React - Day 4

Devpoint Labs - Jake Sorce / Dave Jungst

ES2015

ECMAScript (ES) is a standard for scripting languages. JavaScript is an implementation of ECMAScript. Netscape submitted its first version of LiveScript (later renamed JavaScript) to ECMA and since then every implementation of JS has really been ECMAScript.

ES5 is supported by all major current browsers and backwards compatible to IE9, Safari 6, FF 21, and Chrome 23

ES6 is the newest version and was the most extensive update since 1997. The overhaul was so major the name was later changed to ES2015 with the intention to put out new updates every year.

ES2015

ES2015 compatibility: https://kangax.github.io/compat-table/es6/

Although ES2015 is not fully supported compilers like Babel exist to take ES2015 and convert it at runtime to ES5.

So if ES2015 isn't fully supported and we just transform it to ES5 at runtime why even bother?

Faster, Cleaner, less error prone code!!!

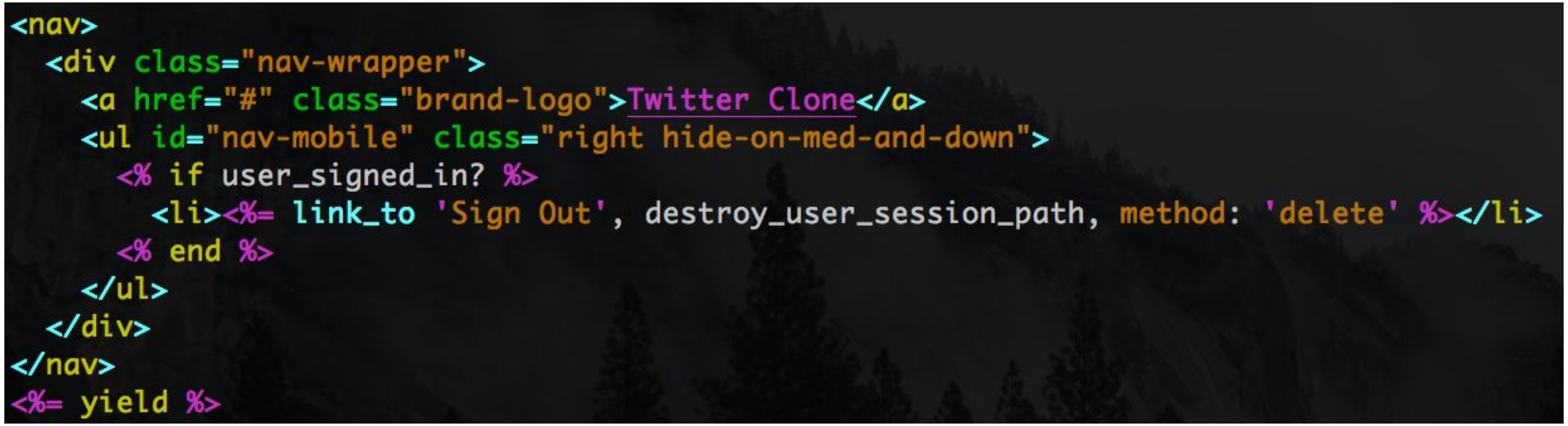
(react-rails comes with full support for ES2015 for files ending in .jsx)

Rails App

```
rails new twitter_clone -d postgresql -T --skip-turbolinks
gem ‘materialize-sass’
gem ‘devise’
gem ‘react-rails’
app/assets/stylesheets/application.css.scss
@import ‘materialize’
app/assets/javascripts/application.js
//= require materialize-sprockets
bundle
rails g model user handle following:text
rails g model tweet body user:belongs_to
bundle exec rake db:create db:migrate
rails g react:install
rails g devise:install
rails g devise User
rails g controller tweets --skip-routes --skip-assets --skip-helper
rails g controller dashboard index --skip-routes --skip-assets --skip-helper
bundle exec rake db:migrate
```

```
app/controllers/application_controller.rb
before_action :authenticate_user!
rails g devise:views

app/controllers/registrations_controller.rb
class RegistrationsController < Devise::RegistrationsController
  private
  def sign_up_params
    params.require(:user).permit(:handle, :email, :password, :password_confirmation)
  end
end

app/views/devise/registrations/new.html.erb
form_for ……
      <div class="field">
        <%= f.label :handle %> <br />
        <%= f.text_field :handle %>
      </div>

Rails.application.routes.draw do
  root ‘dashboard#index'
resources :tweets
  devise_for :users, controllers: { registrations: 'registrations' }
end
```

Nav and dashboard



```
<nav>
  <div class="nav-wrapper">
    <a href="#" class="brand-logo">Twitter Clone</a>
    <ul id="nav-mobile" class="right hide-on-med-and-down">
      <% if user_signed_in? %>
        <li><%= link_to 'Sign Out', destroy_user_session_path, method: 'delete' %></li>
      <% end %>
    </ul>
  </div>
</nav>
<%= yield %>
```

app/views/dashboard/index.html.erb

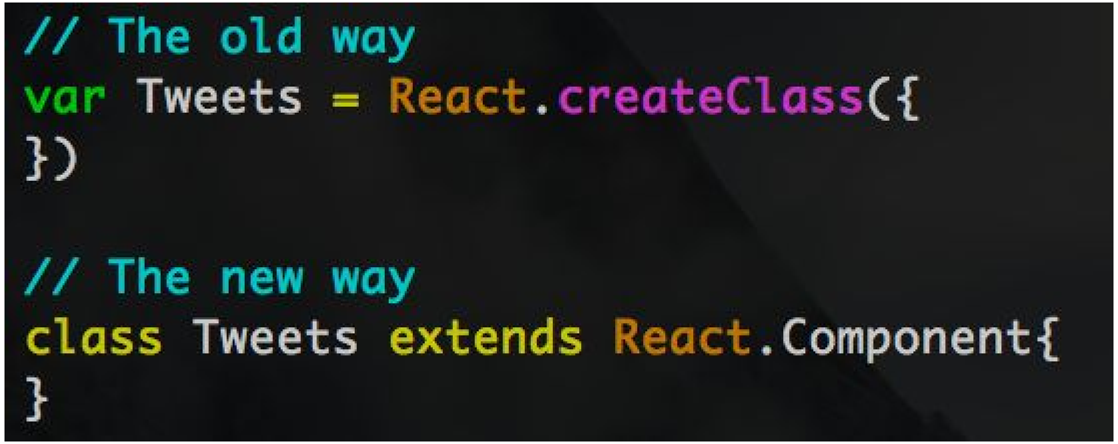
<%= react_component ‘Tweets’ %>

app/models/user.rb

has_many :tweets

ES2015

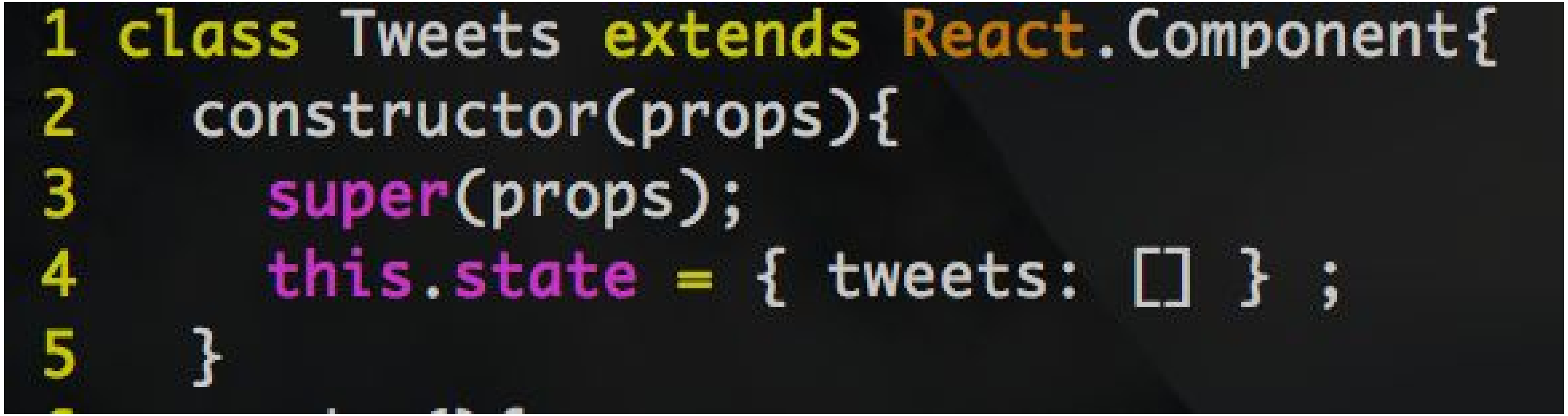
Now that the rails app is all set up we are ready to start on our ES2015 React components. The first difference you will notice is the class syntax for the the component instead of the var syntax.



```
// The old way
var Tweets = React.createClass({
})

// The new way
class Tweets extends React.Component{
}
```

constructor

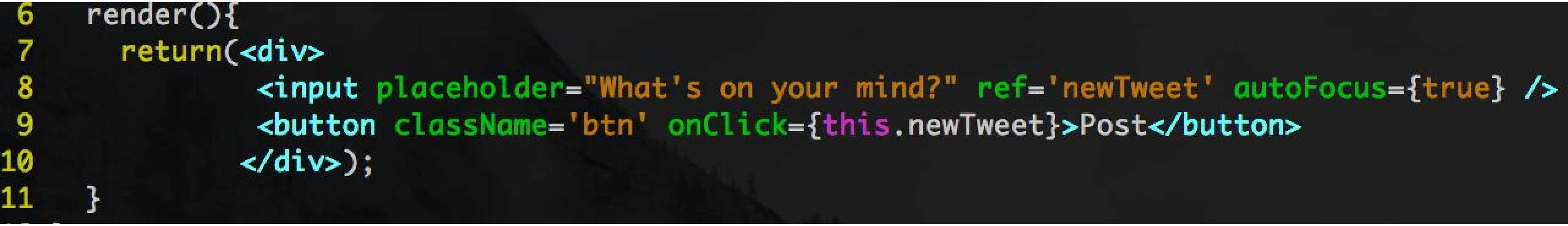
Now that we have a class we need to add a constructor. By adding a constructor we can get rid of the getInitialState and getDefaultState methods completely.



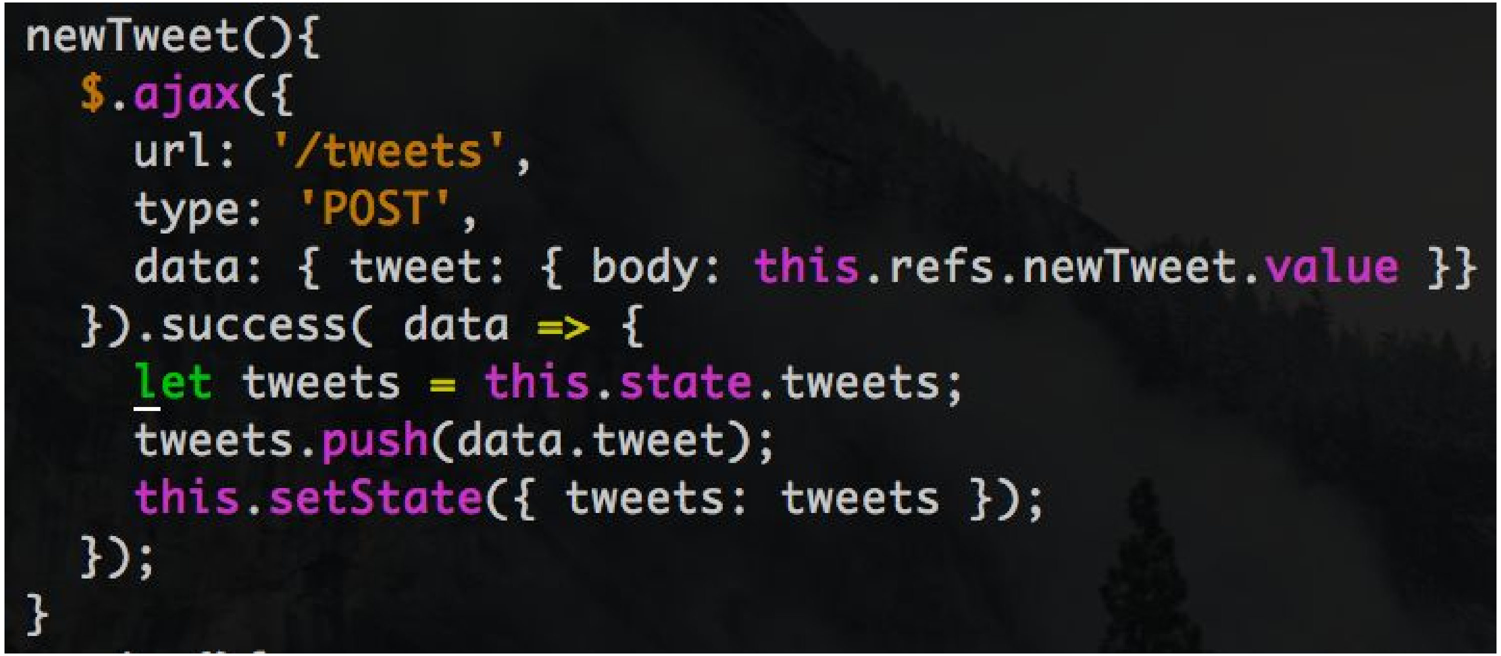
```
class Tweets extends React.Component{
  constructor(props){
    super(props);
    this.state = { tweets: [] } ;
  }
```

functions

Functions have also changed and we will no longer be using the word function. This will also help to control the scope of ‘this’.



```
  render(){
    return(<div>
                <input placeholder="What's on your mind?" ref='newTweet' autoFocus={true} />
                <button className='btn' onClick={this.newTweet}>Post</button>
               </div>);
  }
```



```
newTweet(){
  $.ajax({
    url: '/tweets',
    type: 'POST',
    data: { tweet: { body: this.refs.newTweet.value }}
  }).success( data => {
    let tweets = this.state.tweets;
    tweets.push(data.tweet);
    this.setState({ tweets: tweets });
  });
}
```

Notice that the success method is attached with the . syntax. This allows us to have access to ‘this’ without having to set it outside of the ajax call if we use the => inside of our function.

Notice the use of let instead of var….

Declarations

Previously in JavaScript we used var to declare a variable. With ES2015 let and const are preferred to var.

let can be reassigned but not redeclared

```
let x = 1
let x = 2
```

NOT ALLOWED

```
let x = 1
x = 2
```

ALLOWED

const is READONLY

Why const?

As developers we are guilty of a practice called magic number…

```
if (x > 3)

else if (x < 3)…
```

Does 3 have the same meaning here? and why 3?

```
const MAX_USERS = 3

if (x > MAX_USERS)

else if ( x < MAX_USERS)
```

Why Let?

var becomes ‘hoisted’, hoisting is actually really bad and can produce some unexpected results.

```
loadUsers(users){
  if (users.length > 1)
  var loadingMessage = ‘This could take awhile…’
else
  var flashMessage = ‘Loading…’
}
```

var is scoped to the function and at run time both var’s here would be moved up to the top of the block. This is called hoisting.

More about hoisting

Hoisting can be particularly bad if attempting to call a method in a loop

```
var users = [‘Dave’, ‘Jake’, ‘Bill’, ‘Steve’]
loadUsers(users) {
  for(var i in users) {
    getProfile(“/users/” + users[i], function() {
      console.log(“Fetched users “, users[i]);
     }
   }
}
```

get profile is called 4 times before any of the callbacks are invoked since i is hoisted i will be 4 when the callbacks are called and it will print out steve 4 times.

let

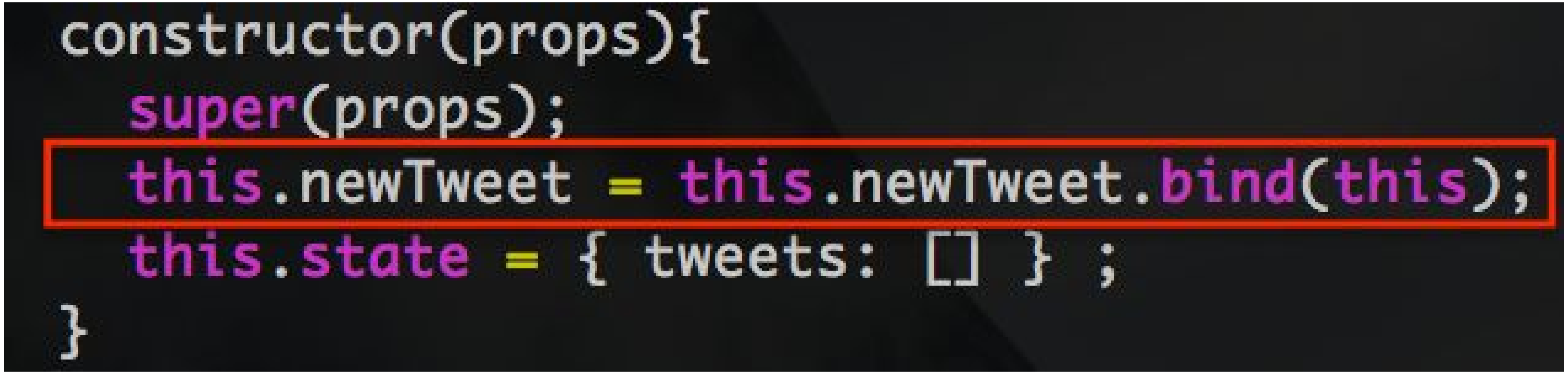
Unlike var let is scoped to its closest block { }

```
getUsers(users) {
  if (users.length > 1)
    let loadingMessage = ‘This could take a while’
  else
    let flashMessage = ‘Loading’
}
```

In this case loadingMessage is scoped only to the if block and flashMessage is scoped only to the else block.

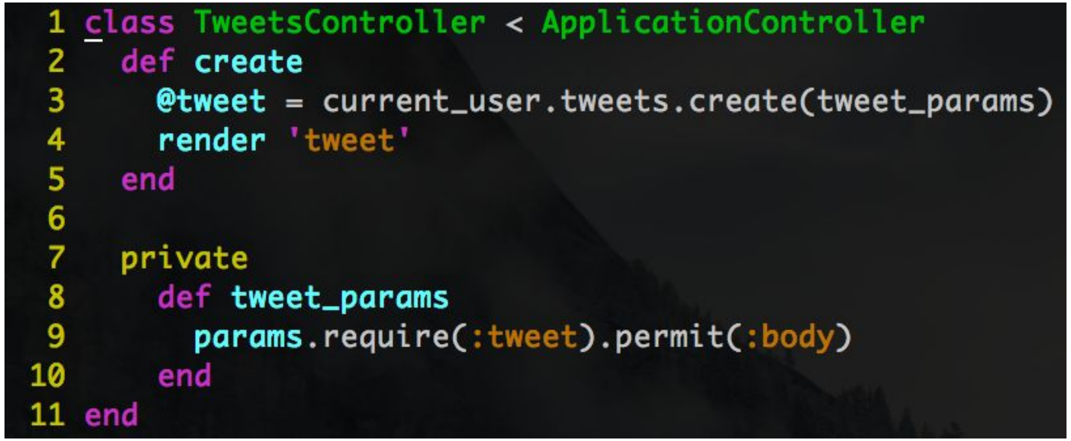
Binding functions

If a function is going to use this referring to the class then it needs to be bound in the constructor.



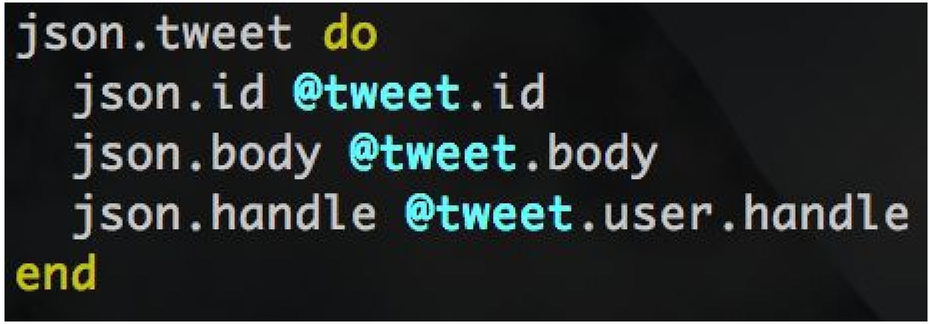
```
constructor(props){
   super(props);
   this.newTweet = this.newTweet.bind(this);
   this.state = { tweets: [] } ;
}
```

Finishing off newTweet



```ruby
class TweetsController < ApplicationController
  def create
    @tweet = current_user.tweets.create(tweet_params)
    render 'tweet'
  end

  private
    def tweet_params
      params.require(:tweet).permit(:body)
    end
end
```

app/views/tweets/tweet.json.jbuilder



```ruby
json.tweet do
  json.id @tweet.id
  json.body @tweet.body
  json.handle @tweet.user.handle
end
```

More about functions

Default params: loadProfiles(users = [])

Named params: setChannel({ message, name, active })

> 0..3 things can be passed in to setChannel and can be accessed as local variables inside the function.

Rest params: displayHashTags(...tags){

> tags is now an array of objects passed in

Spread operator: Like Rest params but used when calling a method instead of in the signature.

String interpolation is back!

```
$.ajax({
  url: `/tweets/${userID}`
})
```

To interpolate use backticks ` and use the ${ } to wrap variables.

This is close to ruby where we use “ #{}

Object initialization shorthand

Before

```
let tweet = { body: body, handle: handle }
```

After

```
let tweet = { body, handle }
```

keys must be the same as varaible names

Object destructuring

before:

```
let tweet = createTweet(“Hello World”, “djungst”)

let body = tweet.body

let handle = tweet.handle
```

After:

```
let { body, handle } = createTweet(“Hello World”, “djungst”)
```

Array Destructuring

```
let users = [“Peter”, “Meg”, “Bryan”]
```

Before:

```
let user1 = users[0]

let user2 = users[1]

let user3 = users[2]
```

After:

```
let [user1, user2, user3] = users
```

Destructuring with rest params

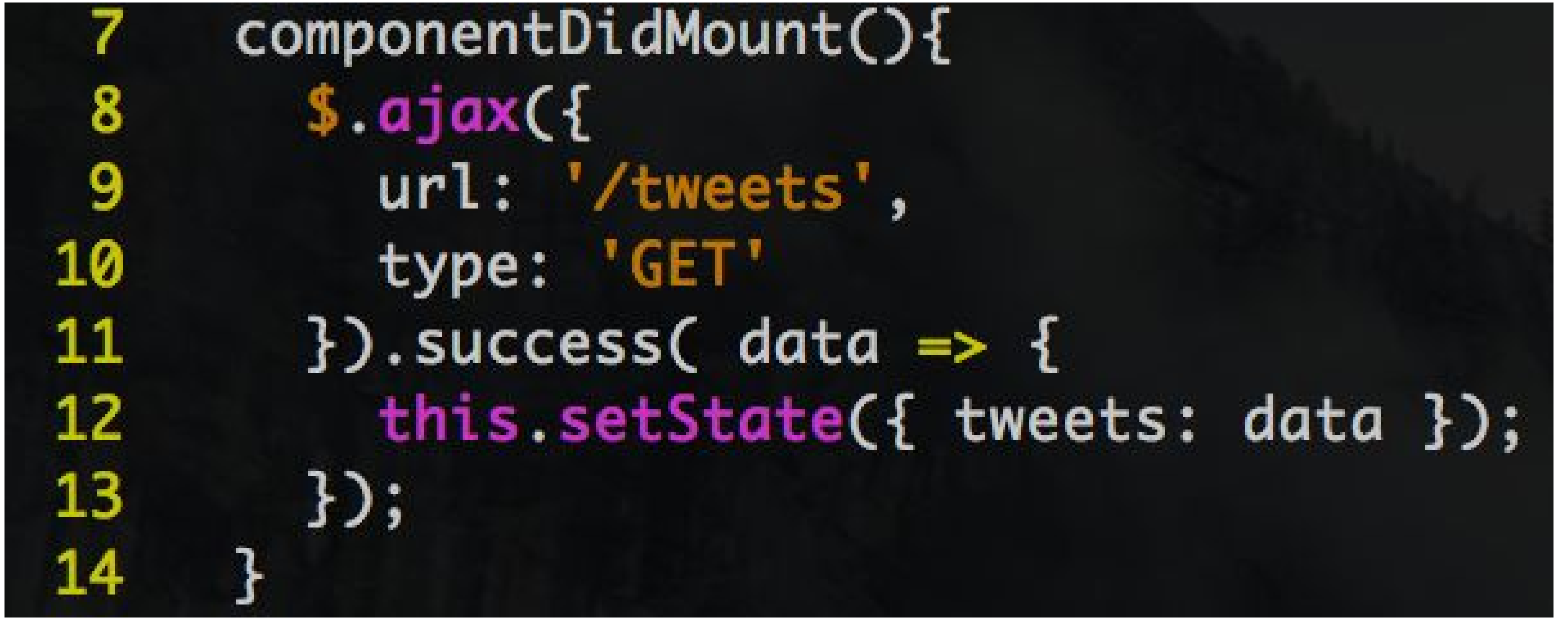
```
let users = [“Peter”, “Bryan”, “Meg”]

let [ first, ...rest] = users

console.log(first, rest)

>>”Peter”, [“Bryan”, “Meg”]
```

for of loop

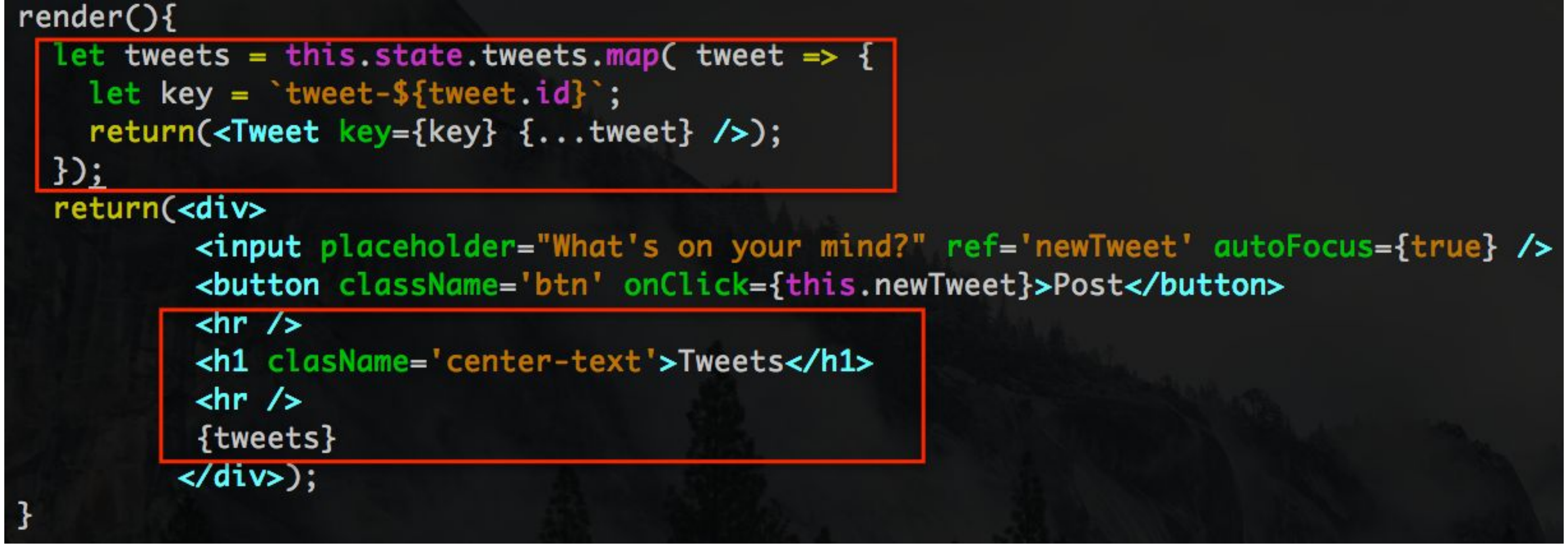
```
let users = [“Phil”, “Ethan”, “Jay”]

for ( let name of users) {

  console.log(name)

}

>>”Phil”

>>”Ethan”

>>”Jay”
```

map...

Let’s get tweets on the page so we can use map



```
componentDidMount(){
  $.ajax({
    url: '/tweets',
    type: 'GET'
  }).success( data => {
    this.setState({ tweets: data });
  });
}
```

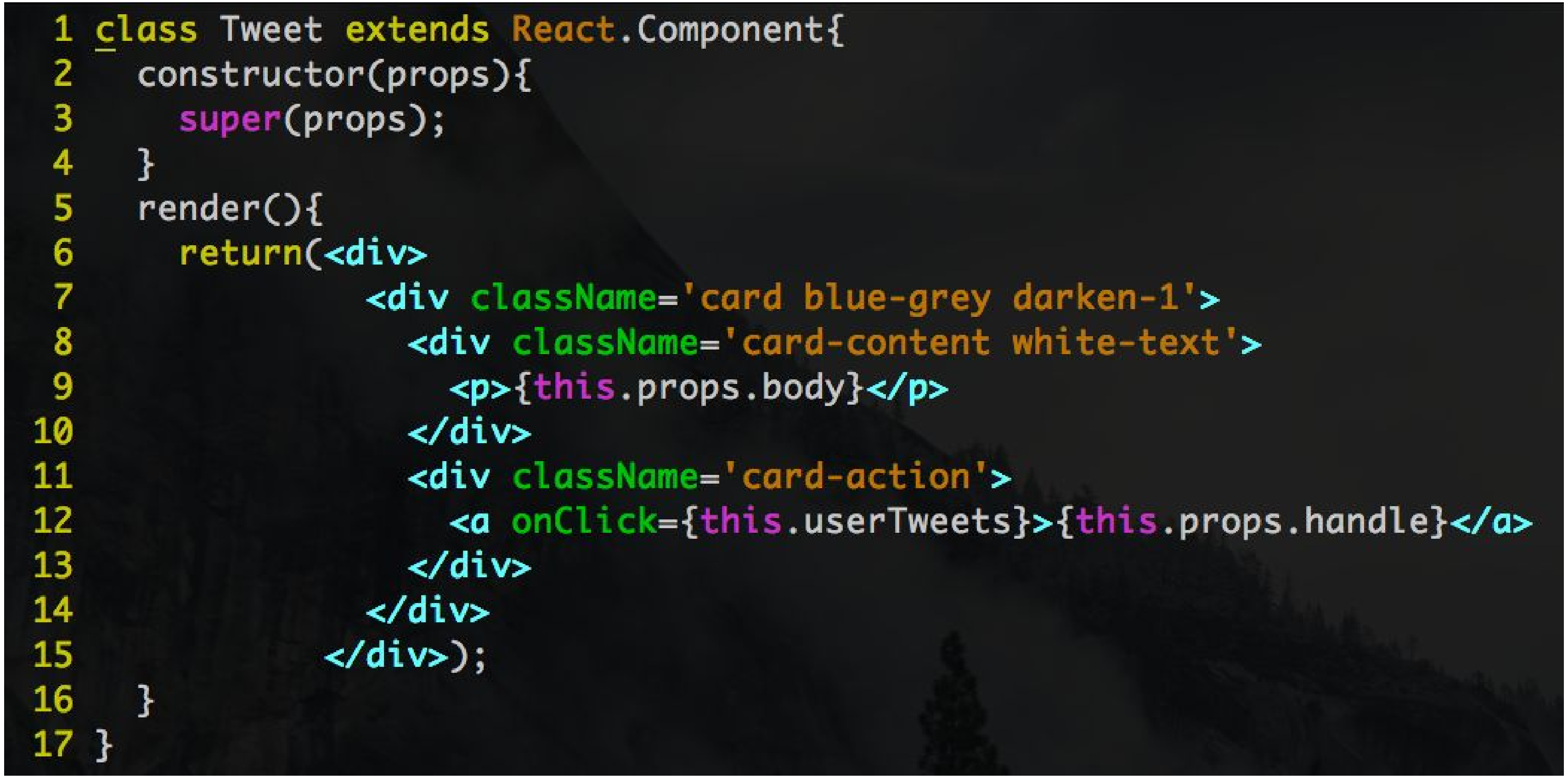


```
render(){
  let tweets = this.state.tweets.map( tweet => {
    let key = `tweet-${tweet.id}`;
    return(<Tweet key={key} {...tweet} />);
  });
  return(<div>
          <input placeholder="What's on your mind?" ref='newTweet' autoFocus={true} />
          <button className='btn' onClick={this.newTweet}>Post</button>
          <hr />
          <h1 clasName='center-text'>Tweets</h1>
          <hr />
          {tweets}
         </div>);
}
```

We are building an array called tweets by calling map on this.state.tweets. Inside the map function we are building <Tweet> components. Notice the destructuring.

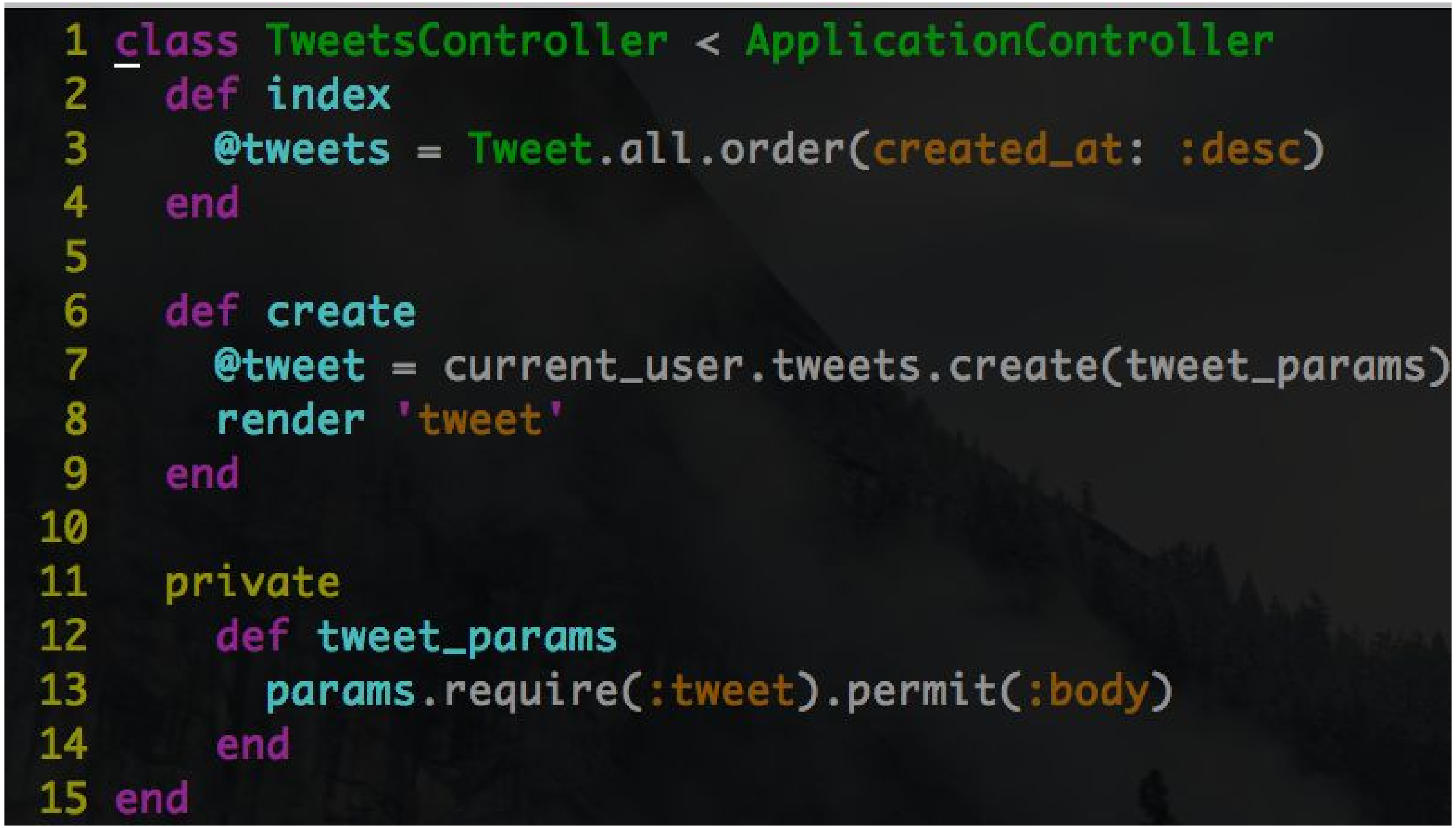
In render we only need to call tweets now.

app/assets/javascripts/components/Tweet.js.jsx



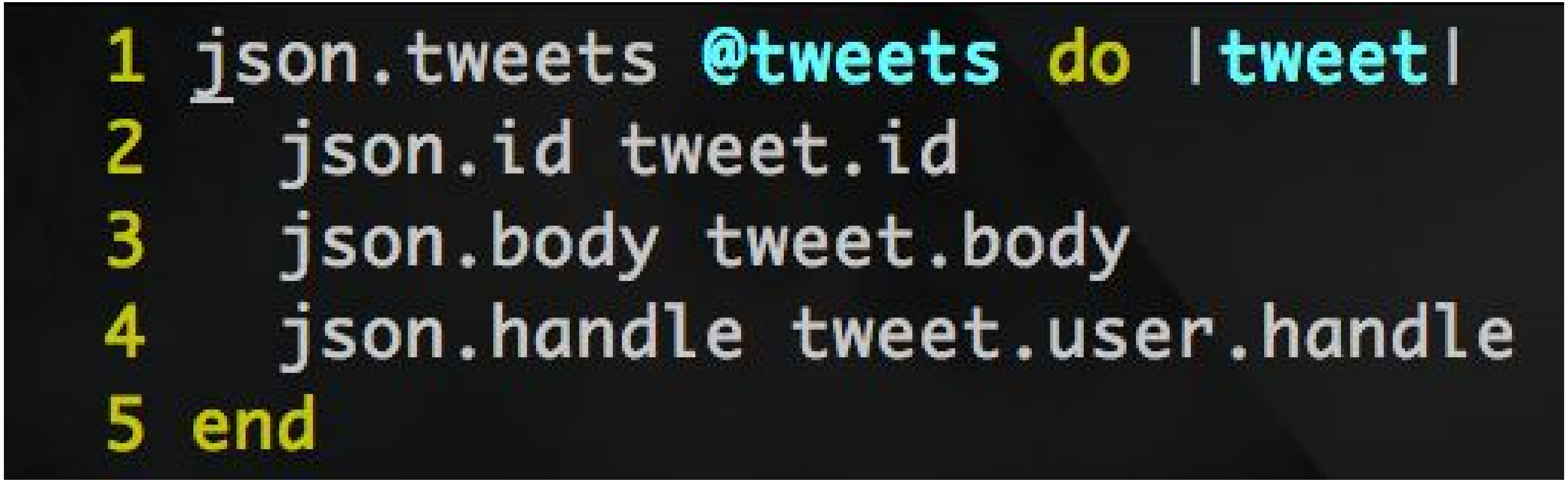
```jsx
class Tweet extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return(<div>
             <div className='card blue-grey darken-1'>
               <div className='card-content white-text'>
                 <p>{this.props.body}</p>
               </div>
               <div className='card-action'>
                 <a onClick={this.userTweets}>{this.props.handle}</a>
               </div>
             </div>
           </div>);
  }
}
```

Tweets controller



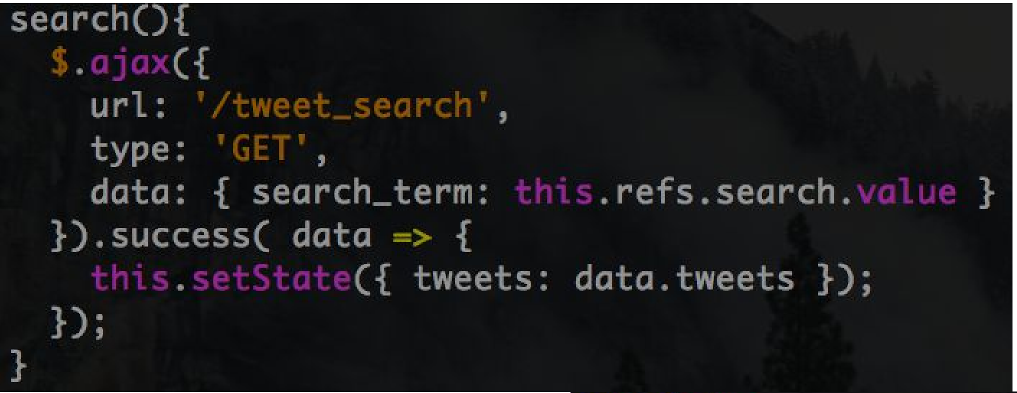
```ruby
class TweetsController < ApplicationController
  def index
    @tweets = Tweet.all.order(created_at: :desc)
  end

  def create
    @tweet = current_user.tweets.create(tweet_params)
    render 'tweet'
  end

  private
    def tweet_params
      params.require(:tweet).permit(:body)
    end
end
```

app/views/tweets/index.json.jbuilder

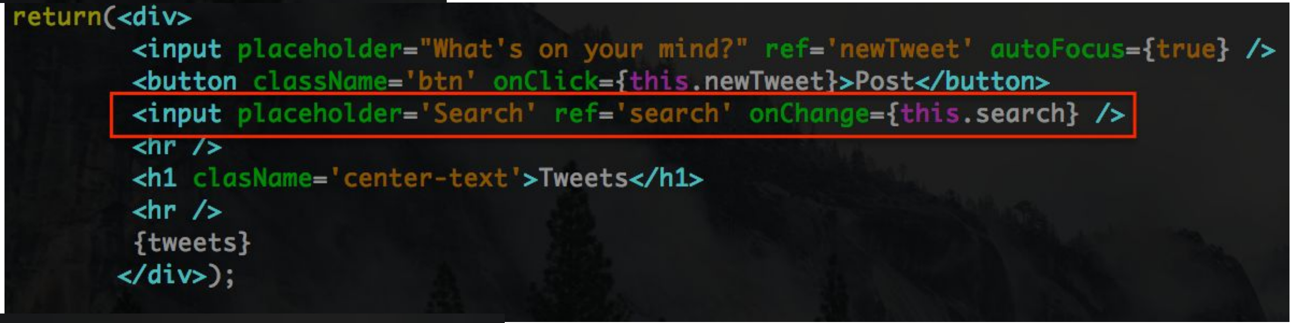


```ruby
json.tweets @tweets do |tweet|
  json.id tweet.id
  json.body tweet.body
  json.handle tweet.user.handle
end
```

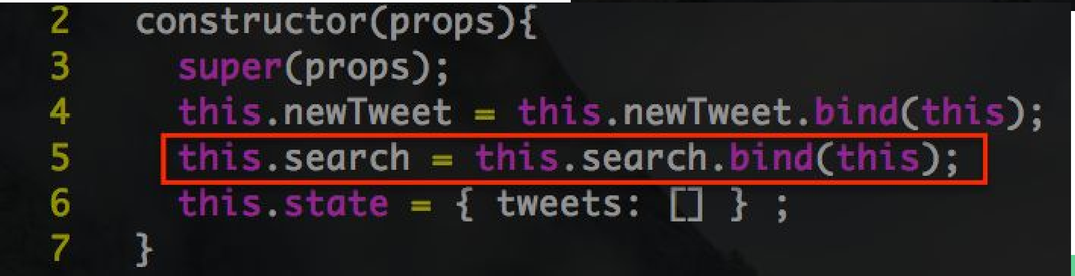
app/assets/javascripts/components/Tweets.js.jsx



```
search(){
  $.ajax({
    url: '/tweet_search',
    type: 'GET',
    data: { search_term: this.refs.search.value }
  }).success( data => {
    this.setState({ tweets: data.tweets });
  });
}
```



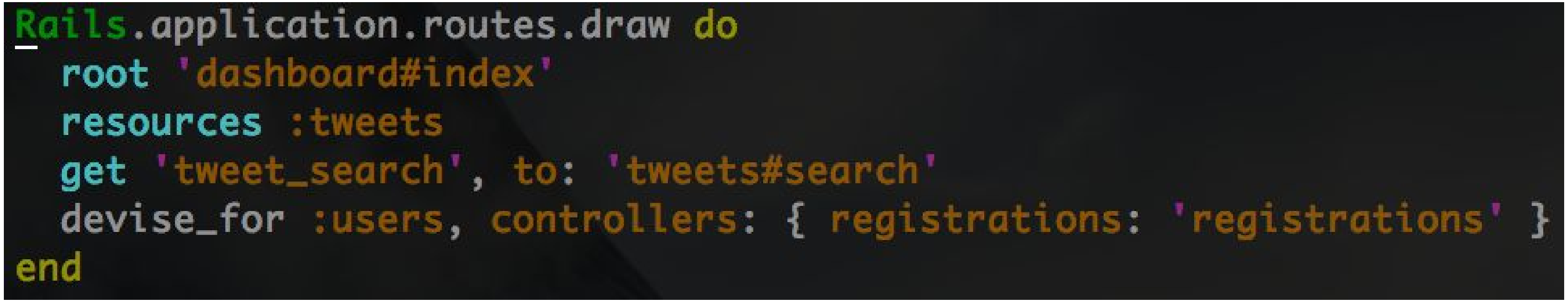
```
return(<div>
        <input placeholder="What's on your mind?" ref='newTweet' autoFocus={true} />
        <button className='btn' onClick={this.newTweet}>Post</button>
        <input placeholder='Search' ref='search' onChange={this.search} />
        <hr />
        <h1 clasName='center-text'>Tweets</h1>
        <hr />
        {tweets}
       </div>);
```



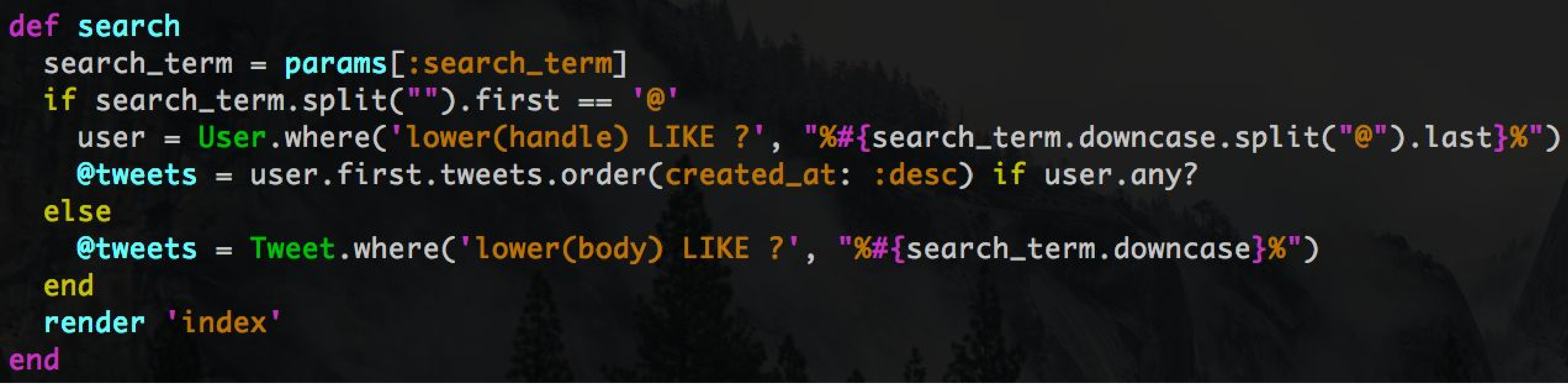
```
constructor(props){
  super(props);
  this.newTweet = this.newTweet.bind(this);
  this.search = this.search.bind(this);
  this.state = { tweets: [] } ;
}
```

On the server side

config/routes



```ruby
Rails.application.routes.draw do
  root 'dashboard#index'
  resources :tweets
  get 'tweet_search', to: 'tweets#search'
  devise_for :users, controllers: { registrations: 'registrations' }
end
```



```ruby
def search
  search_term = params[:search_term]
  if search_term.split("").first == '@'
    user = User.where('lower(handle) LIKE ?', "%#{search_term.downcase.split("@").last}%")
    @tweets = user.first.tweets.order(created_at: :desc) if user.any?
  else
    @tweets = Tweet.where('lower(body) LIKE ?', "%#{search_term.downcase}%")
  end
  render 'index'
end
```

In the search method we first check for @ and if so we search for tweets by a user. If there is no @ then we assume they want to search the body of the tweet.

Using lower in the query and downcase on search term will make a non case sensitive search.