Rozumím! Pojďme na to, PHP je mocný jazyk pro tvorbu dynamických webových aplikací a databázových aplikací. Udělám to podrobně, pokryju všechny základní koncepty, ukázky kódu a postupy, které by ti mohly pomoci pochopit, jak PHP funguje. Takže se těším, že to půjde do detailu a pokryjeme co nejvíc.

**Úvod do PHP**

PHP (Hypertext Preprocessor) je serverový skriptovací jazyk používaný především pro vytváření dynamických webových stránek a aplikací. PHP kód je vykonáván na serveru a výsledkem může být HTML kód, který se posílá zpět do prohlížeče.

**Struktura PHP kódu:**

* PHP kód je uzavřen do tagů <?php ... ?>.
* Výstup z PHP je často HTML, CSS nebo JavaScript.

**Příklad základního PHP kódu:**

<?php

echo "Hello, World!";

?>

* echo je příkaz pro výstup, který ti umožňuje zobrazit text nebo hodnoty na stránce.

**Základní proměnné a datové typy**

PHP je **dynamicky typovaný jazyk**, což znamená, že nemusíš deklarovat typy proměnných předem. PHP automaticky určí typ podle hodnoty, kterou přiřadíš.

**Proměnné v PHP**

* Proměnná začíná znakem $ následovaným názvem proměnné.
* Proměnné jsou **case-sensitive**, takže $var a $Var jsou dvě různé proměnné.

<?php

$moje\_jmeno = "Anna"; // Řetězec

$vek = 25; // Celé číslo

$cislo = 3.14; // Desetinné číslo (float)

$is\_adult = true; // Boolean (pravda/nepravda)

?>

**Datové typy:**

* **String**: Textová data. Například "Ahoj světe!".
* **Integer**: Celá čísla. Například 42.
* **Float**: Desetinná čísla. Například 3.14.
* **Boolean**: Pravda nebo nepravda. Například true nebo false.
* **Array**: Pole. Seznam hodnot. Například array(1, 2, 3).
* **Object**: Objekt (pro práci s objekty v objektově orientovaném programování).
* **NULL**: Speciální datový typ pro "prázdnou" hodnotu.

**Operátory**

V PHP můžeš používat různé **operátory**, které ti umožňují provádět různé operace s daty.

**Aritmetické operátory:**

<?php

$a = 10;

$b = 5;

echo $a + $b; // Sčítání (výstup: 15)

echo $a - $b; // Odečítání (výstup: 5)

echo $a \* $b; // Násobení (výstup: 50)

echo $a / $b; // Dělení (výstup: 2)

echo $a % $b; // Zbytek po dělení (výstup: 0)

?>

**Porovnávací operátory:**

<?php

$a = 10;

$b = 5;

echo ($a == $b); // Porovnání na rovnost

echo ($a !== $b); // Porovnání na nerovnost (typ + hodnota)

echo ($a > $b); // Větší než

echo ($a < $b); // Menší než

?>

**Logické operátory:**

<?php

$a = true;

$b = false;

echo ($a && $b); // AND (logické "a")

echo ($a || $b); // OR (logické "nebo")

echo (!$a); // NOT (logické "ne")

?>

**Řídící struktury**

**Podmínky - if, else, elseif**

<?php

$vek = 18;

if ($vek >= 18) {

echo "Jsi dospělý.";

} elseif ($vek > 12) {

echo "Jsi teenager.";

} else {

echo "Jsi dítě.";

}

?>

* **if**: Pokud je podmínka pravdivá, vykoná se příslušný kód.
* **elseif**: Pokud je první podmínka nepravdivá, zkontroluje další podmínku.
* **else**: Pokud všechny podmínky selžou, vykoná se tento blok.

**Switch-case**

Pokud máš více podmínek, může být efektivnější použít switch místo několika if/elseif.

<?php

$den = "Pondělí";

switch ($den) {

case "Pondělí":

echo "Začíná týden!";

break;

case "Sobota":

echo "Konečně víkend!";

break;

default:

echo "Jsem zmatený, co je dnes za den?";

break;

}

?>

**Cykly**

**for smyčka**

<?php

for ($i = 1; $i <= 5; $i++) {

echo "Číslo: $i<br>";

}

?>

**while smyčka**

<?php

$i = 1;

while ($i <= 5) {

echo "Číslo: $i<br>";

$i++;

}

?>

**foreach smyčka (pro pole)**

<?php

$ovocny\_sladek = array("jablko", "banán", "hruška");

foreach ($ovocny\_sladek as $ovoce) {

echo "$ovoce<br>";

}

?>

* **foreach** se používá k iterování přes pole a pracuje s každým prvkem pole.

**Funkce**

Funkce v PHP ti umožní zabalit kód do opakovatelných bloků.

**Definice funkce**

<?php

function pozdrav($jmeno) {

echo "Ahoj, $jmeno!";

}

pozdrav("Anna"); // Výstup: Ahoj, Anna!

?>

* **function** slouží k definici nové funkce.
* Parametry se přebírají do funkce a mohou být použity uvnitř těla funkce.

**Funkce s návratovou hodnotou**

<?php

function soucet($a, $b) {

return $a + $b;

}

$result = soucet(10, 5);

echo $result; // Výstup: 15

?>

**Práce s formuláři a POST/GET metodami**

**Základní HTML formulář s PHP**

Formuláře mohou být použity pro získání uživatelského vstupu a odeslání dat na server.

<form action="formular.php" method="POST">

Jméno: <input type="text" name="jmeno">

<input type="submit" value="Odeslat">

</form>

**Zpracování formuláře v PHP**

<?php

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$jmeno = $\_POST['jmeno'];

echo "Ahoj, $jmeno!";

}

?>

* **$\_POST**: Superglobální proměnná, která obsahuje data odeslaná metodou POST.
* **$\_GET**: Superglobální proměnná, která obsahuje data odeslaná metodou GET (při odesílání dat v URL).

**Práce s databázemi (MySQL)**

PHP je silně spojeno s databázemi, zejména MySQL. Pro práci s databázemi se používají knihovny jako **MySQLi** nebo **PDO**.

**Připojení k databázi (MySQLi)**

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "mojedb";

// Vytvoření připojení

$conn = new mysqli($servername, $username, $password, $dbname);

// Kontrola připojení

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

echo "Připojeno úspěšně";

?>

**Dotazování do databáze**

<?php

$sql = "SELECT id, jmeno FROM uzivatele";

$result = $conn->query($sql);

if ($result->num\_rows > 0) {

// Výstup každého řádku

while($row = $result->fetch\_assoc()) {

echo "id: " . $row["id"]. " - Jméno: " . $row["jmeno"]. "<br>";

}

} else {

echo "0 výsledků";

}

$conn->close();

?>

**Závěr**

PHP je nesmírně silný jazyk pro webové aplikace a jeho schopnost pracovat s databázemi a formuláři je jednou z hlavních výhod. Pokrýváme základní syntaxi, práce s proměnnými, podmínkami, cykly, funkcemi a také interakci s databázemi. Tímto způsobem budeš schopná se ponořit do jakéhokoliv PHP úkolu.

Skvěle, pokračujme dál! Teď se podíváme na některé pokročilejší funkce PHP, jako jsou pokročilé techniky pro práci s databázemi, zabezpečení, a také práci s objekty (OOP) a třídami. Zaměříme se i na formátování a validaci vstupu, což je klíčové pro bezpečnost aplikace.

**Pokročilé práce s databázemi a MySQL**

**Vkládání dat do databáze**

Pro vkládání dat do databáze použijeme **prepared statements** (předpřipravené dotazy). Tyto dotazy jsou bezpečné proti SQL injekcím.

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "mojedb";

// Vytvoření připojení

$conn = new mysqli($servername, $username, $password, $dbname);

// Zkontroluj připojení

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

// Připravíme dotaz

$stmt = $conn->prepare("INSERT INTO uzivatele (jmeno, email) VALUES (?, ?)");

$stmt->bind\_param("ss", $jmeno, $email);

// Parametry pro dotaz

$jmeno = "Anna";

$email = "anna@example.com";

// Spuštění dotazu

$stmt->execute();

echo "Nový záznam vytvořen";

// Uzavření připojení

$stmt->close();

$conn->close();

?>

* **prepare** připraví dotaz, ale neprovádí ho hned.
* **bind\_param** připojí proměnné k parametrům dotazu, kde první argumenty určují typ (např. "ss" pro dva řetězce).

**Aktualizace dat v databázi**

Pokud chceme aktualizovat data, opět použijeme **prepared statement**:

<?php

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

$stmt = $conn->prepare("UPDATE uzivatele SET jmeno = ?, email = ? WHERE id = ?");

$stmt->bind\_param("ssi", $jmeno, $email, $id);

$jmeno = "Anna Nováková";

$email = "anna.novakova@example.com";

$id = 1; // ID uživatele, kterého chceme aktualizovat

$stmt->execute();

echo "Záznam aktualizován";

$stmt->close();

$conn->close();

?>

**Mazání dat v databázi**

Pokud chceš odstranit data, můžeš použít DELETE dotaz:

<?php

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

$stmt = $conn->prepare("DELETE FROM uzivatele WHERE id = ?");

$stmt->bind\_param("i", $id);

$id = 1; // ID uživatele, kterého chceme odstranit

$stmt->execute();

echo "Záznam smazán";

$stmt->close();

$conn->close();

?>

**Bezpečnost v PHP (Ochrana proti SQL injekcím a XSS)**

Bezpečnost je klíčová, aby se chránily aplikace před různými typy útoků. Podíváme se na ochranu proti **SQL injekcím** a **Cross-Site Scripting (XSS)**.

**Ochrana proti SQL injekcím**

Jak již bylo zmíněno, pro bezpečné připojení k databázi a práci s daty používáme **prepared statements**. Předpřipravené dotazy se postarají o správné ošetření dat, což chrání před SQL injekcemi.

**Ochrana proti XSS (Cross-Site Scripting)**

XSS útoky jsou útoky, kdy útočník může vkládat skripty do webových stránek, které mohou následně spustit škodlivý JavaScript.

Proti XSS se chráníme pomocí **funkcí pro escapování výstupů**.

<?php

$user\_input = "<script>alert('XSS');</script>";

// Použijeme htmlspecialchars k ochraně před XSS

echo htmlspecialchars($user\_input);

?>

* **htmlspecialchars()**: Tato funkce převádí speciální znaky, jako < a >, na HTML entity (&lt; a &gt;), čímž zabraňuje vykonání skriptů v prohlížeči.

**Validace a sanitace uživatelských vstupů**

Je důležité vždy validovat a sanitovat vstupy od uživatele. PHP poskytuje různé funkce pro kontrolu, zda vstupy odpovídají požadovanému formátu.

* **filter\_var()** pro validaci emailu nebo URL:

<?php

$email = "anna@example.com";

if (filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

echo "Email je platný.";

} else {

echo "Email není platný.";

}

?>

* **Sanitace pro odstranění nebezpečných znaků:**

<?php

$user\_input = "<b>Hello!</b>";

$sanitized\_input = filter\_var($user\_input, FILTER\_SANITIZE\_STRING);

echo $sanitized\_input; // Výstup: Hello!

?>

**Objektově orientované programování (OOP) v PHP**

OOP je velmi užitečné pro organizaci a strukturování kódu. V PHP můžeš definovat **třídy** a **objekty** a používat principy jako dědičnost, zapouzdření a polymorfismus.

**Třídy a objekty**

Třídu v PHP definuješ pomocí klíčového slova **class**:

<?php

class Osoba {

public $jmeno;

public $vek;

function \_\_construct($jmeno, $vek) {

$this->jmeno = $jmeno;

$this->vek = $vek;

}

function pozdrav() {

return "Ahoj, jmenuji se $this->jmeno a je mi $this->vek let.";

}

}

// Vytvoření objektu

$osoba1 = new Osoba("Anna", 25);

echo $osoba1->pozdrav();

?>

* **\_\_construct()** je speciální metoda, která se automaticky zavolá při vytvoření objektu.
* **public** znamená, že vlastnosti a metody jsou přístupné zvenčí. Můžeš použít i **private** nebo **protected** pro kontrolu přístupu.

**Dědičnost**

V PHP můžeš vytvářet podtřídy, které dědí vlastnosti a metody nadtřídy.

<?php

class Student extends Osoba {

public $skola;

function \_\_construct($jmeno, $vek, $skola) {

parent::\_\_construct($jmeno, $vek);

$this->skola = $skola;

}

function pozdrav() {

return parent::pozdrav() . " Studuji na $this->skola.";

}

}

$student = new Student("Petr", 22, "Univerzita");

echo $student->pozdrav();

?>

* **parent::\_\_construct()** volá konstruktor nadtřídy.

**Polymorfismus**

Polymorfismus umožňuje objektům různých tříd reagovat na stejný způsob.

<?php

class Zviratko {

public function zvuk() {

return "Zvíře dělá zvuk.";

}

}

class Pes extends Zviratko {

public function zvuk() {

return "Haf!";

}

}

class Kocka extends Zviratko {

public function zvuk() {

return "Mňau!";

}

}

$pes = new Pes();

echo $pes->zvuk(); // Výstup: Haf!

$kocka = new Kocka();

echo $kocka->zvuk(); // Výstup: Mňau!

?>

**Závěr**

S těmito technikami a koncepty jsi vybavena solidními základy pro práci s PHP. Pokryly jsme dynamickou práci s databázemi, bezpečnostní postupy a základy objektově orientovaného programování

Skvěle! Tady máš několik užitečných tipů a triků pro práci s PHP, které ti mohou výrazně usnadnit vývoj a zlepšit kód. Pojďme na to:

**1. Krátký zápis pro podmínky a výrazy**

Místo psaní dlouhých podmínek můžeš použít **ternární operátor** (krátký zápis podmínky), což je zjednodušší kód.

<?php

$vek = 18;

echo $vek >= 18 ? "Plnoletý" : "Neplnoletý";

// Výstup: Plnoletý

?>

Tento zápis znamená:

(je-li $vek >= 18) ? "Plnoletý" : "Neplnoletý";

**2. Krátký zápis pro výchozí hodnoty (Null coalescing operator)**

Pokud chceš zkontrolovat, zda nějaká hodnota není null a použít výchozí hodnotu, použij **null coalescing operator (??)**.

<?php

$uzivatel = null;

echo $uzivatel ?? "Není přihlášen"; // Výstup: Není přihlášen

?>

To je ekvivalentní následujícímu zápisu:

echo isset($uzivatel) ? $uzivatel : "Není přihlášen";

**3. Použití implode() a explode()**

**implode()** slouží pro spojení prvků pole do jednoho řetězce, zatímco **explode()** pro rozdělení řetězce na pole.

<?php

// Spojení pole do řetězce

$pole = ['Anna', 'Petr', 'Jana'];

$seznam = implode(", ", $pole);

echo $seznam; // Výstup: Anna, Petr, Jana

// Rozdělení řetězce na pole

$text = "Anna, Petr, Jana";

$pole2 = explode(", ", $text);

print\_r($pole2); // Výstup: Array ( [0] => Anna [1] => Petr [2] => Jana )

?>

**4. Použití array\_map(), array\_filter() a array\_reduce() pro práci s poli**

Toto jsou velmi silné funkce pro práci s poli.

* **array\_map()**: Aplikuje funkci na každý prvek pole.

<?php

$pole = [1, 2, 3, 4];

$novepole = array\_map(fn($x) => $x \* 2, $pole);

print\_r($novepole); // Výstup: Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 )

?>

* **array\_filter()**: Filtruje pole podle podmínky.

<?php

$pole = [1, 2, 3, 4, 5];

$parne = array\_filter($pole, fn($x) => $x % 2 == 0);

print\_r($parne); // Výstup: Array ( [1] => 2 [3] => 4 )

?>

* **array\_reduce()**: Redukuje pole na jednu hodnotu.

<?php

$pole = [1, 2, 3, 4];

$soucet = array\_reduce($pole, fn($carry, $item) => $carry + $item, 0);

echo $soucet; // Výstup: 10

?>

**5. Použití filter\_var() pro validaci a sanitaci vstupů**

Pro validaci a sanitaci vstupů od uživatele je důležité použít funkci **filter\_var()**, která má mnoho filtrů.

* **Validace e-mailu:**

<?php

$email = "anna@example.com";

if (filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

echo "Platný e-mail";

} else {

echo "Neplatný e-mail";

}

?>

* **Sanitace URL:**

<?php

$url = "http://example.com";

$sanitized\_url = filter\_var($url, FILTER\_SANITIZE\_URL);

echo $sanitized\_url; // Výstup: http://example.com

?>

**6. Generování náhodných čísel**

Pro generování náhodných čísel můžeš použít funkci **rand()** nebo **mt\_rand()** (rychlejší a lepší pro rozsáhlé aplikace).

<?php

echo rand(1, 100); // Náhodné číslo mezi 1 a 100

// Lepší alternativa pro generování náhodných čísel

echo mt\_rand(1, 100); // Rychlejší a spolehlivější

?>

**7. Zjednodušený zápis pro funkce s proměnnými argumenty**

Pokud chceš vytvořit funkci, která přijímá proměnný počet argumentů, můžeš použít **func\_get\_args()**.

<?php

function soucet() {

$args = func\_get\_args();

return array\_sum($args);

}

echo soucet(1, 2, 3, 4); // Výstup: 10

?>

**8. Konec souboru a práce s \_\_DIR\_\_**

Pokud chceš získat cestu k aktuálnímu adresáři, můžeš použít **\_\_DIR\_\_**, což ti dá naprostou cestu k adresáři souboru.

<?php

echo \_\_DIR\_\_; // Výstup: C:\xampp\htdocs\projekt

?>

Takto můžeš bezpečně zahrnout soubory relativně, nezávisle na tom, odkud aplikaci spouštíš.

**9. Použití isset() pro kontrolu existence proměnné**

Funkce **isset()** slouží pro kontrolu, zda proměnná existuje a není null.

<?php

$promenna = "Test";

if (isset($promenna)) {

echo "Proměnná je nastavena.";

} else {

echo "Proměnná není nastavena.";

}

?>

**10. Práce s regulárními výrazy v PHP**

PHP má funkce pro práci s regulárními výrazy, které jsou skvělé pro validaci a manipulaci s textem.

* **Kontrola formátu e-mailu:**

<?php

$email = "anna@example.com";

if (preg\_match("/^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/", $email)) {

echo "E-mail je platný.";

} else {

echo "E-mail není platný.";

}

?>

* **Vyhledávání a nahrazení textu:**

<?php

$text = "Mám rád PHP a HTML.";

$novy\_text = preg\_replace("/PHP/", "JavaScript", $text);

echo $novy\_text; // Výstup: Mám rád JavaScript a HTML.

?>

**11. Práce se soubory (file\_get\_contents, file\_put\_contents)**

Pokud chceš číst nebo zapisovat soubory, můžeš použít funkce **file\_get\_contents()** a **file\_put\_contents()**.

* **Čtení souboru:**

<?php

$data = file\_get\_contents("soubor.txt");

echo $data;

?>

* **Zápis do souboru:**

<?php

file\_put\_contents("soubor.txt", "Toto je nový text.");

?>

**12. Generování výstupu do prohlížeče**

Můžeš snadno generovat **dynamické obrázky** nebo jiné formáty pomocí hlaviček.

<?php

header("Content-type: image/png");

$image = imagecreatetruecolor(100, 100);

$barva = imagecolorallocate($image, 255, 0, 0);

imagefilledrectangle($image, 0, 0, 100, 100, $barva);

imagepng($image);

imagedestroy($image);

?>

Tento kód vytvoří červený čtverec.

**🧠 Co je PHP?**

* PHP je **skriptovací jazyk**, který běží na serveru (na rozdíl od JavaScriptu, který běží v prohlížeči).
* Používá se hlavně k **generování webových stránek dynamicky** a ke komunikaci s databází (MySQL).
* Kód PHP se píše **mezi <?php ... ?>** a spouští se na serveru.

**📦 Co potřebuješ?**

* PHP server – třeba [XAMPP](https://www.apachefriends.org/index.html)
* V něm zapnout **Apache** (web server) a **MySQL** (databáze)
* Soubory dáváš do složky htdocs

**👶 Začínáme! – První PHP soubor**

Vytvoř soubor: test.php

<?php

echo "Ahoj světe!";

?>

➡ Otevři v prohlížeči: http://localhost/test.php  
📣 Výsledek: Ahoj světe!

**📥 Získávání dat z formuláře**

<!-- uložit jako index.html -->

<form action="zpracuj.php" method="post">

Jméno: <input type="text" name="jmeno"><br>

<input type="submit" value="Odeslat">

</form>

<?php

// zpracuj.php

$jmeno = $\_POST['jmeno'];

echo "Ahoj, $jmeno!";

?>

➡ PHP načítá data přes $\_POST['jmeno']

**🗄️ Připojení k databázi**

Vytvoř si databázi maturita a tabulku:

CREATE TABLE studenti (

id INT AUTO\_INCREMENT PRIMARY KEY,

jmeno VARCHAR(100)

);

**vloz.php**

<?php

$conn = new mysqli("localhost", "root", "", "maturita");

$jmeno = $\_POST['jmeno'];

$sql = "INSERT INTO studenti (jmeno) VALUES ('$jmeno')";

$conn->query($sql);

echo "Student $jmeno byl uložen.";

?>

**📋 Výpis dat**

**vypis.php**

<?php

$conn = new mysqli("localhost", "root", "", "maturita");

$result = $conn->query("SELECT \* FROM studenti");

while ($radek = $result->fetch\_assoc()) {

echo $radek['id'] . ": " . $radek['jmeno'] . "<br>";

}

?>

**✅ Shrnutí základních PHP dovedností:**

| **Co** | **Jak se to dělá** |
| --- | --- |
| Výpis na obrazovku | echo "text"; |
| Získání dat z formuláře | $\_POST['jmeno'] |
| Připojení k DB | $conn = new mysqli(...); |
| SQL dotaz | $conn->query("SQL"); |
| Výpis výsledků | while ($radek = $result->fetch\_assoc()) { ... } |

Super, jdeme na to! Mám pro tebe úkol, který ti pomůže procvičit všechno, co jsme si zatím probrali.

**📝 Úkol: Správa knihovny**

Vytvoř aplikaci pro správu knihy v databázi. Tato aplikace bude mít 3 funkce:

1. Přidání nové knihy (název, autor, rok vydání)
2. Zobrazení seznamu knih
3. Smazání knihy podle ID

**🏗️ Kroky:**

1. **Vytvoření databáze a tabulky knih**
2. **Formulář pro přidání nové knihy**
3. **Seznam knih (zobrazení všech knih)**
4. **Smazání knihy podle ID**

**1. Vytvoř si databázi a tabulku:**

CREATE DATABASE knihovna;

USE knihovna;

CREATE TABLE knihy (

id INT AUTO\_INCREMENT PRIMARY KEY,

nazev VARCHAR(255),

autor VARCHAR(255),

rok INT

);

**2. Vytvoř formulář pro přidání knihy:**

<!-- form.php -->

<form action="pridej.php" method="post">

Název knihy: <input type="text" name="nazev"><br>

Autor knihy: <input type="text" name="autor"><br>

Rok vydání: <input type="number" name="rok"><br>

<input type="submit" value="Přidat knihu">

</form>

**3. Zpracování formuláře – pridej.php:**

Napojíš PHP na databázi a uložíš knihu.

<?php

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

$nazev = $\_POST['nazev'];

$autor = $\_POST['autor'];

$rok = $\_POST['rok'];

$sql = "INSERT INTO knihy (nazev, autor, rok) VALUES ('$nazev', '$autor', $rok)";

if ($conn->query($sql) === TRUE) {

echo "Kniha byla přidána!<br>";

} else {

echo "Chyba při přidávání knihy: " . $conn->error;

}

}

?>

<a href="vypis.php">Zobrazit seznam knih</a>

**4. Zobrazení knih – vypis.php:**

Na této stránce vypíšeš všechny knihy z databáze.

<?php

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

$sql = "SELECT \* FROM knihy";

$result = $conn->query($sql);

if ($result->num\_rows > 0) {

while($row = $result->fetch\_assoc()) {

echo "ID: " . $row["id"]. " - Název: " . $row["nazev"]. " - Autor: " . $row["autor"]. " - Rok: " . $row["rok"]. "<br>";

}

} else {

echo "Žádné knihy v databázi.";

}

$conn->close();

?>

**5. Smazání knihy – smazat.php:**

Přidej funkci pro smazání knihy podle ID.

<?php

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

if (isset($\_GET['id'])) {

$id = $\_GET['id'];

$sql = "DELETE FROM knihy WHERE id=$id";

if ($conn->query($sql) === TRUE) {

echo "Kniha byla smazána!<br>";

} else {

echo "Chyba při mazání knihy: " . $conn->error;

}

}

?>

<a href="vypis.php">Zobrazit seznam knih</a>

**🧐 Co teď?**

1. Zkus si vytvořit všechny části – databázi, tabulku, formulář, zpracování a zobrazení knih.
2. Pokud něco nebude jasné nebo narazíš na problém, napiš a společně to napravíme.

Jasně, pojďme to rozebrat krok po kroku ještě podrobněji!

**1. Vytvoření databáze a tabulky**

Nejdříve musíme v MySQL (nebo MariaDB) vytvořit databázi, která bude uchovávat informace o knihách.

1. **Vytvoření databáze**:
2. CREATE DATABASE knihovna;

Tohle vytvoří novou databázi s názvem knihovna.

1. **Přepneme se do databáze**:
2. USE knihovna;

Tím říkáme MySQL, že chceme pracovat s touto databází.

1. **Vytvoření tabulky pro knihy**:
2. CREATE TABLE knihy (
3. id INT AUTO\_INCREMENT PRIMARY KEY,
4. nazev VARCHAR(255),
5. autor VARCHAR(255),
6. rok INT
7. );

Tato tabulka bude mít 4 sloupce:

* + **id** – jedinečný identifikátor knihy (bude se automaticky zvyšovat)
  + **nazev** – název knihy (textový sloupec)
  + **autor** – autor knihy (textový sloupec)
  + **rok** – rok vydání knihy (číselný sloupec)

**2. Formulář pro přidání knihy**

Chceme vytvořit formulář, kde uživatel zadá název knihy, autora a rok vydání, a my tyto informace uložíme do databáze. Tento formulář bude ve **formátu HTML**.

<!-- form.php -->

<form action="pridej.php" method="post">

Název knihy: <input type="text" name="nazev"><br>

Autor knihy: <input type="text" name="autor"><br>

Rok vydání: <input type="number" name="rok"><br>

<input type="submit" value="Přidat knihu">

</form>

* **<form action="pridej.php" method="post">** znamená, že formulář po odeslání zavolá soubor pridej.php a použije metodu POST pro odeslání dat.
* **<input type="text" name="nazev">** je pole pro název knihy.
* **<input type="number" name="rok">** je pole pro rok vydání.

**3. Zpracování formuláře v PHP**

Teď musíme napsat PHP kód, který přijme data z formuláře a uloží je do databáze.

**pridej.php**

<?php

// Připojení k databázi

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

// Zpracování formuláře

if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {

$nazev = $\_POST['nazev']; // získání názvu knihy z formuláře

$autor = $\_POST['autor']; // získání autora knihy z formuláře

$rok = $\_POST['rok']; // získání roku vydání z formuláře

// SQL dotaz pro vložení dat do databáze

$sql = "INSERT INTO knihy (nazev, autor, rok) VALUES ('$nazev', '$autor', $rok)";

if ($conn->query($sql) === TRUE) {

echo "Kniha byla přidána!<br>";

} else {

echo "Chyba při přidávání knihy: " . $conn->error;

}

}

$conn->close();

?>

Co se zde děje:

* **Připojení k databázi** – použijeme new mysqli() pro připojení k databázi. Pokud se připojení nezdaří, vypíše se chybová zpráva.
* **Kontrola metody POST** – pomocí $\_SERVER['REQUEST\_METHOD'] zjistíme, že formulář byl odeslán metodou POST.
* **Získání hodnot z formuláře** – použijeme $\_POST['nazev'], $\_POST['autor'], a $\_POST['rok'], abychom získali data, která uživatel zadal.
* **SQL dotaz** – vytvoříme SQL dotaz, který vloží tyto hodnoty do databáze. Pokud je dotaz úspěšný, zobrazí se zpráva o úspěchu, jinak chybová zpráva.

**4. Zobrazení seznamu knih**

Nyní chceme zobrazit seznam všech knih, které jsme uložili do databáze.

**vypis.php**

<?php

// Připojení k databázi

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

// SQL dotaz pro získání všech knih

$sql = "SELECT \* FROM knihy";

$result = $conn->query($sql);

// Výpis všech knih

if ($result->num\_rows > 0) {

while($row = $result->fetch\_assoc()) {

echo "ID: " . $row["id"]. " - Název: " . $row["nazev"]. " - Autor: " . $row["autor"]. " - Rok: " . $row["rok"]. "<br>";

}

} else {

echo "Žádné knihy v databázi.";

}

$conn->close();

?>

Co se zde děje:

* **SQL dotaz pro získání knih** – použijeme SQL dotaz SELECT \* FROM knihy, který nám vrátí všechny knihy v tabulce.
* **Výpis výsledků** – pomocí while ($row = $result->fetch\_assoc()) projdeme všechny řádky a zobrazíme informace o každé knize.

**5. Smazání knihy**

Přidáme možnost smazat knihu podle jejího ID.

**smazat.php**

<?php

// Připojení k databázi

$conn = new mysqli("localhost", "root", "", "knihovna");

if ($conn->connect\_error) {

die("Chyba připojení: " . $conn->connect\_error);

}

if (isset($\_GET['id'])) {

$id = $\_GET['id']; // Získáme ID knihy, kterou chceme smazat

// SQL dotaz pro smazání knihy

$sql = "DELETE FROM knihy WHERE id=$id";

if ($conn->query($sql) === TRUE) {

echo "Kniha byla smazána!<br>";

} else {

echo "Chyba při mazání knihy: " . $conn->error;

}

}

$conn->close();

?>

Tento soubor použije parametr id z URL (např. smazat.php?id=1) a smaže knihu s tímto ID.

**6. Závěr**

Takto máš základní aplikaci pro správu knihy v PHP. Můžeš:

* Přidávat knihy přes formulář.
* Zobrazit seznam knih.
* Smazat knihy podle jejich ID.