Esercitazione PHP 8: Gestione di un Magazzino di Prodotti Tecnologici

Tema: Magazzino di Prodotti Tecnologici

In questa esercitazione svilupperai un'applicazione web che gestisce un magazzino di prodotti tecnologici (computer, smartphone, tablet, ecc.) utilizzando **PHP 8**. L'applicazione sarà organizzata secondo il pattern **MVC** (Model-View-Controller) e sfrutterà la **programmazione orientata agli oggetti (OOP)**, con l'uso di **namespace** per una corretta gestione del codice. Implementerai anche funzionalità come il caricamento di file (immagini dei prodotti), sessioni, autenticazione degli utenti, e un'API REST in formato **JSON**. La connessione al database sarà gestita tramite **PDO** e il database **MySQL** ospiterà i dati dei prodotti e degli utenti.

Obiettivi dell'Esercitazione

1. Strutturare l'applicazione seguendo il pattern **MVC**.
2. Implementare l'autenticazione degli utenti con login, logout e gestione delle sessioni.
3. Consentire l'upload delle immagini dei prodotti.
4. Gestire il database **MySQL** tramite **PDO** per archiviare utenti e prodotti.
5. Fornire un'**API REST** per accedere e gestire i prodotti in formato **JSON**.
6. Organizzare il codice con **namespace** per migliorarne modularità e manutenibilità.

Struttura del Progetto

Il progetto sarà organizzato con la seguente struttura:

```
/project_root
    /app
        /controllers
            ProductController.php
            UserController.php
        /models
            Product.php
            User.php
        /views
            /products
                index.php
                create.php
            /user
                login.php
        /config
            database.php
            routes.php
    /public
        index.php
        /uploads
```

```
            /products
    /api
        products.php
    /vendor
    .htaccess
    composer.json
```

Passi dell'Esercitazione

1. Setup del Database

Crea un database MySQL chiamato `tech_warehouse` con due tabelle principali: `users` e `products`. La tabella `users` gestirà gli utenti (per l'autenticazione), mentre la tabella `products` terrà traccia dei prodotti tecnologici nel magazzino.

```sql
CREATE DATABASE tech_warehouse;

USE tech_warehouse;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL
);

CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT NULL,
    price DECIMAL(10,2) NOT NULL,
    stock INT NOT NULL,
    image VARCHAR(255) NULL,
    category VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Dati fake per utenti
INSERT INTO users (username, password, email) VALUES
('admin', 'password_hash1', 'admin@example.com'),
('user1', 'password_hash2', 'user1@example.com');

-- Dati fake per prodotti
INSERT INTO products (name, description, price, stock, image, category)
VALUES
('Laptop X1', 'Laptop ad alte prestazioni', 1200.00, 15, 'laptop_x1.png',
'Laptop'),
('Smartphone Z2', 'Smartphone di ultima generazione', 800.00, 50,
'smartphone_z2.png', 'Smartphone'),
```

```
('Tablet A3', 'Tablet per professionisti', 600.00, 30, 'tablet_a3.png',
'Tablet');
```

2. Struttura MVC

a. Controller

Crea un controller `ProductController.php` per gestire le operazioni CRUD (Create, Read, Update, Delete) sui prodotti. Il controller includerà metodi per visualizzare tutti i prodotti, creare nuovi prodotti, modificarli ed eliminarli.

b. Model

Crea il modello `Product.php` che rappresenterà la tabella `products` del database. Il modello dovrà gestire le interazioni con il database, come l'inserimento e il recupero di dati, tramite **PDO**.

c. View

Crea le viste `index.php` e `create.php` nella cartella `/views/products` per visualizzare rispettivamente la lista dei prodotti e un form per l'aggiunta di nuovi prodotti. Le viste utilizzeranno le variabili e i dati passati dal controller.

3. Login e Sessioni

Implementa un sistema di autenticazione degli utenti utilizzando **sessioni** PHP. Gli utenti potranno accedere al sistema per gestire i prodotti. Crea un controller `UserController.php` che gestirà il login e il logout. Usa funzioni come `password_hash()` per memorizzare le password in modo sicuro e `password_verify()` per il login.

4. Upload Immagini dei Prodotti

Aggiungi una funzionalità di **upload immagini** per i prodotti. Le immagini saranno caricate nella cartella `/public/uploads/products`. Nel controller `ProductController.php`, implementa la logica di caricamento del file.

```
// Esempio di funzione di caricamento immagini
public function uploadImage($file) {
    $target_dir = "uploads/products/";
    $target_file = $target_dir . basename($file["name"]);
    move_uploaded_file($file["tmp_name"], $target_file);
    return $target_file;
}
```

5. Connessione al Database con PDO

Crea un file `database.php` in `/config` per la connessione al database. Utilizza **PDO** per gestire tutte le operazioni con il database MySQL.

```
class Database {
    private $host = "localhost";
    private $db_name = "tech_warehouse";
    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 "Connection error: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
```

6. Creazione di API REST

Crea un'API REST che permetta di interagire con i prodotti tramite richieste **GET**, **POST**, **PUT** e **DELETE** in formato **JSON**. Ad esempio, il file `products.php` in `/api` gestirà le operazioni relative ai prodotti.

```
header("Content-Type: application/json");

require_once '../config/database.php';
require_once '../models/Product.php';

$database = new Database();
$db = $database->getConnection();

$product = new Product($db);

// Esempio di gestione delle richieste GET per recuperare tutti i prodotti
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    $products = $product->getAll();
    echo json_encode($products);
}
```

7. Organizzazione dei Namespace

Organizza il codice utilizzando **namespace** per modularizzare i componenti. Ad esempio, utilizza il seguente namespace nel modello `Product.php`:

```
namespace App\Models;

class Product {
    // Implementazione del modello Product
}
```

8. Rotte

Configura le rotte nel file `routes.php` per definire le URL dell'applicazione e associarle ai controller corrispondenti.

Esempio di Logica CRUD per i Prodotti

- **Creazione di un prodotto**: Implementa un form nella vista `create.php` per aggiungere nuovi prodotti. Il form dovrà inviare i dati a `ProductController.php` per salvarli nel database.
- **Visualizzazione dei prodotti**: Nella vista `index.php`, elenca tutti i prodotti presenti nel magazzino. I dati verranno recuperati dal controller attraverso il modello `Product.php`.

Conclusione

Concludendo questa esercitazione, avrai creato un sistema di gestione di un magazzino di prodotti tecnologici completo, utilizzando **PHP 8** e il pattern **MVC**. La gestione dei dati sarà realizzata tramite **PDO** e MySQL, con il supporto di sessioni, login, caricamento di file e API REST per una gestione dei prodotti in formato JSON.