

## RAPPORT DE PROJET

### Projet Carte X

**Objectif** : Créer un gestionnaire de cartes Yu-Gi-Oh! et intégrer l'API externe  
<https://db.ygoprodeck.com/api/v7/cardinfo.php>.

#### Outils et langages :

- React JS et HTML/CSS pour le frontend
- MySQL/MariaDB pour la base de données
- Git/GitHub pour le versionning
- PHP POO pour la page d'administration
- PHPUnit pour les tests unitaires

**Durée** : Du 18/12/2023 à 10:00 au 21/12/2023 à 16:45

**Groupe** : Théo MENANT–FERRY + Alexandre DO + Raphaël ROMERO + Rémi KORZENIOWSKI

**Classe** : BTC2 25.1 Ipssi Paris

## TABLE DES MATIÈRES :

1.	Consignes détaillées.....	3
2.	Récupération et installation du projet.....	4
3.	Présentation du projet.....	5
	a. Répartition des tâches.....	5
	b. Travail sur le frontend.....	6
	c. Travail sur le backend, la base de données et l'api.....	9
	d. Travail sur le backoffice (partie phpadmin).....	14
	e. Travail sur les tests unitaires.....	17
4.	Difficultés rencontrées et solutions.....	22
5.	Remerciements.....	23
6.	Conclusion et retours d'expérience.....	24

## 1. CONSIGNES DÉTAILLÉES :

Développer une application web de gestionnaire de cartes Yu-Gi-Oh! et intégrer l'API externe <https://db.ygoprodeck.com/api/v7/cardinfo.php>.

Le gestionnaire de cartes :

- Les utilisateurs ont la capacité de lire, créer, mettre à jour, supprimer des cartes
- Les données des cartes sont stockées dans une base de donnée MySQL
- Les données des cartes sont exploitées via Node.js pour les utilisateurs
- Les données des cartes sont exploitées via PHP avec PDO pour les administrateurs

Intégration de l'API externe :

- Les données de l'API <https://db.ygoprodeck.com/api/v7/cardinfo.php> sont stockées localement dans une base de donnée

Interface utilisateur :

- Doit respecter l'univers de Yu-Gi-Oh!
- Doit permettre la visualisation de toutes les cartes stockées dans la base de données
- Doit permettre l'ajout, la modification et la suppression de cartes personnalisées (via Node.js et PHP)
- Doit permettre d'afficher les informations détaillées des cartes
- Doit permettre une recherche parmi toutes les cartes importées via système de filtres et de recherche (filtre par nom, par prix et par rareté)

Tests unitaires :

- Tests unitaires adéquats pour la page administrateur en PHP

## 2. RÉCUPÉRATION ET INSTALLATION DU PROJET :

Prérequis :

- WAMP/LAMP/XAMP ou autre plateforme de développement web utilisant MariaDB
- Node.js

Si besoin, télécharger à partir des sites officiels :

- <https://www.wampserver.com/>
- <https://nodejs.org/en/download/>

Récupérer le projet à l'aide de la commande “git clone  
<https://github.com/ttnmeoeayhrnefr/projet-cartex>“

Utiliser le script “ScriptProjetCarteX.sql” pour mettre en place la base de données.

Dans le répertoire “front”, saisir les commandes :

- “npm install”
- “npm start”

Dans le répertoire “back”, saisir les commandes:

- “npm install”
- “node server.js”

Dans le répertoire “backoffice”, saisir les commandes:

- “composer require –dev phpunit/phpunit”
- “npm install”
- “npm start”

Vous êtes prêt à visiter notre site.

### 3. PRÉSENTATION DU PROJET :

#### a. Répartition des tâches :

Travail sur le frontend : Raphaël et Alexandre

Travail sur le backend, la base de donnée et l'API: Alexandre, Théo, Rémi et Raphaël

Travail sur le backoffice : Rémi et Théo

Travail sur les tests unitaires : Théo et Rémi

Travail sur le PowerPoint : Rémi

Travail sur le rapport : Théo

## b. Travail sur le frontend :

Inspiration : <https://www.thunder-stock.com/>

Présentation des pages les plus importantes :

### Page d'accueil :

The screenshot shows the homepage of CarteX. At the top, there's a navigation bar with links for ACCUEIL, MARKETPLACE, INFOS, CRÉATION, BOOSTER, and COLLECTION, along with INSCRIPTION and CONNEXION buttons. The main content area features a large slogan: "Collectionneurs, joueurs, et amateurs, CarteX a la carte qu'il vous faut." Below it is a message from the founder: "Je ne sais pas ce que je raconte mais c'est la pour faire craci. En tout cas ce site il a l'air bien lourd sah ! Je sais pas ce que vous en pensez mais on va forcément avoir 20." There are three cards displayed: "DARK SAIL, THE THIEFLESS MAGICIAN" (Level 4, ATK 1800, DEF 1800), "GROZIK THE THUNDERCLAP TYRANT" (Level 4, ATK 2000, DEF 1800), and "GRAVITY PROTECTION" (Level 1, ATK 0, DEF 0). At the bottom, there's a search bar labeled "Rechercher" and a row of small card thumbnails.

Particularités :

- Header "collant" en "position: fixed"
- Effet accrocheur à l'ouverture du site avec slogan évolutif
- Effet au survol des cartes de la page de présentation
- Carrousel déroulant automatiquement des cartes sélectionnées de manière aléatoire de la base de données
- Feedback utilisateur avec des effets et animations sur chaque élément cliquable

### Marketplace :

The screenshot shows the Marketplace page of CarteX. The top navigation bar is identical to the homepage. The main content area has a heading "Bienvenue sur CarteX," with the subtext "Trouve toutes les cartes que tu désires !". Below this is a search bar with filters for "A - Z", "Prix", "Rareté", "Puissance", and a "Rechercher" button. A "Rechercher" button is also located at the top right of the search bar. The main area displays a grid of six cards: "'A' Cell Breeding Device" (#34541863), "'A' Cell Incubator" (#6416397), "'A' Cell Recombination Device" (#91221901), "'A' Cell Scatter Burst" (#73262676), "Infernible Arms - Almace" (#98319530), and "Infernible Arms - Durendal" (#9748723). Each card includes its name, ID, type, and price.

## Particularités :

- Recherche et filtrage en temps réel
- Ouverture des détails de la carte en cliquant sur cette dernière
- Affichage rotatif des prix Amazon, EBay, TCGPlayer et Cardmarket

## Détails des cartes :

**"A" CELL RECOMBINATION DEVICE**

ID: 3 Card ID: 91231901

Race: Quick-Play Type: Spell Card Attribut: None

Attaque: / Défense: /

0.09\$ 0.20\$  
0.99\$ 0.50\$

Collection: Invasion: Vengeance  
Rareté: Common

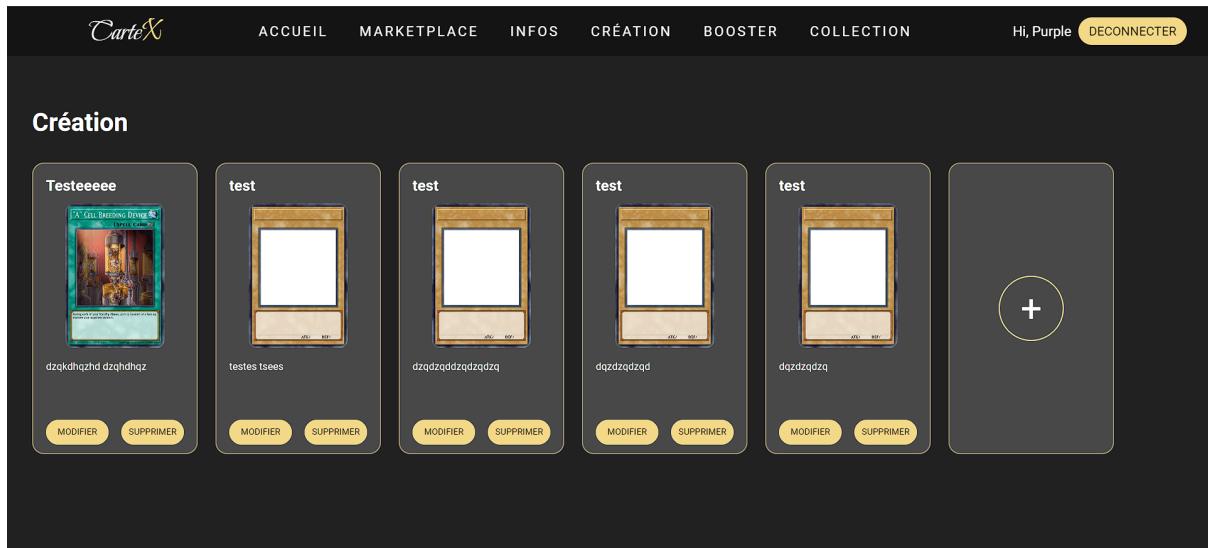
## Collection :

Collection

Abyss Actor - Evil Heel  
Ally of Justice Light Gazer  
Abominable Chamber of the Unchained  
Ancient Gear Ballista  
7

Particularité : Notre site inclut un système de booster à ouvrir, et les cartes ouvertes par les joueurs se retrouvent dans leur collection, à partir de laquelle il est également possible de construire des decks.

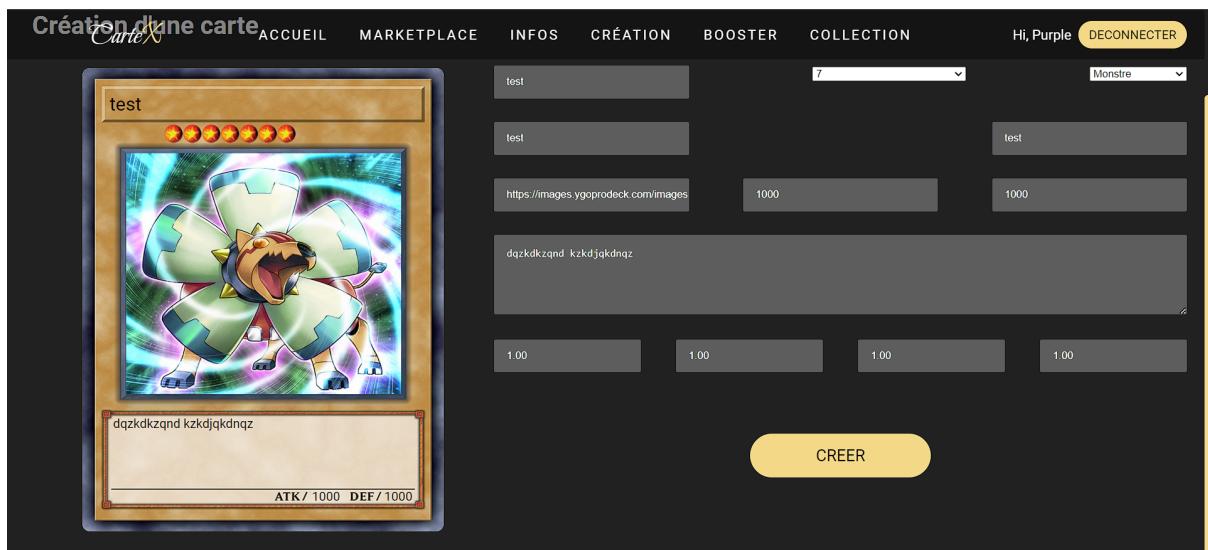
## Menu de création des cartes :



## Particularités :

- Crédation d'un bouton modifier et supprimer qui permettent d'interagir avec la carte en fonction de l'id\_user actuellement connecté.
- Crédation d'un bouton "+", qui permet de créer une carte.

## Menu de création d'une carte :



## Particularités :

- Mise à jour des informations sur l'image de la carte en temps réel
- Page difficile à réaliser
- Difficultés rencontrées à cause d'un problème avec la requête SQL et aussi à cause de la manière dont je récupère les données.

c. Travail sur le backend, la base de donnée et l'API :

Première étape : exploration de l'API <https://db.ygoprodeck.com/api/v7/cardinfo.php>.

Sélection des données utiles :

- nom
- image
- image\_small
- image\_cropped
- id\_carte\_konami
- description
- type
- race
- attaque
- défense
- étoiles
- archetype
- attribut
- prix sur cardmarket
- prix sur tcgplayer
- prix sur ebay
- prix sur amazon
- nom de set
- rareté

Sauvegarde des données utiles des 500 premières cartes de l'API dans une base de donnée avec le modèle suivant :

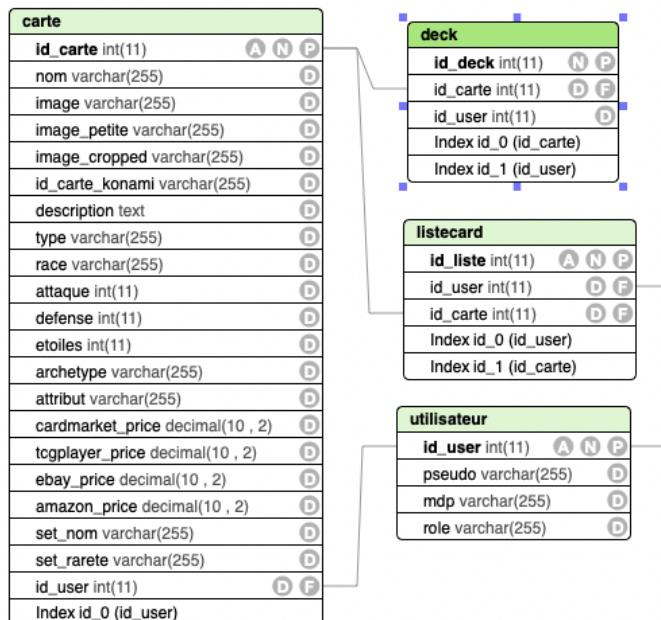


Schéma produit avec SQL Editor

Légende :

- P = Primary key
- A = Auto increment
- N = Not null
- F = Foreign Key
- D = Null

Deuxième étape : mise en place d'un serveur.

```
const express = require("express");
const mariadb = require("mariadb");
const bcrypt = require("bcryptjs");
const cors = require("cors");
const { reset } = require("nodemon");
require("dotenv").config();
const app = express();
const port = 3001;

const pool = mariadb.createPool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PWD,
  connectionLimit: 20,
});
```

*Extrait de server.js*

Explication des imports :

- Express : framework web pour Node.js qui simplifie la création d'applications web en fournissant des fonctionnalités pour gérer les routes, les requêtes et les réponses HTTP
- mariadb : client de base de données pour MariaDB, qui est une base de données relationnelle similaire à MySQL, permet la connexion à notre base de données
- bcrypt : bibliothèque pour le hachage de mots de passe
- cors : permet de gérer les requêtes HTTP entre différents domaines
- dotenv : module utilisé pour charger des variables d'environnement depuis un fichier ".env"
- mariadb.createPool() : crée un pool de connexions à la base de données MariaDB

Troisième étape : écriture des CRUDS carte et utilisateur et tests avec Postman.

Fonctions particulières :

- Création d'un CRUD deck appelé en fonction de la clé étrangère id\_user stockée dans le localstorage.

# .POSSI

```
//CRUD DECK
app.get("/deck/:id", async (req, res) => {
  let conn;
  try {
    console.log("lancement de la console");
    conn = await pool.getConnection();
    console.log("lancement de la requête");
    const rows = await conn.query("select * from deck where id_user = ?", [req.params.id]);
    console.log(rows);
    res.status(200).json(rows);
  } catch (err) {
    console.log(err);
  }
});
```

*CRUD deck*

- Créations de fonctions sélectionnant des cartes aléatoires à partir de la base de données pour l'affichage du carrousel et pour l'ouverture des boosters.

```
// GET 10 RANDOM CARDS
app.get('/cartes/random/random', async (req, res) => {
  let conn;
  try {
    console.log('Lancement de la connexion');
    conn = await pool.getConnection();
    console.log('Lancement de la requête');
    const rows = await conn.query('SELECT * FROM carte ORDER BY RAND() LIMIT 300');
    console.log(rows);
    res.status(200).json(rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Erreur lors de la récupération des cartes aléatoires.' });
  } finally {
    if (conn) conn.release();
  }
});

//GET 5 RANDOM CARDS FOR BOOSTER
app.get('/cartes/random/booster', async (req, res) => {
  let conn;
  try {
    console.log('Lancement de la connexion');
    conn = await pool.getConnection();
    console.log('Lancement de la requête');
    const rows = await conn.query('SELECT * FROM carte ORDER BY RAND() LIMIT 5');
    console.log(rows);
    res.status(200).json(rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Erreur lors de la récupération des cartes aléatoires.' });
  } finally {
    if (conn) conn.release();
  }
});
```

*Gets aléatoires*

# .IPSSI

Dernière étape : gestion de l'inscription et la connexion des utilisateurs via le localstorage. Gestion de la rétention des informations utilisateur via le localstorage. Gestion de la redirection des utilisateurs en fonction de leur rôle administrateur ou utilisateur standard. Gestion des erreurs possibles.

Le composant “register.jsx” permet à un utilisateur de s'inscrire en fournissant un pseudo et un mot de passe. Il vérifie également si un utilisateur est déjà connecté avant de permettre une nouvelle inscription.

```
// DATA UTILISATEUR
const new_user = {
  pseudo: pseudo,
  mdp: mdp,
  role: 0,
};
```

*Informations enregistrées lors de l'inscription d'un utilisateur*

Le composant “connect.jsx” gère le formulaire de connexion des utilisateurs en envoyant les informations d'identification à une API, stockant certaines données importantes localement et redirigeant ensuite l'utilisateur vers la page d'accueil après la connexion réussie.

```
try {
  const response = await fetch("http://localhost:3001/connexion", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ pseudo, mdp })
  });

  const data = await response.json();

  if (response.ok) {
    console.log(data.message);
    localStorage.setItem("pseudo", pseudo);

    const recupererInfoUser = await axios.get(`http://localhost:3001/utilisateurs/nom/${pseudo}`);
    const id_user = recupererInfoUser.data[0].id_user;
    let role = recupererInfoUser.data[0].role;

    console.log(id_user);
    console.log(role);

    if (role === "0") {
      role = "utilisateur";
    } else if (role === "1") {
      role = "admin";
    }

    console.log(id_user);
    console.log(role);

    localStorage.setItem("id_user", id_user);
    localStorage.setItem("role", role);

    window.location.href = '/';
  } else {
    console.log(data.message);
  }
} catch (err) {
  console.log(err);
}
```

*Logique de connexion*

# .POSSI

```
// VERIFICATION
useEffect(() => {
  const user_connected = localStorage.getItem("pseudo");
  if (user_connected) {
    set_user_connected(true);
    alert("Un utilisateur est déjà connecté.");
    document.getElementById("pseudo").disabled = true;
    document.getElementById("mdp").disabled = true;
  }
}, []);
```

Vérifications

## d. Travail sur le backoffice :

Création de la classe Cards ainsi que de son CRUD.

Création de la classe Utilisateurs ainsi que de son CRUD.

Création du DAO ainsi que de toutes les méthodes nécessaires au bon fonctionnement des pages d'administration.

Brève explication du contenu de notre page DAO.php :

### **Classe Cards :**

Cette classe est utilisée pour créer des objets représentant des cartes.

Elle possède des propriétés privées telles que id, nom, image, id\_konami, description, etc., qui sont initialisées via le constructeur.

Elle a des méthodes pour récupérer (get) et définir (set) ces propriétés.

### **Classe Utilisateurs :**

Cette classe est utilisée pour créer des objets représentant des utilisateurs.

Elle possède des propriétés privées comme id, pseudo, password, role, initialisées via le constructeur.

Elle a des méthodes pour récupérer (get) et définir (set) ces propriétés.

### **Classe DAO :**

Cette classe contient des méthodes qui agissent comme une interface entre l'application et la base de données.

Elle utilise PDO pour interagir avec la base de données.

listAllUsers() : Récupère tous les utilisateurs de la base de données.

listUserById(\$id\_user) : Récupère un utilisateur spécifique en fonction de son ID.

addUser(\$pseudo, \$mdp, \$role) : Ajoute un nouvel utilisateur à la base de données.

updateUserById(\$pseudo, \$mdp, \$role, \$id) : Met à jour les informations d'un utilisateur existant en fonction de son ID.

removeUserById(\$id\_user) : Supprime un utilisateur de la base de données.

# .IPSSI

En plus de ces méthodes pour les utilisateurs, il y a également des méthodes similaires pour les cartes :

addCard(...) : Ajoute une nouvelle carte à la base de données.

listAllCards() : Récupère toutes les cartes de la base de données.

listCardById(\$idCarte) : Récupère une carte spécifique en fonction de son ID.

updateCardById(...) : Met à jour les informations d'une carte existante en fonction de son ID.

removeCardById(\$idCarte) : Supprime une carte de la base de données.

## Remarques supplémentaires :

Le code utilise des **requêtes SQL préparées** pour éviter les attaques par injection SQL.

Il gère également certaines validations, comme **s'assurer que les champs obligatoires sont remplis lors de l'ajout d'une carte ou d'un utilisateur.**

## Création de la page de gestion des utilisateurs :

ID	Pseudo	Mot de passe	Role	Actions
1	admin	\$2a\$10\$WNRUR0i3AKlchAczsfEr.YelCgu7gdy3.eumuEk64MvJ9o3zBaaS1		<a href="#">Modifier</a> <a href="#">Supprimer</a>
2	test	\$2a\$10\$XFeM5qxbmixuFUfwhNzjuB/Ghrct4lJo9dt6KwB8a8nJ2HlcA11.0		<a href="#">Modifier</a> <a href="#">Supprimer</a>
3	purple	\$2a\$10\$Q.HLOVA5t9juajJ6FqsoPKJykRFuFTIH5c4rm4AT1kI2W1Ktp/mS0		<a href="#">Modifier</a> <a href="#">Supprimer</a>
6	purplezer	\$2a\$10\$.EYj5ZGg9D6LPwso.IlyZ06opBbRG/Dk/gzyMloMo2H/TgcVGrbkK		<a href="#">Modifier</a> <a href="#">Supprimer</a>

## Création de la page de gestion des cartes :

Panneau d'administration [→]

### Gestion cartes Yu-Gi-Oh !

Ajouter carte																	
ID	Nom	Image	Img Petite	Img Cropped	Id Konami	Description	Type	Race	Attaque	Défense	Étoile	Archetype	Attribut	Prix Cardmarket	Prix Tcgplayer	Prix Ebay	Prix Amazon
1	"A" Cell Breeding Device				34541863		Spell Card	Continuous				Alien		0.15	0.19	0.99	24.45
2	"A" Cell Incubator				64163367		Spell Card	Continuous				Alien		0.15	0.22	1.25	0.50
3	"A" Cell Recombination Device				91231901		Spell Card	Quick-Play				Alien		0.09	0.20	0.99	0.50
4	"A" Cell Scatter Burst				73262676		Spell Card	Quick-Play				Alien		0.09	0.10	2.00	9.76
5	"Infernoble Arms - Almace"				98319530		Spell Card	Equip				Infernoble Arms		0.00	0.83	0.00	0.00
6	"Infernoble Arms -				37478723		Spell Card	Equip				Noble Knight		0.10	0.13	0.99	2.96

## Création des pages de modification ou suppression des cartes et des utilisateurs.

### Création d'une page dédiée à l'import du fichier SQL de la base de données.

Panneau d'administration [→]

### Attention DANGER !

**Importation de script SQL**

Sélectionnez le fichier SQL à importer :  Aucun fichier choisi

**Formatage de la DB**

[→]

## Création d'un script d'import de la base de données.

## e. Travail sur les tests unitaires :

**Première étape** : installation de PHPUnit dans notre projet.

**Seconde étape** : création de notre fichier phpunit.xml, qui sera responsable du lancement de chacun de nos tests unitaires et de nos tests d'intégration.

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit colors="true" bootstrap="vendor/autoload.php">
    <php>
        <!-- <env name="DB_CONNECTION" value="mysql"></env>
        <env name="DB_HOST" value="127.0.0.1"></env>
        <env name="DB_PORT" value=""></env>
        <env name="DB_DATABASE" value="test"></env>
        <env name="DB_USERNAME" value="root"></env>
        <env name="DB_PASSWORD" value="root"></env> -->
    </php>
    <testsuites>
        <testsuite name="Unit Tests">
            <directory>tests/Intégration</directory>
            <directory>tests/Unit</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

fichier phpunit.xml

Ce fichier de configuration est conçu pour exécuter des tests PHPUnit.  
Il spécifie deux répertoires (tests/Intégration et tests/Unit) où les tests sont organisés.

**Troisième étape** : identification des besoins et des exceptions à gérer :

- Un test par fonction utilisée par le DAO
- Des tests réalisés sur la base de donnée
- Des tests réalisés en SQLite avec une base de donnée simulée
- Une exception pour tous les champs ne pouvant pas être laissés vides
- Une exception pour les utilisateurs avec un rôle autre que 0 (utilisateur standard) ou 1 (administrateur)
- Une exception pour le remplissage des champs des cartes différents entre les cartes monstre et les cartes magie ou piège

```
public function testAddUser()
{
    $pseudo = 'test_user';
    $password = 'test_password';
    $role = '0';

    $userId = $this->dao->addUser($pseudo, $password, $role);

    $this->assertNotEmpty($userId, 'id_user ne doit pas etre vide');

    $user = $this->dao->listUserById($userId);

    $this->assertNotEmpty($user, 'utilisateur ne doit pas etre vide');
    $this->assertEquals($pseudo, $user['pseudo']);
    $this->assertTrue(password_verify($password, $user['mdp']), 'verification du mdp raté');
    $this->assertEquals($role, $user['role']);

    return $userId;
}
```

exemple de test unitaire

# .POSSI

Ce test unitaire vise à vérifier si l'ajout d'un utilisateur dans la base de données fonctionne correctement, et il teste également la cohérence des données ajoutées (pseudo, mot de passe, rôle) avec les données récupérées après l'ajout.

```
protected function setUp(): void
{
    $this->pdo = new PDO('sqlite::memory:');
    $this->pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $this->pdo->exec('CREATE TABLE carte (
        id_carte INTEGER(11) PRIMARY KEY,
        nom VARCHAR(255),
        image VARCHAR(255),
        image_small VARCHAR(255),
        image_cropped VARCHAR(255),
        id_carte_konami VARCHAR(255),
        description TEXT,
        type VARCHAR(255),
        race VARCHAR(255),
        attaque INTEGER(11),
        defense INTEGER(11),
        etoiles INTEGER(11),
        archetype VARCHAR(255),
        attribut VARCHAR(255),
        cardmarket_price DECIMAL(10, 2),
        tcgplayer_price DECIMAL(10, 2),
        ebay_price DECIMAL(10, 2),
        amazon_price DECIMAL(10, 2),
        set_nom VARCHAR(255),
        set_rarete VARCHAR(255)
    ');
    $this->pdo->exec('CREATE TABLE utilisateur (
        id_user INTEGER(11) PRIMARY KEY,
        pseudo VARCHAR(255),
        mdp VARCHAR(255),
        role VARCHAR(255)
    ');
}
```

*mise en place de la base de donnée simulée en SQLite*

Cette méthode `setUp()` est utilisée pour créer une base de données SQLite en mémoire avec deux tables (`carte` et `utilisateur`) préalablement définies, fournissant ainsi un environnement de test propre et isolé pour les tests unitaires.

# .POSSI

```
public function testListAllUsers()
{
    $dao = new DAO($this->pdo);

    $dao->addUser('user1', 'password1', 'role1');
    $dao->addUser('user2', 'password2', 'role2');
    $dao->addUser('user3', 'password3', 'role3');

    $users = $dao->listAllUsers();

    $this->assertCount(3, $users);

    $this->assertEquals('user1', $users[0]['pseudo']);
    $this->assertEquals('role1', $users[0]['role']);
}
```

*exemple de test d'intégration*

Ce test unitaire vise à vérifier le fonctionnement de la méthode listAllUsers() de la classe DAO en s'assurant qu'elle retourne la liste attendue des utilisateurs à partir de la base de données, en vérifiant la correspondance des pseudonymes et des rôles des utilisateurs récupérés.

```
public function __construct($id, $pseudo, $password, $role)
{
    if (empty($pseudo)) {
        throw new InvalidArgumentException("Le pseudo de l'utilisateur est requis.");
    }

    if (empty($password)) {
        throw new InvalidArgumentException("Le mot de passe de l'utilisateur est requis.");
    }

    if ($role !== '0' && $role !== '1') {
        throw new InvalidArgumentException("Le rôle de l'utilisateur doit être soit 0 soit 1.");
    }
    $this->id = $id;
    $this->pseudo = $pseudo;
    $this->password = $password;
    $this->role = $role;
}
```

*exemple de vérification des champs*

Ce constructeur vérifie et initialise les propriétés d'un objet utilisateur (id, pseudo, password, rôle) en effectuant des vérifications sur les paramètres passés pour s'assurer qu'ils correspondent aux critères attendus.

# .POSSI

```
public function addCard($nom,$image,$image_small,$image_cropped,$id_konami,$description,$type,$race,$attack,$defense,$stars) {
    if ($type === 'monstre') {
        $requiredMonsterType = [
            'race' => $race,
            'attack' => $attack,
            'defense' => $defense,
            'stars' => $stars,
            'attribute' => $attribute
        ];
        foreach ($requiredMonsterType as $field => $value) {
            if (empty($value)) {
                throw new InvalidArgumentException("Le champ '$field' est obligatoire pour une carte de type 'monstre'.");
            }
        }
    }
    $required = [
        'nom' => $nom,
        'image' => $image,
        'image_small' => $image_small,
        'image_cropped' => $image_cropped,
        'id_konami' => $id_konami,
        'description' => $description,
        'type' => $type,
        'cardmarket_price' => $cardmarket_price,
        'tcgplayer_price' => $tcgplayer_price,
        'ebay_price' => $ebay_price,
        'amazon_price' => $amazon_price,
        'collection' => $collection,
        'rarete' => $rarete
    ];
    foreach ($required as $field => $value) {
        if (empty($value)) {
            throw new InvalidArgumentException("Le champ '$field' de la carte est requis.");
        }
    }
}
```

*exemple de vérification des différences entre les différents types de cartes*

La méthode addCard vérifie les champs obligatoires pour une carte de type "monstre" et insère les détails de la carte dans une base de données, renvoyant son identifiant ou signalant une erreur en cas d'échec.

# .POSSI

```
public function testAddUserInvalidUsername()
{
    $this->expectException(InvalidArgumentException::class);
    $dao = new DAO($this->pdo);

    $dao->addUser('', 'password', '0');
}

public function testAddUserInvalidPassword()
{
    $this->expectException(InvalidArgumentException::class);
    $dao = new DAO($this->pdo);

    $dao->addUser('pseudo', '', '1');
}

public function testAddUserInvalidRole()
{
    $this->expectException(InvalidArgumentException::class);
    $dao = new DAO($this->pdo);

    $dao->addUser('pseudo', 'password', '2');
}

public function testAddCardInvalidArgument()
{
    $this->expectException(InvalidArgumentException::class);

    $dao = new DAO($this->pdo);

    $dao->addCard('', 'image.jpg', 'image_small.jpg', 'image_');
}

public function testAddCardExceptionForMonsterType()
{
    $dao = new DAO($this->pdo);

    $this->expectException(InvalidArgumentException::class);
    $dao->addCard('name', 'image.jpg', 'image_small.jpg', 'ima
}
```

*exemple de tests des exceptions attendues*

Ces fonctions de test vérifient respectivement la validation des arguments pour l'ajout d'utilisateur, y compris les noms d'utilisateur invalides, les mots de passe vides et les rôles non valides, ainsi que la validation des arguments pour l'ajout de carte, y compris les champs vides requis pour une carte de type "monstre".

## 4. DIFFICULTES RENCONTREES ET SOLUTIONS :

### **Théo :**

Les principales difficultés que j'ai rencontrées lors de la réalisation de ce projet ont été des difficultés techniques. En effet, mon manque d'expérience sur GitHub m'a poussé à solliciter l'aide de mes camarades à plusieurs reprises lorsque je restais coincé dans une branche sans savoir pourquoi. Une autre difficulté a été de réadapter des fonctionnalités que nous avions déjà développées dans des projets précédents pour qu'elles soient fonctionnelles pour ce projet. Enfin, que ce soit pour des fonctionnalités du site ou pour les tests unitaires il est arrivé à plusieurs reprises que je push des fonctionnalités qui fonctionnaient pour moi mais pas pour les autres que ce soit à cause des différentes versions de base de données utilisées ou des différents outils utilisés par chacun (WAMP/XAMPP), ce qui a causé plusieurs pertes de temps inutiles.

### **Alex :**

Au cours du projet, j'ai surmonté deux principaux défis. Initialement, l'extraction et le transfert des données de l'API vers la base de données étaient entravés par une approche trop complexe. En sollicitant l'aide de mes collègues, j'ai simplifié le processus avec succès. La création de cartes a également posé des problèmes, d'abord avec la requête SQL, puis avec la transmission des données, générant des cartes "null". J'ai résolu cela en révisant ma méthode de récupération des données utilisateur, en optant pour une approche plus claire. Ces obstacles ont renforcé ma compréhension pratique et ma collaboration avec l'équipe, illustrant ma capacité à résoudre des problèmes et à améliorer mes compétences tout au long du projet.

### **Rémi :**

Plusieurs difficultés techniques se sont présentées lors du déroulé de ce projet, qu'il s'agisse de problèmes matériels, de problèmes logiciels ou de problèmes de méthodologie au sein du groupe.

### **Raphaël :**

Aucune difficulté particulière n'a été rencontrée pendant ce projet.

## 5. REMERCIEMENTS :

Nous tenons à exprimer notre gratitude à l'égard de **l'IPSSI Paris**, institution qui a été le cadre propice à l'épanouissement de notre projet de gestionnaire de cartes Yu-Gi-Oh!. Ce projet, essentiel à notre formation, nous a permis d'explorer et d'appliquer des connaissances pratiques cruciales dans le domaine du développement web.

Nous tenons à remercier chaleureusement Monsieur **Maxime OUDOT** et Madame **Laetitia PINTO**, nos professeurs, pour leur précieux soutien, leur expertise et leurs conseils avisés tout au long de la préparation à ce projet. Leur dévouement et leur disponibilité ont été des atouts majeurs dans l'aboutissement de ce travail.

La conception de ce gestionnaire de cartes Yu-Gi-Oh! nous a offert une opportunité exceptionnelle d'approfondir nos **compétences en développement web, en intégration d'API et en gestion de bases de données**, tout en nous permettant de répondre aux exigences spécifiques du projet.

Ce projet a été une expérience formatrice et enrichissante, nous permettant d'acquérir des compétences précieuses et de consolider nos connaissances dans un environnement stimulant.

Nous sommes reconnaissants envers **l'IPSSI Paris, nos professeurs et tous ceux qui ont contribué de près ou de loin à la réalisation de ce projet**. Cette expérience restera un pilier important dans notre parcours académique et professionnel.

## 6. CONCLUSION ET RETOURS D'EXPÉRIENCE :

En conclusion, la réalisation de ce gestionnaire de cartes Yu-Gi-Oh! a été une expérience enrichissante et formatrice pour notre équipe. Ce projet nous a permis d'explorer en profondeur divers aspects du développement web, de l'intégration d'API et de la gestion de bases de données, consolidant ainsi nos compétences techniques.

La répartition des tâches en fonction des forces de chacun a été cruciale pour le succès de ce projet, permettant à chacun d'entre nous de se concentrer sur des domaines spécifiques et de contribuer de manière significative au développement global de l'application.

Les défis rencontrés, qu'ils soient techniques ou liés à la collaboration en équipe, ont été des opportunités d'apprentissage. Les difficultés rencontrées ont été surmontées grâce à notre collaboration, notre détermination et notre capacité à trouver des solutions ensemble. Nous tenons à exprimer notre gratitude envers l'IPSSI Paris, nos professeurs, et tous ceux qui ont contribué de près ou de loin à la réalisation de ce projet. Leur soutien, leurs conseils et leur expertise ont été des atouts précieux tout au long de cette aventure.

Enfin, ce projet restera un jalon significatif dans notre parcours académique et professionnel. Il nous a permis d'acquérir des compétences pratiques, de consolider nos connaissances et de nous préparer pour de futurs défis dans le domaine du développement web.

Nous sommes fiers du travail accompli et des résultats obtenus, et nous sommes convaincus que cette expérience nous servira de base solide pour nos projets futurs.