**static getDerivedStateFromProps(nextProps, prevState)**

Yeni metod getDerivedStateFromProps static bir fonksiyondur ve React 16.3 ile birliktecomponentWillReceiveProps yerine kullanılmaktadır.

Bir bileşen başlatıldıktan sonra ve yeni bir prop aldığında çağrılır. this anahtarına erişimi yoktur, bunun yerine sizden bir nesne döndürmeniz beklenir. Eğer değerlerinizde bir değişiklik yok ise null ya da {} döndürmelisiniz.

Kullanılan bileşende her defasında yeniden çağrılır. Eğer yalnızca güncellenen değerleri kontrol etmek istiyorsanız önceki ve sonraki değerlerin koşul kontrolleri bu noktada çok önemlidir.

Metodun static olması bize şu şekilde bir soru doğuruyor olabilir: this.setState nasıl kullanabiliriz? Kullanmıyoruz. Bunun yerine fonksiyon, state verilerini direkt olarak güncellemektedir.

**ComponentDidMount =>**

Sayfa yuklenir yuklenmez calisir.

Constructor > getDerivedStateFromProps > render > ComponentDidMount

Sayfa acilisinda birseyler yuklemek istersek bu Lifecycle metodu kullanabiliriz.

**ComponentWillMount =>**

Komponentin Real Dom’a aktarılmadan önceki andır. Yani render işleminden hemen önceki tetiklenen Event’tir. Aynı constructor gibi özel tanımlı bir fonksiyondur.

Constructor > componentWillMount > getDerivedStateFromProps > render > ComponentDidMount

Not \*=\* Render fonksiyonu daha sonra çalışacağı için SetState fonksiyonu çalışmaz.

**ComponentWillUnmount**

Dom üzerinden kaldırılması işlemi diyebiliriz kısaca. Yani bütün işlemler bittikten sonra bileşen dom üzerinden kaldırılmadan önce çalışacak Event’ımızdır.

React’a ufak ufak ilgilenen biri olarak az çok öğrendiklerimden hazırladığım küçük bir makale hazırlamak istedim. Yanlışım varsa affola. Faydalı olması dileğiyle.

**ShouldComponentUpdate(nextProps, nextState) =>**

Props veya state değiştiği anda biz komponentimizin baştan render edilip edilmemesi gerektiğini manual olarak belirleyebiliyoruz. Bu event’ımız sadece boolen değer döndürür. True ise komponent baştan render edilir false ise render edilmez. Varsayılan olarak True değer döndürür. İki tane parametre alır bunlar nextProps ve nextState, bunlarda güncellenmiş olan verileri bize veriyor.

 //Eger burasi false donerse componentlarin renderlanmasina izin verilmeyecektir.

  shouldComponentUpdate(nextProps, nextState) {

    console.log("[App.js] shouldComponentUpdate");

    return true;

  }

**ComponentDidMount**

Komponent Dom’a render edildikten sonra çalışan Event’ımızdır. Yani render işleminden sonra çalışır.

Yani artık sırasıyla  
Constructor>ComponentWillMount>Render>ComponentDidMount

Not \*=\* ComponentDidMount’ta setState işlemi yapılırsa tekrardan Render işlemi yapılır.

Not \*=\* Eğer komponentinizin bir child komponenti varsa ve aynı eventlara sahipse çalışma sıraları şu şekilde olacaktır.  
Construtor>ComponentWillMount>Render>Child Constructor> Child ComponentWillMount>Child Render>Child ComponentDidMount>ComponentDidMount

Yani buradan şunu anlamamız gerekiyor arkadaşlar. Ebeveyn bir komponentin içinde child bir komponent varsa ebeveyn komponentimizin didMount olayından önce Child komponentimizin aksiyonları sırasıyla bildiğimiz sırayla işlenecektir.

**ComponentDidUpdate**

Bu event’ımızda benzer mantıkta yine iki parametre alıyor. ShouldComponentUpdate den true döndükten sonra ComponentWillUpdate çalışır ve Render işlemi çalışıyor. Bu işlemdeki Render işleminden sonra çalışacak event’ımız ComponenDidUpdate olacaktır.

getDerivedStateFromProps > App.js render > shouldComponentUpdate > Components render > getSnapshotBeforeUpdate > componentDidUpdate

**componentWillUpdate** => [LEGACY] Eski projelerde kullaniliyor yenioolerde artik tercih edilmiyor.

**FUNCTIONAL LIFECYCLE METHODS // REACT HOOKS**

**useEffect**

  //Bu sekilde kullanildiginda her renderda calisir.

  useEffect(() => {

    console.log("[Cockpit.js] useEffect");

// Burada http istekleride yapabiliriz.

  });

İstedigimzi kadar useEffect alt alta kullanabiliriz.

  //,useEffect fonksiyonu bitiminde , koyup [] icine bir kosul tanimlarsak sadece o deger degistiginde yani update oldugunda eseEffectimiz calisir. Tipki componentDidUpdate gibi.

  useEffect(() => {

    console.log("[Cockpit.js] useEffect");

    setTimeout(() => {

      alert("Saved data to cloud");

    }, 1000);

  }, [props.persons]);

Eger asagidaki gibi [] bu alani bos birakirsak bu sefer useEffect sadece componentDidMount gibi sadece ilk seferde render isleminden sonra site ilk acildiginda calisir.

  useEffect(() => {

    console.log("[Cockpit.js] useEffect");

    setTimeout(() => {

      alert("Saved data to cloud");

    }, 1000);

  }, []);

Eger icinde asagidaki gibi bir return degeri donderirsek, componentWillUnmount gibi component kapandiktan sonra calistiracagi islemleri belirlemis oluruz.

  useEffect(() => {

    return () => {

      console.log("[Cockpit.js] cleanup work in useEffect");

    };

  }, []);

[] bu second argument onemli eger asagidaki gibi bu olmazsa her update ve islemde useEffect calisir. Once return calisir daha sonra console.log yani useEffectin icindekiler calisir.

  useEffect(() => {

console.log("[Cockpit.js] useEffect")

    return () => {

      console.log("[Cockpit.js] cleanup work in useEffect");

    };

  });

**OPTIMIZATION**

**With React.memo**

export default React.memo(cockpit);

Bu sekilde kullandigimizda React.memo ile herhangi bir update ve degisiklik olmaksizin componentimizi tekrar tekrar render etmeyecek ve stored versiyonunu kullanacaktir bir degisiklik olmadiginda.

**With shouldComponentUpdate**

  shouldComponentUpdate(nextProps, nextState) {

    console.log("[Persons.js] shouldComponentUpdate");

    if (

      nextProps.persons !== this.props.persons ||

      nextProps.changed !== this.props.changed ||

      nextProps.click !== this.props.click

    ) {

      return true;

    }

    return false;

  }

* Parent componentta bir degisiklik oldugunda ve bu komponentimizida etkiliyorsa ve deigisiklik yapilmasi gerekiyorsa bu komponentimizdada o zaman shouldComponentUpdate veya react.memo kullanmaliyiz.
* Yani bir Cockpit.js dosyamiz olsun bu componentimiz degistiginde Persons.js de bir degisiklik olmadigini dusunun ama olmadigi halde render ediliyor Cockpitteki her deigisiklikte, iste bu zamanlarda React.mem yada shouldComponentUpdate kullanmaliyiz. Diger durumlarda luzumsuz olur ve sistemimizi fazladan is yaparak yavaslatir.

**PURE COMPONENT**

import React, { PureComponent } from "react";

class Persons extends PureComponent {

Bu sekildede yukaridaki shouldComponentUpdate ‘in yapacaklarini yapabiliriz. Cunku PureComponent lar shouldComponent’a bizim tanimlayabilecegimiz tum degisiklik kontrollerini otomatik olarak icinde tutar ve herseyi otomatik yaparlar. PureComponentlar bu yuzden ise yararlidir.