

# [IF977] Engenharia de Software

---

Prof. Vinicius Cardoso Garcia  
[vcg@cin.ufpe.br](mailto:vcg@cin.ufpe.br) :: [@vinicius3w](https://twitter.com/vinicius3w) :: [assertlab.com](http://assertlab.com)



# Licença do material

---

Este Trabalho foi licenciado com uma Licença

**Creative Commons - Atribuição-NãoComercial-Compartilhagual 3.0 Não Adaptada.**

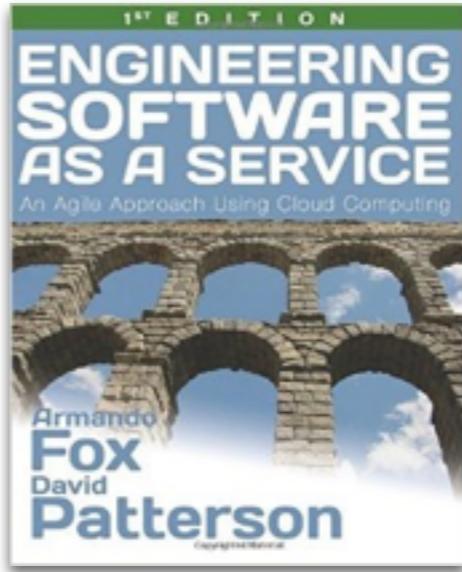
Mais informações visite

<http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt>



# Referências

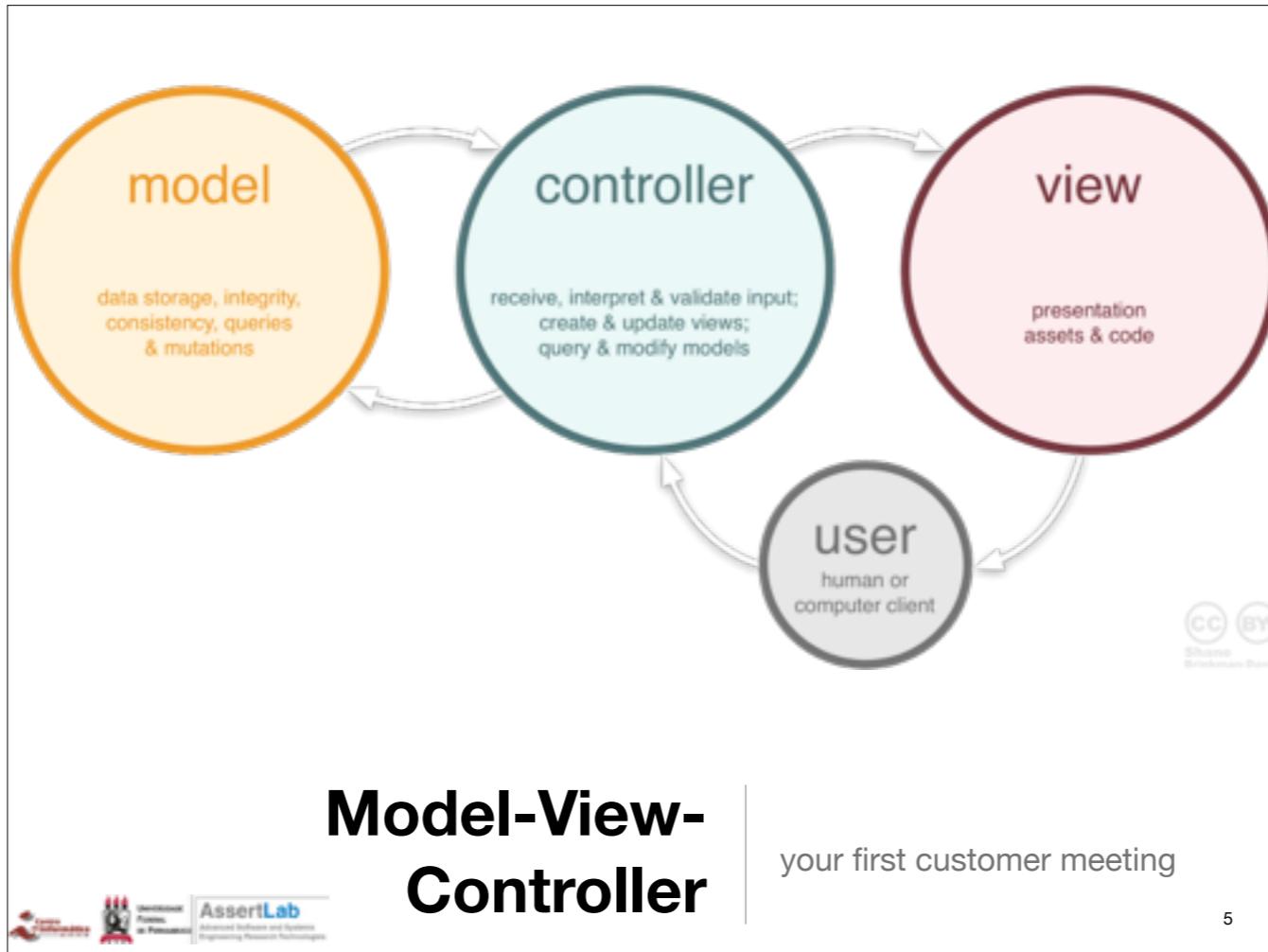
- A biblioteca do Desenvolvedor de Software dos dias de hoje
  - <http://bit.ly/TDOA5L>
- SWEBOK
  - Guide to the Software Engineering Body of Knowledge (SWEBOK): <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing (Beta Edition)
  - <http://www.saasbook.info/>



# Outline

---

- (ESaaS 2.5) Model-View-Controller
- (ESaaS 2.6) Active Record for Model
- (ESaaS 2.7) Routes, Controllers, and REST
- (ESaaS 2.8) Template Views
- (ESaaS 2.9) Fallacies and Pitfalls
- (ESaaS 2.10) Patterns, Architecture, and Long-Lived APIs

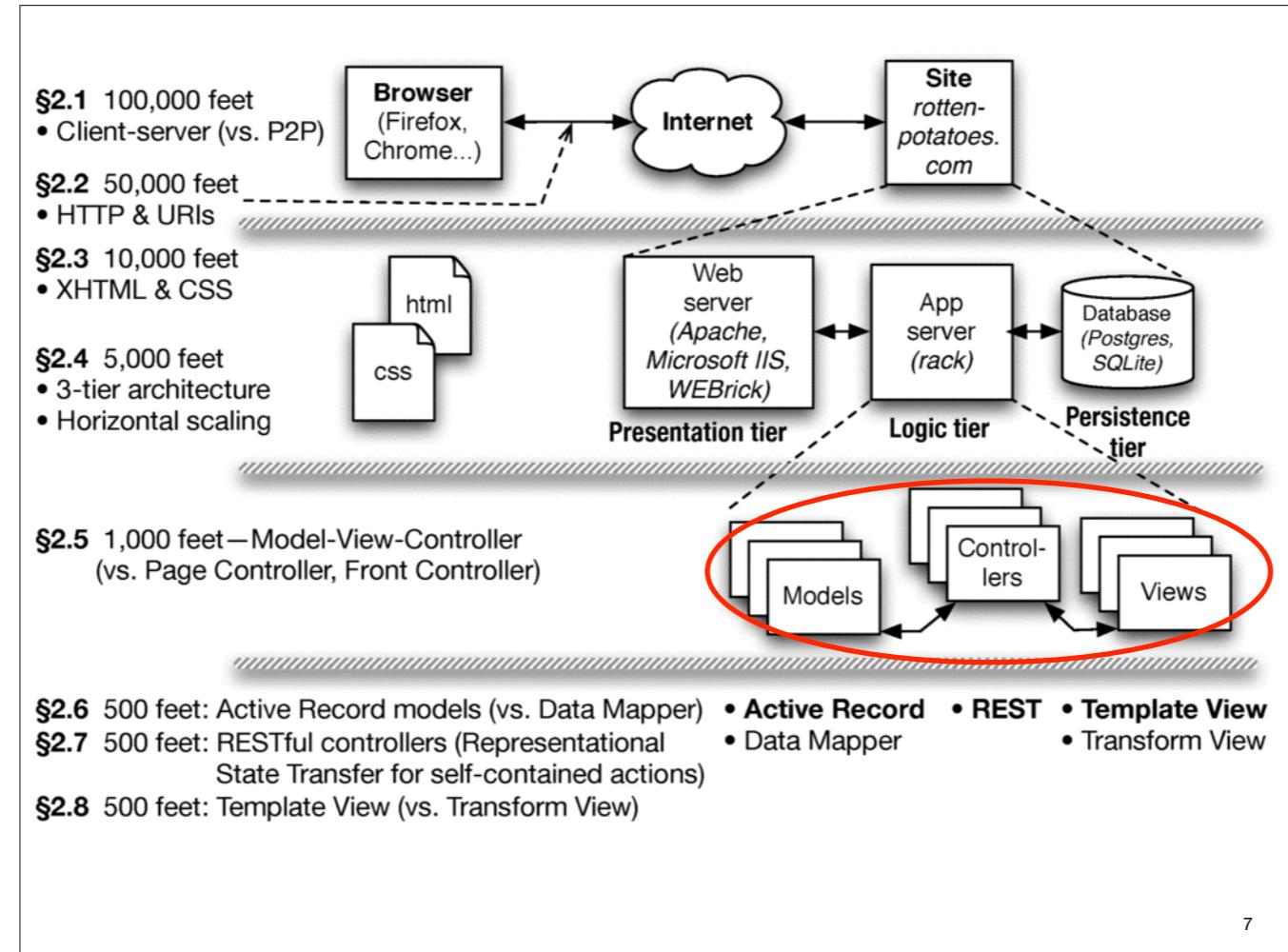


Engineering Software as a Service §2.5

# Para onde vão os frameworks?

---

- Existe **estrutura de aplicação comum** ...
- em aplicações interativas **voltadas para o usuário**...
- ... que poderiam **simplificar** o desenvolvimento de aplicativos, se os capturássemos em um framework?

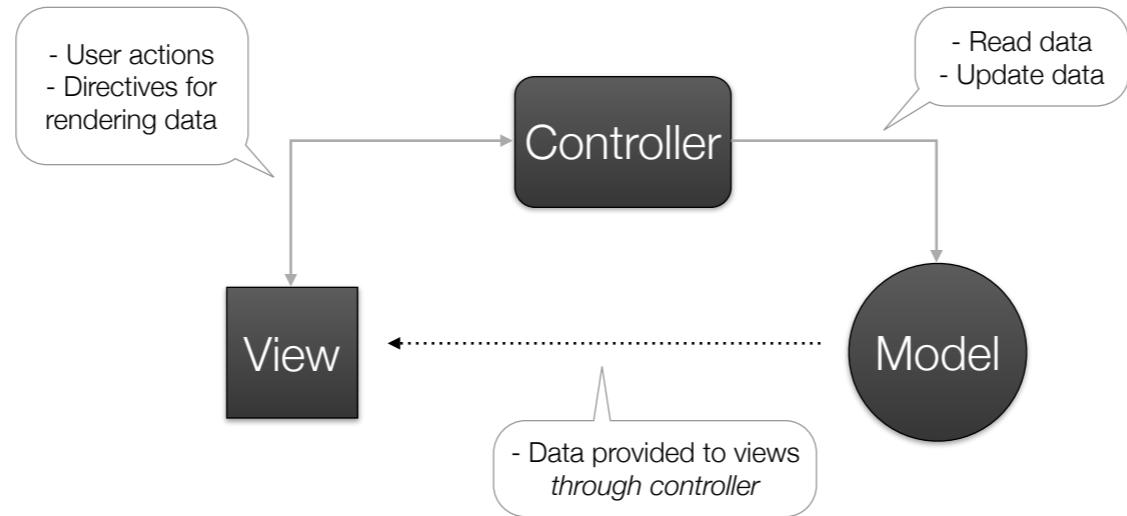


## Padrão arquitetural Model-View-Controller (MVC)

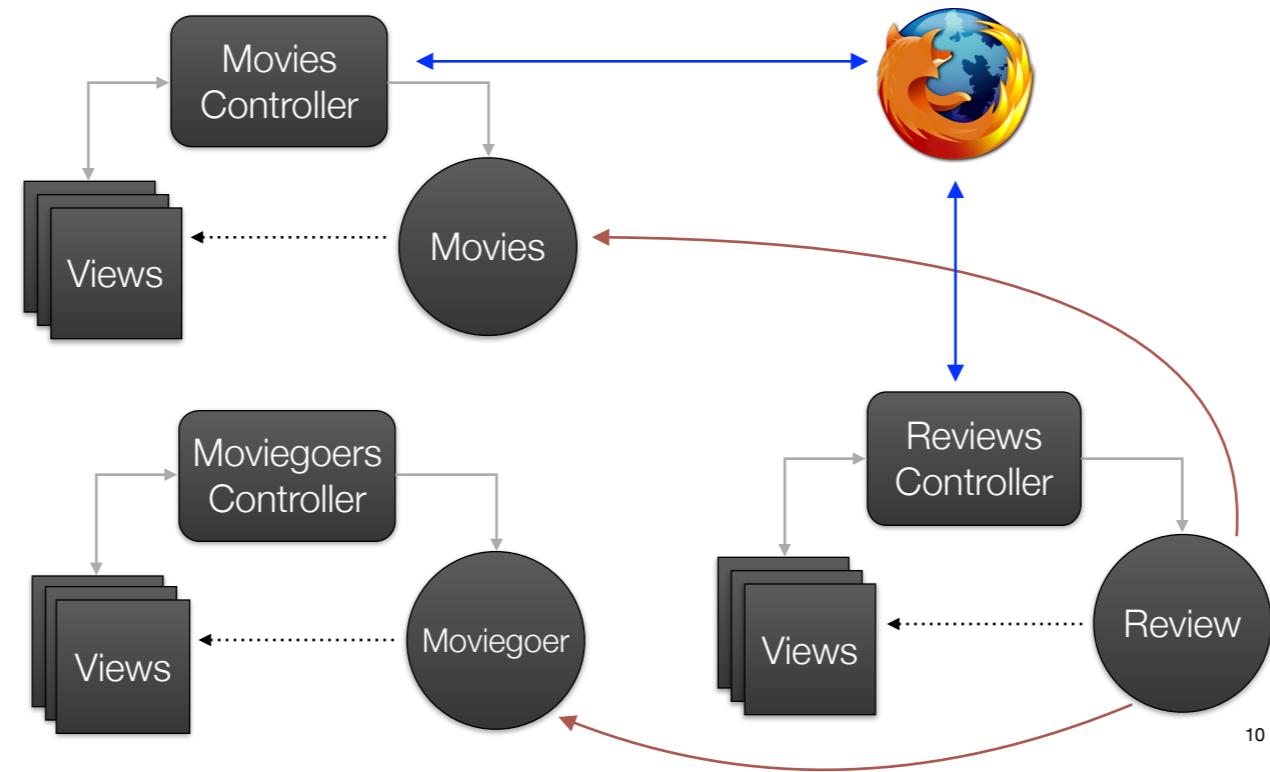
---

- Padrão arquitetural utilizado para ajudar na separação da camada de interface com o usuário das demais partes da aplicação
  - O model contém as classes cujas instâncias devem ser visualizadas e manipuladas
  - O view contém os objetos utilizados para renderizar a aparências dos dados do modelo na interface do usuário
  - O controller contém os objetos que controlam e tratam as interações do usuário com o view e o model.
  - O padrão de projeto Observable é normalmente utilizado para separar o model do view.

## Padrão arquitetural Model-View-Controller (MVC)

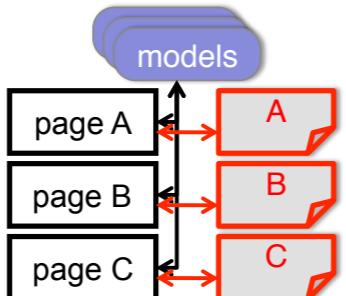


Cada entidade tem um model, controller e um conjunto de views

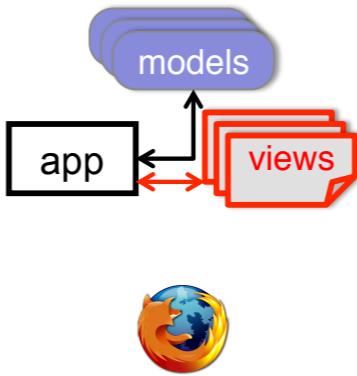


# Alternativas ao MVC

Page Controller  
(Ruby Sinatra)



Front Controller  
(J2EE servlet)



Template View  
(PHP)



Rails dá suporte a apps SaaS estruturadas como MVC, mas outra arquitetura pode ser uma melhor opção para outras apps.

# Pergunta

---

Qual alternativa **NÃO** é verdade sobre o padrão arquitetural Model-View-Controller (MVC)

- A. Nas apps SaaS na Web, controller actions e conteúdos da view são transmitidos usando HTTP
- B. Todas apps MVC possuem tanto uma parte “cliente” (ex. Browser Web) e a parte da “nuvem” (ex. App Rails na nuvem)
- C. MVC é somente uma das diversas possibilidades de se estruturar uma app SaaS
- D. Apps Peer-to-Peer podem ser estruturadas como MVC

# Pergunta

---

Qual alternativa **NÃO** é verdade sobre o padrão arquitetural Model-View-Controller (MVC)

- A. Nas apps SaaS na Web, controller actions e conteúdos da view são transmitidos usando HTTP
-  B. Todas apps MVC possuem tanto uma parte “cliente” (ex. Browser Web) e a parte da “nuvem” (ex. App Rails na nuvem)
- C. MVC é somente uma das diversas possibilidades de se estruturar uma app SaaS
- D. Apps Peer-to-Peer podem ser estruturadas como MVC



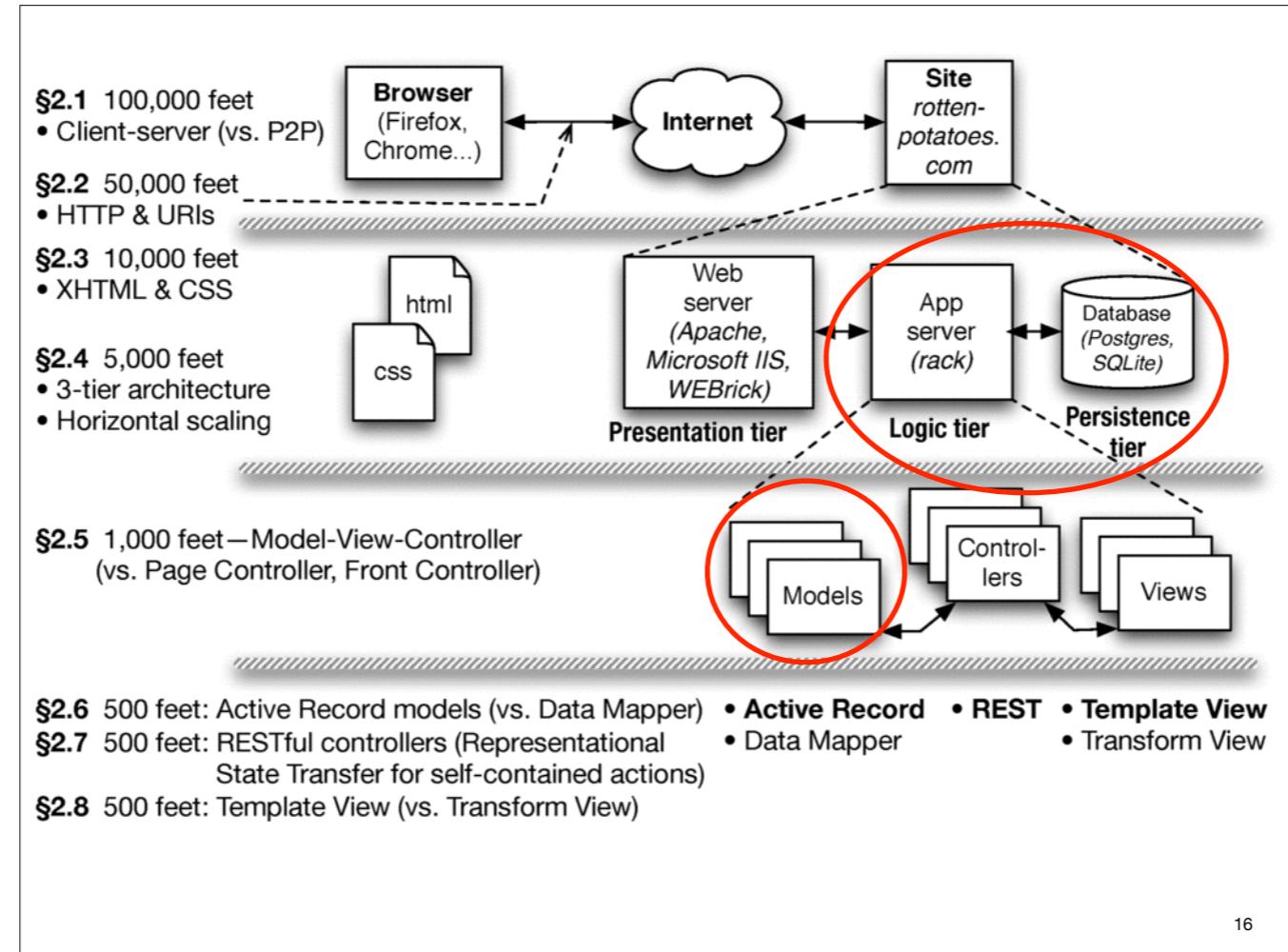
## Models, Databases, and Active Record

# Warm up

---

**Como deveríamos armazenar e recuperar estruturas de dados orientadas a registros?**

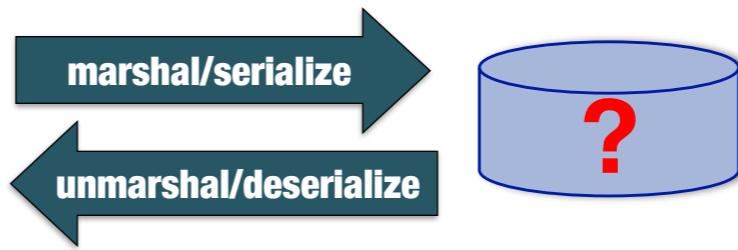
**Qual a relação entre dado armazenado e dado manipulado em uma linguagem de programação?**



# In-Memory vs. In-Storage objects

#<Movie:0x1295580>  
m.name, m.rating, ...

#<Movie:0x32ffe416>  
m.name, m.rating, ...



- Como representar objetos persistidos em armazenamento
  - Exemplo: Filmes e Avaliações
- Operações básicas no objeto: CRUD (Create, Read, Update, Delete)
- ActiveRecord: todo modelo conhece como fazer o CRUD em si mesmo, utilizando mecanismos comuns

# Modelos Rails Armazenam Dados em RDBMS

- Cada tipo de model recupera sua própria tabela no BD
  - Todas as linhas em uma tabela possuem estrutura idêntica
  - 1 linha na tabela == uma instância do modelo
  - Cada coluna armazena o valor de um atributo do modelo
  - Cada linha possui um valor único para primary key (por convenção, em Rails é um inteiro chamado ID)

<b>id</b>	<b>rating</b>	<b>title</b>	<b>release_date</b>
2	G	Gone With the Wind	1939-12-15
11	PG	Casablanca	1942-11-26
...	...	...	...
35	PG	Star Wars	1977-05-25

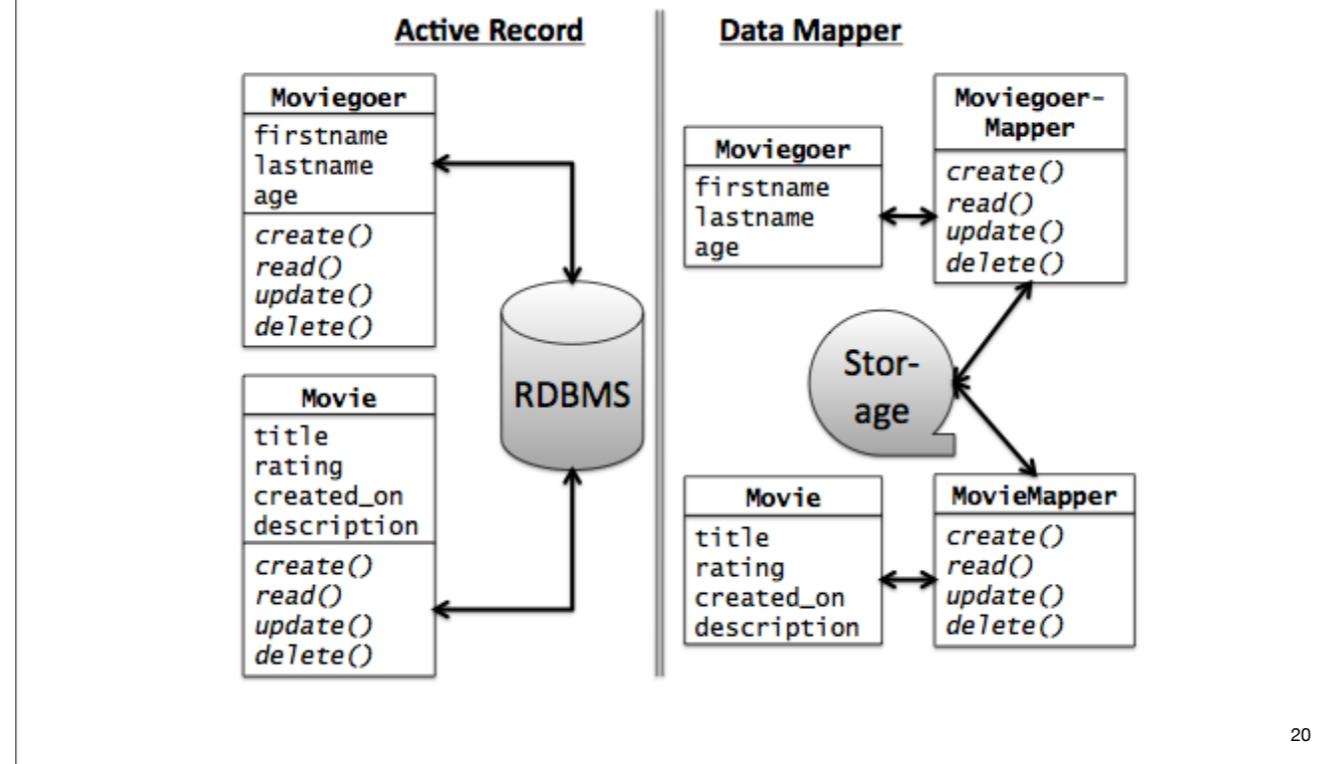
- Schema: Coleção de todas as tabelas e suas estruturas

# Alternativa: DataMapper

---

- Data Mapper associa mapeamentos separados para cada modelo
  - Ideia: manter mapeamentos independentes de armazenamentos de dados => trabalha melhor com mais tipos de BDs
  - Usado pelo Google AppEngine
  - Cons: não pode explorar os recursos de RDBMS para simplificar as consultas e relações complexas

# Active Record vs Data Mapper



# Pergunta

---

- Qual alternativa **NÃO** é verdadeira sobre Modelos no MVC:
  - A. As ações de CRUD são somente aplicadas a modelos que são apoiados por um banco de dados que dá suporte ao Active Record
  - B. Parte do trabalho do Model é converter a representação dos dados entre o In-memory e o armazenamento
  - C. Embora os dados do Model sejam exibidos pela View, a interação direta dele é com os controladores
  - D. Embora o Data Mapper não utilize bancos de dados relacionais, é uma forma válida de implementar a Model

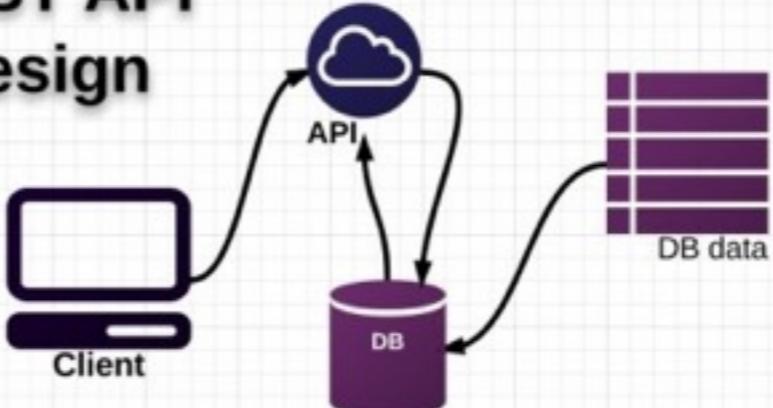
# Pergunta

---

- Qual alternativa **NÃO** é verdadeira sobre Modelos no MVC:  
 A. As ações de CRUD são somente aplicadas a modelos que são apoiados por um banco de dados que dá suporte ao Active Record  
B. Parte do trabalho do Model é converter a representação dos dados entre o In-memory e o armazenamento  
C. Embora os dados do Model sejam exibidos pela View, a interação direta dele é com os controladores  
D. Embora o Data Mapper não utilize bancos de dados relacionais, é uma forma válida de implementar a Model

# REST API Design

**GET** /tasks - display all tasks  
**POST** /tasks - create a new task  
**GET** /tasks/{id} - display a task by ID  
**PUT** /tasks/{id} - update a task by ID  
**DELETE** /tasks/{id} - delete a task by ID

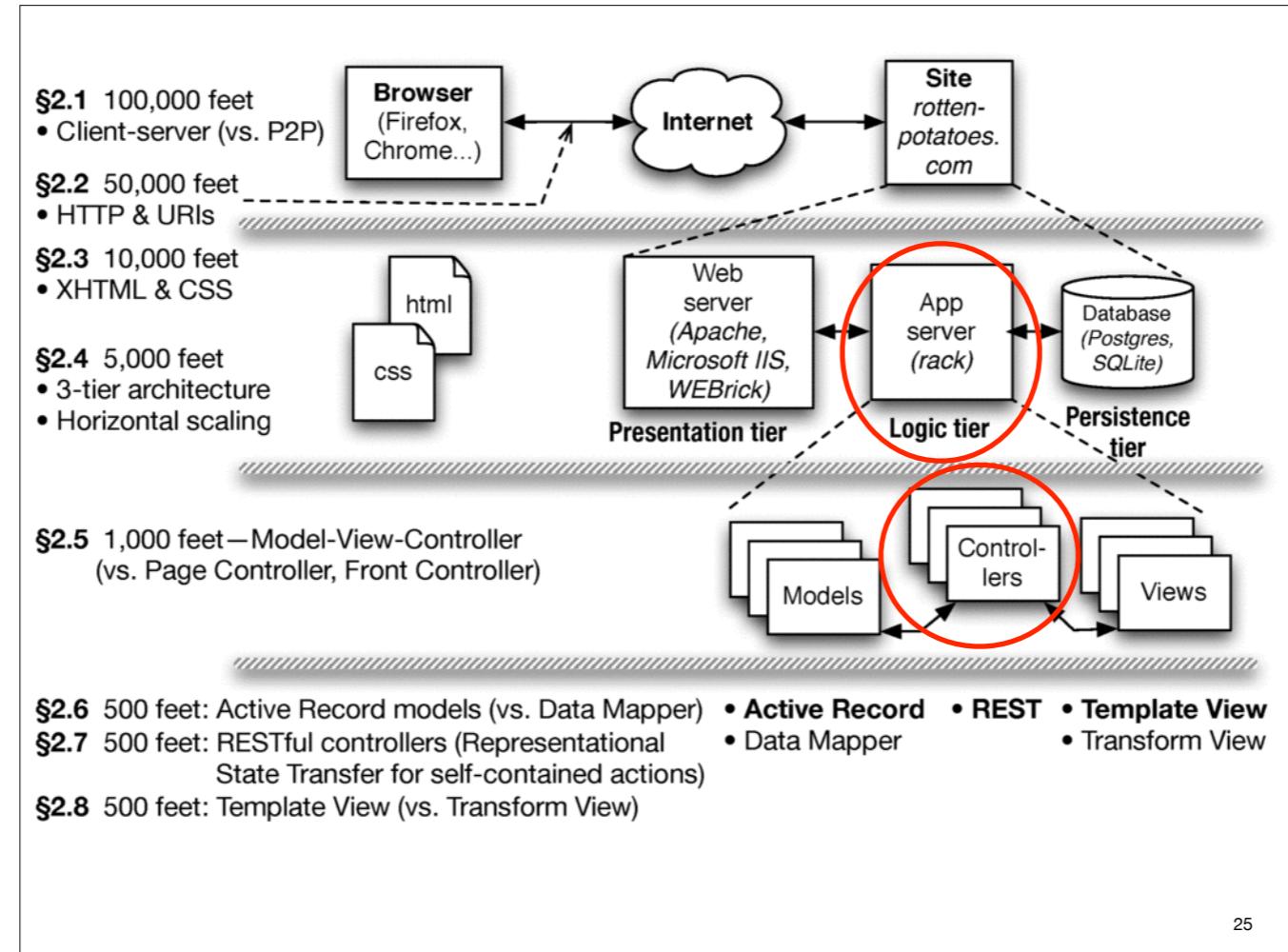


Controllers, Routes, and  
RESTfulness

# Warm up

---

**Quais decisões de projeto vão permitir que nosso aplicativo dê suporte a Arquitetura Orientada a Serviço?**



## REST (Representational State Transfer) — R. Fielding, 2000

---

- Ideia: Requisições independentes especificam em qual recurso operar e o que fazer com ele
  - Tese de PhD de Roy Fielding
  - Wikipedia: “a post hoc description of the features that made the Web successful”
- Um serviço (no contexto de SOA) cujo as operações são como essa, são serviços RESTFul
- Idealmente, RESTFul URI nomeia as operações
- Vamos ver um exemplo...

<http://pastebin.com/edF2NzCF>

# Rotas

---

- No MVC, cada interação que o usuário pode fazer é tratada por uma ação de um controlador (controller action)
  - Ruby possui um método que trata esta interação
- Uma rota mapeia <HTTP method, URI> para uma ação de um controlador

Route	Action
GET /movies/3	Show info about movie whose ID=3
POST /movies	Create new movie from attached form data
PUT /movies/5	Update movie ID 5 from attached form data
DELETE /movies/5	Delete movie whose ID=5

## Breve Introdução ao subsistema de rotas do Rails

- dispara `<method, URI>` para a ação correta do controlador
- provê métodos auxiliares que geram um par `<method, URI>` dada uma ação do controlador
- analisa os parâmetros da consulta de ambas URI e cria a submissão em um hash
- atalhos embutidos para geração de todas as rotas de CRUD (embora a maioria dos apps também tenham outras rotas)

**rake routes**

I	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"}

28

## GET /movies/3/edit HTTP/1.0

- Encontre rota:

- `GET /movies/:id/edit  
{:action=>"edit", :controller=>"movies"}`

- Analise parâmetros coringas (wildcard): `params[:id] = "3"`

- Envie para o método `edit` em `movies_controller.rb`

- Para incluir a URI na visão gerada que irá submeter o formulário para a ação de atualização no controlador com `params[:id]==3`, chama o auxiliar:

- `update_movie_path(3) # => PUT /movies/3`

```
rake routes
```

I	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 alternativa **NÃO** é verdadeira sobre rotas Rails RESTFul e recursos com os quais elas se relacionam

- A. Um recurso pode ser o conteúdo existente ou uma solicitação para modificar alguma coisa.
- B. Cada rota deve, eventualmente, desencadear uma ação do controlador.
- C. Um conjunto comum de ações RESTful são as ações CRUD em modelos.
- D. A rota contém sempre um ou mais parâmetros, como por exemplo : ID, para identificar o recurso

# Pergunta

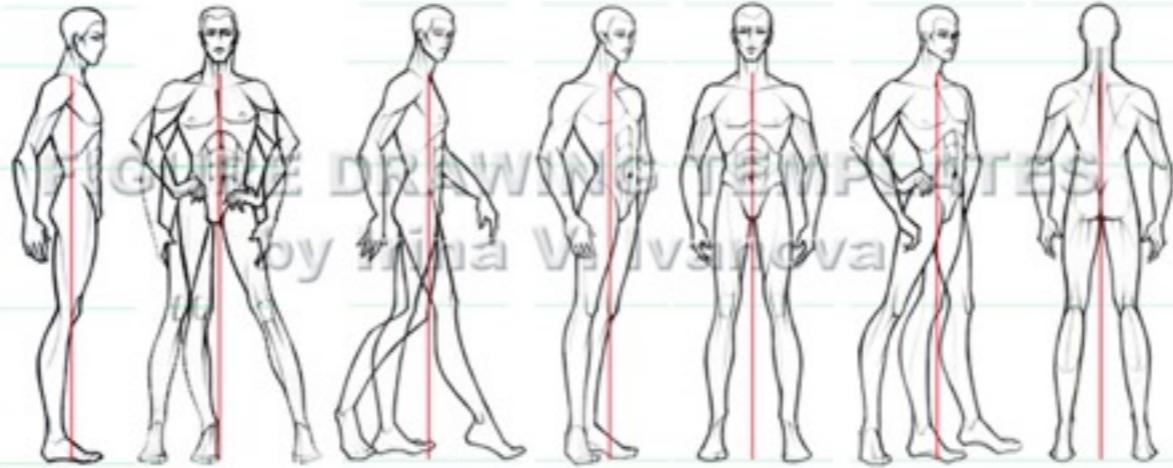
---

Qual alternativa **NÃO** é verdadeira sobre rotas Rails RESTFul e recursos com os quais elas se relacionam

- A. Um recurso pode ser o conteúdo existente ou uma solicitação para modificar alguma coisa.
- B. Cada rota deve, eventualmente, desencadear uma ação do controlador.
- C. Um conjunto comum de ações RESTful são as ações CRUD em modelos.
- D. A rota contém sempre um ou mais parâmetros, como por exemplo : ID, para identificar o recurso



## MALE FIGURE



[www.FigureDrawingTemplates.com](http://www.FigureDrawingTemplates.com)

Ivanova, Irina V. Figure Drawing Templates Set #2 Standard Man's Figure ISBN-10: 0-9843560-0-2 ISBN-13: 978-0-9843560-0-3

## Template Views and Haml



32

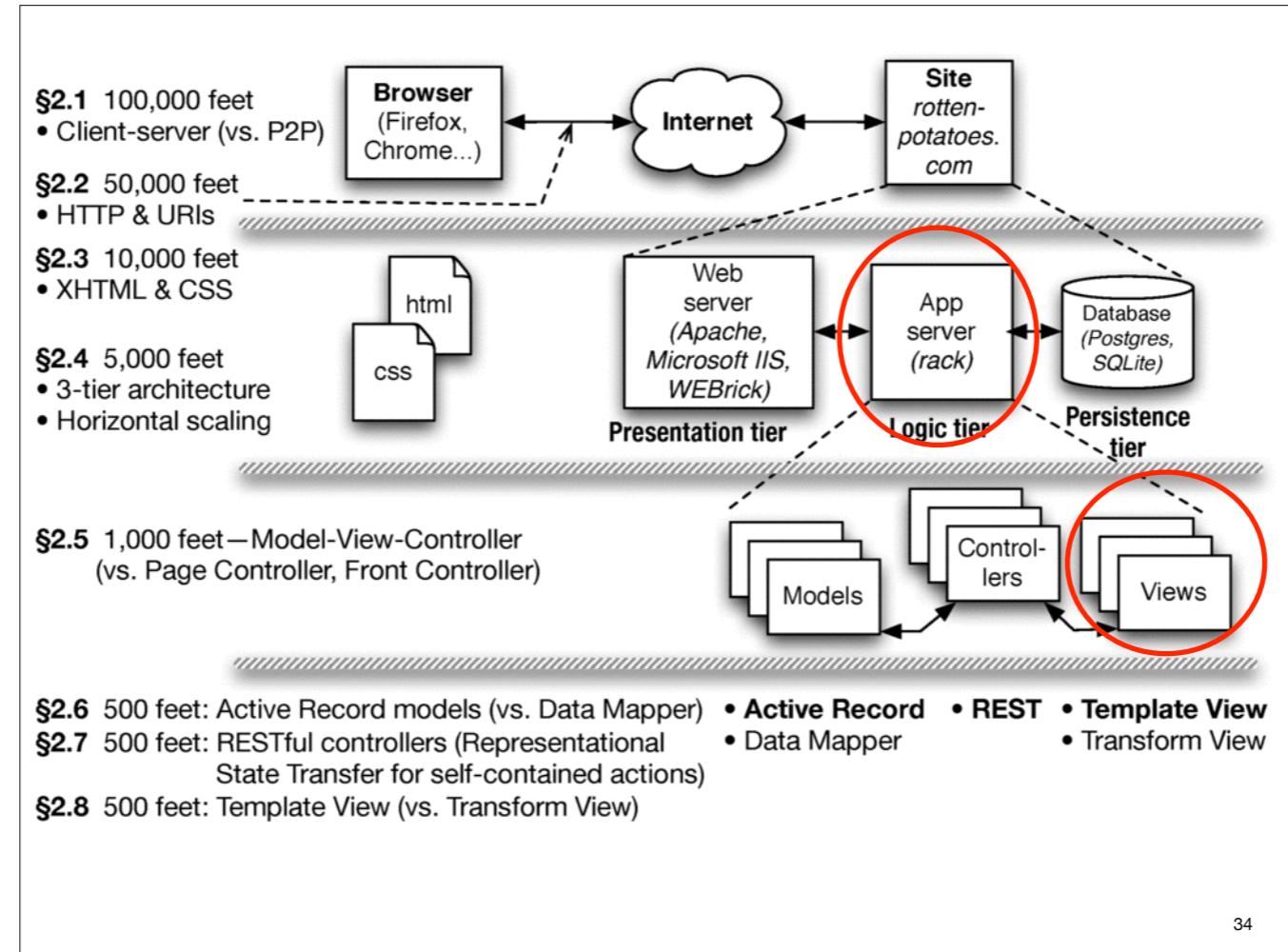
Engineering Software as a Service §2.8

# Warm up

---

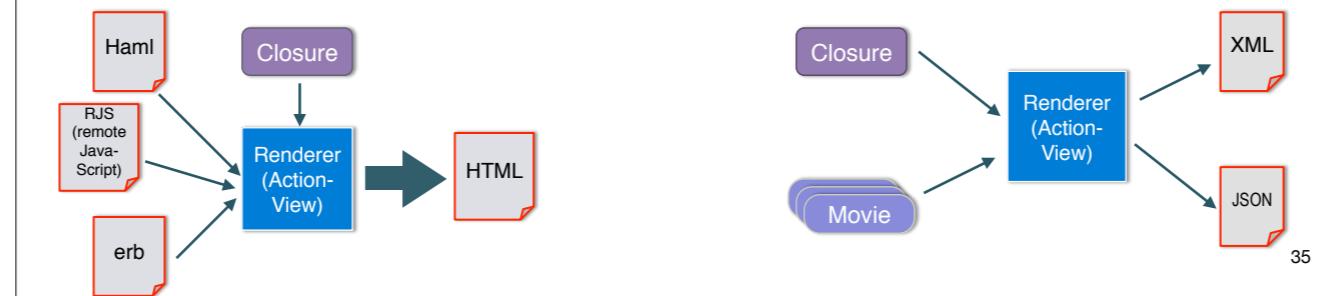
**HTML é como devemos apresentar conteúdo para os browsers...**

**... mas qual processo pela qual a resposta da nossa app se torna HTML?**



# Padrão de Projeto Template View

- Visão consiste em marcação com interpolação selecionada para acontecer em tempo de execução
  - Usualmente, valores de variáveis ou resultados de avaliações de pequenos bits de código
  - “Nos tempos de outrora”, essa foi a app: PHP
- Alternativa: Transform View



35

# Haml é HTML de dieta

```
%h1.pagename All Movies
%table#movies
  %thead
    %tr
      %th Movie Title
      %th Release Date
      %th More Info
  %tbody
    - @movies.each do |movie|
      %tr
        %td= movie.title
        %td= movie.release_date
        %td= link_to "More on #{movie.title}", |
          movie_path(movie) |
      = link_to 'Add new movie', new_movie_path
```

36

no open/close tags: use indentation

tag entirely on single line, or use indentation to indicate nesting

how to specify ID & class

code that is executed but NOT substituted

note lack of 'end' for 'do': implied by indentation

code that is executed and substituted

multi-line – deliberately awkward

# Não coloque código nas suas Views

---

- Sintaticamente, você pode colocar qualquer código nas views
- Porém, MVC advoga por views magras & controladores
  - Haml torna deliberadamente complicado inserir trechos de código
- Métodos Helpers (que “embelezam” objetos para serem incluídos nas views) possuem seu espaço próprio em app Rails
- Alternativa ao Haml: html.erb (Embedded Rubt) templates, e outros tipo PHP

# Pergunta

---

O que acontece se você inserir código nas suas views em Rails que acessam diretamente o modelo?

- A. Ele vai funcionar, mas não é uma boa prática e viola as diretrizes do MVC
- B. Ele vai funcionar quando desenvolver com um BD “toy”, mas não em produção
- C. Não vai funcionar porque as Views não podem se comunicar diretamente com os Modelos
- D. O comportamento vai variar de acordo com a app

# Pergunta

---

O que acontece se você inserir código nas suas views em Rails que acessam diretamente o modelo?

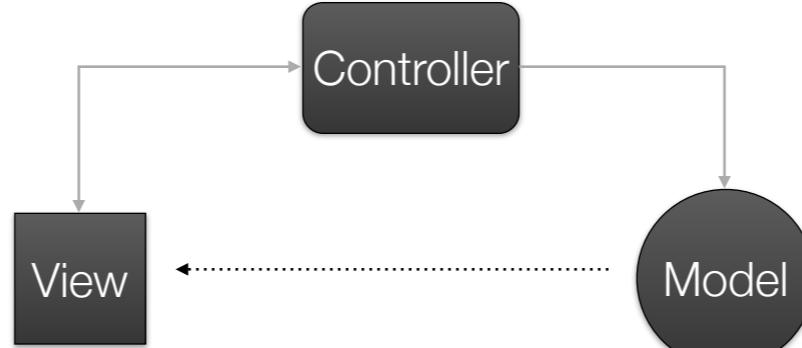
- A. Ele vai funcionar, mas não é uma boa prática e viola as diretrizes do MVC
- B. Ele vai funcionar quando desenvolver com um BD “toy”, mas não em produção
- C. Não vai funcionar porque as Views não podem se comunicar diretamente com os Modelos
- D. O comportamento vai variar de acordo com a app



## Conclusão e Reflexões: Arquitetura SaaS

# The big picture

- URI's, HTTP, TCP/IP stack
- REST & RESTful routes



- HTML & CSS
- XML & XPath

- Databases & migrations
- CRUD

41

# “Rails doesn’t scale”

---

- Escalabilidade é uma preocupação arquitetural – não está confinada a uma linguagem ou framework
- As camadas stateless da arquitetura 3-tier escalam
  - Com computação em nuvem, só precisa se preocupar com as constantes
- RDBMS tradicionais não escalam
- Várias soluções combinam armazenamento relacional e não-relacional (NoSQL) escalam melhor
  - Data Mapper trabalha bem com elas
- Uso inteligente de caching podem melhorar consideravelmente os fatores das constantes

# Frameworks, Apps, Design patterns

---

- Muitos padrões de projeto até o momento, muito mais ainda estão por vir
- Em 1995, era o “velho oeste”: os grandes Web sites eram minicomputadores !!!
  - Nada de 3-tier ou cloud
- Boas práticas (padrões) foram “extraídas” da experiência e capturadas em frameworks
- Porém APIs trancenderam isso: 1969 protocolos + 1960s linguagem de marcação + 1990 browser + 1992 Web server  
-> funciona hoje em dia!

# Arquitetura é sobre alternativas

Pattern we're using	Alternatives
Client-Server	Peer-to-Peer
Shared-nothing (cloud computing)	Symmetric multiprocessor, shared global address space
Model-View-Controller	Page controller, Front controller, Template view
Active Record	Data Mapper
RESTful URIs (all state affecting request is explicit)	Same URI does different things depending on internal state

**Como você irá trabalhar em outros aplicativos SaaS para além deste curso, você pode se encontrar considerando diferentes opções de arquitetura e questionar as escolhas que estão sendo feitas.**

# Conclusões: Arquitetura & Rails

---

- Model-View-Controller é um padrão arquitetural bem conhecido para estruturar apps
- Rails codifica a estrutura de apps SaaS como MVC
- Zend (PHP), Django (Python), Grails (Groovy), Play! (java)...
- Views são Haml com código Ruby embarcado, transformado em HTML quando enviado ao browser
- Models são armazenados em tabelas de um RDBMS, acessados utilizando Active Record (ou Data Mapper)
- Controllers unem views e models através de rotas e código nos métodos controladores

# Pergunta

---

Outros fatores sendo iguais, qual alternativa **NÃO** é verdadeira sobre escalabilidade em SaaS?

- A. Clusters shared-nothing escalam melhor que sistemas construídos em mainframes
- B. RDBMS escalam melhor do que banco de dados “NoSQL”
- C. A linguagem de programação utilizada (Ruby, Java, etc) não é o principal fator em termos de escalabilidade
- D. Escalabilidade pode ser impedida por qualquer parte do app que se torne um gargalo

# Pergunta

---

Outros fatores sendo iguais, qual alternativa **NÃO** é verdadeira sobre escalabilidade em SaaS?

- A. Clusters shared-nothing escalam melhor que sistemas construídos em mainframes
-  B. RDBMS escalam melhor do que banco de dados “NoSQL”
- C. A linguagem de programação utilizada (Ruby, Java, etc) não é o principal fator em termos de escalabilidade
- D. Escalabilidade pode ser impedida por qualquer parte do app que se torne um gargalo