**Roteiro para Experimento Grupo 2**

Para este experimento é necessário realizar o download do repositório **Git:** https://github.com/AllanKaio21/experimento-testes.gitem uma pasta.

Abram o cmd (Windows) ou terminal (Linux) dentro da pasta do repositório baixado(base) e execute: *docker volume create postgres\_database2*. Verifique se a porta “880” está em uso em seu desktop, se estiver entre na pasta “base” da aplicação e dentro do arquivo “docker-compose.yml” altere no escopo “app” a porta para uma disponível. Após verificar a porta execute: *docker-compose up -d*. Acesse [http://*localhost:880*](http://localhost:8080/) *ou* [http://*localhost:<porta*](http://localhost:8080/) *alterada> no seu browser*. Com a aplicação aberta, poderão ver que são dois simples *CRUD* (*Create, Read, Update, Delete*).

Neste experimento deve se cronometrar o tempo gasto para realizar as tarefas propostas, este tempo deve ser anotado em um arquivo especifico informado.

**Iniciando o Experimento**

Antes de iniciar o experimento, deve-se **Iniciar a Cronometragem**.

Nesta parte e necessário criar os testes automáticos manualmente seguindo exemplo do teste automático já informado, sendo necessário apenas os testes funcionais, dentre os para testar o CRUD da aplicação, sendo um que aplica formulário apenas para dados do modelo, teste para alterar dados já cadastrados e um para deletar dados cadastrados de acordo com o exemplo a seguir.

Exemplo de um arquivo de teste automático funcional para o modelo “Teste”, que contem os dados nome, cpf e idade. Sendo um teste para criação e atualização de dados. Este exemplo deve ser usado para criar os testes para os modelos pedidos a seguir. (Todos os exemplos estão na aplicação do git na pasta: *tests/functional/testeExemple*).

<?php  
class TesteCest  
{  
 // Create  
 public function TesteCreate(FunctionalTester $I)  
 {  
 $I->expectTo('Verify exception for form');  
 $I->amOnRoute('teste/create');  
 $I->submitForm('form',[  
 'Teste[nome]' => 'João',  
 'Teste[cpf]' => '00011122290',  
 'Teste[idade]' => '21',  
 ]);  
 $I->seeRecord('app\models\Teste', [  
 'nome' => 'João',  
 ]);  
 $I->seeRecord('app\models\Teste', [  
 'cpf' => '00011122290',  
 ]);  
 $I->seeRecord('app\models\Teste', [  
 'idade' => '21',  
 ]);  
 }  
  
 // Update  
 public function testeUpdate(FunctionalTester $I)  
 {  
 $I->expectTo("Verify exception for Update");  
 $model = $I->grabRecord('app\models\Teste', array('nome' => 'João'));  
 $I->amOnRoute('teste/update',['id' => $model->id]);  
 $I->submitForm('form',[  
 'Teste[nome]' => 'Jorge',  
 'Teste[cpf]' => $model->cpf,  
 'Teste[idade]' => $model->idade,  
 ]);  
 $I->seeRecord('app\models\teste', [  
 'id' => $model->id,  
 'nome' => 'Jorge',  
 ]);  
 }  
}

O próximo exemplo e o delete usado apenas após executar os demais testes para apagar os dados criados. Onde é feito uma pesquisa de um dados já cadastrado no banco e acessado a rota de deleção, passando o id do resultado.

<?php  
class TesteDeleteCest  
{  
 // Delete  
 public function testeDelete(FunctionalTester $I)  
 {  
 $I->wantTo('Verify exception for Delete');  
 $model = $I->grabRecord('app\models\Teste', array('nome' => 'Jorge'));  
 $id = $model->id;  
 $I-> amOnRoute ('/Teste/delete', ['id' => $id]);  
 $I->dontSeeRecord('app\models\Teste', [  
 'id' => $id,  
 'nome' => $model->nome,  
 ]);  
 }  
}

Com base nestes exemplos, deve ser criado testes automáticos para os modelos pessoa e imóvel de forma manual, sendo teste. Após isso deve-se executar os testes usando o terminal, execute: docker exec -it basic-app-1 vendor/bin/codecept run tests/functional/<Pasta>/testeCest.php --steps. Os testes devem ser executados na seguinte ordem dos modelos: pessoa, imóvel. Pois é necessário criar uma pessoa para vincular a um imóvel.

A execução de testes retorna resultados como *PASSED*,*FAIL* e *ERRO*. No caso de *PASSED* não foi encontrado nenhum problema para aquele tipo de teste executado, em *FAIL* ocorreu algum problema que deve ser corrigido na aplicação ou na escrita do teste, em *ERRO* o teste teve algum problema ao ser executado, que pode ser um problema de implementação do teste.

Após isso executar os testes de delete na ordem: imóvel, pessoa. Todos os testes devem retorna PASSED devendo verificar na aplicação se todos os dados foram deletados corretamente, assim concluindo o experimento. Em caso de erros em testes verifique os retornos de FAIL gerados na pasta *tests/\_output*. Sendo arquivos HTML contendo informações sobre o que gerou o problema.

Após encerrar a criação dos testes funcionais automáticos, deverá **Finalizar a Cronometragem** e anotar no arquivo b*asic/tests/acceptance/<seunome>/TempoGasto.txt* o tempo utilizado para a atividade acima (Exemplo: “Tempo gasto para documentação parte 1: 10:51 minutos”).