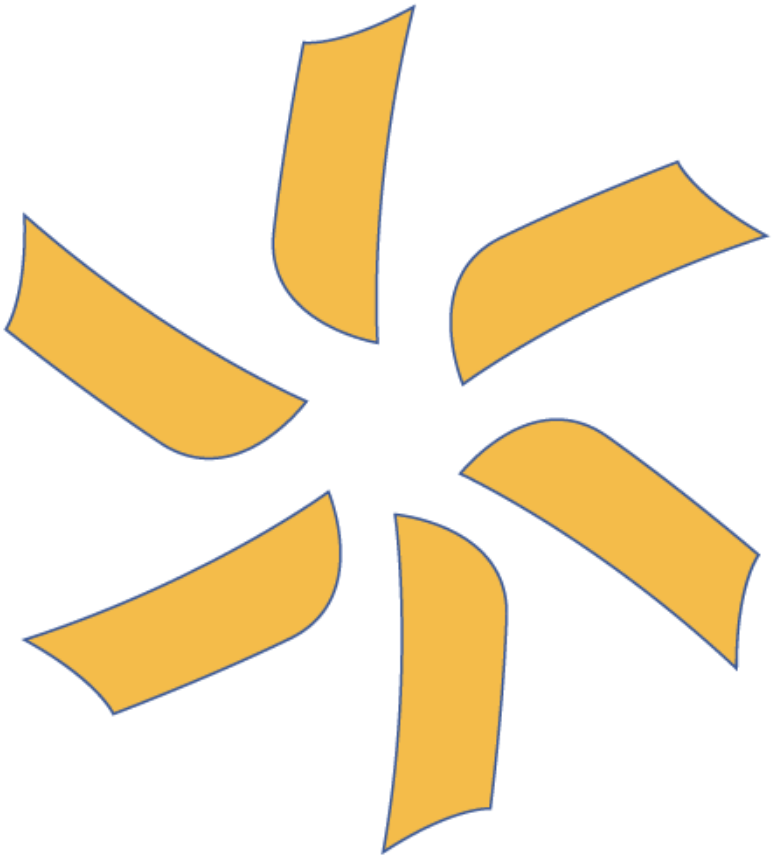

Documentação Easy Framework

Release 2.x

Easy Framework Foundation

Sumário

Iniciando

O EasyFw fornece uma robusta base para a sua aplicação. Ela pode controlar diversos aspectos, desde a requisição inicial do usuário até a renderização da página web. O framework segue os princípios do MVC, permitindo a você uma fácil customização e extensão na maioria dos aspectos da aplicação.

O framework também fornece uma organização lógica de sua estrutura, desde nomes de arquivos a nomes de tabelas do banco, deixando sua aplicação inteiramente lógica e consistente. Esse é um conceito simples mas poderoso. Siga as convenções e você saberá exatamente os as coisas estão e como elas estão organizadas.

A melhor maneira de aprender e experimentar o EasyFw é sentar e construir alguma coisa. Para começar vamos construir um Blog simples.

Construindo um blog

Bem-vindo ao EasyFw. Você está conferindo esse tutorial provavelmente porque quer conhecer mais sobre como o EasyFw funciona. É nossa meta aumentar a produtividade e deixar a tarefa de codificar mais prazerosa: nós esperamos que você veja isso enquanto você codifica.

Esse tutorial vai levar você através da criação de um blog simples. Nós vamos baixar e instalar o EasyFw, criar e configurar o banco de dados e criar a lógica suficiente para listar, adicionar, editar e excluir postagens do blog.

Aqui está o que nós vamos precisar:

1. Um servidor web. Vamos assumir que você está usando o Apache, apesar disso as instruções para usar outro servidor web são bem similares. Talvez nós precisaremos brincar um pouco com as configurações do servidor, mas na maioria das vezes você colocará o EasyFw para rodar sem configurar nada. Tenha certeza que você está rodando o PHP 5.3.0 ou maior.

2. Um servidor de banco de dados. Nós vamos utilizar o servidor MySQL nesse tutorial. Você precisará entender bem a linguagem SQL para criarmos um banco de dados. Como nós vamos utilizar MySQL, tenha certeza que você tem a extensão pdo_mysql instalada (normalmente já vem instalada por padrão nos pacotes).

3. Conhecimento básico de PHP. Quanto mais orientação a objetos você conhecer melhor, mas não tema caso você seja fã de programação procedural.

4. Finalmente, você precisará conhecer o básico do padrão de programação MVC. Uma pequena revisão pode ser encontrada em *Entendendo o Model-View-Controller*. Não se preocupe, é apenas meia página ou menos.

Vamos começar!

Baixando o Easy Framework

Vamos baixar um esqueleto de uma aplicação do EasyFw, para isso vamos clonar o repositório do Easy Skeleton Application.

git clone git://github.com/LellysInformatica/easyskeleton.git --recursive

Dessa maneira o EasyFw já virá junto com a aplicação dentro da pasta vendor.
Caso você não queira clonar o EasySkeleton, apenas baixa-lo normalmente, então você deverá baixa o EasyFw também e coloca-lo na pasta vendor com o nome easyframework.

Por padrão o EasyFw virá com a seguinte estrutura:

- Sua_aplicacao
    - App
    - Vendors
        - Easyframework
    - .htaccess
    - README.md

Agora seria uma boa hora para você conhecer a Eastrutura de pastas do EasyFw

Criando o Banco de dados

Agora, vamos criar o banco de dados para nosso blog. Se você ainda não fez, crie um banco de dados em branco para usarmos nesse tutorial, com um nome de sua escolha. Vamos criar apenas uma tabela para guardarmos nossas postagens. Nós também vamos adicionar algumas postagens para testarmos nossa aplicação. Execute as seguintes instruções SQL:

```
/* Primeiro, criamos nossa tabela de postagens: */
CREATE TABLE posts ( id BIGINT UNSIGNED AUTO_INCREMENT
        PRIMARY KEY, titulo VARCHAR(50), conteudo TEXT, created
        TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ;

/* E vamos inserir alguns dados para testes: */
INSERT INTO posts ( title,body) VALUES ('O titulo, 'Isso é o conteúdo da postagem.');
INSERT INTO posts ( title,body) VALUES ('Outro titulo', 'outro conteudo da postagem.');
INSERT INTO posts ( title,body) VALUES ('Titulo 3', 'Isso realmente é legal.');
```

AS escolhas dos nomes das colunas não são por acaso. Se você seguir as convenções de banco de dados e de classes do EasyFw você poderá ter as vantagens de não precisar especificar nada em arquivos de configurações, pois o EasyFw já irá mapear tudo automaticamente (você pode conferir as convenções na seção Convenções do Easy Framework), mas nada impede que você use outro padrão.

Seguindo as convenções vamos ficar com a tabela com o nome de "posts", automaticamente ela será ligada ao modelo Posts.

Configuração do Banco de dados

Vamos dizer para o EasyFw onde nosso banco de dados está localizado. Para a maioria, essa será a primeira e última vez que você precisará configurar alguma coisa.

O arquivo de configuração do banco de dados está localizado em */App/Config/database.yml.*

O arquivo de configuração é bem simples, basta você trocar os valores para as informações do seu servidor de banco de dados. Em uma máquina local o arquivo ficará parecido com o abaixo:

```
datasource:

 development:
  default:
   driver: mysql
```

```
    host: localhost
    user: root
    password: '"'
    database: blog_db
    encoding: utf8
    prefix: ''

  production:
    default:
    driver: mysql
    host: localhost
    user: theUsername
    password: "justAPassToProduction"
    database: theDatabaseNameAtProduction
    encoding: utf8
    prefix: ''
```

Uma vez configurado o arquivo de banco de dados, você já pode acessar no seu browser a sua aplicação, e ver a tela de boas-vindas do EasyFw.

Note: Lembre-se que você precise ter a extensão PDO, e pdo_mysql habilitados no seu php.ini.

Configurações opcionais

Existem mais três itens interessantes que podem ser configurados. A maioria dos desenvolvedores completa essa sequência de configurações, mas elas não são obrigadas para esse tutorial. A primeira é definir uma string aleatória (ou salt) que será usado nas criptografias de segurança. O segundo é definir um número randômico (ou seed) para encriptação. E o terceiro item é permitir que o EasyFw escreva no diretório *tmp.*

O salt de segurança é utilizado para hashes. Para alterar o valor padrão basta abrir o arquivo de configurações gerais */App/Config/application.yml* na linha 52.

```
Security:
  salt: 'DYhG93b0qyJfIxfs2guVoUubWwvniR2G0FgaC9mi'
```

O chipher seed é utilizado para encriptar e decriptar strings. Para alterar o valor padrão basta abrir o arquivo de configurações gerais */App/Config/application.yml* na linha 53.

```
Security:

  cipherSeed: '768593096574535424967496836 45'
```

A última tarefa é dar permissão para pasta *tmp*.

```
$ chown -R www-data app/tmp
```

Se por alguma razão o EasyFw não puder escrever no diretório *tmp* você será informado com um alerta, contanto que não esteja em modo de produção.

Uma observação no mod_rewrite

Ocasionalmente um novo usuário pode encontrar problemas com o mod_rewrite, por isso vamos falar um pouco sobre ele. Se por alguma motivo sua página de boas-vindas parecer estranha, sem css, imagens ou até mesmo um erro 500 no servidor, isso pode ser algum problema com o mod_rewrite. Então aqui vão algumas dicas para tudo dar certo:

- Tenha certeza que a sobrescrita do .htaccess está permitida no seu httpd.config, lá deve ter uma seção que permite essa sobrescrita. Tenha certeza que a opção AllowOverride está setada para All.
- Garanta que você está editando o httpd.config correto ao invés do httpc.config do website ou usuário específicos.

- Tenha certeza que o Apache está carregando o mod_rewrite corretamente!

Isso permitirá que suas URLs fiquem legíveis, por exemplo, ao invés www.example.com/index.php/controllername/actionname/param ficará www.example.com/controllername/actionname/param.

Se você está instalando o EasyFw em um servidor diferente do Apache, você poderá encontrar as instruções de reescrita de URLs na Instalação Avançada.

Tutorial do Blog – Adicionando camadas

Criando um Modelo

O modelo é o pão e a manteiga das aplicações do EasyFw. Criando um modelo no EasyFw nos permitirá interagir com o nosso banco de dados, vamos ter as operações de um CRUD básico já moldadas para utilizarmos mais tarde.

O modelo do EasyFw deve ser colocado na pasta dos modelos /App/*Model* para nosso exemplo vamos salva-lo da seguinte maneira */App/Model/Post.*php. O arquivo deverá conter a classe Post, mais ou menos dessa forma:

```
<?php

namespace App\Model;

class Post extends AppModel
{

}
```

A convenção de nomes no EasyFw é muito importante. Colocando o nome de nosso modelo como Post automaticamente o EasyFw infere que esse modelo será utilizado pelo controller PostsController, e está ligado a uma tabela no banco de dados com o nome posts.

Nota: O EasyFw dinamicamente cria um objeto do modelo para você no seu controller, contanto que os nomes sigam o padrão e o modelo esteja localizado em */App/*Model. Isso significa que se por acidente o nome da classe ou arquivo de modelo estiver incorreto, o EasyFw não irá reconhece-lo e você precisará dizer explicitamente que o nome é diferente.

Criando o Controller de postagem

Próximo passo vamos criar o controller para nosso post. O controller é o que irá ligar a lógica de negócio do nosso modelo e mostrará uma View para o usuário.

Nós vamos grava o nosso novo controller com o nome *PostsController.php* dentro da pasta */App/Controller* Aqui está como o controller deve estar:

```
<?php

namespace App\Controller;

class PostsController extends AppController
{

}
```

Agora, vamos adicionar uma action para nosso controller. Uma Action é a representações de funcionálidades da nossa aplicação, nada mais são que métodos. Por exemplo, quando um usuário acessar www.example.com/posts/index (que é o mesmo que www.example.com/posts), eles esperarão ver uma listagem das postagens. O código para essa action é algo assim:

```php
<?php

namespace App\Controller;

use Easy\Model\FindMethod;

class PostsController extends AppController
{
   public function index()
   {
      $postagens = $this->Post->getEntityManager()->find(null, FindMethod::ALL);
      $this->postagens =  $postagens;
   }
}
```

Deixe-me explicar essa action. Definindo uma action index() no nosso PostsController, os usuários podem acessar sua lógica através do cominho www.example.com/posts. Logicamente, se nós definirmos um método chamado foobar(), os usuários poderão acessa-la pela url www.example.com/posts/foobar

Atenção: Você pode se sentir tentado a nomear certos controllers e action para obter certas URLs. Resista a essa tentação siga as convenções de nomes do EasyFw (controllers no plural, etc.) e crie actions com os nomes legíveis e entendíveis. Você pode mapear suas URLs para seu código usando rotas, que vamos abordar mais a frente.

Na primeira linha nós atribuímos a uma variável $postagens nossa listagem das postagens do banco, para isso utilizamos nosso modelo $this->Post e acessamos o EntityManager, é com ele que temos as operações de CRUD do banco de dados. Após isso nós pedimos para ele utilizar o método find() e passamos dois parâmetros, o primeiro são as condições da busca (como filtros, ordenações, etc.) e o segundo é o tipo da busca, no nosso caso é FindMethod::ALL para buscar todos os registros.

Na outra linha, nós criamos uma variável dinamicamente, ela será nossa variável na View. Atribuímos a ela todas as nossas postagens e pronto já podemos acessa-la na View.

Para aprender mais sobre os Controllers, visite a seção de Controllers.

Criando a View de postagens

Agora que já temos o nosso fluxo de dados do nosso modelo e nossa lógica da aplicação, vamos criar uma view para nossa ação index() que criamos anteriormente.

As views do EasyFw são apresentações para o usuário final, o EasyFw trabalha com uma abstração das views para isso usamos o Smarty Template Engine. Para a maioria das aplicações as views consistem em HTML junto com o Smarty, XML, JSON, CSV.

Os layouts são representações de código que ficam ao redor da view, isso evita que nós tenhamos que ficar acresentando headers e footers em cada view que criarmos. Os layouts podem ser definidos para um controller ou action específicos, para esse tutorial vamos utilizar o Layout.tpl já existente.

As views são salvas na pasta */App/View/Pages* nessa pasta criamos outra pasta com o nome do controller (no nosso caso “Posts”) e salvamos nossa view com o nome da action, no caso index.tpl Vamos colocar as postagens em uma tabela, essa View ficara mais ou menos dessa form

```
{block name=”content”}

<div class="page-header">
   <h1>{__("Postagens do
Blog")}</h1>
</div>

<table class="table table-bordered table-striped">
       < tr >
              < th>Id</th >
              < th>Título</th >
              < th>Data</th >
              < th>Ações</th >
       < /tr >
       {foreach $postagens as $postagem}
        < tr >
              <td>{$postagem->titulo}</td>
              <td>{$postagem->created|date_format: “%x”}</td>
              <td>{$Html->actionLink(“Editar”, “view”, “posts”, $postagem->id)} |
              {$Html->actionLink(“Excluir”, “delete”, “posts”, $postagem->id)}
              </td>
       < /tr >
       {/foreach}
< /table >

{/block}
```

Você deve ter notado o uso de um objeto $Html->actionLink(). Esse tipo de objeto são chamados de Helpers. O EasyFw vem com diversos Helpers que você pode utilizar em suas views, deixando mais fácil de trabalhar. Por exemplo o helper que utilizamos acima irá gera o HTML de um link para a action view no controller posts.

O link criado pelo helper irá seguir o formato /controller/action/param1/param2.

Nesse ponto você já pode acessar seu browser http://www.example.com/posts/index. Você deve ver todas as postagens que o nosso banco tem.

Caso você clique no link de edição você será informado pelo EasyFw que a action ainda não existe. Então nós vamos cria-la agora:

```php
<?php

namespace App\Controller;

use Easy\Model\FindMethod;

class PostsController extends AppController
{
   public function index()
   {
      $postagens = $this->Post->getEntityManager()->find(null, FindMethod::ALL);
      $this->postagens =  $postagens;
   }
   public function view($id =null)
   {
      $postagem = $this->Post->getEntityManager()->read($id);
      $this->postagem =  $ postagem;
   }

}
```

O código para leitura de uma postagem é bem simples. Notem que estamos usando o *read()* ao invés do *find()* pois nós só queremos recuperar uma postagem.

Notem também que a nossa action recebe um id como parâmetro, esse id é da postagem que queremos ver. Esse parâmetro é provido para a action através da URL. Se o usuário acessar /posts/view/3, então o valor '3' é passado como o $id.

Agora vamos criar a View para nossa action 'view' e salva-la em /App/View/Posts/view.tpl.

```
{block name=”content”}

<div class="page-header">

   <h1>{$postagem->titulo}< </h1>

</div>

<p><small>Data: {$postagem->created|date_format: “%x”}< /small></p >

<p>{$postagem->conteudo}< /p >

{/block}
```

Veja que isso irá funcionar acessando o link pela tabela das nossas postagens ou acessando manualmente um post pela URL. Ex.:/posts/view/1.

Adicionando postagens

Ler registros do banco de dados e mostrá-los é um bom começo, mas vamos aprender como adicionar registros.

Primeiramente, vamos criar uma action chamada add() no nosso PostsController:

```
<?php

namespace App\Controller;

use Easy\Model\FindMethod;

class PostsController extends AppController
{
   public function index()
   {
      $postagens = $this->Post->getEntityManager()->find(null, FindMethod::ALL);
      $this->postagens =  $postagens;
   }
   public function view($id =null)
   {
```

```
    $postagem = $this->Post->getEntityManager()->read($id);
    $this->postagem =  $ postagem;
  }

      public function add()
      {
          if ($this->request->is('post')) {
              if ($this->Post->save($this->request->data)) {
                  $this->Session->setFlash('Sua postagem foi salva com sucesso.') ;
                  $this->redirectToAction('index') ;
              } else {
                  $this->Session->setFlash('Sua postagem não foi salva') ;
              }
          }
      }
}
```

A action add() faz o seguinte: se o método HTTP foi via POST, tente salvar os dados usando o modelo Post. Se por alguma razão não pode ser salvo, ele apenas irá renderizar a view. Isso nos dá a chance de mostrar os erros de validação para o usuário.

Cada request no EasyFw está encapsulado em um objeto chamado Request que pode ser acessado do seu controller pela propriedade $this->request. O objeto Request contém informações uteis sobre o request realizado, e pode ser utilizado para controlar o fluxo da sua aplicação. Nesse caso, nós usamos o método *Request::is()* para verificar se o request foi via POST.

Quando um usuário utiliza um formulário via POST para enviar dados para nossa aplicação, esses dados estão disponíveis através da propriedade $this->data ou $this->request->data.

Nós utilizamos o método *SessionComponent::setFlash()* do componente de Sessão para setar uma mensagem na sessão e mostrar para o usuário depois do redirecionamento. Na View nós utilizamos *SessionHelper::flash()* que vai mostrar a mensagem e limpar a variável da sessão. O método *Controller::redirectToAction()* redireciona a aplicação para outra action e executa seus procedimentos.

Chamar o método save() irá validar o modelo e se encontrar algum erro irá abortar o processo de salvamento. Vamos abordar os error de modelos mais a frente.

Validação de dados

O EasyFw trabalha com validações de uma forma fácil e rápida para que você não precise perder tempo com enormes rotinas de validação.

Para ter as vantagens da validação em formulários, nós vamos precisar utilizar o FormHelper na nossa view:

Aqui está nossa View de adição de postagens:

```
{block name=”content”}

<div class="page-header">
   <h1>{__("Adicionar Postagem")}</h1>
</div>

{$Form->create(‘add’, ‘posts’)}
   {$Form->inputTextLabel(‘titulo’)}
   {$Form->textAreaLabel(‘conteudo’, [‘rows’=>3])}
   {$Form->submit(‘Salvar’, [‘class’ => ‘btn’])}{$Form->close()}
{/block}
```

Aqui nós utilizamos o FormHelper para gerar nossa tag HTML de um formulário. Ele irá gerar a seguinte tag:

```
<form method="post" action="/posts/add">
```

O método $Form->inputTextLabel() é usado para gerar um element input. O primeiro parâmetro corresponde a qual campo ele pertence, o Segundo parâmetro é um array com opções para personalização do input, como css, onclick, id personalizado, etc. São chamados de HtmlOptions. O terceiro parâmetro é para a personalização da label que irá referenciar o input.

O método $Form->textAreaLabel() irá gerar uma tag textArea para o conteúdo da postagem, ela segue basicamente o mesmo princípio que o inputText.

O método $Form->submit() irá gerar um input do tipo submit para enviarmos nosso formulário. Ele aceita um parâmetro para personalizarmos o submit, seja para alterar o css ou para alterar de input para button.

A última linha é o método $Form->close(), que irá gerar uma tag de fechamento do formulário.

Agora vamos voltar para nosso index e atualiza-lo com um link para adicionarmos as postagens. Após o fechamento da tabela adicione:

```
{$Html->actionLink('Adicionar Postagem', 'add', 'posts')}
```

Você deve estar se perguntando como eu falo para o EasyFw sobre minhas regras de validações. Elas são informadas no nosso modelo. Vamos ver nosso modelo Post com algumas alterações

```
< ?php
namespace App\Model;

class Post extends AppModel {
    public $id;
    /**
   * @NotEmpty
   */
    public $titulo;

    /**
   * @NotEmpty
   */
    public $conteudo;
}
```

A validação é chamada quando chamamos o método save(). Aqui nós especificamos que as propriedades $titulo e $conteudo não podem ser vazias (@NotEmpty). A API de validação do EasyFw já vem com diversas validações pré-definidas (cartão de crédito, email, telephone, etc.) e flexíveis para você criar suas próprias validações. Para mais detalhes confira a sessão Validação de Dados.

Agora que temos nossa validação pronta, tente incluir uma postagem sem o conteúdo e veja o que acontece. Desde que nós utilizamos FormHelper::inputText() as mensagens de validação são mostradas automaticamente.

Editando Postagens

A partir desse momento você já é um professional no EasyFw, então você deve escolher um padrão. Crie a action e depois a view. Aqui nós temos a action edit() do nosso PostsController:

```
< ?php
public function edit($id = null) {
$this->Post->id =  $id;
if ($this->request->is('get')) {
                $postagem = $this->Post->read() ;
                $this->postagem = $postagem;
        } else {
                if ($this->Post->save($this->data)) {
$this->Session->setFlash('Sua postagem foi atualizada.') ;
                        $this->redirectToAction('index') ;
                } else {
$this->Session->setFlash('Não pudemos atualizar sua postagem') ;
                }
        }
}
```

A ação verifica se o request veio via GET. Se veio, então nós procuramos a postagem no banco de dados e passamos para a view. Se a requisição vier via POST então nós vamos salvar o registro, ou retornar os erros de validação.

A view da action edit() será assim.

```
{block name="content"}

<div class="page-header">
   <h1>{__("Editar Postagem")}</h1>
</div>
{$Form->create('edit, 'posts', $postagem->id)}
   {$Form->inputTextLabelFor($postagem->titulo, 'titulo')}
   {$Form->textAreaLabelFor($postagem->conteudo, 'conteudo', ['rows'=>3])}
   {$Form->submit('Salvar', ['class' => 'btn'])}
{$Form->close()}

{/block}
```

A view irá renderizar o formulário já populado com a postagem.

Uma coisa importante é saber que o EasyFw irá assumir que você quer atualizar um registro graças ao parâmetro id, que nós estamos informando no formulário.

Excluindo Postagens

A lógica de exclusão das postagens é bem simples. O fluxo segue a mesma lógica da edição: o usuário irá acessar a action delete passando um id como parâmetro, se a requisição for via GET iremos mostrar uma tela de confirmação da exclusão do registro, nessa tela enviaremos uma requisição post para a action delete, então poderemos deletar o registro e redireciona-lo para a index novamente.

Vamos criar a action para excluir a postagem no nosso PostsController.

```
< ?php
public function delete($id = null) {

if ($this->request->is('post')) {
   $this->Post->delete($id);
   $this->redirectToAction('index');
} else {
   $this->postagem = $this->Post->getEntityManager()->read($id);
}
```

Agora precisamos criar a view de confirmação da exclusão da postagem

```
{block name="content"}

<div class="page-header">
   <h1>{__("Exclusão da postagem %s", $postagem->id)}</h1>
</div>
{$Form->create('delete, 'posts', $postagem->id)}
   {$Form->label('Título da Postagem')}
   {$Form->label($postagem->titulo)}
   {$Form->submit('Excluir', ['class' => 'btn'])}
{$Form->close()}

{/block}
```

Com isso nós já temos o fluxo da exclusão das postagens. Muito simples não?

Rotas

Para alguns, as rotas padrão do EasyFw são o suficiente para trabalhar. Alguns desenvolvedores são mais sensíveis quanto as URL amigáveis que ajudam na indexação das Engines de busca. Então para esses desenvolvedores desenvolvemos uma forma de mapear rotas na aplicação. Então para deixar tudo mais interessante vamos fazer algumas mudanças nesse tutorial.

For more information on advanced routing techniques, see *Routes Configuration*.

Por padrão, quando o usuário realiza uma requisição para a raiz da aplicação (ex.: http://www.example.com) o EasyFw irá direciona-lo para o controller padrão HomeController e irá chamar uma action chamada index(). Vamos alterar esse comportamento e pedir para o EasyFw setar como controller padrão nosso PostsController.

O arquivo de rotas do EasyFw está em *App/Config/route.yml* E ficara mais ou menos assim:

```
Routing:

 connect:

  # Here, we are connecting '/' (base path) to controller called 'Home', its action called 'index'

  '/': {controller: Posts, action: index}
```

Essa linha conecta a URL “/” que é a raiz da aplicação com o PostsController chamando a action index(), isso significa que podemos criar diversas rotas diferentes e direcioná-los para qualquer controller em qualquer action().

Uma observação, o EasyFw usa as rotas reversas, ou seja, você pode passar o nome do controller o nome da action como um array indexado, igual no exemplo acima, mas também poderia passar apenas como uma string, dessa forma “posts/index”.

Conclusão

Criar aplicações dessa forma irá poupar tempo e dinheiro, além de ficar muito mais divertido codificar sua aplicação. Simples né? Tenha em mente que esse foi apenas um simples tutorial, o EasyFw tem muito mais ferramentas interessantes que merecem ser estudas, elas deixaram sua aplicação mais flexível e poderosa.

Agora que você criou uma aplicação básica como o EasyFw você está pronto para um projeto de verdade. Comece o seu próprio projeto e leia nossa API (http://easyframework.net/2.x/api).

Se você precisar de ajuda, participe do nosso fórum. Bem-vindo ao EasyFw!

Sugestões de leitura

Essas são tarefas comuns quando utilizamos o EasyFw:

1. *Layouts*: Customizando o layout de sua aplicação
2. *Uma pequena aplicação utilizando Autenticação e Autorização*: Autenticação e Autorização de usuários

Leitura Adicional

Uma requisição típica no EasyFw

Nós cobrimos os ingredientes básicos do EasyFw, então vamos dar uma olhada em uma requisição básica. Continuando com o exemplo principal, imagine que nosso amigo Ricardo clicou em “comprar um livro”.

1. Ricardo clica em um link que aponta para http://www.example.com/livros/comprar e o browser dele fez uma requisição para o servidor.

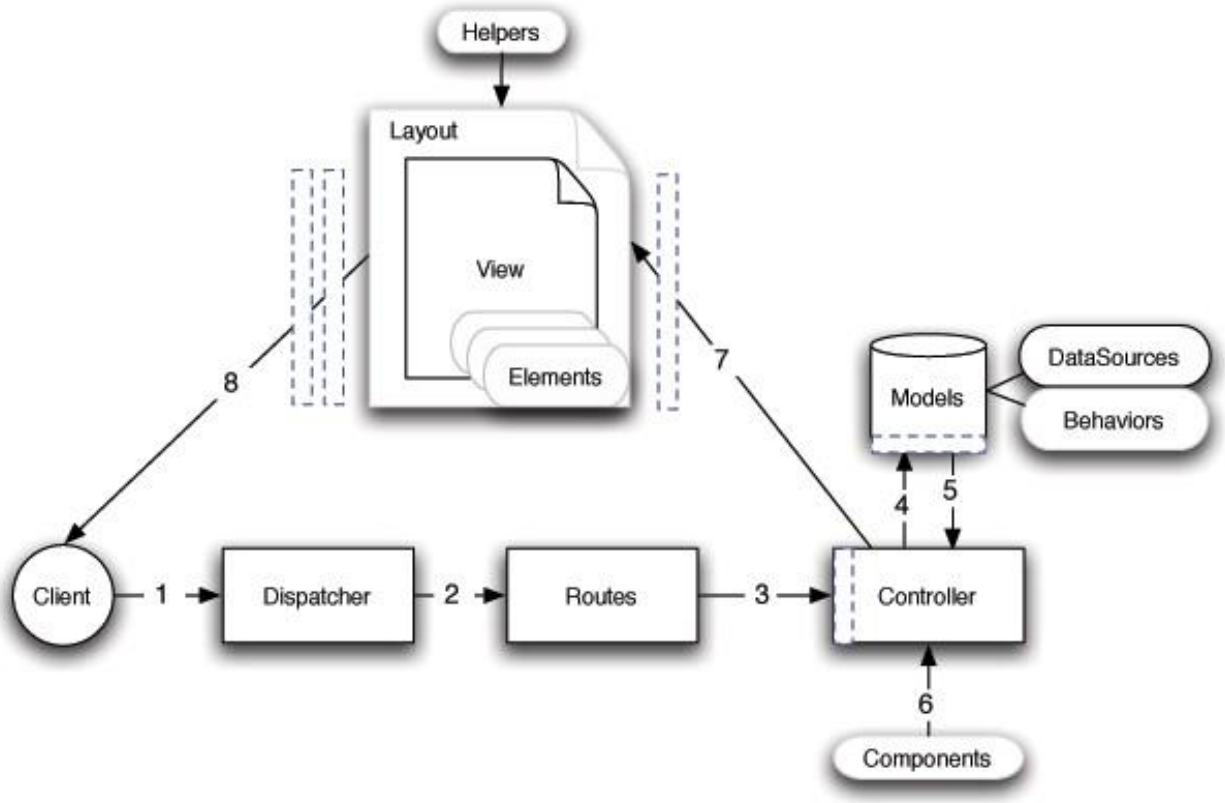

Figura 1Diagrama de Fluxo do EasyFw

2. O Router irá decodificar a URL para extrair parâmetros para a requisição, nessa ordem: O controller, action, e qualquer argumento que afete a lógica de negócio da aplicação.

3. Usando rotas, a URL requisitada é mapeada para uma action de um controller. Nesse caso, é o método comprar() do controller LivrosController. O callback beforeFilter() é chamado antes que qualquer action seja executada.

4. O controller poderá estar utilizando Modelos para acessar dados da aplicação. Nesse exemplo o controller usa um modelo para acessar as últimas compras do Ricardo. Qualquer call-back do modelo, datasources e behaviors serão utilizados nesse instante.

5. Depois que o modelo recuperou os dados e retornou para o controller, outros call-backs do modelo serão disparados.

6. O controller poderá utilizar componentes para realizar operações especificas (manipulação de sessão, autenticação ou enviar e-mails por exemplo).

7. Uma vez que o controller utilizou os modelos e componentes para ter dados suficientes para realizar a operação, esses dados são passados para a view caso necessário. Os call-backs são disparados antes de renderizar a view. A lógica da view é realizada, podendo incluir elementos e helpers. Por padrão uma view e renderizada dentro de um layout.

8. Callbacks adicionais são disparados (como o afterFilter). A renderização é completada e enviada para o browser do Ricardo.

Convenções do EasyFw

Nós gostamos de utilizar configurações por convenções, isso irá ajudá-lo a poupar tempo: seguindo as convenções, você tem funcionalidades prontas e evita estar sempre reconfigurando os arquivos de configurações. As convenções são bem flexíveis, e permitem que você tenha suas próprias convenções, evitando estar “engessado” ao framework. Programe da maneira que você gosta, mas siga um padrão.

Convenções dos Controllers

Os nomes dos controllers devem ser no plural e no formato CamelCase terminado com Controller. PostsController e LivrosController são exemplos dessa convenção.

O primeiro método que você deve ter em seu controller é o index(). Quando uma requisição especificar apenas o controller e não a action, automaticamente esse método é chamado no controller. Por exemplo, uma requisição para http://www.example.com/produtos/ é mapeado para chamado o método index() do controller ProdutosController.

Atenção: Métodos privados não podem ser chamados por uma URL.

Convenções para Arquivos e nomes de Classes

No geral, o nome dos arquivos seguem o mesmo nome das classes, que são em CamelCase. Então se você tiver uma classe MinhaClasse, o nome do arquivo deverá ser MinhaClasse.php. Abaixo seguem alguns exemplos:

- O controller BeijosEAbracosController terá o arquivo com o nome BeijosEAbracosController.php
- O componente MeuComponenteComponent terá o arquivo MeuComponenteComponent.php
- O modelo Opcoes será encontrado no arquivo Opcoes.php

Convenções de Modelos e Banco de dados

Os modelos normalmente são classes no singular e no formato CamelCase. Pessoa e AmigosFavoritos são exemplos de modelos.

As tabelas do banco de dados são no plural e com underscore. As tabelas para os modelos acima seriam: pessoas e amigos_favoritos.

Chaves estrangeiras com relacionamentos hasMany, belongsTo ou hasOne são reconhecidos por padrão pelo tabela que está sendo relacionada seguido da chave primária. Por exemplo uma relação entre uma tabela de Vendas que tem uma pessoa relacionada, a chave estrangeira de pessoa na tabela vendas ficaria assim – pessoa_id.

Convenções das Views

As views seguem o mesmo padrão das actions, ou seja, caso você tinha uma action comprarLivros() a view deverá ser nomeada como comprarLivros.tpl.

Seguindo essas convenções ficará bem mais simples trabalhar, já que tudo será realizado automaticamente pelo EasyFw.

Aqui está o exemplo final das convenções:

- Tabela do Banco: “pessoas”
- Classe Modelo: “Pessoa”, encontrada em /App/Model/Pessoa.php
- Classe Controller: “PessoasController”, encontrada em /App/Controller/PessoasController.php
- Template da View, encontrada em /App/View/Pages/Pessoas/index.tpl

Estrutura de pastas do EasyFw

Depois de baixar o Easy Skeleton Application você verá as seguintes pastas:

- App
- vendors
- .htaccess
- README

Você notará duas pastas principais:

- A pasta App onde você irá trabalhar e criar sua aplicação.
- A pasta vendors, onde colocaremos todas as bibliotecas PHP de terceiros. Nessa pasta também está a pasta do EasyFw, seguindo o padrão do *Composer*.

A pasta App

Vamos analisar a principal pasta do EasyFw onde, você irá trabalhar.

- Config – Estão todas as configurações da aplicação.
- Controller – Contém os controllers e components da aplicação.
- Locale – Armazena às strings de internacionalização.
- Model – Contém os models, behaviors e datasources da aplicação.
- tmp – Aqui o EasyFw armazena dados temporários como cache, logs e descrições dos modelos.
- View - Arquivos de apresentação são armazenados aqui. Elementos, Pages, Layouts.
- webroot – São seus arquivos públicos, ou seja, os CSS, JS e imagens de sua aplicação.

Estendendo o EasyFw

Controllers e models cada um deles tem sua classe mãe para definir comportamentos para todos eles. AppController (localizado em /App/Controller/AppController.php) e AppModel (localizado em /App/Model/AppModel.php)

Estendendo Controllers – Components

Um componente é uma classe que adiciona lógica ao controller. Se você tem alguma lógica que pode ser compartilhada entre os controllers (ou aplicações) um componente pode ser uma ótima jogada. Um exemplo, o EmailComponent do EasyFw facilita o envio e recebimento de e-mail. Melhor do que escrever um controller que irá conter uma lógica engessada, você pode empacota-la para que ela seja compartilhada.

Controllers também possuem callbacks. Esses callbacks estão disponíveis para uso, apenas no caso de você precisar de alguma lógica entre as operações do EasyFw. Os call-back disponíveis são:

- beforeFilter(), executado antes que qualquer action seja executada
- beforeRender(), executado antes que uma view seja renderizada
- afterFilter(), executado após que uma action seja chamada incluindo a renderização da view.

Estendendo às Views - Helpers

Um Helper é uma classe que adiciona lógica nas view. Bem parecido com um componente, helpers poderão ser compartilhados entre as views.