**The Hitchhiker’s Guide to React Router v4: [match, location, history] — your best friends!**

[[](https://medium.freecodecamp.org/@evedes?source=post_header_lockup)](https://medium.freecodecamp.org/@evedes?source=post_header_lockup)

[Eduardo Vedes](https://medium.freecodecamp.org/@evedes)

Sep 2, 2018

“assorted texts wall signages” by [Bruno Martins](https://unsplash.com/@brunus?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral" \t "_blank)

Hey! Welcome to the Hitchhiker’s Guide to React Router v4, Part II!

Now that we’ve [set the ball rolling](https://medium.freecodecamp.org/hitchhikers-guide-to-react-router-v4-a957c6a5aa18) with our first small App, let’s focus on your three travel mates: match, location and history.

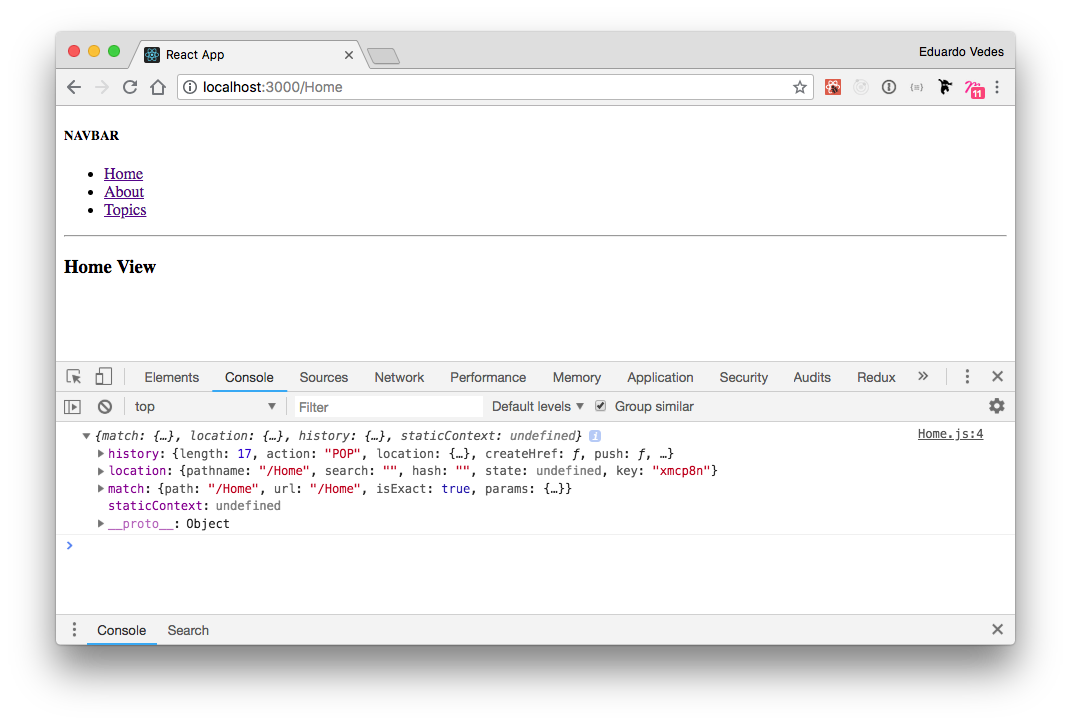
What happens if you get inside your Home Component code and put a console.log there to check the props?

<https://medium.freecodecamp.org/hitchhikers-guide-to-react-router-v4-4b12e369d10>



console.log(props) inside <Home /> Component

Router introduces into your component the following objects:



console.log(props) result in Chrome console

Wow! Where does that came from? 😄

Well, every view, component, or whatever that’s wrapped by Router has these objects. <Router />does its job as an Higher Order Component and wraps your components or views and injects these three objects as props inside them.

So… why are they there and what use can I make of them? 😸

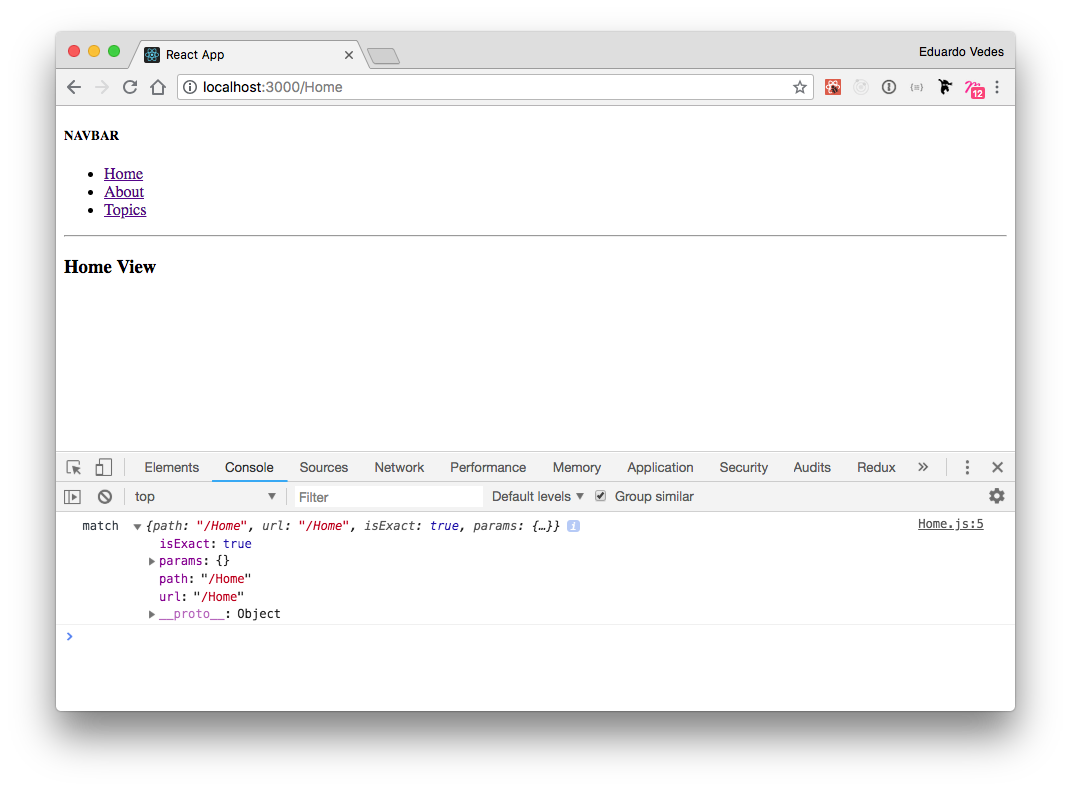
They’ll be your best friends! Trust me! 😻

**Match**

The matchobject contains information about how a <Route path> matched the URL.

* **params**: (object), key/value pairs parsed from the URL corresponding to the dynamic segments of the path
* **isExact**: (boolean), true if the entire URL was matched (no trailing characters)
* **path**: (string), the path pattern used to match. Useful for building nested routes**.** We’ll take a look at this later in one of the next articles.
* **url**: (string), the matched portion of the URL. Useful for building nested links.

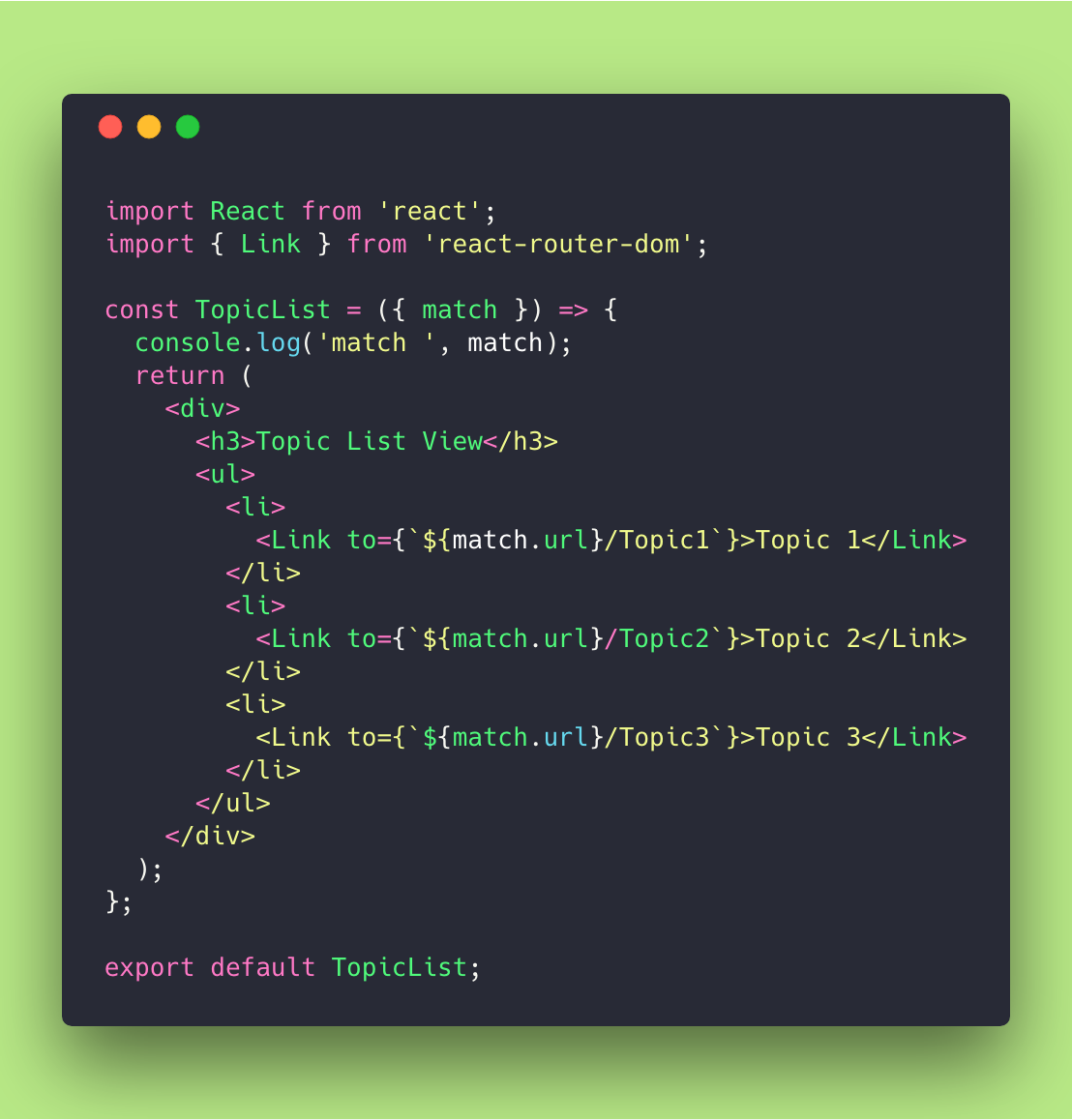
So in the Home component we have this match object:



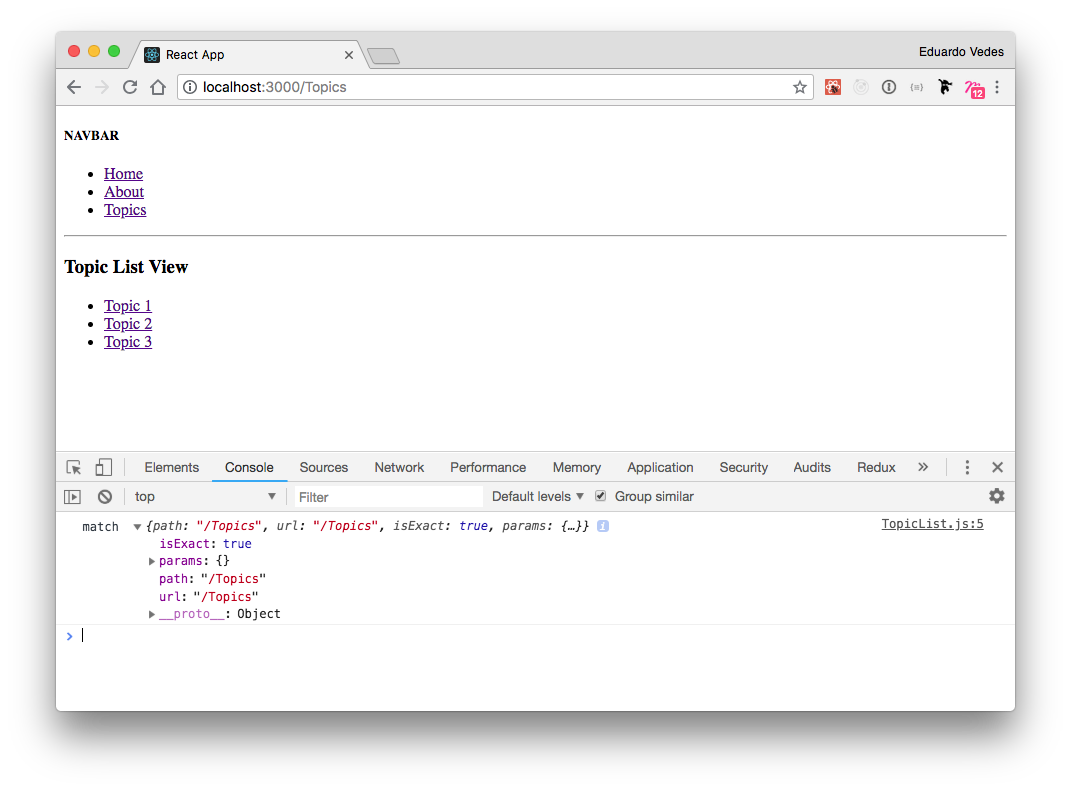
**match** object inside Home Component

isExact is true because the entire URL was matched, the paramsobject is empty because we didn’t pass anything into it, the path and the url key values are equal confirming that isExact is true.

Now let’s take a look at the TopicList View:



TopicList View code



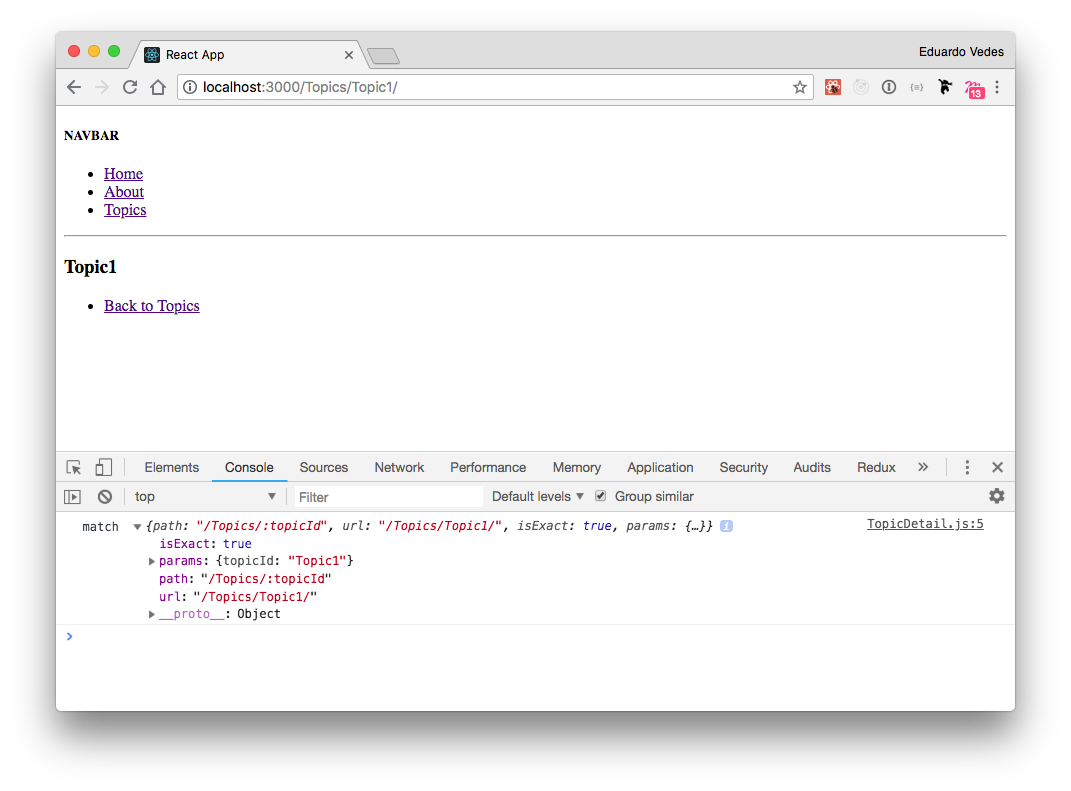
console.log(match) inside TopicList View

Nothing new until now, same story as in the Home View, showing the path and the url of TopicList.

But what if we take a look at TopicDetails?



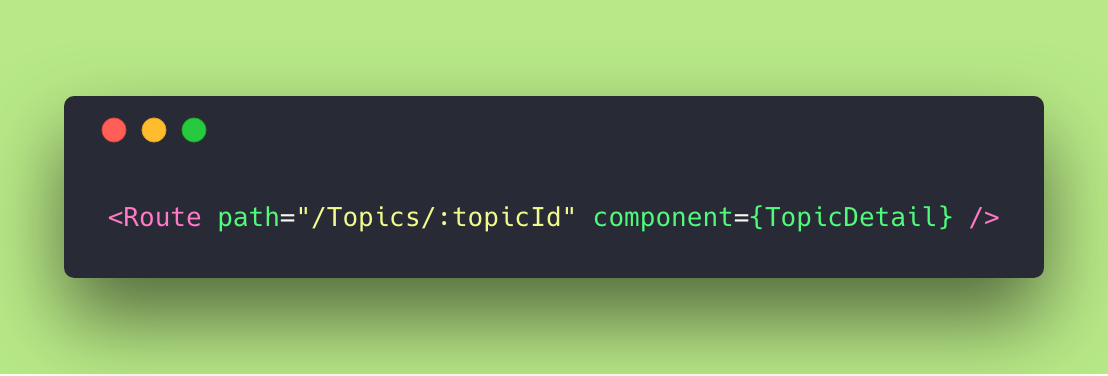
TopicDetails



console.log(match) inside TopicDetails

Okay, what do we have here?

isExactcontinues to be true because the entire URL was matched. params object brings the topicId info that was passed into the component.



Route for TopicDetail component in routes.js

Pay attention to how the topicId is a variable.

But where does it assume the Topic1 value?

Simple, you’re invoking it in an explicit way in TopicList Links.

Check how we’ve used match for the TopicList to know its URL.

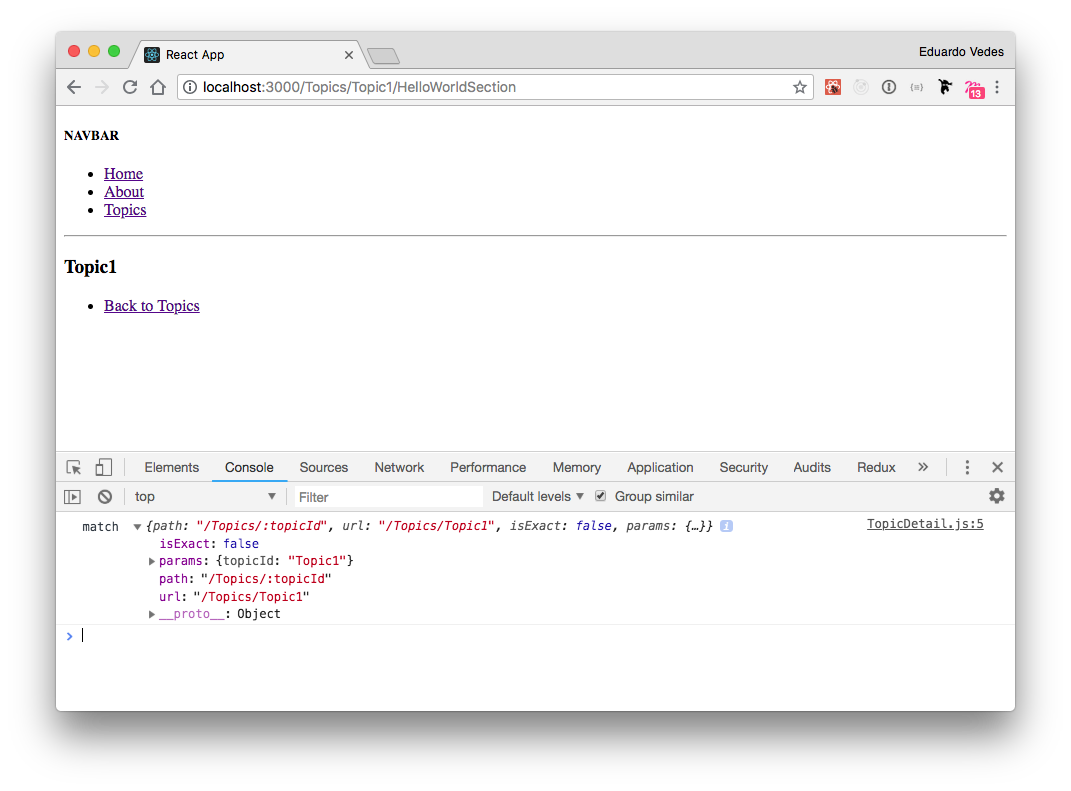


This link could be **dynamic**. Later on we’ll do an example where you Link to a relative path where you don’t know previously if it’s Topic1 or Topic3520.

But…

In which situation is the isExact false?

Well… let me give you an example:



isExact false example

In this situation we’ve introduced the /HelloWorldSection into the browser URL.

What happens is that the Router doesn’t know the full path to the HelloWorldSection so it routes you up until where it knows the way.

isExact shows false telling you precisely that the “**entire URL was not matched**”.

This is very useful, as you’ll see as soon as you start doing SPAs with RRv4!

Just to finish our approach to match check this out:



TopicDetails code

We’ve used the match.params.topicId to print in the screen our topic name.

This is one of the most common usages for match.

Of course it has a multitude of applications. Suppose we need to fetch an API with this topicId information. 😅

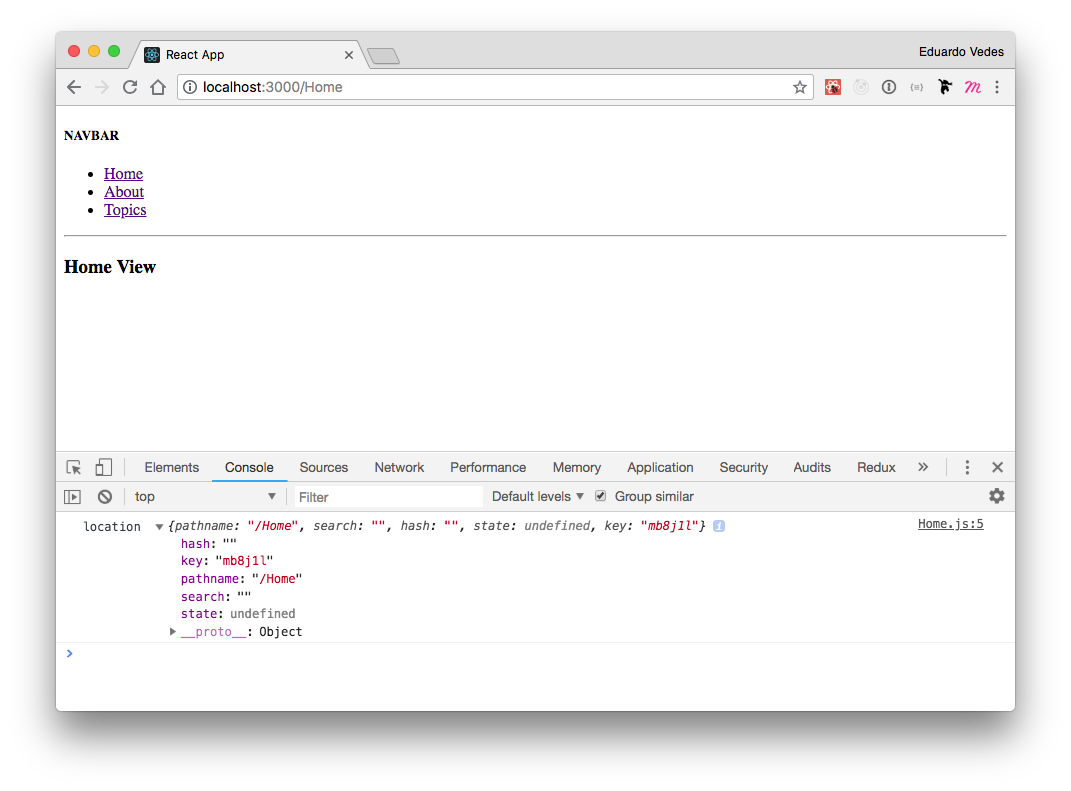
**Location**

The location object represents where the app is now, where you want it to go, or even where it was.

It’s also found on history.locationbut you shouldn’t use that because it’s mutable.

A location object is never mutated so you can use it in the lifecycle hooks to determine when navigation happens. This is really useful for data fetching or DOM side effects.

Let’s console.log(location) inside Home View:

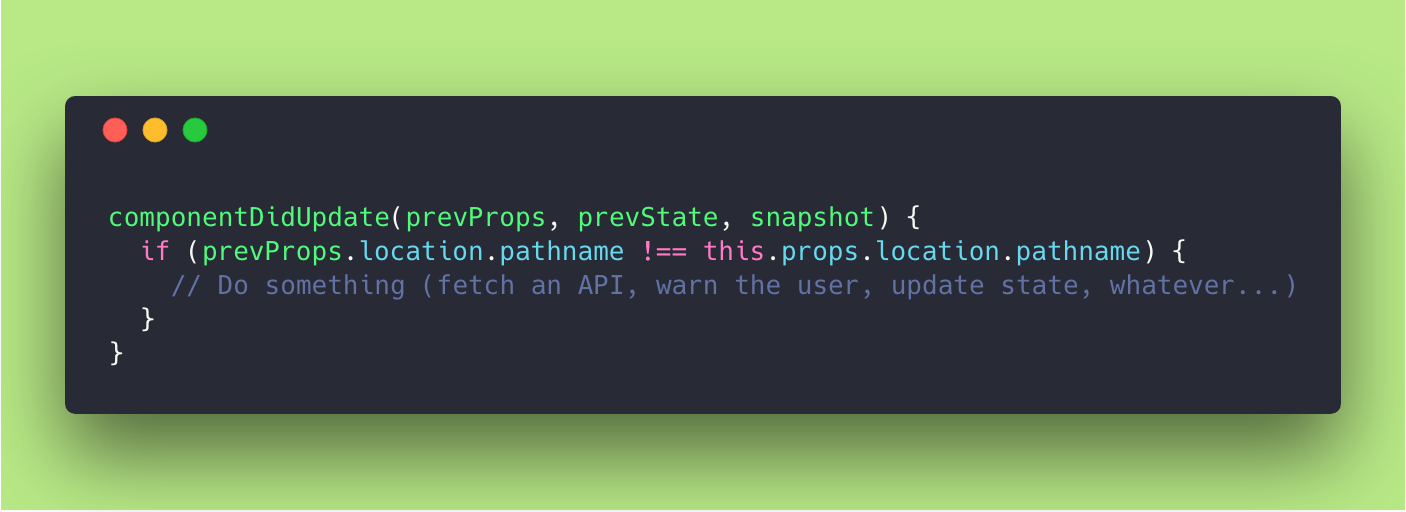


console.log(location) inside Home View

Let’s not deep dive a lot and keep with its simple functionality.

You have the pathname key/value.

You can use it for example to check if pathname has changed:



usage of location inside lifecycle method componentDidUpdate()

You can <Link /> or <Redirect /> to it. You can do an history.push or history.replace as we’re going to see later.



You can create a custom object and use it

* <Redirect to={locationX} />
* <Link to={locationX} />
* history.push(locationX)

You can also pass it into <Route /> and <Switch /> components.

This will prevent them from using the actual location in the router’s state. Maybe you want to trick a Component to render in a different location than the real one 😅

Enough of location now…

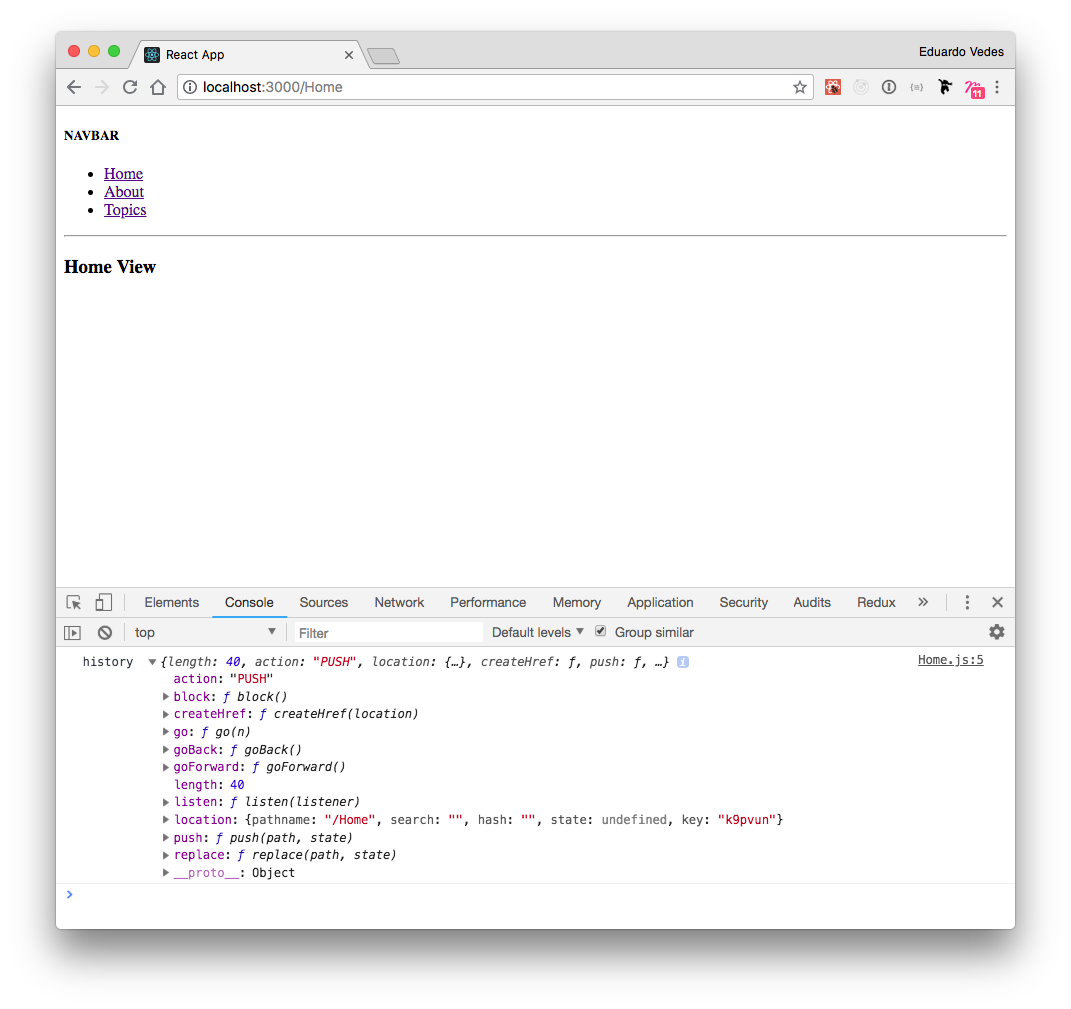
Let’s move to history!

**History**

The historyobject allows you to manage and handle the browser history inside your views or components.

* **length**: (number), the number of entries in the history stack
* **action**: (string), the current action (PUSH, REPLACE or POP)
* **location**: (object), the current location
* **push(path, [state])**: (function), pushes a new entry onto the history stack
* **replace(path, [state])**: (function), replaces the current entry on the history stack
* **go(n)**: (function), moves the pointer in the history stack by n entries
* **goBack()**: (function), equivalent to go(-1)
* **goForward()**: (function,) equivalent to go(1)
* **block(prompt)**: (function), prevents navigation

So let’s console.log the history object in our Home View and see what it shows:



console.log(history) inside Home View

Okay, exactly what we were expecting.

It tells us that we’ve arrived here with a PUSH action, that the length of the object is 40 (as you navigate thru your app history grows to 50 and stops there, discarding the older entries and keeping its size each time the app pushes another history entry into the object).

It gives us the location information.

Again, the history object is **mutable**. Therefore it is recommended to access the location from the render props of Route, not from history.location.

This ensures your assumptions about React are correct in lifecycle hooks.

For example:



example of the usage of location, right and wrong

Typically you can use it to change the browser URL path.

In the example below we avoid <Link> and create a button that does a history push:



example of history.push usage

Of course you can use it to trigger the URL change after some data fetching or side effects.

It’s comfortable to use it in the middle of JSX where you don’t want to invoke components. You can simply return a history push and trigger Router to update the Browser URL.

**Last but not least**

I think that by this time you already have a good idea on how to use match, location and history.

I didn’t make any changes to our initial boilerplate, so feel-free to play with it in the same [repo](https://github.com/evedes/React-Boilerplate-01) supplied in [Part 1](https://medium.freecodecamp.org/hitchhikers-guide-to-react-router-v4-a957c6a5aa18) of this guide.

**05. Bibliography**

To make this article I’ve used the React Router documentation that you can find [here](https://reacttraining.com/react-router/core/guides/philosophy).

All the other sites I’ve used are linked along the document to add info or provide context to what I’ve tried to explain to you.

This article is part 2 of a series called “Hitchhiker’s Guide to React Router v4”

* [Part I: Grok React Router in 20 minutes](https://medium.freecodecamp.org/hitchhikers-guide-to-react-router-v4-a957c6a5aa18)
* Part III: recursive paths, to the infinity and beyond!
* Part IV: route config, the hidden value of defining a route configuration array

😻 Thank you very much!