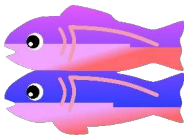
You Got This!

Zine

In this issue:

- Creating Web Apps with React
- Paths into Programming
- Top Tips from React Experts
- What Makes Developers Happy?
- Glitch Creator Profiles

v2.0



Glitch is the friendly community where you'll build the app of your dreams.

With working example apps to remix, a code editor to modify them, instant hosting and deployment – anybody can build a web app on Glitch, for free.

Learn more at **glitch.com**

Contents

p2. When Should I Use React?

p5. Top Tips from React Experts

Paths into Programming:

p6. Suz Hinton p12. Justin Falcone p16. Florida Elago

p8. Essential Parts of a React App UI

p11. What Makes Developers Happy?

p14. What's Routing?

p18. Understanding Redux

Glitch Creator Profiles:

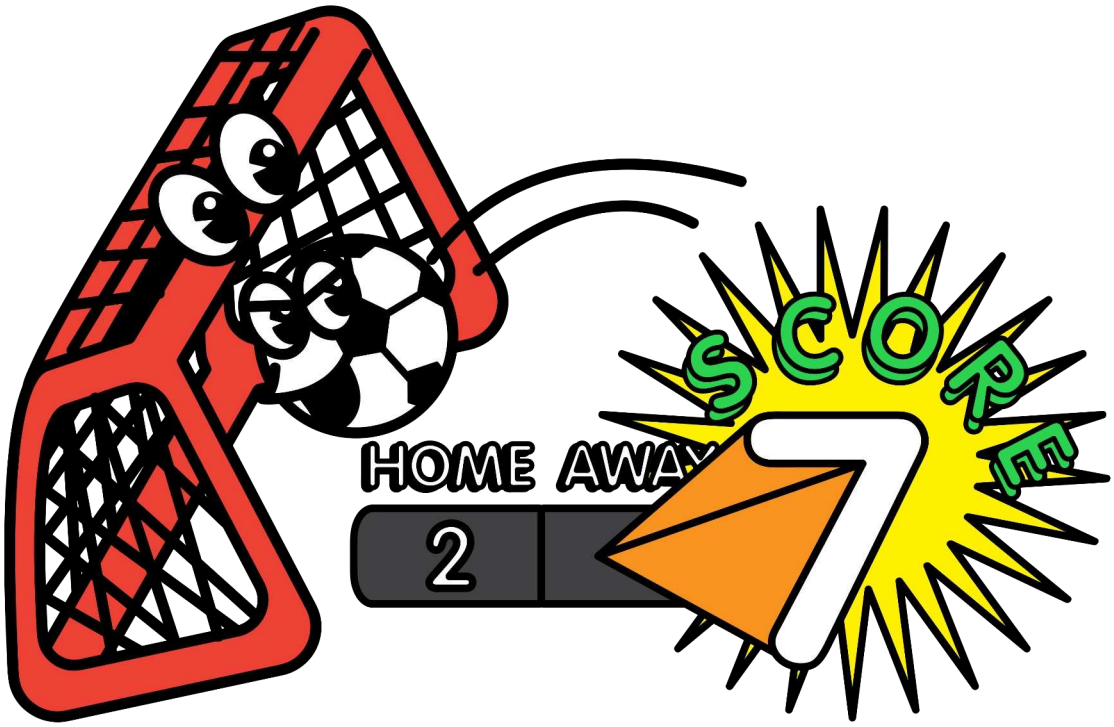
p22. Monica Dinculescu p23. Omayeli Arenyeka

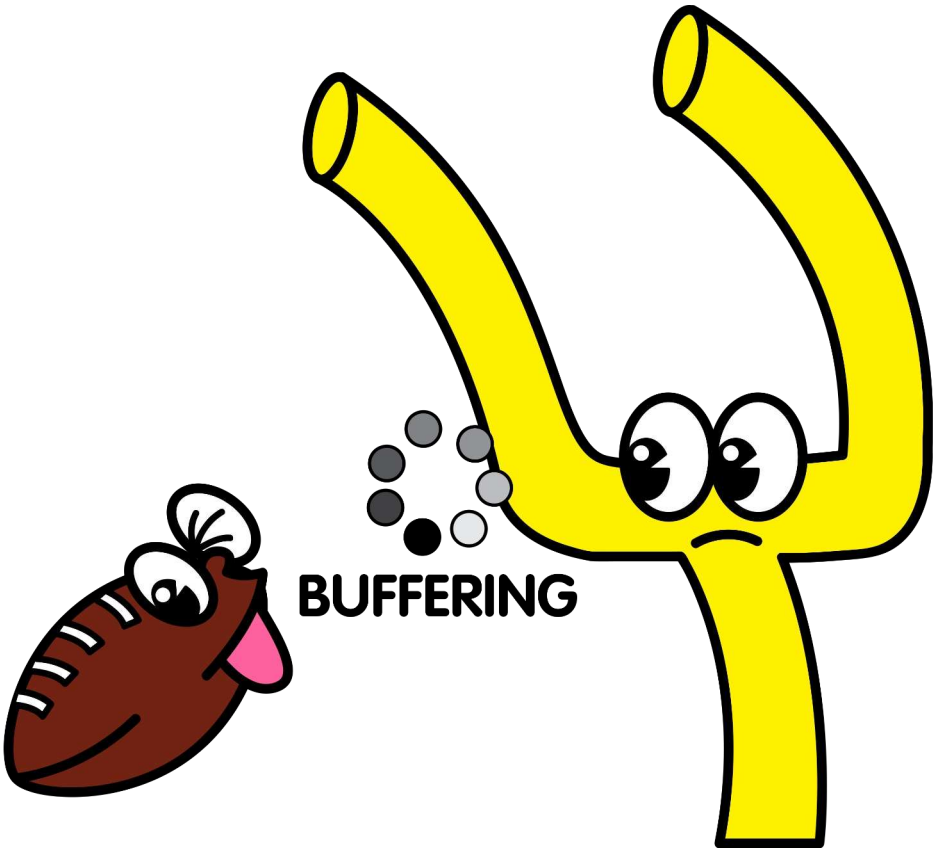
When Should I Use React?

React is a popular JavaScript library created by engineers at Facebook. It’s used for creating and managing a user interface (UI) in an app or website.

You can use it for all types of web apps, but it’s most useful when creating large apps that have data which changes a lot.

If you have a portfolio site or a blog where the data doesn’t change very often, then React may not be the best choice. But imagine you were creating an app that shows live football scores and stats in real time as the teams are playing. This would be a complex app that has to display data which is changing all the time. You’d need to keep the UI up to date so people knew what was happening, which is what React is great at doing.





Without React you can update a webpage, but with some methods it’s like clicking refresh in your browser – everything has to load again. If only one thing changed, like a team scored and only the score needed to update, then reloading the whole page would be inefficient.

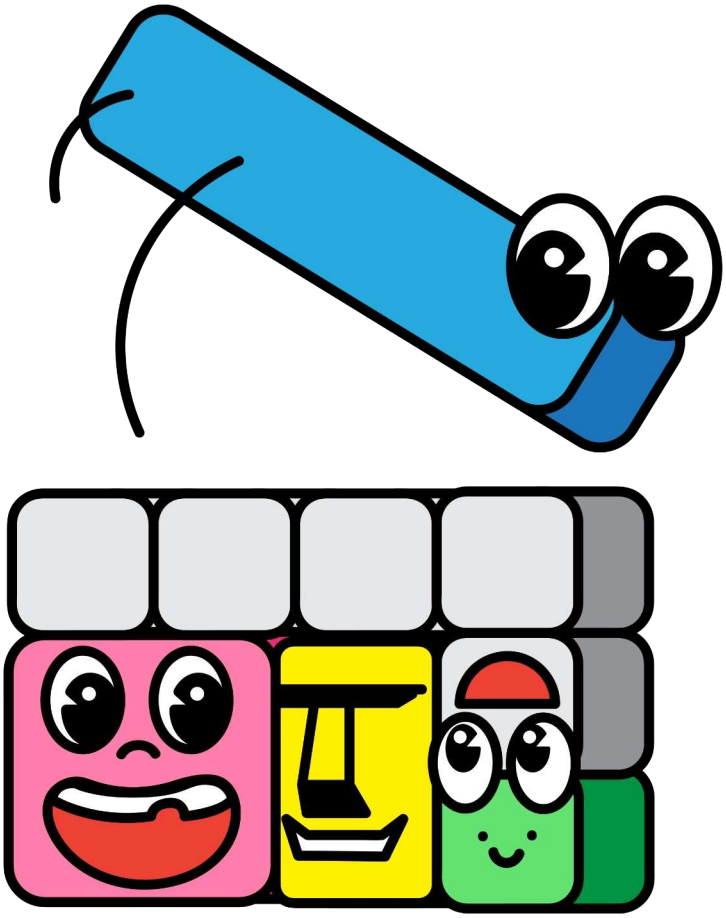
There are other ways of keeping the page updated, like looking out for events and replacing bits of code when things change. But these can make it difficult to understand your app’s code and can be hard to manage.

React apps are different. You build them in small, reusable chunks known as components. React helps you to make sure each of these components shows the latest data by updating the relevant one only when the data has changed.

By creating lots of small components, they’re easy to test and you can reuse them. This means you don’t have to write code that does the same thing elsewhere in your app.

Compared to other libraries that help solve similar problems, many people prefer React because it’s fast, it uses native browser features and you can use it alongside standard HTML, CSS and JavaScript.

It’s well documented, has strong tooling, and there are many resources for learning how to use it. React is also supported by a large and passionate community.



Top Tips from React Experts

"Create a folder for a component instead of multiple files."

Nirmalya Ghosh, Founder at Infismash

"Help React out by telling it when it should and shouldn't render: overwrite *shouldComponentUpdate* to give it more 'smarts'."

Lucy Bain, Developer Team Lead at Atlassian

"Libraries come and go, but good patterns are immortal. Learn the common patterns that inspire your favorite React.js projects."

Bertalan Miklos, Full-Stack Developer at RisingStack

"*propTypes* are a convenient way to document what props your component expects and what they do."

Alex Early, Developer at NPM

"Make your components significantly smaller than you think they need to be."

Cam Jackson, Senior Developer at ThoughtWorks

"React dev tools allow you to do really cool things, especially if you use Redux to go back in time and history to view the state that your application was in after a certain interaction."

Suz Hinton, Cloud Developer Advocate at Microsoft

Suz Hinton

Suz is a Cloud Developer Advocate at Microsoft!

She previously worked in senior web development and tech lead roles at Kickstarter and Zappos, and often codes live on Twitch.

How did Suz get into coding?

> “I learn things by reading in depth about a topic...

I got into coding when I was 9 years old. My dad brought home a second-hand Commodore 64 from my uncle and I was pretty excited for us to have an actual computer!

It had no GUI, so it was just a command line, and programming it was necessary to use it. I fell in love with coding right then and there.

Around five years later, I discovered in the library that webpages were made with code. At the time I was lucky to have a computer running Windows with Internet Explorer, so I moved on to learn HTML/CSS and Lingo/ActionScript. I had no Internet for the first ten years that I was programming; therefore, most of my early HTML/CSS knowledge was picked up in books from the library.

The first website I made from scratch by myself was a Pokémon website all about the games. I used web features such as image maps for menus and a page view counter at the bottom of the site. I was so proud of it and told everyone who would listen about it! You can see the last updated version of the website at:

https://pokemonshack.glitch.me

These days, I learn things by reading in depth about a topic, and then trying it out in code myself once I feel confident enough about where to start.

I am definitely more effective at solving problems when I know how everything I'm using works underneath the abstractions.

Essential Parts of a React App UI

The UI in a React app is modular and based around three main things: components, props and elements.

1. Components

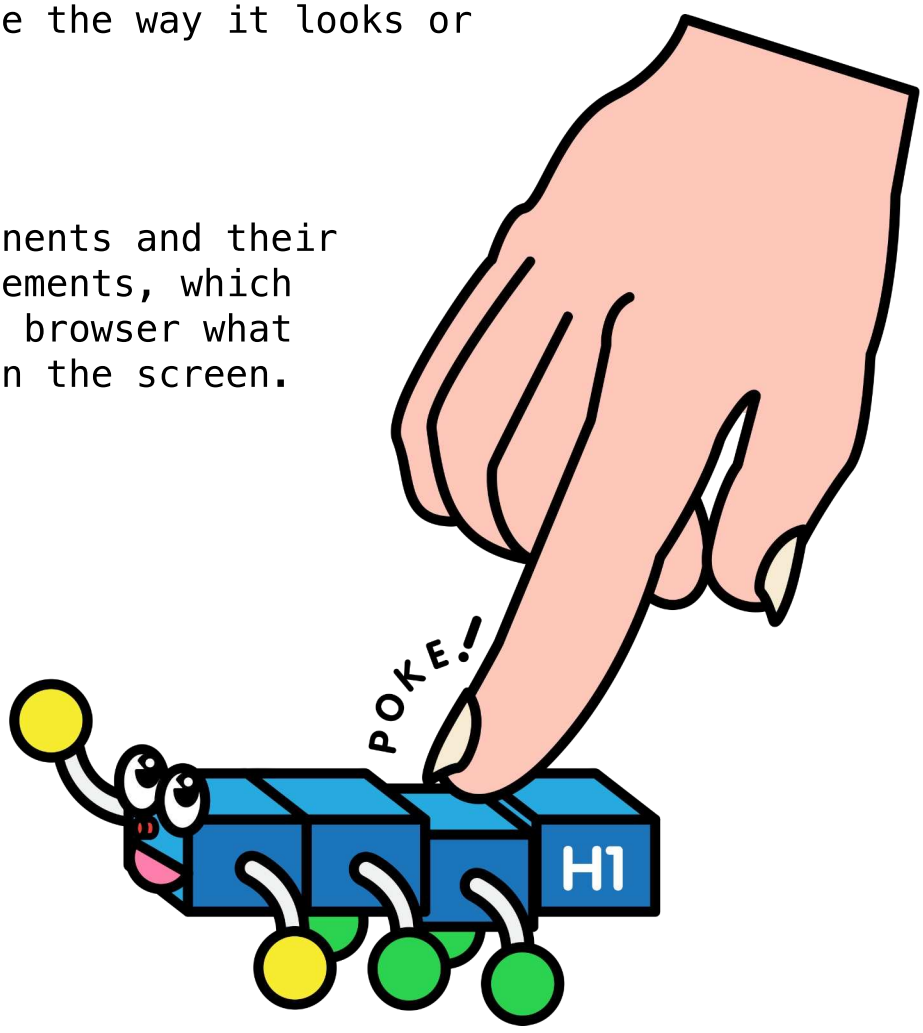
Components are visual blocks that make up your web app's user interface. They're a collection of parts of the UI, like buttons, paragraphs, or links, and are made with a combination of HTML, CSS and JavaScript code.

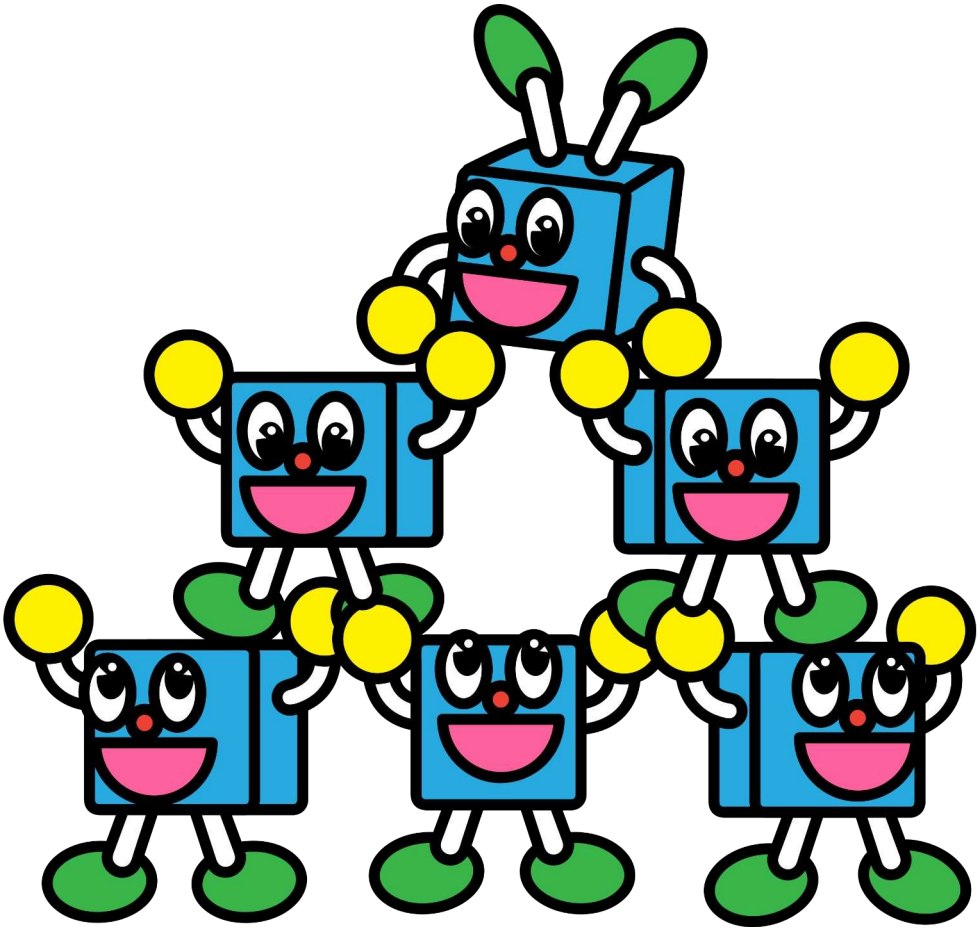
2. Props

You can pass data called "props" (short for properties) into a component, such as a name or the latest football score, which can change the way it looks or behaves.

3. Elements

Together, components and their props create elements, which describe to the browser what should appear on the screen.



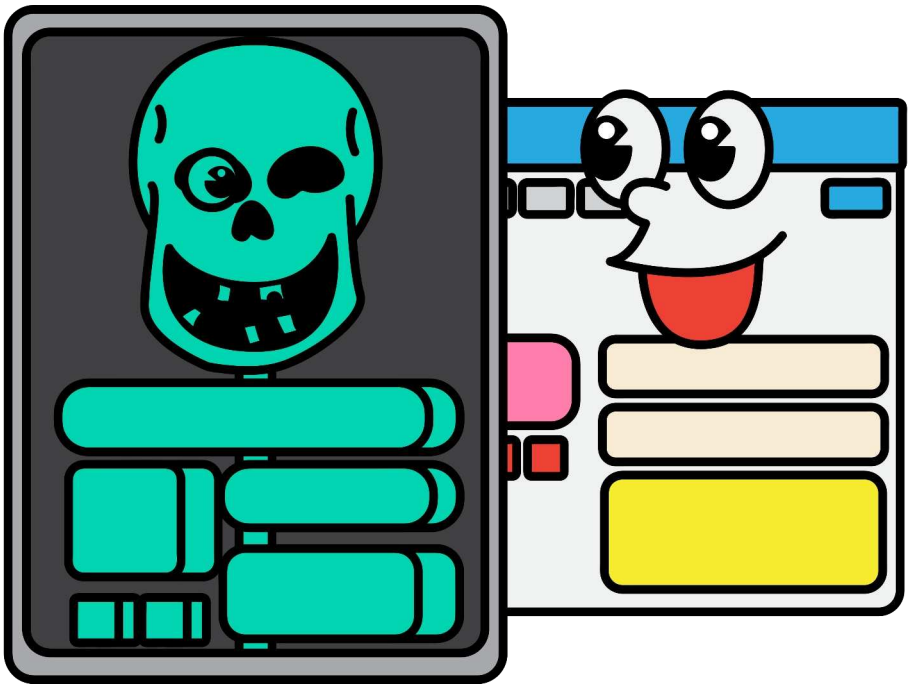


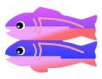
Components are organized in a component tree, meaning components can have a parent-child relationship.

Think of it like a family tree – a grandparent might have some children, and those children can have kids of their own.

Similarly, if you create an activity feed in a web app as a component, it might have child components, which would be the individual activity updates. Each of those activity updates can have child components too: the UI elements it needs to display the update, like a title, a paragraph and a date and time.

Because web apps in React are made up of these trees of visual blocks, it can be useful to map out which parts of the UI could be grouped together and what data will need to be shared between them – a bit like putting together a skeleton of your UI. By doing this, you can work out the components you'll need to create when you come to code up your web app.





glitch.com/~starter-react

In starter-react, we take a look at a simple app that uses React and the tools and concepts you need to understand it.

What Makes Developers Happy?

We asked some software engineers when they're happiest while coding.

"When I have the Eureka moment. Sometimes I struggle for hours to solve one thing, and then all of sudden the answer comes from nowhere."

Tomomi Imura, Developer Relations at Slack

"Simply when I'm actually coding, instead of doing the things I need to get to the point where I can be programming."

Chris Fidao, Software Engineer at UserScape

"I love reading code that is beautiful and concise. And of course, I always relish getting tests to pass."

Lindi Emoungu, Senior Software Engineer at Google

"I'm happiest when I know what I'm doing and I enter this zen mode. I love when I get to focus on getting things just right and paying attention to the details of the code."

Saron Yitbarek, Founder at CodeNewbie

"I like to plan ahead. I'm happiest when I have a solid plan and I'm working towards completion."

Leah Culver, CTO at Breaker and author of OAuth and oEmbed specs

"I'm happiest when I'm coding something where I know I have enough time to do the complete right thing."

Casey Muratori, Independent Games Programmer

How did Justin get into coding?

> "I spent a lot of my childhood playing on hand-me-down computers.

I've been around computers my whole life: my grandfather wrote software for an insurance company, my grandmother taught computer animation, and my mother was a graphic designer.

I spent a lot of my childhood playing on hand-me-down computers.

Sometime between fifth and sixth grade, a friend of mine got me into a program called *HyperStudio*, which was a little bit like a programmable PowerPoint. We made simple point-and-click games.

Around that time I was also getting into "customizing" my software and modding games; on the old Mac there was a program called *ResEdit* that let you open up programs and system files to change graphics and text. It was quite easy to break stuff this way, which became its own learning experience. I wasn't writing any code, I was manipulating all the other parts of the program.

In high school, I made a personal website full of in-jokes and information about my one-man band. I also started making games, websites and even school presentations in Flash. I was taking computer programming courses, though somehow it didn't even occur to me that my programming class and my Flash toys were at all related.

After I flunked out of a computer engineering program at college, I spent several years working retail jobs and not programming at all.

But after I moved from Boston to New York, I got into it again -- suddenly I understood how both the programming and the "non-programming" I had done when I was young were interconnected.

Justin Falcone

Justin is a Software Engineer at Buzzfeed!

He co-organizes *Write to Speak*, New York's favorite tech talk writers' workshop, and is an avid karaoke culture fan.

What's Routing?

Imagine you're walking along a path that splits into several smaller paths, each one going in a different direction.

To know which path you need to follow, you might look at a signpost that tells you the path that leads to your desired destination.

Similarly, behind the scenes of a web app, you need to direct visitors to the destination web page they've requested. If someone typed a URL in their browser like *'yourwebsite.com'*, then they'd expect to land on the homepage.

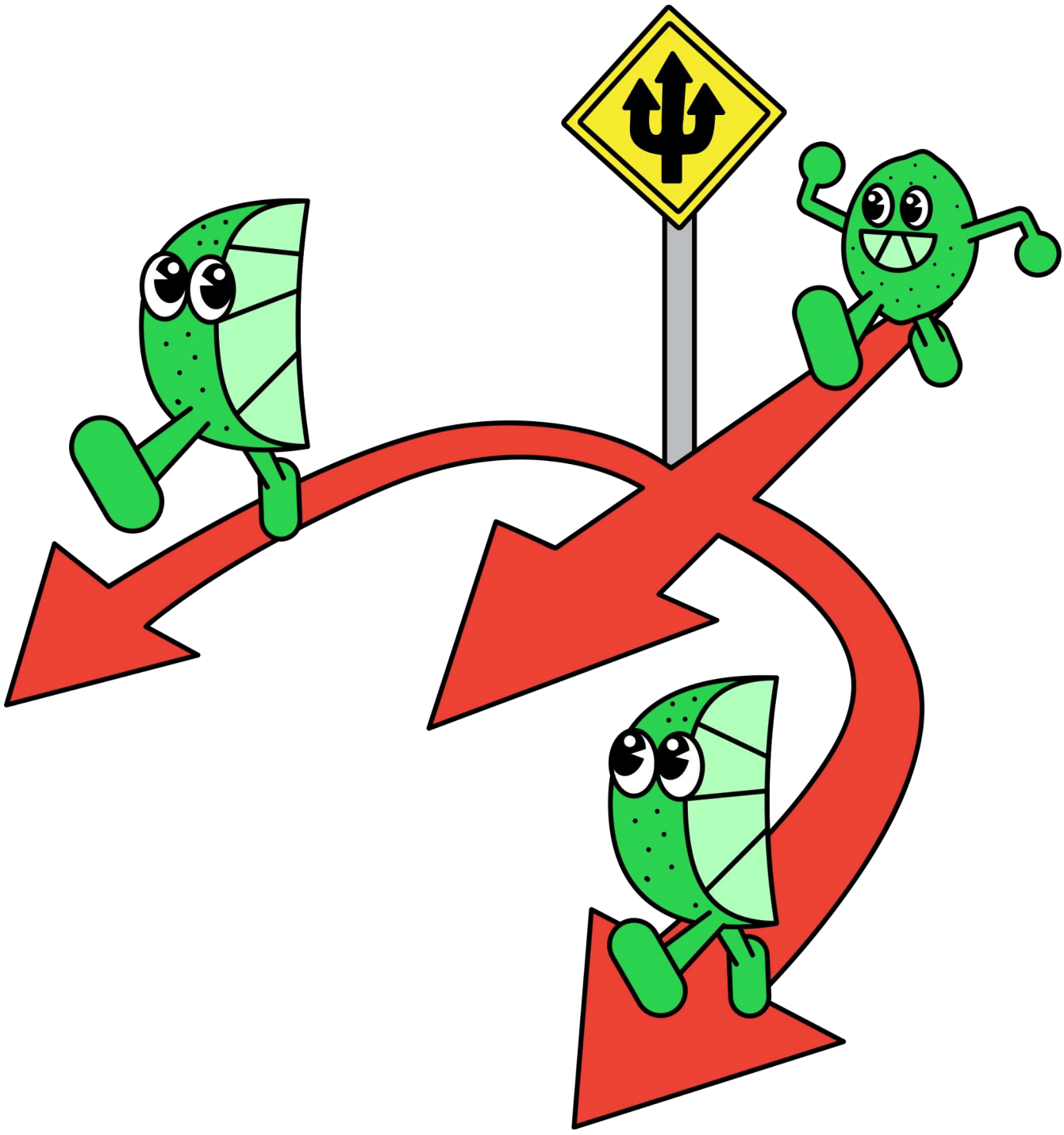
But if they specified a path like *'yourwebsite.com/route/to?something=specific'*, then they'd want to be taken to a particular page or section of the site.

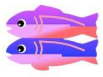
You can let your web app know what to do with a request using routing. Routing is a way of telling your app which part of your code should handle a particular request.

For example, let's say your website has three pages: *About*, *Scores* and *Contact Us*.

These could be mapped to three separate routes or URLs: */about*, */scores* and */contact-us*.

To do this in React, you can use React Router: a library that lets you handle routing. With it, you can map URLs to components or groups of components, and specify how to change the way your app renders when someone clicks a link on your site.





glitch.com/~starter-react-router

Starter-react-router introduces you to routing, and explains how you can use React Router to add links to pages.

Florida Elago

Florida works as a Software Engineer at Casper!

They previously worked in software development roles at both Hootsuite and Shopify, among others.

How did Florida get into coding?

> “Friendster, believe it or not, played a huge role...

This was the first line of code I ever wrote:
<marquee>WELCOME TO MY PAGE</marquee>

It was written after a friend of mine showed me how to get those “moving text and images” I kept seeing on her *Friendster* page.

My curiosity then lead me towards discovering “View Source” on my IE6 browser displaying plain-text HTML, CSS and JavaScript. I barely knew what anything meant back then -- I thought that these cryptic random texts would even contain emails and passwords of any profile I looked at if I looked hard enough.

I just understood that if I typed '*<marquee>text</marquee>*' it would make something move on the page.

I thought that I could just type whatever I wanted inside an HTML tag. I thought that *<jump>* would make something jump and *<fly>* would make something fly up my page. I soon discovered that this wasn’t the case, but I did discover *<blink>*, which was great.

My marquee tags started to develop. I learned about attributes that you could pass in such as *behavior*, *width*, and *direction*. I thought I was UNSTOPPABLE. I had my marquee bouncing around, going in different directions -- I even figured out how to put glitter text images inside them.

Friendster, believe it or not, played a huge role in my early years as a young coder -- I would spend *hours* on that website.

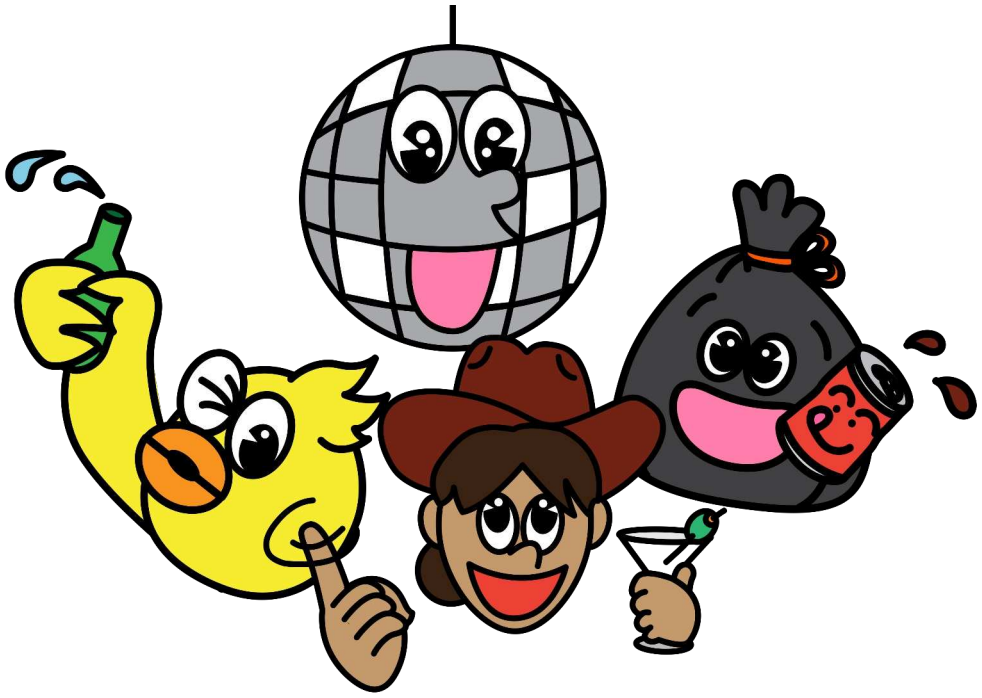
Soon enough, I started making random websites for myself, almost creating one every other day.

Understanding Redux

A button in a UI can have what’s known as a state: a condition that indicates whether it has been pressed or not.

It’s useful to know a button’s state because you can then make it appear or behave differently based on that state. In fact, we can capture almost anything we want to keep track of with state; it’s just a description of what's going on in your app. You can use state throughout your web apps to update and change all aspects of the UI. Each UI element will be part of a component, and every component can create, update and use state. Although this can become trickier to manage as your application becomes more complex.

Imagine you’re a member of a remote tribe and each year you throw a party together. You don’t have phones or laptops, so you’ve always made sure the event ran well by talking to each other.

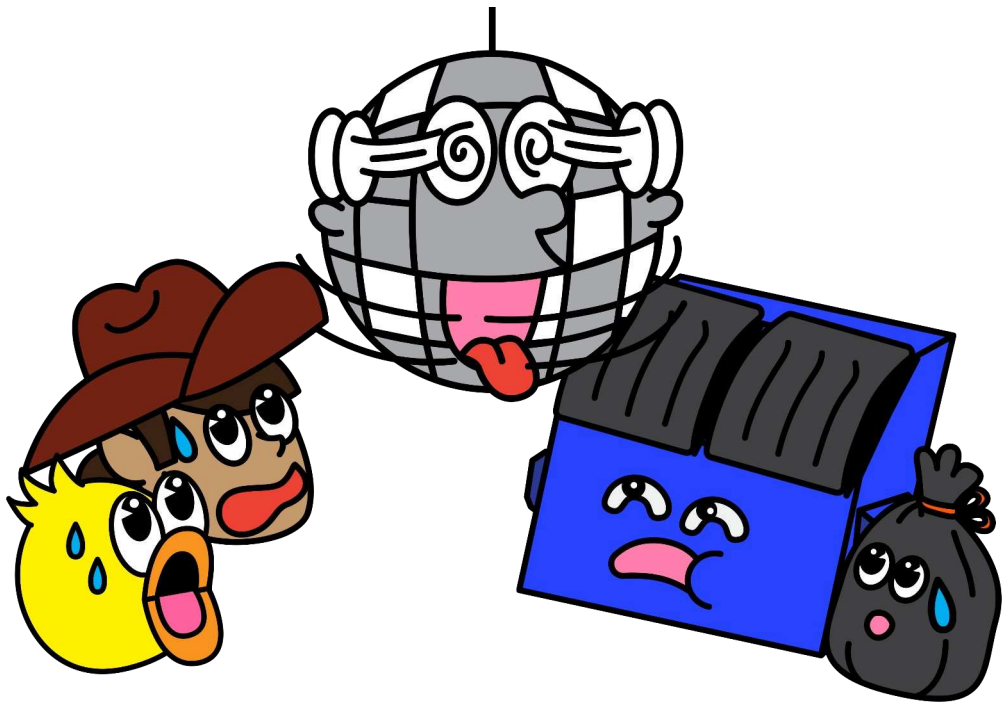


When something happened during the party, you would run and let the others know, and you would all figure out what you needed to do about it.

This used to work well, but as the tribe and the party has grown in size, it’s become too difficult to keep track of who needs to know what and when. If something happens that a lot of people need to know about at once, it becomes particularly hard to manage.

This can happen in React apps too, especially as your UI becomes more complicated. What’s more, React's component tree model doesn’t work well for things that you need to access globally – like a user or a notification, which don’t map directly to a UI element. This is why React has a library known as *Redux*.

Redux is responsible for managing state for your whole application. It’s not something you’ll need to use in every app, but you will sometimes want to add it to help you when managing state gets tough.



You can think of Redux as being like the tribe's god – a powerful, fourth-dimensional being that knows everything about what's happening during the party thanks to its all-seeing eyes. In React, Redux is the god of components. It works by adding a main state tree, which has a *store* and uses *actions*, a *dispatcher*, and *reducers* to make changes and talk to everything.

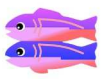
The store holds the whole state tree of your application. The state tree knows about everything in the app, including our button, and what has happened to our button.

If someone presses our button, it sends an action. Actions are little data payloads that describe a thing that happened in your app. A user being added is an action, for example. Actions are sent via the dispatcher, which is the central hub of data flow in your React app.

Editor note: Any resemblance god has to Alanis Morissette is purely coincidental ;)

Actions get sent by the dispatcher to what are called reducers. Reducers specify how the application's state changes in response to actions sent to the store. They use the information provided to figure out what the global state needs to be. From there, data then flows back down to components so that they can update themselves.

Because actions go up and data goes down, this means our components don't need to manage state themselves or talk to each other. If the tribe used this technique, they would no longer have to discuss everything among themselves - they could just ask their god. They'd send a message to their god describing what has happened, and their god would tell them what to do in response, leaving them free to enjoy the party!



glitch.com/~starter-react-redux

The starter-react-redux project explains how to manage 'state' in your app, and how actions work with reducers.

Glitch Creator Profiles

Creative coders, designers, artists, activists, students, and educators are all building the apps of their dreams on Glitch. Meet two of these creators:

Monica Dinculescu

is an *Emojineer* at Google, where she works on Polymer, web components, and Chrome.

She's passionate about emoji, web fonts, and cheese.

Monica's creations run the gamut from tutorials that go into nitty-gritty technical issues, to what she describes as "a lot of pretty silly side projects." Here are just some of her projects:

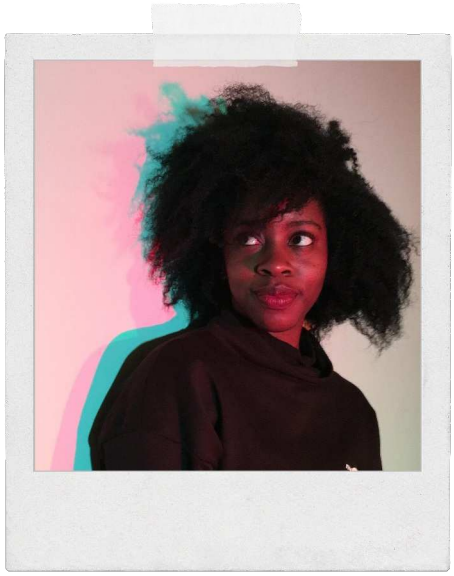
Font-Display: An explainer about loading web fonts, covering potential problems and how to avoid them.

Float-Layout: Describes how the algorithm that browsers use to position floats actually works.

Emoji-Garden: A random emoji garden generator that makes cute little gardens.

Omayeli Arenyeka

is better known as Yeli. She's a creative technologist who doesn't think of herself as a coder, but "more of an artist that uses code as a tool."

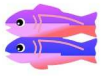


Yeli's projects explore the intersection of art, design and technology. From Twitter bots to data analysis and visualizations, here are some of her projects:

Y-Under-Y: Created as a response to youth-obsessed culture, this bot generates awards for any age.

Art-Connoisseur: A bot that creates tweets interpreting art alongside images from Artsy.

Twit-Original: A project which analyses your tweets determining whether you're a tweeter or retweeter.



Check out these and other projects on Glitch:

Monica: glitch.com/@notwaldorf
Yeli: glitch.com/@oa495

"A lot of beginners underappreciate the degree to which Googling things is part of a programmer’s job."

Steve Klabnik, Rust Docs Team Lead at Mozilla

Keep on Googling! 💪🏽

Credits

Illustration: Good Boy Graphics

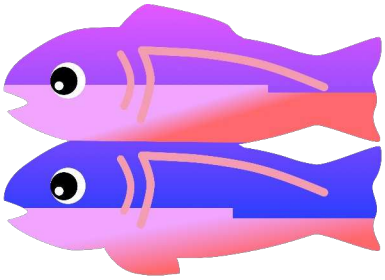
Layout: Gareth Wilson

Content: Jenn Schiffer, Gareth Wilson

Edited by: Maurice Cherry

Technical Consultant: Jude Allred

Guest Contributors: Suz Hinton, Justin Falcone, Florida Elago



CC BY-NC-SA 4.0
Print & Share!