





★
★
8,660 reads
Curry away

 Sign in to Hacker Noon with Google 



leo reny
leoreny8@gmail.com



International Finance
internationalfassociation@gmail.com

8 more accounts

i n R e a c t

by

Ste

ve

Rob

ins

on



Nov

em

ber

27t

h,

201
7



Au
dio
Pre
sen
ted
by MongoDB

Read
by **Dr.
One**



Disclaimer: You may or may not have improved your curry making skills by the end of this blog

So you're going to build the next-gen eCommerce company selling dog supplies and you've decided to use React to build it with coz, you know, its all the rage now and choosing a front-end framework is the "first and most important responsible step" towards building any successful online business.

I'd like to share a neat trick I learned while building my own eCommerce website, obviously next-gen coz React!

Filtering products based on various aspects is a staple in any shopping site and so let's add some filters to the product results page.

Refine results

Price Range

Ages

☐ 0 - 6m

☒ 6m - 1y

☐ 1y - 2y

☐ 2y - 4y

☐ 4y - 8y

☐ 9y and older

Brands

☒ Choostix

☐ Smarty Pet

☐ Super Dog

☐ That Dog In Tuxedo



☐ Isle of Dogs



☒ Dog Gone Smart



☒ Dog Toys



☐ Himalayan Dog Chew

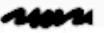

Search Results





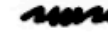







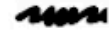







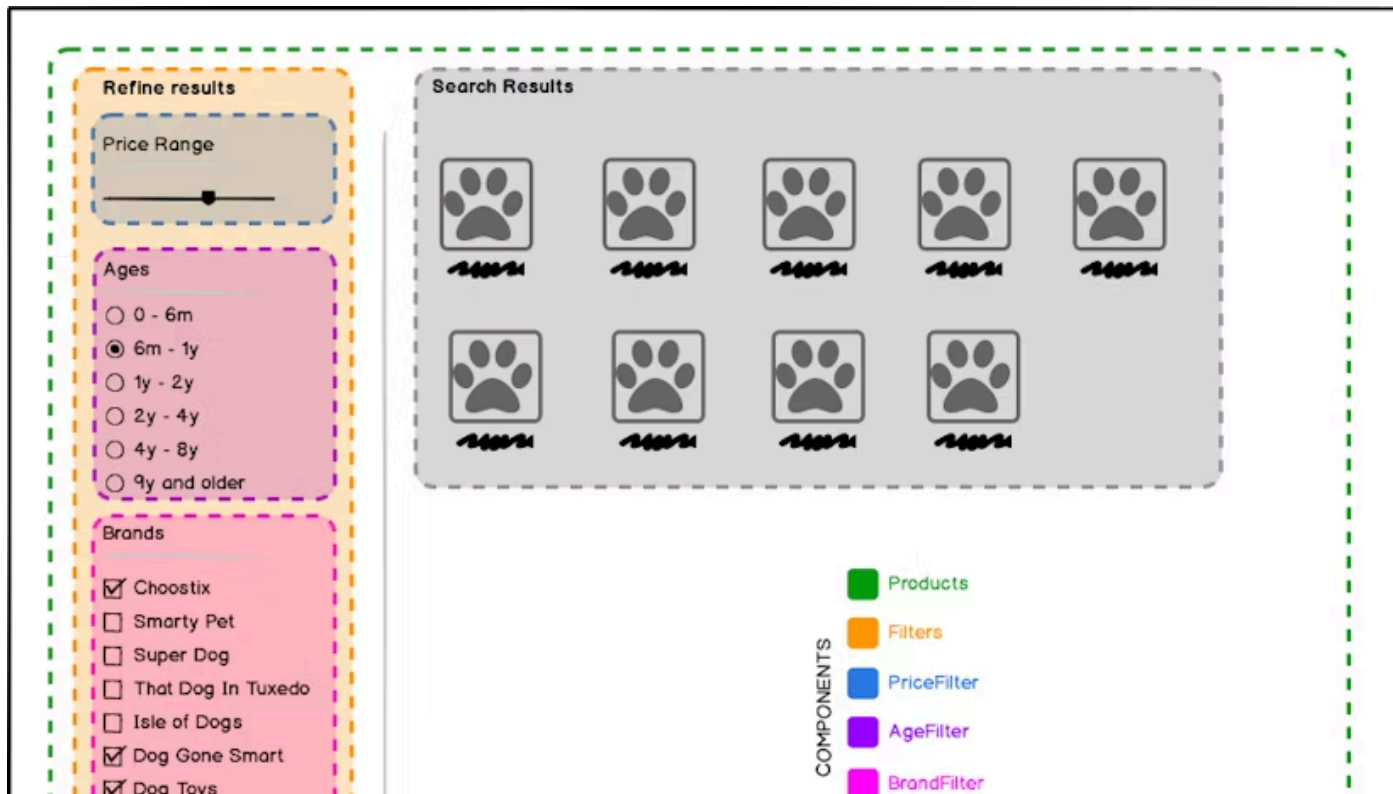






This is where our customers filter and find the right products for their dogs :)

As any 'good' React developer would, let's break up this UI into components which is pretty straight forward in our case.



Discover Anything




Login

Read

Write



 MongoDB. Join the generative AI revolution



Curry away in React by @steverbob



`filterSelections` is an object which contains the selected value for each of the filters in our page.

```
filterSelections = {price: ...,ages: ...,brands: ...},
```

Since the `Products` component is the common parent that contains the components that need `filterSelections` (the `Filters` and `ProductResults` components), it would be the appropriate home for this *state* to live in.

The `Filters` component, after receiving the `filterSelections` as a prop, will pass down relevant filter values to each of its children.

`filterSelections` will also be passed down into the `ProductResults` component so that the filters can be applied and only the relevant products are shown.

These filters can't be just static controls, we need to them to update `filterSelections` as the customer adjusts the filters.

We're passing down `filterSelections` as *props* to the filters and we know props are immutable in React. So where do we mutate `filterSelections` then? The answer to that would be the `Products` component as its the owner of `filterSelections` having it as its *state*.

Let's go ahead and add a bunch of change handler functions so that any changes to the filter selections are propagated up to the `Products` component where the actual mutation happens.

Here's the `Products` component.

And the `Filters` component.

This is pretty straightforward and would work. But we've got a bunch of functions in the `Filters` component, all of which seem to be doing very similar things.

And if we add a new kind of filter in future, we'll need to write yet another similar looking function in this component.

Currying to the rescue

This is where the trick comes in and it's a popular & fancy sounding functional programming concept called **currying**.

Before talking about currying, I'd like to show you how it can help clean up our component. Take a look for yourself —

Neat right? Now the `updatedSelections` function is said to be *curried*. For the functional programming aficionados out there, this is a pretty simple and straightforward concept. But for folks like me who's always lived in the imperative & object oriented side of things, this was confusing to understand. So let's unpack it.

You might have wondered when I said that the `updatedSelections()` function has been curried, what the "uncurried" version of this function would look like. Here's how —



If we were to use this instead of our curried version, we'd need to make each of the Filter component children call `updateSelections` themselves and thus having to make them aware of the attribute name in `filterSelections` that they need to update, which is too much coupling.

The alternative is to use a dedicated function for each of the Filter components which leads to a mess like we saw already. Our solution is to *curry* this function.

What's currying?

Currying is transforming a function `f` into a function `f'` which takes part of the arguments that `f` originally needed and would return another function which could take in rest of the arguments, returning the result of `f` or could be curried itself.

Concretely, take this simple add function,

```
add = (x, y) => x + y;
```

when curried, it becomes —

```
curriedAdd = (x) => {return (y) => {return x + y;}}
```


So instead of calling `add(1, 2)` we could call `curriedAdd(1)` which returns a new function, which we could subsequently call with the other argument of `add`, like `curriedAdd(1)(2)` and get the final result 3.

Technically calling, `curriedAdd(x)` is called “partial application”, as in, we’re applying only part of the arguments that the original function `add(X, y)` needed.

Currying a regular function let’s us perform partial application on it.

This is a naive toy example which does not have much utility.

Now if we go back to our original example —

we can see how partial applications of the `updateSelections` function —

```
updateSelections('ages');updateSelections('brands');updateSelections('price');
```

helped us clean up our component and reduce coupling. That’s currying :)

Like all problems in software, this is not the only solution to this problem. Here’s an alternate solution —

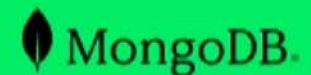
I hope you found this technique useful. If you’ve used currying in your **JavaScript** / **React** project, do share snippets in the comments below and also your other comments and feedback are welcome ❤️.

Are you looking to hire JavaScript, React or Ruby developers to work on your projects? Visit spritel.com to hire our kick-ass team that can bring your ideas to reality.

Go where you want to go with
generative AI, and get there faster

LEARN MORE

Develop Your AI Future



About Author



Steve Robinson @steverob

name@company.com

Subscribe

Technical Consultant

Read My Stories

Comments

Add Comment

TOPICS

PROGRAMMING

#functional-programming

#react

#javascript

#curry-away-in-react

#currying

THIS ARTICLE WAS FEATURED IN...

Permanent on Arweave



Terminal



Lite

MENTIONED IN THIS STORY

companies

BUNCH

RELATED STORIES

ActiveRecord — Named Query Fragment Placeholders

by **steverob** Jul 22, 2017 **#ruby-on-rails**

Crypto Curry with Elixir

by **iacobson** Jan 13, 2018 **#elixir**

Functional Programming Paradigms in modern JavaScript: Currying

by **KondovAlexander** May 24, 2018 **#functional-programming**

I ♥ Ramda — Partial Application with a Special Placeholder

by [joelthoms](#) May 11, 2018 [#javascript](#)


Javascript and Functional Programming: Currying (Pt.4)


by [omergoldberg](#) Dec 16, 2017 [#javascript](#)

Join HackerNoon

Latest technology trends. Customized Experience. Curated Stories. Publish Your Ideas





ABOUT	READ	WRITE	BUSINESS
Careers	Archive	 Distribution	Billboard
Contact	Categories	Editing Protocol	Book Demo Meeting
Cookies	Image Gallery	Editor Tips	Business Blogging
Emails	Leaderboard	Guidelines	Case Studies
Help	Learn Repo	Help	Company Directory
Privacy	Noonification	New Story	Crypto Directory
Sitemap	Signup	Perks	Live Business Posts
Shareholders	Tech Beat	Process	Newsletters
Startups 2023	Tech Brief	Subscribers	Niche Targetting
Testimonials	Tech Tags	Story Templates	Partnerships
Terms	Terminal Reader	Testimonials	Startup Package
Updates	Top Stories	Why Write	Writing Contests

THE HACKERNOON NEWSLETTER

Quality Reads About Technology Infiltrating Everything

name@company.com

Subscribe

☐ Yes, I agree to receive electric content at Noon by HackerNoon

Get our mobile app on
App Store

Get our mobile app on
Google Play

© 2023 HackerNoon. All rights reserved - PO Box 2206, Edwards, Colorado 81632, USA

HACKERNOON

