Daphné Avias
Léo Garcia
Jalil Kanboui
Cyril Tamine
Thomas Tran

# R6.A06 - Maintenance

# Analyse de la documentation – Projet UGSEL Web

# Contexte général

Le projet UGSEL Web est une application web destinée à gérer l'inscription en ligne des licenciés UGSEL aux compétitions sportives. La documentation fournie date de 2012, ce qui implique un contexte technologique ancien et justifie une réflexion approfondie sur la maintenabilité et la pertinence d'une refonte complète.

Dans le cadre du module de maintenance logicielle, le choix a été fait de **repartir de zéro**, en conservant les besoins fonctionnels existants tout en proposant une architecture moderne, maintenable et évolutive.

---

# Analyse fonctionnelle

## Objectif du système

UGSEL Web est une application web permettant :

- l'inscription en ligne des licenciés UGSEL aux compétitions sportives,
- par les établissements scolaires, via leurs UGSEL départementales et régionales.

L'application vise à centraliser la gestion des inscriptions et à simplifier les démarches administratives liées aux compétitions.

## Acteurs

- **Administrateur UGSEL** (implicite) : responsable de la gestion globale des compétitions et de l'ouverture ou fermeture des inscriptions.

## Fonctionnalités principales

- Consultation des informations suivantes :
    - établissements
    - licenciés
    - compétitions
- Gestion des inscriptions aux compétitions :
    - inscriptions individuelles
    - inscriptions en relais ou par équipes
- Opérations de type CRUD :
    - ajout
    - modification
    - suppression
- Fonctionnalités transverses :

    - tri des données
    - filtrage
    - export des données au format CSV
    - impression des listes
- Gestion des états des compétitions :
    - inscriptions ouvertes
    - inscriptions fermées

---

# Analyse de l’existant

## Ancienneté

La documentation analysée est datée du **30 octobre 2012**, soit plus de douze ans. Cette ancienneté laisse supposer l’utilisation de technologies aujourd’hui dépassées et soulève des enjeux importants en matière de maintenance et de sécurité.

## Indices techniques implicites

Plusieurs éléments de la documentation permettent d’émettre des hypothèses sur les choix techniques initiaux :

- forte dépendance au navigateur web,
- recommandation explicite de l’utilisation de Firefox,
- interface utilisateur très orientée tableaux HTML,
- actions majoritairement synchrones,
- export CSV géré côté client,
- impression reposant sur les fonctionnalités du navigateur.

## Hypothèses techniques raisonnables

En l’absence du code source, les hypothèses suivantes peuvent être formulées :

- **Backend** :
    - PHP ancien ou Java (JSP / Servlets)
- **Frontend** :
    - HTML et CSS basiques
    - JavaScript peu structuré
- **Architecture** :
    - application monolithique
    - fort couplage entre le frontend et le backend

## Problèmes de maintenance identifiés

L’analyse met en évidence plusieurs limites majeures :

- technologies non mises à jour,
- sécurité incertaine (gestion des sessions, authentification),
- expérience utilisateur dépassée,
- absence de responsive design,
- difficulté d’évolution pour intégrer de nouveaux besoins,
- dépendance forte à un navigateur spécifique.

Ces éléments rendent la maintenance corrective et évolutive complexe et coûteuse.

---

# Contraintes et objectifs de la refonte

## Contraintes techniques

Les technologies retenues pour la refonte doivent répondre aux critères suivants :

- éviter les technologies trop anciennes (obsolescence),
- éviter les technologies trop récentes (manque de recul),
- être éprouvées,
- être bien documentées,
- favoriser la maintenabilité à long terme.

## Architecture proposée

Une architecture web classique **3-tiers** est retenue :

- **Frontend** : interface utilisateur
- **Backend (API)** : logique métier et sécurité
- **Base de données** : persistance des données

Cette architecture permet de limiter le couplage, de faciliter les tests et d’améliorer l’évolutivité du système.

---

# Choix techniques proposés

## Backend

Le backend pourra être implémenté avec :

- **Symfony (PHP)** ou
- **Spring Boot (Java)**

Ces frameworks ont été choisis pour les raisons suivantes :

- maturité et stabilité,
- large communauté et documentation abondante,
- sécurité intégrée,
- compatibilité avec des projets à long cycle de vie,
- adaptation aux applications institutionnelles.

## Frontend

Le frontend reposera sur :

- HTML, CSS et JavaScript standards,
- éventuellement un framework JavaScript léger (Vue.js ou React), utilisé avec modération.

Les objectifs sont :

- une séparation claire des responsabilités,
- une meilleure expérience utilisateur,
- un design responsive,
- une facilité d'évolution et de maintenance.

## Base de données

Les systèmes de gestion de base de données envisagés sont :

- **MySQL** ou
- **PostgreSQL**

Ces solutions sont reconnues pour leur stabilité, leurs performances et leur compatibilité avec les outils d'export et d'analyse de données.

## Sécurité

La refonte intégrera dès la conception les aspects de sécurité suivants :

- authentification sécurisée,
- gestion des rôles et des droits,
- protection contre les attaques courantes :
    - injections SQL,
    - failles XSS,
    - attaques CSRF.

# Qualité logicielle et maintenabilité

Dans le cadre de la refonte complète du projet UGSEL Web, une attention particulière est portée à la qualité logicielle et à la maintenabilité du code. L'objectif n'est pas uniquement de reproduire les fonctionnalités existantes, mais de concevoir une application durable, évolutive et facile à maintenir.

## Principes de conception

### Principes SOLID

L'architecture du projet sera conçue en respectant les principes SOLID :

- **Responsabilité unique** : chaque classe ou service aura un rôle précis et clairement défini (par exemple : gestion des inscriptions, gestion des compétitions, gestion des exports), évitant ainsi les classes dites « fourre-tout ».
- **Ouvert / Fermé** : le système sera conçu pour permettre l'ajout de nouvelles fonctionnalités (nouveaux types de compétitions, nouveaux exports) sans modifier le code existant.
- **Substitution de Liskov** : les relations d'héritage seront utilisées avec précaution afin de garantir un comportement cohérent des classes filles par rapport aux classes parentes.
- **Ségrégation des interfaces** : des interfaces spécifiques seront privilégiées afin d'éviter que des classes dépendent de méthodes inutiles.
- **Inversion des dépendances** : le recours à des abstractions et à l'injection de dépendances (notamment via le conteneur de services de Symfony) permettra de réduire le couplage et d'améliorer la lisibilité et la testabilité du code.

### Loi de Demeter

La loi de Demeter sera respectée afin de limiter les dépendances excessives entre les composants. Les objets interagiront uniquement avec leurs dépendances directes, ce qui permettra de réduire le couplage et de faciliter l'évolution du code.

### Principes KISS et DRY

- **KISS (Keep It Simple, Stupid)** : l'architecture et les choix techniques resteront volontairement simples, sans sur-ingénierie, afin de faciliter la compréhension et la maintenance du projet.
- **DRY (Don't Repeat Yourself)** : les duplications de code seront évitées grâce à l'utilisation de services, de composants réutilisables et de mécanismes de factorisation.

## Documentation

Une documentation claire et structurée accompagnera le projet afin de faciliter sa prise en main et sa maintenance :

- documentation technique du code (PHPDoc ou équivalent),
- génération automatique de la documentation,
- fichier README détaillant :
    - les prérequis,
    - les versions des outils utilisés,
    - les dépendances,
    - les étapes d'installation et de lancement du projet,
    - les commandes liées aux tests et à la qualité du code.

---

## Qualité du code, outils et métriques

### Linter et règles de codage

Un linter sera mis en place afin de garantir une qualité de code homogène (par exemple PHP-CS-Fixer ou PHP_CodeSniffer). Les règles de codage seront appliquées automatiquement et intégrées au workflow de développement.

### Métriques

Des métriques de qualité seront utilisées pour évaluer et améliorer la maintenabilité du projet :

- indice de maintenabilité,
- complexité cyclomatique,
- couverture de tests.

Ces métriques permettront d'identifier les zones à risque et d'orienter les améliorations du code.

---

## Tests

La refonte du projet s'appuiera sur une stratégie de tests complète, intégrée dès le début du développement :

- **tests unitaires** : validation du bon fonctionnement des composants isolés,
- **tests d'intégration** : vérification des interactions entre les différents modules,
- **tests de comportement** (par exemple avec Behat) : validation des scénarios fonctionnels,
- **tests finaux automatisés** : simulation du comportement global de l'application.

Cette approche vise à limiter les régressions et à sécuriser les évolutions futures.

---

### Workflow de développement et gestion du projet

Le projet adoptera un workflow Git structuré :

- branches principales (`main`, `develop`),
- branches de fonctionnalités (`feature/*`),
- utilisation de pull requests,
- exécution automatique des tests et des outils de qualité lors des intégrations.

Ce fonctionnement permet d'assurer un suivi rigoureux du projet et de renforcer la qualité globale du développement.

---

# Conclusion

Compte tenu de l'ancienneté de l'application UGSEL Web et des technologies probablement utilisées lors de sa conception, une **refonte complète** apparaît plus pertinente qu'une simple maintenance corrective. Le projet ne se limite pas à une réécriture fonctionnelle, mais vise à appliquer les bonnes pratiques de conception logicielle, de tests et de documentation. Les choix techniques retenus s'appuient sur des technologies éprouvées et une méthodologie orientée qualité, garantissant un bon compromis entre modernité, stabilité et maintenabilité à long terme.