# Publishing Process

* 1. Ativar código que sobrescreva o window.console (não tenho certeza sobre isso)

// Disable console

window.console = {

log : function(){},

info : function(){},

error : function(){},

warn : function(){}

};

* 1. Modificar arquivo .env
  2. No layout máster, trocar a fonte do javascript de “main.js” para “main.min.js”
  3. Usar a task “gulp build” para preparar os recursos

Comecei tudo de novo. Dessa vez, farei o site através do codeanywhere. Iniciei um container do Wordpress lá.

# Enviando repositório do computador para o Bitbucket

Para iniciar um repositório novo no Bitbucket, a forma mais fácil parece ser criar o repositório no computador e depois enviar o repositório para o bitbucket. Para fazer isso, seguir os seguintes passos:

* 1. Criar pasta do projeto
  2. Dar um gitbash na pasta
  3. Dar um “git init”
  4. Entrar no SourceTree e dar o primeiro commit no projeto (ainda local).
  5. Voltar a pasta e dar os seguintes comandos para enviar para o Bitbucket:
     1. git remote add origin https://fabiomolinar@bitbucket.org/fabiomolinar/photo2me-android.git
     2. git push -u origin –all
     3. git push origin --tags # pushes up any tags

# Criando password protection com htaccess.

Dois arquivos são necessários. Um chamado “.htaccess” e outro “.htpasswd”. Como o windows não deixa criar esses arquivos sem um nome manualmente, precisamos abrir o notepad, clicar em “Salvar como” e no nome do arquivo escrever “.htaccess” e no tipo de arquivo selecionar “All types”.

The .htaccess tem as seguintes linhas de código:

AuthType Basic

AuthName "Password Protected Area"

AuthUserFile /path/to/.htpasswd

Require valid-user

E o .htpasswd tem que ter a seguinte forma:

Login:password

Por exemplo, “admin:senhaespecial”.

Se não estiver funcionando, é provável que na parte “AuthUserFile” eu não esteja colocando o caminho correto para o arquivo com senhas. O caminho complete tem que ser especificado. O caminho completo pode ser descoberto adicionando a linha de código abaixo em um arquivo html que esteja no caminho que queremos descobrir:

<?php echo $\_SERVER['DOCUMENT\_ROOT'];?>

O código acima irá retornar o caminho correto para se chegar na pasta onde está o arquivo html.

Outro detalhe, a senha no arquivo htpasswd não é a senha em si, mas a versão codificada da senha. Para fazer a versão decodificada da senha, posso usar esse site: <http://www.htaccesstools.com/htpasswd-generator/>

# Modificando pasta onde ficará o index.php do domínio

[AINDA NÃO TESTADO] In deciding what file to serve for a given request, httpd's default behavior is to take the URL-Path for the request (the part of the URL following the hostname and port) and add it to the end of the [DocumentRoot](https://httpd.apache.org/docs/trunk/mod/core.html" \l "documentroot) specified in your configuration files. **Therefore, the files and directories underneath the [DocumentRoot](https://httpd.apache.org/docs/trunk/mod/core.html" \l "documentroot) make up the basic document tree which will be visible from the web**.

For example, if [DocumentRoot](https://httpd.apache.org/docs/trunk/mod/core.html#documentroot) were set to /var/www/html then a request for http://www.example.com/fish/guppies.html would result in the file/var/www/html/fish/guppies.html being served to the requesting client.

Ou seja, cada vez que uma pessoa entra no site, se eu quiser que outro index.php seja servido, tenho que modificar o DocumentRoot que é configurado no arquivo “.htaccess”.

# Evitando directory browsing/listing

Apenas adicionar a seguinte linha de código ao .htaccess:

Options -Indexes

# Estratégia de configuração e .htaccess

Dentro do subdomain, criei um .htaccess e habilitei a proteção por senha (login: admin, senha: Brasil2016). Em seguida, na pasta públic no laravel, editei o .htaccess dessa pasta para que o acesso seja público com o seguinte código:

Require all granted

Satisfy Any

Além disso, adicionei a opção que evita listagem do diretório como descrito no ponto 4 deste documento.

* Redirecionando endereço do subdomain para a pasta public do laravel

Basta adicionar o código abaixo:

*RewriteEngine on*  
*RewriteCond %{HTTP\_HOST} ^domain-name.com$ [NC,OR]*  
*RewriteCond %{HTTP\_HOST} ^www.domain-name.com$*  
*RewriteCond %{REQUEST\_URI} !folder/*  
*RewriteRule (.\*) /folder/$1 [L]*

Lembrar que “folder” é o caminho completo. Por exemplo:

RewriteEngine on

RewriteCond %{HTTP\_HOST} ^photo.optykopoczno.pl$ [NC,OR]

RewriteCond %{HTTP\_HOST} ^www.photo.optykopoczno.pl$

RewriteCond %{REQUEST\_URI} !home/optykopo/photo/photo2me/public/

RewriteRule (.\*) /home/optykopo/photo/photo2me/public/$1 [L]

# Criando a API

* 1. Desabilitar CSRF

Primeira coisa a fazer é remover a proteção CSRF que vem automaticamente no laravel para que não seja necessário a presença do token nos post requests. É possível fazer isso ao adicionar as uri que não precisam de proteção CSRF ao array $except dentro do middleware “VerifyCsrfToken”.

* 1. JSON response

Para retornar JSONs, basta usar o código abaixo:

return response()->json([arrayDeDados],httpResponseCode);

Por exemplo:

return response()->json([

'status' => '400',

'evento-nao-encontrado' => Lang::get('messages.evento-nao-encontrado')

], 400);

* 1. Configurando php.ini para aceitar arquivos maiores que 2MB

O PHP parece que tem alguns limites com relação ao tamanho dos arquivos que podem ser enviados ao servidor. Para alterar essa configuração, ir ao php.ini e mudar as seguintes variáveis: memory\_limit, upload\_max\_filesize, file\_uploads

# Database

NÃO ESQUECER DE CONFIGURAR AS VARIÁVEIS NO ARQUIVO .env.

Para trabalhar com fusos horários diferentes, melhor que o banco de dados seja do tipo DATETIME do que TIMESTAMP. Com o tipo TIMESTAMP, o banco de dados pega a data no fuso do servidor, converte para UTC e salva no banco em UTC. Quando retiramos o dado do banco, o banco pega o dado em UTC e converte para o fuso do servidor. Ou seja, se eu quiser trabalhar com fusos, é melhor usar o tipo DATETIME e salvar o fuso em questão de forma separada.

# NPM

Usar NPM como package manager para bibliotecas como jquery, semantic-ui, bootstrap e etc. Para adicionar uma biblioteca que seja necessária para rodar o código, usar o seguinte comando:

npm install semantic-ui –save

O comando acima irá adicionar a dependência ao nosso package.json e criara uma pasta chamada “node\_modules” onde irá se encontrar o código para essa biblioteca.

Para bibliotecas que são usadas apenas como ferramentas (como o Gulp), usar o seguinte comando:

npm install gulp –save-dev

# GULP

O Gulp é foda. Será muito bom para desenvolvimento do semantic-ui e para automatização de algumas tarefas (como uglifying, minifying e outras). As configurações do Gulp são feitas dentro de um arquivo chamado gulpfile.js.

O Gulp é foda, mas nada fácil ou super intuitivo de ser usado. Uma coisa que é necessária é rodar vários gulpfiles.js de um único gulpfile.js, em vez de ficar indo de pasta em pasta executando as tarefas. Para colocar as tarefas que estão em uma determinada pasta em um outro gulpfile.js, temos que usar o truque do “require-dir”. Primeiro, fazer o requerimento:

requireDir = require('require-dir');

Depois, podemos criar um objeto que vai guardar todos as tarefas que estão salvas em uma certa pasta.

var objetoTarefas = requireDir(‘caminho/para/a/pasta/com/tarefas’);

Esse objeto será composto de todas as tarefas que estão dentro do caminho especificado. Então, por exemplo, se tiver uma tarefa nessa pasta chamada “build.js”, podemos ativar essa task no nosso gulpfile.js principal chamando objetoTarefas.build, assim:

gulp.task('nossoNomeParaEssaTarefa', objetoTarefas.build);

# Semantic

Super importante: configurar corretamento o arquivo semantic.json! E não só ele. Também é importante as configurações contidas dentro da pasta “resources/assets/semantic/src/site”. Aqui estão as definições das variáveis que irão ser usadas para construir o .css e .js do semantic. Ou seja, se eu quiser mudar a cor principal do site ou design dos itens, é nessa pasta que eu tenho que fazer a configuração das variáveis. As variáveis que são definidas aqui serão usadas para sobrescrever os padrões do semantic. Para descobrir o nome das variáveis, basta “caçá-las” dentro dos arquivos originais que estão em “resources/assets/semantic/src/themes”. Dentro de cada um dos temas eu posso encontrar suas devidas configurações. Por exemplo, para setar qual a cor primaria a ser usada, definir a variável no arquivo “semantic/src/site/globals/site.variables”.

Depois de mudar os padrões, basta abrir a linha de comando e digitar o comando “gulp”. Esse comando irá buscar o arquivo gulpfile.js e executar as ações definidas nele.

# Validação de formulários

Para fazer a validação dos formulários, farei a validação tanto no front quanto no back end. No front-end usarei o semantic. No back-end irei criar uma classe para cada um dos requests dentro da pasta “App/Http/Requests”. Essa classe estende a classe request. Dessa forma, podemos sobrescrever os métodos que fazer a validação automaticamente. Se a requisição não passar a validação, o usuário é automaticamente retornado para a página onde ele estava e as mensagens de erro são enviadas junto.

# Abreviações

* -BD: botão direito

# Criando o git

Esse foi fácil. Apenas -BD na pasta onde ficará o projeto a abri o Git Bash.

Inserir o comando “git init”.

Depois, só adicionei a pasta ao Source Tree.

# Criando o laravel

Na pasta do projeto, através do composer, usar o comando “composer create-project --prefer-dist laravel/laravel photo2me”.

# Configurando o servidor

Como eu já tinha o projeto da Evolme rodando nessa máquina e não queria ficar modificando o arquivo httpd.conf toda hora que tivesse que mudar a pasta do projeto apontada para o Apache, modifiquei o arquivo de virtual hosts do Apache para que um url pré-definido me levasse apontasse ao apache uma pasta pré-definida.

No arquivo de hosts do Windows (“c:\windows\system32\drivers\etc\hosts”), adicionei “127.0.0.1 photo2me.com”, para que esse endereço aponte para o loopback.

No arquivo “httpd-vhosts.conf” que se encontra em “C:\xampp\apache\conf\extra”, adicionei as seguintes linhas:

|  |
| --- |
| <VirtualHost \*:80>  ServerName localhost  DocumentRoot "E:\Profissional\EvolMe\Repos v2\public"  </VirtualHost>  <VirtualHost \*:80>  ServerName photo2me.com  DocumentRoot "E:\Profissional\photo2me\Repos\photo2me\public"  <Directory "E:\Profissional\photo2me\Repos\photo2me\public">  Options Indexes FollowSymLinks Includes ExecCGI  AllowOverride All  Require all granted  </Directory>  </VirtualHost> |

Dessa forma, o localhost continua apontando para a pasto do projeto Evolme enquanto o novo url “photo2me.com” aponta para a pasta desse projeto.

# Rotas

Irei criar a seguinte estrutura para as rotas:

* Todas
  + Web
    - Publicas
      * “/”
      * “Cadastro”
      * “Login”
      * “Contato” (não sei se será usado)
    - Privadas
  + App
    - Publicas
    - Privadas

# Layouts padrões

* 1. Publico

O layout foi feito de tal forma que cada página que usa esse layout terá adicionado à ela um conteúdo padrão, estabelecido pelos “@include” e outro conteúdo específico de cada página que pode ser definido, no código da página, através de “@section”.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>photo2me</title>  {{-- Incluir os itens que estarão presentes em todos os heads --}}  @include('includes.head')  {{-- Espaço para heads específicos de cada página --}}  @yield('head')  </head>  <body>  <div class='master' id='header'>  {{-- Incluir os itens que estarão presentes em todos os headers --}}  @include('includes.header')  {{-- Espaço para headers específicos de cada página --}}  @yield('header')  </div>  <div class='master' id='conteudo'>  {{-- Espaço para o conteúdo de cada página --}}  @yield('conteudo')  </div>  <div class='master' id='footer'>  {{-- Espaço para o footer que estará presente em todas as páginas --}}  @include('includes.footer')  </div>  </body>  </html> |

* 1. Input design

Para o design dos inputs, irei usar um código em less que cria um css para os inputs bem legais. Para usar o less, basta:

* Abrir a command line na pasta onde está o arquivo .less e digitar “lessc less.less inputs.css”
* Depois, para usar o design nos inputs, basta usar o seguinte padrão para os inputs:

|  |
| --- |
| <div class="form-group">  <input type="text" name="email" required=''>  <label alt="{{ trans('mensagens.qual-seu-email') }}" placeholder="{{ trans('mensagens.email') }}"></label>  </div> |

* 1. Social buttons design

Para os botões para log in e sign up por redes sociais estou usando uma biblioteca (.css) chamada bootstrap-social.css. E um exemplo de botão do facebook se parece assim:

|  |
| --- |
| <a class="btn btn-block btn-social btn-facebook">  <span class="fa fa-facebook"></span> Facebook  </a> |

* 1. É possível, em um layout, herdar de outro layout. É o que estou fazendo com as páginas do dashboard.
  2. Escondendo/mostrando itens de acordo com viewport no bootstrap

Use a single or combination of the available classes for toggling content across viewport breakpoints.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | xs (<768px) | sm (≥768px) | md (≥992px) | lg (≥1200px) |
| .visible-xs-\* | Visible | Hidden | Hidden | Hidden |
| .visible-sm-\* | Hidden | Visible | Hidden | Hidden |
| .visible-md-\* | Hidden | Hidden | Visible | Hidden |
| .visible-lg-\* | Hidden | Hidden | Hidden | Visible |
| .hidden-xs | Hidden | Visible | Visible | Visible |
| .hidden-sm | Visible | Hidden | Visible | Visible |
| .hidden-md | Visible | Visible | Hidden | Visible |
| .hidden-lg | Visible | Visible | Visible | Hidden |

# Conteúdo padrões do head

# Conteúdo padrões do header

# Conteúdo padrões do footer

# Autenticação

Foi complicado configurar os esquemas de registro, login e logout. Importante de lembrar:

* Adicionar os middlewares responsáveis por cookies e seções às rotas com autenticação.

|  |
| --- |
| Route::group(['middleware' => ['web'], 'as' => 'autenticacao'], function(){  Route::get('auth/login','Auth\AuthController@getLogin');  Route::post('auth/login','Auth\AuthController@postLogin');  Route::get('auth/logout','Auth\AuthController@getLogout');  Route::get('auth/register','Auth\AuthController@getRegister');  Route::post('auth/register','Auth\AuthController@postRegister');  }); |

* E, dentro do AuthController, como minha função de logout se chama getLogout, tive que adicionar essa função a lista de exceções do middleware que faz redirecionamentos automáticos.

|  |
| --- |
| public function \_\_construct()  {  $this->middleware('guest', ['except' => ['logout','getLogout']]);  } |

* E modificar no AuthController quais são os campos que são necessários. Como eu quero que apenas o email e senha seja necessário, preciso retirar o nome da lista do validator.

|  |
| --- |
| protected function validator(array $data)  {  return Validator::make($data, [  //retirei o ‘name’ dessa linha  'email' => 'required|email|max:255|unique:users',  'password' => 'required|confirmed|min:6',  ]);  } |

* O método de registro padrão do Laravel utiliza pelo menos três campos, email, senha e confirmação de senha. Então, não esquecer de colocar um <input> com o “verificar minha senha”.

# JWT-Auth

**Depois de algumas tentativas, decidi que na parte de web usarei a parte de autenticação que vem do próprio laravel. Depois terei que criar uma interface entre o back-end e o aplicativo.** As instruções abaixo são para adicionar o JWT-Auth ao projeto.

Primeiro, segui as seguintes instruções:

|  |
| --- |
| composer require tymon/jwt-auth  php artisan jwt:generate  php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\JWTAuthServiceProvider" |

Agora tive que partir para a criação do banco de dados.

Para usar o JWT-Auth, TENHO QUE ADICIONÁ-LO AOS CONTROLLERS!

# Banco de dados

* 1. Criação

Primeiro, vamos criar o banco de dados no <http://localhost/phpmyadmin>.

Modificado as seguintes linhas do .env:

|  |
| --- |
| DB\_HOST=localhost  DB\_DATABASE=photo2me  DB\_USERNAME=root  DB\_PASSWORD= |

Para criar novas tabelas, usaremos o commando do artisan “make:migration”.

* 1. Atualização

Para fazer a atualização de alguma tabela do banco, criamos uma nova migration relacionada à tabela em questão. Podemos fazer isso através do comando:

|  |
| --- |
| php artisan make:migration update\_nome\_table --table= nome |

E, no arquivo da migration, adicionar as colunas que queremos. Um exemplo de código abaixo.

|  |
| --- |
| public function up()  {  Schema::table('users', function (Blueprint $table) {  $table->enum('tipo\_cliente',array('unico','rede'));  $table->enum('tipo\_usuario',array('usuario','admin'));  $table->date('data\_nascimento')->nullable();  $table->string('sobrenome')->nullable();  $table->boolean('marketing');  });  } |

# Models

* 1. User
  2. Festa

Para fazer validação de cada tentativa de criar uma festa, irei usar do que o laravel chama de “validation”. Ver capitulo sobre validation.

* 1. Foto
  2. Ordem

# Validation

Para fazer uma validação especial, tive que criar um extensão da classe validator dentro do appserviceprovider->boot().

|  |
| --- |
| //Validador extra para evitar eventos com mais de "x" dia(s).  Validator::extend('mais\_x\_dias', function($attribute,$value,$parameters,$validator){  $inicio = Carbon::createFromTimestamp(strtotime($parameters[0]));  $fim = Carbon::createFromTimestamp(strtotime($value));  $dias = $fim->diffInDays($inicio);  if ((strtotime($value) - strtotime($parameters[0])) < 0){  //Data de fim é menor que a data de início. Outra validação é que tem que ser usada nesse caso.  return true;  } else {  if ($dias < $parameters[1]){  return true;  } else {  return false;  }  }  }); |

Com essa nova função de validação, pude usar o validator::make no controller das festas para fazer a validação do formulário de criação da festa.

|  |
| --- |
| $validator = Validator::make($request->all(),[  'numConvidados' => 'required|numeric|min:1',  'inicioFesta' => 'required|date|before:' . $request->fimFesta,  'fimFesta' => 'required|date|after:' . $request->inicioFesta . '|mais\_x\_dias:' . $request->inicioFesta . ',1'  ]); |