Belépő a tudás közösségébe

Informatika szakköri segédanyag

Dinamikus weboldalak előállítása szerveroldali technológiákkal

Horváth Győző, Visnovitz Márton

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2017-ben.

Eötvös Loránd Tudományegyetem
Informatikai Kar

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

DINAMIKUS WEBOLDALAK ELŐÁLLÍTÁSA SZERVEROLDALI TECHNOLÓGIÁKKAL

Horváth Győző, Visnovitz Márton

Belépő a tudás közösségébe

Informatika szakköri segédanyag

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2017-ben.

ISBN 978-963-284-998-0

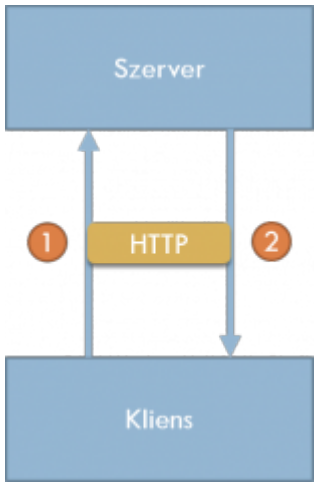
TARTALOMJEGYZÉK

1 BEVEZETÉS

1.1 Dinamikus weboldalak

Manapság számítógépes tevékenységeink tekintélyes részét a böngészőprogram használata jelenti. Információkat keresünk, híreket olvasunk, videókat nézünk, kapcsolatot teremtünk az ismerőseinkkel, egyre több hivatalos ügyet el tudunk intézni online, egyszerűbb játékokat is játszhatunk. Mindezeket valamilyen webes alkalmazás segítségével tudjuk megtenni, így joggal mondhatjuk, hogy a böngészők a webes technológiákkal együtt modern alkalmazásfejlesztési platformmá nőtték ki magukat.

A webes alkalmazások – a web jellegéből fakadóan – *kliens-szerver architektúrában* működnek. A szerver közzéteszi az elérhető erőforrásokat (HTML dokumentumok, képek, stb.), ezeket pedig klienssel, azaz böngészővel kérhetjük el a szervertől. *Dinamikus weboldalakról* akkor beszélünk, ha a megjelenített dokumentum előállításához, működtetéséhez, módosításához valamilyen számítógépes programot használunk. Ez a program futhat szerveroldalon, ekkor a böngészőnek leküldendő tartalmat dinamikusan állítjuk elő ezzel a programmal; vagy futhat kliensoldalon, ekkor a böngészőbe már betöltött HTML oldal dinamikus működtetése a cél. Egy összetettebb webalkalmazásban mindkét oldalon használhatunk programot.



Kliens-szerver architektúra

1.2 A tananyag célja

Ebben a tananyagban a szerveroldali webfejlesztésre fókuszálunk. Azt fogjuk megnézni, hogy hogyan használható a webszerver dinamikus, azaz bizonyos körülményektől (mint pl. felhasználói bemenettől vagy külső adatforrástól) függően változó tartalmú weboldalak előállítására. Az alkalmazás felhasználói felületét a böngésző jeleníti meg, szerkezetéhez HTML, megjelenéséhez CSS technológiát fogunk használni. A böngészőben megjelenő weboldal azonban nem lesz előre adott (mint a statikus oldalak esetén), hanem *előállítását egy program fogja szerveroldalon elvégezni*, ehhez pedig a PHP programozási nyelvet választottuk.

Szerveroldali alkalmazásokra elsősorban akkor van szükség, amikor több kliens között egy közös erőforrás (pl. adat, logika) megosztására van szükség, vagy ha nem szeretnénk érzékeny információt a kliensre juttatni. Tipikusan adatok központi tárolásakor vagy üzleti jellegű számítások elvégzésekor használják, amelyek igen gyakori feladatok.

A szerveroldali alkalmazásokat széles körben használják az egyszerű portáloktól kezdve a bank informatikáig. Az ebben a tananyagban megismert technológiát használják a legelterjedtebb tartalomkezelő rendszerek a Wordpress és a Drupal, de a Facebook webes alkalmazását is eredetileg PHP-ban írták.

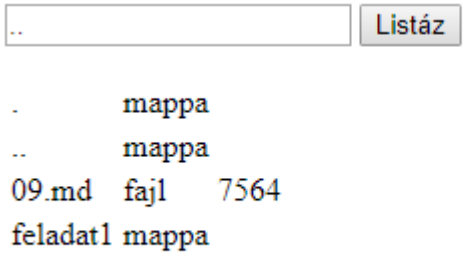
A tananyag elsajátításával a diákok képesek lesznek olyan dinamikus oldalak elkészítésére, amelyek:

- űrlapadatokat központilag dolgoznak fel;
- központi adattárolással nyilvántartási rendszert kezelnek;
- képesek a felhasználók azonosítására, felhasználóra szabott egyéni megjelenésre;

Űrlapkezelés és adatfeldolgozás

Kosár megvalósítása, adatbázisból érkező adatok

Regisztráció, bejelentkezés



Fájl- és könyvtárkezelés

1.3 A tananyag felépítése

A tananyag több, egymásra épülő fejezetet tartalmaz. Minden fejezet elején röviden ismertetjük az adott témakör *elméleti tudnivaló*t, majd azok használatát *több, kisebb feladat*on keresztül mutatjuk be. A tananyagot végigkísérik *nagyobb feladat*ok is, ezeket minden témakör végén adjuk meg. A fejezetek végén további *gyakorló feladatok* kapnak helyet.

A tananyagban a következő jelöléseket használjuk:

> **Példa**
>
> A tananyaghoz tartozó gyakorlati feladatokat, szemléltető szöveg- és kódrészleteket tartalmazó blokkba.

> **Ismétlés**
>
> A feltételezett ismeretek gyors áttekintésére szolgáló blokk.

Apróbetűs

A törzsanyagon túlmutató, további részleteket, kapcsolódó érdekességeket bemutató vagy továbblépési lehetőségeket felvillantó ismereteket ilyen blokkban közöljük.

Feladat

A tananyagot végigkísérő nagyobb feladathoz tartozó részfeladatokat tartalmazzák ezek a blokkok.

Letöltés

A letölthető anyagok ilyen blokkokban jelennek meg.

1.4 Szükséges előismeretek

Habár a tananyag kezdőknek szól, bizonyos mértékben épít korábbi tapasztalatokra, ismeretekre. A feltételezett előismeretek az alábbiak:

- HTML jelölőnyelv ismerete,
- opcionálisan CSS szelektorok és tulajdonságok alapfokú ismerete,
- alapvető algoritmizálási és programozási ismeretek (programozási tételek).

2 SZERVEROLDALI DINAMIKUS WEBPROGRAMOZÁS ÉS ESZKÖZEI

2.1 A világháló és technológiái

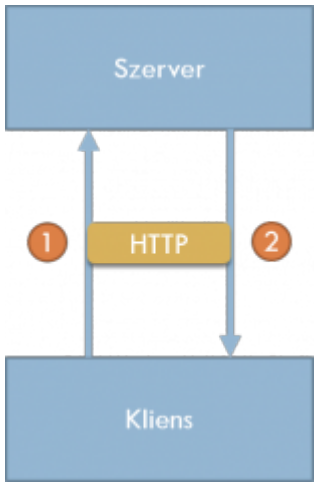
A *világháló* (angolul World Wide Web, röviden web) egy olyan információs rendszer, amelyben dokumentumokat és más erőforrásokat egységes címmel azonosítunk, ezeket hiperhivatkozásokkal kötjük össze, elérhetőségüket pedig internetre kötött szerverek segítségével biztosítjuk. A web több komponensből épül fel, működését számos szabvány, protokoll és technológia biztosítja:

- A weben keresztül elérhető *dokumentumok* (weboldalak) leírása *HTML nyelv* ↪ segítségével történik. Ezek további erőforrásokra (dokumentumok, képek, stb) való *hiperhivatkozások*at tartalmazhatnak.
- Az erőforrásokat *egységes cím*mel azonosítjuk (**URL** ↪), amely tartalmazza, hogy milyen protokollon melyik szerveren hol is van az adott erőforrás (`http://pelda.szerver.hu/ut/az/eroforrashoz.html`)
- Az erőforrásokat *webszerver*ek teszik elérhetővé más gépek számára az interneten keresztül.
- A webdokumentumokat webkliensek, *böngészők* kérik le a szervertől, és jelenítik meg azokat.
- A kliens és szerver közötti kommunikációt a *HTTP protokoll* biztosítja, ez írja le, hogyan kell egy böngészőnek a kérést, a szervernek pedig a választ elküldenie.

Ezek az elemek szükségesek a web használatához. Ezeken kívül azonban "webesnek" hívunk minden olyan technológiát, amely a fenti elemek bármelyikéhez kapcsolódik. Általában a HTTP fölött zajló kommunikációval vagy a HTML-lel leírt dokumentumokkal kapcsolatos technológiák webesnek számítanak. (Ld. még a **World Wide Web Consortium szabványait** ↪.)

2.2 Kliens-szerver architektúra és a dinamikus webprogramozás

A webes dokumentumok kiszolgálása kliens-szerver architektúrában történik. A böngésző kliensként egy HTTP kérést küld a szervernek. A szerver a kérésben foglalt információk alapján összeállítja a HTTP választ, és visszaküldi a böngészőnek. A böngésző a választ feldolgozza, ami általában a válaszban kapott dokumentum megjelenítését jelenti.



Kliens-szerver architektúra

*Statikus weboldalak*ról akkor beszélünk, ha az a tartalom, amit meg szeretnénk jeleníteni már a kérés pillanatában készen áll a szerveren, és a betöltődése után sem változik meg a szerkezete. Ilyenkor a szerver szempontjából is statikus az oldal, hiszen a kikeresett fájlt változatlan formában küldi vissza, és a kliens is statikus, hiszen a megjelenítés után a böngésző nem módosítja az oldal tartalmát.

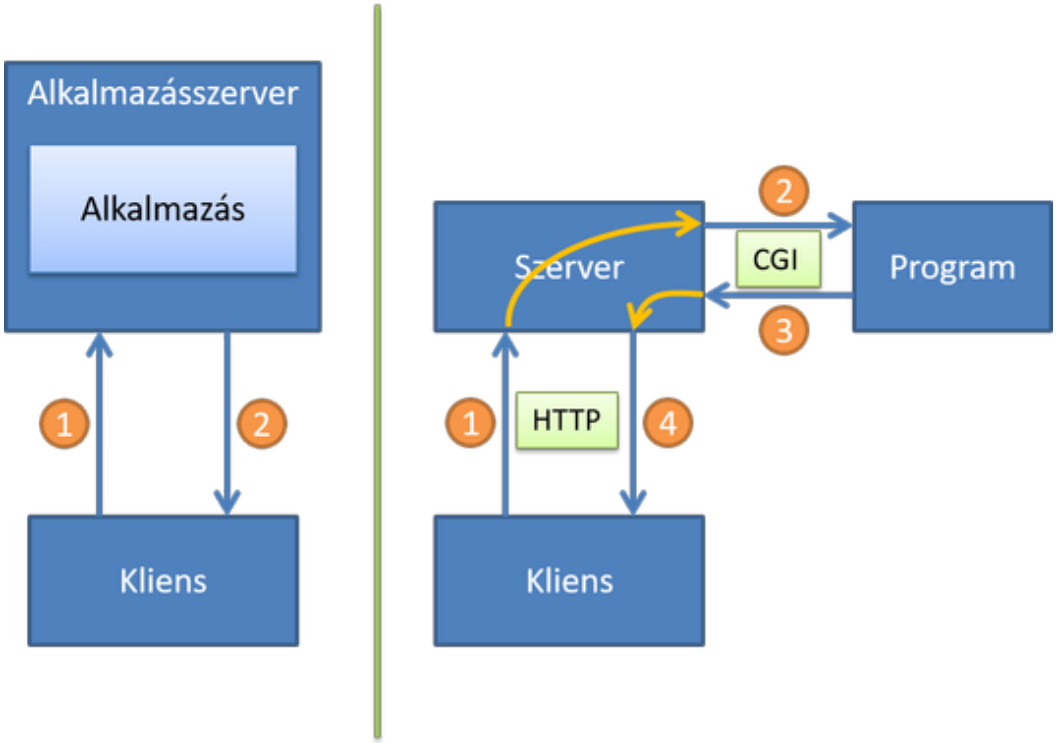
Dinamikus weboldalakról akkor beszélünk, ha a megjelenített dokumentum előállításához, működtetéséhez, módosításához programot használunk. Mivel az architektúránkban két komponens van, ezért a dinamikusságot mindkét komponens szemszögéből vizsgálhatjuk. *Szerveroldali dinamikus kiszolgálásról* akkor beszélhetünk, ha szerveroldalon a HTML válasz egy program futásának eredményeképpen születik meg. *Kliensoldali dinamizmus* esetén a böngészőben futó program változtatja a megjelenített oldal állapotát.

2.3 Szerveroldali webprogramozás

Ez a tananyag *dinamikus szerveroldali weboldalak programozásáról* szól. Azt mutatja meg, hogy hogyan lehet a böngésző kérésére *egy program segítségével HTML oldalakat előállítani*. Szerveroldalon nagyon sok programozási nyelv közül választhatunk (Python, C#, Java, JavaScript), de ebben a tananyagban a PHP nyelvet használjuk, mert ezzel a nyelvvel lehet a legkevesebb beállítással és plusz koncepcióval dinamikus oldalakat generálni.

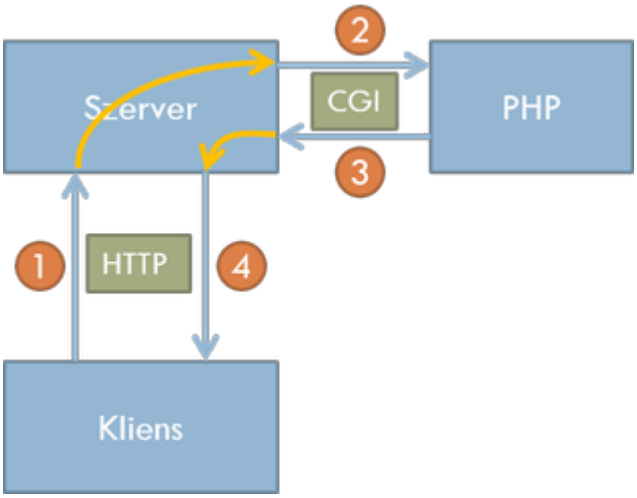
A HTTP kérésre választ adó programok alapvetően kétféle architektúrában működnek. Az egyik lehetőség, hogy egy meglévő webszerver funkcionalitását bővítjük úgy, hogy bizonyos kiterjesztésű kérések esetén a kikeresett forrásállományt először lefuttatja, és annak eredményét adja vissza a böngészőnek. A szkriptnyelvek – mint például a PHP, Python, Ruby, Perl – általában ezt a megközelítést használják. A webszerver és a program közötti kommunikációt az ún. Common Gateway Interface (CGI) protokoll vezérli. Ez írja le, hogyan kell a programot elindítani és milyen módon történik az adatcsere a szerver és a program között (standard bemenet és kimenet, valamint környezeti változók).

A másik lehetőség, hogy a programok nemcsak a HTML oldal generálására szolgálnak, hanem a HTTP kapcsolat figyelését és feldolgozását is ők maguk végzik. Ezt a koncepciót alkalmazásszervernek hívják, és például a Java, C#, JavaScript környezetek alkalmazzák.



Webszerver architektúrák

A PHP-t tipikusan Apache webszerver moduljaként szokták telepíteni. Ebben az esetben az Apache webszerver kezeli a HTTP kommunikációt. Ha egy `.php` kiterjesztésű állomány kiszolgálására érkezik kérés, akkor a szerver kikeresi a fájlrendszerből az állományt, majd ahelyett, hogy azonnal leküldené a forráskódot, átadja a PHP értelmezőnek, ami lefuttatja a programot. A program az eredményét a szabványos kimenetére írja, amit figyel a webszerver, és az oda érkező tartalmat pedig HTTP válaszként továbbítja a böngésző felé.



Az Apache webszerver működésének sematikus ábrája

2.4 A böngészők

A webes világban a böngészők szolgálnak a különböző webes erőforrások megjelenítésére, futtatására. Az erőforrások lehetnek HTML oldalak, képek, stílusállományok, JavaScript programfájlok. A böngésző a HTML oldalakat, képeket megjeleníti, a JavaScript kódot futtatja.

Sokféle böngészőprogram közül lehet választani, ezek közül néhány elterjedtebb:

- Google Chrome ↪
- Mozilla Firefox ↪
- Microsoft Edge ↪
- Opera ↪
- Safari ↪

A böngészőprogramok általában a felhasználói felületükben és az általuk nyújtott szolgáltatásokban térnek el egymástól. Fejlesztés szempontjából az a lényeges, hogy a HTML és CSS állományokat helyesen jelenítsék meg, a JavaScript kódot egységesen futtassák. Szerencsére manapság a böngészők között e tekintetben nincsenek nagy eltérések, ezért bármelyik választható.

2.5 Fejlesztői eszközök a böngészőkben

Szerveroldali fejlesztéshez csupán pár eszközre lehet szükségünk a böngészőkben. Az egyik az oldal forrásának megtekintése, amelyet általában a jobb egér gomb megnyomásával előugró menüben lévő menüpont vagy a `Ctrl-U` billentyűkombinációval tudunk előhívni. Mivel a szerveroldali programozás a megfelelő HTML szerkezet generálásáról szól, ezért legjobban a generált forráskód vizsgálatával tudunk meggyőződni programunk helyes működéséről.

A másik hasznos eszköz a böngészőprogram hálózati forgalmának figyelése. Ez a böngészőkbe épített webfejlesztési eszköztár része, és a legtöbb böngészőprogramban az `F12` billentyű lenyomásával érhetjük el, de a menüben mindig található rá hivatkozás (pl. Google Chrome esetén *További eszközök/Fejlesztői eszközök* menüpont). A megfelelő fülön nyomon követhető, hogy a böngésző milyen HTTP kéréseket küld, és azokra milyen válaszokat kap a szervertől.

2.6 Szerveralkalmazások

A szerveroldalon is számos alkalmazásra lesz szükségünk:

- Webszerver (Apache/nginx/IIS): a HTTP kérések kiszolgálására szolgál.
- Adatbázis-kezelő rendszer (SQLite/MySQL/PostgreSQL): az adatok tárolására szolgál.
- Programozási nyelv (PHP): a HTML oldal előállítását végzi.

2.7 Szerkesztők

Az alkalmazásfejlesztéshez megfelelő szerkesztőprogramra is szükség van. A webes dokumentumok forráskódjai egyszerű szövegfájlok. Olyan szerkesztő kell, amelyik képes HTML, CSS, PHP kódot kezelni, és kényelmes, fejlesztőbarát funkciókat nyújt, mint például kódszínezés, kódkiegészítés, automatikus behúzások, projekt kezelése, nyomkövetés, stb. Kétféle lehetőség közül választhatunk: vannak a kisebb méretű, de funkciókban gazdag kódszerkesztők, és vannak az ún. integrált fejlesztőkörnyezetek, amelyek általában nagyobbak, lassabbak, de rengeteg funkcióval rendelkeznek. Mindenki a maga preferenciái szerint választja ki az általa használt eszközt. A webes fejlesztők között az alábbi szerkesztőprogramok a legelterjedtebbek:

- **Microsoft Visual Studio Code** ↪ (ingyenes kódszerkesztő)
- **Atom** ↪ (ingyenes kódszerkesztő)
- **PHPStorm** ↪ (oktatási célra ingyenes IDE)

Ezeken kívül az olyan általános szerkesztőprogramok is használhatók, mint pl. a **Notepad++** ↪.

2.8 Javasolt fejlesztői eszköztár

2.8.1 Lokális rendszer kialakítása

Egy egy gépen futó fejlesztői környezet kialakításához az alábbi szoftvereket javasoljuk:

- Szerveralkalmazások (Apache, PHP, MySQL)
  - **XAMPP** ↪ (Windows)
  - **WAMP** ↪ (Windows)
  - **LAMP** ↪ (Linux)
  - **MAMP** ↪ (MacOS)
- Szerkesztők
  - **Microsoft Visual Studio Code** ↪
  - **Atom** ↪
- Böngészők
  - Google Chrome
  - Mozilla Firefox

A szerverkörnyezetet telepítve el kell indítani a webszervert (és az adatbázis-kezelő rendszert). Ehhez általában valamilyen vezérlőpulton van lehetőség. A szerverek tipikusan a 80-as porton figyelnek a beérkező kérésekre.

A forrásállományokat a feltelepített webszerver valamelyik alkönyvtárába kell elhelyezni. Ezeket helyben a szerkesztőprogramok segítségével lehet szerkeszteni. Megtekintéshez nem a forrásfájlt kell a böngészőben megnyitni, hanem a webszerveren keresztül a megfelelő URL-t (pl. `localhost/pelda/alma.php`).

2.8.2 Központi szerver használata

Ha a szerveralkalmazások egy központi szerveren kerülnek kialakításra, akkor a következő szoftverkomponensekre lesz szükség:

- Szerveralkalmazások (Apache, PHP, MySQL): a szerveren telepítendő és beállítandó
- Szerkesztők
  - **Microsoft Visual Studio Code** ↪
  - **Atom** ↪
- Böngészők
  - Google Chrome

        - Mozilla Firefox
    - FTP vagy SFTP/SCP kliens
        - **WinSCP** ↪ (Windows)
        - **FileZilla** ↪ (Linux)
        - **Cyberduck** ↪ (MacOS)

A fejlesztés egyik lehetséges folyamata a következő:

- Az SCP klienssel fellépünk a szerverre.
- Kikeressük a szerkesztendő állományt a webes dokumentumok közül.
- Szerkesztésre megnyitjuk az állományt lokálisan. Ekkor a helyi szerkesztőben megnyílik az állomány.
- Mentéskor az állomány (beállítástól függően) automatikusan feltöltésre kerül a szerverre.
- A böngészőben megnyitott oldalon lehet a változásokat nyomon követni (pl. `http://szerverem.hu/pelda/alma.php`).

2.9 Szerver beállítási segédlet tanároknak

- **LAMP – Apache, PHP, MySQL és PHPMyAdmin telepítése Linuxra** ↪

3 A PHP PROGRAMOZÁSI NYELV

> **Feladat**
>
> Célunk, hogy a következő pár fejezetekben megoldjuk nagyobb méretű feladatokat is. Ezeket a feladatokat önálló munkára szánjuk és ehhez hasonló blokkokkal jelöljük.

A szerver oldali programozásban számos nyelv elterjedt. A kiszolgálókon gyakran használják a többek között a *PHP*, *Java*, *Python*, *Ruby*, *JavaScript* nyelveket. Tananyagunkban a *PHP* nyelv segítségével mutatjuk be a legfontosabb szerveroldali technológiákat és alkalmazásfejlesztési módszereket, alapelveket.

3.1 A PHP nyelvről általában

A PHP egy úgynevezett általános célú interpretált szkriptnyelv, ami annyit tesz, hogy a programkód egy futtatókörnyezetben (például egy paranccssori értelmező) fut közvetlenül, fordítás nélkül. A PHP szintaxisában a "C-stílusú" nyelvekhez tartozik, így a vezérlési szerkezetek, nyelvi elemek nagyon hasonlóak a C, C++, Java és C# nyelvek azonos elemeihez.

```
// JavaScript
$x = 1;
for ($i = 2; $i < 10; $i++) {
  $x = $x * $i;
}
```

```
// C++
int x = 1;
for (int i = 2; i < 10; ++i) {
  x = x * i;
}
```

Első ránézésre is szembetűnő, hogy a PHP nyelvben a változók azonosítóját `$` jel előzi meg. Másik érdekesség, hogy PHP-ban nem szükséges a változókat külön deklarálni, azok az első értékadásnál jönnek létre. Amennyiben olyan változót akarnánk használni, aminek még korábban nem adtunk értéket, azt a futtató környezet hibaüzenettel jelzi. A C-hez, C++-hoz hasonlóan az utasításokat `;` zárja. Megjegyzéseket szintén az ezekből a nyelvekből ismerős `//` vagy `/* */` módon írhatunk a kódba, de használhatjuk a Perl nyelvben használt `#` jelet is kommentek készítéséhez.

A nyelv további jellemzői:

- **Részben case-sensitive**, vagyis a változók, konstansok, tömbindexek érzékenyek, a függvény- és osztálynevek, és kulcsszavak nem érzékenyek a kis- és nagybetűk közötti különbségekre.
- **Dinamikusan típusos**, vagyis egy változó típusa mindig az éppen aktuális értéktől függ.
- **Többparadigmás**, vagyis támogat számos programozási stílust, mint például az procedurális, objektum-orientált és funkcionális programozást.

A nyelvhez tartozó **dokumentáció** ↪ a weben elérhető számos nyelven (angol, német, francia, stb).

3.2 Típusok

Mint a legtöbb programozási nyelvben, a PHP-ben is definiálva vannak bizonyos alapvető típusok. Habár a változóink nem, csak az egyes értékek rendelkeznek típussal, mégis fontos típusokról beszélni. A nyelvben elérhető típusok feloszthatóak *egyszerű*, *összetett* és *speciális* típusokra:

3.2.1 Egyszerű típusok:

- **egész** (`integer`): Tetszőleges egész számérték. Megadható többféle számrendszerben, például 10-es (`1234`), 8-as (`02322`) vagy 16-os (`0x4D2`) számrendszerben.
- **tizedestört** (`float`): Tetszőleges tizedes tört, lebegő pontos számábrázolással. Többféle formátumban megadható, például tizedestörtként (`112.7`) vagy normálalakban (`1.127e2`). Más néven `double`-ként is ismert. Fontos figyelni arra, hogy a tizedestörtek nem ábrázolhatóak tetszőleges pontossággal a számítógépen, így kerekítési hibák előfordulhatnak.
- **szöveg** (`string`): Tetszőleges szöveg érték, melyet idézőjel (`"`), aposztróf (`'`) jelöl. Az idézőjellel jelölt szövegekben van lehetőség változó behelyettesítésre (`"Helló, ${nev}"`), illetve a speciális, úgynevezett "escape karakterek", mint például az új sor karakter (`\n`) is kiértékelésre kerülnek. Külön karakter típus nem létezik, a karakterek 1 hosszúságú szövegek a nyelvben.

  > **Apróbetűs**
  >
  > A szövegek megadhatóak további módokon is, ezek az úgynevezett **"heredoc"** ↪ és **"nowdoc"** ↪ szintaxis.

- **logikai** (`boolean`): Igaz (`TRUE`) és hamis (`FALSE`) értékek. Logikai értékek között az ÉS (`&&`, `and`) illetve VAGY (`||`, `or`) műveletek definiáltak.

3.2.2 Összetett típusok:

- **tömb** (`array`): Nem klasszikus értlemben vett tömb, hanem olyan összetett adatszerkezet, mely rendezett kulcs-érték párok halmaza. A kulcsok (indexek) lehetnek egész számok vagy szövegek. A típus használható klasszikus tömbként, listaként, sorként,

veremként, vagy a Pyhtonból ismert `Dictionary`, vagy a JavaScriptből ismert `object` adatszekezetként (ezt a változatot asszociatív tömbnek nevezzük). A tömbben tárolt értékek tetszőleges, akár különböző típusúak is lehetnek. Létrehozni az elemek szögletes zárójelben (`[]`) történő felsorolásával, vagy az `array` utasítással lehet:

Példa

```
// tömb változatos típusú elemekkel
$tomb = [1, "alma", TRUE, []];
// tömb létrehozás a régi szintaxissal
$tomb = array(1, "alma", TRUE, array());
```

Az elemek elérése szintén `[]` és a 0 kezdőelemű index megadásával lehetséges.

Példa

```
$tomb[2] == "alma"; // TRUE
```

Tömbökbe új elemet úgy is fel tudunk venni, hogy a végére szúrunk be. Ehhez a `[]` operátort tudjuk használni:

```
$tomb[] = "új elem";
```

Amennyiben nevesíteni szeretnénk (egyedi kulcssal ellátni) a tömb elemeit, akkor a kulcsokat a `=>` jellel kell elválasztani az értéktől. Kulcsoknak szám vagy szöveg típusú értékeket használhatunk.

Példa

```
$asszociativ_tomb = [
  10 => "alma",
  "korte" => "narancs",
  "20" => "banán"
];
```

- **objektum** (`Object`): Az **objektumorientált paradigma** ↪ alapvető típusa, valamely osztály példánya.

- **meghívható** (`callable`): Egy olyan értéknek a típusa, ami függvényként futtatható kódot tartalmaz. Olyan függvények paramétere, amelyek maguk is valamilyen futtatható kódot várnak paraméterül. Létrehozni a függvényekhez hasnolóan a `function` kulcsszóval lehet annyi különbséggel, hogy nem adunk nevet a létrehozott függvénynek.

  Példa

  ```
  $meghivhato = function () { /* ... */ };
  ```

  Minden névvel létrehozott függvény is áthadható ilyen módon paraméterként, ilyenkor a függvény nevét szövegként kell átadni.

3.2.3 Speciális típusok

- **erőforrás** (`resource`): Speciális típus, ami valamilyen külső erőforrást (pl. fájl) jelképez. Ezeket az erőforrásokat speciális függvények használják.
- **null**: Speciális típus, melybe egy darab speciális érték, a `NULL` tartozik. Ez az típus valamilyen érték hiányát jelzi, ami akkor fordulhat elő, ha *még nem állítottunk be értéket*, *kézzel töröltünk egy értéket az* `unset` *művelettel*, vagy kézzel beállítottunk egy értéket `NULL`-ra.

3.3 Vezérlési szerkezetek

A PHP nyelv vezérlési szerkezetei szinte pontosan megegyeznek a más, hasonlóan C szintaxisú nyelvek azonos elemeivel:

```
for (/* kezdőérték */; /* feltétel */; /* ciklusléptés */) {
  // utasítás
}

while (/* feltétel */) {
  // utasítás
}

do {
  // utasítás
} while (/* feltétel */);

if (/* feltétel */) {
  // utasítás
} else {
  // utasítás
}
```

```
switch (/* változó */) {
  case /* érték */:
    // utasítás
    break;
  default:
    // utasítás
}
```

Ezeken kívül létezik még a `foreach` ciklus egy változata, mely egy tömb (precízebben tetszőleges felsorolható adatszerkezet) elemein halad végig egyesével.

```
foreach ($tomb as $elem) {
  // utasítás
}
```

Amennyiben szükségünk van a felsorolásnál az adott elem indexére is, akkor ez a `foreach` egy alternatív változatával elkérhető:

```
foreach ($tomb as $index => $elem) {
  // utasítás
}
```

3.4 Függvények

PHP-ban, mint szinte minden másik programozási nyelvben lehetőségünk van saját függvényeket létrehozni. A függvények létrehozása a `function` kulcsszóval történik.

```
// függvény megadása a `function` kucsszóval
function fuggveny($param1, $param2) {
  // utasítások
  return /* visszatérési érték */;
}

// például
function osszead($x, $y) {
  return $x + $y;
}

// használata
osszead(10, 32); // --> 42
```

A fenti példában, a függvények paramétereit nem láttuk el típussal, illetve a visszatérési érték típusát sem határoztuk meg. A PHP 7.0-ás verziójától kezdve van lehetőségünk megadni a

függvényparaméterek és visszatérési értékek paramétereinek típusát is, ezzel biztonságosabbá téve a kódunkat.

Példa

```
function fuggveny(int $param1, string $param2) : string {
  /* ... */
}

// például
function osszead(int $x, int $y): int {
  return $x + $y;
}
```

Apróbetűs

Létezik a PHP-nak olyan változata, melyeben teljes mértékben jelen van a statikus típuskezelés, vagyis minden változónak, függvényparaméternek és visszatérési értéknek kötelezően definiált típusa van. Ezt a változatot **Hack** ↪-nek hívják. Hátránya a Hack nyelvnek, hogy saját, **speciális futtatókörnyezetre** ↪ van szüksége a működéshez.

3.5 A PHP kipróbálása

A PHP nyelv kipróbálásához szükséges valamilyen futtatókörnyezet. Ez lehet helyi gépre telepített PHP értelmező, webszerver vagy valamilyen online "sandbox" eszköz. Amennyiben a gépre vagy a szerverre telepítve van a PHP értelmező azt könnyedén elindíthatjuk interaktív módban a `php -a` utasítás kiadásával.

Az egyik legegyszerűbb utasítás az egyszerű kiírása, amit az `echo` utasítással tehetünk meg.

```
echo "Hello világ";
```

Mivel az interaktív parancssori értelmezőbe egyszerre csak egy utasítást írhatunk be, ezért ha hosszabb kódokkal szeretnénk kísérletezni, akkor már érdemesebb a kódunkat egy fájlba elhelyezni. A fájlnak a `.php` kiterjesztést kell adni. Ügyeljünk arra, hogy csak azok a részek kerülnek értelmezésre programkódként, amik a `<?php` és `?>` jelek közé kerülnek. Minden, ami ezeken a jeleken kívül található, az egyszerű szövegként kiírásra kerül. Ilyen "PHP blokkból" több is lehet egy fájlban. Ennek a megoldásnak az az oka, hogy a PHP képes sablonnyelvként is viselkedni, és ezzel a megoldással a statikus és dinamikus részek egymást válthatják egy oldalon belül.

Több olyan online eszköz is létezik, melyek segítségével azonnal futtathatjuk az általunk írt PHP kódot, melynek az eredménye is azonnal megjelenik. Ilyen eszköz például a **Repl.it** ↪ webalkalmazás.

3.6 Feladatok

1. Készíts függvényt, ami beolvas egy számot és eldönti, hogy az páros vagy páratlan! Használd a maradék (`%`) operátort!

```php
function paros_e($szam) {
  return ($szam % 2 == 0);
}
```

2. Egy tömbben adott számoknak a sorozata, adjuk meg az összegüket!

```php
$tomb = [1, 4, 2, 6, 1, 23];
$osszeg = 0;
foreach ($tomb as $szam) {
  $osszeg += $szam;
}
echo $szam;
```

3. Készíts függvényt `faktorialis` néven, ami egy ciklussal kiszámítja az `n` faktoriális értékét!

```php
function faktorialis($n) {
  $eredmeny = 1;
  for ($i = 2; $i <= $n; $i++) {
    $eredmeny *= $i;
  }
  return $eredmeny;
}
```

4. Készíts függvényt, ami egy tömb elemei közül megszámolja, hogy hány darab `x` érték található!

```php
function darab($tomb, $x) {
  $darab = 0;
  foreach ($tomb as $elem) {
    if ($elem === $x) {
      $darab++;
    }
  }
  return $darab;
}
```

3.7 Adatszerkezetek ábrázolása

A PHP nyelv tömbje, egy nagyon sokféleképpen használható adatszerkezet. Lehet használni egyszerű tömbként, de akár veremként, sorként de használhatjuk tetszőleges kulcs-érték párok tárolására is. Ezeknél komplexebb adatszerkezetet is felépíthetünk, ha tömböket egymásba ágyazunk, vagyis ha a tömbünk egyik eleme maga is egy tömb.

Példa

Készítsünk egy adatszerkezetet, ami egy iskolának és annak osztályainak adatait tartalmazza!

Kiindulásképp vegyünk egy asszociatív tömböt, melynek a mezői az iskola alapvető adatait tartalmazzák:

```php
[
  "nev" => "PHP Általános Iskola",
  "cim" => "1337 Világháló utca 404.",
  "om" => "528272478"
]
```

Ez az adatszerkezet tovább bővíthető, ha egy értéken belül egy tömbben eltároljuk, hogy milyen osztályok vannak az iskolában:

```php
[
  "nev" => "PHP Általános Iskola",
  "cim" => "1337 Világháló utca 404.",
  "om" => "528272478",
  "osztalyok" => [
    "1.a",
    "1.b",
    "2.a",
    "2.b"
```

```
    ]
]
```

Ebben a példában csak az osztályok nevét tároljuk el, de semmi egyebet nem tudunk az osztályról. Ha további információkat szeretnénk tárolni (pl. osztálylétszám, osztályfőnök), akkor megtehetjük, hogy minden egyes osztályt egy újabb asszociatív tömb reprezentál, és így tovább tetszőleges mélységig bővítve az adatszerkezetünket.

Apróbetűs

A való életben, a komolyabb programokban az összetartozó értékeket (pl. egy iskola, vagy egy iskolai osztály adatai) objektumokban tárolják, melyekhez osztályokat hoznak létre. Ezzel a megoldással a bonyolult adatszerkezetek leírása egyszerű tömbök és objektumok egymásba ágyazott halmazaivá válik.

3.8 Feladatok

1. Készíts egy olyan adatszerkezetet, amely egy könyv adatait írja le. A könyvnél a következő adatokat tároljuk:
    - szerző
    - cím
    - kiadás éve
    - kiadó
    - ISBN szám

```
$konyv = [
  "szerzo"     => "J.R.R Tolien",
  "cim"        => "A gyűrűk ura",
  "kiadas_eve" => 1954,
  "kiado"      => "Allen & Unwin",
  "isbn"       => "ISBN 0-618-34099-7"
];
```

2. **Feladat**

   Készíts olyan adatszerkezetet, melybe egy bevásárlólista információit tárolhatjuk. A listán többféle dolgot szeretnénk tárolni, és mindegyik elemről tudni szeretnénk, hogy mi az és mennyit szeretnénk belőle vásárolni, illetve, hogy ki a felelős a beszerzéséért.

4 ADATOK DINAMIKUS MEGJELENÍTÉSE

Egy szerveroldali program elsődleges feladata a HTTP kérésnek megfelelő HTML tartalom előállítása. A programban így a kimenet generálás kitüntetett szerepet kap. Másképpen megközelítve: a szerveroldali programokra is jellemző az az általános felépítés, amit például a konzolos programoknál láthattunk, vagyis ezekben a programokban is be kell olvasni az adatokat, fel kell dolgozni őket, majd megfelelő formában (HTML) meg kell őket jeleníteni. Ez a fejezet ez utóbbira, a kimenet-generálásra, azaz a HTML oldal előállítására összpontosít.

Az előző fejezetben láthattuk, hogyan használható a PHP nyelv adatszerkezetek definiálására és feldolgozására. Ebben a fejezetben az így előálló adatokat adottnak tekintjük, és azt nézzük meg, hogy a PHP nyelv segítségével miként tudjuk azokat HTML formában a kliensre küldeni.

Ebben a megközelítésben az az elv tükröződik, hogy az adatok definiálását és feldolgozását minél határozottabban elválasszuk azok megjelenítésétől. Ebből az is következik, hogy a kiírás során új adat definiálására vagy bármiféle feldolgozásra, számításra, háttértárhoz fordulásra már nem kerül sor, így a kiíráshoz használt nyelvi elemkészlet is megfelelően kicsi lehet.

Mivel a generálandó HTML-ben óhatatlanul lesznek dinamikusan előállítandó részek, ezeket a részeket valamilyen módon jelölni kell. A HTML kódból egy szerveroldali alkalmazásban sablon lesz, amelybe majd behelyettesítődnek a dinamikus tartalmak. A kiírás során tehát a PHP *sablonnyelv*ként fog szolgálni.

4.1 Kiírás PHP-val

A PHP kód a HTML kódba ágyazva jelenik meg. A HTML kódban `<?php` nyitó- és `?>` záróelemek közé kell helyezni a PHP kódot. Egy oldalon belül ilyen elempárokból akárhova és akárhány elhelyezhető. PHP oldalról megközelítve: egy PHP fájlon belül mindaz a szövegrészlet, ami nincsen `<?php ?>` elemek között, automatikusan kiírásra kerül. PHP kódon belül kiírni az `echo` utasítással lehet.

```
<p>Automatikusan kiíródó szöveg</p>
<?php
echo "<p>Programmal kiírt szöveg</p>";
?>
```

4.2 HTML generálása

A HTML generálásakor két szempontot tartunk szem előtt:

1. Csak a *dinamikus részek* előállításához van szükségünk programra. Statikus tartalmak kiírásához nem használunk PHP-t.

2. Jelezvén, hogy a kiírás során a PHP sablonnyelvként szolgál, a kimenet generálásakor a *PHP alternatív szintaxisát* fogjuk használni (ld. az alábbi példákat).

4.2.1 Statikus részek generálása

Statikus HTML generálásához nincsen szükség PHP-ra, minden automatikusan mehet a program kimenetére úgy, ahogy van:

```
<h1>Hello világ!</h1>
<p>Ebben a részben semmi dinamikus tartalom nincsen</p>
```

4.2.2 Változó értékének kiírása

Változó kiírásához a `<?= ?>` formát használjuk.

```
<h1>Hello <?= $name ?>!</h1>
```

4.2.3 Ciklus

Gyűjtemények kiírásához ciklust használunk (`foreach-endforeach`).

```
<ul>
  <?php foreach($lista as $elem) : ?>
    <li><?= $elem ?></li>
  <?php endforeach ?>
</ul>
```

4.2.4 Elágazás

Feltételes kiíráshoz az `if-else-endif` párost használjuk.

```
<div>
  <?php if ($bejelentkezve) :?
    <p>Hello <?= $nev ?>!</p>
    <a href="logout.php">Kijelentkezés</a>
  <?php else: ?>
    <a href="login.php">Bejelentkezés</a>
  <?php endif ?>
</div>
```

4.3 Általános elvek

PHP sablon létrehozásakor mindig induljunk ki a generálás végeredményeként előállítandó statikus HTML-ből. Abban azonosítsuk azokat az elemeket, amelyek a program futása során változhatnak, és cseréljük le őket a fenti szerkezetekre!

Példa

Adott egy hibákat tartalmazó tömb. Jelenítsük meg a hibalistát felsorolásként!

Első lépésként magát a HTML listát hozzuk létre:

```
<ul>
  <li>hiba1</li>
  <li>hiba2</li>
  <li>hiba3</li>
</ul>
```

Miután meggyőződtünk, hogy ez hibátlanul megjelenik, akkor azonosítjuk benne a változó tartalmat:

1. A listaelemek száma a hibatömb számosságától függ (–> ciklus)
2. A listaelemek tartalma az adott tömbelem értéke (–> kiírás)

```
<ul>
  <?php foreach($hibak as $hiba) : ?>
    <li><?= $hiba ?></li>
  <?php endforeach; ?>
</ul>
```

4.4 Feladatok

1. **Feladat**

   Adott egy szótár pár bejegyzése. Jelenítsük ezt meg definíciós listaként!

   ```
   $szotar = [
     "alma"       => ["apple"],
      "azonnal"     => ["immediately", "instantly", "at once",
   "now"],
     "segít"      => ["help", "assist", "aid"],
   ];
   ```

   Letöltés

   ↧ A feladat megoldása ↪

2. **Feladat**

   Adott tanulók adatai (név, tanulmányi azonosító, osztály, cím). Írassuk ki a tanulók listáját táblázatos formában!

   Letöltés

   ↧ A feladat megoldása ↪

5 ADATBÁZIS HASZNÁLATA

Szerveroldali programra akkor van szükség, amikor valamilyen közös erőforrást szeretnénk az alkalmazás használói között megosztani. Az egyik legfontosabb közös erőforrás az adat. Az adatot egy helyen tároljuk, és ebből szolgáljuk ki az egyes klienseket.

> **Példa**
>
> - Egy webbolt az árukészletét egy központi szerveren tartja nyilván. A vásárlók a weboldalon keresztül a központi adatokat böngészhetik.
> - Bizonyos játékokat lehet csak a böngészőben játszani. Az elért pontszámokat el is menthetjük a böngészőben, de ekkor mindig csak a saját eredményeink jelennek meg. Ha szeretnénk tudásunkat összemérni másokéval, akkor szükséges egy olyan hely, ahol az eredményeket központilag tároljuk, így a toplistában a játékot használó összes játékos eredménye megjelenik. Ráadásul másik gépen használva a játékot, a lokálisan mentett pontszámok nem érhetők el.

5.1 Adatbázisok fajtái

Az adatot alapvetően kétféleképpen tudjuk tárolni:

- fájlban, vagy
- adatbázisban.

Adatok tárolását illetően az adatbázisok használatának számos előnye van:

- strukturált tárolás,
- típusos adatok,
- biztonságos,
- bonyolult adatszerkezetek,
- összetett kereshetőség,
- fejlett konkurenciakezelés.

Az adatbázisoknak is több fajtája van különböző szempontok szerint. Az adatok szerkezetét illetően megkülönböztetünk:

- *relációs adatbázisokat*: ezek a "klasszikus" adatbázisok, amelyekben az adatokat relációs adatmodellnek megfelelően egymással összekapcsolt táblákban kezeljük. Ilyen adatbázis például a MySQL, SQLite, PostgreSQL.
- *dokumentum-alapú adatbázisokat*: ezeknek központi fogalma a dokumentum, amely egy valamilyen formában (például JSON-ben) tárolt információ. A dokumentumok általában

gyűjteményeket alkotnak. Fő eltérése a relációs adatmodelltől az, hogy egy dokumentum szerkezete egy gyűjteményen belül nem kötött. Emellett az egyes gyűjtemények közötti kapcsolat leírása is korlátozott. Erre példa a MongoDB adatbázis-kezelő szoftver.

A tárolás módja szerint a következő kategóriák lehetnek:

- *adatbázisszerver*: egy külön kiszolgáló, mely helyileg futhat ugyanazon a szerveren, mint a webszerver, de akár egy külön interneten keresztül elérhető gépen. Így működik a MySQL és a PostgreSQL például.
- *fájl-alapú adatbázis*: az adatok tárolása lokálisan egy fájlban valósul meg. Jó példa erre az SQLite.

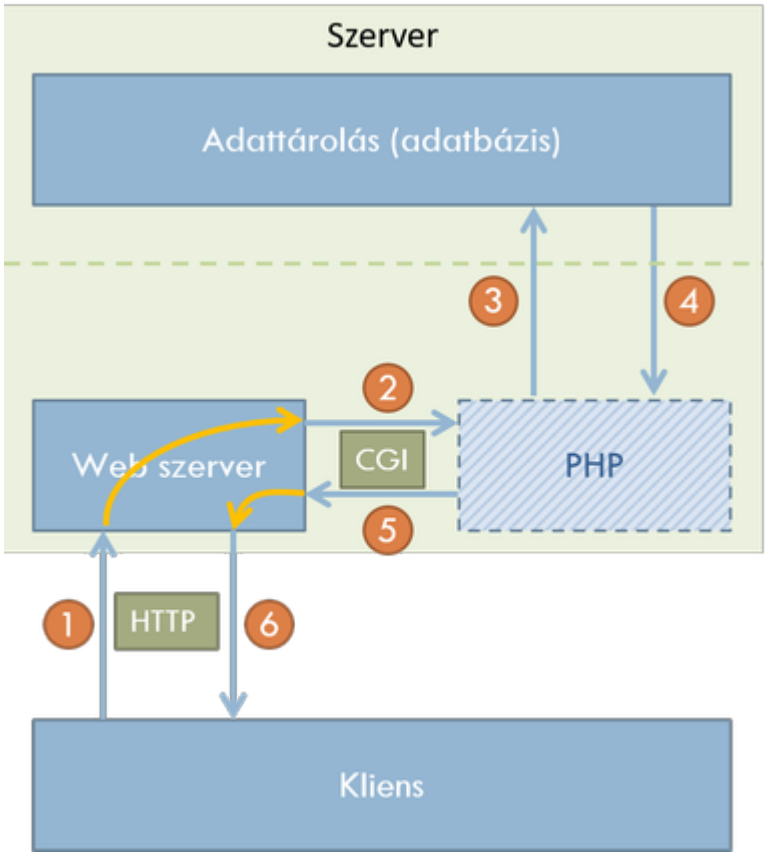
Ebben a tananyagban az egyszerűség kedvéért a szélesebb körben ismert relációs adatmodellel működő és külön szervert nem igénylő *SQLite adatbázisok*at fogjuk használni.

5.2 Program és adatbázis kapcsolata általában

Egy bizonyos típusú adatbázis használatához egy tetszőleges programozási nyelvben *adatbáziskezelő-specifikus függvények* állnak rendelkezésre. Külön függvénykönyvtár szükséges a MySQL, PostgreSQL, SQLite, stb kezelésére. Ha ezeket feltelepítettük, akkor a programban általában a következő lépések követik egymást:

- kapcsolódás az adatbázis-kezelő rendszerhez;
- az adatbázis kiválasztása;
- SQL utasítások futtatása az adatbázis-kezelő rendszerben;
- az SQL utasítás eredményének lekérdezése (ha van);
- a kapcsolat bontása.

5.3 Adatbázisok használata PHP-ban



Kliens, szerver és adatbázis kapcsolata

PHP-ban **számos adatbázis-kezelő rendszer** ↪ elérése támogatott a megfelelő függvénykönyvtárakon keresztül, amelyeket általában külön telepíteni szükséges. A PHP azonban biztosít egy olyan függvénykönyvtárat, amely egy egységes programozási interfészen keresztül teszi lehetővé a különböző adatbázis-kezelők használatát. Így nem kell megtanulnunk az egyes adatbáziskezelő-specifikus függvényeket, mi esetünkben az SQLite-hoz tartozókat, hanem csak ezt a szóban forgó interfészt. Ennek a függvénykönyvtárnak a neve **PHP Data Objects** ↪, azaz PDO.

A PDO ugyanolyan műveletcsoportokat biztosít, mint a többi függvénykönyvtár. Segítségével kapcsolódni lehet egy adatbázishoz, SQL utasításokat lehet kiadni, az eredményeket lekérdezni, stb. A PDO ehhez két osztályt definiál:

- `PDO`: az adatbázishoz való kapcsolódásért, és az SQL utasítások futtatásáért (`query`, `exec`, `prepare`) felel.
- `PDOStatement`: az adatok lekérdezéséért, illetve előkészített lekérdezések futtatásáért felel (ld. később).

Főbb utasításai:

- `$pdo = new PDO("adatbázis_kapcsolat")`: kapcsolódás az adatbázishoz. Ebben jelenik meg egyedül adatbázis-specifikus információ a paraméterként megadott kapcsolati paramétereket tartalmazó szövegben.
- `$pdo = null`: kapcsolat bontása

- `$stmt = $pdo->query("sql")`: lekérdezések (`SELECT`) futtatására szolgál, egy PDOStatement objektummal tér vissza.
- `$db = $pdo->exec("sql")`: nem lekérdezések esetén használandó, az érintett sorok számával tér vissza.
- `$stmt = $pdo->prepare("sql")`: paraméteres SQL utasítások előkészítésére szolgál.
- `$siker = $stmt->execute(paraméter_tömb)`: az előkészített SQL utasítás futtatása a paraméterek behelyettesítésével.
- `$adattomb = $stmt->fetchAll()`: lekérdezés eredményének eltárolása tömbben.

Hibakezelés a következőképpen történhet:

- a hibás utasítások *hamis értékkel* térnek vissza: ekkor mind a `PDO` és a `PDOStatement` osztályok a következő műveleteket biztosítják a hiba lekérdezésére:
  - `errorCode()`: szabványos (ANSI SQL-92) hibakód
  - `errorInfo()`: részletes hibaüzenet
- `PDOException` típusú hiba dobása: a hibás utasítások ekkor kivételt dobnak, amit `try-catch` blokkal kezelhetünk. Ezt a működési módot expliciten kell beállítanunk rögtön a kapcsolódás után a következő kódrészlettel:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

5.4 Alapvető SQL utasítások

Ismétlés

Az adatokkal általában tipikusan négyféle műveletet kell tudnunk elvégezni: lekérdezni, újat felvenni, módosítani és törölni. Az ezeknek megfelelő alapvető SQL utasításokat a következőkben tekintjük át.

Lekérdezés

```
SELECT <oszloplista>     -- oszlopok szűrése
FROM   <tábla>
WHERE  <feltételek>;     -- sorok szűrése

-- például
SELECT `nev`, `kor`
FROM   `dolgozok`
WHERE  `kor` > 60;
```

Beszúrás

```
INSERT INTO <tábla>
  (<oszloplista>)
VALUES
  (<értéklista>);

-- például
INSERT INTO `dolgozok`
  (`nev`, `kor`)
VALUES
  ('Buhera Vera', 55);
```

Módosítás

```
UPDATE <tábla>
SET <oszlop1> = <érték1>,
    <oszlop2> = <érték2>
WHERE <feltételek>;       -- sorok szűrése

-- például
UPDATE `dolgozok`
SET    `kor` = 56
WHERE  `id` = 231;
```

Törlés

```
DELETE FROM <tábla>
WHERE <feltételek>;       -- sorok szűrése

-- például
DELETE FROM `dolgozok`
WHERE  `kor` > 80;
```

5.5 Az adatbázis-használat tipikus esetei

5.5.1 Adatbázis előkészítése

Mielőtt PHP-ból elkezdenénk az adatbázist használni, készítsük elő! Hozzuk létre a táblákat, a kapcsolatokat, és ha szükséges, akkor töltsük fel példaadatokkal. Megtehetnénk ezt PHP segítségével is, de az egyszerűség kedvéért tegyük ezt meg külön a megfelelő adatbáziskliens használatával.

SQLite esetén használhatjuk például a **DB Browser for SQLite** ↪ alkalmazást. Ebben létrehozhatunk egy új adatbázist, majd elkészíthetjük a táblastruktúrát és feltölthetjük adatokkal.

Példa

Az alábbiakban egy zeneszámokat tároló adatbázist építünk. Két tábla lesz benne:

- albumok
- zeneszámok

Az adatbázist az alábbi SQL utasítással létre is hozhatjuk:

```
CREATE TABLE `albumok` (
  `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
  `egyuttes`  TEXT,
  `cim`   TEXT NOT NULL,
  `ev`    INTEGER NOT NULL
);
INSERT INTO `albumok` (`id`, `egyuttes`, `cim`, `ev`) VALUES
(1,'Guns n'' Roses','Appetite for destruction',1986);
INSERT INTO `albumok` (`id`, `egyuttes`, `cim`, `ev`) VALUES
(2,'Aerosmith','Get a grip',1993);

CREATE TABLE `zeneszamok` (
  `id`    INTEGER PRIMARY KEY AUTOINCREMENT,
  `szerzo`    TEXT,
  `cim`   TEXT NOT NULL,
  `hossz` INTEGER,
  `album_id`  INTEGER NOT NULL,
  FOREIGN KEY(`album_id`) REFERENCES `albumok`(`id`) ON UPDATE
RESTRICT ON DELETE RESTRICT
);
INSERT INTO `zeneszamok` (`id`, `szerzo`, `cim`, `hossz`,
`album_id`) VALUES (1, NULL, 'Cryin', 309, 2);
INSERT INTO `zeneszamok` (`id`, `szerzo`, `cim`, `hossz`,
`album_id`) VALUES (2, NULL, 'Crazy', 314, 2);
INSERT INTO `zeneszamok` (`id`, `szerzo`, `cim`, `hossz`,
`album_id`) VALUES (6, NULL, 'Paradise City', 406, 1);
INSERT INTO `zeneszamok` (`id`, `szerzo`, `cim`, `hossz`,
`album_id`) VALUES (7, NULL, 'Sweet Child o'' Mine', 355, 1);
```

5.5.2 Kapcsolódás

```
$pdo = new PDO("sqlite:./zene.sqlite");
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Segédfüggvényben, felkészülve azokra az adatbázisokra, ahol felhasználónevet és jelszót is meg kell adni:

```
function kapcsolodas($kapcsolati_szoveg, $felhasznalonev = '', $jelszo
= '') {
  $pdo = new PDO($kapcsolati_szoveg, $felhasznalonev, $jelszo);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  return $pdo;
}

// használata
$kapcsolat = kapcsolodas("sqlite:./zene.sqlite");
```

5.5.3 Egyszerű lekérdezések

Ezeknél a lekérdezéseknél nem jelenik meg paraméter.

```
$stmt = $kapcsolat->query("SELECT * FROM `albumok`");
$albumok = $stmt->fetchAll();
var_dump($albumok);
```

5.5.4 Paraméteres lekérdezések

Sokszor fordul elő, hogy az SQL utasításainkat paraméterekkel kell általánosítanunk. Ilyenkor biztonsági okokból soha ne szövegösszefűzéssel állítsuk elő az SQL utasítást, hanem használjunk előkészített SQL utasításokat és adatkötést.

Az *előkészített SQL utasítások*ban speciális jelölőkkel jelezzük a paraméterek helyét. Ezt az utasítást az adatbázis-kezelővel előkészíttetjük, amely során az előzetesen feldolgozza az SQL utasítást és a paraméterek helyét előkészíti. Végül az így előkészített utasítást konkrét értékekkel lefuttatjuk. Az előkészítés alapvetően akkor hasznos, ha ugyanazt az SQL utasítást többször szeretnénk futtatni más és más paraméterekkel, de mi főleg azt használjuk ki, hogy így a paraméterek átadása biztonságos csatornán keresztül történik.

```
$stmt = $kapcsolat->prepare("SELECT * FROM `zeneszamok` WHERE
`album_id` = :album_id");
$stmt->execute([
  "album_id" => 1
]);
$zeneszamok = $stmt->fetchAll();
var_dump($zeneszamok);
```

Természetesen az utasítások ezen csoportja paraméterek nélkül is működik. Így a lekérdezés egy segédfüggvényben általánosítható:

```php
function lekerdezes($kapcsolat, $sql, $parameterek = []) {
  $stmt = $kapcsolat->prepare($sql);
  $stmt->execute($parameterek);
  return $stmt->fetchAll();
}

// használata
$zeneszamok = lekerdezes($kapcsolat,
  "SELECT * from zeneszamok where album_id = :album_id",
  [ "album_id" => 1 ]
);

var_dump($zeneszamok);
```

Apróbetűs

A szövegösszefűzés nagy hátránya, hogy alkalmazásunkat kitesszük az ún. SQL beszúrásos támadás (SQL Injection) veszélyének. A paraméterek ugyanis tipikusan a felhasználói felületről érkeznek. Ha ezeket nem vizsgáljuk vagy nem megfelelő módon készítjük elő, akkor egy rosszindulatú támadó olyan szövegrészletet küldhet az alkalmazásunkba, amely nem várt módon változtatja meg az SQL utasításunkat.

Példa

Adott a következő SQL utasítás:

```
$query = "SELECT `id`, `name`, `inserted`, `size` FROM
`products`
          WHERE `size` = '${size}'";
```

Ha a `$size` értékeke a következő:

```
'
union select '1', concat(uname||'-'||passwd) as name, '1971-
01-01', '0' from usertable;
--
```

akkor a végső SQL utasítás ez lesz:

```
SELECT `id`, `name`, `inserted`, `size` FROM `products`
      WHERE `size` = ''
union select '1', concat(uname||'-'||passwd) as name, '1971-
01-01', '0' from usertable;
--'
```

> ami a termékek listázása mellett felhasználóneveket és jelszavakat is visszaad.

5.5.5 Módosító utasítások (beszúrás, módosítás, törlés)

Paraméter nélküli módosító utasításokat a `PDO::exec` metódusával lehet kiadni. Általában azonban paraméteres SQL utasításokat kell itt is írni, amiben az előkészített utasítások vannak segítségünkre. Az alábbi példa azt is megmutatja, hogy az utoljára beszúrt sor azonosítóját hogyan lehet lekérni a `PDO::lastInsertId` segítségével.

```
$kapcsolat
  ->prepare("
    INSERT INTO `albumok` (`egyuttes`, `cim`, `ev`) VALUES (:egyuttes,
:cim, :ev)
  ")
  ->execute([
    "egyuttes"  => "Roxette",
    "cim"       => "Joyride",
    "ev"        => 1991
  ]);

$id = $kapcsolat->lastInsertId();
var_dump($id);
```

Természetesen ez a logika is általánosítható:

```
function vegrehajtas($kapcsolat, $sql, $parameterek = []) {
  return $kapcsolat
    ->prepare($sql)
    ->execute($parameterek);
}

// használata
vegrehajtas($kapcsolat,
  "INSERT INTO albumok (egyuttes, cim, ev) values (:egyuttes, :cim,
:ev)",
  [
    "egyuttes"  => "Roxette",
    "cim"       => "Joyride",
    "ev"        => 1991
  ]
);

$id = $kapcsolat->lastInsertId();
var_dump($id);
```

5.5.6 Hibakezelés

```
try {
  $kapcsolat = kapcsolodas("sqlite:./zene.sqlite");

  vegrehajtas($kapcsolat,
    "INSERT INTO `zeneszamok` (`cim`) VALUES (:cim)",
    [ "cim" => "Joyride" ]
  );
}
catch (PDOException $e) {
  var_dump($e->errorInfo);
}
```

5.5.7 Segédfüggvények

```
function kapcsolodas($kapcsolati_szoveg) {
  $pdo = new PDO($kapcsolati_szoveg);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  return $pdo;
}

function lekerdezes($kapcsolat, $sql, $parameterek = []) {
  $stmt = $kapcsolat->prepare($sql);
  $stmt->execute($parameterek);
  return $stmt->fetchAll();
}

function vegrehajtas($kapcsolat, $sql, $parameterek = []) {
  return $kapcsolat
    ->prepare($sql)
    ->execute($parameterek);
}
```

Példa

Jelenítsük meg az albumokat felsorolásban! A lista fölé tegyünk egy szűrőmezőt, amit kitöltve és elküldve az együttesek nevében keres! A fenti segédfüggvények jelenlétét feltételezzük!

```
<?php
$szuro = $_GET["szuro"] ?? "";

$kapcsolat = kapcsolodas("sqlite:./zene.sqlite");

$albumok = lekerdezes($kapcsolat,
  "SELECT * FROM `albumok` WHERE `egyuttes` LIKE :egyuttes",
  [ ":egyuttes" => "%${szuro}%" ]
);
?>
<h1>Albumok</h1>
<form action="">
  <input name="szuro">
  <button type="submit">Szűr</button>
</form>
<ul>
  <?php foreach ($albumok as $album) : ?>
    <li>
      <?= $album["egyuttes"] ?>:
      <?= $album["cim"] ?>
      (<?= $album["ev"] ?>)
    </li>
  <?php endforeach ?>
</ul>
```

6 BEMENET KEZELÉSE

6.1 Bemeneti adatok elérése és használata

Mint minden számítógépes programnak, úgy a szerveroldali alkalmazásoknak is szüksége van valamilyen bemenetre, ahhoz, hogy érdemben működni tudjanak. Szerveralkalmazások esetén ez a bemenet a klienstől, vagy valamilyen adattárból (fájl, adatbázis) érkezik valamilyen formában.

A felhasználói bemenet leggyakoribb formája az, amikor a szerverhez érkező HTTP kérés tartalmazza a működéshez szükséges bemenetet. Ez a **HTTP protokoll megfelelő metódusai** ↪ révén jut el a szerverhez. A PHP programhoz ezek az klienstől érkező adatok már előre részben feldolgozott formában érkeznek.

Habár van lehetőség a böngészőben, a címsoron keresztül is adatokat küldeni a szervernek, mégis a leggyakoribb módja a bemenetnek az űrlapok kitöltése és szervernek való elküldése. Ehhez a HTML űrlapot el kell látni a megfelelő paraméterekkel:

- Az adatokat feldolgozó PHP oldal címe (a `form` HTML elem `action` attribútuma)
- Az adatküldés módja (a `form` HTML elem `method` attribútuma - ez lehet `get` vagy `post`)
- Az egyes beviteli mezők neve (az `input`, `select` és `textarea` HTML elemek `name` attribútuma)
- Elküldő gomb (`input` vagy `button` elem `type="submit"` attribútummal)

Példa

```
<form method="post" action="feldolgoz.php">
  Név:
  <input type="text" name="nev">
  Életkor:
  <input type="number" name="eletkor">
  Nem:
  <select name="nem">
    <option>Nő</option>
    <option>Férfi</option>
  </select>
  <input type="submit" value="Küldés">
</form>
```

Apróbetűs

A címsoron keresztül adatokat a webcímet követő kérdőjel után, `&` jelekkel elválasztva tudunk küldeni `get` metódussal:

Példa

```
http://www.pelda.hu/oldal.php?nev=Anna&eletkor=19
```

A beérkező adatokat feldolgozó PHP oldalon úgynevezett *szuperglobális változókon* keresztül érhetjük el. Ha az adatok `get` metódussal (a címsoron keresztül, a HTTP kérés fejlécében) érkeztek, akkor a `$_GET`, ha `post` metódussal (a HTTP kérés törzsében), akkor a `$_POST` változóban találhatóak az értékek.

Apróbetűs

A szuperglobális változók (mint a `$_GET` és a `$_POST`) olyan változók, melyek a PHP program bármelyik pontján elérhetőek.

Apróbetűs

A `get` metódus használata a sok esetben nem javasolt, mivel az így elküldött adatok a felhasználó számára szabad szemmel is láthatóak a címsorban. Az ilyen módon küldött jelszavak szövegesen láthatóak a böngészőprogramban, így a felhasználó mögött álló személy, vagy bárki, akinek hozzáférése van a böngésző előzményeihez nagyon könnyen hozzáférhet ezekhez.

A `$_GET` és a `$_POST` esetében is egy olyan asszociatív tömböt (kulcs-érték párok sorozata) kapunk, melyben a kulcsok a HTML űrlap mezők `name` attribútumai, a hozzájuk tartozó értékek pedig az adott űrlap mezők kitöltött tartalma.

Apróbetűs

- A jelölőnégyzetek esetében nem jelenik meg az adott kulcs, ha a jelölő nem volt bepipálva, ha pedig igen, akkor `"on"` értéket kapunk.
- Választógomboknál az összetartozó választógombokat (melyek közül csak egyet lehet kiválasztani) azonos `name` attribútummal kell ellátni. Értékként a kiválasztott választókapcsoló `value` attribútumának értékét kapjuk. Ha nincs kitöltve a `value`, akkor csak egy `"on"` értéket kapunk, melyből nem derül ki, hogy melyik lehetőséget választotta a felhasználó.
- A legördülő lista esetén értékként a kiválasztott elem szövegét kapjuk értékként, vagy az adott `option` elem `value` attribútumát, ha az ki volt töltve.

Példa

A fenti példában bemutatott HTML oldalon elküldött adatokat az alábbi módon érhetjük el PHP-ból:

```
// az elküldött név:
$_POST["nev"];
// az elkdüldött életkor:
$_POST["eletkor"];
// az elküldött nem:
$_POST["nem"];
```

6.2 Bemeneti adatok ellenőrzése

Minden olyan esetben, amikor egy számítógépes program felhasználói bemenettel dolgozik, felmerül a kérdés, hogy a felhasználó által megadott érték helyes-e. A webes világban a bemeneti adatok helyességének ellenőrzése kifejezetten fontos, mivel azon túl, hogy a helytelen bemenet a program hibás működését okozhatja, komoly biztonsági kockázatot is jelenthet.

Apróbetűs

A bemenet ellenőrzésénél felmerülhet ötletként, hogy az ellenőrzést végezzük kliens oldalon, a böngészőben, a HTML elemekre történő mintaillesztéssel. Ez a megoldás az elgépeléses hibák ellen ugyan védelmet nyújt, de a rosszindulatú támadásokkal szemben nem, mivel a HTTP kérések akár egy egyszerű böngészőprogrammal is tetszőlegesen módosíthatók. Ez az oka annak, hogy egy biztonságos rendszer létrehozásához mindenképpen szükség van az adatok helyességének szerveroldali ellenőrzésére.

A bemenet ellenőrzésénél kétféle tulajdonságát vizsgáljuk a kapott adatnak: *megfelelő-e a típusa* és *megfelel-e a programunk előfeltételének* (például egy adott szám valamilyen intervallumba esik-e, vagy a szöveg hossza nem halad-e meg valamilyen előre definiált értéket). Egy jellemző vizsgálat például az, amikor egy szövegről azt akarjuk eldönteni, hogy üres-e. Ezeket az ellenőrzéseket legegyszerűbben egyszerű elágazásokkal tudjuk elvégezni. Ügyeljünk arra, hogy csak akkor próbáljuk vizsgálni a bementi paraméterek értékét, ha volt egyáltalán valamilyen bemenet. Ezt például úgy tudjuk megvizsgálni, hogy a `$_GET` vagy a `$_POST` tömbök elemszámát (`count`) vizsgáljuk, hogy nagyobb-e mint nulla.

```
if (count($_POST) > 0) {
  /* bemenet ellenőrzése */
}
```

Egy változó típusának ellenőrzésére a PHP nyelv tartalmaz beépített segédfüggvényeket. Ezek a segédfüggvények mind `in_*` formátumúak, ahol a csillag egy adott típust jelöl (pl. `is_float`). Mivel az űrlapokon keresztül érkező adatok mindig szöveges formában érkeznek, ezért a valódi típus helyett érdemesebb azt vizsgálni, hogy a szöveg átalakítható-e a kívánt típussá. Számok esetében erre az `is_numeric` függvényt tudjuk használni. Ha a bemenet nem felel meg valamelyik feltételnek, akkor az adott hibához generálunk egy hibaüzenetet, amit egy tömbben tárolunk. Az ellenőrzés végén, ha nem volt hiba, akkor a tömb üres maradt, ha pedig volt, akkor a tömbben lévő hibaüzenetet megjelenítjük a

Példa

A korábbiakban bemutatott példa űrlap bemenetének egy lehetséges vizsgálata:

```
// itt fogjuk tárolni a hibaüzeneteket
$hibak = [];

if (count($_POST) > 0) {
  // a név legyen szöveg, legalább három karakter hosszú, és van
benne legalább egy szóköz
  if (!is_string($_POST["nev"]) ||
      !str_len($_POST["nev"]) >= 3 ||
      !substr_count($_POST["nev"], " ") >= 1) {
    $hibak[] = "A név formátuma nem megfelelő";
  }
  if (!is_numeric($_POST["eletkor"]) ||
    $hibak[] = "Az életkor csak szám lehet!";
  } else {
    if ($_POST["eletkor"] < 13) {
      $hibak[] = "Legalább 13 évesnek kell lenned!";
    }
  }
  if (!isset($_POST["nem"]) ||
      !in_array($_POST["nem"], ["férfi", "nő"])) {
    $hibak[] = "Nem választottad ki a nemedet!";
  }
}
```

Az ellenőrzést követően a HTML sablonban meg tudjuk jeleníteni a hibaüzeneteket:

```
<ul class="hibauzenetek">
  <?php foreach($hibak as $hiba) : ?>
    <li><?= $hiba ?></li>
  <?php endforeach; ?>
</ul>
```

Apróbetűs

Ha nem csak hibaüzeneteket szeretnénk tárolni, hanem például egy művelet sikerességét jelző üzenetet is, akkor arra érdemes külön tömböt használni. Ezzel a módszerrel a sablonban külön helyen és különböző formátummal tudjuk megjeleníteni a hiba- és egyéb üzeneteket.

Példa

```php
$hibak = [];
$uzenetek = [];

if (count($_POST) > 0) {
  /*
    bemenet ellenőrzése
  */
  if (count($hibak) == 0) {
    $uzenetek[] = "Sikeres feldolgozás!";
  }
}
```

```php
<ul class="hibauzenetek">
  <?php foreach($hibak as $hiba) : ?>
    <li><?= $hiba ?></li>
  <?php endforeach; ?>
</ul>
<ul class="uzenetek">
  <?php foreach($uzenetek as $uzenet) : ?>
    <li><?= $uzenet ?></li>
  <?php endforeach; ?>
</ul>
```

6.3 Feladatok

1. **Feladat**

   Készíts egy egyszerű űrlapot, ami két darab szöveg beviteli mezőt tartalmaz, valamint egy legördülő listát, amiből a négy alap matematikai műveletet (+, -, ×, ÷) lehet kiválasztani. Az űrlap küldje el magának az adatokat `post` metódussal. Hibák esetén írj ki hibaüzenetet. Ha minden rendben van, akkor számítsd ki az eredményt és jelenítsd meg. A beérkezett adatokat az allábiak szerint ellenőrizd:

- Mindkét szám valóban szám legyen, osztás esetén a második szám ne legyen 0,
- a műveleti jel az előre meghatározottak valamelyike.

Letöltés

⭳ **A feladat megoldása ↪**

2. **Feladat**

Készíts egy űrlapot, amin egy bevásárlólistába lehet új elemet felvenni! Lehessen megadni, hogy miből mennyit kell venni, illetve külön mértékegységet is. Egy legördülő listából lehessen kiválasztani (előre beégetett nevek közül), hogy kinek a feladata a beszerzés. A bemenetet értelemszerű feltételekkel ellenőrizd. Ha minden rendben van, akkor mentsd el az adatokat adatbázisba és értesítsd a felhasználót, hogy sikeres volt a felvétel. Ha hiba volt, azt is jelezd. Az oldal listázza ki az összes eddig felvett bevásárlólista-tételt.

7 KÓDSZERVEZÉS

Nagyobb szerver oldali alkalmazások készítésekor felmerül a kérdés, hogy hogyan érdemes a programkódunkat részekre osztani, hogy jobban átlátható legyen a program. Erre azért van szükség, mert ahogy nő az alkalmazásunk mérete, egyre több funkció, nézet (különböző elrendezésű oldal) jelenik meg, ezek ha nincsenek megfelelően szervezve, akkor átláthatatlanná teszik a kódot. Ehhez érdemes meghatározni az alkalmazás funkcióban jól elkülönülő részeit.

- Az adatbázis elérése + maga az adatbázis
- A bemenet feldolgozása, számítások elvégzése, adatok eltárolása
- Adatok kiolvasása az adatbázistól, azok megjelenítése sablonok segítségével

> **Apróbetűs**
>
> Ez a hármas felosztás az úgynevezett **MVC architektúra** ↪ hármas felosztásán alapul. Ez az alkalmazás-architektúra széles körben elterjedt mind a weben, mind pedig más platformokon.

A kódszervezés szükségességének legszembetűnőbb formája az, amikor egy szerveroldali webes alkalmazás több oldalból áll. Ezek az oldalak részben független tartalommal rendelkeznek, melyhez külön bemenet-feldolgozás tartozhat. Amellett, hogy más formában jelenik meg a tartalom, a legtöbb esetben vannak a HTML sablonnak a különböző oldalak közötti *közös részei*, melyek minden oldalon ismétlődnek (általában ilyen rész a fejléc és a lábléc). Ezeket az ismétlődő részeket minden oldal sablonja tartalmazza, és ha egy helyen módosítani kell valamit, akkor azt a módosítást az összes többi oldalon is meg kell tenni.

Erre a problémára megoldást jelent az, hogy ha az ilyen ismétlődő részeket egy külön fájlban helyezzük el, és azok tartalmát a megfelelő helyen beillesztjük az oldalunkba az `include` utasítás segítségével. Az `include` tulajdonképpen nem csinál mást, mint egy másik fájl tartalmát beilleszti a megfelelő helyen a forráskódunkba. Ebből az is következik, hogy az `include` utasítás használatának pillanatában létező összes változó elérhető a külső fájlban is. Az `include` használható mind statikus HTML részek, dinamikus sablonok, vagy tisztán PHP kódok betöltésére is.

Példa

```
<!-- az oldal fejléce -->
<?php include "fejlec.php"; ?>

<!-- az oldal egyedi része, tartalmak megjelenítése -->

<!-- az oldal lábléce -->
<?php include "lablec.php"; ?>
```

Apróbetűs

Az `include` utasításnak létezik három további változata is. Ezek működését az alábbi táblázat foglalja össze:

UTASÍTÁS	BETÖLTI TÖBBSZÖR UGYANAZT?	HA NEM TALÁLHATÓ A KÜLSŐ FÁJL?
`include`	igen	`warning` szintű hiba, a program tovább fut
`include_once`	nem	`warning` szintű hiba, a program tovább fut
`require`	igen	`fatal` szintű hiba, a program leáll
`require_once`	nem	`fatal` szintű hiba, a program leáll

Az `include` utasítás segítségével tisztán PHP kódot tartalmazó fájlokat is be tudunk tölteni. Ez olyan kódrészletek esetében hasznos, melyeket több oldal is használ. Elsősorban függvények és osztályok definícióit érdemes ilyen módon kivinni külső fájlba, például az adatbáziskezelés fejezetben bemutatott `kapcsolodas`, `lekerdezes`, `vegrehajtas` segédfüggvények. Így bármelyik oldalon, ahol szeretnénk használni az adatbázist csak annyit kell tennünk, hogy betöltjük a külső fájlt, ami a segédfüggvényeket tartalmazza.

```
include `adatbazis.php`;

/* adatbázis-függvények használata */
```

Kódszervezésre természetesen nem csak fájlok szintjén, hanem fájlokon belül is lehetőség van. Az egyes oldalak kódjainak jobb átláthatósága érdekében érdemes egy előre meghatározott

szisztéma szerint felépíteni a kódunkat. A PHP oldalakon belül a javasolt felépítés:

1. Tisztán PHP kódot tartalmazó külső függőségek betöltése (`include`-ok) – ezeket az oldal további részei használják.
2. Adatbázis csatlakozás létrehozása, ha szükséges – az adatbázis-kapcsolatra szintén szüksége lehet a többi részben.
3. Ha szükséges, akkor a bemenet feldolgozása, adatbázis módosítása, ha a szükséges – azért kerül ide, hogy az itt végzett módosítások az oldalon már megjelenjenek.
4. Az oldalon szükséges adatok lekérdezése az adatbázisból – a sablon enélkül nem jeleníthető meg.
5. A sablon megjelenítése a szükséges adatokkal és üzenetekkel, az ismétlődő részek betöltése külső fájlokból – itt már minden szükséges módosítást elvégeztünk és minden adat rendelkezésünkre áll.

Példa

```php
<?php
// 1. külső állományok betöltése
include "adatbazis.php";

// 2. adatbázis-kapcsolat létrehozása
$kapcsolat = kapcsolodas("sqlite:./adatbazis.sqlite");

// 3. bemenet feldolgozása
$hibak = [];
$uzenetek = [];
if (count($_POST) > 0) {
  // Bemenet ellenőrzése
  if (count($hibak) == 0) {
    // Bemenet feldolgozása (`vegrehajtas`)
  }
}

// 4. adatok lekérdezése
$adatok = lekerdezes($kapcsolat, /* "SELECT ..." */);
?>
<!-- 5. sablon megjelenítése -->
<?php include "fejlec.html"; ?>

<main>
  <!-- adatok, űrlapok megjelenítése -->
</main>

<?php include "lablec.html"; ?>
```

Apróbetűs

A példában is látszik, hogy ezzel a felosztással a teljes oldal két nagy egységre tagolódik - egy tisztán PHP kódból álló részből, illetve egy dinamikus PHP részeket tartalmazó HTML sablonból. Komolyabb MVC architektúrára épülő keretrendszerek ezt a két egységet is szétszedik külön fájlokra, ezek alkotják az úgynevezett *vezérlő* (controller) és *nézet* (view) egységeket.

Apróbetűs

Az általunk bemutatott alkalmazásokban és példákban az oldalak közötti navigációt linkek segítségével oldjuk meg, a fájlokat a webszerver (pl. Apache, nginix) szolgálja ki. Komolyabb rendszerek esetén a nagyobb biztonság és a nagyobb kontroll érdekében ezt az alapértelmezést felül szokták írni és az egyes fájlok kiszolgálását is PHP kód végzi `include`-ok segítségével. Ezt a megoldást nevezzük "routing"-nak.

7.1 Feladatok

1. **Feladat**

   Készítsd el a családi bevásárlás alkalmazáshoz a szükséges fejléc és lábléc fájlokat, szervezd ki az adatbázis kapcsolatot külön fájlba, és alakítsd át az oldaladat a bemutatott strukúrának megfelelően.

8 MUNKAMENET-KEZELÉS

A központi adattárolás nagy előnye, hogy az adatok bármikor bármelyik kliens által hozzáférhetőek az alkalmazáson keresztül. Ezek az adatok viszont közösek minden kliensre nézve. Sokszor azonban szükség van arra, hogy a klienseket megkülönböztessük és bizonyos adatokat kliensenként tároljunk. *A kliensek megkülönböztetését és a kliensenkénti adattárolást munkamenet-kezelésnek hívjuk.*

> **Példa**
>
> Egy webáruház esetén jó dolog az, hogy a bolt árukészletét mint központi adatot az egyes vásárlók elérik, böngészhetik. Mindenki ugyanazt látja. A kosár tartalma azonban vásárlónként, pontosabban kliensenként eltérő. Nem egy nagy közös bevásárló kosár van, hanem mindenkinek külön-külön van egy.
>
> Jó pár további példa van a mindennapi életben arra, hogy az adatokat felhasználónként szükséges megkülönböztetni:
>
> - email fiók,
> - internetbank,
> - dokumentumkezelés.

8.1 A munkamenet-kezelés működése

A klienshez tartozó adat mind a böngészőben, mind szerveroldalon tárolható. Ez a tananyag a szerveroldali megoldást tárgyalja. Ennek több előnye van:

- a tárolandó adat nem közlekedik a kliens és a szerver között minden egyes kérésnél, valamint
- kliensoldalon nem lehet hozzáférni, így biztonságosabb is.

A munkamenet kezelése, azaz a kliensek megkülönböztetése és kliensenkénti adattárolás a következőképpen történik.

1. A megkülönböztetés végett minden kliens első kérésekor kap egy egyedi azonosító kulcsot a szervertől. Ez tipikusan egy jó hosszú véletlenszerűen előállított szöveg.
2. A kliens ezt a kulcsot a szerver által meghatározott név alatt (pl. `PHPSESSID`) sütiként tárolja el. A süti a böngészőbeli adattárolás egyik lehetséges formája, ahol a böngésző minden kéréshez az adott szerverhez tartozó süti-adatokat automatikusan elküldi.
3. Amikor a böngésző a szerver felé kérést intéz (hivatkozás, űrlap), akkor a böngésző a szerverhez tartozó kulcsot is elküldi.

4. A szerver a kérés feldolgozásakor megnézi, érkezett-e adott nevű süti. Ha igen, akkor a szerveroldali program rendelkezésére bocsátja az adott kulcshoz tartozó adatokat. Ha nem, akkor egy új kulcsot generál, és leküldi a kliensnek.
5. A munkamenet mindaddig él, amíg a kliens küldi a munkamenet-azonosító kulcsot. Ha az ezt tároló süti lejár, törlődik, akkor a munkamenet is véget ér.

A munkamenetkulcs és a hozzá tartozó adatot úgy is elképzelhetjük, mint egy banki széfszolgáltatást. Bérléskor kapunk egy kulcsot, és csak ezzel a kulcssal férhetünk hozzá a széf tartalmához. Ha elveszítjük a kulcsot, akkor az abban tárolt adathoz már nem férünk hozzá. Ha valaki ellopja a kulcsunkat, akkor az adatainkhoz a mi nevünkben fér hozzá. Ezért nagyon fontos a süti adatokra vigyázni (ld. a **Weblabor ezzel foglalkozó cikkét ↪**)!

A munkamenet által tárolt kulcsot a böngésző fejlesztői eszköztárával tudjuk megfigyelni. Chrome böngészőben az Application fül alatt a Cookies kategóriában tudjuk a megfelelő szerver alatt kikeresni és akár kézzel módosítani, törölni.

8.2 Munkamenet-kezelés PHP-ban

PHP-ban a `session_start` utasítással lehet a munkamenet-kezelést aktiválni. Ez a függvény nézi meg, érkezik-e munkamenetkulcsot tartalmazó süti, ha igen, akkor elérhetővé teszi az adatokat, ha nem, akkor új kulcsot generál és gondoskodik sütiként való leküldésében. Mivel az adatokra igen hamar szükség lehet a kódban, így a `session_start` az első utasítások egyike a PHP kód elején. Minden fájlban meg kell jelennie, ahol munkamenetet szeretnénk kezelni.

```
<?php
session_start();
// többi utasítás
```

Az adatok a `$_SESSION` szuperglobális asszociatív tömbben érhetők el. A munkamenetből olvasni egyet jelent a tömb megfelelő elemére való hivatkozással, munkamenetbe írni pedig úgy kell, hogy a tömbben egy új kulcs alatt helyezzük el a tárolandó adatot. Egy munkamenetbeli adat törlését a megfelelő kulcs törlésével végezzük el. A `$_SESSION` tömb a `session_start` hívás után töltődik fel adatokkal, ezért csak azután használjuk!

```
session_start();

// olvasás
$valami = $_SESSION["valami"];

// írás (új létrehozás, módosítás)
$_SESSION["valami"] = 12;

// törlés
unset($_SESSION["valami"]);
```

A munkamenet megszüntetését a `$_SESSION` tömb ürítésével és a `session_destroy` utasítással végezhetjük el. Ezeket is meglévő munkameneten lehet értelmezni, azaz a `session_start` után helyezzük el.

```
// munkamenet indítása
session_start();

// munkamenetadatok törlése
$_SESSION = array();

// munkamenet törlése
session_destroy();
```

8.3 Sütik

Apróbetűs

A süti (angolul cookie) leírását a HTTP protokollban találjuk. Eredetileg arra találták ki, hogy a szerver kezdeményezésére adatot lehessen tárolni a böngészőben. A böngésző pedig az adott szerverhez tartozó sütiket automatikusan visszaküldte. Adattárolásra volt tehát lehetőség a kliens-szerver viszonylatában.

Mivel kliensoldalon más adattárolási lehetőség kezdetben nem volt, így a megfelelő JavaScript függvények segítségével a sütiket használták a kliensoldali alkalmazások adatainak tárolására olyan esetekben is, ahol az adatnak a szerverhez semmi köze nem volt. Ráadásul a sütiket JavaScript segítségével szabadon lehetett manipulálni, ami komoly biztonsági kockázatokat vetett fel (süti és így azonosítólopásokhoz vezetett). Mivel a sütiket eredendően nem erre találták ki, megjelentek tipikusan a kliensoldali tárolásra alkalmas JavaScript interfészek (Web Storage, IndexedDB). Ezekkel együtt a sütik alkalmazása is visszatért eredeti funkciójához, a szervertől kapott adatok tárolásához.

8.4 Hitelesítés és jogosultságkezelés

Egy webes alkalmazás esetén sokszor van szükségünk arra az információra, hogy ki is használja az alkalmazást. Ennek eldöntését nevezzük *hitelesítés*nek. A hitelesítés eredménye alapján megkülönböztetünk azonosítatlan (vendég) és azonosított felhasználót. Annak eldöntése, hogy a hitelesítés eredményeképpen kapott felhasználó mihez fér hozzá, *jogosultságkezelés*nek hívjuk. Előfordulhat, hogy egész oldalakat csak belépett, azaz azonosított felhasználóknak teszünk hozzáférhetővé; de az is elképzelhető, hogy csak az oldal egy része változik ennek az információnak megfelelően (ld. pl. a belép/kilép gomb váltakozását).

8.4.1 Hitelesítés munkamenettel

A hitelesítés általában egy egyszeri lépés: a felhasználónevet és a jelszót csak egyszer szükséges helyesen begépelnünk. Jó lenne, ha a többi kérésünknél a szerveroldali kód már látná, hogy ezen az egyszeri lépésen sikeresen túl vagyunk. A hitelesítés kliensenként változik: ki bejelentkezett már, ki nem. Ezt az információt tehát érdemes kliensenként tárolni, erre egyik alkalmas eszköz a munkamenet használata.

Munkamenettel a hitelesítés folyamata a következő:

1. A felhasználó beírja felhasználónevét és jelszavát.
2. Ha ez helyes, akkor a szerver a felhasználó munkamenetében egy egyszerű adat tárolásával jelzi, hogy a felhasználó azonosítása sikeresen megtörtént.
3. További kérésekkor – ha szükséges – meg kell vizsgálni, hogy az adott bejegyzés jelen van-e a munkamenetben. Ha igen, akkor a felhasználó továbbra is azonosított. Ha nem, akkor vendég felhasználóval van dolgunk.
4. A felhasználó kijelentkeztetése egyszerűen az adott bejegyzés munkamenetből való törlésével egyenértékű.

A továbbiakban nézzük végig egy hitelesítési folyamat lépéseit!

8.4.2 Adatbázis

A hitelesítés adatainkat egy adatbázis (mi esetünkben SQLite) `felhasznalok` nevű táblájában fogjuk tárolni:

```
CREATE TABLE `felhasznalok` (
  `id`                INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  `felhasznalonev`    TEXT NOT NULL UNIQUE,
  `jelszo`            TEXT NOT NULL
);
```

8.4.3 Regisztráció

A regisztráció a hitelesítés előszobája, amely során a rendszerbe felvisszük azonosító adatainkat (pl. felhasználónév és jelszó). A folyamat a következő lépésekből áll:

1. Egy erre szolgáló űrlapon megadjuk a regisztrálandó felhasználónév-jelszó párost (esetleg egyéb adatokat is).
2. Az adatokat leellenőrizzük (kitöltöttség, helyes formátum, nem létező felhasználónév).
3. Helyes megadás esetén elmentjük az adatbázisba.

A következőkben az adatbázis műveleteket adottnak tekintjük. Két függvényt hozunk létre a felhasználók kezelésére: a `letezik` függvény megvizsgálja, hogy adott nevű felhasználó létezik-e már az adatbázisban; a `regisztral` függvény pedig elmenti a megadott felhasználónév és kódolt jelszó párost az adatbázisba.

```php
<?php
function letezik($kapcsolat, $felhasznalonev) {
  $felhasznalok = lekerdezes($kapcsolat,
    "SELECT * FROM `felhasznalok` WHERE `felhasznalonev` =
:felhasznalonev",
    [ ":felhasznalonev" => $felhasznalonev ]
  );
  return count($felhasznalok) === 1;
}
function regisztral($kapcsolat, $felhasznalonev, $jelszo) {
  $db = vegrehajtas($kapcsolat,
    "INSERT INTO `felhasznalok` (`felhasznalonev`, `jelszo`)
      values (:felhasznalonev, :jelszo)",
    [
      ":felhasznalonev"   => $felhasznalonev,
      ":jelszo"           => password_hash($jelszo, PASSWORD_DEFAULT),
    ]
  );
  return $db === 1;
}

$hibak = [];
if (count($_POST) > 0) {
  $felhasznalonev = $_POST["felhasznalonev"];
  $jelszo = $_POST["jelszo"];

  $kapcsolat = kapcsolodas("sqlite:./zene.sqlite");

  if (letezik($kapcsolat, $felhasznalonev)) {
    $hibak[] = "Már létező felhasználónév!";
  }

  if (count($hibak) === 0) {
    regisztral($kapcsolat, $felhasznalonev, $jelszo);
    header("Location: bejelentkezik.php");
    exit();
  }
}
?>
<?php var_dump($hibak); ?>
<form action="" method="post">
  Felhasználónév:
  <input type="text" name="felhasznalonev"> <br>
  Jelszó:
  <input type="password" name="jelszo"> <br>
  <button type="submit">Regisztrál</button>
</form>
```

> **Apróbetűs**
>
> Soha ne tároljuk a jelszót egy az egyben, mert az komoly biztonsági kockázatot jelent. Mindig kódoljuk le egyirányú kódoló algoritmusokkal, és későbbiekben mindig a kódolt jelszóval dolgozzunk! PHP-ban jelszókódolásra külön függvény, a `password_hash` szolgál.

8.4.4 Bejelentkezés

A hitelesítési folyamat legfontosabb része a bejelentkeztetés. Ennek során győződünk meg arról, hogy a megadott felhasználónév és jelszó páros helyes, és ennek sikeres tényét a munkamenetbe eltároljuk.

```
<?php
function ellenoriz($kapcsolat, $felhasznalonev, $jelszo) {
  $felhasznalok = lekerdezes($kapcsolat,
    "SELECT * FROM `felhasznalok` WHERE `felhasznalonev` =
:felhasznalonev",
    [ ":felhasznalonev" => $felhasznalonev ]
  );
  if (count($felhasznalok) === 1) {
    $felhasznalo = $felhasznalok[0];
    return password_verify($jelszo, $felhasznalo["jelszo"])
      ? $felhasznalo
      : false;
  }
  return false;
}

function beleptet($felhasznalo) {
  $_SESSION["felhasznalo"] = $felhasznalo;
}

session_start();

$hibak = [];
if (count($_POST) > 0) {
  $felhasznalonev = $_POST["felhasznalonev"];
  $jelszo = $_POST["jelszo"];

  $kapcsolat = kapcsolodas("sqlite:./zene.sqlite");
  $felhasznalo = ellenoriz($kapcsolat, $felhasznalonev, $jelszo);

  if ($felhasznalo === false) {
    $hibak[] = "Hibás adatok!";
  }

  if (count($hibak) === 0) {
```

```
    beleptet($felhasznalo);
    header("Location: fooldal.php");
    exit();
  }
}
?>
<?php var_dump($hibak); ?>
<form action="" method="post">
  Felhasználónév:
  <input type="text" name="felhasznalonev"> <br>
  Jelszó:
  <input type="password" name="jelszo"> <br>
  <button type="submit">Bejelentkezik</button>
</form>
```

8.4.5 Hitelesítés ellenőrzése

A további hitelesítéshez elegendő a munkamenet megfelelő kulcsának meglétét ellenőrizni.

```
function azonositott_e() {
  return isset($_SESSION["felhasznalo"]);
}
```

8.4.6 Kijelentkezés

Kijelentkeztetéshez elegendő a megfelelő kulcs törlése a munkamenetből.

```
function kijelentkeztet() {
  unset($_SESSION["felhasznalo"]);
}
```

8.4.7 Jogosultságvizsgálat

Jogosultságvizsgálat során azt nézzük meg, hogy az oldalt használónak van-e joga egy erőforrást elérni. Egyszerűbb esetekben elegendő annak megkülönböztetése, hogy vendég vagy azonosított felhasználó használja-e az oldalt. Ehhez kiváló eszköz az `azonositott_e` függvény. A következő példában egy teljes oldal levédéséhez használjuk:

```
session_start();
if (!azonositott_e()) {
  header("Location: bejelentkezik.php");
  exit();
}
```

Apróbetűs

Bonyolultabb esetekben a felhasználókat általában szerepkörökbe csoportosítják, és az egyes szerepkörök jogosultságait állítják be, illetve vizsgálják az adott erőforrás elérése során.

8.5 Feladatok

1. **Feladat**

   Egy webáruház főoldalán kategóriák szerint lehet termékeket megjeleníteni, és a választott terméket kosárba helyezni.

   1. Tegyél fel egy legördülő mezőt, amiben a kategóriákat tartalmazza. Egy mellette lévő gombot megnyomva, az oldalon jelenjenek meg az adott kategóriába eső termékek.
   2. Minden termék mellett legyen egy "Kosárba" feliratú gomb. Erre kattintva a termék a kosárba kerül, aminek tartalmát ki is íratjuk a terméklista alatt.
   3. A kosárbeli termékek mellett legyen egy "Eltávolít" feliratú gomb, ezzel a kosárból lehet törölni az elemeket.

   Az alábbi SQL utasítással lehet az adatbázist gyorsan létrehozni.

```
CREATE TABLE `kategoriak` (
  `id`  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  `nev` TEXT NOT NULL UNIQUE
);
INSERT INTO `kategoriak` (id,nev) VALUES (1,'laptop');
INSERT INTO `kategoriak` (id,nev) VALUES (2,'asztali gép');
INSERT INTO `kategoriak` (id,nev) VALUES (3,'okostelefon');
INSERT INTO `kategoriak` (id,nev) VALUES (4,'tablet');

CREATE TABLE `termekek` (
  `id`  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  `nev` TEXT NOT NULL,
  `ar`  INTEGER NOT NULL,
  `kategoria_id`  INTEGER NOT NULL,
   FOREIGN KEY(`kategoria_id`) REFERENCES `kategoriak`(`id`)
ON UPDATE RESTRICT ON DELETE RESTRICT
);
INSERT INTO `termekek` (id,nev,ar,kategoria_id) VALUES
(1,'Lenovo Thinkpad 12345',200000,1);
INSERT INTO `termekek` (id,nev,ar,kategoria_id) VALUES
(2,'Asus Zenbook P12345',250000,1);
INSERT INTO `termekek` (id,nev,ar,kategoria_id) VALUES
(3,'Lenovo Desktop Dream 2',230000,2);
INSERT INTO `termekek` (id,nev,ar,kategoria_id) VALUES (4,'HP
```

```
EliteWork A6235',280000,2);
INSERT   INTO   `termekek`   (id,nev,ar,kategoria_id)   VALUES
(5,'Xiaomi Redmi 4X',45000,3);
INSERT   INTO   `termekek`   (id,nev,ar,kategoria_id)   VALUES
(6,'Samsung Galaxy S8',145000,3);
INSERT   INTO   `termekek`   (id,nev,ar,kategoria_id)   VALUES
(7,'Samsung A123',65000,4);
INSERT   INTO   `termekek`   (id,nev,ar,kategoria_id)   VALUES
(8,'Apple iPad5',265000,4);
```

Letöltés

⭳ A feladat megoldása ↪

2. **Feladat**

Készíts egy teendőket kezelő alkalmazást!

1. Az adatbázisban tárold a felhasználókat és a teendőket. Minden felhasználónak több teendője is lehet, de egy teendő csak egy felhasználóhoz tartozhat (egy-sok kapcsolat).

```
CREATE TABLE `felhasznalok` (
  `id`                    INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT,
  `felhasznalonev`    TEXT NOT NULL UNIQUE,
  `jelszo`            TEXT NOT NULL
);
CREATE TABLE `teendok` (
  `id`  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  `nev` TEXT NOT NULL,
  `kesz`  INTEGER NOT NULL DEFAULT 0,
  `felhasznalo_id`  INTEGER NOT NULL,
        FOREIGN    KEY(`felhasznalo_id`)    REFERENCES
`felhasznalok`(`id`)  ON  UPDATE  RESTRICT  ON  DELETE
RESTRICT
);
```

2. A teendőlistát csak bejelentkezett felhasználó érheti el. Védd le a főoldalt. Bejelentkezés nélkül irányítson át a bejelentkező oldalra!
3. Legyen lehetőség regisztrálásra is! Regisztrálás után a bejelentkező oldalra kerüljünk!
4. Sikeres bejelentkezés után a főoldalra kerüljünk!
5. A főoldalon üdvözöljük a felhasználót és tegyük lehetővé a kijelentkezést.

6. A főoldalon listázzuk ki a felhasználóhoz tartozó teendőket! Ha kész van a teendő, akkor áthúzva jelenítsük meg!
7. Ugyanitt a teendőlista fölött legyen lehetőség új teendőt megadni.
8. A teendőlistában minden teendő mellett jelenjen meg egy gomb. Amíg nincs kész, addig "Megold" felirat jelenjen meg, ha kész van, akkor az "Újra" felirat. Erre kattintva a kész állapotot tudjuk váltogatni.

Letöltés

↓ **A feladat megoldása ↪**

9 FÁJLKEZELÉS

A szerveroldali webes alkalmazások képesek fájlok, fájlrendszerek kezelésére is. Ez egyrészről azt jelenti, hogy képesek (megfelelő jogosultságokkal) a szerver fájlrendszerén lévő fájlokat elérni, másrészről pedig képesek arra, hogy a klienstől fájlokat fogadjanak, azokat feldolgozzák vagy akár el is tárolják. Számos módon lehet képes egy szerveroldali program fájlokkal dolgozni. Ezekből mutatunk be néhányat.

9.1 Fájlrendszerek elérése

A PHP programok [teljes eszköztárral (**fájl- ↪** és **mappakezelő műveletek ↪**) rendelkeznek, amivel elérhetik, olvashaták vagy módosíthatják a szerver fájlrendszerét. Fáljok írására és olvasására a `fopen`, `fgets`, `fwrite` és `feof` függvényeket használhatjuk.

Példa

```
$fajl1 = fopen("szoveg1.txt", "r"); // a szoveg1.txt fájl
megnyitása olvasásra
$fajl2 = fopen("szoveg2.txt", "a"); // a szoveg2.txt fájl
megnyitása hozzáírásra

// amíg végére nem érünk a fájlnak, addig soronként olvasunk
while (!feof($fajl1)) {
  $adat = fgets($fajl1);
  fwrite($fajl2, $adat);
}

// fájlok lezárása
fclose($fajl1);
fclose($fajl2);
```

Fájlok olvasásán és módosításán kívül lehetőségünk van fájlok és mappák adatait is lekérdezni, illetve vizsgálhatjuk, hogy egy adott elérési út létezik-e, illetve, hogy fájlt vagy mappát tartalmaz.

Példa

```php
$utvonal = "/home/user/mappa";
// ha a megadott útvonal mappa
if (is_dir($utvonal)) {
  // akkor gyűjtsük ki a benne található fájlokat
  $fajlok = scandir($utvonal);
} else {
  // különben írjunk ki egy hibaüzenetet
  echo "A megadott útvonal nem mappát jelöl.";
}
```

Apróbetűs

A legtöbb parancssori fájl- és mappakezelő műveletnek létezik PHP utasítás megfelelője. Minden művelet, amit a PHP program végrehajt, az a webszervert futtató felhasználó (pl. `www_data`) kerül végrehajtásra. Ebből adódik, hogy fájlok, mappák, amiket a PHP program hoz létre ennek a felhasználónak a tulajdonában lesznek.

9.2 Feladatok

1. **Feladat**

   Készíts egy egyszerű programot, ami egy szöveges beviteli mezőben elkér egy útvonalat, és ha az mappa, akkor kilistázza a mappában található elemeket úgy, hogy minden elemnél jelzi, hogy az adott elem mappa-e vagy fájl, illetve ha fájl, akkor annak mérete.

   Letöltés

   ↧ A feladat megoldása ↪

9.3 Fájlok feltöltése

Mivel a PHP hozzáfér a fájlrendszerhez, így lehetőségünk van arra is, hogy a felhasználótól érkező feltöltött fájlokat kezeljük, és ha szükséges, akkor el is tároljuk. Ezáltal lehetőségünk van nem csak egyszerű szöveges tartalmak, de akár fájlok központi kiszolgálására is a felhasználók számára.

Ahhoz, hogy fájlokat tudjunk feltölteni, ki kell alakítanunk egy erre alkalmas űrlapot a felhasználó felületen. Ehhez két dologra van szükségünk: beállítani, hogy a megfelelő kódolással (`multipart/form-data`) küldje el az űrlap az adatokat, illetve szükség van egy fájl típusú űrlapmezőre (`type="file"`). Fontos, hogy fájlokat csak `post` metódussal van lehetőségünk elküldeni.

```
<form method="post" action="fajlfeltoltes.php"
enctype="multipart/form-data">
  <label for="fajl">Válaszd ki a feltöltendő fájlt: </label>
  <input type="file" name="fajl" id="fajl">
  <input type="submit" value="Feltölt!">
</form>
```

Mikor ilyen módon elküldünk egy fájlt a webszervernek, akkor az először átmegy egy kezdezi ellenőrzésen. Ekkor a szerver vizsgálja például azt, hogy ne haladja meg a megengedett legnagyobb fájlméretet a fájl (ez a beállítás a `php.ini` konfigurációs fájlban van benne). Ha minden feltételnek megfelel a fájl, akkor azt a szerver egy ideiglenes mappában eltárolja (pl. `/tmp`), és a feltöltést kezelő PHP szkript megkapja a feltöltött fájllal kapcsolatos információkat a `$_FILES` szuperglobális asszociatív tömbben. A `$_FILES` a `$_POST`-hoz hasonlóan a `name` attribútumnak megfelelő mezőben tárolja a beérkezett adatokat.

```
// így vizsgáljuk, hogy érkezett-e feltöltött fájl
if (count($_FILES) > 0) {}
```

A feltöltött fájlról számos információ rendelkezésünkre áll a tömbben, például a fájl típusa (`"type"`), eredeti neve (`"name"`), mérete (`"size"`), illetve, hogy milyen ideiglenes néven mentette el a szerver (`"tmp_name"`). Ezen információk alapján már könnyedén tudjuk kezelni a feltöltött fájlt, például szűrni a feltölthető fájlokat típus, vagy méret alapján. Ha szeretnénk eltárolni a fájlt, akkor mindenképp szükséges, hogy azt áthelyezzük, mivel az ideiglenes mappa a legtöbb operációs rendszeren rendszeresen törlésre kerül.

Áthelyezésre a `move_uploaded_file` függvény szolgál.

Példa

Képfájl feltöltése és mentése a szerveren.

```
$hibak = [];
$uzenetek = [];
if (count($_FILES) > 0) {
  // ha hiba volt a feltöltés közben
  if ($_FILES["fajl"]["error"] != UPLOAD_ERR_OK) {
    $hibak[] = "Hiba a fájl feltöltésekor!";
  } else {
    // ha nem kép, vagyis a típusában szerepel az, hogy "image"
```

```
    if (!substr_count($_FILES["fajl"]["type"], "image") > 0) {
      $hibak[] = "A feltöltött fájl nem kép!";
    } else {
      // ha minden rendben, akkor eltároljuk a fájlt a `fajlok`
mappába
      $cel_utvonal = "fajlok/" . $_FILES["fajl"]["name"];
      // a `move_uploaded_file` függvény eredményéből tudjuk meg,
hogy sikeres volt-e a másolás
      $siker = move_uploaded_file($FILES["fajl"]["tmp_name"],
$cel_utvonal);
      if ($siker) {
        $uzenetek[] = "Sikeres fájlfeltöltés!";
      } else {
        $hibak[] = "Sikertelen fájlfeltöltés!";
      }
    }
  }
}
```

Apróbetűs

A HTML szabvány lehetőséget biztosít olyan fájlfeltöltő űrlapmezőre is, ami egyszerre több fájlt képes küldeni. Ebben az esetben a `$_FILES` megfelelő elemének minden tulajdonságot leíró mezője egy sorszámozott tömb lesz, mely sorban tartalmazza az egyes feltöltött fájlok adatait.

9.4 Feladatok

1. **Feladat**

   Készíts egy képgaléria alkalmazást, melyben lehetőség van új képek feltöltésére! A feltöltésnél ellenőrizd, hogy a feltöltött fájl mindenképp kép legyen, illetve azt is, hogy ne legyen a mérete több, mint 2 MB! A feltöltött képek jelenjenek meg egy galéria formájában!

2. **Feladat**

   Alakítsd át úgy a családi bevásárló-lista alkalmazást, hogy egy külön oldalon lehetőség legyen a rendszerbe a fontosabb számlák szkennelt változatát képként feltölteni, majd keresni közöttük dátum szerint! Ily módon sosem fognak elveszni a számlák!

Apróbetűs

A webszerver csak bizonyos fájlok kiszolgálására képes a szerver számítógépen. Hogy melyek azok a mappák, amiket kiszolgálhat, az a webszerver konfigurációjától függ. Ezeket a mappákat hívjuk "webroot" mappáknak. Ezzel szemben a PHP a szerver számítógép összes fájlját eléri (legfeljebb nincs jogosultsága az íráshoz-olvasáshoz). Ez az alapértelmezett beállítás módosítható a PHP beállításai között az `open_basedir` paraméter átállításával. Amennyiben az `open_basedir` adott, a PHP csak az ott megadott mappán belüli fájlokat látja.