Aplicações Web com Ruby On Rails

Com.pensar 2016

PUC Minas

Poços de Caldas

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

3 de maio de 2016

Agenda

PUC Minas
Poços de Caldas

Aplicação Web (1)

PUC Minas
Poços de Caldas

- Executada pelos usuários via um endereço de um servidor web na rede
- Utiliza um navegador (em inglês: *browser*) para iniciar sua execução
- Consiste de uma coleção de *scripts* no cliente e no servidor, páginas HTML, folhas de estilos e etc.
  - outros recursos que podem estar espalhados por vários servidores.
- Exemplos: webmail, lojas virtuais, homebanking, wikis, blogs e etc.

Aplicação Web (2)

PUC Minas
Poços de Caldas

- Há um pouco mais do que isso:
  - Rede de Computadores:
    - a Internet, um sistema global de redes de computadores interconectadas.
    - utiliza o conjunto de protocolos TCP/IP.
  - Web (World Wide Web):
    - um sistema de documentos (em inglês: *web pages*) vinculados que são acessados através da Internet via protocolo HTTP.
    - Web pages contêm documentos hypermedia: textos, gráficos, imagens, vídeos e outros recursos multimídia, juntamente com *hiperlinks* para outras páginas
    - Hiperlinks formam a estrutura básica da Web.
    - A estrutura da Web é a que a torna útil e de valor.
- Vantagens:
  - Conveniência pela utilização um web browser como cliente.
  - Compatibilidade inerente entre plataformas.

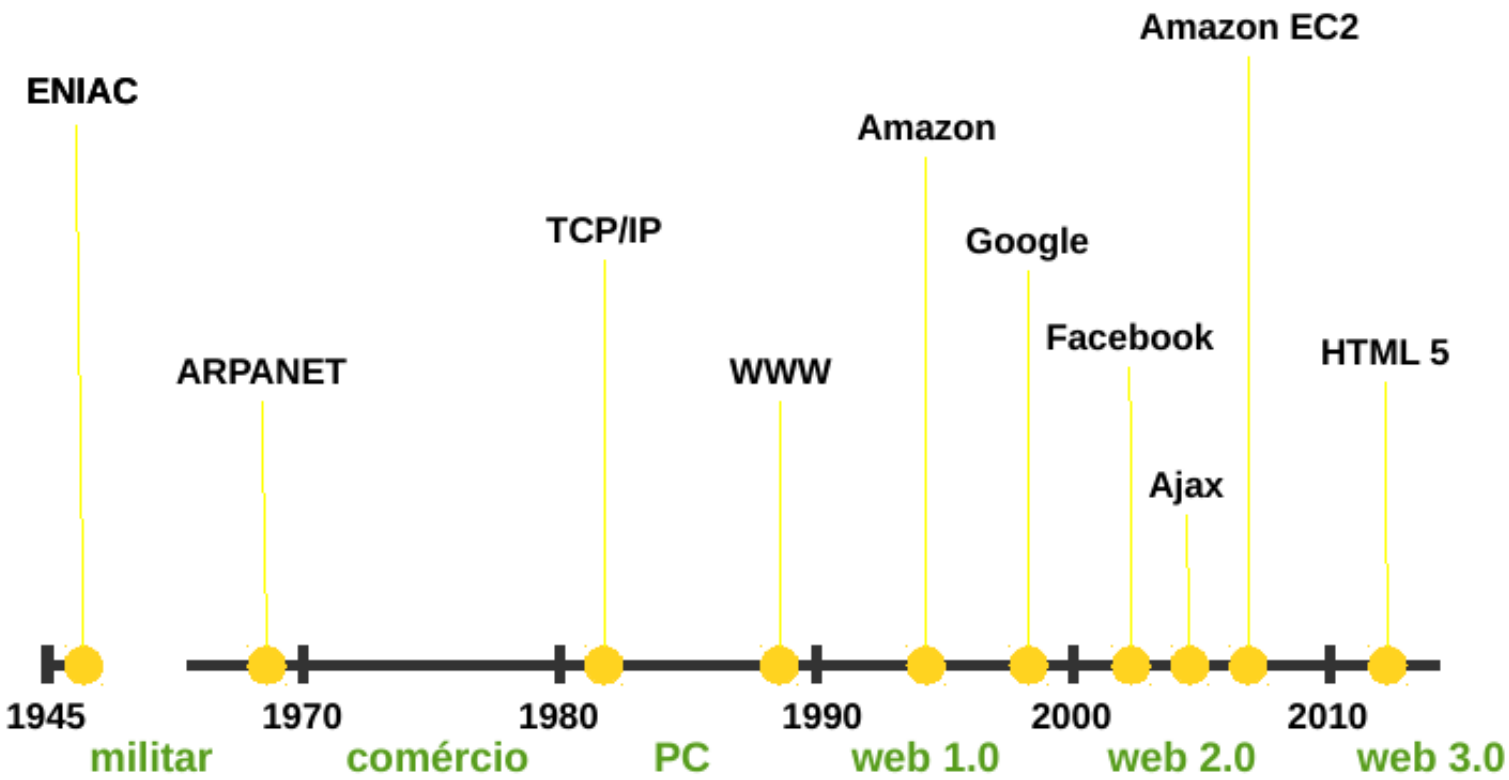
Aplicação Web (3)

PUC Minas
Poços de Caldas

- Habilidade de atualizar e manter as aplicações web sem instalação e distribuição de software em vários clientes em potencial.
- Redução dos custos de TI.

- Desvantagens:
  - Interfaces com usuário ainda não são tão boas quanto as das aplicações tradicionais.
  - Maior risco de comprometimento da privacidade e segurança dos dados.
  - Mais difícil de desenvolver e depurar do que uma aplicação tradicional, pois existem mais partes a se considerar.

Histórico


ENIAC
ARPANET
TCP/IP
WWW
Amazon
Google
Facebook
Ajax
Amazon EC2
HTML 5
1945
1970
1980
1990
2000
2010
militar
comércio
PC
web 1.0
web 2.0
web 3.0

Web 1.0, 2.0 e 3.0

PUC Minas
Poços de Caldas

- Web 1.0 : páginas estáticas e primeiros modelos de negócios.
- Web 2.0 : interactividade(Ajax), redes sociais e comércio eletrônico.
- Web 3.0 : 'Web Inteligente', interpretação da informação auxiliada por máquina
  - exemplo: sistemas de recomendação.
- Base tecnológica da Web 2.0 e 3.0.
  - javascript, xml, json(ajax).
  - interoperabilidade via Web Services.
  - infraestrutura via modelos de computação em nuvem (IAAS, PAAS e SAAS)
  - aplicações móveis

Modelos de Computação em Nuvem (1)

PUC Minas
Poços de Caldas

- IAAS (Infraestructure As A Service) : fornece a insfraestrutura computacional física ou máquinas virtuais e outros recursos discos, firewalls, endereços IP e etc.
  - exemplos: Amazon EC2, Windows Azure, Google Compute Engine.
- PAAS (Platform as a Service) : fornece plataformas computacionais que tipicamente incluem sistemas operacionais, ambientes para execução de programas, bancos de dados, servidores web e etc.
  - exemplos: AWS Elastic Beanstalk, Windows Azure, Heroku e Google App Engine
- SAAS (Software as a Service) : fornece acesso sob demanda às aplicações de software, sem que o usuário tem que se preocupar com sua instalação, configuração e execução.
  - exemplos: Google Apps e Microsoft 365.

Arquiteturas de Aplicações Web (1)

PUC Minas
Poços de Caldas

- As aplicações web modernas envolvem uma quantidade significativa de complexidade.
  - especialmente no lado do servidor.
- Uma típica aplicação web envolve inúmeros protocolos, linguagens de programação e tecnologias que compõem a pilha de tecnologia web.
- Desenvolver, manter e ampliar uma aplicação web complexa é difícil.
  - mas, construindo-o usando uma base de princípios de sólidos de projeto pode-se simplificar cada uma dessas tarefas.
- Engenheiros de software usam abstrações para lidar com este tipo de complexidade.
  - *Design patterns* fornecem abstrações úteis para sistemas orientados a objetos.

Design Patterns (1)

PUC Minas
Poços de Caldas

Definição (Design Patterns)

Um padrão de projeto é uma descrição da colaboração de objetos que interagem para resolver um problema de software em geral dentro de um contexto particular.

- Um design pattern é um modelo abstrato que pode ser aplicado recorrentemente.
- A idéia é aplicar padrões de projeto, a fim de resolver problemas específicos que ocorrem durante a construção de sistemas reais.
- Os padrões de projeto fornecem uma maneira de comunicar as soluções em um projeto, ou seja, é a terminologia que engenheiros de software usam para falar sobre projetos.

Modelo Cliente-Servidor (1)

PUC Minas
Poços de Caldas

- A arquitetura cliente-servidor é a arquitetura mais básica para descrever a cooperação entre os componentes de uma aplicação web.
- A arquitetura cliente-servidor pode ser subdividia em:
  - servidor que "escuta"por requisições e fornece os serviços ou recursos de acordo com cada uma.
  - cliente que estabelece a conexão com o servidor para requisitar serviços ou recursos.
- Existe um protocolo request/response associado com qualquer arquitetura cliente-servidor.

Modelo Cliente-Servidor (2)

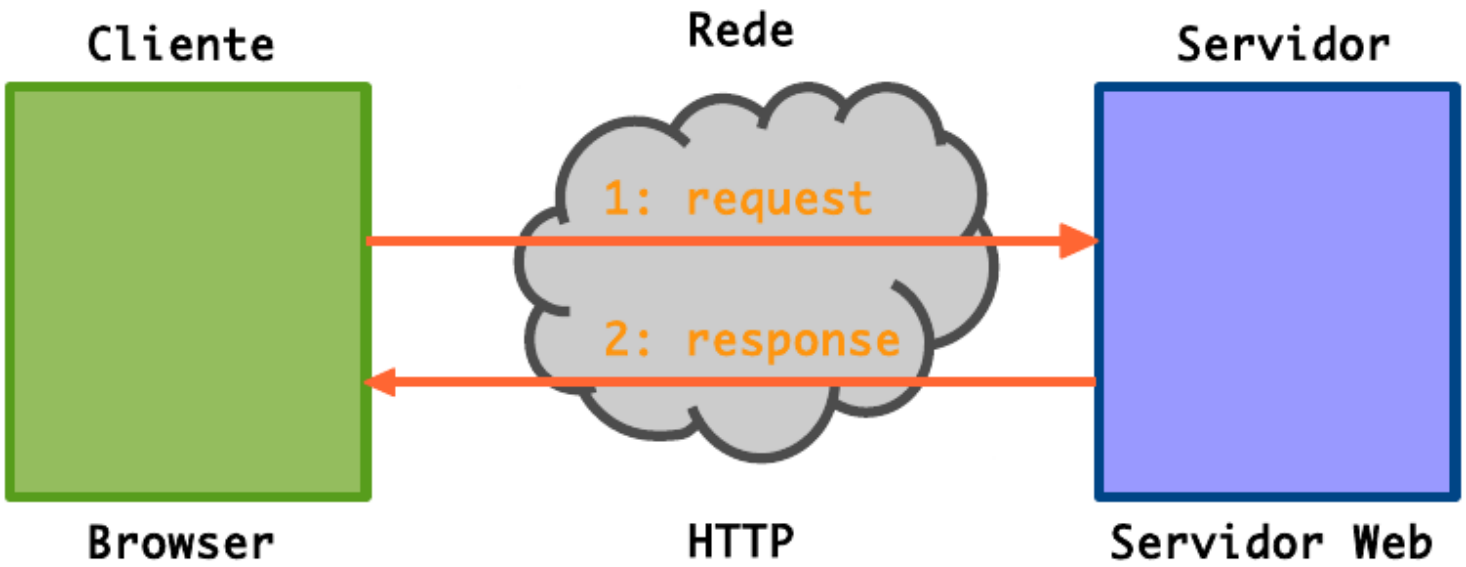


Figura: Arquitetura cliente servidor.

Modelo Cliente-Servidor (3)

PUC Minas
Poços de Caldas

- É sem dúvida é o padrão de projeto de arquitetura mais conhecido
- O ponto chave de uma arquitetura cliente-servidor é distribuir os componentes de uma aplicação entre o cliente o servidor de alguma forma.
  - o servidor realiza as tarefas, consultas e transações
  - o cliente fica com uma responsabilidade menor: a de receber informações
- A fim de construir aplicações web complexas, vários design patterns ajudam a organizar como peças são dispostas dentro da arquitetura cliente-servidor.

Arquitetura N-Tier (1)

PUC Minas
Poços de Caldas

Definição (Arquitetura N-Tier)

A arquitetura n-tier é um *design pattern* muito útil que estrutura o modelo cliente-servidor.

- Este padrão de projeto é baseado no conceito de quebrar um sistema em partes diferentes ou camadas que podem ser separados fisicamente:
  - cada camada é responsável por fornecer uma funcionalidade específica ou coesa.
  - uma camada apenas interage com as camadas adjacentes a ela por meio de uma estrutura bem definida por meio de interfaces.

Arquitetura N-Tier (2)

PUC Minas
Poços de Caldas

Exemplos (Arquitetura 2-Tier)

- Servidores de impressão
- Aplicações web antigas:
  - Interface com o usuário (navegador) residia no cliente (thin).
  - Servidor fornecia as páginas estáticas (HTML).
  - Interface entre os dois via *Hypertext Transfer Protocol* (HTTP).

- Camadas adicionais aparecem quando a funcionalidade do aplicativo é ainda mais dividida.
- Quais são as vantagens de um tal projeto?
  - A abstração fornece um meio para gerenciar a complexidade.
  - Camadas podem ser atualizados ou substituídos de forma independente a medida que os requisitos ou tecnologia.

Arquitetura N-Tier (3)

PUC Minas
Poços de Caldas

- a nova só precisa usar as mesmas interfaces que a antiga utilizada.

- Ele fornece um equilíbrio entre inovação e padronização.
- Sistemas tendem a ser muito mais fáceis de construir, manter e atualizar.

Arquitetura 3-Tiers (1)

PUC Minas
Poços de Caldas

- Uma das mais comuns é a arquitetura em 3 camadas:
  - Apresentação
    - a interface com o usuário.
  - Aplicação (lógica)
    - recupera modifica e/ou exclui dados na camada de dados, e envia os resultados do processamento para a camada de apresentação.
  - Camada de dados
    - a fonte dos dados associados ao aplicativo.
- As aplicações web modernas frequentemente são construídas utilizando uma arquitetura em 3 camadas:
  - Apresentação
    - o navegador web do usuário.
  - Aplicação (lógica)

Arquitetura 3-Tiers (2)

PUC Minas
Poços de Caldas

- o servidor web e lógica associada com geração de conteúdo web dinâmico.
- por exemplo, a coleta e formatação do resultados de uma pesquisa.

- Camada de dados
  - um banco de dados.

Agenda

PUC Minas
Poços de Caldas

Ruby on Rails (1)

PUC Minas
Poços de Caldas

- Ruby on Rails (Rails) é framework construído na linguagem Ruby para o desenvolvimento de aplicações web
  - Rails é fornecido em uma gem Ruby (gem é um pacote Ruby)
- Rails fornece uma extenso conjunto de geradores de código e scripts de automação de testes
- Um conjunto de ferramentas adicionais são fornecidos como parte do ecossistema Rails:
  - Rake - utilitário similar ao **make do Unix** para criar e migrar bancos de dados, limpar sessões de uma Web app
  - WEBrick - servidor web de desenvolvimento para execução de aplicações Rails
  - SQLite - um servidor de banco de dados simples pré-instalado como o Rails

Ruby on Rails (2)

PUC Minas
Poços de Caldas

- Rack Middleware - interface padronizado para interação entre um servidor web e uma Web App

- Algumas empresas que utilizam Rails: Twitter, Hulu, GitHub, Yellow Pages e etc

Filosofia do Rails (1)

PUC Minas
Poços de Caldas

- Ruby on Rails é 100% open-source, disponível por meio da MIT License: ⟨http://opensource.org/licenses/mit-license.php⟩.
- Convenção acima da Configuração (em inglês: *Convention over Configuration* (CoC))
  - se nomeação segue certas convenções, não há necessidade de arquivos de configuração.

Exemplo:

```
FilmesController#show -> filmes_controler.rb
FilmesController#show -> views/filmes/show.html.erb
```

- "Don't Repeat Yourself" (DRY) sugere que escrever que o mesmo código várias vezes é uma coisa ruim

Filosofia do Rails (2)

PUC Minas
Poços de Caldas

- O *Representational State Transfer* (REST) é o melhor padrão para desenvolvimento de aplicações web
  - organiza a sua aplicação em torno de recursos e padrões HTTP (verbs)

Histórico (1)

PUC Minas
Poços de Caldas

- David Heinemeier Hanson derivou o Ruby on Rails a partir do BaseCamp – uma ferramenta de gestão de projetos da empresa 37Signals.
  - a primeira versão de código aberto (em inglês: *open source*)foi liberada em julho de 2004.
  - mas direitos para que outros desenvolvedores colaborassem com o projeto foram liberados em fevereiro de 2005.
- Em agosto de 2006, o Ruby on Rails atingiu um marco importante quando a Apple dicidiu distribuído juntamente com a versão do seu sistema operacional Mac OS X v10.5 "Leopard"
  - nesse mesmo no o Rails começou a ganhar muita atenção da comunidade de desenvolvimento web.
- Rails é utilizado por diversas companhias, como por exemplo:

Histórico (2)

PUC Minas
Poços de Caldas

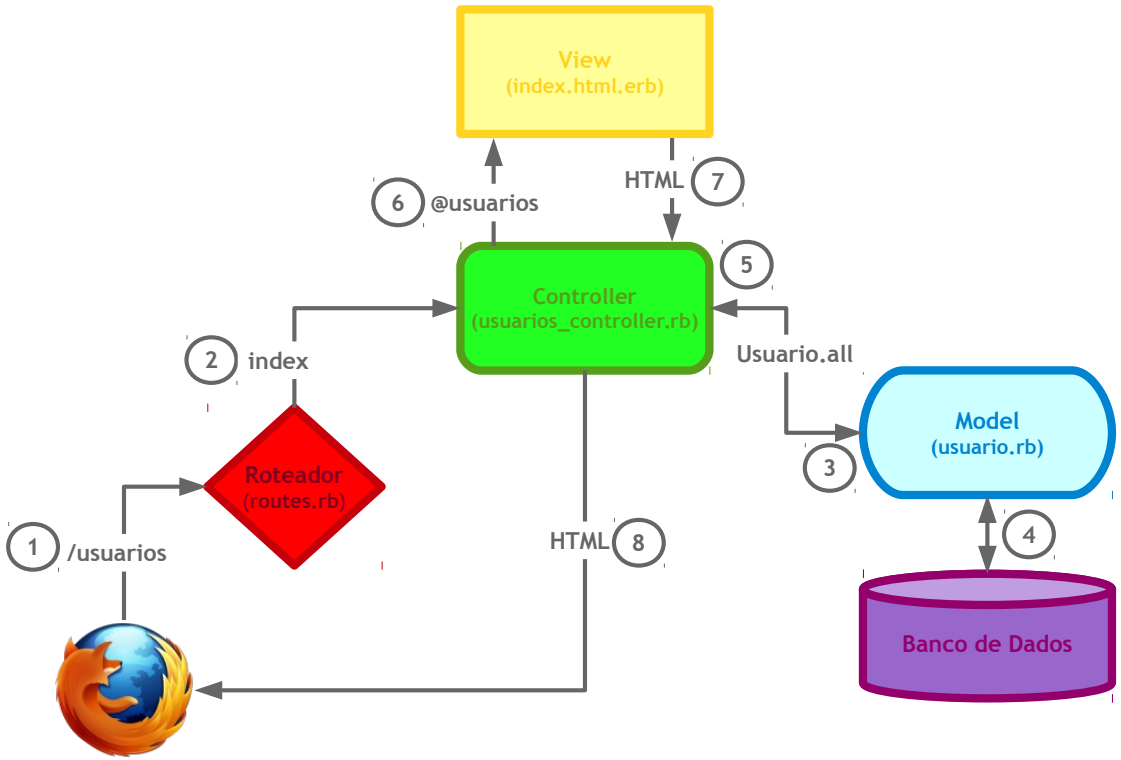
- Airbnb, BaseCamp, Disney, GitHub, Hulu, Kickstarter, Shopify e Twitter.

Versão	**Data**
1.0	13 de dezembro de 2005
1.2	19 de janeiro de 2007
2.0	07 de dezembro de 2007
2.1	01 de junho de 2008
2.2	21 de novembro de 2008
2.3	16 de março de 2009
3.0	29 de agosto de 2010
3.1	31 de agosto de 2011
3.2	20 de janeiro de 2012
4.0	25 de junho de 2013
4.1	08 de abril de 2014

Tabela: Evolução histórica do Ruby on Rails

Model-View-Controller

- O framework Rails é contruído em cima do Design Pattern Model View Controller(MVC):



Hora de Colocar a Mão na Massa

PUC Minas
Poços de Caldas

- Conecte-se na máquina com o usuário `a1550099999` e senha 333333
  1. Inicie uma janela de terminal e digite no prompt:

```
$ rails new my_app
```

  2. Mude para o diretório da aplicao (RAILS.root)

```
$ cd new my_app
```

  3. Execute o servidor web embutido:

```
$ rails s
```

  4. Abra uma janela do navegador e digite:

```
$ http://localhost:3000
```

Estrutura de uma Aplicação Rails (1)

PUC Minas
Poços de Caldas

Arquivo/Pasta	**Descrição**
app	Arquivos contendo os principais códigos da aplicação, incluindo modelos, visões, controladores e auxiliares(*helpers*)
app/assets	Arquivos contendo folhas de estilos (CSS), códigos Javascript e imagens da aplicação
bin	Arquivos ou scripts executáveis
config	Configurações da aplicação
db	Migrações, esquema e outros arquivos relacionados ao banco de dados
doc	Documentação do sistema
lib	Bibliotecas auxiliares
lib/assets	Arquivos contendo folhas de estilos (CSS), códigos Javascript e imagens das bibliotecas

Estrutura de uma Aplicação Rails (2)

PUC Minas
Poços de Caldas

Arquivo/Pasta	Descrição
log	Informações de log
public	Páginas que podem ser acessadas publicamente via navegador, tais como páginas de erros
test	Testes da nossa aplicação
tmp	Arquivos temporários como cache e informações de sessões
vendor	Dependências e bibliotecas de terceiros
vendor/assets	Arquivos contendo folhas de estilos (CSS), códigos Javascript e imagens de terceiros
README.rdoc	Uma breve descrição da aplicação
Rakefile	Tarefas que podem ser executadas pelo comando rake
Gemfile	Pacotes(gems) necessários para a aplicação
Gemfile.lock	Uma lista de gems utilizadas para garantir que todas as cópias da aplicação utilizam as mesmas versões de gems
config.ru	Um arquivo de configuração para o Rack Middleware
.gitignore	Define de arquivos ou padrões de arquivos que deverão ser ignorados pelo Git

Agenda

PUC Minas
Poços de Caldas

Especificação do Blog App (1)

PUC Minas
Poços de Caldas

1. Blog é uma contração de "weblog", um site de discussão ou troca de informações publicado na Web.
2. Existem dois tipos de participantes: o administrador e o usuário
3. O administrador do blog deve ser capaz de entrar novas postagens, tipicamente em ordem cronológica inversa.
4. Os usuários devem ser capazes de visitar o blog e escrever comentários sobre as postagens.
5. O administrador do blog deve ser capaz de modificar e ou remover qualquer postagem ou comentário.
6. Os usuários não devem ser capazes de modificar postagens ou comentários de outros usuários.

Passos Iniciais do Blog App (1)

PUC Minas
Poços de Caldas

1. Inicie uma janela de terminal e digite no prompt:

```
$ cd
$ rails new blog
```

2. Utilize o gerador `scaffold` para criar os componentes MVC para as postagens e os comentários

```
$ rails generate scaffold post \
    title:string body:text
$ rails generate scaffold comment post_id:integer \
 body:text
```

Passos Iniciais do Blog App (2)

PUC Minas
Poços de Caldas

3. Gere as tabelas `post` e `comment` no banco de dados

```
$ rake db:migrate
```

4. Visualize todas as URLs reconhecidas pela sua aplicação digitando:

```
$ rake routes
```

5. Inicie o servidor web embutido:

```
$ rails s
```

6. Abra uma janela do navegador e digite:

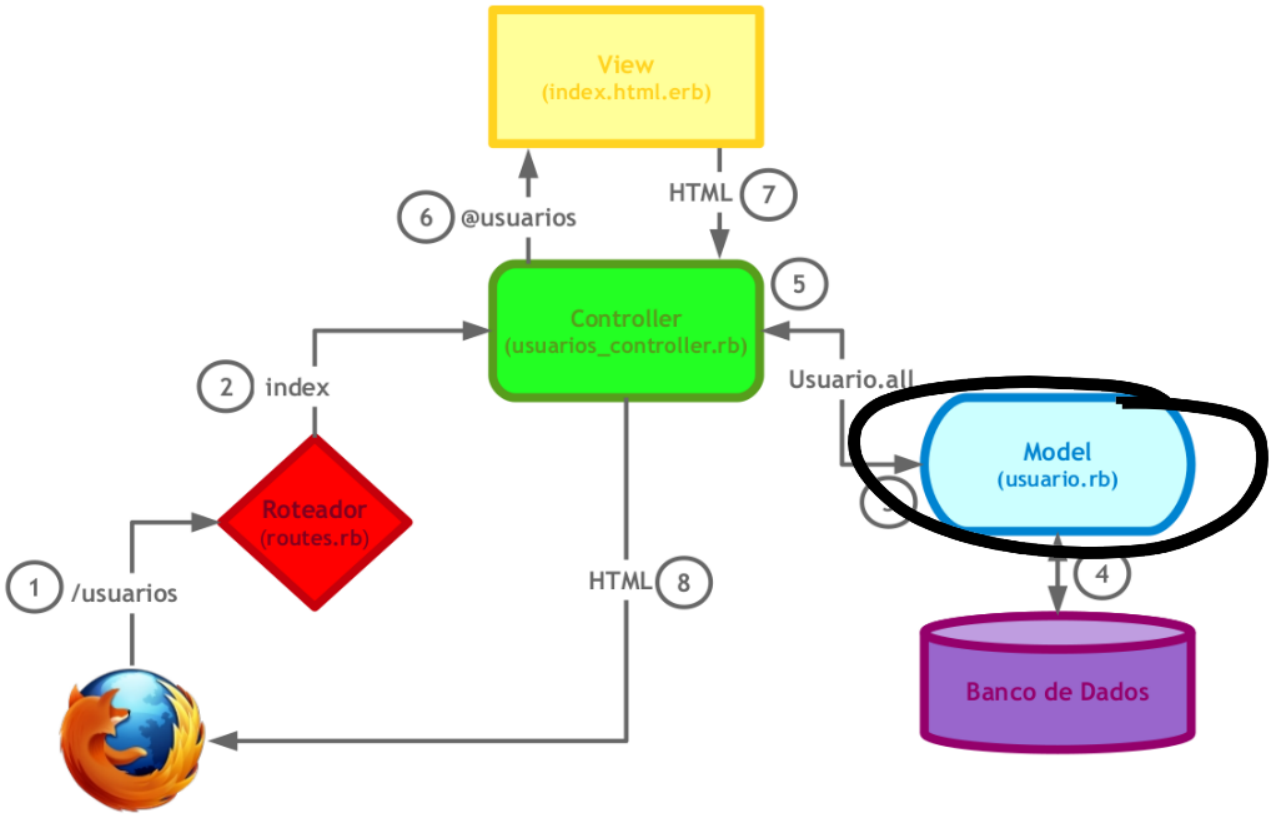
```
$ http://localhost:3000/posts
```

Agenda

PUC Minas
Poços de Caldas

Model Component

- O modelo gerencia os dados, a lógica e as regras de negócios da aplicação.



Banco de Dados Relacionais (1)

PUC Minas
Poços de Caldas

- Um aspecto importante da programação web é a habilidade de coletar, armazenar e recuperar diferentes formas de dados
  - uma das formas mais populares são os bancos de dados relacionais
- Um banco de dados relacional é baseado entidades, denominadas tabelas, no relacionamento, associações, entre elas
- O contêiner fundamental em um banco de dados relacional é denominado de database ou schema
  - podem incluir estruturas de dados, os dados propriamente ditos e permissões de acesso

Banco de Dados Relacionais (2)

PUC Minas
Poços de Caldas

- Os dados são armazenados em tabelas e as tabelas são divididas em linhas e colunas. Por exemplo:

Tabela: comment

id	post_id	body
10	1	Ruby realmente...
11	2	Rails facilita...
13	2	Concordo, ...

Banco de Dados Relacionais (3)

PUC Minas
Poços de Caldas

- Relacionamentos são estabelecidos entre tabelas para que a consistência dos dados seja mantida em qualquer situação e podem ser:
  - 1:1
  - 1:N
  - N:M

Tabela: comment

id	post_id	body
10	1	Ruby realmente...
11	2	Rails facilita...
13	2	Concordo, ...

Tabela: post

id	title	body
1	A Linguagem Ruby	Ruby é legal.
2	O Framework Rais	O Rais facilita...

SQLite (1)

PUC Minas
Poços de Caldas

- O banco de dados que o Rails utiliza em diversos ambientes (desenvolvimento, teste e produção) é especificado em: config/database.yml

```
1 default: &default
2   adapter: sqlite3
3   pool: 5
4   timeout: 5000
5 development:
6   <<: *default
7   database: db/development.sqlite3
8 test:
9   <<: *default
10  database: db/test.sqlite3
11 production:
12  <<: *default
13  database: db/production.sqlite3
```

SQLite (2)

PUC Minas
Poços de Caldas

- Rails usa por padrão o SQLite como gerenciador padrão
  - relacional, embutido, sem servidor, configuração zero, transacional, suporta SQL

ATENÇÃO: SQLite não um banco de dados para produção !

- Banco de dados de produção populares: MySQL e PostgreSQL

Database Console

PUC Minas
Poços de Caldas

- O comando rails db fornece uma console para acesso aos bancos dados MySQL, PostgreSQL e SQLite.

```
$ rails db
SQLite version 3.8.7.1 2014-10-29 13:59:56
Enter ".help" for usage hints.
sqlite> .headers on
sqlite> .mode columns
sqlite> select * from posts;
id          title             body            created_at                  updated_at
----------  ----------------  --------------  --------------------------  --------------------------
5           A Linguagem Ruby  Ruby e legal.  2016-04-30 22:45:20.636363  2016-04-30 22:45:20.636363
sqlite>
```

- Dica: utilize headers on e mode coluns

Hora de Colocar a Mão na Massa (1)

PUC Minas
Poços de Caldas

- Inicialize na pasta da aplicação a console do banco de dados e configure a sua exibição:

```
$ rails db
sqlite> .headers on
sqlite> .mode columns
```

- Exiba os colunas da tabela `posts`:

```
sqlite> .schema posts
```

Hora de Colocar a Mão na Massa (2)

PUC Minas
Poços de Caldas

- Crie um novo post e salve no banco de dados:

```
sqlite> INSERT INTO posts
(title, body, created_at, updated_at)
VALUES ("Com.pensar 2016", "Tem varios cursos",
"2016-05-03 19:50:00", "2016-05-03 19:50:00");
```

- Exiba todos os posts:

```
sqlite> SELECT * FROM posts;
```

- Exiba todos os posts ordenados pelo título (title):

```
sqlite> SELECT * FROM posts ORDER BY title;
```

Hora de Colocar a Mão na Massa (3)

PUC Minas
Poços de Caldas

- Exiba um `post`:

```
sqlite> SELECT * FROM posts LIMIT 1
```

- Exiba o `post` cujo `id` é 2:

```
sqlite> SELECT * FROM posts WHERE id=2;
```

- Atualize o título `post` cujo o `id` é 2:

```
sqlite> UPDATE posts SET title="Novo titulo"
WHERE id=2;
```

- Remova `post` cujo o `id` é 2:

```
sqlite> DELETE FROM posts WHERE id=2;
```

Migrations (1)

PUC Minas
Poços de Caldas

- Como podemos rastrear e desfazer alterações em um banco de dados?
- Não existe uma maneira fácil - manualmente é confuso e propenso a erros.
- Tipicamente, comandos SQL são dados para criar e modificar tabelas em um banco de dados
- Mas se houver a necessidade de trocar o banco de dados "durante o voo"?
  - por exemplo, desenvolve-se em SQLite e implanta-se em MySQL.

SOLUÇÃO: Migrations

Migrations (2)

PUC Minas
Poços de Caldas

- A cada vez que o scaffold é executado na aplicação, o Rails cria um arquivo de migration de banco de dados. Este arquivo é armazenado em db/migrate
- Por exemplo: o arquivo 20160430140114_create_posts.rb

```
1 class CreatePosts < ActiveRecord::Migration
2   def change
3     create_table :posts do |t|
4       t.string :title
5       t.text :body
6
7       t.timestamps null: false
8     end
9   end
10 end
```

Migrations (3)

PUC Minas
Poços de Caldas

- Rails utiliza o comando rake para executar os migrations e fazer as alterações no banco de dados.

```
$ rake db:migrate
```

Object-Relational Mapping (1)

PUC Minas
Poços de Caldas

- Um ORM preenche a lacuna entre banco de dados relacionais e as linguagens de programação orientadas a objetos
- Simplifica bastante a escrita de códigos para acessar o banco de dados.
- Tipicamente, comandos SQL são dados para criar e modificar tabelas em um banco de dados
- No Rails, o Model do MVC utiliza algum framework de ORM

Active Record (1)

PUC Minas
Poços de Caldas

- ActiveRecord é o nome do ORM padrão do Rails?

Onde está código ?
R: Metaprogramação + Convenção

Código 1: app/models/post.rb

```
1 class Post < ActiveRecord::Base
2 end
```

- Para que "mágica" ocorra:
  - o ActiveRecord tem que saber como encontrar o banco de dados (ocorre via config/database.yml)
  - **(Convenção)** existe uma tabela com o nome no plural da subclasse `ActiveRecord::Base`
  - **(Convenção)** espera-se que a tabela tenha uma chave primário denominada id

Object-Relational Mapping (1)

PUC Minas
Poços de Caldas

- Um ORM preenche a lacuna entre banco de dados relacionais e as linguagens de programação orientadas a objetos
- Simplifica bastante a escrita de códigos para acessar o banco de dados.
- Tipicamente, comandos SQL são dados para criar e modificar tabelas em um banco de dados
- No Rails, o Model do MVC utiliza algum framework de ORM

Hora de Colocar a Mão na Massa (1)

PUC Minas
Poços de Caldas

- Inicialize na pasta da aplicação a console do Rails (não a do banco de dados):

```
$ rails c
```

- Exiba os atributos da classe `Post`:

```
irb(main):004:0> Post.column_names
```

- Crie um novo `post` e salve no banco de dados:

```
irb(main):005:0> p1 = Post.new
irb(main):006:0> p1.title="Temperatura em Pocos"
irb(main):007:0> p1.body="Esta muito frio..."
irb(main):008:0> p1.save
```

Hora de Colocar a Mão na Massa (2)

PUC Minas
Poços de Caldas

- Exiba todos os `posts`:

```
irb(main):007:0> Post.all
```

- Exiba todos os `posts` ordenados pelo título (title):

```
irb(main):007:0> Post.all.order(title: :asc)
```

- Exiba um `post`:

```
irb(main):007:0> Post.first
```

- Exiba o `post` cujo `id` é 2:

```
irb(main):007:0> Post.find_by(id: 2)
```

Hora de Colocar a Mão na Massa (3)

PUC Minas
Poços de Caldas

- Atualize o título do primeiro `post`:

```
irb(main):007:0> p1=Post.first
irb(main):008:0> p1.update(title: "um novo titulo")
```

- Remova do primeiro `post`:

```
irb(main):007:0> p1=Post.first
irb(main):008:0> p1.destroy
```

Validação em Aplicações Web

PUC Minas
Poços de Caldas

- Validação de Dados é o processo para garantir que a aplicação web operem corretamente. Exemplo:
  - garantir a validação do e-mail, número do telefone e etc
  - garantir que as "regras de negócios"sejam validadas
- A vulnerabilidade mais comum em aplicação web é a injeção SQL

Client Side

PUC Minas
Poços de Caldas

- Envolve a verificação de que os formulários HTML sejam preechidos corretamente
  - JavaScript tem sido tradicionalmente utilizado.
  - HTML5 possui "input type"específicos para checagem.
  - Funciona melhor quando combinada com validações do lado do servidor.

Server Side

PUC Minas
Poços de Caldas

- A validação é feita após a submissão do formulário HTML
  - banco de dados(stored procedure) - dependente do banco de dados
  - no controlador - veremos mais tarde que não se pode colocar muita lógica no controlador (controladores magros)
  - no modelo - boa maneira de garantir que dados válidos sejam armazenados no banco de dados (database agnostic)
  - Funciona melhor quando combinada com validações do lado do servidor.

Validação em Rails (1)

PUC Minas
Poços de Caldas

- Objetos em um sistema OO como tendo um ciclo de vida
  - eles são criaddos, atualizados mais tarde e também destruidos.
- Objetos ActiveRecord têm métodos que podem ser chamados, a fim de assegurar a sua integridade nas várias fases do seu ciclo de vida.
  - garantir que todos os atributos são válidos antes de salvá-lo no banco de dados
- Callbacks são métodos que são invocados em um ponto do ciclo de vida dos objetos ActiveRecord
  - eles são "ganchos"para gatilhos para acionar uma lógica quando houver alterações de seus objetos

Validação em Rails

PUC Minas
Poços de Caldas

- Validations são tipo de callbacks que podem ser utilizados para garantir a validade do dado em um banco de dados
- Validação são definidos nos modelos. Exemplo:

```
1 class Person < ActiveRecord::Base
2     validates_presence_of :name
3     validates_numeracality_of :age, :only_integer => true
4     validates_confirmation_of :email
5     validates_length_of :password, :in => 8..20
6 end
```

Hora de Colocar a Mão na Massa

PUC Minas
Poços de Caldas

- Modifique o arquivo `app/models/post.rb` para exigir que o usuário digite o título e o texto do blog:

```
1 class Post < ActiveRecord::Base
2   validates_presence_of :title, :body
3 end
```

- Modifique o arquivo `app/models/comment.rb` para exigir que o usuário digite texto do comentário blog:

```
1 class Post < ActiveRecord::Base
2     validates_presence_of :body
3 end
```

- Inicie o servidor web embutido e teste se a validação está funcionando

Associações em Rails (1)

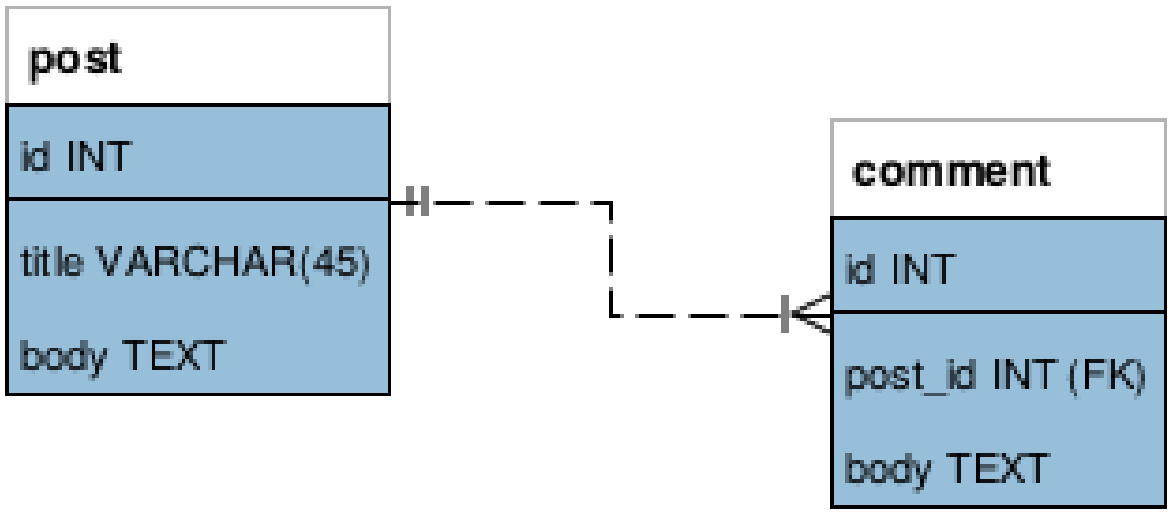
PUC Minas
Poços de Caldas

- O gerador `scaffold` utiliza por padrão o ActiveRecord. Isto significa:
  - Tabelas para postagens e comentários foram criadas quando executamos as migrações
  - Um conexão com o banco de dados é estabelecida
  - O ORM é configurado para as postagens e comentátios foi criado - o "M" do MVC.
- No entanto, uma coisa está faltando:
  - tem-se que assegurar que qualquer comentários sejam associados às suas postagens
- Para tornar os modelos em Rails totalmente funcionais precisamos adicionar associações:
  - cada postagem precisa saber os comentários associado a ele
  - cada comentário precisa saber qual é a postagem ele pertence

Associações em Rails (2)

PUC Minas
Poços de Caldas

- Há uma relação muitos-para-um entre comentários e postagens uma:



- O ActiveRecord contém um conjunto de métodos de classe para vinculação de objetos por meio de chaves estrangeiras

Associações em Rails (3)

PUC Minas
Poços de Caldas

- Para habilitar isto, deve-se declarar as associações dentro dos modelos usando:

Associação	**Modelo Pai**	**Modelo Filho**
Um-para-um	`has_one`	`belongs_to`
Muitos-para-um	`has_many`	`belongs_to`
Muitos-para-muitos	`has_and_belongs_to_many`	*na tabela junção

Hora de Colocar a Mão na Massa (1)

PUC Minas
Poços de Caldas

- Modifique o arquivo `app/models/post.rb` para associar o post aos seus comentário:

```
1 class Post < ActiveRecord::Base
2   validates_presence_of :title, :body
3   has_many :comments
4 end
```

- Modifique o arquivo `app/models/comment.rb` para associar o comentário ao seu post:

```
1 class Comment < ActiveRecord::Base
2   validates_presence_of :body
3   belongs_to :post
4 end
```

Hora de Colocar a Mão na Massa (2)

PUC Minas
Poços de Caldas

- Crie um novo `post` e salve no banco de dados:

```
irb(main):005:0> p1 = Post.new
irb(main):006:0> p1.title="Associacao"
irb(main):007:0> p1.body="Vinculando com o comentario"
irb(main):008:0> p1.save
```

- Crie um novo `comment` e o vincule a um `post`:

```
irb(main):005:0> c1 = Comment.new
irb(main):006:0> c1.body="Comentario vinculado"
irb(main):007:0> c1.save
irb(main):008:0> p1.comments << c1
```

Hora de Colocar a Mão na Massa (3)

PUC Minas
Poços de Caldas

- Consulte os comentários do `post` `p1`:

```
irb(main):005:0> p1.comments.all
```

- Consulte os comentários 2 do `post` `p1`:

```
irb(main):005:0> p1.comments.where(id: 2)
```

- Consulte o `post` do comentário `c1`:

```
irb(main):005:0> c1.post
```

Agenda

PUC Minas
Poços de Caldas

Action Controller

PUC Minas
Poços de Caldas

- Um Action Controller é classe Ruby contendo uma ou mais ações
- Cada ação é responsável pela resposta a uma requisição
- Quando uma ação é concluída a visão de mesmo nome é renderizada
- Uma ação deve estar mapeada no arquivo routes.rb que é gerado pelo scaffold:

```
1 Rails.application.routes.draw do
2 resource :posts
3 resource :comments
4 end
```

Representational State Transfer

PUC Minas
Poços de Caldas

- Rails utilza Representational State Transfer(REST) para mapear os recursos (resources) de uma aplicação:
  - List todos os recursos disponíveis
  - Show um recurso específico
  - Destroy um recurso existente
  - Provide a way to create um novo recurso
  - Create um novo recurso
  - Provide a way to update um recurso existente
  - Update um recurso existente

Ação: Index (1)

PUC Minas
Poços de Caldas

- Ação que recupera todas as postagens do blog
- (Implicitamente) procura pelo template index.html.erb para renderizar a resposta

Código 2: controllers/posts_controller.rb

```
1 Class PostsController < ApplicationController
2
3 # GET /posts
4 # GET /posts.json
5 def index
6 @posts = Post.all
7 end
```

Ação: Index (2)

PUC Minas
Poços de Caldas

- index.html.erb:

Código 3: views/posts/index.html.erb

```
1 ...
2 <tbody>
3 <% @posts.each do |post| %>
4 <tr>
5 <td><%= post.title %></td>
6 <td><%= post.body %></td>
7 <td><%= link_to 'Show', post %></td>
8 <td><%= link_to 'Edit', edit_post_path(post) %></td>
9 <td><%= link_to 'Destroy', post, method: :delete, ....
10 </tr>
11 <% end %>
12 </tbody>
13 ....
```

Ação: Show (1)

PUC Minas
Poços de Caldas

- Recupera uma postagem específica no parâmetro id passado como parte da URL
- (Implicitamente) procura pelo show.html.erb para renderizar a resposta

Código 4: controllers/posts_controller.rb

```
1 Class PostsController < ApplicationController
2 before_action :set_post, only: [:show, :edit, :update, :destroy]
3
4 #GET /posts/1
5 #GET /posts/1.json
6 def show
7 end
8
9 private
10 def set_post
```

Ação: Show (2)

PUC Minas
Poços de Caldas

```
11 @post = Post.find(params[:id])
12 end
```

Ação: Show (3)

PUC Minas
Poços de Caldas

- show.html.erb:

Código 5: views/posts/show.html.erb

```
1 <p>
2 <strong>Title:</strong>
3 <%= @post.title %>
4 </p>
5 <p>
6 <strong>Body:</strong>
7 <%= @post.body %>
8 </p>
9 <%= link_to 'Edit', edit_post_path(@post) %> |
10 <%= link_to 'Back', posts_path %> end
```

respond_to (1)

PUC Minas
Poços de Caldas

- Rails helper que especifica como responder a uma requisição baseado no formato da requisição

Código 6: controllers/posts_controller.rb

```
1 Class PostsController < ApplicationController
2
3   # POST /posts
4   # POST /posts.json
5   def create
6     @post = Post.new(post_params)
7
8     respond_to do |format|
9       if @post.save
10        format.html { redirect_to @post, notice: 'Post was successfully created.'
11        format.json { render :show, status: :created, location: @post }
12      else
```

respond_to (2)

PUC Minas
Poços de Caldas

```
13        format.html { render :new }
14        format.json { render json: @post.errors, status: :unprocessable_entity }
15      end
16    end
17  end
```

redirect_to

PUC Minas
Poços de Caldas

- Ao invés de renderizar um template - envia uma resposta ao navegador: "go here"
- Usualmente utiliza uma URL completa como um parâmetro
  - pode ser tanto uma URL ou uma rota nomeada
- Se o parâmetro é um objeto - Rails tentara gerar uma URL para aquele objeto

Ação: Destroy (1)

PUC Minas
Poços de Caldas

- Remove uma postagem específica pelo parâmetro id passado como parte da URL

Código 7: posts_controller.rb

```
1 Class PostsController < ApplicationController
2   before_action :set_post, only: [:show, :edit, :update, :destroy]
3
4   #DELETE /posts/1
5   #DELETE /posts/1.json
6   def destroy
7     @post.destroy
8     respond_to do |format|
9       format.html {redirect_to posts_url, notice: 'Post was ....'}
10      format.json {head :no_content}
11    end
12  end
13
```

Ação: Destroy (2)

PUC Minas
Poços de Caldas

```
14  private
15  # Use callbacks to share common setup or constraints between actions.
16  def set_post
17    @post = Post.find(params[:id])
18  end
```

Ação: New (1)

PUC Minas
Poços de Caldas

- Cria um novo objeto post(vazio)
- (Implicitamente) procura pelo new.html.erb para renderizar a resposta

Código 8: posts_controller.rb

```
1 Class PostsController < ApplicationController
2
3   #GET /posts/new
4   def new
5     @post = Post.new
6   end
```

Ação: New (2)

PUC Minas
Poços de Caldas

- new.html.erb:

Código 9: views/posts/new.html.erbb

```
1 <h1>New Post</h1>
2 <%= render 'form' %>
3 <%= link_to 'Back', posts_path %>
```

Ação: Create (1)

PUC Minas
Poços de Caldas

- Cria um novo objeto post como os parâmetros que foram passados pelo formulário new
- Tenta salvar o objeto no banco de dados
- Se sucesso, redireciona para o template show
- Se insucesso, renderiza o template new novamente

Ação: Create (2)

PUC Minas
Poços de Caldas

Código 10: controllers/posts_controller.rb

```
1 Class PostsController < ApplicationController
2   def create
3     @post = Post.new(post_params)
4
5     respond_to do |format|
6       if @post.save
7         format.html { redirect_to @post, notice: 'Post was successfully create
8         format.json { render :show, status: :created, location: @post }
9       else
10        format.html { render :new }
11        format.json { render json: @post.errors, status: :unprocessable_entity
12      end
13    end
14  end
15
16  private
17  # Never trust parameters from the scary internet,
18 # only allow the white list through.
```

Ação: Create (3)

PUC Minas
Poços de Caldas

```
19  def post_params
20    params.require(:post).permite(:title, :content)
21  end
```

- a linha 20 implementa strong parameters para aumentar a segurança da aplicação

Hora de Colocar a Mão na Massa

PUC Minas
Poços de Caldas

- Modifique o post_params para o código abaixo:

Código 11: controllers/posts_controller.rb

```
1Class PostsController < ApplicationController
2  private
3    # Never trust parameters from the scary internet,
4# only allow the white list through.
5    def post_params
6      #params.require(:post).permite(:title, :content)
7      params
8    end
```

  - tente, agora, criar um post (volte agora para o código original).

Flash (1)

PUC Minas
Poços de Caldas

- **Problema:** Queremos redirecionar um usuário para uma página diferente do nosso site, mas ao mesmo tempo fornecer a ele algum tipo de mensagem. Exemplo: "Postagem criada !"
- **Solução:** flash - uma hash onde a dado persiste por exatamente UMA requisição APÓS a requisição corrente
- Um conteúdo pode ser colocado em um flash assim:

Código 12: controllers/posts_controller.rb

```
1 flash[:attribute] = value
```

- Dois atributos comuns são :notice(good) e :alert (bad)
- Estes dois atributos (:notice ou :alert) podem ser colocados no redirect_to

Flash (2)

PUC Minas
Poços de Caldas

- show.html.erb:

Código 13: views/posts/show.html.erb

```
1 <p id="notice"><%= notice %></p>
2 <p>
3 <strong>Title:</strong>
4 <%= @post.title %>
5 </p>
6 <p>
7 <strong>Body:</strong>
8 <%= @post.body %>
9 </p>
10 <%= link_to 'Edit', edit_post_path(@post) %> |
11 <%= link_to 'Back', posts_path %> end
```

Ação: Edit (1)

PUC Minas
Poços de Caldas

- Recupera uma postagem específica no parâmetro id passado como parte da URL
- (Implicitamente) procura pelo edit.html.erb para renderizar a resposta

Código 14: controllers/posts_controller.rb

```
1 Class PostsController < ApplicationController
2   before_action :set_post, only: [:show, :edit, :update, :destroy]
3
4   #GET /posts/1/edit
5   def edit
6   end
7
8   private
9    def set_post
10    @post = Post.find(params[:id])
```

Ação: Edit (2)

```
11    end
```

- edit.html.erb:

Código 15: controllers/posts_controller.rb

```
1 <h1>Editing Post</h1>
2 <%= render 'form' %>
3 <%= link_to 'Show', @post %> |
4 <%= link_to 'Back', posts_path %>
```

Ação: Update (1)

PUC Minas
Poços de Caldas

- Recupera um objeto post utilizando o parâmetro id
- Atualiza o objeto post com os parâmetros que foram passados pelo formulário edit
- Tenta atualizar o objeto no banco de dados
- Se sucesso, redireciona para o template show
- Se insucesso, renderiza o template edit novamente

Ação: Update (2)

PUC Minas
Poços de Caldas

Código 16: posts_controller.rb

```
1 Class PostsController < ApplicationController
2   # PATCH/PUT /posts/1
3   # PATCH/PUT /posts/1.json
4   def update
5     respond_to do |format|
6       if @post.update(post_params)
7         format.html { redirect_to @post, notice: 'Post was successfully update
8         format.json { render :show, status: :ok, location: @post }
9       else
10        format.html { render :edit }
11        format.json { render json: @post.errors, status: :unprocessable_entity
12      end
13    end
14  end
15  private
16    # Never trust parameters from the scary internet,
17  # only allow the white list through.
18    def post_params
```

Ação: Update (3)

PUC Minas
Poços de Caldas

```
19      params.require(:post).permite(:title, :content)
20    end
```

Hora de Colocar a Mão na Massa (1)

PUC Minas
Poços de Caldas

- Modifique o arquivo de rotas para aninhar os comentários às postagens e reinicie o servidor:

Código 17: config/routes.rb

```
1 Rails.application.routes.draw do
2   resources :comments
3   resources :posts do
4     resources :comments
5   end
6 end
```

- Crie um novo post e salve no banco de dados:

Hora de Colocar a Mão na Massa (2)

PUC Minas
Poços de Caldas

```
irb(main):005:0> p1 = Post.new
irb(main):006:0> p1.title="Whatsapp bloqueado"
irb(main):007:0> p1.body="A justia bloqueou o Whatsapp..."
irb(main):008:0> p1.save
```

- Crie um novo comment e o vincule a um post:

```
irb(main):005:0> c1 = Comment.new
irb(main):006:0> c1.body="O que fazer agora ???"
irb(main):007:0> c1.save
irb(main):008:0> p1.comments << c1
```

Hora de Colocar a Mão na Massa (3)

PUC Minas
Poços de Caldas

- Digite no navegador no endereço ⟨http://localhost:3000/posts/id/comments⟩. Onde o id é o id do post criado anteriormente.
- Agora crie um novo blog e digite novamente ⟨http://localhost:3000/posts/id/comments⟩. Onde o id do blog que acabou de ser criado.(Temos um problema)
- Modifique o código do template views/posts/show.html.erb. Insira o código abaixo, após a linhas 10 (abaixo do parágrafo do body).

Hora de Colocar a Mão na Massa (4)

PUC Minas
Poços de Caldas

```
1 <h2>Comments</h2>
2 <div id="comments">
3   <% @post.comments.each do |comment| %>
4     <%= div_for comment do %>
5     <p>
6       <strong>Posted <%= time_ago_in_words(comment.created_at) %></strong>
7       <%= h(comment.body) %>
8     </p>
9     <% end %>
10  <% end %>
11 </div>
```

- Agora no navegador visualize uma postagem que tenha comentários.
- Acrescente o código a seguir logo abaixo do código anterior no arquivo views/posts/show.html.erb:

Hora de Colocar a Mão na Massa (5)

PUC Minas
Poços de Caldas

```
1 <%= form_for([@post, Comment.new]) do |f| %>
2 <p>
3 <%= f.label :body, "New Comment" %><br>
4 <%= f.text_area :body %>
5 </p>
6 <p>
7 <%= f.submit "Add Comments" %>
8 </p>
9 <% end %>
```

- Modifique a acão create do controlador controllers/comments_controller.rb:

Hora de Colocar a Mão na Massa (6)

PUC Minas
Poços de Caldas

```ruby
def create
  @post = Post.find(params[:post_id])
  @comment = @post.comments.create(comment_params)

  respond_to do |format|
    if @comment.save
      format.html { redirect_to @post, notice: 'Comment was successfully creat
      format.json { render :show, status: :created, location: @comment }
    else
      format.html { render :new }
      format.json { render json: @comment.errors, status: :unprocessable_entit
    end
  end
end
```

- Escolha uma postagem qualquer e escreva alguns comentários.

Hora de Colocar a Mão na Massa (7)

PUC Minas
Poços de Caldas

- Remova a rota absoluta para comentários no arquivo de rotas e reinicie o servidor:

Código 18: config/routes.rb

```
1 Rails.application.routes.draw do
2   #resources :comments
3   resources :posts do
4     resources :comments
5   end
6 end
```

Agenda

PUC Minas
Poços de Caldas

Action View

PUC Minas
Poços de Caldas

- Arquivo HTML com a extensão .erb
  - ERb é uma biblioteca que permite a colocação de código Ruby no HTML
- Dois padrões a aprender:
  - <% ...código ruby..%> avalia o código Ruby
  - <%= ...código ruby..%> retorna o resultado do código avaliado

Partials (1)

PUC Minas
Poços de Caldas

- Rails encoraja o princípio DRY
- O laioute da aplicação é mantida em um único local no arquivo application.html.erb
- O código comum dos templates ser reutilizado em múltiplos templates
- Por exemplo, os formulários do edit e do new - são realmente muito diferentes ?
- Partials são similares aos templates regulares, mas ele possuem capacidades mais refinadas
- Nomes de partials começam com underscore (_)
- Partials são renderizados com render 'partialname' (sem underscore)

Partials (2)

PUC Minas
Poços de Caldas

- render também aceita um segundo argumento, um hash com as variáveis locais utilizadas no partial
- Similar a passagem de variáveis locais, o render pode receber um objeto
- <%= render @post %> renderizara app/views/posts/_posts.html.erb com o conteúdo da variavel @post
- <%= render @posts %> renderiza uma coleção e é equivalente a:

Código 19: controllers/posts_controller.rb

```
1 <% @posts.each do |posts | %>
2 <%= render post %>
3 <% end %>
```

Partials (3)

PUC Minas
Poços de Caldas

- _form.html.erb

Código 20: views/posts/_form.html.erb

```
1<%= form_for(@post) do |f| %>
2  <% if @post.errors.any? %>
3    <div id="error_explanation">
4      <h2><%= pluralize(@post.errors.count, "error") %>
5    prohibited this post from being saved:</h2>
6      <ul>
7      <% @post.errors.full_messages.each do |message| %>
8        <li><%= message %></li>
9      <% end %>
10     </ul>
11   </div>
12 <% end %>
13
14 <div class="field">
15   <%= f.label :title %><br>
```

Partials (4)

PUC Minas
Poços de Caldas

```
16      <%= f.text_field :title %>
17    </div>
18    <div class="field">
19      <%= f.label :body %><br>
20      <%= f.text_area :body %>
21    </div>
22    <div class="actions">
23      <%= f.submit %>
24    </div>
25  <% end %>
```

Form Helpers (1)

PUC Minas
Poços de Caldas

- form_for gere a tag form para o objeto passado como parâmetro
- Rails utiliza a método POST por padrão
- Isto faz sentido:
  - uma password não é passada como parâmetro na URL
  - qualquer modificação deverá ser feita via POST e não GET

Código 21: views/posts/_form.html.erb

```
1 <%= form_for(@post) do |f| %>
2 ...
3 <% end %>
```

f.label

PUC Minas
Poços de Caldas

- Gera a tag HTML label
- A descrição pode ser personalizada passando um segundo parâmetro

```
1 <div class="field">
2   <%= f.label :title, "Titulo" %><br>
3   <%= f.text_field :title %>
4 </div>
```

f.text_field

PUC Minas
Poços de Caldas

- Gera o campo input type=”text”
- Utilize :placeholder para mostrar um valor dentro do campo

```
1 <div class="field">
2   <%= f.label :title, "Titulo" %><br>
3   <%= f.text_field :title, placeholder: "Escreva o titulo aqui." %>
4 </div>
```

f.text_area

PUC Minas
Poços de Caldas

- Similar ao f.text_field, mas gera um text area de tamanho (40 cols x 20 rows)
- O tamanho pode ser modificado através do atriburo size:

```
1 <div class="field">
2   <%= f.label :body, "Conteudo" %><br>
3   <%= f.text_area :body, size: "10x3" %>
4 </div>
```

Outros Form Helpers

PUC Minas
Poços de Caldas

- date_select
- search_field
- telephone_field
- url_field
- email_field
- number_field
- range_field

f.submit

PUC Minas
Poços de Caldas

- Renderiza o botão submit
- Aceita o nome do botão submit como primeiro argumento
- Se o nome não for fornecido - gera um baseado no modelo e na ação. Por exemplo: "Create Post"ou "Update Post"

```
1 <div class="actions">
2   <%= f.submit "Postar"%>
3 </div>
```

- Mais form helpers: ⟨http://guides.rubyonrails.org/form_helpers.html⟩