1 começando

Rails Muito Veloz

Quer fazer o seu desenvolvimento de aplicação web alçar voo? Então você precisa conhecer o **Rails**. Rails é o **framework de desenvolvimento** mais **rápido** e **legal** na cidade, que lhe permite **desenvolver aplicações web funcionais de forma completa** mais rápido do que você pensa. É muito simples começar, tudo o que você precisa fazer é **instalar o Rails** e virar as páginas deste livro. Antes que se dê conta, já estará **quilômetros à frente na competição**.

Sexta, 9 da manhã

O primeiro e-mail que você abriu é de um velho amigo que está com problema:

Oi – como você está?
Preciso de um grande favor! Recorda-se daquela aplicação de venda de ingressos, na qual eu disse que estava trabalhando? Não está funcionando direito. Já passamos semanas trabalhando nisso! A equipe está tendo sérios problemas.

Você acha que consegue criar a aplicação para nós?
Nós precisamos de um site que possa:
Listar todos os ingressos vendidos;
Criar uma nova venda de ingressos;
Ler e exibir um ingresso único;
Atualizar as informações de venda;
Excluir uma venda de ingresso.

Sei que parece ter muitas funções, mas o meu chefe disse que esta lista é a quantidade mínima de características que o site precisa ter – e você sabe que ele não é uma pessoa fácil de convencer! Aqui está a estrutura de dados:

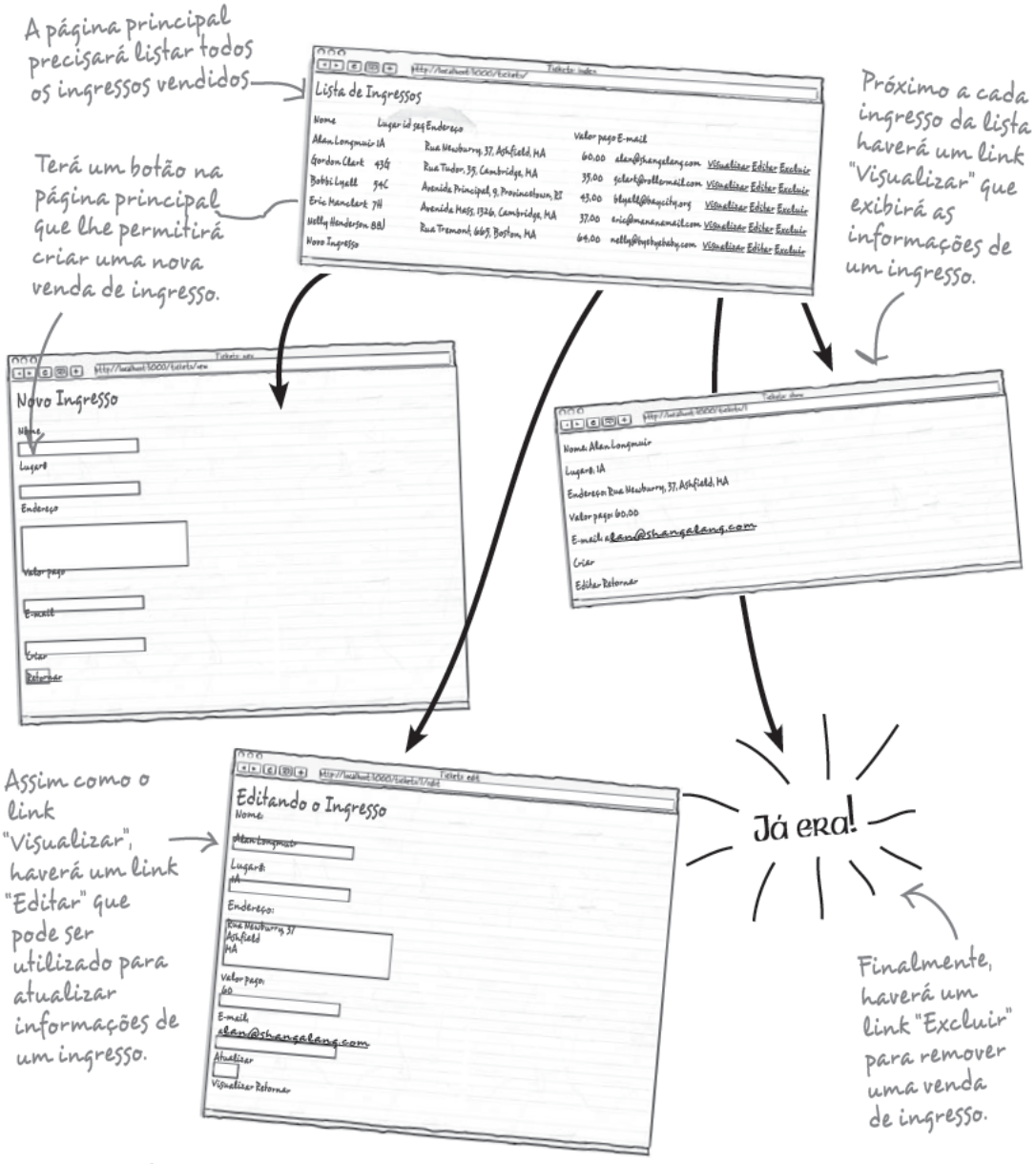
Ingresso:
name – nome do comprador (string)
seat_id_seq – o número do lugar p.ex. E14 (string)
address – endereço do comprador (string longa)
price_paid – valor do ingresso (decimal)
email_adress – e-mail do comprador (string)

Também anexei os rascunhos das páginas para que você saiba o que queremos. Ah! Precisamos de tudo isso até segunda, ou estarei no olho da rua. Ajude-me!

O sistema foi desenhado para ser utilizado pelos funcionários do caixa na arena do show. O banco de dados será restaurado a cada show, assim ele só precisará gravar as informações de um show por vez. Você acha que pode me ajudar?

A aplicação precisa fazer muitas coisas

Aqui estão os rascunhos das páginas web. Como eles se encaixam nos requisitos do sistema?



PODER DO CÉREBRO

De quais tipos de software você precisará para criar e executar a aplicação?

Então, o que precisamos para a aplicação?

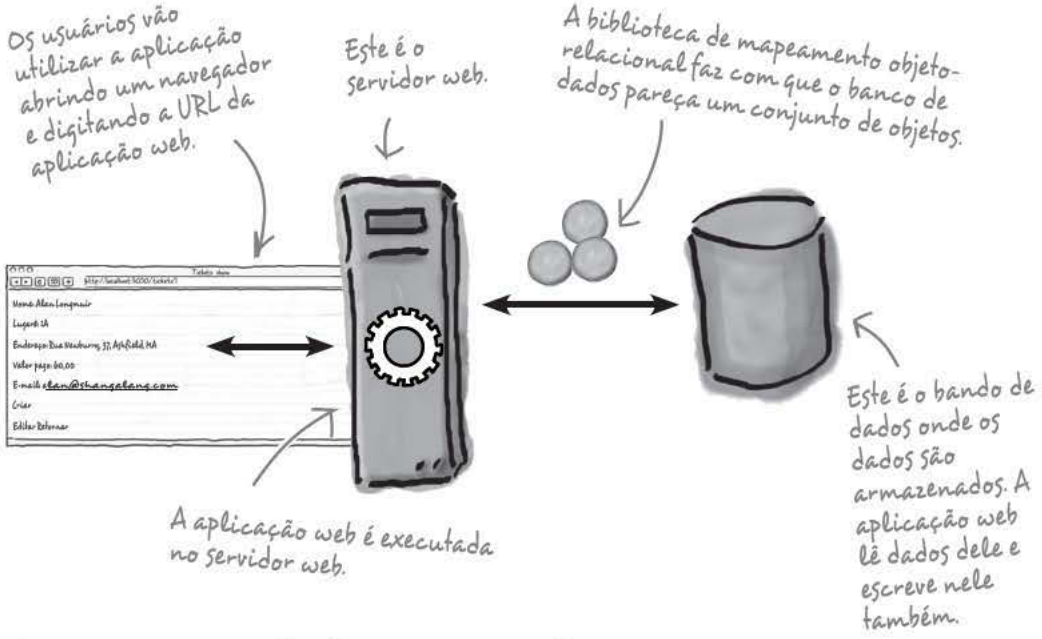
Há muitas coisas de que precisamos para executar uma aplicação no servidor da arena de shows. Precisamos de:

1. **Um framework de aplicação.**
   Um conjunto de código pré-escrito para que forneça a fundação para a

2. **Um sistema de banco de dados.**
   Algum tipo de banco de dados onde possamos armazenar os dados.

3. **Um servidor web.**
   Algum lugar para executar a aplicação web.

4. **Uma biblioteca de mapeamento objeto-relacional.**
   Para simplificar o acesso ao banco de dados, muitas aplicações web utilizam atualmente uma biblioteca de mapeamento objeto-relacional para converter registros do banco de dados em objetos.



Então, como o Rails nos ajuda?

Independente da linguagem quevocê usa para desenvolver seu código, provavelmente precisará destes três recursos para ter sua aplicação totalmente implantada. Uma das melhores qualidades do Rails é que ele contém tudo o que você precisa para seu software – *tudo agrupado gratuitamente.*

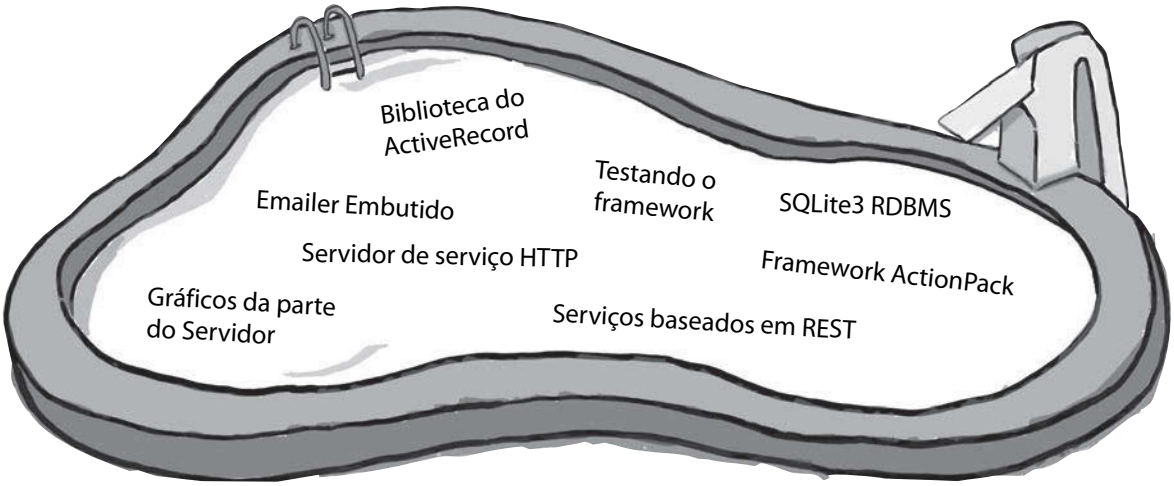
Vamos ver como isso funciona.

Quebra-Cabeça

Há muitas partes e características internas do Rails. O seu trabalho é adivinhar de quais delas, listadas dentro da piscina, precisamos para aplicação web e, então, colocá-las nas linhas em branco abaixo. Você não precisará de todas elas.

1. .....
2. .....
3. .....
4. .....

Lembrete: cada coisa dentro piscina só pode ser utilizada uma vez!



O Rails serve para aplicações de banco de dados como um sistema de vendas de ingressos

Muitas aplicações têm um banco de dados como o coração da mesma. Elas existem principalmente para que os usuários consigam acessar e modificar os conteúdos do banco de dados ***sem*** utilizar SQL de forma direta.

Quais os problemas que precisam ser resolvidos ao conectar o banco de dados a uma aplicação web?

Bem, a aplicação web precisará permitir ao usuário acessar e modificar dados, então, o Rails inclui um **framework de aplicação** chamado de **ActionPack**, que lhe ajudará a gerar dados e páginas interativas.

Em seguida, as aplicações web precisam ser executadas em um **servidor de web** para exibir essas páginas, e o Rails vem com servidor um interno.

Depois, você precisará de um **banco de dados**. O Rails cria aplicações que são configuradas para funcionar com um banco de dados integrado ao **SQLite3**.

Outra coisa de que você precisará é uma **biblioteca de mapeamento objeto-relacional** e o Rails fornece uma, chamada **ActiveRecord**. Isto faz com o que o seu banco de dados se pareça com uma coleção de simples *objetos Ruby*.

Além de tudo isto, o Rails também inclui um conjunto de ferramentas de script para ajudá-lo a gerenciar a aplicação. Então, se você estiver criando uma aplicação web focada em banco de dados, descobrirá que o **Rails fornece tudo o que você precisa.**

Quebra-Cabeça Solução

Há muitas partes e características internas do Rails. O seu trabalho é adivinhar de quais delas, listadas dentro da piscina, precisamos para aplicação web e, então, colocá-las nas linhas em branco acima. Você não precisará de todas elas.

Framework ActionPack

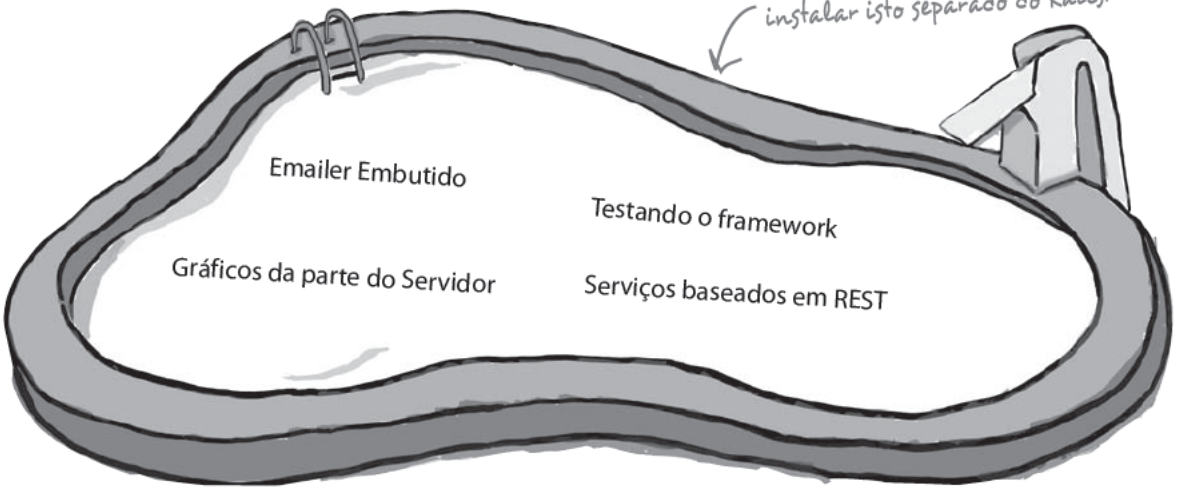
SQLite3 RDBMS

Servidor de serviço HTTP

Biblioteca do ActiveRecord

Em alguns sistemas operacionais, você precisará instalar isto separado do Rails.

Com o Rails você também tem tudo isso, porém nesta aplicação você não precisa deles. Em alguns sistemas operacionais você precisa instalar isto separado do Rails.



Você pode criar uma nova aplicação com o comando rails

Então, como começar a utilizar o Rails?

Criar uma aplicação web para a venda de tickets é, na verdade, muito simples com Rails.

Tudo o que você precisa é abrir o prompt de comando ou o terminal do windows e digitar **rails tickets**, onde tickets é o nome da aplicação que você quer criar.

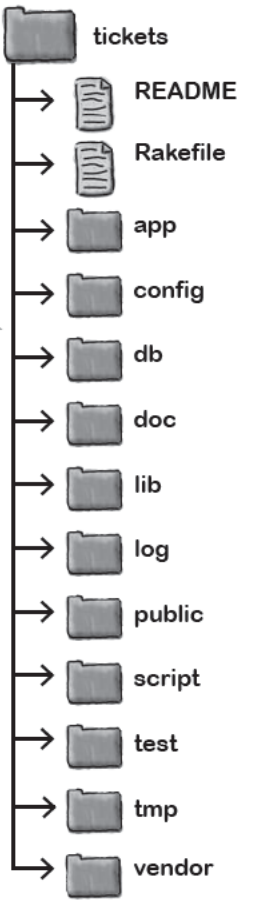

```
File Edit Window Help RailsRules
> rails tickets
```

Apenas digite "rails tickets" no prompt de comando.

Então, o que isso faz?

Digitar `rails tickets` gera, de forma bem esperta, uma aplicação web em um novo diretório chamado "tickets" (ingressos). Não é só isso, dentro do diretório de ingressos, o Rails gera todo um conjunto de outros diretórios e arquivos que formam a estrutura básica da nova aplicação.

Isto significa que você efetivamente criou uma aplicação web básica inteira com apenas um pequeno comando.



O Rails gera todo um conjunto de arquivos e pastas para você com apenas um comando. Esta é a estrutura de toda a aplicação web.

O Rails gera muitos arquivos e pastas, mas não se preocupe.

Todos estão lá por um motivo e você entenderá o que todos eles fazem até o final do livro.

Test Drive

Como a aplicação que você acabou de criar é uma aplicação *web*, você precisa iniciar o servidor de web interno para ver esta aplicação executando.

No prompt de comando ou terminal, mude para dentro da pasta tickets e digite **`ruby script/server.`**

Este é o console. Você consegue acessá-la pelo comando prompt no Windows, ou por um terminal no Linux ou Mac.

Entre na pasta da aplicação que foi criada...

...e inicie o servidor de web.

```
File  Edit  Window  Help
> cd tickets
> ruby script/server
```

Algumas mensagens vão aparecer na tela, isso confirmará que o servidor está executando. Agora você consegue ver a *home page* padrão ao abrir o navegador no endereço:

`http://localhost:3000/`

Esta é a home page padrão do servidor de web.

Dicas Geek

O Rails inicia o seu servidor na porta 3000 como padrão. Se você quiser utilizar outra porta, por exemplo, 8000, execute

```
ruby script/server
-p 8000
```

Agora você precisa adicionar o seu próprio código à aplicação padrão

O Rails cria a estrutura básica da sua aplicação inicial, mas você ainda precisa adicionar o código que é específico para o que **você** quer.

Como a aplicação de cada pessoa é diferente, mas o Rails tem algumas ferramentas ou artimanhas que o tornam mais fácil para a criação de um código customizado?

Bem, para falar a verdade, tem sim. Você percebeu como o Rails criou toda a estrutura do arquivo para você, quase como se ele soubesse do que você precisaria?

Isso ocorre porque as aplicações do Rails seguem firmemente a nomeação de convenções.

Princípio Rails: Convenção Sobre Configuração

Aplicações Rails sempre seguem convenções

Todas as aplicações Rails seguem a mesma estrutura básica de arquivos e utilizam nomes consistentes para tudo. Isto torna a aplicação mais fácil de entender, mas também significa que as ferramentas internas do Rails entenderão como as suas aplicações funcionam.

Por que isto é importante? Bom, se as ferramentas souberem como a sua aplicação está estruturada, você pode utilizá-las para automatizar muitas tarefas de codificação. Desta forma, o Rails consegue utilizar convenções para gerar o código para você, sem que você tenha que configurar a sua aplicação web. Em outras palavras, o Rails segue ***convenção sobre configuração.***

Vamos olhar uma das ferramentas mais poderosas do Rails: o scaffold.

não existem Perguntas Idiotas

P: Falando de Ruby e Rails. Qual a diferença entre os dois?

R: *Ruby é uma linguagem de programação. Rails é uma coleção de scripts Ruby. Assim, o servidor web, o framework de aplicação ActionPack e os scripts de ferramenta embutidos são todos scripts Ruby, portanto, parte do Rails.*

P: Como edito a página principal do meu site?

R: *usando o arquivo* `HTML` *no arquivo* `public/index.html`*, abaixo do diretório da aplicação. O diretório público contém todo o conteúdo estático para a aplicação.*

P: E se eu quiser usar outro servidor de web? Como faço?

R: *Faz sentido utilizar o servidor web embutido durante o desenvolvimento. Se você quiser, pode trocar o servidor atual da sua aplicação por outro servidor.*

P: Faz diferença a pasta em que estou ao executar **`ruby script/server`**?

R: *Sim, claro. Você precisa estar na pasta que contém a sua aplicação web.*

P: Isto é o que compila o meu código?

R: *Ruby é uma linguagem interpretada, como JavaScript. Isto significa que não é necessário compilação. Você pode alterar o seu código e executá-lo imediatamente.*

Scaffold e o código GERADO

O que a sua aplicação precisa fazer? Vamos dar uma olhada no e-mail de novo:

Este é o mesmo e-mail que vimos antes.

Oi – como você está?

Preciso de um grande favor! Recorda-se daquela aplicação de venda de ingressos, na qual eu disse que estava trabalhando? Não está funcionando direito. Já passamos semanas trabalhando nisso! A equipe está tendo sérios problemas.

Você acha que consegue criar a aplicação para nós?

- *Nós precisamos de um site que possa:*
- *Listar todos os ingressos vendidos;*
- *Criar uma nova venda de ingressos;*
- *Ler e exibir um ingresso único;*
- *Atualizar as informações de venda;*
- *Excluir uma venda de ingresso*

A aplicação web precisa desempenhar todas estas funções. Precisa conseguir criar, ler, atualizar e excluir dados.

Sei que parece ter muitas funções, mas o meu chefe disse que esta lista é a quantidade mínima de características que precisa ter – e você sabe que ele não é uma pessoa fácil de convencer! Aqui está a estrutura de dados:

A operação precisa agir nesta estrutura de dados do ingresso.

Ingresso:

name – nome do comprador (string)
seat_id_seq – o número do lugar e.x. E14 (string)
address – endereço do comprador (string longa)
price_paid – valor do ingresso (decimal)
email_adress – e-mail do comprador (string)

Também anexei os rascunhos das páginas para que você saiba o que queremos. Ah – e precisamos de tudo isso até segunda, ou eu estarei no olho da rua. Ajude-me!

Então, precisamos criar páginas web que nos permitam ***Criar***, ***Ler***, ***Atualizar*** e ***Excluir*** ingressos. As letras iniciais em inglês de cada operação são ***C***, ***R***, ***U*** e ***D*** (*Create*, *Read*, *Update* e *Delete*, respectivamente), por isso são conhecidas como ***operações CRUD***. São operações muito comuns nas aplicações de bancos de dados – tão comuns que o Rails tem um jeito de gerar, de forma mais rápida, todo o código e todas as páginas de que você precisa. Ele faz tudo isso utilizando o comando ***scaffold***.

Imãs de Geladeira

Há um comando simples que você pode utilizar no console para gerar um código usando o scaffold. Veja se você pode organizar os ímãs para completar este comando.

```
script/generate ruby _____ ticket name: _____

_____ : _____ _____ : _____

_____ : _____ _____ : _____
```

string

seat_id_seq

scaffold

string

price_paid

decimal

text

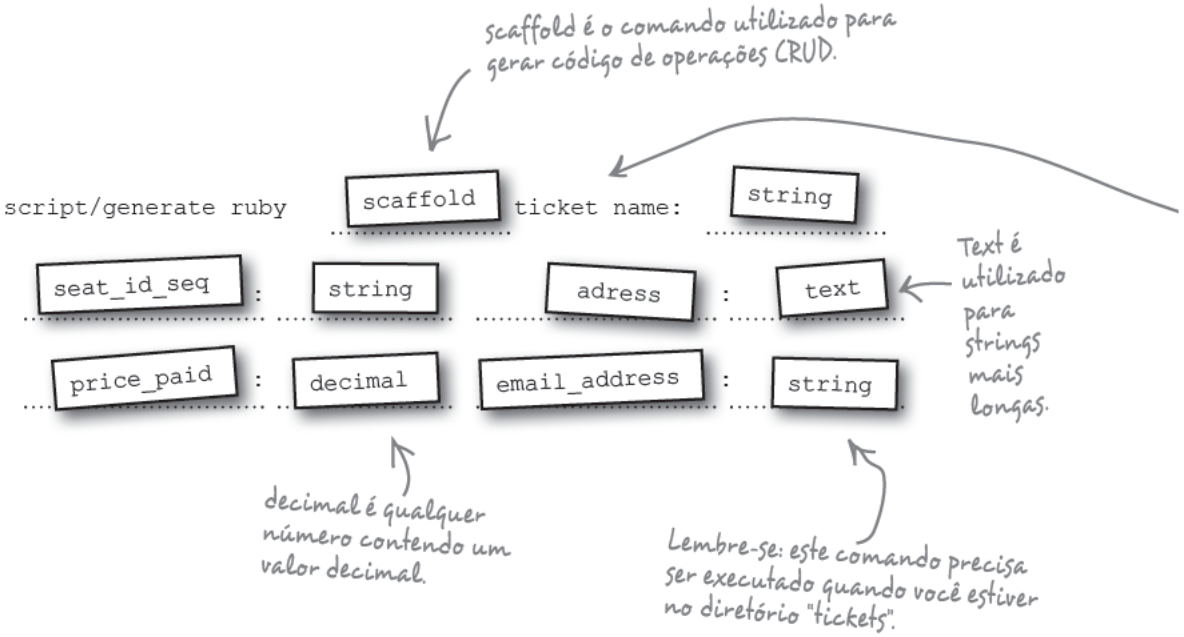
adress

email_address

string

Imãs de Geladeira Solução

Há um comando simples que você pode utilizar no console para gerar um código usando o scaffold. Veja se você pode organizar os imãs para completar este comando.



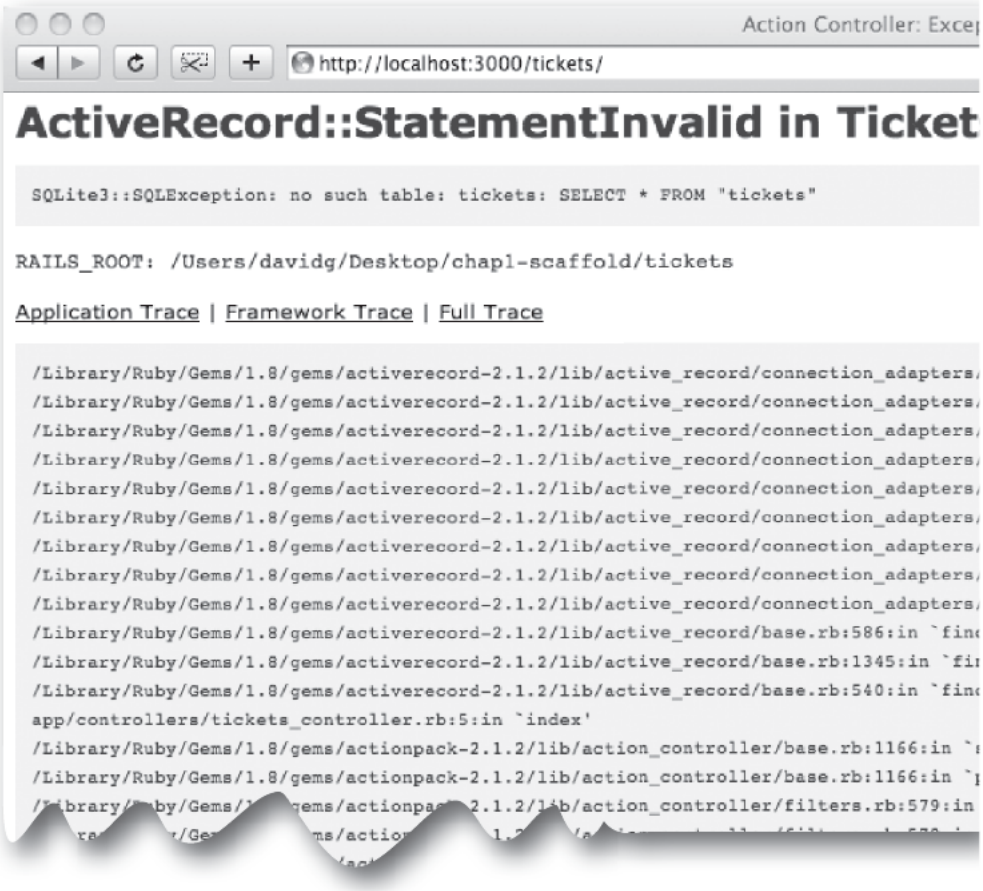
Mas o que realmente o comando scaffold faz?

O Scaffold cria o código que permitirá ao usuário criar, ler, atualizar e excluir dados no banco de dados.

Se você tem uma aplicação web de banco de dados padrão que precisa criar, ler, atualizar e excluir dados, o scaffold consegue economizar muito do seu tempo e esforço.

Digite o comando scaffold para a tabela ticket dentro do console e vamos ver o que acontece:


```
File Edit Window Help CRUD
> ruby script/generate scaffold ticket name:string
seat_id_seq:string address:text price_paid:decimal
email_address:string
```



TEST DRIVE

Agora é hora de ver se a sua aplicação realmente está funcionando. Para ver novas páginas de ingressos, com o servidor web ativo, digite em seu navegador web o comando

`http://localhost:3000/tickets`

Isto combina com o nome dado no comando scaffold. Veja como o Rails está no plural.

ActiveRecord::StatementInvalid in Ticket

```
SQLite3::SQLException: no such table: tickets: SELECT * FROM "tickets"
```

```
RAILS_ROOT: /Users/davidg/Desktop/chap1-scaffold/tickets
```

Application Trace | Framework Trace | Full Trace

```
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/connection_adapters/
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:586:in `fin
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:1345:in `fi
/Library/Ruby/Gems/1.8/gems/activerecord-2.1.2/lib/active_record/base.rb:540:in `fin
app/controllers/tickets_controller.rb:5:in `index'
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/base.rb:1166:in `
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/base.rb:1166:in `
/Library/Ruby/Gems/1.8/gems/actionpack-2.1.2/lib/action_controller/filters.rb:579:in
```

Hmmm... isto definitivamente não está certo.

Então, o que deu errado? Mesmo tendo gerado corretamente o código usando o scaffold, há um erro no servidor web. Tudo o que conseguimos foi um monte de mensagens de erro que estão retornando.

PODER DO CÉREBRO

Pense um pouco sobre essa mensagem de erro no navegador.
Por que você acha que a execução da aplicação deu errado?

Ainda não há tabelas no banco de dados!

A aplicação deveria ter exibido uma lista vazia de ingressos vendidos, mas não fez isso. **Por que não?** Ela precisava ler a lista da tabela chamada **tickets,** no banco de dados, porém ainda não criamos nenhuma tabela no banco de dados.

Só devemos conectar ao banco de dados e criar a tabela? Mas então, o que devemos fazer? Já demos informações suficientes para que o Rails crie uma tabela para nós. Dê mais uma olhada em nosso comando `scaffold`:

Lembrete: o comando scaffold refere-se a ticket e a tabela será chamada de tickets (no plural).

```
File Edit Window Help DRY
> ruby script/generate scaffold ticket name:string
seat_id_seq:string address:text price_paid:decimal
email_address:string
```

tickets	
name	string
seat_id_seq	string
address	text
price_paid	decimal
email_address	string

Já demos ao Rails as informações da estrutura de dados quando executamos o comando `scaffold`, então, aí está um princípio importante do Rails: ***Não Seja Repetitivo!*** Se você já disse algo ao Rails, não deve repetir a informação.

Então, como podemos fazer o Rails criar uma tabela?

Um dos princípios Rails:

Não
Seja
Repetitivo

Você vai ouvir falar do princípio chamado Não Seja Repetitivo ao conversar sobre programação com seus amigos.

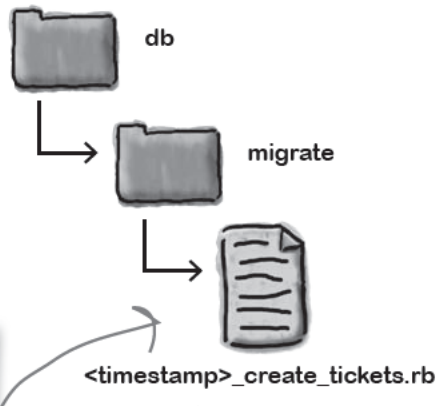
Dicas Geek

O Rails vem com um banco de dados embutido, o SQLite3. Onde ele está? O banco de dados está localizado dentro da pasta db, no arquivo development.sqlite3.

Criar uma tabela ao executar uma migração

Quando o Rails gerar a aplicação usando o scaffold, também será gerado um pequeno script Ruby chamado **migration** para criar a tabela. Uma migração (mudança) é um script que altera a estrutura subjacente do banco de dados.

Dê uma olhada na pasta `db/migrate`. Você vê um arquivo chamado `<timestamp>_create_tickets.rb` onde `<timestamp>` é o marcador de tempo **UTC** (horário padrão) de quando o arquivo foi criado. Se você abrir o arquivo em um editor de texto, o conteúdo vai parecer com isso:



O nome do arquivo que o Rails criou inclui a data e hora da criação.

```
class CreateTickets <
ActiveRecord::Migration
  def self.up
    create_table :tickets do |t|
      t.string :name
      t.string :seat_id_seq
      t.text :address
      t.decimal :price_paid
      t.string :email_address
      t.timestamps
    end
  end
  def self.down
    drop_table :tickets
  end
end
```

Aqui estão os conteúdos do arquivo de migração.

Não se preocupe, vamos falar mais disso lá na frente.

Veja bem! – **O Rails utiliza CamelCase para classes, mas underline para nomes de arquivos.**

É por isso que a migração é chamada "CreateTickets" e fica dentro de um arquivo chamado "..._create_tickets.rb"

O migration é um script Ruby pequeno. Ao invés de executar este script de forma direta, você deveria executá-lo utilizando outra ferramenta do Rails, chamada **rake**. Para executar a migração, digite **`rake db:migrate`** no prompt de comando. Isto executa o código de migração e criará a tabela:

```
File Edit Window Help DRY
> rake db:migrate
```

EXERCITANDO O CÉREBRO

Por que a migração inclui a data e horário no nome da mesma?

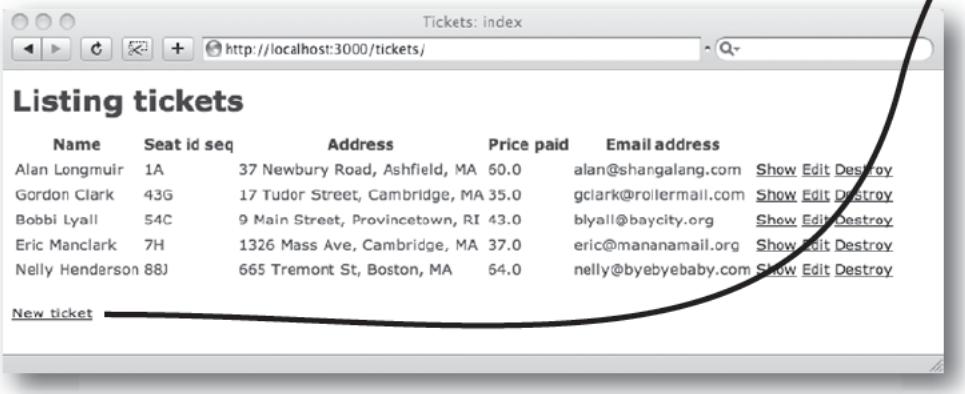
Test Drive

Tenha certeza que você criou a sua tabela de ingressos (tickets) com o comando rake.

Depois, retorne ao navegador web e atualize essa página:

```
http://localhost:3000/tickets
```

A nova aplicação web está finalmente funcionando! Dentro de poucos minutos você poderá inserir algumas informações de teste:

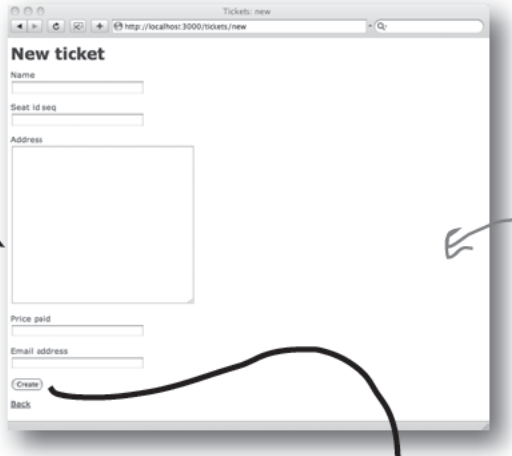


Aqui estão algumas informações que adicionamos. Agora é a sua vez, tente adicionar algumas também!

Espere! Sem chance! Só inserimos alguns comandos no console e isso criou **toda a aplicação?**

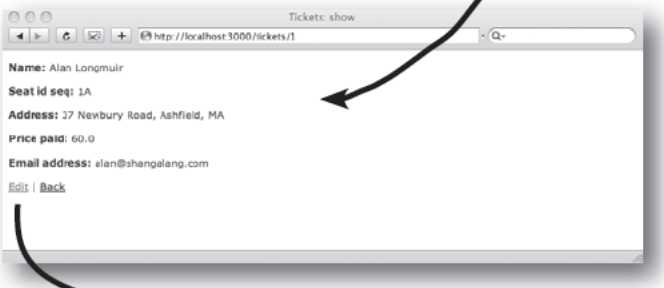
Sim – Criamos muito mais do que uma simples página inicial. Criamos todo um sistema.

O scaffold gerou todo um conjunto de páginas que nos permite criar, modificar e excluir informações dos ingressos. Para ver como a aplicação é consistente, vamos criar e editar outro registro. Siga as telas e faça os testes.



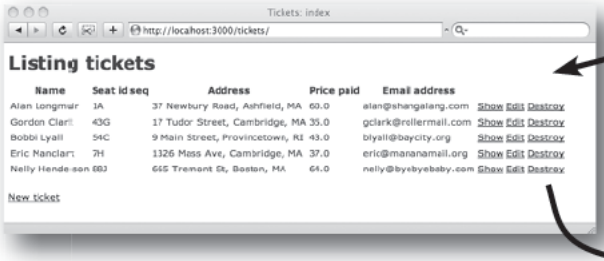
Clicar no link "Novo ingresso" na página principal direciona-o a um formulário onde você pode criar um novo ingresso.

Quando você enviar o formulário, conseguirá ler o novo ingresso, que retornou do banco de dados, e exibi-lo.



O botão "Editar", na página de exibição, permite-lhe atualizar qualquer informação que você quiser.

Clicar no botão "Retornar" vai retornar para a página principal...



... e neste link é onde nós podemos excluí um ingresso.

PONTO DE BALA

- *O comando* **`rails <nome de aplicação>`** *gera uma aplicação web na pasta* `<appname>`*. O Rails também cria as pastas e arquivos que formam a estrutura básica da sua aplicação.*
- *O Rails vem com um servidor http embutido. Para iniciar a execução do servidor, utilize o comando* **`ruby script/server`** *A home page padrão pode ser acessada assim:* **`http://localhost:3000/`**
- *As aplicações do Rails seguem a ideia de Conversão sobre Configuração*
- *Criar, Ler, Atualizar e Excluir no banco de dados são conhecidas como operações CRUD (**Create**, **Read**, **Update**, **Delete**).*
- *Scaffold gera o código CRUD para você. Para criar uma aplicação usando o scaffold, execute:*

```
ruby script/generate scaffold
"X"
<nome da coluna 1>:<tipo da
coluna 1>
<nome da coluna 2>:<tipo da
coluna 2>
...
```

- *Para visualizar o seu scaffold, digite a URL abaixo no navegador* **http://localhost:3000/things**
- *As aplicações Rails seguem o princípio Não Seja Repetitivo.*
- *Uma migração é um script que altera a estrutura do banco de dados selecionado. Você executa a migração utilizando o comando* ***rake db:migrate***

não existem Perguntas Idiotas

P: Alguns comandos começam com `rails`, outros começam com `ruby` e alguns com `rake`. Qual a diferença?

R: O comando `rails` é utilizado para criar uma nova aplicação. O `ruby` é o interpretador Ruby e é utilizado para executar os scripts das ferramentas, que são armazenados na pasta `scripts`. Os comandos `ruby` e `rake` são utilizados praticamente para tudo no Rails.

P: Então, o que é o rake?

R: O `rake` é o comando utilizado para executar a migração do banco de dados. O nome significa "Ruby faz" e é utilizado por algumas das mesmas tarefas que `make` e `ant` são, em outras linguagens como C e Java, respectivamente. Quando uma tarefa é dada ao `rake` (como executar migrações), ele serve para analisar, de forma bem sábia, a aplicação e decidir quais scripts executar. Ele é um pouco mais esperto do que o `ruby` e é utilizado para tarefas mais complicadas, como modificar a estrutura do banco de dados e executar testes.

P: Eu não entendo essa coisa de "Convenção Sobre Configuração". O que quer dizer?

R: Muitas linguagens oferecem-lhe muitas opções, como escolher os opcionais para um carro novo. Se você tem uma linguagem assim, que tenha muitas opções disponíveis, você precisa armazenar as configurações do desenvolvedor em algum lugar – geralmente em arquivos grandes de XML. O Rails tem uma abordagem diferente. Esses elementos no Rails são nomeadas de forma consistente e armazenadas em lugares padronizados. Isto se chama abordagem "convencional" – não porque é algo fora de moda, mas porque segue "convenções" ou "padrões".

P: Então não posso alterar a forma como o Rails funciona?

R: Você pode alterar praticamente tudo no Rails, mas se seguir as convenções, descobrirá que pode desenvolver sua aplicação muito mais rapidamente e que as outras pessoas terão mais facilidade para entender seu código.

Que amor! Você salvou o emprego do seu amigo!

O seu trabalho veloz com o Rails salvou o dia do seu amigo. Pelo menos, por enquanto. Veja que há outro e-mail, com outro pedido:

Obrigado!

É maravilhoso ver a aplicação pronta e funcionando – e você fez isso tão rápido! O Rails é incrível! As alterações aparecem assim que você edita o código. Sem compilar. Sem implantar. Maravilhoso.

Você salvou mesmo o meu pescoço desta vez!

Mais uma coisa – as descrições para seat_id_seq deveriam ser algo mais inteligível, talvez como "Lugar #". Você acha que você pode arrumar isso?

Como mudamos as descrições?

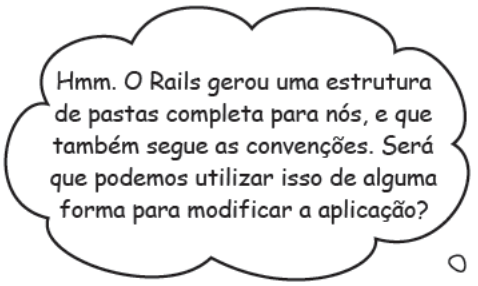
O Rails gerou uma aplicação web rapidamente, que nos economizou muito tempo e esforço. Mas, se quisermos fazer pequenas alterações na aparência das páginas, então, como devemos proceder?

Então é fácil modificar as páginas que Rails gerou?

Para modificar uma aplicação, você precisa mexer dentro da arquitetura da mesma

O comando Scaffold apenas gera o código básico para nós. Uma vez gerado, a responsabilidade pela personalização do código é sua. Se você olhar na pasta da aplicação (APP), verá que há muitos códigos gerados que você pode querer personalizar.

Então, se você precisa fazer uma alteração na aplicação "Scaffold" – como modificar as descrições da página – por onde devemos começar?



Recorrendo às convenções do Rails.

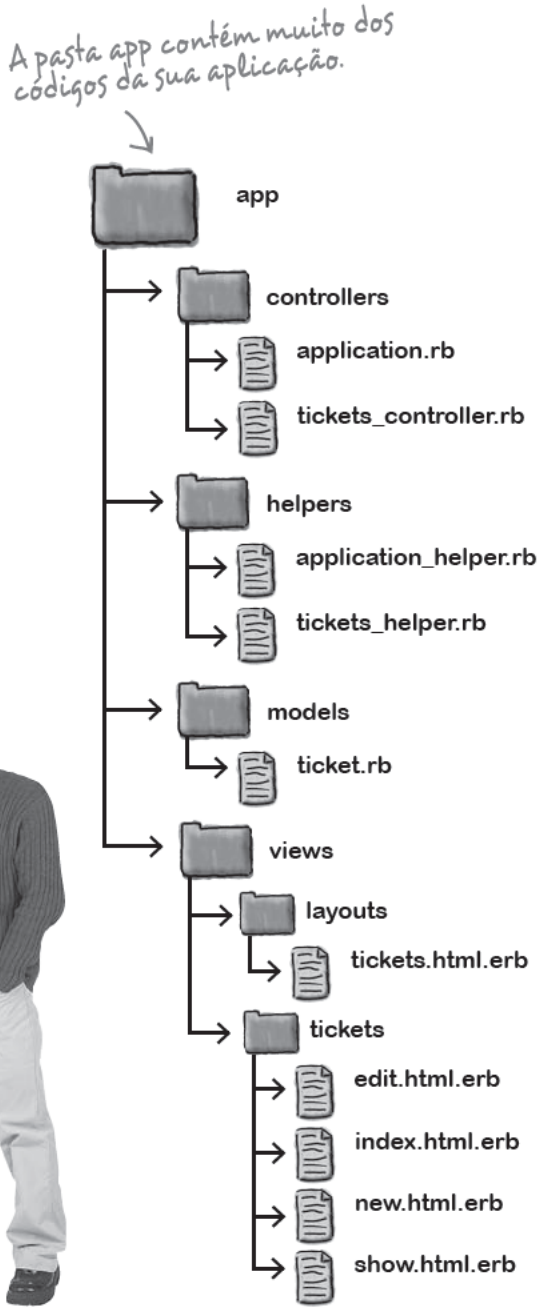
Lembra-se de quando dissemos que as aplicações do Rails seguem convenções sobre configurações?

Isso vai fazer com que a modificação das aplicações seja algo mais fácil. Por quê? Bem, porque o código é separado de acordo com a sua **função**. O que significa que os scripts Rails que fazem *coisas parecidas* fiquem em *lugares parecidos*.

Então, se você precisar alterar o comportamento da sua aplicação Rails, deve, primeiramente, saber identificar onde está o código que precisa ser alterado e comofazer isso.

Mas é claro que para fazer isso você precisa entender a:

Arquitetura Padrão do Rails.



As 3 partes da sua aplicação myc: modelo, visualização e controlador

Praticamente todos os códigos em uma aplicação Rails entram em uma destas 3 categorias:

1 Código Modelo (Model)

O código modelo gerencia como os dados são escritos e lidos no seu banco de dados. Os **objetos** do código modelo representam coisas que existem no *domínio do problem* do sistema – como os **ingressos** no sistema de tickets.

Isso significa que a sua aplicação está tentando resolver os problemas e necessidades.

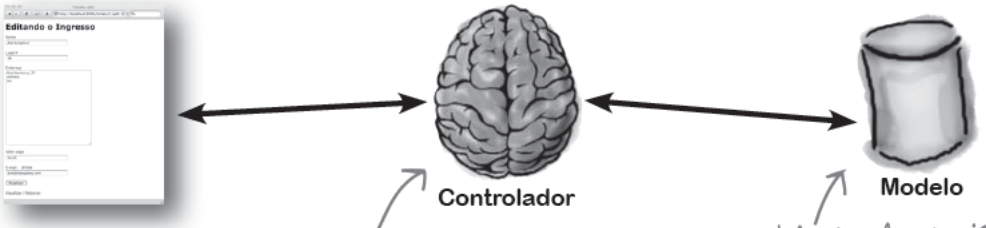
2 Código Visualização (View)

A visualização ou "visão" é a parte da aplicação que é apresentada ao usuário. Por isso, às vezes é chamada de camada de apresentação. Para uma aplicação web, a visualização normalmente gera as páginas web.

3 Código Controlador (Controller)

O controlador é o verdadeiro *cérebro* da aplicação. Ele decide como o usuário **interage** com o sistema, controlando quais dados são acessados a partir do modelo (model) e quais partes da visualização (view) vão apresentá-los.

É assim que os tipos diferentes de códigos são conectados em uma aplicação Rails.



A visualização (que também pode ser conhecido como "visão") consiste das páginas que apresentam a aplicação ao usuário.

O controlador é a chave que toma a decisão. Ele decide quais dados precisam ser acessados a partir do modelo (model) e quais partes da visualização (view) vão apresentá-los.

O modelo simula as coisas no mundo real que a aplicação está gerenciando. No sistema de vendas de ingressos, os objetos "ingressos" (tickets) ficam no modelo.

Fala Rails

Na entrevista desta semana:
Perguntamos ao framework mais radical do momento o que o faz ser tão maneiro:

Use a Cabeça: Olá, Rails! Estamos muito felizes em ter você no nosso grupo.

Rails: Por favor, me chame de Ray. Também estou feliz por fazer parte.

UC: Deve ser complicado encontrar uma pausa na sua agenda tão agitada.

Rails: Com certeza! Estou ocupado com conexões de banco de dados, lógicas de aplicação e páginas de web para servir e não tenho muito tempo para o que vocês chamam de "Vida particular", mas tudo bem, tenho bons amigos.

UC: Uma coisa que fico me perguntando, se não for lhe ofender, é claro, é por que tantos diretórios ao criar uma aplicação nova?

Rails: O que posso dizer? Sou um cara prestativo! Aprendi com o tempo como as pessoas querem fazer em suas aplicações. Não gosto de ver as pessoas fazendo as mesmas coisas várias e várias vezes, ainda mais manualmente.

UC: Mas isso não é um pouco… como posso dizer... confuso?

Rails: Ah! Por favor! Sou um cara convencional, sem surpresas. Uma vez que você aprender como eu trabalho, vai ver que é fácil e vai querer usar sempre.

UC: Soube que você não gosta de ser configurado?

Rails: Você consegue me configurar se quiser, mas a maioria das pessoas prefere trabalhar da forma que eu gosto: Convenção sobre Configuração. Capiche?

UC: Ah, sim! É um dos seus princípios de design, não é?

Rails: Sim, e também Não Seja Repetitivo.

UC: E o quê?

Rails: Não Seja Repetitivo.

UC: E o quê?

Rails: Não… Ei, você é engraçadinho.

não existem Perguntas Idiotas

P: Onde a lógica de negócio deveria ficar na minha aplicação web?

R: *Bom, tudo vai depender do que você entende por "lógica de negócio". Algumas pessoas definem como as regras associadas ao gerenciamento de dados. Neste caso, a lógica de negócio fica no modelo. Outras pessoas definem como as regras que controlam o fluxo de trabalho do sistema – como que características a aplicação tem e em que seqüência o usuário as acessa. Neste caso, a lógica de negócio fica no controlador. Daqui por diante, utilizaremos "lógica de modelo" e "lógica de aplicação" para diferenciar estes dois casos.*

P: Qual a diferença entre o controlador e a visualização?

R: *A visualização decide como a aplicação vai parecer e o controlador decide como ela vai funcionar. Sendo assim, a visualização vai definir a cor do botão em uma página web e qual texto aparece na mesma, mas o controlador decidirá o que acontece quando o botão é pressionado.*

P: Então qual código vou escrever mais?

R: *Vai depender da aplicação e do desenvolvedor. Se você descobrir que estão adicionando mais um tipo de código do que outro, vai querer pensar com cuidado se a próxima parte do código que está adicionando é do tipo apresentação, interação ou modelação.*

QUAL É MEU PROPÓSITO?

Ligue com uma seta a descrição do código à parte da aplicação a que ela pertence.

O desenho das cartas em seu jogo Monty 3-card.

Em uma aplicação bancária on-line, este código decide se você quis depositar ou transferir dinheiro de determinada conta.

Um objeto de "consulta" em uma aplicação de agenda.

Em um sistema de blog, isto decide quando exibir comentários, na forma de tabela ou lista.

Este código grava um lance em site de leilões.

Decide se você precisa logar em uma aplicação de e-mail.

Um menu de links.

Modelo (Mode)

Visualização (View)

Controlador (Controller)

QUAL É MEU PROPÓSITO?
Solução

Você tinha que ligar a descrição do código à parte da aplicação a que ele pertence. Como você se saiu?

O desenho das cartas em seu jogo Monty 3-card.

Em uma aplicação bancária on-line, este código decide se você quis depositar ou transferir dinheiro de determinada conta.

Um objeto de "consulta" em uma aplicação de agenda.

Em um sistema de blog, isto decide quando exibir comentários, na forma de tabela ou lista.

Este código grava um lance em site de leilões.

Decide se você precisa logar em uma aplicação de e-mail.

Um menu de links.

Modelo (Mode)

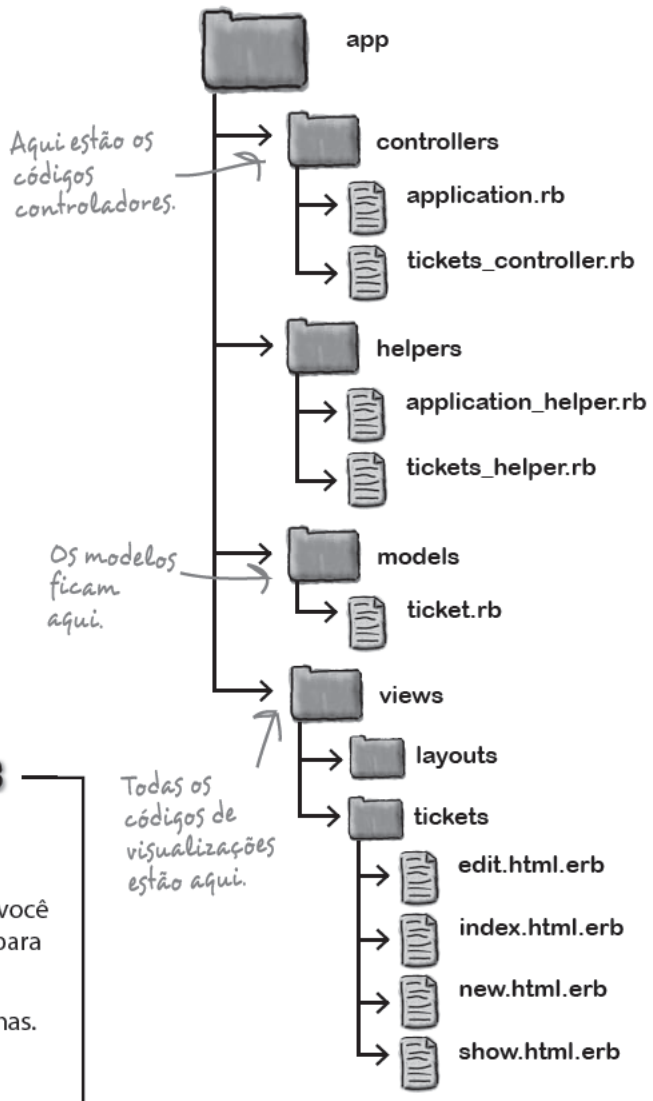
Visualização (View)

Controlador (Controller)

Os 3 tipos de código estão guardados em pastas SEPARADAS

O Rails favorece convenção sobre configuração e utiliza arquitetura MVC (*Model-view-controller*, Modelo-visualização-controlador). E daí?

Como a arquitetura MVC nos ajuda a alterar as descrições em nossas páginas web de venda de ingressos e a concertar a aplicação? Vamos voltar aos arquivos e estrutura de pasta que foram gerados pelo scaffold mais uma vez. Como o código é separado organizadamente em três tipos diferentes – modelo, visualização e controle – o Rails coloca cada tipo em uma pasta separada.



Aponte seu Lápis

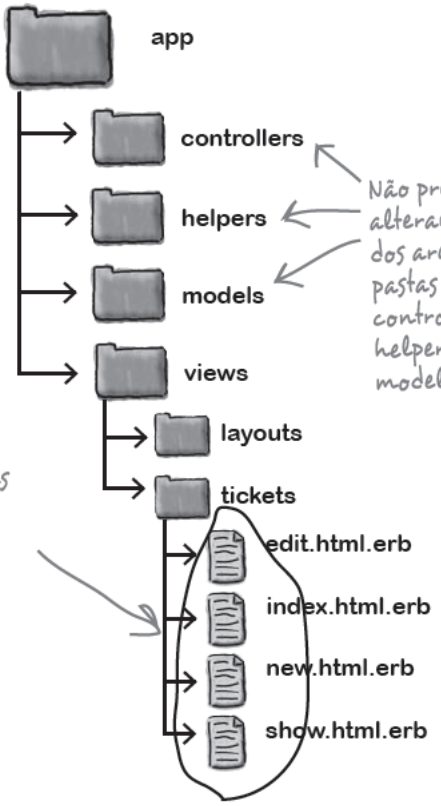
No diagrama com as pastas a sua direita, destaque os arquivos que você acha que precisarão ser editados para alterar as descrições nas páginas.

Depois, anote o ***porquê*** das escolhas.

Aponte seu Lápis
Solução

Você tinha que destacar os arquivos que você acha que precisarão ser editados para alterar as descrições nas páginas.

Como começamos a mudar a aparência das páginas, precisamos alterar os códigos de visualização. Os arquivos que precisamos atualizar são encontrados na pasta views e têm a extensão .html.erb.



Os arquivos em VIEW precisam ser editados

Se quisermos alterar as descrições e textos nas páginas web, precisamos modificar o código de visualização das mesmas. Todos os códigos de visualização ficam dentro da pasta `app/views`.

Os arquivos de visualização geram páginas web e são chamados ***templates (ou modelos) de página***. Então, o que é um template (modelo) de página e o que contém estes templates?

Editar o HTML da visualização

Então, com o que se parecem os templates (modelos) de página? Para começar, abra os quatro arquivos `html.erb` na pasta `views/tickets` utilizando um editor de texto. Os conteúdos dos arquivos parecem horríveis, muitos com instruções HTML.

Queremos mudar as descrições de seat_id_seq para Lugar#. Para fazer isso, procure pelo texto "Seat id seq" nos quatro arquivos, altere-os para "Lugar#" e salve as alterações.

Faça isso!

Entre em cada um dos quatro arquivos **`.html.erb`**, na pasta **`views/tickets`**, e altere o texto **`Seat id seq`** para **`lugar #.`** Isto vai alterar a descrição nas páginas para Lugar#.

```
<p>
  <b>Nome:</b>
  <%=h @ticket.name %>
</p>
<p>
  <b>Lugar #id seq:</b>
  <%=h @ticket.seat_id_seq %>
</p>
<p>
  <b>Endereço:</b>
  <%=h @ticket.address %>
</p>
```

Este é o texto que você precisa alterar para Lugar#.

```
<h1>Editando o Ingresso</h1>
<% form_for(@ticket) do |f| %>
  <%= f.error_messages %>
  Não se esqueça de adicionar aspas, porque este é uma string.
<p>
  <%= f.label :name %><br />
  <%= f.text_field :name %>
</p>
<p>
  "lugar #"
  <%= f.label :seat_id_seq %><br />
  <%= f.text_field :seat_id_seq %>
</p>
```

Este símbolo precisará ser alterado para a string "Lugar#".

Se você editar as descrições no seu HTML, essas alterações tornam-se imediatamente visíveis em seu navegador. Se você fizer as alterações *agora* e atualizar, elas ficarão visíveis *imediatamente*. Você pode usar estas mesmas dicas e traduzir os demais campos.

não existem — Perguntas Idiotas

P: Você chamou :seat_id_seq de símbolo. O que é um símbolo?

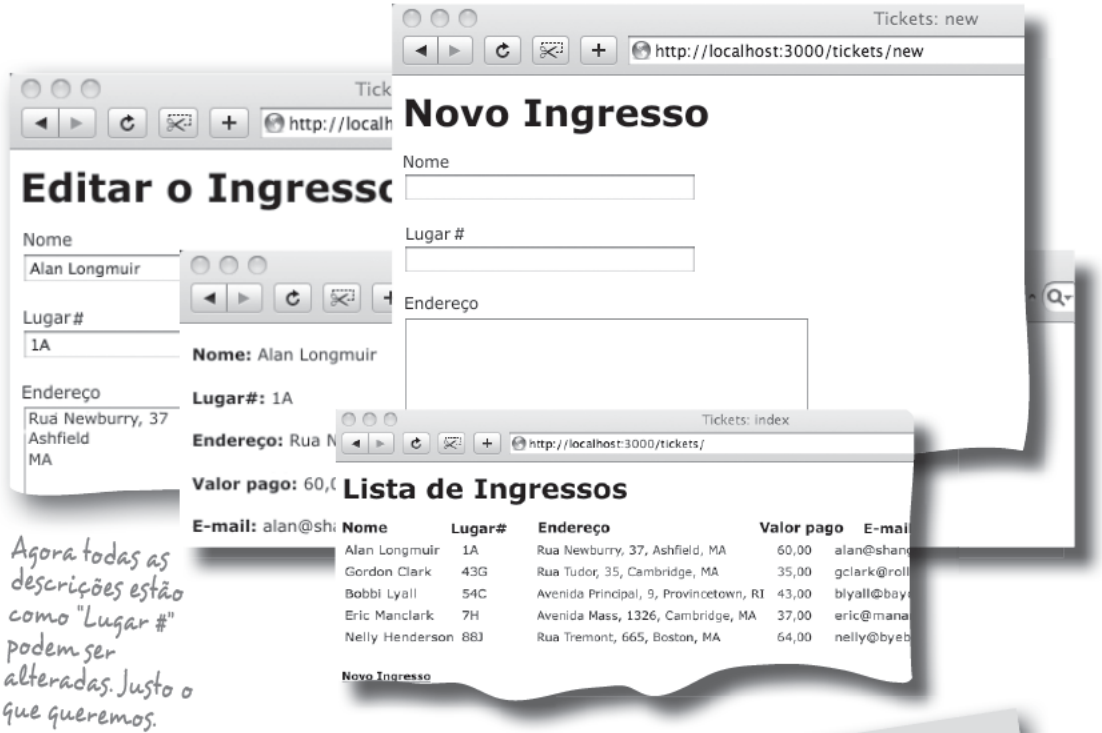
R: *Um símbolo é um pouco parecido com uma string. Uma string vem cercada por aspas ("") mas um símbolo sempre é iniciado com dois pontos (:).*

Os símbolos geralmente são utilizados para nomear coisas no Rails, porque eles são um pouco mais eficientes na memória. Na maioria dos casos, símbolos e strings podem ser utilizados permutavelmente.

Test Drive

Atualize esta instrução no seu navegador:

```
http://localhost:3000/tickets/
```



Agora todas as descrições estão como "Lugar #" podem ser alteradas. Justo o que queremos.

Você terá a rapidez com que a alteração apareceu na sua aplicação?

Isso acontece porque o Rails é feito com Ruby, e o código Ruby não precisa ser compilado. Assim, o servidor de web do Rails pode apenas executar seu código fonte atualizado. Mas isso é uma coisa boa?

Você tem que passar por muito menos etapas para testar o código que você alterou. Não precisa compilar o código, a propósito, você não precisa fazer um pacote com o código e instalá-lo onde quer que seja. Tudo o que precisa é escrever o código e executá-lo. O ciclo de desenvolvimento do Rails é bem rápido e é rápido também para fazer alterações na sua aplicação web existente.

Estes passos são irrelevantes quando você está desenvolvendo no Rails.

Domingo, 8 da manhã

Dois problemas resolvidos, mas o seu telefone está tocando... e agora?

Oi! Bom saber que a aplicação está funcionando bem. Escute... Acho que você precisa saber que o meu chefe ligou e quis saber onde você mora. Ele quer muito conhecer mais sobre o trabalho que você fez, mas ainda está meio preocupado se ficará pronto até amanhã pela manhã, então ele quer ver hoje. Obrigado por todo o trabalho feito. Ah, a propósito, eu cheguei a falar que quero que grave o telefone de contato e também o e-mail para cada ingresso vendido? Desculpa – eu tinha esquecido.

TOC!

TOC!

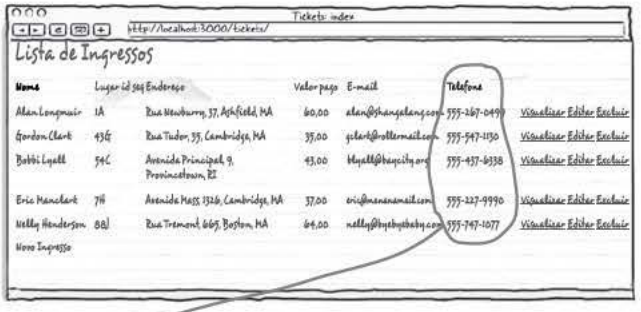

Então, **você é** o único responsável pela nova aplicação. Espera-se que as primeiras reservas para o show estejam disponíveis em menos de 24 horas, sendo assim é melhor que esta aplicação esteja pronta ou **eu vou** querer saber o motivo...

Agora a aplicação precisa armazenar mais informações

Tudo estava pronto até que o seu amigo mencionou que os números de telefone precisam ser gravados. Precisamos armazenar mais dados, mas o que isso significa para a nossa aplicação?

1 Precisamos exibir mais um campo em cada página.

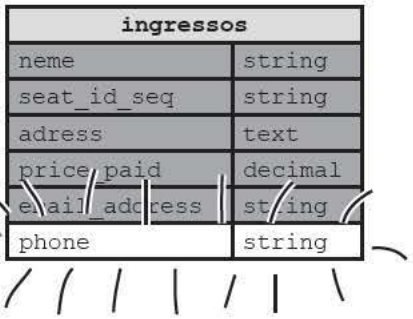
Felizmente, sabemos como alterar modelos (template) de página, então isso não deve ser um grande problema.



Precisamos adicionar e exibir mais um campo (coluna) como este à página.

2 Precisamos armazenar mais uma coluna no banco de dados.

Precisamos armazenar mais uma coluna em nosso banco de dados, mas como?



precisamos adicionar a coluna phone (telefone) na tabrla tickets no banco de dados

Aponte seu Lápis

Precisamos armazenar mais uma coluna no banco de dados. Escreva a seguir qual foi o tipo de script que nós utilizamos anteriormente para alterar a estrutura do banco de dados.

.....

Aponte seu Lápis
Solução

Você deveria responder que que foi o tipo de script nós utilizamos antes para alterar a estrutura do banco de dados.

Uma migração

Uma migração é apenas um script Ruby

Nós precisamos de uma migração para adicionar uma coluna a uma tabela. Mas, na verdade, o que *é* uma migração? Vamos relembrar ao que criou nossa tabela de ingressos.

```
class CreateTickets < ActiveRecord::Migration
  def self.up
    create_table :tickets do |t|
      t.string :name
      t.string :seat_id_seq
      t.text :address
      t.decimal :price_paid
      t.string :email_address
      t.timestamps
    end
  end
  def self.down
    drop_table :tickets
  end
```

Precisamos criar um código que seja parecido com isso, mas ao invés de criarmos uma tabela, precisamos da nossa migração (que na verdade seria uma alteração) para adicionar uma coluna.

Nós precisamos CRIAR o código, mas não significa que precisamos ESCREVER o código.

não existem Perguntas Idiotas

P: Alguns códigos na migração parecem que estão recriando a tabela. Por que isso?

R: *Migrações podem fazer muito mais do que estamos mostrando aqui. Por exemplo, cada migração tem a habilidade de desfazer-se. É por isso que o código criará uma nova tabela combinada pelo código que pode recriar a tabela. Você ainda não precisa saber muito sobre isso.*

P: Não preciso entender o código? É importante entender de código Ruby para dominar o Rails?

R: *Quanto mais você entender Ruby, mais controle do Rails você terá. No percurso deste livro você aprenderá mais e mais sobre a linguagem Ruby.*

P: Se a migração é só um script Ruby, porque tenho que usar o rake? Por que não posso só executar o script?

R: *Boa pergunta. Algumas partes do Ruby são projetadas para serem executadas diretamente e outras não. Migrações são projetadas para serem executadas diretamente.*

P: Sim, tudo bem – mas por quê?

R: `rake` *é "mais esperto" do que ruby. Ao chamar* `rake db:migrate`*, você está, na verdade, dizendo ao* `rake`*: "Tenha certeza de que todas as migrações foram executadas".* `rake` *pode decidir não fazer uma migração e mudanças se ela não for necessária. Ruby não consegue tomar essas decisões sozinho.*

P: Eu posso editar a minha tabela de ingressos manualmente?

R: *Você poderia, mas é melhor gerenciar a sua estrutura de banco de dados com migrações. Quando a sua aplicação já está funcionando, você precisará recriar as estruturas de dados no banco de dados em produção. Se você utilizar as migrações, então o rake poderá fazer com que as estruturas de dados, em seu banco de dados de produção, forneçam o que você precisa para a sua aplicação. Se você modificar a sua estrutura de dados manualmente, as coisas podem sair da sincronização facilmente. Com a maioria das coisas no Rails, se você seguir a forma convencional de utilizar o Rails, fará com que tudo fique mais fáceis para você mesmo.*

Rails pode gerar migrações

Lembre-se de quando geramos a aplicação via scaffold utilizando:

```
ruby script/generate scaffold ticket name:string seat_id_seq:string
address:text price_paid:decimal email_address:string
```

Que o `generate` é um script para criar código Ruby. A boa notícia é que `generate` não *só* escreve código padrão scaffold, mas também gera migrações.

Agora suponhamos que você tenha digitado esse comando:

Na verdade, não digite isso.

```
ruby script/generate migration PhoneNumber
```

Isto geraria um novo arquivo de migração em branco. Poderíamos, então, adicionar o código Ruby para modificar a tabela. O problema é: até o momento não sabemos como escrever o código para completar o processo de migração.

O que fazemos então? E o que o bondoso Rails pode fazer por nós?

Dê um nome "esperto" à sua migração e o Rails escreve o código para você

Provavelmente, você notou até aqui que nomes são realmente importantes no Rails.
Quando criamos a aplicação com o comando scaffold a chamamos de "tickets". O Rails deixou a aplicação visível em http://localhost:3000/tickets e gerou uma migração para criar a tabela chamada tickets.

Convençoes em nomes são importantes no Rails porque isto poupará o seu trabalho. Isto também ocorre na forma com que você nomeia as migrações. Ao invés de apenas dar um velho nome à migração, tente dar um novo como esse:

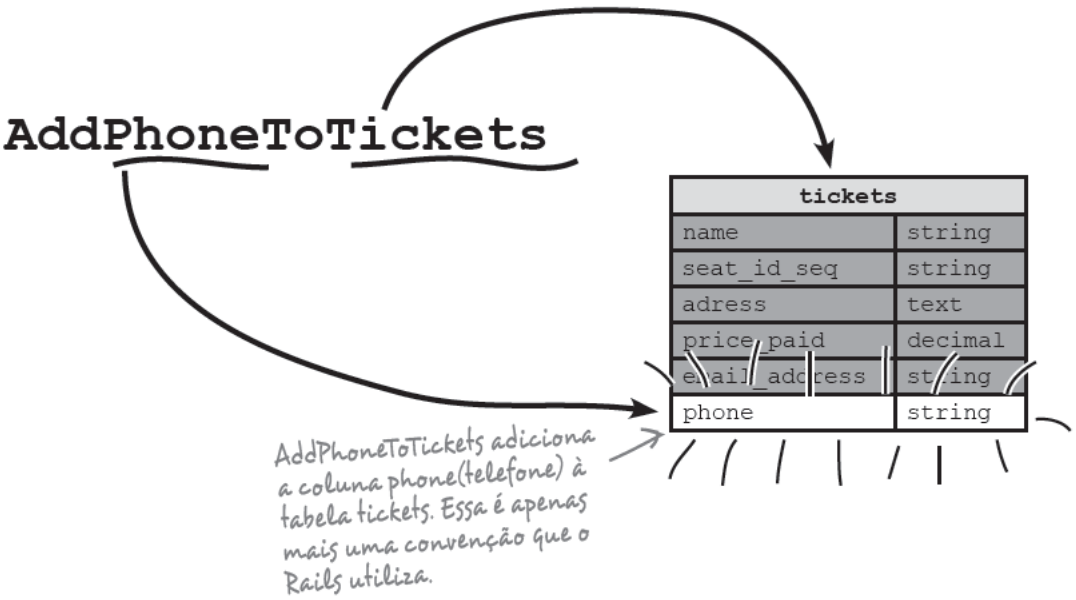
A parte importante é este nome aqui. Ele leva a forma Add...To...(Adicionar)

```
File Edit Window Help NamesMatter
> ruby script/generate migration AddPhoneToTickets phone:string
```

Você precisa executar esse comando.

Por que o nome faz diferença?

O Rails sabe que um migração chamada Add...To... estará, provavelmente, adicionando um coluna específica a uma tabela específica, ao invés de apenas gerar uma migração em branco para você preencher, ***o Rails vai, na verdade, escrever para você o seu código para a migração.***

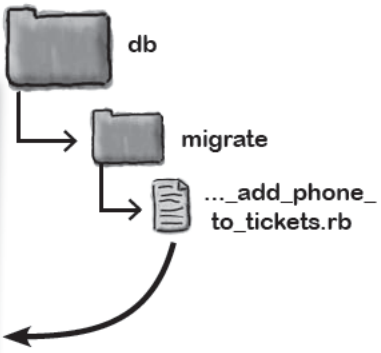


Agora você precisa executar a sua migração com rake.

Aqui está uma migração que o Rails gerou para você.

```
class AddPhoneToTickets < ActiveRecord::Migration
  def self.up
    add_column :tickets, :phone, :string
  end

  def self.down
    remove_column :tickets, :phone
  end
end
```



Quando quisermos executar uma migração, utilizaremos o já conhecido comando `rake :`

```
rake db:migrate
```

Mas podemos fazer isso dessa vez? Afinal, não queremos que o `rake` execute a primeira migração novamente, o que, provavelmente, levará a um erro.

Marcadores de tempo (timestamps) dizem ao rake quais migrações executar e em que ordem.

O Rails grava a última timestamp de todas as migrações executadas. Isto permite que o `rake` saiba quais migrações foram executadas e quais não foram. Isto significa que sempre que você executar `rake db:migrate`, ***o Rails executará apenas a última migração.***

Vamos testar isso. Execute `rake db:migrate` novamente e adicione a coluna telefone à tabela ingressos.


```
> rake db:migrate
```

Mas alterar o banco de dados não é suficiente

O Scaffold *gera* código - e isso é muito bom, porque você pode implementar e executar rapidamente algo. Mas a parte não tão boa é que, uma vez que o código foi gerado, é **responsabilidade do desenvolvedor** mantê-lo **atualizado**.

Acabamos de adicionar o atributo telefone ao banco de dados. Mas como os formulários já foram criados pelo scaffold, eles não usarão o novo campo automaticamente. Então, teremos que voltar aos modelos (templates) de página e adicionar as referências para editar o número de telefone:

```
<p>
   <%= f.label :email_address %><br />
   <%= f.text_field :email_address %>
</p>
<p>
   <%= f.label :phone %><br />
   <%= f.text_field :phone %>
</p>
<p>
   <%= f.submit "Update" %>
</p>
```

Isto está dentro do arquivo edit.html.erb.

Este é o código adicionado ao arquivo show.html.erb.

```
<b>Valor pago:</b>
  <%=h @ticket.price_paid %>
</p>

<p>
  <b>E-mail:</b>
  <%=h @ticket.email_address %>
</p>

<p>
```

```
<p>
  <%= f.label :email_address %><br />
  <%= f.text_field :email_address %>
</p>
<p>
  <%= f.label :phone %><br />
  <%= f.text_field :phone %>
</p>
<p>
```

É um new.html.erb - a página contendo o formulário "Criar".

E, finalmente, no arquivo index.html.erb, que gera a lista de todos os ingressos. Precisamos adicionar código em dois lugares aqui: no cabeçalho da coluna e em cada linha da tabela.

```
    <th>E-mail</th>
    <th>Telefone</th>
  </tr>
<% for ticket in @tickets %>
  <tr>
    <td><%=h ticket.name %></td>
    <td><%=h ticket.seat_id_seq %></td>
    <td><%=h ticket.address %></td>
    <td><%=h ticket.price_paid %></td>
    <td><%=h ticket.email_address %></td>
    <td><%=h ticket.phone %></td>
    <td><%= link_to 'Exibir', ticket %></td>
```

não existem Perguntas Idiotas

P: Por que tem <%=h ... %> em alguns lugares? O que significa "h"?

R: *"h" é um método auxiliar (ou Helper). **Auxiliares** são utilizados para coisas como formatar uma saída. O auxiliar "h" escapa caracteres especiais no campo como "<" e "&". Isto evitará que as pessoas enviem texto para sites que contenham JavaScript ou outro tipo de código.*

P: Por que as strings são utilizadas em alguns lugares e os símbolos em outros?

R: *Strings são utilizados em modelos (templates) de página onde uma parte simples de texto é necessária. Símbolos (as palavras que começam com ":") são, comumente, utilizadas em descrições/títulos de campos.*

P: Por quê?

R: *Símbolos são mais eficientes memorizados e muitos dos métodos (como f.label), que aceitam parâmetros e que preferem ter símbolos a strings. Mas na maioria dos casos, os métodos do Rails permitem-lhe optar por usar strings ao invés de símbolos se eles são fáceis de formatar.*

Longo exercício

O chefe está satisfeito com o jeito que a aplicação está funcionando e agora ele quer gravar dados e informações dos eventos e, também, as vendas de ingressos. Essa é a estrutura de dados de eventos:

```
Evento:
artist – o artista (string)
description – pequena biografia (text)
price_low – ingressos mais baratos (decimal)
price_high – valor do ingresso (decimal)
event_date – quando vai acontecer (date)
```

Que comando você insere na console para criar um código scaffold para os dados do evento?

.....

.....

O que você digitaria para criar a tabela de eventos no banco de dados?

.....

O chefe quer que a descrição dos campos na página web para price_low nas páginas seja "Preços a partir de", e para price_high, "Até", e para event_date, seja "Data". Você precisará editar quatro modelos para fazer a alteração. Escreva as alterações necessárias no modelo (template) de página new.html.erb mostrado abaixo:

new.html.erb

```
<h1>Novo Evento</h1>
<% form_for(@event) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :artist %><br />
    <%= f.text_field :artist %>
  </p>
  <p>
    <%= f.label :description %><br />
    <%= f.text_area :description %>
  </p>
  <p>
    <%= f.label :price_low %><br />
    <%= f.text_field :price_low %>
  </p>
  <p>
    <%= f.label :price_high %><br />
    <%= f.text_field :price_high %>
  </p>
  <p>
    <%= f.label :event_date %><br />
    <%= f.date_select :event_date %>
  </p>
  <p>
    <%= f.submit "Criar" %>
  </p>
<% end %>
<%= link_to 'Retornar', events_path %>
```

Quais são os nomes dos outros três modelos no diretório app/views/events que precisam ser alterados?

Longo Exercício Solução

O chefe está satisfeito com o jeito que a aplicação está funcionando e agora ele quer gravar dados e informações dos eventos e, também, as vendas de ingressos. Essa é a estrutura de dados de eventos:

Evento:
artist – o artista (string)
description – pequena biografia (text)
price_low – ingressos mais baratos (decimal)
price_high – valor do ingresso (decimal)
event_date – quando vai acontecer (date)

Que comando você insere na console para criar um código scaffold para os dado do evento?

```
ruby script/generate scaffold event artist:string description:text price_low:decimal
price_high:decimal event_date:date
```

O que você digitaria para criar a tabela de eventos no banco de dados?

```
rake db:migrate
```

O chefe quer que a descrição dos campos na página web para price_low nas páginas seja "Preços a partir de", e para price_high, "Até", e para event_date, seja "Data". Você precisará editar quatro modelos para fazer a alteração. Escreva as alterações necessárias no modelo (template) de página new.html.erb mostrado abaixo:

```
<h1>Novo Evento</h1>
<% form_for(@event) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :artist %><br />
    <%= f.text_field :artist %>
  </p>
  <p>
    <%= f.label :description %><br />
    <%= f.text_area :description %>
  </p>
  <p>
    <%= f.label :price_low %><br />
    <%= f.text_field :price_low %>
  </p>
  <p>
    <%= f.label :price_high %><br />
    <%= f.text_field :price_high %>
  </p>
  <p>
    <%= f.label :event_date %><br />
    <%= f.date_select :event_date %>
  </p>
  <p>
    <%= f.submit "Criar" %>
  </p>
<% end %>
<%= link_to 'Retornar', events_path %>
```

:preçoas à partir de

"Preços a partir de" também funcionaria - mas utilizar um símbolo é melhor.

:até

Ou "Até"

:data

Ou "Data"

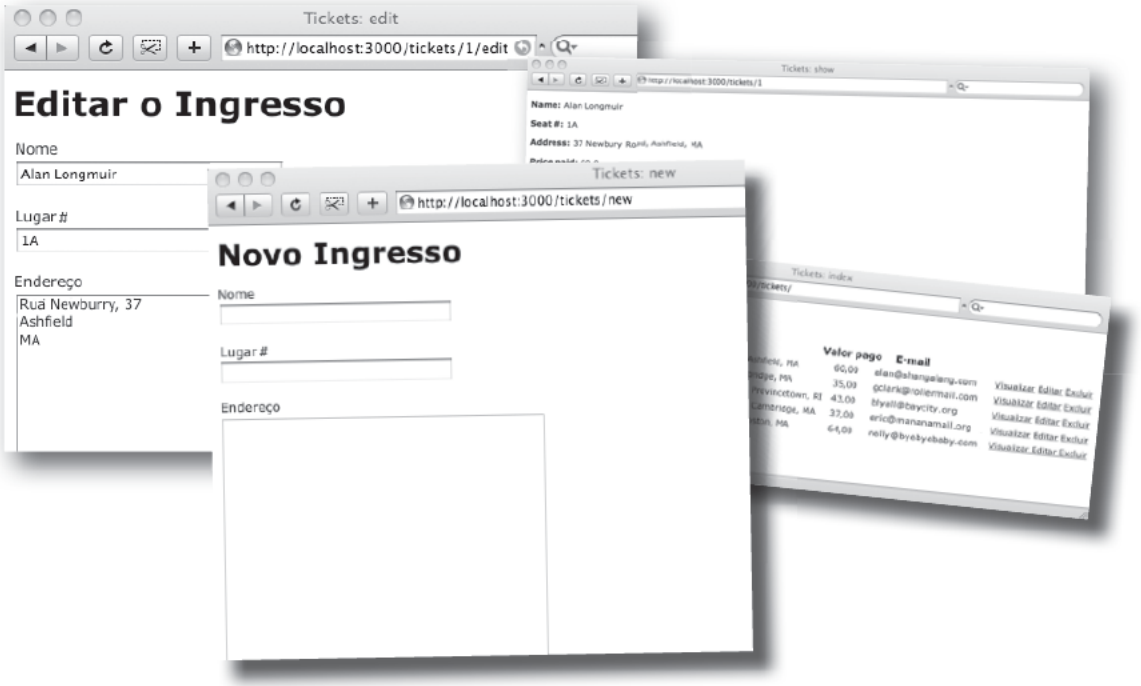
new.html.erb

Quais são os nomes dos outros três modelos no diretório app/views/events que precisam ser alterados?

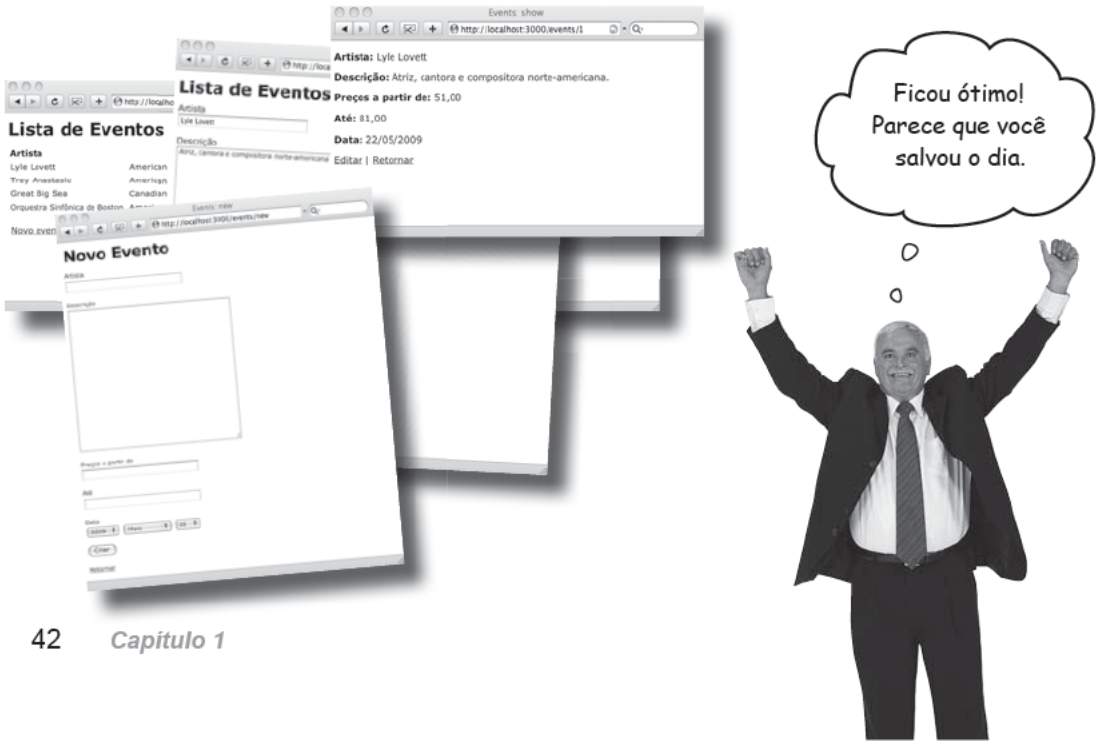
edit.html.erb, show.html.erb e index.html.erb

Test Drive

Agora, a aplicação tem todas as informações de contato nas páginas de venda de ingressos:



E todas as informações dos eventos também ficarão gravadas:



E o show é um sucesso!

A aplicação funcionou perfeitamente a semana inteira e os lugares estão esgotados para a próxima sexta a noite.

PONTOS DE BALA

- *O Rails segue uma arquitetura Model-View-Controller (**Modelo-Visualização-Controle**), conhecida como arquitetura MVC.*
- *O Rails gera pastas separadas por modelo (model), visualização (view) e para controlador (controller) do código.*
- *Quaisquer alterações que você faça na sua aplicação podem ser visualizadas assim que você salvá-las e recarregar as páginas do navegador. Isto ocorre porque o Rails é feito em Ruby e não precisa ser compilado.*
- *Você pode fazer mudanças na estrutura da sua tabela utilizando o proceddo conhecido como migração. Para gerar uma migração (mudança) que adicione uma coluna a uma tabela, utilize o seguinte comando:*

```
ruby script/generate migration
Add<nome da coluna>To<nome da
tabela> <nome da coluna>:<tipo
da coluna>
```

- *Para executar uma migração, utilize o comando:*

```
rake db:migrate
```

Algumas ferramentas para a sua Caixa de Ferramentas do Rails

Você concluiu o capítulo 1 com sucesso, agora você sabe criar aplicações básicas com o Rails, e agora você já pode incluir essas habilidades na sua caixa de ferramentas.

Ferramentas do Rails

rails nome_da_aplicação
Cria uma aplicação.

ruby script/server
Inicia uma aplicação.

ruby script/generate scaffold...
Gera um código do tipo CRUD para um modelo.

ruby script/generate migration
Gera uma migração para alterar a estrutura do banco de dados.

rake db:migrate
Executa novas migrações no banco de dados.