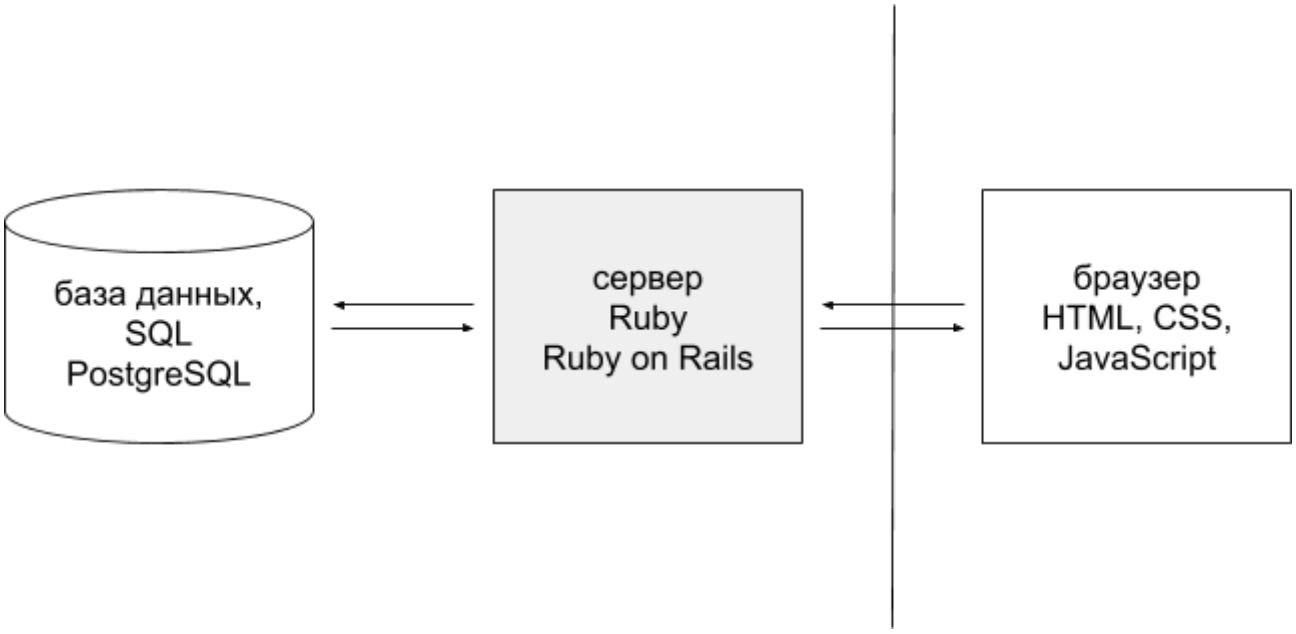
Урок 7

Frontend

Assets Pipeline. Гем Webpacker. Yarn. Webpack. Гем Foreman. Фреймворк Bootstrap.

- Assets Pipeline
  - Соединение ресурсов
  - Подключение ресурсов
  - Организация ресурсов
  - Подключение JS-кода из гемов
- Гем Webpacker
  - Менеджер зависимостей Yarn
  - Сборка проектов Webpack
  - Установка гема Webpacker
  - Подключение Webpacker к проекту
  - Гем Foreman
  - Компонентный подход
  - CSS-фреймворк Bootstrap
- Домашнее задание
- Дополнительные материалы
- Используемая литература

На прошлых уроках мы разбирали устройство Rails-приложений со стороны backend. Генерация шаблонов, обработка маршрутов, валидация моделей − все эти операции обрабатываются на стороне сервера. Клиент, то есть браузер пользователя, об этом ничего не «знает». Однако современные приложения обычно требуют выполнения кода и на стороне клиента. Для этого был создан язык JavaScript. Кроме того, оформление страницы осуществляется при помощи каскадных таблиц стилей CSS, шрифтов и небольших графических изображений. Все эти файлы называются *ассетами* (assets).



Код приложения сосредоточен на серверной стороне, поэтому перед передачей его клиенту он обрабатывается. Например, для разработки JavaScript-кода используются CoffeeScript или ES6/ES2017. Перед отправкой клиенту они преобразуются в JavaScript. Шаблоны каскадных таблиц в форматах SASS, SCSS или Stylus преобразуются в CSS.

В development-окружении коды JavaScript и CSS во время разработки постоянно меняются, поэтому Rails транслирует код каждый раз. Для production-среды Rails предоставляет инструмент для прекомпиляции кода.

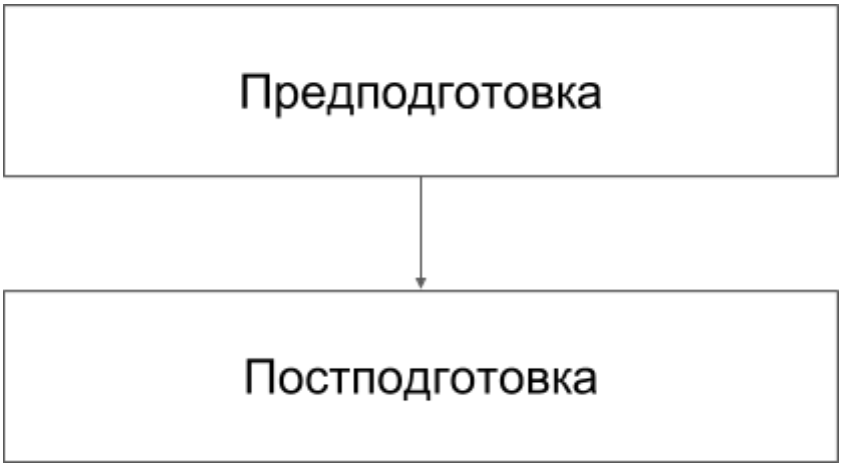
В современном Ruby on Rails есть два подхода для организации ассетов: Assets Pipeline, реализованный через гем Sprockets, и компонентный подход к сборке ассетов, реализованный через гем Webpacker.

Assets Pipeline

Assets Pipeline организован в виде своеобразного конвейера, в котором ассеты проходят пред- и постподготовку. Предподготовка включает преобразование языка шаблона (SASS, CoffeeScript), который не известен браузеру, в понятный ему формат (CSS, JavaScript).

Если JavaScript-кода много, выгоднее, чтобы клиент загрузил его только один раз при первом посещении сайта, а в дальнейшем использовал версию из кеша браузера. В процессе постподготовки

Assets Pipeline собирает множество js- и ccs-файлов в один, а затем минифицирует их: убирает комментарии, лишние переводы строк, пробелы и назначает переменным более короткие имена.



Таким образом, Assets Pipeline представляет собой библиотеку для объединения ресурсов JavaScript и CSS. В его задачу входит:

- Организация ассетов − расположение файлов по соглашению.
- Сжатие файлов, чтобы уменьшить время их отдачи клиенту.
- Сборка многочисленных файлов в один, чтобы уменьшить количество запросов от клиента.
- Организация обработки файлов для разных окружений: максимальная скорость в production-окружении и режим отладки в development-окружении.

Assets Pipeline реализован в геме sprockets-rails. В Rails он по умолчанию подключен в Gemfile, если при развертывании проекта rails new не использовался параметр --skip-sprockets. Там же по умолчанию подключены гемы sass-rails для SASS и uglifier для сжатия JS-кода.

В цепочку предподготовки uglifier подключается в файле config/application.rb. Настройки находятся в соответствующих файлах в директории config/environments. Например, в config/environments/production.rb можно обратить внимание на следующие строки:

config/environments/production.rb

```
Rails.application.configure do
  ...
  # Compress JavaScripts and CSS.
  config.assets.js_compressor = :uglifier
  ...
  # Do not fallback to assets pipeline if a precompiled asset is missed.
  config.assets.compile = false
  ...
end
```

Здесь указаны настройки отключения компиляции и используемый uglifier (программа для минификации JS-кода). В production-окружении, как было указано выше, будет использоваться прекомпиляция.

Соединение ресурсов

Sprockets соединяет все JavaScript-файлы в один «главный» .js и все CSS-файлы в один «главный» .css. В production-окружении в имя каждого файла Rails вставляет специальную метку в виде MD5-хеша. Так он позволяет браузеру кэшировать файлы и скачивать их новые версии только тогда, когда файл изменяется. Индикатором изменения файла и служит метка MD5.

Выполним в консоли команду rails assets:precompile:

```
I, [2018-01-21T17:37:02.361451 #86800]  INFO -- : Writing
/www/blog/public/assets/application-b80b8dc311e201e9a6650f7b0dfe48178ac006da8e27
3af7fb1d7d77b45d9a41.js
I, [2018-01-21T17:37:02.361687 #86800]  INFO -- : Writing
/www/blog/public/assets/application-b80b8dc311e201e9a6650f7b0dfe48178ac006da8e27
3af7fb1d7d77b45d9a41.js.gz
I, [2018-01-21T17:37:02.365369 #86800]  INFO -- : Writing
/www/blog/public/assets/application-f0d704deea029cf000697e2c0181ec173a1b47464546
6ed843eb5ee7bb215794.css
I, [2018-01-21T17:37:02.365738 #86800]  INFO -- : Writing
/www/blog/public/assets/application-f0d704deea029cf000697e2c0181ec173a1b47464546
6ed843eb5ee7bb215794.css.gz
```

Sprockets создал 4 файла, добавив в конец каждого метку хеша – fingerprint. Файлы с расширением .gz – это архивированные версии файлов, сжатые алгоритмом gzip, который понимают все популярные браузеры.

Если в gz-файлах нет необходимости, их можно отключить, установив параметру config.assets.gzip значение false.

config/environments/production.rb

```
...
config.assets.gzip = false
...
```

Подключение ресурсов

Rails автоматически генерирует правильные пути с нужными метками для собранных ресурсов с помощью функций stylesheet_link_tag и javascript_include_tag. Посмотрим на содержимое файла app/views/layouts/application.slim.html.

app/views/layouts/application.slim

```
doctype html
html
  head
    title MyBlojek
    = csrf_meta_tags
    = stylesheet_link_tag 'application', media: 'all'
    = javascript_include_tag 'application'
```

```
body
  = render 'header'
  #content
    = yield
```

Указанные в теге <head> методы javascript_include_tag и stylesheet_link_tag подключают файлы application.js и application.css.

Организация ресурсов

Ресурсы для Assets Pipeline по умолчанию размещаются в одной из трёх директорий:

- **app/assets** предназначена для хранения ресурсов, которые принадлежат приложению и изготовлены специально для него: изображений, JavaScript, шрифтов и т.п.
- **lib/assets** предназначена для хранения кода библиотек: собственных, которые не вписываются в сферу применения приложения, или общих для нескольких приложений.
- **vendor/assets** предназначена для хранения ресурсов, принадлежащих сторонним субъектам: кодов плагинов JavaScript и CSS-фреймворков.

Помимо этих папок можно добавлять дополнительные, указывая путь к ним в параметре config.assets.paths. Например, при использовании гема webpacker js-пакеты устанавливаются в папку node_modules, которая прописывается в config.assets.paths.

config/initializers/assets.rb

```
...
Rails.application.config.assets.paths << Rails.root.join('node_modules')
...
```

Подключение JS-кода из гемов

Для подключения ресурсов из гемов к приложению в манифесте используют специальный комментарий вида: //= require your_library_or_file. Манифест − это файл, который является основной точкой входа ресурса. Обычно это application.css и application.js.

app/assets/javascripts/application.js

```
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Здесь подключаются библиотеки jquery и jquery_ujs, а также все файлы в директории, в которой расположен файл application.js.

Гем Webpacker

Подход, принятый при классическом использовании гема Sprockets, сегодня считается устаревшим.

Сборка единого css- и js-файла для всех приложений, даже если не на всех страницах требуются все css-инструкции и js-компоненты, − это довольно расточительный подход. На смену ему пришел новый компонентный подход, в котором страница собирается из отдельных компонентов: набора HTML, CSS и JS-кода. Он позволяет загружать только тот код, который необходим для выполнения на текущей странице.

Кроме того, сборка ассетов посредством Assets Pipeline происходит довольно медленно. Можно значительно ускорить процесс, если воспользоваться современными frontend-сборщиками.

Менеджер зависимостей Yarn

Долгое время в JS-мире в качестве менеджера пакетов использовался npm. Недавно ему на смену пришел yarn, который реализует идеи, схожие с менеджером ruby-гемов bundler. В частности, он позволяет зафиксировать версии пакетов в специальном файле yarn.lock. Это гарантирует одинаковые версии библиотек на машинах разработчиков и серверах.

Для установки пакетов предназначен файл package.json. Его можно редактировать самостоятельно, а можно добавлять в него пакеты при помощи команды yarn add.

```
yarn add bootstrap
```

Чтобы иметь возможность использовать команду yarn, необходимо установить Yarn.

```
npm install -G yarn
```

Сборка проектов Webpack

Сборщик Webpack играет в мире JS такую же роль, какую играет устаревший Assets Pipeline в RoR-приложениях. Задача Webpack заключается в организации ассетов, их минификации и трансляции ES6-кода, а также в разделении окружений для разработки, тестирования и эксплуатации.

Помимо этого Webpack позволяет запускать сервер, производящий все эти действия, «на лету» без перезапуска.

Остается только вопрос интеграции JS и CSS-кода, сгенерированного Webpack с Ruby on Rails. Эту задачу выполняет специальный гем Webpacker. Он развивался независимо, но с версии 5.1 включен в состав Ruby on Rails.

Установка гема Webpacker

Чтобы включить поддержку Webpacker, необходимо развернуть проект с использованием параметра --webpacker:

```
rails new blog --skip-test --skip-turbolinks --skip-spring --skip-coffee
--skip-action-cable
 --webpack --database=postgresql
```

Этот параметр добавляет в Gemfile строку с подключением гема:

Gemfile

```
...
gem 'webpacker'
...
```

Кроме того, в результате постустановки проекта запускается генератор rails webpacker:install.

```
rails webpacker:install
```

Подключить гем webpacker и запустить генератор webpacker:install можно и самостоятельно. Особенно, если проект создавался без параметра --webpack.

Генератор создает несколько файлов и папок:

- **config/webpacker.yml** – конфигурационный файл Webpack;
- **config/webpack** – конфигурационные js-файлы для каждого из окружений;
- **.babelrc** – конфигурационный файл библиотеки Babel.js, переписывающей ES6-код в ES5;
- **app/javascript** – папка с компонентами и js-кодом точки или точек входа приложения.

Подключение Webpaker к проекту

Webpack позволяет собирать проект в фоне. Для этого в папке bin есть утилита webpack-dev-server. Она позволяет запустить Webpack-сервер, который в фоне пересобирает JS-код. Сервер необходимо запускать одновременно с Rails-сервером, например в двух разных консолях.

```
rails s
```

```
./bin/webpack-dev-server
```

При этом Rails-сервер запускается на 3000-порту, а Webpack-сервер – на 3035. Для включения поддержки Webpacker на уровне приложения в представлениях необходимо заменить хелперы javascript_include_tag и stylesheet_link_tag на javascript_pack_tag и stylesheet_pack_tag соответственно.

app/views/layouts/application.html.slim

```
doctype html
html
  head
    title
      | Blog
    = csrf_meta_tags
    = javascript_pack_tag 'application'
    = stylesheet_pack_tag 'application'
  body
    = render 'shared/header'
    = yield
```

Для проверки работоспособности добавим в файл app/javascript/packs/application.js проверочный код.

app/javascript/packs/application.js

```
import "./application.css"

window.onload = function() {
  document.body.insertAdjacentHTML("afterbegin", "Hello Webpack!");
};
```

Кроме того, рядом с файлом application.js создадим файл application.css для каскадных таблиц стилей.

app/javascript/packs/application.css

```
body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  font-size: 16px;
  line-height: 24px;
}
```

Теперь, если обратимся к главной странице сайта http://localhost:3000/, обнаружим в начале страницы фразу «Hello Webpack!», которая была добавлена JS-кодом.

Гем Foreman

Для функционирования приложения с использованием Webpacker приходится запускать два сервера. Чтобы свести эти две команды к одной, следует воспользоваться гемом Foreman.

```
gem install foreman
```

В корне проекта следует создать конфигурационный файла Procfile. В нем задаются команды, которые необходимо стартовать при запуске foreman.

Procfile

```
server: bin/rails server
assets: bin/webpack-dev-server
```

После этого запуск обоих серверов сводится к одной команде foreman start.

```
foreman start
```

Следует иметь в виду, что Foreman по умолчанию запускает приложение на 5000-порту: http://localhost:5000.

Компонентный подход

Чтобы использовать на каждой странице только необходимые на ней компоненты JS и CSS, следует реорганизовать содержимое папки app/javascript.

```
blog
└── app
        └── javascript
                ├── components
                │       ├── page
                │       │       ├── _page.html.slim
                │       │       ├── page.css
                │       │       └── page.js
                ├── init
                │       ├── index.css
                │       └── index.js
                └── packs
                        └── application.js
```

Файл application.js будет служить точкой входа и не должен включать никакой JS-код, кроме инструкций import. Весь существующий JS и CSS-код сосредоточим в папке app/javascript/init.

app/javascript/packs/application.js

```
import "init";
import "components/page/page";
```

app/javascript/init/index.js

```
import "./index.css";
window.onload = function() {
  document.body.insertAdjacentHTML("afterbegin", "Hello Webpack!");
};
```

app/javascript/init/index.css

```
body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  font-size: 16px;
  line-height: 24px;
}
```

app/javascript/components/page/page.js

```
import "./page.css";

window.onload = function() {
```

```
  let elem = document.getElementById('hello');
  console.log(elem.innerText);
  document.body.insertAdjacentHTML("afterbegin", elem.innerText);
}
```

app/javascript/components/page/page.css

```
.page {
  height: 100vh;
  width: 700px;
  margin: 0 auto;
  overflow: hidden;
}
```

app/javascript/components/page/_page.html.slim

```
.page#hello
  = yield
```

Чтобы Ruby on Rails смог обнаружить partial-представление, необходимо в ApplicationController добавить вызов метода prepend_view_path, передав ему путь до папки app/javascript.

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  ...
  prepend_view_path Rails.root.join('app/javascript')
  ...
end
```

Теперь можно воспользоваться компонентом и подключить partial-шаблон _page.html.slim к одной из страниц, например views/home/index.html.slim.

app/views/home/index.html.slim

```
h1 Home#index
p Find me in app/views/home/index.html.slim

= render "components/page/page" do
  p Hello from our first component!
```

CSS-фреймворк Bootstrap

Для оформления страниц используются каскадные таблицы стилей. Это технология, которая требует отдельного изучения. Для каскадных таблиц стилей созданы специальные фреймворки, облегчающие процесс оформления страниц. Один из самых известных − Bootstrap.

Bootstrap предоставляет 12-колоночную сетку и множество готовых CSS-компонентов: меню, кнопки, элементы управления форм и пр. Одна из зависимостей фреймворка − библиотека jQuery, которую также придется установить.

Для подключения bootstrap воспользуемся командой yarn add.

```
yarn add popper.js
yarn add jquery
yarn add bootstrap
```

После этого Bootstrap следует подключить в init-файлы app/javascript/init.

app/javascript/init/index.css

```
@import "bootstrap";
...
```

app/javascript/init/application.js

```
import "bootstrap";
import "./index.css";
...
```

Теперь в шаблонах сайта есть возможность использовать классы Bootstrap.

app/views/home/index.html.slim

```
h1 Home#index
p.alert.alert-primary Find me in app/views/home/index.html.slim

= render "components/page/page" do
  p.alert.alert-success Hello from our first component!
```

Домашнее задание

1. Настроить проект на использование гема Webpacker.
2. Подключить к проекту Bootstrap. Использовать его возможности для оформления страниц.
3. *Реализовать рендер ошибок в формах приложения без перезагрузки страницы с помощью AJAX.

   «*» задание повышенной сложности (по желанию)

Дополнительные материалы

1. Assets Pipeline.
2. Подробнее про SASS.
3. Webback.
4. Гем Webpacker.
5. CSS-фреймворк Bootstrap.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Assets Pipeline
2. Evil Front Part 1.
3. Evil Front Part 2.
4. Evil Front Part 3.
5. Майкл Хартл. Ruby on Rails для начинающих. Изучаем разработку веб-приложений на основе Rails.
6. Ruby on Rails Guides.
7. Obie Fernandez. The Rails 5 Way.