Ruby

BD

- Usamos SQLite em ambiente de desenvolvimento
- Em ambiente de produção usamos o que está disponível (ex: MySQL)
- Há um arquivo com o resultado do BD e as migrations (schema.rb)
- Migrations dizem como foram as alterações do BD.
- As migrations normalmente permitem o rollback
- Muito úteis as migrations, mas tem como não usar de forma simples
- $ rails generate migration AddPartNumberToProducts

Frameworks

- RAILS é o muito mais famoso;
- Há outros, como o Sinatra;
- Ruby é bacana, mas muito mais com o Rails;

Simples

- Respeito a existência de alguns patterns, mas eles são necessários em Ruby?
- O que vale mais? Funcionar ou bloquear 100% algo errado. Não é mais simples fazer certo?
- O que vc acha mais simples por exemplo? Rspec ou PHPUnit?
- Quando precisei pensar em um container de injeção de dependência em Ruby? Nada contra o conceito. Nunca!

Rotas

- Dizem que URL deve ser acessada, o verbo HTTP (GET, por exemplo) e que controller e método
- No CakePHP, por exemplo, isto é feito por convenção
- Simples

Rails

- Completo;
- Permite até a execução de testes;
- Tem até um servidor web embutido;
- Simples
- Copiado (ex frameworks PHP, como o CakePHP)
- Na versão 5, tem o modo API, mais enxuto

Gems

- Quando empacotados um conjunto de arquivos para realizar algo, como uma biblioteca para gráficos, devemos fazer isto na forma de uma gem;
- https://rubygems.org ;
- Rails é uma gem .

Falemos de Ruby

- Totalmente OO;
- Permite tipos como string, booleana e float
- Cada tipo tem seus métodos, como teste.upcase (se teste for uma string)
- Há convenções como o “!”, que sinaliza perigo, usado por exemplo para alterar a própria variável
- Se usado um método que não percence à classe, obviamente dá erro

rails c

- rails console;
- Um irb com Rails;
- Muito poderoso;
- Ajuda testes “na mão”

Testes

- Rspec é o padrão para testes unitários;
- Ex
- describe User do
- it "é inválido caso já exista um e-mail igual" do
- user = User.create(firstname: 'Steve',
- lastname: 'Harris', email: 'contato@ironmaiden.com')
- user = User.new(firstname: 'Bruce',
- lastname: 'Dickinson', email: 'contato@ironmaiden.com')
-
- user.valid?
- expect(user.errors[:email]).to include('has already been taken')
- end
- end

Rspec

- Usa o diretório “tests”
- Existe o Cucumber também, por exemplo
- Retorna o resultado no console
- Padrão Rails
- Não precisa fornecer uma cobertura 100%
- Fácil
- Pode não fazer sentido para determinados tipos de teste

Cucumber

- O Cucumber permite testes de sistema:
- Mais complexo
- Nem sempre usado pela complexidade

Geral

- Rails é tão bom que é um framework copiado até em outras linguagens, como PHP (CakePHP por exemplo)
- Mais completo, porém mais pesado quer o Sinatra;
- Usa Convention Over Configuration
- Funcionou perfeitamente no trabalho, por exemplo em APIs
- No Rails 5 há como usar o modo API, mais enxuto
- Simples

Características

- Respeita boas práticas, mas evita teorias pesadas;
- Já usei em PHP um container de injeção de dependência, mas em Ruby não, sem desrespeitar nada que eu saiba; 100% testável;
- Boa parte das coisas que precisa já existem como gems;
- Mundo complexo: só tenho que injetar dependências ou usar um container;
- Mundo simples: Ruby

fcl-rails-daemon

- Gem gratuita gerada na FCL;
- Pelo Washington;
- Funciona super bem
- Ao contrário do cron, tasks não encavalam
- Usamos em projetos de APIs para requisições constantes que não devem depender de humanos

Rails (outros)

- Há como fazer um scaffold rápido e bem básico (nunca usei como view final);
- Exemplo de controller:
- class CheckoutController < ApplicationController
- def create
- checkout = Checkout.process(checkout_params)
- respond_with javascript, location: -> { order_path(checkout.order_id) }
- end
-
- end

Exemplo:

$ rails new app

$ cd app

$ rake generate scaffold new custo direção:string, gasto: varchar(100), valor number(7,2), tipo varchar(30), classificacao varchar(30) observações text;

MVC padrão criado para custo

Rails (outros):

- O MVC e suas responsabilidades é o mesmo que em outros frameworks;
- Logo o model cadastra no BD e valida, por exemplo;
- Se for muito simples, como o exemplo abaixo, as coisas podem ser feitas por herança
- class Article < ApplicationRecord
- validates :title, presence: true,
- length: { minimum: 5 }
- end
- A rota para o controller e método é obrigatória

Exemplo:

```
Agenda::Application.routes.draw do
  get 'contatos', controller: 'contatos', action: 'index' → lista os contatos
  get 'contatos/:id', controller: 'contatos', action: 'show', as: 'contato' → recupera o contato com o identificador passado
  put 'contatos/:id', controller: 'contatos', action: 'update' → atualiza os dados do contato com as informações passadas
  post 'contatos', controller: 'contatos', action: 'create' → cria um contato com os dados passados
  delete 'contatos/:id', controller: 'contatos', action: 'destroy' → exclui o contato pelo identificador passado
end
```

-não tem os comentários, a não ser que estejam com # antes

Exemplo:

```
- class AddDetailsToProducts < ActiveRecord::Migration[5.0]
-   def change
-     add_column :products, :part_number, :string
-     add_column :products, :price, :decimal
-   end
- end
-
- Um projeto tem várias migrations
```

Scaffold

- Gera o MVC
- Simples
- Exemplo:
- $ rails generate scaffold Time
- Caso não haja ambiguidade, podemos usar comandos mais enxutos (ex: g=generate)

Servidor web

- Webrick
- Simples
- Exemplo: rails s 0.0.0.0 -p 8080

Rails console

- Um IRB com Rails;
- Linha de comando;
- Permite testar lógicas rapidamente

Muito simples

- Por onde entro (controller e método): rota
- Lá faço a responsabilidade do controller
- O Model salva e valida por exemplo
- A view gera o HTML, JavaScript e CSS

Ou é melhor?

- Estudar 1.000 teorias por 1 ano para REALMENTE ficar bom em apenas 1 framework, como o Zend (no flames);
- Bora, rápido, está pronto já?
- O básico do sistema pode ser feito em horas
- CakePHP, Rails x Zend
- Já trabalhei em uma campanha de incentivos gigante SEM PROBLEMAS cujo software foi desenvolvido sob o CakePHP, inclusive com carrinho de compras;
- Não falei qual é o mais leve e sim que Rails e CakePHP são utilizáveis
- Tenho prática em um sistema grande com Zend 1, mas eu tenho...

Zend x Rails

- Os 2 são MVC, mas o primeiro é muito complexo
- Nem a mãe do Andy deve achar o Zend simples…
- Temos um sistema administrativo relativamente simples com Zend 2 persistindo em uma API Rest;
- De tão complexo que ficou, queremos migrar
- Parte já está migrada, já que resolvemos no front com Angular
- Nada contra o desenvolvedor
- Temos APIs mais complexas com Rails, mas o resultado final é mais simples
- Mas esse pessoal de Rails tem o mesmo papinho… você é o próximo!