PMR3304 - Sistemas de Informação - 07

Aula 07

Prof. Dr. Marcos de Sales Guerra Tsuzuki

11 de Outubro de 2019

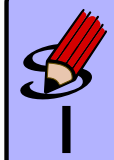
PMR-EPUSP

Armazenamento de Variáveis em Cookies e Autenticação de Usuário

Armazenamento de Variáveis em Cookies - Introdução

Nesta aula, vamos verificar as formas de armazenamento em cookies disponíveis no Ruby on Rails. Não há a necessidade de acessar cookies diretamente. As ferramentas de suporte do Rails para sessões e autenticação de usuário podem auxiliar neste gerenciamento. Vamos criar uma nova aplicação:

```
rails new guestnote
cd guestnote
rails generate controller entries
```



Para você fazer em casa

Execute os comandos descritos.

Armazenamento de Variáveis em Cookies - Introdução

Vamos modificar o arquivo
app/controller/entries_controller.rb para o conteúdo abaixo:

```
class EntriesController < ApplicationController
  def sign_in
    @previous_name = cookies[:name]
    @name = params[:visitor_name]
    cookies[:name] = @name
  end
end
```

A primeira linha coleta o nome anteriormente fornecido e armazenado no cookie, e o armazena em *@previous_name* assim a **view** poderá exibi-lo. O cookie chega até o servidor pela requisição HTTP header. A segunda linha, recebe o nome do campo do form *:visitor_name* e a terceira linha armzena o *@name* como um cookie.

Armazenamento de Variáveis em Cookies - Introdução

Vamos criar o arquivo **app/views/entries/sign_in.html.erb** com o conteúdo abaixo:

```
<h1>Hello <%= @name %></h1>
    <%= form_tag action: 'sign_in' do %>
        <p>Enter your name:
        <%= text_field_tag 'visitor_name', @name %></p>
        <%= submit_tag 'Sign in' %>
    <% end %>
    <% unless @previous_name.blank? %>
      <p>Hmmm... the last time you were here, you said you were
      <%= @previous_name %>.</p>
    <% end %>
```

Resta modificar o **config/routes.rb** para que o roteamento encontre o controller.

```
Rails.application.routes.draw do
  match ':controller(/:action(/:id(.:format)))', via: [:get, :post]
end
```

Para você fazer em casa

Execute o desenvolvimento.

O header HTTP que carrega o cookie entre o browser e o servidor está invisível. Existem mais alguns parâmetros para associar ao cookie, como *:value*, *:domain*, *:path*, *:expires*, *:secure* e *:https_only*.

Armazenamento de Variáveis em Cookies - Introdução

- *:value* - valor para o cookie;
- *:domain* - indica o domínio para o qual o cookie se aplica. Se a aplicação estiver sob um domínio `http://myapp.example.com` então o domínio deverá configurado como o mesmo ou `http://example.com`.
- *:path* - indica o path para o qual o cookie se aplica. Se for `/entries/sig_in`, path poderá ser `/`, `/entries` ou `/entries/sign_in`.
- *:expires* - indica quando o cookie expirará. Poderá ser *5.minutes* ou *12.hours.from.now*.
- *:secure* - se configurado como *true* o cookie será transmitido utilizando apenas HTTP segura *HTTPS*.
- *:http_only* - indica que o cookie será transmitido apenas por HTTP ou HTTPS, e não por javascript.

Armazenamento Dados entre Sessões

Armazenamento Dados entre Sessões - Introdução

Cookies são úteis para preservar informações isoladas entre mudanças de página. Mas, o Rails permite que o gerenciamento dos cookies seja feito por ele, e que nós fiquemos em um nível superior. É o que veremos neste exemplo.

Sessões permitem que seja mantida uma coerência entre o que os usuários estão executando. O Rails não sabe nada específico sobre oo usuário, apenas que ele possui um cookie com um específico valor.

Vamos modificar o arquivo **app/controllers/entries_controller.rb** para ter um novo código.

Armazenamento Dados entre Sessões - Introdução

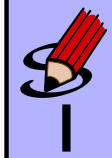
```
class EntriesController < ApplicationController
  def sign_in
    @names=session[:names]
    unless @names
      @names=[]
    end
    @name = params[:visitorName]
    if @name
      @names << @name
    end
    session[:names]=@names
  end
end
```

O novo código está funcionando com um vetor ao invés de ser um único campo. O grande problema é inserir uma nova entidade a um vetor não existente. O método *sign_in* recupera o vetor *names* a partir do objeto *session* e o atribui para *@names*. Em seguida o método recupera o último *visitor_name* do formulário e o inclui no vetor *@names*. A última linha de código atualiza *session[:names]*.

A **view app/views/entries/sign_in** precisará também ser refeita e está exibida abaixo:

```
<h1>Hello <%= @name %></h1>
<%= form_tag :action => 'sign_in' do %>
    <p>Enter your name:
    <%= text_field_tag 'visitorName', @name %></p>
    <%= submit_tag 'Sign in' %>
<% end %>
<% if @names %>
<ul><% @names.each do |name| %>
      <li><%= name %></li>
  <% end %></ul>
<% end %>
```

Neste exemplo, as variáveis estarão disponíveis enquanto a sessão estiver ativa. Caso o browser seja fechado e uma nova sessão criada, as variáveis disponíveis para o primeiro caso, não estarão disponíveis para o segundo.



Para você fazer em casa

Execute o desenvolvimento.

Autenticação de Usuários

Autenticação de Usuários - Introdução

Praticamente qualquer aplicação requer a autenticação de usuários. O Rails não possui nenhum mecanismo específico para gerenciar usuários. Existem algumas **gems** que permitem realizar a autenticação de usuários. O *Devise* `http://bit.ly/2aTEqdW` e o mais popular na comunidade Rails. E o *OmniAuth* `http://bit.ly/2aTEpqz` possui a vantagem de autenticar utilizando terceiros como *Twitter* e *Facebook*.

O ideal é desenvolver o núcleo da aplicação, e apenas depois incluir a autenticação. Em nosso caso, vamos incluir a autenticação à aplicação *students* que desenvolvemos em aulas anteriores. Vamos para o diretório **students** que contém a aplicação.

Internamente ao diretório vamos executar o comando:

```
rails g resource user email password_digest
```

Foi utilizado **g** ao invés de **generate**, apenas uma abreviação. Foi utilizado **resource** que substitui **scaffold**. Será criado um modelo contendo *email* e *password_digest* como campos to tipo **string** (não foi escrito nada e o Rails assume). Não serão criadas as **views** automaticamente, isto evitará que todos os usuários sejam exibidos em uma única **view**, o que é um ponto fraco. Em seguida execute **rails db:migrate** para criar a tabela *users*.

Vamos modificar o arquivo **app/models/user.rb** para incluir *has_secure_password*.

```
class User < ApplicationRecord
  has_secure_password
end
```

Ao incluir esta linha, será necessário descomentar a utilização da **gem 'bcrypt', '~> 3.1.7'** no arquivo **Gemfile**. Após descomentar, será necessário executar **bundle** para efetivar a modificação.

Também será necessário modificar o arquivo **app/controllers/user_controller.rb**.

Autenticação de Usuários - Introdução

```
class UsersController < ApplicationController
  def new
    @user = User.new
  end
  def create
    @user = User.new(user_params)
    if @user.save
      session[:user_id] = @user.id
      redirect_to root_url, notice: "Thank you for signing up!"
    else
      render "new"
    end
  end
  ...
```

```
  ...
  private
  # Use callbacks to share common setup or constraints between
      actions.
  def set_user
    @user = User.find(params[:id])
  end
  # Never trust parameters from the scary internet, only allow the
      white list through.
  def user_params
    params.require(:user).permit(:email, :password, :
        password_confirmation)
  end
end
```

Método *new* o objeto *user* é criado, e no método *create* um novo objeto *user* é criado com os parâmetros do usuário. Se o usuário for criado com sucesso, será redirecionado para a URL *root*, e será exibida uma mensagem. Em caso contrário será redirecionado para a página *new*.

Vamos criar a página *new* no arquivo **app/views/users/new.html.erb**.

```
<h1>Sign Up</h1>

<%= form_for @user do |f| %>
  <% if @user.errors.any? %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% @user.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
```

Autenticação de Usuários - Introdução

```
  <div class="field">
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </div>
  <div class="field">
    <%= f.label :password %><br />
    <%= f.password_field :password %>
  </div>
  <div class="field">
    <%= f.label :password_confirmation %><br />
    <%= f.password_field :password_confirmation %>
  </div>
  <div class="actions"><%= f.submit "Sign Up" %></div>
<% end %>
```

Vamos criar o *root* URL no arquivo **config/routes.rb**.

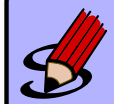
```
Rails.application.routes.draw do
  resources :users
  ...
  root to: "students#index"

  # For details on the DSL available within this file, see http://guides.
      rubyonrails.org/routing.html
end
```

Resta apenas incluir um link no menu principal para realizar o *sign_in*. Vamos modificar o arquivo **app/views/application/_navigation.html.erb**.

```
<p>
  <%= link_to "Students", students_path %> |
  <%= link_to "Courses", courses_path %> |
  <%= link_to "Sign Up", new_user_path %>
</p>

<hr>
```



Para você fazer em casa

Vamos testar a execução em `http://localhost:3000/students`.

Após executar, clique em *Sign Up* e crie um novo usuário.

Resta agora, criar uma página para realizar a autenticação.
Para este fim, vamos criar um **controller** *sessions*.

```
rails g controller sessions new
```

Vamos modificar o arquivo **config/routes.rb** novamente, vamos remover o **GET** route para *sessions* e incluir *resources :sessions*.

```
Rails.application.routes.draw do
  resources :sessions
  resources :users
  '''
  root to: "students#index"

  # For details on the DSL available within this file, see http://guides.
      rubyonrails.org/routing.html
end
```

O arquivo **app/views/sessions/new.html.erb** será modificado para a listagem abaixo:

```
<h1>Log In</h1>
<%= form_tag sessions_path do %>
  <div class="field">
    <%= label_tag :email %><br />
    <%= text_field_tag :email, params[:email] %></div>
  <div class="field">
    <%= label_tag :password %><br />
    <%= password_field_tag :password %></div>
  <div class="actions"><%= submit_tag "Log In" %></div>
<% end %>
```

O arquivo **app/controllers/sessions_controller.rb** será modificado para a listagem abaixo:

```
class SessionsController < ApplicationController
  def create
    user = User.find_by_email(params[:email])
    if user && user.authenticate(params[:password])
      session[:user_id] = user.id
      redirect_to root_url, notice: "Logged in!"
    else
      flash.now.notice = "Email or password is invalid"
      render "new"
    end
  end
  ...
```

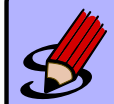
Autenticação de Usuários - Introdução

```
  ...
  def new
  end
end
```

Resta apenas incluir um link no menu principal para realizar o *login*. Vamos modificar o arquivo **app/views/application/_navigation.html.erb**.

```
<p>
  <%= link_to "Students", students_path %> |
  <%= link_to "Courses", courses_path %> |
  <%= link_to "Sign Up", new_user_path %> |
  <%= link_to "Log In", new_session_path %>
</p>

<hr>
```



Para você fazer em casa

Vamos testar a execução em `http://localhost:3000/students`.

Após executar, clique em *Log In* com uma senha correta e veja que você realiza o log in.

Vamos agora modificar o menu para refletir este estado. Para este fim, será criado um método *current_user* para o **controller** da aplicação. Assim, este método estará visível para toda a aplicação. O arquivo modificado será **app/controllers/application_controller.rb**.

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  private

  def current_user
        @current_user ||= User.find(session[:user_id]) if session[:
            user_id]
  end
  helper_method :current_user
end
```

Agora podemos modificar o menu **app/views/application/_navigation.html.erb** utilizando este **helper**.

```
<p><%= link_to "Students", students_url %> |
  <%= link_to "Courses", courses_url %> |
  <% if current_user %>
    Logged in as <%= current_user.email %>.
    <%= link_to "Log Out", logout_path %>
  <% else %>
    <%= link_to "Sign Up", signup_path %> |
    <%= link_to "Log In", login_path %>
  <% end %></p>
<hr />
```

Precisamos incluir o log out, será o equivalente a destruir a sessão. Vamos modificar o arquivo **app/controllers/sessions_controller.rb**.

```
class SessionsController < ApplicationController
  def new
  end
  ...
  def destroy
    session[:user_id] = nil
    redirect_to root_url, notice: "Logged out!"
  end
end
```

Uma última modificação será incluir no roteamento simplificações para facilitar a leitura (arquivo **config/routes.rb**):

```
Rails.application.routes.draw do
  get 'signup', to: 'users#new', as: 'signup'
  get 'login', to: 'sessions#new', as: 'login'
  get 'logout', to: 'sessions#destroy', as: 'logout'

  resources :sessions
  ...
end
```

Conectando a Autenticação ao Restante da Aplicação

No momento, não existe diferença entre autenticação e não autenticação para o restante da aplicação. Ou seja, usuários não autenticados e usuários autenticados são iguais para o restante da aplicação. Vamos diferenciá-los. Para isto, será criado um método no arquivo **app/controllers/application_controller.rb**.

```
class ApplicationController < ActionController::Base
  private
  ...
  def authorize
        redirect_to login_url, alert: "Not Authorized" if current_user.
            nil?
  end
end
```

Este método poderá ser inserido em todos os **controlllers** como uma **before_action**. Vamos acrescentar isto ao **app/controllers/atudent_controller.rb**.

```
class StudentsController < ApplicationController
  before_action :set_student, only: [:show, :edit, :update, :destroy]
  before_action :authorize, only: [:edit, :update, :destroy]

  # GET /students
  # GET /students.json
  def index
    @students = Student.all
  end
```

Caso ninguém esteja logado, podemos esconder os links para editar e remover *students*. Vamos acrescentar isto ao **app/views/students/index.html.erb**.

```
<td><%= link_to 'Show', student %></td>
<td><%= link_to 'Awards', student_awards_path(student) %>
    </td>
<% if current_user %>
  <td><%= link_to 'Edit', edit_student_path(student) %></td>
  <td><%= link_to 'Destroy', student, method: :delete, data: {
      confirm: 'Are you sure?' } %></td>
<% end %>
```

Os Vários Atores

Todas as aplicações devem possuir mais que um ator. Vamos considerar o caso de um ator *admin*. Vamos incluir uma flag em *user* indicando se é do tipo *admin*. Vamos executar o comando **rails generate migration AddAdminFlagToUsers**. Isto criará a seguinte migration:

```
class AddAdminFlagToUsers < ActiveRecord::Migration[5.2]
  def change
    add_column :users, :admin, :boolean, default: false, null: false
  end
end
```

Execute **rails db:migrate**.

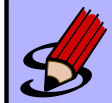
Vamos incluir um checkbox na **view** para criar usuários e identificar se é do tipo *admin*. O arquivo **app/views/users/new.html.erb** deve ser modificado para:

```
  <div class="field">
    <%= f.label :password_confirmation %><br />
    <%= f.password_field :password_confirmation %>
  </div>
  <div class="field">
    <%= f.label :admin %><br />
    <%= f.check_box :admin %>
  </div>
  <div class="actions"><%= f.submit "Sign Up" %></div>
<% end %>
```

Também devemos incluir o parâmetro *admin* ao conjunto de parâmetros necessários para verficação no **controller app/controllers/users_controller.rb**:

```
class UsersController < ApplicationController
    ...

    # Never trust parameters from the scary internet, only allow the
        white list through.
    def user_params
      params.require(:user).permit(:email, :password, :
          password_confirmation, :admin)
    end
end
```



Para você fazer em casa

Agora faça um teste e crie um usuário do tipo administrador.

Vamos definir alguns requisitos para cada um dos atores:

- Usuários não autenticados não podem fazer nenhuma ação;
- Usuários comuns (não administradores) podem apenas visualizar os *students* (ações como *show* e *index*);
- Usuários administradores podem fazer qualquer coisa (*new*, *edit* e *destroy*).

O primeiro requisito pode ser implementado modificando o método **before_action** do **controller** de *students*.

```
class StudentsController < ApplicationController
  before_action :set_student, only: [:show, :edit, :update, :destroy]
  before_action :authorize
```

Agora, ao tentar acessar a lista de *students* o sistema será redirecionado para a tela de log in.

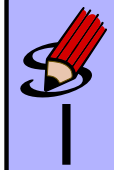
O próximo requisito é que usuários comuns (não administradores) podem apenas visualizar *students* (ações do tipo *show* e *index*). Simultaneamente, vamos também considerar o terceiro requisito que diz, usuários administradores podem realizar todas as ações. Vamos criar um método *is_admin?* no **controller** de applicação (arquivo **app/controllers/application_controller.rb**).

```
class ApplicationController < ActionController::Base
  ...
  def is_admin?
    redirect_to students_path, alert: "Not authorized" if current_user.nil?
          or !current_user.admin?
  end
end
```

O método **before_action** do **controller** de *students* será novamente modificado para refletir esta modificação:

```
class StudentsController < ApplicationController
  before_action :set_student, only: [:show, :edit, :update, :destroy]
  before_action :authorize
  before_action :is_admin, only: [:new, :edit, :update, :destroy]
```

Agora, ao tentar acessar os métodos *new*, *edit*, *update* e *destroy* este teste será executado e não permitirá a sua execução.



Para você fazer em casa

Agora faça um teste e verifique a situação.

Apesar de que o método não permite o seu acionamento, a interface exibe o link, fornecendo a idéia de que a função está acessível. O próximo passo é esconder o link para os casos em que as funções não estão disponíveis. Vamos iniciar com *new* . Vamos modificar o arquivo **app/views/students/index.html.erb**.

```
<br>

<% if current_user.admin? %>
   <%= link_to 'New Student', new_student_path %>
<% end %>
```

A última modificação será com relação às ações *edit* e *destroy* que também devem ser habilitadas apenas para o usuário administrador. Vamos modificar o arquivo **app/views/students/index.html.erb**.

```
<% if current_user.admin? %>
  <td><%= link_to 'Edit', edit_student_path(student) %></td>
  <td><%= link_to 'Destroy', student, method: :delete, data: {
      confirm: 'Are you sure?' } %></td>
<% end %>
```

The End!