



IUT - CRAZY CREW

EQUIPE MIXTE

Cahier des Charges : Projet ToyBoxing

Crazy Charly Day 2026

Commanditaire :
Association Toys Academy

Réalisé par :
2 Crazy Crew

19 février 2026

Table des matières

1	Présentation du projet	2
2	Objectifs techniques	2
3	Spécifications Fonctionnelles	2
3.1	Base de données	2
3.2	Conteneurisation	3
4	Architecture Logicielle	3
4.1	Backend (PHP)	3
4.2	Frontend (Vue.js)	4
5	Pipeline de Déploiement (CI/CD)	4
6	Plan de Tests	4

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.

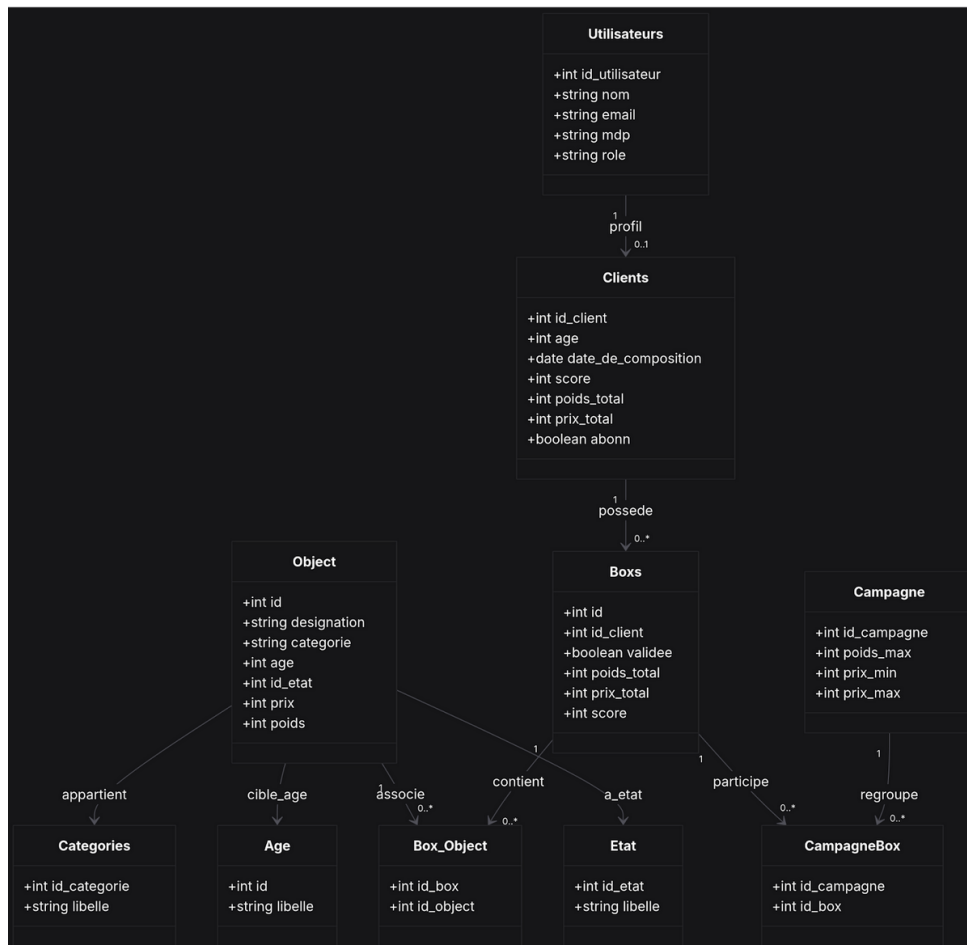
3 Spécifications Fonctionnelles

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 des responsabilités :

4.1 Backend (PHP)

Le backend suit une **architecture hexagonale** (ou Clean Architecture) pour garantir la maintenabilité :

- **Application Core** : Contient la logique métier (`Service.php`) et les cas d'utilisation (usecases).

- **Infrastructure** : Gère les détails techniques comme l'accès à la base de données via des dépôts (**repositories**).
- **API (Interface)** : Point d'entrée de l'application géré par des Actions (**LoginAction**, **RegisterAction**) et sécurisé par des middlewares (JWT, CORS, Authz).

4.2 Frontend (Vue.js)

Le frontend est développé avec le framework **Vue.js 3** et l'outil de build **Vite**.

- Architecture basée sur des composants réutilisables (**components**).
- Communication avec le backend via des appels asynchrones à l'API REST.

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.
- **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 Plan de Tests

- **Tests Unitaires** : Vérification de la logique métier dans le **Application Core**.
- **Tests d'Intégration** : Validation de la connexion entre l'API et la base de données PostgreSQL.
- **Tests de Disponibilité** : Vérification de la réponse HTTP 200 sur les points d'entrée critiques (Home, Liste des articles).