React in Rails

gem webpacker

Rails 5.1

Webpack in Rails

From Scratch

```
rails new myapp --webpack                 // webpack stand alone
rails new myapp --webpack=react           // react ready

// and also
rails new myapp --webpack=angular     // angular ready
rails new myapp --webpack=vue         // etc...
rails new myapp --webpack=elm
```

Webpack in Rails

On an existing project

```
# Gemfile
# [...]
gem 'webpacker', '~> 3.0'
```

```
rails webpacker:install
```

Then to setup a particular JS framework preconfig:

```
rails webpacker:install:react  # OR angular OR vue OR elm
```

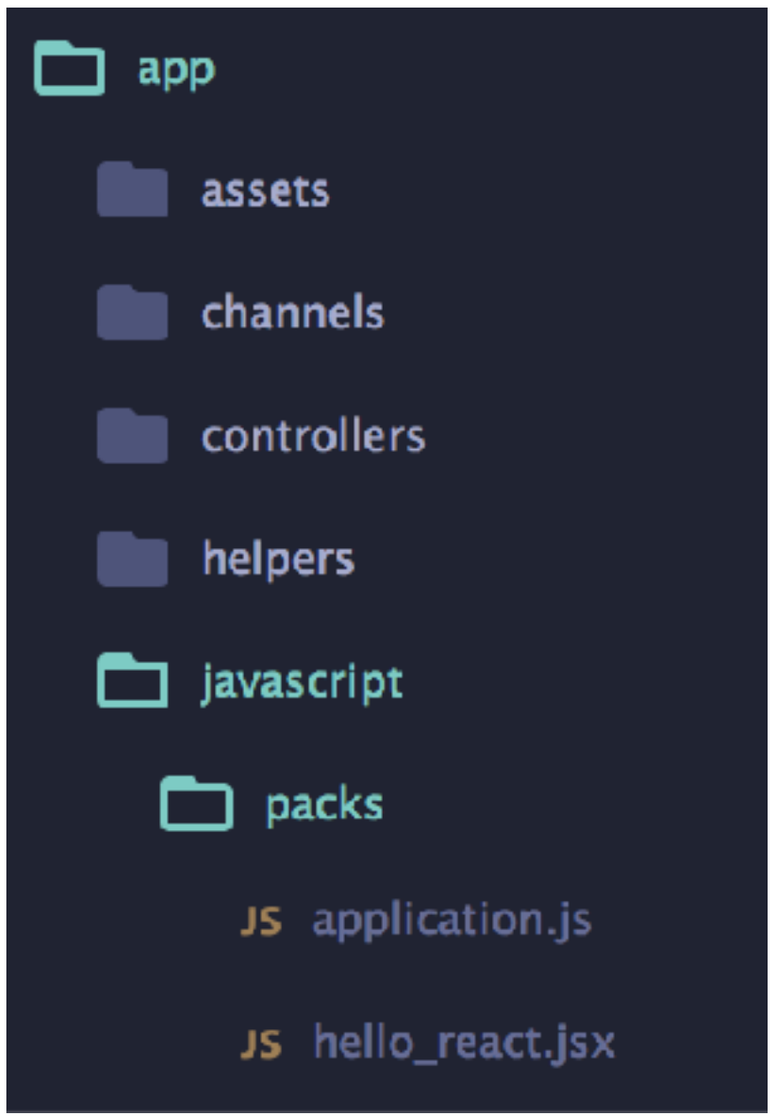
Webpack in Rails

With a Le Wagon minimal rails template

```
rails new \
  --database postgresql \
  --webpack=react \
  -m https://raw.githubusercontent.com/lewagon/rails-templates/
master/minimal.rb \
  CHANGE_THIS_TO_YOUR_RAILS_APP_NAME
```

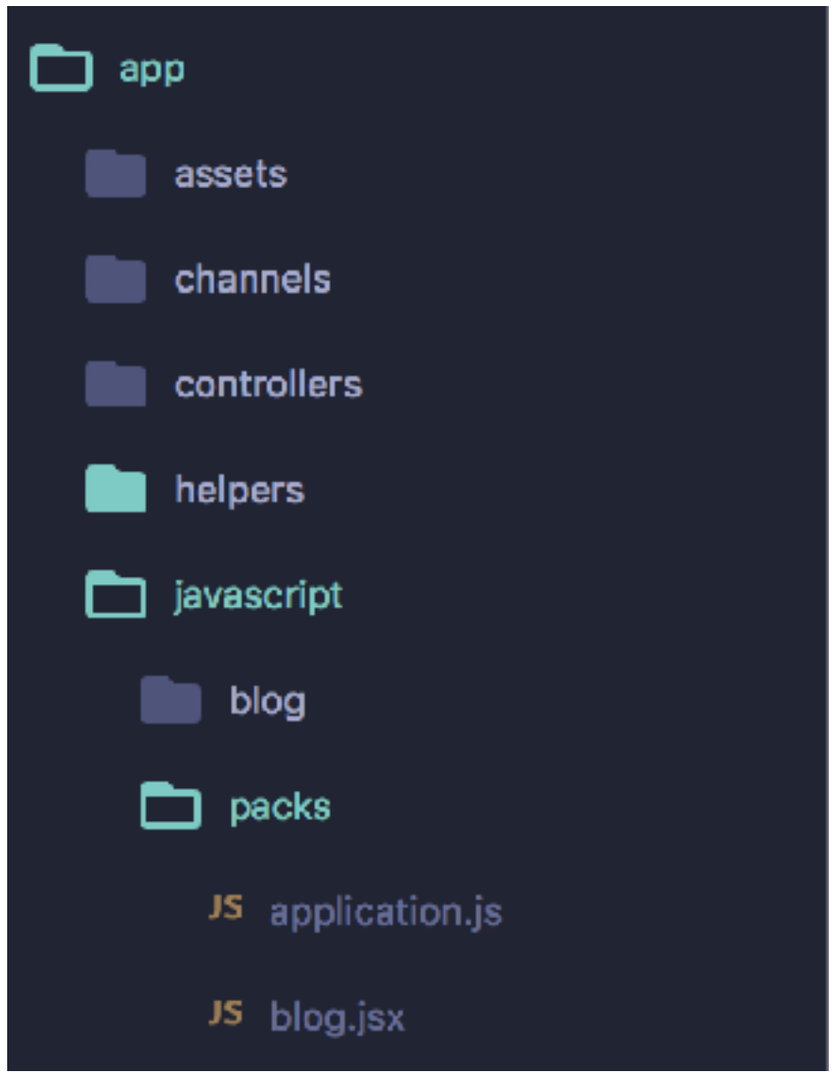
Architecture

Alongside many configuration files,
this created the **`app/javascript`** folder



One folder / app

You can now easily inject
front-end apps in a Rails project



Webpack entry files

Let's import the blog-redux code

You should only place webpack entry files in the `app/javascript/packs` folder

```
// app/javascript/packs/blog.js
import '../blog'; // loads the blog app (needs an index.js)
```

Inject your React App in some view

```
// app/views/pages/home.html.erb
<div class="container"></div>

<%= javascript_pack_tag "blog" %>
```

package.json

Compare your React app's **`package.json`**
with the existing one in Rails.
Add the ones that are missing

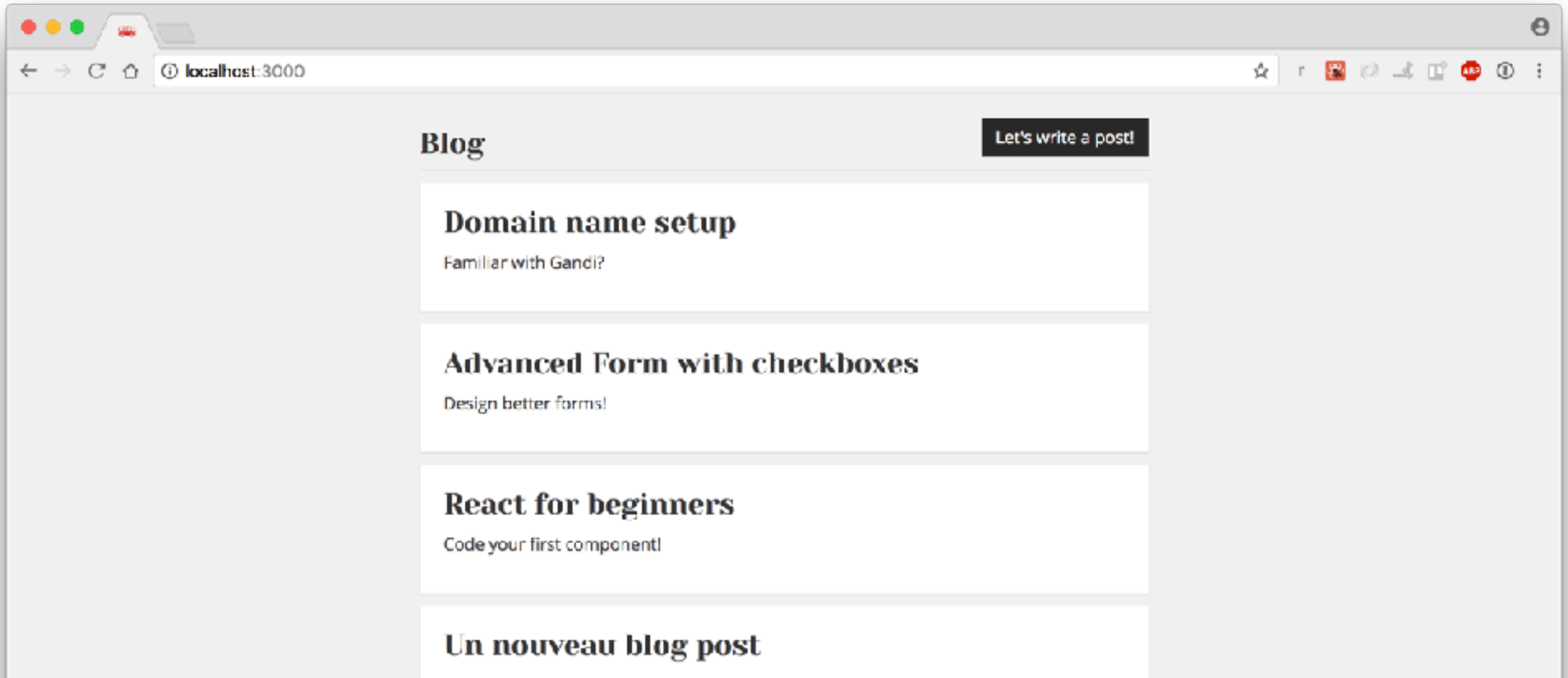
```
yarn add history redux react-redux react-router-dom redux-form
redux-promise redux-logger
```

Copy paste your React app's **src** folder
in the **app/javascript/blog** folder

localhost:3000

launch a `rails s` and navigate to http://localhost:3000

It works 🎉



React Router

Make sure the URLs in your React-Router App meet your webpack entry point

```
# config/routes.rb
root to: 'pages#home'
get "posts/:id", to: 'pages#home'
get "posts/new", to: 'pages#home'
```

API

Strategy

Let's create our own internal API to provide data to our front-end app.

Model

```
rails g model Post title content:text
rails db:migrate
```

```
# app/models/post.rb
class Post < ApplicationRecord
  validates :title, presence: true
  validates :content, presence: true
end
```

Routing

```
# config/routes.rb

Rails.application.routes.draw do
  root to: 'pages#home'
  get "posts/:id", to: 'pages#home'
  get "posts/new", to: 'pages#home'

  # API routing
  namespace :api, defaults: { format: :json } do
    namespace :v1 do
      resources :posts, only: [ :index, :show, :create ]
    end
  end
end
```

Controller

```
# app/controllers/api/v1/posts_controller.rb
class Api::V1::PostsController < ActionController::Base
  def index
    @posts = Post.order(created_at: :desc)
    render json: @posts
  end

  def show
    @post = Post.find(params[:id])
    render json: @post
  end

  def create
    @post = Post.create(post_params)
    render json: @post
  end

  private

  def post_params
    params.require(:post).permit(:title, :content)
  end
end
```

Update your action creators

action creators

```
// app/javascript/blog/

const ROOT_URL = 'http://reduxblog.herokuapp.com/api/posts';
const API_KEY = 'LEWAGON-BLOG';
const ROOT_URL = '/api/v1';

export function fetchPosts() {
  const promise = fetch(`${ROOT_URL}?key=${API_KEY}`)
    .then(response => response.json());

  return {
    type: FETCH_POSTS,
    payload: promise
  };
}
// [...]
```

Preload data

dataset

```ruby
# app/controllers/pages_controller.rb
class PagesController < ApplicationController
  def home
    @posts = Post.all
  end
end
```

```erb
// app/views/pages/home.html.erb
<div id="root" data-posts="<%= @posts.to_json %>"></div>

<%= javascript_pack_tag "blog" %>
```

```jsx
// app/javascript/blog/index.jsx

const root = document.getElementById("root");
const initialState = { posts: JSON.parse(root.dataset.posts) };
const store = createStore(reducers, initialState, middlewares)
```

Your turn!