MODELOS, BANCOS DE DADOS E ACTIVE RECORD

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

Daniel Cordeiro

1º de setembro de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

AVISOS

5 e 8/set Semana da Pátria

12/set entrega da 1ª iteração e acompanhamento do andamento do projeto com a monitora + introdução ao Pivotal Tracker

16/set não haverá aula

- Como armazenar e recuperar dados restruturados como registros?
- Qual a relação entre a forma que os dados estão armazenados com a forma como eles são manipulados em uma linguagem de programação?

OBJETOS IN-MEMORY VS. IN-STORAGE

- Como representar objetos que foram gravados em um dispositivo de armazenamento?
  - Exemplo: `Movie` com os atributos `name` e `rating`
- Operações básicas em um objeto: CRUD (`Create, Read, Update, Delete`)
- ActiveRecord: todo modelo sabe como realizar as operações CRUD em si mesmos, usando mecanismos comuns

- Cada tipo de modelo possui sua própria tabela no BD
  - todas as linhas da tabela tem uma estrutura idêntica
  - uma linha na tabela == uma instância da classe do modelo
  - cada coluna armazena o valor de um atributo do modelo
  - cada linha tem um único valor como *chave primária* (por convenção, o Rails usa um inteiro e o chama de `id`

`id`	`rating`	`title`	`release_date`
2	Livre	Pets – A Vida Secreta dos Bichos	2016-08-25
11	Livre	Procurando Dory	2016-06-30
22	14 anos	Quando as Luzes se Apagam	2016-08-18

- *Schema*: coleção de todas as tabelas e de suas estruturas

CONTROLADORES, ROTAS E RECURSOS REST

- Quais decisões de projeto permitirão ao nosso app implementar uma Arquitetura Orientada a Serviços (SOA)?

ROTAS

- Em MVC, cada interação do usuário pode ser tratada por um `controller action`
  - método Ruby que trata a interação
- Uma *rota* mapeia uma tupla `<método HTTP, URI>` para uma ação do controlador

Rota	Ação
`GET /movies/3`	Mostra informação sobre o filme cujo ID=3
`POST /movies`	Cria novo filme usando os dados da requisição
`PUT /movies/5`	Atualiza o filme com ID=5 usando dados da req.
`DELETE /movies/5`	Apaga o filme cujo ID=5

SISTEMA DE ROTEAMENTO DO RAILS

- despacha <método, URI> para a ação do controlador correto
- provê métodos auxiliares (`helper methods`) que geram um par <método,URI> dada uma ação de controlador
- analisa parâmetros da requisição (da URI ou formulário) e cria um hash para acessá-los
- métodos `built-in` para gerar todas as rotas CRUD

`rake routes`

```
  GET /movies          {:action=>"index", :controller=>"movies"}
C POST /movies         {:action=>"create", :controller=>"movies"}
  GET /movies/new      {:action=>"new", :controller=>"movies"}
  GET /movies/:id/edit {:action=>"edit", :controller=>"movies"}
R GET /movies/:id      {:action=>"show", :controller=>"movies"}
U PUT /movies/:id      {:action=>"update", :controller=>"movies"}
D DELETE /movies/:id   {:action=>"destroy", :controller=>"movies"}
```

GET /movies/3/edit HTTP/1.0

- Casa com a rota `GET /movies/:id/edit :action=>"edit", :controller=>"movies"`
- Analisa os parâmetros curinga: `params[:id] = "3"`
- Despacha a requisição para o método `edit` em `movies_controller.rb`
- Para incluir na visão gerada uma URI que irá submeter o formulário para a ação `update` do controlador com `params[:id] = "3"`, chame o método auxiliar `update movie path(3) # => PUT /movies/3`

`rake routes`

```
  GET /movies          {:action=>"index", :controller=>"movies"}
C POST /movies         {:action=>"create", :controller=>"movies"}
  GET /movies/new      {:action=>"new", :controller=>"movies"}
  GET /movies/:id/edit {:action=>"edit", :controller=>"movies"}
R GET /movies/:id      {:action=>"show", :controller=>"movies"}
U PUT /movies/:id      {:action=>"update", :controller=>"movies"}
D DELETE /movies/:id   {:action=>"destroy", :controller=>"movies"}
```

PERGUNTA

Qual afirmação não é verdadeira em relação às rotas RESTful do Rails e os recursos referenciados por elas:

- Um recurso pode representar um conteúdo existente ou uma requisição para modificar algo
- Em um app MVC, toda rota deve disparar uma ação do controlador
- Um conjunto comum de ações RESTful são as ações CRUD nos modelos
- Uma rota sempre deve conter um ou mais parâmetros "curingas" tais como `:id`, para identificar uma instância particular de um recurso

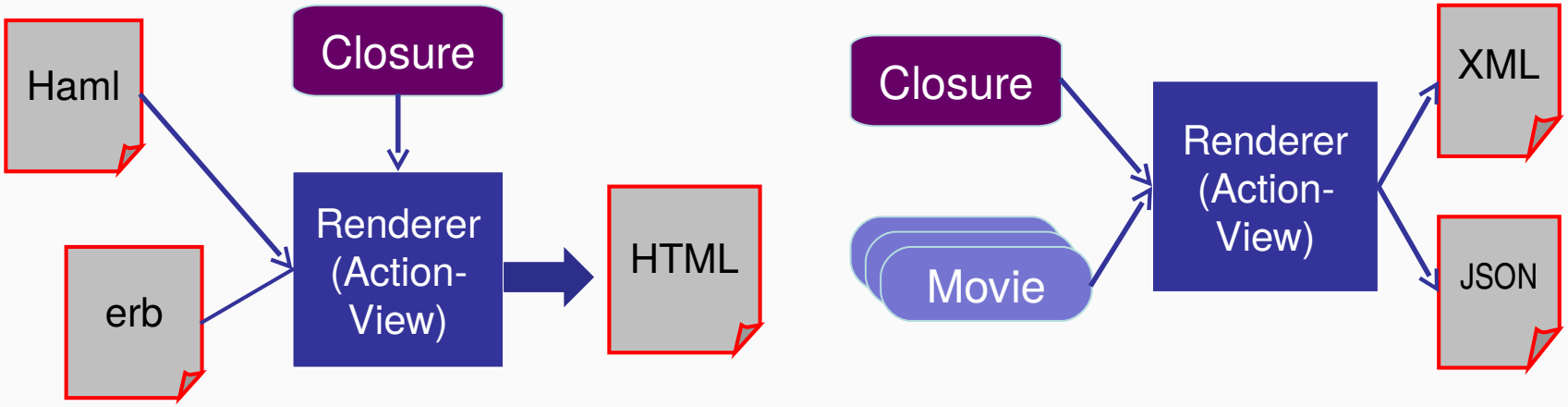
TEMPLATE VIEWS E HAML

TEMPLATE VIEWS E HAML

- HTML é como representamos conteúdo para os navegadores
- ... mas como é o processo pelo qual a saída do nosso app se torna HTML?

PADRÃO TEMPLATE VIEW

- A visão consiste de um *markup* intercalado com trechos que serão preenchidos em tempo de execução
  - Normalmente, valores de variáveis ou o resultado da avaliação de um pedaço pequeno de código
- Antigamente, *isso era o app* (ex: PHP)
- Alternativa: padrão *Transform View*



HAML É UM “HTML DE REGIME”

```
%h1.pagename All Movies
%table#movies
  %thead
    %tr
      %th Título do livro
      %th Data de lançamento
      %th Mais informações
  %tbody
    - @movies.each do |movie|
      %tr
        %td= movie.title
        %td= movie.release_date
        %td= link_to "Mais em #{movie.title}",  |
               movie_path(movie) |
= link_to 'Adicionar novo filme', new_movie_path
```

NÃO COLOQUE CÓDIGO NAS SUAS VIEWS

- Sintaticamente, você pode colocar qualquer código na view
- Mas MVC defende que a visão e controle devem ser enxutos
    - A sintaxe de Haml, de propósito, dificulta incluir um monte de código
- *Helpers* (métodos que "embelezam" os objetos para inclusão nas views) tem um espaço reservado em uma app Rails
- Alternativa para Haml: *templates* html.erb (Embedded Ruby), têm mais cara de PHP

SUMÁRIO & REFLEXÕES: ARQUITETURA SAAS

Visão geral (arquitetura)

View	HTML & CSS; XML & XPath
Controller	URIs, HTTP, TCP/IP + REST & rotas RESTful
Model	Bancos de dados e migrações + CRUD

2008: “Rails não é escalável”

- Escalabilidade é uma preocupação arquitetural; não é restrita à linguagem ou arcabouço
- Arquiteturas de 3 camadas escalam, desde que *stateless*
- Bancos de dados **relacionais** tradicionais não escalam bem
- Várias soluções que combinam sistemas relacionais com não-relacionais (“NoSQL”) escalam muito melhor
- Uso inteligente de *caching* pode melhorar consideravelmente os custos constantes

ARCABOUÇOS, APPS, PADRÕES DE PROJETOS

- Vimos vários padrões de projeto, veremos mais no futuro
- Em 1995, a situação era caótica: os maiores sites Web eram minicomputadores e não 3-camadas/nuvem
- As melhores práticas (padrões) foram “extraídos” e capturados em arcabouços
- Mas a API transcende isso: protocolos de 1969 + linguagem de marcação de 1960 + navegador de 1990 + servidores Web de 1992 continuam funcionando até hoje

ARQUITETURA: SEMPRE HÁ ALTERNATIVAS

Padrão usado	Alternativas
Cliente–servidor	Peer-to-Peer
Shared-nothing (cloud)	Multiprocessamento simétrico (SMP), espaço de endereçamento global e compartilhado
Model–View–Controller	Page controller, Front controller, Template View
ActiveRecord	Data Mapper
URIs RESTful	a mesma URI faz coisas diferentes dependendo do estado do sistema

No futuro sua experiência com apps SaaS vai te fazer questionar qual arquitetura é mais adequada à cada situação.

SUMÁRIO: ARQUITETURA & RAILS

- Model–View–Controller é um padrão arquitetural muito usado para a estruturação de apps
- Rails assume que a app SaaS segue MVC
- Views são Haml com código Ruby embutido que são convertidos em HTML antes de serem enviados ao navegador
- Models são armazenados em tabelas em um BD relacional, acessados com ActiveRecord
- Controllers amarram as views aos modelos com rotas e código nos métodos de controle