

# Sigma

## konfigurációs beállítások

### 1. Url átirányítás.

Ahhoz, hogy az Apache2 engedélyezze az átirányítást első körben engedélyezni kell azt a apache konfigurációs állományában. Ezt Linux esetében a `/etc/apache2/sites-enabled/000-default.conf` konfigurációs állományban tehetjük meg, még pedig az alábbi ábra szerint.

Ugyan akkor elképzelhető, hogy eltérő operációs rendszerek esetében ez az alapértelmezett beállítás.

```
etc > apache2 > sites-enabled > 000-default.conf
1  <VirtualHost *:80>
2      # The ServerName directive sets the request scheme, hostname and port that
3      # the server uses to identify itself. This is used when creating
4      # redirection URLs. In the context of virtual hosts, the ServerName
5      # specifies what hostname must appear in the request's Host: header to
6      # match this virtual host. For the default virtual host (this file) this
7      # value is not decisive as it is used as a last resort host regardless.
8      # However, you must set it for any further virtual host explicitly.
9      #ServerName www.example.com
10
11     ServerAdmin webmaster@localhost
12     DocumentRoot /var/www/html
13
14     # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
15     # error, crit, alert, emerg.
16     # It is also possible to configure the loglevel for particular
17     # modules, e.g.
18     #LogLevel info ssl:warn
19
20     ErrorLog ${APACHE_LOG_DIR}/error.log
21     CustomLog ${APACHE_LOG_DIR}/access.log combined
22
23     <Directory /var/www/html>
24         Options Indexes FollowSymLinks MultiViews
25         AllowOverride All
26         Require all granted
27     </Directory>
28
29     # For most configuration files from conf-available/, which are
30     # enabled or disabled at a global level, it is possible to
31     # include a line for only one particular virtual host. For example the
32     # following line enables the CGI configuration for this host only
33     # after it has been globally disabled with "a2disconf".
34     #Include conf-available/serve-cgi-bin.conf
35 </VirtualHost>
36
37 # vim: syntax=apache ts=4 sw=4 sts=4 sr noet
38
```

A `<Directory ...></directory>` tagek közé felveszünk egy plusz sort, ami az url átírás engedélyezését állítja be: **AllowOverride All** Ügyeljünk rá, hogy a konfigurációs állomány **Case Sensitive**

**megj.:** A `000-default.conf` fájl átírása után a szervert újra kell indítani!

A 2. lépés, hogy a szervernek megmondjuk, hogy mely mappára irányuló kéréseket kell átirányítani, milyen fájlok/mappák kivételével és hova. Ezt az egyes mappákban elhelyezett `.htaccess` konfigurációs fájlba írjuk bele. A szerver minden kérés esetében megvizsgálja, hogy a hivatkozott mappa tartalmaz-e `.htaccess` állományt és, ha igen, beolvassa azt felülírja az alapértelmezett beállításait. A kérés kiszolgálása után a beállítások visszaállításra kerülnek a kérés előtti állapotba és egyszerű újabb kérésnél a folyamat kezdődik előlről.

```
.htaccess
1 | # Átirányítást végző motor bekapcsolása
2 | RewriteEngine on
3 |
4 | # Átirányításra vonatkozó kivétel hozzáadása
5 | RewriteCond %{REQUEST_FILENAME} !/Application/Style
6 |
7 | # átirányítás cél fájlja
8 | RewriteRule ^ index.php [L]
9 |
10 | # Mappák kilistázásának letiltása
11 | Options -indexes
12 |
13 |
14 |
```

























**Megj.:** A kommenteket `#` jel után írhatjuk.

- A 2. sorban bekapcsoljuk az url átírást
- Az 5. sorban az átirányítás alá veszünk fel kivételt. Ez azt jelenti, hogy ha ebben az esetben az **Application/Style** mappában lévő fájlra érkezik kérés a szerverhez, akkor ezeket kiszolgálja, vagyis közvetlenül továbbadja a kliens, vagy valamilyen értelmező felé.
- A 8. sorban megadjuk annak a fájlnak a nevét, amelyiket a szerver futtasson, vagy adjon át a kliens felé. Ha ez egy `.js/.html/.css/...` fájl, akkor közvetlenül elküldi a kérést félnek. Ha egy `.php` állomány, akkor átadja a php interpreternek, ami lefuttatja azt. Ha a megjelölt fájl egy programfájl, abban az esetben az URL és mindenféle, a szervertől kapott környezeti változó elérhetővé válik ( pl. `php` → `$_SERVER` )
- A 11. sorban a mappák kilistázását kapcsoljuk ki.

**Figyelmeztetés.:** A `.htaccess` fájl is Case Sensitive!!

### 1. Mappastruktúra.

/

-  .htaccess
-  index.php
-  favicon.jpeg
-  config.json
-  Doc /
  -  14SZ\_FW.pdf
-  Application /
  -  Core /
    -  controllers.php
    -  core.php
    -  functions.php
  -  Database /
    -  database.php
  -  Log /
    -  dberror.log
  -  Style /
    -  style.css
  -  Templates /
    -  \_404View.php
    -  \_errorView.php
    -  \_layout.php
    -  aboutView.php
    -  headerView.php
    -  homeView.php

#### Projekt mappa

*Konfigurációs állomány*

*Program belépési pont.*

*Böngésző fülön megjelenő képállomány*

*Adatbázis felhasználói adatok*

*Dokumentációk*

*Működési, konfigurálási leírás*

#### Alkalmazás mappa

Alkalmazás réteg (alkalmazás logika)

*Kontollerek*

*Routing funkció*

*Segédfüggvények*

*Adatbázis réteg (üzleti logika)*

*Adatbázis kezelő függvények*

*Naplók*

*Adatbázis műveletek naplózása*

*Stílus*

*Fő stílus állomány*

*Megjelenítési réteg*

*404*

*adatelérési hiba*

*oldalszerkezeti sablon*

*az oldalról*

*fejléc és navigáció*

*kezdőoldal*

## 2. Belépési pont, index.php

```
index.php > ...
1 <?php
2     define('APPPATH', 'Application/');
3
4     /**
5      * hibaüzenetek kimenetre írásának ki/be kapcsolása.
6      */
7     ini_set('display_errors', 0);
8
9     // Logolás szintje -> error level
10    error_reporting(E_ALL);
11
12    // Logolás be/ki kapcsolása
13    ini_set("log_errors", 1);
14
15    // Logfájl útvonala
16    ini_set("error_log", APPPATH."Log/sys.log");
17
18    /**
19     * Az APPROOT szükséges ahhoz, hogy amennyiben nem a webszerver
20     * gyökérmappájában lakik az alkalmazásunk, a routing helyesen működjön.
21     *
22     * Pl.: winsql.vereb.dc/diakXX/feladat/index.php
23     *      ekkor:          --> /feladat
24     */
25    define('APPROOT', '');
26
27    /**
28     * Az adatbázis kapcsolódásho szükséges adatokat tartalmazó json fájl.
29     */
30    define('CONFPATH', 'config.json');
31
32    // Könyvtárak behúzása
33    require_once APPPATH.'Core/functions.php';
34    require_once APPPATH.'Core/controllers.php';
35    require_once APPPATH.'Database/database.php';
36
37    /**
38     * A core.php-ban megyünk tovább.
39     */
40    require_once APPPATH.'Core/core.php';
41
```

- 2. APPPATH konstans deklarációja.
- 7. Beállításra kerül az interpreter, hogy HTTP Respons-ba, azaz a kimenetre írassa a hibaüzeneteket.
- 10. Logolás szintjének beállítása. <https://www.php.net/manual/en/errorfunc.constants.php>

- 13., 16. Logfájl engedélyezése és útvonal megadása.
- 25. Definiáljuk az APPROOT konstans. Ez minden esetben a szerver domain és az index.php közti útvonalat jelenti.
- 30. CONFPATH konstans. Az adatbázis felhasználó adatait tartalmazó json fájl elérési útja.

Példa 1: Nézzünk egy alap urlt:  
`winsql.vereb.dc/userXX/datum/feladat/index.php?p=231`

Ebben a következő részek rejlenek:

Domain, a host neve  
Útvonal, mely a lekért állományt tartalmazza  
A kért állomány neve  
Query string – get paraméterek

Ezen url részek közül a zöldeskék, azaz az útvonal értéke amit minden esetben aktualizálnunk kell az APPROOT konstansban.

Példa 2.: Egy otthoni localhost-os esetben, amikor a projekt a szerver gyökérmappájában van, az url az alábbi: `localhost/`. Ebben az esetben az APPROOT „” üres marad.

- 21-19. Meghívásra kerülnek az egyes állományok. Sorban a segédfüggvények, kontrollerek, adatbázis függvények. Ha ezek már mind elérhetőek, jöhet a core.php, ahol az url feldolgozás és útválasztás történik.

### 3. Alkalmazás logika

Core.php

```
Application > Core > core.php > ...
1  <?php
2
3  /**
4   * Az URL szétszedése. Első körben megtisztítjuk a ?-jel utáni
5   * query_string-től.
6   */
7  $uriPart = explode('?', $_SERVER['REQUEST_URI'])[0];
8
9  /**
10   * Második körben az APPROOT értékét kivágjuk az URL-ből.
11   */
12  $cleanedUri = str_replace(APPROOT, '', $uriPart);
13
```

- 7., 12. A szerver által átadott adatokat a `$_SERVER` szuper-globális tömbben érhetjük el. A `REQUEST_URI` tartalmazza a kérés url-t. Ebben a szekcióban levágjuk a végéről a *query\_string*-et és az elejéről az APPPATH-t. Így megkapjuk azt az értéket/elérési utat, amelyből információt nyerhetünk ki.



```

Application > Core > core.php > ...
14  /**
15   * Aktuális gyökérmappa visszakeresése és a visszalépés (STEPBACK)
16   * konstans definiálása. A példában a böngésző a /jhhj/Application/Style/style.css-t
17   * keresné, mert az url-ben a /jhhj/ az aktuális mappa, amihez hozzáfűzi
18   * a kapott path. Ezért nekünk ki kell számolni, hogy hány mappával feljebb található
19   * az Application mappa, hogy a kérés helyes útvonalra mutasson. Mivel az explode függvény
20   * a /jhhj/ stringet a / jelek mentén három részre osztja, de nekünk innen csak egyet kell
21   * visszalápnunk, ezért:
22   * Pl.: /jhhj/
23   * 0 => string '' (length=0)
24   * 1 => string 'jhhj' (length=4)
25   * 2 => string '' (length=0)
26   *
27   * ...a ciklusnak 2-től kell indulnia
28   */
29  $stepBack = '';
30  for($i = 2; $i < count(explode('/', $cleanedUri)); $i++)
31  {
32      $stepBack .= '../';
33  }
34  define('STEPBACK', $stepBack);
35  /**
36   * A $routes tömb tartalmazza majd az egyes útvonalakhoz tartozó
37   * kontrollereket.
38   */
39  $routes = [];
40
41  /**
42   * Az addRoute függvény végzi a regisztrációt.
43   * Ezen a ponton vesszük fel az egyes kontrollereket a kapott útvonalakhoz.
44   */
45  addRoute('/?', 'homeController');
46  addRoute('/about/?', 'aboutController');
47
48
49  /**
50   * A routing függvény végzi a kikeresést és a controller függvény meghívását.
51   */
52  routing($cleanedUri);

```

29 - 34. Szükséges a külön letöltendő állományoknak ahhoz, hogy pontos címet – url-t kapjanak egy konstans létrehozása. Ez STEPBACK -nek lett elnevezve. A mechanizmus lényege, hogy amikor a kliens lekéri az alkalmazás egy példányát a szerverről, a kérés útvonala sokszor nem a projekt gyökérmappájára mutat, hanem lejjebb a fájlstruktúrában.

Példa: localhost/Module/1

Mivel a projekt fájlstruktúrája állandó, a style.css mindig az alábbi cím alatt található meg.

/Application/Style/style.ccs

Vagyis a gyökérmappától 2-vel lejjebb. A böngésző amikor nem egy teljes „https://...” kezdetű címet kap a linkelésnél, hanem egy sima mappa útvonalat, akkor a kapott dokumentum, vagyis ebben az esetben „http://localhost/Module/1” url -nek aktuális könyvtárához fűzi azt hozzá. A példában a Module a munkakönyvtár.

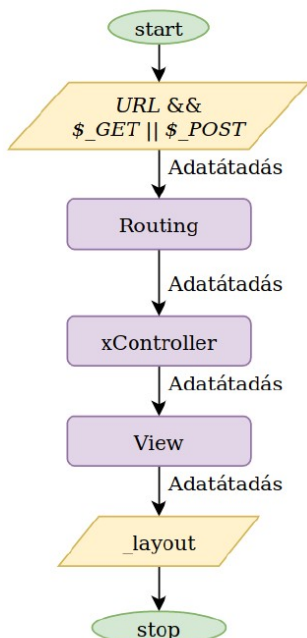
http://localhost/Module/Application/Style/style.css

Ebből következik, hogy ahhoz, hogy a stíluslapot a böngésző a helyes címről kérje le, elő kell állítani azt. A STEPBACK konstans minden esetben annyi mappa visszalépést fog tartalmazni (...), amennyire szükség van.

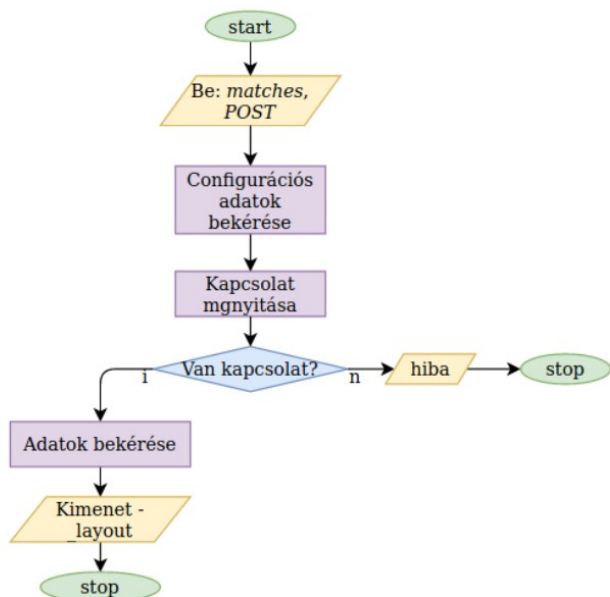
`http://localhost/Module/./Application/Style/style.css`

39. *\$routes* tömb deklarálása. Ez a tömb fogja tartalmazni a felregisztrált kontrollerek nevét és a regex mintát, amelyre meghívódnak.
- 45, 46. *HomeController*<sub>controllers.php</sub> és *AboutController*<sub>controllers.php</sub> felregisztrálása a programba. A regisztrációt az *addRoute*<sub>functions.php</sub> függvény végzi a rendszerben.
52. A kapott elérési útvonal alapján kiválasztásra kerül a felregisztrált controller függvény. Ha nincsen az útvonalra illeszkedő minta a *\$routes* tömbben, úgy a *notFoundController*<sub>controllers.php</sub> függvény hívódik meg.

A meghívott controller elvégzi az adatgyűjtést és/vagy módosításokat az adatbázis-kezelő függvények és segédfüggvények közreműködésével, majd meghívja a *view* függvényt, az adatokat tömbként, a *\_layout.php*-n keresztül átadja a nézeteknek, amelyek megjelenítik azokat.



Adatlekérdezéses megjelenítés. A controller nem végez módosítást az adatbázisban, így a vezérlés átadódik a *view* függvénynek (1).





## Segédfüggvények – functions.php

```
Application > Core > functions.php > PHP Intelephense > routing
2  /**
3   * Kiolvassa a konfigurációs állomány tartalmát és tömbben adja vissza.
4   *
5   * @param string $path Path of configuratin file
6   *
7   * @return array Content of readed file
8   */
9  function getConfig($confPath)
10 {
11     return json_decode( file_get_contents($confPath),true);
12 }
13
```

```
15 /**
16 * Az addRout függvény végzi a regisztrációt. A global kulcsszóval lehet elérni
17 * külső változót függvényen belülről
18 *
19 * @param string $pattern pattern of searched part of url
20 * @param string $controller Name of conteroller
21 *
22 * @return void
23 */
24 function addRoute($pattern, $controller)
25 {
26     global $routes;
27     $routes['%'.$pattern.'$'] = $controller;
28 }
29
```

Útvonal hozzáadása a `$routes` tömbhöz. A függvény delimiterek közé illeszti a kapott mintát és kiegészíti azt egy `^` és egy `$` karakterrel. Ezek határozzák meg, hogy a mintán kívül semmi más ne legyen az url-ben.

```

Application > Core > functions.php > PHP Intelephense > routing
30 /**
31  * A routing függvény végzi a kikeresést. Ha nem talál egyezést a tisztított url
32  * és a regisztrált minta között, meghívja a notFoundController-t.
33  *
34  * @param string $cleanedUri Part of url
35  *
36  * @return void
37  */
38 function routing($cleanedUri)
39 {
40     global $routes;
41
42     foreach($routes as $pattern => $controller)
43     {
44         if(preg_match($pattern, $cleanedUri, $matches))
45         {
46             $controller($matches);
47             return;
48         }
49     }
50     notFoundController();
51 }

```

A **routing** függvény ha megtalálta a *\$routes* ban tárolt minták közül az url-re illeszkedőt, meghívja annak értékéül megadott controller függvényt, különben a **notFoundController** hívódik meg.

```

54 /**
55  * A view függvény a kapott adatokat kibontja és átadja a layoutnak.
56  * A $datas-nak tartalmaznia kell egy 'view' és egy 'title' kulcsot!
57  *
58  * @param array $datas Datas from controller to view
59  */
60 function view($datas)
61 {
62     extract($datas);
63     require_once APPPATH.'Templates/_layout.php';
64 }

```

A **view** függvény a nézet megjelenítéséért felelős. Meg kapja az adott kontrollertől a megjelenítéshez szükséges adatokat, így azok elérhetőek lesznek a belinkelt nézetek számára is.

```

66  /**
67  * A kapott hibaüzenetet írja ki az Application/Log/dberror.log fájlba
68  * Ha a Log/ mappa nem létezik, a fájl létrehozása nem hajtódik végre.
69  *
70  * @param string $message Kiírandó hibaüzenet
71  */
72  function errorLog($message)
73  {
74
75      $path      = APPPATH.'Log/dberror.log';
76      $msg       = "[".date('Y-m-d H:i:s')."].".$message.PHP_EOL;
77
78      return file_put_contents($path, $msg);
79  }

```

Az `errorLog` függvény a logolást teszi egyszerűbbé. A kapott hibaüzenetet egy aktuális időbélyeg prefixel és egy sortöréssel egészíti ki.

## Alap kontrollerek - controllers.php

```

Application > Core > controllers.php > ...
1  <?php
2
3  function homeController($datas)
4  {
5      view([
6          "home" => 'active',
7          "title" => "- Home",
8          'view' => 'home'
9      ]);
10 }
11

```

Kezdőlapot kezelő controller

```

11
12 function aboutController($datas)
13 {
14     view([
15         "about" => 'active',
16         "title" => "- About",
17         'view' => 'about'
18     ]);
19 }
20
21

```

Oldalleírást kezelő controller

```
21
22 function notFoundController()
23 {
24     view([
25         "title" => "- Page Not Found",
26         'view' => '_404'
27     ]);
28 }
29
```

### Rossz címezést – 404 – kezelő kontroller

A kontrollereknek minden esetben át kell adniuk a `view` függvénynek két kulcsot: `title` és `view`. Ezeket a `_layout.php` várja. A `$view` változó függvénye, hogy melyik nézet kerüljön behúzásra a html törzsbe, míg a `$title` adja böngészőfél szövegét.