2024-2025

Ruzsinszki Zita

BACKEND - PHP

13-ik évfolyam

Tartalom

[1. Bevezetés a PHP-be 5](#_Toc179455672)

[1.1 Mi az a PHP? 5](#_Toc179455673)

[1.2 A PHP Fejlődése 5](#_Toc179455674)

[1.3 Népszerűség és Használat 5](#_Toc179455675)

[1.4 Előnyök és Hátrányok 6](#_Toc179455676)

[1.4.1 Előnyök 6](#_Toc179455677)

[1.4.2 Hátrányok 6](#_Toc179455678)

[1.5 Első PHP Szkript 6](#_Toc179455679)

[1.5.1 Echo és Print 6](#_Toc179455680)

[1.5.2 Fájl létrehozása és futtatása 7](#_Toc179455681)

[1.6 Környezet Beállítása 7](#_Toc179455682)

[1.6.1 PHP Konfigurációs Fájl 7](#_Toc179455683)

[1.6.2 Hibakezelés 7](#_Toc179455684)

[1.7 Összefoglalás 7](#_Toc179455685)

[2. PHP Szintaxis és Változók 8](#_Toc179455686)

[2.1 PHP Szintaxis 8](#_Toc179455687)

[2.1.1 Alapvető Szintaxis 8](#_Toc179455688)

[2.1.2 Példa Kód 8](#_Toc179455689)

[2.2 Változók 8](#_Toc179455690)

[2.2.1 Változók Definiálása és Használata 8](#_Toc179455691)

[2.2.2 Példa Kódok 8](#_Toc179455692)

[2.3 Adattípusok 9](#_Toc179455693)

[2.3.1 Integer és Float 9](#_Toc179455694)

[2.3.2 Stringek 9](#_Toc179455695)

[2.3.3 Boolean 9](#_Toc179455696)

[2.3.4 Array 10](#_Toc179455697)

[2.4 Konstansok 10](#_Toc179455698)

[2.5 Operátorok 13](#_Toc179455699)

[2.5.1 Aritmetikai Operátorok 13](#_Toc179455700)

[2.5.2 Összehasonlító Operátorok 13](#_Toc179455701)

[2.5.3 Logikai Operátorok 13](#_Toc179455702)

[2.6 Feladatok 14](#_Toc179455703)

[3. Alapvető vezérlési szerkezetek: If-else, switch-case, logikai operátorok 16](#_Toc179455704)

[3.1 If-else 16](#_Toc179455705)

[3.1.1. Ternáris Operátor Szerkezete 17](#_Toc179455706)

[3.2 Switch-case 18](#_Toc179455707)

[3.3 Logikai Operátorok 18](#_Toc179455708)

[Önállóan Megoldandó Feladatok 19](#_Toc179455709)

[4. Ciklusok PHP-ben: For, While, Do-While, Foreach Ciklusok 26](#_Toc179455710)

[4.1 Elöl- és hátultesztelő ciklusok 26](#_Toc179455711)

[4.1.1 Elöltesztelő ciklus: while 26](#_Toc179455712)

[4.1.2 Hátultesztelő ciklus: do-while 26](#_Toc179455713)

[4.2 for Ciklus 27](#_Toc179455714)

[4.3 foreach Ciklus 27](#_Toc179455715)

[4.4 for...of Ciklus PHP-ben 27](#_Toc179455716)

[4.5 Feladatok 28](#_Toc179455717)

[5. Függvények Definiálása és Használata: Függvények, Paraméterek, Visszatérési Értékek 29](#_Toc179455718)

[5.1 Függvények Definiálása 29](#_Toc179455719)

[5.2 Függvények Meghívása 30](#_Toc179455720)

[5.3 Paraméterek a Függvényekben 30](#_Toc179455721)

[5.4 Visszatérési Értékek 30](#_Toc179455722)

[5.5 Alapértelmezett Értékek 31](#_Toc179455723)

[5.6 Több Visszatérési Érték 31](#_Toc179455724)

[5.7 Feladatok 32](#_Toc179455725)

[6. PHP Szuperglobális Változók 34](#_Toc179455726)

[6.1 $\_GET 34](#_Toc179455727)

[6.2 $\_POST 36](#_Toc179455728)

[6.3 $\_SESSION 38](#_Toc179455729)

[6.4 $\_COOKIE 40](#_Toc179455730)

[6.5 $\_REQUEST 42](#_Toc179455731)

[6.6 $\_FILES 44](#_Toc179455732)

[6.7 $\_SERVER 47](#_Toc179455733)

[7. Formuláris adatfeldolgozás: Formok kezelése, input validáció. 49](#_Toc179455734)

[7.1. Adatok feldolgozása: 50](#_Toc179455735)

[7.2. Input Validáció 50](#_Toc179455736)

[7.3. Formok biztonsága 51](#_Toc179455737)

[7.4. SQL Injection elkerülése: 51](#_Toc179455738)

[7.5. Összetett űrlap feldolgozási feladat: 52](#_Toc179455739)

[7.6. Implode() függvény használata 52](#_Toc179455740)

[8. Érzékeny Adatok Biztonságos Kezelése .env Fájlban 65](#_Toc179455741)

[9. Adatbázis műveletek a PHP-ben összefoglalás 68](#_Toc179455742)

[10. PHP adatbázis műveletek továbbfejlesztése: Kérések optimalizálása, indexek használata. 70](#_Toc179455743)

[10.1. SQL kérdések optimalizálása: 70](#_Toc179455744)

[10.2. Indexek használata: 70](#_Toc179455745)

[10.3. Adatbázis normalizálása és optimalizálása: 71](#_Toc179455746)

[10.4. Caching használata: 71](#_Toc179455747)

[10.5. Query Plan elemzése: 71](#_Toc179455748)

[11. PHP adatkezelés haladó technikák: Adatbázis migrációk, készletkezelés. 72](#_Toc179455749)

[12. PHP és AJAX: Aszinkron adatküldés és fogadás PHP és JavaScript használatával. 74](#_Toc179455750)

[13. PHP fájlkezelés haladó technikák: Fájlok és könyvtárak manipulálása, jogosultságok kezelése 83](#_Toc179455751)

[14. A RESTful API alkalmazás fő követelményei 87](#_Toc179455752)

[15. Összetett feladat megoldása különböző módszerekkel: Névjegyek 91](#_Toc179455753)

[14.1. Adatbázis 91](#_Toc179455754)

[14.2. Adatbázis kapcsolat 98](#_Toc179455755)

[14.3. Read művelet megvalósítása 99](#_Toc179455756)

[14.3.1. READ művelet egyetlen fájl létrehozásával 99](#_Toc179455757)

[Gyakran használt PHPDoc annotációk összefoglalása 107](#_Toc179455758)

[Gyakorlati feladatok 107](#_Toc179455759)

[14.3.2. Végpont azaz JSON kimenet generálása 109](#_Toc179455760)

[14.3.3. Végpont beolvasása 112](#_Toc179455761)

[14.3.1. Lapozó választó, hány rekord jelenjen meg egy lapon 114](#_Toc179455762)

[14.3.2. Végpont beolvasása AngularJs használatával 117](#_Toc179455763)

[14.4. Admin felület és a CRUD műveletek 121](#_Toc179455764)

[14.5. Listázás az admin részére 121](#_Toc179455765)

[14.6. Felvitel 124](#_Toc179455766)

[14.7. Módosítás 127](#_Toc179455767)

[14.8. Törlés 131](#_Toc179455768)

[14.9. A CRUD műveletek megvalósítása a RESTful API követelményeinek megfelelően 132](#_Toc179455769)

[14.9.1. Végpont 132](#_Toc179455770)

[14.9.2. Listázás – READ 134](#_Toc179455771)

[14.9.3. Felvitel – CREATE 136](#_Toc179455772)

[Felépítés 137](#_Toc179455773)

[14.9.4. Módosítás – UPDATE 142](#_Toc179455774)

[14.9.5. Módosítás – PUT 145](#_Toc179455775)

[14.9.6. Törlés – DELETE 147](#_Toc179455776)

[14.10. Titkosítás – Lapvédelem 149](#_Toc179455777)

[14.10.1.htaccess 149](#_Toc179455778)

[14.10.2. Sütik 150](#_Toc179455779)

[14.10.3. A session 152](#_Toc179455780)

[14.10.4. Lapvédelem 153](#_Toc179455781)

[14.10.5. HASH függvények 156](#_Toc179455782)

[14.11. Az új RESTful Api alkalmazásnak megfelelő titkosítás 160](#_Toc179455783)

[14.11.1. composer.json (Telepítjük a JWT csomagot) 160](#_Toc179455784)

[14.10.2. connect.php (Már meglévő adatbázis kapcsolat) 160](#_Toc179455785)

[14.10.3. middleware.php (Token ellenőrzése) 160](#_Toc179455786)

[14.10.4. login.php (Belépés) 161](#_Toc179455787)

[14.10.5. index.php (Lapvédelem JWT használatával) 162](#_Toc179455788)

[14.10.6. htaccess (RESTful API védelme) 162](#_Toc179455789)

[14.10.7. login.html – Felhasználónév és jelszó űrlap 163](#_Toc179455790)

[14.10.8. login.php – A bejelentkezési adatok kezelése 164](#_Toc179455791)

[14.10.9. dashboard.php – Admin oldal, amely védelem alatt áll 165](#_Toc179455792)

[14.10.10. logout.php – Kijelentkezés 165](#_Toc179455793)

[Források: 168](#_Toc179455794)

# 1. Bevezetés a PHP-be

## 1.1 Mi az a PHP?

A PHP (Hypertext Preprocessor) egy népszerű szerveroldali programozási nyelv, amelyet weboldalak dinamikus tartalmának előállítására használnak. A PHP kódot a szerver végzi el, mielőtt az HTML oldal véglegesen megjelenne a felhasználó böngészőjében. A PHP használatával interaktív weboldalakat, űrlapokat, adatbázis-műveleteket és sok más dinamikus funkciót lehet létrehozni.

## 1.2 A PHP Fejlődése

A PHP eredetileg Rasmus Lerdorf által készült 1994-ben. Az első verzió, a PHP/FI (Personal Home Page Tools/FI), alapvetően egy egyszerű eszköz volt, amely lehetővé tette a weboldalak dinamikus tartalmának előállítását. Azóta a PHP jelentős fejlődésen ment keresztül, és a következő főbb verziókat tartalmazza:

* **PHP 3**: 1998-ban jelent meg, és jelentős frissítést hozott az előző verzióhoz képest, mivel a kódot teljesen újraírták, és már objektum-orientált programozási támogatást is nyújtott.
* **PHP 4**: 2000-ben jelent meg, és az új Zend Engine révén gyorsabb és hatékonyabb teljesítményt biztosított. Az objektum-orientált programozás tovább fejlődött.
* **PHP 5**: 2004-ben kiadott verzió, amely az objektum-orientált programozás jelentős fejlesztéseit tartalmazta, például az osztályokat, öröklődést, és a MySQL támogatást is jelentősen javította.
* **PHP 7**: 2015-ben jelent meg, és jelentős teljesítménybeli javulásokat hozott, valamint új funkciókat és optimalizálásokat. Az objektum-orientált programozás és a hibakezelés is tovább fejlődött.
* **PHP 8**: 2020-ban jelent meg, és további teljesítménybeli javulásokat hozott, valamint új nyelvi funkciókat, mint például a JIT (Just-In-Time) fordítót, és számos új szintaktikai elemet.

## 1.3 Népszerűség és Használat

A PHP az egyik legnépszerűbb szerveroldali programozási nyelv a webfejlesztés terén. A statisztikák szerint a weboldalak jelentős része, köztük népszerű CMS rendszerek, mint a WordPress, Drupal, és Joomla, PHP-t használ. A PHP népszerűsége az alábbiaknak köszönhető:

* **Széleskörű Támogatás**: A PHP széles körben támogatott a legtöbb webhoszting szolgáltatásnál, és könnyen integrálható különböző adatbázisokkal és technológiákkal.
* **Nagy Közösség**: A PHP közössége aktív és támogató, rengeteg dokumentáció, fórum, és könyvtár érhető el.
* **Nyílt Forráskód**: A PHP ingyenes és nyílt forráskódú, amely lehetővé teszi bárki számára a nyelv tanulmányozását és testreszabását.

## 1.4 Előnyök és Hátrányok

### 1.4.1 Előnyök

* **Könnyen Tanulható**: A PHP viszonylag könnyen tanulható, különösen kezdők számára, mivel egyszerű szintaxissal rendelkezik.
* **Széleskörű Kompatibilitás**: A PHP jól működik különböző operációs rendszereken (Windows, Linux, macOS), és könnyen integrálható más webtechnológiákkal.
* **Erős Közösségi Támogatás**: Az aktív közösség és a széleskörű dokumentáció segíti a fejlesztőket a problémák gyors megoldásában.
* **Teljesítmény**: A PHP 7 és 8 verziói jelentős teljesítménybeli javulásokat hoztak a korábbi verziókhoz képest.

### 1.4.2 Hátrányok

* **Biztonsági Kérdések**: Mivel a PHP egy nyílt forráskódú nyelv, a biztonsági problémák és sebezhetőségek kezelése kihívást jelenthet, különösen a rosszul megírt kód esetén.
* **Elavult Kódok**: A PHP elavult kódokkal rendelkező régi rendszerek továbbra is fennállnak, amelyek karbantartása és frissítése problémás lehet.
* **Teljesítmény Különbségek**: Bár az újabb verziók jelentős teljesítményjavulást hoztak, a PHP teljesítménye még mindig elmaradhat bizonyos modern nyelvekhez képest, különösen nagy forgalmú alkalmazások esetén.

## 1.5 Első PHP Szkript

A PHP fájlok .php kiterjesztéssel rendelkeznek, és a PHP kódot a <?php ... ?> címkék közé kell írni. Az alábbi példa bemutatja, hogyan lehet egy egyszerű "Hello, World!" üzenetet megjeleníteni echo és print használatával:

### 1.5.1 Echo és Print

A PHP-ben két gyakran használt függvény a echo és a print, amelyek mindkettő a szöveg kiírására szolgál, de van néhány fontos különbség közöttük.

* **echo**: Több értéket is ki tud írni egyszerre, és nem ad vissza értéket. Az echo egy nyelvi konstrukció, nem pedig egy valódi függvény, így nincs visszatérési értéke.
* **print**: Csak egy értéket tud kiírni egyszerre, és mindig visszatér egy értékkel (0-t jelent, ha sikeres a művelet).

**echo példa:**

<?php

echo "Hello, World!<br>";

echo "Welcome to PHP!";

?>

**print példa:**

<?php

print "Hello, World!<br>";

print "Welcome to PHP!";

?>

**Különbségek példával:**

* **echo használatával több paramétert adhatunk meg:**

<?php

echo "Hello, ", "World!", "<br>";

?>

* **print csak egy paramétert tud kezelni:**

<?php

print "Hello, World!" . "<br>";

?>

### 1.5.2 Fájl létrehozása és futtatása

1. **Fájl létrehozása**: Nyiss meg egy szövegszerkesztőt (pl. Notepad++ vagy VS Code), és írd be az alábbi PHP kódot. Mentsd el a fájlt hello.php néven az XAMPP telepítési könyvtárában található htdocs mappába.

<?php

echo "Hello, World!<br>";

print "Welcome to PHP!";

?>

1. **Fájl futtatása**: Nyisd meg a böngészőt, és írd be az http://localhost/hello.php címet. A böngészőben meg kell jelennie a "Hello, World!" és a "Welcome to PHP!" üzeneteknek, a echo által hozzáadott sorok közé szúrva.

## 1.6 Környezet Beállítása

### 1.6.1 PHP Konfigurációs Fájl

A PHP konfigurációs fájl neve php.ini, és az alapértelmezett helye az XAMPP telepítési könyvtárában található php mappában van. A fájl módosításával szabályozhatjuk a PHP beállításait, például a memória limitet vagy a hibakezelést.

### 1.6.2 Hibakezelés

A PHP hibakezelése fontos része a fejlesztésnek. A hibák megjelenítését a php.ini fájlban található display\_errors beállítással szabályozhatjuk. A hibaüzenetek hasznosak lehetnek a hibák keresésében és javításában.

display\_errors = On

## 1.7 Összefoglalás

Ebben a fejezetben megismerkedtünk a PHP alapjaival, a telepítés folyamatával, és az első PHP szkript elkészítésével. Részleteztük a PHP fejlődését, népszerűségét, valamint az előnyeit és hátrányait. A print és echo funkciókat is bemutattuk, és rávilágítottunk a köztük lévő különbségekre. A következő fejezetekben részletesebben foglalkozunk a PHP szintaxisával, változókkal és adattípusokkal.

# 2. PHP Szintaxis és Változók

## 2.1 PHP Szintaxis

### 2.1.1 Alapvető Szintaxis

A PHP kódot a <?php ... ?> címkék között helyezzük el, és minden utasításnak pontosvesszővel (;) kell végződnie. A kommentárok segítenek a kód dokumentálásában.

### 2.1.2 Példa Kód

<?php

// Változók definiálása

$felhasznalo = "Alice";

$kor = 25;

// Kimenet generálása

echo "Név: " . $felhasznalo . ", Kor: " . $kor . "<br>";

// Többsoros kommentár

/\*

A fenti kód kiírja a felhasználó nevét és korát.

\*/

?>

## 2.2 Változók

### 2.2.1 Változók Definiálása és Használata

PHP változók dinamikusan típusúak, ami azt jelenti, hogy a változók típusát automatikusan a hozzárendelt érték határozza meg.

### 2.2.2 Példa Kódok

* **Egész számok és lebegőpontos számok**:

<?php

$e = 100; // Egész szám

$f = 25.5; // Lebegőpontos szám

// Kiíratás

echo "Egész szám: $e<br>";

echo "Lebegőpontos szám: $f<br>";

?>

* **Stringek**:

<?php

$szoveg1 = "Hello";

$szoveg2 = 'World';

// Kiíratás

echo $szoveg1 . " " . $szoveg2 . "<br>";

?>

* **Tömbök**:

<?php

// Asszociatív tömb

$szemely = array(

"nev" => "Bob",

"kor" => 35

);

// Indexált tömb

$gyumolcsok = array("alma", "banán", "szilva");

// Kiíratás

echo "Név: " . $szemely["nev"] . "<br>";

echo "Kedvenc gyümölcsök: " . implode(", ", $gyumolcsok) . "<br>";

?>

## 2.3 Adattípusok

A PHP dinamikus típusú, ami azt jelenti, hogy a változók típusát automatikusan kezeljük az értékük alapján.

### 2.3.1 Integer és Float

<?php

$integer = 42; // Egész szám

$float = 3.14; // Lebegőpontos szám

// Kiíratás

echo "Egész szám: $integer<br>";

echo "Lebegőpontos szám: $float<br>";

?>

### 2.3.2 Stringek

A stringek lehetnek egyszerű vagy többsorosak. A PHP támogatja az escape karaktereket, például \n az új sorhoz.

<?php

$szoveg = "Ez egy \"időjárás-jelentés\".\nSzeretném bemutatni, hogyan működik.";

echo nl2br($szoveg); // nl2br() konvertálja az új sor karaktereket <br> tagekké

?>

### 2.3.3 Boolean

A Boolean típus csak két értéket vehet fel: true vagy false.

<?php

$valasz = true;

if ($valasz) {

echo "Az érték igaz.<br>";

} else {

echo "Az érték hamis.<br>";

}

?>

### 2.3.4 Array

Az tömbök lehetnek indexáltak vagy asszociatívak. Az indexált tömbök numerikus indexekkel rendelkeznek, míg az asszociatív tömbök kulcs-érték párokat tartalmaznak.

<?php

$indexelt = array("alma", "banán", "cseresznye");

$asszociativ = array("Nev" => "Kati", "Kor" => 28);

// Kiíratás indexált tömbből

foreach ($indexelt as $gyumolcs) {

echo $gyumolcs . "<br>";

}

// Kiíratás asszociatív tömbből

foreach ($asszociativ as $kulcs => $ertek) {

echo "$kulcs: $ertek<br>";

}

?>

### 2.4 Konstansok

#### 2.4.1 Konstansok Definiálása

A PHP-ben a konstansok olyan változók, amelyek értéke a definiálás után nem változtatható meg. Konstansokat a define() függvénnyel vagy a const kulcsszóval definiálhatunk.

* **define() függvény**: A define() függvény az első paraméter a konstans neve, a második pedig az értéke.

<?php

define("PI", 3.14159);

echo "A PI értéke: " . PI . "<br>";

?>

* **const kulcsszó**: A const kulcsszó a classokon belül is használható, de globálisan is alkalmazható.

<?php

const GRAVITACIO = 9.81;

echo "A gravitációs gyorsulás értéke: " . GRAVITACIO . " m/s²<br>";

?>

**Különbség a define() és const kulcsszó használata között:**

A const a fordítási időben, a define() futási időben határozza meg a változókat.

**define() függvény**:

* A define() függvényt arra használják, hogy egy állandó (konstans) értéket hozzunk létre, amely nem módosítható a program futása során.
* A konstansokat a futási időben hozza létre, és azok globálisan elérhetők az egész programban.
* Példa: define('PI', 3.14); – Ezzel a PI névhez 3.14 értéket rendelsz.

**const kulcsszó**:

* A const kulcsszóval szintén állandókat hozunk létre, de ezek jellemzően osztályokon belül használatosak.
* Ezek az állandók már a fordítás során rögzülnek, és nem módosíthatók a futás során.
* Példa: const PI = 3.14; – Ezzel egy PI nevű konstans változót deklarálsz.

A lényegi különbség tehát az, hogy a define() futási időben hoz létre konstansokat, míg a const kulcsszóval deklarált állandók már a kód fordítása során rögzülnek, és jellemzően osztályokon belül használatosak.

**define() használata**

define('MINIMUM\_KOR', 18);

$kor = 20;

if ($kor >= MINIMUM\_KOR) {

echo "Elég idős vagy.";

} else {

echo "Nem vagy elég idős.";

}

**const használata**

class KorEllenorzo {

const MINIMUM\_KOR = 18;

}

$kor = 20;

if ($kor >= KorEllenorzo::MINIMUM\_KOR) {

echo "Elég idős vagy.";

} else {

echo "Nem vagy elég idős.";

}

* Az első példában a define('MINIMUM\_KOR', 18); segítségével létrehoztunk egy globális konstansot, amelyet az if statement-ben használunk.
* A második példában a const MINIMUM\_KOR = 18; egy osztályon belül van deklarálva, és ezt az osztály nevén keresztül érjük el (KorEllenorzo::MINIMUM\_KOR) az if statement-ben.

Mindkét példa ugyanazt az eredményt adja: ha az életkor ($kor) eléri vagy meghaladja a minimális korhatárt, akkor az "Elég idős vagy." üzenetet írja ki, különben a "Nem vagy elég idős." üzenetet.

**define() helytelen használata**

define('MINIMUM\_KOR', 18);

$kor = 20;

// Nem működik, mert a konstansokat nem lehet újra definiálni

define('MINIMUM\_KOR', 21);

if ($kor >= MINIMUM\_KOR) {

echo "Elég idős vagy.";

} else {

echo "Nem vagy elég idős.";

}

**Hiba:**

* A define() használatával egyszer definiált konstansot nem lehet újra definiálni. A fenti példa hibát eredményez, mivel próbáljuk a MINIMUM\_KOR konstansot újra definiálni, ami nem megengedett.

#### const helytelen használata

class KorEllenorzo {

const MINIMUM\_KOR = 18;

}

$kor = 20;

if ($kor >= KorEllenorzo::MINIMUM\_KOR) {

echo "Elég idős vagy.";

} else {

echo "Nem vagy elég idős.";

}

// Nem működik, mert a const értékét nem lehet futás közben megváltoztatni

KorEllenorzo::MINIMUM\_KOR = 21;

**Hiba:**

* A const kulcsszóval deklarált konstansok értékét nem lehet megváltoztatni futásidőben. A fenti példa hibát eredményez, mert megpróbáljuk megváltoztatni a KorEllenorzo::MINIMUM\_KOR értékét, ami nem megengedett.

Ezek a példák megmutatják, hogy a define() és a const konstansok egyaránt egyszeri értékek, amelyeket nem lehet futásidőben módosítani vagy újra definiálni.

#### 2.4.2 Példák Konstansok Használatára

* **Konstansok használata matematikai műveletekhez**:

<?php

define("ALAP", 5);

$terulet = ALAP \* ALAP;

echo "A négyzet területe: " . $terulet . "<br>";

?>

* **Konstansok alkalmazása a program konfigurálásában**:

<?php

const MAX\_LOGIN\_ATTEMPTS = 3;

$attempts = 2;

if ($attempts >= MAX\_LOGIN\_ATTEMPTS) {

echo "Túl sok sikertelen bejelentkezési kísérlet!<br>";

} else {

echo "További próbálkozás engedélyezett.<br>";

}

?>

## 2.5 Operátorok

### 2.5.1 Aritmetikai Operátorok

* **Összeadás (+)**: Két szám összeadása.
* **Kivonás (-)**: Két szám kivonása.
* **Szorzás (\*)**: Két szám szorzása.
* **Osztás (/)**: Két szám osztása.
* **Modulus (%)**: Két szám maradékának kiszámítása.

<?php

$a = 10;

$b = 4;

echo "Összeadás: " . ($a + $b) . "<br>";

echo "Kivonás: " . ($a - $b) . "<br>";

echo "Szorzás: " . ($a \* $b) . "<br>";

echo "Osztás: " . ($a / $b) . "<br>";

echo "Modulus: " . ($a % $b) . "<br>";

?>

### 2.5.2 Összehasonlító Operátorok

* **Egyenlőség (==)**: Két érték egyenlősége.
* **Nem egyenlőség (!=)**: Két érték eltérése.
* **Nagyobb mint (>)**: Az első érték nagyobb, mint a második.
* **Kisebb mint (<)**: Az első érték kisebb, mint a második.
* **Nagyobb vagy egyenlő (>=)**: Az első érték nagyobb vagy egyenlő a második.
* **Kisebb vagy egyenlő (<=)**: Az első érték kisebb vagy egyenlő a második.

<?php

$x = 10;

$y = 20;

echo "Egyenlőség: " . ($x == $y ? "Igaz" : "Hamis") . "<br>";

echo "Nem egyenlőség: " . ($x != $y ? "Igaz" : "Hamis") . "<br>";

echo "Nagyobb mint: " . ($x > $y ? "Igaz" : "Hamis") . "<br>";

echo "Kisebb mint: " . ($x < $y ? "Igaz" : "Hamis") . "<br>";

echo "Nagyobb vagy egyenlő: " . ($x >= $y ? "Igaz" : "Hamis") . "<br>";

echo "Kisebb vagy egyenlő: " . ($x <= $y ? "Igaz" : "Hamis") . "<br>";

?>

### 2.5.3 Logikai Operátorok

* **ÉS (&&)**: Két feltétel együttes ellenőrzése.
* **VAGY (||)**: Két feltétel közül az egyik igaz.
* **NEGÁCIÓ (!)**: Feltétel tagadása.

php

Kód másolása

<?php

$a = true;

$b = false;

echo "ÉS: " . (($a && $b) ? "Igaz" : "Hamis") . "<br>";

echo "VAGY: " . (($a || $b) ? "Igaz" : "Hamis") . "<br>";

echo "NEGÁCIÓ: " . (!$a ? "Igaz" : "Hamis") . "<br>";

?>

## 2.6 Feladatok

1. **Változók és Konstansok Használata**: Írj egy PHP szkriptet, amely kiszámítja és kiírja a kör területét és kerületét egy adott sugár (r) alapján, ahol a sugár értékét egy változóban tároljuk, és a PI értékét egy konstansban.
2. **Tömbök és Kiíratás**: Készíts egy PHP szkriptet, amely létrehoz egy asszociatív tömböt, amely tartalmazza a következő kulcsokat: "Név", "Kor", "Ország". Írj egy ciklust, amely kiírja az összes kulcsot és azok értékeit a weboldalra.
3. **Összehasonlító Operátorok**: Írj egy PHP szkriptet, amely összehasonlít két számot, és kiírja, hogy az első szám nagyobb, kisebb vagy egyenlő a második számhoz képest.
4. **Logikai Operátorok**: Készíts egy PHP szkriptet, amely ellenőrzi, hogy egy adott szám páros-e, és nagyobb-e mint 10, majd írja ki az eredményt.

Megoldások:

<?php

// *1. Konstans definíciója: a PI értéke*

define("PI", 3.14159);

// *2. Függvény a kör területének kiszámításához*

function terulet($sugar) {

    return PI \* $sugar \* $sugar;

}

// *3. Függvény a kör kerületének kiszámításához*

function kerulet($sugar) {

    return 2 \* PI \* $sugar;

}

// *4. Sugár változó definiálása*

$sugar = 7; // *Például 7 egység hosszú sugár*

// *5. Terület és kerület kiszámítása a függvények segítségével*

$terulet = terulet($sugar);

$kerulet = kerulet($sugar);

// *6. Eredmények kiíratása*

echo "A kör sugara: " . $sugar . " cm<br>";

echo "A kör területe: " . $terulet . " cm2<br>";

echo "A kör kerülete: " . $kerulet . " cm<br>";

echo "--------------------------------------------------------<br>";

// *3. Terület és kerület számítása*

$terulet = PI \* $sugar \* $sugar;  // *A kör területe: PI \* r^2*

$kerulet = 2 \* PI \* $sugar;       // *A kör kerülete: 2 \* PI \* r*

// *4. Eredmények kiíratása*

echo "A kör sugara: " . $sugar . " cm<br>";

echo "A kör területe: " . $terulet . " cm2<br>";

echo "A kör kerülete: " . $kerulet . " cm<br>";

2.)

<?php

// *1. Asszociatív tömb létrehozása*

$szemely = array(

    "Nev" => "Anna",

    "Kor" => 29,

    "Orszag" => "Magyarország"

);

// *2. Ciklus az asszociatív tömb kulcsainak és értékeinek kiíratásához*

foreach ($szemely as $kulcs => $ertek) {

    echo $kulcs . ": " . $ertek . "<br>";

}

//*$kulcs tartalmazza az aktuális kulcsot,*

//*$ertek pedig az ahhoz tartozó értéket.*

3.)

<?php

// *1. Két szám definiálása*

$szam1 = 15;

$szam2 = 10;

// *2. Összehasonlító operátorok használata és eredmények kiíratása*

if ($szam1 > $szam2) {

    echo "$szam1 nagyobb, mint $szam2<br>";

} elseif ($szam1 < $szam2) {

    echo "$szam1 kisebb, mint $szam2<br>";

} else {

    echo "$szam1 egyenlő $szam2-vel<br>";

}

4.)

<?php

// *1. Adott szám definiálása*

$szam = 12;

// *2. Páros szám ellenőrzése és nagyobb mint 10 ellenőrzése*

if ($szam % 2 == 0 && $szam > 10) {

    echo "$szam páros és nagyobb mint 10<br>";

} elseif ($szam % 2 == 0) {

    echo "$szam páros, de nem nagyobb mint 10<br>";

} elseif ($szam > 10) {

    echo "$szam nem páros, de nagyobb mint 10<br>";

} else {

    echo "$szam nem páros és nem nagyobb mint 10<br>";

}

# 3. Alapvető vezérlési szerkezetek: If-else, switch-case, logikai operátorok

A PHP programozás során a vezérlési szerkezetek segítségével irányíthatjuk a program futását. Ezek a szerkezetek lehetővé teszik, hogy különböző feltételek alapján különböző kódblokkokat hajtsunk végre. A leggyakrabban használt vezérlési szerkezetek az if-else, a switch-case, valamint a logikai operátorok. Az alábbiakban részletesen bemutatjuk ezeket a szerkezeteket mintapéldákkal, valamint önállóan megoldandó feladatokat is adunk.

## 3.1 If-else

Az if-else szerkezet lehetővé teszi, hogy a program különböző ágakat hajtson végre a megadott feltételek alapján.

#### Szerkezet

if (feltétel) {

// Ha a feltétel igaz, ez a blokk végrehajtódik

} else {

// Ha a feltétel hamis, ez a blokk végrehajtódik

}

#### Példa

<?php

$szam = 8;

if ($szam % 2 == 0) {

echo "$szam páros szám.<br>";

} else {

echo "$szam páratlan szám.<br>";

}

?>

* Az if blokk ellenőrzi, hogy a $szam páros-e ($szam % 2 == 0).
* Ha a feltétel igaz, kiírja, hogy páros szám; ha nem, az else blokk kiírja, hogy páratlan szám.

A ternáris (vagy más néven conditional) operátor egy rövidített formája az if-else szerkezetnek, amelyet akkor használunk, ha csak egy egyszerű feltételt szeretnénk ellenőrizni és két értéket akarunk visszaadni attól függően, hogy a feltétel igaz vagy hamis.

### Ternáris Operátor Szerkezete

A ternáris operátor szerkezete a következő:

feltétel ? érték\_ha\_igaz : érték\_ha\_hamis;

* feltétel: Az a kifejezés, amelyet ellenőrizni akarunk.
* érték\_ha\_igaz: Az az érték, amelyet visszaadunk, ha a feltétel igaz.
* érték\_ha\_hamis: Az az érték, amelyet visszaadunk, ha a feltétel hamis.

#### 1. Egyszerű Példa

Az alábbi példában ellenőrizzük, hogy egy szám páros-e, és a ternáris operátort használjuk az eredmény kiíratásához.

<?php

$szam = 7;

// Ternáris operátor használata

$result = ($szam % 2 == 0) ? "$szam páros szám." : "$szam páratlan szám.";

echo $result . "<br>";

?>

* A feltétel: $szam % 2 == 0
* Ha a feltétel igaz, a $result változó értéke: $szam páros szám.
* Ha a feltétel hamis, a $result változó értéke: $szam páratlan szám.
* Az echo utasítással kiíratjuk az eredményt.

Példa, ahol a ternáris operátort használjuk az életkor osztályozására:

<?php

$age = 25;

// Ternáris operátor használata az életkor osztályozására

$category = ($age < 18) ? "Fiatalkorú" : (($age <= 65) ? "Felnőtt" : "Nyugdíjas");

echo "A személy kora: $age év. Kategória: $category<br>";

?>

* A feltétel: $age < 18
* Ha a feltétel igaz, a $category változó értéke: "Fiatalkorú".
* Ha a feltétel hamis, további feltételt ellenőrzünk: ($age <= 65)
* Ha az új feltétel igaz, a $category értéke: "Felnőtt".
* Ha az új feltétel is hamis, a $category értéke: "Nyugdíjas".
* Az echo utasítással kiíratjuk a személy életkorát és kategóriáját.

### 3.2 Switch-case

A switch-case szerkezet hasznos, amikor több lehetséges értéket szeretnénk ellenőrizni egy változó esetében.

#### Szerkezet

switch (kifejezés) {

case érték1:

// Ha a kifejezés értéke egyenlő az érték1-tel

break;

case érték2:

// Ha a kifejezés értéke egyenlő az érték2-tel

break;

default:

// Ha a kifejezés értéke egyik esetnek sem felel meg

}

#### Példa

<?php

$nap = "kedd";

switch ($nap) {

case "hétfő":

echo "Ma hétfő van.<br>";

break;

case "kedd":

echo "Ma kedd van.<br>";

break;

case "szerda":

echo "Ma szerda van.<br>";

break;

default:

echo "Ma nem hétfő, kedd, vagy szerda.<br>";

}

?>

* A switch szerkezet a $nap változó értékét ellenőrzi.
* Ha a $nap értéke "kedd", a megfelelő case blokk hajtódik végre, és kiírja, hogy ma kedd van.
* Ha egyik eset sem teljesül, a default blokk fut le.

### 3.3 Logikai Operátorok

A logikai operátorok segítségével több feltételt kombinálhatunk.

#### Logikai Operátorok

* **ÉS (&& vagy and)**: Két feltétel közül mindkettőnek igaznak kell lennie.
* **VAGY (|| vagy or)**: Két feltétel közül legalább az egyiknek igaznak kell lennie.
* **NEGÁCIÓ (!)**: Feltétel tagadása.

#### Példa

<?php

$szam = 15;

$kor = 18;

if ($szam > 10 && $kor >= 18) {

echo "A szám nagyobb mint 10 és a kor 18 vagy annál több.<br>";

} elseif ($szam > 10 || $kor >= 18) {

echo "A szám nagyobb mint 10 vagy a kor 18 vagy annál több.<br>";

} else {

echo "A feltételek egyike sem teljesül.<br>";

}

?>

**Magyarázat**:

* Az if blokk ellenőrzi, hogy a $szam nagyobb-e 10-nél és a $kor 18 vagy annál több-e. Ha mindkét feltétel igaz, a megfelelő üzenet jelenik meg.
* Az elseif blokk ellenőrzi, hogy legalább az egyik feltétel igaz-e ($szam > 10 vagy $kor >= 18).
* Az else blokk akkor fut le, ha egyik feltétel sem teljesül.

### Önállóan Megoldandó Feladatok

1. **Feltételes Kiírás**: Készíts egy PHP szkriptet, amely bekéri a felhasználótól az életkorát. A program írja ki, hogy a felhasználó fiatalkorú (18 év alatti), felnőtt (18 és 65 év között), vagy nyugdíjas (65 év felett).
2. **Hét Napja**: Írj egy PHP szkriptet, amely a hét napját megjeleníti (például "hétfő", "kedd", stb.), és az adott nap alapján írja ki, hogy az a hét eleje, közepe, vagy vége. Használj switch-case szerkezetet a napok kezelésére.
3. **Számellenőrzés**: Készíts egy PHP szkriptet, amely ellenőrzi, hogy egy felhasználó által megadott szám páros-e, és hogy osztható-e 5-tel. Ha mindkét feltétel teljesül, írd ki, hogy a szám páros és osztható 5-tel; ha csak az egyik, írd ki annak megfelelő üzenetet; ha egyik sem, akkor is adj vissza egy üzenetet.
4. **Komplex Logikai Feltételek**: Készíts egy PHP szkriptet, amely ellenőrzi, hogy egy felhasználó által megadott szám pozitív, és egy másik szám negatív-e, valamint hogy az első szám nagyobb-e mint 10 vagy a második szám kisebb-e mint -5. Használj &&, ||, és ! operátorokat a feltételek kombinálására, és írd ki az eredményeket.
5. **Feltételes Kiírás Ternáris Operátorral**: Készíts egy PHP szkriptet, amely ellenőrzi, hogy egy felhasználó által megadott szám nagyobb-e mint 100. Használj ternáris operátort az üzenet kiírásához, amely jelzi, hogy a szám nagyobb mint 100 vagy sem.
6. **Számkategorizálás**: Készíts egy PHP szkriptet, amely egy felhasználó által megadott szám alapján eldönti, hogy a szám negatív, nulla vagy pozitív. Használj ternáris operátort az eredmény kiírásához.
7. **Diákok Érdemjegyének Kiértékelése**: Írj egy PHP szkriptet, amely egy diák érdemjegyét (például 0-tól 100-ig) kategorizálja az alábbiak szerint: 0-49 = "Elégtelen", 50-69 = "Közepes", 70-89 = "Jó", 90-100 = "Kiváló". Használj ternáris operátort az érdemjegy kategorizálásához.
8. **Havi Költségvetés**: Készíts egy PHP szkriptet, amely ellenőrzi, hogy egy felhasználó havi költségvetése ($koltsegvetes) elegendő-e a költségek ($koltseg) fedezésére. Használj ternáris operátort az üzenet kiírásához, amely jelzi, hogy a költségvetés elegendő vagy sem.

Megoldások:

1.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Életkor Ellenőrzés</title>

</head>

<body>

    <form method="post" action="">

        <label for="age">Add meg az életkorodat:</label>

        <input type="number" id="age" name="age" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['age'])) {

        $age = (int)$\_POST['age'];

        if ($age < 18) {

            echo "Fiatalkorú (18 év alatti).<br>";

        } elseif ($age >= 18 && $age <= 65) {

            echo "Felnőtt (18 és 65 év között).<br>";

        } else {

            echo "Nyugdíjas (65 év felett).<br>";

        }

    }

    ?>

</body>

</html>

2.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Hét Napja</title>

</head>

<body>

    <form method="post" action="">

        <label for="day">Add meg a hét napját:</label>

        <input type="text" id="day" name="day" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['day'])) {

        $day = strtolower(trim($\_POST['day']));

        switch ($day) {

            case 'hétfő':

            case 'kedd':

            case 'szerda':

                echo "$day a hét eleje.<br>";

                break;

            case 'csütörtök':

            case 'péntek':

                echo "$day a hét közepe.<br>";

                break;

            case 'szombat':

            case 'vasárnap':

                echo "$day a hét vége.<br>";

                break;

            default:

                echo "Érvénytelen nap.<br>";

                break;

        }

    }

    ?>

</body>

</html>

3.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Számellenőrzés</title>

</head>

<body>

    <form method="post" action="">

        <label for="number">Add meg a számot:</label>

        <input type="number" id="number" name="number" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['number'])) {

        $number = (int)$\_POST['number'];

        $isEven = ($number % 2 == 0);

        $isDivisibleBy5 = ($number % 5 == 0);

        if ($isEven && $isDivisibleBy5) {

            echo "$number páros és osztható 5-tel.<br>";

        } elseif ($isEven) {

            echo "$number páros, de nem osztható 5-tel.<br>";

        } elseif ($isDivisibleBy5) {

            echo "$number nem páros, de osztható 5-tel.<br>";

        } else {

            echo "$number nem páros és nem osztható 5-tel.<br>";

        }

    }

    ?>

</body>

</html>

4.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Komplex Logikai Feltételek</title>

</head>

<body>

    <form method="post" action="">

        <label for="num1">Add meg az első számot:</label>

        <input type="number" id="num1" name="num1" required>

        <br>

        <label for="num2">Add meg a második számot:</label>

        <input type="number" id="num2" name="num2" required>

        <br>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['num1']) && isset($\_POST['num2'])) {

        $num1 = (int)$\_POST['num1'];

        $num2 = (int)$\_POST['num2'];

        $isPositive = ($num1 > 0);

        $isNegative = ($num2 < 0);

        $isGreaterThan10 = ($num1 > 10);

        $isLessThanMinus5 = ($num2 < -5);

        if ($isPositive && $isNegative && ($isGreaterThan10 || $isLessThanMinus5)) {

            echo "A feltételek teljesülnek: az első szám pozitív, a második szám negatív, és az első szám nagyobb mint 10 vagy a második szám kisebb mint -5.<br>";

        } else {

            echo "A feltételek egyike sem teljesül.<br>";

        }

    }

    ?>

</body>

</html>

5.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Ternáris Operátor</title>

</head>

<body>

    <form method="post" action="">

        <label for="number">Add meg a számot:</label>

        <input type="number" id="number" name="number" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['number'])) {

        $number = (int)$\_POST['number'];

        // *Ternáris operátor használata*

        $result = ($number > 100) ? "$number nagyobb mint 100." : "$number nem nagyobb mint 100.";

        echo $result . "<br>";

    }

    ?>

</body>

</html>

6.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Számkategorizálás</title>

</head>

<body>

    <form method="post" action="">

        <label for="number">Add meg a számot:</label>

        <input type="number" id="number" name="number" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['number'])) {

        $number = (int)$\_POST['number'];

        // *Ternáris operátor használata*

        $category = ($number < 0) ? "Negatív" : (($number == 0) ? "Nulla" : "Pozitív");

        echo "A szám: $number. Kategória: $category.<br>";

    }

    ?>

</body>

</html>

7.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Érdemjegy Kiértékelés</title>

</head>

<body>

    <form method="post" action="">

        <label for="grade">Add meg pontszámodat:</label>

        <input type="number" id="grade" name="grade" min="0" max="100" required>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['grade'])) {

        $grade = (int)$\_POST['grade'];

        // *Ternáris operátor használata*

        $evaluation = ($grade < 50) ? "Elégtelen" : (($grade < 70) ? "Közepes" : (($grade < 90) ? "Jó" : "Kiváló"));

        echo "Az érdemjegy: $grade. Kiértékelés: $evaluation.<br>";

    }

    ?>

</body>

</html>

8.)

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <title>Költségvetés Ellenőrzés</title>

</head>

<body>

    <form method="post" action="">

        <label for="budget">Add meg a havi bevételeket:</label>

        <input type="number" id="budget" name="budget" required>

        <br>

        <label for="cost">Add meg a havi költségeket:</label>

        <input type="number" id="cost" name="cost" required>

        <br>

        <button type="submit">Ellenőrizd</button>

    </form>

    <?php

    if (isset($\_POST['budget']) && isset($\_POST['cost'])) {

        $budget = (int)$\_POST['budget'];

        $cost = (int)$\_POST['cost'];

        // *Ternáris operátor használata*

        $message = ($budget >= $cost) ? "A költségvetés elegendő a költségek fedezésére." : "A költségvetés nem elegendő a költségek fedezésére.";

        echo $message . "<br>";

    }

    ?>

</body>

</html>

# 4. Ciklusok PHP-ben: For, While, Do-While, Foreach Ciklusok

A ciklusok a programozás alapvető eszközei, amelyek lehetővé teszik, hogy egy adott műveletsort többször, ismétlődően végrehajtsunk. A PHP nyelvben többféle ciklusszerkezet is létezik, amelyek különböző helyzetekben használhatók hatékonyan. Ebben a fejezetben a PHP nyelvben elérhető ciklusokat, valamint azok működését és felhasználási módjait tárgyaljuk.

## 4.1 Elöl- és hátultesztelő ciklusok

Az elöl- és hátultesztelő ciklusok abban különböznek egymástól, hogy mikor vizsgálják a ciklusfeltételt: az elején vagy a végén.

### 4.1.1 Elöltesztelő ciklus: while

A while ciklus a PHP-ben egy elöltesztelő ciklus, ami azt jelenti, hogy a ciklus belsejében lévő utasítások csak akkor hajtódnak végre, ha a ciklusfeltétel igaz.

**Példa:**

<?php

$szam = 1;

while ($szam <= 5) {

echo "A szám értéke: $szam\n";

$szam++;

}

?>

A fenti kódban a while ciklus addig fut, amíg a $szam változó értéke kisebb vagy egyenlő 5-tel. Minden egyes iterációban növeljük a $szam értékét 1-gyel.

### 4.1.2 Hátultesztelő ciklus: do-while

A do-while ciklus egy hátultesztelő ciklus, ami azt jelenti, hogy legalább egyszer végrehajtja a ciklus törzsét, mielőtt ellenőrzi a ciklusfeltételt.

**Példa:**

<?php

$szam = 1;

do {

echo "A szám értéke: $szam\n";

$szam++;

} while ($szam <= 5);

?>

A do-while ciklus esetében a kód először végrehajtja a ciklus törzsét, majd ezután ellenőrzi a feltételt. Ez garantálja, hogy a ciklus legalább egyszer lefut.

## 4.2 for Ciklus

A for ciklus akkor hasznos, ha egy ismert számú iterációt kell végrehajtani. A ciklus három fő részből áll: inicializálás, feltétel, és a ciklusváltozó frissítése.

**Példa:**

<?php

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

echo "A ciklus száma: $i\n";

}

?>

**Magyarázat:** A for ciklus első része, $i = 1, a ciklusváltozó inicializálását végzi. A második rész, $i <= 5, a ciklusfeltételt határozza meg, míg a harmadik rész, $i++, a ciklusváltozó frissítését végzi minden egyes iteráció után.

## 4.3 foreach Ciklus

A foreach ciklust tömbök és más iterálható adatszerkezetek bejárására használjuk. Ez a ciklus közvetlenül a tömb elemein iterál, így a kód egyszerűbbé válik.

**Példa:**

<?php

$gyumolcsok = ["alma", "banán", "cseresznye"];

foreach ($gyumolcsok as $gyumolcs) {

echo "Gyümölcs: $gyumolcs\n";

}

?>

A foreach ciklus bejárja a $gyumolcsok tömb minden egyes elemét, és az aktuális elemet a $gyumolcs változóba helyezi. Minden iterációban a tömb egy-egy elemét jeleníti meg.

## 4.4 for...of Ciklus PHP-ben

Bár a PHP-ben nincs natív for...of szintaxis, a foreach ciklus tölti be ezt a szerepet. A JavaScript for...of ciklusához hasonlóan a PHP foreach ciklusa is a tömb vagy iterálható objektum elemein megy végig.

**Példa:**

<?php

$nevek = ["John", "Jane", "Doe"];

foreach ($nevek as $nev) {

echo "Név: $nev\n";

}

?>

A fenti foreach ciklus végigmegy a $nevek tömb elemein, és kiírja azokat. Ez a megközelítés hasonló a JavaScript for...of ciklusához.

## 4.5 Feladatok

1. **Feladat:** Írj egy while ciklust, amely 1-től 10-ig kiírja a páros számokat.
2. **Feladat:** Módosítsd az előző feladatot do-while ciklusra.
3. **Feladat:** Írj egy for ciklust, amely 5-től 15-ig kiírja az összes számot.
4. **Feladat:** Adott egy tömb, amely a következő szavakat tartalmazza: ["alma", "banán", "cseresznye"]. Írj egy foreach ciklust, amely kiírja a tömb minden elemét.
5. **Feladat:** Írj egy foreach ciklust, amely végigmegy egy asszociatív tömbön, és kiírja a kulcsokat és az értékeket. Például: ["név" => "John", "kor" => 25, "város" => "Budapest"].

Megoldások:

<?php

// *1. Feladat: Írj egy while ciklust, amely 1-től 10-ig kiírja a páros számokat.*

echo "1. Feladat: Páros számok 1-től 10-ig (while ciklussal):<br>";

$szam = 1;

while ($szam <= 10) {

    if ($szam % 2 == 0) {

        echo "$szam<br>";

    }

    $szam++;

}

echo "<br>";

// *2. Feladat: Módosítsd az előző feladatot do-while ciklusra.*

echo "2. Feladat: Páros számok 1-től 10-ig (do-while ciklussal):<br>";

$szam = 1;

do {

    if ($szam % 2 == 0) {

        echo "$szam<br>";

    }

    $szam++;

} while ($szam <= 10);

echo "<br>";

// *3. Feladat: Írj egy for ciklust, amely 5-től 15-ig kiírja az összes számot.*

echo "3. Feladat: Számok 5-től 15-ig (for ciklussal):<br>";

for ($i = 5; $i <= 15; $i++) {

    echo "$i<br>";

}

echo "<br>";

// *4. Feladat: Adott egy tömb, amely a következő szavakat tartalmazza: ["alma", "banán", "cseresznye"].*

// *Írj egy foreach ciklust, amely kiírja a tömb minden elemét.*

$gyumolcsok = ["alma", "banán", "cseresznye"];

echo "4. Feladat: Gyümölcsök listája (foreach ciklussal):<br>";

foreach ($gyumolcsok as $gyumolcs) {

    echo "$gyumolcs<br>";

}

echo "<br>";

// *5. Feladat: Írj egy foreach ciklust, amely végigmegy egy asszociatív tömbön, és kiírja a kulcsokat és az értékeket.*

// *Például: ["név" => "John", "kor" => 25, "város" => "Budapest"].*

$ember = ["név" => "John", "kor" => 25, "város" => "Budapest"];

echo "5. Feladat: Asszociatív tömb elemei (foreach ciklussal):<br>";

foreach ($ember as $kulcs => $ertek) {

    echo "$kulcs: $ertek<br>";

}

?>

# 5. Függvények Definiálása és Használata: Függvények, Paraméterek, Visszatérési Értékek

A függvények a programozás egyik legfontosabb eszközei. Lehetővé teszik, hogy egy kódrészletet többször is felhasználjunk a program különböző részein. A függvények segítségével elérhetjük a kód újrafelhasználhatóságát, könnyebben karbantarthatóvá tesszük a programot, és jobban szervezhetjük a kódot.

## 5.1 Függvények Definiálása

PHP-ban a függvényeket a function kulcsszóval definiáljuk. Egy függvénynek lehetnek paraméterei, amiket a függvény hívásakor adunk át, valamint lehet visszatérési értéke, amelyet a return kulcsszóval adunk vissza.

**Egyszerű függvény definiálása:**

<?php

function koszontes() {

echo "Üdvözöllek a PHP világában!\n";

}

?>

Ebben a példában a koszontes nevű függvény egy egyszerű üzenetet ír ki. A függvény definíciójában nincsenek paraméterek, és nem tér vissza semmilyen értékkel.

## 5.2 Függvények Meghívása

Miután definiáltunk egy függvényt, azt meghívhatjuk a nevének használatával. Ha a függvény fogad paramétereket, akkor azokat a függvény neve után, zárójelek között kell megadni.

**Függvény meghívása:**

<?php

koszontes(); // Üdvözöllek a PHP világában!

?>

**Magyarázat:**  
A koszontes függvény meghívásakor a függvény végrehajtja a definiált utasításokat, ebben az esetben egy üzenetet ír ki.

## 5.3 Paraméterek a Függvényekben

A függvények paramétereket is fogadhatnak, amelyek lehetővé teszik, hogy a függvény működése dinamikus legyen. A paramétereket a függvény definíciójában, a zárójelek között kell megadni.

**Példa paraméterekkel rendelkező függvényre:**

<?php

function koszontes($nev) {

echo "Üdvözöllek, $nev!\n";

}

?>

Ebben a példában a koszontes függvény egy $nev nevű paramétert fogad, és azt az üzenet részeként használja.

**Függvény meghívása paraméterrel:**

<?php

koszontes("Anna"); // Üdvözöllek, Anna!

koszontes("Béla"); // Üdvözöllek, Béla!

?>

Amikor a koszontes függvényt meghívjuk, átadunk egy nevet paraméterként, amit a függvény kiír az üzenet részeként.

## 5.4 Visszatérési Értékek

Egy függvény visszatérhet egy értékkel a return kulcsszó használatával. A visszatérési értéket a függvény meghívásakor egy változóba tárolhatjuk, vagy közvetlenül felhasználhatjuk.

**Példa visszatérési értékkel rendelkező függvényre:**

<?php

function osszeadas($a, $b) {

return $a + $b;

}

?>

Az osszeadas függvény két paramétert fogad, $a és $b, és visszatér a két szám összegével.

**Függvény meghívása visszatérési értékkel:**

<?php

$eredmeny = osszeadas(3, 5);

echo "Az eredmény: $eredmeny\n"; // Az eredmény: 8

?>

**Magyarázat:**  
Ebben a példában az osszeadas függvény visszatérési értéke az $eredmeny változóban kerül tárolásra, amelyet aztán kiírunk a képernyőre.

## 5.5 Alapértelmezett Értékek

A függvények paraméterei alapértelmezett értékeket is kaphatnak. Ha a függvény hívásakor nem adunk meg értéket a paraméternek, akkor az alapértelmezett érték kerül felhasználásra.

**Példa alapértelmezett értékekkel rendelkező függvényre:**

<?php

function koszontes($nev = "Vendég") {

echo "Üdvözöllek, $nev!\n";

}

?>

**Magyarázat:**  
Ebben a példában, ha a koszontes függvényt hívjuk meg anélkül, hogy átadnánk egy nevet, akkor az alapértelmezett "Vendég" nevet használja.

**Függvény meghívása alapértelmezett értékkel:**

<?php

koszontes(); // Üdvözöllek, Vendég!

koszontes("Anna"); // Üdvözöllek, Anna!

?>

## 5.6 Több Visszatérési Érték

PHP-ban egy függvény egyszerre több értéket is visszaadhat egy tömb vagy asszociatív tömb használatával.

**Példa több visszatérési értékre:**

<?php

function szamok() {

return [1, 2, 3];

}

list($elso, $masodik, $harmadik) = szamok();

echo "$elso, $masodik, $harmadik\n"; // 1, 2, 3

?>

A szamok függvény egy tömböt ad vissza, amelyet a list függvénnyel külön változókba bontunk.

## 5.7 Feladatok

1. **Feladat:** Definiálj egy függvényt, amely egy adott számnak kiszámolja a négyzetét, és adja vissza az eredményt.
2. **Feladat:** Írj egy függvényt, amely két számot ad össze, és adja vissza az eredményt. A függvény alapértelmezett paraméterként adjon 0-t a második számnak, ha az nem kerül megadásra.
3. **Feladat:** Készíts egy függvényt, amely egy adott nevet fogad, és köszönő üzenetet ír ki az illető számára. Ha a név nem kerül megadásra, a függvény "Vendég"-et használjon alapértelmezett értékként.
4. **Feladat:** Definiálj egy függvényt, amely egy tömböt fogad paraméterként, és visszatér a tömb elemeinek összegével.
5. **Feladat:** Írj egy függvényt, amely visszatér egy tömbbel, amelyben az első elem egy név, a második elem az életkor, és a harmadik elem a város neve. Majd bontsd ki a visszaadott értékeket külön változókba és jelenítsd meg őket.

Megoldások:

<?php

// *1. Feladat: Definiálj egy függvényt, amely egy adott számnak kiszámolja a négyzetét, és adja vissza az eredményt.*

function negyzet($szam) {

    return $szam \* $szam;

}

// *Tesztelés*

echo "1. Feladat: A 4 négyzete: " . negyzet(4) . "<br>"; // *16*

echo "<br>";

// *2. Feladat: Írj egy függvényt, amely két számot ad össze, és adja vissza az eredményt.*

// *A függvény alapértelmezett paraméterként adjon 0-t a második számnak, ha az nem kerül megadásra.*

function osszeadas($a, $b = 0) {

    return $a + $b;

}

// *Tesztelés*

echo "2. Feladat: Az 5 + 10 összege: " . osszeadas(5, 10) . "<br>"; // *15*

echo "2. Feladat: Az 5 + 0 (alapértelmezett) összege: " . osszeadas(5) . "<br>"; // *5*

echo "<br>";

// *3. Feladat: Készíts egy függvényt, amely egy adott nevet fogad, és köszönő üzenetet ír ki az illető számára.*

// *Ha a név nem kerül megadásra, a függvény "Vendég"-et használjon alapértelmezett értékként.*

function koszones($nev = "Vendég") {

    echo "Üdvözöllek, $nev!<br>";

}

// *Tesztelés*

koszones("Anna"); // *Üdvözöllek, Anna!*

koszones(); // *Üdvözöllek, Vendég!*

echo "<br>";

// *4. Feladat: Definiálj egy függvényt, amely egy tömböt fogad paraméterként, és visszatér a tömb elemeinek összegével.*

function osszeg($tomb) {

    return array\_sum($tomb);

}

// *Tesztelés*

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

echo "4. Feladat: A tömb elemeinek összege: " . osszeg($szamok) . "<br>"; // *15*

echo "<br>";

// *5. Feladat: Írj egy függvényt, amely visszatér egy tömbbel, amelyben az első elem egy név, a második elem az életkor,*

// *és a harmadik elem a város neve. Majd bontsd ki a visszaadott értékeket külön változókba és jelenítsd meg őket.*

function szemelyAdatok() {

    return ["név" => "John", "kor" => 30, "város" => "Budapest"];

}

// *Tesztelés*

$adatok = szemelyAdatok();

echo "5. Feladat: Név: " . $adatok['név'] . ", Kor: " . $adatok['kor'] . ", Város: " . $adatok['város'] . "<br>";

// *Név: John, Kor: 30, Város: Budapest*

# PHP Szuperglobális Változók

A PHP szuperglobális változói speciális típusú változók, amelyek előre meghatározottak és globális hatókörrel rendelkeznek. Ez azt jelenti, hogy a PHP-szkript bármely részéből, beleértve a függvényeket, osztályokat és fájlokat, elérhetők.

#### Miért hasznosak a szuperglobálisok?

* **Globális hatókör**: Bárhonnan elérhetők a kódban, így könnyű adatokat továbbítani a különböző részei között.
* **Előre meghatározott értékek**: Automatikusan feltöltődnek adatokkal a felhasználói bemenetből, szerverinformációkból vagy HTTP-fejlécekből.
* **Elnevezési konvenció**: Mindig egy aláhúzással kezdődnek, pl. $\_GET, $\_POST, segítve a megkülönböztetést a szokásos változóktól.

## 6.1 $\_GET

A **$\_GET** szuperglobális változó egy asszociatív tömb, amely a böngésző URL-jén keresztül, a HTTP GET módszerrel átadott adatokat tartalmazza. A GET módszert általában olyan kérések esetén használják, amelyek nem módosítanak adatokat a szerveren (pl. adatkeresésnél). Ezek az adatok az URL-ben jelennek meg a kérdőjel (?) után, ahol a kulcs-érték párokat amperszand (&) jel választja el egymástól.

**Hogyan működik a $\_GET?**

Az URL egy adott oldalhoz való hozzáférést biztosít, és gyakran tartalmazhat paramétereket, amelyeket a szerver felhasználhat a válasz személyre szabásához. Például, ha a felhasználó egy űrlapot tölt ki és a GET módszerrel küldi el az adatokat, akkor ezek az adatok az URL részeivé válnak.

**Példa URL**:  
example.com/index.php?name=János&age=25

Ebben az esetben az URL két paramétert tartalmaz:

* name: ennek az értéke János
* age: ennek az értéke 25

A PHP **$\_GET** tömbje ezeket a paramétereket automatikusan feldolgozza, és kulcs-érték párokként tárolja. Tehát a fenti példa esetén a **$\_GET['name']** értéke János, és a **$\_GET['age']** értéke 25.

**Az URL-ből kapott paraméterek használata a $\_GET-tel**

Amikor a szerver fogad egy GET kérésű URL-t, a PHP a **$\_GET** tömbben tárolja az összes átadott paramétert. Ezeket a paramétereket a szkript bárhonnan elérheti és feldolgozhatja.

// Ellenőrizzük, hogy a GET paraméterek léteznek-e

if (isset($\_GET['name']) && isset($\_GET['age'])) {

// Ha igen, kiírjuk a nevét és az életkorát

echo "Név: " . $\_GET['name'] . ", Kor: " . $\_GET['age'];

}

* Az isset() függvény használatával ellenőrizzük, hogy a paraméterek léteznek-e a **$\_GET** tömbben. Ez azért fontos, mert ha egy paraméter nincs megadva az URL-ben, a PHP hibaüzenetet adna, ha közvetlenül próbálnánk meg elérni.
* Ha a paraméterek léteznek, akkor azokat a **$\_GET** tömbből kiolvassuk, és egyszerűen megjelenítjük a felhasználónak.

**Példa URL**:  
example.com/index.php?name=János&age=25

**URL-ből kiolvasott eredmény**:

Név: János, Kor: 25

**Mire használható a $\_GET?**

1. **Adatok továbbítása URL-ben**: A **$\_GET** segítségével egyszerűen továbbíthatunk adatokat az URL-en keresztül, például keresési lekérdezéseket vagy szűrési paramétereket. Ezek az adatok felhasználhatók a szerveroldalon a keresési eredmények vagy a tartalom szűréséhez.
   * Példa: example.com/search.php?q=php+tutorials keresési lekérdezés küldése.
2. **Oldalak személyre szabása**: A **$\_GET** tömb felhasználható arra, hogy különböző paraméterek alapján módosítsuk a weboldal tartalmát. Például egy webshopban a felhasználó kiválaszthat egy kategóriát, és a GET paraméterek alapján csak az adott kategória termékeit jelenítjük meg.
   * Példa: example.com/shop.php?category=electronics
3. **Kis méretű adatok továbbítása**: A GET módszer korlátozott mennyiségű adatot képes továbbítani (általában néhány ezer karakter), így ideális kis méretű adatok, például űrlapadatok vagy linkek továbbítására.

**Fontos megjegyzések:**

1. **Biztonság**:
   * A GET módszerrel küldött adatok az URL-ben jelennek meg, így láthatók a böngésző címsorában és elmenthetők a böngésző előzményeibe. Emiatt érzékeny adatokat (például jelszavakat vagy banki adatokat) soha nem szabad GET-tel továbbítani.
   * A felhasználó az URL-ben lévő adatokat manipulálhatja. Ezért mindig ellenőrizni és szűrni kell a GET paramétereket, hogy elkerüljük a biztonsági problémákat, például az SQL injekciót vagy a XSS (cross-site scripting) támadásokat.

**Példa szűrésre**:

$name = htmlspecialchars($\_GET['name']); // Speciális karakterek átalakítása

1. **GET vs POST**: A GET módszert főként olyan kérésekhez használjuk, amelyek nem változtatják meg az adatokat a szerveren, például keresési vagy szűrési kérésekhez. A POST módszert akkor használjuk, ha a felhasználó adatokat küld be (pl. űrlapok), és ezek az adatok nagyobbak, vagy érzékenyebbek.
2. **Adatmennyiség korlátozása**: A GET módszerrel küldött adatok mérete általában korlátozott (a böngésző és a szerver beállításaitól függően), míg a POST módszerrel nagyobb mennyiségű adat is továbbítható.

A **$\_GET** szuperglobális változó egy rendkívül hasznos eszköz, amely lehetővé teszi a fejlesztők számára, hogy az URL-ből könnyedén hozzáférjenek a felhasználói adatokhoz. Ez különösen hasznos keresések, szűrések és egyéb egyszerű adatok továbbítására a böngészőből a szerverre. Azonban mindig figyelni kell a biztonsági szempontokra, és megfelelően szűrni kell a beérkező adatokat.

## 6.2 $\_POST

A **$\_POST** szuperglobális változó egy asszociatív tömb, amely a HTTP POST módszerrel küldött adatokat tartalmazza. A POST módszer az egyik leggyakoribb módja annak, hogy az űrlapok adatait a szerverre továbbítsuk, különösen olyan esetekben, amikor nagyobb mennyiségű adatot (pl. szöveges mezők, fájlok) vagy érzékeny információkat (pl. jelszavakat) kell küldeni. A **$\_POST** adatokat nem jelenítik meg az URL-ben, ami biztonságosabbá teszi bizonyos típusú adatküldések esetén.

**Hogyan működik a $\_POST?**

Amikor egy felhasználó egy HTML formot kitölt és elküldi, a böngésző az űrlap adatait a szerverre küldi egy HTTP kérésben. Ha az űrlap **POST** módszerrel van elküldve, az adatokat nem látjuk az URL-ben, hanem a kérés testében kerülnek továbbításra. A PHP a szerveren automatikusan feldolgozza ezt a kérést, és az adatokat a **$\_POST** szuperglobális tömbben tárolja.

**Példakód:**

Egy egyszerű HTML űrlap, amely POST módszerrel küldi el az adatokat a szerverre:

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

<input type="text" name="name" placeholder="Név">

<input type="submit" value="Küldés">

</form>

* **method="POST"**: Ez meghatározza, hogy a form adatait a POST módszerrel küldjük el.
* **action="submit.php"**: Az űrlap adatainak céloldala, amely a form feldolgozásáért felelős.

**PHP rész:**

// submit.php fájl

if (isset($\_POST['name'])) {

echo "Küldött név: " . $\_POST['name'];

}

* Az **isset()** függvényt használjuk annak ellenőrzésére, hogy a felhasználó ténylegesen küldött-e adatot az űrlapról. Ha igen, az adat a **$\_POST['name']** tömbben található, és kiírjuk a megadott nevet.

**Példa működés:**

Tegyük fel, hogy a felhasználó az űrlapon a következő adatokat adja meg:

* **Név**: Péter

Miután a felhasználó rákattint a „Küldés” gombra, a böngésző elküldi az adatokat a szerverre POST kéréssel. A **submit.php** feldolgozza az adatokat, és a következő eredményt adja:

Küldött név: Péter

**A POST módszer előnyei**

1. **Adatvédelem**: Mivel a POST módszerrel küldött adatok nem jelennek meg az URL-ben, biztonságosabb, különösen érzékeny információk, mint például jelszavak vagy személyes adatok továbbításakor.
2. **Nagyobb adatmennyiség**: A POST módszer lehetővé teszi, hogy nagyobb mennyiségű adatot küldjünk a szerverre, míg a GET módszernél van egy korlátozás az URL maximális hosszára (böngésző és szerver korlátok miatt).
3. **Bonyolultabb űrlapok kezelése**: A POST módszer különösen hasznos összetettebb űrlapok esetében, amelyek fájlokat, több soros szöveget vagy más bonyolult struktúrákat tartalmaznak.

**Mire használható a $\_POST?**

1. **Űrlapadatok továbbítása**: A **$\_POST** használható egyszerű űrlapok adatainak továbbítására, például felhasználói regisztrációs űrlapokra, bejelentkezési oldalakra, vagy kapcsolatfelvételi űrlapokra.
2. **Fájlok feltöltése**: A **$\_POST** módszerrel könnyen kezelhetjük a fájlokat is, amikor egy űrlapon keresztül fájlokat töltenek fel. Ehhez kombinálhatjuk a **$\_FILES** szuperglobális változóval.
3. **Jelszavak és érzékeny adatok kezelése**: A POST módszer sokkal biztonságosabb az olyan űrlapokhoz, amelyek érzékeny adatokat kérnek, mivel ezek nem lesznek láthatók az URL-ben.

**Példakód jelszavak küldésére:**

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

<input type="text" name="username" placeholder="Felhasználónév">

<input type="password" name="password" placeholder="Jelszó">

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

</form>

A fent említett űrlap a felhasználónevet és a jelszót POST módszerrel küldi el a szerverre, anélkül, hogy ezek megjelennének az URL-ben.

**Fontos megjegyzések:**

1. **Biztonság**:
   * Bár a POST módszer biztonságosabb a GET-nél, nem jelent önmagában teljes védelmet. Mindig használni kell biztonsági intézkedéseket, például HTTPS-t az adatok titkosított továbbításához, és érvényesíteni kell az adatokat a szerver oldalon is.

**Példa érvényesítésre**:

$name = htmlspecialchars($\_POST['name']); // HTML karakterek átalakítása

1. **Nagy mennyiségű adat kezelés**: A POST módszerrel nagy mennyiségű adatot küldhetünk, például hosszabb szöveges adatokat vagy több mezővel rendelkező űrlapokat.
2. **Hozzáférés szűrés**: A POST módszer használatával küldött adatok mindig a **$\_POST** szuperglobálison keresztül érhetők el, függetlenül attól, hogy hány oldal van érintve. Ez azt jelenti, hogy minden egyes oldal, amelyhez hozzáférünk, hozzáférhet az űrlap által küldött adatokhoz is.

**GET vs POST összehasonlítás:**

* **GET**:
  + Az adatok az URL-ben jelennek meg.
  + Korlátozott mennyiségű adat küldhető (általában néhány ezer karakter).
  + Nem alkalmas érzékeny adatok továbbítására.
  + Általában adatlekérdezésre vagy oldalfrissítésekre használják.
* **POST**:
  + Az adatok a kérés testében vannak, nem láthatók az URL-ben.
  + Nagyobb mennyiségű adat küldhető.
  + Alkalmas érzékeny adatok küldésére (pl. jelszavak).
  + Általában adatbeküldésre és fájlok feltöltésére használják.

A **$\_POST** szuperglobális változó hatékony és biztonságos módszer a felhasználói adatok szerverre történő továbbítására. Ez a módszer különösen hasznos olyan helyzetekben, amikor érzékeny adatokat kell továbbítani, vagy nagyobb mennyiségű adatot kell feldolgozni. A POST módszer segítségével könnyedén elérhetők és feldolgozhatók a felhasználói adatok, miközben az adatok védelmét is biztosítja.

## 6.3 $\_SESSION

A **$\_SESSION** szuperglobális lehetővé teszi a munkamenetek kezelését a PHP-ban. Ez különösen hasznos, ha több oldalkérésen keresztül szeretnénk megőrizni a felhasználói adatokat. A munkamenet (session) egy szerveroldalon tárolt adat, amely egyedi felhasználókhoz van kötve. Minden felhasználó saját munkamenet-azonosítót (session ID) kap, amely alapján a szerver nyomon tudja követni a felhasználót, még akkor is, ha az átlép egyik oldalról a másikra.

**Hogyan működik a munkamenet?**

1. **Munkamenet indítása**: A PHP-ban a munkamenet-kezelést a session\_start() függvénnyel indítjuk. Ennek hatására a szerver létrehozza vagy folytatja az adott felhasználóhoz tartozó munkamenetet, ami a felhasználó böngészőjében egyedileg azonosítva van egy **session ID** nevű cookie segítségével.
2. **Adatok tárolása**: Miután a munkamenet elindult, a $\_SESSION szuperglobális tömbön keresztül bármilyen adatot tárolhatunk a szerveren, amely a felhasználóra vonatkozik. Az adatokat ezután több oldalkérés során is elérhetjük.
3. **Adatok elérése**: A felhasználóra vonatkozó adatokat bármelyik oldalról elérhetjük, ahol a session\_start() függvény hívva van.
4. **Munkamenet befejezése**: Ha már nincs szükség a munkamenetre, vagy a felhasználó kijelentkezik, a munkamenet megszüntethető a session\_destroy() függvénnyel. Ez törli a szerveren tárolt adatokat és érvényteleníti a felhasználó session ID-jét.

**Részletes példa a munkamenet használatára:**

**1. Oldal: Munkamenet indítása és adat tárolása; pl: belépés egy oldalra**

// Munkamenet indítása

session\_start();

// Adat tárolása a munkamenetben

$\_SESSION['user'] = "Lucifer";

// Az adat elérése

echo "Üdvözöllek, " . $\_SESSION['user'];

* A session\_start() elindítja a munkamenetet.
* A $\_SESSION['user'] változóban tároljuk a felhasználó nevét, amelyet most "Lucifer"-nek állítunk be.
* Ez az adat elérhető minden későbbi oldalon, amely ugyanennek a munkamenetnek a része.

**2. Oldal: Munkamenet adatainak elérése; a személyes adatok elérése**

session\_start();

// Ellenőrzés, hogy van-e munkamenet adat

if (isset($\_SESSION['user'])) {

echo "Üdv újra, " . $\_SESSION['user'];

} else {

echo "Nincs bejelentkezett felhasználó.";

}

* A session\_start() függvény folytatja a már meglévő munkamenetet, így elérhetjük a korábban tárolt adatokat.
* A $\_SESSION['user'] változóban elérjük a felhasználó nevét, amit korábban "János"-nak állítottunk be.

**3. Oldal: Munkamenet törlése -> kijelentkezés**

// Munkamenet indítása

session\_start();

// Munkamenet adatainak törlése

session\_unset(); // Törli a $\_SESSION tömb összes elemét

// Munkamenet megszüntetése

session\_destroy(); // A munkamenet teljes megszüntetése

* A session\_unset() törli a munkamenetben tárolt összes adatot.
* A session\_destroy() teljesen megszünteti a munkamenetet, ami azt jelenti, hogy a felhasználó session ID-je érvénytelen lesz, és a következő kéréskor új munkamenetet kell indítani.

**Munkamenetek előnyei:**

* **Állapot megtartása**: Lehetővé teszi, hogy az alkalmazás "emlékezzen" a felhasználóról több kéréssel kapcsolatban.
* **Biztonság**: A munkamenet-adatok szerveroldalon vannak tárolva, így kevésbé érzékenyek a kliensoldali manipulációkra, mint például a cookie-k.
* **Egyszerű használat**: Könnyű implementálni és felhasználni bármilyen PHP projektben.

**Fontos tudnivalók:**

* **session\_start()**: Ezt minden olyan oldal tetején meg kell hívni, ahol a munkamenet adatokat használni szeretnénk.
* **session ID**: A munkamenet azonosításához szükséges az egyedi session ID, amit a PHP automatikusan kezel.
* **Biztonság**: Fontos a munkamenetek biztonságos kezelése, például titkosított kapcsolat (HTTPS) használatával, hogy elkerüljük a session hijacking-et (munkamenet eltérítését).

Ez az eljárás biztosítja, hogy a felhasználó által megadott információk több oldal kérésén keresztül megőrződjenek, például bejelentkezéskor vagy kosár funkció esetén.

## 6.4 $\_COOKIE

A $\_COOKIE szuperglobális a sütik segítségével tárolt adatokat tartalmazza.

// Süti beállítása

setcookie("username", "János", time() + (86400 \* 30), "/"); // 30 napig érvényes

// Süti elérése

if (isset($\_COOKIE['username'])) {

echo "Süti tartalma: " . $\_COOKIE['username'];

}

**Mi az a süti?**

A süti (cookie) egy kis adatcsomag, amelyet a webkiszolgáló (szerver) küld a felhasználó böngészőjébe, és amelyet a böngésző minden későbbi kéréskor visszaküld a szervernek. A sütik általában olyan információkat tárolnak, mint például a felhasználó bejelentkezési állapota, nyelvi preferenciák vagy egyéb beállítások.

**Sütik beállítása**

A setcookie() függvénnyel állíthatunk be sütiket. A függvény szintaxisa a következő:

setcookie(name, value, expire, path, domain, secure, httponly);

* **name**: A süti neve.
* **value**: A süti értéke.
* **expire**: Az időpont, ameddig a süti érvényes lesz, UNIX időbélyeg formájában. (pl. time() + (86400 \* 30) - 30 nap múlva)
* **path**: A weboldal azon útvonala, amelyhez a süti érvényes. Ha "/"-ot adunk meg, akkor az egész weboldalon érvényes lesz.
* **domain**: Az a domain, amelyhez a süti érvényes.
* **secure**: Ha true, akkor a süti csak HTTPS kapcsolaton keresztül lesz elküldve.
* **httponly**: Ha true, akkor a süti csak HTTP kéréseken keresztül érhető el, nem JavaScriptből.

**Példa sütik beállítására:**

setcookie("username", "János", time() + (86400 \* 30), "/"); // 30 napig érvényes

Ebben a példában a süti neve username, értéke János, és 30 napig lesz érvényes, az egész weboldalon elérhető.

**Sütik elérése**

A $\_COOKIE szuperglobális tömb tartalmazza az összes sütit, amelyet a böngésző visszaküldött a szervernek. Az egyes sütik értékei a tömb kulcsai alapján érhetők el.

**Példa sütik elérésére:**

if (isset($\_COOKIE['username'])) {

echo "Süti tartalma: " . $\_COOKIE['username'];

} else {

echo "A süti nem létezik.";

}

Ebben a példában ellenőrizzük, hogy a username nevű süti létezik-e, és ha igen, kiírjuk annak értékét. Ha nem létezik, akkor egy üzenetet adunk vissza.

**Fontos tudnivalók**

1. **Sütik és biztonság**: A sütik nem titkosítva tárolják az adatokat, ezért érzékeny információk tárolására nem javasoltak. Ha érzékeny adatokat kell tárolni, használjunk HTTPS-t, és lehetőség szerint titkosítsuk az adatokat.
2. **Sütik élettartama**: A sütik élettartamát az expire paraméter határozza meg. Ha nem állítunk be expire időpontot, a süti csak a böngésző munkamenete alatt lesz elérhető.
3. **Sütik és adatvédelem**: A sütik használata adatvédelmi kérdéseket vethet fel, ezért fontos, hogy a felhasználókat tájékoztassuk arról, hogy sütiket használunk, és biztosítsuk, hogy megfeleljünk a vonatkozó adatvédelmi törvényeknek (pl. GDPR).

## 6.5 $\_REQUEST

A **$\_REQUEST** változó egy speciális szuperglobális tömb a PHP-ban, amely lehetővé teszi, hogy a felhasználói bemenetet egyetlen helyről érjük el, függetlenül attól, hogy az adatokat GET, POST vagy COOKIE módszerrel küldték. Ez a tulajdonság kényelmes lehet az adatok lekérésekor, de bizonyos biztonsági kockázatokat is hordoz, mivel nem különbözteti meg az adatforrásokat.

**Hogyan működik a $\_REQUEST?**

A PHP automatikusan feltölti a **$\_REQUEST** tömböt az alábbi adatokkal:

* **$\_GET**: Az URL-ben lévő paraméterekkel érkező adatokat tartalmazza.
* **$\_POST**: A HTTP POST módszerrel küldött adatokat tartalmazza (például űrlapok adatai).
* **$\_COOKIE**: A felhasználói böngészőben tárolt sütiket tartalmazza, amelyeket a szerver felé továbbítanak.

A $\_REQUEST egy **kombinált** tömb, ami azt jelenti, hogy ezek az adatok egyesítve vannak egyetlen tömbben, és a tömb kulcsai a változók nevei (mint $\_GET, $\_POST vagy $\_COOKIE esetében).

**Előnyei:**

1. **Egyszerűsített hozzáférés**: Nem kell külön ellenőrizni, hogy az adat GET, POST vagy COOKIE módszerrel érkezett. A $\_REQUEST segítségével egyetlen helyről elérhetők.
2. **Rugalmasság**: Ha az adat forrása nem fontos, és csak a változó létezése érdekel, a $\_REQUEST egyszerűbbé teszi az adatok kezelését.

**Hátrányai:**

1. **Biztonsági kockázat**: Mivel a $\_REQUEST nem különbözteti meg az adatforrásokat, nehéz lehet biztosítani, hogy egy adat megbízható forrásból származik-e. Például egy **$\_POST** változó könnyen felülírható egy **$\_GET** vagy **$\_COOKIE** változóval. Ezért a biztonságos és pontos adatkezelés érdekében inkább konkrétan a **$\_GET**, **$\_POST**, vagy **$\_COOKIE** szuperglobálisokat érdemes használni.
2. **Nem specifikus**: A $\_REQUEST nem jelzi, hogy az adat melyik módszerrel érkezett, ezért elveszhet az az információ, hogy egy adatot GET, POST vagy COOKIE módszerrel kaptunk-e meg. Ez zavaró lehet egyes esetekben, például ha POST és GET változók azonos nevűek.

**Példa a $\_REQUEST használatára:**

**Adatküldés GET vagy POST módszerrel:**

// Ha egy űrlap GET vagy POST módszerrel küldi az adatokat, mindkettőt elérhetjük $\_REQUEST segítségével:

if (isset($\_REQUEST['name'])) {

echo "Név: " . $\_REQUEST['name'];

}

**GET módszer esetén:**

// URL: example.com/page.php?name=János

// Eredmény: Név: János

**POST módszer esetén:**

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

<input type="text" name="name" value="Péter">

<input type="submit" value="Küldés">

</form>

// POST kérés után: Név: Péter

**COOKIE használata:**

// Beállítunk egy cookie-t:

setcookie("name", "Kata", time() + 3600); // 1 órán át érvényes

// $\_REQUEST segítségével elérhetjük a cookie-t:

if (isset($\_REQUEST['name'])) {

echo "Név: " . $\_REQUEST['name'];

}

Ebben az esetben a $\_REQUEST['name'] tartalmazhatja a sütit is, ha létezik.

**$\_REQUEST használatának kockázatai:**

Egy egyszerű példával bemutatom, miért kevésbé biztonságos a $\_REQUEST használata:

**Probléma: GET és POST ütközés** Ha ugyanazt a kulcsot használjuk GET és POST változókban, a $\_REQUEST a két forrásból érkező adatokat összemossa, és előnyben részesítheti az egyiket (általában a GET-et).

**Példa:**

// URL: example.com/page.php?name=Éva

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

<input type="text" name="name" value="Kata">

<input type="submit" value="Küldés">

</form>

if (isset($\_REQUEST['name'])) {

echo "Név: " . $\_REQUEST['name'];

}

Ha egy felhasználó POST módszerrel küldi az adatot ("Kata"), de a GET kérés tartalmaz egy "name" paramétert is ("Éva"), akkor a $\_REQUEST['name'] valószínűleg az URL-ben lévő GET paramétert fogja visszaadni ("Éva"), és nem a POST kéréssel küldött adatot ("Kata"). Ez biztonsági kockázatot jelenthet, például amikor érzékeny adatokat POST módszerrel továbbítunk.

**Mikor érdemes használni a $\_REQUEST-t?**

* Ha az adat forrása nem lényeges, és egyszerűen csak el akarjuk érni a változót, függetlenül attól, hogy GET, POST vagy COOKIE módszerrel érkezett.
* Ha gyors prototípust készítünk, és a biztonság nem elsődleges szempont.

**Mikor ne használjuk a $\_REQUEST-t?**

* Ha a biztonság kritikus, például érzékeny adatokat továbbítunk (pl. felhasználói bejelentkezés, jelszó). Ilyen esetben a **$\_POST** használata javasolt.
* Ha pontosan tudni akarjuk, hogy az adat GET vagy POST módszerrel érkezett.

**Összefoglalás:**

* **$\_REQUEST**: Kényelmes módja az adatok elérésének, ha nem számít, hogy GET, POST vagy COOKIE módszerrel érkeztek.
* **Használata biztonsági kockázatot rejthet**: Ügyeljünk arra, hogy a $\_REQUEST nem különbözteti meg az adatforrásokat, így könnyen ütközhetnek a változók.
* **Biztonságosabb alternatívák**: Ha fontos, hogy az adat milyen módszerrel érkezett, inkább használjuk a konkrét szuperglobális változókat: **$\_GET**, **$\_POST**, vagy **$\_COOKIE**.

A $\_REQUEST tehát kényelmes, de fokozott figyelmet igényel a biztonságos használat érdekében.

## 6.6 $\_FILES

A **$\_FILES** szuperglobális segítségével a PHP képes kezelni a HTTP POST kéréseken keresztül érkező fájlokat. Ezt általában űrlapokban használják, amikor a felhasználóknak lehetőségük van fájlokat feltölteni a szerverre. A **$\_FILES** egy tömb, amely tartalmazza a feltöltött fájlokkal kapcsolatos információkat, például a fájl nevét, típusát, méretét és az ideiglenes helyét, ahova a szerver elmenti a fájlt a feldolgozás előtt.

**Hogyan működik a fájlfeltöltés a PHP-ban?**

**HTML űrlap létrehozása fájl feltöltéséhez**

Az űrlapban használnunk kell a enctype="multipart/form-data" attribútumot, hogy a fájlokat megfelelően továbbítsuk a szervernek. Ez az attribútum lehetővé teszi, hogy a böngésző többféle adatot is küldjön, például szöveget és fájlokat egyaránt.

**Példa űrlap:**

html

Kód másolása

<form method="POST" action="upload.php" enctype="multipart/form-data">

<input type="file" name="fileToUpload">

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

</form>

* Az űrlap egy fileToUpload nevű input mezőt tartalmaz, amely lehetővé teszi a felhasználó számára, hogy fájlt válasszon ki a feltöltéshez.
* Az enctype="multipart/form-data" fontos, mert enélkül a fájl nem kerülne megfelelően továbbításra a szervernek.

**A feltöltött fájl feldolgozása a szerveren**

Amikor a felhasználó rákattint az "Feltöltés" gombra, a PHP a **$\_FILES** tömbbe tölti a fájllal kapcsolatos adatokat.

Az alábbi **$\_FILES** tömb tartalmazza a következő kulcsokat minden fájlhoz:

* **name**: A feltöltött fájl eredeti neve.
* **type**: A fájl MIME típusa (pl. image/jpeg, application/pdf).
* **tmp\_name**: Az ideiglenes hely, ahol a fájl tárolva van a szerveren, amíg fel nem dolgozzuk.
* **error**: Hibakód, amely jelzi, hogy történt-e hiba a feltöltés során.
* **size**: A fájl mérete bájtokban.

**PHP kód a fájl feltöltéséhez:**

// Ellenőrizzük, hogy történt-e fájlfeltöltés

if (isset($\_FILES['fileToUpload'])) {

// A feltöltött fájl neve

$fileName = $\_FILES['fileToUpload']['name'];

// A fájl ideiglenes helye a szerveren

$fileTmpName = $\_FILES['fileToUpload']['tmp\_name'];

// A fájl áthelyezése a cél könyvtárba

if (move\_uploaded\_file($fileTmpName, "uploads/" . $fileName)) {

echo "A fájl sikeresen feltöltve: " . $fileName;

} else {

echo "Hiba történt a fájl feltöltése során.";

}

}

**Részletezve a feltöltési folyamatot:**

1. **$\_FILES['fileToUpload']['name']**: A feltöltött fájl eredeti neve, amelyet a felhasználó a fájlkiválasztó mezőn keresztül tölt fel. Például: kep.jpg.
2. **$\_FILES['fileToUpload']['tmp\_name']**: Ez az ideiglenes hely, ahol a fájl tárolva van a szerveren, amíg fel nem dolgozzuk. Ez egy átmeneti fájl, amit a PHP automatikusan létrehoz.
3. **move\_uploaded\_file() függvény**: Ez a funkció áthelyezi a fájlt az ideiglenes helyről egy állandó helyre a szerveren. Például a uploads/ könyvtárba helyezhetjük át. Fontos, hogy a cél könyvtár létezzen, és a szervernek megfelelő írási jogai legyenek a fájl elmentéséhez.
4. **Hibaellenőrzés**: A feltöltés során előfordulhatnak hibák, amelyeket a **$\_FILES['fileToUpload']['error']** változó tartalmaz. Ennek segítségével ellenőrizhetjük, hogy történt-e probléma.

**Hibakódok és kezelésük:**

A **$\_FILES['fileToUpload']['error']** egy hibakódot tartalmaz, amely jelzi, hogy mi történt a fájl feltöltése során. A leggyakoribb hibakódok:

* **UPLOAD\_ERR\_OK (0)**: A fájl sikeresen feltöltődött.
* **UPLOAD\_ERR\_INI\_SIZE (1)**: A fájl mérete meghaladja a upload\_max\_filesize irányelvben meghatározott méretet a php.ini fájlban.
* **UPLOAD\_ERR\_FORM\_SIZE (2)**: A fájl mérete meghaladja az űrlapon beállított maximális méretet (MAX\_FILE\_SIZE irányelv).
* **UPLOAD\_ERR\_PARTIAL (3)**: A fájl csak részben lett feltöltve.
* **UPLOAD\_ERR\_NO\_FILE (4)**: Nem töltöttek fel fájlt.

Például egy egyszerű hibakezelés így nézhet ki:

if ($\_FILES['fileToUpload']['error'] === UPLOAD\_ERR\_OK) {

// Fájl sikeresen feltöltve

} else {

// Hiba történt

echo "Hiba történt: " . $\_FILES['fileToUpload']['error'];

}

**A fájlok méretének korlátozása**

A PHP beállításaiban korlátozhatjuk a feltöltött fájl maximális méretét a php.ini fájlban a következő irányelvek segítségével:

* upload\_max\_filesize: A feltöltött fájlok maximális mérete.
* post\_max\_size: A POST adatok maximális mérete (beleértve a fájlokat).

Például:

ini fájlban:

upload\_max\_filesize = 2M

post\_max\_size = 8M

**Biztonsági megfontolások fájlfeltöltésnél:**

1. **Fájl típusa és kiterjesztése ellenőrzése**: Mielőtt elfogadnánk a fájlokat, mindig ellenőrizzük a fájl típusát és kiterjesztését. Ezt megtehetjük a **$\_FILES['fileToUpload']['type']** vagy a **pathinfo()** függvény segítségével.

Például csak képfájlokat engedélyezhetünk:

$fileType = mime\_content\_type($\_FILES['fileToUpload']['tmp\_name']);

if ($fileType != "image/jpeg" && $fileType != "image/png") {

echo "Csak JPEG és PNG fájlokat tölthet fel!";

exit();

}

1. **Fájlnevek kezelése**: Ügyelni kell arra, hogy a fájl neve ne tartalmazzon veszélyes karaktereket (pl. speciális karakterek vagy útvonalak). Használhatjuk a basename() függvényt a fájl nevének biztonságos kezelésére.

$fileName = basename($\_FILES['fileToUpload']['name']);

1. **Az ideiglenes fájl eltávolítása**: Ha nem helyezzük át a feltöltött fájlt, az ideiglenes fájl automatikusan törlődik a szkript futásának végén. Azonban ha valamilyen hiba történik, manuálisan is törölhetjük az ideiglenes fájlt a unlink() függvénnyel.

A **$\_FILES** szuperglobális egy hatékony eszköz a PHP-ban a fájlok kezelésére. Lehetővé teszi, hogy a felhasználók fájlokat töltsenek fel a szerverre, és ezeket az adatokat könnyen kezelhessük, feldolgozhassuk és áthelyezhessük a kívánt helyre. A fájlfeltöltés során azonban különös figyelmet kell fordítani a biztonsági megfontolásokra, például a fájl típusának és méretének ellenőrzésére.

## 6.7 $\_SERVER

A **$\_SERVER** szuperglobális változó számos olyan információt tartalmaz, amelyek a szerverrel, az aktuális kéréssel és a környezeti változókkal kapcsolatosak. Ez egy asszociatív tömb, amely hozzáférést biztosít a szerver által feldolgozott HTTP kérés részleteihez. Ezek az információk hasznosak lehetnek például a kérési metódusok megértéséhez, a felhasználó által látogatott oldal meghatározásához, vagy a kiszolgáló környezetének lekérdezéséhez.

**A $\_SERVER tömb kulcsai és gyakran használt elemei:**

**1. $\_SERVER['REQUEST\_METHOD']:**

Ez a kulcs megadja, hogy milyen módszerrel érkezett a kérés a szerverhez, például GET, POST, PUT, vagy DELETE. Általában a GET és POST a leggyakrabban használt HTTP módszerek a webalkalmazásokban.

echo "A kérés módszere: " . $\_SERVER['REQUEST\_METHOD'];

* Ha a kérés GET módszerrel érkezett, akkor az eredmény: A kérés módszere: GET
* Ha a kérés POST módszerrel érkezett, akkor az eredmény: A kérés módszere: POST

**2. $\_SERVER['SERVER\_NAME']:**

Ez a kulcs megadja a szerver nevét, amely az aktuális szkriptet futtatja. Ez általában a domain nevet jelenti, például example.com.

echo "A szerver neve: " . $\_SERVER['SERVER\_NAME'];

* Eredményként a szerver neve jelenik meg, például: A szerver neve: example.com.

**3. $\_SERVER['SCRIPT\_NAME']:**

Ez a kulcs megadja a futtatott PHP szkript elérési útját a szerveren. Ez különösen hasznos lehet, ha dinamikusan szeretnénk meghatározni a jelenlegi szkript helyét vagy URL-jét.

echo "A szkript neve: " . $\_SERVER['SCRIPT\_NAME'];

* Eredmény: A szkript neve: /folder/file.php, ami az aktuális szkript elérési útját jelzi.

**4. $\_SERVER['REMOTE\_ADDR']:**

Ez a kulcs a kliens (felhasználó) IP-címét adja meg, amely kapcsolatba lépett a szerverrel. Ez hasznos lehet például a felhasználói tevékenységek nyomon követéséhez vagy biztonsági ellenőrzésekhez.

echo "A felhasználó IP címe: " . $\_SERVER['REMOTE\_ADDR'];

* Eredmény: A felhasználó IP címe: 192.168.1.1, vagy más IP-cím, amely a felhasználót azonosítja.

**5. $\_SERVER['HTTP\_USER\_AGENT']:**

Ez a kulcs a felhasználó böngészőjéről és eszközéről tartalmaz információkat. Az HTTP\_USER\_AGENT általában egy hosszú karakterlánc, amely leírja a felhasználó által használt böngészőt, az operációs rendszert, és egyéb releváns adatokat.

echo "A böngésző adatai: " . $\_SERVER['HTTP\_USER\_AGENT'];

* Eredmény: A böngésző adatai: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36
* Ez az adat segítséget nyújt abban, hogy megértsük, milyen böngészőt és operációs rendszert használ a felhasználó.

**6. $\_SERVER['SERVER\_PORT']:**

Ez a kulcs azt a portot adja meg, amelyen a szerver a kérést fogadta. Például a legtöbb webhely alapértelmezett portja a 80-as (HTTP) vagy a 443-as (HTTPS).

echo "A szerver portja: " . $\_SERVER['SERVER\_PORT'];

* Eredmény: A szerver portja: 80, ha a kérés HTTP-n keresztül érkezik, vagy A szerver portja: 443, ha HTTPS-en keresztül.

**7. $\_SERVER['REQUEST\_URI']:**

Ez a kulcs a teljes URI-t tartalmazza, amelyet a felhasználó kért. Ez magában foglalja az útvonalat és az esetleges kérés paramétereit is. Ez hasznos lehet, ha dinamikusan szeretnénk dolgozni az URL-ben található adatokkal.

echo "A kérés URI-je: " . $\_SERVER['REQUEST\_URI'];

* Eredmény: A kérés URI-je: /products?category=books&item=123, amely a kért oldal teljes URI-ját jelzi.

**8. $\_SERVER['HTTP\_REFERER']:**

Ez a kulcs megadja annak az oldalnak az URL-jét, amelyről a felhasználó érkezett az aktuális oldalra. Ez a referer URL hasznos lehet statisztikák gyűjtésére vagy átirányítások kezelésére.

echo "Honnan érkezett a felhasználó: " . $\_SERVER['HTTP\_REFERER'];

* Eredmény: Honnan érkezett a felhasználó: https://example.com/previous-page, ami az előző oldal URL-jét jelzi.

**Biztonsági megfontolások:**

* **Adatvalidálás**: Bár a **$\_SERVER** szuperglobális számos hasznos adatot szolgáltat, mindig ellenőrizni és validálni kell az adatokat, különösen, ha külső forrásokból (pl. HTTP\_USER\_AGENT, HTTP\_REFERER) származnak. Ezeket az információkat a felhasználók manipulálhatják, ezért fontos, hogy ne bízzunk meg bennük vakon.
* **IP-címek és user agent**: Az IP-címek és a böngészők adatai hasznosak lehetnek, de nem mindig megbízhatóak, mivel proxyk és VPN-ek használatával a felhasználók elrejthetik valódi IP-jüket vagy megváltoztathatják a böngészőjük azonosítóját.

A **$\_SERVER** szuperglobális sokféle információt tartalmaz a szerverről és az aktuális kérésről, így fontos eszköz a szerver állapotának és a felhasználók tevékenységeinek megfigyelésére. A kérés módszere, a szerver neve, a szkript útvonala, a felhasználói IP-cím és a böngésző adatai mind könnyen hozzáférhetők a **$\_SERVER** változóval, de fontos figyelembe venni a biztonsági szempontokat és az adatok megfelelő validálását.

**Összefoglalás**

* **$\_GET**: URL-paraméterek kezelésére.
* **$\_POST**: Űrlapok POST metódusával küldött adatok kezelésére.
* **$\_SESSION**: Munkamenet alapú adatokat tárol több oldalon keresztül.
* **$\_COOKIE**: Kliens oldalon tárolt sütik kezelésére.
* **$\_REQUEST**: GET, POST, és COOKIE adatok egyesített elérésére.
* **$\_FILES**: Feltöltött fájlok kezelésére.
* **$\_SERVER**: Szerver és kérés információinak lekérésére.

A PHP szuperglobálisai hatékony eszközök a webes alkalmazásokban az adatok kezelésére és továbbítására.

# Formuláris adatfeldolgozás: Formok kezelése, input validáció.

**Formok létrehozása:** A formok lehetővé teszik a felhasználók számára, hogy adatokat küldjenek a szerverre. Az alábbi példában egy egyszerű formot hozunk létre, amely egy nevet és egy email címet kér be.

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

<label for="name">Név:</label>

<input type="text" id="name" name="name">

<label for="email">Email:</label>

<input type="email" id="email" name="email">

<input type="submit" value="Küldés">

</form>

* **action attribútum**: Meghatározza a cél URL-t, ahol az űrlap adatai elküldésre kerülnek.
* **method attribútum**: Meghatározza, hogy az adatokat GET vagy POST metódussal küldjük el. A POST metódus biztonságosabb, mivel az adatokat nem jeleníti meg az URL-ben.

## ****Adatok feldolgozása:****

Az űrlap által beküldött adatokat a process.php fájl kezeli. A $\_POST szuperglobális változó tartalmazza a POST metódussal elküldött adatokat.

<?php

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

// Adatok lekérése

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

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

// Adatok megjelenítése

echo "Név: " . htmlspecialchars($name) . "<br>";

echo "Email: " . htmlspecialchars($email);

}

?>

* **$\_SERVER["REQUEST\_METHOD"]**: Ellenőrizzük, hogy a kérés POST metódussal történt-e.
* **htmlspecialchars()**: Megakadályozza, hogy a felhasználó által megadott HTML kód futtatható legyen a böngészőben, így elkerülve az XSS támadásokat.

## Input Validáció

**Kötelező mezők ellenőrzése:** A kötelező mezők ellenőrzése biztosítja, hogy a felhasználó ne küldjön üres mezőket.

if (empty($\_POST['name'])) {

echo "A név mező kitöltése kötelező!";

}

**Email cím ellenőrzése:** Az email cím formátumának ellenőrzéséhez a filter\_var() függvényt használjuk.

if (!filter\_var($\_POST['email'], FILTER\_VALIDATE\_EMAIL)) {

echo "Érvénytelen email cím!";

}

* **FILTER\_VALIDATE\_EMAIL**: Ellenőrzi, hogy az email cím megfelelő formátumban van-e.

**HTML karakterek eltávolítása:** A felhasználók által megadott adatokat mindig kezeljük biztonságosan, hogy elkerüljük a potenciális XSS támadásokat.

$name = htmlspecialchars($\_POST['name']);

* **htmlspecialchars()**: Átkonvertálja a speciális HTML karaktereket (pl. <, >, &) HTML entitásokra.

**Adattípusok ellenőrzése:** Ha bizonyos típusú adatokat várunk, például számokat, ellenőrizzük azok típusát.

if (!is\_numeric($\_POST['age'])) {

echo "A kor mező csak számokat tartalmazhat!";

}

* **is\_numeric()**: Ellenőrzi, hogy az érték számként értelmezhető-e.

## Formok biztonsága

**CSRF védelem:** A Cross-Site Request Forgery (CSRF) támadások elkerülésére CSRF tokeneket használunk. Ez egy véletlenszerűen generált, egyedi azonosító, amelyet az űrlaphoz adunk, és a kiszolgálón is ellenőrizzük.

// Token generálása

session\_start();

$\_SESSION['csrf\_token'] = bin2hex(random\_bytes(32));

// Token hozzáadása a formhoz

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

<input type="hidden" name="csrf\_token" value="<?php echo $\_SESSION['csrf\_token']; ?>">

<!-- további mezők -->

</form>

* **bin2hex(random\_bytes(32))**: Generál egy 64 karakter hosszú hexadecimális token-t.
* **session\_start()**: Indítja a PHP session-t, amely lehetővé teszi a token tárolását.

Az űrlap adatainak feldolgozása során ellenőrizzük a token érvényességét:

session\_start();

if ($\_POST['csrf\_token'] !== $\_SESSION['csrf\_token']) {

die("Érvénytelen CSRF token!");

}

## ****SQL Injection elkerülése:****

Az SQL injection támadások elkerülésére használjunk előkészített utasításokat. Ez biztosítja, hogy a felhasználói adatok nem változtatják meg az SQL lekérdezések struktúráját.

// Adatbázis kapcsolat létrehozása

$pdo = new PDO('mysql:host=localhost;dbname=test', 'root', '');

// Előkészített utasítás használata

$stmt = $pdo->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");

$stmt->bindParam(':name', $name);

$stmt->bindParam(':email', $email);

// Adatok beszúrása

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

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

$stmt->execute();

* **PDO::prepare()**: Előkészíti az SQL lekérdezést.
* **bindParam()**: Hozzárendeli a változókat a lekérdezés paramétereihez.

## Összetett űrlap feldolgozási feladat:

 **HTML Űrlap**:

* Készítsd el a HTML űrlapot a megadott mezőkkel.
* Használj megfelelő input típusokat (text, tel, email, radio, checkbox) a formában.
* Adj meg validációs mintákat a telefon mezőhöz és biztosítsd, hogy az email cím érvényes formátumban legyen.
* JavaScript segítségével végezz frontend validációt: ellenőrizd, hogy minden mező ki van-e töltve, a telefonszám helyes formátumban van-e, az email cím érvényes-e, a felhasználó választott-e tanfolyamot, és elfogadta-e a felhasználási feltételeket.

 **PHP Feldolgozás**:

* Ellenőrizd, hogy a form adatai POST metódussal érkeztek-e.
* Sanitize-eld a beérkező adatokat HTML karakterek eltávolításával.
* Végezze el a backend validációt: ellenőrizd az űrlap mezőit, hogy megfelelnek-e a követelményeknek.
* Ha bármelyik validációs teszt megbukik, jeleníts meg megfelelő hibaüzenetet.
* Ha az adatok helyesek, írd ki a feldolgozott adatokat.

## Implode() függvény használata

Az implode() egy hasznos PHP függvény, amely tömb elemeit egy sztringgé (karakterlánccá) egyesíti. A függvény megadja, hogy milyen elválasztó karaktert (például vesszőt, szóközt vagy más karaktert) szeretnénk használni a tömb elemei között a sztringben. Ez gyakran hasznos például checkboxok vagy többválasztásos elemek feldolgozásakor, ahol a felhasználó több elemet is választhat.

**Szintaxis:**

string implode ( string $separator , array $array )

* **$separator**: Ez az az elválasztó karakter vagy karakterlánc, amit a tömb elemei közé helyezünk.
* **$array**: A tömb, amelynek elemeit szeretnénk egyesíteni.

**Példa:**

Tegyük fel, hogy van egy tömbünk, ami tartalmazza a felhasználó által választott tananyagokat:

php

Kód másolása

$tananyagok = ["HTML", "CSS", "JavaScript"];

Ha ezt a tömböt vesszővel elválasztva szeretnénk egy sztringgé alakítani, az alábbi módon használhatjuk az implode() függvényt:

$eredmeny = implode(", ", $tananyagok);

echo $eredmeny;

Ez kiírja: HTML, CSS, JavaScript

**Magyarázat:**

1. **Tömb elemeinek egyesítése**: Az implode() függvény a tömb minden egyes elemét egyesíti egy sztringbe, és az elemek közé beteszi a megadott elválasztót.
2. **Rugalmasság**: Az elválasztó karakter lehet bármi, amit meg szeretnél adni, például vessző, szóköz, pont vagy akár szavak is.
3. **Üres tömb kezelése**: Ha a tömb üres, az implode() egy üres sztringet ad vissza.

**További példák:**

#### Szóközzel elválasztva:

$eredmeny = implode(" ", $tananyagok);

echo $eredmeny;

Eredmény: HTML CSS JavaScript

#### Ponttal elválasztva:

$eredmeny = implode(". ", $tananyagok);

echo $eredmeny;

Eredmény: HTML. CSS. JavaScript

#### Elválasztó nélkül:

$eredmeny = implode("", $tananyagok);

echo $eredmeny;

Eredmény: HTMLCSSJavaScript

**Példa az űrlapok feldolgozásában:**

Amikor több checkboxot használsz egy űrlapban, a PHP tömbként kezeli azokat. Az implode() ilyenkor segít a kiválasztott elemek egyesítésében és megjelenítésében. Például:

if (isset($\_POST['anyag']) && is\_array($\_POST['anyag'])) {

$anyagok = implode(", ", $\_POST['anyag']);

echo "A választott tananyagok: $anyagok";

} else {

echo "Nem választottál tananyagot!";

}

Ez esetben, ha a felhasználó HTML-t és PHP-t választ, a kimenet a következő lesz:

A választott tananyagok: HTML, PHP

**Összegzés:**

Az implode() függvény rendkívül hasznos, ha tömb elemeit szeretnéd egyszerűen, tetszőleges elválasztóval egy sztringbe fűzni. A tömbben lévő elemek rendezésére, megjelenítésére és más hasonló műveletekre nagyszerű eszköz.

**Feladat**

Hozz létre egy űrlapot, amelyben a felhasználó megadhatja a nevét, választhat egy napot rádiógombokkal, több tananyagot checkboxokkal, és kiválaszthat egy csoportot egy legördülő menü segítségével. Az űrlap adatainak feldolgozásához PHP-t használunk, amely ellenőrzi a bevitt adatokat, és visszajelzést ad a felhasználónak.

1. **HTML űrlap létrehozása**:
   * Hozz létre egy HTML oldalt, amely az alábbi elemeket tartalmazza:
     + Egy szövegmezőt a név megadásához.
     + Két rádiógombot, amelyekkel a felhasználó a napok közül választhat (Hétfő, Kedd).
     + Két checkboxot, amelyekkel a felhasználó a tananyagok közül választhat (HTML, PHP).
     + Egy legördülő menüt, amelyben csoportot választhat (Első, Második).
   * Az űrlap az adatokat egy PHP fájlhoz küldje POST metódussal.
2. **Adatok feldolgozása PHP-ben**:
   * A PHP fájl ellenőrizze, hogy a névmező kitöltött-e.
   * Ellenőrizze, hogy a napok közül választott-e a felhasználó, és a megfelelő napot jelenítse meg.
   * A tananyagok közül választhat több opciót is a felhasználó; ezeket egy sztringben listázd ki.
   * Ellenőrizze, hogy a csoportot választotta-e, és adjon visszajelzést a kiválasztott csoport alapján.

#### Részletes követelmények:

1. **HTML oldalon:**
   * A név mező legyen kötelezően kitöltendő.
   * A rádiógombokkal biztosítsd, hogy csak egy napot lehessen kiválasztani.
   * A checkboxok segítségével több tananyag is választható.
   * A legördülő menü első eleme a „Válassz csoportot” legyen, ami nem választható érvényes opcióként.
2. **PHP feldolgozó fájl:**
   * Ellenőrizze a POST kérelem típusát.
   * Ellenőrizze a név mezőt, és ha üres, adjon visszajelzést („Nem adtad meg a neved!”).
   * Ellenőrizze, hogy választottak-e napot, és írja ki a választott nap nevét („Hétfő” vagy „Kedd”).
   * Ellenőrizze, hogy legalább egy tananyagot kiválasztottak-e, és listázza ki azokat.
   * A csoport kiválasztását switch-el kezelje, és adjon visszajelzést az első vagy második csoport választása esetén.

**Példa kimenet:**

Amennyiben a felhasználó a „Hétfő” napot, „HTML” tananyagot és az első csoportot választja, a következő kimenetet várjuk:

**Neved: Példa Név**

**A választott nap: Hétfő**

**Választott tananyagok: HTML**

**Az első csoportot választottad.**

Ha valamilyen adat hiányzik, a megfelelő hibaüzenet jelenik meg.

#### Tesztelés:

* Töltsd ki az űrlapot különböző kombinációkkal, és ellenőrizd a PHP kimenetét.
* Próbáld ki, mi történik, ha nem adsz meg nevet vagy nem választasz csoportot.

**Megoldások:**

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Adatbekérő űrlap</title>

    <style>

        body {

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            margin: 0;

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

        }

        form {

            background-color: #fff;

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

        }

        input[type="text"],

        input[type="email"],

        input[type="tel"] {

            width: 100%;

            padding: 10px;

            margin: 10px 0;

            border: 1px solid #ccc;

            border-radius: 5px;

        }

        input[type="submit"] {

            background-color: #4CAF50;

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        input[type="submit"]:hover {

            background-color: #45a049;

        }

        .course-options {

            margin: 10px 0;

        }

    </style>

</head>

<body>

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

    <label for="name">Név (kötelező):</label>

    <input type="text" id="name" name="name" required>

    <label for="email">Email (kötelező):</label>

    <input type="email" id="email" name="email" required>

    <label for="phone">Telefonszám (kötelező):</label>

    <input type="tel" id="phone" name="phone" required>

    <fieldset class="course-options">

        <legend>Válasszon tanfolyamot:</legend>

        <input type="radio" id="frontend" name="course" value="Frontend" required>

        <label for="frontend">Frontend</label><br>

        <input type="radio" id="backend" name="course" value="Backend">

        <label for="backend">Backend</label><br>

        <input type="radio" id="fullstack" name="course" value="Fullstack">

        <label for="fullstack">Fullstack</label><br>

    </fieldset>

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

</form>

</body>

</html>

**form.php**

<?php

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

    // *Kötelező mezők ellenőrzése*

    $name = htmlspecialchars($\_POST['name']);

    $email = filter\_var($\_POST['email'], FILTER\_VALIDATE\_EMAIL);

    $phone = htmlspecialchars($\_POST['phone']);

    $course = htmlspecialchars($\_POST['course']);

    if ($name && $email && $phone && $course) {

        echo "<h2>Beküldött adatok:</h2>";

        echo "Név: " . $name . "<br>";

        echo "Email: " . $email . "<br>";

        echo "Telefonszám: " . $phone . "<br>";

        echo "Választott tanfolyam: " . $course . "<br>";

    } else {

        echo "Kérjük, töltse ki az összes kötelező mezőt!";

    }

/\**} else {*

*echo "Hibás kérés!";*

*}*\*/

**Űrlap feldolgozása AngularJs használatával:**

<!DOCTYPE html>

<html lang="hu" ng-app="formApp"> <!-- *ng-app: Ez az AngularJS alkalmazás inicializálása. Az "formApp" az alkalmazás modulja* -->

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <link rel="stylesheet" href="css/style.css">

    <title>Adatbekérő űrlap</title>

</head>

<body ng-controller="FormController"> <!-- *ng-controller: A "FormController" vezérlőt társítja az aktuális nézethez* -->

    <!-- *Ha az űrlap nem lett beküldve* -->

    <div ng-if="!formSubmitted"> <!-- *ng-if: Feltételes megjelenítés. Ha a "formSubmitted" false, akkor ez a rész jelenik meg* -->

        <!-- *Az űrlap a következő AngularJS direktívákkal* -->

        <form id="userForm" method="post" enctype="multipart/form-data" ng-submit="submitForm()" novalidate> <!-- *ng-submit: Az űrlap beküldésekor a submitForm() függvény kerül meghívásra* -->

            <label for="name">Név (kötelező):</label>

            <!-- *ng-model: A formData objektumban tárolt név változóhoz köti az input értékét* -->

            <input type="text" name="name" id="name" ng-model="formData.name" required>

            <label for="email">Email (kötelező):</label>

            <!-- *ng-model: A formData objektumban tárolt email változóhoz köti az input értékét* -->

            <input type="email" name="email" id="email" ng-model="formData.email" required>

            <label for="phone">Telefonszám (kötelező, legalább 8 karakter):</label>

            <!-- *ng-model: A formData objektumban tárolt phone változóhoz köti az input értékét* -->

            <input type="tel" name="phone" id="phone" ng-model="formData.phone" required minlength="8">

            <fieldset>

                <legend>Válasszon tanfolyamot:</legend>

                <!-- *ng-model: A formData objektumban tárolt course változóhoz köti az input értékét* -->

                <input type="radio" id="frontend" name="course" value="Frontend" ng-model="formData.course" required>

                <label for="frontend">Frontend</label><br>

                <input type="radio" id="backend" name="course" value="Backend" ng-model="formData.course" required>

                <label for="backend">Backend</label><br>

                <input type="radio" id="fullstack" name="course" value="Fullstack" ng-model="formData.course" required>

                <label for="fullstack">Fullstack</label><br>

            </fieldset>

            <label for="photo">Fotó feltöltése (nem kötelező):</label>

            <input type="file" name="photo" id="photo">

            <!-- *ng-submit: A form elküldésekor a submitForm() függvény hívódik meg* -->

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

            <div class="error">{{ error }}</div> <!-- *{{ error }}: AngularJS interpoláció a hibák megjelenítéséhez* -->

        </form>

    </div>

    <!-- *Ha az űrlap már beküldésre került* -->

    <div ng-if="formSubmitted"> <!-- *ng-if: Ha a "formSubmitted" true, akkor ez a rész jelenik meg* -->

        <h2>Beküldött adatok:</h2>

        <!-- *Az AngularJS interpolációs szintaxisa: {{ expression }}* -->

        <p>Név: {{ submissionResult.name }}</p> <!-- *{{ submissionResult.name }}: A beküldött név megjelenítése* -->

        <p>Email: {{ submissionResult.email }}</p> <!-- *{{ submissionResult.email }}: A beküldött email megjelenítése* -->

        <p>Telefonszám: {{ submissionResult.phone }}</p> <!-- *{{ submissionResult.phone }}: A beküldött telefonszám megjelenítése* -->

        <p>Választott tanfolyam: {{ submissionResult.course }}</p> <!-- *{{ submissionResult.course }}: A kiválasztott tanfolyam megjelenítése* -->

        <div ng-if="submissionResult.photo"> <!-- *ng-if: Ha van feltöltött fotó, ez a rész jelenik meg* -->

            <p>Feltöltött fotó:</p>

            <!-- *ng-src: AngularJS által biztosított direktíva a képek helyes betöltéséhez, a {{ submissionResult.photo }} az URL-t adja meg* -->

            <img ng-src="{{ submissionResult.photo }}" alt="Feltöltött fotó" style="max-width:200px;">

        </div>

        <div ng-if="!submissionResult.photo"> <!-- *ng-if: Ha nincs feltöltött fotó, ez a rész jelenik meg* -->

            <p>Nem lett fotó feltöltve.</p>

        </div>

    </div>

    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

    <script src="form.js"></script>

</body>

</html>

**form.js fájl:**

// *AngularJS modul létrehozása*

const app = angular.module('formApp', []);

app.controller('FormController', ['$scope', '$http', function($scope, $http) {

    // *Az adatok inicializálása*

    $scope.formData = {};

    $scope.error = '';

    $scope.submissionResult = {};

    $scope.formSubmitted = false; // *Az űrlap beküldésének állapota*

    // *Az űrlap adatainak küldése a PHP fájlnak*

    $scope.submitForm = function() {

        // *Hibák törlése*

        $scope.error = '';

        $scope.submissionResult = {};  // *Töröljük az előző eredményeket*

        // *Kötelező mezők ellenőrzése*

        if (!$scope.formData.name || !$scope.formData.email || !$scope.formData.phone || !$scope.formData.course) {

            alert('Kérjük, töltse ki az összes kötelező mezőt!');

            return;  // *Megakadályozza az űrlap elküldését, ha bármelyik kötelező mező üres*

        }

        // *Adatküldés a PHP backend felé*

        const formData = new FormData();

        formData.append('name', $scope.formData.name);

        formData.append('email', $scope.formData.email);

        formData.append('phone', $scope.formData.phone);

        formData.append('course', $scope.formData.course);

       // *Fotó feltöltésének ellenőrzése és hozzáadása, ha van*

       if (document.getElementById('photo').files.length > 0) {

        formData.append('photo', document.getElementById('photo').files[0]);

    }

        console.log(formData); //*itt nem látjuk az adatokat*

        /\*\**Az entries() egy metódus a JavaScript-ben, amely bizonyos beépített objektumok, például a FormData, Map, vagy Array esetén elérhető. Ez a metódus egy iterátort ad vissza, amely tartalmazza az objektum összes kulcs-érték párját, így lehetővé teszi az elemek végigjárását.*

*\* Ezzel tudjuk tesztelni, hogy milyen adatokat küldünk tovább a php-nek*

\*/

        for (let i of formData.entries()) {

            console.log(i[0] + ': ' + i[1]);

        }

       /\* *for (let [key, value] of formData.entries()) {*

*console.log(key, value); // Ellenőrizd az adatokat a konzolon*

*}*\*/

        $http.post('form.php', formData, {

            transformRequest: angular.identity,

            headers: { 'Content-Type': undefined }

        })

        .then(function(response) {

            //*ell.hogy mit küldünk el*

            console.log('Válasz:', response);

            $scope.submissionResult = response.data;

            $scope.formSubmitted = true; // *Az űrlap sikeres beküldésének jelzése*

        }, function(error) {

            $scope.error = 'Hiba történt az adatok küldésekor: ' + error.status + ' - ' + error.statusText;

        });

    };

}])

**form.php**

<?php

// *Adatok kezelésére szolgáló osztály*

class FormDataHandler {

/\* *A private $name; sor az osztályon belüli $name tulajdonság deklarációját jelöli.*

*private: Ez a láthatósági módosító, amely azt jelzi, hogy a $name tulajdonság csak az adott osztályon belül érhető el. Más osztályok és külső kód nem férhet hozzá közvetlenül ehhez a tulajdonsághoz. Ez segít megvédeni az osztály állapotát, és biztosítja, hogy az adatokat csak az osztály metódusain keresztül lehessen módosítani, ami növeli a kód biztonságát és karbantarthatóságát.* \*/

    private $name;      // *Felhasználó neve*

    private $email;     // *Felhasználó email címe*

    private $phone;     // *Felhasználó telefonszáma*

    private $course;    // *Választott tanfolyam*

    private $photo;     // *Feltöltött fénykép*

      // *Névrendező függvény: a név első betűje nagybetűs, a többi kisbetűs*

      private function formatName($name) {

        return ucwords(strtolower($name));

    }

    // *Konstruktor, amely inicializálja az osztály tulajdonságait*

    public function \_\_construct($name, $email, $phone, $course, $photo) {

/\* *$this->name: Az osztály egy tulajdonsága, amely a felhasználó nevét tárolja.*

*$this->formatName($name): Egy privát metódus, amely a bemeneti $name értéket formázza a kívánt módon (pl. első betű nagybetűs, a többi kisbetűs).*

*$name: A konstruktor paramétere, amelyet a felhasználó ad meg, és amelyet formázni szeretnénk.*

*Nevet formázunk: első betű nagybetűs és a többi kisbetűs*\*/

        $this->name = $this->formatName($name);

        // *Email címet kisbetűssé alakítjuk*

        $this->email = strtolower($email);

        // *Telefonszámot HTML entitásokra kódoljuk, hogy biztonságos legyen*

        $this->phone = htmlspecialchars($phone);

        // *Tanfolyamot HTML entitásokra kódoljuk, hogy elkerüljük a XSS támadásokat*

        $this->course = htmlspecialchars($course);

        // *Fényképet eltároljuk*

        $this->photo = $photo;

    }

    // *Az űrlap feldolgozása és válasz készítése*

    public function processForm() {

        $response = [];  // *Üres válasz tömb inicializálása*

        // *Ellenőrizzük, hogy minden kötelező mező ki van-e töltve*

        if ($this->name && $this->email && $this->phone && $this->course) {

            // *Ha minden adat megvan, hozzáadjuk a választ tartalmához*

            $response['name'] = $this->name;

            $response['email'] = $this->email;

            $response['phone'] = $this->phone;

            $response['course'] = $this->course;

            // *Ha van fotó és nincs feltöltési hiba*

            if ($this->photo && $this->photo['error'] === UPLOAD\_ERR\_OK) {

                // *Feltöltjük a fotót és elmentjük az URL-t*

                $photoUrl = $this->uploadPhoto();

                $response['photo'] = $photoUrl;  // *Hozzáadjuk az URL-t a válaszhoz*

            } else {

                // *Ha nincs fotó, vagy hiba történt, null értéket adunk*

                $response['photo'] = null;

            }

        } else {

            // *Ha bármely kötelező mező hiányzik*

            $response['error'] = "Kérjük, töltse ki az összes kötelező mezőt!";

        }

        // *JSON válasz visszaküldése*

        header('Content-Type: application/json');  // *Beállítjuk a válasz típusát JSON-ra*

        echo json\_encode($response);  // *JSON formátumban visszaadjuk a választ*

    }

    // *Fénykép feltöltésének kezelése*

    private function uploadPhoto() {

        $uploadDir = 'uploads/';  // *Feltöltési könyvtár*

        // *Ha a könyvtár nem létezik, létrehozzuk*

        if (!is\_dir($uploadDir)) {

            mkdir($uploadDir, 0777, true);  // *Könyvtár létrehozása az összes szükséges szülőkönyvtárral*

        }

        // *Fénykép egyedi névvel ellátása*

        $photoName = uniqid() . '\_' . basename($this->photo['name']);

        $destination = $uploadDir . $photoName;  // *Célmappa elérési útja*

        // *Fénykép áthelyezése a célkönyvtárba*

        if (move\_uploaded\_file($this->photo['tmp\_name'], $destination)) {

            return $destination;  // *Ha sikerült a feltöltés, visszaadjuk az elérési utat*

        } else {

            return null;  // *Hiba esetén null értéket adunk vissza*

        }

    }

}

// *POST kérés feldolgozása*

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

    // *Az űrlap adatait a POST változóból nyerjük ki*

    $name = $\_POST['name'];

    $email = $\_POST['email'];

    $phone = $\_POST['phone'];

    $course = $\_POST['course'];

    // *Ha van fotó, akkor azt a FILES tömbből nyerjük ki*

    $photo = isset($\_FILES['photo']) ? $\_FILES['photo'] : null;

    // *Létrehozzuk az adatkezelő példányát és feldolgozzuk az űrlapot*

    $formHandler = new FormDataHandler($name, $email, $phone, $course, $photo);

    $formHandler->processForm();

} else {

    // *Ha nem POST kérés érkezik*

    echo "Hibás kérés!";  // *Jelzi, hogy nem POST kérést kaptunk*

}

**form.test.php**

<?php

// *A tesztek eredményeinek tárolása*

$testResults = [];

// *Egyszerű tesztellenőrző függvény*

/\**assertEqual függvény: Ez a függvény ellenőrzi, hogy az expected és actual értékek megegyeznek-e. Ha igen, sikeres tesztet jelöl, különben sikertelen tesztet, és megjeleníti az elvárt és tényleges értékeket.*\*/

function assertEqual($expected, $actual, $testName) {

    global $testResults;

    if ($expected === $actual) {

        $testResults[] = " $testName: Teszt sikeres!";

    } else {

        $testResults[] = "$testName: Sikertelen teszt. Várt érték: '$expected', tényleges: '$actual'";

    }

}

// *Include-oljuk az osztályt tartalmazó fájlt*

include\_once 'form.php';

// *1. teszt: Név formázása*

/\**Itt létrehozol egy új FormDataHandler példányt a new kulcsszóval, és inicializálod azt néhány bemeneti értékkel. Az itt megadott adatok a FormDataHandler osztály konstruktorába kerülnek.*

*Ez a sor a ReflectionClass osztályt használja a FormDataHandler osztály metódusainak elérésére:*

*new ReflectionClass('FormDataHandler') létrehoz egy ReflectionClass példányt a FormDataHandler osztályról.*

*getMethod('formatName') lekéri a formatName nevű metódust az osztályból.*

*invokeArgs($formHandler, ['john doe']) meghívja a formatName metódust a korábban létrehozott $formHandler objektumon, és átadja neki a ['john doe'] argumentumot. A invokeArgs lehetővé teszi a metódus dinamikus meghívását és argumentumok átadását.*

*Miért használjuk a ReflectionClass-t?*

*Privát és védett metódusok tesztelése: A ReflectionClass lehetővé teszi, hogy hozzáférjünk és teszteljük az osztály privát és védett metódusait, amelyeket egyébként nem érhetnénk el közvetlenül.*

*Dinamikus metódushívás: Ha dinamikusan akarjuk meghívni a metódusokat, például a metódus nevét csak futási időben tudjuk, a ReflectionClass segítségünkre lehet.*

\*/

$formHandler = new FormDataHandler("john doe", "test@example.com", "123456789", "frontend", null);

$formattedName = (new ReflectionClass('FormDataHandler'))->getMethod('formatName')->invokeArgs($formHandler, ['john doe']);

assertEqual("John Doe", $formattedName, "Név formázása");

// *2. teszt: Helyes adatok feldolgozása*

$photo = ['error' => UPLOAD\_ERR\_OK, 'name' => 'test.jpg', 'tmp\_name' => '/tmp/test.jpg'];

$formHandler = new FormDataHandler("John Doe", "test@example.com", "123456789", "frontend", $photo);

ob\_start(); // *Kimenet elkapása*

$formHandler->processForm();

$output = ob\_get\_clean();

$response = json\_decode($output, true);

assertEqual("John Doe", $response['name'], "Név feldolgozása");

assertEqual("test@example.com", $response['email'], "Email feldolgozása");

assertEqual("123456789", $response['phone'], "Telefonszám feldolgozása");

assertEqual("frontend", $response['course'], "Tanfolyam feldolgozása");

assertEqual(true, isset($response['photo']), "Fénykép feldolgozása");

// *3. teszt: Hiányzó mezők tesztelése*

$formHandler = new FormDataHandler("", "test@example.com", "123456789", "frontend", null);

ob\_start(); // *Kimenet elkapása*

$formHandler->processForm();

$output = ob\_get\_clean();

$response = json\_decode($output, true);

assertEqual("Kérjük, töltse ki az összes kötelező mezőt!", $response['error'], "Hiányzó mezők kezelése");

// *4. teszt: Feltöltési hiba kezelése*

$photo = ['error' => UPLOAD\_ERR\_NO\_FILE];

$formHandler = new FormDataHandler("John Doe", "test@example.com", "123456789", "frontend", $photo);

ob\_start(); // *Kimenet elkapása*

$formHandler->processForm();

$output = ob\_get\_clean();

$response = json\_decode($output, true);

assertEqual(null, $response['photo'], "Feltöltési hiba kezelése");

// *Az eredmények megjelenítése*

echo "<h2>Teszt eredmények:</h2>";

foreach ($testResults as $result) {

    echo $result;

}

?>

# Érzékeny Adatok Biztonságos Kezelése .env Fájlban

A webfejlesztés során gyakran szükség van érzékeny információkra, például adatbázis hitelesítő adatokra, API kulcsokra, és más konfigurációs beállításokra. Ezek az adatok kritikus fontosságúak a webalkalmazások működéséhez, és megfelelő kezelésük elengedhetetlen a biztonságos működéshez.

#### ****Mi az a**** .env ****Fájl?****

A .env fájl egy egyszerű szöveges fájl, amely környezeti változókat tárol. Általában a projekt gyökérkönyvtárában található, és az alkalmazás konfigurációs adatait tartalmazza. A .env fájl nem tartalmaz semmilyen formázást, csak kulcs-érték párokat, például:

DBHOST=localhost

DBUSER=root

DBPASS=

DBNAME=nevjegyek

#### ****Miért Használjunk**** .env ****Fájlt az Érzékeny Adatok Tárolására?****

1. **Biztonságosabb Adatkezelés**: A .env fájl használatával az érzékeny adatokat elválaszthatjuk a kódunk többi részétől. Ez lehetővé teszi, hogy az adatokat külön fájlban tartsuk, amelyet könnyen kezelhetünk és védhetünk.
2. **Egyszerűbb Konfigurációkezelés**: A .env fájl segítségével egyszerűen módosíthatjuk az alkalmazás konfigurációját anélkül, hogy a kódunkat változtatnunk kellene. Ez különösen hasznos, ha különböző környezetekben (fejlesztés, tesztelés, éles) eltérő konfigurációkat használunk.
3. **Könnyebb Verziókezelés**: A .env fájl nem kerül be a verziókezelés alá, ha a .gitignore fájlban szerepel. Ez megakadályozza, hogy érzékeny információk véletlenül nyilvánosságra kerüljenek.
4. **Elkerüli a Hardcoded Értékeket**: Az érzékeny adatok közvetlen kódba írása (hardcoding) növeli a biztonsági kockázatokat. A .env fájl lehetővé teszi, hogy elkerüljük a hardcoded értékeket, és biztonságosabbá tegyük az alkalmazást.

#### ****Hogyan Használjuk a**** .env ****Fájlt?****

1. **Létrehozás**: Készíts egy .env fájlt a projekt gyökérkönyvtárában, és add hozzá az érzékeny adatokat.
2. **Telepítés**: Használj egy könyvtárat, mint például vlucas/phpdotenv, hogy betöltsd a .env fájlban található változókat a PHP alkalmazásodba.

Például, a vlucas/phpdotenv telepítéséhez használhatod a Composer-t:

composer require vlucas/phpdotenv

1. **Betöltés**: A .env fájl betöltése a következő kód segítségével történik:

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

$dotenv = Dotenv\Dotenv::createImmutable(\_\_DIR\_\_);

$dotenv->load();

1. **Használat**: Használhatod a $\_ENV vagy getenv() függvényeket az adatok elérésére:

$dbHost = $\_ENV['DBHOST'];

1. **Git Ignore**: Ne felejtsd el hozzáadni a .env fájlt a .gitignore fájlhoz, hogy elkerüld annak verziókezelés alá kerülését:

.env

#### ****Összegzés****

A .env fájl használata az érzékeny adatok biztonságos kezelésére egy hatékony módszer a konfigurációk kezelésére és a biztonság növelésére. Segít elkerülni a hardcoded értékeket, és biztosítja, hogy a fontos információk ne kerüljenek nyilvánosságra.

**Feladat:** Írj egy adatkapcsolatot tartalmazó php kódot a .env felhasználásával.

**Megoldás:**

.env fájl tartalma

DBHOST=localhost

DBUSER=root

DBPASS=

DBNAME=nevjegyek //-> itt az adatbázisod nevét kell megadni

kapcsolat.php

<?php

//*telepítsük: composer require vlucas/phpdotenv ez létre hoz egy vendor mappát*

//*majd hozzuk létre a .env fájlt a gyökér könyvtárban*

//*.gitignore létrehozása .env tartalommal*

/\**\*a .env fájlt a .gitignore fájlhoz. Ez megakadályozza, hogy a .env fájl a Git verziókezelésébe kerüljön.*\*/

// *Load Composer's autoloader*

require\_once \_\_DIR\_\_ . '/vendor/autoload.php';

// *Load environment variables from the .env file*

$dotenv = Dotenv\Dotenv::createImmutable(\_\_DIR\_\_);

$dotenv->load();

// *HTTP fejléc beállítása a megfelelő tartalomtípussal és karakterkódolással*

header("Content-Type: text/html; charset=utf-8");

// *Adatbázis kapcsolódáshoz szükséges adatok definíciója*

define("DBHOST", $\_ENV['DBHOST']);

define("DBUSER", $\_ENV['DBUSER']);

define("DBPASS", $\_ENV['DBPASS']);

define("DBNAME", $\_ENV['DBNAME']);

// *Adatbázis kapcsolat létrehozása, és az esetleges hibák kezelése*

$dbconn = @mysqli\_connect(DBHOST, DBUSER, DBPASS, DBNAME) or die("Hiba az adatbázis csatlakozásakor!");

// *Karakterkódolás beállítása az adatbáziskapcsolaton keresztül*

mysqli\_query($dbconn, "SET NAMES utf8");

Ha ezt egyszer már megírtad, akkor tárol el, könnyen elérhető helyen, hogy a következő projektednél is fel tudd használni a már megírt kódot. Csak az adatbázisod nevét kell mindig átírni a .env fájlban.

# Adatbázis műveletek a PHP-ben összefoglalás

-- Minden érték lekérése

SELECT \* FROM tablanev;

-- Adott oszlopok lekérése

SELECT oszlop1, oszlop2, oszlop3 FROM tablanev;

-- Az oszlopok nevének körülírása backtick-kel

SELECT `oszlop1`, `oszlop2`, `oszlop3` FROM tablanev;

-- Feltétellel adatlekérés

SELECT \* FROM tablanev WHERE feltetel;

-- Adott oszlopok lekérése feltétellel

SELECT oszlop1, oszlop2, oszlop3 FROM tablanev WHERE feltetel;

-- Keresés LIKE segítségével

SELECT oszlop1, oszlop2, oszlop3 FROM tablanev WHERE oszlop1 LIKE 'kifejezes%'; -- Kezdő kifejezés

SELECT oszlop1, oszlop2, oszlop3 FROM tablanev WHERE oszlop1 LIKE '%kifejezes%'; -- Bármely helyen

-- Rendezés ASC [A-Z-ig], DESC [Z-A-ig]

SELECT \* FROM tablanev ORDER BY oszlop1 ASC; -- Növekvő sorrend

SELECT \* FROM tablanev ORDER BY oszlop1 DESC; -- Csökkenő sorrend

SELECT \* FROM tablanev WHERE feltetel ORDER BY oszlop1 DESC; -- Feltétellel és csökkenő sorrend

SELECT \* FROM tablanev WHERE feltetel ORDER BY oszlop1 DESC, oszlop2 ASC; -- Több oszlop szerinti rendezés

-- Adott darabszám lekérése

SELECT \* FROM tablanev LIMIT 10; -- Az első 10 rekord

SELECT \* FROM tablanev LIMIT 10, 10; -- 10-től kezdődő következő 10 rekord

-- Rekord beszúrása

INSERT INTO tablanev (oszlop1, oszlop2, oszlop3) VALUES ('ertek1', ertek2, ertek3);

-- Rekordok törlése

DELETE FROM tablanev WHERE feltetel;

-- Rekordok módosítása

UPDATE tablanev SET oszlop1 = ertek1, oszlop2 = 'ertek2' WHERE feltetel;

-- Több tábla összekapcsolása JOIN használatával

-- Inner Join: Két tábla közös rekordjait adja vissza

SELECT t1.oszlop1, t2.oszlop2 FROM tabla1 t1

JOIN tabla2 t2 ON t1.kozos\_oszlop = t2.kozos\_oszlop;

-- Left Join: Az első táblából minden rekordot visszaad, és a második táblából csak a megfelelő rekordokat

SELECT t1.oszlop1, t2.oszlop2 FROM tabla1 t1

LEFT JOIN tabla2 t2 ON t1.kozos\_oszlop = t2.kozos\_oszlop;

-- Right Join: A második táblából minden rekordot visszaad, és az első táblából csak a megfelelő rekordokat

SELECT t1.oszlop1, t2.oszlop2 FROM tabla1 t1

RIGHT JOIN tabla2 t2 ON t1.kozos\_oszlop = t2.kozos\_oszlop;

-- Union: Két SELECT eredményének egyesítése

SELECT oszlop1 FROM tabla1

UNION

SELECT oszlop1 FROM tabla2;

# PHP adatbázis műveletek továbbfejlesztése: Kérések optimalizálása, indexek használata.

A PHP alapú alkalmazásoknál az adatbázisműveletek hatékonyságának javítása kulcsfontosságú a teljesítmény optimalizálása érdekében. Itt van néhány lényeges pont a kérdések optimalizálásával és az indexek használatával kapcsolatban:

## ****SQL kérdések optimalizálása:****

* **Minimális adatok lekérése**: Minden SQL lekérdezésben érdemes csak a szükséges adatokat lekérni. Ha nincs szükséged az összes oszlopra, akkor ne használj SELECT \*, hanem kifejezetten határozd meg azokat az oszlopokat, amelyekre szükséged van (SELECT name, email FROM users).
* **WHERE feltételek optimalizálása**: A WHERE feltétel használatával pontosíthatod a lekérdezéseket, így csak a releváns rekordokat kapod meg. Fontos, hogy a WHERE feltételben lévő oszlopokat érdemes indexelni, hogy a lekérdezés gyorsabb legyen.
* **LIMIT használata**: Ha nem kell az összes találat, akkor érdemes LIMIT-et alkalmazni, különösen nagy adatbázisok esetén. Ez segít a memóriaterhelés csökkentésében és gyorsabbá teheti az oldalak betöltését. Pl.:

SELECT name, email FROM users WHERE status = 'active' LIMIT 10;

* **Csoportosítás és aggregálás**: Ha szükséged van összegző műveletekre (pl. átlagok, összeg, csoportosítás), akkor használd a GROUP BY, COUNT(), SUM(), AVG() stb. műveleteket, hogy közvetlenül az adatbázisban számold ki az adatokat, így nem kell utólagos feldolgozást végezned a PHP-ben.
* **Kapcsolt táblák használata (JOIN)**: Több táblából történő adatlekérés esetén a JOIN műveletek (pl. INNER JOIN, LEFT JOIN) használatával optimalizálhatod az adatlekérdezést, és elkerülheted a több különálló lekérdezésből adódó felesleges műveleteket.

## ****Indexek használata:****

Az indexek kulcsfontosságú szerepet játszanak az adatbázis teljesítményének növelésében, mivel gyorsabbá teszik az adatok lekérdezését, különösen nagy táblák esetén.

* **Indexek szerepe**: Az indexek olyan struktúrák, amelyek segítenek gyorsan megtalálni a rekordokat anélkül, hogy végig kellene nézni az egész táblát (full table scan). Az indexek a táblák bizonyos oszlopaira épülnek, és azok gyors keresését támogatják.
* **Indexek létrehozása**: Az indexeket általában azokon az oszlopokon hozzuk létre, amelyeken gyakran keresünk, szűrünk, vagy amelyekkel gyakran kapcsolunk más táblákhoz (JOIN). Például, ha van egy users tábla, ahol gyakran keresel a email mező alapján, akkor érdemes egy indexet létrehozni az email oszlopon:

sql

Kód másolása

CREATE INDEX idx\_email ON users (email);

* **Több oszlopos indexek (Composite Indexek)**: Ha a lekérdezésed több oszlopra építve keres adatokat (pl. WHERE name = 'John' AND age = 30), érdemes egy kombinált indexet létrehozni:

CREATE INDEX idx\_name\_age ON users (name, age);

* **Elsődleges kulcs és egyedi indexek**: Minden táblában van egy elsődleges kulcs (primary key), amely automatikusan indexelve van. Emellett használhatsz egyedi indexeket (unique index) is, amelyek biztosítják, hogy egy adott oszlop (vagy oszlopok kombinációja) értékei egyediek legyenek a táblában.
* **Indexek hátrányai**: Bár az indexek nagyban felgyorsíthatják a lekérdezéseket, van néhány hátrányuk is. Az indexek karbantartása (pl. adatok beszúrásakor, módosításakor) extra erőforrást igényel, így túl sok index lassíthatja az adatmódosító műveleteket (INSERT, UPDATE, DELETE).

## ****Adatbázis normalizálása és optimalizálása:****

* **Adatbázis normalizálás**: Ez a folyamat segít megszüntetni az adatok redundanciáját, és biztosítja az adatok konzisztenciáját. A normalizálás során a nagy táblákat kisebb, logikai egységekre bontod, amelyeket kulcsokkal kapcsolsz össze (pl. idegen kulcsokkal).
* **Denormalizálás**: Bizonyos esetekben, ahol az adatok elérése elsődleges szempont, előnyös lehet a denormalizálás, vagyis adatredundancia bevezetése a gyorsabb lekérdezések érdekében. Ez általában nagy volumenű olvasási műveleteknél fordul elő.

## ****Caching használata:****

* **Adatok cache-elése**: Ha gyakran lekérdezel ugyanazokat az adatokat az adatbázisból, érdemes caching mechanizmust alkalmazni (pl. Memcached, Redis), hogy ne kelljen mindig az adatbázis-lekérdezést végrehajtani. Az adatok gyorsítótárazása nagy teljesítménynövekedést eredményezhet.

## ****Query Plan elemzése:****

* **Query Plan**: Az adatbázisok (pl. MySQL) lehetővé teszik, hogy megnézd a lekérdezések végrehajtási tervét (query plan) az EXPLAIN kulcsszó segítségével. Ezáltal láthatod, hogy a kérdésed hogyan hajtódik végre, és megértheted, hogy az adatbázis milyen indexeket használ, vagy épp miért fut lassan egy lekérdezés.

EXPLAIN SELECT name, email FROM users WHERE email = 'example@example.com';

Összefoglalva, a PHP-alapú adatbázis-műveletek optimalizálása során fontos a hatékony SQL-lekérdezések megírása, a megfelelő indexek használata, az adatbázis struktúrájának optimalizálása és az esetleges cache-elés bevezetése a jobb teljesítmény érdekében.

# PHP adatkezelés haladó technikák: Adatbázis migrációk, készletkezelés.

A **PHP adatkezelés haladó technikái** közé tartozik az adatbázis-migrációk és a készletkezelés. Ezek a technikák segítenek az adatstruktúrák, táblák változásainak kezelésében, valamint a valós idejű adatkezelésben, például egy készlet vagy raktárkészlet esetében. Nézzük meg részletesen ezeket a fogalmakat.

**1. Adatbázis-migrációk:**

Az adatbázis-migrációk lehetővé teszik az adatbázis-séma verziókövetését, karbantartását és automatikus frissítését. Főként nagyobb projektek esetében hasznosak, amikor gyakran van szükség az adatbázis struktúrájának módosítására, és ezeket a változásokat több fejlesztői környezetben, valamint a produkciós környezetben is egyidejűleg kell bevezetni.

#### Jellemzői és előnyei:

* **Verziókövetés**: A migrációk segítségével nyomon követheted, hogy az adatbázis mikor és hogyan változott meg, és visszaállíthatod az adatbázist korábbi állapotára, ha szükséges.
* **Könnyű frissítés**: Amikor új adatbázis-változásokra van szükség (például új táblák létrehozása, oszlopok hozzáadása vagy módosítása), a migrációk segítségével ezek a változások könnyen alkalmazhatók minden környezetben.
* **Automatizálás**: A legtöbb modern PHP keretrendszer, mint például Laravel, beépített eszközöket biztosít az adatbázis-migrációk kezelésére, így a fejlesztőnek nem kell manuálisan végrehajtania az SQL parancsokat minden környezetben.

#### Laravel adatbázis-migrációk példája:

Laravelben az adatbázis-migrációkat PHP osztályokban definiáljuk. Ezek az osztályok két fő metódust tartalmaznak: up() és down(). Az up() metódus tartalmazza az adatbázis szerkezetének módosításait, míg a down() metódus ezek visszavonására szolgál. (A Laravel keretrendszerrel később ismerkedünk meg részletesen.)

Például egy új products tábla létrehozása:

// Példa egy migrációs fájlra Laravelben

public function up()

{

Schema::create('products', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->integer('quantity');

$table->decimal('price', 8, 2);

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('products');

}

#### További eszközök migrációk kezelésére:

* **Doctrine Migrations**: Ez egy ORM-alapú migrációs eszköz, amely lehetővé teszi a migrációk létrehozását és futtatását egy PHP alapú alkalmazásban.
* **Phinx**: Egy másik népszerű migrációs eszköz, amely a migrációk kezelését és verziózását könnyíti meg.

2. **Készletkezelés**:

A készletkezelés a készletek valós idejű nyilvántartásával, nyomon követésével, beszerzésével és eladásával foglalkozik. PHP-alapú rendszerekben általában adatbázis-táblák, lekérdezések és üzleti logika segítségével valósítjuk meg.

**Készletkezelés alapelvei:**

* **Termékek nyilvántartása**: Egy készletkezelő rendszerben minden termékhez kapcsolódik egy rekord, amely tartalmazza a termék nevét, árát, és aktuális készletét.
* **Készletmozgások kezelése**: A készletkezelés során figyelembe kell venni a készletbeérkezéseket (beszerzés, gyártás), valamint a készletcsökkenéseket (eladás, selejtezés). Ezeket külön táblákban tároljuk.
* **Automatikus riasztások**: Gyakran szükséges automatikus értesítéseket létrehozni, ha egy adott termék készlete kritikus szint alá csökken.

**Készletkezelési példák:**

* **Bevétel nyilvántartás**: Amikor új készlet érkezik, egy bevételi rekordot hozunk létre, és frissítjük a termék mennyiségét.
* **Kiadás nyilvántartás**: Ha egy terméket eladnak vagy felhasználnak, akkor csökkentjük az adott termék készletét, és rögzítjük a kiadás részleteit.

**Példa készletkezelésre PHP-ben:**

// Adatbázis kapcsolat

$connection = new mysqli('localhost', 'username', 'password', 'inventory');

// Készlet lekérdezése

$product\_id = 1;

$sql = "SELECT name, quantity FROM products WHERE id = ?";

$stmt = $connection->prepare($sql);

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

$stmt->execute();

$result = $stmt->get\_result();

$product = $result->fetch\_assoc();

echo "Termék neve: " . $product['name'] . ", Készlet: " . $product['quantity'];

// Készlet frissítése

$new\_quantity = 100;

$update\_sql = "UPDATE products SET quantity = ? WHERE id = ?";

$update\_stmt = $connection->prepare($update\_sql);

$update\_stmt->bind\_param("ii", $new\_quantity, $product\_id);

$update\_stmt->execute();

**Készletmozgások nyilvántartása:**

// Bevétel rögzítése

$quantity\_in = 50;

$sql = "INSERT INTO stock\_movements (product\_id, movement\_type, quantity) VALUES (?, 'in', ?)";

$stmt = $connection->prepare($sql);

$stmt->bind\_param("ii", $product\_id, $quantity\_in);

$stmt->execute();

// Készlet frissítése bevétel után

$new\_quantity = $product['quantity'] + $quantity\_in;

$update\_sql = "UPDATE products SET quantity = ? WHERE id = ?";

$update\_stmt = $connection->prepare($update\_sql);

$update\_stmt->bind\_param("ii", $new\_quantity, $product\_id);

$update\_stmt->execute();

**Készletkezelés valós idejű rendszerekben:**

* **API integrációk**: A modern készletkezelő rendszerek gyakran integrálnak API-kat harmadik fél rendszerekkel (például webshopokkal vagy logisztikai rendszerekkel), hogy valós időben frissüljenek a készletadatok.
* **Több raktár kezelése**: Ha több raktár van, a készletkezelő rendszernek képesnek kell lennie raktáronként nyilvántartani a készletmozgásokat és készletmennyiségeket.

**Összefoglalás:**

A **PHP adatbázis-migrációk** segítenek az adatbázis-sémák karbantartásában és verziókövetésében, különösen nagyobb projektek esetén, ahol folyamatosan változik az adatbázis struktúrája. A **készletkezelés** olyan rendszer létrehozását jelenti, amely képes valós időben nyomon követni a készletek mozgását, beleértve a készletbevételeket és kiadásokat, valamint biztosítja, hogy a készletinformációk naprakészek legyenek.

# PHP és AJAX: Aszinkron adatküldés és fogadás PHP és JavaScript használatával.

A **PHP és AJAX (Asynchronous JavaScript and XML)** egy hatékony módszer az aszinkron adatküldésre és -fogadásra, amely lehetővé teszi a weboldalak számára, hogy frissítsék az adatokat anélkül, hogy a teljes oldalt újra kellene tölteni. Ez javítja a felhasználói élményt, mivel a felhasználó valós időben kap visszajelzést, és gyorsabb adatfeldolgozás történik. Az AJAX használatával JavaScript segítségével küldhetsz adatokat egy szerverre (általában PHP-vel), majd a szerver visszaküldi a feldolgozott adatokat anélkül, hogy újratöltené az oldalt.

1. **Az AJAX működése:**

Az AJAX aszinkron adatküldést és fogadást tesz lehetővé a következő lépések szerint:

1. A JavaScript egy aszinkron kérést indít a szerver felé (pl. PHP script).
2. A PHP script feldolgozza a kérést (adatok mentése, lekérdezése, stb.), majd válaszol.
3. A JavaScript fogadja a PHP-től érkező választ, és frissíti a weboldalon az adatokat, anélkül, hogy újratöltődne az oldal.

2. **AJAX használata JavaScript-ben:**

Az AJAX lekérések megvalósítása JavaScript-ben többféle módon történhet. A két legnépszerűbb módszer a XMLHttpRequest objektum és a modern fetch() API használata.

#### ****a. XMLHttpRequest használata:****

Példa egy egyszerű AJAX kérésre XMLHttpRequest-tel:

// Új XMLHttpRequest objektum létrehozása

let xhr = new XMLHttpRequest();

// A kérés konfigurálása

xhr.open('POST', 'backend.php', true);

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

// A kérés elküldése a szerver felé

xhr.send('name=John&email=john@example.com');

// A válasz kezelése

xhr.onreadystatechange = function() {

if (xhr.readyState === 4 && xhr.status === 200) {

// A válasz sikeresen megérkezett

document.getElementById('result').innerHTML = xhr.responseText;

}

};

Ebben a példában a JavaScript küld egy POST kérést a backend.php felé, amely adatokat dolgoz fel, majd visszaküldi az eredményt, amely megjelenik az oldalon egy result ID-jú elemen.

#### ****b. fetch() API használata:****

A fetch() API egy modernebb és egyszerűbb módszer az aszinkron kérésekhez.

// fetch() API használata POST kérés küldéséhez

fetch('backend.php', {

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: 'name=John&email=john@example.com'

})

.then(response => response.text()) // A válasz feldolgozása textként

.then(data => {

document.getElementById('result').innerHTML = data;

})

.catch(error => console.error('Hiba:', error));

A fetch() egyszerűbbé és olvashatóbbá teszi az aszinkron kéréseket, és támogatja az ígéretek (promises) használatát is.

#### 3. ****PHP oldal kezelése:****

A szerveroldali feldolgozáshoz általában PHP-t használunk. A PHP fogadja a JavaScript által küldött adatokat (GET vagy POST kérés), majd feldolgozza azokat (például adatbázisba mentés, adatlekérdezés), és visszaad egy választ a JavaScript számára.

#### Egyszerű PHP script a JavaScript által küldött POST adat fogadásához:

<?php

// Adatok fogadása

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

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

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

// Adatfeldolgozás (pl. adatbázisba mentés)

// ...

// Válasz küldése a kliens felé

echo "Az Ön neve: $name, email címe: $email";

}

?>

Ez a PHP kód fogadja a JavaScript által küldött name és email adatokat, majd visszaküldi a feldolgozott eredményt a kliens oldal felé.

4. **Gyakorlati példa PHP és AJAX integrációjára:**

#### ****HTML űrlap:****

<form id="ajaxForm">

<label for="name">Név:</label>

<input type="text" id="name" name="name" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<button type="submit">Beküldés</button>

</form>

<div id="result"></div>

<script src="app.js"></script>

#### ****JavaScript az űrlap adatok aszinkron küldéséhez:****

document.getElementById('ajaxForm').addEventListener('submit', function(e) {

e.preventDefault(); // Megakadályozza az űrlap hagyományos elküldését

// Az űrlap adatainak összegyűjtése

const formData = new FormData(this);

// Aszinkron kérés küldése

fetch('backend.php', {

method: 'POST',

body: formData

})

.then(response => response.text())

.then(data => {

// Az eredmény megjelenítése

document.getElementById('result').innerHTML = data;

})

.catch(error => console.error('Hiba:', error));

});

Ez a JavaScript kód megakadályozza az űrlap hagyományos beküldését, majd AJAX kérést küld a PHP szervernek az űrlap adataival, és megjeleníti az eredményt.

#### ****PHP az adatok feldolgozásához:****

<?php

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

$name = htmlspecialchars($\_POST['name']);

$email = htmlspecialchars($\_POST['email']);

// Feldolgozás (pl. adatbázisba mentés)

echo "Beküldött név: $name<br>";

echo "Beküldött email: $email";

}

?>

Ebben a példában a PHP fogadja a JavaScript által beküldött adatokat, és megjeleníti azokat a böngészőben.

5. **Hibakezelés AJAX kérések esetén:**

Az AJAX kérések során fontos a hibakezelés is. A PHP hibás válaszai esetén a JavaScript-ben fel kell készülni a hibák kezelésére, és megfelelő üzeneteket kell megjeleníteni a felhasználónak.

Például a fetch() API-ban a hibák kezelésére használhatjuk a .catch() blokkot, amelyben a hálózati hibákat és más problémákat kezelhetjük:

fetch('backend.php', { method: 'POST', body: formData })

.then(response => {

if (!response.ok) {

throw new Error('Hálózati hiba történt');

}

return response.text();

})

.then(data => {

document.getElementById('result').innerHTML = data;

})

.catch(error => {

console.error('Hiba:', error);

document.getElementById('result').innerHTML = 'Hiba történt az adatok beküldésekor.';

});

6. **Biztonság AJAX használatakor:**

Mivel az AJAX lehetővé teszi az adatokat aszinkron módon, ügyelni kell az adatok biztonságos kezelésére:

* **Adat validáció a szerveroldalon**: Mindig ellenőrizzük és szűrjük a beérkező adatokat a PHP oldalon, hogy elkerüljük a XSS és SQL injection támadásokat.
* **CSRF védelem**: Különösen formok esetében érdemes CSRF tokent használni, hogy megakadályozzuk a jogosulatlan adatküldést.
* **HTTPS használata**: Biztosítsuk, hogy a kérés titkosított csatornán (HTTPS) keresztül történjen, így a küldött adatok biztonságban vannak.

**Összefoglalás:**

Az **AJAX** segítségével könnyen küldhetünk és fogadhatunk adatokat PHP szerverrel anélkül, hogy újratöltenénk az oldalt. Ez a technika javítja a felhasználói élményt és hatékonyabb adatfeldolgozást tesz lehetővé. A modern böngészők támogatják az XMLHttpRequest objektumot és a fetch() API-t is, amelyekkel egyszerűen küldhetünk és fogadhatunk adatokat. Fontos azonban a megfelelő biztonsági intézkedések bevezetése a PHP oldalon.

**Űrlap feldolgozása fotó feltöltésével együtt:**

HTML:

<!DOCTYPE html>

<html lang="hu">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Adatbekérő űrlap</title>

    <style>

        body {

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            margin: 0;

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

        }

        form {

            background-color: #fff;

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

        }

        input[type="text"],

        input[type="email"],

        input[type="tel"],

        input[type="file"] {

            width: 100%;

            padding: 10px;

            margin: 10px 0;

            border: 1px solid #ccc;

            border-radius: 5px;

        }

        input[type="submit"] {

            background-color: #4CAF50;

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        input[type="submit"]:hover {

            background-color: #45a049;

        }

        .error {

            color: red;

        }

    </style>

</head>

<body>

<form id="userForm" enctype="multipart/form-data">

    <label for="name">Név (kötelező):</label>

    <input type="text" name="name" id="name">

    <label for="email">Email (kötelező):</label>

    <input type="email" name="email" id="email">

    <label for="phone">Telefonszám (kötelező, legalább 8 karakter):</label>

    <input type="tel" name="phone" id="phone">

    <fieldset>

        <legend>Válasszon tanfolyamot:</legend>

        <input type="radio" id="frontend" name="course" value="Frontend">

        <label for="frontend">Frontend</label><br>

        <input type="radio" id="backend" name="course" value="Backend">

        <label for="backend">Backend</label><br>

        <input type="radio" id="fullstack" name="course" value="Fullstack">

        <label for="fullstack">Fullstack</label><br>

    </fieldset>

    <label for="photo">Fotó feltöltése (nem kötelező):</label>

    <input type="file" name="photo" id="photo">

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

    <div class="error" id="error"></div>

</form>

<script src="form.js"></script>

</body>

</html>

form.js

document.addEventListener('DOMContentLoaded', function() {

    // *Az űrlap elemének elérése az ID segítségével*

    const form = document.getElementById('userForm');

    // *Az elem elérése, ahol majd a hibák megjelennek*

    const errorDiv = document.getElementById('error');

    // *Eseményfigyelő hozzáadása az űrlaphoz, hogy a beküldést elkapjuk*

    form.addEventListener('submit', function(event) {

        // *Megakadályozza az űrlap alapértelmezett beküldési viselkedését*

        event.preventDefault();

        // *A hibadiv tartalmának törlése, hogy a régi hibák eltűnjenek*

        errorDiv.innerHTML = '';

        // *Az űrlap mezők értékeinek lekérése*

        const name = document.getElementById('name').value.trim();

        const email = document.getElementById('email').value.trim();

        const phone = document.getElementById('phone').value.trim();

        const course = document.querySelector('input[name="course"]:checked');

        const photo = document.getElementById('photo').files[0];

        // *Adatok logolása a teszteléshez*

        console.log('Név:', name);

        console.log('Email:', email);

        console.log('Telefonszám:', phone);

        console.log('Tanfolyam:', course ? course.value : 'Nincs kiválasztva');

        console.log('Fénykép:', photo ? photo.name : 'Nincs fájl kiválasztva');

//*gyűjtsük össze a hibákat*

        let errors = [];

        // *Ellenőrzés: a név mező nem lehet üres*

        if (name === '') {

            errors.push('A név megadása kötelező.');

        }

        /\* *Email formátum ellenőrzése; egy reguláris kifejezést (regexet) használ az email-címek érvényesítésére*\*/

        const emailPattern = /^[a-zA-Z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$/;

        if (!emailPattern.test(email)) {

            errors.push('Az email formátuma helytelen.');

        }

        // *Telefonszám ellenőrzése: legalább 8 karakter hosszú legyen*

        if (phone.length < 8) {

            errors.push('A telefonszámnak legalább 8 karakter hosszúnak kell lennie.');

        }

        // *Tanfolyam kiválasztás ellenőrzése*

        if (!course) {

            errors.push('Válasszon egy tanfolyamot.');

        }

        // *Hibák logolása*

        console.log('Hibák:', errors);

        // *Ha van hiba, megjelenítjük*

        if (errors.length > 0) {

            errorDiv.innerHTML = errors.join('<br>');

        } else {

            // *FormData objektum létrehozása*

            const formData = new FormData();

            formData.append('name', name);

            formData.append('email', email);

            formData.append('phone', phone);

            formData.append('course', course.value);

            if (photo) {

                formData.append('photo', photo);

            }

            // *FormData tartalmának logolása*

            console.log('FormData:', formData);

//*ezen a ponton teszteljünk!!! Addig ne adjuk át az adatokat*

            // *Fetch API használata*

            fetch('form.php', {

                method: 'POST',

                body: formData

            })

            .then(response => response.text()) // *Válasz logolása*

            .then(data => {

                console.log('Szerver válasza:', data);

                errorDiv.innerHTML = data;

            })

            .catch(error => {

                console.error('Fetch hiba:', error);

                errorDiv.innerHTML = 'Hiba történt az adatok küldésekor: ' + error.message;

            });

        }

    });

});

form.php

<?php

// *A nevet kisbetűsítjük és minden szó első betűjét nagybetűre állítjuk*

/\**function formatName($name) {*

*$name = strtolower($name); // Minden karaktert kisbetűsítünk*

*$name = ucwords($name); // Minden szó első betűjét nagybetűsítjük*

*return $name;*

*}*\*/

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

    // *Név és email formázása kiszervezhető függvényekbe*

   // *$name = formatName($\_POST['name']);*

   $name = htmlspecialchars(ucwords(strtolower($\_POST['name'])));

    $email = strtolower($\_POST['email']); // *Emailt kisbetűsítjük*

    $phone = htmlspecialchars($\_POST['phone']);

    $course = htmlspecialchars($\_POST['course']);

    if ($name && $email && $phone && $course) {

        echo "<h2>Beküldött adatok:</h2>";

        echo "Név: " . $name . "<br>";

        echo "Email: " . $email . "<br>";

        echo "Telefonszám: " . $phone . "<br>";

        echo "Választott tanfolyam: " . $course . "<br>";

        // *Fotó ellenőrzése és feltöltése*

        if (isset($\_FILES['photo']) && $\_FILES['photo']['error'] === UPLOAD\_ERR\_OK) {

            $photoTmpPath = $\_FILES['photo']['tmp\_name'];

            $photoName = uniqid() . '\_' . $\_FILES['photo']['name'];

            $uploadDir = 'uploads/';

/\**\*Ez a feltétel azt ellenőrzi, hogy létezik-e egy adott mappa a megadott útvonalon. Az is\_dir() függvény visszaadja true-t, ha a megadott útvonal egy létező mappa, és false-t, ha nem létezik.* \*/

            if (!is\_dir($uploadDir)) {

                mkdir($uploadDir, 0777, true);

            }

/\**\*A mkdir() függvény egy új mappát hoz létre a megadott útvonalon.*

*Paraméterek:*

*$uploadDir: Ez az a változó, amely tartalmazza a létrehozandó mappa nevét vagy útvonalát.*

*0777: Ez a mappa jogosultsága, amely azt jelenti, hogy mindenki (felhasználó, csoport, mások) számára teljes hozzáférést biztosít (olvashat, írhat, futtathat). (A 0777 az oktális számrendszerben van megadva, ahol minden számjegy egy csoportnak a jogosultságait jelenti.)*

*true: Ez a harmadik paraméter azt mondja meg a mkdir() függvénynek, hogy rekurzívan hozza létre a mappát, vagyis, ha szükséges, létrehozza az összes szülő mappát is, amelyek még nem léteznek az útvonalban.* \*/

            $destination = $uploadDir . basename($photoName);

            if (move\_uploaded\_file($photoTmpPath, $destination)) {

                echo "Feltöltött fotó:<br>";

                echo "<img src='" . $destination . "' alt='Feltöltött fotó' style='max-width:200px;'><br>";

            } else {

                echo "Hiba történt a fotó feltöltésekor.";

            }

        } else {

            echo "Nem lett fotó feltöltve.";

        }

    } else {

        echo "Kérjük, töltse ki az összes kötelező mezőt!";

    }

} else {

    echo "Hibás kérés!";

}

?>

# PHP fájlkezelés haladó technikák: Fájlok és könyvtárak manipulálása, jogosultságok kezelése

A **PHP fájlkezelés haladó technikái** lehetővé teszik a fájlok és könyvtárak különféle műveleteinek végrehajtását, beleértve a létrehozást, olvasást, írást, törlést és jogosultságok kezelését. A fájlkezelés elengedhetetlen számos alkalmazás számára, például fájlok feltöltésénél, adatnaplózásnál vagy akár teljes fájlrendszer-adminisztrációs feladatok végrehajtásánál.

#### 1. ****Fájlok kezelése:****

A fájlkezelés során olyan műveleteket végezhetünk, mint a fájlok létrehozása, olvasása, írása, módosítása és törlése. PHP-ben ezekhez számos beépített függvény áll rendelkezésre.

#### a. ****Fájlok olvasása:****

* **file\_get\_contents()**: Egy fájl tartalmának egy lépésben történő beolvasására használható, és az eredményt egy stringben adja vissza.

$fileContent = file\_get\_contents('path/to/file.txt');

echo $fileContent;

* **fopen() és fread()**: Ezek a függvények részletesebb kontrollt biztosítanak a fájlkezelés felett, és lehetővé teszik a fájlok olvasását bájtonként vagy karakterenként.

$file = fopen('path/to/file.txt', 'r');

while (!feof($file)) {

echo fgets($file); // Egy sor beolvasása

}

fclose($file);

#### b. ****Fájlok írása:****

* **file\_put\_contents()**: Lehetővé teszi, hogy egy lépésben adatokat írjunk egy fájlba.

$data = "Ez egy új fájl tartalma.";

file\_put\_contents('path/to/file.txt', $data);

* **fwrite()**: Több kontrollt biztosít az írás felett, például a fájl megnyitását különböző módokban (pl. írásra, hozzáfűzésre).

$file = fopen('path/to/file.txt', 'a'); // Hozzáfűzési mód

fwrite($file, "Új tartalom a fájl végén.");

fclose($file);

#### c. ****Fájlok törlése:****

* **unlink()**: Ezzel a függvénnyel fájlokat lehet törölni.

php

Kód másolása

unlink('path/to/file.txt');

#### 2. ****Könyvtárak kezelése:****

A fájlok mellett a könyvtárak kezelése is fontos lehet, például amikor fájlokat másolunk, mozgatunk vagy létrehozunk új könyvtárakat.

#### a. ****Könyvtárak létrehozása:****

* **mkdir()**: Ez a függvény új könyvtárat hoz létre.

mkdir('path/to/new\_directory', 0755, true); // Jogosultságok beállítása (0755), és az összes szükséges szülő könyvtár létrehozása

#### b. ****Könyvtárak olvasása:****

* **scandir()**: A könyvtár tartalmát listázhatjuk ezzel a függvénnyel.

php

Kód másolása

$files = scandir('path/to/directory');

foreach ($files as $file) {

echo $file . "<br>";

}

#### c. ****Könyvtárak törlése:****

* **rmdir()**: Ez a függvény egy üres könyvtárat töröl.

rmdir('path/to/directory');

Ha a könyvtár nem üres, akkor rekurzívan törölhetjük az összes benne lévő fájlt és könyvtárat egy egyéni funkcióval.

#### 3. ****Fájlok és könyvtárak jogosultságainak kezelése:****

A fájlok és könyvtárak jogosultságai kritikusak lehetnek a biztonság szempontjából, különösen olyan webhelyeken, ahol több felhasználó hozzáférhet a rendszerhez.

#### a. ****Jogosultságok beállítása:****

* **chmod()**: Ezzel a függvénnyel fájlok és könyvtárak jogosultságait módosíthatjuk.

chmod('path/to/file.txt', 0644); // Fájl jogosultságainak beállítása

Az értékek bináris alapon értelmezhetők (pl. 0644 jelenti, hogy a tulajdonos olvashatja és írhatja a fájlt, míg a csoport és a többiek csak olvashatják).

#### b. ****Jogosultságok ellenőrzése:****

* **fileperms()**: Ezzel a függvénnyel lekérdezhetjük egy fájl vagy könyvtár jogosultságait.

$permissions = fileperms('path/to/file.txt');

echo decoct($permissions & 0777); // A jogosultságok bináris formátumban

#### 4. ****Fájlok feltöltése PHP-ben:****

A fájlfeltöltések gyakoriak a webes alkalmazásokban, például profilképek, dokumentumok vagy más típusú fájlok feltöltésénél. A PHP beépített támogatást nyújt fájlok feltöltésére az űrlapokon keresztül.

#### Példa egy fájl feltöltésére PHP-ben:

HTML űrlap:

<form action="upload.php" method="POST" enctype="multipart/form-data">

Válassz fájlt: <input type="file" name="fileToUpload">

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

</form>

PHP feldolgozás:

<?php

$target\_dir = "uploads/";

$target\_file = $target\_dir . basename($\_FILES["fileToUpload"]["name"]);

$uploadOk = 1;

// Ellenőrizzük, hogy a fájl már létezik-e

if (file\_exists($target\_file)) {

echo "A fájl már létezik.";

$uploadOk = 0;

}

// Fájl feltöltése, ha nincs hiba

if ($uploadOk == 1) {

if (move\_uploaded\_file($\_FILES["fileToUpload"]["tmp\_name"], $target\_file)) {

echo "A fájl feltöltése sikerült: " . basename($\_FILES["fileToUpload"]["name"]);

} else {

echo "Hiba történt a fájl feltöltésekor.";

}

}

?>

#### 5. ****Haladó fájlműveletek:****

#### a. ****Fájlok másolása, mozgatása és átnevezése:****

* **copy()**: Egy fájl másolása egyik helyről a másikra.

copy('path/to/source.txt', 'path/to/destination.txt');

* **rename()**: Egy fájl vagy könyvtár átnevezése vagy áthelyezése.

rename('path/to/old\_name.txt', 'path/to/new\_name.txt');

#### b. ****Fájlméret és létrehozás dátumának lekérdezése:****

* **filesize()**: Egy fájl méretének lekérdezése bájtokban.

echo filesize('path/to/file.txt') . " bájt";

* **filemtime()**: Egy fájl utolsó módosítási időpontjának lekérdezése.

echo "Utolsó módosítás: " . date("F d Y H:i:s.", filemtime('path/to/file.txt'));

#### 6. ****Biztonsági megfontolások:****

A fájlkezelés során különösen fontos a biztonság:

* **Fájlnevek ellenőrzése**: Soha ne fogadjuk el közvetlenül a felhasználótól kapott fájlneveket, mindig ellenőrizzük azokat.
* **Könyvtárfeltörés megakadályozása**: Használjunk relatív útvonalakat és ellenőrizzük, hogy a felhasználó nem próbál meg könyvtárszintet ugrani (pl. ../ karakterekkel).
* **Fájlméret és típus ellenőrzése**: Fájlok feltöltésekor korlátozzuk a fájlok méretét és típusát, hogy elkerüljük a rosszindulatú fájlok feltöltését.

#### 6. ****Biztonsági megfontolások:****

* **Fájlméret és típus ellenőrzése**: Amikor fájlokat töltünk fel, mindig érdemes korlátozni a fájlok maximális méretét és ellenőrizni a fájl típusát, hogy csak megengedett típusokat fogadjunk el (például képeket, dokumentumokat). Ezt a következőképpen lehet megvalósítani:

// Ellenőrizzük a fájlméretet (max 2 MB)

if ($\_FILES['fileToUpload']['size'] > 2000000) {

echo "A fájl túl nagy.";

$uploadOk = 0;

}

// Ellenőrizzük a fájltípust (csak képek)

$fileType = strtolower(pathinfo($target\_file, PATHINFO\_EXTENSION));

if (!in\_array($fileType, ['jpg', 'png', 'jpeg', 'gif'])) {

echo "Csak JPG, JPEG, PNG és GIF fájlok engedélyezettek.";

$uploadOk = 0;

}

* **Fájlnév szűrése**: Mindig biztonságos fájlneveket használjunk, és szűrjük a fájlneveket, hogy elkerüljük a könyvtárszint ugrásokat vagy rosszindulatú fájlok feltöltését. Például, ha a fájl tartalmaz tiltott karaktereket (pl. ../), azt cserélhetjük biztonságos alternatívára.
* **Jogosultságok kezelése**: Győződj meg arról, hogy a fájlokhoz és könyvtárakhoz megfelelő jogosultságok vannak beállítva, hogy csak az engedélyezett felhasználók férjenek hozzá az adatokhoz. Például a feltöltött fájlokat be lehet állítani, hogy csak olvashatók legyenek a nyilvánosság számára, de ne legyenek írhatók:

chmod('uploads/somefile.txt', 0644); // Csak olvasható jogosultság a felhasználók számára

#### Összefoglalás:

A **PHP fájlkezelés haladó technikái** számos eszközt kínálnak a fájlok és könyvtárak kezelésére, beleértve az olvasást, írást, törlést, másolást és áthelyezést. Az ilyen műveletek során különösen fontos a jogosultságok megfelelő kezelése és a biztonságos fájlműveletek végrehajtása, például a fájlok feltöltésekor. A megfelelő biztonsági intézkedések és jogosultságkezelés segít elkerülni a lehetséges támadásokat, mint például a fájlbeillesztési támadások vagy a rosszindulatú fájlok feltöltése.

Ha egy PHP alapú rendszerben fájlokat és könyvtárakat kezelsz, ezek a technikák segíthetnek a hatékony és biztonságos működés biztosításában.

# A RESTful API alkalmazás fő követelményei

Az eddigiekben számos megoldással ismerkedtünk, hogyan hajthatunk végre különféle műveleteket. A záródolgozat kimeneti követelménye, hogy az alkalmazásnak meg kell felelnie a RESTful API követelményeinek. A következőkben megnézzük, hogy melyek ezek a követelmények, melyet később a gyakorlatba is átültetünk.

#### 1. ****Kliens-szerver architektúra****

A RESTful API egyértelműen elválasztja a klienst (például egy böngésző, mobilalkalmazás vagy más API-felhasználó) és a szervert. A szerver nem tárol kliens oldali állapotokat (stateless), csak válaszokat küld a kliens által küldött kérésekre. Ez a felépítés biztosítja a feladatok tiszta szétválasztását.

#### 2. ****Stateless (Állapotmentesség)****

A RESTful API-k stateless-ek, ami azt jelenti, hogy minden egyes kérésnek teljesnek kell lennie önmagában. A szerver nem tárolhat kliens oldali adatokat egyik kéréstől a másikig. Minden kérésnek tartalmaznia kell minden szükséges információt (pl. hitelesítési adatok, kliens állapot), hogy a szerver megfelelően válaszolhasson.

#### 3. ****HTTP protokoll használata****

A REST API-k az HTTP-t használják, és kihasználják az HTTP-szabvány által biztosított metódusokat, például:

* **GET**: Adatok lekérdezésére használjuk.
* **POST**: Új erőforrás létrehozására.
* **PUT**: Meglévő erőforrás frissítésére vagy felülírására.
* **PATCH**: Egy meglévő erőforrás részleges frissítésére.
* **DELETE**: Erőforrás törlésére.
* **OPTIONS**: Lekérdezés a szerver által támogatott műveletekről.

#### 4. ****Erőforrás-alapú URL struktúra****

A RESTful API-kban minden erőforráshoz egy egyedi URL társul. Az URL-eket úgy kell megtervezni, hogy könnyen érthetők és logikailag szervezettek legyenek. Például:

* **GET /contacts**: Az összes névjegy lekérdezése.
* **GET /contacts/1**: Az 1-es azonosítójú névjegy lekérdezése.
* **POST /contacts**: Új névjegy létrehozása.
* **PUT /contacts/1**: Az 1-es névjegy frissítése.
* **DELETE /contacts/1**: Az 1-es névjegy törlése.

Az URL-eknek az erőforrásokra kell utalniuk (pl. contacts, users, orders), és azonosítókkal lehet konkrét erőforrásokra hivatkozni.

#### 5. ****Állapotkódok használata****

A RESTful API-k használják a HTTP válaszokban megadott állapotkódokat annak jelzésére, hogy egy kérés sikeres volt-e, vagy hibába ütközött. Példák:

* **200 OK**: Sikeres kérés.
* **201 Created**: Erőforrás sikeresen létrehozva (pl. új felhasználó hozzáadása).
* **400 Bad Request**: Rosszul formázott kérés.
* **401 Unauthorized**: Jogosultság hiánya a kéréshez.
* **403 Forbidden**: Tiltott hozzáférés.
* **404 Not Found**: Erőforrás nem található.
* **500 Internal Server Error**: Szerveroldali hiba.

#### 6. ****HATEOAS (Hypermedia as the Engine of Application State)****

A RESTful API ideális esetben követi a HATEOAS elvet, ami azt jelenti, hogy a válaszok tartalmaznak további hivatkozásokat más releváns erőforrásokra. Például egy felhasználó lekérdezésére adott válasz tartalmazhat hivatkozásokat a felhasználó névjegyeire vagy rendeléseire:

{

"id": 1,

"name": "John Doe",

"links": [

{ "rel": "self", "href": "/users/1" },

{ "rel": "contacts", "href": "/users/1/contacts" }

]

}

#### 7. ****JSON használata (vagy más formátum)****

A RESTful API-k általában **JSON** formátumban adnak vissza adatokat, mivel ez egy könnyen olvasható és platformfüggetlen formátum. Azonban XML, YAML vagy más formátumok is használhatók, attól függően, hogy mi szükséges az alkalmazás szempontjából. A válaszok JSON formátuma például:

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com"

}

#### 8. ****Biztonság (HTTPS, Hitelesítés)****

A RESTful API-k védelméhez **HTTPS** használata szükséges az adatok titkosítása érdekében. Az API-k hozzáféréséhez szükség lehet hitelesítésre, amelyet különféle módszerekkel lehet megvalósítani, például:

* **OAuth**: Token-alapú hitelesítés.
* **API kulcsok**: Minden kérésnél egy kulcs átadása, amely hitelesíti a klienst.
* **JWT (JSON Web Token)**: Kriptografikusan aláírt token, amely tartalmazza a felhasználó adatait.

#### 9. ****Cache-elhetőség****

A RESTful API-kban a válaszok cache-elhetők, ha az erőforrások nem változnak gyakran. A megfelelő HTTP cache fejlécek használata biztosítja, hogy a kliens (pl. böngésző) helyi gyorsítótárából használhatja az adatokat, ami javítja a teljesítményt és csökkenti a szerver terhelését. Például a Cache-Control fejléc:

Cache-Control: max-age=3600

#### 10. ****Verziózás****

Az API-k folyamatos fejlesztése miatt fontos az API-k verziózása, hogy az új fejlesztések ne törjék meg a régi implementációkat. Ez többféleképpen valósítható meg, például az URL-ben:

/v1/contacts

/v2/contacts

#### 11. ****Idempotencia****

A RESTful API-kban az idempotens műveletek olyan műveletek, amelyek többszöri végrehajtása ugyanazt az eredményt adja. Például a **GET** és **DELETE** műveletek idempotensek, mert bármennyiszer hajtjuk végre őket, ugyanaz lesz az eredmény.

#### Összegzés

Egy RESTful API-nak tehát a következő elveket kell követnie:

1. Kliens-szerver architektúra.
2. Állapotmentesség (stateless).
3. HTTP metódusok használata (GET, POST, PUT, DELETE, stb.).
4. Erőforrás-alapú URL-ek.
5. Megfelelő állapotkódok használata.
6. HATEOAS támogatás (opcionális, de ajánlott).
7. JSON vagy más formátumú válaszok.
8. Biztonság és hitelesítés (HTTPS, OAuth, API kulcsok stb.).
9. Cache-elhetőség a megfelelő HTTP fejlécekkel.
10. Verziózás.
11. Idempotencia a megfelelő műveleteknél.

Ha egy alkalmazás megfelel ezeknek a követelményeknek, akkor **RESTful API**-nak tekinthető.

# Összetett feladat megoldása különböző módszerekkel: Névjegyek

Egy olyan komplex feladatot készítünk el, melynek során a gyakorlatban fogjuk azonnal alkalmazni az új ismereteket. Első lépésként megtanuljuk, hogyan tudunk kapcsolatot létre hozni az adatbázisunkkal, procedurális illetve objektum orientált módszerrel, majd megnézzük a tesztelését sikeresen létre jön-e a kapcsolat.

A feladat megoldása során a CRUD azaz create, read, update és delete műveleteket fogjuk megvalósítani, amely a szerveroldali programozás legfontosabb alapköve. A különböző műveletek végrehajtására nagyon sokféle módszert fogunk használni, míg végül eljutunk a RESTFUL API alkalmazás követelményeiig, amelyeket a záró dolgozatban is alkalmazni kell.

Kiolvasunk adatokat az adatbázisból, melyeket megjelenítünk a böngészőben megfelelően formázva, az eddigi ismereteink felhasználásával. Ezt követően űrlapot fogunk készíteni, melynek segítségével a böngészőből viszünk fel adatokat az adatbázisunkba. A feladat megoldása során ellenőrizzük az űrlapba beírt adatokat, és különböző függvényekkel ismerkedünk meg, melyek segítségével megtisztíthatjuk az adatokat, pl. levághatjuk a fölösleges szóközöket a szöveg elejéről és végéről, megakadályozhatjuk, hogy valaki káros scriptet vigyen fel.

Harmadik lépésként a módosítást valósítjuk meg, amely nagyon hasonlatos a felvitelhez, azzal a fő különbséggel, hogy az űrlapba be kell olvasnunk előzetesen a módosítani kívánt adatokat.

A negyedik lépés a törlés művelete lesz, mely során egy gombnyomással törölhetünk majd az adatbázisból adatot a böngészőből.

Ezeken túlmenően megnézzük, milyen lehetőségeket alkalmazhatunk kép feltöltése során, ha mindig egyedi nevet szeretnék adni a képünknek, valamint ennek a megvalósítási lépéseit.

Végezetül pedig a titkosítással és hasító függvényekkel ismerkedünk meg a feladat során. Beléptető felületet készítünk, hogy illetéktelenek ne tudjanak webesen törölgetni, módosítgatni az adatbázisban, csakis azok, akik rendelkeznek felhasználónévvel és jelszóval. Ehhez kiegészítésként a kiléptetést is elkészítjük.

## Adatbázis

Jelen feladatban egy egyszerű egy táblás adatbázist és fiktív adatokat használunk a különféle megoldásai lehetőségek vizsgálatához.

A tábla neve: **nevjegyek**. Ebben a táblában tárolódnak a személyek különböző adatai, amelyek a következők:

* **id**:
  + Típus: smallint(5) unsigned NOT NULL AUTO\_INCREMENT
  + Ez a tábla elsődleges kulcsa (PRIMARY KEY), amely egyedi azonosítót biztosít minden rekordhoz. A smallint típus azt jelenti, hogy 0 és 65535 közötti értékeket tárolhat, ami egy kisebb tartomány, de elégséges lehet a névjegyek számára. Az AUTO\_INCREMENT lehetővé teszi, hogy az új rekordok automatikusan új, egyedi azonosítót kapjanak.
* **foto**:
  + Típus: varchar(64) NOT NULL DEFAULT 'nincskep.png'
  + Ez a mező a névjegyhez tartozó kép fájlnevét tárolja. Ha nincs kép feltöltve, akkor a **'nincskep.png'** alapértelmezett kép kerül tárolásra.
* **nev**:
  + Típus: varchar(20)
  + A név mező a névjegyen szereplő személy nevét tartalmazza, legfeljebb 20 karakter hosszúságban.
* **cegnev**:
  + Típus: varchar(77)
  + A személy cégét vagy munkahelyét tárolja, legfeljebb 77 karakterrel.
* **foglakozas**:
  + Típus: varchar(60)
  + A személy foglalkozását vagy pozícióját tárolja, legfeljebb 60 karakter hosszúságban.
* **email**:
  + Típus: varchar(28)
  + A névjegyen szereplő személy email címét tartalmazza, legfeljebb 28 karakter hosszúságban. Ez kissé rövidnek tűnhet, mivel egyes e-mail címek hosszabbak lehetnek.
* **mobil**:
  + Típus: varchar(13)
  + A mobiltelefonszám mezője. Ebben a mezőben legfeljebb 13 karakter hosszúságú számot lehet tárolni, ami elég lehet a nemzetközi formátumú mobiltelefonszámokhoz.

#### Megjegyzések a tábláról:

* **Képfájl tárolás**: A képfájl neve van tárolva (nem a kép maga), ami általánosan használatos megoldás. A képeket egy különálló fájlrendszerben tároljuk, és a fájlneveket használjuk a hivatkozásokhoz.
* **Alapértelmezett értékek**: A foto mező alapértelmezett értéke a 'nincskep.png', ami egy üres vagy alapértelmezett kép jelenítésére szolgál, ha nincs fotó feltöltve.
* **NULL értékek**: Az e-mail, cég név, foglalkozás és mobil mezők NULL-t tartalmazhatnak, ami azt jelenti, hogy ezek a mezők üresen is maradhatnak.

#### 2. ****Adatok betöltése (INSERT INTO)****

A megadott **INSERT INTO** parancsokkal különböző névjegyadatokat töltöttél be. Ezek a parancsok számos személyt reprezentálnak, különféle munkahelyi pozíciókkal és elérhetőségekkel. Az adatok tartalmazzák:

* **Személyek neveit** (pl. 'Bognár Csanád', 'Kis Alex').
* **Cégneveket** (pl. 'K&H Bank', 'Concorde Csoport').
* **Foglalkozásokat** (pl. 'Bolti pénztáros', 'Kereskedelmi részegység vezetője').
* **Email címeket és telefonszámokat**.

#### 3. ****Adatbázis és alkalmazás következményei****

Az adatbázis egy viszonylag egyszerű **névjegykezelő** rendszert szolgál ki, amely a következő funkciókat biztosíthatja:

* **Névjegyek létrehozása (CREATE)**: Új névjegyek adhatók hozzá (pl. egy új alkalmazott adatai).
* **Névjegyek megjelenítése (READ)**: A rendszer képes lekérdezni az összes névjegyet és azok részletes adatait.
* **Névjegyek frissítése (UPDATE)**: A meglévő névjegyek adatai frissíthetők (pl. munkahely vagy elérhetőségek változása).
* **Névjegyek törlése (DELETE)**: Lehetőség van névjegyek törlésére.

CREATE TABLE IF NOT EXISTS nevjegyek (

    id smallint(5) unsigned NOT NULL AUTO\_INCREMENT,

    foto varchar(64) NOT NULL DEFAULT 'nincskep.png',

    nev varchar(20) DEFAULT NULL,

    cegnev varchar(77) DEFAULT NULL,

    foglakozas varchar(60) DEFAULT NULL,

    email varchar(28) DEFAULT NULL,

    mobil varchar(13) DEFAULT NULL,

    PRIMARY KEY (id)

);

INSERT INTO nevjegyek (id, foto, nev, cegnev, foglakozas, email, mobil) VALUES

(4, 'nincskep.png', 'Bognár Csanád', 'Rimai Coalitio', 'Bolti pénztáros', 'bognar.csanad@freemail.hu', '(70) 126-1342'),

(5, 'nincskep.png', 'Katona Taksony', 'K&H Bank', 'Területi közig., igazságszolg. közép vezetője', 'taksony\_katona@vipmail.hu', '(70) 668-3833'),

(6, 'nincskep.png', 'Antal Ilona', 'Genesis Energy Nyrt.', 'Kereskedelmi részegység vezetője', 'antal.ilona@hotmail.com', '(20) 346-3148'),

(7, 'nincskep.png', 'Barta Kristóf', 'NIK Központi Autótervező Iroda', 'Cipész, cipőkészítő, -javító', 'k.barta@gmail.com', '(30) 618-9109'),

(8, 'nincskep.png', 'Orbán Anasztáz', 'HEINEKEN Hungária Sörgyárak Zrt.', 'Középfokú oktatási intézmény szakoktatója', 'anasztaz.orban@yahoo.com', '(20) 470-1547'),

(9, 'nincskep.png', 'Kis Alex', 'Pécsi Dohánygyár', 'Vájár, segédvájár', 'kis\_alex@gmail.com', '(30) 356-1144'),

(10, 'nincskep.png', 'Nagy Aletta', 'Toti Sport', 'Bolti pénztáros', 'a.nagy@yahoo.com', '(70) 194-8179'),

(11, 'nincskep.png', 'Nemes Eulália', 'Digic Pictures', 'Fegyveres erők felsőfokú foglalkozásai', 'eulalia\_nemes@gmail.com', '(20) 907-7812'),

(12, 'nincskep.png', 'Bakos Emil', 'Moskovits Sámuel vasöntödéje és gépgyára', 'Szállító- és rakodómunkás', 'bakos.emil@freemail.hu', '(30) 614-7177'),

(13, 'nincskep.png', 'Kis Diána', 'Concorde Csoport', 'Egyéb műszaki foglalkozások', 'diana\_kis@mailbox.hu', '(20) 425-8210'),

(14, 'nincskep.png', 'Nagy Kelemen', 'Concorde Csoport', 'Egyéb szakképzett oktatók, nevelők', 'k.nagy@freemail.hu', '(20) 549-3060'),

(15, 'nincskep.png', 'Horváth Margit', 'InfoRádió', 'Bútorasztalos', 'margit.horvath@hotmail.com', '(70) 531-6569'),

(16, 'nincskep.png', 'Kiss Dénes', 'Goodwill Communications', 'Fegyveres erők felsőfokú foglalkozásai', 'denes\_kiss@hotmail.com', '(30) 587-8352'),

(17, 'nincskep.png', 'Szűcs Andrea', 'Offset és Játékkártya Nyomda Zrt.', 'Egyéb műszaki foglalkozások', 'andrea.szucs@freemail.hu', '(70) 218-2486'),

(18, 'nincskep.png', 'Kocsis Krisztina', 'Dunai Vasmű', 'Szociális munkás', 'kocsis.krisztina@gmail.com', '(30) 582-3853'),

(19, 'nincskep.png', 'Vincze János', 'Első Magyar Cérnagyár', 'Minőségi, műszaki, biztonsági ellenőr', 'vincze\_janos@freemail.hu', '(20) 107-4404'),

(20, 'nincskep.png', 'Hegedűs György', 'Cartographia', 'Általános orvos', 'gyorgy\_hegedus@freemail.hu', '(30) 653-4082'),

(21, 'nincskep.png', 'Molnár Jenő', 'Első Nemzeti Közműszolgáltató', 'Szövő', 'jeno.molnar@citromail.hu', '(20) 459-4843'),

(22, 'nincskep.png', 'Sándor Apor', 'Nézőpont Intézet', 'Mosónő, vasalónő', 'sandor.apor@freemail.hu', '(70) 995-1103'),

(23, 'nincskep.png', 'Halász Rozália', 'Merck', 'Hegesztő, lángvágó', 'rozalia.halasz@gmail.com', '(20) 766-7351'),

(24, 'nincskep.png', 'Balla Ibolya', 'Centrum Áruházak', 'Kazángépkezelő (vizsgázott kazánfűtő)', 'balla.ibolya@gmail.com', '(20) 925-4219'),

(25, 'nincskep.png', 'Szabó Virgínia', 'Raiffeisen Bank (Magyarország)', 'Felsőfokú tanintézeti tanár, oktató', 'v.szabo@yahoo.com', '(20) 915-5028'),

(26, 'nincskep.png', 'Fekete Ferenc', 'Vidanet', 'Óvónő', 'fekete.ferenc@gmail.com', '(30) 976-6445'),

(27, 'nincskep.png', 'Hegedűs Benedek', 'Index (internetes újság)', 'Számviteli és pénzügyi részegység vezetője', 'benedek\_hegedus@freemail.hu', '(30) 682-1804'),

(28, 'nincskep.png', 'Fodor Magdolna', 'Zoltán Hugó és Társa', 'Egyéb pedagógusok', 'magdolna\_fodor@vipmail.hu', '(20) 751-8304'),

(29, 'nincskep.png', 'Szűcs Gusztáv', 'T-Mobile (Magyar Telekom Csoport)', 'Mechanikai műszerész', 'szucs\_gusztav@vipmail.hu', '(20) 213-1955'),

(30, 'nincskep.png', 'Németh Taksony', 'Kner nyomda', 'Pénzügyi ügyintéző', 't.nemeth@hotmail.com', '(20) 626-9632'),

(31, 'nincskep.png', 'Magyar Teodóra', 'Hajós- és Szántó Elektromos Gyára Rt.', 'Vegyészmérnök', 't.magyar@upc.hu', '(30) 634-9864'),

(32, 'nincskep.png', 'Király Édua', 'HospInvest Zrt.', 'Oktatási részegység vezetője', 'edua.kiraly@hotmail.com', '(30) 569-2536'),

(33, '1443198181.jpg', 'Antal F', 'Virtual Call Center Solutions Zrt.', 'Középiskolai tanár, oktató', 'felix.antal@vipmail.hu', '(70) 843-4487'),

(34, 'nincskep.png', 'Pintér Kartal', 'Pick Szeged Zrt.', 'Kereskedelmi részegység vezetője', 'pinter.kartal@yahoo.com', '(30) 411-8240'),

(35, 'nincskep.png', 'Budai Frigyes', 'Magyar Turizmus Zrt.', 'Nehézföldmunkagép-kezelő', 'budai\_frigyes@vipmail.hu', '(70) 438-5219'),

(36, 'nincskep.png', 'Vincze Vazul', 'HospInvest Zrt.', 'Fegyveres erők felsőfokú foglalkozásai', 'vincze.vazul@gmail.com', '(20) 159-6994'),

(37, 'nincskep.png', 'Farkas Damján', 'Raiffeisen Bank (Magyarország)', 'Egyéb termelői, szolg. részegységek vezetői', 'd.farkas@indamail.hu', '(30) 670-2868'),

(38, 'nincskep.png', 'Sándor Xénia', 'Kinizsi Bank', 'Egészségügyi és szoc. szolg. részegység vezetője', 'xenia.sandor@freemail.hu', '(70) 718-7929'),

(39, 'nincskep.png', 'Varga Viola', 'Elektronikus Mérőkészülékek Gyára (Sashalom)', 'Ügyvéd', 'v.varga@gmail.com', '(30) 132-3593'),

(40, 'nincskep.png', 'Dudás Edgár', 'Minimal Art Family', 'Újságíró', 'edgar\_dudas@gmail.com', '(20) 391-8580'),

(41, 'nincskep.png', 'Horváth Hilária', 'Hungarotex', 'Fodrász, borbély', 'h.horvath@freemail.hu', '(30) 712-4304'),

(42, 'nincskep.png', 'Kiss Lőrinc', 'Toti Sport', 'Általános orvos', 'kiss\_lorinc@freemail.hu', '(30) 939-2149'),

(43, 'nincskep.png', 'Barna Valter', 'Bárdi Autó Zrt.', 'Felszolgáló, vendéglátóipari eladó', 'barna\_valter@vipmail.hu', '(30) 476-6957'),

(44, 'nincskep.png', 'Balázs Ágoston', 'IFUA Horváth & Partners', 'Kereskedelmi kisszervezet vezetője', 'agoston.balazs@hotmail.com', '(30) 308-7837'),

(45, 'nincskep.png', 'Szőke Egon', 'Dorogi hanglemezgyár', 'Fordító, tolmács', 'egon.szoke@gmail.com', '(70) 881-3975'),

(46, 'nincskep.png', 'Kis Aida', 'CTnetwork', 'Számviteli és pénzügyi részegység vezetője', 'kis.aida@gmail.com', '(30) 758-4737'),

(47, 'nincskep.png', 'Balog Flóra', 'FM1 Konzorcium', 'Ipari kisszervezet vezetője', 'balog.flora@freemail.hu', '(70) 179-3710'),

(48, 'nincskep.png', 'Jakab Mária', 'Aegon Magyarország Pénztárszolgáltató Rt.', 'Fodrász, borbély', 'jakab.maria@yahoo.com', '(30) 423-1600'),

(49, 'nincskep.png', 'Papp Tihamér', 'Murányi Unió', 'Bérelszámoló', 'papp\_tihamer@freemail.hu', '(70) 518-7226'),

(50, 'nincskep.png', 'Balázs Szeréna', 'Diósgyőri Gépgyár', 'Felsőfokú tanintézeti tanár, oktató', 'balazs.szerena@mailbox.hu', '(20) 885-1579'),

(51, 'nincskep.png', 'Fábián Annamária', 'Nemzeti Infrastruktúra-fejlesztő Zrt.', 'Épületasztalos', 'a.fabian@indamail.hu', '(20) 649-1835'),

(52, 'nincskep.png', 'Jakab Ulrika', 'Tarr Kft.', 'Szakasszisztens (orvosi)', 'jakab\_ulrika@gmail.com', '(20) 939-3100'),

(53, 'nincskep.png', 'Sándor Veronika', 'Dunai Vasmű', 'Erősáramú villamosmérnök', 'veronika\_sandor@gmail.com', '(30) 377-6067'),

(54, 'nincskep.png', 'Bogdán Bánk', 'MAL Magyar Alumínium Termelő és Kereskedelmi Zrt.', 'Kézi anyagmozgató, csomagoló', 'b.bogdan@gmail.com', '(30) 830-1382'),

(55, 'nincskep.png', 'Pásztor Elizeus', 'ÁPISZ', 'Mechanikai műszerész', 'e.pasztor@yahoo.com', '(30) 756-7914'),

(56, 'nincskep.png', 'Szabó Gergely', 'Dorogi hőerőmű', 'Általános iskolai tanár, tanító, oktató', 'g.szabo@freemail.hu', '(70) 783-5243'),

(57, 'nincskep.png', 'Veres Aranka', 'Fehér Ferenc Precíziós Mechanikai és Gépműhelye', 'Egyéb szakképzett oktatók, nevelők', 'veres.aranka@freemail.hu', '(30) 707-4449'),

(58, 'nincskep.png', 'Simon Brúnó', 'Auguszt Cukrászda', 'Kereskedelmi részegység vezetője', 'bruno\_simon@gmail.com', '(70) 756-1051'),

(59, 'nincskep.png', 'Boros Zakariás', 'Melléktermék- és Hulladékhasznosító Vállalat', 'Kézi anyagmozgató, csomagoló', 'zakarias\_boros@mailbox.hu', '(70) 636-4264'),

(60, 'nincskep.png', 'Farkas Tétény', 'IFUA Horváth & Partners', 'Háztömbfelügyelő, házfelügyelő, házgondnok', 'farkas.teteny@gmail.com', '(20) 440-1633'),

(61, 'nincskep.png', 'Virág Dzsenifer', 'Vodafone', 'Szakasszisztens (orvosi)', 'dzsenifer.virag@freemail.hu', '(20) 937-9599'),

(62, 'nincskep.png', 'Somogyi Cecília', 'ANY Biztonsági Nyomda', 'Oktatási szolgáltatási kisszervezet vezetője', 'c.somogyi@gmail.com', '(70) 187-3210'),

(63, 'nincskep.png', 'Magyar Márta,Flóra', 'ProfiPower', 'Területi közig., igazságszolg. közép vezetője', 'm.magyar@hotmail.com', '(70) 906-8973'),

(64, 'nincskep.png', 'Oláh Paula', 'Sanoma Budapest Zrt.', 'Egyéb növénytermesztési és kertészeti foglalkozások', 'olah.paula@yahoo.com', '(70) 647-6588'),

(65, 'nincskep.png', 'Pataki Matilda', 'Globex Holding', 'Mosónő, vasalónő', 'pataki.matilda@gmail.com', '(20) 369-3664'),

(66, 'nincskep.png', 'Boros Szaniszló', 'Petőfi Nyomda', 'Szakács', 'szaniszlo.boros@indamail.hu', '(70) 725-7827'),

(67, 'nincskep.png', 'Orosz Egyed', 'Salgótarjáni Öblösüveggyár', 'Oktatási szolgáltatási kisszervezet vezetője', 'orosz\_egyed@gmail.com', '(20) 988-2478'),

(68, 'nincskep.png', 'Dudás Valentin', 'FŐTÁV Zrt.', 'Bér- és társadalombiztosítási ügyintéző', 'valentin.dudas@gmail.com', '(70) 485-4700'),

(69, 'nincskep.png', 'Király Jusztina', 'Offset és Játékkártya Nyomda Zrt.', 'Gyógypedagógus', 'kiraly\_jusztina@citromail.hu', '(30) 603-8769'),

(70, 'nincskep.png', 'Balog Frigyes', 'MVM Magyar Villamos Művek Zrt.', 'Szakápoló', 'balog.frigyes@gmail.com', '(70) 586-9614'),

(71, 'nincskep.png', 'Borbély Kunigunda', 'Telefongyár', 'Szállítási és raktározási részegység közv. term.ir.', 'k.borbely@yahoo.com', '(70) 270-9717'),

(72, 'nincskep.png', 'Dudás Oszkár', 'Röck István Gépgyára', 'Iparművész', 'dudas.oszkar@vipmail.hu', '(20) 637-5737'),

(73, 'nincskep.png', 'Deák Paulina', 'Auguszt Cukrászda', 'Egyéb irodai jellegű foglalkozások', 'p.deak@indamail.hu', '(70) 124-7532'),

(74, 'nincskep.png', 'Fazekas Richárd', 'Miskolci Gőztéglagyár', 'Ipari részegység vezetője', 'r.fazekas@gmail.com', '(70) 494-6946'),

(75, 'nincskep.png', 'Hajdú Valér', 'Magyar Pénzügyi Közvetítő Zrt.', 'Fegyveres erők felsőfokú foglalkozásai', 'valer.hajdu@freemail.hu', '(30) 650-3078'),

(76, 'nincskep.png', 'Kozma Kartal', 'Moskovits Sámuel vasöntödéje és gépgyára', 'Építőipari részegység vezetője', 'kartal.kozma@yahoo.com', '(70) 559-2828'),

(77, 'nincskep.png', 'Borbély Mária', 'Versys Clinics', 'Vezeték- és csőhálózat-szerelő', 'maria.borbely@mailbox.hu', '(30) 569-8800'),

(78, 'nincskep.png', 'Budai Franciska', 'Herendi Porcelánmanufaktúra', 'Védőnő', 'budai\_franciska@yahoo.com', '(30) 134-6204'),

(79, 'nincskep.png', 'Halász Emánuel', 'Labor Műszeripari Művek', 'Egyéb magasan képzett ügyintézők', 'halasz\_emanuel@mailbox.hu', '(70) 512-1472'),

(80, 'nincskep.png', 'Virág Szilárd', 'K&H Bank', 'Egyéb magasan képzett ügyintézők', 'virag\_szilard@gmail.com', '(70) 895-8552'),

(81, 'nincskep.png', 'Hajdú Árpád', 'Zsolnay Porcelánmanufaktúra Zrt.', 'Szarvasmarhatartó és -tenyésztő', 'hajdu.arpad@gmail.com', '(20) 885-1150'),

(82, 'nincskep.png', 'Bogdán Imre', 'Borsodi Sörgyár Zrt.', 'Fegyveres erők felsőfokú foglalkozásai', 'imre.bogdan@gmail.com', '(30) 887-6175'),

(83, 'nincskep.png', 'Faragó Imelda', 'Pick Szeged Zrt.', 'Egyéb szakképzett oktatók, nevelők', 'i.farago@indamail.hu', '(70) 816-5736'),

(84, 'nincskep.png', 'Tamás Filippa', 'Dorogi hőerőmű', 'Óvónő', 'tamas\_filippa@hotmail.com', '(70) 846-6079'),

(85, 'nincskep.png', 'Budai Gellért', 'Petőfi Nyomda', 'Ipari kisszervezet vezetője', 'g.budai@citromail.hu', '(30) 832-3064'),

(86, 'nincskep.png', 'Szilágyi Atanáz', 'MAL Magyar Alumínium Termelő és Kereskedelmi Zrt.', 'Szakorvos', 'szilagyi\_atanaz@vipmail.hu', '(30) 333-6057'),

(87, 'nincskep.png', 'Bogdán Botond', 'Invitel', 'Gyengeáramú villamosmérnök', 'bogdan.botond@gmail.com', '(30) 891-4250'),

(88, 'nincskep.png', 'Szűcs Margit', 'MATERal', 'Fegyveres erők középfokú foglalkozásai', 'margit.szucs@yahoo.com', '(70) 187-6512'),

(89, 'nincskep.png', 'Szőke Placida', 'Gyulaj Erdészeti és Vadászati Zrt.', 'Cipész, cipőkészítő, -javító', 'szoke.placida@freemail.hu', '(70) 661-2364'),

(90, 'nincskep.png', 'Gál János', 'ELTE Idegennyelvi Továbbképző Központ', 'Szállító- és rakodómunkás', 'j.gal@freemail.hu', '(30) 198-4578'),

(91, 'nincskep.png', 'Lengyel Izidor', 'Gamma (gyár)', 'Hegesztő, lángvágó', 'izidor\_lengyel@gmail.com', '(30) 149-9353'),

(92, 'nincskep.png', 'Németh Géza', 'Pécsi Bőrgyár', 'Postai kézbesítő', 'g.nemeth@gmail.com', '(30) 168-6167'),

(93, 'nincskep.png', 'Hegedűs Péter', 'Buda-Cash Brókerház Zrt.', 'Piackutató, marketingtevékenységet végző', 'p.hegedus@hotmail.com', '(70) 673-9769'),

(94, 'nincskep.png', 'Váradi Leó', 'Senior', 'Egészségügyi, oktatási szolgáltatási foglalkozások', 'l.varadi@gmail.com', '(30) 376-3507'),

(95, 'nincskep.png', 'Vincze Vince', 'Budapesti Finomkötöttárugyár', 'Egyéb szakképzett oktatók, nevelők', 'vince\_vincze@indamail.hu', '(30) 114-3636'),

(96, 'nincskep.png', 'Fekete Eulália', 'REMIX Rádiótechnikai Vállalat', 'Ipari kisszervezet vezetője', 'eulalia.fekete@gmail.com', '(20) 212-7642'),

(97, 'nincskep.png', 'Pásztor Apor', 'Melléktermék- és Hulladékhasznosító Vállalat', 'Egyéb fémmegmunkálók, felületkezelők', 'a.pasztor@gmail.com', '(30) 649-4540'),

(98, 'nincskep.png', 'Oláh Emília', 'KDB Bank (Magyarország)', 'Mezőgazdasági vontatóvezető', 'olah.emilia@gmail.com', '(30) 660-4635'),

(99, 'nincskep.png', 'Balla Léda', 'Első Magyar Koncessziós Autópálya Rt.', 'Egyszerű szolgáltatási jellegű foglalkozások', 'l.balla@vipmail.hu', '(30) 878-3177'),

(100, 'nincskep.png', 'Hajdú Péter', 'Röck István Gépgyára', 'Egyéb növénytermesztési és kertészeti foglalkozások', 'p.hajdu@hotmail.com', '(30) 332-2332'),

## Adatbázis kapcsolat

A következőkben bemutatom, hogyan hozhatod létre a **PHP** alkalmazásod adatbázis-kapcsolatát a **Dotenv** használatával, hogy az érzékeny adatokat, például az adatbázis-hitelesítési adatokat, ne közvetlenül a kódban tárolja, hanem a .env fájlban. Ez a módszer biztonságosabb, különösen, ha verziókezelést használsz (pl. **Git**).

Természetesen vannak egyszerűbb módszerek is: <https://www.w3schools.com/php/php_mysql_connect.asp>

#### 1. ****PHP-Dotenv telepítése****

A kódrészlet első lépése a **Dotenv** csomag telepítése a **Composer** segítségével:

composer require vlucas/phpdotenv

Ez létrehozza a vendor könyvtárat és hozzáadja a csomaghoz szükséges fájlokat, hogy a környezeti változókat be tudja olvasni.

#### 2. ****.env fájl használata****

A .env fájl a projekt gyökérkönyvtárában lesz létrehozva, és ebben tárolódnak az adatbázis eléréséhez szükséges adatok:

DBHOST=localhost

DBUSER=root

DBPASS=secret\_password

DBNAME=database\_name

#### 3. ****Biztonság: .gitignore fájl****

A .env fájlban tárolt érzékeny információk biztonságos kezelése érdekében a .gitignore fájlhoz kell adni a .env fájlt:

.env

Ez megakadályozza, hogy a .env fájl véletlenül bekerüljön a verziókezelésbe.

#### 4. ****Adatbázis kapcsolat inicializálása****

A kódrészlet betölti a .env fájl tartalmát és használja az adatbázis-kapcsolat létrehozásához:

// Adatbázis kapcsolódás

define("DBHOST", $\_ENV['DBHOST']);

define("DBUSER", $\_ENV['DBUSER']);

define("DBPASS", $\_ENV['DBPASS']);

define("DBNAME", $\_ENV['DBNAME']);

// Kapcsolat létrehozása

$dbconn = @mysqli\_connect(DBHOST, DBUSER, DBPASS, DBNAME) or die("Hiba az adatbázis csatlakozásakor!");

// Karakterkódolás beállítása

mysqli\_query($dbconn, "SET NAMES utf8");

Az adatbáziskapcsolat létrehozása a környezeti változókkal történik, amelyek a .env fájlban vannak definiálva.

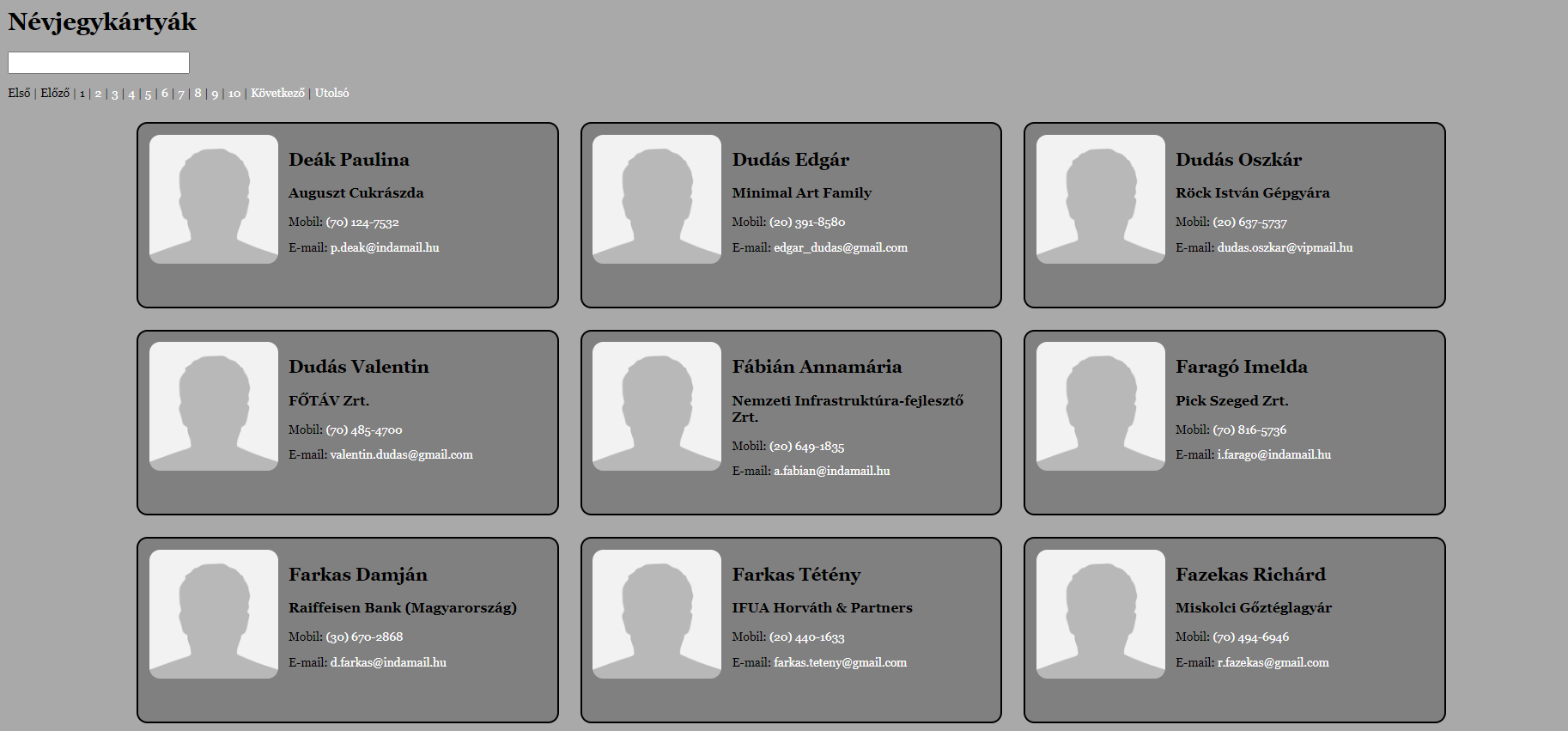
#### 5. ****Fontos lépések****

* **Composer használata**: A Composer által kezelt csomagok közül a **Dotenv** felelős a környezeti változók betöltéséért.
* **Karakterkódolás**: A "SET NAMES utf8" parancs biztosítja, hogy az adatbázis-műveletek UTF-8 karakterkódolással történjenek.

Ez a megközelítés biztonságosabbá teszi az adatbázis-csatlakozás kezelését, mivel az érzékeny adatok nincsenek közvetlenül a kódban. Ezzel csökkented az adatok kiszivárgásának kockázatát, különösen nyílt forrású projektek vagy megosztott környezetek esetén.

## Read művelet megvalósítása

A következőkben különböző módszereket mutatok be, hogyan olvashatunk be adatokat az adatbázisból, és jeleníthetjük meg a böngészőben:



Amint az látható, egy lapzó is beépítésre került, hogy ne kelljen a felhasználónak a végtelenségig lefelé görgetnie.

### READ művelet egyetlen fájl létrehozásával

Ez a PHP kód egy névjegykezelő alkalmazás részét képezi, amely lehetővé teszi a névjegyek megjelenítését az adatbázisból, valamint keresési és lapozási lehetőségeket biztosít a felhasználóknak. Az alábbiakban részletesen bemutatom a kód működését:

#### 1. ****Adatbázis kapcsolat létrehozása****

A kód az elején betölti a kapcsolat.php fájlt, amely valószínűleg az adatbázis-kapcsolatot definiálja.

require "kapcsolat.php";

#### 2. ****Alapvető változók és beállítások****

* A kód az összes névjegyet lekéri az adatbázisból egy egyszerű **SELECT** lekérdezéssel:

$sql = "SELECT \* FROM nevjegyek";

$eredmeny = mysqli\_query($dbconn, $sql);

* Az összes névjegy számát megszámolja:

$osszes = mysqli\_num\_rows($eredmeny);

* Ezután beállítja, hogy oldalanként hány névjegyet jelenítsen meg:

$mennyit = 9;

* Kiszámolja, hány oldal szükséges a névjegyek megjelenítéséhez:

$lapok = ceil($osszes / $mennyit); // A "ceil" felfelé kerekíti az eredményt

* Az aktuális oldalt a **GET** paraméterekből (oldal) olvassa be. Ha nincs beállítva, az aktuális oldal az 1-es lesz.

$aktualis = (isset($\_GET['oldal'])) ? (int)$\_GET['oldal'] : 1;

* Kiszámítja, hogy melyik rekordnál kezdje az adatbázisból az aktuális oldal lekérdezését:

$honnan = ($aktualis - 1) \* $mennyit;

#### 3. ****Lapozó felépítése****

* A lapozó vezérlők (pl. **Első**, **Előző**, **Következő**, **Utolsó**) HTML formátumban jelennek meg. A kód dinamikusan hozza létre az egyes oldalak közötti linkeket a lapozáshoz:

$lapozo = "<p>";

$lapozo .= ($aktualis != 1) ? "<a href=\"?oldal=1\">Első</a> | " : "Első | ";

$lapozo .= ($aktualis > 1 && $aktualis <= $lapok) ? "<a href=\"?oldal=" . ($aktualis - 1) . "\">Előző</a> | " : "Előző | ";

* A ciklus végigmegy minden oldalon, és linkeket generál az oldalszámokhoz:

for ($oldal = 1; $oldal <= $lapok; $oldal++) {

$lapozo .= ($aktualis != $oldal) ? "<a href=\"?oldal={$oldal}\">{$oldal}</a> | " : $oldal . " | ";

}

#### 4. ****Keresés megvalósítása****

* A keresési funkció a POST metódussal történik. Ha a felhasználó valamilyen kifejezést keres, az input értékét betölti a kód:

$kifejezes = (isset($\_POST['kifejezes'])) ? $\_POST['kifejezes'] : "";

* Az SQL lekérdezés úgy módosul, hogy a keresett kifejezést figyelembe vegye. A név, cég, mobil, vagy e-mail mezőkben keres:

$sql = "SELECT \*

FROM nevjegyek

WHERE (

nev LIKE '%{$kifejezes}%'

OR cegnev LIKE '%{$kifejezes}%'

OR mobil LIKE '%{$kifejezes}%'

OR email LIKE '%{$kifejezes}%'

)

ORDER BY nev ASC

LIMIT {$honnan}, {$mennyit}";

* Az SQL lekérdezés eredményét ismét lefuttatja és feldolgozza:

$eredmeny = mysqli\_query($dbconn, $sql);

#### 5. ****Eredmények megjelenítése****

* Ha nincs találat, megjelenik egy üzenet:

if (@mysqli\_num\_rows($eredmeny) < 1) {

$kimenet = "<article><h2>Nincs találat a rendszerben!</h2></article>\n";

}

* Ha van találat, akkor minden egyes névjegyet HTML elemekbe ágyazva jelenít meg:

while ($sor = mysqli\_fetch\_assoc($eredmeny)) {

$kimenet .= "<article>

<img src=\"kepek/{$sor['foto']}\" alt=\"{$sor['nev']}\">

<h2>{$sor['nev']}</h2>

<h3>{$sor['cegnev']}</h3>

<p>Mobil: <a href=\"tel:{$sor['mobil']}\">{$sor['mobil']}</a></p>

<p>E-mail: <a href=\"mailto:{$sor['email']}\">{$sor['email']}</a></p>

</article>\n";

}

#### 6. ****HTML oldal felépítése****

* A kimeneti tartalom a HTML oldalon kerül megjelenítésre. A lapozó vezérlők és a névjegyek dinamikusan kerülnek be az oldalra:

<?php print $lapozo; ?>

<div class="container">

<?php print $kimenet; ?>

</div>

#### Összefoglalás

Ez a kód egy egyszerű névjegykártya-kezelő alkalmazás PHP-ben, amely:

1. **Adatbázisból** olvassa be a névjegyeket.
2. **Lapozó funkcióval** osztja fel a névjegyeket oldalanként.
3. Keresési funkcióval biztosít lehetőséget a felhasználóknak a névjegyek szűrésére.
4. Az eredmények dinamikusan kerülnek megjelenítésre, és a **képek**, **nevek**, **cégek**, valamint a kapcsolattartási információk is elérhetők a felhasználó számára.

A következő példában bemutatom, hogyan tudsz kiszervezni kódrészleteket, hogy az átláthatóbb legyen, valamint újra hasznosítható.

#### index2.html ****(PHP és HTML)****

Ez a fájl az alkalmazás fő belépési pontja, amely névjegykártyákat (üzleti kapcsolatok adatait) jelenít meg, keresési és lapozási funkcióval. Dinamikusan állítja elő a HTML-t a lekérdezett adatbázis eredményei alapján.

#### ****Fontosabb szakaszok:****

1. **Függőségek:**
   * A db\_functions.php fájl szükséges az adatbázis-műveletekhez (pl. névjegyek lekérése).
   * A pagination.php fájl a lapozási logikát biztosítja.
   * A header("Expires: 0") biztosítja, hogy az oldal ne kerüljön gyorsítótárba, így mindig friss adatok töltődnek be.
2. **Kezdeti beállítások:**
   * Lekérdezi az összes rekord számát a getTotalRecords($dbconn) függvénnyel.
   * Meghatározza, hány elemet jelenítsen meg oldalanként ($itemsPerPage = 9).
   * Kiszámítja az összes szükséges oldalszámot, és beállítja az aktuális oldalt az URL-ből ($\_GET['oldal']).
   * A keresési kifejezést az űrlap POST metódusából veszi át, ha van megadva ($\_POST['kifejezes']).
3. **Adatok lekérdezése:**
   * A fetchContacts() függvénnyel lekérdezi az adatbázisból a névjegyeket, a keresési kifejezés és az adott oldal alapján.
   * Ha nem találhatóak névjegyek, egy megfelelő üzenetet jelenít meg.
4. **Lapozás:**
   * A lapozó linkeket a generatePagination() függvény generálja az aktuális oldal és az összes oldal számával.
5. **HTML kimenet:**
   * Minden névjegy egy article blokkban jelenik meg, beleértve a képet, nevet, cégnevet, mobilszámot és emailt.
   * A kereső űrlap és a dinamikusan generált lapozó linkek az oldal tetején találhatók.

<?php

require("db\_functions.php"); // Az adatbázis műveletekért felelős függvények betöltése

require("pagination.php"); // A lapozási logikát tartalmazó fájl betöltése

header("Expires: 0"); // Az oldal gyorsítótárasodásának megakadályozása

// Az adatbázisból lekérdezi az összes rekord számát

$totalRecords = getTotalRecords($dbconn);

// Lapozási beállítások: hány elemet jelenítsünk meg oldalanként

$itemsPerPage = 9;

#### 2. pagination.php ****(Lapozás logika)****

Ez a fájl a lapozási logikát valósítja meg, ami biztosítja, hogy a felhasználók több oldalon keresztül navigálhassanak az adatok között.

#### ****Függvények:****

1. **generatePagination($currentPage, $totalPages)**
   * Ez a függvény generálja a lapozó linkeket, figyelembe véve az aktuális oldalt és az összes oldalszámot.
   * Az első, előző, következő és utolsó oldalak linkjeit is dinamikusan hozza létre a megfelelő feltételek alapján.

Itt szintén a lapozó megvalósítását láthatod, más módszerrel függvénybe szervezve:

<?php

function generatePagination($currentPage, $totalPages) {

$pagination = "<p>";

// Link az első oldalra (ha nem az első oldalon vagyunk)

$pagination .= ($currentPage != 1) ? "<a href=\"?oldal=1\">Első</a> | " : "Első | ";

// Link az előző oldalra (ha nem az első oldalon vagyunk)

$pagination .= ($currentPage > 1 && $currentPage <= $totalPages) ? "<a href=\"?oldal=" . ($currentPage - 1) . "\">Előző</a> | " : "Előző | ";

// Az összes oldal linkjeit generáljuk

for ($page = 1; $page <= $totalPages; $page++) {

$pagination .= ($currentPage != $page) ? "<a href=\"?oldal={$page}\">{$page}</a> | " : $page . " | ";

}

// Link a következő oldalra (ha nem az utolsó oldalon vagyunk)

$pagination .= ($currentPage < $totalPages) ? "<a href=\"?oldal=" . ($currentPage + 1) . "\">Következő</a> | " : "Következő | ";

// Link az utolsó oldalra (ha nem az utolsó oldalon vagyunk)

$pagination .= ($currentPage != $totalPages) ? "<a href=\"?oldal={$totalPages}\">Utolsó</a>" : "Utolsó";

$pagination .= "</p>";

return $pagination; // A lapozó linkek visszaadása

}

#### 3. db\_functions.php ****(Adatbázis függvények)****

Ez a fájl tartalmazza az adatbázis interakciókat kezelő függvényeket. Az adatokat a kapcsolat.php fájlban definiált adatbáziskapcsolatot használva éri el.

#### ****Függvények:****

1. **fetchContacts($dbconn, $kifejezes = "", $honnan = 0, $mennyit = 9)**
   * Lekérdezi az adatbázisból a névjegyeket, beleértve a keresési kifejezés alapján történő szűrést is.
   * Az adatokat egy tömbbe olvassa be, és visszaadja.
   * Hibakezelést is tartalmaz, ha a lekérdezés nem sikerül.
2. **getTotalRecords($dbconn)**
   * Lekérdezi az adatbázisból az összes névjegy számát.
   * A visszatérési érték egy egész szám, ami a rekordok számát tartalmazza.

#### ****Példa kódkommentek:****

Ebben a kódrészletben figyeljétek meg a kommentezést, a /\*\* \*/ segítéségével paraméterezheted a kommentjeidet, ha az adott parancsra, változóra húzod az egeret, ha hozzá csatolt megjegyzést megjeleníti a felugró ablakban.

<?php

require("kapcsolat.php"); // Az adatbázis kapcsolati beállításokat betöltő fájl

/\*\*

\* Lekérdezi az összes névjegyet az adatbázisból, és visszaadja őket.

\* @param mysqli $dbconn Az adatbázis kapcsolati objektum

\* @param string $kifejezes A keresési kifejezés

\* @param int $honnan A lekérdezett adatok kezdő indexe

\* @param int $mennyit Hány rekordot jelenítsünk meg oldalanként

\* @return array Az adatbázisból lekért adatok tömbje

\*/

function fetchContacts($dbconn, $kifejezes = "", $honnan = 0, $mennyit = 9) {

// SQL lekérdezés, amely szűri a névjegyeket a keresési kifejezés alapján

$sql = "SELECT \* FROM nevjegyek

WHERE (nev LIKE '%{$kifejezes}%'

OR cegnev LIKE '%{$kifejezes}%'

OR mobil LIKE '%{$kifejezes}%'

OR email LIKE '%{$kifejezes}%')

ORDER BY nev ASC

LIMIT {$honnan}, {$mennyit}";

// Lekérdezés végrehajtása

$result = mysqli\_query($dbconn, $sql);

// Hibakezelés, ha a lekérdezés nem sikerül

if (!$result) {

http\_response\_code(500); // HTTP 500-as hiba küldése

echo json\_encode(["error" => "Adatbázis lekérdezési hiba: " . mysqli\_error($dbconn)]);

exit();

}

$data = array(); // Üres tömb létrehozása az adatoknak

while ($row = mysqli\_fetch\_assoc($result)) {

$data[] = $row; // Adatok hozzáadása a tömbhöz

}

return $data; // Visszaadja az adatok tömbjét

}

/\*\*

\* Visszaadja az összes rekord számát az adatbázisból.

\* @param mysqli $dbconn Az adatbázis kapcsolati objektum

\* @return int Az összes rekord száma

\*/

function getTotalRecords($dbconn) {

// SQL lekérdezés az összes rekord számának lekérdezésére

$sql = "SELECT COUNT(\*) AS count FROM nevjegyek";

// Lekérdezés végrehajtása

$result = mysqli\_query($dbconn, $sql);

$row = mysqli\_fetch\_assoc($result); // Eredmény sor beolvasása

return (int)$row['count']; // Az összes rekord száma

}

Ezekkel a kommentekkel és magyarázatokkal a kód könnyebben érthető és karbantartható.

#### PHPDoc stílusú kommentezés

A PHPDoc egy dokumentációs szabvány, amely lehetővé teszi a kódolók számára, hogy pontosan kommentálják a PHP kódjukat, így más fejlesztők és maga a fejlesztő is könnyebben megérthesse a kód működését. Ezenkívül az IDE-k (például PHPStorm) és a dokumentációs eszközök automatikusan feldolgozhatják ezeket a kommenteket, hogy hasznos információkat nyújtsanak.

#### Kommentezés szintaxisa

A PHPDoc kommentek többsora zárójelben történik:

/\*\*

\* Ez a komment leírja, hogy mit csinál a funkció.

\* További részletek is itt jelennek meg.

\*/

#### Főbb PHPDoc annotációk

1. **@var** Használat: Egy változó típusát és célját adja meg.  
   Formátum: @var típus $változónév  
   Példa:

/\*\* @var int $age A felhasználó életkora. \*/

public $age;

1. **@param** Használat: Egy függvény/módszer paramétereit dokumentálja.  
   Formátum: @param típus $paraméterLeírás  
   Példa:

/\*\*

\* Kiszámítja a felhasználó életkorát.

\*

\* @param int $yearOfBirth A születési év.

\* @return int A felhasználó életkora.

\*/

public function calculateAge($yearOfBirth) {

return date('Y') - $yearOfBirth;

}

1. **@return**  
   Használat: Egy függvény visszatérési értékét írja le.  
   Formátum: @return típus Leírás  
   Példa:

/\*\*

\* @return string A felhasználó teljes neve.

\*/

public function getFullName() {

return $this->firstName . ' ' . $this->lastName;

}

1. **@method** Használat: Egy osztály által dinamikusan elérhető függvényeket dokumentál.  
   Formátum: @method típus methodName(paramétertípus $paraméterLeírás)  
   Példa:

/\*\*

\* @method void prepare(string $sql)

\* @method void bind\_param(string $types, mixed ...$vars)

\*/

1. **@property** Használat: Egy osztály olyan tulajdonságait írja le, amelyeket nem közvetlenül definiáltak, de elérhetők dinamikusan.  
   Formátum: @property típus $tulajdonságLeírás  
   Példa:

/\*\*

\* @property string $name A felhasználó neve.

\*/

1. **@throws** Használat: Ha egy függvény kivételt (exception) dobhat, dokumentálja, hogy melyik típusú kivétel dobható.  
   Formátum: @throws ExceptionTípus Leírás  
   Példa:

/\*\*

\* @throws InvalidArgumentException Ha a bemeneti adat érvénytelen.

\*/

public function setAge($age) {

if ($age < 0) {

throw new InvalidArgumentException('Érvénytelen életkor');

}

}

1. **@deprecated** Használat: Jelzi, hogy egy függvény, metódus, vagy tulajdonság elavult, és nem ajánlott a használata.  
   Formátum: @deprecated Leírás  
   Példa:

/\*\*

\* @deprecated Ez a metódus elavult. Használja a getNewData() függvényt.

\*/

public function getData() {

// régi logika

}

### Gyakran használt PHPDoc annotációk összefoglalása

* **@var**: Egy változó típusát írja le.
* **@param**: Függvény vagy metódus paramétereinek leírása.
* **@return**: A visszatérési érték típusát és célját dokumentálja.
* **@method**: Dinamikusan elérhető metódusok leírása.
* **@property**: Olyan osztály tulajdonságait írja le, amelyeket dinamikusan lehet elérni.
* **@throws**: Kivétel dobásának leírása.
* **@deprecated**: Elavult kód jelölése.

### Gyakorlati feladatok

1. **Annotációk azonosítása és helyes használata:**
   * Írj egy egyszerű PHP osztályt, amely egy adatbázis csatlakozást és egy select() függvényt tartalmaz.
   * Kommentezd az osztályt PHPDoc annotációkkal, például @var, @param, @return és @throws.
2. **Dokumentáció generálása:**
   * Használj eszközt (pl. phpDocumentor), hogy automatikusan generálj dokumentációt a kommentjeid alapján.

<?php

/\*\*

\* Adatbázis kapcsolatot kezelő osztály.

\*

\* Ez az osztály kapcsolatot létesít egy MySQL adatbázissal,

\* és lehetővé teszi az SQL utasítások végrehajtását.

\*/

class DatabaseConnection {

/\*\*

\* @var mysqli $connection A MySQL adatbázis kapcsolat objektuma.

\*/

private $connection;

/\*\*

\* Létrehozza az adatbázis kapcsolatot.

\*

\* @param string $host Az adatbázis kiszolgáló neve.

\* @param string $user Az adatbázis felhasználóneve.

\* @param string $password Az adatbázis felhasználó jelszava.

\* @param string $database Az adatbázis neve.

\*

\* @throws Exception Ha nem sikerül a kapcsolat létrehozása.

\*/

public function \_\_construct($host, $user, $password, $database) {

$this->connection = new mysqli($host, $user, $password, $database);

if ($this->connection->connect\_error) {

throw new Exception('Kapcsolódási hiba: ' . $this->connection->connect\_error);

}

}

/\*\*

\* Végrehajt egy SELECT lekérdezést és visszaadja az eredményt.

\*

\* @param string $sql A végrehajtandó SQL lekérdezés.

\* @param int $id A lekérdezésben használt azonosító.

\*

\* @return array A lekérdezés eredménye tömbként.

\*/

public function select($sql, $id) {

$stmt = $this->connection->prepare($sql);

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

$stmt->execute();

$result = $stmt->get\_result();

return $result->fetch\_all(MYSQLI\_ASSOC);

}

/\*\*

\* Bezárja az adatbázis kapcsolatot.

\*/

public function close() {

$this->connection->close();

}

}

?>

1. **PHPDoc dokumentáció generálása**

Ehhez a kódhoz használhatod a phpDocumentor nevű eszközt, amely automatikusan feldolgozza a PHPDoc stílusú kommenteket, és HTML alapú dokumentációt készít belőlük.

* Telepítsd a phpDocumentor eszközt:

composer require --dev phpdocumentor/phpdocumentor

* Futtasd a phpDocumentor-t a projekt könyvtárában:

./vendor/bin/phpdoc

Ez generál egy HTML fájlt, ahol a fenti kód dokumentációját szépen formázva megtekintheted.

### Végpont azaz JSON kimenet generálása

A következő példában végpontokat hozok létre, amelyek JSON végpontot generálnak, amit beolvasunk különböző módszerekkel. Ez a megközelítés már megfelel a RESTful API követelményeinek.

Ez a kód egy egyszerű adatbázis-lekérdezést hajt végre, majd az eredményeket JSON formátumban adja vissza. A kód először lekéri az összes rekordot a nevjegyek nevű adatbázis táblából, és a visszakapott adatokat JSON formátumban alakítja át, amelyet a böngészőnek küld.

#### Részletes Magyarázat:

1. **Adatbázis kapcsolódás**:

require("kapcsolat.php");

* + A kapcsolat.php fájl tartalmazza az adatbázis-kapcsolat részleteit, például a szerver, felhasználó, jelszó és adatbázis nevét. Ezzel a require() paranccsal történik meg a csatlakozás az adatbázishoz.

1. **Adatbázis lekérdezés**:

$query = "SELECT \* FROM nevjegyek";

$result = mysqli\_query($dbconn, $query);

* + A SELECT \* FROM nevjegyek SQL utasítás lekéri az összes rekordot a nevjegyek táblából.
  + A mysqli\_query() függvény végrehajtja a lekérdezést az adatbázis kapcsolat ($dbconn) segítségével, és az eredményt a $result változóban tárolja.

1. **Adatok tömbbe rendezése**:

$data = [];

while ($row = mysqli\_fetch\_assoc($result)) {

$data[] = $row;

}

* + A mysqli\_fetch\_assoc() függvénnyel az adatokat soronként kérdezi le, és minden egyes sort asszociatív tömbként ad vissza, ahol a mezőnevek a kulcsok.
  + A while ciklus addig fut, amíg van lekérhető adat a $result eredményhalmazból, és ezeket az adatokat egyenként hozzáadja a $data tömbhöz.

1. **JSON formátumba alakítás**:

$json = json\_encode($data, JSON\_PRETTY\_PRINT | JSON\_UNESCAPED\_UNICODE);

* + A json\_encode() függvénnyel alakítja át a tömböt JSON formátumba.
  + A JSON\_PRETTY\_PRINT opció biztosítja, hogy a JSON kimenet szépen formázott legyen, új sorokkal és behúzásokkal.
  + A JSON\_UNESCAPED\_UNICODE opció gondoskodik arról, hogy a kimenet tartalmazza az ékezetes karaktereket anélkül, hogy azokat Unicode escape formában (\uXXXX) jelenítené meg.

1. **HTTP fejléc beállítása**:

header('Content-Type: application/json');

* + Ezzel a sorral a PHP a megfelelő HTTP fejléccel állítja be a válasz tartalomtípusát. A Content-Type: application/json fejléc megmondja a böngészőnek, hogy a visszaküldött adat formátuma JSON.

1. **JSON válasz visszaküldése**:

echo $json;

* + Az elkészített JSON-t egyszerűen kiírja a böngésző felé.

#### Példa Kimenet:

Ha a nevjegyek táblában két névjegy szerepel, akkor a kimenet így nézhet ki:

[

{

"nev": "Kiss János",

"cegnev": "Kiss Kft.",

"mobil": "06301234567",

"email": "kiss.janos@example.com"

},

{

"nev": "Nagy Éva",

"cegnev": "Nagy Bt.",

"mobil": "06309876543",

"email": "nagy.eva@example.com"

}

]

Nézzük ugyanennek a feladatnak egy másik megoldását is:

Ez a kód is ugyanazt az adatbázis-lekérdezést végezi, amely az összes adatot lekéri a nevjegyek táblából, majd az adatokat JSON formátumban adja vissza a felhasználónak. A kód tartalmazza a hibakezelést is, ami gondoskodik arról, hogy ha a lekérdezés sikertelen, akkor egy megfelelő hibajelzést küld vissza.

#### ****Részletes Magyarázat:****

1. **Adatbázis Kapcsolódás:**

require("kapcsolat.php");

* + Ez a sor betölti a kapcsolat.php fájlt, amely tartalmazza az adatbáziskapcsolat beállításait (például szervernév, felhasználónév, jelszó). Ezáltal létrejön a kapcsolat az adatbázissal, amit a későbbi lekérdezések során használunk.

1. **fetchContacts() Függvény:** Ez a függvény felelős az adatok lekérdezéséért a nevjegyek nevű adatbázis táblából.

$query = "SELECT \* FROM nevjegyek";

$result = mysqli\_query($dbconn, $query);

A SELECT \* FROM nevjegyek SQL utasítás minden adatot lekér az adatbázisból, amely a nevjegyek táblában található. A mysqli\_query() függvény végrehajtja ezt a lekérdezést, és az eredményt a $result változó tárolja.

* + **Hibakezelés:**

if (!$result) {

http\_response\_code(500);

echo json\_encode(["error" => "Adatbázis lekérdezési hiba: " . mysqli\_error($dbconn)]);

exit();

}

Ez a blokk ellenőrzi, hogy a lekérdezés sikeresen lefutott-e. Ha a mysqli\_query() függvény hibával tér vissza, a http\_response\_code(500) segítségével HTTP 500-as hibakódot küldünk, és JSON formátumban egy hibaüzenetet adunk vissza. A mysqli\_error($dbconn) az adatbázis hibaüzenetét is megjeleníti.

* + **Adatok Tömbe Rendezése:**

$data = [];

while ($row = mysqli\_fetch\_assoc($result)) {

$data[] = $row;

}

return $data;

Az eredményhalmazt a mysqli\_fetch\_assoc() segítségével kérjük le soronként, ahol minden sor asszociatív tömbként adódik vissza (a kulcsok a táblában lévő oszlopok nevei). Minden egyes sort hozzáadunk a $data nevű tömbhöz, és a függvény végül visszaadja ezt a tömböt.

1. **Adatok Lekérdezése:**

$data = fetchContacts($dbconn);

* + A fetchContacts($dbconn) függvény meghívásával lekérdezzük az adatbázisból az összes adatot, és eltároljuk a $data változóban.

1. **JSON Formátumba Alakítás:**

$json = json\_encode($data);

* + A json\_encode() függvénnyel a $data tömböt JSON formátumra alakítjuk. Ez a JSON formátum lesz az, amit a szerver válaszként visszaküld.

1. **HTTP Fejléc Beállítása:**

header('Content-Type: application/json');

* + A header() függvény beállítja a HTTP válasz fejléceit, ez esetben a tartalom típusát application/json-ra állítjuk. Ez megmondja a böngészőnek (vagy más kliensnek), hogy a szerver JSON formátumú adatokat küld vissza.

1. **JSON Válasz Visszaküldése:**
   * A echo $json paranccsal a kód kiírja a JSON formátumú adatokat a válaszban, amelyet a böngésző vagy más kliens megkap.

#### ****Összegzés:****

Ez a kód egy alapvető API funkciót valósít meg, amely egy adatbázisban található rekordokat JSON formátumban küld vissza. A kód ellenőrzi a lekérdezések sikerességét, és szükség esetén megfelelő hibajelzéseket küld. Ez az API hasznos lehet, ha névjegyadatokat szeretnénk egy webes alkalmazásban vagy külső rendszerekben felhasználni.

### Végpont beolvasása

A következő kód a névjegyek adatait egy JSON fájlból olvassa be, majd megjeleníti azokat az oldalon. A kód a névjegyeket HTML formátumban jeleníti meg, és lapozási funkciót is biztosít a lapozo.php fájl segítségével.

1. **JSON Fájl Beolvasása:**

$json\_data = file\_get\_contents('vegpont.php');

* + A file\_get\_contents() függvénnyel a vegpont.php fájl tartalmát olvassuk be, ami egy JSON fájl. Ez a fájl JSON formátumú adatokat ad vissza, amelyek tartalmazzák a névjegykártyák információit.

1. **Adatbázis és Lapozási Modulok Betöltése:**

require "kapcsolat.php";

require "lapozo.php";

* + A kapcsolat.php fájl az adatbázis-kapcsolat beállításait tartalmazza, amire szükség lehet a lapozási funkcióhoz vagy egyéb adatbázis-műveletekhez.
  + A lapozo.php fájl tartalmazza a lapozási logikát, amely a névjegyeket több oldalra bontja.

1. **JSON Adatok Dekódolása:**

$data = json\_decode($json\_data, true);

* + A json\_decode() függvénnyel a beolvasott JSON formátumú adatokat egy asszociatív tömbbé alakítjuk (true paraméterrel biztosítjuk, hogy tömb legyen, ne objektum). Ez a tömb tartalmazza a névjegykártyák adatait, amelyeket később megjelenítünk az oldalon.

1. **Adatok Ellenőrzése és Feldolgozása:**

if (!empty($data)) {

// JSON adatok feldolgozása

} else {

// Ha nincs adat a JSON fájlban

}

* + **Ellenőrzés:** A !empty($data) feltétel ellenőrzi, hogy vannak-e adatok a JSON fájlban. Ha nincsenek adatok, egy üres eredményt jelenít meg, jelezve, hogy nincs találat.
  + **Adatok feldolgozása:** Ha a JSON fájl tartalmaz adatokat, akkor egy foreach ciklussal végigmegyünk minden névjegyen, és dinamikusan HTML article blokkokat generálunk, amelyek tartalmazzák a névjegy képeit, nevét, cégnevét, telefonszámát, valamint email címét.

**HTML Kimenet Generálása:**

$kimenet .= "

<article>

<img src=\"kepek/{$sor['foto']}\" alt=\"{$sor['nev']}\">";

$kimenet .= "<h2>{$sor['nev']}</h2>";

$kimenet .= "<h3>{$sor['cegnev']}</h3>";

$kimenet .= "<p>Mobil: <a href=\"tel:{$sor['mobil']}\">{$sor['mobil']}</a></p>";

$kimenet .= "<p>E-mail: <a href=\"mailto:{$sor['email']}\">{$sor['email']}</a></p>

</article>\n";

* + Itt az adatokból HTML elemeket készítünk. Minden névjegyhez tartozik egy kép (<img>), a név (<h2>), a cég neve (<h3>), valamint a mobiltelefonszám és az email cím hivatkozásokként jelenik meg. Ez lehetővé teszi, hogy a felhasználó egy kattintással hívhassa vagy emailt küldhessen a névjegynek.

1. **Ha Nincs Találat:**

$kimenet = "<article><h2>Nincs találat a rendszerben!</h2></article>\n";

* + Ha a JSON fájl üres, akkor egy üzenetet jelenít meg, hogy "Nincs találat a rendszerben".

1. **HTML Fejléc és Stílusok:**

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<title>Névjegykártyák</title>

<link href="stilus.css" rel="stylesheet">

</head>

<body>

* + A HTML dokumentum fejlécében megadjuk, hogy a tartalom magyar nyelvű (lang="hu"), és hogy a karakterkódolás UTF-8, amely biztosítja, hogy az ékezetes karakterek megfelelően jelenjenek meg.
  + A stilus.css fájl tartalmazza a weboldal stílusait, amelyeket a névjegykártyák megjelenítéséhez használunk.

1. **Lapozás és Kimenet Megjelenítése:**

<?php print $lapozo; ?>

<?php print $kimenet; ?>

* + **Lapozás:** A print $lapozo sor a lapozási linkeket jeleníti meg, amelyeket a lapozo.php fájl generál.
  + **Kimenet:** A print $kimenet sor megjeleníti a dinamikusan generált névjegykártyákat.

1. **Oldal Struktúrája:**
   * Az oldal tartalma egy container div elemben jelenik meg, amely tartalmazza az oldalon lévő névjegykártyák listáját.

Ez a kód dinamikusan jelenít meg névjegykártyákat egy JSON forrásból származó adatok alapján. A kód tartalmaz egy ellenőrzést, hogy van-e adat a JSON fájlban, és megfelelő HTML formátumban jeleníti meg a névjegyeket. Ha nincs adat, akkor egy üzenetet jelenít meg, amely tájékoztatja a felhasználót. Ezen felül a lapozo.php biztosítja a lapozási funkciókat, amelyek lehetővé teszik a felhasználó számára, hogy több oldalon keresztül navigáljon a névjegykártyák között.

### Lapozó választó, hány rekord jelenjen meg egy lapon

Épp rendeltem egy webshopról, amikor eszembe jutott, jó lenne kiegészíteni a lapozó, olyan választási lehetőséggel, mint amit az adott webshopon láttam. Íme az eredmény:

Ez a PHP alapú kód egy weboldalon jeleníti meg a névjegykártyákat keresési és lapozási funkcióval. A felhasználó kereshet név, cég, mobil vagy email alapján, valamint kiválaszthatja, hogy hány elem jelenjen meg oldalanként (9, 30 vagy 60 elem). A kód dinamikusan generálja a lapozási linkeket is.

#### 1. ****Adatbázis-kapcsolat****

require("kapcsolat.php");

* A kapcsolat.php fájl tartalmazza az adatbázis-kapcsolat beállításait. Ez biztosítja, hogy a kód kapcsolatba lép az adatbázissal, ahol a névjegykártyák tárolódnak.

#### 2. ****Keresési kifejezés****

$kifejezes = $\_GET['kifejezes'] ?? "";

* A keresési kifejezést az URL-ből ($\_GET['kifejezes']) kapjuk meg. Ha a felhasználó nem adott meg keresési kifejezést, akkor az üres string lesz az alapértelmezett érték.
* A ?? (nullás egyesítés operátor) biztosítja, hogy ha a keresési kifejezés nem létezik, akkor az alapértelmezett érték egy üres string legyen.

#### 3. ****Oldalanként megjelenítendő elemek száma****

php

Kód másolása

$mennyit = isset($\_GET['mennyit']) ? (int)$\_GET['mennyit'] : 9;

* Az oldalanként megjelenítendő elemek számát a felhasználó határozhatja meg a legördülő listából. Az alapértelmezett érték 9. A $\_GET['mennyit'] az URL-ből veszi át az adatot.
* Ha a felhasználó nem adta meg ezt az értéket, akkor az alapértelmezés 9 elem.

#### 4. ****SQL lekérdezés: keresés és lapozás****

$sql = "SELECT \*

FROM nevjegyek

WHERE (

nev LIKE '%{$kifejezes}%'

OR cegnev LIKE '%{$kifejezes}%'

OR mobil LIKE '%{$kifejezes}%'

OR email LIKE '%{$kifejezes}%'

)";

* A nevjegyek táblából történő lekérdezés során a név, cég, mobil vagy email alapján szűrjük az adatokat. A LIKE operátor segítségével bármilyen egyezést keresünk a megadott keresési kifejezéssel.

#### 5. ****Lapozási beállítások****

$osszes = mysqli\_num\_rows($eredmeny);

$lapok = ceil($osszes / $mennyit);

$aktualis = (isset($\_GET['oldal'])) ? (int)$\_GET['oldal'] : 1;

$honnan = ($aktualis-1)\*$mennyit;

* **Összes rekord száma**: A mysqli\_num\_rows($eredmeny) adja meg, hogy hány rekord egyezik a keresési kifejezéssel.
* **Lapok száma**: A lapok számát az összes rekord és az oldalanként megjelenítendő elemek számának arányából számítjuk ki. A ceil() függvénnyel kerekítünk felfelé, hogy az utolsó oldal akkor is megjelenjen, ha maradnak fennmaradó rekordok.
* **Aktuális oldal**: Az aktuális oldalt az URL-ből ($\_GET['oldal']) kapjuk meg. Ha nincs megadva, akkor az 1. oldal jelenik meg.
* **Kezdő elem indexe**: A lapozás megkezdéséhez ki kell számolnunk, hogy hányadik rekordtól kell megjeleníteni az elemeket az adott oldalon. Ezt a (aktuális oldal - 1) \* elemek száma oldalanként képlettel számítjuk ki.

#### 6. ****Lapozási logika****

$lapozo = "<p>";

$lapozo.= ($aktualis != 1) ? "<a href=\"?oldal=1&kifejezes={$kifejezes}&mennyit={$mennyit}\">Első</a> | " : "Első | ";

$lapozo.= ($aktualis > 1 && $aktualis <= $lapok) ? "<a href=\"?oldal=".($aktualis-1)."&kifejezes={$kifejezes}&mennyit={$mennyit}\">Előző</a> | " : "Előző | ";

for ($oldal=1; $oldal<=$lapok; $oldal++) {

$lapozo.= ($aktualis != $oldal) ? "<a href=\"?oldal={$oldal}&kifejezes={$kifejezes}&mennyit={$mennyit}\">{$oldal}</a> | " : $oldal." | ";

}

$lapozo.= ($aktualis > 0 && $aktualis < $lapok) ? "<a href=\"?oldal=".($aktualis+1)."&kifejezes={$kifejezes}&mennyit={$mennyit}\">Következő</a> | " : "Következő | ";

$lapozo.= ($aktualis != $lapok) ? "<a href=\"?oldal={$lapok}&kifejezes={$kifejezes}&mennyit={$mennyit}\">Utolsó</a>" : "Utolsó";

$lapozo.= "</p>";

* **Első és Előző gomb**: Az első oldalra, illetve az előző oldalra navigáló linkeket csak akkor jelenítjük meg, ha nem az első oldalon vagyunk.
* **Következő és Utolsó gomb**: Ezek a linkek akkor jelennek meg, ha nem az utolsó oldalon vagyunk.
* **Oldalak közötti navigáció**: A for ciklus generálja a lapozó linkeket az oldalak között. Minden oldal kap egy linket, kivéve az aktuális oldalt, amely csak egy számként jelenik meg.

#### 7. ****Adatok lekérdezése lapozáshoz****

$sql .= " ORDER BY nev ASC

LIMIT {$honnan}, {$mennyit}";

$eredmeny = mysqli\_query($dbconn, $sql);

* A lapozáshoz hozzáadunk egy LIMIT klauzulát az SQL lekérdezéshez, amely megadja, hogy hány rekordot kérünk le, és honnan kezdve.

#### 8. ****Adatok megjelenítése****

if (@mysqli\_num\_rows($eredmeny) < 1) {

$kimenet = "<article>

<h2>Nincs találat a rendszerben!</h2>

</article>\n";

} else {

$kimenet = "";

while ($sor = mysqli\_fetch\_assoc($eredmeny)) {

$kimenet.= "<article>

<img src=\"kepek/{$sor['foto']}\" alt=\"{$sor['nev']}\">

<h2>{$sor['nev']}</h2>

<h3>{$sor['cegnev']}</h3>

<p>Mobil: <a href=\"tel:{$sor['mobil']}\">{$sor['mobil']}</a></p>

<p>E-mail: <a href=\"mailto:{$sor['email']}\">{$sor['email']}</a></p>

</article>\n";

}

}

* **Nincs találat**: Ha nincs eredmény a lekérdezésre (vagyis nem található névjegy a megadott keresési kifejezés alapján), akkor megjelenítünk egy üzenetet.
* **Adatok megjelenítése**: Ha vannak találatok, akkor egy while ciklus segítségével minden találatot HTML formátumban jelenítünk meg. Minden névjegykártya tartalmazza a fotót, nevet, cégnevet, mobilt és emailt, valamint ezek kattintható hivatkozásokat is kapnak.

#### 9. ****Keresési űrlap és lapozó beállítások****

<form method="get" action="">

<input type="search" id="kifejezes" name="kifejezes" value="<?php echo htmlspecialchars($kifejezes); ?>">

<input type="submit" value="Keresés">

<label for="mennyit">Elemek száma oldalanként:</label>

<select id="mennyit" name="mennyit" onchange="this.form.submit()">

<option value="9" <?php echo $mennyit == 9 ? 'selected' : ''; ?>>9</option>

<option value="30" <?php echo $mennyit == 30 ? 'selected' : ''; ?>>30</option>

<option value="60" <?php echo $mennyit == 60 ? 'selected' : ''; ?>>60</option>

</select>

</form>

* **Keresési mező**: A felhasználó itt adhatja meg a keresési kifejezést.
* **Oldalankénti elemek száma**: Egy legördülő menü segítségével a felhasználó kiválaszthatja, hogy hány névjegyet szeretne látni oldalanként. Az onchange="this.form.submit()" azt biztosítja, hogy az űrlap automatikusan elküldésre kerüljön, ha a felhasználó megváltoztatja a kiválasztott értéket.

#### 10. ****Végleges HTML kimenet****

<?php echo $lapozo; ?>

<div class="container">

<?php echo $kimenet; ?>

</div>

* A lapozási linkek és a névjegykártyák HTML formátumban kerülnek megjelenítésre a weboldalon.

#### Összegzés

Ez a PHP kód egy komplex keresési és lapozási funkciót biztosít, amely lehetővé teszi a felhasználók számára, hogy hatékonyan böngésszenek az adatbázisban tárolt névjegykártyák között. Az oldal dinamikusan frissül, és a felhasználó kiválaszthatja, hány elemet szeretne látni oldalanként, valamint kereshet a névjegyek különböző mezői alapján.

### Végpont beolvasása AngularJs használatával

A következő megoldáshoz, szükséges némi AngularJs alapismeret.

Ez a kód egy AngularJS alapú webes alkalmazás, amely a névjegykártya adatokat egy PHP API-ból (a vegpont.php-ből) olvassa be JSON formátumban, majd az adatokat megjeleníti a felhasználó számára lapozási lehetőséggel. A lapozás lehetővé teszi, hogy több névjegykártyát jelenítsünk meg különböző oldalakon, egy oldalon egyszerre 9 névjegykártya jelenik meg.

#### 1. ****AngularJS Integráció****

Az AngularJS egy egyszerű és hatékony frontend keretrendszer, amely lehetővé teszi az interaktív elemek létrehozását egy weboldalon, mint például a dinamikus adatbetöltést és lapozást.

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

* Ezzel a sorral töltjük be az AngularJS könyvtárat. A 1.8.2 verziót használjuk, amely a stabil AngularJS verziók közé tartozik.

<html lang="hu" ng-app="nevjegyekApp">

* Az ng-app="nevjegyekApp" attribútum az AngularJS alkalmazás indításához szükséges. Ez határozza meg a root modult, amelyhez minden további logika és funkcionalitás kapcsolódik.

#### 2. ****Controller és Scope****

Az AngularJS vezérlője a névjegyek adatait kezeli, és megadja az oldalnak, hogyan dolgozzon az adatokkal és hogyan jelenítse meg azokat.

app.controller('NevjegyekController', ['$scope', '$http', function($scope, $http) {

// Adatok betöltése PHP API-ról

$http.get('vegpont.php').then(function(response) {

$scope.nevjegyek = response.data;

$scope.itemsPerPage = 9; // 9 elem megjelenítése oldalanként

$scope.currentPage = 1; // Kezdő oldal

}, function(error) {

console.error('Hiba történt az adatok lekérése közben:', error);

});

// Oldalak számítása

$scope.numberOfPages = function() {

return Math.ceil($scope.nevjegyek.length / $scope.itemsPerPage);

};

}]);

* A NevjegyekController vezérlő a $scope objektummal dolgozik, amely az AngularJS adatkapcsolatot biztosítja az adatmodell és a felhasználói felület között.
* **Adatok betöltése**: A $http.get() függvénnyel egy HTTP GET kérést küldünk a vegpont.php címre, ami a PHP API-n keresztül JSON formátumban küldi vissza a névjegyek adatait.
  + **Sikeres válasz esetén**: Az adatokat (response.data) a $scope.nevjegyek változóba mentjük. Ez az a változó, amelyet a HTML oldalon használunk az adatok megjelenítésére.
  + **Hibakezelés**: Ha hiba történik az adatok lekérése közben, egy üzenet kerül kiírásra a konzolra.
* **Lapozás**: A numberOfPages() függvény segítségével kiszámítjuk, hogy hány oldalt kell megjeleníteni. Ezt a névjegyek számából ($scope.nevjegyek.length) és az oldalon megjelenítendő névjegyek számából ($scope.itemsPerPage) számítjuk ki.

#### 3. ****Egyedi AngularJS szűrő a lapozáshoz****

app.filter('startFrom', function() {

return function(input, start) {

start = +start; // Átalakítás számmá

return input.slice(start);

};

});

* Ez a szűrő a lapozáshoz szükséges, mert lehetővé teszi, hogy csak egy adott oldalon lévő névjegyek jelenjenek meg.
* A startFrom szűrő az input tömböt úgy szeleteli, hogy az input.slice(start) csak a kívánt indexről kezdje a megjelenítést.
* Például ha a felhasználó a második oldalra lép, akkor a megjelenítés a (currentPage - 1) \* itemsPerPage számított értékkel kezdődik.

#### 4. ****HTML megjelenítés AngularJS kifejezésekkel****

<article ng-repeat="nevjegy in nevjegyek | startFrom: (currentPage-1) \* itemsPerPage | limitTo: itemsPerPage">

<img ng-src="kepek/{{nevjegy.foto}}" alt="{{nevjegy.nev}}">

<h2>{{nevjegy.nev}}</h2>

<h3>{{nevjegy.cegnev}}</h3>

<p>Mobil: <a ng-href="tel:{{nevjegy.mobil}}">{{nevjegy.mobil}}</a></p>

<p>E-mail: <a ng-href="mailto:{{nevjegy.email}}">{{nevjegy.email}}</a></p>

</article>

* A ng-repeat direktíva segítségével iterálunk végig a $scope.nevjegyek tömbön, és minden névjegyet egy külön article elemként jelenítünk meg.
* Az adatok dinamikus megjelenítése AngularJS kifejezésekkel történik ({{nevjegy.nev}}, {{nevjegy.cegnev}}, stb.).
* A ng-src és ng-href attribútumok biztosítják, hogy az adatok csak akkor kerülnek be a HTML-be, amikor azok már készen állnak a megjelenítésre, elkerülve ezzel a hivatkozási hibákat.

#### 5. ****Lapozási funkció AngularJS-ben****

<div class="pagination">

<button ng-disabled="currentPage == 1" ng-click="currentPage = 1">Első</button>

<button ng-disabled="currentPage == 1" ng-click="currentPage = currentPage - 1">Előző</button>

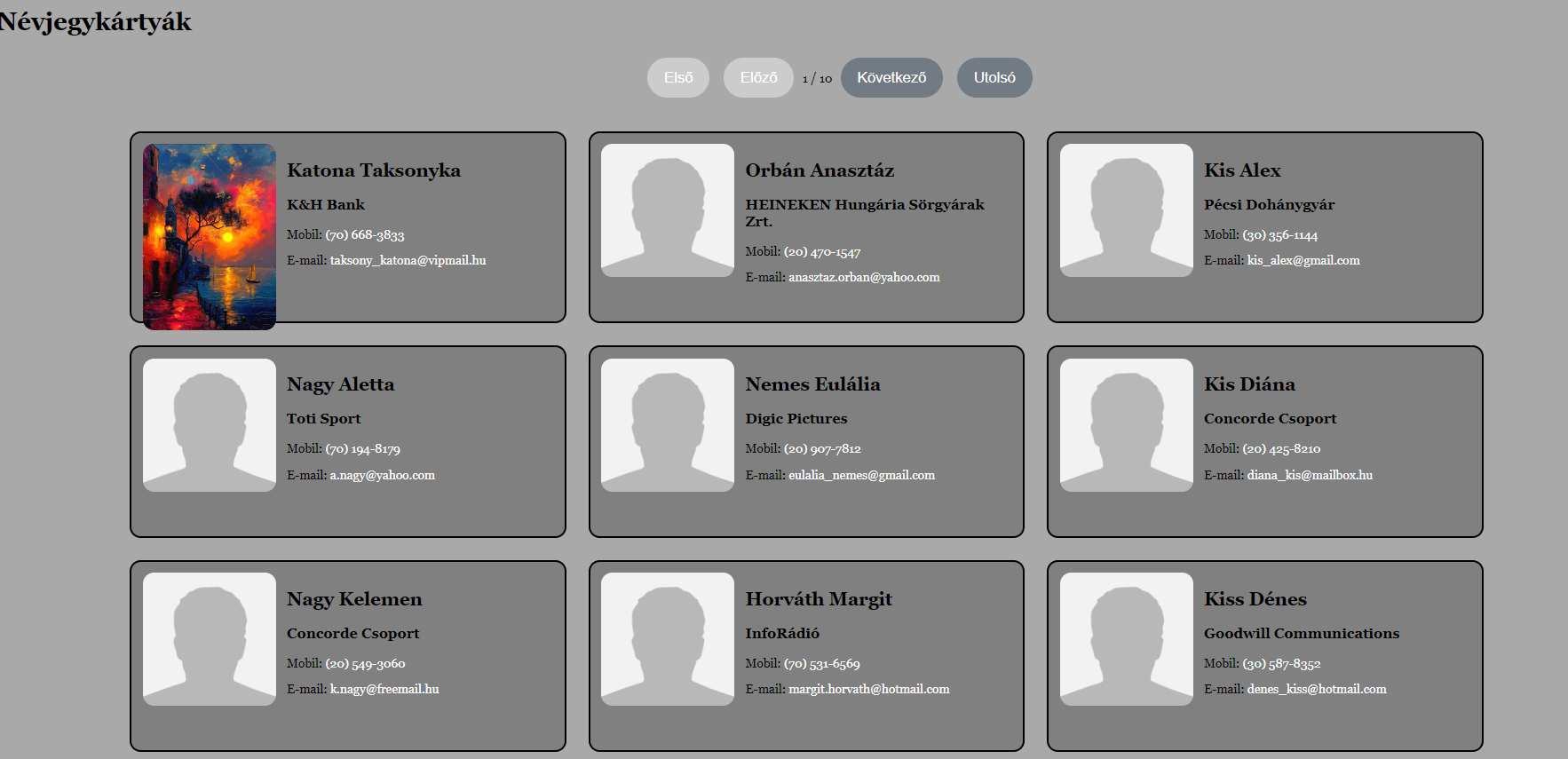
<span>{{currentPage}} / {{numberOfPages()}}</span>

<button ng-disabled="currentPage == numberOfPages()" ng-click="currentPage = currentPage + 1">Következő</button>

<button ng-disabled="currentPage == numberOfPages()" ng-click="currentPage = numberOfPages()">Utolsó</button>

</div>

* A lapozási gombok lehetővé teszik az előre- és visszalépést az oldalak között.
* Az ng-disabled attribútum használatával a gombokat deaktiváljuk, ha az aktuális oldal már az első vagy az utolsó.
* A ng-click attribútummal határozzuk meg, hogy mi történik, amikor egy adott gombra kattintanak (pl. visszalépés az első oldalra vagy továbblépés a következő oldalra).
* Az aktuális oldalszámot a currentPage változó tartja nyilván, amelyet növelhetünk vagy csökkenthetünk a gombok kattintásakor.



a kártyák felett az Angular beépített lapozót láthatod

#### 6. ****Stílus és megjelenés (CSS)****

.pagination {

margin: 20px 0;

text-align: center;

}

.pagination button {

background-color: #727a83;

color: white;

border: none;

padding: 10px 15px;

margin: 0 5px;

cursor: pointer;

border-radius: 20px;

}

.pagination button:disabled {

background-color: #cccccc;

cursor: not-allowed;

}

* A pagination osztály a lapozási gombokat formázza, középre igazítva azokat, és hozzáadva némi margót a megjelenéshez.
* A gombok alapértelmezett háttere szürke (#727a83), de ha egy gomb deaktivált (:disabled), akkor világosabbá válik, jelezve, hogy az nem kattintható.

#### Összegzés

Ez az AngularJS alkalmazás dinamikusan jeleníti meg a névjegykártyákat egy PHP API-ból JSON formátumban lekért adatok alapján. A lapozási logika lehetővé teszi, hogy a felhasználó egy oldalon egyszerre 9 névjegykártyát láthasson, és egyszerűen navigálhasson a különböző oldalak között. Az AngularJS dinamikus adatkötési funkciói biztosítják, hogy a felhasználói felület folyamatosan szinkronban legyen az adatokkal.

## Admin felület és a CRUD műveletek

A következő fájlok létrehozásához, hozz létre a projekt mappádban egy admin mappát, és a további fájlokat ebben hozd létre, hogy a lapvédelmet majd a feladatok végén ki tudd építeni.

## Listázás az admin részére

#### Névjegykártya adminisztráció, SQL lekérdezés előkészítéssel és rendezéssel

Ez a PHP alapú kód egy névjegykártya-kezelő rendszert valósít meg, amely tartalmazza a névjegykártyák keresését, rendezését és a találatok megjelenítését. A felhasználó kereshet név, cég, mobil vagy email alapján, a találatokat pedig több szempont alapján rendezheti. A kód biztonságos SQL lekérdezéseket használ az előkészített utasítások (prepared statements) segítségével, ezáltal csökkentve az SQL injekció veszélyét.

#### 1. ****Session kezelés és felhasználói jogosultságok ellenőrzése****

session\_start();

if (!isset($\_SESSION['belepett'])) {

header("Location: false.html"); // Átirányítás, ha a felhasználó nincs bejelentkezve

exit();

}

* **Session indítása**: A session\_start() segítségével kezdjük a session-t. Ez biztosítja, hogy a felhasználó bejelentkezési állapota megmaradjon.
* **Bejelentkezés ellenőrzése**: A $\_SESSION['belepett'] változót ellenőrizzük, hogy a felhasználó bejelentkezett-e. Ha nem, akkor átirányítjuk egy false.html oldalra.

#### 2. ****Adatbázis kapcsolat****

require "../kapcsolat.php";

* Betöltjük az adatbázis kapcsolatot kezelő fájlt, ami tartalmazza az adatbázis-kapcsolat beállításait (pl. adatbázis szerver, felhasználónév, jelszó).

#### 3. ****Rendezési és keresési kifejezések kezelése****

$rendez = $\_GET['rendez'] ?? "nev";

$kifejezes = $\_POST['kifejezes'] ?? "";

* **Rendezés**: A $\_GET['rendez'] paraméterből vesszük a rendezési szempontot (pl. név, cégnév, mobil, email). Ha nincs megadva, az alapértelmezett rendezési szempont a nev.
* **Keresési kifejezés**: A $\_POST['kifejezes'] tartalmazza a keresési kifejezést, amelyet a felhasználó adott meg az űrlapon.

#### 4. ****Rendezési oszlopok validálása****

$validColumns = ['nev', 'cegnev', 'mobil', 'email'];

$rendez = in\_array($rendez, $validColumns) ? $rendez : 'nev';

* Itt létrehozunk egy **fehér listát** a megengedett oszlopokról (név, cég, mobil, email), amelyek alapján rendezhetjük az adatokat. Ez a módszer megakadályozza, hogy rosszindulatú felhasználók manipulálják az SQL lekérdezést más oszlopok vagy kód injektálásával.

#### 5. ****SQL lekérdezés előkészítése (Prepared Statement)****

$sql = "SELECT \*

FROM nevjegyek

WHERE (nev LIKE ?

OR cegnev LIKE ?

OR mobil LIKE ?

OR email LIKE ?)

ORDER BY {$rendez} ASC";

$stmt = $dbconn->prepare($sql);

* **Előkészített lekérdezés**: Az SQL lekérdezést előkészítjük úgy, hogy a keresési kifejezés helyére helyettesítő karaktereket (?) helyezünk, és a rendezési szempont alapján rendezzük a találatokat.
* Az előkészítés során a prepare() függvényt használjuk, ami biztonságos SQL lekérdezést biztosít, mivel a felhasználói bemeneteket később paraméterként adjuk át a lekérdezéshez.

#### 6. ****Paraméterek bekötése és lekérdezés végrehajtása****

$searchTerm = "%{$kifejezes}%";

$stmt->bind\_param('ssss', $searchTerm, $searchTerm, $searchTerm, $searchTerm);

$stmt->execute();

$eredmeny = $stmt->get\_result();

* **Paraméterek bekötése**: A bind\_param() függvénnyel kötjük össze a helyettesítő karaktereket a felhasználó által megadott keresési kifejezéssel. Mivel négy különböző oszlopban keresünk (név, cégnév, mobil, email), négyszer kell bekötni ugyanazt a keresési kifejezést.
* **Lekérdezés végrehajtása**: A stmt->execute() függvénnyel futtatjuk az előkészített lekérdezést.
* **Eredmények lekérése**: A get\_result() függvénnyel lekérjük az eredményeket.

#### 7. ****Eredmények megjelenítése táblázatban****

$kimenet = "<table>

<tr>

<th>Fotó</th>

<th><a href=\"?rendez=nev\">Név</a></th>

<th><a href=\"?rendez=cegnev\">Cégnév</a></th>

<th><a href=\"?rendez=mobil\">Mobil</a></th>

<th><a href=\"?rendez=email\">E-mail</a></th>

<th>Művelet</th>

</tr>";

while ($sor = $eredmeny->fetch\_assoc()) {

$foto = htmlspecialchars($sor['foto'], ENT\_QUOTES, 'UTF-8');

$nev = htmlspecialchars($sor['nev'], ENT\_QUOTES, 'UTF-8');

$cegnev = htmlspecialchars($sor['cegnev'], ENT\_QUOTES, 'UTF-8');

$mobil = htmlspecialchars($sor['mobil'], ENT\_QUOTES, 'UTF-8');

$email = htmlspecialchars($sor['email'], ENT\_QUOTES, 'UTF-8');

$kimenet .= "<tr>

<td><img src=\"../kepek/{$foto}\" alt=\"{$nev}\"></td>

<td>{$nev}</td>

<td>{$cegnev}</td>

<td>{$mobil}</td>

<td>{$email}</td>

<td><a href=\"torles.php?id={$sor['id']}\">Törlés</a> | <a href=\"modositas.php?id={$sor['id']}\">Módosítás</a></td>

</tr>";

}

$kimenet .= "</table>";

* **Táblázat generálása**: Az eredményeket HTML táblázatként jelenítjük meg, ahol minden sor egy névjegykártya adatainak megfelelően töltődik fel.
* **HTML special characters kezelése**: Az htmlspecialchars() funkcióval védekezünk a XSS támadások ellen, így a megjelenített adatok biztonságosan kerülnek a weboldalra.

#### 8. ****Űrlap és rendezési funkciók****

<form method="post" action="">

<input type="search" id="kifejezes" name="kifejezes">

</form>

<p><a href="felvitel.php">Új névjegy</a> | <a href="kilepes.php">Kilépés</a></p>

<?php print $kimenet; ?>

<p><a href="felvitel.php">Új névjegy</a> | <a href="kilepes.php">Kilépés</a></p>

* **Keresési űrlap**: A felhasználó egy keresési mezőbe írhatja be a kifejezést, ami alapján a névjegykártyák szűrésre kerülnek.
* **Rendezés**: A táblázat fejlécében kattintható linkek találhatók, amelyekkel a felhasználó rendezheti a találatokat név, cég, mobil vagy email alapján.

#### Összegzés

Ez a PHP kód egy biztonságos és felhasználóbarát névjegykártya-kezelő rendszert valósít meg. A felhasználók kereshetnek és rendezhetnek a névjegykártyák között, miközben az előkészített SQL lekérdezések segítségével minimalizálják az SQL injekció támadások kockázatát. Az eredmények egy táblázatban jelennek meg, és a felhasználók módosíthatják vagy törölhetik a névjegyeket.



## Felvitel

Ez a PHP kód egy névjegykártya hozzáadási funkciót valósít meg. A felhasználó feltölthet egy képet, megadhat nevet, céget, mobilszámot és email címet, amelyeket a rendszer ment az adatbázisba. A kód tartalmaz bemeneti adatellenőrzést, fájlformátum-ellenőrzést és feltöltési korlátozásokat.

#### 1. ****Session kezelés****

session\_start();

if (!isset($\_SESSION['belepett'])) {

header("Location: false.html");

exit();

}

* **Session indítása**: A session\_start() szükséges ahhoz, hogy a PHP alkalmazás kezelje a felhasználó bejelentkezési állapotát.
* **Felhasználó ellenőrzése**: A $\_SESSION['belepett'] változó alapján ellenőrizzük, hogy a felhasználó be van-e jelentkezve. Ha nincs, akkor a false.html oldalra irányítjuk át a felhasználót.

#### 2. ****Űrlap Adatok Feldolgozása****

if (isset($\_POST['rendben'])) {

* Itt kezdődik az űrlap feldolgozása. Ha a felhasználó elküldi az űrlapot (a "Rendben" gombbal), akkor a kód végrehajtja a benne lévő műveleteket.

#### 3. ****Adatok Tisztítása****

$nev = htmlspecialchars(strip\_tags(ucwords(strtolower(trim($\_POST['nev'])))));

$cegnev = strip\_tags(trim($\_POST['cegnev']));

$mobil = strip\_tags(trim($\_POST['mobil']));

$email = strip\_tags(strtolower(trim($\_POST['email'])));

* **trim()**: Eltávolítja a bemenet elején és végén található szóközöket.
* **strtolower()**: Kisbetűssé alakítja az e-mail címeket és neveket.
* **ucwords()**: A név esetében minden szó első betűjét nagybetűssé alakítja.
* **strip\_tags()**: Megakadályozza a HTML vagy egyéb kódok beszúrását a bemenetbe.
* **htmlspecialchars()**: Védi az alkalmazást az XSS (cross-site scripting) támadások ellen.

#### 4. ****Adatellenőrzések****

if (empty($nev)) {

$hibak[] = "Nem adtál meg nevet!";

} elseif (strlen($nev) < 5) {

$hibak[] = "A névnek legalább 5 karakter hosszúnak kell lennie!";

}

if (!empty($mobil)) {

if (strlen($mobil) < 9) {

$hibak[] = "A mobil szám legalább 9 karakter hosszú kell legyen!";

} elseif (!preg\_match("/^[0-9]{9,}$/", $mobil)) {

$hibak[] = "A mobil szám csak számokat tartalmazhat!";

}

}

if (!empty($email)) {

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

$hibak[] = "Az e-mail cím formátuma érvénytelen!";

}

}

* **Név ellenőrzése**: A névnek kötelezően meg kell adva lennie, és legalább 5 karakter hosszúnak kell lennie.
* **Mobil szám ellenőrzése**: Ha van megadva mobil szám, annak legalább 9 karakter hosszúnak kell lennie, és csak számokat tartalmazhat.
* **Email ellenőrzése**: Ha van megadva e-mail, akkor az e-mail cím formátumának helyesnek kell lennie a filter\_var() függvény segítségével.

#### 5. ****Fájl feltöltési ellenőrzés****

$mime = ["image/jpeg", "image/pjpeg", "image/gif", "image/png"];

if ($\_FILES['foto']['error'] == 0 && $\_FILES['foto']['size'] > 2000000)

$hibak[] = "Túl nagy méretű képet töltöttél fel!";

if ($\_FILES['foto']['error'] == 0 && !in\_array($\_FILES['foto']['type'], $mime))

$hibak[] = "Nem megfelelő képformátum!";

* A feltöltött fájl mérete nem lehet nagyobb 2 MB-nál.
* Csak a megengedett fájltípusok tölthetők fel: JPEG, GIF és PNG.

#### 6. ****Fájlnevek generálása****

switch($\_FILES['foto']['type']) {

case "image/png": $kit = ".png"; break;

case "image/gif": $kit = ".gif"; break;

default: $kit = ".jpg";

}

$foto = date("U").$kit;

* A fájlnév egyedi az aktuális Unix időbélyeg (másodperc) alapján kerül generálásra, és a megfelelő kiterjesztést adja hozzá a fájl típusától függően.

#### 7. ****Hibakezelés****

if (isset($hibak)) {

$kimenet = "<ul>\n";

foreach($hibak as $hiba) {

$kimenet.= "<li>{$hiba}</li>\n";

}

$kimenet.= "</ul>\n";

}

* Ha vannak hibák (például helytelen mobil szám, e-mail vagy túl nagy fájl), akkor ezek listaként jelennek meg a felhasználónak.

#### 8. ****Adatok Mentése az Adatbázisba****

else {

require("../kapcsolat.php");

$sql = "INSERT INTO nevjegyek

(foto, nev, cegnev, mobil, email)

VALUES

('{$foto}', '{$nev}', '{$cegnev}', '{$mobil}', '{$email}')";

mysqli\_query($dbconn, $sql);

// Kép mozgatása a végleges helyére

move\_uploaded\_file($\_FILES['foto']['tmp\_name'], "../kepek/{$foto}");

header("Location: lista.php");

}

* **Adatbázisba írás**: Ha nincs hiba, az adatokat beszúrjuk a nevjegyek táblába.
* **Fájl mozgatása**: A move\_uploaded\_file() függvénnyel a feltöltött képet a végleges helyére, a ../kepek/ mappába helyezzük.
* A felhasználót a névjegylista oldalra (lista.php) irányítjuk, miután a névjegy sikeresen hozzá lett adva.



#### 9. ****HTML Űrlap és Fájl Feltöltés****

<form method="post" action="" enctype="multipart/form-data">

<?php if (isset($kimenet)) print $kimenet; ?>

<input type="hidden" name="MAX\_FILE\_SIZE" value="2000000" />

<p><label for="foto">Fotó:</label><br>

<input type="file" id="foto" name="foto"></p>

<p><label for="nev">Név\*:</label><br>

<input type="text" id="nev" name="nev" value="<?php if (isset($nev)) print $nev; ?>"></p>

<p><label for="cegnev">Cégnév:</label><br>

<input type="text" id="cegnev" name="cegnev" value="<?php if (isset($cegnev)) print $cegnev; ?>"></p>

<p><label for="mobil">Mobil:</label><br>

<input type="tel" id="mobil" name="mobil" value="<?php if (isset($mobil)) print $mobil; ?>"></p>

<p><label for="email">E-mail:</label><br>

<input type="email" id="email" name="email" value="<?php if (isset($email)) print $email; ?>"></p>

<p><em>A \*-gal jelölt mezők kitöltése kötelező!</em></p>

<input type="submit" id="rendben" name="rendben" value="Rendben">

<input type="reset" value="Mégse">

<p><a href="lista.php">Vissza a névjegyekhez</a></p>

</form>

* Az űrlap a következő mezőket tartalmazza: fotó feltöltése, név, cégnév, mobil, email. A név kötelező mező.
* Az enctype="multipart/form-data" szükséges a fájlfeltöltéshez.

#### Összegzés

Ez a kód lehetővé teszi névjegyek hozzáadását egy adatbázishoz, beleértve képek feltöltését is. A bemenetek ellenőrzése biztosítja, hogy a felhasználó csak megfelelő formátumú adatokat adjon meg. A fájlfeltöltés során a fájl méretét és formátumát is ellenőrizzük, és a feltöltött képeket biztonságos módon kezeljük.

## Módosítás

Ez a kód egy névjegykártya frissítéséhez készített űrlapot és a hozzá tartozó feldolgozó logikát valósítja meg. A felhasználó módosíthatja a névjegykártya adatait (név, cégnév, mobil, email), illetve feltölthet vagy frissíthet egy fotót. A kód tartalmaz bemeneti adatellenőrzést, fájlméret és fájltípus ellenőrzést, valamint adatbázis-műveleteket a névjegykártya frissítésére.

#### 1. ****Adatbázis Kapcsolat****

php

Kód másolása

require "../kapcsolat.php";

* Ez a sor betölti az adatbázis kapcsolatot kezelő fájlt, ami tartalmazza az adatbázis elérési adatait, mint a szerver neve, felhasználónév, jelszó stb.

#### 2. ****Fájl Típusok és Méret Ellenőrzése****

$mime = ["image/jpeg", "image/pjpeg", "image/gif", "image/png"];

$maxFileSize = 2000000; // 2MB maximális fájlméret

* **Engedélyezett MIME-típusok**: Csak bizonyos fájltípusokat fogad el a feltöltött fotókra: JPEG, GIF és PNG.
* **Maximális fájlméret**: 2 MB méretkorlátot állítunk be a fájlfeltöltésre.

#### 3. ****Űrlap Feldolgozása és Adatok Ellenőrzése****

if (isset($\_POST['rendben'])) {

* Az űrlap feldolgozása akkor indul el, ha a felhasználó elküldte az űrlapot a "Rendben" gomb megnyomásával.

##### ****Adatok Tisztítása és Biztonsági Ellenőrzés****

$nev = mysqli\_real\_escape\_string($dbconn, strip\_tags(ucwords(strtolower(trim($\_POST['nev'])))));

$cegnev = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['cegnev'])));

$mobil = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['mobil'])));

$email = mysqli\_real\_escape\_string($dbconn, strip\_tags(strtolower(trim($\_POST['email']))));

* **trim()**: Eltávolítja a felesleges szóközöket.
* **strtolower()**: Kisbetűssé alakítja az e-mail címeket és neveket.
* **ucwords()**: A név esetében minden szó első betűjét nagybetűssé alakítja.
* **strip\_tags()**: Megakadályozza a HTML vagy egyéb kódok beszúrását a bemenetbe.
* **mysqli\_real\_escape\_string()**: Megvédi a kódot az SQL injekcióktól.

##### ****Adatok Ellenőrzése és Hibakezelés****

if (empty($nev)) {

$hibak[] = "Nem adtál meg nevet!";

} elseif (strlen($nev) < 5) {

$hibak[] = "Rossz nevet adtál meg!";

}

if (!empty($mobil) && strlen($mobil) < 9) {

$hibak[] = "Rossz mobil számot adtál meg!";

}

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

$hibak[] = "Rossz e-mail címet adtál meg!";

}

* **Név ellenőrzése**: A név megadása kötelező, és legalább 5 karakter hosszúnak kell lennie.
* **Mobil ellenőrzése**: Ha a mobil szám meg van adva, annak legalább 9 karakter hosszúnak kell lennie.
* **E-mail ellenőrzése**: Az e-mail cím validálása a FILTER\_VALIDATE\_EMAIL segítségével történik.

#### 4. ****Fájl Feltöltés Ellenőrzése****

if ($\_FILES['foto']['error'] == 0) {

if ($\_FILES['foto']['size'] > $maxFileSize) {

$hibak[] = "Túl nagy méretű képet töltöttél fel!";

}

$finfo = finfo\_open(FILEINFO\_MIME\_TYPE);

$mimeType = finfo\_file($finfo, $\_FILES['foto']['tmp\_name']);

finfo\_close($finfo);

if (!in\_array($mimeType, $mime)) {

$hibak[] = "Nem megfelelő képformátum!";

}

}

* **Fájl méret ellenőrzése**: A feltöltött kép nem lehet nagyobb 2 MB-nál.
* **Fájl MIME-típus ellenőrzése**: A finfo\_file() segítségével ellenőrizzük, hogy a fájl típusa megegyezik-e az engedélyezett MIME-típusokkal.

##### ****Fájlnév Generálása****

if (empty($hibak)) {

switch($mimeType) {

case "image/png": $kit = ".png"; break;

case "image/gif": $kit = ".gif"; break;

default: $kit = ".jpg";

}

$foto = date("U") . $kit;

} else {

$foto = $\_POST['regi\_foto'];

}

* Ha nincs hiba a fájl ellenőrzés során, akkor a fájlhoz egyedi nevet generálunk az aktuális Unix időbélyeg alapján.
* Ha nem töltöttek fel új képet, akkor a régi fájlnevet használjuk.

#### 5. ****Adatok Mentése és Frissítése az Adatbázisba****

if (!empty($hibak)) {

// Hibaüzenetek megjelenítése

} else {

if ($\_FILES['foto']['error'] == 0 && empty($hibak)) {

move\_uploaded\_file($\_FILES['foto']['tmp\_name'], "../kepek/{$foto}");

}

$id = (int)$\_POST['id'];

$sql = "UPDATE nevjegyek

SET foto = '{$foto}', nev = '{$nev}', cegnev = '{$cegnev}', mobil = '{$mobil}', email = '{$email}'

WHERE id = {$id}";

if (mysqli\_query($dbconn, $sql)) {

$kimenet = "<p style='color: green;'>Az adatok sikeresen frissültek!</p>";

} else {

$kimenet = "<p style='color: red;'>Hiba történt az adatok frissítésekor!</p>";

}

}

* Ha nincs hiba, és feltöltötték a képet, akkor a fájl a megfelelő mappába kerül.
* Az adatbázis frissítése történik a UPDATE SQL parancs segítségével, amely módosítja a névjegy adatait.
* Ha a művelet sikeres, megjelenik egy zöld színű üzenet, ha sikertelen, akkor piros hibaüzenet.

#### 6. ****Űrlap Előzetes Kitöltése (Szerkesztéshez)****

else {

$id = (int)$\_GET['id'];

$sql = "SELECT \* FROM nevjegyek WHERE id = {$id}";

$eredmeny = mysqli\_query($dbconn, $sql);

$sor = mysqli\_fetch\_assoc($eredmeny);

$nev = htmlspecialchars($sor['nev'], ENT\_QUOTES, 'UTF-8');

$cegnev = htmlspecialchars($sor['cegnev'], ENT\_QUOTES, 'UTF-8');

$mobil = htmlspecialchars($sor['mobil'], ENT\_QUOTES, 'UTF-8');

$email = htmlspecialchars($sor['email'], ENT\_QUOTES, 'UTF-8');

$foto = ($sor['foto'] != "nincskep.png") ? htmlspecialchars($sor['foto'], ENT\_QUOTES, 'UTF-8') : "nincskep.png";

}

* Ha a felhasználó az űrlap megnyitásakor szerkeszt egy meglévő névjegykártyát, akkor a rekord adatait előzetesen kitöltjük az űrlap mezőiben.
* A meglévő adatokat SQL lekérdezéssel kérdezzük le, majd biztonságosan megjelenítjük azokat.

#### 7. ****HTML Űrlap****

<form method="post" action="" enctype="multipart/form-data">

<?php if (isset($kimenet)) echo $kimenet; ?>

<input type="hidden" id="id" name="id" value="<?php echo htmlspecialchars($id, ENT\_QUOTES, 'UTF-8'); ?>">

<input type="hidden" id="regi\_foto" name="regi\_foto" value="<?php echo htmlspecialchars($foto, ENT\_QUOTES, 'UTF-8'); ?>">

<input type="hidden" name="MAX\_FILE\_SIZE" value="2000000" />

<img src="../kepek/<?php echo htmlspecialchars($foto, ENT\_QUOTES, 'UTF-8'); ?>" alt="<?php echo htmlspecialchars($nev, ENT\_QUOTES, 'UTF-8'); ?>">

<p><label for="foto">Fotó:</label><br>

<input type="file" id="foto" name="foto"></p>

<p><label for="nev">Név\*:</label><br>

<input type="text" id="nev" name="nev" value="<?php echo $nev; ?>"></p>

<p><label for="cegnev">Cégnév:</label><br>

<input type="text" id="cegnev" name="cegnev" value="<?php echo $cegnev; ?>"></p>

<p><label for="mobil">Mobil:</label><br>

<input type="tel" id="mobil" name="mobil" value="<?php echo $mobil; ?>"></p>

<p><label for="email">E-mail:</label><br>

<input type="email" id="email" name="email" value="<?php echo $email; ?>"></p>

<p><em>A \*-gal jelölt mezők kitöltése kötelező!</em></p>

<input type="submit" id="rendben" name="rendben" value="Rendben">

<input type="reset" value="Mégse">

<p><a href="lista.php">Vissza a névjegyekhez</a></p>

</form>

* Az űrlap lehetővé teszi a név, cégnév, mobil és email megadását vagy frissítését, valamint egy fotó feltöltését.
* Az előzőleg feltöltött fotó is megjelenik, ha a felhasználó nem tölt fel újat.

#### Összegzés

Ez a kód egy biztonságos, felhasználóbarát névjegykártya frissítő rendszer, amely lehetővé teszi a felhasználó számára a névjegyek adatainak módosítását és a képek frissítését. Az ellenőrzések biztosítják, hogy csak megfelelő formátumú és méretű képeket tölthessenek fel, és az adatok megfelelően formázottak és validáltak legyenek. Az adatbázis-műveletek SQL injekció elleni védelemmel vannak ellátva.

## Törlés

Ez a kód egy névjegykártya törlését valósítja meg az adatbázisból, valamint törli a hozzá tartozó képet is, ha az nem egy alapértelmezett kép. A törlés folyamatában a rekordok és a fájlok kezelésére is figyelmet fordítunk. A folyamat a GET paraméterben kapott id alapján történik, amely az egyes névjegyek egyedi azonosítója.

#### 1. ****Törlendő Névjegy Azonosítása****

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

* A kód ellenőrzi, hogy az URL-ből érkező id paraméter létezik-e. Az id paraméter a névjegykártya egyedi azonosítója, amit az adatbázisban található rekordok azonosítására használunk.

#### 2. ****Adatbázis Kapcsolat Betöltése****

require("../kapcsolat.php");

$id = (int)$\_GET['id'];

* Az adatbázis kapcsolatot a kapcsolat.php fájl biztosítja.
* Az id értéket lekérjük a $\_GET paraméterből, és egész számmá alakítjuk, hogy elkerüljük az SQL injekció támadásokat. Ez biztonságosabbá teszi az alkalmazást.

#### 3. ****Kép Lekérdezése és Törlése****

$sql = "SELECT foto FROM nevjegyek WHERE id = {$id}";

$eredmeny = mysqli\_query($dbconn, $sql);

$sor = mysqli\_fetch\_assoc($eredmeny);

if ($sor['foto'] != "nincskep.png") {

unlink("../kepek/{$sor['foto']}");

}

* **Kép lekérdezése**: A SELECT foto SQL lekérdezéssel kiválasztjuk a törölni kívánt névjegykártyához tartozó fotó nevét az adatbázisból.
* **Kép törlése**: Ha a kép neve nem nincskep.png (ami valószínűleg egy alapértelmezett kép), akkor az unlink() függvénnyel töröljük a képet a ../kepek/ mappából.
  + Az unlink() függvény a fájlok törlésére szolgál.

#### 4. ****Rekord Törlése az Adatbázisból****

$sql = "DELETE FROM nevjegyek WHERE id = {$id}";

mysqli\_query($dbconn, $sql);

* **Rekord törlése**: A DELETE SQL parancs segítségével töröljük az adatbázisból a névjegykártya rekordot, amelynek azonosítója megegyezik az URL-ben kapott id értékkel.

#### 5. ****Átirányítás a Névjegykártyák Listájára****

header("Location: lista.php");

* Miután a kép és a rekord is törlésre került, a felhasználót átirányítjuk a lista.php oldalra, ahol az összes névjegykártyát listázza az alkalmazás.

#### Összegzés

Ez a kód biztosítja, hogy egy adott névjegykártya törlésre kerüljön mind az adatbázisból, mind a hozzá kapcsolódó fájlok közül, ha van hozzárendelt fotó. Az adatok biztonságos kezelése érdekében az id paramétert egész számmá alakítjuk, és csak akkor töröljük a képet, ha az nem egy alapértelmezett kép. A törlés után a felhasználót visszairányítjuk a névjegyek listájára.

## A CRUD műveletek megvalósítása a RESTful API követelményeinek megfelelően

A következő megoldásokhoz AngularJS előzetes ismeretére van szükség.

### Végpont

Ez a kód egy API végpontot hoz létre, amely lekérdezi a névjegykártyákat az adatbázisból, és JSON formátumban adja vissza az adatokat. A lekérdezés történhet egy konkrét névjegy alapján, amelyet az ID alapján kérdezünk le, vagy az összes névjegy lekérdezésére is van lehetőség.

#### 1. ****Adatbázis Kapcsolat Betöltése****

require "../kapcsolat.php";

* Betölti az adatbázis kapcsolatot biztosító fájlt, amely tartalmazza az adatbázis eléréséhez szükséges beállításokat.

#### 2. ****Karakterkódolás Beállítása****

header('Content-Type: application/json; charset=utf-8');

* A kód beállítja a HTTP válasz fejlécét úgy, hogy az JSON típusú legyen UTF-8 karakterkódolással. Ez biztosítja, hogy a JSON válasz megfelelően jelenítse meg az ékezetes karaktereket is.

#### 3. ****Egyedi Névjegy Lekérdezése ID Alapján****

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

$id = (int)$\_GET['id'];

$sql = "SELECT \* FROM nevjegyek WHERE id = ?";

$stmt = $dbconn->prepare($sql);

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

$stmt->execute();

$result = $stmt->get\_result();

* **ID ellenőrzése**: Ha az URL-ben van id paraméter és az numerikus, akkor folytatjuk az egyedi névjegy lekérdezésével.
* **SQL lekérdezés előkészítése**: A SELECT \* FROM nevjegyek WHERE id = ? lekérdezéssel kérdezzük le az adatokat az id alapján.
* **Előkészített utasítás (Prepared Statement)**: Az előkészített SQL utasítás megakadályozza az SQL injekció támadásokat. A lekérdezés során az id értékét biztonságosan kötjük az SQL parancshoz a bind\_param() függvénnyel.

#### 4. ****Lekérdezés Eredménye és JSON Válasz****

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

$contact = $result->fetch\_assoc();

echo json\_encode($contact, JSON\_UNESCAPED\_UNICODE | JSON\_PRETTY\_PRINT);

} else {

http\_response\_code(404);

echo json\_encode(['error' => 'Névjegy nem található!'], JSON\_UNESCAPED\_UNICODE | JSON\_PRETTY\_PRINT);

}

* **Adatok visszaadása**: Ha találunk rekordot, az eredményt JSON formátumban adjuk vissza a json\_encode() függvény segítségével. A JSON\_UNESCAPED\_UNICODE opció biztosítja, hogy az ékezetes karakterek helyesen jelenjenek meg, míg a JSON\_PRETTY\_PRINT olvashatóbbá teszi a JSON formátumot.
* **404-es Hiba**: Ha nincs találat az id alapján, akkor egy 404-es HTTP státuszkódot küldünk vissza és egy JSON formátumú hibaüzenetet (Névjegy nem található!).

#### 5. ****Az Összes Névjegy Lekérdezése****

else {

$sql = "SELECT \* FROM nevjegyek";

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

$contacts = [];

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

$contacts[] = $row;

}

echo json\_encode($contacts, JSON\_UNESCAPED\_UNICODE | JSON\_PRETTY\_PRINT);

}

* **Az összes névjegy lekérdezése**: Ha az URL-ben nincs id paraméter, akkor az összes névjegyet kérdezzük le a SELECT \* FROM nevjegyek SQL parancs segítségével.
* **Adatok összegyűjtése**: Az összes rekordot egy asszociatív tömbbe ($contacts) gyűjtjük össze a while ciklus segítségével.
* **JSON válasz**: A teljes tömböt JSON formátumban adjuk vissza.

#### Összegzés

Ez a kód kétféle lekérdezést tesz lehetővé:

1. **Egyedi névjegy lekérdezése**: Ha az URL-ben szerepel egy id paraméter, akkor az ahhoz tartozó névjegyet kérdezi le.
2. **Az összes névjegy lekérdezése**: Ha nincs megadva id, akkor az összes névjegyet visszaadja.

Mindkét esetben az adatokat JSON formátumban kapjuk meg, amelyet a kliens (pl. egy frontend alkalmazás) könnyen tud kezelni.

### Listázás – READ

Ez a kód egy AngularJS-alapú névjegykártya-kezelő alkalmazást valósít meg, amely lehetővé teszi a felhasználóknak, hogy névjegyeket listázzanak és töröljenek. A névjegyeket egy API-ról (PHP backend segítségével) tölti be, és a törlés is egy API hívás segítségével történik.

#### 1. ****HTML Fejléc és AngularJS Betöltése****

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<link rel="stylesheet" href="../stilus.css">

* Az AngularJS könyvtár betöltése a Google CDN-ből.
* A külső stíluslap (stilus.css) betöltése biztosítja az oldal megjelenésének formázását.

#### 2. ****AngularJS Alkalmazás Létrehozása****

const app = angular.module('contactsApp', []);

* Létrehozzuk az AngularJS alkalmazást contactsApp néven.

#### 3. ****Controller Létrehozása****

app.controller('ContactsController', ['$scope', '$http', function($scope, $http) {

* A ContactsController AngularJS kontroller az alkalmazás logikáját tartalmazza.
* A $scope az AngularJS-ben használt objektum, amely a nézet és a kontroller között közvetít.
* Az $http szolgáltatás felelős az API hívásokért (GET és POST kérések kezeléséért).

#### 4. ****Névjegyek Betöltése****

$scope.loadContacts = function() {

$http.get('RA-get-contact.php').then(function(response) {

$scope.contacts = response.data;

}, function(error) {

console.error('Adatok betöltési hiba:', error);

});

};

* A loadContacts funkció az oldal betöltésekor automatikusan lefut, és a RA-get-contact.php API hívás segítségével lekérdezi az adatbázisból az összes névjegyet.
* A sikeres válasz (response.data) eredményeként a névjegyeket hozzárendeljük a $scope.contacts-hoz, amelyet az AngularJS a nézetben megjelenít.

#### 5. ****Törlés Funkció****

$scope.deleteContact = function(id) {

if (confirm("Biztosan törölni szeretnéd?")) {

$http({

method: 'GET',

url: 'RA-delete-contact.php?id=' + id

}).then(function(response) {

if (response.status === 200) {

$scope.loadContacts(); // Újra betöltjük a névjegyeket

$scope.successMessage = 'Névjegy sikeresen törölve!';

$scope.errorMessage = null;

} else {

$scope.errorMessage = 'Valami hiba történt a törlés során.';

$scope.successMessage = null;

}

}, function(error) {

console.error('Törlési hiba:', error);

$scope.errorMessage = 'Törlési hiba történt!';

$scope.successMessage = null;

});

}

};

* A deleteContact funkció felelős egy adott névjegy törléséért.
* A funkció először megjelenít egy megerősítő párbeszédablakot (confirm), hogy a felhasználó biztosan törölni akarja-e a névjegyet.
* Ha a felhasználó megerősíti a törlést, egy GET kérést küldünk az API-nak (RA-delete-contact.php?id=), amely a névjegy törlését végzi.
* Ha a válasz sikeres (HTTP 200), újra betöltjük a névjegyeket és megjelenítünk egy sikerüzenetet. Hibás válasz esetén hibaüzenetet jelenítünk meg.

#### 6. ****HTML Struktúra: Névjegyek Megjelenítése****

<tr ng-repeat="contact in contacts">

<td><img ng-src="../kepek/{{contact.foto}}" alt="{{contact.nev}}" width="50"></td>

<td>{{contact.nev}}</td>

<td>{{contact.cegnev}}</td>

<td>{{contact.mobil}}</td>

<td>{{contact.email}}</td>

<td>

<a href="RA-update-angular.html?id={{contact.id}}">Módosítás</a> |

<a href="" ng-click="deleteContact(contact.id)">Törlés</a>

</td>

</tr>

* A HTML táblázatban az AngularJS ng-repeat direktívája segítségével jelenítjük meg az összes névjegyet.
* **Fotó megjelenítése**: A névjegyekhez tartozó képet a ng-src direktívával jelenítjük meg.
* **Műveletek**: A módosítás és törlés linkek különböző funkciókat látnak el. A módosítás egy másik oldalra irányít, ahol a felhasználó szerkesztheti a névjegyet, a törlés pedig az AngularJS ng-click direktívájával indítja a törlés funkciót.

#### 7. ****Siker és Hibaüzenetek Kezelése****

<p ng-if="successMessage" class="success">{{ successMessage }}</p>

<p ng-if="errorMessage" class="error">{{ errorMessage }}</p>

* **Sikerüzenet**: Ha a törlés sikeres, egy zöld színű üzenet jelenik meg, amely jelzi, hogy a névjegy sikeresen törölve lett.
* **Hibaüzenet**: Ha hiba történik (pl. a törlés nem sikerült), egy piros színű hibaüzenet jelenik meg.

#### Összegzés

Ez az AngularJS alapú névjegykártya-kezelő alkalmazás egyszerű módot biztosít a névjegyek megjelenítésére és törlésére. A felhasználó könnyen törölhet névjegyeket, és az alkalmazás azonnal frissíti a névjegylistát. Az API hívások (GET kérések) a háttérben zajlanak, és a válaszok alapján a felhasználó értesítést kap a műveletek sikerességéről vagy hibájáról.

### Felvitel – CREATE

Ez az oldal egy névjegykártya létrehozásához készült űrlapot tartalmaz, amely lehetővé teszi a felhasználók számára, hogy megadják egy névjegykártya adatait (név, cégnév, mobil, e-mail), illetve feltöltsenek egy képet a névjegyhez. Az űrlap JavaScript segítségével helyben validálja az adatokat és elküldi azokat a szerverre a **Fetch API** segítségével.

#### Fő funkciók:

* Név, cégnév, mobil, e-mail és fotó megadása.
* Helyi validáció a JavaScript segítségével.
* Hibaüzenetek megjelenítése helytelen bemenet esetén.
* Adatok elküldése a szerverre a **Fetch API** segítségével.

## Felépítés

#### HTML felépítés

1. **Form** (űrlap):
   * Az űrlap tartalmazza a következő bemeneti mezőket:
     + **Fotó** (fájlfeltöltés): lehetőséget ad a felhasználónak, hogy egy képet töltsön fel.
     + **Név** (szöveges mező): kötelező mező a név megadására.
     + **Cégnév** (szöveges mező): opcionális mező a cégnév megadására.
     + **Mobil** (telefonszám mező): opcionális mező a telefonszám megadására.
     + **E-mail** (e-mail mező): opcionális mező az e-mail cím megadására.
   * Két gomb található az űrlapon:
     + **Küldés**: beküldi az űrlapot.
     + **Mégse**: visszaállítja az űrlapot.
2. **Hibaüzenetek**:
   * Az űrlap felett egy div elem (id="hiba-uzenet") található, ahol az esetleges hibák megjelennek.

#### CSS stílusok

Az oldalhoz kapcsolódik egy külső CSS fájl, amely az oldal megjelenését szabályozza.

#### JavaScript

#### Form validáció:

* A beküldés (submit) eseményt megelőzően validáció történik:
  + **Név**: nem lehet üres, és minimum 5 karakter hosszúnak kell lennie.
  + **Mobil**: ha meg van adva, akkor minimum 9 karakter hosszúnak kell lennie.
  + **E-mail**: ha meg van adva, ellenőrizzük az e-mail formátumát.
  + **Fotó**: ha fotót töltenek fel, annak mérete nem haladhatja meg a 2 MB-ot.
* Ha a validáció hibákat talál, akkor ezeket egy listában megjelenítjük a felhasználó számára a hiba-uzenet div-ben.

#### Fetch API

* A validáció sikeres teljesítése után a **Fetch API** használatával a FormData objektumban elküldjük az adatokat a RA-create-contact.php fájlra, amely feldolgozza azokat a szerveren.
* A szerver visszajelzése alapján az alábbiak történhetnek:
  + **Siker**: ha a beküldés sikeres, a felhasználót átirányítjuk a lista.php oldalra, ahol az összes névjegy listája található.
  + **Hiba**: ha hiba történik, a hibaüzenetek megjelennek az oldalon.

#### Form működésének részletei

* Az űrlapot a JavaScript validálja, mielőtt beküldésre kerülne.
* Ha a form kitöltése során hibák adódnak (pl. túl rövid név, helytelen e-mail cím), a hibákat a felhasználó azonnal látja az oldal tetején, pirossal kiemelve.
* A Fetch API segítségével az adatokat JavaScript küldi el POST metódussal a szerverre, ahol a névjegy létrehozása megtörténik.
* Ha minden rendben van, a névjegy bekerül a rendszerbe, és a felhasználót átirányítják a névjegylistára.

#### Kód részletezés

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Névjegykártyák - Űrlap</title>

<link rel="stylesheet" href="../stilus.css">

</head>

<body>

<h1>Névjegykártyák</h1>

<form id="nevjegy-form" enctype="multipart/form-data">

<div id="hiba-uzenet"></div>

<input type="hidden" name="MAX\_FILE\_SIZE" value="2000000">

<!-- Fotó feltöltése -->

<p><label for="foto">Fotó:</label><br>

<input type="file" id="foto" name="foto"></p>

<!-- Név -->

<p><label for="nev">Név\*:</label><br>

<input type="text" id="nev" name="nev"></p>

<!-- Cégnév -->

<p><label for="cegnev">Cégnév:</label><br>

<input type="text" id="cegnev" name="cegnev"></p>

<!-- Mobil -->

<p><label for="mobil">Mobil:</label><br>

<input type="tel" id="mobil" name="mobil"></p>

<!-- E-mail -->

<p><label for="email">E-mail:</label><br>

<input type="email" id="email" name="email"></p>

<p><em>A \*-gal jelölt mezők kitöltése kötelező!</em></p>

<!-- Küldés és Mégse gombok -->

<button type="submit">Küldés</button>

<button type="reset">Mégse</button>

<!-- Link a névjegylistára -->

<p><a href="lista.php">Vissza a névjegyekhez</a></p>

</form>

<!-- JavaScript -->

<script>

document.getElementById('nevjegy-form').addEventListener('submit', function(e) {

e.preventDefault(); // Alapértelmezett form beküldés megakadályozása

// Hibák összegyűjtése

let hibak = [];

let nev = document.getElementById('nev').value.trim();

let cegnev = document.getElementById('cegnev').value.trim();

let mobil = document.getElementById('mobil').value.trim();

let email = document.getElementById('email').value.trim();

let foto = document.getElementById('foto').files[0];

// Validáció

if (nev === "" || nev.length < 5) {

hibak.push("Nem adtál meg érvényes nevet!");

}

if (mobil !== "" && mobil.length < 9) {

hibak.push("Rossz mobil számot adtál meg!");

}

if (email !== "" && !/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {

hibak.push("Rossz e-mail címet adtál meg!");

}

if (foto && foto.size > 2000000) {

hibak.push("Túl nagy méretű képet töltöttél fel!");

}

// Ha vannak hibák, jelenítsük meg

if (hibak.length > 0) {

let hibaUzenet = "<ul>";

hibak.forEach(function(hiba) {

hibaUzenet += "<li>" + hiba + "</li>";

});

hibaUzenet += "</ul>";

document.getElementById('hiba-uzenet').innerHTML = hibaUzenet;

} else {

// Adatok elküldése a backendnek

let formData = new FormData();

formData.append('nev', nev);

formData.append('cegnev', cegnev);

formData.append('mobil', mobil);

formData.append('email', email);

formData.append('foto', foto);

fetch('RA-create-contact.php', {

method: 'POST',

body: formData

})

.then(response => response.json())

.then(data => {

if (data.message) {

window.location.href = 'lista.php'; // Siker esetén átirányít

} else if (data.errors) {

let hibaUzenet = "<ul>";

data.errors.forEach(function(hiba) {

hibaUzenet += "<li>" + hiba + "</li>";

});

hibaUzenet += "</ul>";

document.getElementById('hiba-uzenet').innerHTML = hibaUzenet;

}

})

.catch(error => console.error('Hiba:', error));

}

});

</script>

</body>

</html>

Ez az oldal lehetőséget ad új névjegyek létrehozására és az adatokat biztonságosan elküldi a szerverre. Az űrlap JavaScript validálást tartalmaz, és a modern Fetch API segítségével kommunikál a backenddel.

**Backend feldolgozó fáj: RA-create-contact.php**

Ez a PHP script egy névjegy hozzáadását kezeli egy MySQL adatbázisban. Az űrlapról érkező adatok (név, cégnév, mobil, e-mail és fotó) ellenőrzésre kerülnek, majd egy MySQL adatbázisba kerülnek mentésre. A kód a következő műveleteket hajtja végre:

* Adatok tisztítása és validálása.
* Hibakezelés a bemeneti adatokkal kapcsolatban.
* Fájl feltöltése és kezelése.
* Adatok beszúrása az adatbázisba.
* Válasz küldése JSON formátumban az AJAX kérésre.

#### Fő funkciók

1. **POST Kérés Kezelése**: Az adatok csak akkor kerülnek feldolgozásra, ha POST kérés érkezik a szerverhez.
2. **Adatok Tisztítása**: A bemeneti adatok (név, cégnév, mobil, e-mail) tisztításra és ellenőrzésre kerülnek SQL injekció ellen és egyéb biztonsági kockázatok elkerülése érdekében.
3. **Hibakezelés**: Az űrlapadatok validációja során fellépő hibákat összegyűjtjük és JSON formátumban visszaküldjük a kliensnek.
4. **Fájl Feltöltés**: Ha egy kép is feltöltésre kerül, annak típusa és mérete ellenőrzésre kerül, és biztonságosan a szerver megfelelő könyvtárába kerül feltöltésre.
5. **Adatbázis Beszúrás**: Ha az adatok helyesen kerültek megadásra, akkor azok az adatbázisba kerülnek beszúrásra, és a kliens egy sikerüzenetet kap.
6. **Hibaválasz**: Ha bármilyen hiba történik a folyamat során, a rendszer megfelelő hibakódokat és üzeneteket küld vissza a kliensnek.

#### Kód Felépítése

#### 1. ****Adatbázis Kapcsolat Betöltése****

require "../kapcsolat.php"; // Kapcsolat az adatbázissal

Ez a sor betölti a kapcsolatot az adatbázishoz, amely a kapcsolat.php fájlban van definiálva. Ez biztosítja, hogy a továbbiakban a $dbconn objektumon keresztül hozzáférhetünk az adatbázishoz.

#### 2. ****POST Kérés Ellenőrzése****

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

Ez a feltétel ellenőrzi, hogy a kérés egy **POST** metódussal érkezett, és csak abban az esetben dolgozza fel az adatokat. Ha a kérés más módszerrel érkezik (pl. GET), akkor nem történik semmi.

#### 3. ****Adatok Tisztítása****

Az űrlapból érkező adatok ($\_POST tömbből) tisztításra kerülnek. Az alábbi függvényeket használjuk:

* trim(): eltávolítja a szóközöket a bemenet elejéről és végéről.
* strip\_tags(): eltávolítja az összes HTML és PHP címkét a bemenetből.
* mysqli\_real\_escape\_string(): megvédi az adatokat az SQL injekciótól.

$nev = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['nev'])));

$cegnev = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['cegnev'])));

$mobil = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['mobil'])));

$email = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['email'])));

#### 4. ****Hibakezelés****

A bemenetek érvényességét ellenőrizzük, és a hibákat egy tömbben tároljuk, ha érvénytelen adatokat találunk:

* **Név**: kötelező mező.
* **E-mail**: ha meg van adva, akkor ellenőrizzük a formátumot.
* **Mobil**: ha meg van adva, akkor ellenőrizzük, hogy legalább 9 számjegyből álljon.

if (empty($nev)) {

$hibak[] = "A név mező nem lehet üres!";

}

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

$hibak[] = "Az e-mail cím formátuma érvénytelen!";

}

if (!empty($mobil) && !preg\_match('/^[0-9]{9,}$/', $mobil)) {

$hibak[] = "A mobil szám formátuma nem megfelelő!";

}

#### 5. ****Fájl Feltöltés****

Ha a felhasználó egy fotót is feltölt, ellenőrizzük a fájl típusát (JPEG, PNG, GIF), és ha megfelelő, akkor azt feltöltjük egy kepek könyvtárba.

if (isset($\_FILES['foto']) && $\_FILES['foto']['error'] === UPLOAD\_ERR\_OK) {

// MIME típusok ellenőrzése

$allowedMimeTypes = ['image/jpeg', 'image/png', 'image/gif'];

$mimeType = mime\_content\_type($\_FILES['foto']['tmp\_name']);

if (!in\_array($mimeType, $allowedMimeTypes)) {

$hibak[] = "Nem megfelelő képformátum!";

}

// Fájl feltöltés és név generálás

if (empty($hibak)) {

$extension = pathinfo($\_FILES['foto']['name'], PATHINFO\_EXTENSION);

$fotoFileName = time() . '.' . $extension; // Egyedi fájlnév időbélyeggel

move\_uploaded\_file($\_FILES['foto']['tmp\_name'], "../kepek/{$fotoFileName}");

}

} else {

// Ha nincs feltöltött kép, használjunk alapértelmezett képet

$fotoFileName = 'nincskep.png';

}

#### 6. ****Adatok Adatbázisba Mentése****

Ha minden adat érvényes és nincs hiba, akkor az adatokat beszúrjuk az adatbázisba egy SQL lekérdezéssel. Az adatokat előkészítjük és paraméterkötéssel adjuk hozzá a lekérdezéshez, hogy elkerüljük az SQL injekciós támadásokat.

if (empty($hibak)) {

$sql = "INSERT INTO nevjegyek (nev, cegnev, mobil, email, foto) VALUES (?, ?, ?, ?, ?)";

$stmt = $dbconn->prepare($sql);

$stmt->bind\_param('sssss', $nev, $cegnev, $mobil, $email, $fotoFileName);

if ($stmt->execute()) {

http\_response\_code(201); // 201 Created státusz

echo json\_encode(['message' => 'Névjegy sikeresen létrehozva!'], JSON\_UNESCAPED\_UNICODE);

} else {

http\_response\_code(500); // 500 Internal Server Error

echo json\_encode(['error' => 'Hiba történt az adatbázis művelet során!'], JSON\_UNESCAPED\_UNICODE);

}

}

#### 7. ****Hibakezelés és Válasz Küldése****

Ha hibák vannak a feldolgozás során (például érvénytelen adat vagy hibás fájl), akkor a hibákat JSON formátumban küldjük vissza a kliensnek, 400-as (rossz kérés) státuszkóddal.

if (!empty($hibak)) {

http\_response\_code(400); // 400 Bad Request

echo json\_encode(['errors' => $hibak], JSON\_UNESCAPED\_UNICODE);

}

Ez a PHP script biztonságos módszereket alkalmaz az adatok beszúrására és fájlok feltöltésére, miközben kezel minden lehetséges hibát, amely az adatbevitel során felmerülhet. A válaszok JSON formátumban történnek, hogy jól integrálható legyen egy modern JavaScript front-endhez.

### Módosítás – UPDATE

Ez az AngularJS alkalmazás lehetőséget biztosít egy névjegy adatainak frissítésére, beleértve a név, cégnév, mobil, e-mail és fotó módosítását. A módosítási művelet egy API-n keresztül történik, amely POST kéréssel fogadja a frissített adatokat. A képfeltöltést is támogatja, és ha a felhasználó nem tölt fel új képet, akkor az aktuális képet használja tovább.

#### 1. ****HTML Fejléc és AngularJS Betöltése****

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

<link rel="stylesheet" href="../stilus.css">

* Az AngularJS könyvtár betöltése a Google CDN-ből, amely a kliensoldali JavaScript logikát biztosítja.
* A külső stíluslap (stilus.css) formázza az oldal megjelenését.

#### 2. ****Form és Input mezők****

<form ng-submit="updateContact()" enctype="multipart/form-data">

<label>Név:</label>

<input type="text" ng-model="contact.nev" required><br>

<label>Cégnév:</label>

<input type="text" ng-model="contact.cegnev"><br>

<label>Mobil:</label>

<input type="tel" ng-model="contact.mobil"><br>

<label>E-mail:</label>

<input type="email" ng-model="contact.email"><br>

<!-- Jelenlegi fotó megjelenítése -->

<label>Jelenlegi fotó:</label><br>

<img ng-src="../kepek/{{contact.foto}}" alt="{{contact.nev}}" width="100"><br>

<!-- Új fotó feltöltése -->

<label>Új fotó:</label>

<input type="file" file-model="contact.newFoto"><br>

<button type="submit">Frissítés</button>

</form>

* A felhasználó módosíthatja a névjegy különböző adatait (név, cégnév, mobil, email) a megfelelő input mezőkben.
* A felhasználó a jelenlegi fotót is megtekintheti, és lehetősége van új fotót feltölteni.
* Az ng-model attribútum segítségével az input mezők szinkronizálva vannak az AngularJS modell adataival, így azok közvetlenül frissülnek.

#### 3. ****Custom AngularJS Direktíva: File Upload****

app.directive('fileModel', ['$parse', function($parse) {

return {

restrict: 'A',

link: function(scope, element, attrs) {

const model = $parse(attrs.fileModel);

const modelSetter = model.assign;

element.bind('change', function() {

scope.$apply(function() {

modelSetter(scope, element[0].files[0]);

});

});

}

};

}]);

* Ez a custom direktíva (fileModel) a fájlok kiválasztásához és feltöltéséhez szükséges, mivel az alapértelmezett AngularJS nem kezeli a fájlokat automatikusan.
* A fileModel direktíva lehetővé teszi, hogy a fájlt (pl. új fotó) a $scope változók között kezeljük.

#### 4. ****Adatok Betöltése a Szerverről****

const urlParams = new URLSearchParams(window.location.search);

const contactId = urlParams.get('id');

$http.get('RA-get-contact.php?id=' + contactId).then(function(response) {

$scope.contact = response.data;

}, function(error) {

console.error('Hiba a névjegy adatainak betöltésekor:', error);

});

* Az URL-ből lekérdezzük az id paramétert (amely meghatározza a szerkesztendő névjegyet).
* Az AngularJS $http.get() metódusával kérjük le a névjegy adatait egy PHP API-ról (RA-get-contact.php), majd az adatokat a $scope.contact változóba mentjük.

#### 5. ****Névjegy Frissítése****

$scope.updateContact = function() {

const formData = new FormData();

formData.append('id', contactId);

formData.append('nev', $scope.contact.nev);

formData.append('cegnev', $scope.contact.cegnev);

formData.append('mobil', $scope.contact.mobil);

formData.append('email', $scope.contact.email);

// Ha van új fotó, akkor hozzáadjuk

if ($scope.contact.newFoto) {

formData.append('foto', $scope.contact.newFoto);

}

$http({

method: 'POST',

url: 'RA-put-contact.php',

data: formData,

headers: { 'Content-Type': undefined },

}).then(function(response) {

$scope.successMessage = response.data.message;

setTimeout(function() {

$window.location.href = 'RA-angular-list.html';

}, 2000);

}, function(error) {

console.error('Hiba történt a frissítés során:', error);

});

};

* A frissítés egy űrlap beküldésével történik (ng-submit="updateContact()"), amely a FormData objektumba csomagolja az adatokat, beleértve a fájlokat is, ha a felhasználó új fotót töltött fel.
* A $http.post() metódus a RA-put-contact.php végponthoz küldi az adatokat, ahol a névjegyet frissítjük az adatbázisban.
* Ha a frissítés sikeres, a felhasználót 2 másodperc múlva visszairányítja a névjegyek listájára (RA-angular-list.html).

#### 6. ****Üzenet Kezelés****

<p ng-if="successMessage">{{ successMessage }}</p>

* Ha a névjegy frissítése sikeres, megjelenítünk egy üzenetet a felhasználónak, amely jelzi a sikeres műveletet.

#### Összegzés

Ez az AngularJS alapú névjegy módosító felület lehetővé teszi a felhasználó számára, hogy frissítse egy névjegykártya adatait, beleértve az új fotó feltöltésének lehetőségét is. Az API-val való kommunikáció AJAX hívások segítségével történik, a fájlok kezelése pedig a FormData objektumon keresztül történik, amely lehetővé teszi, hogy fájlokat is küldjünk a szerverre POST kérésekben.

### Módosítás – PUT

Ez a PHP kód egy névjegykártya frissítési műveletet valósít meg. A kód támogatja a névjegy különböző adatainak frissítését (név, cégnév, mobil, e-mail) és kezeli a fotók feltöltését is. Ha új fotót töltenek fel, akkor a régi fotó törlésre kerül, és az új fájl kerül a helyére. Az adatbázis műveletek biztonságosak az előkészített SQL utasításokkal.

#### 1. ****Adatbázis Kapcsolat****

require "../kapcsolat.php";

* Az adatbázis kapcsolatot kezelő fájl betöltése, amely a mysqli kapcsolatot biztosítja a további műveletekhez.

#### 2. ****POST Kérés Ellenőrzése****

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

* A kód csak akkor fut le, ha a kérés POST módszerrel érkezik. Ez biztosítja, hogy az adatok frissítése ne GET módszerrel történjen, mivel a POST az adatok módosítására szolgál.

#### 3. ****Bemeneti Adatok Biztonságos Kezelése****

$id = (int) $\_POST['id'];

$nev = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['nev'])));

$cegnev = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['cegnev'])));

$mobil = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['mobil'])));

$email = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['email'])));

* **Bemenetek ellenőrzése és tisztítása**: A strip\_tags() eltávolítja az esetleges HTML tag-eket a bemenetekből, a trim() pedig eltávolítja a felesleges szóközöket a mezők elejéről és végéről.
* **SQL injekció elleni védelem**: A mysqli\_real\_escape\_string() biztosítja, hogy az adatok biztonságosan kerüljenek az SQL lekérdezésbe.

#### 4. ****Kép Feltöltés és Ellenőrzés****

if (isset($\_FILES['foto']) && $\_FILES['foto']['error'] === UPLOAD\_ERR\_OK) {

// MIME típus ellenőrzése

$allowedMimeTypes = ['image/jpeg', 'image/png', 'image/gif'];

$mimeType = mime\_content\_type($\_FILES['foto']['tmp\_name']);

if (in\_array($mimeType, $allowedMimeTypes)) {

$extension = pathinfo($\_FILES['foto']['name'], PATHINFO\_EXTENSION);

$fotoFileName = time() . '.' . $extension; // Egyedi fájlnév generálása

move\_uploaded\_file($\_FILES['foto']['tmp\_name'], "../kepek/{$fotoFileName}");

// Régi fotó törlése

$oldFotoQuery = "SELECT foto FROM nevjegyek WHERE id = ?";

$stmt = $dbconn->prepare($oldFotoQuery);

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

$stmt->execute();

$result = $stmt->get\_result();

$oldFoto = $result->fetch\_assoc()['foto'];

if ($oldFoto && file\_exists("../kepek/{$oldFoto}") && $oldFoto !== 'nincskep.png') {

unlink("../kepek/{$oldFoto}");

}

} else {

http\_response\_code(400);

echo json\_encode(['error' => 'Nem megfelelő képformátum!']);

exit();

}

} else {

// Ha nincs új fotó, megtartjuk a régit

$fotoFileName = $\_POST['regi\_foto'] ?? 'nincskep.png';

}

* **Fájl ellenőrzése**: A kód ellenőrzi, hogy a feltöltött fájl létezik és nincs-e hiba a feltöltés során.
* **Fájl MIME típus ellenőrzése**: Csak a JPEG, PNG és GIF típusú fájlok megengedettek.
* **Fájl feltöltése**: Ha a fájl típusa megfelelő, akkor egy egyedi nevet generálunk (az aktuális időbélyeg segítségével), és a fájlt áthelyezzük a ../kepek/ mappába.
* **Régi fotó törlése**: Ha van már egy meglévő fotó és az nem az alapértelmezett "nincskep.png", akkor azt töröljük az új fájl feltöltése után.

#### 5. ****Adatbázis Frissítése****

$sql = "UPDATE nevjegyek SET nev = ?, cegnev = ?, mobil = ?, email = ?, foto = ? WHERE id = ?";

$stmt = $dbconn->prepare($sql);

$stmt->bind\_param('sssssi', $nev, $cegnev, $mobil, $email, $fotoFileName, $id);

if ($stmt->execute()) {

http\_response\_code(200);

echo json\_encode(['message' => 'Névjegy sikeresen frissítve!']);

} else {

http\_response\_code(500);

echo json\_encode(['error' => 'Hiba történt a frissítés során!']);

}

* **SQL lekérdezés előkészítése**: Az UPDATE SQL utasítással frissítjük az adatokat a névjegykártyában, beleértve a fotó nevét is, ha új fotót töltöttek fel.
* **Biztonságos SQL végrehajtás**: A bind\_param() segítségével biztonságosan kötjük az értékeket az SQL parancshoz, így elkerülve az SQL injekció támadásokat.
* **Sikeres frissítés**: Ha az SQL utasítás sikeresen lefut, egy 200-as HTTP státuszkódot küldünk vissza, amely jelzi a sikeres műveletet, és egy JSON formátumú üzenetet adunk vissza.
* **Hibakezelés**: Ha a frissítés során hiba történik, egy 500-as HTTP hibakódot küldünk vissza és egy hibaüzenetet.

#### Összegzés

Ez a PHP kód biztonságosan kezeli a névjegykártya frissítését az adatbázisban, beleértve az új fotó feltöltését és a régi fotó törlését. A bemeneti adatok tisztítása és az SQL injekció elleni védelem biztosítja, hogy a kód biztonságos legyen. Az új fotó fájl kezelése megfelelő MIME típus-ellenőrzéssel és egyedi fájlnév-generálással történik.

### Törlés – DELETE

Ez a PHP kód egy névjegykártya törlési műveletet valósít meg. A GET kérés segítségével egy adott id alapján törli a névjegyet az adatbázisból, majd visszairányítja a felhasználót egy HTML oldalra. A kód UTF-8 karakterkódolást használ a JSON válaszoknál, és biztonságos SQL lekérdezéseket hajt végre előkészített utasításokkal.

#### 1. ****Adatbázis Kapcsolat****

require "../kapcsolat.php";

* Az adatbázis kapcsolatot biztosító fájl betöltése, amely a mysqli kapcsolatot biztosítja.

#### 2. ****UTF-8 Karakterkódolás Biztosítása****

header('Content-Type: application/json; charset=UTF-8');

* Ez a sor beállítja a válasz karakterkódolását UTF-8 formátumra, biztosítva, hogy a JSON válaszok helyesen jelenjenek meg ékezetes karakterekkel is.

#### 3. ****GET Kérés és Azonosító Ellenőrzése****

if ($\_SERVER['REQUEST\_METHOD'] === 'GET' && isset($\_GET['id'])) {

* **GET kérés ellenőrzése**: A törlési művelet csak akkor hajtódik végre, ha a kérés GET módszerrel érkezik, és van id paraméter az URL-ben.
* **Azonosító ellenőrzése**: Az id paramétert először egész számmá alakítjuk, hogy elkerüljük a nem kívánt bemeneteket és SQL injekció támadásokat.

#### 4. ****Névjegy Törlése az Adatbázisból****

$sql = "DELETE FROM nevjegyek WHERE id = ?";

$stmt = $dbconn->prepare($sql);

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

* **SQL törlési parancs előkészítése**: A DELETE SQL parancs segítségével töröljük a névjegyet az adatbázisból, ahol az id megegyezik a kapott értékkel.
* **Előkészített SQL utasítás**: A prepare() és bind\_param() függvények biztosítják, hogy az SQL lekérdezés biztonságos legyen az SQL injekciókkal szemben. A bind\_param('i', $id) azt jelenti, hogy az id egy egész szám (i), amit az SQL parancsban használunk.

#### 5. ****Törlési Művelet Eredménye****

if ($stmt->execute()) {

// Sikeres törlés esetén átirányítás az RA-angular-list.html oldalra

header('Location: RA-angular-list.html');

exit();

} else {

// Hiba esetén visszaküldjük a hibaüzenetet UTF-8 kódolással

http\_response\_code(500);

echo json\_encode(['error' => 'Hiba történt a törlés során!'], JSON\_UNESCAPED\_UNICODE);

}

* **Sikeres törlés**: Ha a DELETE parancs sikeresen végrehajtódik, akkor a felhasználót átirányítjuk a RA-angular-list.html oldalra a header('Location: RA-angular-list.html') segítségével.
* **Hiba esetén**: Ha a törlés nem sikerül, egy 500-as HTTP státuszkódot küldünk vissza, és egy hibaüzenetet jelenítünk meg JSON formátumban.

#### 6. ****Hibakezelés: Hiányzó**** id ****vagy Hibás Kérés****

} else {

// Hibás kérés vagy hiányzó azonosító esetén hibaüzenet visszaküldése

http\_response\_code(400);

echo json\_encode(['error' => 'Hibás kérés vagy hiányzó azonosító!'], JSON\_UNESCAPED\_UNICODE);

}

* Ha a kérés nem GET módszerrel érkezik, vagy ha hiányzik az id paraméter, egy 400-as hibakódot küldünk vissza, amely azt jelzi, hogy a kérés érvénytelen. Egy JSON formátumú hibaüzenetet is visszaküldünk.

#### Összegzés

Ez a PHP kód egy névjegykártya törlésére szolgál, amely az id alapján törli a névjegyet az adatbázisból. A GET kérés során kapott azonosítót először biztonságosan kezeljük, és ha a törlési művelet sikeres, a felhasználót visszairányítjuk egy megadott HTML oldalra. Hibás kérés vagy törlési hiba esetén megfelelő hibaüzenetet és HTTP státuszkódot küldünk vissza.

## Titkosítás – Lapvédelem

Ha beírjuk a böngésző címsorába a http://localhost/nevjegyek/admin/ címet, és az oldalt az index.php néven mentettük, akkor az megjelenít egy listát. Ez viszont azt jelenti, hogy gyakorlatilag bárki hozzáférhet az oldalunkhoz, amit nem biztos, hogy szeretnénk. Egyáltalán nem célunk, hogy az oldal egy nyitott közösségi platformmá váljon, ahol mindenki szabadon böngészhet. A védelmi szintek kiépítése többféleképpen megvalósítható, és attól függ, mennyi időt tudunk rászánni a fejlesztésre.

**Ha csak 3 másodpercünk van**, mert például sietnünk kell elérni a buszt, akkor egy gyors, de ideiglenes megoldás lehet, hogy létrehozunk egy üres index.php fájlt. Ez elrejti a mögöttes fájlokat, mivel a böngészők alapértelmezésben ezt keresik a könyvtárban. Persze, ez egyáltalán nem tökéletes megoldás, mert ha valaki kitalálja, hogy a lista.php fájl is létezik, akkor gond nélkül bejuthat a rendszerbe. Ez tehát nem igazi védelem, csak annyit érünk el vele, hogy az oldalunk nem 10 másodperc alatt, hanem talán 15 másodperc alatt lesz feltörhető.

### 14.10.1.htaccess

**A másik megoldás**, amely már sokkal biztonságosabb, a .htaccess fájl használata. Ha beírjuk a Google-be, hogy ".htaccess generator", több olyan weboldalt találhatunk, ahol segítenek létrehozni ezt a fájlt. A .htaccess fájlnak nincs neve, csak kiterjesztése van, és valójában egy rejtett fájl a Linux rendszereken. Ezt a fájlt az Apache szerver könyvtárába helyezve mindenféle "okosságot" írhatunk bele, például beállíthatjuk, hogy a rendszer csak jelszó megadása után engedjen be bárkit. Ehhez viszont szükség lesz egy másik fájlra, a .htpasswd fájlra, amely a felhasználónév és jelszó párokat tartalmazza.

A [**https://www.htaccessredirect.net/**](https://www.htaccessredirect.net/) oldalon például generálhatunk egy .htaccess fájlt, amely jelszavas védelmet biztosít, de szükségünk lesz egy jelszót titkosító fájlra is. Ha például ellátogatunk a [**https://hostingcanada.org/htpasswd-generator/**](https://hostingcanada.org/htpasswd-generator/) oldalra, ahol megadhatunk egy felhasználónevet és egy jelszót (például admin és 1234), a rendszer egy olyan fájlt hoz létre, amely titkosítja ezeket az adatokat. Így egy ilyen titkosított sort kapunk: latiza:{SHA}cRDtpNCeBiql5KOQsKVyrA0sAiA=. Ezt a kódot elmentjük .htpasswd fájlként, és máris van egy alapvető jelszavas védelmünk. Ez a módszer könyvtárszintű védelmet biztosít, tehát az adott könyvtárban található összes fájlra érvényes lesz.

**Azonban mi lap szintű védelmet szeretnénk kiépíteni**, ami azt jelenti, hogy egy kicsit jobban bele kell merülnünk a sütik és a session-ök világába. Valószínűleg mindannyian találkoztunk már sütikkel, amikor egy weboldalon arra kérnek, hogy fogadjuk el őket. A sütik olyan fájlok, amelyeket a böngésző kezel, és amelyek kulcs-érték párok formájában tárolnak információkat. Ezeket a sütiket egy adott domainből akár 20 példányban is tárolhatja a böngésző. A sütikben olyan adatokat tárolhatunk, mint például a kiválasztott nyelv, amelyet legközelebb, amikor az oldalra látogatunk, már nem kell újra beállítani. Ezek az adatok a böngészőn belül, a Network fül alatt a Cookies részben megtekinthetők.

### Sütik

**A sütik korlátozott élettartamúak**, ami azt jelenti, hogy ha a böngészőt bezárjuk, vagy valamilyen tisztító programot futtatunk (például a Windows tisztító programjait), akkor ezek a sütik törlődhetnek. Ráadásul minden süti rendelkezik egy lejárati idővel, amit a böngésző automatikusan figyel, és amikor az idő letelik, a böngésző automatikusan törli a sütit. Van olyan süti is, amely a böngésző bezárásakor azonnal megszűnik, ha annak lejárati idejét a múltba állítjuk. A sütik problémája azonban az, hogy az adatok a felhasználó gépén tárolódnak, és manipulálhatók. Ezt "süti mérgezésnek" hívják, amikor valaki meghosszabbítja vagy módosítja egy süti lejárati idejét.

**Ezért nem ajánlott érzékeny adatokat sütikben tárolni.** Van egy szerveroldali alternatíva, amelyet **session**-nek hívunk, és amely a szerver gépen tárolódik. A session kezelésével a szerver biztonságosabb módon tud adatokat tárolni, bár ennek is vannak sebezhetőségei. A session azonban sokkal megbízhatóbb, mivel a szerveren tárolódik, és a felhasználó nem fér hozzá közvetlenül. A session adatok kezelésére szolgál a **$\_SESSION** szuperglobális tömb, míg a sütik esetében a **$\_COOKIES** tömb áll rendelkezésünkre.

A GET és POST módszerek szintén fontos szerepet játszanak az adatok továbbításában, például amikor egy űrlapot küldünk be. Ha GET módszert használunk, az adatok a böngésző címsorában is megjelennek, ami nem ajánlott érzékeny adatok (például jelszavak) továbbítására. A POST módszer viszont biztonságosabb, mivel az adatok a HTTP üzenet törzsében kerülnek továbbításra, nem láthatók a címsorban, és nem tárazódnak el a böngészőben.

A **session alapú védelem** megvalósításához először el kell indítanunk a session-t minden oldalon, amit védeni szeretnénk. Ehhez a következő kódot használjuk:

session\_start();

if (!isset($\_SESSION['belepett'])) {

header("Location: index.php");

exit();

}

Ezzel a módszerrel csak azok a felhasználók férhetnek hozzá az admin oldalhoz, akik be vannak jelentkezve. A bejelentkezés során a session-be mentjük, hogy a felhasználó sikeresen bejelentkezett, és ezt a státuszt ellenőrizzük minden védett oldalon.

Végül, ha a felhasználó ki szeretne lépni, egy egyszerű kijelentkező kóddal törölhetjük a session-t, és visszairányíthatjuk a beléptető oldalra:

session\_start();

unset($\_SESSION['belepett']);

session\_destroy();

header("Location: index.php");

**Összegzésképpen**, a lap szintű védelem egy hatékony módszer arra, hogy csak a megfelelő jogosultságokkal rendelkező felhasználók férjenek hozzá az adminisztrációs felülethez. A session alapú védelem jó alapot biztosít egy beléptető rendszerhez, amely megakadályozza, hogy bárki közvetlenül hozzáférhessen az oldalhoz anélkül, hogy bejelentkezett volna.

A másik a post típusú módszer. A request viszont a get és post tulajdonképpeni úniója, ugyanis ha gettel és posttal küldök adatot, akkor az a request tömbben is megjelenik, tehát bármelyiket el lehet csípni a request tömb segítségével.

A get és post parancsokat http protokolloknak hívják, a fejlesztőben a network fülön ha betöltjük a w3school oldalt és a network fület nézzük, akkor vizsgálhatjuk mik töltődnek le, amiket szét is lehet válogatni, hogy mik töltődnek le, js stb.

Gyakorlatilag mikor betöltődik az oldal, akkor ezek letöltődnek, és betöltési időt és méretet is meg tudjuk nézni, státuszt tudjuk vizsgálni,a 304 akkor az cache-be betöltődött, tehát gyorsító tározódott, a böngésző letöltéskor megnézi van e neki ilyen eszköz és fölöslegesen nem terheli a hálózatot és nem tölti le újra.

Type a mime típusok, amit használtunk. Ha pl belemegyünk egy képbe, akkor meg lehet nézni milyen tulajdonságai vannak egy ilyen képnek, ez minden fájltípusnál más, a headersben pedig, látható a lezajlott kérés válasz, mennyi az idő, milyen szervert használ, mi fut azon a szerveren, milyen karakterkódolást használnak a rendszerben, a kliens elküldi saját magát, hogy ő pl egy chrome, ezek alapján eldönti a rendszer ha van egy fájl, és ugyanakkora a fájl és ugyanaz a neve, akkor azt gyorstárból használja. (network fül alatt rá kell kattintani valamelyik betöltődő elemre és ott jelenik meg)

Amikor ajaxot használ valaki ami azt jelenti javascript és szerver kommunikáció a háttérben, és az oldalt úgy frissítem, hogy menet közben kicserélek bizonyos részeket benne, és ezek a válaszok különböző formátumban érkeznek, vagy html-ként vagy xml-ként, vagy json-ként ez a javascript object notation,.

A get és post különböző dolgokra való igazából, a get mint módszer látszik a címsorban, az azt jelenti hogy nagyon érzékeny adatokat ne tegyünk bele, ne szülessen olyan link, hogy / default.asp?jelszo=nagyontitkos, mert ezt gyakorlatilag mindenki fogja látni, tehát a get típusú paraméter átadásnál megjelenik a címsorban, ezért nem szabad használni érzékeny adatokat.

Ha feltöltünk fájlt, az is furán nézne ki, ha a címsorban mindenféle krixkraxok zöldségek megjelennének, tehát fájl feltöltésnél sem ajánlatos használni, ez igazából mivel az url címben látszik, ezért könyvjelzőkbe tehető, tehát ha pl a w3s oldalánn kattintgatunk, látszik a böngészőben az oldal és szépen könyvjelzőbe el tudjuk menteni az aktuális állapotot, aztán a böngésző előzményeibe is bekerül, tehát ez a vissza gomb, ezzel folyamatosan bekerül a history objectumba javascriptnél, viszont van egy méret megkötése, hogy **2048 karaktert** **tudok átvinni getes átadással**, ezek inkább feldolgozó programoknál érdekes, ha valaki később készít egy restfull apit, ami egy oldalas alkalmazást szolgál ki.

A getes módszernél ha érdekes karaktert kell átadni pl szóközt, akkor annak url változata jelenik meg pl % és ezek a paraméterek a fejrészben adódnak át. bud.hu-nál is látszik, a network fülnél ki lehet keresni, ha valamit keresünk látszik a getes paraméter, ha viszont postos küldés van az a http üzenetben megy át a testében a bodyban, tehát a lényeg, hogy a post soha nem tárazódik, soha nem lehet könyvjelzőbe tenni, nem látható az url címben, nincsen méret megkötése, adatok küldéséhez használható, kétfajta kódolása van a multipart bináris adatokhoz, amit a képfeltöltsénél néztünk, tehát nincsen kódolási megkötés, és bináris adat is küldhető. Elsősorban mindig azt szokták mondani hogy lehetőség szerint post típusú kérést használjunk, mert biztonságosabb. Ezzel át is néztük a post és getes kéréseknek a lényegét, lépjünk vissza az index.php oldalunkra, tehát a get és post azért volt hogy adatokat töltsünk fel űrlapokról, vagy pedig linkekre rátegyük különböző értékeket, a filest is megnéztük, a fájl feltöltő űrlapnál, és van itt ez a kettő újdonság a cookies és session.

A cookies nevű szuperglobális tömbbe tudok beállítani különböző sütiket, kulcs érték párokkal, és lejárati idővel. Ami viszont nekünk most a feladathoz kell, azt a munkamenet változókkal a sessionnal meg tudjuk oldani.

### A session

A get és post esetében, mindig van valami ami előállítja, és mindig van valami ami elkapja az adatokat. A session viszont egy picit másképpen működik, ennek is kell értéket beállítani és ezt is el lehet olvasni, viszont ennek egy ilyen saját emlékezete lesz a rendszerben, és ezáltal ameddig ezt a böngésző ablakot nem csukom be, addig gyakorlatilag egy ilyen memóriája alakul ki, ezt így tudjuk beaktiválni, hogy először is ki kell írni session start ekkor lehet elindítani ezt a munkamenet kezelőt, ami után ebbe a session globális változóba fel lehet venni különböző értékeket, pl: $\_SESSION

<?php

session\_start();

$\_SESSION[’belépett’] = true; // itt éppen egy logikai értéket adok neki, mert ezt még nem nagyon használtuk,

?>

tehát ezt a sessiont felveszem ezen az index.php oldalon így, és egy másik oldalon, pl az admin oldalon belül a listak.php-t és ott a második sorba beírjuk: session\_start(); utána print\_r($\_SESSION);

gyakorlatilag ha ezt így meghívjuk, akkor array üresen kerül a tetejére, viszont van egy másik weboldal ahol adtunk neki egy értéket, az itt megjelenik, és ezzel gyakorlatilag lesz egy ilyen kollektív tudata az oldalaknak, és nem kell mindig feldobni elkapni, mint a posttal és a gettel. de csak akkor látszik a lista.php-ben az arrayban a belépett, amíg meg van nyitva az index.php, ha bezárom az index.php, és újra indítom a lista.php –t akkor az array-ből eltűnik a belépett felirat, tehát gyakorlatilag az oldal bezárásáig fog működni. Kivétel ez alól mondjuk ha valaki úgy hagy egy gépet több mint fél órára, akkor újból úgymond kijelentkezteti a rendszer. Tehát van egy alap beállítási idő fél órát ad egy munkamenetre, ami persze tetszőlegesen átállítható más időre, de a default érték az 30 perc.

Ha egy oldalt szeretnék megvédeni, akkor ezt úgy tudom megtenni, hogy van egy beléptető űrlap, jelen esetben az index.php, és ha ezen a beléptető űrlapon valaki jól lép be, akkor kap egy ilyen értéket, hogy pl session belépett true, és csak itt lehet jogosítványt szerezni arra, hogy az oldalra belépjek. Tehát nem lehet az ablakon bemászni, hogy beírom az adott oldalt, mert én kitaláltam a nevét, pl a lista.php-t, hanem hogyha kitaláltam a nevét, de nincs megfelelő jogosítványom, akkor vissza zavarnak a bejárathoz. Gyakorlatilag mi is egy ilyet fogunk készíteni, hogy lesz először is a védett oldalakon egy figyelő, ami az illetékteleneket elzavarja a bejárathoz, tehát a lista.php oldalamra felvisszük:

### Lapvédelem

session\_start();

if (!isset($\_SESSION['belepett'])) { //ha nem létezi a belépett, akkor menjen vissza bejárathoz

header("Location: index.php"); // azaz ide

exit(); // Ha valaki tart attól hogy a header tovább fut, mert vannak ilyen kósza híresztelések, akkor adjunk meg itt egy exit parancsot és innentől kezdve biztos, hogy a php-meg fogja szüntetni az oldal futását.

} de ezt be kell másolni az összes többi oldalra. az index esetén csak session\_start()

Tehát a headernél vissza pattan akkor ha nincsen ilyen belépett jogosítványa, ha nem pattanna vissza, akkor is leáll a programnak a futása az exit-tel. Gyakorlatilag ez egy lapvédelem, amit minden egyes oldalra be kell tenni, hogy ne lehessen azt hogy illetéktelenek itt kószálhassanak, tehát gyakorlatilag, minden ami az adminban van azt meg kell védeni, úgyhogy a többi oldalt is meg kell nyitni és be kell másolni. Ezek után próbálj meg bebújni a kertek alatt azzal, hogy beírod lista.php nem fog beengedni, ha minden rendben van.

A modositast ki kell egészíteni ezzel is: mert enélkül hibát ad

if (!isset($\_REQUEST['id'])) header("Location: lista.php");

tehát ha nem létezik a header location, akkor is vissza a feladónak, csak itt a táblázathoz ugrasszuk vissza, ami a lista. Ezzel megakadályoztuk azt, hogyha nem kap id-t a rendszer, akkor ne írjon ki hiba üzenetet, ha kap id-t akkor jól fog működni. Ha nem működne zárjunk be minden ablakot, hogy elvesszen a munkamenet és indítsuk újra, csak előtte kommenteljük ki az index.php-n a sesssion true sort, mert emiatt beenged mindenhova.

Tehát egy beléptető oldalra van szükségünk, hogy ne legyen beleégetve a session-ba a true, írjuk meg a html szakaszát, majd folytassuk a phpt

**<?php**

**session\_start();**

**if (isset($\_POST['rendben'])) {**

// itt tehát ha a post rendben akkor, utána történhet valamiféle beavatkozás. Próbáljuk ki, hogy a html formban ha átírjuk a method postot getre, és beírunk e-mail címet és jelszót, akkor az urlben meg fog jelenni a nagyontitkos kulcsszó. A másik amiért nem érdemes a getet használni, hogy így bele lehet nyúlni az adatokba, és bele tudunk írni valami olyasmit a linkbe amit nem lehetne, és ha ezt a szerver oldal feldolgozza, akkor igazából meg lehet hegeszteni a szerver oldalt, ez az url befecskendezés, ezt néztük ugye a lapozónál is ezt a biztonsági problémát, és ha nincs erre felkészítve a szerver oldal, akkor nagyon nagy problémák sülhetnek ki ebből. Tehát fontos, hogy a form tagben mindenképpen az kerüljön bele, hogy method= post!!! Azt is mondhatjuk, hogy a jelszavas mezőknél, és a fájl feltöltő űrlapoknál ez kötelező, a többinél meg csak ajánlott, ha megnyomták a rendben gombot, ugyanúgy mint az űrlap feldolgozásnál először is tisztítani kell az átadott értékeket. Tehát van egy mail cím, amit átveszünk a post emailből, és van egy jelszó amit szintén átveszünk.

// Változók tisztítása – w3s oldalán sok példa van php form validation ha többmindent akarsz beleépíteni.validate name – reguláris kifejezések mint preg\_match, vannak még itt további szörnyűségek, pl a valdate url alatt, (nem kezdő szint) de akár be lehet ezt is másolni a rendszerünkbe.

**$email = strip\_tags(strtolower(trim($\_POST['email'])));**

**$jelszo = $\_POST['jelszo'];** //itt sokmindent nem kell megadni

// Változók ellenőrzése, másolhatunk a w3s oldaláról, itt majd nem csinálunk hibák tömböt, mert nem akarjuk segíteni a hackert, hanem csak egyszerű üzenetet küldünk vissza. Az a lényeg, hogy ne akarjuk megtanítani a hackernek, hogy jó a mail cím már rendben van, már csak a jelszót kell kitalálnod, tehát azt azért igazán elvárhatjuk a felhasználótól, hogy két adatot jól adjon meg, ezért nem is bontogatjuk úgy szét, mint ahogyan a múltkor tettük.

**if (empty($email) ||** //ha mail cím nem üres, mert lehet a beviteli mezőt telenyomta spacekkel, akkor tisztás után még lehet üres, majd megnézzük filterrel, nem valós e a mail cím, mert a vagynál azt kell tudni, hogy ha vagy vagy nem igaz, de egy igaz, akkor az egész feltétel igaz lesz, a php feldolgozás úgy működik, ha a első igaz, akkor ami tőle jobbra található, azokat már ki sem értékeli, hanem kiszáll belőle igazzal. Ha ez az első feltétel itt nem igaz, akkor meg tovább a filterre, ha az nem igaz, vagyis igaz, akkor ott kiszáll, tehát az a lényeg, hogy a vagy feltétel esetén az első vagy feltételnél kiszáll igazzal. Az és feltételnél meg az első hamisnál száll ki, és az egész hamis lesz. Ez azért érdekes majd hogyha if feltételbe több tagot teszek bele, akkor a sorrenden lehet esetleg a program sebességén változtatni, gyorsítani. ha a jelszó nem olyan, mint amit kértünk, hogy a-tól z-ig karakterek szerepeljenek benne,

**!filter\_var($email, FILTER\_VALIDATE\_EMAIL) ||**

**!preg\_match("/^[a-zA-Z ]\*$/", $jelszo)) {**

**$hiba = "Rossz e-mail címet vagy jelszót adtál meg!";**

**}**

// Beléptetés: először bedrótozott jelszavak lesznek a rendszerben:\*zöldet később, ha kijön a beégetett jelszó és mail cím

**else {**

// Sikeres ide megadhatjuk így is:\* itt kezdjük

if ($email == [jancsi@gmail.com](mailto:jancsi@gmail.com) && $jelszo == „juliska”){ //de ez így nem az igazi mert látszik a jelszó

**if (mysqli\_num\_rows($eredmeny) > 0) {**

**$\_SESSION['belepett'] = true;**

**header("Location:lista.php");**

**}**

// Sikertelen, ezt a htmlben adjuk meg hogy írja ki:

**else {**

**$hiba = "Hibás e-mail címet vagy jelszót adtál meg!";**

**}**

}

}

?><!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Névjegykártyák</title>

<link href="stilus.css" rel="stylesheet">

</head>

<body>

<h1>Belépés</h1>

<form method="post" action="">

<?php if (isset($hiba)) print $hiba; ?>

<p><label for="email">E-mail:\*</label><br>

<input type="email" id="email" name="email" required></p>

<!--1 ez a require nem a legszerencsésebb ez a natív védelem, mert a böngészőből ki lehet szedni egy f12 és inspektorral,ráadásul régebbi explorereknél nem is működik. A type=email is text tipussá butul vissza a régi böngészőkben, így átengedik a rossz mail címet is. Tehát a lényeg, hogy az kevés, hogy a html5-ben vannak lehetőségek,a hibák ellenőrzésére. Akár itt használhatunk mintákat, reguláris kifejezéseket,html5 patterns.com-on meg lehet nézni a lehetőségeket, tehát sok mintát lehet találni telefonszám, bankkártyaszám stb.ellenőrzésére. -->

<p><label for="jelszo">Jelszó:\*</label><br>

<input type="password" id="jelszo" name="jelszo" required></p>

<p><em>A \*-gal jelölt mezők kitöltése kötelező!</em></p>

<input type="submit" id="rendben" name="rendben" value="Belépés">

</form>

</body>

</html>

**Szükségünk lesz kilepes.php oldalra is:**

**<?php**

**session\_start();**

**unset($\_SESSION['belepett']);**

**session\_destroy();** //tokkal vonóval megszünteti az összes változót, webshopnál is ezt érdemes használni és csak azt kell megadni, hova pattanjon és ennyi a kijelentkezés, tehát vissza dobjuk a beléptető felülethez. a listánál érdemes egy kilépés gombot is feltenni.

**header("Location: index.php");**

**?>**

**lista.php htmljébe fel kell vinni a kilépést linkkel!!**

<p><a href=”felvitel.php”>Új névjegy</a> | <a href=”kilepes.php”>Kilépés</a></p>

próbáljuk ki, és ezzel meg van oldva a rendszerünk egyszerű védelme, ami tényleg nagyon egyszerű, nincsen túl bonyolítva, viszont a valóságban ez annyira nem menő. Egyrészt ha valaki megnézi a php kódot ott van a jelszó, amit csak a rendszergazda tud megváltoztatni, tehát ha a megrendelőnek oda adjuk az oldalt ő nem tud innen kitiltani, mert nem tudja megváltoztatni, ezért ez ilyen szempontból nem túl jó. Van egy következő evolúciós változat. Ezért nevezzük át egyszerű index.php- nek, hogy megmaradjon, és nézzük a következő szintet.

### HASH függvények

A jelszavak biztonságos tárolása és feldolgozása az informatikában egy kulcsfontosságú terület, amelyhez különféle titkosítási módszereket használhatunk. Először is nézzünk egy kis titkosítási történelmet.

A **hash függvények** olyan algoritmusok, amelyek bármilyen hosszúságú adatot rögzített hosszúságúra képeznek le. Ezeket az eljárásokat hash értékek előállítására használják, és elsődleges céljuk az adat integritásának ellenőrzése, például annak megállapítása, hogy az adatot manipulálták-e vagy sem.

A hash függvények egy adott bemenet alapján mindig ugyanazt a kimenetet produkálják, azonban visszafejtésük nagyon nehéz, és gyakorlatilag egyirányú folyamatnak tekinthetők. Az 1980-as évek végén váltak különösen fontossá, amikor az elektronikus aláírás és az adatok biztonságos tárolásának szükségessége felmerült. A tárolástechnika terén a hash függvények már az 1950-es évek elején is megjelentek.

A következő példában egy egyszerű jelszót titkosítunk különböző hash algoritmusok segítségével:

<?php

$jelszo = "juliska"; // A titkosítandó jelszó

print "<p><b>Jelszó:</b> {$jelszo}<br>";

Ebben a kódban egy egyszerű szöveget, a "juliska" jelszót használjuk, amelyet különféle hash algoritmusokkal fogunk titkosítani. Az első ilyen algoritmus az **MD5**:

print "<p><b>md5():</b> ".md5($jelszo)."<br>";

Az **MD5** egy hash függvény, amely 32 karakter hosszú hexadecimális karakterláncot generál. Régebben széles körben használták a jelszavak és adatok titkosítására, de mára már kevésbé biztonságosnak számít, mivel feltörhető a brute-force és más támadási módszerekkel.

A következő algoritmus a **SHA1**:

print "<p><b>sha1():</b> ".sha1($jelszo)."<p>";

A **SHA1** egy másik hash algoritmus, amely 40 karakter hosszú hexadecimális kimenetet generál. Ez már valamivel biztonságosabb, mint az MD5, de manapság ez sem számít elég biztonságosnak, különösen olyan esetekben, ahol magas szintű védelemre van szükség.

Mindkét hash függvény az adott jelszóból egy olyan "zöldség"-et, azaz hexadecimális karakterekből álló sorozatot készít, amely a bemeneti adat alapján mindig ugyanazt az eredményt adja, viszont a visszafejtése rendkívül nehéz. Bár elméletileg visszafejthetetlenek, a gyakorlatban speciális módszerekkel (például rainbow table támadásokkal) mégis feltörhetők.

**Példa kimenet**:

* MD5: 5b6c2fbd7a43c2f215becc015e7fc4b5
* SHA1: 49cef48df229f6e608f4b57c11ef05c4f014f0c6

Ez a két titkosítási módszer nem visszafejthető, azonban a jelszavak hash értékei összehasonlíthatók, így ha valaki beírja a "juliska" jelszót, és a hash értéke megegyezik az előre eltárolttal, akkor a rendszer hitelesítheti a felhasználót anélkül, hogy a jelszót valójában tárolnia kellene.

Ha biztonságosabb titkosítást szeretnénk, választhatjuk a **SHA1** algoritmust az MD5 helyett. Ehhez egy egyszerű módosítást kell végeznünk az eredeti jelszóellenőrzési kódban. Tegyük fel, hogy van egy hitelesítési fájlunk, amely a felhasználók bejelentkezését kezeli. A jelszó ellenőrzésénél most már az SHA1 algoritmussal hash-elt jelszót fogjuk használni:

if ($email == "jancsi@gmail.com" && sha1($jelszo) == "49cef48df229f6e608f4b57c11ef05c4f014f0c6") {

// Sikeres bejelentkezés

}

Ebben a példában a jelszót hash-eltük a SHA1 algoritmussal, és csak akkor léptetjük be a felhasználót, ha a megadott jelszó hash értéke egyezik az adatbázisban tárolt hash értékkel.

**Összegzés**:

* A hash algoritmusok, mint az MD5 és a SHA1, biztonságosabb módot kínálnak a jelszavak tárolására, mint a sima szöveges tárolás.
* Ezek a módszerek nem visszafejthetők, ami azt jelenti, hogy a jelszavak biztonságosabbak, de vannak módszerek, amelyekkel ezek az algoritmusok is feltörhetők.
* A gyakorlatban már léteznek sokkal erősebb hash algoritmusok is, például a SHA-256 vagy a bcrypt, amelyek biztonságosabbak a modern alkalmazásokban.

Ez a megoldás már jelentős előrelépés a biztonság terén az előzőekhez képest, bár léteznek ennél modernebb és még erősebb titkosítási megoldások is.

A hash függvények, mint az **MD5** és a **SHA1**, korábban széles körben használtak voltak a jelszavak biztonságos tárolására, azonban a modern kriptográfia és a számítástechnika fejlődésével már nem számítanak biztonságosnak, nagyon rövid idő alatt történelemmé váltak az infótanárok nagy örömére, hogy nem győzik fejleszteni minden évben a feladataiakat.

A brute-force és a rainbow table támadások révén ezek az algoritmusok könnyen feltörhetők. Ezért ma már ajánlott modernebb és biztonságosabb megoldásokat használni, mint például a **SHA-256**, a **bcrypt**, az **argon2**, vagy a **scrypt**.

#### 1. ****SHA-256****:

Ez az algoritmus a Secure Hash Algorithm (SHA) család része, és erősebb védelmet nyújt, mint az MD5 vagy a SHA-1. A **SHA-256** 256 bites hash-t állít elő, amely sokkal nagyobb és bonyolultabb, így ellenállóbb a brute-force támadásokkal szemben.

Például egy SHA-256 hash előállítása PHP-ben így néz ki:

$jelszo = "juliska";

$sha256Hash = hash('sha256', $jelszo);

print "<p><b>SHA-256 Hash:</b> {$sha256Hash}<br>";

**Példa kimenet**: c4993768e75b150cc91785a83cb6a4fda12f453018a36f71d35c2f07ac7dd435

A **SHA-256** már sokkal biztonságosabb, mint az MD5 vagy a SHA1, de mivel még mindig egyetlen iterációval működik, az extrém számítási kapacitással rendelkező támadók (például botnetek vagy speciális hardverek) ellen nem nyújt teljes védelmet.

#### 2. ****bcrypt****:

A **bcrypt** egy olyan titkosítási algoritmus, amely nem csak egyszerűen hash-eli a jelszavakat, hanem beépített lassítást is tartalmaz, amely ellenállóvá teszi a brute-force támadásokkal szemben. A **bcrypt** ráadásul egy "salt"-ot is generál minden egyes jelszóhoz, ami védi a hash-t a rainbow table támadásoktól.

A **bcrypt** PHP-ben a password\_hash() függvénnyel könnyen használható:

$jelszo = "juliska";

$bcryptHash = password\_hash($jelszo, PASSWORD\_BCRYPT);

print "<p><b>bcrypt Hash:</b> {$bcryptHash}<br>";

**Példa kimenet**: $2y$10$TkhvUrfbQ6EwBhORroMnT.blxZcdD4RjG5nbkUIBTXer9eCH2NYZq

Az itt látható bcrypt hash egy véletlenszerű salt-ot is tartalmaz, így még ha két felhasználó ugyanazt a jelszót adja meg, a hash értékük akkor is eltérő lesz. A **bcrypt** ráadásul beépített "lassító" mechanizmussal dolgozik, amitől nehézkesebb és időigényesebb lesz a brute-force támadások kivitelezése.

A **bcrypt** esetében a jelszó ellenőrzése is egyszerű:

if (password\_verify($jelszo, $bcryptHash)) {

print "Jelszó helyes!";

} else {

print "Jelszó helytelen!";

}

#### 3. ****argon2****:

Az **argon2** a legújabb és legbiztonságosabb algoritmus, amelyet jelszavak titkosítására használnak. Az **argon2** algoritmust úgy tervezték, hogy ellenálljon a brute-force támadásoknak, és memória-költséges támadások ellen is védelmet nyújt. 2015-ben megnyerte a Password Hashing Competition-t, így jelenleg a legbiztonságosabbnak tartott algoritmus.

A PHP támogatja az **argon2**-t is, a használata nagyon hasonlít a bcrypt-hez:

$jelszo = "juliska";

$argon2Hash = password\_hash($jelszo, PASSWORD\_ARGON2I);

print "<p><b>argon2 Hash:</b> {$argon2Hash}<br>";

**Példa kimenet**: $argon2i$v=19$m=65536,t=4,p=1$...

Az **argon2** biztonságossága abban rejlik, hogy sokkal több számítási és memóriakapacitást igényel a hash előállítása, így a támadók számára nehezebb és költségesebb ezt brute-force technikákkal megtámadni.

A jelszó ellenőrzése az **argon2** esetében is hasonlóan történik:

if (password\_verify($jelszo, $argon2Hash)) {

print "Jelszó helyes!";

} else {

print "Jelszó helytelen!";

}

#### 4. ****scrypt****:

A **scrypt** hasonló a bcrypthez, de még nagyobb hangsúlyt fektet a memóriaköltségek növelésére, ami további védelmet nyújt a brute-force támadásokkal szemben. A **scrypt** különösen hatékony a speciális hardverrel (pl. ASIC) történő támadásokkal szemben.

PHP-ben a **scrypt** használatához külön csomagokra lehet szükség, de a hasonló hash-elési algoritmusokhoz képest ez is egy robusztus megoldás, amely biztonságosabbá teszi a jelszókezelést.

#### Miért érdemes erős algoritmusokat használni?

* **Salt használata**: A modern algoritmusok, mint a bcrypt vagy argon2, automatikusan használnak salt-ot, ami egy véletlenszerű adatdarab, amit minden jelszóhoz hozzáadnak, így még ha két felhasználó ugyanazt a jelszót adja meg, a hash értékük eltérő lesz.
* **Lassító mechanizmus**: Az algoritmusok, mint a bcrypt és az argon2, beépített lassítást alkalmaznak, hogy a brute-force támadások kivitelezése időigényes legyen.
* **Erős titkosítás**: A SHA-256, bcrypt, argon2 és scrypt olyan algoritmusok, amelyek napjainkban megfelelő biztonságot nyújtanak, és ellenállóak a modern támadásokkal szemben.

#### Összegzés:

A jelszavak tárolása és feldolgozása során mindig fontos olyan algoritmusokat használni, amelyek megfelelő védelmet nyújtanak a támadások ellen. Bár az **MD5** és **SHA1** algoritmusok már nem számítanak biztonságosnak, a **SHA-256**, **bcrypt**, **argon2** és **scrypt** algoritmusok már korszerű, erős védelmet biztosítanak. A legjobb gyakorlat, ha minden jelszóhoz egyedi salt-ot használunk, és beépített lassító mechanizmussal nehezítjük meg a brute-force támadásokat.

## Az új RESTful Api alkalmazásnak megfelelő titkosítás

### ****composer.json**** (Telepítjük a JWT csomagot)

A composer.json fájlban meghatározzuk a JWT csomagot:

{

"require": {

"firebase/php-jwt": "^6.0"

}

}

Ezután a parancssorban a gyökérkönyvtáradból futtasd a következő parancsot a **JWT csomag telepítéséhez**:

composer install

Ez létrehozza a vendor/ mappát, amely tartalmazza a JWT csomag összes függőségét.

### ****connect.php**** (Már meglévő adatbázis kapcsolat)

### ****middleware.php**** (Token ellenőrzése)

Ez a fájl biztosítja, hogy az adminisztrációs oldalakhoz csak hitelesített felhasználók férhessenek hozzá.

<?php

require\_once '../vendor/autoload.php'; // Composer autoload fájl

use \Firebase\JWT\JWT;

class Middleware {

private static $secretKey = 'titkos\_kulcs'; // Titkos kulcs a JWT számára

public static function validateToken() {

// Ellenőrizzük, hogy van-e token a fejlécben

$headers = apache\_request\_headers();

if (!isset($headers['Authorization'])) {

http\_response\_code(401);

echo json\_encode(['error' => 'Token nem található.']);

exit();

}

// Kivesszük a "Bearer " részt a tokenből

$token = str\_replace('Bearer ', '', $headers['Authorization']);

try {

// Token dekódolása

$decoded = JWT::decode($token, self::$secretKey, array('HS256'));

// Token validálása sikeres

return $decoded;

} catch (Exception $e) {

// Hibás vagy lejárt token

http\_response\_code(401);

echo json\_encode(['error' => 'Hibás vagy lejárt token.']);

exit();

}

}

}

?>

### ****login.php**** (Belépés)

Ez a fájl a felhasználó belépését kezeli, és sikeres belépés esetén JWT-t generál.

<?php

require\_once '../connect.php';

require\_once '../vendor/autoload.php';

use \Firebase\JWT\JWT;

$secretKey = 'titkos\_kulcs'; // A JWT titkos kulcsa

// Ellenőrizzük, hogy POST kérés érkezett-e

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

// Adatok beolvasása és tisztítása

$email = mysqli\_real\_escape\_string($dbconn, strip\_tags(trim($\_POST['email'])));

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

// Felhasználó keresése az adatbázisban

$sql = "SELECT \* FROM felhasznalok WHERE email = ?";

$stmt = $dbconn->prepare($sql);

$stmt->bind\_param('s', $email);

$stmt->execute();

$result = $stmt->get\_result();

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

$user = $result->fetch\_assoc();

// Jelszó ellenőrzése

if (password\_verify($jelszo, $user['jelszo'])) {

// Token generálása

$payload = [

'iat' => time(), // Token létrehozásának időpontja

'exp' => time() + (60 \* 60), // Token lejárata (1 óra)

'userId' => $user['id'], // Felhasználó ID

'email' => $user['email'] // Felhasználó e-mail címe

];

$token = JWT::encode($payload, $secretKey);

// Válasz a kliensnek

echo json\_encode([

'message' => 'Sikeres belépés',

'token' => $token

]);

} else {

// Hibás jelszó

http\_response\_code(401);

echo json\_encode(['error' => 'Hibás jelszó.']);

}

} else {

// Nem található felhasználó

http\_response\_code(401);

echo json\_encode(['error' => 'Nem található ilyen felhasználó.']);

}

}

?>

### ****index.php**** (Lapvédelem JWT használatával)

Ez egy védett oldal, amelyhez csak hitelesített felhasználók férhetnek hozzá. A **middleware.php** segítségével ellenőrizzük a token érvényességét.

<?php

require\_once '../middleware.php';

// Token ellenőrzése

$decodedToken = Middleware::validateToken();

// Ha a token érvényes, folytatjuk az oldal megjelenítését

?>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Admin Oldal</title>

</head>

<body>

<h1>Üdv az adminisztrációs oldalon!</h1>

<p>Felhasználó ID: <?= $decodedToken->userId; ?></p>

<p>Email: <?= $decodedToken->email; ?></p>

</body>

</html>

### ****htaccess**** (RESTful API védelme)

Az **Apache** alapú szerverek esetén használhatjuk a .htaccess fájlt, hogy ne lehessen közvetlenül elérni a fájlokat. Egy egyszerű .htaccess fájl az admin mappában, hogy csak a meghatározott fájlok legyenek elérhetőek:

apache

Kód másolása

<Files \*.php>

Require all denied

</Files>

<Files "login.php">

Require all granted

</Files>

<Files "register.php">

Require all granted

</Files>

bash

Kód másolása

composer install

### ****login.html**** – Felhasználónév és jelszó űrlap

Ez az a fájl, ahol az admin beírhatja az e-mail címét és jelszavát, és elküldi a szervernek a hitelesítési adatokat.

html

Kód másolása

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Admin Belépés</title>

<link rel="stylesheet" href="../stilus.css">

</head>

<body>

<h1>Admin Belépés</h1>

<form id="login-form">

<p><label for="email">E-mail:</label><br>

<input type="email" id="email" name="email" required></p>

<p><label for="jelszo">Jelszó:</label><br>

<input type="password" id="jelszo" name="jelszo" required></p>

<button type="submit">Belépés</button>

</form>

<div id="hiba-uzenet"></div>

<script>

document.getElementById('login-form').addEventListener('submit', function(e) {

e.preventDefault(); // Ne küldje el automatikusan az űrlapot

let email = document.getElementById('email').value.trim();

let jelszo = document.getElementById('jelszo').value.trim();

// Fetch API használata a bejelentkezéshez

fetch('login.php', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: email,

jelszo: jelszo

})

})

.then(response => response.json())

.then(data => {

if (data.success) {

window.location.href = 'admin/dashboard.php'; // Átirányít a védett admin felületre

} else {

document.getElementById('hiba-uzenet').innerHTML = data.error;

}

})

.catch(error => console.error('Hiba:', error));

});

</script>

</body>

</html>

### ****login.php**** – A bejelentkezési adatok kezelése

Ez a fájl ellenőrzi az admin által megadott e-mailt és jelszót az adatbázisban, és elindítja a munkamenetet, ha sikeres a belépés.

<?php

require "../connect.php"; // Az adatbázis kapcsolódás

header('Content-Type: application/json; charset=utf-8');

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

// Az űrlapadatok JSON formátumban érkeznek, így beolvassuk és dekódoljuk

$input = json\_decode(file\_get\_contents('php://input'), true);

$email = mysqli\_real\_escape\_string($dbconn, trim($input['email']));

$jelszo = mysqli\_real\_escape\_string($dbconn, trim($input['jelszo']));

// Ellenőrizze, hogy az e-mail cím és jelszó nem üres

if (empty($email) || empty($jelszo)) {

http\_response\_code(400); // 400 Bad Request

echo json\_encode(['error' => 'Az e-mail cím és jelszó megadása kötelező!']);

exit();

}

// Az admin felhasználó ellenőrzése az adatbázisban

$sql = "SELECT \* FROM felhasznalok WHERE email = ? LIMIT 1";

$stmt = $dbconn->prepare($sql);

$stmt->bind\_param('s', $email);

$stmt->execute();

$result = $stmt->get\_result();

if ($result->num\_rows === 1) {

$admin = $result->fetch\_assoc();

// Ellenőrzés: jelszó megfelel-e a titkosított verziónak

if (password\_verify($jelszo, $admin['jelszo'])) {

// Indítsunk munkamenetet és tároljuk az admin státuszát

session\_start();

$\_SESSION['admin'] = true;

$\_SESSION['email'] = $admin['email'];

http\_response\_code(200); // Sikeres

echo json\_encode(['success' => true]);

} else {

http\_response\_code(401); // 401 Unauthorized

echo json\_encode(['error' => 'Helytelen jelszó!']);

}

} else {

http\_response\_code(404); // 404 Not Found

echo json\_encode(['error' => 'Az e-mail cím nem található!']);

}

}

### ****dashboard.php**** – Admin oldal, amely védelem alatt áll

Ez az oldal csak akkor érhető el, ha a felhasználó be van jelentkezve. Ellenőrzi, hogy a munkamenetben tárolt-e admin jogokat.

<?php

session\_start();

// Ellenőrizze, hogy a felhasználó be van-e jelentkezve admin státusszal

if (!isset($\_SESSION['admin'])) {

header("Location: ../login.html");

exit();

}

?>

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<title>Admin Felület</title>

</head>

<body>

<h1>Üdvözlünk az admin felületen!</h1>

<p><a href="logout.php">Kilépés</a></p>

</body>

</html>

### ****logout.php**** – Kijelentkezés

Ez a fájl kijelentkezteti az admin felhasználót, és visszairányítja a bejelentkezési oldalra.

<?php

session\_start();

session\_unset(); // Munkamenet adatainak törlése

session\_destroy(); // Munkamenet lezárása

header("Location: ../login.html"); // Visszairányít a bejelentkezési oldalra

exit();

#### Fájlstruktúra

nevjegyek/

│

├── connect.php // Adatbáziskapcsolat

├── login.html // Admin bejelentkezési űrlap

├── login.php // Bejelentkezés kezelése

├── admin/

│ ├── dashboard.php // Védett admin oldal

│ ├── logout.php // Kijelentkezés

│ └── ... // Egyéb admin funkciók

Ez az alapbeállítás egy egyszerű és biztonságos rendszer, amely RESTful módon működik az admin bejelentkezéséhez, védi az admin felületet, és biztosítja a kijelentkezési funkciót is.

Az **admin mappa** index.php fájlja opcionális, de érdemes létrehozni, különösen ha az admin felületről szeretnél közvetlen belépési pontot. Ha valaki beírja az admin mappa URL-jét (pl. http://localhost/nevjegyek/admin/), akkor ez a fájl töltődne be alapértelmezetten.

Ha szeretnél egy **belépési pontot** biztosítani, ahol az admin a belépése után az admin dashboard-ot látja, akkor ezt az oldalt létrehozhatod. Ez az oldal irányítaná az admin felhasználót az összes többi védett oldalra.

#### Példa az admin/index.php tartalmára:

<?php

session\_start();

// Ellenőrizze, hogy az admin be van-e jelentkezve

if (!isset($\_SESSION['admin'])) {

header("Location: ../login.html"); // Ha nincs bejelentkezve, irányítsa át a bejelentkezési oldalra

exit();

}

?>

<!DOCTYPE html>

<html lang="hu">

<head>

<meta charset="UTF-8">

<title>Admin Felület</title>

</head>

<body>

<h1>Üdvözlünk az Admin Felületen!</h1>

<p>Itt tudsz navigálni az adminisztrációs funkciók között.</p>

<ul>

<li><a href="dashboard.php">Admin Dashboard</a></li>

<li><a href="some-other-admin-page.php">Egyéb Admin Funkció</a></li>

<li><a href="logout.php">Kilépés</a></li>

</ul>

</body>

</html>

#### Miért hasznos lehet az index.php az admin mappában?

1. **Navigációs központ**: Ez lehet a központi oldal, ahol az admin felhasználó az összes elérhető admin funkciót könnyen megtalálhatja.
2. **Védelem**: Ha valaki az admin mappa URL-jét írja be közvetlenül, és nincs bejelentkezve, akkor átirányíthatod a bejelentkezési oldalra.
3. **Alapértelmezett oldal**: Amikor az admin mappába navigál, automatikusan betöltődhet ez az oldal, így az admin felhasználó számára barátságosabb kezdőpont lehet.

Az index.php tehát hasznos lehet, ha van több adminisztrációs funkció, és egy központi oldalról akarod ezeket elérhetővé tenni. Ha nincs erre szükség, akkor elhagyható.

# Források:

**PHP Alapok és Procedurális Programozás**

1. <https://www.php.net/manual/en/>
2. <https://www.w3schools.com/php/>
3. <https://phptherightway.com/>
4. <https://www.tutorialspoint.com/php/index.htm>
5. <https://www.codecademy.com/learn/learn-php>

#### ****OOP PHP (Objektumorientált Programozás)****

1. <https://www.php.net/manual/en/language.oop5.php>
2. <https://www.tutorialspoint.com/php/php_object_oriented_programming.htm>
3. <https://www.w3schools.com/php/php_objects.asp>
4. <https://phptherightway.com/#object_oriented_programming>
5. <https://www.codecademy.com/learn/learn-php/modules/learn-php-classes>