Il collezionista

Creare un'applicazione RESTful per catalogare oggetti da collezione utilizzando PHP 8, PDO, OOP e il pattern MVC è un progetto ideale per capire come sviluppare un'architettura moderna e scalabile. Di seguito mostro uno schema base e un esempio pratico di come implementare una tale applicazione.

Struttura del Progetto (MVC)

```
- app/
  - Controllers/
    - ItemController.php
  - Models/
    - Item.php
  - Views/
    - json.php
- public/
  - index.php
- config/
  - database.php
- uploads/
- vendor/
- .htaccess
```

1. **Configurazione del Database**

Nel file `config/database.php`, puoi configurare la connessione al database utilizzando PDO.

```
<?php
// config/database.php
namespace Config;

use PDO;
use PDOException;

class Database {
    private $host = 'localhost';
    private $db_name = 'catalogo_collezioni';
    private $username = 'root';
    private $password = '';
    public $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname="
. $this->db_name, $this->username, $this->password);
            $this->conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
```

```
        } catch (PDOException $exception) {
            echo "Errore nella connessione: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
```

2. **Modello (Model)**

Il modello rappresenta l'oggetto da collezione. Il file `app/Models/Item.php` gestisce le interazioni con il database.

```
<?php
// app/Models/Item.php
namespace App\Models;

use PDO;

class Item {
    private $conn;
    private $table = 'items';

    public function __construct($db) {
        $this->conn = $db;
    }

    // Creare un nuovo oggetto
    public function create($data) {
        $query = "INSERT INTO " . $this->table . " (title, description,
size, price, image) VALUES (:title, :description, :size, :price, :image)";
        $stmt = $this->conn->prepare($query);

        // Bind dei parametri
        $stmt->bindParam(':title', $data['title']);
        $stmt->bindParam(':description', $data['description']);
        $stmt->bindParam(':size', $data['size']);
        $stmt->bindParam(':price', $data['price']);
        $stmt->bindParam(':image', $data['image']);

        if ($stmt->execute()) {
            return true;
        }
        return false;
    }

    // Leggere tutti gli oggetti
    public function read() {
        $query = "SELECT * FROM " . $this->table;
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt;
```

```
    }
}
```

3. **Controller**

Il `Controller` gestisce le richieste HTTP e richiama il `Model`. Il file `app/Controllers/ItemController.php` gestisce le operazioni CRUD (creazione, lettura).

```php
<?php
// app/Controllers/ItemController.php
namespace App\Controllers;

use App\Models\Item;
use Config\Database;

class ItemController {
    private $db;
    private $itemModel;

    public function __construct() {
        $database = new Database();
        $this->db = $database->getConnection();
        $this->itemModel = new Item($this->db);
    }

    // Creazione di un nuovo oggetto
    public function create() {
        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            $data = [
                'title' => $_POST['title'],
                'description' => $_POST['description'],
                'size' => $_POST['size'],
                'price' => $_POST['price'],
                'image' => $this->uploadImage()
            ];

            if ($this->itemModel->create($data)) {
                $this->respond(['message' => 'Oggetto creato con
successo']);
            } else {
                $this->respond(['message' => 'Errore nella creazione
dell\'oggetto'], 500);
            }
        }
    }

    // Recuperare tutti gli oggetti
    public function read() {
        $result = $this->itemModel->read();
        $items = $result->fetchAll(PDO::FETCH_ASSOC);
        $this->respond($items);
```

```
    }

    // Upload dell'immagine
    private function uploadImage() {
        if (isset($_FILES['image']) && $_FILES['image']['error'] == 0) {
            $targetDir = "uploads/";
            $targetFile = $targetDir . basename($_FILES["image"]["name"]);
            move_uploaded_file($_FILES["image"]["tmp_name"], $targetFile);
            return $targetFile;
        }
        return null;
    }

    // Risposta JSON
    private function respond($data, $status = 200) {
        http_response_code($status);
        header('Content-Type: application/json');
        echo json_encode($data);
    }
}
```

4. **Route e Punto di Ingresso**

Il file `public/index.php` gestisce le richieste e instrada le chiamate ai controller appropriati. Qui potresti avere un semplice router che chiama i metodi del controller in base all'endpoint.

```
<?php
// public/index.php
require_once '../vendor/autoload.php';

use App\Controllers\ItemController;

$controller = new ItemController();

// Router semplice
$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
if ($uri === '/items' && $_SERVER['REQUEST_METHOD'] === 'GET') {
    $controller->read();
} elseif ($uri === '/items' && $_SERVER['REQUEST_METHOD'] === 'POST') {
    $controller->create();
} else {
    http_response_code(404);
    echo json_encode(['message' => 'Endpoint non trovato']);
}
```

5. **File .htaccess**

Per fare in modo che tutte le richieste vengano gestite da `public/index.php`, puoi creare un file `.htaccess` nella cartella `public/`:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php [QSA,L]
```

6. **Database SQL**

Qui un esempio della tabella `items` per il database.

```
CREATE TABLE `items` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `size` varchar(100) NOT NULL,
  `price` decimal(10, 2) NOT NULL,
  `image` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

7. **Esempio di Chiamate API**

1. **Creare un nuovo oggetto (POST)**
   - Endpoint: `/items`
   - Metodo: `POST`
   - Parametri: `title`, `description`, `size`, `price`, `image` (file upload)
2. **Recuperare tutti gli oggetti (GET)**
   - Endpoint: `/items`
   - Metodo: `GET`

Conclusione

Questo esempio fornisce un'implementazione semplice di un'app RESTful con PHP 8, utilizzando OOP, PDO, e il pattern MVC. La promozione dei parametri nel costruttore di PHP 8 potrebbe ulteriormente semplificare la gestione delle proprietà delle classi, rendendo il codice ancora più leggibile e gestibile.

Spiegazione .htaccess

Il file `.htaccess` è utilizzato per configurare a livello di directory il comportamento del server web Apache. Le istruzioni che hai fornito fanno parte di una configurazione comune per gestire le richieste di URL tramite un unico file di front controller, spesso usato in applicazioni PHP che seguono l'architettura MVC.

Spiegazione delle Istruzioni

1. `RewriteEngine On`

Questa direttiva abilita il modulo di riscrittura (`mod_rewrite`) di Apache. Il modulo permette di riscrivere gli URL in base a regole definite, consentendo la creazione di URL "puliti" e l'instradamento delle richieste a specifici script PHP.

2. `RewriteCond %{REQUEST_FILENAME} !-f`

Questa è una **condizione di riscrittura** che viene valutata prima della regola successiva (`RewriteRule`). La condizione controlla se il file richiesto **non** esiste fisicamente sul server.

- `%{REQUEST_FILENAME}` rappresenta il percorso completo del file richiesto.
- `!-f` significa "se non esiste un file fisico". Quindi, questa condizione è vera solo se il file specificato nell'URL non esiste.

3. `RewriteCond %{REQUEST_FILENAME} !-d`

Simile alla condizione precedente, ma controlla se la richiesta non corrisponde a una directory.

- `%{REQUEST_FILENAME}` rappresenta il percorso completo della directory richiesta.
- `!-d` significa "se non esiste una directory fisica". Questa condizione è vera solo se la directory specificata nell'URL non esiste.

4. `RewriteRule ^(.*)$ index.php [QSA,L]`

Questa è la regola di riscrittura vera e propria. Dice ad Apache come riscrivere l'URL quando le condizioni precedenti sono soddisfatte.

- `^(.*)$`: Questo è un'espressione regolare che cattura qualsiasi stringa dopo il nome del dominio. `^(.*)$` corrisponde a qualsiasi URL.
- `index.php`: Se la richiesta non corrisponde a un file o a una directory esistente, la richiesta viene riscritta per essere gestita da `index.php`.
- `[QSA,L]`:
  - `QSA` (Query String Append): Se nell'URL originale era presente una query string (es. `?id=123`), questa viene mantenuta e aggiunta alla query string di destinazione.
  - `L` (Last): Indica che questa è l'ultima regola da applicare. Una volta applicata questa regola, Apache non considera ulteriori regole di riscrittura.

Cosa fa il codice nel contesto MVC?

Queste regole sono comunemente usate in applicazioni MVC per inviare tutte le richieste a un singolo punto di ingresso (`index.php`), che funge da front controller.

- **Gestione centralizzata**: In un'applicazione MVC, tutte le richieste (ad esempio `https://tuosito.com/prodotti/dettaglio/5`) vengono reindirizzate a `index.php`. Da lì, il codice PHP può analizzare l'URL, caricare il controller e l'azione corretti, e infine restituire la risposta appropriata.

- **URL puliti**: L'uso di `.htaccess` consente di creare URL puliti e user-friendly, senza estensioni di file o parametri query string visibili.

Esempio di Flusso:

1. L'utente richiede `https://tuosito.com/prodotti/dettaglio/5`.
2. Apache controlla se il file o la directory corrispondente esistono fisicamente:
    - Se esistono, Apache serve il file o la directory.
    - Se non esistono, Apache riscrive l'URL per essere gestito da `index.php`.
3. `index.php` riceve la richiesta e decide come elaborarla (ad esempio, potrebbe estrarre il controller `prodotti`, l'azione `dettaglio`, e l'ID `5` per visualizzare un prodotto specifico).