IUT - Crazy Crew

Equipe mixte

Cahier des Charges : Projet ToyBoxing

Crazy Charly Day 2026

Commanditaire :
Association Toys Academy

Lien vers le site :
https://front.cesareuh.fr/

Réalisé par :
Crazy Crew
19 février 2026

Table des matières

1 Présentation du projet

Ce projet est réalisé pour l'association **Toys Academy** dans le cadre des **Crazy Charly Day 2026**. L'objectif est de concevoir une application Web permettant la gestion et l'optimisation de "Toy Boxes" : des boîtes de jouets reconditionnés envoyées aux abonnés selon leurs préférences et l'âge de l'enfant.

2 Objectifs techniques

Le développement s'articule autour de certains axes principaux :

— Une application Web de gestion des articles et des abonnements.

— Une optimisation de la composition des box.

— Un déploiement de la solution globale sur un serveur public.

— Une collaboration inter-départements pour enrichir les fonctionnalités et l'expérience utilisateur.

— Une interface responsive pour les abonnés, accessible sur mobile.

3 Spécifications Fonctionnelles

3.1 Base de données

La base de données doit inclure les tables suivantes :

— **Article** : id : UUID , désignation : string , id_categ : entier , id_age : entier , id_etat : entier , prix : numeric(10,2) , poids : numeric(10,3).

— **Utilisateur** : id : UUID , nom : string , mail : string , mdp : string , admin : booléen.

— **Client** : id : UUID (FK utilisateur) , age : entier , categ_1 à categ_6 : entier (FK catégorie).

— **Catégorie** : id : entier , libellé : string.

— **Age** : id : entier , libellé : string.

— **Etat** : id : entier , libellé : string.

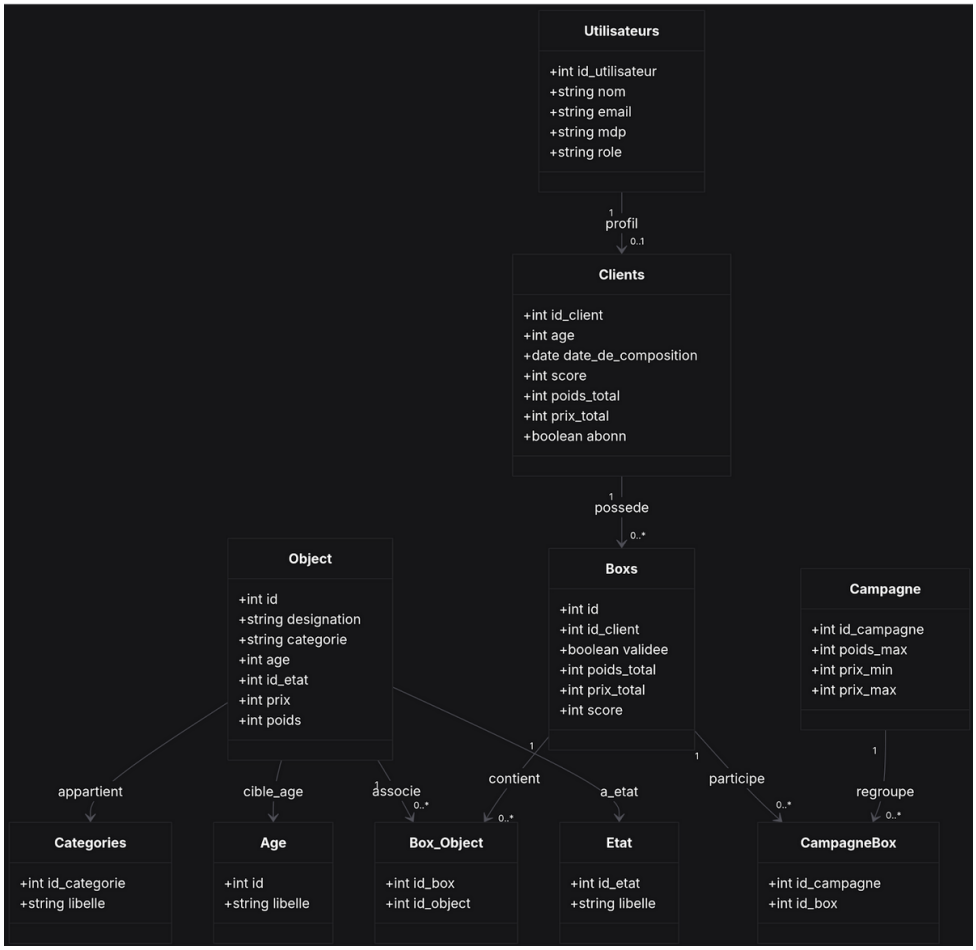
— **Box** : id : UUID , id_client : UUID , prix : numeric(10,2) , poids : numeric(10,3) , score : entier.

— **Campagne** : id : UUID , poids_max : numeric(10,3) , prix_min : numeric(10,2) , prix_max : numeric(10,2).

— **Boxobj** : id_box : UUID , id_article : UUID.

— **Boxcampagne** : id_box : UUID , id_campagne : UUID.

Voici un schéma de la base de données :



3.2 Conteneurisation

L'application doit être conteneurisée à l'aide de Docker, avec les services suivants :

— **Backend** : Un conteneur basé sur PHP 8.2+ hébergeant l'API REST.

— **Frontend** : Un conteneur Node.js pour le développement (Vite) et un serveur Nginx pour la production.

— **Base de données** : Un conteneur PostgreSQL 15+ pour la persistance des données.

— **Orchestration** : Utilisation de `docker-compose` pour lier les services, gérer les volumes de données et les réseaux internes.

4 Architecture Logicielle

L'arborescence du projet témoigne d'une volonté de séparation stricte des responsabilités.

4.1 Backend (PHP)

Le backend suit une **architecture hexagonale** rigoureuse, utilisant le framework Slim 4 et l'injection de dépendances (PHP-DI) :

— **Application Core** : La logique métier est isolée dans le namespace `application_core`. Elle définit les ports (DTOs comme `AbonneDTO`, `ArticleDTO`) et les services métiers (`Service.php`).

— **Infrastructure** : Implémentation de la persistance via le *Pattern Repository* (`UserRepository.ph` `Repository.php`) assurant l'indépendance vis-à-vis du SGBD PostgreSQL.

— **Couche API** :

  — **Actions** : Chaque point de terminaison est géré par une classe unique (ex : `AjouterArticleAction`, `RegisterAction`).

  — **Middleware Stack** : Sécurisation par couches incluant la gestion du `CORS`, l'authentification JWT (`AuthnMiddleware`), et les autorisations (`AuthzMiddleware`).

4.2 Frontend (Vue.js 3)

L'interface est une *Single Page Application* (SPA) développée avec **Vue.js 3** et l'outil de build **Vite**. Elle est structurée en deux espaces distincts :

— **Espace Client (Front Office)** : Gestion du tunnel d'achat et des préférences (`RegisterView`, `LoginView`, `BoxView`).

— **Espace Gestionnaire (Back Office)** : Tableaux de bord pour le suivi des campagnes et des abonnés (`DashboardView`, `CampaignView`, `SubscribersView`).

— **Navigation** : Utilisation de `vue-router` pour assurer un routage fluide sans rechargement de page.

5 Pipeline de Déploiement (CI/CD)

Afin de garantir la qualité et la stabilité du code à chaque modification :

— **Intégration Continue** : Utilisation de GitHub Actions pour automatiser les tests en déployant les docker pour vérifier si ils se lancent correctement.

— **Validation Docker** : À chaque *push* sur la branche principale, une pipeline vérifie que les conteneurs se construisent (`build`) et démarrent (`up`) correctement.

— **Sécurité** : Gestion des variables d'environnement sensibles via les *Secrets* GitHub pour éviter toute fuite de données de connexion.

6 Déploiement

6.1 Choix de l'infrastructure

Initialement prévu sur l'infrastructure `Dockétu`, le déploiement a été migré vers un serveur privé accessible publiquement. Ce choix stratégique a permis de lever plusieurs barrières techniques rencontrées en environnement partagé :

— **Gestion des ports** : Éviter les conflits de ports dynamiques rencontrés sur Dockétu, permettant d'utiliser les ports standards pour l'API et le Frontend.

— **Configuration Réseau** : Résolution des blocages liés aux hôtes autorisés (*Allowed Hosts*) de Vite.js et aux politiques de sécurité CORS.

— **Disponibilité** : Accès permanent via une adresse publique stable pour les démonstrations et tests utilisateurs.