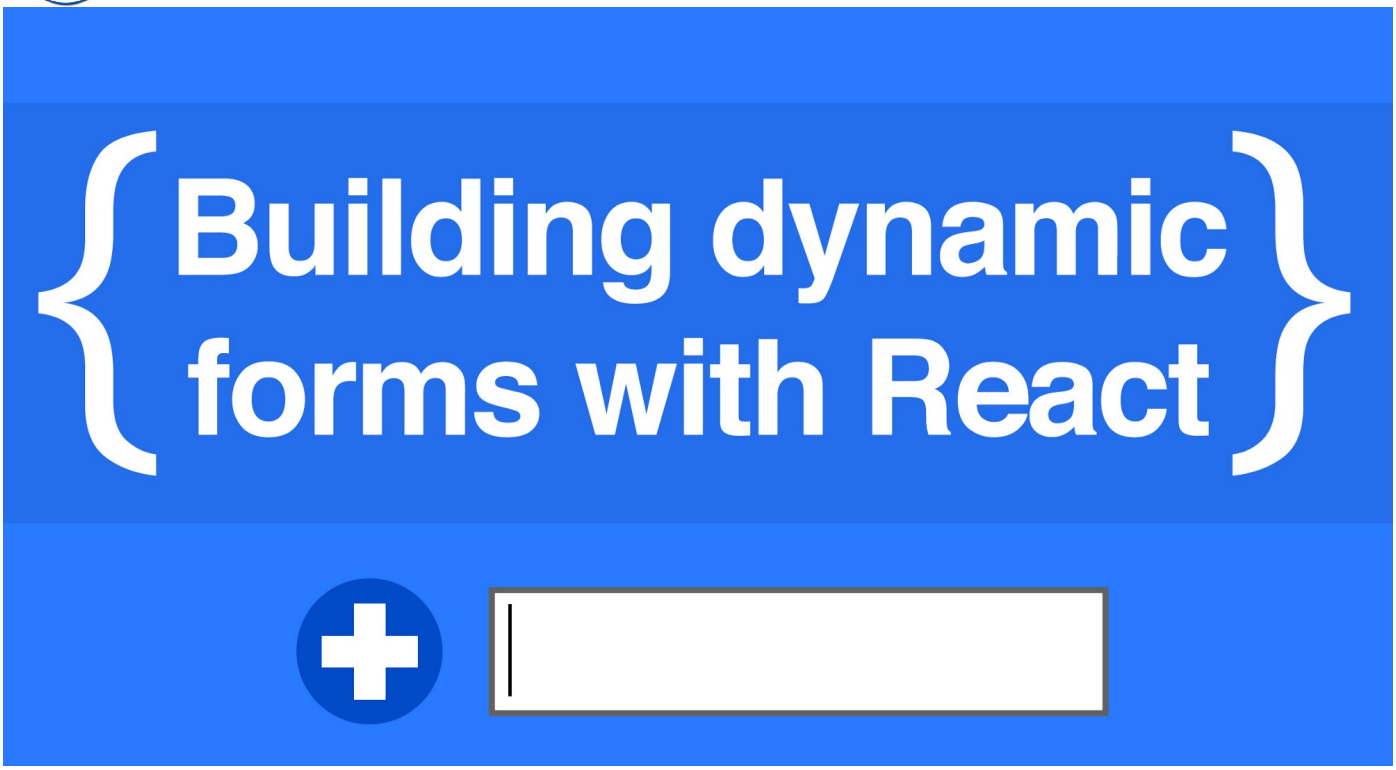
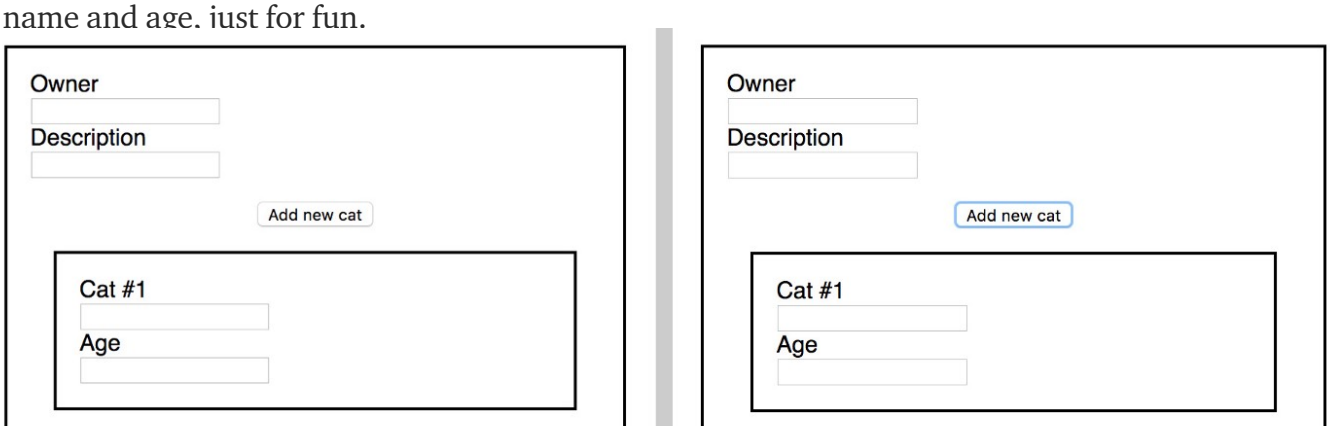
Building a Dynamic, Controlled Form with React

Mike Cronin in ITNEXT Follow
Jul 4, 2018 · 6 min read ★



React makes building UI's in JavaScript easy, but some things can still be a bit tricky to wrap your head around, things like dynamic forms. A **dynamic form** is one where the user is able to decide how many inputs there should be. For this code along, we're going to have standard inputs for a cat owner's name and a short description, and then dynamically add cat inputs. Each new cat will have a name and age, just for fun.



This is the bare minimum of what we're designing today

You can code along with me, or just skip to the end and see the full code, and refer back to any parts that confuse you.

Deciding on what we want

Planning is crucial with React, it can save you so much time if you wireframe out what you want before you start coding. We know what it will look like, but how will it be made? I like to have a little React checklist:

1. What components will I need?
2. Which of them will have state?
3. Are there events that we need to deal with?

In this case, I feel like it would read nicely to have a main Form component, and then CatInputs components that we can just render as we go. As for state, our Form component will need it, since we'll control each input. But our CatInputs components will not require state, since we can just pass everything in as props.

Finally, there are 3 events: we need to handle the form getting submitted, we need to handle changes to each input, and we need to handle the button that adds new inputs.

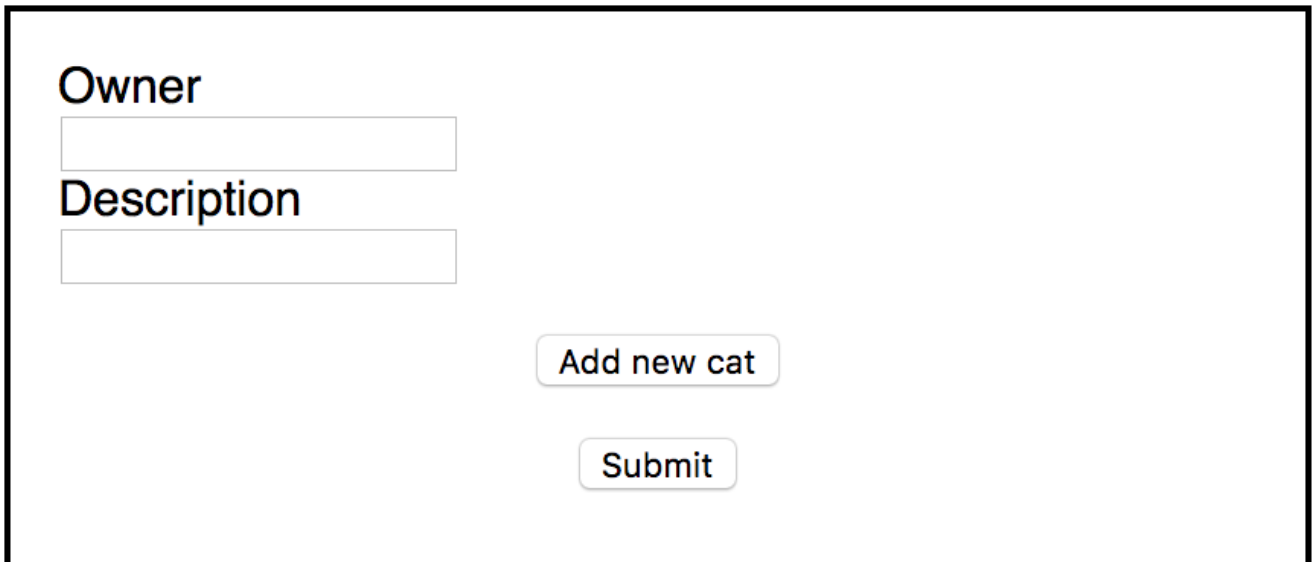
Order of Attack

In general, I like to render whatever I need first, and then start working on the interactivity. In this case, we'll render the static base Form, then figure out how to render the new inputs, and then finally we'll deal with controlling them. For the sake of this tutorial, I will build everything in the Form component, and then once everything works, I'll refactor the proper sections into a CatInputs component.

Getting started: render()

Lets build out the non interactive part of the form first:

If you're following along, this is basically what we made (yours will look different, I added some styling):

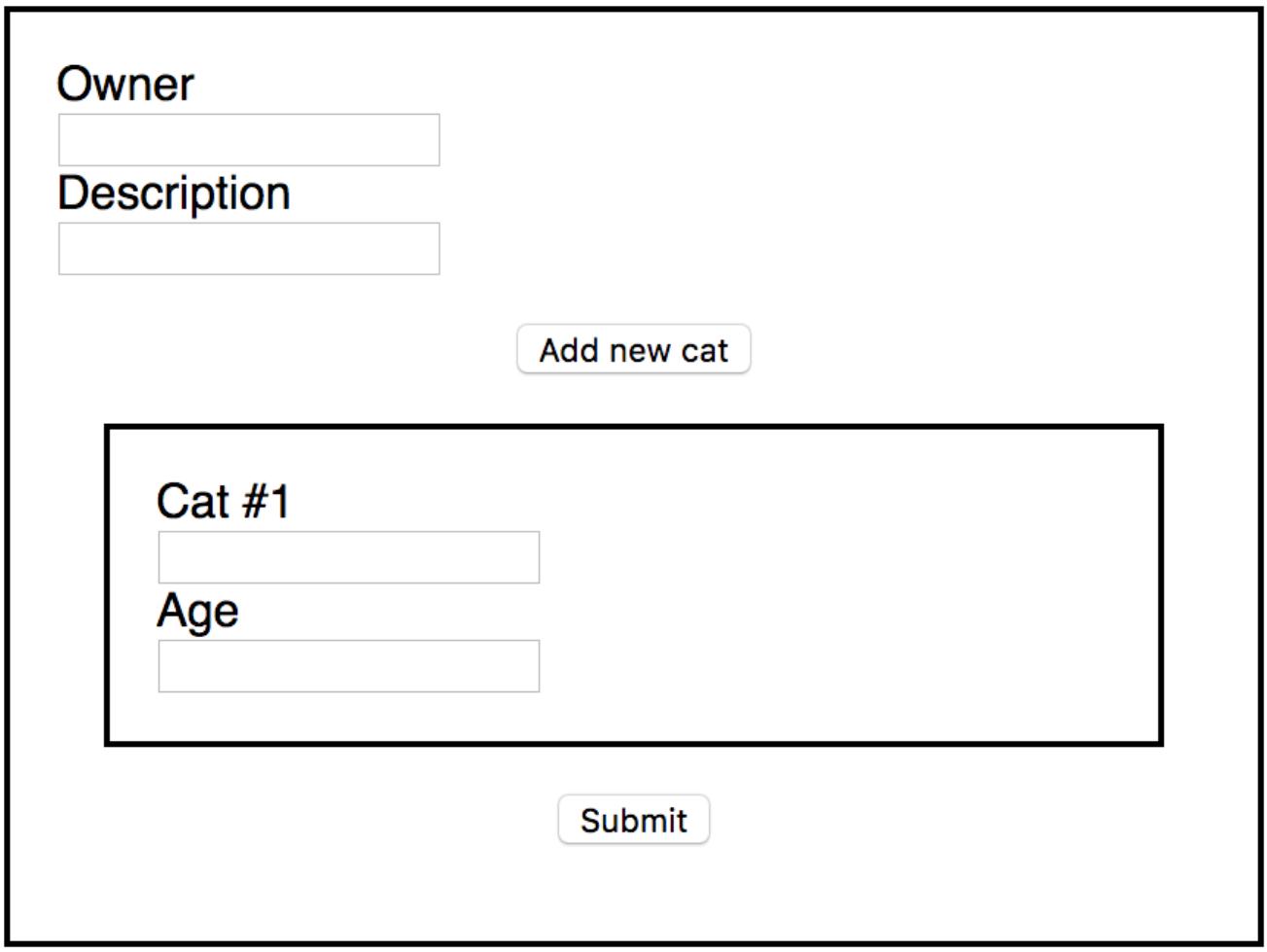


Beautiful.

Using arrays for dynamic inputs

Before we code, let's talk about how we are going to do this. Basically, we're going to have an array of cat objects in our state. Each object will have a `name` and `age` value. Our Form will iterate over this list and create two new inputs for the `name` and `age`. When we click the "add new cat" button, we'll add a new object to our array. Since this will change our state, it will trigger a re-render. Then, our form will iterate over this *new* list of cats, it will add another pair of inputs.

To start, lets just worry about putting the first blank cat object into our state, and then render that:



Looking good but not actually dynamic yet

that dataset will be crucial to controlling our inputs later, since it matches the inputs to the index of the corresponding cat object in the cats array.

Adding inputs

Since our form is creating two new inputs, we know that the iteration aspect is working. But for it to truly be dynamic, we have to be able to let the user add the inputs. React's state feature makes this really easy. We just need to give our component a method that adds a new blank cat to our array. Since we're clicking a button inside a form now, we'll also need to add a `handleSubmit` method which will stop our form from automatically reloading the page:

All `addCat` does is set the state with a spread of the previous state's `cats` array with a new blank `cat` object tagged on the end. This will make our component re-render, and when it does, the form will iterate over the new length of the array, giving us another pair of inputs.

Controlling the forms

Now that we have our dynamic inputs down, let's actually control them. Here is the method that will control changes in *all* our inputs, static and dynamic, and our final state:

Let's cover the `else` condition first, since it refers to our static elements. We're using good old `e.target.value` to grab the value of the input that the user is typing into, nothing shocking there. But we're using `[]` so that we can dynamically match our state using each input's name attribute. These are called Computed Property Names, check them out. So our `owner` input has a name of `owner`, which means our `setState` translates to `owner: whatever-was-typed`.

Now for the fancy part; handling our dynamic inputs. First, we're checking whether the event's class matches our dynamic inputs. If it does, we make a copy of our `cats` array of objects using the spread operator. Now it gets *real* fancy. We use `e.target`'s dataset to match the input to its corresponding object, and then we use the `e.target`'s classname to grab either the cat object's `name` or `age` value. So, if a user types into the first cat name input this is what it would translate to:

```
cats[e.target.dataset.id][e.target.className] = e.target.value
// cats[0][name] = whatever-was-typed
```

We're just using brackets to dynamically access elements of our array and attributes on the objects. Now that our copied list has been mutated to reflect the new input, we use `setState` to save the change to our state and trigger a re-render of our form.

That may take a second to get comfortable with, but by making our change handler dynamic, we can put it on the whole form instead of each input:

```
<form onSubmit={this.handleSubmit} onChange={this.handleChange} >
```

Putting it all together

Here is our finished code in one component. It has all the inputs controlled, and to prove it, it uppercases everything:

There you have it. A basic run through of making a dynamic form in react. Follow this basic pattern as a jumping off point for your next project, and try to find some parts to streamline once you understand the process.
happy coding everyone,
mike
my other articles

React JavaScript Coding Flatiron School Programming

WRITTEN BY

Mike Cronin

I'm Mostly Focused on everything. Follow me on Instagram to keep in touch: @mostlyfocusedmike

ITNEXT

ITNEXT is a platform for IT developers & software engineers to share knowledge, connect, collaborate, learn and experience next-gen technologies.

What is the N+1 Problem in GraphQL?

Mike... Jun 26... 539

JavaScript Fundamentals: Mastering Functions

Timot... Jul 4 · ... 138

Creating Scroll Animations in Flutter

Kenne... Jul 5 · ... 201