# Learning Objectives

- **Define ReactJS**
- **Add React to a website**
- **Define React components**
- **Add a button component that responds to a click**
- **Mount a component to the DOM**

# Clone Blango Repo

## Clone Blango Repo

Before we continue, you need to clone the `blango` repo so you have all of your code. You will need the SSH information for your repo.

### In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/blango.git
```

- You should see a `blango` directory appear in the file tree.

You are now ready for the next assignment.

# Intro and Purpose

## Intro and Purpose

ReactJS is a JavaScript framework for building user interfaces. These interfaces are composed of different *components*. Each component has an internal state and knows how to render itself. Components may have child components, and data is passed between components using properties – kind of like arguments to a function.

Frameworks like ReactJS exist because in the early days of JavaScript development, keeping the UI and data state in sync was a manual process. For example, a user would click a button to fire an event. The developer would have to write JavaScript code to handle this event and update the state. Then, write more code to manually locate the element in the UI that needed updating, and even more code to correctly update its content.

With React, we just need to update the component's state, and it knows if it needs to be re-rendered, and its `render()` method is called automatically.

## Adding React to a Website

There are different "levels" of how ReactJS can be integrated with your website. You could build the entirety of your UI with it, and forgo Django templating entirely. Or, you can just include the ReactJS script files on pages in which you want to have some interactivity. We're going to use the second approach.

In JavaScript, variables and functions defined in script files or in `<script>` elements are accessible by subsequent JavaScript code further down the page (assuming they are in the global scope). This means we must include the ReactJS scripts in an HTML page, and then our scripts further down the page, and we can then access the ReactJS methods in our own JavaScript file.

There are ReactJS files that are hosted for developers to include on their own page, which means we don't have to download the scripts and put them in our local `static` directory. To use the scripts on your own pages, include these `<script>` tags:

```
<script
        src="https://unpkg.com/react@17/umd/react.development.js
        " crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-
        dom.development.js" crossorigin></script>
```

▼ Production

When going to production, you can use the production versions of these scripts by replacing `development` with `production.min`. This will include a minified (stripped down) version of these files that is quicker to load.

The other thing that needs to be added to our HTML page is an element to contain the React components. This is usually an empty `<div>` – React will dynamically populate it with HTML elements. The only attribute the `<div>` needs is an ID so we can find it. This could be called the *root element* or *container element*. Something like this is fine:

```
<div id="react_root"></div>
```

## Try It Out

Let's start by making the necessary changes to the template, and the rest of the work is in the JavaScript file. Open `blog/templates/blog/post-table.html`. We'll add the containing row and column divs without using our helper template tags so we can see more explicitly what elements are being used. Inside the column, we'll add the root React element. Put this HTML directly after the opening `{% block content %}` tag:

```
<div class="row">
    <div class="col">
        <div id="react_root"></div>
    </div>
</div>
```

Then underneath this, add the ReactJS `<script>` tags:

```
<script
        src="https://unpkg.com/react@17/umd/react.development.js
        " crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-
        dom.development.js" crossorigin></script>
```

You should leave the existing `<script>` tag that includes `blog.js` where it is, after the above `<script>` tags and before the `{% endblock %}` for `content`.

There's no need to refresh the page since it won't look any different, and the `blog.js` file hasn't been updated yet, but we'll do that next.

# Building React Elements and Components

## Building React Elements

React *components* are the classes that contain the state, event handlers and render methods. What is rendered on the page are actually React *elements*.

▼ **Function-based components**
React components can also be built using functions, which is actually the preferred way as they are more performant. However, they can be a bit harder for a beginner to understand, so we'll stick to class-based components in this course.

The component's `render()` method should return a React element, which will be added to a page.

The element can be created with the `React.createElement()` function. This takes three arguments:

- The first is either the name of the element to create, such as `div`, `span`, etc. Or, it can be another React component as a child.
- The second argument object (dictionary) of properties/attributes to set on the element. These can be standard attributes like `id`, `onClick` or `href` (depending on what the element supports). Or custom properties that the child React component supports can be used.
- The third is a list of children, this can be a single string, or an array of strings or other elements.

An important thing to note regarding properties: since `class` is a reserved keyword in JavaScript, we can't see the `class` of an element using it. Instead we need to pass `className`.

▼ **React.createElement**
`React.createElement` is sometimes aliased to `e`, for example: `const e = React.createElement`. This is to reduce the amount of code to write. You might see this used in some other code samples. We won't be using this alias as we'll only be using `React.createElement` twice, and only temporarily.

Let's see `React.createElement` in action. This will create a `<button>` with the Bootstrap button classes and the text `Click Me`:

```
React.createElement('button', { className: 'btn btn-primary' },
        'Click Me')
```

Next let's build a component that renders the button with an `onClick` handler.

## Building React Components

Basic React components are actually quite easy to write. They are just classes that inherit from `React.Component`. At the minimum you need to implement the `render()` method, and usually add some `state` (although that's not required). The `render()` method takes no arguments, so it must vary its return value by looking at the internal state.

Here we will build a simple component that renders a button. When the button is clicked the text of the button will change to `Clicked!`. Here's the class first, then we'll go through what it does:

```
class ClickButton extends React.Component {
  state = {
    wasClicked: false
  }

  handleClick () {
    this.setState(
      {wasClicked: true}
    )
  }

  render () {
    let buttonText

    if(this.state.wasClicked)
      buttonText = 'Clicked!'
    else
      buttonText = 'Click Me'

    return React.createElement(
      'button',
      {
        className: 'btn btn-primary mt-2',
        onClick: () => {
          this.handleClick()
        }
      },
      buttonText
    )
  }
}
```

First the class starts by extending from `React.Component`, the base component class.

Then, we set the `state` attribute to an object (remember, this is like Python dictionary). It has one item, `wasClicked`, which will store the "clicked" status of the button. We default it to `false`.

Next we define a method called `handleClick()`, which will be called when the button is clicked. Note that there is nothing special about the name `handleClick`, it could just as easily be called `dontHandleClick` but that would be confusing! We wire up the connection between the click and the handler method later.

Inside the `handleClick()` method we call the `setState()` method. This takes an object and updates the given fields in the component's state. Only fields that are set on the passed-in object are changed, any other fields in state

are unaffected. The state should only be updated using `setState()` because after the `state` is updated, then `render()` method is called automatically to update the display of the page. If we were to try to update the state manually, e.g. by doing `this.state.wasClicked = true` then the page would not update.

Finally, the `render()` method. It's safe to read the state variables directly (but not update them). We check the status of `wasClicked` and update the `buttonText` appropriately. We then create a `<button>` element with the following properties:

- `className` is rendered into the `class` attribute. The `mt-2` class adds a **m**argin on the **t**op, of size `2`. This is a Bootstrap specific size and if you're curious what it actually means you can read the spacing documentation.
- `onClick` is the attribute that sets the handler that's called when the button is clicked. We're setting it to an anonymous arrow function that in turn calls `handleClick()` on the component. We can't just pass `this.handleClick` directly in the object as the context would be incorrect when `handleClick()` was called, and `this` would refer to the event not the component.

The last argument to `React.createElement` is the child we want to set on the `<button>`, in this case, just the `buttonText`.

## Try It Out

Now you can add the `ClickButton` component to your `blog.js` file. Empty the contents of the file and add this instead:

```
class ClickButton extends React.Component {
  state = {
    wasClicked: false
  }

  handleClick () {
    this.setState(
      {wasClicked: true}
    )
  }

  render () {
    let buttonText

    if (this.state.wasClicked)
      buttonText = 'Clicked!'
    else
      buttonText = 'Click Me'

    return React.createElement(
      'button',
      {
        className: 'btn btn-primary mt-2',
        onClick: () => {
          this.handleClick()
        }
      },
      buttonText
    )
  }
}
```

Remember that since you are including the `blog.js` file in your template after the React scripts, you will have access to the `React` variable automatically.

The `/post-table/` page still won't change as there's one last thing we need to do: mount the component into the container. Let's see how to do that now.

# Mounting a Component

## Mounting a Component

To mount a component onto the page (or the *DOM*) we use the `ReactDOM.render()` function.

▼ **DOM**
*DOM* stands for Document Object Model and is a way of representing the HTML page (document) as a tree of objects. It's kind of like an in-memory representation of what has been written in HTML code.

`ReactDOM.render()` takes two arguments: a react element to render, and a DOM element in which to render it.

There are a few ways you could get a reference to DOM element, we'll use the `document.getElementById()` method. As you might expect, this takes the ID of the element you want to retrieve as an argument. We want a reference to our `<div>` with `id="react_root"`, which we can get like this:

```
const domContainer = document.getElementById('react_root')
```

Then, to generate a React element from our component, we use the `React.createElement()` function again. Since our Component takes no properties nor does it have children, we can pass it as the only argument.

Putting this all together, we get:

```
ReactDOM.render(
  React.createElement(ClickButton),
  domContainer
)
```

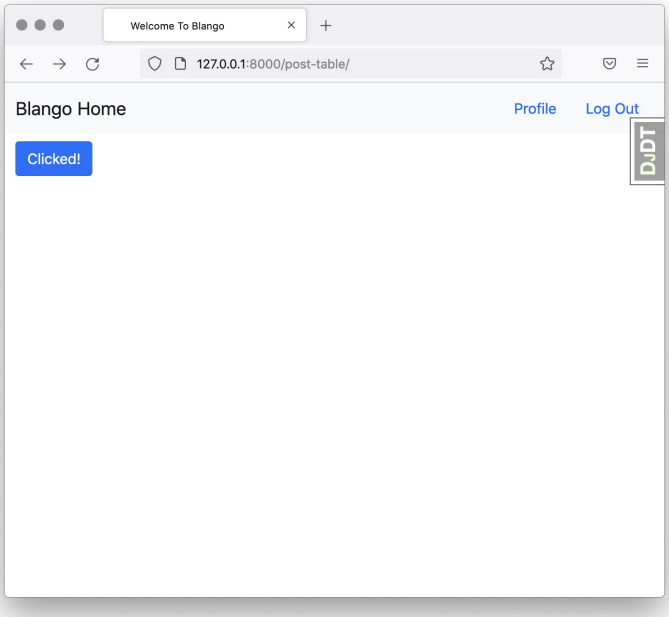
## Try It Out

Now you'll put in the final piece of the puzzle, and mount a `ClickButton` onto your page. Add this code to the bottom of `blog.js`:

```
const domContainer = document.getElementById('react_root')
ReactDOM.render(
  React.createElement(ClickButton),
  domContainer
)
```

Now load `/post-table/` in your browser and you should see the `Click Me` button. Click on it, and the text will change to `Clicked!`.

View Blog




click button component

Congratulations on building your first React component!

We mentioned earlier that we're only going to be using `React.createElement()` temporarily. In the next section we'll look at replacing it with JSX.

# Pushing to GitHub

## Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Finish ReactJS"
```

- Push to GitHub:

```
git push
```