XUnit, NUnit - learn these to add to CV.

**Class Components**

Funksiyani class componente bind etmek

this.handleIncreaseCounterClick = this.handleIncreaseCounterClick.bind(this);

In React, you can create components using two primary syntaxes: class components and functional components (also known as function components). Both types of components serve the same fundamental purpose, which is to define reusable UI elements, but they have differences and similarities:

**Differences:**

**Syntax:**

*Class Components*: Class components are defined as ES6 classes that extend the React.Component class. They use a render method to define the component's output.

*Functional Components*: Functional components are regular JavaScript functions that take in props as their argument and return JSX to describe the UI.

**State:**

*Class Components*: Class components can have state using the state property. You can use this.setState to update the state.

*Functional Components*: Functional components didn't traditionally have state before React 16.8. With the introduction of hooks (like useState), you can now use state in functional components.

**Lifecycle Methods:**

*Class Components*: Class components have access to lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount. These methods allow you to perform actions at different stages of a component's life.

*Functional Components*: Functional components can achieve similar functionality using React hooks like useEffect.

**Similarities:**

**Props:**

Both class and functional components can accept and use props (short for properties) to receive data from their parent component. Props are read-only and help in passing data from parent to child components.

**Render Output:**

Both types of components are responsible for rendering the UI. They return JSX that describes what should be displayed on the screen.

**Reusability:**

Both class and functional components are reusable. You can use them to build complex UIs by composing smaller components together.

**Context API:**

Both types of components can access and consume data from the React Context API, which allows you to share data across your component tree without having to pass props explicitly.

**Event Handling:**

Both types of components can handle events (e.g., click events) and respond to user interactions.

**Children:**

Both types can have children components nested within them and can render these children components.

**Considerations:**

Functional components with hooks have become the preferred way to write components in React due to their simplicity and the ability to use state and lifecycle features. However, class components are still widely used in existing codebases and can be useful in certain situations, especially when dealing with legacy code.

If you are starting a new React project or working on a codebase using React 16.8 or later, it's recommended to use functional components with hooks for their simplicity and the direction in which React development is heading.

**Class Componented Consturctor yazmagin 2 meqsedi var:**

1. State leri Initialize etmek ucun.
2. Functionalari Class Componente bind etmek ucun.

**Class Component Hooks**

**componentDidCatch()**

The componentDidCatch() method is invoked if some error occurs during the rendering phase of any lifecycle methods or any children component

**componentDidMount()**

The componentDidMount() method allows us to execute the React code when the component is already placed in the DOM (Document Object Model).

**componentWillUnmount()**

componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().

**componentdidupdate()**

componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.

Use this as an opportunity to operate on the DOM when the component has been updated. This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed).

Example : [codesandbox.io/s/tmtv63?file=/App.js&utm\_medium=sandpack](https://codesandbox.io/s/tmtv63?file=/App.js&utm_medium=sandpack)