Documentation technique bestiarum api

“La documentation technique de l’api de bestiarum présente l’architecture de l’application, ses endpoints API, son modèle de données ainsi que les choix technologiques réalisés. Elle est destinée aux développeurs souhaitant installer, comprendre ou maintenir le projet.”

Dernière version le 14/11/2025 à 21h

Par BOHARD hugo

1. Introduction / explication du besoin

l'api bestiarum est un projet de conception, de création et de réflexion sur la création d'une API PHP vanilla (pour une utilisation future). Son besoin est le suivant : créer une API de création de créature mystique, avec la possibilité de faire hybrider ces créatures, les faire combattre. Cet API devra faire appel à des requêtes API d'intelligence artificielle comme OPEN IA, Claude…

Pour ce fait, une deadline d'un peu plus d'un mois a été mise en place pour la réalisation de cet API et de son écosystème

2. Technologies & Environnement

Pour ce projet d'API, quelque contrainte niveau technologie et environnement avait été mise en place avant tout début de créations, donc voici un résumé des différentes technologies de ce projet

Technologies & Environnement

Stack back end : Php vanilla
Bdd : SQlite
Authentification : JSON Web Token (JWT)
Gestion des dépendances : Composer

Outils de Développement

Postman : outil permettant de tester les endpoints de l’API et de vérifier les réponses HTTP.
VS Code : éditeur de code principal.
Git/GitHub : versioning et sauvegarde du projet.

3. Architecture du Projet

3.1 Architecture globale (schéma)

Aujourd'hui l'application repose sur une architecture 2 couches :

- API REST : logique métier et endpoints
- Base de donnée SQLite : stockages des donnée de l'applications

Dans le futur de l'api ainsi que du projet en globalitée, l'application pourras reposée sur une architecture 4 couches :

- API REST : logique métier et endpoints
- Base de donnée SQLite : stockages des donnée de l'applications
- Interface web : interface utilisateur (site web)
- Interface mobile : interface utilisateur (Application mobile

3.2 Structure des dossiers (schéma)

Pour la réalisation de cet API, j'ai optée pour une composition qui s'inspire au modèle MVC (model, view, controller), pour cela voici la structure et l'explication de chaque fichier du code API

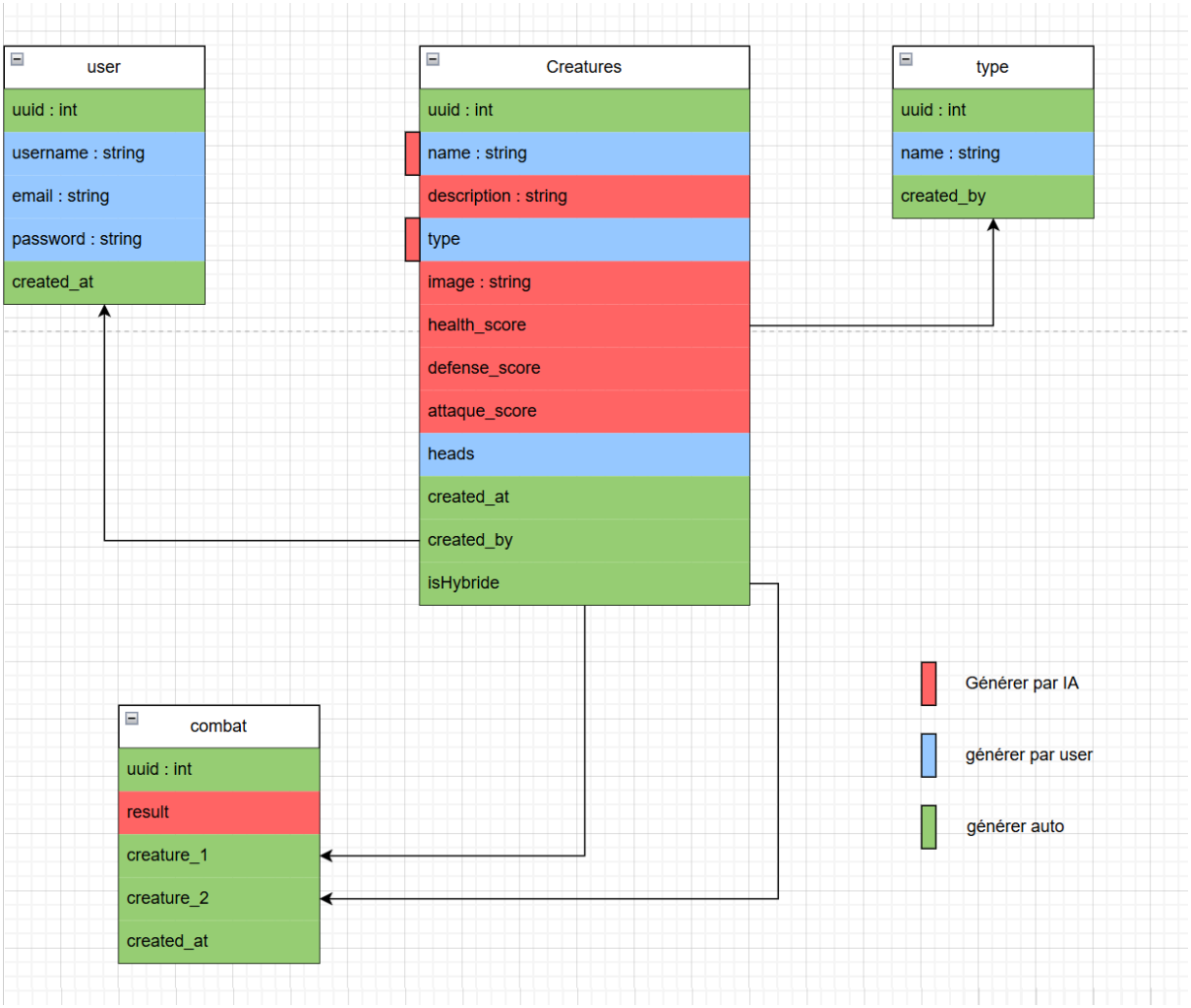
```
├──includes
│   ├──controllers
│   │   ├──auth.controller.php
│   │   ├──hybrids.controller.php
│   │   ├──matchs.controller.php
│   │   ├──monsters.controller.php
│   │   ├──types.controller.php
│   │   └──users.controller.php
│   ├──db
│   │   └──db.connector.php
│   ├──models
│   │   ├──__include__all.php
│   │   ├──Hybrid.class.php
│   │   ├──Match.class.php
│   │   ├──Monster.class.php
│   │   ├──Type.class.php
│   │   └──User.class.php
│   └──pollinations
│   │   ├──monster.description.prompt
│   │   ├──monster.image.prompt
│   │   └──Pollinations.class.php
├──vendor
├──.gitignore
├──.htaccess
├──composer.json
├──composer.lock
├──database.sqlite
├──index.php
└──README
```

4. Modélisation (UML / MERISE)

4.1 Diagramme Use Case (cas d’utilisation)

4.2 MLD

Pour une meilleure approche et une meilleure compréhension de notre logiciel, j'ai réalisé un MLD avec les différentes tables de notre application, bien sûr ce MLD et une V1 (simplifier) pour correspondre à la deadline de 1 mois



5. Documentation de l'API (ENDPOINTS)

Cette section décrit l'ensemble des endpoints exposés par l'API, organisés par domaine fonctionnel.

Chaque route indique la méthode HTTP, l'URL, et l'objectif principal.

5.1 Endpoints de l'API

5.1.1 Authentification

Méthode	Endpoint	Description
POST	`/auth/login`	Permet la connexion d'un utilisateur
POST	`/auth/register`	Crée un nouvel utilisateur
POST	`/auth/logout`	Déconnecte l'utilisateur en invalidant son token

5.1.2 Utilisateurs

Méthode	Endpoint	Description
GET	`/users/{uuid}`	Récupère les informations d'un utilisateur identifié par son UUID
GET	`/users/monstres/{uuid}`	Récupère l'ensemble des monstres associés à un utilisateur

5.1.2 Création de monstres

Méthode	Endpoint	Description
POST	`/monstres/create`	Crée un nouveau monstre

4.2.4. Création d'hybrides

Méthode	Endpoint	Description
POST	`/hybrids/create`	Génère un monstre hybride à partir de deux monstres existants

4.2.5. Matchs entre monstres

Méthode	Endpoint	Description
POST	`/match`	Lance un match entre deux monstres et renvoie le gagnant

4.3. Codes de statut utilisés par l'API

Code	Signification
200	Requête traitée avec succès
201	Ressource créée avec succès
400	Requête invalide ou mal formée
401	Authentification requise ou incorrecte
404	Ressource demandée introuvable

6. Documentation des fonctionnalités (par contrôleur)

Cette section présente les principales fonctions implémentées dans chaque contrôleur, ainsi que leur rôle dans le fonctionnement global de l'application.

6.1 Auth Controller

login($email, $password)

Permet à un utilisateur de se connecter en validant ses identifiants.
Retourne un token d'authentification en cas de succès.

register($username, $email, $password)

Crée un nouveau compte utilisateur et stocke les informations de manière sécurisée.
Retourne un statut HTTP 201 en cas de réussite.

logout($token)

Invalide le token d'un utilisateur pour assurer sa déconnexion.

6.2 Hybrids Controller

create($userId, $monstre1, $monstre2)

Crée un monstre hybride en combinant deux monstres existants.
Les deux monstres sont identifiés via leurs UUID.
Retourne un statut HTTP 201 en cas de création réussie.

6.3 Matchs Controller

match($monstre1, $monstre2)

Réalise un combat entre deux monstres.
Retourne le résultat du match ainsi que le monstre gagnant.
Statut HTTP 200 en cas de réussite.

6.4 Monsters Controller

generate_image($heads, $types)

Génère une image de monstre en fonction du nombre de têtes et du type, via un générateur externe.
Retourne le résultat du match ainsi que le monstre gagnant.
Retourne une image encodée ou une URL.

generate_monster_info($name, $heads, $types)

Génère une description détaillée du monstre à partir de ses caractéristiques.
Retourne un texte descriptif structuré.

6.5 Types Controller

createType($name)

Crée un nouveau type de monstre, sous certaines conditions (existence préalable vérifiée).
Retourne un statut HTTP 201 si la création est valide.

6.6 Users Controller

getUser($uuid)

Récupère l'ensemble des informations d'un utilisateur (hors données sensibles telles que le mot de passe).

getMonsterByUser($uuid)

Récupère tous les monstres appartenant à un utilisateur donné.
Retourne un tableau contenant les données des monstres associés.

7. Justification des choix techniques

Pour ce projet, et indiqué plus haut, j'ai dû réaliser des choix techniques pour la réalisation de mon petit projet de créations d'API, voici mon explication

SQLite : base légère, idéale pour un "side projet" ou un projet d'école, ne nécessite pas de serveur MySQL
PHP vanilla : Le choix technique pour PHP vanille était de pouvoir comprendre la logique dans la réalisation d'une API en vanilla (sans framework). J'aurais pu choisir Laravel, mais ce framework est assez lourd et énormément de fichiers ne servent pas à grand-chose

8. Installation & Lancement pour Dev

Pour tout ce qui est installation et lancement, merci de se référer au README du projet initié dans le repo GitHub du projet

9. Limites & évolutions possibles

Pour ce projet de création d'API vanille, plusieurs limites et questions techniques étaient présentes, l'utilisation d'une API externe Pollinations et de la relier à notre projet était une question très compliquée mais la mise en œuvre était plus simple.