In this video, I want to introduce you to React and go over the core concepts I think every React developer should aim to learn and master. When it came to putting these concepts together, I selected them based on things you'll need to build out most of the functionality that you see in websites today and things somebody interviewing you would probably expect you to know. But before we get started, I want to quickly mention my React crash course that's linked in a video description. In this course, you will learn more about the concepts I mentioned here while building a fun Notes application So what is React? React is a JavaScript library for building out user interfaces. When you look at websites like Facebook, Netflix and Airbnb, you're looking at UIs built in React. React provides us with a set of tools and structure for building out these user interfaces and makes this process much faster and easier. Single Page Applications With React, it's very common to build out single page applications. So before we get into the React concepts, I want to quickly recap single page applications and how they work for anybody that's not familiar with this concept yet. So in traditional websites we have a template for each page on our website and return that template back to the user whenever they request it. With single page applications, however, we are working with one single template and are simply updating all the components within the dom. Personally, I think the term single page application is a bit misleading as it makes me think there is only one page in our application when really we're just using one single template and modifying all the contents within it. Components are what make up the visual layer of our application and let us split up our UI into independent reusable pieces, while how you build and structure your application is completely up to you. Traditionally each part of our UI would be built out separately as its own component and then added into a parent component making up the UI for a specific page. A Component is a JavaScript class or function that returns back some HTML. Well, this is actually something called JSX, but more on that in a second. One thing to note about components is that they can be nested as deep as you want. A component can hold another component and that component can hold more components. While I do think you should learn both class based components and function based components at some point, with the addition of React hooks, the trend is shifting more towards

using functional components. So if you're trying to decide which to learn first, you can probably start with functional components. JSX Instead of writing traditional HTML tags, we're going to be using something called JSX which stands for JavaScript XML. JSX actually looks a lot like HTML with some slight syntax differences and also gives us some added functionality. Take a look at this example and you'll see how you can use the curly braces to pass in variables and adding JavaScript logic directly into your HTML. JSX tags are actually very similar to HTML tags. Some notable differences are things like class declarations which are written as class name and how event handlers are added. Browsers can't actually read jsx, so this code will actually be run through a compiler and converted into traditional HTML and JavaScript code once it's output in the browser. React Router Using a React router is how we can have multiple pages in a single page application. With React, we typically handle URL routing with something called a router that keeps our UI in sync with a URL. Because we're not actually changing pages, the router will take care of rendering out components into the DOM based on the current URL. Props when you need to pass data down from one component to another, you can pass it down as a prop. A prop can be passed down like a function parameter. Once a prop is passed down into a component, you can now use that prop anywhere in the child. Component props can be passed down multiple layers if needed. The term for this is called prop drilling. Prop drilling can get messy, so we'll talk about some solutions to this in a minute. State State is simply a JavaScript object used to represent information in or about a particular component. Traditionally we use class based components to set our state and its values, but more modernly we use React hooks like Use State to create a component state. So let's imagine for a second that we have a list of notes that we want to render out in our app. We can set an initial state and then map through that state and output all that data in our component. We can also update our state. In this example, we can set our initial state as an empty array. Then we request some data from our API and update that state with new data. This state update will trigger our component lifecycle effect, which we'll talk about next. The Component Lifecycle Understanding the component lifecycle is a must for every React developer and is probably one of the most common

interview question for junior developers. A React component has a lifecycle that it goes through, and there are three main phases that we need to know about in this life cycle. Each component has a mounting phase for when it's first being added to the dom, an updating phase for when we are modifying something and that component needs to update and an unmounting phase for when this component will be removed from the dom. With class components we have these three methods. To take care of these lifecycle methods, we have Component did mount, Component did update, and and Component will unmount. With functional components, however, we have a method called UseEffect that allows us to work with each part of a component lifecycle. React Hooks React hooks only apply to functional components, but due to the popularity of using function based components, hooks have become essential to learn in this process. Hooks let us add state and other features without using class based components. Before hooks, functional components could not hold any state. Hooks are simply functions that allow us to hook into and manage state. The two most common hooks that you'll probably use when you first start are going to be use State, which lets us set and update our state in a component, and useeffect, that is simply a function that allows us to manage our component lifecycle. React gives us a whole list of built in hooks along with the ability to create our own custom hooks. State Management While we can create and manage state inside of our components, there will likely be a time when we need to create some form of global state to make data available across multiple components. Think of something like holding data for a logged in user. You may need this user across multiple components like your header bar or a profile component, and passing this data down through props may not be practical, especially when this information is updated somewhere inside of those components. We have several options to handle this, like using the built in Context API or using a third party package like Redux and many others out there. With these we are able to create some form of global state and use it across multiple components in our component tree without having to deal with prop Drilling the virtual dom. At some point in the process of learning React, you will want to have an understanding of how the virtual DOM works. Understanding this will help you understand and make sense of how React builds and

updates our DOM and the complete lifecycle of a React component. In short, React creates something called a virtual dom, which is a virtual representation of the real dom. When we're updating our components, we're actually updating the virtual DOM and not the real dom. Using this method, React can find the most efficient way to update the real DOM by updating only areas where changes have been made without having to update the entire dom. The Key Prop when it comes to rendering out a list of data in your components, there is one thing you should be aware of and that is the key proposal. Each item in a dynamically rendered list should contain the key prop or else you'll get this annoying error in the console. This prop should be unique and helps React identify which items have been changed, added or removed. So React knows which part of the virtual DOM to update. Event Listeners Handling events with React is very similar to how we would do this in traditional JavaScript with a few differences. In React, we camel case event names and we pass in the function we want to call directly in line between two curly braces. So so there is no need for methods like AddEventListener because our JavaScript code and HTML are mixed together. Handling Forms with React how we handle forms is a little bit different from the traditional method because we are trying to keep all our information in some form of state inside of our component. HTML elements such as Input, text, area and select typically maintain their own state and update based on a user's input. With React, however, we typically add in event listeners to each field and update our component state whenever any one of these inputs change change. So methods like OnChange and OnSubmit would directly update our state and would be controlled by our own functions instead of letting the form handle all of this on its own. Conditional Rendering There is always a chance that you will need to render out some content conditionally depending on other values inside of your application. Think of something like a user's name inside of a navigation bar. Depending on the user's authentication status, you will either display the user's name or display nothing. One way we can go about handling this is using the logical and operator. We can also use the inline if else conditional operator if we want to add in some extra logic. In both of these examples, the rendered output will depend on the conditions we provide. Common Commands

There are three main commands I want to mention here because these are commands you will use in every project. We have the Create React App command which creates the boilerplate files for a React application application. We have the Start command that starts up our development server so we can view our project right away, and we have the Run Build command that builds a directory for a production build of our app for deployment. All right, so that's my list of core concepts every React developer should master. Don't forget to subscribe if you enjoyed this video, and make sure to check out the React crash course linked in the video description if you want to learn more.