**1. Virtual DOM Nedir ve Nasıl Çalışır?**

Virtual DOM (Sanal DOM), React'in performans optimizasyonunu sağlamak için kullandığı bir teknolojidir. Virtual DOM(**Document Object Model)**, gerçek DOM'un hafifletilmiş bir kopyasıdır. React, UI değişikliklerini önce Virtual DOM'da yapar ve ardından bu değişiklikleri gerçek DOM'a yansıtır.

**Nasıl Çalışır:**

* **Değişikliklerin Yapılması:** Kullanıcı etkileşimleri veya veri değişiklikleri olduğunda, React bileşenlerdeki değişiklikleri önce Virtual DOM'a uygular.
* **Farkların Hesaplanması:** React, eski Virtual DOM ile yeni Virtual DOM arasındaki farkları hesaplar. Bu süreç "Reconciliation" olarak adlandırılır.
* **Güncellemelerin Yapılması:** Sadece bu farklar (yani değişiklikler) gerçek DOM'a uygulanır. Bu sayede gereksiz güncellemelerden kaçınılır ve performans iyileştirilir.

**2. Real DOM ile Farkları Nelerdir?**

* **Performans:** Real DOM'da her değişiklikte tüm DOM yapısı yeniden render edilir, bu da performans sorunlarına yol açabilir. Virtual DOM ise sadece değişiklik yapılan kısmı günceller, bu nedenle daha hızlıdır.
* **Güncellemeler:** Real DOM'da her değişiklikte doğrudan gerçek DOM güncellenir. Virtual DOM'da ise değişiklikler önce Virtual DOM'da yapılır ve ardından minimum güncellemeyle gerçek DOM'a yansıtılır.
* **Hafıza Kullanımı:** Virtual DOM ek bir hafıza gerektirir çünkü gerçek DOM'un bir kopyasını tutar. Ancak, bu ek hafıza kullanımı performans iyileştirmesi sağlar.

**3. Props Nedir, Nasıl Kullanılır?**

**Props** (Properties), React bileşenlerine dışarıdan veri iletmek için kullanılan bir mekanizmadır. Props, bileşenlerin dinamik ve yeniden kullanılabilir olmasını sağlar.

**Nasıl Kullanılır:**

* **Tanımlama:** Bir bileşende, props özelliği bir nesne olarak tanımlanır. Örneğin:

const MyComponent = (props) => {

return <h1>Hello, {props.name}</h1>;

};

* **Kullanma:** Props, bileşenlerin kullanıldığı yerde atanır:

<MyComponent name="John" />

Burada, name prop'u John olarak atanır ve MyComponent içinde kullanılabilir.

**4. Componentler Arası Veri Geçişi Nasıl Yapılır?**

Veri, genellikle üst bileşenden alt bileşene props aracılığıyla geçilir. Ayrıca, aşağıdaki yöntemlerle de veri paylaşabilirsiniz:

* **Props:** Üst bileşenden alt bileşene veri geçişi.
* **Callback Fonksiyonları:** Alt bileşenden üst bileşene veri göndermek için callback fonksiyonları kullanabilirsiniz.

// Üst Bileşen

const ParentComponent = () => {

const handleData = (data) => {

console.log(data);

};

return <ChildComponent onData={handleData} />;

};

// Alt Bileşen

const ChildComponent = ({ onData }) => {

const sendData = () => {

onData("Hello from child");

};

return <button onClick={sendData}>Send Data</button>;

};

* **State Yönetim Kütüphaneleri:** Daha karmaşık uygulamalar için Redux, Context API gibi state yönetim kütüphaneleri kullanılabilir.

**5. Props'un Component Lifecycle’ına Etkisi Nedir?**

Props, bileşenlerin yaşam döngüsü boyunca çeşitli aşamalarda etkili olabilir:

* **Mounting:** Bileşen ilk kez oluşturulduğunda props bileşene atanır.
* **Updating:** Props değiştiğinde, bileşen componentDidUpdate (veya useEffect ile) bu değişiklikleri yakalar ve gerekli güncellemeleri yapar.
* **Unmounting:** Bileşen DOM'dan kaldırıldığında props kullanılarak temizlik işlemleri yapılabilir.

Props'un bileşen lifecycle'ı üzerindeki etkisi, bileşenlerin dinamik olarak güncellenmesini ve kullanıcı etkileşimlerine göre doğru şekilde render edilmesini sağlar.