

React Nedir?

Facebook developerları tarafından geliştirilmiş,
Declarative bir UI (kullanıcı arayüzü) kütüphanesidir.

1. Componentlarla çalışmayı kolaylaştırır
2. DOM işlemlerini daha performanslı yapmayı sağlar
3. Kodların yönetilebilirliğini arttırmır.

Imperative - Declarative kodlar

Imperative - **NASIL** yapılacak?

```
<script>
  const app = document.getElementById('app');

  function addQuote() {
    const post = document.createElement('div');
    post.classList.add('post');

    const clapContainer = document.createElement('div');
    clapContainer.classList.add('post-clap');

    const quote = document.createElement('p');
    quote.classList.add('post-p');
    quote.textContent =
      "Yalnız tek bir şeye ihtiyacımız vardır, çalışan olmak. Servet ve onun
      tabii neticesi olan refah ve saadet yalnız ve ancak çalışanların
      hakkıdır.";

    const clapCount = document.createElement('span');
    clapCount.classList.add('post-clapCount');
    clapCount.textContent = 10;

    const clapButton = document.createElement('button');
    clapButton.classList.add('post-clapButton');
    clapButton.textContent = 'Alkışla';

    clapButton.addEventListener('click', () => {
      clapCount.textContent = Number(clapCount.textContent) + 1;
    });

    clapContainer.appendChild(clapCount);
    clapContainer.appendChild(clapButton);
    post.appendChild(quote);
    post.appendChild(clapContainer);

    return post;
  }

  app.appendChild(addQuote());
```

Declarative - **NE** yapılacak?

```
const post = (
  <div class="post">
    <p class="post-p">
      "Yalnız tek bir şeye ihtiyacımız vardır, çalışan olmak. Servet ve onun
      tabii neticesi olan refah ve saadet yalnız ve ancak çalışanların
      hakkıdır."
    </p>
    <div class="post-clap">
      <span class="post-clapCount">10</span>
      <button class="post-clapButton">Alkışla</button>
    </div>
  </div>
)
```

React'ın kullandığı temel teknolojiler

1. JSX formatı

- HTML, JS ve CSS'in aynı dosyada yazılabildiği bir dosya formatı

2. Virtual DOM (Sanal DOM)

- Sayfadaki etkileşimleri hızlı ve performanslı yapmak için bir yöntem

Component Nedir? jsx return eden fonksiyon

```
const title = <h1>Hi, React!</h1>;  
const container = document.getElementById('app');  
const root = ReactDOM.createRoot(container);  
root.render(title);
```

- İlk harfi büyük yazılır.
- HTML tagleri gibi <...> arasına yazılır.

```
function Title() {  
  return <h1>Hi, React!</h1>;  
}  
  
const container = document.getElementById("app");  
const root = ReactDOM.createRoot(container);  
root.render(<Title />);
```

React Component anatomisi

Componentlar

- **JSX** return eden fonksiyonlardır
- Kendi fonksiyonları olabilir
- React'ın özel fonksiyonlarını kullanabilir (**hooks**)

used libraries

```
import { useState } from 'react';
import { sculptureList } from './data.js';
```

```
export default function Gallery() {
  const [index, setIndex] = useState(0);
```

```
  const [showMore, setShowMore] = useState(false);
```

```
  function handleNextClick() {
    setIndex(index + 1);
  }
```

```
  function handleMoreClick() {
    setShowMore(!showMore);
  }
```

```
  let sculpture = sculptureList[index];
  return (
    <>
```

```
    <button onClick={handleNextClick}>
      Next
    </button>
```

```
    <h2>
      <i>{sculpture.name}</i>
      by {sculpture.artist}
    </h2>
```

```
    <h3>
      ({index + 1} of {sculptureList.length})
    </h3>
```

```
    <button onClick={handleMoreClick}>
      {showMore ? 'Hide' : 'Show'} details
    </button>
```

```
    {showMore && <p>{sculpture.description}</p>}
```

```
    <img
      src={sculpture.url}
      alt={sculpture.alt}
    />
  </>
);
```

```
}
```

hooks

functions

returned JSX

component (function)

JSX kuralları

1. Tüm **taglerin kapanma** / işaretleri eklenmeli
(Özellikle img ve br gibi taglere dikkat!)

2. **class** yerine **className** yazılmalı (çünkü class kelimesinin JS için başka bir anlamı var)
3. 2 kelimededen oluşan attributelar, **camelCase** yazılmalı
4. Tüm jsx kodları **tek bir parenta** sahip olmalı:

JSX içinde JS kullanmak

- `{ ... }` ile return edilen kodun içinde JS dünyasına bir pencere açıp JS yazılabilir
- Fakat **return** içinde olduğundan, kısa ve return eden şeyler yazılabilir
 - Ternary if
 - `&&`, `||` gibi operatörler

```
function Title() {  
  const greet = true;  
  const text = "Hi, React!";  
  
  return <h1>{greet && text}</h1>;  
}
```

```
function Title() {  
  const language = "tr";  
  const textEn = "Hi, React!";  
  const textTr = "Selam, React!";  
  
  return <h1>{language === "tr" ? textTr : textEn}</h1>;  
}
```

React hooks nedir?

Bir component'in state'ine ve hayat döngüsüne (lifecycle) bağlı işlem yapmaya yarayan metodlardır.

- eventListener'a benzer - event değil, state ve lifecycle dinliyor
- En çok kullanacağımız: **useState**, **useEffect**
- Kütüphaneler kendi hooklarını yazar, biz de onları kullanırız.
- Kendi hooklarını da yazabiliriz!

State kavramı ve useState hook'u

STATE: Bir componentin hafızasıdır.

- Değiştiğinde, **component fonksiyonu tekrar çalıştırılır**, bu esnada state kullanılan ve state'e bağlı yerler değişir, diğer yerler sabit kalır.
- **Özel bir fonksiyonla değiştirilir (setter)**

```
const [begeni, setBegeni] = React.useState(10);
```

state değişkeni state değiştiren
fonksiyon (setter) başlangıç değeri

Ekstra: Click eventi ile çalışmak

- **onClick={calisacakFonksiyon}**
- **onClick={event => calisacakFonksiyon(event)}**
- **onClick** gibi farklı eventler de kullanıcak: **onChange**, **onSubmit** vb.

JS Modules

2016'da **JavaScript'e eklenen bir özellik!**

1. Dosyaları bölmek ve birbiri içinde kullanmayı,
2. Dosyalar arası bir scope oluşturmayı,
3. Kütüphaneleri, sadece ihtiyaç duyulan dosyalar içinde kullanmayı sağlar.

Module'leri export ve import etmek

Export: Dışarı aktarmak, **Import:** İçeri aktarmak

Her şey export/import edilebilir:

- Değişkenler(array, object, boolean,...)
- Fonksiyonlar
- **Componentlar**

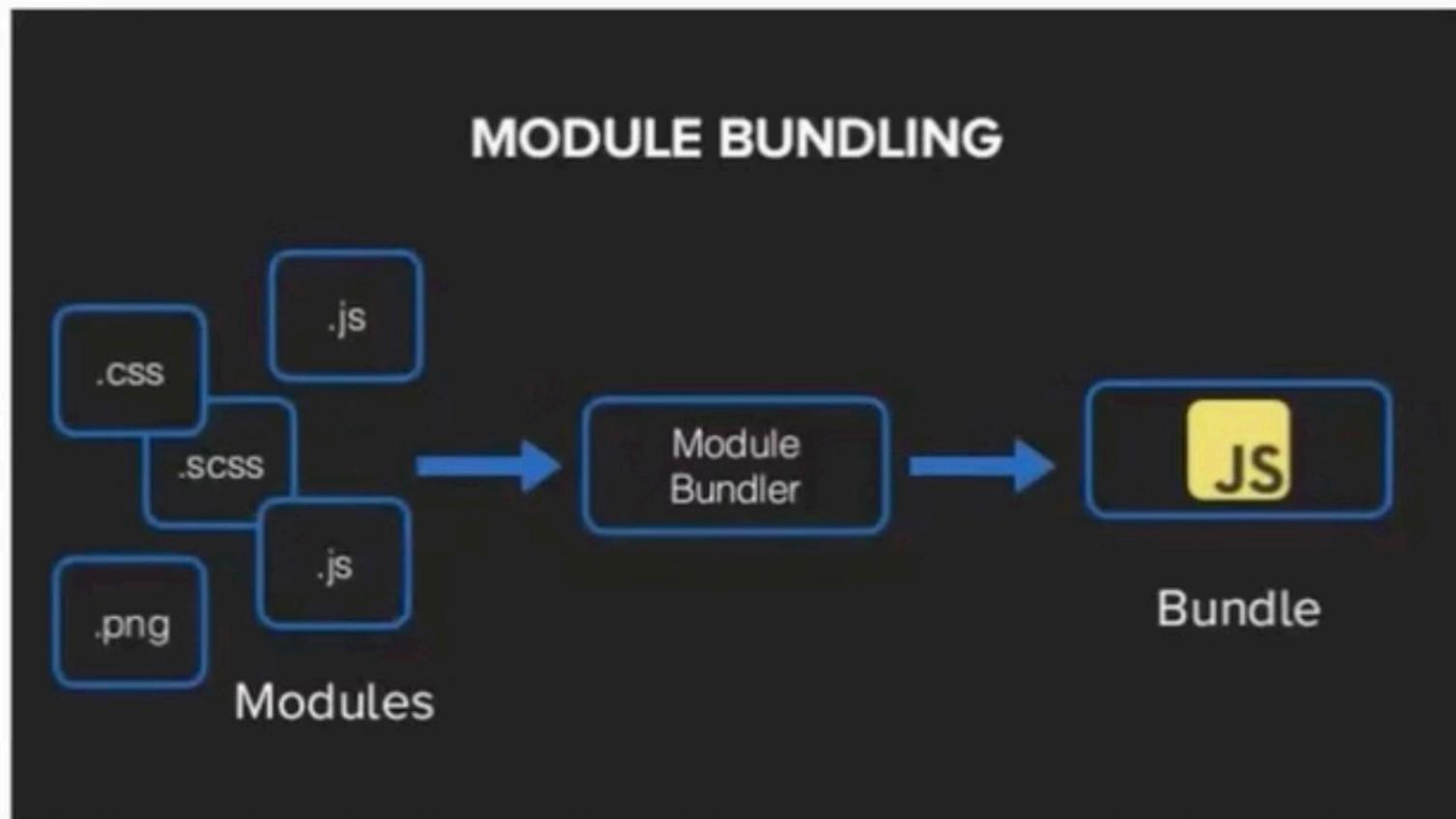
Default export 1 defa, **named export** sınırsız kullanılabilir.

Syntax	Export statement	Import statement
Default	<code>export default function Button() {}</code>	<code>import Button from './Button.js';</code>
Named	<code>export function Button() {}</code>	<code>import { Button } from './Button.js';</code>

Module Bundlers

Bir uygulamanın tüm JavaScript dosyalarını ve bağımlılıklarını tek bir dosyada birleştiren araçlar.

Örn: [Webpack](#), [Rollup](#), [Parcel](#), [Vite](#)



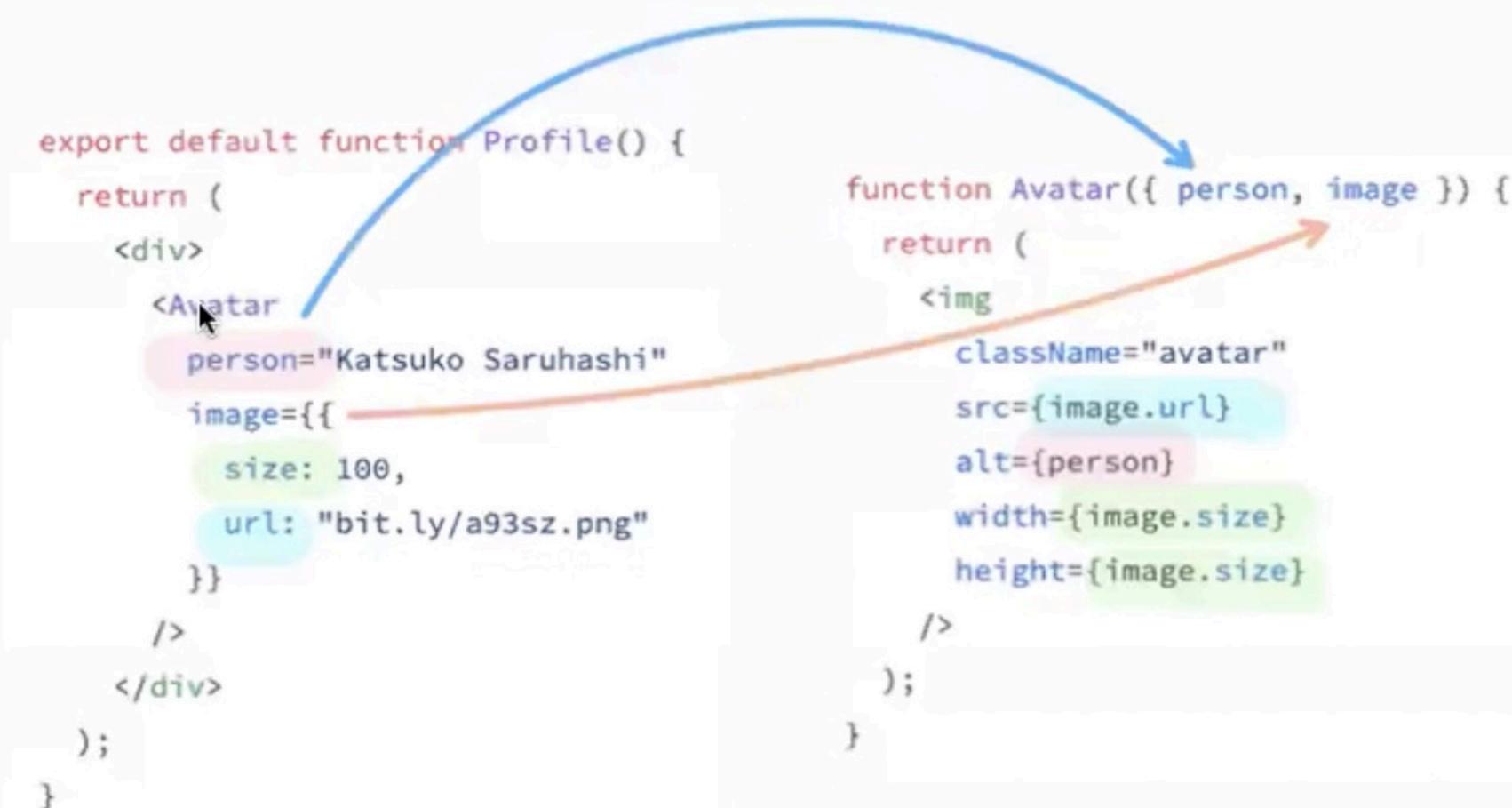
1. Module bundler'lar kodu optimize eder, gereksiz kodları kaldırır (**tree shaking**), kodu küçültür (**minification**) ve böylece uygulamanın genel boyutunu azaltır.
2. Bu işlem tarayıcının yüklemesi gereken dosya sayısını azaltır. Bu da web uygulamalarının yüklenme süresini azaltır ve performansı artırır.
3. ES6 modülleri gibi modern JavaScript özelliklerini daha geniş bir tarayıcı yelpazesinde uyumlu hale getirir.

Props Kavramı

React Component'ları, birbirleriyle iletişim kurmak için **props** kullanır.

1. Her parent component => child component'lara props vererek onlara bilgi aktarabilir.
2. Prop'lar child componentlar içinde sadece okunabilir, **child'da değiştirilemez!**
3. Props ile her türlü JavaScript değerini -objeleri, dizileri ve fonksiyonları, aktarabiliriz.

Prop Gönderme ve Props Destructuring



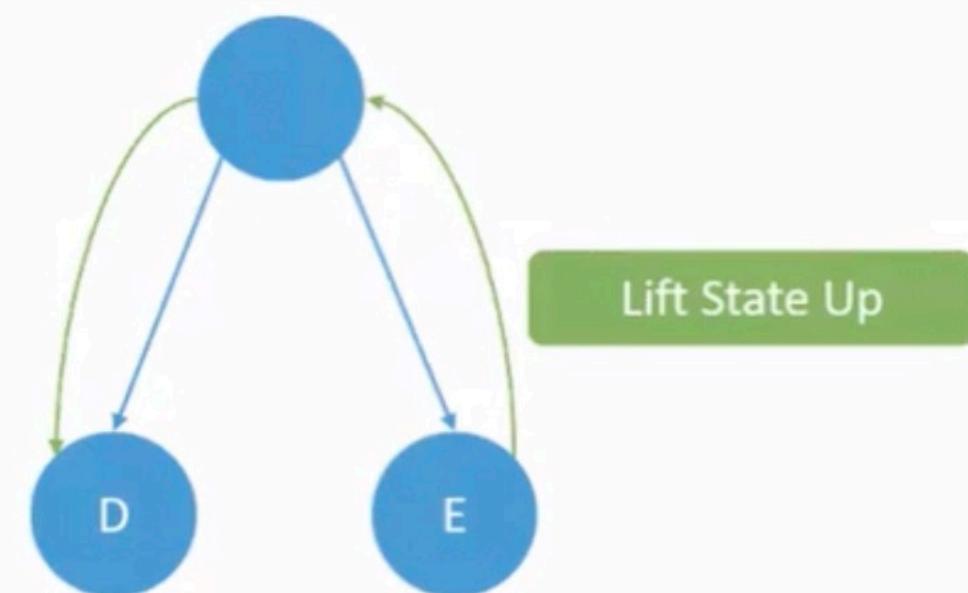
Destructure, Javascript özelliğidir.
props.person yazmanın kısa yoludur.

```
function Avatar(props) {
  let person = props.person;
  let size = props.size;
  // ...
}
```

1. Proplar parent'dan child'a html attribute'u gibi gönderilir.
2. Gönderilen tüm proplar tek bir objenin içinde key&value çifti olarak yer alır.
3. **children** propu Component tagleri arasında kalan kısımdır
 - `<Title>Selam Dünya!</Title>` children prop'u “Selam Dünya!”

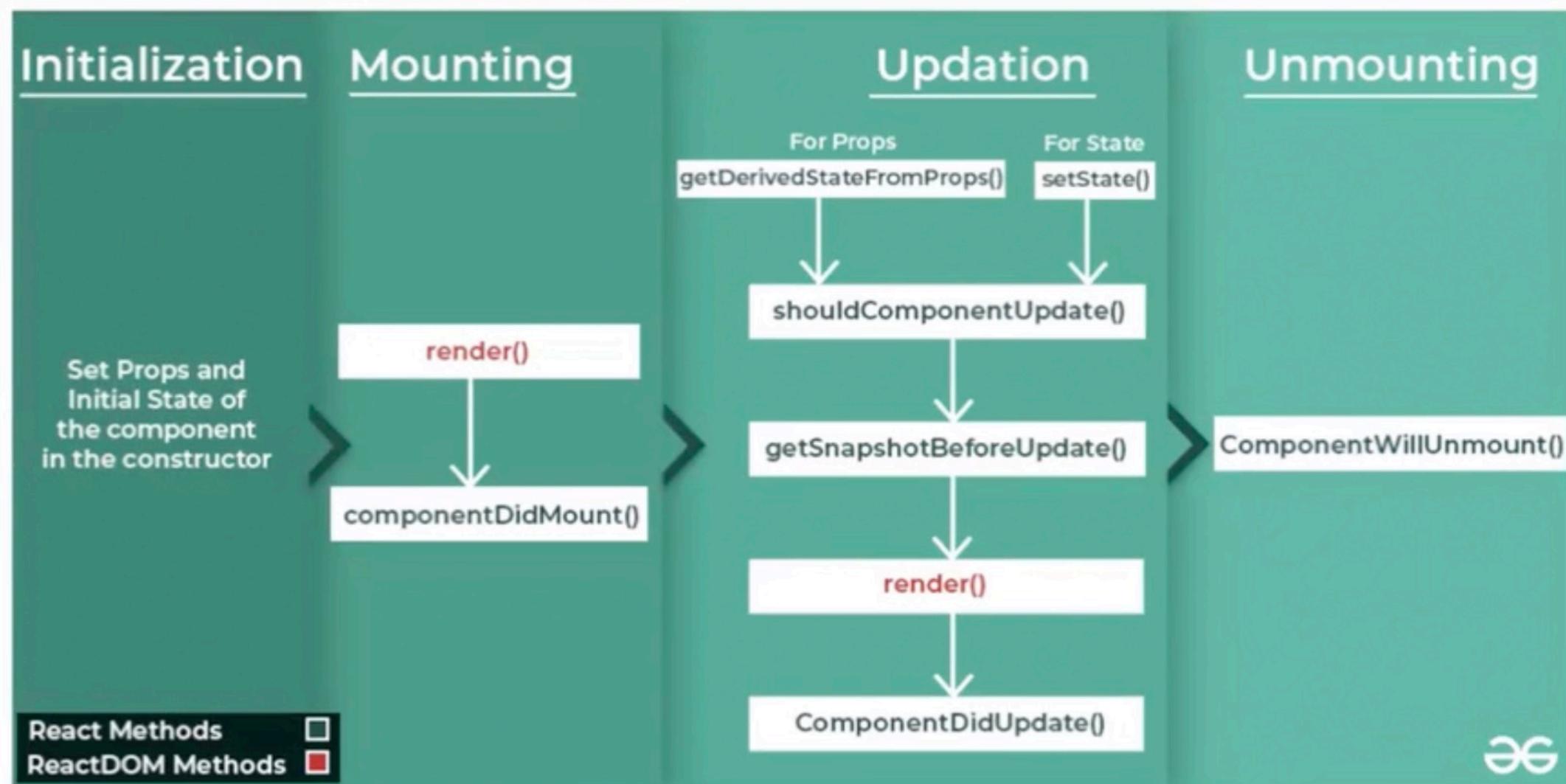
Veri akışı ve yönü

1. React uygulamalarında (ve genelde) **veri parent component'tan child component'a doğru akar**, yani bu yönde paylaşılır.
2. Child'larda olan verileri Parent'a taşımaya **State Kaldırmak** (lifting state up) denir.
3. **Parent'daki veriyi değiştirmek** için;
 - veriyi değiştirecek fonksiyon yazıp,
 - prop olarak child'a iletiriz.
 - child'da bu fonksiyonu kullanarak parent'daki veriyi değiştiririz.



Component Lifecycle

Projedeki component'lar, kullanım yerine göre sayfada gösterilir ya da çıkarılırlar.



Mount: Component'ın ekranda görüntülenmesi

Update: Component'ın güncellenmesi

Unmount: Component'ın ekrandan çıkarılması

Component lifecycle

- Herhangi bir **state değişir ise** component fonksiyonu baştan(**tekrar**) çalışır.
- Gereksiz state değişikliklerinden kaçınmak iyi bir pratiktir.

useEffect Hook

Component lifecycle dinleyen bir hook'tur.

Axios ile yaptığımız API isteklerini genelde burada yaparız.

```
useEffect(() => {}, [])
```

arrow function

Dependency array

- Dinlemek istediğimiz değerler
- Opsiyonel!

useEffect Hook

3 farklı kullanımı vardır.

```
useEffect(() => {
  console.log("Sayfa yükleniğinde, bir kere çalışır.");
}, []);
```

mount

```
useEffect(() => {
  console.log("Sayaç değişti.");
}, [sayac]);
```

update

```
useEffect(() => {
  console.log("Her değişiklikte çalışır.");
});
```

```
useEffect(() => {
  return () => {
    console.log("Sayfa kaldırılırken çalıştı.");
  }
}, []);
```

unmount

Array/Object türündeki stateler

- Listeleri tutmak için **array** kullanabiliriz.
- Form elemanları gibi key&value çiftlerini tutmak için **object** kullanabiliriz.
- State'i güncellemek için setter'a yeni bir array/object veririz => **spread operatörü**

```
const [movies, setMovies] = useState(movilerListesi);

const yeniListe = [...movies, yeniMovie]
setMovies(yeniListe);
```

```
const [movies, setMovies] = useState(movilerListesi);

setMovies( [...movies, yeniMovie] );
```

- **.map()** ve **.filter()** metodlarını kullanabiliriz.
- State güncelleme örnekleri: <https://state-updates.vercel.app/#State%20Updates>

Ekstra: .filter(...) ile listeyi güncellemek

```
const [movies, setMovies] = useState(movilerListesi);

const yeniState = movies.filter((movie) => movie.id != id);
setMovies(yeniState);
```

- Bir arrayin tüm elemanlarında verilen koşula uygun elamanları filtreler
- Sonuçları içeren **yeni bir array** return eder
- Array'deki sırayı bozmadır

Ekstra: .map(...) metodu ile bir listeyi ekranada göstermek

```
<div className="pt-3 text-sm">
  {
    favoriteMovies.map((movie) => (
      <Movie key={movie.id} movie={movie} />
    ))
  }
</div>
```

- Bir arrayin tüm elemanlarına parametre olarak aldığı işlemi uygular
- Sonuçları içeren **yeni bir array** return eder
- Array'deki sırayı bozmadır
- **key:** map kullanırken elemanlara verilmeli, unique olmalıdır. Hatasız renderlar için gerekli.

JSX içinde CSS kullanmak

1. CSS dosyası import ederek

2. inline style olarak

- stiller string olarak değil bir style objesi olarak verilir
- CSS propertyleri camelCase yazılır: `backgroundColor`
- String değerler tırnak içinde yazılır: `fontSize: "21px"`
- Property-value çiftleri virgül ile ayrılır.
- JS code da yazabiliriz.

```
<div style={{  
    fontSize: "21px",  
    color: isImportant ? "red": "blue",  
    textAlign: "center",  
    borderRadius: "4px"  
}}>
```

JS'in Çözemediği CSS Durumları

1. CSS içinde bir scope kavramı yok

- `p { color: red; }` Tüm paragrafları kırmızı yapar
- **Çözüm:** Scope'a benzer bir yapı combined selector kullanmak
 - `section header nav p { color: blue }`
 - Bu yapı hiyerarşije çok bağlı ve bu yüzden kırılgan

2. Bir elemanı stillerin geri kalanından izole etmek sadece CSS kullanarak zor!

- **Çözüm:** CSS-in-JS kütüphaneleri: emotion, JSS, [styled-components](#)

<paint> Styled-Components <https://styled-components.com/>

- Elemanlara/componentlara stil yazabildiğimiz bir wrapper(sarmalayıcı) eleman oluşturarak çalışır.
- SC rastgele class isimleri üreterek çakışma ihtimalini ortadan kaldırır: **scope** ve **izolasyon**
- **Önemli:** SC wrapperları asıl component scopu dışında tanımlanmalı.

```
const Button = styled.button`  
  color: #BF4F74;  
  font-size: 1em;  
  margin: 1em;  
  padding: 0.25em 1em;  
  border: 2px solid #BF4F74;  
  border-radius: 3px;  
`;
```

Styled-Component Kullanımı

1. projeye styled-component kütüphanesini yükle

```
› npm install styled-components
```

2. import et ve kullan.

```
import styled from 'styled-components'

const Button = styled.button`
  background: transparent;
  border-radius: 3px;
  border: 2px solid #BF4F74;
  color: #BF4F74;
  margin: 0 1em;
  padding: 0.25em 1em;
`
```

Component Kütüphaneleri

React uygulamalarında kullanılmak üzere önceden tasarlanmış componentlar sunar.

- form elemanları, butonlar, menüler, kartlar, ...
- **Artıları:**
 - Hızlı prototipleme
 - Tutarlı tasarımlar
 - Bakım ve güncelleme yükünü azaltması
 - Erişilebilirlik avantajları
- **Eksileri:** Learning Curve, defaultlara yeni yöntemler önerme vs
- Popüler olanlarından bazıları: Reactstrap, MUI, Chakra UI, Ant Design

Reactstrap Kullanımı

1. reactstrap ve bootstrap kütüphaneleri projeye yüklenir.

```
› npm install reactstrap bootstrap
```

2. Bootstrap css dosyasını import et. (App.jsx dosyasında yapabilirsin)

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

3. Componentleri import et ve kullan. (dokümantasyondan örnek kodlara bak)

```
import { Button } from 'reactstrap';

export default function Example(props) {
  return <Button color="danger">Danger!</Button>;
};
```

Ekstra: ⚡ Vite ile Sıfırdan React Projesi Oluşturmak

1. Terminal ekranında vite oluşturma komutunu çalıştır.

```
› npm create vite@latest
```

2. Yükleme adımlarını takip et: Proje adını seç -> framework seç -> ...

```
[✓] Project name: ... vite-project
? Select a framework: > - Use arrow-keys. Return to submit.
  Vanilla
  Vue
  > React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
```

3. Önerdiği komutları çalıştır.

```
Done. Now run:
```

```
cd vite-project
npm install
npm run dev
```