به نام یگانه برنامه‌نویس هستی

مستندات هسته

نسخه 1.1

# فهرست مطالب

[فهرست مطالب 2](#_Toc408088351)

[بررسی اجمالی 4](#_Toc408088352)

[ویژگی‌های کلیدی 4](#_Toc408088353)

[اصول اولیه 5](#_Toc408088354)

[بررسی اجمالی لایه‌ها 5](#_Toc408088355)

[افزودن ماژول جدید 7](#_Toc408088356)

[ماژول مستقل 7](#_Toc408088357)

[ماژول وابسته 8](#_Toc408088358)

[بررسی کامل فایل‌های هر ماژول 10](#_Toc408088359)

[مشترکات فایل‌ها 10](#_Toc408088360)

[پیشوند‌ها 10](#_Toc408088361)

[مدل 11](#_Toc408088362)

[اتصال به دیتابیس 13](#_Toc408088363)

[ساخت کوئری خصوصی 15](#_Toc408088364)

[ساخت کلاس کوئری عمومی 15](#_Toc408088365)

[ویو 16](#_Toc408088366)

[کنترلر 18](#_Toc408088367)

[دیسپلی 18](#_Toc408088368)

[کار با فرم‌ها 19](#_Toc408088369)

[اقدامات اولیه 19](#_Toc408088370)

[تعیین ویژگی های هر کنترل 20](#_Toc408088371)

[اعتبارسنجی 21](#_Toc408088372)

[ساخت فرم 22](#_Toc408088373)

[ساخت یک فرم عادی 22](#_Toc408088374)

[ارسال اطلاعات 24](#_Toc408088375)

[ساخت فرم عمومی 24](#_Toc408088376)

[ساخت فرم خصوصی 26](#_Toc408088377)

[کار با کتاب‌خانه فرم‌ساز 26](#_Toc408088378)

[ساخت فرم از طریق متدهای پیش‌تعریف 28](#_Toc408088379)

[ساخت فرم از طریق دیتابیس 28](#_Toc408088380)

[کنترل صفحه 29](#_Toc408088381)

[روتر 29](#_Toc408088382)

[قوانین مقادیر ورودی روتر 29](#_Toc408088383)

[انتقال دهنده صفحه 31](#_Toc408088384)

[روش معرفی توابع 32](#_Toc408088385)

# بررسی اجمالی

برای راه اندازی پروژه با زبان پی‌اچ‌پی نیاز به یک سری نیازهای اولیه داریم که این نیازها در قالب هسته پیش رو برطرف گردیده است. به جهت حجم پروژه و نیاز برای توسعه و نگهداری آن، هسته به صورت چند لایه بر مبنای الگوی معماری [[1]](#footnote-1)MVC طراحی گردیده است. این الگو منطق برنامه را از رابط کاربری به طور کامل جدا ساخته و به توسعه دهندگان این امکان را می‌دهد که توسعه، تست و نگهداری نرم‌افزار را در واحدهای مستقل انجام دهند.

هم‌چنین با توجه به نیاز پیش بینی شده برای ارتباط سیستم با خارج از آن، سیستم بر مبنای api بنا شده است. بدین معنا که هر گونه عملیات انجام شده توسط سیستم، قابلیت انجام از طریق خارج از آن را با داشتن سطح دسترسی‌های مورد نیاز دارد.

علاوه بر این به دلیل سبک طراحی صورت گرفته، سیستم قابلیت افزودن کلاس‌های مورد نیاز توسط برنامه نویسان یا جایگزینی کلاس‌های موجود توسط آنان را در بخش های مورد نیاز داراست.

## ویژگی‌های کلیدی

برترین ویژگی‌های هسته به شرح زیر است.

* طراحی بر مبنای معماری MVC
* قابلیت استفاده به عنوان api
* طراحی سیستم فرم ساز
* طراحی سیستم اعتبار سنجی خودکار
* دارای سیستم یکپارچه برای اتصال به دیتابیس
* امنیت بالا
* سریع و سبک

# اصول اولیه

در تعریف ما، کلاس به معنی یک آدرس بعد از نام دامنه در مروگر است. برای مثال آدرس زیر را در نظر بگیرید که اکانت به عنوان یک کلاس شناخته می‌شود.

test.com/account

در ادامه آدرس بعد از کلاس به عنوان ماژول شناخته می‌شود. آدرس زیر را در نظر بگیرید که لاگین به عنوان ماژول شناخته می‌شود.

test.com/account/login

در این مثال لاگین به عنوان یک ماژول در نظر گرفته می‌شود که میتواند به صورت مستقل یا وابسته باشد. تمام ماژول‌ها باید در پوشه content موجود در پروژه قرار گیرند. به همین دلیل از این پس آدرس ها بر اساس این روش نمایش داده خواهند شد.

ساختار پوشه بندی پیش فرض با وجود ماژول اکانت که شامل ماژول لاگین است، به شکل زیر خواهد بود.

Content/home

Content/account/home

Content/account/login

پوشه home همان index است، یعنی در صورتی که هیچ چیزی وجود نداشته باشد، کدهای موجود در این پوشه خوانده می‌شود. برای ماژول اکانت پوشه account/home و برای ماژول لاگین پوشه account/login خوانده می‌شود. در صورتی که فایل‌های مورد نیاز در هر یک از ماژول‌ها وجود نداشته باشد، به ترتیب به یک پوشه بالاتر مراجعه می‌شود و در این مدل پوشه home به عنوان آخرین آدرس مورد بررسی قرار می‌گیرد.

## بررسی اجمالی لایه‌ها

هر ماژول می تواند باید شامل لایه های زیر باشد که در ادامه به تشریح آن می‌پردازیم.

model.php

view.php

controller.php

display.html

**مدل**

این لایه تمامی پروسه‌ها و عملگرهای سمت سرور از قبیل ایجاد و حذف فایل، فولدر، اتصال به پایگاه داده ها. تغییر سایز عکس ها و ... را کنترل می کند. توابعی که در مدل ساخته می‌شوند را می‌توان در ویو و کنترلر فراخوانی کرد. اما بهترین راه فراخوانی استفاده از روترهاست.

مدل لایه‌ای برای ارتباط با دیتابیس و ذخیره سازی داده‌های دائمی است. نکته مهم این است که این بخش از اتفاقاتی که در بخش های کنترلر و ویو می افتد هیچ اطلاعاتی ندارد. این لایه به دنبال پاسخ از دیگر لایه‌ها نیست و تنها هدف آن ذخیره سازی دائمی داده ها یا آماده سازی آنها برای انتقال به لایه‌های دیگر است.

هم‌چنین این لایه تنها زمانی فراخوانی می‌شود که دستور post به صفحه ارسال شود و یا نیاز داشته باشیم در ویو از کوئری خاصی استفاده کنیم، قابل توجه است که اگر قصد هدایت به این لایه را داشتیم و دستور ما از نوع post نبود، می‌توان یک post مجازی ایجادکرد.

**ویو**

این لایه داده ها را از مدل درخواست کرده و داده های نهایی را برای نمایش در خروجی نمایش آماده می‌کند. به عبارت دیگر این لایه تنظیمات پویا جهت آماده سازی حالت نمایشی و ارسال به فایل display.html را بر عهده دارد. این لایه به کمک فایل دیسپلی عمل می‌کند و اگر از روترها استفاده کرده باشید به صورت اتوماتیک داده‌های بازگشتی مدل به تابع های خاص این کلاس ارسال می شود. در این لایه می‌توان تمامی حالت های نمایشی صفحه را کنترل کرد و به عهده گرفت.

**کنترلر**

فایل مادر هر ماژول است، و بودنش به معنی استقلال ماژول به حساب می‌آید. اگر ماژولی دارای این فایل نباشد به معنی عدم استقلال ماژول است و نمی توان از آدرس های تعریف شده استفاده کرد، به گونه دیگر نمی‌توان گفت که پوشه ما ماژول است بلکه حکم زیرماژول را دارد. اگر ماژولی دارای این فایل نباشد از کنترلر اولین ماژول home استفاده می‌کند. به عنوان مثال ماژول user/logout دارای فایل controller نیست. پس سیستم در فایل user/home به دنبال کنترلر می‌گردد و اگر این پوشه نیز وجود نداشت یا فایل controller نداشت به پوشه home منتقل می‌شود و از آن پوشه کنترلر را درخواست می‌کند.

همانگونه که از اسم این فایل مشخص است، وظیفه این فایل به عهده گیرنده پردازش‌های اصلی و کنترل کننده فایل است، از قبیل توابع مشترک در model, view و کنترل سطوح دسترسی، کنترل آدرس‌بار صفحه و دیگر خصیصه‌های اصلی. در هنگام فراخوانی یک صفحه قبل از هر اتفاقی کنترلر فرخوانی می‌شود.

**دیسپلی**

این فایل که با پسوند html ذخیره می‌شود تم و حالت نمایشی صفحه را به عهده می‌گیرد که با ساختار نوشتاری twig به متغیرهای view متصل می‌شود.

# افزودن ماژول جدید

اگر درون پوشه مربوطه به ماژول فایل controller.php وجود داشته باشد، آن ماژول به عنوان یک ماژول مستقل شناخته شده و در غیر این‌صورت در تعریف ما یک ماژول وابسته است. هر ماژول مستقل علاوه بر فایل controller.php که اجباری است و مشخص کننده استقلال ماژول است می تواند فایل‌های زیر را نیز داشت باشد.

model.php

view.php

controller.php

display.html

پس از درخواست کاربر به هر صفحه اولین لایه‌ای که مورد خوانده می‌شود، کنترلر است. سلسه مراتب دسترسی به کنترلر در پوشه ها به شرح زیر است.

content/account/login/controller.php => content/account/login/controller.php

content/account/login/no controller! => content/account/home/controller.php

content/account/home/no controller! => content/home/controller.php

content/home/no controller! => bigbang :(

## ماژول مستقل

اطلاعات هر ماژول درون یک پوشه با نام دلخواه ذخیره می‌شود. البته می‌توان برای سهولت کار ماژول ها را دسته‌بندی کرد تا چند ماژول درون یک پوشه قرار گیرد. با توجه به نیاز موجود، ما می‌توانیم تا دو ماژول مستقل تو در تو داشته باشیم و از آدرس سوم به بعد باید از طریق ماژول وابسته پیاده سازی شود.

برای مثال اگر شما ماژولی برای ورود کاربران دارید، می‌توانید به دو صورت زیر عمل کنید.

content/login

content/account/login

از این به بعد تمامی توابع سازنده (constructor) که در php با نام \_\_construct نامگذاری می شود، را فراموش کنید، چون ما زودتر از شما این کلاس ها را اشغال کردیم و برای شما تابع سازنده ای با نام config رزور کردیم که می‌توانید از آن استفاده کنید و عدم استفاده از این تابع یعنی نیاز به سازنده ندارید، این کلاس کار پیکربندی را انجام می دهد که بعدا بیشتر با آن آشنا می شویم.

برای ساخت یک ماژول مستقل، یک پوشه درون پوشه content پروژه خود ساخته و چهار فایل زیر را درون آن قرار دهید.

/content/home/controller.php

<?php

namespace content\home;

class controller extends MVCcontroller{

function config(){

}

}

?>

/content/home/view.php

<?php

Namespace content\home;

class view extends MVCview{

function config(){

}

}

?>

/content/home/model.php

<?php

namespace content\home;

class model extends MVCmodel{

function config(){

}

}

?>

/content/home/display.html

{% extends layout %}

{%block content%}

{%endblock%}

## ماژول وابسته

ماژول وابسته بدین معناست که این ماژول برای اجرا نیاز به یک ماژول دیگر دارد و به تنهایی قابل اجرا نیست. در ماژول وابسته، ماژول دیگر فایل کنترلر را ندارد، البته با توجه به سبک طراحی باید یک کنترلر وظایف مورد نیاز را بر عهده گیرد که در ادامه روش کار را شرح خواهیم داد. البته ماژول می تواند سایر لایه ها را هم نداشته باشد و یا حتی به طور کل مجازی بوده و مثل لاگ اوت توسط یک تابع کنترل شود.

می خواهیم صفحات پویای زیر را درست کنیم.

/javad/noshahr

/saman=ramhormoz

/hasan/cityNumber=12

/javad/cityNumber=13

/saman/cityNumber=14

اگر آدرس های بالا را در آدرس بار مرورگر وارد کنید متوجه می شوید که صفحه پیغام 404 را داده؛ چون این ماژول ها کنترلر و حتی شاید پوشه نداشته باشند. اگر درست متوجه شده باشید، قرار است همه این ماژول های وابسته از کنترلر home استفاده کنند. پس در کنترلر قواعدی را می سازیم تا دیگر خطای 404 تولید نشود، و آدرس های ما معتبر باشد.

# بررسی کامل فایل‌های هر ماژول

پس از آشنایی اولیه با ماژول مستقل و وابسته و همچنین شناخت اولیه لایه‌ها، در ادامه به تشریح هر لایه خواهیم پرداخت. قبل از تشریح هر کدام از این فایل‌ها لازم است تا با مشترکات آن‌ها آشنا شویم.

## مشترکات فایل‌ها

Public function ‌‌‌‌config() { }

همان constructor ما می‌باشد که عملیات‌های ابتدایی و اصلی در این تابع صورت می‌گیرد.

$this->sql()

همانگونه که از اسمش مشخص است یک شی از کوئری را برایمان می سازد.

$this->redirect()

یک ریدایرکتور برای صفحه ایجاد می کند که می‌تواند بدون آرگومان باشد یا تا دو آرگومان بگیرد. آرگومان اول به صورت دستی نام صفحه ای که قرار است ریدایرکت شود را مشخص می کنید.

$this->redirect("/profile");

آرگومان دوم مقدار بولین می‌گیرد که صحیح یعنی از طریق کد php و تابع header صفحه ریدایرکت می‌شود و مقدار false از طریق متاتگ‌های html این کار صورت می‌گیرد. توضحیات تکمیلی در عنوان مخصوص به منتقل کننده‌ها آمده است.

این تابع اگر در دو فایل controller, view خوانده شود باید آرگومانی با پیشوند @ و . داشته باشد و اگر در مدل خوانده شود می‌تواند بدون پیشوند آرگومان می‌گیرد یا اصلا آرگومان نمی‌گیرد. اگر در مدل به این تابع آرگومانی اختصاص یابد، در هرکجای ماژول بعد از ساخته شدن این کوئری می‌توان با یکتای آن این کوئری را فراخوانی کرد.

### پیشوند‌ها

@: بعد از این پیشوند نام تابعی در مدل نوشته می‌شود که آن تابع با پیشوند sql\_ در مدل تعریف شده است.

View: $this->sql("@Uid");

Model: function sql\_Uid();

می‌توان در خط فراخوانی آرگومان‌های اضافی داد که آرگومان‌های دوم تا n به تابع اصلی ارجاع داده می‌شوند.

View: $this->sql("@Uid", "test", true);

Model: function sql\_Uid($name, $syntax);

.(نقطه): به کوئری‌های عمومی ارجاع داده می‌شوند. این کوئری‌ها در تمامی صفحات قابل دسترسی هستند.مثال زیر را بررسی کنید.

View: $this->sql(".Uid")

OR Model: $this->sql(".Uid")

OR Controller: $this->sql(".Uid")

Uid Query: class Uid function config()

در این حالت نیز می‌توان آرگومان‌هایی ارسال کرد. توجه داشته باشین که اگر تنها یک نقطه گذاشته شود کوئری از تابع config درخواست می‌شود، اما اگر با دو نقطه مشخص گردد کوئری در کلاس دنبال نام تابع می‌گردد. به مثال زیر دقت کنید.

View,Model,Controller: $this->sql(".Uid.login");

Uid Query: class Uid function login()

توضحیات تکمیلی در بخش مخصوص به دیتابیس‌ها آمده است.

حال که با مشترکات لایه‌ها آشنا شدید لازم است بدانید که هر پوشه شامل چهار فایل یا چهار لایه است که باید با نام مشخص شده در پوشه هر ماژول وجود داشته باشند.

## مدل

فایل model.php وظیفه ارتباط با دیتابیس و تهیه خروجی برای استفاده در سایر لایه ها را بر عهده دارد. برای آشنایی بیشتر با طریقه ارتباط با دیتابیس مستندات بخش sql را مطالعه فرمایید. در ساختار مهندسی MVC و به دلیل یکپارچه بودن سیستم تنها کانال انتقال و دریافت کلاس model می‌باشد. پس ناگذیر باید ساختاری طراحی شود که شما در هرکجای سیستم که هستید بتوانید از model استفاده کنید و اطلاعات خود را ارسال و یا دریافت نمایید.

قبل از شروع، باید بدانید که برای تولید و نمایش یک پیغام در مدل ما از کلاس debug\_lib و توابع آن به صورت زیر استفاده می کنیم. این پیغام ها در دیسپلی به صورت خودکار هندل شما و پیغام را به کاربر نمایش خواهند داد.

lib\debug::true(“operation complete”);

lib\debug::error(“insert users failed”);

lib\debug::warn(“tel is empty!”);

lib\debug::title(“presses title”)

lib\debug::msg(msg\_name, msg\_value)

lib\debug:: property(msg\_name, msg\_value)

برای دریافت مقادیر پست شده از صفحهات display.html کافیست به صورت زیر عمل کنید. فرض کنید شما فرم زیر را در صفحه display.html طراحی کرده اید.

/content/users/add/display.html

{%extends layout%}

{%block content%}

<form class="ajx" method="post">

<input type=”hidden” name=”\_post” value=”add\_users”>

<input type=”text” name=”username” placeholder=”Users Name”>

<input type=”submit”>

</form>

{%endblock%}

توجه داشته باشید تمامی فرم‌های شما باید یک input از نوع hidden با نام post\_ داشته باشد و مقدار value آن، همان نام تابعی است که در مدل ماژول اطلاعات به آن ارسال می شود. یعنی در مثال بالا اطلاعات این فرم با submit کردن آن به تابعی با نام post\_add\_users ارسال می‌شود.

/content/users/add/model.php

<?php

namespace content\users\add;

class model extends MVCmodel

{

function post\_add\_users()

{

$usersname = lib\utility\post::username();

// query to insert users

$this->commit( function ($username = false) {

// if query is ok (status = 1)

debug\_lib::true(“operation complete. username is $username”);

},$username);

$this->roolback( function() {

// if query error (status = 0)

debug\_lib::fatal(“insert users failed”);

});

}

public function sql\_listofusers($username = false)

{

// return result query

// var\_dump($username);

Return 0;

}

}

?>

در مثال بالا مقدار پست شده نام کاربر هم توسط تابع مربوطه دریافت می‌شود. در حقیقت نام متغیری که به سمت php ارسال می‌شود به صورت یک تابع بعد از کلاس post به صورت static قرار دهید تا اطلاعات پست شده در دسترس شما باشند.

هم‌چنین در این فایل شما می‌توانید مشخص کنید که اگر کد شما به صورت صحیح و کامل اجرا شد سیستم تغییرات را در دیتابیس به صورت کامل ذخیره کند(commit) و یا اگر در جایی از کد خطا رخ داده است، سیستم مقادیر وارد شده در دیتابیس را برگرداند علاوه بر این خطاهایی که در ظاهر فرم‌ها نمایش داده می‌شود در این فایل تنظیم می‌شود(roolback). در مثال بالا ما مقدار نام کاربری را به بخش کامیت پاس داده و در نمایش خطا از آن استفاده کرده‌ایم.

تابع دوم که با پیشوند sql\_ تعریف شده است قابلیت فراخوانی در سایر لایه‌ها را دارد که در بخش بعدی از آن استفاده خواهیم کرد. این تابع با متصل شدن به دیتابیس و دریافت نتایج مورد نظر، مقادیر را return می‌کند. از این روش نامگذاری برای دسترسی به یک تابع مدل در سایر لایه ها استفاده کنید. همچنین ما توانایی ارسال پارامتر به مدل را از طریق سایر لایه‌ها داریم که این موضوع هم در ادامه تشریح خواهد شد.

### اتصال به دیتابیس

با استفاده از نمونه مشابه قطعه کد زیر می توانید اتصال به دیتابیس را برقرار کنید.

$query = $this->sql()->tableUser()

->whereUser\_id(12)

->andPassword(md5(111111))

->select();

$this->sql()

این تابع جدول مقصد کوئری مارا مشخص می‌کند. توجه داشته باشین که در سرتاسر این کتاب‌خانه اولین حرف بزرگ تعیین کننده است. همانگونه که در مثال قبل از آن مشاهده می‌کنید، پارامترهای تعریفی را از اولین حرف بزرگ شروع کردیم و نام اصلی تابع قبل اولین حرف بزرگ است، به عنوان مثال در بالا tableTableName() نام تابع table است که TableName نام جدول پایگاه داده‌ای است. به مثال زیر دقت کنید.

->tableUsers() // fn = table; table = users; SELECT \* FROM users

->tablePermisions() // fn = table; table=permisions; SELECT \* FROM permisions

در خط اول مثال بالا ما جدول users را تعریف کردیم و در خط دوم جدول permisions را مشخص نمودیم.

برای تنظیم یک مقدار خاص از روش زیر استفاده می‌کنیم که کلمه کلیدی set در ابتدا و نام فیلد با حروف بزرگ نوشته خواهد شد. به مثال زیر توجه کنید.

->setField(insert)

->setUsrname("ali") // SELECT … SET username="ali"

->setPassword(md5(110)) // SELECT … SET password=md5(110);

->setEmail("ali@world.com") // SELECT … SET email="[ali@world.com](mailto:ali@world.com)"

متنی که برای کوئری ساخته می‌شود اگر برای دستور select باشد خط اول متن زیر را تولید می کند و اگر برای دستور update باشد خط دوم متن زیر تولید می‌شود.

// SELECT … (username, password, email) VALUES ("ali", md5(110), "[ali@world.com](mailto:ali@world.com)");

// UPDATE … SET username="ali", password=md5(110), email="[ali@world.com](mailto:ali@world.com)"

هم‌چنین شما توانایی استفاده از توابع زیر را نیز دارید. این دستورات شرط کوئری را می سازند و مطابق مثال زیر قابل استفاده‌اند.

$this->where, if, or, and, like, orlike, andlike

->whereId(10)->andName("ali")->orlikeEmail("%.com")

// where id=10 and name = "ali" or email like "%.com"

->groupOpen, groupClose

این دستورارت شرط ما را داخل یک پرانتز قرار می‌دهند برای شروط پویا مطابق مثال زیر عمل کنید.

->wehreId(10)

->groupOpen()

->andName("ali")

->orlikeEmail("%.com")

->groupClose()

->orCode(114)

// where id = 10 and (name = "ali" or email like "%.com") or code = 114

->limit(len), limit(start, len)

این تابع اگر یک آرگومان داشته باشد مقدار limit کوئری را از 0(صفر) تا len ادامه می‌دهد و اگر دو آرگومان داشته باشد مقادر limit کوئری را از start تا len ادامه می‌دهد.

->limit(20) // … LIMIT 0, 20

->limit(20, 40) LIMIT 20, 40

->orderId()

این تابع نحوه چینش کوئری را مشخص می‌کند.

->orderId() // ORDER BY id ASC

->orderId("ASC") // ORDER BY id ASC

->orderId("DESC") // ORDER BY id DESC

->groupbyYear()

این تابع کوئری را گروه‌بندی می‌کند.

->groupbyYear() // … GROUP BY year

این توابع وضعیت اجرای کوئری را مشخص می‌کنند.

->select(), insert(), update(), delete()

### ساخت کوئری خصوصی

برای این کار باید حتما فایل model در فولدر ماژول شما موجود باشد. کافیست یک تابع با پیشوند sql\_ در مدل خود ایجاد کنید و در آن مستقیما کوئری بزنید و از هرکجای ماژول که نیاز بود دسترسی داشته باشید. به مثال زیر دقت کنید.

Model:

function sql\_userName($id)

{

retrun $this->sql()->tableUser()->whereId($id)->assoc('username');

}

View:

$tmp = $this->sql("@userName", 14);

### ساخت کلاس کوئری عمومی

برای اینکه در تمامی صفحات و ماژول‌ها قادر باشید که یک کوئری یا یک مجموعه کوئری یکسان را انجام دهید لازم است به پوشه helper/query رفته و یک کلاس تعریف کنید. در هنگام فراخوانی نیز با پیشوند .(نقطه) می‌توانید به آن کلاس دسترسی داشته باشید.

helper/query/user.cls.php

class query\_user\_cls

{

function config($id)

{

return retrun $this->sql()->tableUser()->whereId($id)->assoc('username');

}

function userPassword($id)

{

return retrun $this->sql()->tableUser()->whereId($id)->assoc('password');

}

}

View:

$tmp = $this->sql(".user", 14);

$tmp = $this->sql(".user.userPassword", 13);

## ویو

فایل view.php وظیفه این فایل ایجاد و مدیریت آیتم ها و مقادیری است که در فایل display.html قرار است نمایش داده شود. فرض کنید می‌خواهید لیستی از کاربران خود را ایجاد کرده و نمایش دهید، ایجاد این لیست در این فایل صورت می‌پذیرد و مقادیر آن به فایل display.html ارسال شده تا در تگ ها جایگزاری شود. به مثال زیر توجه کنید.

/content/users/list/view.php

<?php

namespace content\users\list;

class view extends MVCview

{

function config()

{

$users = array('name' =>'reza');

$this->global->list\_of\_users = $users;

}

}

?>

/content/users/list/display.html

{%extends layout%}

{%block content%}

{%for key, value in global.list\_of\_users%}

{{value.name}}

{%endfor%}

{%endblock%}

در این مثال لیستی که در متغیر $users تعریف شده بود در فایل display.html نمایش داده می‌شود. حال سؤال اینجاست، اگر ما بخواهیم این لیست را از دیتابیس استخراج کنیم، چه کنیم؟ نکته اینجاست که در معماری MVC نمی‌توان از داخل فایل view.php مستقیم به دیتابیس وصل شد و مقادیر را استخراج کرد. برای اتصال به دیتابیس باید یک تابع با پیشوند sql\_ در مدل (مشابه مثال بالا) ساخته و در ویو به روش زیر عمل کنیم.

حال درون فایل view.php اگر به صورت زیر عمل کنیم به مقادیر ارسال شده توسط این تابع از دیتابیس می رسیم. در مثال زیر ابتدا تنها تابع را فراخوانی کردیم و در متغییر users ریختیم، سپس در خط بعد پارامتر نام کاربری را نیز به تابع موجود در مدل ارسال کردیم. ارسال چندین پارامتر نیز به همین روش امکان پذیر است.

این نماد ها از زبان برنامه نویسی css الهام گرفته شده است. اگر شما کوئری داشته باشید که مختص به ماژول خاصی نباشد و در چندین و چند جا از آن استفاده شود می‌توانید آن را مشابه یک کلاس css تعریف کنید. و آن را در پوشه cls/query قرار داده تا در هر جایی از پروژه خواستید از آن استفاده کنید تنها با کاراکتر «.» در دسترس باشد. اما اگر sql شما فقط مختص به همین ماژول و درون مدل همین ماژول است از نماد «#» (فراخوانی آیدی تگ در css) استفاده می‌کنید.

/content/users/list/view.php

<?php

namespace content\users\list;

class view extends MVCview

{

function config()

{

$usersname = “test”;

$users = $this->sql(“#listofusers”);

$users = $this->sql(“#listofusers”, $usersname);

$this->global->list\_of\_users = $users;

}

}

?>

یکی دیگر از کار های این لایه ساختن فرم‌های ورودی اطلاعات جدول‌ها می‌باشد. برای مثال با کد زیر فرم افزودن کاربران را خواهیم ساخت.

/content/users/add/view.php

<?php

namespace content\users\add;

class view extends main\_view

{

function config()

{

$f = $this->createform(“@users”, “add”);

// for edit use below line

// $f = $this->createform(“@users”, “edit”);

}

}

?>

/content/users/add/display.html

{% extends layout%}

{%block content%}

<form class="ajx" method="post">

{{ forms.make(form[users]) }}

</form>

{%endblock%}

تنها با همین کد شما فرم ورودی اطلاعات جدول users را ساخته اید. توجه داشته باشید که حتماً باید جدولی با نام users داشته باشید و فایلی با نام users.sql.php در پوشه sql وجود داشته باشد تا فرم آن ساخته شود و برای فراخوانی فرم ساز یک جدول از نماد @ استفاده می‌شود. متغییر دوم تابع form که add و یا edit است و مشخص می‌کند حالت استفاده شده از این فرم جهت اصلاح اطلاعات و یا ورودی اطلاعات است و فرم مربوطه را نمایش خواهد داد.

علاوه بر این موارد می‌توانید در فایل view.php عنوان صفحه، توضیحات صفحه، آدرس پوشه static صفحه و… را در این فایل تغییر دهید که البته در صورتی که هیچ کدام از این مقادیر را تغییر ندهید مقادیر پیش‌فرض در آن قرار می‌گیرد .

برای تغییر متغیر ها می‌توانید از روش زیر استفاده کنید. برخی از این متغییرها را در زیر مشاهده می‌کنید.

<? Php

namespace content\users\list;

class view extends main\_view

{

public function config()

{

$this->global->domain = DOMAIN;

$this->global->path = PATH;

$this->global->site\_url = 'http://'.DOMAIN.PATH;

$this->global->site\_static = 'http://'.DOMAIN.PATH.’/static/’;

$this->global->site\_title = "Test Content”;

$this->global->site\_desc = "This is for test!”;

$this->global->page\_title\_spliter = true;

$this->global->page\_title = $this->global->site\_title;

$this->global->page\_desc = $this->global->site\_desc;

}

}

?>

## کنترلر

فایل controller.php همان‌طور که از اسم آن مشخص است وظیفه کنترل و مدیریت بین ماژول‌ها را به عهده دارد. در این فایل شما می‌تواند مشخص کنید که چه آدرس‌هایی به چه پوشه هایی منتقل شوند و یا سینتکس درست آدرس‌ها چیست و یا اینکه سطح دسترسی به چه صورت می باشد.

## دیسپلی

کلیه خروجی قابل نمایش به کاربر در فایل display.html آماده خواهد شد. سیستم توئیگ[[2]](#footnote-2) به این فایل اضافه شده و تمام کنترل‌ها بر روی مقادیر از طریق توئیگ امکان‌پذیر است. برای آشنایی بیشتر با توئیگ به داکیومنت آن مراجعه کنید.

یکی از ویژگی‌های توئیگ، استفاده از توابع است. برای مثال با استفاده از تابع فرم‌ساز به راحتی می‌توانید فرم مورد نظر خود را بسازید که نمونه آن را در بخش مدل دیدید.

# کار با فرم‌ها

برای استفاده از فرم ساز خودکار باید طبق دستورالعمل زیر اقدامات اولیه ای را انجام دهید.

## اقدامات اولیه

اگر شما بخواهید از طریق php به جدول هایتان اطلاعاتی را ارسال و یا دریافت کنید حتماً باید فایلی با نام جدول شما و با پسوند sql.php درون پوشه sql وجود داشته باشد. در مقابل جدول هایی هستند که از طریق خود موتور mysql اطلاعات آن‌ها مدیریت می‌شود. آن جدول ها نیازی به داشتن فایل sql. ندارند. درون این فایل، مشخصات فیلد ها و اینکه هر فیلد از چه نوعی است، مقدار پیش‌فرض آن چیست؟، برچسب نمایشی آن و … ثبت می‌شود.

علاوه بر موارد بالا، در این فایل مشخص می‌شود که هر فیلد اگر بخواهد نمایش داده شود با چه ظاهر و با چه مشخصاتی در فرم‌های html نمایش داده شود. برای مثال ما جدولی با نام users داریم. در این صورت نیاز است درون پوشه sql فایلی با نام users.sql.php وجود داشته باشد.

درون این فایل کلاسی با نام جدول (users) ایجاد می کنیم. ساختار این فایل باید مشابه زیر باشد. نوع فیلد همان type درون mysql است.

/includes/sql/users.sql.php

<?php

class users {

public $id = array('type' => 'int[@1](mailto:field@lenght)0' , 'label' => 'id');

public $username = array('type' => 'varchar[@3](mailto:field@lenght)2' , 'label' => 'username');

public $description = array('type' => 'text[@](mailto:field@lenght)' , 'label' => 'description');

public $gender = array('type' => 'enum[@m](mailto:field@lenght)ale,female!male' , 'label' => 'label of field');

public function id () {}

public function username () {}

public function description () {}

public function gender () {}

}

?>

بعد از تعریف نوع و برچسب فیلد ها نوبت به ساخت فرم‌های Html فیلد های می رسد. برای این کار ابتدا باید برای هر فیلد یک تابع با نام همان فیلد ساخت. درون این توابع چند عملیات صورت می‌پذیرد ابتدا اینکه نوع حالت نمایشی فرم در html مشخص می‌شود که مثلاً این فیلد اگر بخواهد نمایش داده شود در قالب input باشد یا radio یا checkbox. در ادامه نوع توابعی که با آن‌ها اعتبار سنجی می‌شود مشخص می‌شود برای مثال این فیلد با تابع number اعتبار سنجی شود یا تابع farsi. فرزندان فیلد اگر از نوع enum باشند در توابع تنظیم می‌شوند که فرزندان نمایش داده شوند یا نه.

## تعیین ویژگی های هر کنترل

با مقادیر وارد شده در تابع مربوط به هر متغییر می‌توانیم کنترل های مربوط به آن متغییر را مدیریت کنیم. برای مثال نوع فرم، اعتبار سنجی آن و افزودن مقادیر مورد نیاز به کنترل. به مثال زیر دقت کنید.

public function username () {

**$this->form(“text”)->name(“username”)**

**->label(“نام کاربری “)**

**->pl(“نام کاربری خود را وارد کنید”)**

**->addClass(“classname”);**

}

با دستور بالا اگر فیلد username بخواهد نمایش داده شود در خروجی به شکل زیر ساخته خواهد شد.

<label for=”**username**”>**نام کاربری** </label>

<input type=”**text**” name=” **username**” placeholder=“**نام کاربری خود را وارد کنید**” class=”**classname**” id=”**username**”>

اما یک سری از فرم‌ها هستند که به صورت تکراری در پروژه استفاده خواهند شد به عنوان مثال در یک پروژه می‌تواند چندین فیلد از نوع input و text باشد. برای این کار لازم نیست ما درون هر کدام از توابع این فلید ها در فایلsql مربوطه فرم آن را ایجاد کنیم بلکه می‌توانیم یک بار فرمی با مشخصات لازم را طراحی کرده و در مواقع نیاز آن را فراخوانی کنیم.

اصطلاحاً به این کار عملیات extend کردن فرم‌ها می گوییم.

فرم‌های مورد نیاز در فایلی ساخته می‌شوند و هر کجا که نیاز به استفاده از آن‌ها بود با صدا زدن نام فرم به همراه کاراکتر «#» فراخوانی می شوند. برای اینکه فرمی را به صورت عمومی و یا حالت extend تولید کنید وارد پوشه forms موجود در پروژه شده و فایل Extends.cls.php را باز کنید. در اینجا به موتور forms\_lib متصل هستیم پس برای ساختن فرم از تابع form استفاده نمی‌کنیم بلکه از تابع make استفاده خواهیم کرد.

برای ساختن یک فرم عمومی ابتدا برای آن نامی اختصاص داده و فرم را درون آن قرار می دهیم.

class forms\_Extends\_cls extends forms\_lib

{

function \_\_construct()

{

$this->username = $this->make("text")->name("username")

->label(“نام کاربری “)

->pl(“نام کاربری خود را وارد کنید”)

->addClass(“classname”);

}

**}**

با ایجاد شدن کد بالا در فایل users.sql.php دیگر نیازی به نوشتن فرم نیست بلکه به صورت زیر فرم username فراخوانی می شود.

public function username () {

**$this->form(“#username”);**

}

اما برای اعتبار سنجی فرم‌ها از توابع validate استفاده می‌کنیم. فرض می‌کنیم که ما قبلاً [توابع اعتبار سنجی](#_اعتبارسنجی) خود را تولید کرده‌ایم و اکنون فقط از آن‌ها استفاده خواهیم کرد.

/includes/sql/users.sql.php

public function username () {

$this->form(“#username”)**->validate()->number();**

}

با فراخوانی تابع number سیستم به فایل validateExtends.cls.php مراجعه کرده و مقداری وارد شده در این فیلد را به آن تابع ارسال می‌کند و آن را اعتبار سنجی می کند.

## اعتبارسنجی

برای اعتبار سنجی در هر نوع تعریف از تعاریف چهارگانه فرم که هستید می توانید این کار را انجام بدهید. در اعتبارسنجی، شما یک سری تابع خاص را فراخوانی می‌کنید که برای اعتبارسنجی روی هر مقدار باید ابتدا تابع آن نوشته شود که مقدار را بگیرد و در صورتی صحیح بودن return true بدهد و اگر مقدار مطابق استاندارد های شما نبود return false را تولید می‌کند. این توابع در فایل validateExtends.cls.php تولید می‌شود. برای مثال تابع مربوط به اعتبارسنجی کلمه عبور را بررسی کنید. از این تابع در بخش اعتبارسنجی خصوصی و عمومی استفاده خواهیم کرد.

/includes/helper/validateExtends.cls.php

<?php

class validateExtends\_cls

{

public function password()

{

if(!preg\_match("/^.{5,20}$/", $this->value)){

return false;

// $this->SetOnError('password');

}else{

$this->value = md5($this->value);

}

return true;

}

{

?>

برای استفاده از توابع نوشته شده و اعتبارسنجی، مثال زیر را بررسی کنید.

/includes/helper/forms/extends.cls.php

$this->password = $this->make("password")->name("password")->label("password");

$this->password->validate()->password()->form->password("password incorrect");

در مثال ما در خط 1 فرم و نام آن را تعریف کردیم و درخط 2 اعتبارسنجی را مشخص نمودیم، توجه داشته باشید که حتما مشابه مثال عمل کنید و در یک خط فرم را تعریف کنید و در خط دیگر اعتبار سنجی را چرا که شما زمانی که تابع validate را استفاده می کنید بازگشت شما تابع اعتبارسنج می باشد نه فرم و در این صورت اگه مستقیم در خط اول این کار را بکنید با مشکل مواجه خواهید شد.

تابع پسورد استفاده شده در خط دوم مثال بالا مربوطه به استفاده از همان تابع تعریف شده در بخش قبل برای کلمه‌عبور می‌باشد. همچنین در انتها متن مورد نظر در صورت اشتباه بودن اعتبارسنجی نمایش داده خواهد شد.

## ساخت فرم

برای ساخت یک فرم استاندارد تنها از دوحالت ساخت گروه فرمی عمومی و ساخت گروه فرمی خصوصی استفاده نمایید. زیر فرم هایی همانند submit, hidden input, gender,… که قرار است در چندین فرم قرار بگیرد را به صورت ساخت عضو خصوصی فرم بنویسید. حتما فرمتان باید دارای یک زیرفرم hidden با نام \_post و مقداری مشخص باشد که بتوانید در سمت مدل به آن دسترسی داشته باشید. همچنین سعی کنید تا حد امکان از اعتبار سنجی ها استفاده کنید و خطاگیری را برای خودتان تنظیم نمایید.

### ساخت یک فرم عادی

در لایه view تابعی وجود دارد به نام form که می توانید در تابع config از آن استفاده نمایید.

$input = $this->form("text")->label("نام")->addClass("b")->addClass("c")->removeClass("b");

$this->data->formElement[] = $input->compile()

حال در display.php در بلوک content کد های زیر را وارد کنید.

<form class="ajx" method="post">

{{forms.make(formElement)}}

</form>

در مثال بالا شما نوع فرم را در اولین اندیس تابع form می گذارید و بعد شروع می کنید به مقدار دهی بجز توابع بالا هر تابع دیگری را اضافه کنید، مساوی با مقدار attribute فرمتان می شود درضمن اگر در اسم تابع یک \_ زیرخط باشد در قسمت attribute به صورت – دش در می آید. این جا را هم ببینید [ساخت عضو خصوصی فرم](#_Toc390211235).

php: $this->form("text")->lable("نام")->data\_type("json")->index\_\_name("one")->href("first.html");

html: <type="text" data-type="json" index\_name="one" href="first.html">

به مثال زیر دقت کنید.

$inputa = $this->form("#faname")->label("نام");

$inputb = $this->form("#faname")->label("نام پدر");

$this->data->formElement[] = $inputa->compile();

$this->data->formElement[] = $inputb->compile();

اگر به آدرسcore/cls/forms/symbol.php بروید متوجه می شوید که در تابع سازند کلاس یکسری شی برای کلاس تعریف شده است علامت # شما را به آن شی ها هدایت می کند، در حقیقت این ها فرم های پیش ساخته شده هستند. که می توانید فرخوانی کنید و در ادامه ویرایش نمایید. برای اطلاعات بیشتر می توانید به [ساخت عضو عمومی فرم](#_ساخت_عضو_عمومی) مراجعه نمایید.

کمی سطح مثال را بالاتر می بریم

$form = $this->form(".login");

// $form = $this->forms->login = $this->data->forms['login']

باز اگر به آدرس core/lib/forms بروید متوجه میشوید که فایلی با نام login.php وجود دارد، این همان نامی است که ما در تابع فرم گذاشتیم و قبلش را با نقطه مشخص کردیم، دیگر شما یک گروه فرمی دارید که می توانید از آن استفاده کنید در ضمن اگر فرمی را در view تولید کردید می توانید با سه حالت نام خصوصی، دسترسی از طریق forms و دسترسی از طریق data->forms['login'] داشته باشید، همانند خط 2 مثال بالا. برای اطلاعات بیشتر [ساخت گروه فرمی عمومی](#_Toc390211234) را بخوانید.

مثال زیر را نیز بررسی کنید.

home/view.php config()

$myForm = $this->form("@login");

home/view.php

…

class forms extends forms\_lib{

function login(){

$this->input = $this->make('#hidden')->value('login');

$this->username = $this->make("#username");

$this->password = $this->make("password")->name("password")->label("گذرواژه");

$this->sumbit = $this->make("#submit")->value("ورود");

}

}

در این مثال شما در همان view که می خواهید صفحه را تنظیم کنید کلاسی را با نام forms می سازید و با نام forms\_lib بسط می دهید، هر تابع در این کلاس مشخص کننده نام فرمی است که می خواهید فراخوانی نمایید، که می‌توانید از هر سه حالت بالاتر آن استفاده کنید. برای اطلاعات بیشتر [ساخت گروه فرمی خصوصی](#_Toc390211235) را بخوانید.

تفاوت حالت 3 و 4 این است که در حالت 3 شما زمانی استفاده می کنید که ممکن است یک فرم به طور کلی و یا بیشتر عضو های آن در بیشتر از یک فرم استفاده شود، همانند فرم login که دو عضو نام کاربری و گذرواژه آن در قسمت register نیز کارایی دارد.

درضمن در حالت های 3 و4 دیگر لازم نیست فرم را compile نموده و نامگذاری کنید بلکه در قسمت display می‌توانید نام فرمتان را بگذارید، ما compile و نامگذاری را به صورت پویا در این دو حالت انجام می دهیم:).

<form class="ajx" method="post">

{{forms.make(form.login)}}

</form>

### ارسال اطلاعات

خب حالا اگر دکمه ارسال را بزنید، شما به صفحه ای منتقل می شوید اما دارای خطا می باشید، چون نه فایل model.php را ساختید و نه تابعتان را مشخص نمودید. برای رفع این مشکل کلاسی همانند کلاس های بالا می سازیم اما با نام model و با بسط دهنده main\_model.

<?php

class model extends main\_model{

function post\_login(){

debug\_lib::true("Test");

}

}

?>

در فرم شماره 4 اولین فرمی که ساخته شد یک فرم بود با نام #hidden برای اینکه فرم هایتان سمت سرور دارای اعتبار باشد باید تابعی با نام فرم وجود داشته باشد و نام فرمتان را هم داشته باشید نام فرم را در یک فرم hidden با نام \_post و با هر مقداری که خواستید می گذارید که ما در مثال نامش را login گذاشتیم، حال سمت model در کلاس model تابعی می سازیم که ابتدای نامش post\_ است و انتهایش نامی که در فرمhidden گذاشته بودیم. برای اطلاعات بیشتر به ساخت یک فرم استاندارد مراجعه کنید.

### ساخت فرم عمومی

این نوع نگارش زمانی صورت می گیرد که شما می خواهید از یک فرم در چند جا استفاده نمایید. برای این که بتوان یک فرمی ساخت که همه جا در دسترس باشد باید در آدرس includes/lib/forms یک فایل ایجاد کرد با نام یکتای فرم. مثال زیر را بررسی کنید.

includes/lib/forms/login.lib.php

<?php

class forms\_login\_lib extends forms\_lib

{

function \_\_construct()

{

$this->input = $this->make(“#hidden”)->value(“loginsend”);

$this->username = $this->make("text")->label("نام کاربری")->name("username")->pl(“نام کاربری”);

$this->password = $this->make("password")->name("password")->label("گذرواژه");

$this->submit = $this->make("#submit")->value("ورود");

}

}

?>

یادتان باشد چه در این حالت و چه در حالت فرم خصوصی باید اعتبارسنجی را در همان تابع انجام دهید، تا در قسمت مدل قابل دریافت باشد. در این حالت برای این که در لایه view قابل دسترسی باشد می توانید نام فرم را با علامت . (نقطه) صدا بزنید. مثلا

$myForm = $this->form(".login");

در خط 2 ما کلاس فرمی login را ایجاد نمودیم در خط 4 تابع ایجاد کننده کلاس را تشکیل دادیم و از خط 6 تا 9 سه مقدار برای فرممان اختصاص دادیم. در خط 7 pl به معنای خصیصه (placeholder) می باشد. توجه داشته باشید که اگر برای فرمتان id و placeholder تعریف نکنید به صورت اتوماتیک id را از name و placeholder را از label می گیرد. شما می توانید از addClass, removeClass, className برای تنظیمات کلاس برروی المان مورد نظر استفاده کنید. درضمن هر تابعی را با نام سفارشی که بکار ببرید تبدیل به نام خصیصه در کد نهایی خواهد شد. به عنوان مثال

$this->make(“text”)->datatype(“json”)->alt(“loading josn”);

در خط بالا ما دو خصیصه datatype, alt را به تگمان اضافه نمودیم که حالت html آن به خروجی زیر تبدیل می‌شود.

<input type=”text” datatype=”json” alt=”loading json” />

اگر در نام خصیصه یک \_ (زیرخط) بود در html می شود – (دش) و اگر دو زیرخط یا بیشتر بود می شود \_ (زیرخط). به مثال زیر دقت کنید

$this->make(“text”)->data\_type(“json”)->data\_\_class(“text/json”);

که در حالت نمایش html تبدیل به خروجی زیر می‌شود.

<input type=”text” data-type=“json” data\_class=”text/json” />

### ساخت فرم خصوصی

این نوع نگارش زمانی صورت می گیرد که شما می خواهید فقط در این ماژول از فرم استفاده نمایید. برای این که بتوان یک فرم خصوصی ساخت باید در همان فایل view.php ماژولتان یک کلاس با نام forms مطابق مثال زیر باز کنید.

login/home/view.php

<?php

class forms extends forms\_lib

{

function loginform()

{

$this->input = $this->make(“#hidden”)->value(“loginsend”);

$this->username = $this->make("text")->label("نام کاربری")->name("username")->pl(“نام کاربری”);

$this->password = $this->make("password")->name("password")->label("گذرواژه");

$this->submit = $this->make("#submit")->value("ورود");

}

}

?>

در فرم خصوصی اعتبارسنجی را فراموش نکنید. در این حالت برای این که در لایه view قابل دسترسی باشد می توانید نام فرم را با علامت @ مطابق مثال زیر صدا بزنید.

$myForm = $this->form("@loginform ");

## کار با کتاب‌خانه فرم‌ساز

این کتاب‌خانه با نام $this->form() نیز در view قابل دسترسی است.

$this->form(type)

نوع فرمی را که قصد ایجادش را داریم مشخص می‌کند.

$this->form("checkbox") // <input type="checkbox">

->attr(name, value)

خصیصه ای با نام name و مقدار value به فرم اضافه می‌کند.

->attr("data-name", "ignore") // <input … data-name="ignore">

توجه داشته باشید که خصیصه‌ها را نیز می‌توان مستقیم با قراردادن نام تابع وارد کرد.

->tabindex("0") // <input … tabindex="0">

->style("color:red") // … style="color:red">

->classname(cls)

کلاس فرم را با مقدار cls ایجاد می‌کند.

->classname("tabvil") // … class="tabvil">

->addClass(clsn)

کلاس clsn را به کلاس فرم اضافه می‌کند و اگر فرم کلاسی نداشت مقدار کلاس را برابر clsn قرار می‌دهد.

->classname("one") // class="one"

->addClass("two") // class="one two"

->removeClass(clsn)

کلاس clsn را از کلاس‌های فرم حذف می‌کند.

->classname("one two") // class="one two"

->removeClass("two") // class="one "

->pl(pl)

مقدار pl را برابر placeholder فرم قرار می‌دهد. شما می‌توانستید این خصیصه را دستی وارد کنید اما برای سهولت مخفف آن را تعبیه کرده‌ام.

->label(lb)

مقدار lb را برابر label فرم قرار می‌دهد.

->child()

یک فرزند به فرم اضافه می‌کند. این تابع برای نوع‌های radio و select لازم است. درضمن اگر از این تابع استفاده نمودید از کلاس formMaker خارج شده و یک کلاس جدید ایجاد میشود برای فرزند.

$x = $this->form("radio")

->name("gender")

->child()

->value("male")

->label("male");

$x->child()

->value("female")

->label("female");

$x->label("gender")

<label for="gender">gender</label>

<lable for="male">male</label> <input type="radio" id="male" name="male" value="male">

<lable for="female">male</label> <input type="radio" id="female" name="female" value="female">

## ساخت فرم از طریق متدهای پیش‌تعریف

شما می‌توانید در آرگومانی که نام فرم را تعریف می‌کنید علامت # را قرار دهید تا از متدهای پیش‌تعریف که در کلاس forms\_Extends\_cls نوشته شده استفاده کنید.

## ساخت فرم از طریق دیتابیس

با قرار دادن علامت @ و نام جدول پایگاه داده‌ها میتوانید لیستی از فرم‌های آن جدول را تهیه کنید.

# کنترل صفحه

برای کنترل آدرس‌های اضافی موجود در آدرس بار مرورگر از این تابع استفاده می شود.

## روتر

برای روتر کردن می توان به صورت مستقیم از خود کلاس روتر استفاده کرد.

new \lib\router\route(conditions, [[afterTrue]..args]);

اما به یاد داشته باشین که اگر در ساختار mvc بودید. یعنی در یکی از کلاسهای مدل ویو و یا کنترلر بودید باید از حالت زیر استفاده کنید.

$this->route(conditions, [[afterTrue]..args]);

پارامتر اول قوانین روتر است که اگر حالت صفحه شما با آن قوانین هماهنگ بود روتر مقدار true را بر می گرداند.

پارامتر دوم یک تابع دلخواه می باشد که اگر روتر مقدار true را برگشت داد این تابع اجرا می‌شود. پارامترهای دوم به بعد مقادیر ارسالی به تابع دلخواه هستند و اگر تابع وجود داشته باشند به آرگومان‌های آن تابع ارسال می‌گردند.

### **قوانین مقادیر ورودی روتر**

در حالت رشته‌ای اگر مقدار قانون رشته باشد، هماهنگ بودن آن با آدرس صفحه چک می‌شود.

در حالت عبارت ویژه اگر مقدار قانون عبارت ویژه باشد، آدرس صفحه با آن عبارت چک می‌شود.

از حالت آرایه ای در گاهی از اوقات که لازم است چندین مقدار برای قانون گذاری دخیل باشند، استفاده می‌شود. برای این کار از آرایه استفاده می کنیم. به نمونه زیر دقت کنید.

url => array() or strign or RegExp

real\_url => array() or strign or RegExp

fn => array(function, ...args) or function

max => int

min => int

sub\_domain => array(), strign, RegExp

domain => array(), strign, RegExp

property => array()

get =>array()

post =>array()

url : مقدار آدرس صفحه را مشخص می‌کند، البته آدرس مجازی. بهتر است در از این مقدار به جای مقدار real\_url استفاده کنید. چرا که ممکن است در مسیر اجرای صفحه جایی real\_url دست خورده باشد و در اصل ما نیاز به url داشته باشیم. مثلا فرض کنید ما سیستم api خود را فعال کرده‌ایم. سیستم api زمانی کار می‌کند که ابتدای همه آدرس‌ها عبارت api نوشته شود. مانند مثال زیر

site.com/api/post

site.com/api/post/10

در این حالت برای اینکه مسیریابی صحیح باشد ما بعد از تشخیص اینکه سیستم api فعال است عبارت api را از url حذف می‌کنیم. حال اگر شما در قوانین از real\_url استفاده کرده باشید باید دو حالت روتر برای حالت api و غیر آن داشته باشید

default router

real\_url => array(“post”, “/\d+/”);

api router

real\_url => array(“api”, “post”, “/\d+”)

اما اگر از حالت url استفاده کرده باشید، فقط یک حالت کافی است. چرا که عبارت api از url حذف شده است.

url => array(“post”, “/\d+/”);

برای حالت‌های url ,real\_url, sub\_domain , domain می توان از سه حالت استفاده کرد.

حالت اول رشته است که دقیقا مقادیر را مطابق آن می‌سنجد. حالت دوم عبارت ویژه است که مقدار درخواستی را با عبارت ویژه می سنجد و حالت سوم آرایه است که درون آرایه ها می‌توان از رشته و عبارت ویژه استفاده کرد. خانه اول آرایه اولین مقدار درخواستی است. مثلا آدرس صفحه ما به صورت زیر است.

/admin/post/10

برای قانون گذاری روتر آدرس url به صورت زیر عمل میشود.

array(“admin”, “post”, “/\d+/”)

min ,max: مقدار max به صورت پیشفرض صفر است. اگر شما مقدار url را عبارت ویژه قرار دهید مقدار max خوانده نشده و بستگی به قانون‌گذاری شما در عبارت ویژه دارد. اما اگر مقدار رشته بود باید max را تعریف کنید و اگر آرایه بود مقدار max از طول آرایه گرفته می‌شود. همه این‌ها زمانی است که شما مقداری برای max تعریف نکرده اید اما اگر تعریف کنید برابر با مقدار شما می‌گردد. مقدار max , min تعداد دایرکتوری های شما در آدرس صفحه می باشد.

Fn: می تواند دو حالت مقداری بگیرید. اگر تابع گرفت تابع اجرا میشود و اگر بازگشت true داشت که عملیات صحیح است اما اگر بازگشت false داشت قانون شما اشتباه است و روتر مقدار false بر می‌گرداند. اما اگر آرایه بود مقدار اول باید تابع باشد که همانند بالا عمل می‌شود و بقیه مقادیر همان آرگومان های تابع هستند.

توجه داشته باشین که هسته در حین بارگذاری قبل از اینکه وارد ساختار mvc شما شود اگر کلاسی به اسم cls/router داشته باشین وارد کلاس شده و اگر در کلاس تابع \_before باشد قبل از روتر اصلی هسته اجرا می‌شود و اگر \_after باشد بعد از روتر اصلی هسته اجرا میشود.

# انتقال دهنده صفحه

ریداریکتور یا همان انتقال دهنده صفحه به ما کمک می کند که صفحه خودمان را به صفحه ای دیگر که مورد نیاز است انتقاد دهیم.

برای ریدایرکت کردن می توان به صورت مستقیم از خود کلاس ریدایرکت استفاده کرد.

$redirector = new \lib\redirector($url = null, $php = true);

آرگومان اول می‌تواند رشته یک آدرس خاص باشد و آرگومان دوم یک مقدار منطقی است که اگر true بود ریدایرکت کردن صفحه از طریق php و با متد header صورت می‌گیرد، اما اگر false بود از طریق تگ meta در html صورت می‌گیرد.

اما اگر قصد دارید در داخل وب‌سایت خود از صفحه‌ای به صفحه دیگر ریدایرکت شوید و یا از ماژولی به ماژول دیگر می‌توانید مقدار url را null گذاشته و از طریق متدهای این کلاس آدرس دهی کنید.

$redirector->set\_protocol(‘ftp’);

این متد پروتکل آدرس صفحه را تنظیم می‌کند. به عنوان مثال اگر آدرس صفحه فعلی شما <http://saloos.ir> باشد، اگر مثال بالا را اجرا کنید، آدرس صفحه به <ftp://saloos.ir> تغییر پیدا می‌کند، بدون نیاز به نام دهی دومین.

$redirector->set\_domain($domain = null)

این متد نام دامنه ریدایرکتور را مشخص می‌کند. اگر نام دامنه null باشد، زیر دامنه‌ها را حذف کرده و نام خالص دامنه را جایگزین می‌کند. به مثال زیر توجه کنید.

url: http://portal.saloos.ir/login

$redirector->set\_domain();

url: <http://saloos.ir/login>

همانطور که مشاهده می‌کنید تنها نام دامنه را از زیر دامنه به دامنه اصلی تغییر داده و هیچ تغییر دیگری در آدرس صفحه داده نشده است، که باید به این امر توجه شود.

$redirector->set\_url($value = '')

این متد آدرس دایرکتوری‌های صفحه را تغییر می‌دهد که می‌تواند دو حالت رشته‌ای و آرایه‌ای بگیرد.

# روش معرفی توابع

اسم تابع

مقدار بازگشتی

شرح مقادیر بازگشتی

پارامترهای ورودی (در صورت وجود)

شرح تابع

مثال

لینک به برخی از توابع هم رده

لینک به برخی از توابع مورد نیاز

1. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [↑](#footnote-ref-1)
2. http://twig.sensiolabs.org/documentation [↑](#footnote-ref-2)