תוכן עניינים

מבוא ל-MVC

גם לך יש את ההרגשה הזו שכל שבוע צץ מדריך חדש באינטרנט על MVC וכל אחד מהם מספר משהו אחר לגמרי, והדבר היחידי שלא משתנה זו השאלות שנשארות בסוף?

"רגע, אז מה זה בעצם MVC ואיך אני משתמש בזה בקוד שלי?"

"זה לא סתם מאריך את הזמן שלוקח לפתח בצורה רגילה? זה יותר יעיל או נכון?"

"כתבתי את הקוד הזה. זה MVC תקין?"

אם אחת או יותר מהשאלות האלו עלו גם לך, קח 15 דקות לקרוא את הספר הזה במלואו ותצא עם כל הידע הדרוש כדי להתחיל לכתוב **מערכות גדולות בצורה מסודרת** בעצמך.

כן, היתרון העקרוני של MVC הוא **לעשות סדר בקוד** וזו הסיבה שרוב הפריימוורקים והאתרים שבנויים על בסיסם עובדים בצורה הזו. מעבר לזה, ה-MVC כיום כל כך פופולארי שמשתמשים בו לא רק ב-PHP ולא רק בשפות WEB אחרות, אלה גם לתוכנות חלונות, לאפליקציות מובייל ומה לא.

MVC היא אחת מתבניות העיצוב בעולם התכנות מונחה עצמים, וכמו רוב דפוסי העיצוב **המטרה של MVC היא לעשות סדר בקוד**.

אם האתר שלך זה שני סקריפטים ושלושה משתמשים, אתה בהחלט הולך להפסיד משימוש ב-MVC. הוא ייקח לך יותר זמן מאשר יוסיף תועלת ותוכל לחסוך לעצמך כמה שעות אם תשכח מ-MVC ובמקום זה תלך לשחק כדורגל. אם אתה בכל זאת רוצה להמשיך להתקדם או ללמוד משהו חדש – תמשיך איתי.

מרכיבי ה-MVC

ראשי התיבות MVC מתורגמות בתור Model — View — Controller
או בעברית מודל — מבט — בקר.

בהמשך אנחנו נסתכל על כל אחד מהחלקים האלו בנפרד, אבל בשביל לעבוד עם התבנית הזו בעולם ה-Web חסרים לנו עדיין כמה רכיבים. תבנית ה-MVC מגיעה אלינו מלפני למעלה משלושים שנה משפת Smalltalk, עוד לפני שהיה קיים הווב שאנחנו מכירים. והאמת היא ש:

מה שאנחנו קוראים לו MVC היום זה ניסיון להתאים את התבנית שהתאימה לתוכנות חלונות של אז - לעולם המודרני של היום. אבל יש כמה חלקים שבדרך כלל שוכחים להסביר בכל מני מדריכים, כמו:

"איפה נכנסת העבודה עם מסד נתונים?"

"מה אם יש יותר ממשתמש אחד במקביל להבדיל מתוכנת חלונות?"

"וזה שהתוכנה לא רצה בצורה קבוע והדפדפן יכול להיסגר מתי שבא לו?"

דווקא מהשאלות האלה אני רוצה להתחיל. בוא נצלול לתוך הרקע התיאורטי עם החלק שקיים כמעט בכל מערכת MVC – הראוטר (הנתב).

ה-Router

יצא לך לראות קוד כזה?

```
if ($_GET['do'] == 'add') {
    // ...
} else if ($_GET['do'] == 'edit') {
    // ...
} else if ($_GET['do'] == 'delete') {
    // ...
} else {
    // ...
}
```

או לתהות למה באתרים מסוימים הכתובות והקישורים נראים יפים ומסודרים ובאתרים אחרים הם מלאים בפרמטרים אחרי סימן השאלה וכנראה גם מצריכים קוד כמו זה שלמעלה?

יכול להיות שכבר יצא לך להתעסק ב-mod_rewrite וביצירת קישורים קריאים ומתאימים למנועי החיפוש, אבל אם לא, הרשה לי להסביר בשני שורות במה מדובר:

כשאתה גולש לכתובת site.com/login.php?action=register&from=facebook השרת מפעיל את הקובץ login.php והסקריפט מתחיל לפעול. אמנם במקרה הזה הקישורים נראים לא יפים במיוחד לעין או למנועי החיפוש, וקישור בדומה ל site.com/fbreg יכול להיות הרבה יותר נחמד.

כדי להשיג את התוצאה הזו, משתמשים בטריק נחמד, מבקשים מהשרת להעביר את כל הבקשות הנכנסות אל הקובץ index.php. לא משנה לאיזו כתובת גלש המשתמש, תמיד יופעל הסקריפט index.php ובתוכו תהיה לנו שליטה רבה יותר על הפעולות הבאות.

נסה בעצמך: שים את חמשת השורות הבאות בקובץ htaccess. (קובץ בלי שם והסיומת htaccess) והפעל בשרת את mod_rewrite ברשימת המודולים אם הוא עדיין לא מופעל בתצורת ברירת המחדל ב-wamp.

```
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# otherwise forward it to index.php
RewriteRule . index.php
```

מעכשיו כל כתובת שתזין בדפדפן תגיע אל הסקריפט index.php, למעט גלישה ישירה לקובץ שקיים פיזית על השרת. מה שיישאר לך הוא להחליט בתוך סקריפט ה-index מה הפעולה שתרצה לעשות, על בסיס הקלט שהגיע.

סקריפט ה-index.php ישמש אותנו בתור נתב, זה שמקבל את כל הבקשות ומחליט מה לעשות איתם ואילו פונקציות להפעיל בתוך המערכת. זהו הסקריפט היחידי שאמורה להיות גישה אליו דרך הדפדפן. זה אומר שאת שאר הסקריפטים אפשר לשים בתיקיה אחרת ולחסום גישה אליהם מהדפדפן.

הניתוב יכול בהחלט להיראות כמו בדוגמה למעלה, בצורה כזו:

```
if ($_GET['do'] == 'add') {
    // ...
} else if ($_GET['do'] == 'edit') {
    // ...
} else if ($_GET['do'] == 'delete') {
    // ...
} else {
    // ...
}
```

או למשל ניתוב על בסיס כתובות:

דוגמה מתוך Slim Framework

```
get('/hello/:name/:surname', function ($name, $surname) {
    echo "Hello, $name $surname";
})

post('/login', function () {
    echo $_POST['email'];
})
```

אם אתה מרגיש מצב רוח קרבי, אתה מוזמן לנסות לממש את הפונקציות get ו-post בעצמך ולהיעזר בחברי הפורום באתר phpguide.co.il כדי לעשות את זה. לאחר שתסיים, קח דקה ונסה להבין איפה כאן הבעיה. איך במקרה הזה יראה פרויקט קצת גדול יותר משני פונקציות?

עם ניתוב כזה, לא רק שלא קיבלנו סדר בקוד של הפרויקט, אלא להיפך: כל הקוד של האתר נמצא עכשיו בקובץ אחד ענק ומסורבל. כדי לפתור את הבעיה הזו אנחנו נפריד חלקים שונים של הפרויקט למחלקות נפרדות ונעביר את הטיפול אליהם.

המחלקות האלה ייקראו קונטרולרים (בקרים).

ה-Controller

ה**קונטרולר** – או ה**בקר** בעברית – הוא החלק שמקבל את הנתונים (במקרה הזה מהנתב) ומחליט מה לעשות איתם. הבקר כאן לא שונה בהרבה מהבקר שיש לך במיקרוגל. המטרה שלו היא לקבל פלט מכל מני מקומות, כמו לחיצות כפתורים שלך או טמפרטורה בתוך המיקרו ולהציג משהו מתאים על הצג או לזרז את המהירות של הצלחת המסתובבת.

בתור דוגמה ניקח טיפול בהזדהות ובהרשמה וניצור מחלקה עם החתימה הבאה:

```
interface ILoginRegController
{
    function actionShowLoginForm();
    function actionShowRegForm();

    function actionCheckLogin();
    function actionRegister();
}
```

יצירת כל מחלקה אנחנו מתחילים מיצירת ממשק (Interface) תואם. זהו אחד הרעיונות החשובים ביותר בעולם התכנות מונחה עצמים.

הממשק שלנו מגדיר ארבע פעולות שלהן תהיה המחלקה אחראית: הצגת טופסי ההזדהות וההרשמה וטיפול בכל אחד מהטפסים שנשלחו לשרת.

כדי להקל עלינו בעתיד, החלטתי להוסיף מוסכמה לקוד: להתחיל ב-`action` את שמות המתודות אליהן מנותבות בקשות מהנתב. באופן כללי, אין חובה להתחיל את שמות הפעולות במילה `action`, אך בעתיד זה יחסוך לך זמן כשתוכל במבט מהיר להבין אילו פעולות עושות מה.

מוסכמה נוספת היא להוסיף את המילה `Controller` לסוף שמם של המחלקות שמתפקדות בתור בקרים, כלומר כאלו, שמכילות את הפעולות אליהם מבוצע הניתוב, ולא כאלה שעסוקות בחיבורים למסדי נתונים.

מעבר לפעולות של הקונטרולר עלינו להחליט גם באיזו כתובת הקונטרולר יהיה נגיש. לרוב, מקובל שהכתובות יורכבו משם הקונטרולר, שם הפעולה בתוך הקונטרולר ולאחריה פרמטרים נוספים.

באופן הזה אל טופס ההזדהות ניגש ב-site.com/loginReg/showLoginForm ואת שליחת טופס ההרשמה נבצע אל-site.com/loginReg/register.

מחברים בין הקונטרולר לבין הניתוב

בשלב הזה הדרישות אל הנתב שלנו עולות שלב. לא רק שהוא צריך לפענח את הכתובת, אלא גם ליצור מופע חדש של מחלקת הקונטרולר המתאימה ולהפעיל בתוכה את המתודה המתאימה. לפעמים הוא גם יצטרך לפענח נתונים מהכתובת ולהעביר אותם בתור פרמטרים למתודות בתוך הקונטרולר.

בשלב מאוחר יותר אולי נרצה גם כתובות שלא בדיוק תואמות למבנה של שם הקונטרולר – שם הפעולה, אלא משהו יותר מתאים למנועי החיפוש וכאן נצטרך לכתוב נתב משלנו או להוריד אחד מוכן.

הדרך המועדפת עלי לקשר בין נתיבים לקונטרולרים היא בצורת מערך, כמו בדוגמה הזו:

```
[
    'posts/<id:\d+>'                          => 'posts/redirectById',
    '<article_url:[a-z0-9]+>.htm'             => 'posts/index',
    ''                                        => 'homepage/index',
    'users/<username:[a-z0-9]+>'              => 'users/user',
    '<controller:[a-z]+>/<action:\w+>'        => '<controller>/<action>'
]
```

כאשר מצד שמאל הכתובת שהמשתמש הזין, יחד עם ביטוי רגולרי שמוציא פרמטרים מהכתובת ומעביר אותם לפעולה מסוימת בתוך הקונטרולר עם ערך מתאים.

יש המעדיפים ניתוב בסגנון אחר, באמצעות פונקציות:

```php
class ProductController {
    public function listAction() {
        return 'product list';
    }
    public function itemAction($id) {
        return "product $id";
    }
}
$mux = new Pux\Mux;
$mux->any('/product', ['ProductController','listAction']);
$mux->get('/product/:id', ['ProductController','itemAction'] , [
    'require' => [ 'id' => '\d+', ],
    'default' => [ 'id' => '1', ]
]);
$mux->post('/product/:id', ['ProductController','updateAction'] , [
    'require' => [ 'id' => '\d+', ],
    'default' => [ 'id' => '1', ]
]);
$mux->delete('/product/:id', ['ProductController','deleteAction'] , [
    'require' => [ 'id' => '\d+', ],
    'default' => [ 'id' => '1', ]
]);
$route = $mux->dispatch('/product/1');
Executor::execute($route);
```

תוכל לנסות לכתוב מערכת ניתוב כזו בעצמך או להוריד מערכת ניתוב מוכנה. אתה יכול לנסות את אחד מספריות הניתוב המהירות הבאות:

- Fast Route
- Pux

מצא את ההבדלים

מצא את ההבדלים בין הקוד באחת הגרסאות הישנות של WordPress:

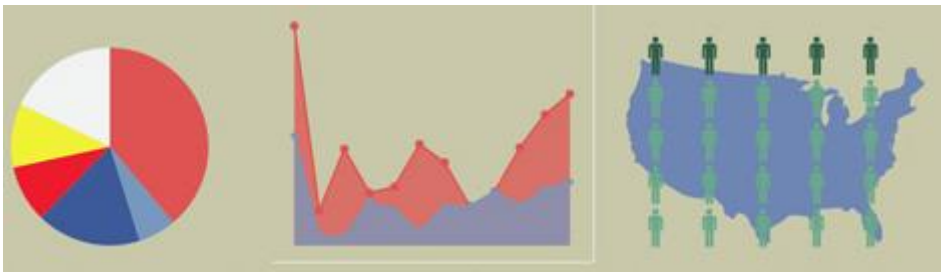
```
<?php
$user_login = $HTTP_POST_VARS["user_login"];
$pass1 = $HTTP_POST_VARS["pass1"];
$pass2 = $HTTP_POST_VARS["pass2"];

/* checking login has been typed */
if ($user_login=='') {
    die ("<b>ERROR</b>: please enter a Login");
}

mysql_select_db("$base") or die ("<b>OOPS</b>: can't select the database $base : ".mysql_error());

/* checking the login isn't already used by another user */
$request =  " SELECT user_login FROM $tableusers WHERE user_login = '$user_login'";
$result = mysql_query($request,$id) or die ("<b>OOPS</b>: can't check the login...");
$lines = mysql_num_rows($result);
mysql_free_result($result);
if ($lines>=1) {
    die ("<b>ERROR</b>: this login is already registered, please choose another one");
}

$message  = "new user registration on your blog $blogname:\r\n\r\n";
@mail($admin_email,"new user registration on your blog $blogname",$message);

?><html>
<head>
<title>b2 > Registration complete</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" href="<?php echo $b2inc; ?>/b2.css" type="text/css">
<style type="text/css">
<!--
<?php
if (!preg_match("/Nav/",$HTTP_USER_AGENT)) {
?>
textarea,input,select {
    background-color: #f0f0f0;
    border-width: 1px;
    border-color: #cccccc;
    border-style: solid;
    padding: 2px;
    margin: 1px;
}
<?php
}
?>
-->
</style>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cccccc" vlink="#cccccc" alink="#ff0000">
```

לבין הדבר הזה:

רמז: הם שניהם ברדק לא נורמאלי, אבל רק אחד מהם תרצה בצלחת שלך.

בקוד למעלה קיים סלט מטורף בין בדיקת קלט, פלט של שגיאות, חיבור למסד נתונים, עיצוב CSS, עדכון טבלאות ותגי HTML. קוד כזה קשה להבין מהצד, עוד יותר קשה לתחזק ולשנות בעתיד. תכפיל את זה בכמות הקבצים שיש בפרויקט גדול ואפשר לקפוץ מהחלון כבר עכשיו. וזה עוד לפני שהבוס ביקש לעשות שההזדהות למעלה תעבוד ב-AJAX.

כדי לעשות קצת סדר, אנחנו הולכים להפריד בין הנתונים הגולמיים שיש לנו לבין צורת התצוגה שלהם. במילים אחרות, את אותם הנתונים אפשר להציג בכמה דרכים שונות:



גם בעולם ה-Web עומדות לרשותנו צורות תצוגה שונות. ברוב המקרים הפלט של הסקריפטים שלנו יהיה תוכן HTML כלשהו, אך לא תמיד. לפעמים הסקריפטים שלנו

יציירו גרפים או תמונות, כמו CAPTCHA, או יחזירו XML דוגמת RSS Feed. הרבה פעמים הסקריפטים שלנו יחזירו JSON בעבור קריאות AJAX או אולי ישלחו אימיילים. בסופו של יום, בכל אחד מהמקרים האלה, אותם הנתונים בדיוק מוכנסים לתבניות תצוגה שונות.

מודלים ותבניות תצוגה

בואו נניח שמערך המשתמשים הבא הגיע מתוך מסד הנתונים ומייצג את המשתמשים במערכת שלנו.

```
$users = [
    new User('Alice',   11),
    new User('Bob',     53),
    new User('Claudio', 52),
    new User('Denzel',  31),
]


class User
{
    public $name;
    public $age;

    public function __construct($name, $age)
    {
        $this->name = $name;
        $this->age = $age;
    }
}
```

המערכת שלנו צריכה לדעת להציג את המשתמשים באתר בתור עמוד HTML עם טבלה או בתור קובץ אקסל שנייצר בפורמט CSV. במילים אחרות, את אותם הנתונים אנחנו צריכים להציג בשני דרכים שונות, כלומר להכניס לשני תבניות שונות. התבניות האלו יכולות להיראות ככה:

תבנית תצוגת משתמשים מבוססת HTML

```
<html>
    <table>
        <? foreach($user in $users): ?>
            <tr>
                <td> <?= htmlSpecialChars($user->name, ..)?> </td>
                <td> <?= htmlSpecialChars($user->age, ...)?> </td>
            </tr>
        <? endforeach; ?>
    </table>
</html>
```

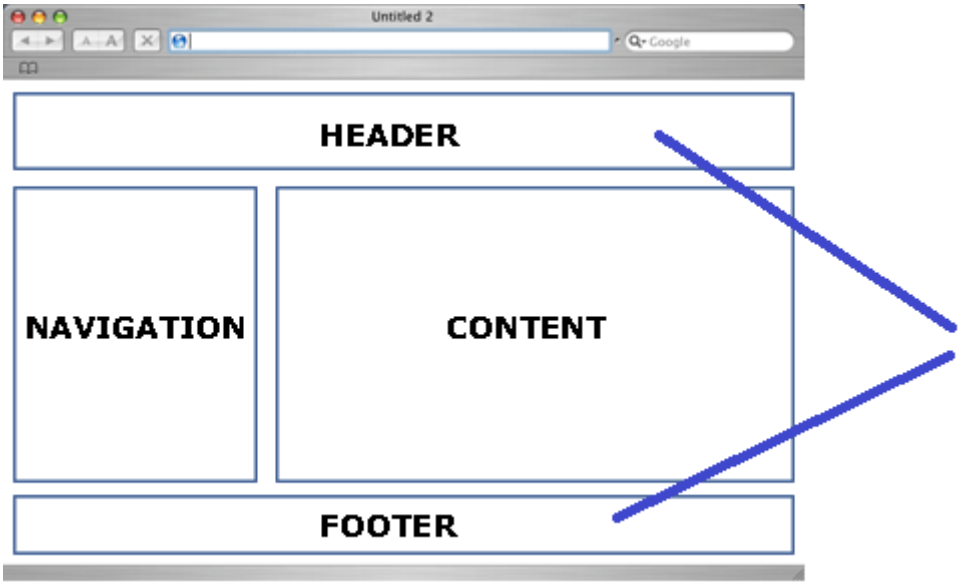
תבנית תצוגת משתמשים CSV לאקסל

```
name,age
<?
    foreach($user in $users)
        echo $user->name, $user->age, "\r\n";
```

בשלב מאוחר יותר תוכל להוסיף תבניות תצוגה נוספות, למשל להחזיר את רשימת המשתמשים בפורמט JSON או גרף התפלגות לפי גילאים.

לפני שאתה מחליט שאתה בחיים לא מייצא שום דבר לאקסל

אם אתה חושב שבאתר שלך אתה תמיד תחזיר רק HTML ואתה לא באמת צריך עשר תבניות שונות של XML, JSON, CSV, וורד ומה לא – **אתה כנראה צודק**. אבל, גם באתר שלך **אותם הנתונים** יכולים להיות מוצגים בכמה מקומות שונים באתר בצורה קצת שונה:



ולא רק שתוכל לנצל את אותם הנתונים בתוך תבניות שונות, אלא גם תוכל לנצל את אותה התבנית להצגת נתונים שונים. פעם את רשימת המשתמשים הכללית, פעם רק את המשתמשים שביקרו שבוע שעבר ממוינים לפי גיל או אפילו את רשימת הרכישות האחרונות מהאתר. ככה שהנתונים עצמם מופרדים לגמרי מהתבנית שבה הם מוצגים.

במונחי `MVC` למערך של המשתמשים למעלה קוראים `model`, שזהו בסך הכול אוסף הנתונים שמועברים לתצוגה. לתבניות התצוגה עצמן קוראים `views`. לא מסובך מדי.

מה הקשר בין MV ל-C

ראינו שהמודלים הם הנתונים שיש להעביר למבט. קל לנחש שכל מה שנשאר לקונטרולר לעשות זה לייצר או לשלוף את הנתונים ולהכניס אותם למבט המתאים. נמשיך עם דוגמת קונטרולר ההזדהות שלנו:

```
class LoginController extends BaseController implements ILoginController
{
    public function actionShowLoginForm()
    {
        $view = 'views/loginform.php';
        $model = null;

        $this->showView($view, $model)
    }

    public function actionCheckLogin()
    {
        $view = 'views/loginform.php';
        $model = [];

        if($_POST['login'] !== 'Jo')
            $model['error'] = 'Wrong username!';

        elseif ($_POST['pass'] !== 'qwerty')
            $model['error'] = 'Wrong password!';

        else
            ....
            return redirect to 'home/main';

        $this->showView($view, $model)
    }
}
```

המטרה של הקונטרולר במקרה הזה היא לייצר את הנתונים המתאימים ולהציג אותם בתבנית המתאימה. בשני המקרים התבנית היא טופס ההזדהות של האתר, כאשר פעם אחת הוא מוצג לבד ופעם אחת מוצג יחד עם הודעות שגיאה.

התבנית מתוך הקובץ `views/loginform.php` יכולה לקבל את הצורה הבאה:

```
<form method="POST" action="login/checkLogin">
    <?php
        if(isset($error) && !empty($error))
            echo "<div class='err'>", $error, "</div>";
    ?>

    <input type="text" name="login" />
    <input type="password" name="pass" />
    <input type="submit" />
</form>
```

במקרה הזה המשתנה error הוא חלק מהמודל, מהנתונים שהועברו על ידי הקונטרולר אל המבט. התבנית עצמה אחראית לתצוגה בלבד. בצורה דומה, קונטרולר אחר יכול לשלוף את רשימת המשתמשים שכרגע באתר מהמסד ולהציג אותם בתבנית אחרת.

רגע, מה זה showView?

את המתודה showView ירשנו ממחלקת האב, BaseController. המטרה שלה לקבל את שם התבנית ואת הנתונים שיש להעביר לה. אתה יכול לנסות לממש אותה בעצמך אם אתה רוצה, אבל ברוב המקרים המימוש שלה הוא כזה:

```
protected function renderView($_viewFile_, array $_model_)
{
    // we use special variable names here to avoid conflict when extracting
    if(is_array($_model_))
        extract($_model_);
    else
        $data=$_model_;

    ob_start();
    ob_implicit_flush(false);
    require($_viewFile_);
    return ob_get_clean();
}
```

הפונקציה extract מקבלת מערך ויוצרת מהמפתחות שלו משתנים בצורה הבאה:

```
$arr = ['key1' => 'val', 'jo' => 'halla'];
extract($arr);
echo $key1, '-', $jo; // val-halla
```

ככה שבזמן ביצוע ה-require לקובץ התצוגה, יש לו גישה למשתנים החדשים שנוצרו.

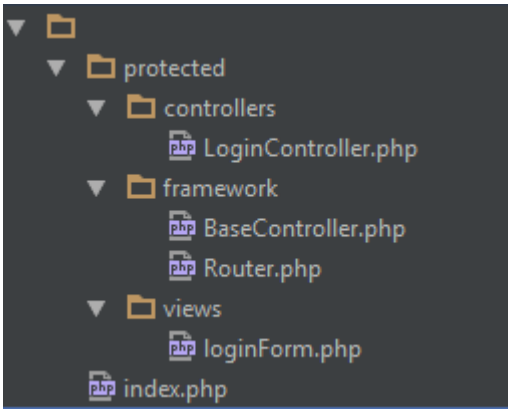
הרחבה נוספת שכדאי לעשות למתודה הזו היא להציג לא רק התבנית הספציפית שנדרשה, אלא להציג את גם את התבנית הכללית של האתר, עם כל התפריטים, עיצובים והחלקים האחרים.

שני דברים נובעים מהקוד למעלה

א. מחלקת אב, בתור בונוס, מאפשרת לנו לא רק לגשת אל מתודות שכל קונטרולר צריך, אלא גם מאפשרת לעשות דברים מעניינים נוספים, כמו למשל להגביל גישה למתודות מסוימות עבור משתמשים לא מזוהים.

ב. עולה כאן מבנה תיקיות מסוים.

מבנה תיקיות ל-MVC



מבנה תיקיות לדוגמה במערכת שבנויה בצורת MVC. הקובץ index.php הוא הקובץ היחידי שנגיש לדפדפן. שאר הקבצים נמצאים בתיקיה protected ומוגבלים לגישה מבחוץ.

בפנים נמצאות שלוש התיקיות שמכילות את הקונטרולרים שאתה יוצר, את התבניות שאתה יוצר ותיקיה נוספת של עזרי תמיכה, כמו מחלקת קונטרולר בסיס ומחלקת ניתוב.

שים לב שאין כאן בפירוש תיקיית models או תיקיה למחלקות BL כלשהן, למרות שבהרבה פרויקטים ניתן לראות תיקיה כזו. לרוב, מה שנמצא בתיקיה הזו, אלו הם מחלקות או מסוג DTO, כאלה שרק מכילות נתונים, או מחלקות שפריימוורק כזה או אחר מחבר למסד, בהתאם לפריימוורק עצמו.

בכדי לא לסבך את עצמך סתם, כרגע תוכל להתעלם לגמרי מדקויות מימוש של פריימוורקים שונים. בתור התחלה תוכל אפילו לבצע את החיבור למסד נתונים מתוך הקונטרולר (למרות שזה רעיון גרוע לעשות את זה בעולם תכנות מונחה העצמים ויש דרכים הרבה יותר טובות).

כרגע זה לא הזמן להיכנס לדרכי פיתוח נכון בתכנות מונחה עצמים ובינתיים יש לך ידע מספק בשביל לבנות מערכת בצורת MVC בעצמך. כל מה שנשאר לך הוא לנסות.

לסיכום

המטרה של MVC היא לעשות סדר בקוד. ככל שהפרויקט גדול יותר, ההפרדה שלו לחלקים קטנים ומסודרים היא משמעותית יותר.

בוא נראה מה למדת בינתיים:

1. הראוטר (נתב) מאפשר לך שליטה על נתיבים וקישורים באתר, עקב זה שהוא מקבל את כל הבקשות הנכנסות ומעביר אותן לביצוע של הקוד המתאים.

2. את הקוד הפרדת לקונטרולרים (בקרים) שונים, כאשר כל קונטרולר אחראי לחלק קטן של האתר, כשהוא מקבל את הקלט המתאים ובהתאם לזה מחליט אילו נתונים להציג ובאיזו צורה.

3. למדת לעבוד עם תבניות שונות, להעביר אליהם מידע ולהציג אותם.

ועל כל זה דיברנו כאן בספרון הזה.

עכשיו תורך לנצל את הידע החדש

בלי לנסות הלימוד לא שווה הרבה. צא לדרך ונסה לממש בפועל את מה שלמדת!

ואם אהבת החומר בספר הזה, אני רוצה שתעשה את הדבר הבא:

1. שלח לי אימייל ל-mail+mvcb@phpguide.co.il עם משהו בסגנון: "אלכס, קראתי את הספרון שלך על MVC. אהבתי."

2. אם יש נושאים נוספים שהיית רוצה ללמוד רשום גם אותם.

3. טפח לעצמך על השכם, קראת ספר שלם על MVC. (: