Criando Aplicativos com o CakePHP 3

Sumário

O CakePHP é um framework que conta com uma ferramenta que automatiza a criação de aplicativos, que é o bake. Esta ferramenta simplifica muito a criação de um aplicativo básico, com o CRUD e vários outros bons recursos.

Aqui mostraremos como usar o bake mas também como criar aplicativos sem o uso do mesmo, além de criar código manualmente e que se comunique entre as camadas do MVC.

Iremos criar um aplicativo chamado **crud** em algumas etapas, um aplicativo para finanças pessoais e um aplicativo tipo controle de estoque mas simplificado.

Algumas Dicas

Customizar CSS do bootstrap numa view

```
echo $this->Form->input('grupo',['style'=>'width: 200px']);
```

CSS e JavaScript

No topo da view

```
echo $this->Html->script(array('ajaxupload.3.5.js'));

echo $this->Html>css(array('new_layout.css'));
```

```
<script type="text/javascript">
function filldetails()
{
   document.getElementById('FirstName').value  = "hjshjsh";
}
</script>
echo $this->Form->select('grupos',['onblur' =>grupo());

<style>
.col1{
   width:50px;
}

.col2{
   width:100px;
}

.col4{
   width:200px;
}
</style>


      <?= $this->Form->input('username', ['label'=>'Login', 'class'=>'col4']) ?>
      <?= $this->Form->input('password',['label'=>'Senha', 'class'=>'col4']) ?>
```

Foco em um campo

```
echo $this->Form->input('grupo',['style'=>'width: 100px', 'type'=>'text', 'autofocus']);
```

Relacionamentos

O campo secundário do relacionamento precisa ser NOT NULL na tabela.

Exemplo
users com clientes

Em clientes temos o campo user_id que relaciona as tabelas.
Na tabela clientes o campo user_id precisa ser:

user_id int not null

Caso contrário teremos sérios problemas de cadastramento de todos os registros vazios e fura o relacionamento.

Relacionamentos no Banco e no Cake

O CakePHP tem uma convenção para relacionamento entre tabelas, como visto acima, de forma que nem precisamos implementar o relacionamento pelo SGBD, pois o Cake cuida disso pra nós.

Acontece que no nosso exemplo de script, onde não implementamos o relacionamento pelo banco, temos 100 registros de servidores, todos relacionados com users e ainda não temos nenhum user cadastrado.

Isto não é coerente. Só deveriamos adicionar um servidor, quando antes tivéssemos cadastrado o user respectivo.

Recomendação: sempre implemente os relacionamentos no banco, que fica mais coerente e mais seguro.

Para implementar relacionamento entre servidores e users na tabela servidores faça assim (mas isso no início, antes de criar servidores):

```
CREATE TABLE servidores (
      id SERIAL PRIMARY KEY,
      nome varchar(55) NOT NULL,
      nascimento date default null, -- Requerido default null em campos data peço cake
      cpf char(11),
      fone varchar(14),
      user_id integer not null,
    created timestamp(0) without time zone DEFAULT NULL,
    modified timestamp(0) without time zone DEFAULT NULL,
      observacao text,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE RESTRICT
);
```

Esta sintaxe funciona no PostgreSQL e também no MySQL.

Combo Estática

Daquelas que não vem do banco, mas são estáticas. Caso precise alterar precisar fazer manualmente no código.

```
          $actions =
['add'=>'add','edit'=>'edit','index'=>'index','vied'=>'view','delete'=>'delete'];
          echo $this->Form->input('action',
['options'=>$actions,'class'=>'col3','empty'=>'Selecione']);
```

Dicas sobre data e hora

Por padrão o Cake mostra apenas os anos de 2011 até 2021 na combo Ano.

Vamos alterar para que o ano mínimo seja 13 anos antes do atual e máximo seja 100 anos antes do atual, ou seja, como estou em 2016, que mostre de 1916 até 2003, mas isso deve ser pensado para atender ao requisito da tabela/aplicativo. No nosso caso, do DNOCS, devemos usar 18 anos antes, ou mais para o primeiro?
E 100 antes do atual ou mais para o segundo.

Precisamos saber da legislação e usar a favor da segurança, deixando uma margem.

```
        echo $this->Form->input('nascimento',['label' => 'Nascimento',
          'dateFormat'    => 'DMY',
          'minYear'       => date('Y') - 100,
          'maxYear'       => date('Y') - 13,
          'empty'         => [
              'day'       => 'Dia',
              'month'     => 'Mês',
              'year'      => 'Ano'
        ]
    ]);
```

Implementação do displayField()

Tenho duas tabelas relacionadas: users e customers.
Customers tem um campo user_id

No Template/Customers/index.ctp no valor do user_id,
mude apenas a primeira ocorrência de $customer->user->id para $customer->user->username:

```
    <td><?= $customer->has('user') ? $this->Html->link($customer->user->username, ['controller' => 'Users', 'action' => 'view', $customer->user->id]) : '' ?></td>
```

Para que nos actions add e edit de customers mostre username ao invés de id mudar em UsersTable.php

No método initialize()

Mudar de id

```
    $this->setDisplayField('id');
```

Para

```
    $this->setDisplayField('username');
```

Agora ao adicionar um customer aparecerá na combo User os usernames e não os ids.

Criação de um Element do tipo topmenu

Este element deve funcionar apenas para aplicativos que aplicaram o plugin Acl e seus usuários chamam-se: admin, manager e user. Caso sejam diferentes faça as devidas alterações.

Abrir o AppController.php

Adicionar as linhas seguintes dentro do método initialize()

```
    $loguser = $this->request->session()->read('Auth.User');
    $loguser = $loguser['username'];
    $this->set('loguser',$loguser);
```

Criar o arquivo topmenu.ctp contendo:

src/Template/Element/topmenu.ctp

```
<?php
    $loggedno = 'Não Logado';
    if(!$loguser) $loguser=$loggedno;

    if($loguser == 'admin'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Funcionários'),
array('plugin'=>null,'controller'=>'Funcionarios','action'=>'index'));
        echo ' '.$this->Html->link(__('Grupos'),
array('plugin'=>null,'controller'=>'Groups','action'=>'index'));
        echo ' '.$this->Html->link(__('Usuários'),
array('plugin'=>null,'controller'=>'Users','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
    }elseif($loguser == 'manager'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Funcionários'),
array('plugin'=>null,'controller'=>'Funcionarios','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
    }elseif($loguser == 'user'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
        /* Modelo
        echo $this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'),['class' => 'button', 'target' =>
'_blank']); */
    }
    echo '   Logado como: '. $loguser;
```

Salvar topmenu.ctp em scr/Template/Element

Editar src\Template\Layout\default.ctp

E adicionar a linha seguinte logo após a linha
<nav class="top-bar expanded" data-topbar role="navigation">:

```
<?php echo $this->element('topmenu');?>
```

Mudar a linha 10 so src\webroot\css\cake.css para
color: #ffffff;

Fluxo de Código

Fluxo de Informações entre controllers, models e view/templates

Temos, por exemplo, o aplicativo clientes

UsersController.php
UsersTable.php
Template/Users/index.ctp

Criar em UsersTable.php a função:

```
public function teste(){
    $query = $this->find('all', [
        'order' => ['Users.id' => 'ASC']
    ]);
    $row = $query->first(); // Ou ->last()
    print "Model<br>";
    return $row->username;
}
```

Chamar no index() do UsersController.php:

```
public function index()
{
    $clientes = $this->paginate($this->Users);

    $this->set(compact('users'));
    $this->set('_serialize', ['users']);

        print "Controller<br>";

        // Mostrar o primeiro username:
        print $this->Users->teste();//exit;

}
```

Chamar pela web

http://localhost/clientes/users/index

Mostrará

Controller
Model
admin

Listagem de Users

View

Veja a ordem:
1) O controller recebe a requisição do usuário para mostrar o endereço:
http://localhost/clientes/users/index

2) O Controller envia para o Model pedindo o primeiro username e a listagem de users

3) O Model processa e devolve

4) Então o controller envia para a view o username e a listagem de users

Passando variáveis pela URL para uma view

Na URL:
localhost/cake-control-demo/pages/home?lang=en&temp=default

No AppController

```
public function initialize()
{
    parent::initialize();

    $this->loadComponent('RequestHandler');
    $this->loadComponent('Flash');

        $lang=$this->request->query('lang');
        $temp=$this->request->query('temp');

        $this->set('lang',$lang);
        $this->set('template',$temp);
}
```

Na View :

```
<div id="content">
    <div class="row">
        <div class="columns large-12 ctp-warning checks">
            <?php print "<h1>Idioma: ".$lang." Template: ".$template."</h1>"; ?>
```

Select com valor default e customização do label

echo $this->Form->control('categoria', $options,['label'=>'Categoria','empty'=>'Selecione a Categoria']);

Select múltiplo (permite selecionar várias opções)

```
print $this->Form->input('pilot_ratings',[
    'type' => 'select',
    'class' => 'listbox',
    'size' => 5,
    'id' => 'pilot_ratings',
    'multiple' => 'multiple',
    'options' => [
        ['name' => 'Habilitación de Vuelo Nocturno Local', 'value' => '1'],
        ['name' => 'Habilitación Cat. II / Cat. III', 'value' => '2'],
        ['name' => 'Habilitación de Remolque de Planeador', 'value' => '5']
    ]
]);
```

Algumas configurações do CakePHP

Pasta config

Os arquivos básicos com configurações a serem customizadas são:
- config/app.php - este guarda configurações sobre:

Debug. Anes de enviar um aplicativo para a produção aterar para false
- 'debug' => filter_var(env('DEBUG', true), FILTER_VALIDATE_BOOLEAN),

Também é importante alterar o salt padrão por conta de segurança, antes de enviar para produção
- 'salt' => env('SECURITY_SALT', 'f504145ed549a05c74bd75e05d7c77b9b6d565de30b06e81dce10144f718ab98'),

Exibição de erros. Por padrão é E_ALL. Usamos o abaixo para não mostrar as mensagens deprecateds
- 'errorLevel' => E_ALL & ~E_USER_DEPRECATED,

Bancos de dados

```
'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            /*
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
```

```
                * the following line and set the port accordingly
                */
               //'port' => 'non_standard_port_number',
               'username' => 'root',
               'password' => 'root',
               'database' => 'crud',
```

Quando for postgresql e estiver usando um esquema diferente do default usamos assim:

```
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Postgres',
            'persistent' => false,
            'host' => 'localhost',
            /*
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
             * the following line and set the port accordingly
             */
            //'port' => 'non_standard_port_number',
            'username' => 'postgres',
            'password' => 'postgres',
            'database' => 'crud',
            'schema' => 'sc_pedidos',
```

Podemos ter várias conexões com bancos de dados e não somente a defualt.

- config/bootstrap.php

Neste podemos configurar data/hora e é muito usado, até a versão 3.5, para armazenar as cargas dos plugins, assim:

- Plugin::load('CakeAclBr', ['bootstrap' => true]);

- config/routes.php

Neste difinimos as rotas do aplicativo, de forma que o aplicativo saiba para onde ir quando se chamar uma URL

- O mais usual e simples é definir quam será o action/método default de um certo controller. Vamos definir o método index do controller Clientes como

o default de um aplicativo:

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Layouts no CakePHP 3

Os layouts são camadas de software que organizam o espaço nas páginas.
Definem o que fica no cabeçalho, menus etc.
O que fica na região de conteúdo.
Na região de cabeçalho,
etc.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title><?= h($this->fetch('title')) ?></title>
<link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
<!-- Include external files and scripts here (See HTML helper for more info.) -->
<?php
echo $this->fetch('meta');
echo $this->fetch('css');
echo $this->fetch('script');
?>
</head>
<body>

<!-- If you'd like some sort of menu to
show up on all of your views, include it here -->
<div id="header">
    <div id="menu">...</div>
</div>

<!-- Here's where I want my views to be displayed -->
<?= $this->fetch('content') ?>

<!-- Add a footer to each displayed page -->
<div id="footer">...</div>
</body>
</html>
```

Podemos definir vários layouts para nosso aplicativo.

Os layouts devem ficar no diretório
src/Template/Layout

O CakePHP já vem com um layout default.ctp.

Atribuindo título para o aplicativo e definindo um layout no AppController:

```
class UsersController extends AppController
{
    public function view_active()
    {
        $this->set('title', 'View Active Users');
        $this->viewBuilder()->layout('default_small_ad');
```

```
    }

    public function view_image()
    {
        $this->viewBuilder()->layout('image');
    }
}
```

View add.ctp

```
<div class="row">
    <div class="col-xs-12">
        <div class="actions">
            <ul class="side-nav btn-group">
                <li class="btn btn-info btn-sm"><?= $this->Html->link(__('List Users'), ['action' =>
'index']) ?></li>
            </ul>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-xs-12">
        <div class="users form">
            <?= $this->Form->create($user, ['role' => 'form']) ?>
            <div class="box box-primary">
                <div class="box-header with-border">
                    <h3 class="box-title"><?= __('Add User') ?></h3>
                </div>
                <div class="box-body">
                    <div class="form-group">
                        <?php
                        echo $this->Form->input('username', ['class' => 'form-control', 'placeholder'
=> __('Enter ...')]);
                        echo $this->Form->input('password', ['class' => 'form-control', 'placeholder'
=> __('Enter ...')]);
                        echo $this->Form->input('email', ['class' => 'form-control', 'placeholder' =>
__('Enter ...')]);
                        echo $this->Form->input('name', ['class' => 'form-control', 'placeholder' =>
__('Enter ...')]);
                        echo $this->Form->input('surname', ['class' => 'form-control', 'placeholder'
=> __('Enter ...')]);
                        ?>
                    </div>
                </div>
                <div class="box-footer">
                    <?= $this->Form->button(__('Submit'), ['class'=>'btn btn-primary']) ?>
                    <?= $this->Html->link(__('Cancel'), ['controller' => 'users'], ['class' => 'btn btn-
warning']) ?>
                </div>
            </div>
            <?= $this->Form->end() ?>
        </div>
```

```
    </div>
</div>
```

Usando o mysql pela linha de comando

Acessar o mysql pela console

mysql -uroot -p

Acessar banco

use banco;

Mostrar tabelas

show tables;

Mostrar estrutura de tabela

desc nome;

Consultas

select * from clientes;

Importar script para um banco

Criar o banco antes
Indicado para quando o phpmyadmin ou o adminer não suportam por ser maior que o php.ini permite

mysql -uroot -p banco <banco sql

Exportar banco para script sql

mysqldump -uroot -p banco > banco .sql

Dicas sobre Validações no CakePHP

As validações no CakePHP são implementadas no Model, especificamente na classe Table.

Algumas Validações no core:

date
datetime
time
timestamp
alphaNumeric
blank
notBlank
boolean
decimal
inList
email
ip
url
numeric
range
userDefined

add() - usado para criar uma validação customizada

Exemplo: src/Model/Table/ClientesTable.php

Campo username unique

```
    $validator
        ->requirePresence('username', 'create')
        ->notEmpty('username')
        ->add('username', 'unique', ['rule' => 'validateUnique', 'provider' => 'table',
'message', 'Este usuário já existe']);
```

Senha

```
    $validator
        ->requirePresence('password', 'create')
        ->notEmpty('password');
```

E-mail requerido

```
    $validator
        ->email('email')
        ->notBlank('email')
        ->add('email', 'unique', [
         'rule' => 'validateUnique',
         'provider' => 'table'
        ]);
```

ou

```
$validator
    ->requirePresence('email')
      ->add('email', 'validFormat', [
          'rule' => 'email',
          'message' => 'E-mail inválido'
      ]);
```

Nome requerido

```
$validator
   ->requirePresence('nome', 'create')
   ->notEmpty('nome');
```

No Model/Table/ClientesTable.php

Rule customizada para o campo nascimento

```
public function validationDefault(Validator $validator)
{
    $validator
       ->integer('id')
       ->allowEmpty('id', 'create');

    $validator
       ->requirePresence('nome', 'create')
       ->notEmpty('nome');

    $validator
       ->email('email')
       ->notBlank('email')
       ->add('email', 'unique', ['rule' => 'validateUnique', 'provider' => 'table']);

    $validator->add('nascimento',[
         'notEmptyCheck'=>[
           'rule'=>[$this,'notEmptyNascimento'],
           'provider'=>'table',
           'message'=>'Favor selecionar uma data de nascimento'
        ]
    ]);

    $validator
       ->allowEmpty('cpf');

    $validator
       ->allowEmpty('cnpj');

    $validator
       ->allowEmpty('fone');
```

```
        $validator
            ->allowEmpty('observacao');

        $validator
            ->notBlank('user_id');

        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Dica: O campo group_id não aparecia por padrão na lista de validações e mesmo que não fosse selecionado nenhum grupo o registro era armazenado. Então adicionei a validação como notBlank e na view add.ctp usei:

```
        echo $this->Form->input('user_id', ['options' => $users,'empty'=>true]);
```

Assim aparece o asterisco vermelho indicando requerido e somente quando alguém esoclhe um grupo na lista é permitodo cadastrar.

Sugestão para lista

```
    public function validationDefault(Validator $validator)
    {
        return $validator
            ->notEmpty('first_name', 'A username is required')
            ->notEmpty('last_name', 'A username is required')
            ->notEmpty('email', 'A username is required')
            ->notEmpty('username', 'A username is required')
            ->notEmpty('password', 'A username is required')
            ->add('role', 'inList', [
                'rule' => ['inList', ['Employer', 'Job Seeker']],
                'message' => 'Please enter a valid role'
            ]);
    }
```

Como o Cake exige que as datas sejam null por default, uma forma de contornar isso exigindo o preenchimento é criando uma validação customizada:

No Model/Table/ClientesTable.php

```
    public function validationDefault(Validator $validator)
    {
            ...
        $validator->add('nascimento',[
             'notEmptyCheck'=>[
```

```
                'rule'=>[$this,'notEmptyNascimento'],
                'provider'=>'table',
                'message'=>'Favor selecionar uma data de nascimento'
            ]
        ]);
            ...
        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Validação de telefone

```
'fone' => [
        'rule' => array('isValidBRFoneFormat')
    ],

// Formatos aceitos: (99) 99999-9999 e (99) 9999-9999

 /*isValidBRFoneFormat() - Custom method to validate US Phone Number
 * @params Int $phone
 */
 function isValidBRFoneFormat($phone){
 $fone=$fone['fone'];
 $errors = array();
    if(empty($fone)) {
        $errors [] = "Favor entrar um tenefone válido";
    }
    else if (!preg_match('/^(\(11\) [9][0-9]{4}-[0-9]{4})|(\(1[2-9]\) [5-9][0-9]{3}-[0-9]{4})|(\([2-9]
[1-9]\) [5-9][0-9]{3}-[0-9]{4})$/', $fone)) {
        $errors [] = "Favor entrar um tenefone válido";
    }

    if (!empty($errors))
    return implode("\n", $errors);

    return true;
}
```

Criação de um aplicativo tipo CRUD com o CakePHP3

O principal objetivo aqui é criar um aplicativo tipo CRUD com bons recursos e de forma bem prática.

Vamos começar usando o recurso mais prático do CakePHP, que é o gerador de aplicativos chamado bake.

Iremos criar um simples CRUD mas já é um recurso muito prático e funcional, que pode ser utilizado para a criação de aplicativos simples para nosso dia a dia. Mais a frente iremos abordar recursos mais avançados.

Estou usando o Linux com diretório web em
/var/www/html

Caso esteja usando outro sistema operacional faça os devidos ajustes

Caso esteja usando outro ambiente ou em outro diretório web altera o caminho abaixo

- Criar o aplicativo base usando o composer na pasta crud
    cd /var/www/html

    composer create-project --prefer-dist cakephp/app crud

Aguarde a instalação e ao final apenas tecle ENTER.

Para saber qual a versão atual do Cake visite este site:

https://github.com/cakephp/cakephp/releases

Ou então, depois de instalado, visualize este arquivos:
vendor/cakephp/cakephp/VERSION.txt

- Criar o banco de dados no MySQL chamado 'crud' com apenas uma tabela chamada 'clientes'

Sugestão de gerenciador de bancos de dados: o gerenciador web em PHP
http://adminer.org

```
CREATE TABLE `clientes` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` char(45) COLLATE utf8mb4_unicode_ci NOT NULL,
  `email` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
);

INSERT INTO `clientes` (`nome`, `email`) VALUES
('Ribamar FS',   'ribafs@gmail.com'),
('João Britu Cunha','joao@gmail.com'),
('Pedro Álvares',      'pedro@gmail.com'),
```

('Antônio',   'ant@gm.com'),
('Jorge','jo@df.com'),
('Cliente',   'cli@cli.com'),
('João Brito',   'pa@cli.com'),
('Caarlos Alberto',   'pa2@cli.com'),
('Roberto Pereira',   'pa22@cli.com');

- Configurar o banco no aplicativo

```
cd /var/www/html/crud
nano config/app.php
```

Como usaremos o mysql basta alterar estar linhas em Datasource

```
'username' => 'root',
'password' => '',
'database' => 'crud',
```

- Acessar o aplicativo pelo navegador

```
http://localhost/crud
```

Ele nos mostra uma página padrão criada automaticamente pelo Cake, mas sem nada do nosso banco

- Configurar o routes.php para Clientes com index

```
nano config/routes.php
```

Mudar apenas esta linha

```
cd /var/www/html/crud
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Para

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

- Gerar o CRUD com o bake

Execute

```
cd /var/www/html/crud
bin/cake bake all clientes
```

Observe que ele mostra mensagens de que criou todo o nosso código: controller, model, view/template entre outros recursos

Chame pelo navegador

```
http://localhost/crud
```

Ele nos mostra um aplicativo completo, com todo o CRUD, com CSS padrão, paginação, ordenação dos campos apensa clicando em seus labels, etc.

Clique em New Criante e crie um novo cliente.

Entre um nome e entre um e-mail incompleto, apenas o login e clique em SUBMIT.
Ele já traz validação para o e-mail. Ele verificou na tabela o nome do campo "email" e já aplicou validação.

Uma beleza.

Dica: Para que o CakePHP nos preste este serviço o campo email na tabela deve ser NOT NULL.

Corrija e clique em SUBMIT

Como a equipe do Cake está se preparando para lançar a versão 4.0, talvez receba a mensagem de erro:

Deprecated (16384): RequestHandlerComponent::beforeRedirect() is deprecated. This functionality will be removed in 4.0.0. Set the `enableBeforeRedirect` option to `false` to disable this warning. - /var/www/html/crud/vendor/cakephp/cakephp/src/Event/EventManager.php, line: 353
You can disable deprecation warnings by setting `Error.errorLevel` to `E_ALL & ~E_USER_DEPRECATED` in your config/app.php. [CORE/src/Core/functions.php, line 307]

Caso receba edite novamente o arquivo

config/app.php

E altere a linha

'errorLevel' => E_ALL,

Para
'errorLevel' => E_ALL & ~E_USER_DEPRECATED,

E chame novamente

http://localhost/crud

Veja que temos um CRUD funcional (Add, Edit, View e Delete, além de outrosr ecursos), que podemos usar para o cadastro de clientes, uma agenda e qualquer outra finalidade.

Sugestões de leitura:

Uma recomendação é instalar o plugin cake-acl-br, que instala um template com o Bootstrap e um controle de acesso para o aplicativo:

https://github.com/ribafs/cake-acl-br

O site oficial do CakePHP é uma das grandes forças do framework, com tutoriais de exemplo de vários aplicativos:
https://book.cakephp.org/3.0/en/tutorials-and-examples.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog/blog.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog-auth-example/auth.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/bookmarks/intro.html

E uma ótima documentação, que pode ser vista online
https://book.cakephp.org/3.0/en/index.html

Ou offline em PDF ou Epub
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.pdf
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.epub

Criação de um aplicativo tipo CRUD com o CakePHP3 via código

O objetivo maior deste tutorial é criar um aplicativo tipo CRUD com o Cake, mas agora apenas codificando, sem a ajuda do bake.

Vamos seguir as dicas do CakePHP para ir criando nosso aplicativo.

Este aplicativo será apenas para mostrar como funcionam alguams coisas no CakePHP. Ele não faz nada de tão útil mas conhecer estas coisas no Cake é muito importante. Quando mais você dominar o funcionamento mais poderá fazer com o framework.

- Criar o aplicativo base usando o composer na pasta crud0
cd /var/www/html

composer create-project --prefer-dist cakephp/app crud0

- Configurar o banco no aplicativo
Neste exemplo não precisaremos configurar o banco de dados, pois usaremos apenas código sem acesso a bancos.

Configure o routes.php de forma semelhante, para Clientes/index

- Acessar o aplicativo pelo navegador
http://localhost/crud0

Ele nos mostra uma página padrão criada automaticamente pelo Cake com diversas informações úteis, mas sem nada do nosso banco

- Vamos tentar acessar nossa tabela clientes
http://localhost/crud0/clientes

Veja os erros:
Error: ClientesController could not be found.

Error: Create the class ClientesController below in file:
src/Controller/ClientesController.php

```
<?php
namespace App\Controller;

use App\Controller\AppController;
class ClientesController extends AppController
{

}
```

Ele diz que precisamos ter um controller chamado Clientes e nos indica uma estrutura básica. Então vamos seguir suas dicas

cd /var/www/html/crud0

Lembre que a pasta src guarda todo o código do aplicativo em CakePHP. Todo o código que adicionamos diretamente para o aplicativo deve ficar nesta pasta.
Crie o arquivo:

nano src/Controller/ClientesController.php
Contendo

```
<?php
namespace App\Controller;

use App\Controller\AppController;

class ClientesController extends AppController
{

}
```

Chamemos agora novamente pelo navegador

http://localhost/crud0/clientes

Veja os erros agora

The action index is not defined in ClientesController

Error: Create ClientesController::index() in file: src/Controller/ClientesController.php.

```
<?php
namespace App\Controller;

use App\Controller\AppController;

class ClientesController extends AppController
{

    public function index()
    {

    }
}
```

Ele está dizendo que precisamos criar o action index() no nosso controller, então vamos criar como indicado dentro da classe

```
public function index()
{

}
```

Chamemos novamente pelo navegador

http://localhost/crud0/clientes

E agora os erros são diferentes:

Error: The view for ClientesController::index() was not found.

Confirm you have created the file: "Clientes/index.ctp" in one of the following paths:

/var/www/html/crud/src/Template/Clientes/index.ctp

Ele nos pede uma view chamada index.ctp em src/Template/Clientes

```
mkdir /var/www/html/crud/src/Template/Clientes
nano /var/www/html/crud/src/Template/Clientes/index.ctp
```

Criando apenas o arquivo index.ctp vazio já atende e não mais veremos erro.
Mas a tela abrirá limpa, somente com o título Clientes

Adicione para o index.ctp:

```
<h1>Meu primeiro aplicativo</h1>
```

Chamemos novamente pelo navegador

http://localhost/crud0/clientes

Porque ao chamar clientes ele abre o action index() que chama o index.ctp correspondente?
É o padrão do Cake, ao chamar um controller, ele chama por padrão o action index().
E ao chamar um action ele chama o template correspondente, no caso o index.ctp.

Agora nada de erro, apenas a mensagem que adicionamos.

Veja toda a sequência:
- Criar o controller
- Criar no controller o método/action ClientesController/index()
- Criar uma view/template correspondente ao método do controller, no caso o Tempalte/Clientes/index.ctp

Veja como o Cake ajuda na criação. Em algumas etapas ele diz exatamente o que precisamos fazer.

- Configurar o routes.php para que nosso aplicativo acesse automaticamente o action index() do controller Clientes

```
cd /var/www/html/crud0
nano config/routes.php
```

Mudar apenas esta linha

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Para

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Fazendo isso podemos chamar nosso aplicativo assim:

```
http://localhost/crud0
```

E ele automaticamente chamará

```
http://localhost/crud0/clientes/index
```

Pois definimos o controller Clientes e o action index como defaults.

O que vem a seguir?

Veja que nosso aplicativo crud0 não é funcional. Seu código não é usável. Não trabalha com o banco de dados.
Na próxima etapa criaremos o aplicativo crud1 e então trabalharemos com o banco de dados.

Criação de um aplicativo tipo CRUD com o CakePHP3

O objetivo maior deste tutorial é criar um aplicativo tipo CRUD com o Cake, mas agora apenas codificando, sem a ajuda do bake.

Vamos seguir de onde paramos no tutorial anterior sobre o crud0.

- Criar o aplicativo base usando o composer na pasta crud1
cd /var/www/html
composer create-project --prefer-dist cakephp/app crud1

- Vamos usar o mesmo banco 'crud' criado no tutorial anterior com a tabela clientes

- Configurar o banco no aplicativo
cd /var/www/html/crud1
nano config/app.php
Como usaremos o mysql basta alterar estar linhas em Datasource
'username' => 'root',
'password' => '',
'database' => 'crud',

Aproveite e altere a linha com
'errorLevel' => E_ALL,
Para
'errorLevel' => E_ALL & ~E_USER_DEPRECATED,

- Configurar o routes.php para que nosso aplicativo acesse automaticamente o action index() do controller Clientes
cd /var/www/html/crud1
nano config/routes.php

Mudar apenas esta linha
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
Para
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);

Fazendo isso podemos chamar nosso aplicativo assim:

http://localhost/crud1

E ele chamará
http://localhost/crud1/clientes/index

Nos mostrará erros dizendo que não temos o controller clientes.

Criação do controller ClientesController.php com o action index()

O CakePHP é um framework que segue o padrão MVC. Quando criamos um aplicativo sem criar explicitamente o model ele cria um model para nós.
Vejamos. Vamos criar o controller

src/Controller/ClientesController.php

Contendo:

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class ClientesController extends AppController
{
    public function index()
    {
        $clientes = $this->paginate($this->Clientes);
        $this->set(compact('clientes'));
    }

}
```

Veja que é basicamente o mesmo método index() criado pelo bake, mas sem os comentários para simplificar.

Este método usa a função set() para enviar a variável clientes, que contém todos os registros da tabela clientes para a view chamada pelo método index, que no caso é src/Template/Clientes/index.ctp

Criação da View/Template Clientes/index.ctp

Então criemos o diretório

src/Template/Clientes

E dentro dele o template

index.ctp

Contendo

```
<?php
foreach ($clientes as $cliente){
    echo $cliente->id.'-'.$cliente->nome.'-'.$cliente->email.'<br>';
}
```

Melhorando um pouco a aparência do nosso template index.ctp:

```
<table>
<?php
print '<tr><td>ID</td><td>Nome</td><td>E-mail</td></tr>';
foreach ($clientes as $cliente){
    print '<tr><td>'.$cliente->id.'</td><td>'.$cliente->nome.'</td><td>'.$cliente-
>email.'</td></tr>';
}
```

```
print '</table>';
```

Aqui recebemos a variável $clientes e varremos a mesma com o laço foreach().

Ele mostrará todos os campos de todos os registros.

Veja que fizemos tudo isso apenas com o controller e o template/view, não tocamos em model. O cake cuidou disso para nós.

Criação do action add() e do tempalte add.ctp

Adicionar o método/action add() ao nosso controller clientes

```
    public function add()
    {
        $cliente = $this->Clientes->newEntity();
        if ($this->request->is('post')) {

            // o método patchEntity(), como o newEntity() validará os dados antes de ser salvo
na entidade
            $cliente = $this->Clientes->patchEntity($cliente, $this->request->getData());

            // Salvar o registro cliente
            if ($this->Clientes->save($cliente)) {
                return $this->redirect(['action' => 'index']);
            }
        }
        // Armazena os dados do registro clientes na variável cliente e envia para o add.ctp
        $this->set(compact('cliente'));
    }
```

É também o método criado pelo bake mas simplificado.

Se chamar pelo navegador agora com:

http://localhost/crud1/clientes/add

Ele reclamará do tempalte add.ctp correspondente

Criar o template src/Template/Clientes/add.ctp

Contendo este código abaixo, que é uma simplificação do criado pelo bake

```
<?= $this->Form->create($cliente) ?>
<legend><?= __('Add Cliente') ?></legend>
    <?php
        echo $this->Form->control('nome');
        echo $this->Form->control('email');
    ?>
```

```
<?= $this->Form->button(__('Submit')) ?>
<?= $this->Form->end() ?>
```

Como não temos nenhum link para esta view podemos chamá-la pelo navegador assim:

http://localhost/crud1/clientes/add

Criando o action a o template view

Adicionar ao controller o método abaixo

```
public function view($id = null)
{
    $cliente = $this->Clientes->get($id);
    $this->set(compact('cliente'));
}
```

Agora chame pelo navegador com

http://localhost/crud1/clientes/view/3

Ele informará que o view.ctp não existe

Então crie o arquivo src/Template/Clientes/view.ctp

Contendo

```
<table class="vertical-table">
    <tr>
        <th>Nome</th>
        <td><?= $cliente->nome ?></td>
    </tr>
    <tr>
        <th>E-mail</th>
        <td><?= $cliente->email ?></td>
    </tr>
    <tr>
        <th>Id</th>
        <td><?= $cliente->id ?></td>
    </tr>
</table>
```

Agora chame pelo navegador com

http://localhost/crud1/clientes/view/3

Criar o action e o template edit

```
public function edit($id = null)
{
    $cliente = $this->Clientes->get($id);
```

```
        if ($this->request->is(['patch', 'post', 'put'])) {
            $cliente = $this->Clientes->patchEntity($cliente, $this->request->getData());
            if ($this->Clientes->save($cliente)) {
                return $this->redirect(['action' => 'index']);
            }
        }
        $this->set(compact('cliente'));
    }
```

```
<?= $this->Form->create($cliente) ?>
    <h3>Editar Cliente<>
    <?php
        echo $this->Form->control('nome');
        echo $this->Form->control('email');
    ?>
<?= $this->Form->button(__('Submit')) ?>
<?= $this->Form->end() ?>
```

Agora chame pelo navegador com:

http://localhost/crud1/clientes/edit/3

Ele mostrará um formulário com o registro 3 para ser editado.

Agora o interessante é a criação de uma grid contendo a listagem de todos os registros (index), um botão para adicioanr novos registros e cada registro contendo um link para editar, visualizar e excluir, como fez o Bake.

Mas ainda falta muito para ficar parecido com o código gerado pelo bake:
- Paginação
- Ordenação dos campos
- Estilo com CSS
- ...

O que vem a seguir?

Veja que nosso aplicativo crud1 não está completo.
Criaremos o aplicativo crud2 que mostrará links para editar, visualizar e excluir os registros no index.ctp.

Criação de um aplicativo tipo CRUD com o CakePHP3 mas na unha

O objetivo maior deste tutorial é criar um aplicativo tipo CRUD com o Cake, mas agora apenas codificando, sem a ajuda do bake.
Vamos seguir de onde paramos no tutorial anterior sobre o crud1.

Desta vez não criaremos outro aplicativo iremos apenas copiar a pasta do crud1 e renomear para crud2 aproveitando tudo que fizemos Caso seja necessário num Linux ajuste as permissões dos arquivos da pasta crud2.

Criar os links para editar, ver e excluir no index.ctp

Chame pelo navegador com:

http://localhost/crud2

Veja que ele lista todos os registros, com respectivos id, nome e email.
Vamos adicionar uma coluna/label chamada Ações após o email e ela conterá em cada registro os três links: editar, ver e excluir, como faz o Bake.

Vamos para isso editar o src/Template/Clientes/index.ctp

Atualmente ele está assim:

```
<table>
<?php
print '<tr><td>ID</td><td>Nome</td><td>E-mail</td></tr>';
foreach ($clientes as $cliente){
    print '<tr><td>'.$cliente->id.'</td><td>'.$cliente->nome.'</td><td>'.$cliente-
>email.'</td></tr>';
}
```

Vamos alterar (aproveitando o código gerado pelo Bake no app crud):

```
<table>
<?php
print '<tr><td><strong>ID</td><td><strong>Nome</td><td><strong>E-
mail</td><td><strong>Ações</td></tr>';
?>
       <tbody>
          <?php foreach ($clientes as $cliente): ?>
          <tr>
             <td><?= $cliente->id ?></td>
             <td><?= h($cliente->nome) ?></td>
             <td><?= h($cliente->email) ?></td>
             <td class="actions">
                <?= $this->Html->link(__('View'), ['action' => 'view', $cliente->id]) ?>
                <?= $this->Html->link(__('Edit'), ['action' => 'edit', $cliente->id]) ?>
                <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $cliente->id],
['confirm' => __('Are you sure you want to delete # {0}?', $cliente->id)]) ?>
             </td>
```

```
        </tr>
        <?php endforeach; ?>
    </tbody>
</table>
```

No crud1 faltou a criação do método delete() para o controller Clientes

Adicione ao controller ClientesController.php

```
public function delete($id = null)
{
    $this->request->allowMethod(['post', 'delete']);
    $cliente = $this->Clientes->get($id);
    $this->Clientes->delete($cliente);
    return $this->redirect(['action' => 'index']);
}
```

Veja que mais uma vez eu simplifiquei o código, removendo validações e mensagens de erro ou acerto apenas para facilitar o entendimento.
Mas o ideal é fazer como faz o Bake, com tudo que tem direito para tornar o código mais amigável e mais seguro.

Agora podemos chamar novamente pelo navegador e testar todas as operações do crud:

http://localhost/crud2

Ops, ainda falta a opção na index para criar um novo registro, ou seja, um link para o add.ctp

Vamos adicioná-lo acima da listagem, logo abaixo da tag <table>.

```
<li><?= $this->Html->link(__('New Cliente'), ['action' => 'add']) ?></li>
```

Assim nosso CRUD está completo. Todo criado linha a linha do código.

O que vem a seguir?

Veja que nosso aplicativo crud2 não está completo mas já está funcional, inserindo, editando, visutalizando e excluindo registros.
Para excluir precisamos incluir o action delete(). Inclua este action:

```
public function delete($id = null)
{
    $this->request->allowMethod(['post', 'delete']);
    $cliente = $this->Clientes->get($id);
    if ($this->Clientes->delete($cliente)) {
        $this->Flash->success(__('The cliente has been deleted.'));
    } else {
        $this->Flash->error(__('The cliente could not be deleted. Please, try again.'));
    }

    return $this->redirect(['action' => 'index']);
```

```
}
```

Falta muita coisa para que fique decente. Mas nesta linha pararemos por aqui.
Daqui pra frente estudaremos as partes mais avançadas do código do Cake:
- Como validar informações usando o model
- Como programar nas três camadas, model, crontroller e view e fazer as informações conversarem entre si, mostrando todo o fluxo, onde ela começa e onde ela termina. Desde a solicitação do usuário ao clicar em um link, em um botão ou chamar um link até a resposta final da informação voltando para o usuário. Com isso precisaremos usar funções que são próprias do model, outras que são próprias do controller e outras que são próprias da view.

Em seguida veremos o aplicativo app_code.

Existe muita coisa para aprendermos.

Onde obter ajuda?

O site do Cake é a melhor fonte de consulta que conheço. Com toda a documentação fornecida online. Também é oferecida em forma de pdf e epub.

Mas não existe documentação perfeita. Seria praticamente impossível chegar perto. Devemos contar com outras fontes de pesquisa que ajudam:

Lista oficial em inglês, onde podemos tirar dúvidas - https://discourse.cakephp.org/
Forum sobre CakePHP em inglês no StackOverflow - https://stackoverflow.com/tags/cakephp/new
Forum do StackOverflow em pt-br sobre o CakePHP 3 - https://pt.stackoverflow.com/questions/tagged/cakephp-3
Claro que também temos os instrumentos de busca caso falte algo.

Criação de Aplicativo tipo Controle de Estoque Simplificado

Com o plugin cake-acl-br para implementar ACL

Este aplicativos será criado em 3 versões e usará um script de banco de dados do CEP do Ceará, meio desatualizado:

Todos iniciando com a versão 3.5.13 e migrando para a atual

- Usando o banco com MySQL

Gerar diagrama do Banco com o dbVisualzier

Aplicativo com o MySQL

Criar o banco e importar o script estoque_my.sql

Por conta das mudanças no CakePHP 3.6::

- Devemos primeiro instalar o CakePHP 3.5.13

https://github.com/cakephp/cakephp/archive/3.5.13.zip

Descompactar
Renomear a pasta para estoque

Obs.: isso somente por enquanto o cake-acl-br está incompatível com o CakePHP 3.6. Alias, não estou conseguindo instalar nenhum plugin nesta versão, inclusive os da equipe como o migrations.

- Instalar e carregar o plugin cake-acl-br

cd /var/www/html/estoque

composer require ribafs/cake-acl-br
bin/cake plugin load CakeAclBr --bootstrap

- Configurar o banco em config/app.php
E o config/routes.php para Clientes/index

- Configurar o plugin cake-acl-br:
Copiar os arquivos sobrescrevendo:
De docs/bootstrap_cli.php para config
De docs/AppController.php para src/Controller
De docs/custom.css para webroot/css
De docs/pdo_error.ctp para src/Template/Error

- Gerar todo o código com o Bake

bin/cake bake all groups -t CakeAclBr
Repetir o procedimento acima para todas as demais

Acessar
http://localhost/estoque

Algo fortuito é que o menu como está está em ordem alfabética.

Como consultar a versão do CakePHP 3?

Tá no arquivo
vendor/cakephp/cakephp/VERSION.txt

Agora façamos alguns ajustes para melhorar nosso aplicativo:

1) Aterar a largura de campos nos forms.
Para isso usaremos as classes criadas no plugin cake-acl-br para o framework Bootstrap, que estão no arquivo webroot/css/custom.css

1.1) Ajustar a largura das colunas em src/Template/Permissions/add.ctp

```
<?php
    echo $this->Form->input('group_id', ['options' => $groups, 'autofocus'=>'true',
'class'=>'col4']);
    echo $this->Form->input('controller', ['class'=>'col4']);
    echo $this->Form->input('action',['class'=>'col4']);
?>
```

1.2) Aproveitar e ajustar também as larguras de src/Template/Permissions/edit.ctp

Implementar a busca nas Permissões

- Editar o arquivo src/Controller/PermissionsController.php

- Remover o action index() ativo
- Mover "use Cake\Datasource\ConnectionManager;" para baixo da linha "use App\Controller\AppController;"
- Remover o final do comentário que está abaixo do index()
- Ajustar o index() que estava comentado para que fique assim, para que possamos pesquisar pelo group e pelo controller:

```
public function index()
{
    $conn = ConnectionManager::get('default');
    $driver = $conn->config()['driver']; // Outros: database, etc.

    if($driver == 'Cake\Database\Driver\Postgres'){
        $this->paginate = [
            'contain' => ['Groups'],
```

```
                'conditions' => ['or' => [
                    'Permissions.group ilike' => '%' . $this->request->query('search') . '%',
                    'Permissions.controller ilike' => '%' . $this->request->query('search') . '%'
                ]],
                'order' => ['Permissions.id' => 'DESC' ]
            ];
        }elseif($driver=='Cake\Database\Driver\Mysql'){
            $this->paginate = [
                'contain' => ['Groups'],
                'conditions' => ['or' => [
                    'Permissions.group like' => '%' . $this->request->query('search') . '%',
                    'Permissions.controller like' => '%' . $this->request->query('search') . '%'
                ]],
                'order' => ['Permissions.id' => 'DESC' ]
            ];
        }else{
              print '<h2>Driver database dont supported!';
              exit;
        }

          $this->set('permissions', $this->paginate($this->Permissions));
        $this->set('_serialize', ['permissions']);
    }
```

- Agora ajustemos o src/Template/Permissions/index.ctp. Veja que o trecho de código da busca está comentado no início. Descomente e deixe assim:

```
    <?php
        echo $this->Form->create(null, ['type' => 'get']);
        echo $this->Form->label('Busca');
        echo $this->Form->input('search', ['class' => 'form-control', 'label' => false,
'style'=>'width:170px;',
          'placeholder' => 'Group_id ou controller', 'value' => $this->request-
>query('search')]);
        echo $this->Form->button('Busca');
        echo $this->Form->end();
    ?>
```

- Agora já pode acessar novamente Permissions e efetuar uma busca para testar

- Vamos trocar os tipos de campos controller e action do form src/Template/Permissions/add.ctp de text para select, já trazendo os respectivos

Action

Comente a linha do campo action e adicione esta abaixo:

```
        $options =
['index'=>'index','add'=>'add','edit'=>'edit','view'=>'view','delete'=>'delete'];
        echo $this->Form->input('action', ['options'=>$options,'required'=>'false',
'class'=>'col-md-12', 'empty'=>'Selecione','class'=>'col4']);
```

Controller apenas a sugestão, se fossem apenas os campos originais seria assim. Lembrando que para permissions somente estes controllers são válidos:

```
$options2 = ['Customers'=>'Customers','Groups'=>'Groups', 'Users'=>'Users',
'Permissions'=>'Permissions', 'Products'=>'Products', 'ProductItems'=>'ProductItems',
'value'=>'Selecione'];
echo $this->Form->input('controller',['options'=>$options2,'required'=>'false', 'class'=>'col-
md-12','empty'=>'Selecione','class'=>'col4']);
```

- Remover a coluna Password do src/Template/Users/index.ctp

Remover as linhas:

```
            <th><?= $this->Paginator->sort('password') ?></th>
e
            <td><?= h($user->password) ?></td>
```

- Mostrar em src/Template/CompraItens/index.ctp (produto->id mudar para produto->descricao)

```
            <td>
                <?= $compraIten->has('compra') ? $this->Html->link($compraIten->compra->id, ['controller' => 'Compras', 'action' => 'view', $compraIten->compra->id]) : '' ?>
            </td>
```

- Mostrar em src/Template/Funcionarios/index.ctp (empresa->id mudar para empresa->razao_social)

```
            <td>
                <?= $funcionario->has('empresa') ? $this->Html->link($funcionario->empresa->razao_social, ['controller' => 'Empresas', 'action' => 'view', $funcionario->empresa->id]) : '' ?>
            </td>
```

- Mostrar em src/Template/Users/index.ctp (user->id mudar para user->groúp_id)

```
            <td>
                <?= $user->has('group') ? $this->Html->link($user->group->name, ['controller' => 'Groups', 'action' => 'view', $user->group->id]) : '' ?>
            </td>
```

- Adicionar 'Selecione' para a combo de User em add.ctp de todas as tabelas,

A linha deve ficar assim:

```
        echo $this->Form->input('user_id', ['options' => $users, 'empty'=>'Selecione']);
```

- Aproveite e mude a largura do campo para a classe col4

```
        echo $this->Form->input('user_id', ['options' => $users, 'empty'=>'Selecione',
'class'=>'col4']);
```

- Mostrar nome do cliente e do funcionario na index de pedidos

src/Tempalte/Pedidos/index.php

Mude nas linhas:

```
                <?= $pedido->has('cliente') ? $this->Html->link($pedido->cliente->nome,
['controller' => 'Clientes', 'action' => 'view', $pedido->cliente->id]) : '' ?>
            </td>
        <td>
                <?= $pedido->has('funcionario') ? $this->Html->link($pedido->funcionario-
>nome, ['controller' => 'Funcionarios', 'action' => 'view', $pedido->funcionario->id]) : '' ?>
```

Mudar
$pedido->cliente->nome
$pedido->funcionario->nome

Observe que em compra_itens ele não mostrou o estoque_id como combo
Então vamos ajustar isso

Editar o CompraItensController.php

Vamos ajustar o action add()

Adicone a linha abaixo:

```
    $estoque = $this->CompraItens->Estoque->find('list', ['limit' => 200]);
```

E altere esta

```
    $this->set(compact('compraIten', 'compras', 'produtos', 'estoque'));
```

Alterar src/Template/CompraItens/add.ctp

Apenas a linha

```
        echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

Agora com o action edit

Adicione

```
    $estoque = $this->CompraItens->Estoque->find('list', ['limit' => 200]);
```

Altere

```
    $this->set(compact('compraIten', 'compras', 'produtos','estoque'));
```

No action edit()

Altere apenas a linha:

```
        echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

Adicionar Estoque para o action index() na linha

```
        'contain' => ['Compras', 'Produtos', 'Estoque']
```

Agora na view edit.ctp

```
echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

No index.ctp adicione logo abaixo da linha

```
<td><?= $this->Number->format($compraIten->preco_unitario) ?></td>
```

Estas:

```
<td>
  <?= $compraIten->has('estoque') ? $this->Html->link($compraIten->estoque_id, ['controller' => 'Estoque', 'action' => 'view', $compraIten->estoque_id]) : '' ?>
</td>
```

Boa parte dos formulários nesta versão do plugin recebe o foco no primeiro campo ao abrir, inclusive o login usando a propriedade autofocus do HTML5:

'autofocus' => 'true'

Nesta etapa estaremos codificando nosso aplicativo para adicionar a camada de negócios do estoque:

Precisamos tomar um grande cuidado quando estivermos modelando um banco de dados, ao criar astabelas, seus campos, suas constraints, chaves primárias, chaves estrangeiras. Esta fase do projeto é vital. Erros aqui se propagam e prejudicam o projeto. Algo que não pode acontecer é uma tabela A ter um relacionametno com uma tabela B e também a tabela B ter o relacionamento com a tabela A. Cuidado para não cometer estes descuidos. Claro que precisamos ter bem claros os conceitos de bancos de dados e das formas normais em mente.1

Cadastros primários que não têm relacionamentos, exceto com users e ceps e também as tabelas administrativas, groups, users e permissions:

Chamadas Tabelas primárias, pois outras se relacionam com ela e estas não guardam chaves estrangeiras

- empresas
- ceps
- armazens
- clientes
- fornecedores
- produtos

Tabelas secundárias, que contém relacionamentos, ou seja, um campo que relaciona com a tabela principal e a chave estrangeira:

- funcionarios
- compras
- compra_itens
- pedidos
- pedido_itens
- bonus
- comissoes
- estoque

São num total 14 tabelas

Tipos de dados

Estoque, comprar, pedidos devem ter um tipo de dados datetime para permitir mais de um por dia. Aqui é uma simplificação

Sequência de Cadastro

- Cadastramos sempre as informações nas tabelas primárias. Não posso cadastrar uma empresa sem o CEP (isso se CEP for obrigatório). Não posso cadastrar um pedido para um cliente se o cliente ainda não foi cadastrado. Primeiro temos que cadastrar o cliente.

- Antes de cadastrar um funcionario precisamos cadastrar a empresa

- Antes de cadastrar uma compra precisamos cadastrar os fornecedores

- Assim sucessivamente, cadastrar primeiro a tabela primária depois a secundária/estrangeira. Para o usuário que está cadastrando, basta abrir o add e olhar os campos. Se existir alguma combo que traz ou trará informações de outra tabela ele precisa cadastrar primeiro esta tabela da combo. Exemplo, ele abre Produtos para cadastrar um novo produto. Sem problema, pois não existem combos. Todas as tabelas cujo nome tem Itens, precisa cadastrar a principal antes (compras, pedidos). Mas se ele quizer cadastrar um novo estoque. Veja que tem uma combo Armazen. Se ele for cadastrar em um dos armazens da combo tá beleza, mas se for em um que não está na relação, ele deve fechar o estoque e abrir Armazens para cadastrar o novo armazen. Ainda no estoque ele percebe que não existe CompraItens cadastrado. Então terá que abrir CompaItens e talvez compras e cadastrar primeiro a compra e depois voltar para Estoque. Também acontece se existirem compraitens cadasrados mas não existir a compra iten que ele deseja. Precisará cadastrar antes a compra e somente depois voltar para o Estoque.

Criação de DER

- PostgreSQL - dbVisualizer
- MySQL - Workbench - Database - Reverse Engineer

Veja o para o MySQL com o dbVisualizer