 3.93.1** : maintien de la qualité et standard PSR-12

- **PHPStan 2.1.38** : analyse statique
- **Doctrine ORM & Migrations** : gestion propre des entités et schéma
- **MySQL** : base stable pour développement et production
- **Panther 2.4.0** : tests fonctionnels end-to-end

Maintenabilité et qualité logicielle

L'ensemble de la configuration (linter, tests automatisés, PHPStan, workflow GitHub) assure :

- code lisible, cohérent et évolutif
- application des bonnes pratiques : SOLID, Loi de Demeter, DRY, KISS
- pérennité et facilité de maintenance

Tests associés :

- PHPUnit : services et repositories
- Behat : scénario « Page championnat » et autres parcours utilisateur
- Panther : tests end-to-end des interactions avec l'interface web