Simplest MVC

PHP

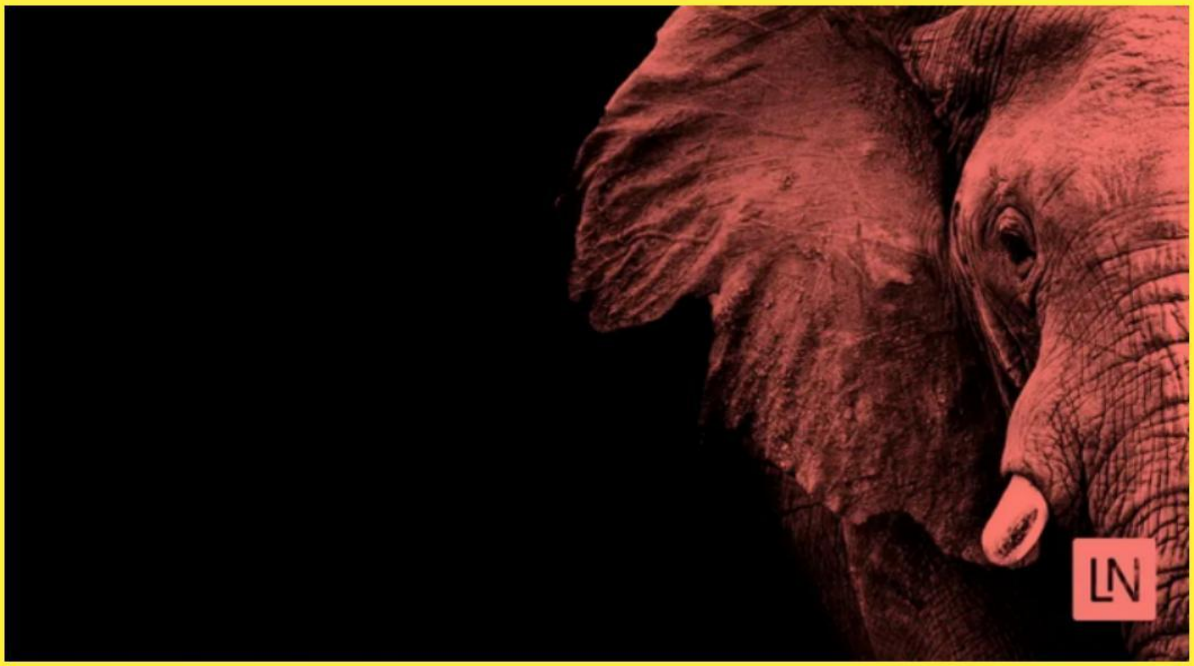

Imagem do https://laravel-news.com

Dezembro de 2020

por Ribamar FS

Acesse em https://github.com/ribafs/simplest-mvc

Table of Contents

1.0 - MVC Simples em PHP sem banco de dados

Arquivos

index.php Controller.php Model.php views/index.php

Fluxo das informações

Quando o index.php é chamado:

- inclui o Controller.php
- Instancia o Controller
- Chama o método index()

O Controller é chamado:

- Ele inclui o Model
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui o index.php da view.

Quando o Model é chamado

- Ele define um array multi dimensional na variável $clients e insere 4 clientes, com código, nome e idade
- Retorna em seu método index() este array
- Este array é recebido pelo Controller

Quando o index.php é chamado

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list, vindo do Controller, que recebeu do Model

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.

2.0 - MVC Simples em PHP com banco de dados

O mais simples possível, com apenas um controller e um model.

Arquivos

index.php Controller.php Model.php views/index.php db.sql

Fluxo das informações

Quando o index.php é chamado:

- inclui o Controller.php
- Instancia o Controller
- Chama seu método index()

Então o Controller é chamado:

- Ele inclui o Model
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui o index.php da view.

Quando o Model é chamado

- Ele define um array multi dimensional na variável $clients e insere 4 clientes, com código, nome e idade
- Retorna em seu método index() este array
- Este array é recebido pelo Controller

Quando o index.php é chamado

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list, vindo do Controller, que recebeu do Model

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.

3.0 - MVC Simples em PHP cem banco de dados

Esta versão adiciona o suporte a vários controllers, models e views

Estrutura de arquivos

- clients.php
- products.php
- index.php
- db.sql
- App
    - Controllers
        - ClientController
        - ProductController
    - Models
        - ClientModel
    - views
        - clients/index.php
        - products/index.php
- Core
    - config.php
    - Model

Fluxo das informações

Quando o index.php é chamado:

- Mostra dois links para "clients" e para "products"

Caso clique em clients

- Incluirá o ClientController
- Cria uma instância do mesmo
- E chama seu método index()

De forma semelhante ao clicar em products

Quando o ClientController é chamado:

- Ele inclui o ClientModel
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui a view App/views/clients/index.php, que mostrará o resultado recebido do model

Quando o ClientModel é chamado

- Ele include o Model pai em Core e extende Model
- Em seu método index() efetua uma consulta para trazer todos os registros da tabela clients
- E retorna a consulta na variável $clients, que será recebida pelo controller

Quando a view views/clients/index.php é chamada

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list do Controller e mostra na tela os registros da tabela clients

De forma semelhante procedi com o ProductController e o ProductModel

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.

4.0 - MVC Simples em PHP cem banco de dados

Este versão implementa o autoload com composer e PSR-4.

Assim não precisaremos estar usando requires, mas em seu lugar usaremos namespace, que é mais tranquilo e profissional.

Foram criados dois namespaces: App e Core, cada um aponta para a pasta com o mesmo nome. Assim fica mais fácil de lembrar qual a pasta de certo namespace.

Após a criação do composer e conclusão do aplicativo, inclusive adicionando os namespaces, então precisamos executar no raiz do aplicativo:

composer du

E executar novamente sempre que alterar o composer.json

Estrutura de arquivos

- composer.json
- index.php
- db.sql
- App
    - Controllers
        - ClientController
        - ProductController
    - Models
        - ClientModel
    - views
        - clients/index.php
        - products/index.php
- Core
    - Model

Fluxo das informações

Quando o index.php é chamado:

- Mostra dois links para "clients" e para "products"
- Inclui o autoload

Se o link clients tiver sido chamado

- Instanciará o ClientController
- E chamará seu método index()

De forma semelhante se products for clicado

Quando o ClientController é chamado:

- Na versão anterior ele incluiu o ClientModel.
    - Agora adicionamos o namespace na primeira linha
    - E cada include é substituido por um "use". Veja como ficou agora no ClientController namespace App\Controllers; use App\Models\ClientModel;
- O restando do código permanece
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui a view App/views/clients/index.php, que mostrará o resultado recebido do model

Quando o ClientModel é chamado

- Adiciona namespace e troca include por use
- Em seu método index() efetua uma consulta para trazer todos os registros da tabela clients
- E retorna a consulta na variável $clients, que será recebida pelo controller

Quando a view views/clients/index.php é chamada

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list do Controller e mostra na tela os registros da tabela clients

De forma semelhante procedi com o ProductController e o ProductModel

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.

5.0 - MVC Simples em PHP cem banco de dados

Este versão implementa um sistema de rotas simples, o front controller e mais alguns bosn recursos.

Agora temos nosso MVC em PHP completo. Isso do meu ponto de vista, pois ao meu ver a sua criação teve a finalidade de nos ensinar a usar alguns dos bons e importantes recursos encontrados nos grandes frameworks: PHPOO, MVC, Rotas, Front Controller, autoload com composer e PSR-4, entre outros. Me parece que um grande framework como o Laravel já nos entre o que existe de melhor pronto, babsta usar. Claro que você pode aproveitar o que fizemos e ir em frente, escolha sua.

Ops, ainda teremos a versão 6, que implementará um recurso desejável, que é o Bootstrap, basicamente seu CSS.

Estrutura de arquivos

- composer.json
- db.sql
- .htaccess
- public
    - .htaccess
    - css
    - img
    - js
    - index.php
- App
    - Controllers
        - ClientController
        - ProductController
    - Models
        - ClientModel
    - views
        - clients
            - index.php
        - error
            - index.php
        - products
            - index.php
        - templates
            - footer.php
            - header.php
- Core
    - config.php
    - ErrorController
    - Model
    - Router

Fluxo das informações

- Veja que não mais existe um index.php. Quando alguém chega no raiz de mvc5 é redirecionado para a apsta public pelo .htaccess

- Quando chega na pasta public será redirecionado para o public/index.php pelo public/.htaccess

Quando o public/index.php é chamado

- Ele define algumas constantes para o aplicativo
- Inclui o autoload
- Inclui o Core/config.php, que também define outras constanes importantes
- Então inclui a classe Core/Router através do use
- Instancia a classe Router. Veja que esta classe tem dois métodos, o construct e o Url(), mas ela chama o Url() dentro do __construct(), portanto logo que a classe Router é instanciada todo o seu código é executado e está disponível, no caso o tratamento das rotas acessadas pelo usuário.
- A captura da URL digitada é feita pelo método Url(), que é executado inicialmente pelo __construct() e então testado, para que de acordo com a URL acessar o respectivo controller.
- Veja que se alguém chama o "clients" pela URL o Router dispara o ClientController, de forma semelhante, se for chamada "products" o ProductController será disparado. Caso a URL digitada ou chamada via link não seja para clients nem para products será disparado o ErrorController.

Quando o ClientController é chamado:

- Na versão anterior ele incluiu o ClientModel.
    - Agora adicionamos o namespace na primeira linha
    - E cada include é substituido por um "use". Veja como ficou agora no ClientController namespace App\Controllers; use App\Models\ClientModel;
- O restando do código permanece
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui a view App/views/clients/index.php com o respectivo template, que mostrará o resultado recebido do model

Quando o ClientModel é chamado

- Adiciona namespace e troca include por use
- Em seu método index() efetua uma consulta para trazer todos os registros da tabela

clients
- E retorna a consulta na variável $clients, que será recebida pelo controller

Quando a view views/clients/index.php é chamada

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list do Controller e mostra na tela os registros da tabela clients

De forma semelhante procedi com o ProductController e o ProductModel

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.

6.0 - MVC Simples em PHP cem banco de dados

Nesta versão apenas troquei o CSS original pelo Bootstrap 4.

Um sistema de rotas simples foi implementado, o front controller e mais alguns bosn recursos.

Agora temos nosso MVC em PHP completo. Isso do meu ponto de vista, pois ao meu ver a sua criação teve a finalidade de nos ensinar a usar alguns dos bons e importantes recursos encontrados nos grandes frameworks: PHPOO, MVC, Rotas, Front Controller, autoload com composer e PSR-4, entre outros. Me parece que um grande framework como o Laravel já nos entre o que existe de melhor pronto, babsta usar. Claro que você pode aproveitar o que fizemos e ir em frente, escolha sua.

Estrutura de arquivos

- composer.json
- db.sql
- .htaccess
- public
    - .htaccess
    - css
    - img
    - js
    - index.php
- App
    - Controllers
        - ClientController
        - ProductController
    - Models
        - ClientModel
    - views
        - clients
            - index.php
        - error
            - index.php
        - products
            - index.php
        - templates
            - footer.php
            - header.php
- Core
    - config.php
    - ErrorController
    - Model
    - Router

Fluxo das informações

- Veja que não mais existe um index.php. Quando alguém chega no raiz de mvc5 é redirecionado para a apsta public pelo .htaccess

- Quando chega na pasta public será redirecionado para o public/index.php pelo public/.htaccess

Quando o public/index.php é chamado

- Ele define algumas constantes para o aplicativo
- Inclui o autoload
- Inclui o Core/config.php, que também define outras constanes importantes
- Então inclui a classe Core/Router através do use
- Instancia a classe Router. Veja que esta classe tem dois métodos, o construct e o Url(), mas ela chama o Url() dentro do __construct(), portanto logo que a classe Router é instanciada todo o seu código é executado e está disponível, no caso o tratamento das rotas acessadas pelo usuário.
- A captura da URL digitada é feita pelo método Url(), que é executado inicialmente pelo __construct() e então testado, para que de acordo com a URL acessar o respectivo controller.
- Veja que se alguém chama o "clients" pela URL o Router dispara o ClientController, de forma semelhante, se for chamada "products" o ProductController será disparado. Caso a URL digitada ou chamada via link não seja para clients nem para products será disparado o ErrorController.

Quando o ClientController é chamado:

- Na versão anterior ele incluiu o ClientModel.
    - Agora adicionamos o namespace na primeira linha
    - E cada include é substituido por um "use". Veja como ficou agora no ClientController namespace App\Controllers; use App\Models\ClientModel;
- O restando do código permanece
- Instancia o model
- Chama seu método index(), que retorna o array $clients e armazena na variável $list;
- Então o Controller inclui a view App/views/clients/index.php com o respectivo template, que mostrará o resultado recebido do model

Quando o ClientModel é chamado

- Adiciona namespace e troca include por use
- Em seu método index() efetua uma consulta para trazer todos os registros da tabela

clients
- E retorna a consulta na variável $clients, que será recebida pelo controller

Quando a view views/clients/index.php é chamada

- Mostra algum texto explicativo na tela
- Cria uma tabela HTML
- Joga na tabela o resultado de um for trazendo o array $list do Controller e mostra na tela os registros da tabela clients

De forma semelhante procedi com o ProductController e o ProductModel

No caso a View recebe do Controller e não diretamente do Model, o que é uma boa prática.