Ruby On Rails

Engenharia de Software

Luiz Alberto Ferreira Gomes

Curso de Ciência da Computação

23 de setembro de 2019

Aplicação Web (1)

- 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)

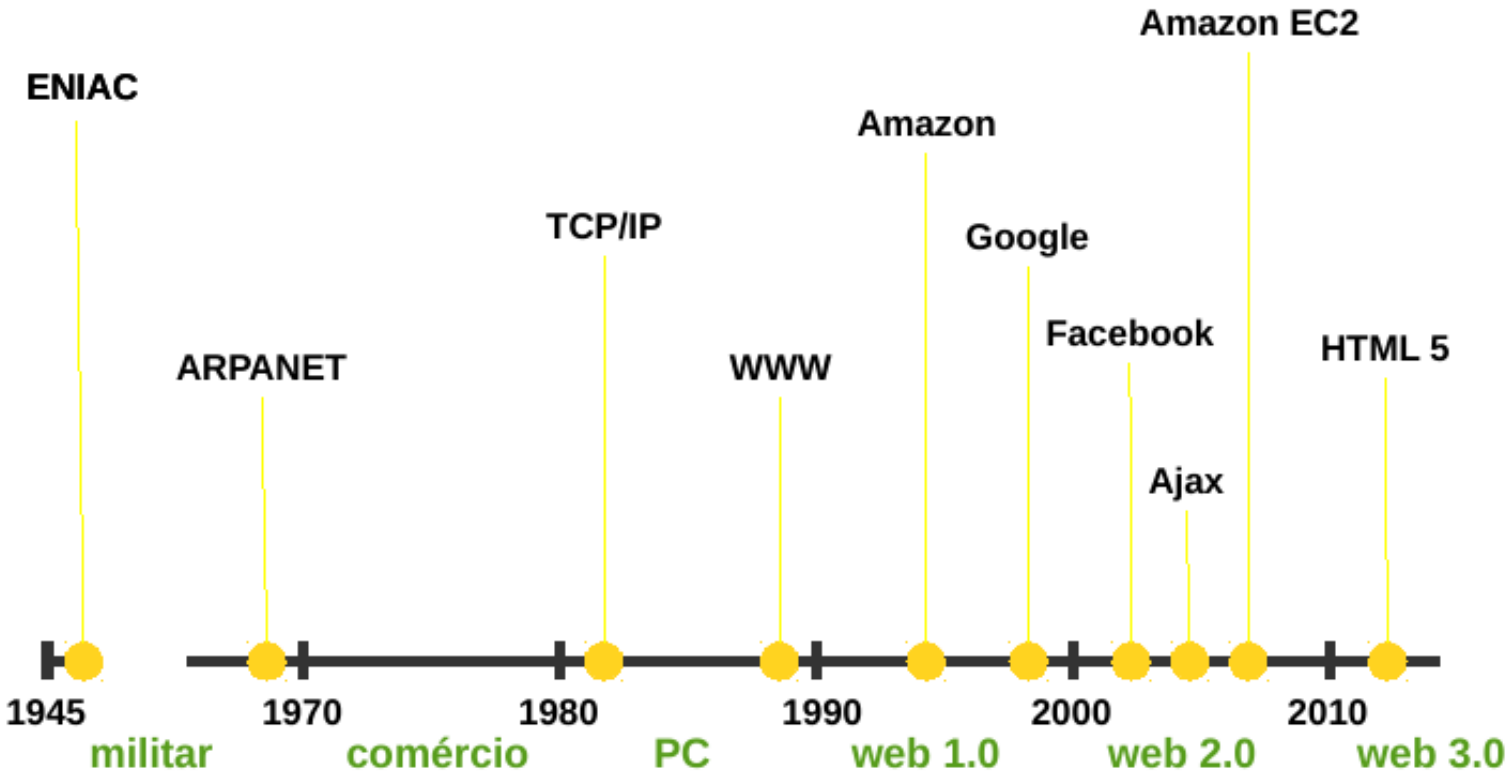
- Há um pouco mais do que isso:
  - Rede de Computadores:
    - a Internet, um sistema global de redes de computadores interconectadas.
    - utiliza o conjunto de protocolos TCP/IP.
  - Web (World Wide Web):
    - um sistema de documentos (em inglês: *web pages*) vinculados que são acessados através 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:

Aplicação Web (3)

- Conveniência pela utilização um web browser como cliente.
- Compatibilidade inerente entre plataformas.
- 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

- 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)

- 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

Modelos de Computação em Nuvem (2)

- 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)

- 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.

Arquiteturas de Aplicações Web (2)

- 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)

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.

Design Patterns (2)

- 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)

- 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)

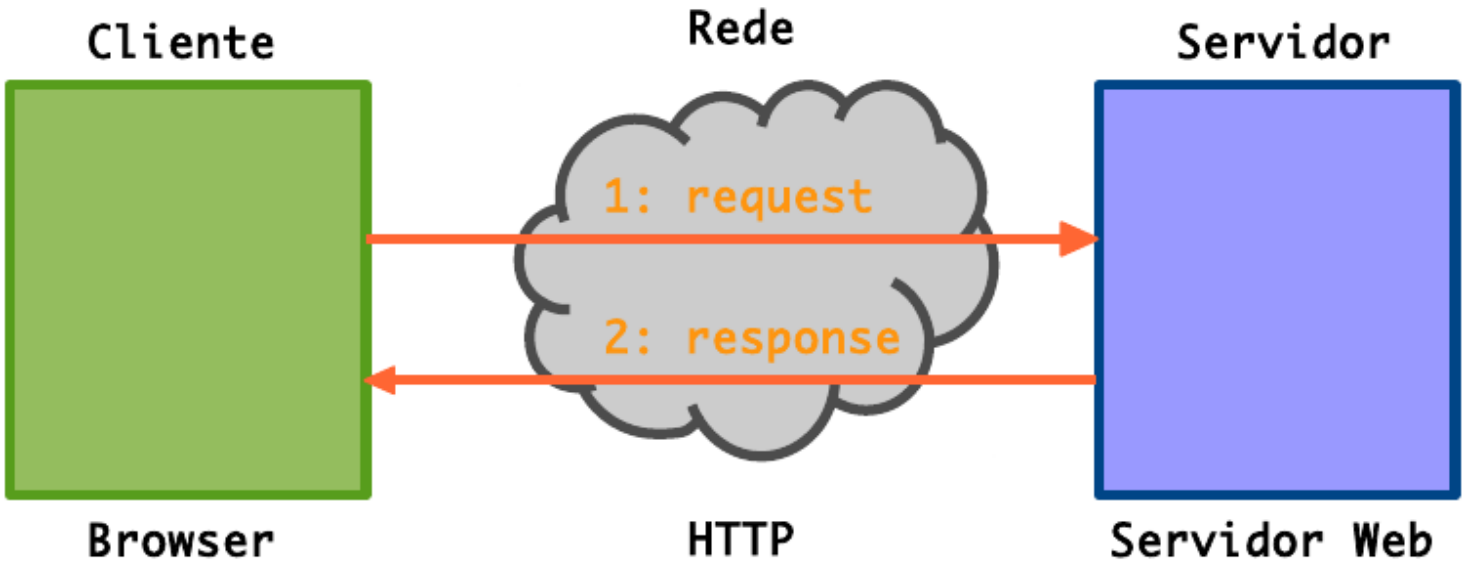


Figura: Arquitetura cliente servidor.

Modelo Cliente-Servidor (3)

- 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)

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)

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.

Arquitetura N-Tier (3)

- Camadas podem ser atualizados ou substituídos de forma independente a medida que os requisitos ou tecnologia.
  - 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)

- 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.

Arquitetura 3-Tiers (2)

- Aplicação (lógica)
  - 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.

Filosofia do Rails (1)

- 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.er
```

Filosofia do Rails (2)

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

Rails (1)

- Rails é um *framework* para construção de aplicações web
- David Heinemeier Hanson derivou o 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"

Rails (2)

- nesse mesmo no o Rails começou a ganhar muita atenção da comunidade de desenvolvimento web.

- Rails é utilizado por diversas companhias, como por exemplo:
  - Airbnb, BaseCamp, Disney, GitHub, Hulu, Kickstarter, Shopify e Twitter.

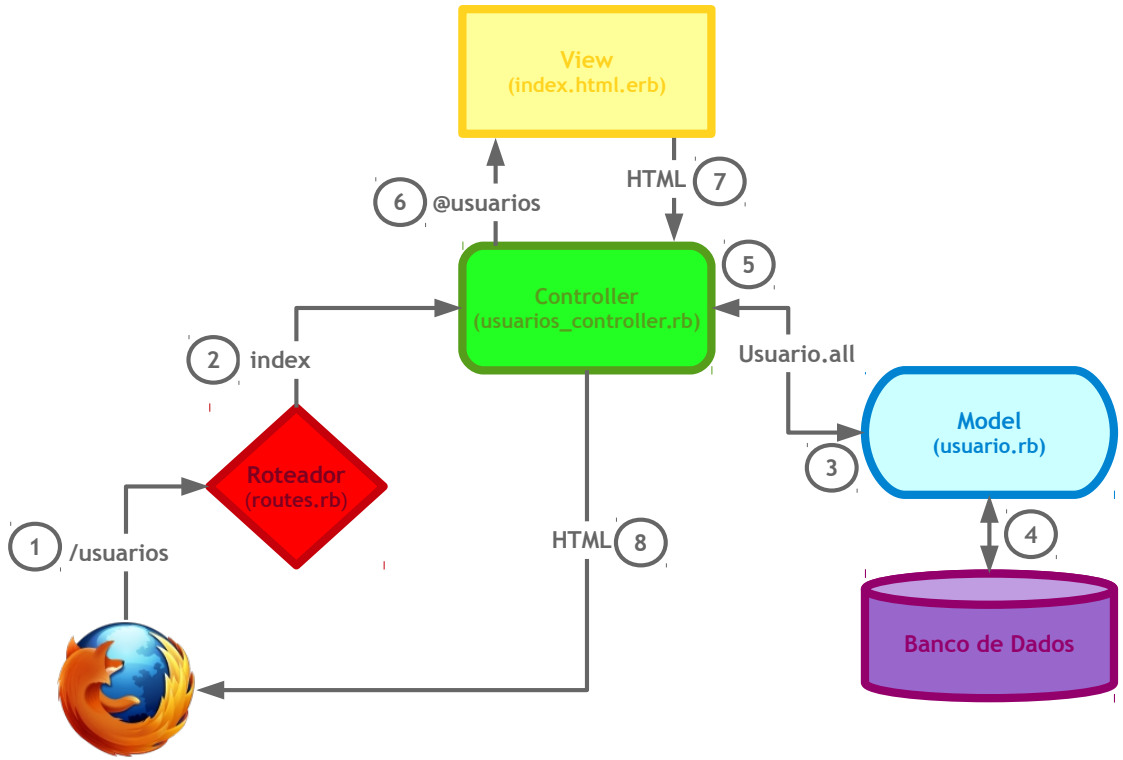
Rails (3)

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

- Conecte-se na máquina com o usuário a1550099999 e senha 333333
  1. Inicie uma janela de terminal e digite no prompt: [style=BashInputBasicStyle] $rails new my_app$
  2. Mude para o diretório da aplicao (RAILS.root) [style=BashInputBasicStyle] $cd new my_app$
  3. Execute o servidor web embutido: [style=BashInputBasicStyle] *rails s*
  4. Abra uma janela do navegador e digite: [style=BashInputBasicStyle] $http://localhost:3000$

I1 - Hora de Colocar as Mãos na Massa (1)

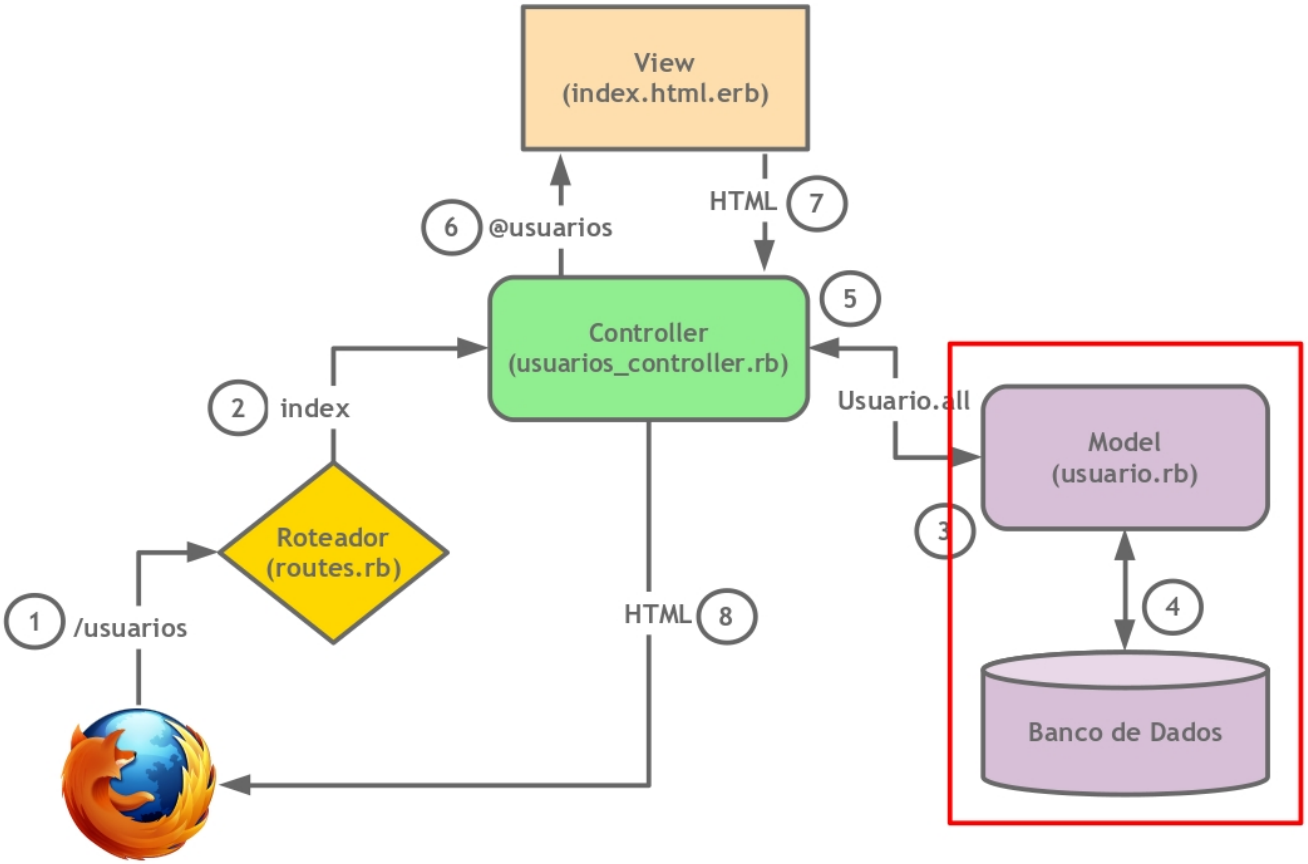
1. Gerar o modelo para os *posts* [style=BashInputBasicStyle] *railsgeneratemodelPosttitle* : *stringbody* : *text*
2. Gerar o modelo para os *comentários* [style=BashInputBasicStyle] *railsgeneratemodelCommentpost*$_i$*d* : *integerbody* : *text*
3. Gere as tabelas `post` e `comment` no banco de dados [style=BashInputBasicStyle] *rakedb* : *create* rake db:migrate

Agenda

Modelo

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

Modelo


View
(index.html.erb)
6
@usuarios
HTML
7
Controller
(usuarios_controller.rb)
5
2
index
Usuario.all
Model
(usuario.rb)
Roteador
(routes.rb)
3
4
1
/usuarios
HTML
8
Banco de Dados

Banco de Dados Relacionais (1)

- 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)

- 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)

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

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

[style=RubyInputStyle, firstline=7]codigos/blog$_1$/*config*/*database.yml*

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

SQLite (2)

ATENÇÃO: SQLite não um banco de dados para produção !

- Banco de dados de produção populares: MySQL e PostgreSQL

Database Console

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

[style=BashInputBasicStyle, basicstyle=, keepspaces=true]

*railsdbSQLiteversion*3.8.7.12014 − 10 − 2913 : 59 : 56*Enter*" .*help*" *forusagehints*.*sqlite* > .*headersonsqlite* >

.*modecolumnssqlite* > *select* ∗ *fromposts*; *idtitlebodycreated$_a$tupdated$_a$t* − − − − − − − − − − − − − − − − −

− −

− −5*ALinguagemRubyRubyelegal*.2016 − 04 − 3022 :

45 : 20.6363632016 − 04 − 3022 : 45 : 20.636363*sqlite* >

- Dica: utilize headers on e mode coluns

Hora de Colocar a Mão na Massa (1)

- Inicialize na pasta da aplicação a console do banco de dados e configure a sua exibição: [style=BashInputBasicStyle] *railsdbsqlite* > *.headersonsqlite* > *.modecolumns*
- Exiba os colunas da tabela `posts`: [style=BashInputBasicStyle] sqlite¿ .schema posts

Hora de Colocar a Mão na Massa (2)

- Crie um novo `post` e salve no banco de dados:
  [style=BashInputBasicStyle] sqlite¿ INSERT INTO posts
  (title, body,
  $created_at, updated_at) VALUES("Seminariosda Computacao","Temv$
  $10-1619:50:00","2017-16-0319:50:00");$
- Crie outro `post` e salve no banco de dados:
  [style=BashInputBasicStyle] sqlite¿ INSERT INTO posts
  (title, body,
  $created_at, updated_at) VALUES("Calorem Pocos","Comoestaquente!$
  $10-1819:50:00","2017-10-1819:50:00");$
- Exiba todos os `posts`: [style=BashInputBasicStyle] sqlite¿
  SELECT * FROM posts;

Hora de Colocar a Mão na Massa (3)

- Exiba todos os `posts` ordenados pelo título (title): [style=BashInputBasicStyle] sqlite¿ SELECT * FROM posts ORDER BY title;
- Exiba um `post`: [style=BashInputBasicStyle] sqlite¿ SELECT * FROM posts LIMIT 1
- Exiba o `post` cujo `id` é 2: [style=BashInputBasicStyle] sqlite¿ SELECT * FROM posts WHERE id=2;
- Atualize o título `post` cujo o `id` é 2: [style=BashInputBasicStyle] sqlite¿ UPDATE posts SET title="O tempo esta louco"WHERE id=2;
- Remova `post` cujo o `id` é 2: [style=BashInputBasicStyle] sqlite¿ DELETE FROM posts WHERE id=2;

Migrations (1)

- 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 (3)

- A cada vez que o generate model 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`

[style=RubyInputStyle]codigos/blog$_1$/*db*/*migrate*/20160430140114$_c$*re*

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

[style=BashInputBasicStyle] *rakedb* : *migrate*

Object-Relational Mapping (1)

- 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)

- ActiveRecord é o nome do ORM padrão do Rails?

[style=RubyInputStyle, caption=app/models/post.rb]codigos/blog_1/app/models/post.rb

Onde está código ?
R: Metaprogramação + Convenção

- 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 `ApplicationRecord`
  - **(Convenção)** espera-se que a tabela tenha uma chave primário denominada id

Object-Relational Mapping (1)

- 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)

- Inicialize na pasta da aplicação a console do Rails (não a do banco de dados): [style=BashInputBasicStyle] *railsc*
- Exiba os atributos da classe `Post`: [style=BashInputBasicStyle] irb(main):004:0¿ Post.column$_n$*ames*
- Crie um novo `post` e salve no banco de dados: [style=BashInputBasicStyle] 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
- Exiba todos os `posts`: [style=BashInputBasicStyle] irb(main):007:0¿ Post.all

Hora de Colocar a Mão na Massa (2)

- Exiba todos os `posts` ordenados pelo título (title): [style=BashInputBasicStyle] irb(main):007:0¿ Post.all.order(title: :asc)
- Exiba um `post`: [style=BashInputBasicStyle] irb(main):007:0¿ Post.first
- Exiba o `post` cujo `id` é 2: [style=BashInputBasicStyle] irb(main):007:0¿ Post.find$_by$(*id* : 2)
- Atualize o título do primeiro `post`: [style=BashInputBasicStyle] irb(main):007:0¿ p1=Post.first irb(main):008:0¿ p1.update(title: "Pensando...")
- Remova do primeiro `post`: [style=BashInputBasicStyle] irb(main):007:0¿ p1=Post.first irb(main):008:0¿ p1.destroy

Validação em Aplicações Web

- 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

- 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

- 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)

- 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

- 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: [style=RubyInputStyle] class Person ¡ ApplicationRecord validates$_p$*resence*$_o$*f* : *name*validates$_n$*umeracality*$_o$*f* : *age*, : *only*$_i$*nteger* => *true*validates$_c$*onfirmation*$_o$*f* : *email*validates$_l$*ength*$_o$*f* : *password*, : *in* => 8..20*end*

I2 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo `app/models/post.rb` para exigir que o usuário digite o título e o texto do blog:
  [style=RubyInputStyle] class Post ¡ ApplicationRecord validates$_p$*resence*$_o$*f* : *title*, : *bodyend*
- Modifique o arquivo `app/models/comment.rb` para exigir que o usuário digite texto do comentário blog:
  [style=RubyInputStyle] class Post ¡ ApplicationRecord validates$_p$*resence*$_o$*f* : *bodyend*

I2 - Hora de Colocar a Mão na Massa (2)

- Reinicie a console do Rails tente criar um Post e um Comment [style=BashInputBasicStyle] irb(main):005:0¿ p1 = Post.new irb(main):006:0¿ p1.body="Tem algo errado..."irb(main):007:0¿ p1.save irb(main):008:0¿ Post.all irb(main):009:0¿ c1 = Comment.new irb(main):010:0¿ c1.save irb(main):011:0¿ Comment.all

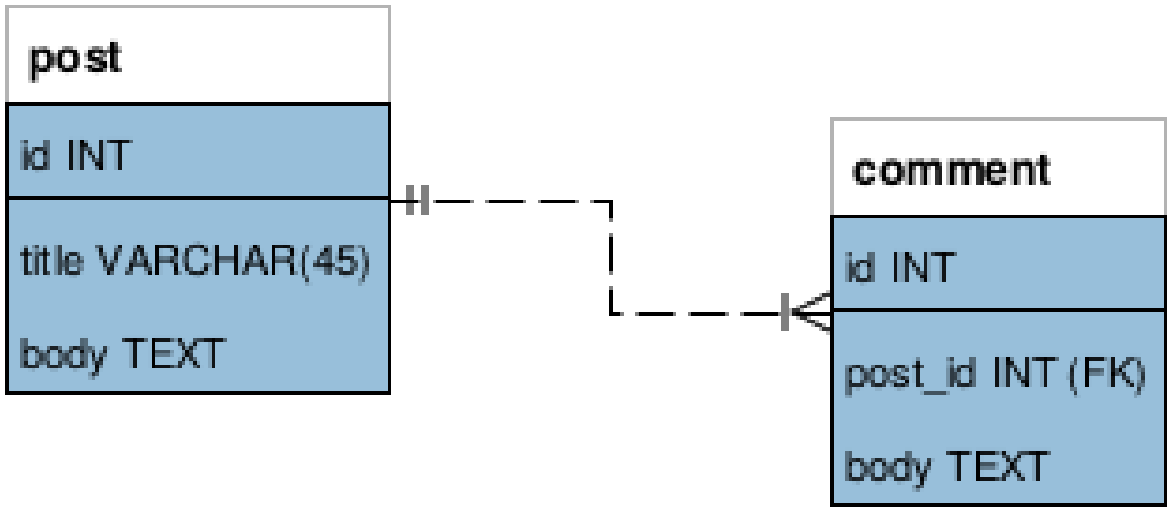
Associações em Rails (1)

- O gerador de modelos 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:

Associações em Rails (2)

- cada postagem precisa saber os comentários associado a ele
- cada comentário precisa saber qual é a postagem ele pertence

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



Associações em Rails (3)

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

I3 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo `app/models/post.rb` para associar o post aos seus comentário: [style=RubyInputStyle] class Post ¡ ApplicationRecord
  validates$_p$*resence*$_o$*f* : *title*, : *bodyhas*$_m$*any* : *commentsend*
- Modifique o arquivo `app/models/comment.rb` para associar o comentário ao seu post: [style=RubyInputStyle] class Comment ¡ ApplicationRecord
  validates$_p$*resence*$_o$*f* : *bodybelongs*$_t$*o* : *postend*

I3 - Hora de Colocar a Mão na Massa (2)

- Crie um novo `post` e salve no banco de dados (Reinicie a console do Rails): [style=BashInputBasicStyle] irb(main):005:0¿ p1 = Post.new irb(main):006:0¿ p1.title="Associacao"irb(main):007:0¿ p1.body="Eu tenho comentarios!"irb(main):008:0¿ p1.save
- Crie um novo `comment` e o vincule a um `post`: [style=BashInputBasicStyle] irb(main):005:0¿ c1 = Comment.new irb(main):006:0¿ c1.body="Eu sou de um post!"irb(main):007:0¿ c1.post = p1 irb(main):008:0¿ c1.save
- Consulte os comentários do `post` p1: [style=BashInputBasicStyle] irb(main):005:0¿ p1.comments.all

I3 - Hora de Colocar a Mão na Massa (3)

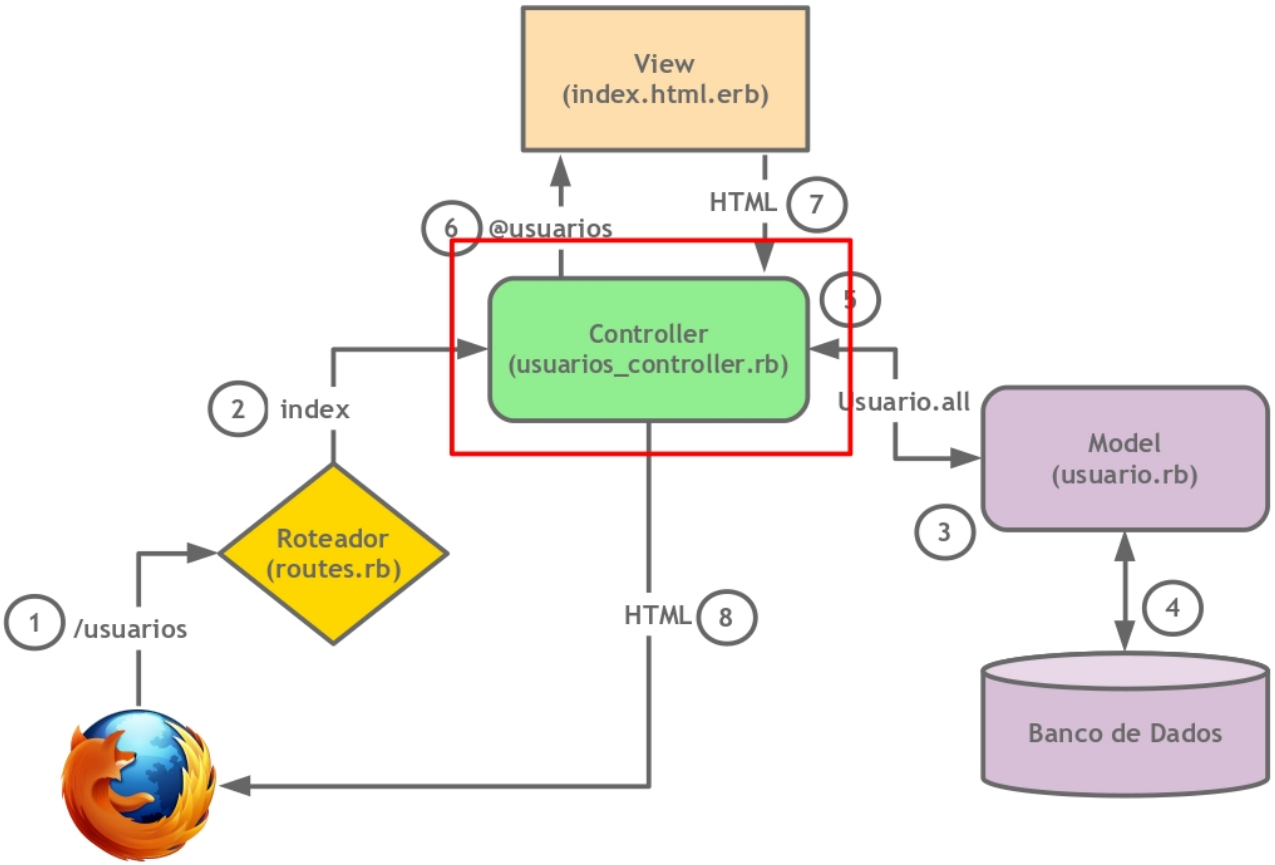
- Consulte os comentários 2 do `post` p1:
  [style=BashInputBasicStyle] irb(main):005:0¿
  p1.comments.where(id: 2)
- Consulte o `post` do comentário c1:
  [style=BashInputBasicStyle] irb(main):005:0¿ c1.post

Agenda

Controlador (1)

- 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:

Controlador (2)



Representational State Transfer (REST)

- Index lista todos os recursos disponíveis
- Show um recurso específico
- Destroy um recurso existente
- New instância um novo recurso
- Create salva um novo recurso
- Edit instância um recurso existente
- Update um recurso existente

Ação: Index (1)

- Ação que recupera todas as postagens do blog
- (Implicitamente) procura pelo template index.html.erb para renderizar a resposta [style=RubyInputStyle, caption=controllers/posts_controller.rb] Class PostsController ¡ ApplicationController def index @posts = Post.all end end

Ação: Index (2)

- index.html.erb: [style=RubyInputStyle, caption=views/posts/index.html.erb] ¡p id=”notice»¡¡h1¿Posts¡/h1¿ ¡table¿ ¡thead¿ ¡tr¿ ¡th¿Titulo¡/th¿ ¡th¿Conteudo¡/th¿ ¡th colspan=”3»¡/th¿ ¡/tr¿ ¡/thead¿ ¡tbody¿ ¡¡tr¿ ¡td¿¡¡td¿¡¡td¿¡¡td¿¡edit$_{p}ost_{p}ath(post) < td >< data : confirm :' Temcerteza?' < /tr ><< /tbody >< /table ><$

respond_to (1)

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

[style=RubyInputStyle, caption=controllers/posts_controller.rb]
Class PostsController ¡ ApplicationController
POST /posts POST /posts.json def create @post = Post.new(post$_p$arams)
respond$_t$odo|format|if @post.saveformat.htmlredirect$_t$o@post, notice :

redirect_to

- 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)

- Remove uma postagem específica pelo parâmetro id passado como parte da URL [style=RubyInputStyle, caption=posts_controller.rb] Class PostsController ¡ ApplicationController
  before$_a$*ction* : *set*$_p$*ost*, *only* : [: *show*, : *edit*, : *update*, : *destroy*]
  def destroy @post.destroy
  redirect$_t$*oposts*$_u$*rl*, *notice* :′ *Postfoiremovidocomsucesso*!′*end*

Flash (1)

- **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: [style=RubyInputStyle, caption=controllers/posts_controller.rb] flash[:attribute] = value
- Dois atributos comuns são :notice(good) e :alert (bad)

Flash (2)

- Estes dois atributos (:notice ou :alert) podem ser colocados no redirect_to
- show.html.erb: [style=RubyInputStyle, caption=views/posts/show.html.erb] ¡p id=”notice≫¡¡p¿ ¡strong¿Title:¡/strong¿ ¡¡/p¿ ¡p¿ ¡strong¿Body:¡/strong¿ ¡¡/p¿ ¡¡

Ação: Edit (1)

- Recupera uma postagem específica no parâmetro id passado como parte da URL
- (Implicitamente) procura pelo edit.html.erb para renderizar a resposta [style=RubyInputStyle, caption=controllers/posts_controller.rb] Class PostsController ¡ ApplicationController
  $before_action : set_post, only : [: show, : edit, : update, : destroy]$
  def edit end
- edit.html.erb: [style=RubyInputStyle, caption=view/posts/edit.html.erb] ¡h1¿Edit Post¡/h1¿ ¡¡¡

Ação: Update (1)

- 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 [style=RubyInputStyle, caption=posts_controller.rb] Class PostsController ¡ ApplicationController def update if @post.update(post$_{p}arams$)$redirect_{t}o@post$, $notice$:′ $Postmodificadocomsucesso$!′$elserender$: $editendend$

I13 - Hora de Colocar a Mão na Massa (1)

- Modifique o arquivo de rotas para aninhar os comentários às postagens e reinicie o servidor: [style=RubyInputStyle, caption=config/routes.rb] Rails.application.routes.draw do resources :posts do resources :comments end end
- Modifique o código do template views/posts/show.html.erb. Insira o código abaixo do parágrafo do body. [style=RubyInputStyle] ¡h2¿Comments¡/h2¿ ¡div id=”comments» ¡¡p¿ ¡strong¿Posted ¡¡i/p¿ ¡¡/div¿
- Agora no navegador visualize uma postagem que tenha comentários.

I13 - Hora de Colocar a Mão na Massa (2)

- Acrescente o código a seguir logo abaixo do código anterior no arquivo views/posts/show.html.erb: [style=RubyInputStyle] ¡¡p¿ ¡¡¡/p¿ ¡p¿ ¡¡/p¿ ¡
- Gere o controlador para os comments: [style=BashInputBasicStyle] *railsgeneratecontrollercomments*

I13 - Hora de Colocar a Mão na Massa (3)

- Modifique a acão create do controlador controllers/comments_controller.rb: [style=RubyInputStyle] before$_a$*ction* : *set*$_c$*omment*, *only* : [: *show*, : *edit*, : *update*, : *destroy*]

def create @post = Post.find(params[:post$_i$*d*])@*comment* = @*post*.*comments*.*create*(*comment*$_p$*arams*)
if @comment.save redirect$_t$*o*@*post*, *notice* :′
Commentfoicriadocomsucesso!′*elseredirect*$_t$*o*@*postendend*
private def
set$_c$*omment*@*comment* = *Comment*.*find*(*params*[: *id*])*end*
def comment$_p$*aramsparams*.*require*(: *comment*).*permit*(: *body*)*end*

I13 - Hora de Colocar a Mão na Massa (4)

- Escolha uma postagem qualquer e escreva alguns comentários.

Agenda

Action View

- 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)

- 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 (2)

- Partials são renderizados com render 'partialname' (sem underscore)
- 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

Partials (3)

- <%= render @posts %> renderiza uma coleção e é equivalente a: [style=RubyInputStyle, caption=controllers/posts_controller.rb] ¡¡¡

Partials (4)

- _form.html.erb [style=RubyInputStyle, caption=views/posts/_form.html.erb] ¡¡¡div id="error_explanation» < h2 >< impedemqueospostsejasalvo :< /h2 >< ul ><< li ><<< /ul >< /div >< ¡div class="field» ¡¡¡/div¿ ¡div class="field» ¡¡¡/div¿ ¡div class="actions» ¡¡/div¿ ¡

Form Helpers (1)

- 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

[style=RubyInputStyle, caption=views/posts/_form.html.erb] ¡...
¡

f.label

- Gera a tag HTML label
- A descrição pode ser personalizada passando um segundo parâmetro [style=RubyInputStyle] ¡div class=”field» ¡¡¡/div¿

f.text_field

- Gera o campo input type=”text”
- Utilize :placeholder para mostrar um valor dentro do campo [style=RubyInputStyle] ¡div class=”field» ¡¡¡/div¿

f.text_area

- 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: [style=RubyInputStyle] ¡div class=”field» ¡¡¡/div¿

Outros Form Helpers

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

f.submit

- 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"[style=RubyInputStyle] ¡div class="actions» ¡¡/div¿
- Mais form helpers: ⟨http://guides.rubyonrails.org/form_helpers.html⟩