PMR3304 - Sistemas de Informação - 01

Aula 01

Prof. Dr. Marcos de Sales Guerra Tsuzuki

02 de Agosto de 2019

PMR-EPUSP

Introdução

Introdução - Estrutura do Curso

- 02/08/2019 - Introdução à Ruby on Rails, Views e CSS;
- 09/08/2019 - Controller, Models, Active Record, Scaffold e REST;
- 16/08/2019 - **Não haverá aula**;
- 23/08/2019 - Models com Forms, Validação;
- 30/08/2019 - Criando Relações no Modelo, Gerenciando Banco de Dados;
- 13/09/2019 - Debugar, Testes, Sessões e Cookies;
- 20/09/2019 - **Primeira Prova**;
- 27/09/2019 - **Não haverá aula**;
- 04/10/2019 - Autenticação de Usuário, Roteamento, CSS e Sass;
- 11/10/2019 - JavaScript e CoffeeScript, enviando e-mails;
- 18/10/2019 - SQL e Modelagem Entidade Relação;
- 25/10/2019 - Padrões de Projeto;
- 01/11/2019 - Exemplo de Projeto Baseado em Objetos;
- 08/11/2019 - Exemplo de Projeto Baseado em Objetos;
- 22/11/2019 - **Segunda Prova**;
- 29/11/2019 - **Não haverá aula**;
- 06/12/2019 - **Prova Substitutiva (fechada)**.

Introdução - Objetivo da Disciplina

- **Ojetivo**: apresentar um método para análise e projeto de sistemas orientado a objetos, baseado na abordagem do Processo Unificado (PU).
- **Processo Unificado**: a motivação para o uso da abordagem de Craig Larman [1] ao Processo Unificado deve-se ao fato de que este é um processo bastante conciso e eficiente para análise e projeto de sistemas orientado a objetos. Neste método, cada artefato (documento ou diagrama) tem uma razão muito clara para existir e as conexões entre os diferentes artefatos são muito precisas.

[1] Larman, Craig – Utilizando UML e Padrões. 2a. Edição, Bookman, 2004.

Introdução - Software Deselegante

- Software deselegante é aquele software feito sem uma **estrutura clara**.
- O software deselegante é aquele do qual não **se consegue reusar partes** e que não se consegue **entender como funciona** sem uma boa carga de documentação (e muitas vezes nem assim).
- É aquele no qual uma **pequena modificação** em uma de suas características pode causar um **não funcionamento generalizado**.

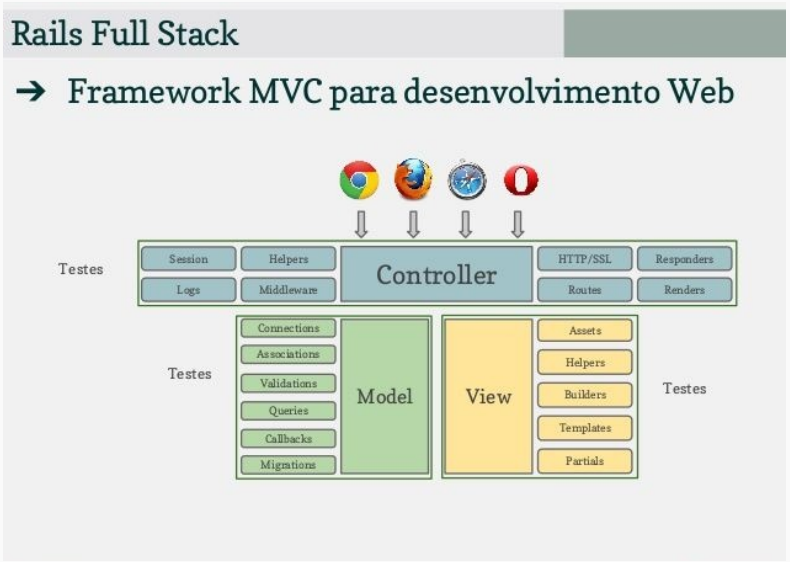
- Criada por Yukihiro Matsumoto em fevereiro de 1993. Primeira release pública (v. 0.95) em dezembro de 1995. Primeira versão estável em dezembro de 1996. "*Ruby is simples in appearence, but is very complex inside, just like our human body*".
- Foi inspirada em Perl, Smalltalk, Eiffel, Ada e Lisp.
- Existem diversos tutoriais online: `https://pine.fm/LearnToProgram`
- Pode ser testada online:
  - `https://repl.it/languages/ruby`.
  - `https://www.tutorialspoint.com/execute_ruby_online.php`
  - `http://rextester.com/l/ruby_online_compiler`

Introdução - Rails Filosofia e Pontos Fortes

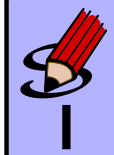
- Ruby - menos código e código mais compreensível, tempo de desenvolvimento menor, simples, mas poderoso, sem ciclo de compilação;
- CoC (Convention Over Configuration) –> praticamente sem arquivos de configuração, estrutura de diretórios pré-definida, convenções de nomes –> menos código, facilidade de manutenção;
- Boas práticas: MVC (Model View and Controller) - Isolamento da Lógica de Negócios e da Interface com o Usuário, DRY (Don't Repeat Yourself) - Facilidade de Manter o Código, Testing;ler
- Praticamente tudo em código Ruby on Rails (SQL e JavaScript são abstraídos);
- Suporte AJAX integrado, Web Services com REST (Representational State Transfer);
- Grande comunidade, ferramentas e documentações;
- Extraído de aplicações reais (Twitter, GitHub, Shopify, Basecamp, Aibnb, ...).

Introdução - Componentes do Rails

- Action Controller: processa as requisições;
- Action View: criar saídas HTML ou XML;
- Active Record: independência de BD;
- Action Mailer: serviços de e-mail;
- Active Resource: fornece suporte para WebServices;
- Active Jobs: fornece suporte para execução de threads;
- Active Support: classes utilitárias.



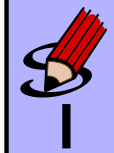
- Para o Tiger (10.4) ou inferior: verificar nos sites `http://brew.sh` (guia Homebrew) e no `http://bit.ly/2aTroNH` (tutorial do RailsGirls).
- Para Linux: Ruby já vem em diversas versões de Linux e OS X. Verifique se ele está presente executando o comando **ruby -v** na linha de comando. Desejamos a versão 2.2.2 ou maior.
- Para Windows: o melhor é utilizar o instalador presente em `http://rubyinstaller.org`.



Para você fazer em casa

Instalar o ruby no sistema operacional de sua escolha.

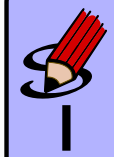
- Você deverá executar **gem install rails**. Isto instalará a última versão do Rails (pelo menos 5.0.1).
- Vamos instalar também o sqlite3, que não vem automaticamente com o rails gem. Execute **gem install sqlite3**.
- Para verificar quais gems estão presentes em seu ambiente execute o comando **gem list --local**.
- Para Windows: utilize o instalador windows localizado em `http://railsinstaller.org/en`. Após a instalação do Rails será necessário instalar também o Node.js disponível em `http:/nodejs.org`. Após estas insalações, o computador deve ser reiniciado.



Para você fazer em casa

Instalar o rails e o sqlite3.

- Uma vez que o rails já está instalado, podemos começar o seu uso. A partir da linha comando tecle **rails new hello01**. Diversos diretórios serão criados e módulos serão instalados.
- Você deverá entrar no diretório recém criado: **cd hello01**.
- A aplicação foi criada, e agora o servidor web deve ser iniciado: **rails server**. Neste momento, o servidor está executando a aplicação recém criada.
- Vamos verificar o funcionamento em um browser acessando `http://localhost:`3000.



Para você fazer em casa

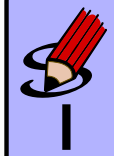
Crie a primeira aplicação rails e execute.

Introdução - O que são todos estes diretórios?

- *app*: diretório que contém o núcleo da aplicação. Ele inclui subdiretórios para controllers, assets (como imagens, stylesheets, e JavaScript)
- *config*: armazena as configurações de banco de dados, regras de roteamento de URL, e as estruturas de ambiente do Rails para desenvolvimento, testes e produção. Um arquivo muito importante é o **config.ru**.
- *db*: armazena scripts para gerenciar as tabelas do banco de dados relacional.
- *doc*: coleta os documentos gerados pelo código Ruby using RubyDoc. Para maiores informações veja `http://www.ruby-doc.org`.
- *lib*: armazena os códigos que não se encaixam nas classificações de model, view ou controller.
- *log*: contém o log - não apenas erros, mas informação muito rica sobre os processamentos.
- *public*: contém HTML estático e os arquivos **favicon.ico** para a aplicação.
- *test*: contém o código para testar a aplicação Rails.
- *tmp*: armazena variáveis de sessão, arquivos temporários, dados cache e outros.
- *vendor*: este diretório deverá ser extinto no futuro.

Rails na Web

- Como criar o “Hello World”. Vamos postar uma mensagem utilizando uma view pela inclusão de HTML que será enviado para o browser.
- Execute o comando: **rails generate controller hello index**.
- Esta linha de comando indica que será criado um controller de nome **hello**, e o último argumento requesita que um uma view de nome **index** seja criada.
- Você deverá editar o arquivo **config/routes.rb** e modificar a linha **get “hello/index”** para **get ’hello’ => ’hello#index’**. Para editar, um possível sofware é o NotePad++[2]. Importante: o caracter ’ não pode ser copiado e colado, precisa ser digitado.
- Vamos verificar o funcionamento em um browser acessando `http://localhost:3000/hello/`.



Para você fazer em casa

Crie a primeira aplicação rails e execute.

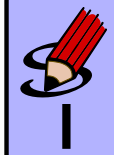
[2] `https://notepad-plus-plus.org`

O conteúdo padrão do arquivo **app/views/hello/index.html.erb**

```
<h1>Hello#index<\h1>
<p>Find me in app/views/hello/index.html.erb</p>
```

Vamos substituir o conteúdo deste arquivo para:

```
<h1>Hello!<</h1>
<p>This is a greeting from app/views/hello/index.html.erb</p>
```



Para você fazer em casa

Execute `http://localhost:3000/hello/` e verifique a saída.

O conteúdo padrão do controller **app/controllers/hello_controller.rb**

```
class HelloController < ApplicationController
  def index
  end
end
```

Vamos substituir o conteúdo deste arquivo para:

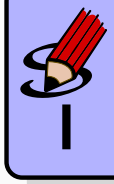
```
class HelloController < ApplicationController
  def index
    @message="Hello!"
    @count=3
    @bonus="This message came from the controller."
  end
end
```

Agora possuímos variáveis.

O conteúdo padrão do view **app/views/hello/index.html.erb**

```
<h1><%= @message %><</h1>
<p>This is a greeting from app/views/hello/index.html.erb</p>
<p><%= @bonus %></p>
```

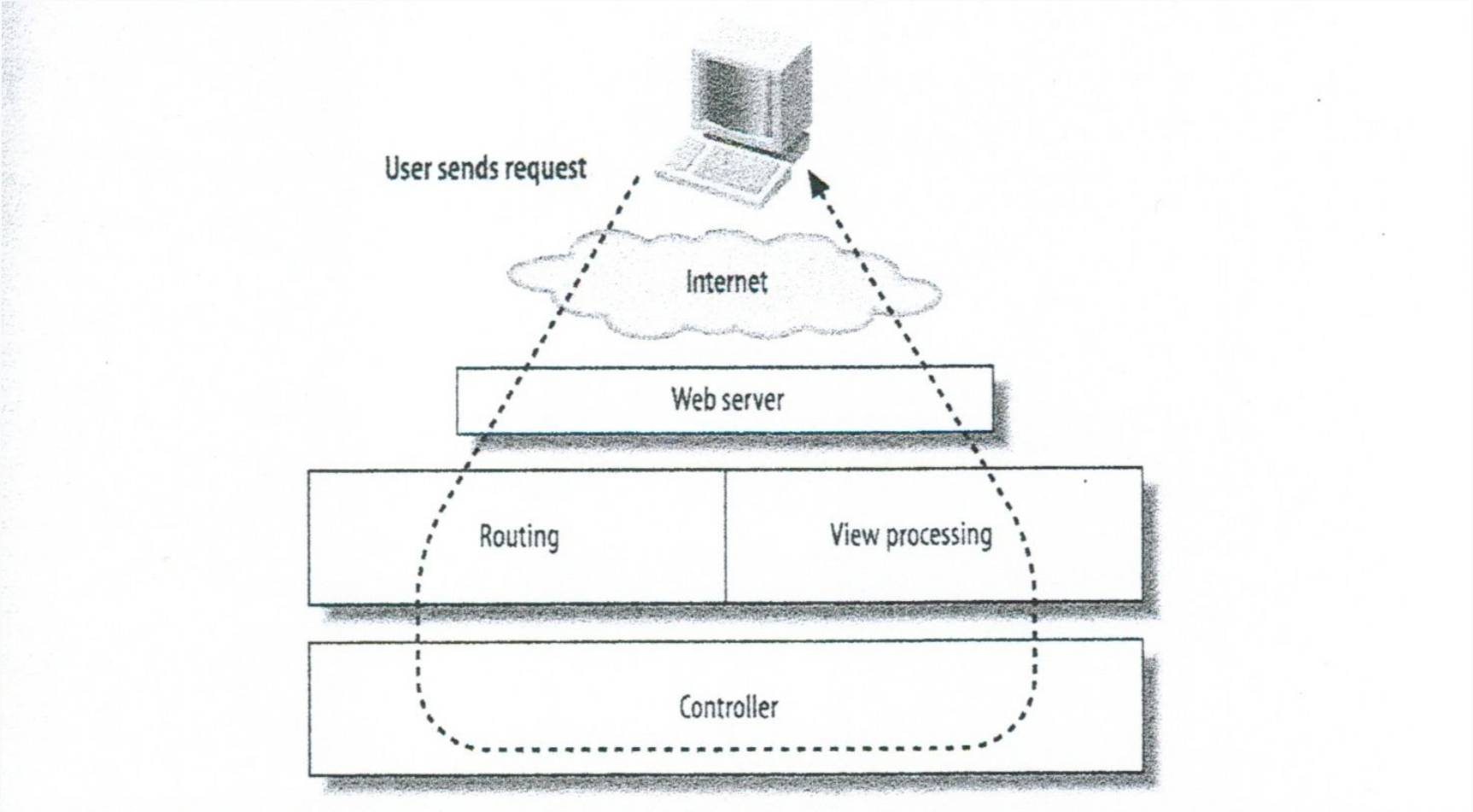
O código limitado por <% e %> são programas Ruby embebidos.



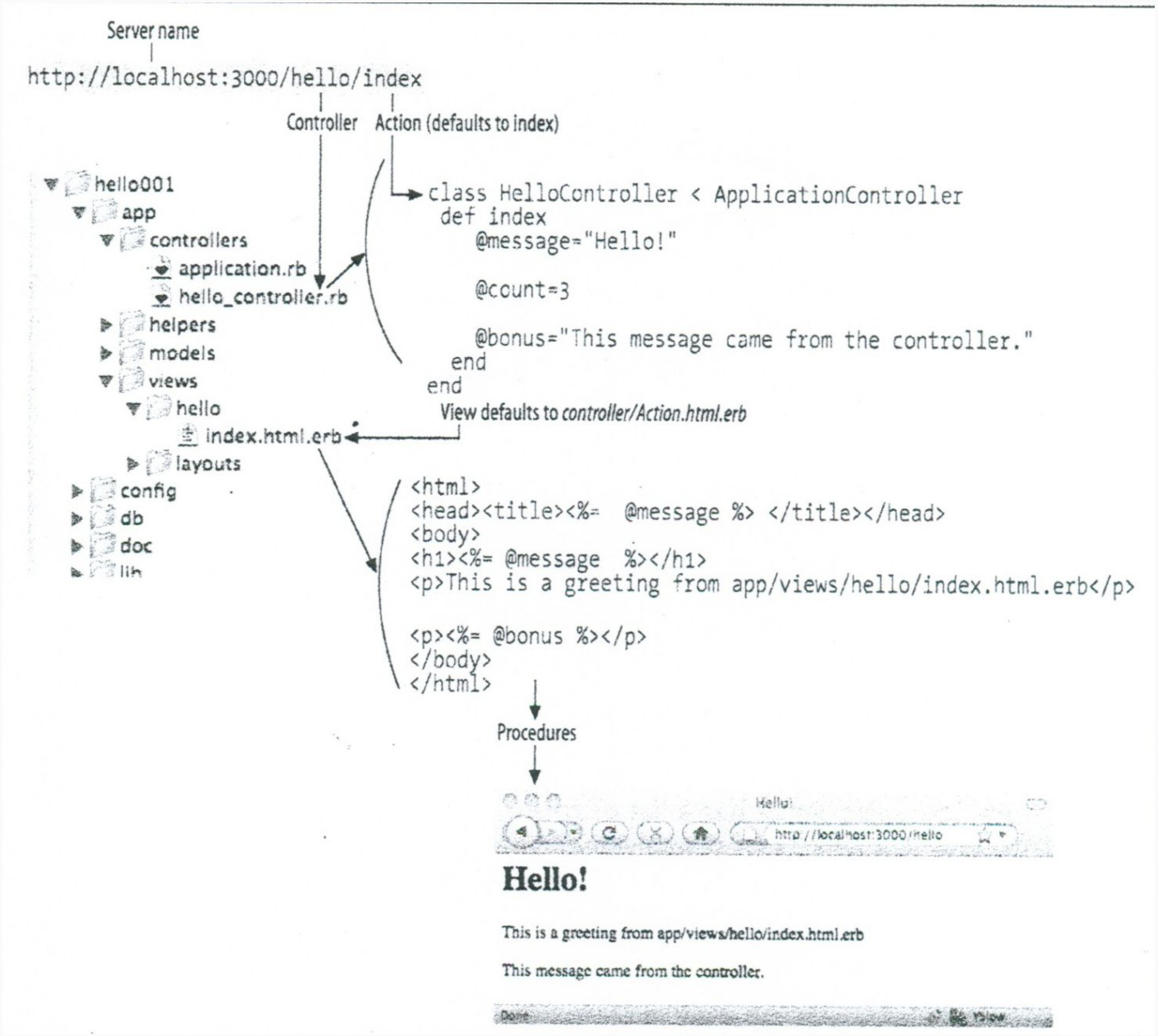
Para você fazer em casa

Execute `http://localhost:3000/hello/` e verifique a saída.

Rails na Web - Comunicando Controller e View



Rails na Web - Comunicando Controller e View

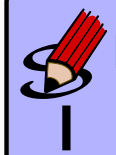


Modificar o view **app/views/hello/index.html.erb** para

```
<h1><%= @message %><</h1>
<p>This is a greeting from app/views/hello/index.html.erb</p>

<% for i in 1..@count %>
  <p><%= @bonus %></p>
<% end %>
```

Se o caracter # for inserido após o <%, significa que este código foi comentado e não será executado. Assim, <%#= @message %> não será executado pela inserção do caracter #.



Para você fazer em casa

Execute `http://localhost:3000/hello/` e verifique a saída.

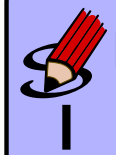
CSS: Acrescentando Estilos

O padrão para a Web W3C recomenda que os estilos configurem como o conteúdo será publicado. Isto ocorre inicialmente pela definição do conteúdo pela estrutura XHTML, e a apresentação será definida pelo CSS.

- Conteúdo é a coleção de termos para textos, imagens, vídeos, sons, animações e arquivos que serão exibidos.
- XHMTL (**eXtensible HyperTex Markup Language**) permite que cada elemento do conteúdo seja definido. É um cabeçalho ou parágrafo? É uma lista de itens, um link, ou uma imagem?
- CSS (**Cascading Style Sheets**) permitem que você configure como cada elemento marked-up do seu conteúdo é apresentado na página. O fonbodyte de um parágrafo será Helvetica ou Times? Ele será bold ou itálico.

As páginas exibidas até o momento podem ser melhoradas pela inclusão de CSS. O código abaixo será incluído no **app/assets/stylesheets/application.css** (que possui apenas comentários) para

```
body { font-family:sans-serif;
    }
h1 {font-family:serif;
   font-size: 24pt;
   font-weight: bold;
   color:#F00 ;
  }
```



Para você fazer em casa

Execute `http://localhost:3000/hello/` e verifique a saída.

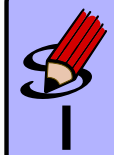
Os layouts são armazenados no diretório **app/views/layouts**. Inicialmente, vamos observar o arquivo **application.html.erb** que será utilizado quando não houverem layouts específicos para views. O seu conteúdo inicial contém declarações HTML5 DOC TYPE, com exibido abaixo.

```
<!DOCTYPE html>
  <html>
    <head>
      <title>Hello01</title>
        <%= csrf_meta_tags %>
        <%= csp_meta_tags %>
        <%= stylesheet_link_tag 'application', media: 'all',
                  data-turbolinks-track: 'reload' %>
        <%= javascript_include_tag 'application', data-turbolinks-track: 'reload' %>
    </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Pontos relevantes são os helpers **stylesheet_link_tag**, **javascript_include_tag**, **csrf_meta_tag** e **yield**.

Código HTML gerado.

```
<!DOCTYPE html><html><head>
   <title>Hello01</title>
   <meta name="csrf-param" content="authenticity_token" />
   <meta name="csrf-token" content="CZPoySi ... As7lRgXzFHT+g==" />
   <link rel="stylesheet" media="all" href="/assets/hello.self- ... 5.css?body=1" data-turbolinks-
         track="reload" />
   <link rel="stylesheet" media="all" href="/assets/application.self- ... 2.css?body=1" data-
         turbolinks-track="reload" />
   <script src="/assets/rails-ujs.self-... b.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/activestorage.self-... 7.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/turbolinks.self-... 8.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/action_cable.self-... 1.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/cable.self-... 9.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/hello.self-... 5.js?body=1" data-turbolinks-track="reload"></script>
   <script src="/assets/application.self-... e.js?body=1" data-turbolinks-track="reload"></script>
 </head>
 <body>
   <h1>Hello#index</h1>
   <p>This is a greeting from app/views/hello/index.html.erb</p>
   <p>This message came from the controller.</p>
   <p>This message came from the controller.</p>
   <p>This message came from the controller.</p>
 </body>
</html>
```



Para você fazer em casa

Verificar no browser o código HTML gerado.

The End!