بسم الله الرحمن الرحیم

# مقدمه

## بررسی اجمالی

با نام و یاد خدا شروع می کنیم

این سیستم توسط سامان سلطانی، جواد عوض زاده کاکرودی و محمد حسن صالحی حاجی آبادی نوشته شده است.

البته از آقایان رضا محیطی و احمد کریمی به خاطر توسعه این سیستم تشکر می کنیم.

## ویژگی ها

این سامانه ویژگی های زیادی خواهد داشت!

فهرست مطالب

[مقدمه 1](#_Toc390211222)

[بررسی اجمالی 1](#_Toc390211223)

[ویژگی ها 1](#_Toc390211224)

[اصول اولیه 2](#_Toc390211225)

[ساخت یک ماژول مستقل 2](#_Toc390211226)

[ساخت ماژول وابسته 4](#_Toc390211227)

[ساخت فرم 7](#_Toc390211228)

[1. ساخت یک فرم عادی 7](#_Toc390211229)

[2. بیایید مثال را کمی پیچیده تر کنیم. 7](#_Toc390211230)

[3. چطور است کمی سطح کار را بالا ببریم! 7](#_Toc390211231)

[4. چطور است یک حالت دیگر را تست کنیم 8](#_Toc390211232)

[ارسال اطلاعات 9](#_Toc390211233)

[ساخت عضو عمومی فرم 9](#_Toc390211234)

[ساخت عضو خصوصی فرم 9](#_Toc390211235)

[صفحه یاب ها 10](#_Toc390211236)

[مقادری مقدار اول 10](#_Toc390211237)

[ساخت یک فرم استاندارد 11](#_Toc390211238)

[ساخت گروه فرمی عمومی 12](#_Toc390211239)

[ساخت گروه فرمی خصوصی 14](#_Toc390211240)

[اعتبارسنجی 15](#_Toc390211241)

[اعتبارسنجی خصوصی 15](#_Toc390211242)

[اعتبارسنجی عمومی 16](#_Toc390211243)

[دسترسی به پایگاه داده ها 18](#_Toc390211244)

[حالت اول اتصال در کلاس مدل 18](#_Toc390211245)

[حالت دوم اتصال توسط تابع محلی 18](#_Toc390211246)

[حالت سوم استفاده از کلاس های عمومی 19](#_Toc390211247)

[ساخت لیست دیتا 21](#_Toc390211248)

[توابع 21](#_Toc390211249)

[مثال 24](#_Toc390211250)

[دریافت اطلاعات در کلاس مدل 26](#_Toc390211251)

[اعتبارسنجی گروه فرم ها (عمومی، خصوصی) 26](#_Toc390211252)

[اعتبارسنجی فرم های عمومی 26](#_Toc390211253)

[اعتبارسنجی دستی 26](#_Toc390211254)

[انتقال دهنده صفحه 28](#_Toc390211255)

[مثال 29](#_Toc390211256)

[ساخت فرزندان مشترک بین کلاس های ماژول 30](#_Toc390211257)

[ساخت تابع مشترک 30](#_Toc390211258)

[ساخت متغییر مشترک 31](#_Toc390211259)

# اصول اولیه

## ساخت یک ماژول مستقل

به صورت کلی برای ساخت هر ماژولی نیاز داریم تا دو پوشه به جلو برویم، این کار برای سهولت دسترسی و گروه بندی نیاز است.

پس ابتدا برای خودتا گروه هایی را مشخص کنید. به عنوان مثال

news => add, edit, view, tags, cat, …

profile => privacy, options, add, …

دو نام news, profile هرکدام یک گروه هستند که فرزندانشان درون این دو پوشه قرار می گیرند که هر پوشه ای نام یکی از فرزندان این گروه می باشد.

پس اگر خواستیم صفحه ای برای ساخت اخبار درست کنیم باید پوشه news/add را بسازیم. لازم به ذکر است اگر در آدرس صفحه اسم اصلی گروه را وارد کردید پوشه home زیرگروه اجرا می گردد، یعنی آدرس news دقیقا مساوی با آدرس news/home می باشد. و اگر خواستید به پوشه / در آدرس بار صفحه یا همان index صفحه دسترسی داشته باشید باید ماژول home/home را بنوسید که این ماژول حتما باید یک ماژول مستقل باشد، چرا که اگر هیچ کدام از آدرس های شما پیدا نشد، نهایتا هسته کدها را به این آدرس منتقل می کند.

البته اگر بعد از نام های home اگر مقداری در آدرس صفحه ارسال نشود نامعتبر بوده و صفحه خطای 404 می دهد. پس زمانی که می خواهید به پوشه news/home دسترسی پیدا کنید کافیست تنها نام اصلی پوشه را وارد نمایید مثل news.

درضمن شما می توانید یک پوشه دیگر هم داخل بروید، اما به نظر من پوشه سوم که اسمش را پوشه فرزند گذاشتم را نگه دارید برای ماژول های وابسطه و فرزندها که در حالت مثال اینگونه می شود.

profile/user/add

حال نحوه استقلال یک ماژول از کنترل (controller) آن ماژول فهمیده می شود، پس ماژولی که کنترلر داشته باشد یعنی یک ماژول مستغل است و می توان با آدرس دهی در صر صفحه مرورگر آن را فراخوانی کرد.

هر ماژول مستقل علاوه بر فایل controller.php که اجباری است می تواند یک یا هردو گروه زیر را داشته باشد

view.php, display.html: این دو فایل برای حالت نمایشی تنظیم می شوند که در view.php تنظیمات پویا صورت می گیرد و در قسمت display.html تنها نشانک های html و کد های twig می باشد.

model.php: این فایل همان ساختار فایل model در حالت اجرایی MVC می باشد. که در این فایل کوئری ها، دریافت پست ها و ... صورت می گیرد.

**مثال**

home/home/controller.php

<?php

class controller extends main\_controller{

function config(){

}

}

?>

حال که دایرکتور اصلی ما / یا همان index برروی آدرس بار معتبر است، بیایید کد را بررسی کنیم:

توجه داشته باشید که چون ما داریم در تعامل با هم کار می کنیم، پس باید از کارهای یکدیگر آشنا باشیم، برای آشناییت شما با ما، ما یکسری قواعد را در داخل این جزوه راهنما به شما آموزش می دهیم، و برای آشناییت ما با شما، لازم است تمامی کلاس های معتبرتان را از طریق کلاس هایی که ما تعریف کردیم بسط (extend) دهید، تا بتوانیم با هم کارها را بدون نقص پیش ببریم. پس شما در اولین کاری که می کنید، یک کلاس controller در controller.php می سازید و با کلاس main\_controller بسط می دهید، همانند خط 2 مثال ما. در ضمن 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، اما ناراحت نباشید، چرا که برای شما تابع سازنده ای با نام config رزور کردیم که در هر کلاسی تعریف شود کار همان سازنده اصلی را انجام می دهد، همانند خط 3، که می توانید از آن استفاده بکنید، 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.

خب چون ما فعلا کاری با controller نداریم، به همین صورت رهایش می کنیم.

home/home/view.php

<?php

class view extends main\_view{

public function config(){

$this->global->title = 'ایران جاویدان';

$this->data->titlepage = 'چون ایران نباشد، طن من مباد';

}

}

?>

در متن کد بالا خط 2 آشناست، تنها نام ها عوض شده اند وگرنه تمامی توضیحات بخش قبل مو به مو در این نوع کلاس سازی باید رعایت شود.

همانطور که در قسمت قبل گفتیم، کار پیکر بندی و یا سازنده کلاس ماژول به عهده تابع config می باشد که فکر کنم در این مثال بهتر متوجه شوید.

مثلا ما وقتی صفحه ای را می سازیم نیاز به یکسری مقادیر پویا داریم، مثل، عنوان صفحه، منوها، متن تیتر صفحه، و یا حتی خود متن اصلی صفحه. تمامی این ها در تابع config عنوان سازی می شوند.

شما در یک قدمی نمایش صفحه تان هستید، چون صفحه نمایش همگی کد های html,twig می باشد، پس مقادیر پویا در تابع config و در زیر شیء $this->data تعریف می شوند، هرنامی را دلتان خواست بگذارید، اما یادتان باشد دو نام global, form رزرو شده است، global برای متغیر های همیشگی همانند نام دامنه، نام زیردامنه، پوشه اصلی سایت، عنوان صفحه و چیزهایی از این قبیل، و form هم برای فرم هایی است که قصد دارید در صفحه بسازید.

و یکسری نام دیگر هم رزرو شده اند:

macro, mainDisplay, extendForm

در خط 4 ما مستقیما در $this->global->title عنوان صفحه را معرفی کردیم، این نام برای سهولت دسترسی می باشد ولی می توانید از محل اصلی خودش فراخوانی کنید که در $this->data->global->title می باشد.

خط 5 هم مثالی از ساخت مقدار پویا می باشد که ما یک نام و مقدار را تعریف کردیم تا نهایتا در display.html جایگذاری نماییم.

{%extends mainDisplay%}

{%block content%}

my title is : {{global.title}}

<h1>{{titlepage}}</h1>

سعدیا مرد نکونام نمیرد هرگز.

{%endblock%}

در خط 1 ما کد های اصلی html سایت که شامل سر صفحه، پاصفحه، منوها و هزاران کد که در هر صفحه ای باید تکرار شود را به صفحه خود اضافه می کنیم، خط 2 و 6 بلوکی تعریف می کنیم، که در پوسته اصلی که در خط 1 جایگذاری شد، رزرو شده است، پس محتویات بین این دو خط در آن بلوک ها جایگذاری می شود.

خط 3 ما از عنوان صفحه را فراخوانی کردیم، این همان عنوانی است که در تب مرورگرتان نمایش داده می شود، برایتان مثال زدیم که اگر نیاز بود در صفحه نام عنوان را داشته باشید، بتوانید استفاده کنید.

خط 4 آن متغیری که در خط 5 فایل view.php تعریف شد.

تبریک می گویم، شما یک ماژول نمایشی ایجاد کردید که به درستی کار می کند.

## ساخت ماژول وابسته

برای این کار شما تمامی حالت های بالا را تکرار می کنید، با این تفاوت که دیگر پوشه و کلاسی به اسم controller نخواهیم داشت. پس در حقیقت کنترل این ماژول چه می شود؟ ناراحت نباشید ما فراموشش نکرده ایم. اگر ما در پوشه سوم یعنی فرزند باشیم، مثلا profile/user/add می خواهیم اگر پوشه add دارای controller نبود کدام ماژول خواند می شود، که در زیر بررسی شده است.

profile/user/add/controller.php => profile/user/add/controller.php

profile/user/add/ no controller! => profile/user/controller.php

profile/user/ no controller! => profile/home/controller.php

profile/home/ no controller! => home/home/controller.php

home/home/ no controller! => bigbang :( go to home sleep

خب امیدوارم که با نقشه بالا روند پیدا کردن کنترل ماژول وابستتون رو پیدا کرده باشین و اینکه هیچ موقع به حالت 5 بر نخورید.

پس شما می توانید، تا حالت 4 ماژول وابسته بسازید، یعنی هرچیزی را می توانید از دید ماژول home/home بگذرانید و همه ماژول هایتان را وابسته کنید، که این به میل خود شماست.

خب بیایید یکسری صفحه پویا درست کنیم، چطور است؟

/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.

بی خیال، فکر کردین می خواهیم برای همه این آدرس ها ماژول بسازیم!

اگر خوب متوجه شده باشید، قرار است همه این ماژول های وابسته از کنترلر /home/home استفاده کنند.

پس بیایید در کنترلر قواعدی را بسازیم تا دیگر خطای 404 تولید نشود، و آدرس های ما معتبر باشد.

در تابع config فایل /home/home/controller.php خط های زیر را اضافه نمایید.

$this->listen(array(

"max" => 2,

"url" => array("javad","noshahr")

)

);

$this->listen(array(

"max" => 1,

"url" => array("saman" => "ramhormoz")

)

);

$this->listen(array(

"max" => 2,

"url" => array("/.\*/", "cityNumber" => "/^\d+$/")

)

);

برای توضیحات تکمیلی شما را ارجاع می دهم به آموزش [صفحه یاب ها](#_صفحه_یاب_ها_1)، همانطور که مشاهده کردید، اگر صفحات را در مرورگر اجرا کنید دیگر خطای 404 نخواهید داشت.

# ساخت فرم

## ساخت یک فرم عادی

در کلاس view که می سازید 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](#_ساخت_عضو_خصوصی).

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">

## بیایید مثال را کمی پیچیده تر کنیم.

$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/lib/forms/Extends.lib.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.

## چطور است کمی سطح کار را بالا ببریم!

$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 اینجا را بخوانید [ساخت گروه فرمی عمومی](#_ساخت_گروه_فرمی_1).

## چطور است یک حالت دیگر را تست کنیم

home/home/view.php config()

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

home/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 می توانید از هر سه حالت بالاتر آن استفاده کنید. برای اطلاعات بیشتر اینجا را بخوانید [ساخت گروه فرمی خصوصی](#_ساخت_گروه_فرمی).

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 می توانید 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(){

$this->debug->msg("تست اطلاعات");

}

}

?>

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](#_ساخت_یک_فرم) مراجعه کنید.

## ساخت عضو عمومی فرم

## ساخت عضو خصوصی فرم

# صفحه یاب ها

برای اینکه شما بتوانید آدرس های اضافی در آدرس بار مرورگر داشته باشین، و یا اصلا صفحه را بسازید که دارای هیچ ماژولی نمی باشد. باید در پوشه کنترلر خود در تابع config از تابع $this->listen استفاده نمایید. این تابع دارای دو مقدار می باشد که هر دو مقدار باید آرایه ای باشند.

## مشخصات قابل قبول در مقدار اول

|  |  |  |
| --- | --- | --- |
| نام فارسی | توضیحات | نام انگلیسی |
| حداکثر آدرس | حداکثر اسلش / که در آدرس بار معتبر است، عددی | max |
| حداقل آدرس | حداقل اسلش / که در آدرس بار معتبر است، عددی | min |
| آدرس ها | نوع آدرس ها که مقدار آرایه ای میگیرد اگر آرایه های شما داری اندیس نباشد می توانید پشت سر هم وارد کنید، و حتی می توانید از عبارات ویژه استفاده نمایید  اما اگر دارای اندیس بود می توانید مقدار اندیس را هم در شرط بگذارید | url |
| ایستا بودن آدرس | اگر این مقدار true باشد آدرس ها به ترتیب اعتبارسنجی می گردند، مقدار پیش فرض false، نوع مقدار بولین.  تنها موقعی استفاده می گردد که در آدرس علامت (=) داشته باشیم | static |

دو مقدار اول که نیاز به توضیح ندارد، اما با ما همراه باشید تا مقدار آخر را توضیح دهیم.

1. user/12 : array("user", "12");
2. user/1, …, user/14 : array("user", "/^\d+/$");
3. user/id=1, …, user/id=14 : array("user", "/^\d+/$");
4. user/id=1, …, user/id=14 : array("user", "id" => "/^\d+$/");
5. user/id=1, …, user/id=14 : array("user", "/^id=\d+$/");

تفاوت مثال 3 با مثال 4 این است که شما در مثال 3 نام اندیس برایتان مهم نیست و تنها میخواهید مقدارش برابر عددی باشد، پس در مثال 3 متن زیر نیز معتبر است.

user/name=5

user/type=8

تفاوت مثال 4 با مثال 5 این است که شما در مثال 4 برایتان مهم نیست که id در کدام خانه از اسلش های مرور آدرس قرار دارد اما در مثال 5 میگویید که id حتما باید در اسلش دوم باشد و اندیس دار باشد که اندیس آن id و مقدارش عددی است.البته اگر مقدار static برابر true باشد دیگر نیازی به مثال 5 نیست.

پس در مثال 4 متن زیر معتبر است.

user/id=14

user/name/id=124

user/name=ali/id=2

## مشخصات قابل قبول در مقدار دوم

توضیح ندادی!

# ساخت یک فرم استاندارد

برای ساخت یک فرم استاندارد تنها از دوحالت [ساخت گروه فرمی عمومی](#_ساخت_گروه_فرمی_1) و [ساخت گروه فرمی خصوصی](#_ساخت_گروه_فرمی) استفاده نمایید.

زیر فرم هایی همانند submit, hidden input, gender,… که قرار از در چندین فرم قرار بگیرد را به صورت [ساخت عضو خصوصی فرم](#_ساخت_عضو_خصوصی) 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.

سعی کنید تا حد امکان از [اعتبار سنجی ها](#_اعتبارسنجی) استفاده کنید و خطاگیری را برای خودتان تنظیم نمایید.

## ساخت گروه فرمی 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 گروه فرمی ساخت که همه جا در دسترس باشد باید در آدرس core/lib/forms یک فایل ایجاد کرد با نام یکتای فرم.

به عنوان مثال:

core/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 کاربریات را این جا وارد کنید”);

$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 گروه فرمی خصوصی باید [اعتبارسنجی](#_اعتبارسنجی) را در همان تابع انجام دهید، تا در قسمت مدل قابل دریافت باشد.

در این حالت برای این که در 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 3 تابع ایجاد کننده کلاس را تشکیل دادیم و از خط 4 تا 7 سه مقدار برای فرممان اختصاص دادیم.

در خط 4 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 تگتان استفاده کنید.

درضمن هر تابعی را می توانید به کار ببرید که نام تابع نام خصیصه شما می شود و مقدار ارسالی آن مقدار خصیصه.

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 می شود:

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

حال فرض کنید نام خصیصه شما دارای – (دش) و (\_) زیرخط بود:

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

که در حالت نمایش html می شود:

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

پس اگر در نام خصیصه یک \_ (زیرخط) بود در html می شود – (دش) و اگر دو زیرخط یا بیشتر بود می شود \_ (زیرخط).

## ساخت گروه فرمی 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 گروه فرمی خصوصی ساخت باید در همان فایل view.php ماژولتان یک کلاس با نام forms باز کرد.

به عنوان مثال:

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 کاربریات را این جا وارد کنید”);

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

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

}

}

?>

[اعتبارسنجی](#_اعتبارسنجی) اعتبار سنجی فراموش نشود.

در این حالت برای این که در view قابل دسترسی باشد می توانید نام فرم را با علامت @ صدا بزنید

مثلا:

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

# اعتبارسنجی

برای اعتبار سنجی در هر نوع تعریف از تعاریف چهارگانه فرم که هستید می توانید این کار را انجام بدهید.

ما دو نوع اعتبارسنجی داریم یکی این که جایی تعریف کنیم و در فرممان از آن استفاده نمیاییم. این حالت زمانی به کار می آید که ما از یک اعتبارسنجی چند جا نیاز داریم، مثلا اعتبارسنجی فرم های فارسی که شامل فرم های نام، نام خانوادگی، نام پدر، نام کلاس، نام صفحه ها و ... می باشد، پس بهتر است از اعتبارسنجی عمومی استفاده کنیم.

گاهی اوقات فقط قرار است که اعتبارسنجی در یک فرم لحاظ شود، خب اینگونه بهتر است که از اعتبارسنجی خصوصی استفاده شود.

و گاهی هم نیاز از از ترکیب این دو استفاده شود.

## اعتبارسنجی خصوصی

در این حالت شما فرم را فراخوانی کرده و تابع validate() را به انتهای آن اضافه می نمایید.

به عنوان مثال در حالت فرم سازی عمومی نگاه کنید:

$name = $this->make(“text”);

$name->validate()->reg(“/^\w+$/”)

->form(“set”, “این فرم تعریف نشده است”)

->form(“reg”, “مقدار فرم باید بیش از یک حرف باشد”)

->sql(“unique”,”این نام موجود می باشد”);

در مثال ما در خط 1 فرم را تعریف کردیم و از خط 2 اعتبارسنجی را مشخص نمودیم، توجه داشته باشید که حتما شبیه ما 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.

در خط 2 ما گفتیم که فرممان باید بیش از یک حرف باشد.

در خط 3 گفتیم که اگر به طور محتویات ارسالی ما این فرم را نداشت چه خطایی تولید کند.

در خط 4 گفتیم که اگر مخالف خط 2 بود چه خطایی تولید کند.

شما هرگونه اعتبارسنجی که برای فرم لحاظ کردید را می توانید با تابع form (که دو ورودی می گیرد، یکی نام اعتبار سنج و دیگری خطای تولید شده) خطایش را تولید کنید.

توابع اعتبارسنج فعلی: reg, username, password, repassword, date, number, farsi و ... می باشد که اگر از این ها همانند خط 2 استفاده کردید موظفید برای تولید خطایش همانند خط 4 استفاده کنید، درضمن می توانید از چندین تابع نیز استفاده نمایید.

در خط 5 ما تابع sql را داریم که همانند form عمل می کند با این تفاوت که خطاهای پایگاه داده ای را نمایش می دهد، یعنی اگر یکتا بودن، ایندکس بودن و متصل بودن ها دچار مشکل شد، این تابع خطاسازی را انجام می دهد.

در خط 5 ما گفتیم که اگر یکتابودن مقدار این فرم دچار مشکل شد چه خطایی نمایش داده شود. شما می توانید همزمان از چندخطاساز استفاده نمایید.

## اعتبارسنجی عمومی

این نوع اعتبارسنجی زمانی به کار می آید که می خواهید برای فرم های متعددی اعتبارسنجی را صورت دهید. مثل همان مثال همیشگی: نام، نام خانوادگی، نام پدر، نام کلاس و ... که همگی این ها باید فارسی و بین 3 تا 32 حرف باشند.

برای این کار لازم است در پوشه core/lib/FormsValidate یک کلاس درست کنید.

مثال:

core/lib/FormsValidate/faname.lib.php

<?php

class FormsValidate\_faname\_lib extends FormsValidate\_lib{

public $farsi = [3, 32];

public $form = array(

"set" => "مقدار فرم فارسی اشتباه است",

"farsi" => "مقدار فرم فارسی باید بین 3 تا 32 حرف فارسی باشد"

);

}

?>

شما می توانید مقادیر خود را به صورت مستقیم به کلاس اضافه کنید همانند مثال بالا و یا این که در تابع config این مقادیر را تنظیم نمایید.

دو کلمه کلیدی این کلاس form, sql است که مقدار آرایه ای ایندکس دار می گیرند. اگر یادتان باشد این ها همان تولید کننده های خطاها هستند در قسمت قبلی.

و یک کلمه کلیدی دیگر هم رزرو شده است به نام extend که یا مقدار حرفی می گیرد یا آرایه ای که می توان قبل از اجرای این کلاس کلاس اعتبارسنج دیگری را در خود گشایش extend دهد.

بقیه مشخصه ها (property) هم اگر در تابع validate وجود داشته باشند اجرا می گردند و مقدارشان یا آرایه است یا غیر آن که یعنی آرگومان های ارسال به آن تابع مثلا در خط 2 ما تابع farsi در اعتبار سنج را با دو مقدار 3,32 فراخوانی کردیم، که در تابع اعتبار سنج این دو مقدار یعنی حداقل و حداکثر حرف فارسی، حال خیلی راحت می توانیم این تابع را به فرم اضافه نماییم.

مثال:

$fathername = $this->make(“text”);

$fathername ->validate(“faname”)->form(“farsi”,”نام پدر اشتباه است”);

$name = $this->make(“text”);

$name ->validate(“faname”)->form(“farsi”,”نام اشتباه است”);

همانگونه که می بینید، ما مقدار faname که همان اعتبارسنج عمومی بود را به فرم اضافه نمودیم و مقدار خطای فارسی را دستی وارد کردیم، شما می توانید بی نهایت اعتبارسنج را به فرم اضافه کنید که به صورت آرگومان به تابع validate ارسال می شود.

شما می توانید از تابع extend نیز استفاده کنید.

مثال$name = $this->make(“text”);

$name ->validate()->extend(“faname”, “text”)->form(“farsi”,”نام اشتباه است”);

در مثال بالا ما اول اعتبارسنج faname را اضافه نمودیم و بعد از آن اعتبارسنج text را اضافه کردیم و در نهایت رویداد form.farsi را اصلاح کردیم.

# دسترسی به پایگاه داده ها

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.

## حالت اول اتصال در کلاس مدل

در این حالت شما به راحتی می‌توانید در خود کلاس مدل از طریق تابع sql به دستورات mysql دسترسی داشته باشین.

این تابع یک آرگومان اختیاری می‌گیرد. این آرگومان شناسه کوئری شما می‌شود که به شما و توسعه دهنده ها کمک می‌کند تا هر موقعی که خواستید اطلاعات کوئری را با آن شناسه دریافت کنید.

model.php

$this->sql()->select(“history”);

$this->sql(“history”)->select(“history”);

## حالت دوم اتصال توسط تابع محلی

در این حالت، شما در هر سه کلاس model, view & controller می توانید از تابع sql استفاده نمایید و اطلاعات خود که در تابعی در مدل تعبیه شده است را دریافت و یا ارسال کنید.

تابع محلی که در مدل می سازید باید پیشوند sql\_ داشته باشد.

نحوه فراخوانی به این صورت است که اسم تابع را بدون پیشوند sql\_ و با علامت @ فراخوانی می کنید.

model.php

make => public function sql\_userLoginTime() {… return sql\_query or anythings}

call =>

model.php => $this->sql(“@userLoginTime”);

view.php => $this->sql(“@userLoginTime”);

controller.php => $this->sql(“@userLoginTime”);

query\_...\_cls => $this->sql(“@userLoginTime”);

توجه داشته باشید که در حین فراخوانی می توانید بی نهایت آرگومان به تابع sql\_userLoginTime ارسال کنید.

## حالت سوم استفاده از کلاس های عمومی

در این حالت فرض کنید که شما یک کوئری خاص و یا یک مجموعه کوئری مرتبط دارید که در اکثر مواقع قصد دارید به آن ها دسترسی داشته باشین، در این حالت قاعدتا باید خارج از محدوده محلی این کلاس ها تعریف شوند، شما می توانید در پوشه core/cls/query کلاس خود را ایجاد کنید و در توابع آن کلاس شبیه حالات یک و دو کوئری های خود را بنویسید.

ساخت یک کلاس برای یک کوئری:

core/cls/query/user.cls.php

<?php

class query\_user\_cls extends query\_cls{

function config($id){

return $this->sql()->select(“users”)->where(“user\_id”, $id)->limit(1)->assoc();

}

}

?>

ما یک کلاس ساختیم به نام user که زیرمجموعه کلاس query می باشد و توسط کلاس query\_cls بسط داده شده است.

تابع config یعنی: زمانی که مستقیما نام کلاس را فراخوانی کردیم منظورمان این تابع است، همان constructor ما می باشد البته زمانی که نام کلاس تنهایی فراخوانی شود.

حال در هر کجا (model, view, controller) که خواستید این تابع را فراخوانی کنید، کافیست در تابع sql نام کلاس را با پیشوند نقطه (.) صدا بزنید.

بیایید باهم کلاسمان را در یک view فراخوانی کنیم.

view.php => config()

$user = $this->sql(“.user”, 12);

در مثال بالا ما در یکی از view ها کلاس user را فراخوانی کردیم که اولین آرگومان درخواستی کلاس ما عدد 12 می باشد.

شما می توانید در mode,controller هم این کار را انجام دهید، و یا حتی در یکی از کلاسه‌های دیگر query\_cls.

حال فرض کنید شما یک مجموعه کوئری دارید که با هم مرتبط اند و نیاز دارند در یک دسته قرار گیرند.

core/cls/query/user.cls.php

<?php

class query\_user\_cls extends query\_cls{

function byId($id){

return $this->sql()->select(“users”)->where(“user\_id”, $id)->limit(1)->assoc();

}

function byUsername($usrname){

return $this->sql()->select(“users”)->where(“username”, $usrname)->limit(1)->assoc();

}

function byEmail($email){

return $this->sql()->select(“users”)->where(“email”, $email)->limit(1)->assoc();

}

}

?>

فرض کنید ما می خواهیم مشخصات کاربرانمان را داشته باشیم، گاهی از طریق id و گاهی هم از طریق username Or email پس این ها همگی می توانند در یک کلاس نوشته شوند.

حال بیایید نحوه فراخوانی را تمرین کنیم.

view.php => config()

$user = $this->sql(“user.byId”, 12);

$user = $this->sql(“user.byUsername”, “baravak”);

$user = $this->sql(“user.byEmail”, “itb.baravak@gmail.com”);

در این ساختار شما دیگر نیاز به پیشوند نقطه ندارید. نام کلاس را می نویسید یک نقطه (.) می گذارید و نام تابعی که قصد ورود به آن را دارید بعد از آن می نویسید، به همین راحتی :) .

# ساخت لیست دیتا

این کلاس به شما کمک می کند تا دیتاهایی را که از طریق کوئری های پایگاه داده ها می‌گرین را برایتان در ساختاری زیبا پیاده سازی کند و با یک ماکروی راحت سمت html آن ها را ترجمه و درون تگ ها قرار دهید.

## توابع

\_\_construct(array $title, array $query)

سازنده کلاس ما می باشد.

آرگومان اول: آرایه ای برای عنوان سلول ها که این آرایه اندیس دار است، اندیس آرایه نام یکتای سلول ها و مقدار آرایه عنوان سلول ها قرار می گیرد.

آرگومان دوم: آرایه ای از کوئری های گرفته شده از پایگاه داده ها.

باید توجه داشت که نام فیلد های کوئری باید برابر اندیس های آرگومان اول باشد.

addCol(int $index, string $name, string $label)

سلولی را به سطر های شما در مکانی مشخص اضافه می کند.

آرگومان اول: مکان قرار گیری سلول. از اندیس صفر (0) تا آخرین اندیس می توانید بگذارید.

آرگومان دوم: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان سوم: عنوان سلول.

addCol(string $name, string $label)

سلولی را به سطر های شما بعد از آخرین سلول اضافه می کند.

آرگومان اول: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان دوم: عنوان سلول.

addColBefore(string $before, string $name, string $label)

سلولی را قبل از سلول مشخص اضافه می کند، اگر سلول مشخص وجود نداشته باشد در اولین سلول ایجاد می شود.

آرگومان اول: سلولی که قرار است قبل از آن ایجاد شود.

آرگومان دوم: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان سوم: عنوان سلول.

addColAfter(string $after, string $name, string $label)

سلولی را بعد از سلول مشخص اضافه می کند، اگر سلول مشخص وجود نداشته باشد در آخرین سلول ایجاد می شود.

آرگومان اول: سلولی که قرار است بعد از آن ایجاد شود.

آرگومان دوم: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان سوم: عنوان سلول.

addColFirst(string $name, string $label)

سلولی را به اندیس اول اضافه می کند

آرگومان اول: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان دوم: عنوان سلول.

addColEnd(string $name, string $label)

سلولی را به اندیس آخر اضافه می کند

آرگومان اول: نام یکتای سلول، اگر این نام موجود باشد جایگزین قبلی می گردد.

آرگومان دوم: عنوان سلول.

removeCol(string $name)

سلولی را از طریق نام یکتای آن حذف می کند.

آرگومان اول: نام یکتایی که قرار است با آن، سلول حذف شود.

removeCol(int $index)

سلولی را از طریق موقعیت آن حذف می کند.

آرگومان اول: موقعیت سلولی که قرار است با آن، سلول حذف شود.

row(int $index)

سطری را انتخاب می‌کند.

اگر قرار باشد یک سطر تغییر کند، باید قاعدتا قبل از آن اتنخاب (select) شود.

آرگومان اول: موقعیت سطر،‌ اگر عدد منفی یک (-1) را وارد کنید،‌ یعنی همه سطرها.

col(string $name)

سلولی را از طریق نام یکتای آن انتخاب می‌کند.

برای اینکه بتوانیم در سلول ها نوشته و یا تگی اضافه و یا حذف کنیم باید از این راه استفاده کنیم.

آرگومان اول: نام یکتای سلول.

col(int $index)

سلولی را از طریق موقعیت آن انتخاب می‌کند.

برای اینکه بتوانیم در سلول ها نوشته و یا تگی اضافه و یا حذف کنیم باید از این راه استفاده کنیم.

آرگومان اول: موقعیت سلول.

select(int $rowIndex, int $colIndex || string $colName)

سطر و سلول را انتخاب می کند.

آرگومان اول: موقعیت سطر را مشخص می کند،‌ اگر عدد منفی یک (-1) را وارد کنید،‌ یعنی همه سطرها.

آرگومان دوم: نام یکتا و یا موقعیت سلول را مشخص می کند.

html(string $string)

مقداردهی داخل سلول ها از طریق متن.

آرگومان اول: متنی که قرار است در سلول یا سلول های انتخاب شده قرار گیرد.

شما می توانید در این حالت به صورت پویا از یکتاهای سلول ها در متون داخل این تابع استفاده کنید مثلا:

$show->html(“<b>%username%</b>”);

اگر نوشته شما بین دو درصد (%) قرار گیرد، به همان سطر رجوع می کند و از یکتای username مقدار آن را بر می گرداند.

html(object $tags)

مقداردهی داخل سلول ها از طریق شی tags\_lib.

در این شی شما می توانید یک ساختار تگ با خصیصه ها و فرزندانش بسازید و می تواند از %colname% مثل حالت بالا مقدار یکتای سلول ها را داخل تگ ها بگنجانید.

آرگومان اول: شی کلاس tags\_lib.

مقدار یکتای سلول هایی که از طریق درصد ٪ مشخص می شوند. می توانند در tagName، اندیس attr و مقدار attr قرار گیرند.

compile()

مقدار آرایه ای و نهایی لیست شما را بر می گرداند و شما می توانید به راحتی از آن در ماکرو ها استفاده کنید.

## مثال

فرض کنید می خواهید لیستی از کاربرانتان را از سمت پایگاه داده ها بگیرین، فلید های user\_id, password را حذف کنید و به جای آن در خانه ای از لیستتان تنظیمات و حذف را بگذارید، این کار کمتر از چند دقیقه طول می کشد.

view.php => config()

$users = $this->sql(“.users”, 100);

$usersTitle = array(“username” => “نام کاربری”, “email”=> “نامه برقی”);

$list = $this->show($userTitle, $users)

->addColEnd(“delete”, “حذف”)

->select(-1, “delete”)

->html(“<a href=’user/delete/id=%user\_id%’>حذف %username%</a>”)

->compile();

خط 1: درخواستی از پایگاه داده ها گرفتیم.

خط 2: عنوان سلول ها را همراه با یکتای آن ها مشخص کردیم.

خط 3: آرگومان اول که عنوان سلول ها و آرگومان دوم که آرایه کوئری ها بود را به کلاس فرستادیم.

خط 4: یک سلول با نام یکتای delete و عنوان حذف در آخرین سلول ساختیم.

خط 5: همه سطر ها و سلول delete را انتخاب کردیم.

خط 6: در سلول انتخاب شده متنی را نوشتیم که تگ html است و از علامت درصد % برای جایگزاری استفاده کردیم.

خط 7: لیست را به صورت آرایه ای ترجمه کردیم.

# دریافت اطلاعات در کلاس مدل

فرض کنید فرممان را در view ساختیم، و کاربر برروی دکمه submit کلید کرده است، حال ما باید در قسمت مدل اعتبارسنجی و آنالیز را انجام دهیم.

نحوه اعتبارسنجی بستگی به نحوه ایجاد فرم می باشد، باید بگویم اگر از سه حالت .,@,# برای ساختن فرمتان استفاده کردید کارتان راحت است. ما تک تک نحوه اعتبارسنجی ها را برایتان توضیح می دهیم.

## اعتبارسنجی گروه فرم ها (عمومی، خصوصی)

اگر از دو حالت .,@ استفاده کردید، کافیست به صورت زیر عمل کنید.

$forms = $this->validate(“@login”); //$forms = $this->validate(“.login”);

$chckeLogin = $this->sql(“checkLogin”)->select(“users”)->filed->(array(

“username” => $forms->username,

“password” => $forms->password

))->num();

خط 1: نام گروه فرمی را به تابع validate سپردیم تا اعتبارسنجی را انجام دهد، بازگشت گروه فرمی یک شی است که اعضایش نام فرم هایی است که ساخته شده. و تک تک اعضا یک کلاس validate\_lib می باشند.

خط 2: یک کوئری از دیتابیس ساختیم و در قسمت فیلد ها مشخص کردیم که هر شی برای کدام فیلد از دیتابیس می باشد، حال کلاس sql خودش قبل از ارسال شی ها را تبدیل به کاراکتر می کند و دیتا را ارسال می نماید.

## اعتبارسنجی فرم های عمومی

اگر از حالت # برای ساختن فرم ها در قسمت view استفاده کردید به صورت زیر عمل کنید

$username = $this->validate(“#username”);

$password = $this->validate(“#password”);

$chckeLogin = $this->sql(“checkLogin”)->select(“users”)->filed->(array(

“username” => $username,

“password” => $password

))->num();

## اعتبارسنجی دستی

اگر از هیچ یک از سه حالت بالا برای ساخت فرم استفاده نکرده اید، یا می خواهید آدرس صفحه خود را اعتبارسنجی کنید از حالت زیر استفاده کنید.

فرض کنید می خواهیم از آدرس صفحه کد یکتای کاربر را دریافت کنیم و اعتبارسنجی نماییم.

url: profile/user/edit/id=160

$idValidate= $this->validate()->reg(“/^\d+$/”)->form(“set”, ‘کد کاربری وارد نشده است’)->set(“reg”, ‘کد کاربری اشتباه است’);

$id = $this->url(‘id’, $idValidate);

خط 1: در خط یک ما یک اعتباسنج ایجاد می کنیم. که تمام دستورات [اعتبارسنجی](#_ساخت_یک_فرم) مو به مو اینجا اجرا می شود.

خط 2: در تابع url آرگومان اول می تواند عدد یا کاراکتر باشد که موقعیت آدرس را مشخص می کند. و آرگومان دوم همان اعتبارسنج ما است.

باید توجه داشت که اگر قصد اعتبارسنجی یک فرم را داشتیم باید از $this->post به جای $this->url استفاده کنیم.

# 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. این کار به صورت پیشفرض در model ها به کار می رود، تا صفحه بعد از آنالیز به مبدا اولی منتهی شود.

اما برای تغییر دستی و یا ساخت یک انتقال دهنده در model, view & controller می توانید از تابع $this->redirect() استفاده کنید. که این تابع بازگشت یک کلاس انتقال دهنده است و می توانید از افراد آن استفاده کنید.

$this->redirect(string $redirec, boolean $exit = true, boolean $php = false)

سازنده تابع انتقال دهنده در model, view & controller.

آرگومان اول: (پیشفرض false) مشخص می کند که می خواهید بدون چون و چرا به این آدرس منتقل شوید. اگر مقدار این آرگومان false باشد، به این معنی است که می خواهید از ساختار همین هسته استفاده کنید، یعنی class, method, child, urls. یعنی شما می خواهید یک یا چند آدرس را از url فعلی ویرایش و یا حذف نمایید.

آرگومان دوم: (پیشفرض true) مشخص می کند که آیا بعد از ارسال دستور انتقال دهنده کد را ببند یا خیر. بهتر است true باشد.

آرگومان سوم: (پیشفرض false)مشخص می کند که آیا دستور انتقال دهنده یک کد php باشد یا یک کد Html. بهتر است false باشد.

->classChange(string $str = false, boolean $toEnd = false)

یکی از توابع انتقال دهنده که توسط این می توانید نام کلاس فعلی را تغییر دهید.

آرگومان اول: (پیشفرض false) نام کلاس را مشخص می کند، اگر false باشد یعنی کلاس home.

آرگومان دوم: (پیشفرض false) مشخص می کند که آیا مابقی آدرس فعلی را حذف کن یا خیر.

->methodChange(string $str = false, boolean $toEnd = false)

یکی از توابع انتقال دهنده که توسط این می توانید نام متد فعلی را تغییر دهید.

آرگومان اول: (پیشفرض false) نام متد را مشخص می کند، اگر false باشد یعنی متد home.

آرگومان دوم: (پیشفرض false) مشخص می کند که آیا مابقی آدرس فعلی را حذف کن یا خیر.

->childChange(string $str = false, boolean $toEnd = false)

یکی از توابع انتقال دهنده که توسط این می توانید نام فرزند فعلی را تغییر دهید.

آرگومان اول: (پیشفرض false) نام کلاس را مشخص می کند، اگر false باشد یعنی بدون فرزند.

آرگومان دوم: (پیشفرض false) مشخص می کند که آیا مابقی آدرس فعلی را حذف کن یا خیر.

->urlChange(string $str = false, string $replace)

یکی از توابع انتقال دهنده که توسط این می توانید مقدار مابقی آدرس ها را تغییر دهید یا حذف کنید.

آرگومان اول: (پیشفرض false) نام کلاس را مشخص می کند، اگر false باشد یعنی آدرس های اضافه بر کلاس، متد و فرزند را حذف کن. می تواند از نوع عددی و یا کاراکتری باشد.

آرگومان دوم: اگر آرگومان اول وجود داشت با این آرگومان تغییر می کند، درغیر اینصورت با این آرگومان ایجاد می شود.

## مثال

view.php => config()

$this->redirect()->classChange(“profile”)->methodChange(“edit”)->urlChange(0, 12)->urlChange(“name”, “baravak”);

باید توجه داشته باشید هربار که از $this->redirect() استفاده می کنید، اگر وجود داشته باشد، جایگزین می شود.

برای اینکه انتقال دهنده را ویرایش کنید نه ایجاد می توانید از $this->redirect استفاده کنید.

# ساخت فرزندان مشترک بین کلاس های ماژول

## ساخت تابع مشترک

فرض کنید در حالتی نیاز است که شما یک پروسه را هم در view و هم در model و یا در هر سه کلاس model, view & controller انجام دهید. خب برای این کار فقط کافیست تابع خود را در کلاس controller فراخوانی کنید. به لیست رزروی های کنترلر اضافه کنید.

لیست رزروی های کنترلر به این معنا است که اگر متد یا متغییری در دو کلاس model or view صدا زده شد و وجود نداشت، اگر در لیست رزروی های کنترلر باشد، آن تابع یا متغییر را در کلاس controller جستجو می کند.

به عنوان مثال فرض کنید ما می خواهیم از آدرس صفحه یک مقداری را هم در view و هم در model بررسی کنیم:

controller.php

function statusChecker($status){

if(isset(config\_lib::$aurl[‘status’]) && config\_lib::$aurl[‘status’] == $status ){

return true;

}

return false;

}

در کلاس کنترلر ماژولمان یک تابع نوشتیم به نام statusChecker تا مقدار status آدرس بار را با مقداری که به او می دهیم چک کنید و درستی و نادرسی تساوی را برایمان بازگرداند.

در مرحله بعد باید این تابع را به لیست رزروی های تابع معرفی کنیم.

controller.php -> config()

$this->addMethod(“statusChecker”);

در این خط ما به لیست رزروی های کنترلر یک متد به نام statusChecker اضافه می کنیم، این همان نام تابعی است که در مرحله قبل انجام دادیم.

حالا کافیست چه در model و چه در view آن ها را فراخوانی کنیم، مثلا:

view.php => config()

$status = $this->statusChecker(‘edit’);

## ساخت متغییر مشترک

همه توضیحات [ساخت تابع مشترک](#_ساخت_تابع_مشترک) در این عنوان هم می آید با این تفاوت که ما یک متغییر مشترک ایجام می کنیم و رزرو می نماییم، و اینکه تابع addMethod جایگزین تابع addPeroperty می شود.