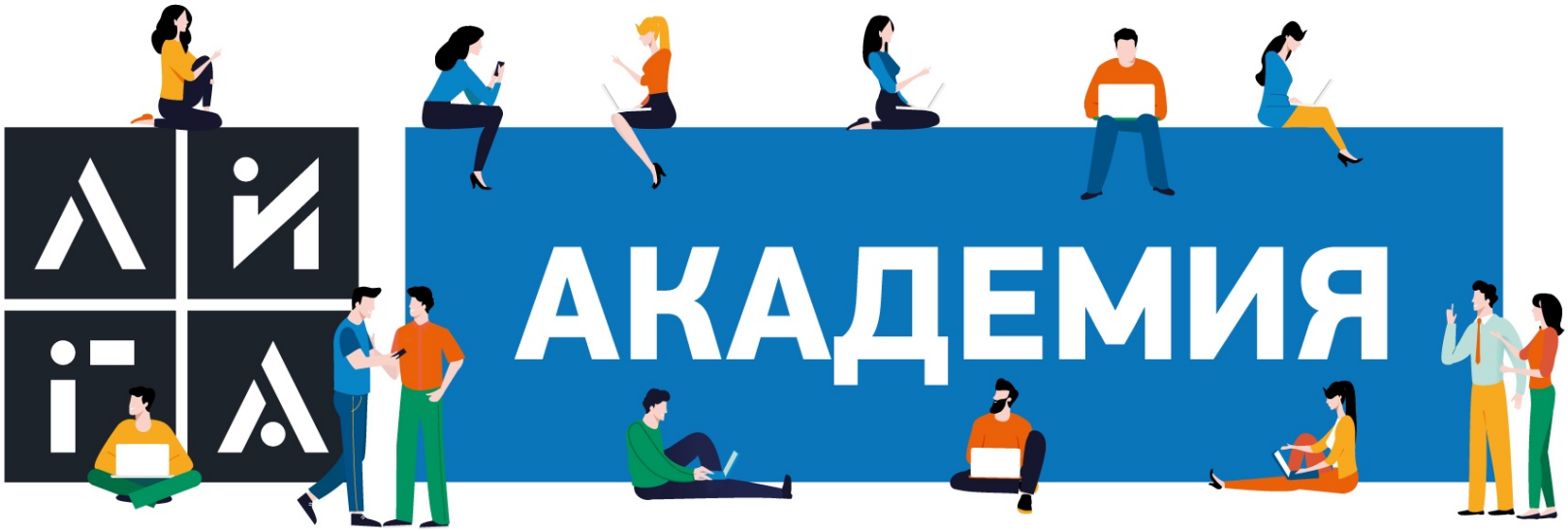

АКАДЕМИЯ

Влада Мамутова

TeamLead, разработчик

Опыт работы

В Лиге: **1 год**

Проекты

СЭД МВД

Frontend

в Ruby on Rails

20 ЛЕТ
МЕНЯЕМ МИР!

План занятия

1. Три кита фронтенда
2. Action View в Ruby on Rails
3. Templates, Layouts, Partials
4. Helpers
5. Coffee Script

Три кита фронтенда

HTML, CSS, JavaScript

Основа фронтенда

HTML

каркас

- Hypertext Markup Language
- язык разметки
- используется для структурирования и отображения веб-страницы и её контента

CSS

внешний вид

- Cascading Style Sheets
- язык иерархических правил (таблица стилей)
- используется для представления внешнего вида документа

JS

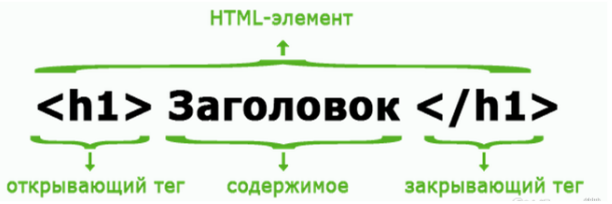
интерактивность

- Java Script
- язык сценариев, интерпретируемый, мультипарадигменный язык
- взаимодействие с пользователем, обработка данных, кэширование, анимация, формирование запросов к серверу

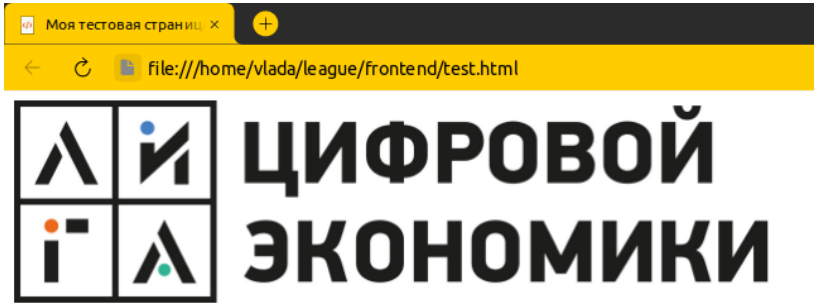
HTML

- Описывается тегами:

<h1>, <h2>, <p>, <table>,
, <ul>, <li>, <a>, <img>...




```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Моя тестовая страница</title>
    <link rel="stylesheet"
href="css/style.css">
    <script src="js/script.js"></script>
  </head>
  <body>
    <img src="logo.png" alt="Логотип">
  </body>
</html>
```



CSS

- Описывается набором правил

- Каждое правило состоит из селектора и блока объявлений - пар "свойство-значение"

```
селектор, селектор {
   свойство: значение;
   свойство: значение;
}
```

- Различают селекторы тегов, классов, идентификаторов, атрибутов...

```
<h1>Заголовок</h1>
<div id="content">
  <p class="error">Ошибка</a>
  <a href="#top">Наверх</a>
</div>
```

```
/* Селекторы тегов */
h1, h2, h3 {
   font-family: sans-serif;
}
/* Селектор класса */
.error {
    color: red;
}
/* Селектор идентификатора */
#content {
   height: 200px;
}
/* Селектор атрибута */
a[href^="#"] {
  font-weight: bold;
}
/* Селектор дочернего эл-та */
div > p {
  color: blue;
}
/* Селектор псевдоклассов */
a:active {
   color: blue;
}
```

JS

- Переменные: `var`, `let`, `const`
- Типы данных: `String`, `Number`, `Boolean`, `Array`, `Object`
- Операторы: математические, логические, операторы циклов...
- Функции: `function fun(arg)`
- События: `onclick`, `onchange`, `onmouseover` и др.

```
<script>
var colors = ["green", "yellow", "blue"];
var i = 0;

function changeColor() {
    document.body.style.background = colors[i];
    i++;
    if (i > colors.length - 1) {
        i = 0;
    }
}
</script>

<button onclick="changeColor();">Change background</button>
```

Методология БЭМ

• БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке.

• Блок - независимый интерфейсный компонент.
• Элемент — это составная часть блока.
• Модификатор — это свойство блока или элемента, задающее изменения в их внешнем виде или поведении.

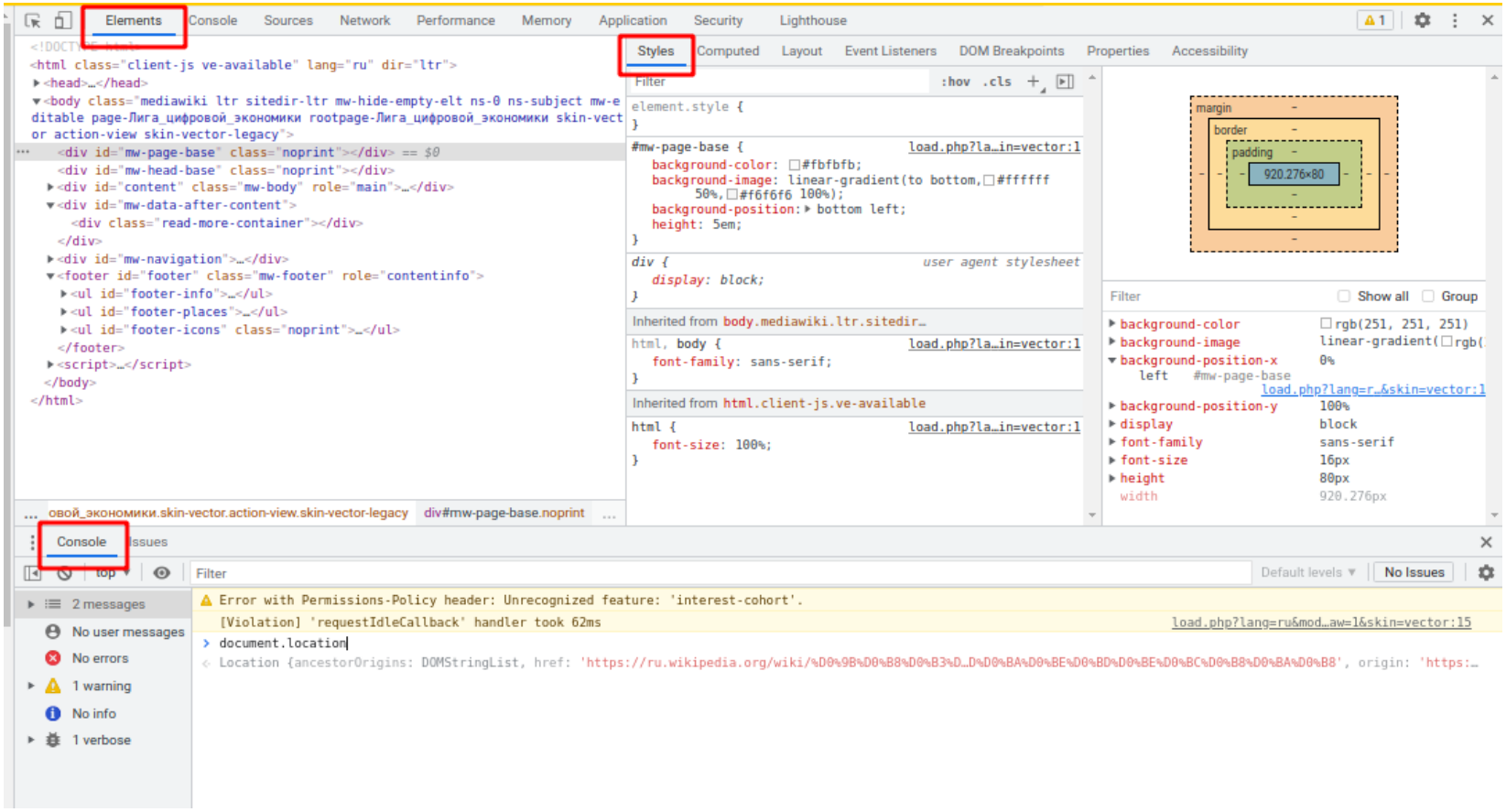
• В основе БЭМ лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности, повторно использовать существующий код, избегая «Copy-Paste», упроситить понимание кода и создать общий язык для всех участников проекта.

Документация:

https://ru.bem.info/methodology/

Панель разработчика

CTRL + SHIFT + I или **F12**

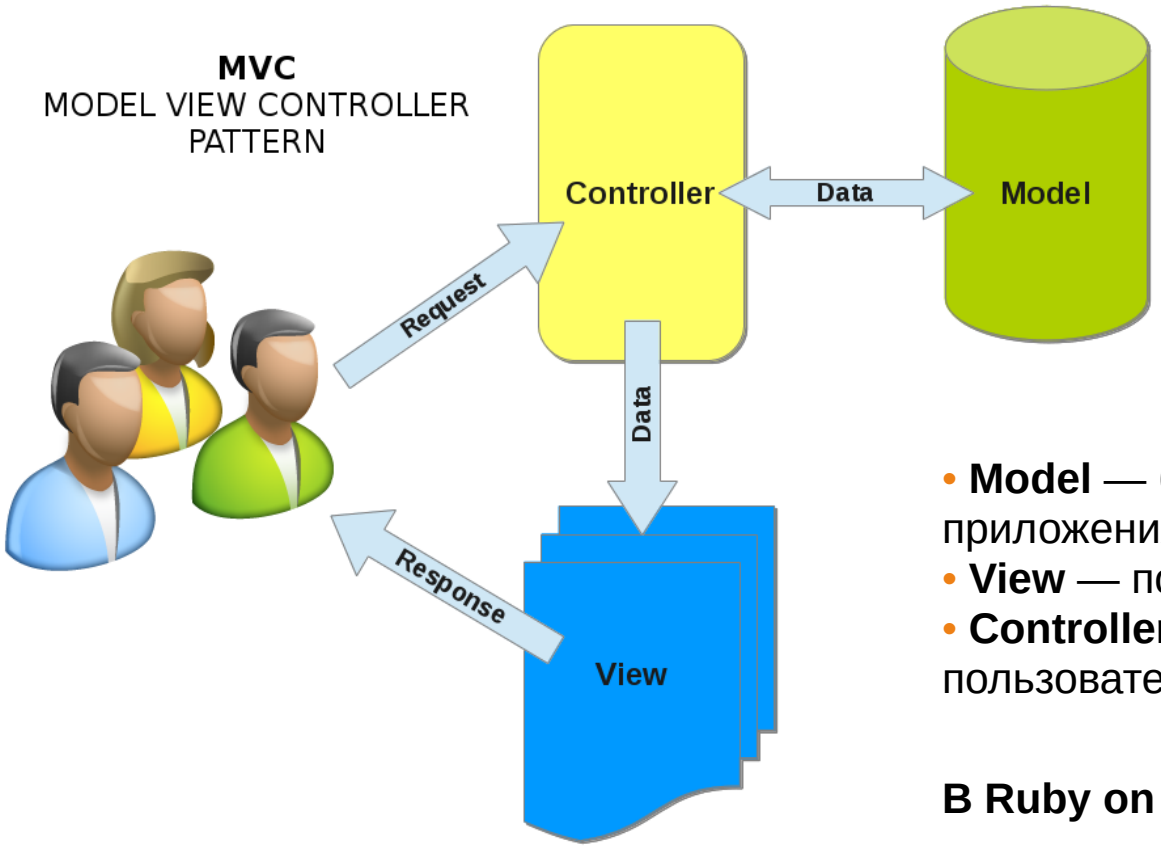


20 ЛЕТ
МЕНЯЕМ МИР!

Action View в Ruby on Rails

20 ЛЕТ
МЕНЯЕМ МИР!

Структура MVC-приложений

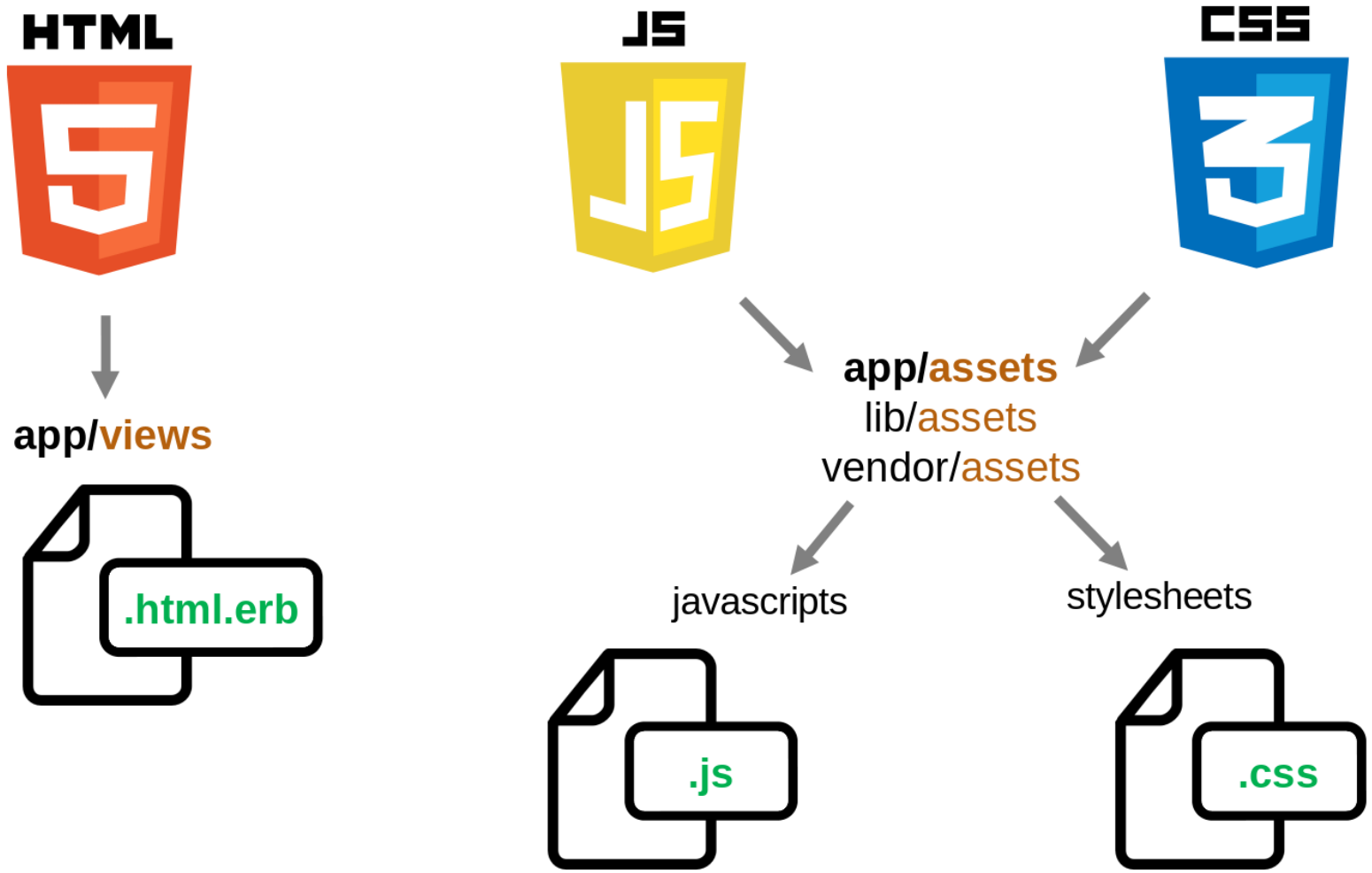


- **Model** — бизнес-модель, бизнес-логика приложения
- **View** — пользовательский интерфейс
- **Controller** — обработка событий пользовательского интерфейса

В Ruby on Rails:

- `Model → Active Record`
- `View → Action View`
- `Controller → Action Controller`

Организация файлов View



Компиляция файлов. Библиотека Sprockets

- **Sprockets** - это библиотека Ruby для компиляции и обслуживания веб-ресурсов.

- **Особенности**:
  - соединение ассетов → *уменьшение количество запросов, необходимых браузеру для отображения страницы*;
  - минимизация или сжатие ассетов → *уменьшение объёма трафика*;
  - использование ассетов на языках более высокого уровня (Sass для CSS, CoffeeScript для JavaScript, ERB для обоих) с дальнейшей прекомпиляцией → *повышение скорости и удобства работы программиста*.

- **Устройство работы**: Sprockets загружает файлы, указанные в *манифесте*, обрабатывает их при необходимости (для erb, sass, coffee), соединяет в отдельный файл и производит сжатие.

Файл манифеста *определяет, какие ассеты и в каком порядке необходимо сгруппировать в один единственный файл, который затем будет использоваться приложением.*

Компиляция файлов. Поиск файлов

1. Файлы манифестов:

app/assets/javascripts/application.js

```
...
//= require jquery
//= require lib/bootstrap
//= require_tree .
```

app/assets/stylesheets/application.css

```
/* ...
*= require_self
*= require_tree .
*/
```

2. Подключение файлов манифеста в индексный файл *.html.erb

app/views/layouts/application.html.erb

```
<html>
  <head>
    ...
    <%= stylesheet_link_tag
"application", media: "all" %>
    <%= javascript_include_tag
"application" %>
  </head>
  <body> ... </body>
</html>
```

3. Поиск файлов в порядке приоритета:

puts Rails.application.config.assets.paths

```
/usr/src/app/app/assets/javascripts
/usr/src/app/app/assets/stylesheets
/usr/src/lib/assets/javascripts
/usr/src/lib/assets/stylesheets
/usr/src/app/vendor/assets/javascripts
/usr/src/app/vendor/assets/stylesheets
```

Статика в production

- В среде production Sprockets использует *схему меток*.
- **Метки (fingerprinting)** — техника, реализующая зависимость имени файла от его содержимого.
- При изменении файла меняется имя файла.
- Для редко обновляемых (статичных) файлов схема меток является простым способом определить, идентичны ли две версии файла, даже если они находятся на разных серверах, загружены в разное время.
- Во время прекомпиляции из содержимого компилированных файлов генерируется SHA256 и вставляется в имена файлов, когда они записываются на диск. Эти имена меток используются хелперами Rails.

- Примеры генерации ссылок в production:

```
<%= javascript_include_tag "application" %>
# => <script src="/assets/application-908e25f4bf641868d8683022a5b62f54.js">
</script>

<%= asset_path "application.js" %>
# => "/assets/application-908e25f4bf641868d8683022a5b62f54.js"
```

Препроцессоры

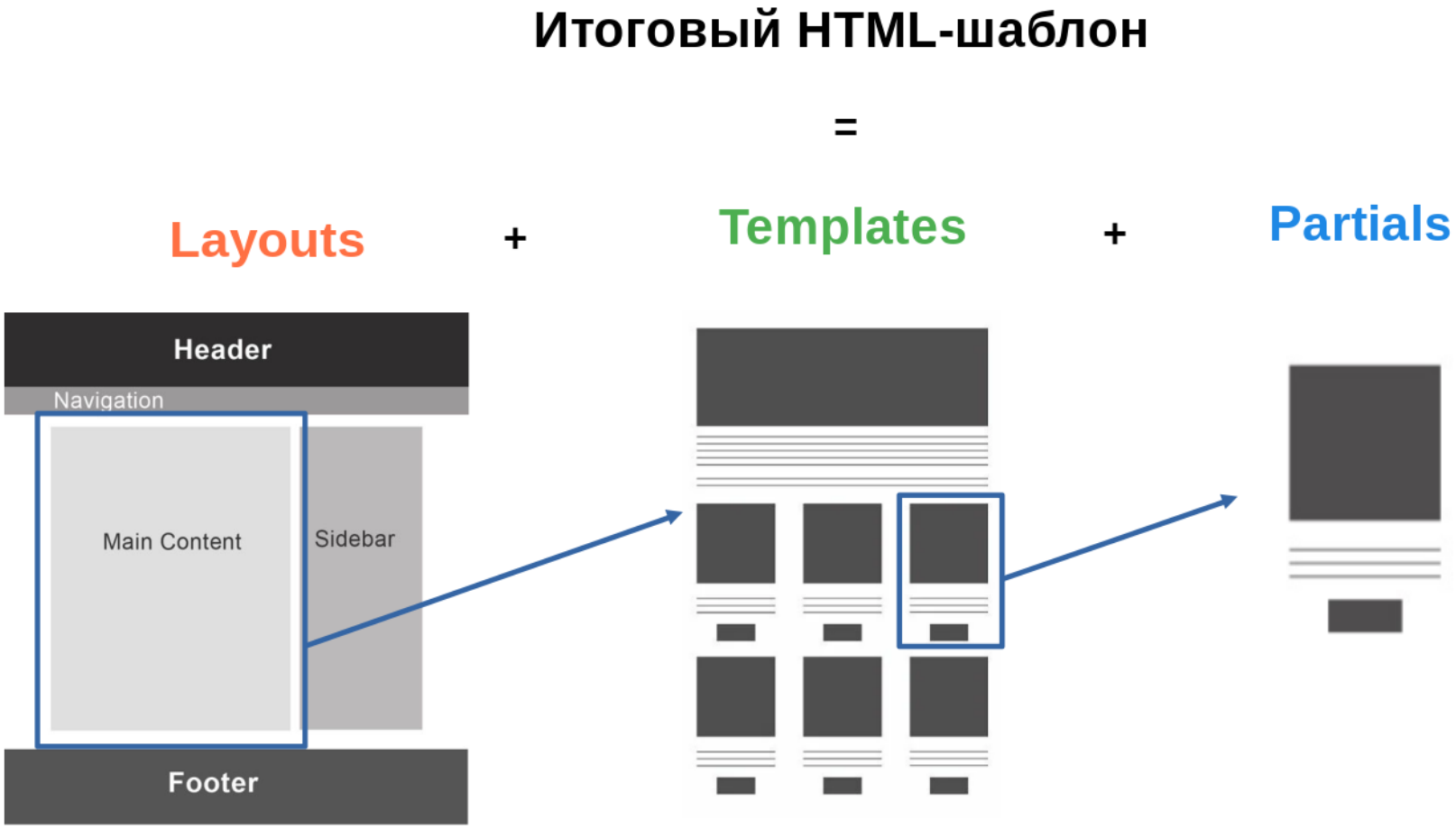
- После сбора ассетов Rails подготавливает их для макета, выполняя их *предварительную обработку*.
- Расширение, используемое в названии файла ассета, указывает на порядок обработки файла с помощью препроцессоров. Порядок обработки - *справа налево*.
- Наиболее употребительными являются препроцессорами являются: Sass для css (`.scss`), CoffeeScript для JS (`.coffee`), и ERb для HTML (`.erb`).

- Примеры порядка обработки файлов препроцессорами:

```
app/assets/stylesheets/books.scss.erb → ERB → SCSS → CSS
app/assets/javascripts/books.coffee.erb → ERB → CoffeeScript → JavaScript
app/assets/javascripts/books.html.erb → ERB → HTML
```

Templates, Partials, Layouts

Формирование HTML-документа



Язык описания шаблонов ERb

• **Embedded Ruby (ERb)** — язык описания шаблонов, который предоставляет возможность вставки в шаблоны исполняемого кода на языке Ruby.

• **Теги для включения кода Ruby в шаблон:**

<% %> - выполнение кода Ruby
<%= %> - включение выполненного кода Ruby в шаблон

```
<h1>Names of all the people</h1>

<%# Выводим список имён людей.%>
<% @people.each do |person| %>
  Name: <%= person.name %><br>
<% end %>
```

Templates. Соглашения по наименованию файлов

- Для каждого контроллера создаётся папка в `app/views`.
- Имя `view` совпадает с соответствующим `action` контроллера.

Action Controller app/controllers/books_controller.rb	Action View app/views/books/*
books_controller#**index**	app/views/books/**index**.html.erb
books_controller#**show**	app/views/books/**show**.html.erb
books_controller#**new**	app/views/books/**new**.html.erb
books_controller#**edit**	app/views/books/**edit**.html.erb
...	...
books_controller#**custom_action**	app/views/books/**custom_action**.html.erb

Templates. Пример шаблона

```
# app/controllers/books_controller.rb
```

```ruby
def index
  @books = Book.all
end
```

```
# app/views/books/index.html.erb
```

```erb
<h1>Book Store</h1>
<% if @books.blank? %>
  <p>В магазине нет книг :(</p>
<% else %>
  <div id='books'>
    <% @books.each do |book| %>
      <p><%= book.title %></p>
    <% end %>
  </div>
<% end %>

<p><%= link_to 'Добавить книгу', { action: :new } %></p>
```

Layouts

- Представляет макет страницы, в который встраивается шаблон.

- По соглашению макеты располагаются в `app/views/layouts`.

- По умолчанию название файла макета совпадает с именем контроллера (макет по умолчанию - `/app/views/layouts/application.html.erb`), но можно задавать собственные названия макетов.

- Для подстановки шаблона в макет используется ключевое слово `yield`.

- Можно дополнительно задавать именованные `yield` (`yield :nav_bar`). Тогда в конкретном шаблоне можно по-разному отрисовать данный компонент, указав его имя в `content_for` (`content_for :nav_bar`).

Layouts. Пример макета

```
# app/controllers/books_controller.rb
class BooksController < ApplicationController
  layout 'main'

  def index
    @books = Book.all
  end
  ...
end
```

```
# app/views/layouts/main.html.erb
<html>
  <head>
    <%= yield :title %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

```
# app/views/books/index.html.erb
<% content_for :title do %>
  <title>Book Store: Книги</title>
<% end %>

...
```

Partials

- Представляет фрагмент шаблона.
- Название файла начинается с нижнего подчеркивания: `_form.html.erb`.
- Для отрисовки partial в шаблоне используется метод `render`.

```
render(options = {}, locals = {}, &block) public
# options:
:partial, :file, :inline, :plain, :html, :body, :layout
  partial: 'item' # имя файла в текущей директории или путь от папки views
  layout: 'layout' # макет для partial из папки views
locals: <argument_hash> - хеш с аргументами, передаваемыми в partial;
# имя файла в :partial и в :layout указывается без нижнего подчёркивания!
```

Примеры:

```
<%= render item %> # рендеринг файла app/views/items/_item.html.erb
<%= render 'item', item: item %> # передача имени партиала и списка аргументов

# Указание дополнительных опций с обязательной передачей опции :partial.
<%= render partial: 'items/item', locals: { item: item } %>
<%= render partial: 'item', layout: 'item_layout', locals: { item: item } %>
```

Partials. Пример фрагмента

 # app/views/books/index.html.erb

```
...
<% @books.each do |book| %>
  <%= render partial: 'book', layout: 'book_layout', locals: { book: book } %>
<% end %>
...
```

 # app/views/books/_book_layout.html.erb

```
<div id="<%= dom_id book %>">
  <%= yield %>
</div>
```

 # app/views/books/_book.html.erb

```
<p><%= book.title %></p>
<p>Автор: <%= book.author %></p>
<p>Жанр: <%= book.genre.name %></p>
```

Helpers

Что такое хелперы?

- Представляют собой вспомогательные методы для рендеринга шаблонов.

- Бывают:
  - встроенными (например, `javascript_include_tag`, `text_field_tag`, `link_to`, `select_tag` и т. д.),
  - пользовательскими (модуль с методами в папке `app/helpers`).

Встроенные хелперы: ERb-теги

ERb tag	HTML tag
javascript_include_tag stylesheet_link_tag	<script/> <link/>
image_tag	<img/>
link_to button_to	<a/> <button/>
form_tag (form_for, form_with) label_tag (label) text_field_tag (text_field) check_box_tag (check_box) submit_tag (submit) select_tag (select) options_for_select, options_from_collection_for_select	<form/> <label/> <input/> ☐ <input type="submit"/> <select/> <option/>

Хелпер изнутри на примере stylesheet_link_tag

```
<%= stylesheet_link_tag "application", :media => "all" %>
```

```
# rails/actionview/lib/action_view/helpers/asset_tag_helper.rb
def stylesheet_link_tag(*sources)
  options = sources.extract_options!.stringify_keys
  ...
  sources_tags = sources.uniq.map { |source|
    href = path_to_stylesheet(source, path_options)
    …
    tag_options = {
      "rel" => "stylesheet",
      "crossorigin" => crossorigin,
      "href" => href
    }.merge!(options)
    ...
    tag(:link, tag_options)
  }.join("\n").html_safe
  ...
  sources_tags
end
```

Примеры использования stylesheet_link_tag

- Относительные ссылки:

```
stylesheet_link_tag("style")
stylesheet_link_tag "style.css"
# => <link href="/assets/style.css" media="screen" rel="stylesheet" />
```

- Абсолютные ссылки:

```
stylesheet_link_tag "http://www.example.com/style.css"
# => <link href="http://www.example.com/style.css" media="screen"
rel="stylesheet" />
```

- Передача дополнительных параметров:

```
stylesheet_link_tag "style", media: "all"
# => <link href="/assets/style.css" media="all" rel="stylesheet" />
```

- Подключение нескольких файлов:

```
stylesheet_link_tag "random.styles", "/css/stylish"
# => <link href="/assets/random.styles" media="screen" rel="stylesheet" />
#    <link href="/css/stylish.css" media="screen" rel="stylesheet" />
```

Хелпер link_to

- генерирует гиперссылку - отображает имя ссылки и ссылается на путь
- в html - тег <a>

```
link_to(name = nil, options = nil, html_options = nil, &block)

# Options:
data: <attribute_hash> # хэш для пользовательских атрибутов:
   confirm: 'Are you sure?' # переход по ссылке после подтверждения
   disable_with: <text> # отображение текста при disabled-ссылке
method: <http_verb> # указание HTTP-метода (например, 'post' или :post)
remote: true # выполнение Ajax-запроса вместо перехода по ссылке
```

** С Rails 7 вместо этих опций используются опции библиотеки Turbo.*

Пример:

```
<%= link_to "Improve Your Ruby Skills", "/ruby-book" %>
```

=>

```
<a href="/ruby-book">Improve Your Ruby Skills</a>
```

Хелпер `link_to`: генерация путей

- добавление `resources :books` в `routes.rb` создаёт хелперы путей:

```
books_path              => /books
new_book_path           => /books/new
book_path(:id)          => /books/:id
edit_book_path(:id)     => /books/:id/edit
...

books_url               => schema://host/books
new_book_url            => schema://host/books/new
book_url(:id)           => schema://host/books/:id
edit_book_url(:id)      => schema://host/books/:id/edit
...
```

*Для просмотра всех путей используйте **`rails routes`**.*

Хелпер link_to: примеры использования

- Варианты использования link_to для метода show

```
<%= link_to "Book", controller: "books", action: "show", id: @book %>
# => <a href="/books/1">Book</a>

<%= link_to "Book", book_path(@book) %>
# => <a href="/books/1">Book</a>

<%= link_to "Book", @book %>
# => <a href="/books/1">Book</a>
```

- link_to для метода index

```
<%= link_to "All books", books_path %>
```

- link_to для метода edit

```
<%= link_to "Edit Book", edit_book_path(@book) %>
```

Хелпер `link_to`: примеры использования

- Передача дополнительных параметров запроса:

```
<%= link_to "Novels of the 20th century",
            books_path(genre: "novel", year_from: "1900") %>
#=> <a href="/books?genre=novel&year_from=1900">Novels of the 20th century</a>
```

- Задание `html_options` (id и CSS-класс):

```
<%= link_to "More Books", books_path, id: "book-link", class: "custom-link" %>
# => <a href="/books" class="custom-link" id="book-link">More Books</a>
```

- Задание параметра `&block` для отображения содержимого ссылки, отличного от текста:

```
<%= link_to book_path(@book) do %>
  <%= image_tag @book.cover" %>
<% end %>
```

- Использование опций `:method` и `:data`

```
<%= link_to "Delete Book", @book, method: "delete", data: { confirm: "Are you
sure?", disable_with: "Processing" } %>
```

Хелперы форм

- генерируют формы - основной интерфейс для пользовательского ввода и отправки данных
- в html - тег `<form>`

Виды хелперов:

- `form_tag` - простая форма
- `form_for` - форма с привязкой к объекту модели
- `fields_for` - привязывает вложенные объекты модели без создания тега `<form>`
- `form_with` - заменяет `form_tag` и `form_for` в версиях Rails >= 5.1

Хелпер form_tag

- создаёт простую форму без привязки к моделям ActiveRecord

```
form_tag(url_for_options = {}, options = {}, &block)

# Options:
multipart: true # установка в отправляемых данных "multipart/form-data"
method: <method> # метод запроса (по умолчанию - POST)
authenticity_token: <authenticity_token> # выполнение запроса с токеном
аутентификации для предотвращения CSRF-атак
remote: true # позволяет отправлять данные формы через AJAX-запрос
```

Пример:

```
<%= form_tag '/posts' do %>
  <%= search_field_tag 'name' %>
  <%= submit_tag 'Save' %>
<% end %>
```

=>

```
<form action="/posts" method="post">
  <input id="name" name="name"
         type="search" />
  <input type="submit" name="commit"
         value="Save" />
</form>
```

Хелперы для использования внутри `form_tag`

- `label_tag`
- `text_field_tag`
- `check_box_tag`
- `select_tag`
- `hidden_field_tag`
- `submit_tag`

Полный список хелперов можно найти в документации по FormTagHelper:

https://api.rubyonrails.org/v7.0.3/classes/ActionView/Helpers/FormTagHelper.html

Хелперы для использования внутри `form_for`: примеры

```ruby
# текстовое поле
text_field_tag(name, value = nil, options = {})

# Options:
disabled: true # отключает возможность редактирования пользователем
size: <symbol_number> # размер поля в длину
maxlength: <max_symbols> # ограничение макс. количества символов для ввода
placeholder: <text> # подсказка для поля
```

```ruby
# кнопка для отправки формы
submit_tag(value = "Save changes", options = {})

# Options:
disabled: true # отключает возможность редактирования пользователем
data: <attribute_hash> # хэш для пользовательских атрибутов:
   confirm: 'Are you sure?' # отправка формы по после подтверждения
   disable_with: <text> # отображение текста для disabled-кнопки
```

** С Rails 7 вместо опций `confirm`, `disable_with` используются опции библиотеки Turbo.*

Хелпер check_box_tag

- генерирует чекбокс - элемент управления, предоставляющий пользователю выбор опции
- в html - тег <input checked="checked">

```
check_box_tag(name, value = "1", checked = false, options = {})

# Options:
disabled: true # для отключения возможности редактирования пользователем
```

Пример:

```
<%= check_box_tag 'receive_email', 'yes', true, class: 'simple-check-box' %>
```

=>

```
<input checked="checked" class="simple-check-box" id="receive_email"
name="receive_email" type="checkbox" value="yes" />
```

Хелпер hidden_field_tag

- генерирует скрытое поле ввода, которое предоставляет возможность передать данные в запросе без необходимости иметь видимое для пользователя поле
- в html - тег `<input type="hidden">`

```
hidden_field_tag(name, value = nil, options = {})
```

Пример:

```
<% = hidden_field_tag 'user_id', 'VUBJKB23UIVI1UU1VOBVI@' %>
```

=>

```
<input id="token" name="token" type="hidden" value="VUBJKB23UIVI1UU1VOBVI@" />
```

Хелпер select_tag

- генерирует выпадающий список для выбора опций (как одной, так и нескольких)
- в html - тег <select>

```
select_tag(name, option_tags = nil, options = {})

# Options:
multiple: true # для выбора нескольких опций
disabled: true # отключает возможность редактирования пользователем
include_blank: true или <text> # создаёт пустую опцию (может содержать
название, но value будет пустым)
prompt: <text> # отображает пустую опцию с подсказкой для выбора значения
```

Пример:

```
<%= select_tag 'people', raw('<option value="david">David</option>') %>
```

=>

```
<select name="people" id="people"><option value="david">David</option>
</select>
```

Хелпер options_for_select

- генерирует список опций для select_tag
- в html - тег <option>

```
options_for_select(container, selected = nil)

# container - hash, array, enumerable, пользовательский тип, которые отвечают
на методы first (для атрибута name) и last (для атрибута value)
# selected - одно или несколько значений, которые будут выбраны по умолчанию
(в те опции, у которых value точно совпадёт с одним из представленных в
selected, будет добавлен атрибут selected)
```

Пример:

```
<%= options_for_select([['Lisbon', 1], ['Madrid', 2]], 2) %>
```

=>

```
<option value="1">Lisbon</option>
<option value="2" selected="selected">Madrid</option>
```

Хелпер form_for

- создаёт форму с привязкой к модели ActiveRecord
- внутри формы используются теги без постфикса _tag (в отличие от хелперов для form_tag)

```
form_for(record, options = {}, &block)

# Options
url: <url> # url страницы, на которые будут отправлены данные формы (по
умолчанию - на страницу с формой)
namespace: <name> # пространство имён для гарантии уникальности элементов
формы
method: <method> # метод запроса (по умолчанию - POST)
authenticity_token: <authenticity_token> # выполнение запроса с токеном
аутентификации для предотвращения CSRF-атак
remote: true # позволяет отправлять данные формы через AJAX-запрос
html: <attribute_hash> # хеш с дополнительными HTML-атрибутами для формы
```

Хелпер form_for: примеры использования

Пример с передачей названия модели (в виде строки или символа):

```
<%= form_for :person do |f| %>
  First name: <%= f.text_field :name %><br />
  Biography : <%= f.text_area :biography %><br />
  Admin?    : <%= f.check_box :admin %><br />
  <%= f.submit %>
<% end %>
```

```
<form action="/action_url" accept-charset="UTF-8" method="post">
  <input name="utf8" type="hidden" value="✓">
  <input type="hidden" name="authenticity_token" value="token">
    First name: <input type="text" name="user[name]" id="user_name">
    <br>
    Biography: <textarea name="user[biography]" id="user_biography">
</textarea><br>
    Admin?: <input name="user[admin]" type="hidden" value="0">
  <input type="checkbox" value="1" name="user[admin]" id="user_admin"><br>
  <input type="submit" name="commit" value="Submit User">
</form>
```

Хелпер form_for: примеры использования

Пример с передачей объекта модели:

```
<%= form_for @user do |f| %>
  First name: <%= f.text_field :name %><br />
  Biography : <%= f.text_area :biography %><br />
  Admin?    : <%= f.check_box :admin %><br />
  <%= f.submit %>
<% end %>
```

- где @user - объект модели User

```
=> @user = User.find(1)
# => #<User id: 1, name: "Петя", biography: "Работает в Лиге", admin: true>
```

Хелперы для использования внутри form_for

- хелперы вызываются у объекта FormBuilder
- большинство хелперов являются обёрткой методов у FormHelper
- примеры: label, text_field, submit др. (без постфикса _tag)

Пример определения text_field

```
text_field(method, options = {})

# method - поле/метод модели
# options - дополнительные опции html
```

Пример использования text_field

```
<%= form_for @user do |f| %>
  <%= f.text_field :name %>
<% end %>
```

=

```
<form action="/users/1" method="post">
  <input type="text" name="user[name]"
         value="Петя">
</form>
```

Документация по FormBuilder с полным перечнем хелперов:

https://api.rubyonrails.org/v7.0.3/classes/ActionView/Helpers/FormBuilder.html

Взаимозаменяемость хелперов форм

```
<%= form_for @user do |form| %>
  <%# Использование хелпера ActionView::Helpers::FormBuilder %>
  <%= form.text_field :name %>

  <%# => <input type="text" value="Петя" name="user[name]" id="user_name"> %>


  <%# Использование хелпера ActionView::Helpers::FormHelper %>
  <%= text_field :user, :name %>

  <%# => <input type="text" value="Петя" name="user[name]" id="user_name"> %>


  <%# Использование хелпера ActionView::Helpers::FormTagHelper %>
  <%= text_field_tag 'user[name]', @user.name %>

  <%# => <input type="text" name="user[name]" id="user_name" value="Петя"> %>
<% end %>
```

Хелпер form_with

- создаёт тег формы на основе смешивания URL-адресов, областей или моделей
- предназначен для замены двух методов form_for и form_tag, начиная с Rails 5

```
form_with(model: nil, scope: nil, url: nil, format: nil, **options) public

# Options
model: <object> # объект модели для определения :url и :scope, а также для
заполнения значений полей ввода.
skope: <skope> # область действия для префикса имен полей ввода и,
следовательно, способ группировки отправленных параметров в контроллерах.
url: <url> # url страницы, на которые будут отправлены данные формы (по
умолчанию - на страницу с формой)
format: <format> # формат запрашиваемых данных по маршруту (например, :json).
```

Пользовательские хелперы

- **Модуль** вспомогательных методов в `app/helpers`.

- **Decorator** - разгружает модель и зачастую хелперы от избыточной логики представления (например, от часто вызываемых в представлениях методов преобразования данных.

В этом случае обычно используется draper.
Документация: https://github.com/drapergem/draper

- **Serializer** - разгружает контроллер и модель от избыточной логики представления, используется для внешней сериализации (для API).

Документация: https://github.com/rails-api/active_model_serializers

Coffee Script

20 ЛЕТ
МЕНЯЕМ МИР!

Итоги

- Рассмотрели, из чего состоит фронтенд
- Разобрались, как организованы файлы пользовательского интерфейса в проекте Ruby on Rails: где хранятся, как компилируются, как происходит поиск
- Познакомились с Action View: Templates, Layouts, Partials
- Узнали, что такое хелперы: как пользоваться встроенными тегами и как писать собственные хелперы
- Создали пользовательский интерфейс для приложения Book Store
- Рассмотрели основы Coffee Script

ВОПРОСЫ?

20 ЛЕТ
МЕНЯЕМ МИР!

Всем спасибо!