Cahier des charges technique – Solution logicielle de gestion pour les Jardins de Cocagne

Présenté par : GIL Jérémie et PETRAZOLLER Johan
Formation : BUT 3 Informatique
Année : 2025-2026

UNIVERSITÉ DE LORRAINE

IUT Saint-Dié

Équipe projet

Jérémie GIL – Étudiant en **BUT 3 Informatique**

Rôles et contributions :

- Conception de l’architecture logicielle
- Développement du **Back Office** sous Symfony
- Mise en place de la base de données
- Participation aux tests et à la validation technique

Johan PETRAZOLLER – Étudiant en **BUT 3 Informatique**

Rôles et contributions :

- Conception des interfaces utilisateur sous Symfony
- Gestion de la sécurité
- Rédaction du cahier des charges technique et documentation finale
- Participation aux tests et à la validation technique

Sommaire

I. Contexte et objectifs du projet

II. Architecture générale du système

III. Structure fonctionnelle

IV. Choix techniques

V. Base de données et modélisation

VI. Conclusion et perspectives

I. Contexte et objectifs du projet

Contexte

- Les Jardins de Cocagne est un réseau de plus de 100 jardins, permettant l'insertion social via le maraîchage biologique.
- Vente principale de paniers de légumes bio aux adhérents.
- L'ancien logiciel de gestion est une application Microsoft Access, locale et obsolète.

Enjeu Central

- Concevoir un logiciel de gestion moderne, accessible en ligne.
- Offrir une interface simple adaptée à tous les profils (encadrants, salariés en insertion, bénévoles).
- Permettre une autonomie des adhérents via un espace personnel.
- Mutualiser l'outil au niveau national tout en laissant chaque Jardin configurer son fonctionnement.

L’outil actuel vieillissant et obsolète

L’ancien outil de gestion est devenu obsolète pour plusieurs raisons :

- On ne peut qu’y accéder localement, il est impossible d’y accéder par le web.
- L’interface utilisateur est peu intuitive et peu poser des problèmes aux utilisateurs peu expérimentés.
- L’installation en locale est obligatoire pour y avoir accès.
- Il n’y a aucune automatisation des tâches récurrentes
- Chaque jardin gère sa propre base, il n’y a pas de mutualisation.
- Les clients ne peuvent pas gérer leurs abonnements via un espace client

II. Architecture générale du système

Pour remplacer l'ancien outil, les Jardins de Cocagnes souhaitent :

- **Modèle SaaS mutualisé (multi-structure)**

→ Une seule application partagée par tous les Jardins, avec des données cloisonnées par structure.

- **Accès 100 % web**

→ Application accessible depuis un navigateur, sans installation locale.

- **Deux espaces principaux :**

Back Office : gestion des clients, abonnements, dépôts, tournées, produits, etc.

Front Office : espace personnel pour les adhérents.

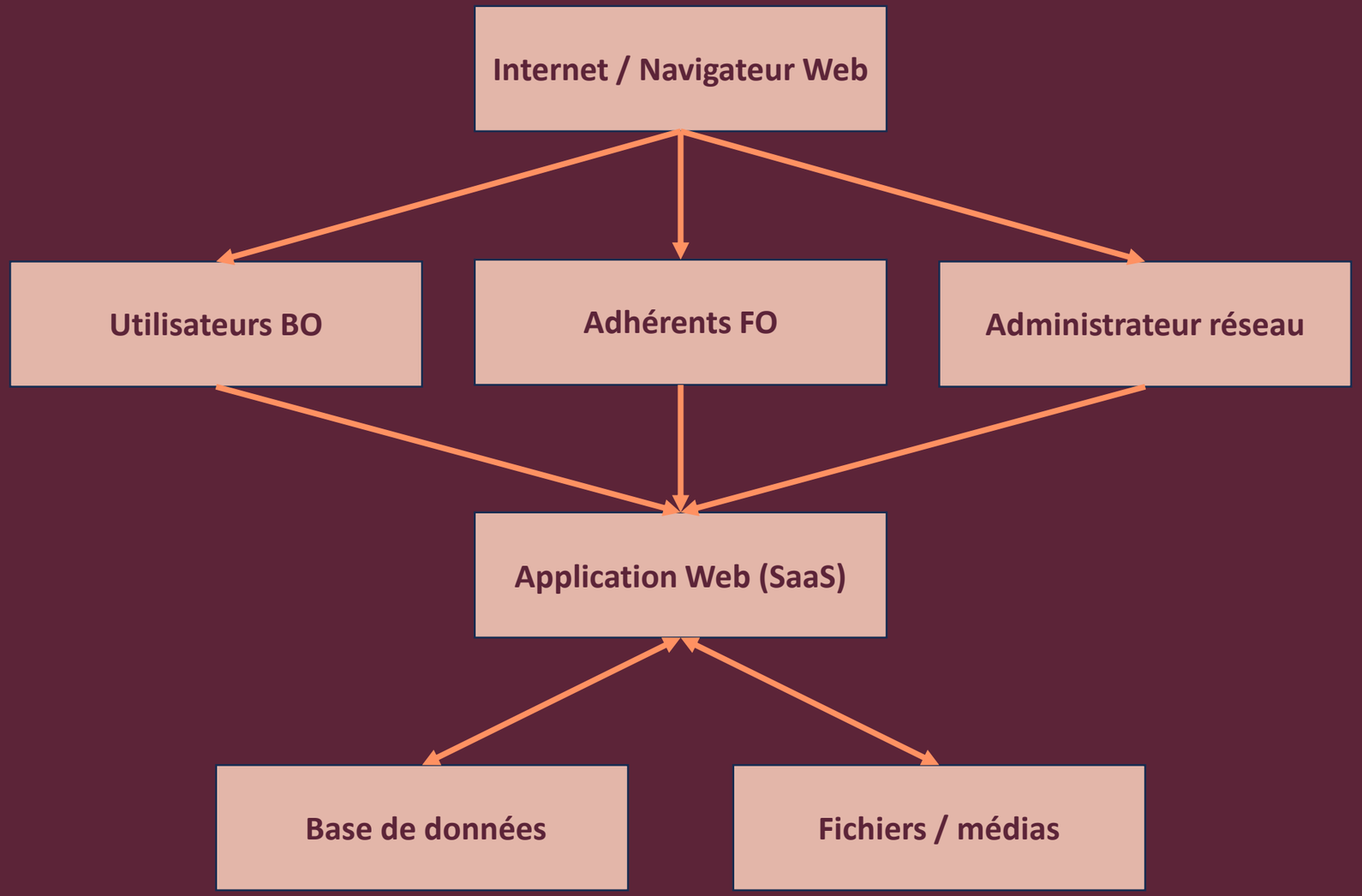
- **Hébergement centralisé et sécurisé**

→ Serveurs administrés par le prestataire, simplifiant les mises à jour.

- **Open source et évolutif**

→ Code libre, permettant aux Jardins de développer ou modifier des modules.

Schéma d’architecture du système



- Accès unique via navigateur (aucune installation locale).
- Données cloisonnées par *Jardin*.
- Hébergement mutualisé sur serveur sécurisé.
- Front Office et Back Office reliés à la même base.

Fonctionnement multi-structure

- **Une seule application partagée** entre tous les Jardins de Cocagne.

- Chaque Jardin possède **ses propres données**, paramétrages et utilisateurs.

- Le système identifie chaque élément par un identifiant de structure (JARDIN_ID).

- Les données sont **totalement cloisonnées** :
→ aucun accès possible aux informations d'un autre Jardin.

- Les mises à jour et correctifs sont déployées **une seule fois** pour tous.

- Permet la **mutualisation des coûts** (hébergement, maintenance, évolutions).

Avantages du modèle SaaS mutualisé

- **Maintenance simplifiée** : une seule version à mettre à jour pour tout le réseau.
- **Accès universel** : disponible partout, sur tout appareil connecté à Internet.
- **Sécurité renforcée** : données hébergées sur un serveur central sécurisé.
- **Mutualisation des coûts** : hébergement et support partagés entre les Jardins.
- **Évolutivité** : ajout de nouvelles fonctionnalités profitant à l'ensemble du réseau.
- **Pérennité** : code libre et documenté, accessible à la communauté des Jardins de Cocagne.

III. Structure fonctionnelle du système

Le futur logiciel doit être composé de **deux interfaces principales** :

1. Back Office (B.O.) : Gestion Interne

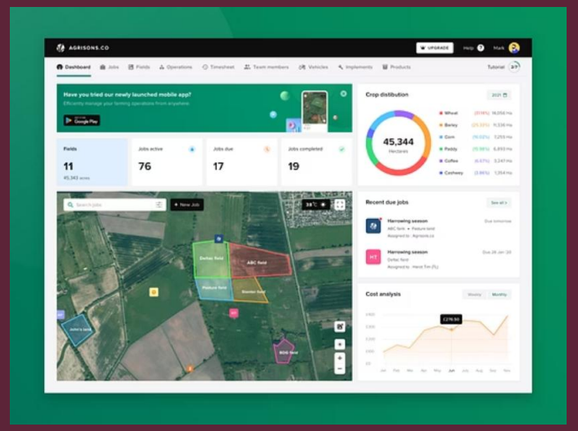
Pour qui : Les équipes du Jardin.

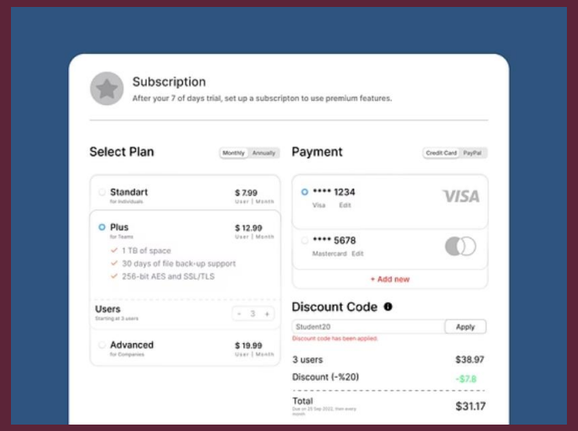
- Gérer les clients et les abonnements.
- Gérer le stockage et les commandes.
- Imprimer les feuilles de route.

2. Front Office (F.O.) : Espace Client

Pour qui : Les Adhérents-consommateurs.

- Permettre l'autonomie du client.
- Gérer et consulter ses abonnements.
- Mettre à jour ses informations.





IV. Choix techniques

Langage de développement

Langage choisi : PHP 8+

→ Langage **libre, robuste et largement documenté**, idéal pour un projet open source.

→ Facile à prendre en main par différents profils (bénévoles, développeurs internes).

→ Excellente compatibilité avec les hébergeurs mutualisés et les frameworks modernes.

💡 *Pourquoi PHP ?*
→ Il permet de **développer rapidement des applications web stables**, tout en assurant une **grande communauté de support et d'extensions**.

IV. Choix techniques

Framework applicatif

Framework choisi : Symfony (architecture MVC)

→ Offre une **structure claire et modulaire** : séparation du code (Modèle / Vue / Contrôleur).

→ **Écosystème complet** : sécurité, formulaires, gestion des données …

→ **Framework mature et maintenu**, utilisé dans de nombreux projets professionnels.

💡 *Pourquoi Symfony ?*
→ Il combine **performance, clarté et maintenabilité**, tout en restant **ouvert aux contributions** du réseau Cocagne.

IV. Choix techniques

Composants Symfony utilisés

- **Doctrine ORM** → Gestion automatique des accès à la base de données.
- **Form & Validator** → Création et validation simplifiée des formulaires.
- **Twig** → Génération de pages HTML claires et structurées.
- **Security & Bcrypt** → Authentification et hashage sécurisé des mots de passe.
- **Symfony Console & Migrations** → Outils intégrés pour les tâches d'administration.

💡 *Avantage :*
→ Ces composants garantissent un développement **homogène, sécurisé et évolutif**.

IV. Choix techniques

Front-end et dynamisme

Moteur de rendu : Twig (HTML côté serveur)
→ Structure claire, facile à modifier par des équipes non techniques.

JavaScript natif (Vanilla JS)
→ Ajout de dynamisme.

Symfony UX
→ Pour des interactions réactives tout en gardant la simplicité du modèle serveur.

IV. Choix techniques

Base de données : MariaDB

→ Base de données **relationnelle**, libre et open source.

→ Compatible avec **MySQL** et parfaitement intégrée à **Doctrine ORM**.

→ Offre de bonnes performances et une grande stabilité pour les environnements web.

→ Gère efficacement le **cloisonnement des données** entre les différentes structures (multi-Jardin).

💡 *Pourquoi MariaDB ?*
→ Fiable, légère, bien supportée, et simple à administrer sur des serveurs mutualisés.

IV. Choix techniques

Sécurité de l’application

- **Authentification sécurisée** avec le composant **Security** de Symfony.
- **Hashage des mots de passe** via l’algorithme **bcrypt**.

→ Empêche toute récupération en clair des identifiants.

- **Gestion des rôles utilisateurs** :

  ROLE_ADMIN → gestion complète de la structure,
  ROLE_GESTION → encadrants / salariés,
  ROLE_ADHERENT → accès client (front office).

- **Protection des échanges** par le protocole **HTTPS**.
- **Vérifications automatiques** : CSRF, validation des formulaires, contrôle d’accès sur chaque route.

IV. Choix techniques

Hébergement : Serveur mutualisé

→ Application hébergée sur un **serveur mutualisé**, administré par le prestataire.
Une seule instance du logiciel pour l'ensemble des Jardins de Cocagne.

→ Environnement : **Linux (Ubuntu)**, **Nginx + PHP-FPM**, **MariaDB**, **Git** pour la gestion du code.

→ Sécurité assurée par le protocole **HTTPS** et des **sauvegardes automatiques quotidiennes**.

💡 **Pourquoi un hébergement mutualisé ?**
Maintenance simplifiée, **mises à jour déployées une seule fois**, et **réduction des coûts** pour l'ensemble du réseau.

IV. Choix techniques

Tests et validation

→ Une **instance de test** est mise en place sur le serveur mutualisé, distincte de la version de production.
Elle permet de **valider les évolutions** avant leur déploiement officiel.

→ Utilisation de **PHPUnit** pour les **tests unitaires et fonctionnels**.
Mise en place d'une **intégration continue (CI)** via **GitHub Actions** pour automatiser les vérifications à chaque mise à jour.

💡 Pourquoi ces tests ?
Garantir une **application stable et fiable**, réduire les régressions, et **assurer la qualité** avant chaque déploiement.

IV. Choix techniques

Synthèse des choix techniques

- **Langage :** PHP 8+
  → fiable, documenté et accessible à tous les profils de développeurs.
- **Framework :** Symfony (architecture MVC)
  → structure claire, sécurité intégrée, développement modulaire.
- **Front-end :** Twig + JavaScript
  → interface fluide, légère et facile à maintenir.
- **Base de données :** MariaDB + Doctrine ORM
  → solution open source, performante et adaptée au multi-Jardin.
- **Sécurité :** bcrypt, HTTPS, rôles utilisateurs, validation des formulaires.
  → protection complète des comptes et des échanges.
- **Hébergement :** Serveur mutualisé (Linux, Nginx, PHP-FPM, MariaDB)
  → Mises à jour centralisées, coûts partagés et maintenance simplifiée.
- **Tests et validation :** PHPUnit
  → Intégration continue, contrôle qualité et stabilité avant déploiement.

V. Base de données et modélisation

Vue d’ensemble du modèle (blocs & entités)

Blocs fonctionnels (avec entités clés) :

Structure (Jardin)
– *Jardin* (paramètres, coordonnées, règles de gestion).

Clients & Adhésions
– *Client* (coordonnées, mot de passe),
– *Adhésion associative* (type, période, statut payé/expiré).

Produits & Abonnements
– *Produit* (nom, unité, prix),
– *Abonnement* (nom + fréquence + période).

Dépôts & Tournées
– *Point de dépôt* (catégorie, capacité, horaires, contact),
– *Tournée* (jour, ordre de livraison).

Calendrier & Saisons
– *Saison* (dates début/fin, semaines non livrables, fériés/décalages),
– *Calendrier livrable* (jours de livraison par tournée).

Modèle de données (entités principales)

Entité	Description	Attributs principaux
Jardin	Structure du réseau Cocagne.	id_jardin, nom, adresse, téléphone, email, paramètres, date_creation
Client	Personne physique ou morale effectuant des achats.	id_client, nom, prénom, adresse, email, téléphone, mot_de_passe
Adhésion	Lien entre le client et l'association du Jardin.	id_adhesion, type, date_debut, date_fin, statut
Produit	Élément vendu (panier, œufs, pain…).	id_produit, nom, prix, unité, marge, categorie
Abonnement	Adhésion d'un client sur une période donnée.	id_abonnement, frequence, date_debut, date_fin, nb_livraisons

Modèle de données (entités principales)

Entité	Description	Attributs principaux
Dépôt	Lieu de livraison des paniers.	id_depot, nom, adresse, jour_livraison, horaire, capacité, categorie
Tournée	Regroupe plusieurs dépôts livrés un même jour.	id_tournee, nom, jour, ordre_livraison
Saison	Période d’activité annuelle du Jardin.	id_saison, libelle, date_debut, date_fin, semaines_fermeture
Calendrier	Jours de livraison rattachés à une saison.	id_calendrier, jour, semaine, ferie, report

Planning prévisionnel

Phase d'analyse et de conception (1 semaine)
- Validation du cahier des charges
- Modélisation de la base de données et des interfaces

Phase d'évolution
- Intégration des retours d'amélioration
- Ajout de nouvelles fonctionnalités

Phase de développement (6 semaines)
- Mise en place du framework Symfony et de la BDD
- Développement du Back Office
- Développement du Front Office

Phase de déploiement (1 semaine)
- Mise en ligne sur le serveur mutualisé
- Formation et documentation utilisateur

Phase de tests et d'intégration (1 semaine)
- Tests unitaires et fonctionnels
- Corrections et ajustements

VI. Conclusion

La pérennité du projet est assurée grâce :

Solution basée sur des technologies **open source éprouvées** (PHP, Symfony, MariaDB).

Documentation claire et code structuré, favorisant la maintenance dans le temps.

Possibilité pour les Jardins de **développer ou adapter eux-mêmes** des modules complémentaires.

Architecture mutualisée assurant une **mise à jour centralisée** et un coût réduit pour chacun.

VI. Conclusion

Les perspectives d’évolution pour ce projet sont multiples :

Ajout futur de **modules spécifiques** : facturation, statistiques, gestion de stock, etc.

Interconnexion via une API avec d’autres outils du Réseau Cocagne.

Développement d’une **application mobile** pour les adhérents.