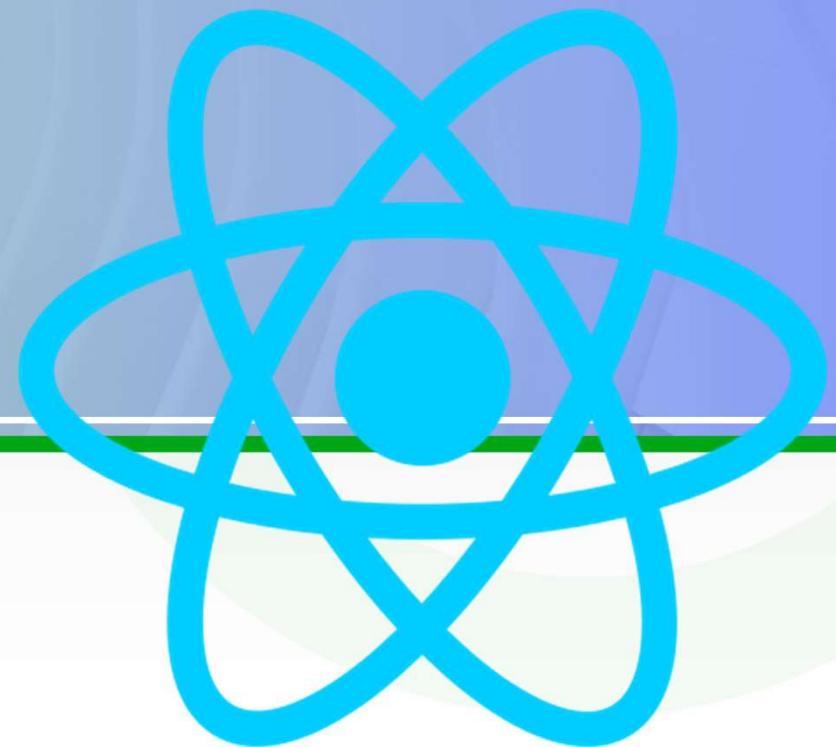




React





REACT

Introduction

- React Nedir?
- Alternatifleri
- Component
- DOM vs Virtual DOM
- One Way Data-Binding
- Dezavantajlar



React Nedir

- › Web uygulamaları için hızlı ve interaktif kullanıcı arayüzleri oluşturmada kullanılan bir **Javascript kütüphanesidir.**
- › Front-end tarafını oluşturur.
- › MVC de View katmanı ile ilgilenir.
- › Projede her bir bölüm bir component (bileşen) olarak değerlendirilir. React projeleri componentlerin bir araya gelmesi ile oluşur.
- › Virtual DOM ile tüm sayfanın yenilenmesi yerine sadece değişiklik olan componentler güncelleyerek ciddi bir hız artışı sağlar





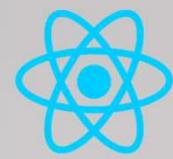
React Nasıl Ortaya Çıktı

- › Facebook tarafından, facebook'un hızlandırılması ve hantallığının ortadan kaldırılması amacıyla geliştirilmiştir.
- › Facebook'un Instagram'ı satın almasıyla birlikte o platformda da kullanılmış.
- › Ardından açık kaynak kodu ile tüm dünya ile paylaşılmış
- › Ancak mevcut html – javascript yapısına ters olduğu için ilk zamanlar kabul görmemiş
- › Airbnb gibi büyük firmalar da kullanmaya başlayınca popüler hale gelmiş



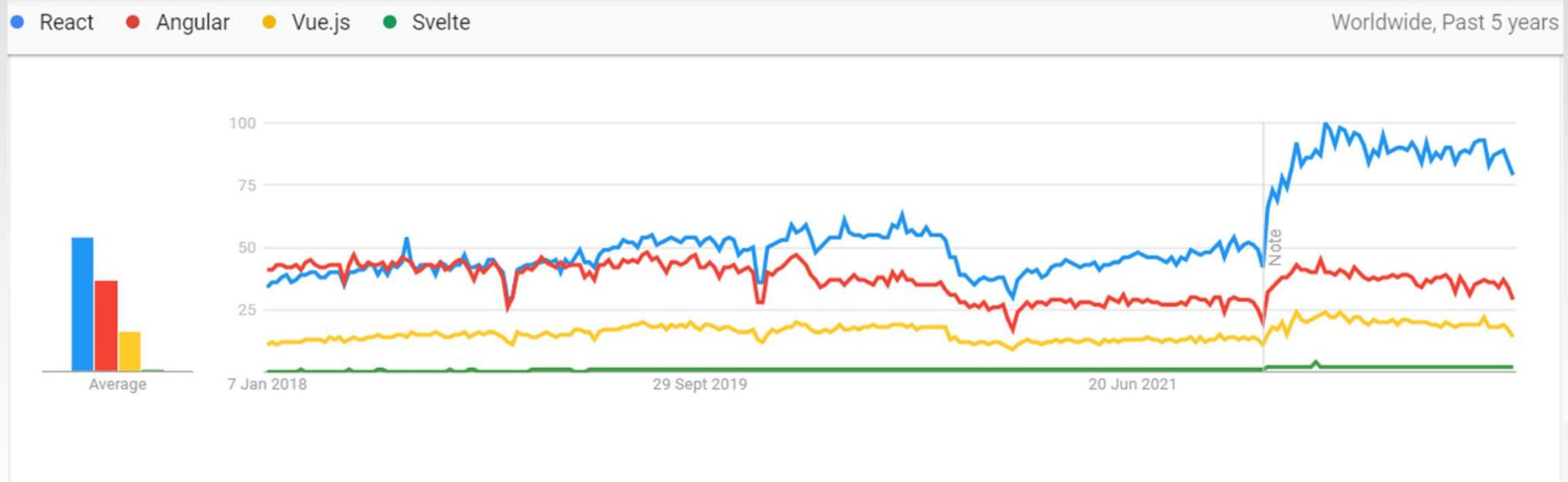


React Alternatifleri

				
Firma / Şahıs	Google (2010)	Facebook (2013)	Rich Harris (2016)	Evan You (2014)
Tanım	Framework	Library	Framework	Framework
Proje Boyutu	180KB	6KB	4KB	63KB
Performans	*	**	***	**
Öğrenme Zorluğu	Zor	Orta	Kolay	Kolay
Popülerlik	Yüksek	En Yüksek	Düşük	Orta
Piyasa Hacmi	Orta	Büyük	Düşük	Orta



Google Trends



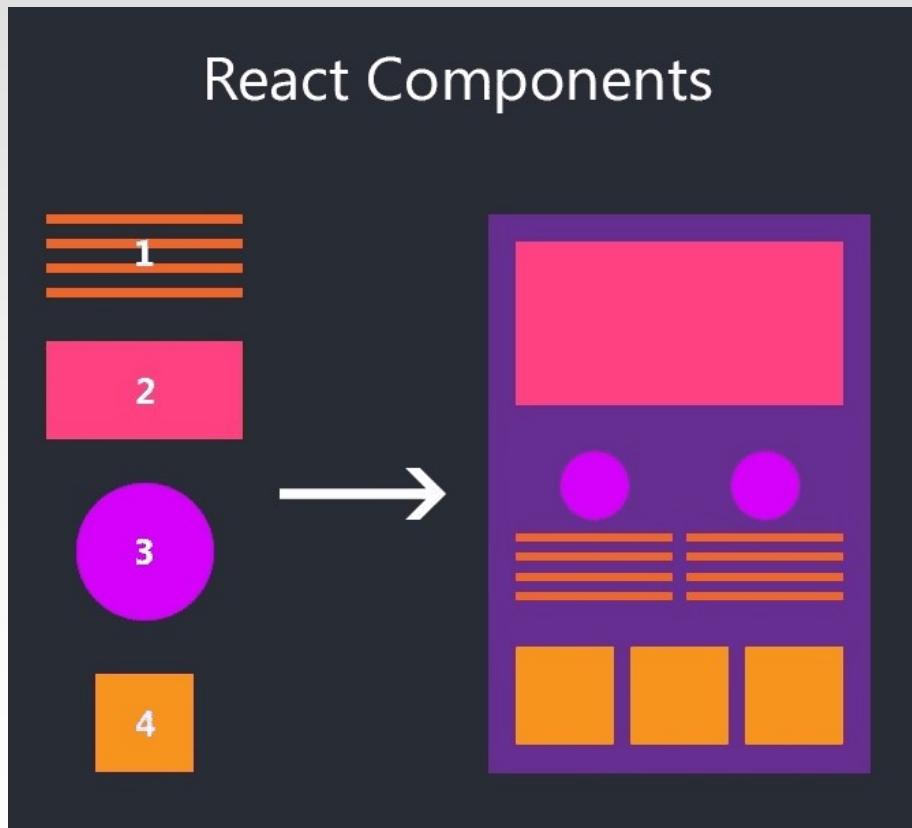


İş hacmi

indeed®



Component

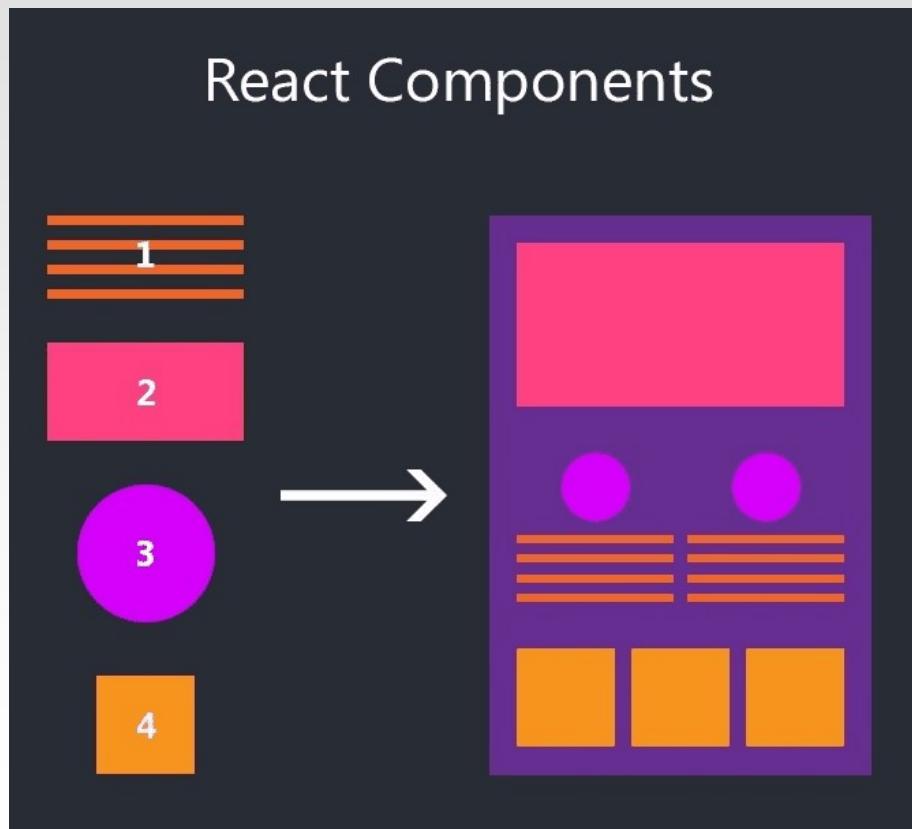


React' ta projeler componentlerden oluşur.

- › Bir uyuglamanın, **bağımsız olarak çalışabilen (isolated)**, **tekrar kullanılabilen (resusable)** her bir parçasına **component** denir
- › Component ler, bir yapbozun parçaları gibi bir araya getirilerek uygulama ortaya çıkarılır
- › Her react uygulaması en azından bir component ten oluşur (root component)
- › Bir component içinde başka componentler barındırılabilir.
- › Componentler arasında veri akışı sağlanabilir.



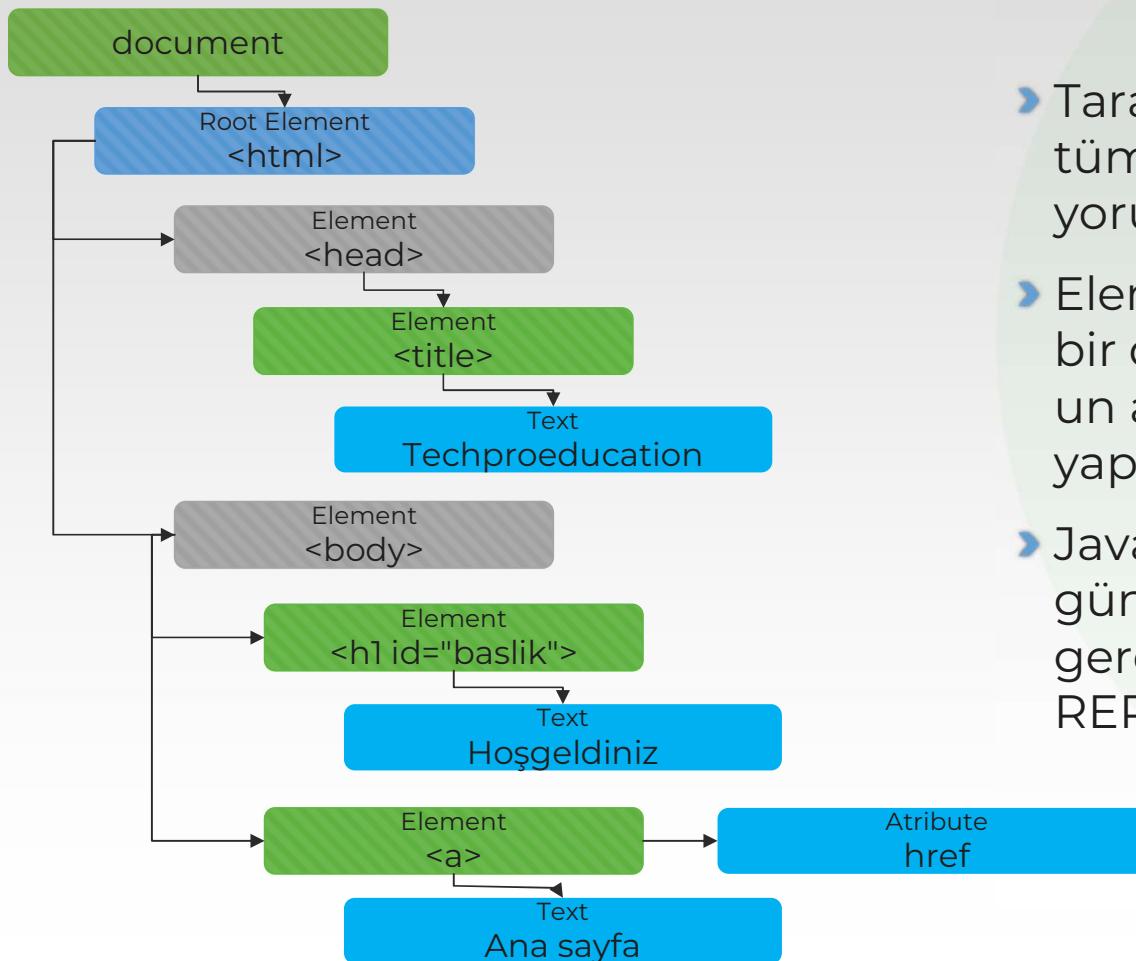
Component State



- › Her component'in kendi özelliklerini içinde barındıran bir **state** i vardır.
- › State ler içinde component'in durumu, datası, stil özellikleri gibi bilgiler tutulabilir.
- › State lerin değişimleri takip edilerek, buna göre çeşitli kodlama hamleleri yapılabilir.
- › State ler child component lere veya parent component lere aktarılabilir.



Document Object Model (DOM)

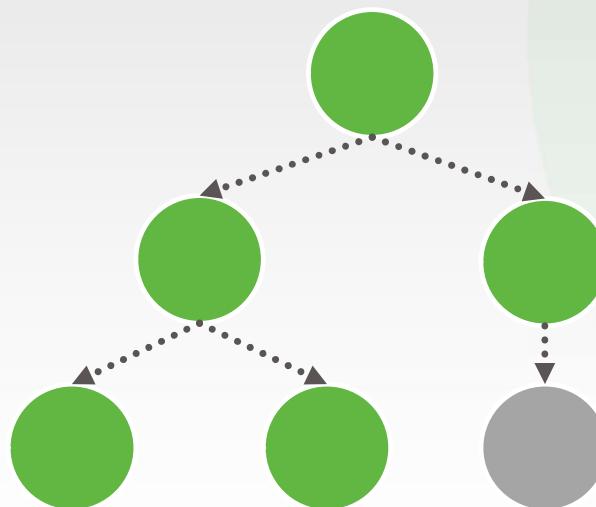


- Tarayıcı sayfayı yükledikten sonra tüm html elementlerini yorumlayarak DOM'u oluşturur.
- Elementlerden herhangi birinde bir değişiklik oluştuğunda DOM'un ağaç yapısında arama yapıldığından yavaşır.
- Javascript ile DOM da bir güncelleme yapıldığında gereksiz yere tüm içeriği REPAINT yapar.



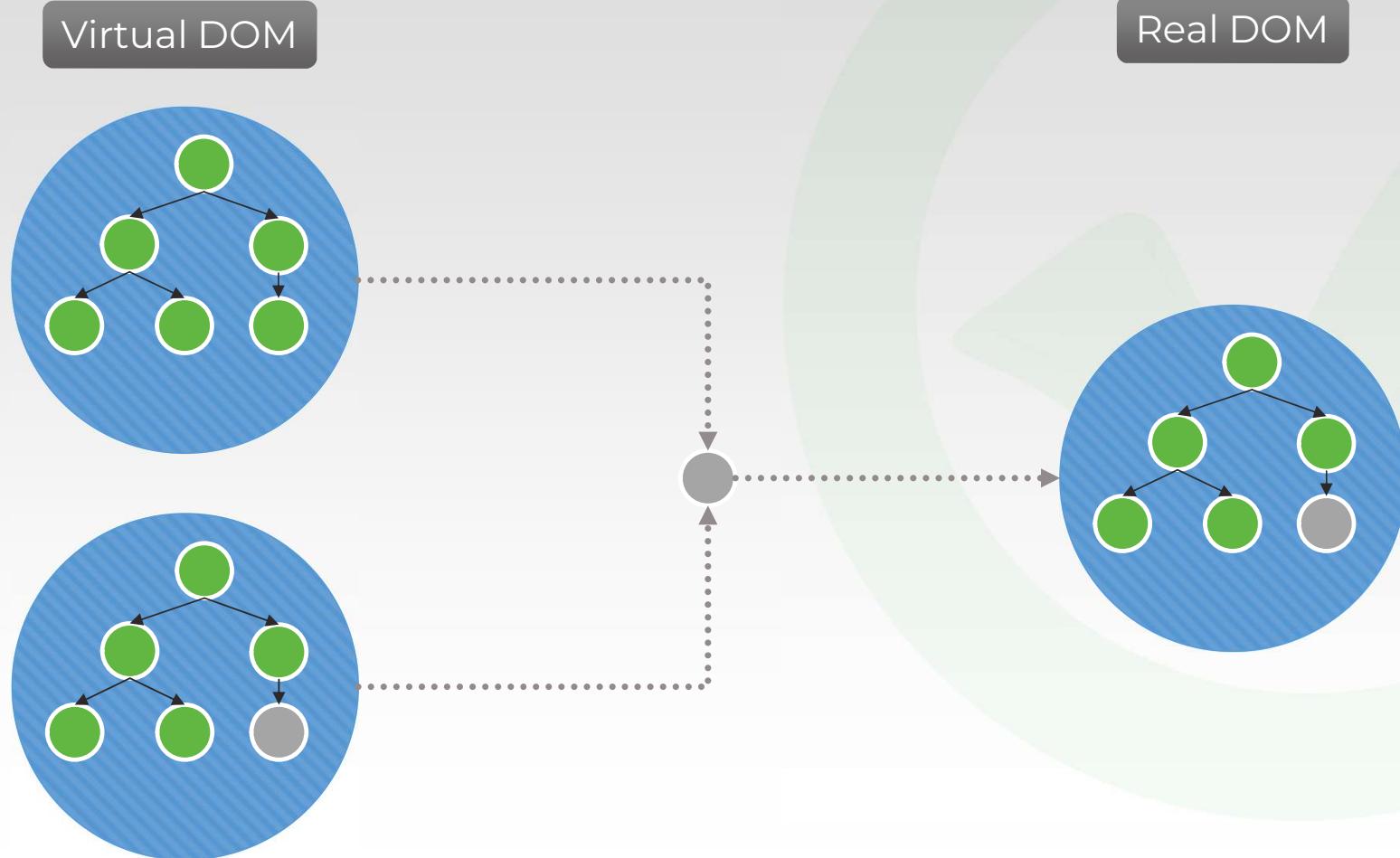
React Neden Hızlı? (JS ye göre)

Virtual DOM





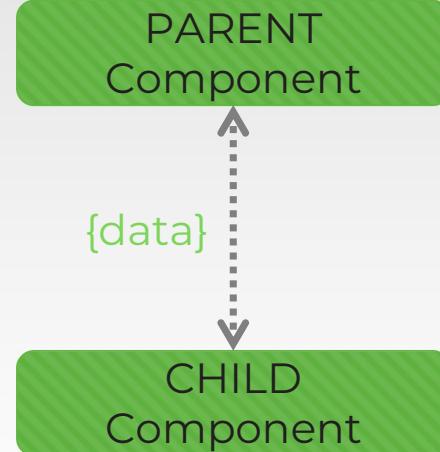
Virtual DOM



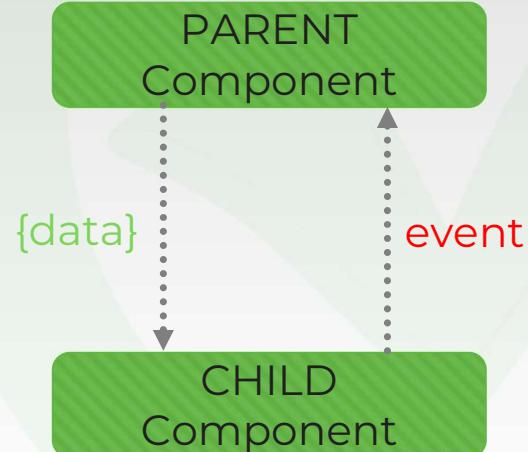


Data Binding

2 Way Binding



1 Way Binding



-vuejs, angular, svelte -

-react-



React Dezavantajları

- Eski tarayıcı desteği yok
 - Internet Explorer 8 ve öncesinde çalışmaz
- En az ES5 kullanmak gereklidir
- Dokümantasyon yetersiz
- JSX' e alışmak zor



REACT

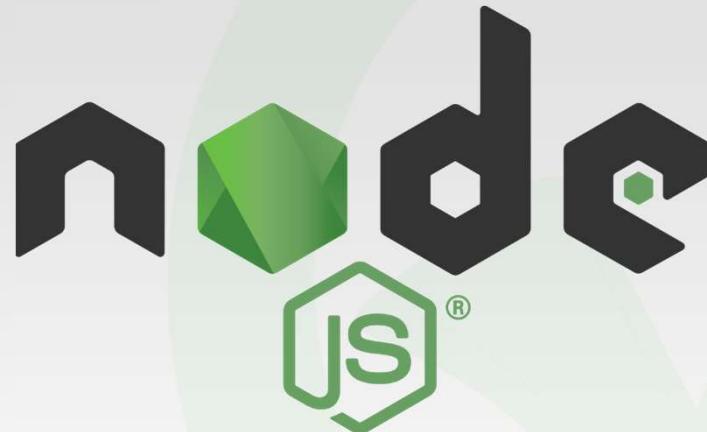
Kurulum

- NodeJs kurulumu
- Proje oluşturma
- Klasör yapısı
- Package.json
- Dokümantasyon



Kurulum

- Nodejs kurulumu
<https://nodejs.org/tr>



Nodejs'i kurma sebebimiz npm (node package manager) ile react projemize farklı kütüphaneler ekleyebilmek içindir.



Uygulama oluşturma

Terminal'de aşağıdaki kodlar sırayla çalıştırılır

```
npx create-react-app my-app  
cd my-app  
npm run start
```

my-app isminde bir react projesi oluşturur

my-app klasörüne girer

Projeyi çalıştırır



Uygulama oluşturma

```
C:\ Windows PowerShell
Starting the development server...
Compiled successfully!

You can now view myapp in the browser.

http://localhost:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Terminate batch job (Y/N)? -
```

Uygulamayı durdurmak için klavyeden
CTRL + C ye basılır



React Klasör Yapısı

```
> build  
> node_modules  
└ public  
    └ favicon.ico  
    └ index.html  
    └ logo192.png  
    └ logo512.png  
    └ manifest.json  
    └ robots.txt  
└ src  
    └ App.css  
    └ App.js  
    └ App.test.js  
    └ index.css  
    └ index.js  
    └ logo.svg  
    └ reportWebVitals.js  
    └ setupTests.js  
└ .gitignore  
└ package-lock.json  
└ package.json  
└ README.md
```

build, proje tamamlandıktan sonra, **npm run build** komutu ile yayına hazırlandığında oluşan klasördür.

node_modules klasörü react ta kullanacağımız tüm kütüphaneleri içerir. **npm install** komutu ile kurduğumuz kütüphaneler burada tutulur. Proje paylaştırılırken bu klasör paylaşılmaz

public, projedeki statik dosyaların tutulduğu klasördür.

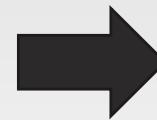
src, projenin kaynak dosyalarını tutmaktadır. React kodlarını barındıran dosyalar burada bulunmalıdır.

Proje ile ilgili genel bilgiler ve projede kullanılan kütüphane bilgileri **package.json** dosyasında bulunur.



Package.json

```
{ package.json > ...
1  {
2    "name": "test",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.14.1",
7      "@testing-library/react": "^11.2.7",
8      "@testing-library/user-event": "^12.8.3",
9      "react": "^17.0.2",
10     "react-dom": "^17.0.2",
11     "react-scripts": "4.0.3",
12     "web-vitals": "^1.1.2"
13   },
14   ▶ Debug
15   "scripts": {
16     "start": "react-scripts start",
17     "build": "react-scripts build",
18     "test": "react-scripts test",
19     "eject": "react-scripts eject"
20   },
21 }
```

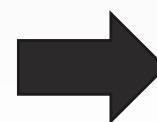


Projede kullanılan npm kütüphanleri. Eğer node_modules klasörü yoksa yada sorun varsa, npm install komutu çalıştırılırsa burada yazan paketler tekrar kurulacaktır.



Package.json

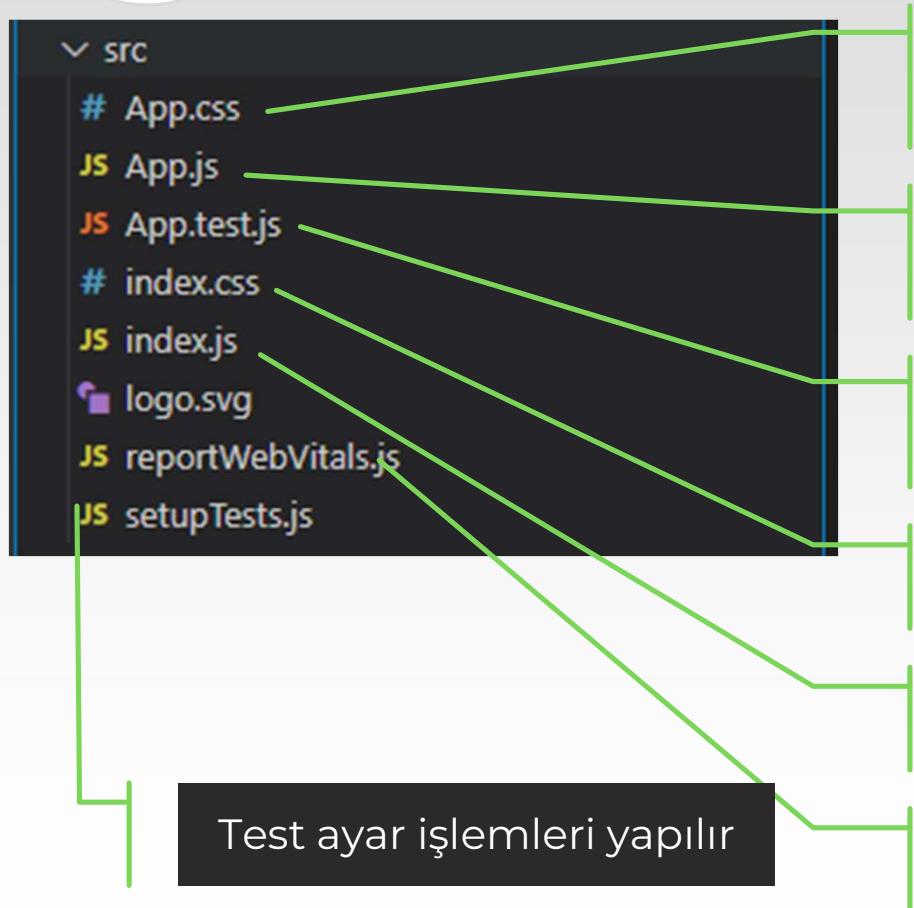
```
{ package.json > ...  
1  {  
2    "name": "test",  
3    "version": "0.1.0",  
4    "private": true,  
5    "dependencies": {  
6      "@testing-library/jest-dom": "^5.14.1",  
7      "@testing-library/react": "^11.2.7",  
8      "@testing-library/user-event": "^12.8.3",  
9      "react": "^17.0.2",  
10     "react-dom": "^17.0.2",  
11     "react-scripts": "4.0.3",  
12     "web-vitals": "^1.1.2"  
13   },  
14   "scripts": {  
15     "start": "react-scripts start",  
16     "build": "react-scripts build",  
17     "test": "react-scripts test",  
18     "eject": "react-scripts eject"  
19   },  
  }> Debug
```



- **npm start**: uygulamayı lokalde çalıştırır
- **npm build**: uygulamayı yayına hazırlar. build klasörünü oluşturur
- **npm test**: test fonksiyonlarını çalıştır
- **npm reject**: react'ın build aracından ve yapılandırma seçeneklerinden memnun değilseniz, kendi build aracınızı oluşturmak için kullanabilirsiniz.



Src klasörü



App isimli componentin css dosyası

App isimli component

App isimli componentin test fonksiyonu

Uygulamanın genel css dosyası

Uygulamanın giriş dosyası

Uygulamanın performansını ölçmek için kullanılan dosya



Dokümantasyon

The screenshot shows the React documentation website. At the top, there is a dark navigation bar with the React logo, navigation links for 'Dokümantasyon', 'Öğretici', 'Blog', and 'Topluluk', a search bar, and version information 'v17.0.2'. Below the navigation bar, the main content area has a dark background featuring a large, faint React logo watermark. The word 'React' is prominently displayed in blue text at the top center. Below it, a subtitle in white text reads 'Kullanıcı arayüzleri geliştirebileceğiniz bir JavaScript kütüphanesi'. At the bottom of the main content area, there are two buttons: a light blue button labeled 'Hemen Başla' and a white button labeled 'Öğreticiye Git >'. The overall design is clean and professional.



React

Fundamentals

- Rendering
- Component hiyerarşisi
- Component oluşturma
- JSX



Uygulamanın render edilmesi

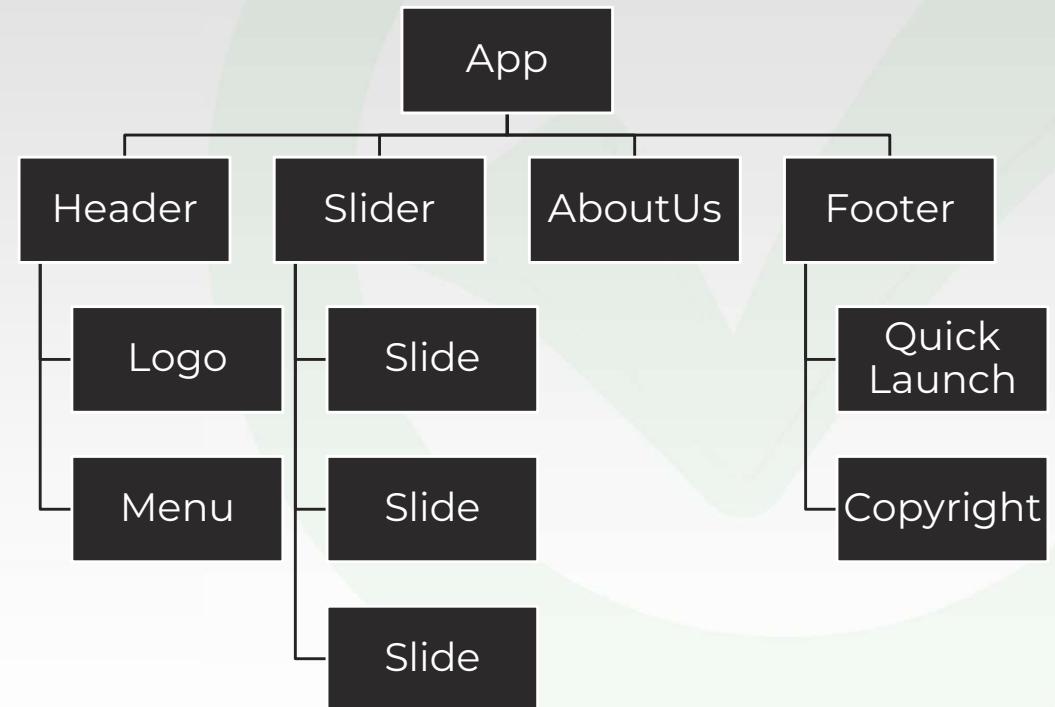
public/index.html

```
<body>
  <div id="root"></div>
</body>
```

src/index.js

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Component Hiyerarşisi





Component oluşturma

```
const MyComponent = () => {  
  return (  
    <div>  
      ...  
    </div>  
  );  
}  
  
export default MyComponent;
```

Component lerin dosya uzantıları **js** veya **jsx** olabilir.

Component ismi büyük harfle başlamalıdır.
(PascalCase)

Dışarıdan parametre alırsa buraya yazılır

Component in içeriği burada oluşturulup kullanıldığı yere return edilir. Bu kısmda JSX yazılmalıdır

Component in başka bir yerden import edilebilmesi için burada export edilmelidir.



Component kullanma

```
import MyComponent from './MyComponent';
```

Component import edilir

```
const App = () => {  
  return (  
    <div className="App">  
      <MyComponent/>  
    </div>  
  );  
}
```

import edilirken
kullanılan isimle
eklenir

```
export default App;
```



JSX

- › Javascript ve XML kelimelerinin kısaltılmış halidir.
- › React ta html ve javascript'in birlikte kullanılmasına izin veren syntax dır.



JSX



JSX

Tüm elementler tek bir root element içinde bulunmalıdır

```
const App = () => {  
  return (  
    <div>  
      <div>Merhaba</div>  
      <MyComponent/>  
    </div>  
  );  
}
```



JSX



JSX

Elementlerin attribute ları camelCase olarak yazılmalıdır.

```
const App = () => {  
  return (  
    <div>  
      <div className="msg">Merhaba</div>  
      <MyComponent/>  
    </div>  
  );  
}
```



JSX



JSX

Expression kullanılacağı zaman, expression {} işaretleri arasına yazılır.

```
const App = () => {  
  const user = {  
    name="Ali",  
    age=34  
  }  
  return (  
    <div className="App">  
      <div>Adı:{user.name}</div>  
      <div>Yaşı:{user.age}</div>  
    </div>  
  );  
}
```





JSX – Conditional Statements

JSX içinde şarta bağlı olarak istenilen componentlerin veya elementlerin gösterilmesi ya da gizlenmesi için kullanılır. Klasik if jsx içinde kullanılmaz

```
const tip = 1;

return(
  <div>
    { tip == 1 ? (
      <div>
        <component1/>
        <component2/>
      </div>
    ) : (
      <component3>
    )
  }
)
```

1.Yöntem:
Ternary





JSX – Conditional Statements

2.Yöntem: Short Circuit Evaluation

```
const a = false;  
  
return(  
  <div>  
    { a && <component/> }  
  </div>  
)
```

a, true ise ya da içinde bir değer var ise component gösterilir

```
const a = false;  
  
return(  
  <div>  
    { a || <component/> }  
  </div>  
)
```

a, false ise veya değersiz ise component gösterilir





React JSX Loops

```
people.map( (person)=> {  
  return (  
    <div>  
      ...  
    </div>  
  )  
})
```

- React' ta JSX içinde yani render methodu içinde bir döngü oluşturulacaksa javascript map kullanılabılır.
- Map methodu dizilere ait bir method olup return ile geriye bir değer döndürmelidir.



JSX içinde sadece geriye değer döndürebilen döngüler kullanılabilir.



Inline Stiller

```
<h1 style={  
    color:"red",  
    backgroundColor:"green",  
    fontSize:"10px"  
} >Merhaba</h1>"
```

Mavi süslü
parentezler object
notasyonu gereği
kullanıldı

Kırmızı süslü
parentezler
expression için
kullanıldı

Stiller camelCase ile
yazılır.



Internal Stiller

```
const App = () => {
  const stil = {
    color:"red",
    backgroundColor:"green",
    fontSize:"10px"
  }

  return (
    <div style={stil}>
      Merhaba
    </div>
  );
}
```

Stiller önceden tanımlanarak
da kullanılabilir



Internal ve Inline Kullanımı

```
const App = () => {
  const stil = {
    color:"red",
    backgroundColor:"green",
    fontSize:"10px"
  }

  return (
    <div style={{...stil, backgroundColor: "red"}>
      Merhaba
    </div>
  );
}
```

Internal ve inline
stiller aynı
element üzerine
uygulanabilir.

Bu durumda
internal stil spread
ile inline stil içinde
acılmalıdır.



External Stiller

App.js

```
import "app.css"
const App = () => {
  return (
    <div className="app">
      <div>Adı:{user.name}</div>
      <div>Yaşı:{user.age}</div>
    </div>
  );
}
export default App;
```

app.css

```
.app{
  color: red;
  align:center;
}
```



External stillerde klasik olarak css kullanılır. inline stillerde olduğu gibi camelCase yapısı kullanılmaz.



SASS

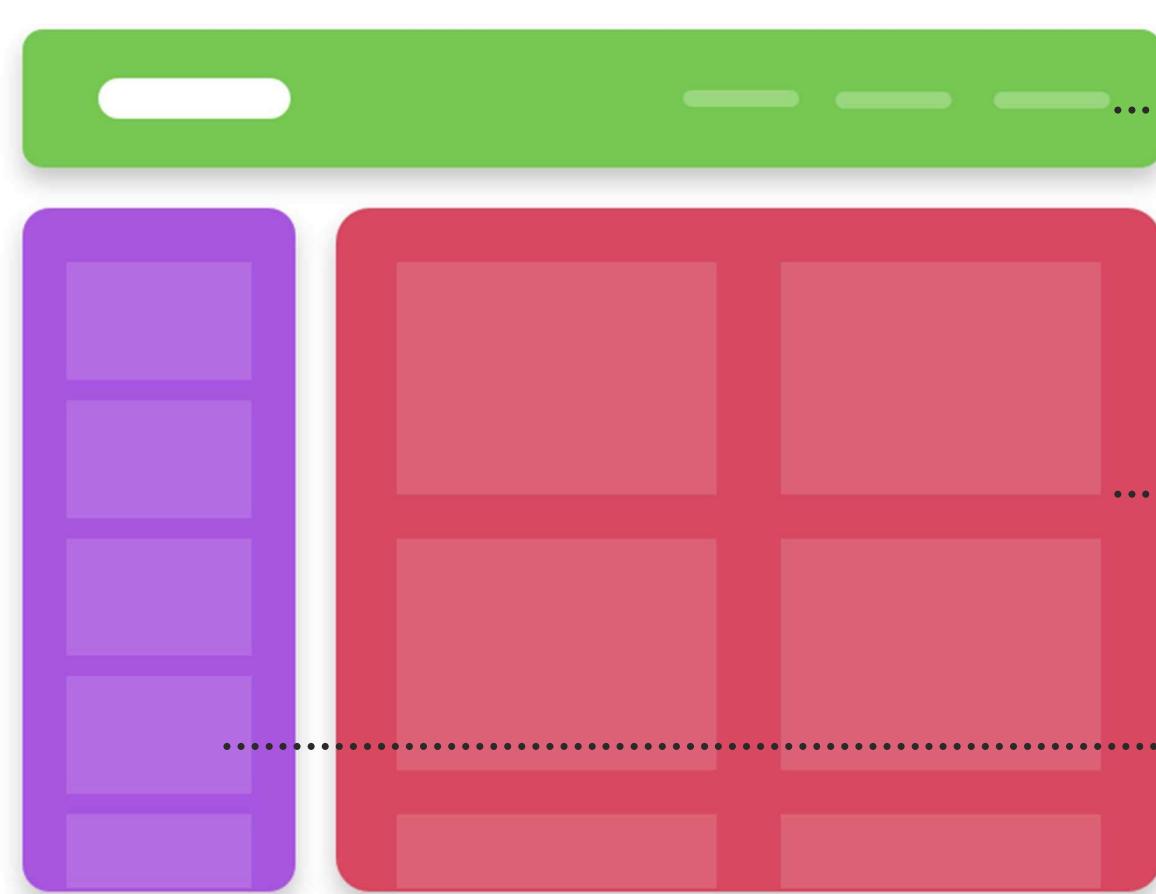
React üzerinde sass modüllerinin çalıştırılabilmesi için sass paketinin yüklenmesi gereklidir.

```
npm i --save-dev sass
```

HOMWORK



Components



HEADER

CONTENT

SIDEBAR

Digital Saat



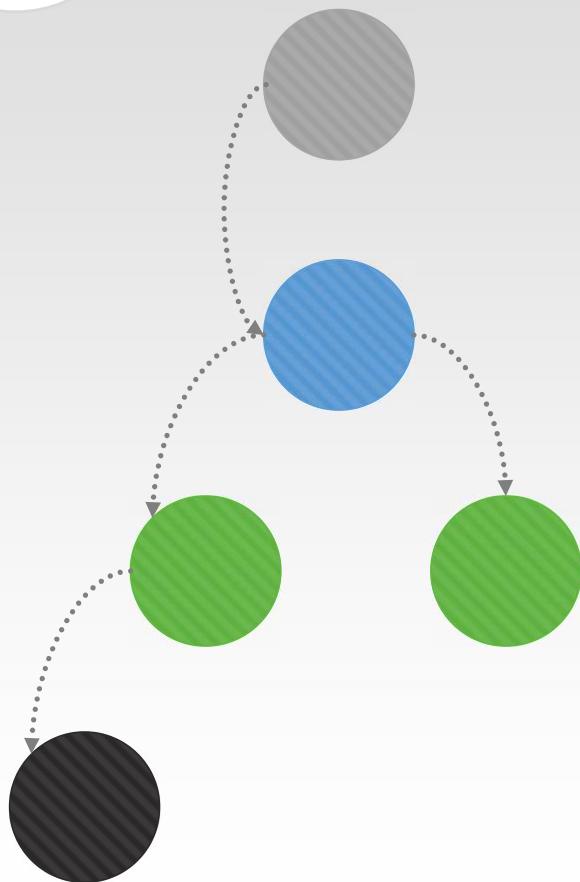
PRACTISE

- › Digital Saat uygulaması yapalım

20:11 AUGUST 13, 2021
FRIDAY EVENING



Props



- › Bir componentin child componentlerine veri aktarımı yapmak için props lar kullanılır.
- › Bu aktarımlar tek yönlüdür (one-way data binding)
- › Fonksiyonların parametre almasına çok benzer
- › Html attribute ları da bu mantıkla çalışmaktadır.



Props

```
const Welcome = (props) => {
  return <h1>{props.name}
    hoşgeldiniz</h1>;
}
```

```
const App = () => {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```



Props

Eğer bir componente, içine aldığı diğer componentleri prop olarak göndermek istersek **children** ifadesinden faydalnırız

```
const Card = (props) => {
  return (
    <div>
      {props.children}
    </div>
  )
}
```

```
const App = () => {
  return (
    <Card>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </Card >
  );
}
```



Digital saat

- › Daha önce yapmış olduğumuz digital saat uygulamasının renklerini parametrik hale getirelim

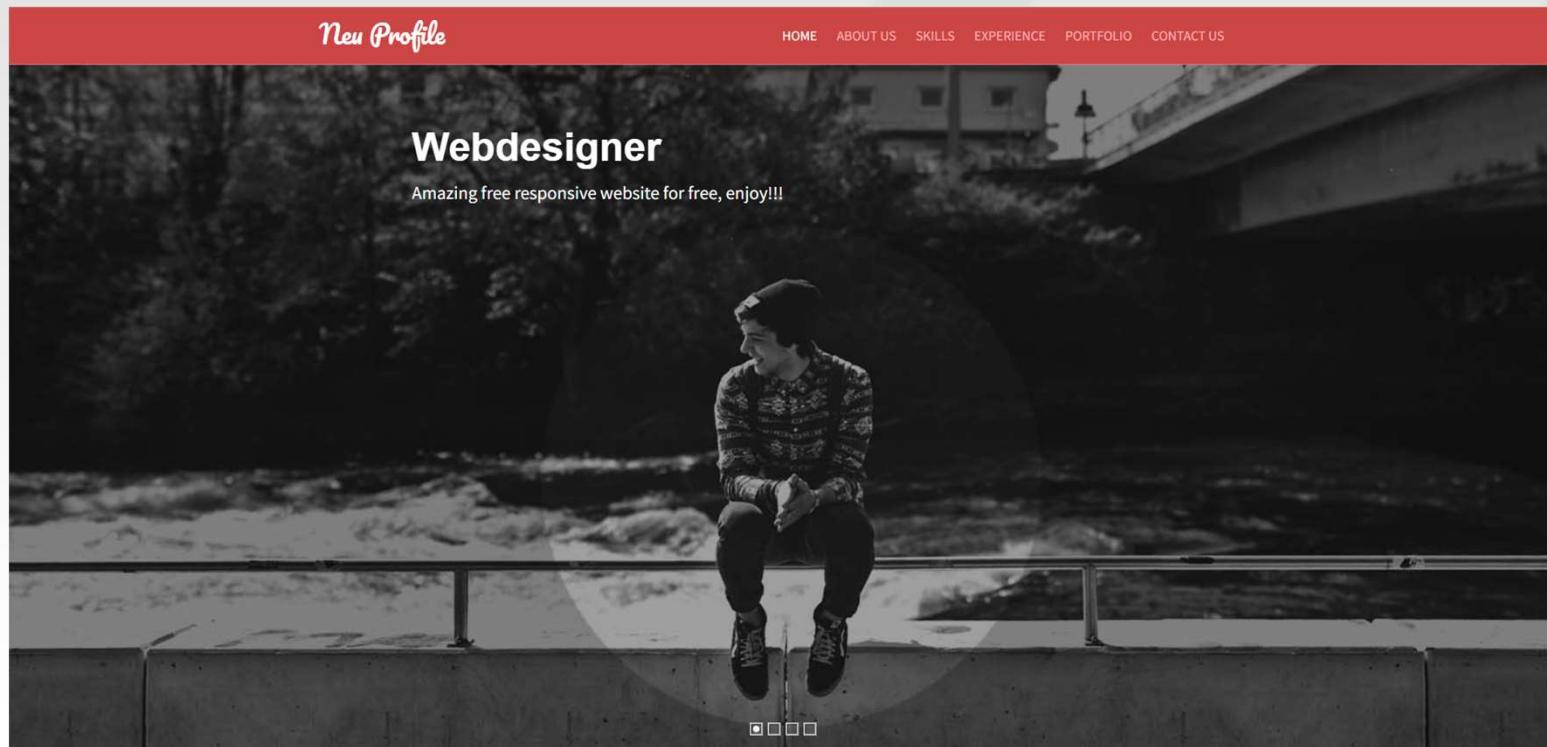


PRACTISE



Component

Hazır bir html template ini react component yapısına çeviriniz.





Görüntü Ekleme

Statik yöntem

```

```

Public klasöründeki görüntülerin eklenmesi için kullanılır

Import yöntemi

```
import profile from "./profile.jpg";
```

```
<img src={profile}/>
```

src klasöründeki görüntülerin eklenmesi için kullanılırlar. Eğer görüntünün adı dinamik olarak bir dış kaynaktan alınıyorsa bu durumda require kullanmak gereklidir.

Require yöntemi

```
<img src={require("./profile")}/>
```



Görüntü ekleme

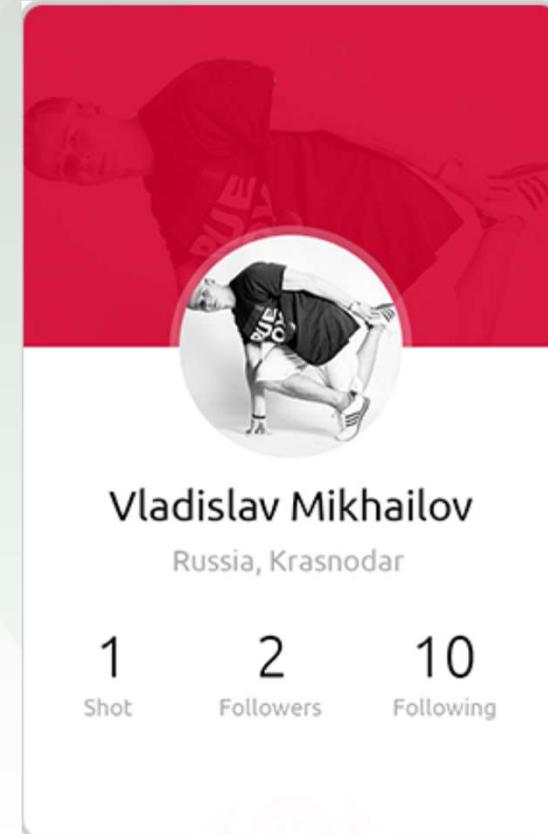
Src/assets/img/gallery klasöründe bulunan görüntüler bir dizi aracılığı ile döngü içerisinde alınarak gösterilecektir.



Görüntü ekleme

Dışarıdan aşağıdaki props'ları alarak bir kişi kartı oluşturan component tasarlaymentınız

- › Name
- › Image
- › Place
- › Shot
- › Followers
- › Following



HOMEWORK



Card desing

A product card for a men's shirt. It features a male model wearing a light blue button-down shirt. Below the image is a rating of 4.5 stars and the text "MEN'S SHIRT" followed by the price "\$65.90".

A product card for a women's top. It features a female model wearing a brown blouse with puffed sleeves and jeans. A green discount badge in the top left corner says "-10%". Below the image is a rating of 5 stars and the text "WOMEN'S TOP" followed by the original price "\$70.00" and the discounted price "\$63.00".

Dışarıdan aşağıdaki props'ları alan bir product card component tasarlayınız

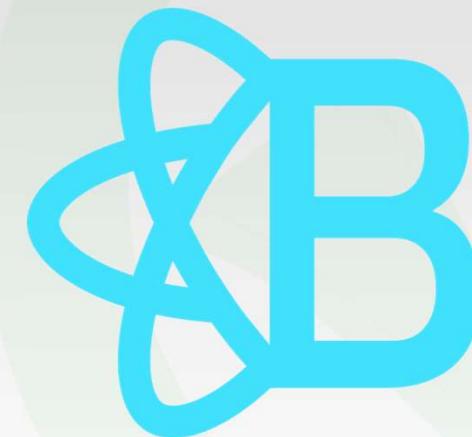
- › Title
- › Image
- › Price
- › Discount
- › Rate



Bootstrap



Bootstrap



reactstrap



Bootstrap

public/index.html

```
<html>
  <head>
    <link rel="stylesheet"
  href="bootstrap.css"/>
  </head>
  <body>
    ...
    <script src="bootstrap.js"/>
  </body>
</html>
```

App.js

```
const App = () => {
  return (
    <div
      className="alert alert-primary"
      role="alert">
      Hello
    </div>
  );
}
```



react-bootstrap

```
npm install react-bootstrap bootstrap
```

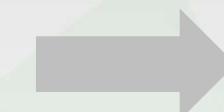
index.js

```
import "bootstrap/dist/css/bootstrap.min.css";
// index.js içerisinde bootstrap.css import edilir
```

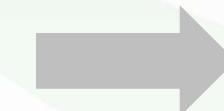
veya

index.scss

```
@import "bootstrap/scss/bootstrap";
// index.scss içerisinde bootstrap.scss import edilir
```



CSS



SCSS



react-bootstrap

App.js

```
import { Alert } from 'react-bootstrap';

const App = () => {
  return (
    <Alert color="primary">
      Hello
    </Alert>
  );
}
```



Icon kütüphanesi

react-icons.github.io

react-icons kütüphanesini yükleyin

```
npm install react-icons
```

İlgili icon kütüphanesini import edin

```
import { FaBeer } from 'react-icons/fa';
```

İstediğiniz ikonu kullanın

```
const Test = () => {
  render() {
    return <h3> Lets go for a <FaBeer />? </h3>
  }
}
```



Events

Parametresiz event handler

Event listener

```
const MyComp = () => {  
  
  const sayHello = () => {  
    alert("Hello there!");  
  };  
  
  return (  
    <div>  
      <div onClick={sayHello}>Say Hello</div>  
    </div>  
  );  
};
```

Event handler



Events

Parametreli event handler

```
const MyComp = () => {  
  
  const sayHello = (name) => {  
    alert(`Hello ${name}`);  
  };  
  
  return (  
    <div>  
      <div onClick={()=> sayHello('Ali Gel')}>Say Hello</div>  
    </div>  
  );  
};
```

PRACTISE



Ürünler

Yanda görüldüğü gibi bir tasarım yapınız.

Oluşturulacak componentler

- Header
- ProductList
- Product
- Footer

TechnoShop

Home Products About Us Contact Us

Search

Search

Anker Soundcore
Hybrid Active Noise Cancelling Headphones
59\$

Add to cart

Sony MDRZX110NC
Noise Cancelling Headphones, Black
79\$

Add to cart

Bose SoundLink
Around Ear Wireless Headphones II - Black
55\$

Add to cart

iJoy Matte
Finish Premium Rechargeable Wireless Headphones
Bluetooth Over Ear

Beats Solo3
Wireless On-Ear Headphones - Apple W1
Headphone Chip

Sony WH-1000XM4
Wireless Industry Leading Noise Canceling
99\$



REACT

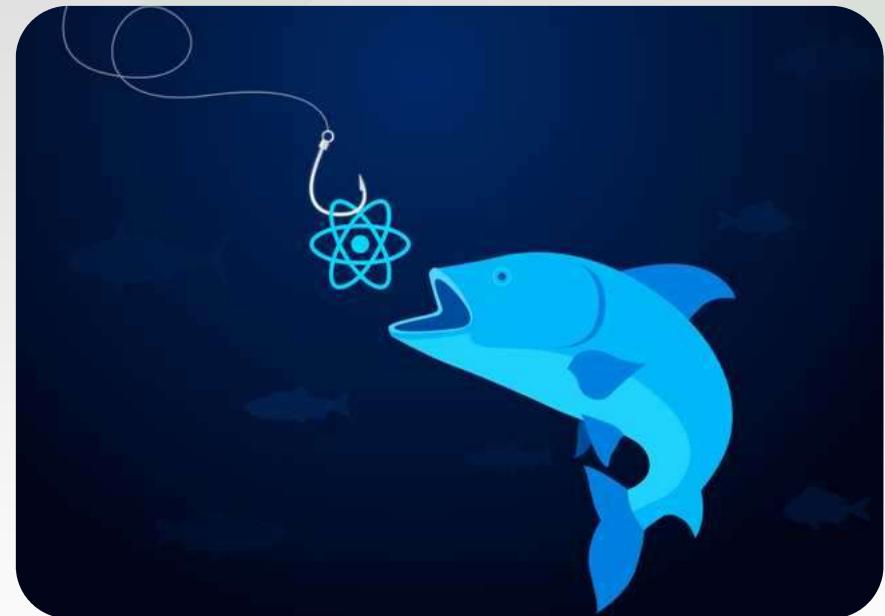
HOOKS

- Hooks nedir?
- State nedir
- useState hook
- useEffect hook
- useRef hook



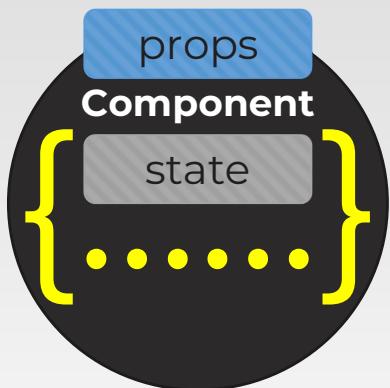
Hooks

- › "Hook" lar bir class component yazmadan react özelliklerini kullanmamıza olanak sağlayan bir yapıdır.
- › Sadece function type componentlerde kullanılabilirler.
- › En çok kullanılan hooklar: *useEffect*, *useState*, *useContext*, *useRef*
- › Hooklar en üst seviyede kullanılır.
Döngülerde, koşullarda kullanılmaz
- › <https://tr.reactjs.org/docs/hooks-intro.html>





State



- Her componentin kendine ait bir state i olur.
- OOP mantığında **private fields** kavramına karşılık gelir.
- State, component içinde kullanılan data yi saklamaktadır. Bir butona basılıp basılmadığı, bir textbox a yazılan yazı, bir hesaplama sonucu veya bir API den gelen ürün bilgisi gibi data, state içinde saklanmaktadır.
- Bir component in state bilgisine başka bir component ten ulaşılamaz. (**Encapsulation**)
- Ancak component izin verirse, props lar üzerinden state dışarıya aktarılabilir.



useState

```
const [counter, setCounter] = useState(0);
```

Değeri saklayan
değişken

Değeri
değiştirmekle
görevli method

useState hook u
değişkenin
saklayacağı ilk
değeri parametre
olarak alır.
Bu ilk değer
numeric, string,
object, array olabilir

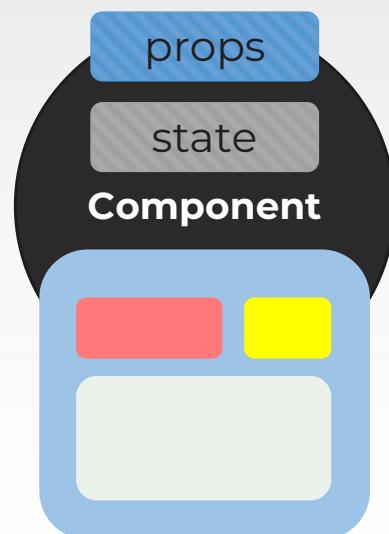


Set fonksiyonları asenkron çalışmaları için peş
peşe çağrılderinde önceki state değerlerini
koruyamayabilirler. Bu sebeple peşpeşe
çağrılmaları tavsiye edilmez.



Rendering & Re-rendering

Component içindeki JS ve JSX kodlarının çalıştırılması işlemine **rendering** denir. Rendering sadece ilk yüklemeye gerçekleşir



Re-rendering ise aşağıdaki durumlarda oluşur.

- State güncellendiğinde
- Prop güncellendiğinde
- Parent component re-render olduğunda



useState

```
import React, { useState } from 'react';

const Example() => {
  const [mode, setMode] = useState("light");
  return (
    <div>
      <button onClick={()=>setMode("dark")}>Dark Mode</button>
      <button onClick={()=>setMode("light")}>Light Mode</button>
    </div>
  );
}
```

PRACTISE



Counter

- Function component kullanarak sayaç tasarılayınız.

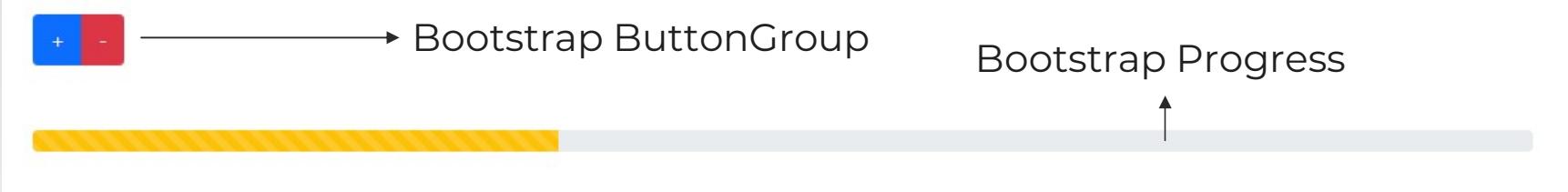


Eğer aynı anda hem state'in mevcut değeri okunup, ona bağlı olarak bir değişiklik yapılacaksaksa aşağıdaki kalıp kullanılmalıdır:

setState(prev=> prev+1)



Progress



- (+) butonuna basıldığında Progress bar 10 birim dolacak,
- (-) butonuna basıldığında 10 birim azalacak.
- 0 – 100 arasında değer alabilecek
- Anlık değerini de gösteriniz

PRACTISE



Bugün doğanlar

Harici bir dosyadan alınacak bir data ile yandaki gibi bir «bugün doğanlar» çalışması yapınız.

Bugün Doğanlar

Bugün doğan 5 kişi bulundu



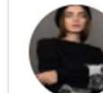
Banu Korkut

27



Hüseyin Şahin

28



Sevda Kara

24



Recep Fırat

27



Sevinç Genç

30

[Temizle](#)



Side Effects

React component ler genellikle basit fonksiyonlarla oluşturulurlar. Bu fonksiyonlar beklenmedik sonuçlar vermezler. Çünkü bizim kontrolümüzdedir.

Bizim kontrolümüzde olmayan, nasıl sonuçlanacağı önceden tahmin edilemeyen, component dışı olaylara **side effects** denir.

API bağlantıları

Doğrudan DOM a erişim

setTimeout veya
setInterval gibi
zamanlayıcı
fonksiyonlar



useEffect Hook

useEffect, side effect lerin kullanılması için oluşturulmuş kod bloklarıdır. *Class type component* lerde side effect ler **life cycle event** (yaşam döngüsü olayları) leri içinde kullanılırken *function type component*lerde bunun için **useEffect** hook kullanılır.

Mounting

Updating

Unmounting

useEffect



useEffect Hook

Mounting

```
useEffect( ()=> {  
  console.log('ilk renderda çalışır')  
}, [] )
```

Sadece ilk render da çalışır.

Updating

```
useEffect( ()=> {  
  console.log('her render da çalışır')  
})
```

Component her render edildiğinde
çalışır



useEffect Hook

Updating

```
useEffect( ()=> {  
  console.log('ilk renderda çalışır')  
}, [sayac] )
```

sayac isimli state veya prop her güncellendiğinde çalışır

Unmounting

```
useEffect( ()=> {  
  return () => {  
    console.log('unmount olduğunda  
    çalışır')  
  }  
}, [] )
```

Component unmount olduğunda çalışır.



Digital Saat

- › Daha önce yapmış olduğumuz digital saat uygulamasını çalışacak şekilde kodlayınız

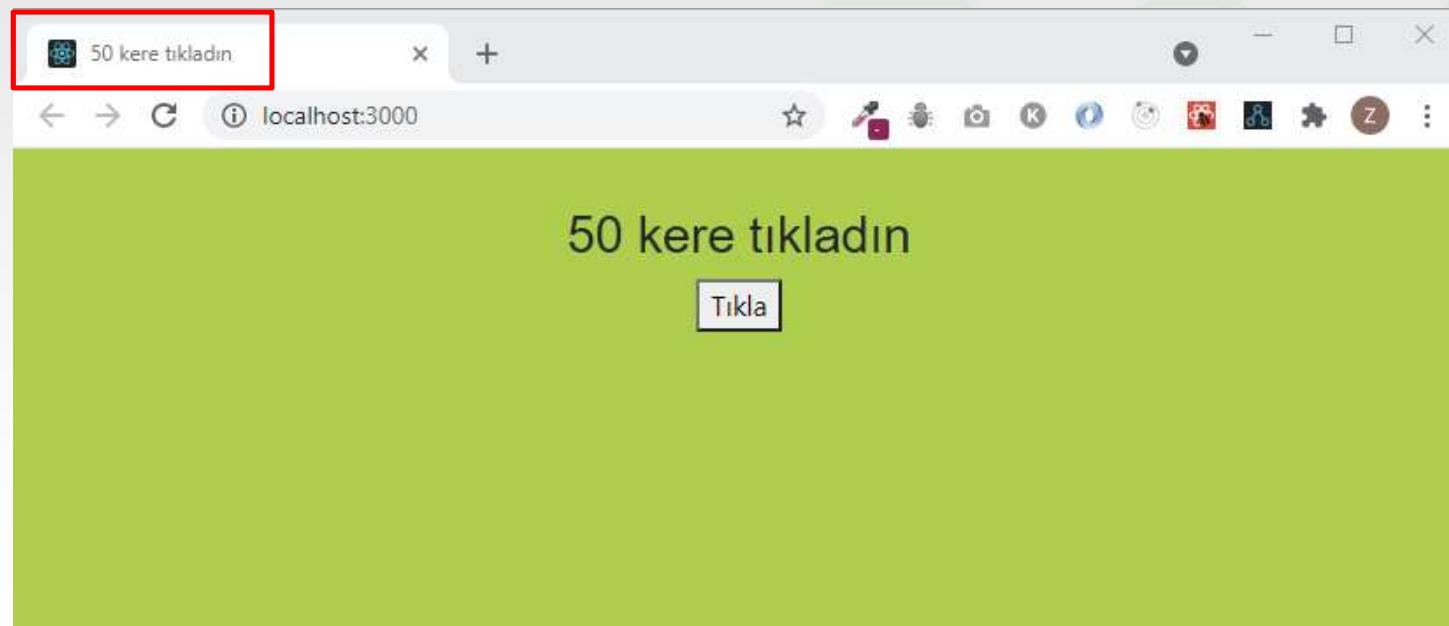
20:11 AUGUST 13, 2021
FRIDAY EVENING

PRACTISE



Counter

- ▶ Butona her basıldığında counter değeri artırılacak, sayfanın title değeri güncellenecek



PRACTISE



Filtreleme

Filter by **value**

Name	Age
Banu Korkut	27
Hüseyin Şahin	28
Sevda Kara	24
Recep Fırat	27
Sevinç Genç	30

Şekildeki gibi bir listeyi textbox'a girilen degree göre filtreleyen bir uygulama yapınız.



useRef Hook

DOM elemanlarına erişmek için kullanılır. Vanila JS deki `document.querySelector` ifadesinin React taki karşılığıdır.

```
const FocusInput = () => {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    inputEl.current.focus();
  };

  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus</button>
    </>
  );
}
```



useRef değeri
değiştiğinde
rerender
gerçekleşmez



REACT

Component Types

- Component tipleri
- Class type components
- Life-cycle methods



Class Component

React ile 2 farklı şekilde component tanımlanabilir.

Function Type Component

Class Type Component



Class Type Component

Class componentlerde çok fazla boilerplate (basma kalıp) kod bulunmaktadır

```
class HelloMessage extends React.Component {  
  
    render() {  
        return ( <div> Merhaba </div> );  
    }  
}
```



Class Type Component

```
class HelloMessage extends React.Component {  
  
  constructor(props) {  
    super(props);  
    this.state = { seconds: 0 };  
  }  
  
  render() {  
    return ( <div> Merhaba </div> );  
  }  
}
```

Constructor

- Constructor class component oluşturulmadan hemen önce çalışır.
- Props lar constructor a gönderilir.
- Super statement ile parent constructor çağrılmalıdır.
- Constructor genelde state in oluşturulduğu yerdır.



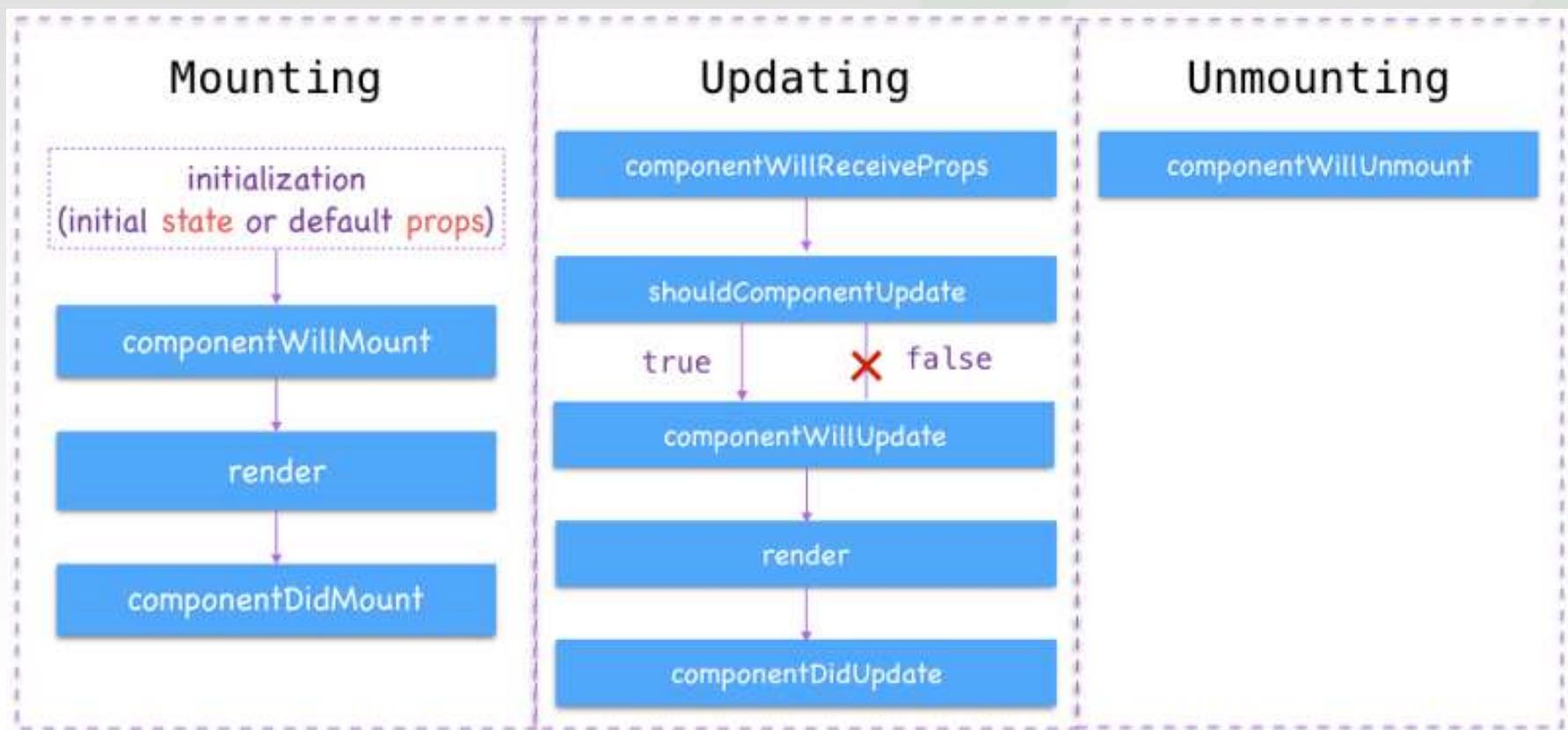
Class Type Component

```
class HelloMessage extends React.Component {  
  componentDidMount() {  
    this.interval = setInterval(() => this.tick(), 1000);  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.interval);  
  }  
}
```

Life Cycle Methods

Lifecycle metodları component'in durumuna göre otomatik olarak çalıştırılan built-in metodlardır.

Life Cycle Methods





REACT

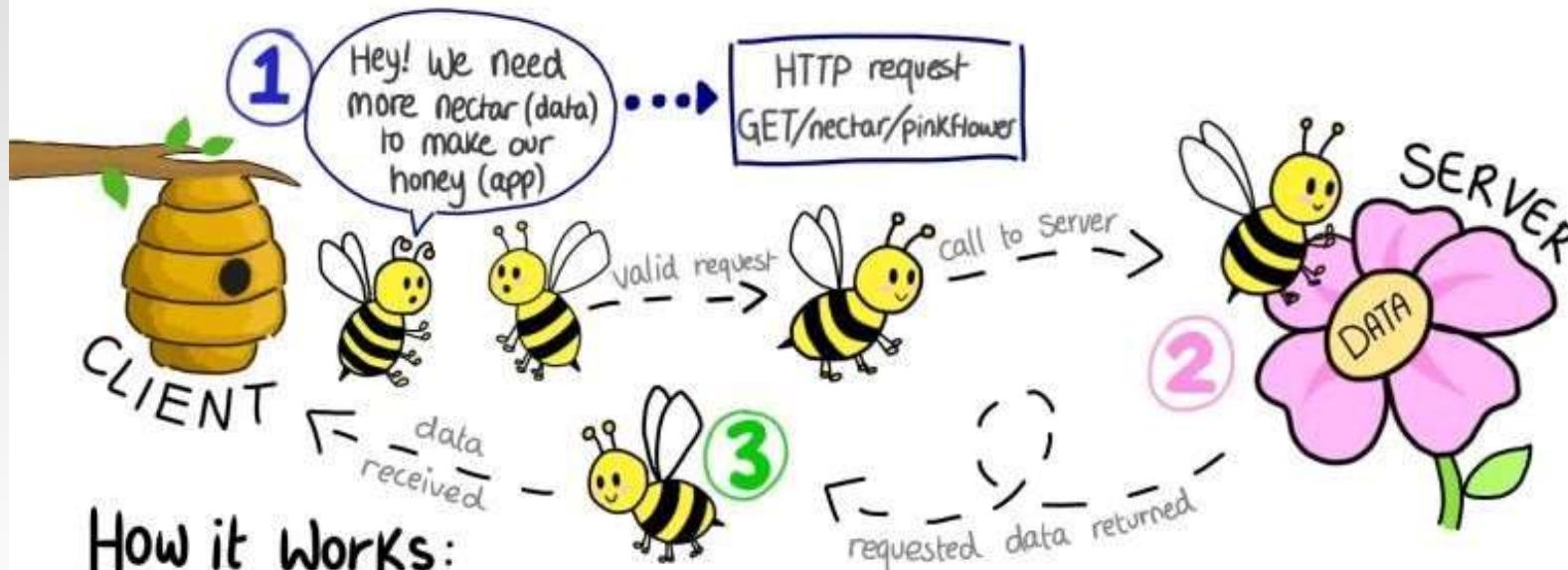
API

- API nedir?
- MockAPI
- Postman
- Fetch
- Axios

What is an API?



An application programming interface allows two programs to communicate. On the web, APIs sit between an application and a web server, and facilitate the transfer of data.



How it Works:

1 Request

API call is initiated by the Client application via a HTTP request

2 Receive

Our worker bee acts as an API, going to a flower (server) to collect nectar (data)

3 Response

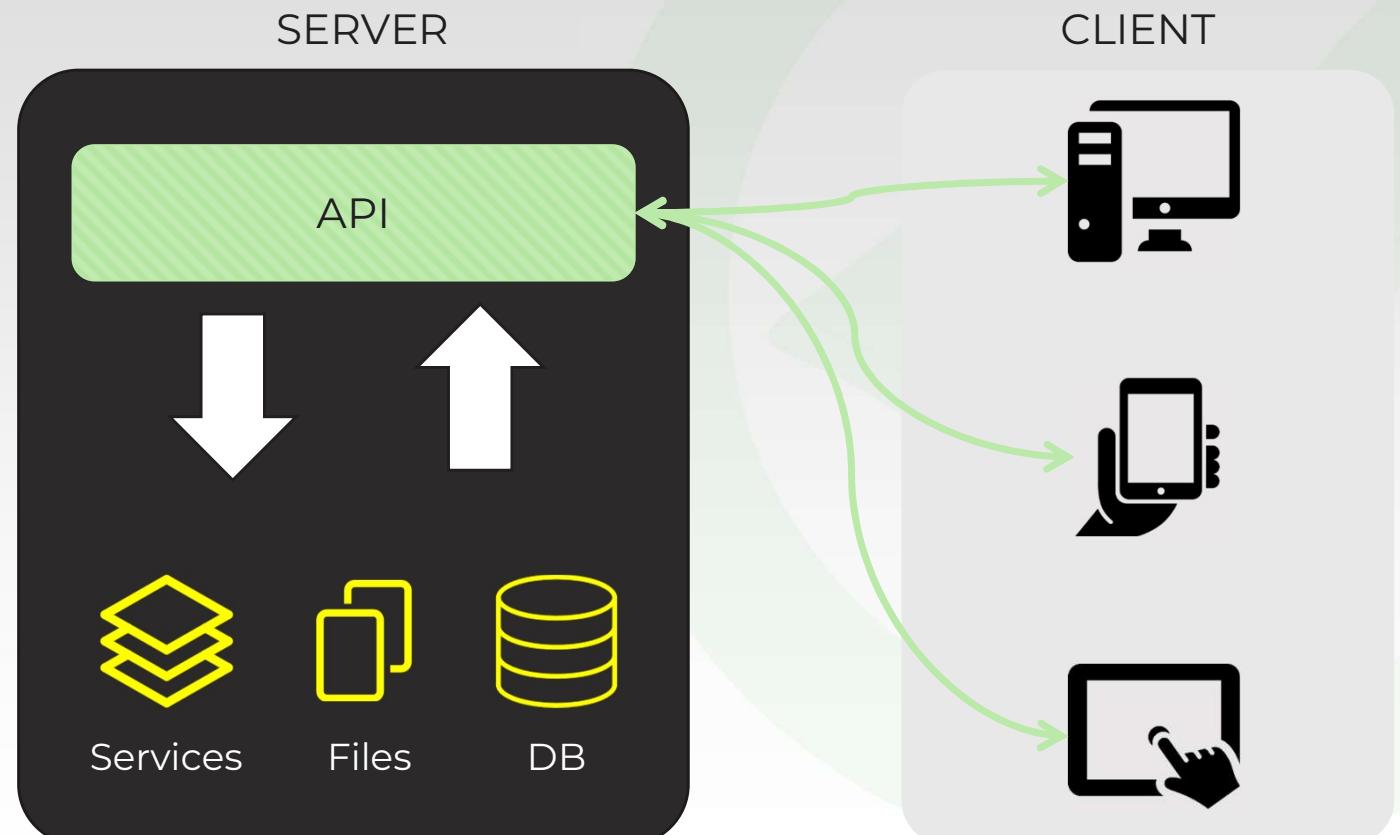
The API transfers the requested data back to the requesting application, usually in JSON format





Web API Nedir

Farklı yazılım sistemleri ile iletişim kurmak için uyulması gereken kurallar veya kullanılması gereken protokollerdir.





En popüler Web API mimarileri

	REST Representatioanl State Transfer	SOAP Simple Object Access Protocol
Type	Principles	Protocol
Bandwith	Low	High
Cacheable	Yes	No
Data formats	XML, Json, Text, ...	XML
Invokation	HTTP(S)	SOAP
Javascript support	Easy	Difficult
Reliability	Not reliable	Reliable
Performance	High	Low
Who use it?	Social media, web apps, mobile apps, IOT...	Financial, Payment Gateways, Telecommunication



RESTful (REpresentational State Transfer)

REST mimarisini uygulayan web uygulamalara RESTful uygulamalar denir. RESTful istekleri ve cevapları belli bir yapıdan oluşur.

İstekler (Request)

- Endpoint
- Method
- Body
- Header

Cevaplar (Response)

- Status
- Body
- Header



RESTful Request

Endpoint **https://example.com/api/v1/users**

Method

- Get Bil almak için
- Post Bilgi göndermek için
- Put Bilgi güncellemek için
- Delete Bilgi silmek için

Body

```
{  
  "name": "Ali",  
  "age": 3,  
  "salary":50000  
}
```

Header

Authorization, Content-Type, ...



RESTful Response

- Status
- Body
- Header

200	Başarılı
300	Redirection
400	Client kaynaklı hatalar
500	Server kaynaklı hatalar

```
{  
    "name": "Ali",  
    "age": 3,  
    "salary": 50000  
}
```

JSESSIONID, Content-Type



Mock API

Frontend tarafında API testleri yapabilmek için kolaylıkla sahte API üretmek için kullanılan ücretsiz bir platformdur.

mockapi.io



Postman

API endpointlerini test etmek için kullanılan bir uygulamadır



postman.com/downloads



Swagger

API endpointlerini, modellerini online olarak gösteren ve testler yapmamızı sağlayan dokümantasyondur.





Javascript Fetch

Fetch' e ilk verilecek parametre api endpoint adresidir. Bu kısımda sunucuya istekte bulunulur

Sunucudan gelen cevap res değişkenine eşitlenecektir. Gelen cevap json a bu aşamada çevrilir.

```
fetch('api_url', {})  
.then( res => res.json() )  
.then(data => console.log(data))  
.catch(err=> console.log(err))
```

Fetch ayaları bu object içinde yapılabilir

Json a çevrilen data bu aşamada kullanılabilir.

Hata olursa bu aşamaya hata gönderilir.

PRACTISE



Mock API

MOCK API kullanarak oluşturduğunuz API'yi kullanarak kullanıcıları listeleyen bir uygulama yapınız



[mojombo](#)



[defunkt](#)



[pjhyett](#)



[wycats](#)



[ezmobius](#)



[ivery](#)



[evanphx](#)



[vanpelt](#)



Axios

Axios' a ilk verilecek parametre api endpoint adresidir. Bu kısımda sunucuya istekte bulunulur

Eğer post, put, delete işlemi yapılacaksa axios.post kalımı kullanılır

```
axios('api_url', {})  
.then(res => console.log(res))  
.catch(err=> console.log(err))
```

Gönderilecek data burda tanımlanır

Axios da response otomatik olarak json a çevrilmektedir.

Hata olursa bu aşamaya hata gönderilir.

PRACTISE



Countries API

- <https://restcountries.com/v2/all> api kullanarak ülke isimleri ve bayraklarını gösteren bir uygulama yapınız.
- Bu listenin başlıklarına tıklandığında listeyi sıralasın

#	Bayrak	Ülke	Nüfus	Başkent
1		Afghanistan	27657145	Kabul
2		Åland Islands	28875	Mariehamn
3		Albania	2886026	Tirana
4		Algeria	40400000	Algiers
5		American Samoa	57100	Pago Pago
6		Andorra	78014	Andorra la Vella
7		Angola	25868000	Luanda
8		Anguilla	13452	The Valley

Child'dan Parent'a prop aktarımı

PARENT

```
const Parent = () => {
  const [count, setCount] = useState(0)

  const callback = (val) => {
    setCount(count + val);
  }

  return (
    <div>
      <Child parentCallback={callback}/>
      <h2>You clicked {state} times</h2>
    </div>
  )
}
```

1

Child componente
gönderilecek bir method
tanımlanır.

2

Bu method bir prop
aracılığı ile Child a
gönderilir.



Child'dan Parent'a prop aktarımı

CHILD

```
const Child = (props) => {  
  
  const artir = () => {  
    props.parentCallback(2); ←.....  
  };  
  
  return (  
    <button onClick={artir}>Click me  
    </button>  
  );  
};
```

3

Child component içinde,
props ile parent tan
gonderilen method
cağrılaraK data parent a
geri gönderilir.

HOMEWORK



Todo app

Title

Create Note

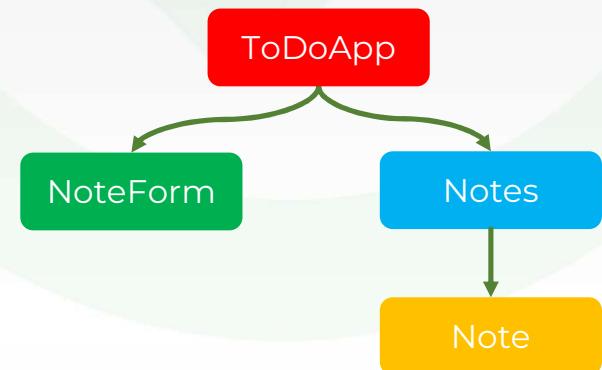
Çocukları okuldan al

Derne hazırlan

Ödevleri kontrol et

Dil çalış

MOCK API kullanarak yapınız





REACT

Form Handling

- Form oluşturma
- Form handling
- Form validation



Formlar

```
const Form01 = () => {
  const [name, setName] = useState("");
  const x = (e) => {
    setName(e.target.value);
  };
  return (
    <div>
      <h1>Merhaba {name}</h1>
      <input type="text" className="form-control"
        value={name} onChange={x}
      />
    </div>
  );
};
```

Form elemanları üzerinde işlem yapabilmek için state' e bağlanması gereklidir. Bunun için yandaki kalıp kullanılabilir.

PRACTISE



Form

Ziyaretçiden ad, soyad, email, telefon bilgilerini alıp API a gönderen uygulamayı yapınız.

Telefon numaranızı giriniz

Ad

dfsdf

Soyad

sdfsd

Email

fsdfsf

Telefon

 !

Gönder



Formlar

```
const Form01 = () => {
  const [formData, setFormData] = useState({
    ad:"",
    soyad:"",
    eposta:"",
    telefon:""
  });

  const handleFormData = (e) => {
    const {name, value} = e.target;
    setFormData({...formData, [name]:value});
  };

  return (
    // form kodları
  );
};
```

- ◎ Form işlemlerinde form elementlarının sayısı arttığında her biri için state ve ve statelerin güncellenmesi için handle foknsiyonlarını tanımlamak gereksiz kod kalabılığına sebep olur.
- ◎ Bunu ortadan kaldırmak için tek state içinde tüm form elemanları object şeklinde saklanabilir.

PRACTISE



Formlar

Önceki yaptığınız
uygulamayı kodlama
olarak daha sade hale
getiriniz.

Telefon numaranızı giriniz

Ad

dfsdf

Soyad

sdfsd

Email

fsdfsf

Telefon

 !

Gönder



Form Validation

Form handling, form elemanlarının istenilen kriterlere uygun olup olmadığını kontrol etmek demektir. Form handling işlemleri manuel yöntem veya çeşitli 3th party library ler ile yapılabilir.



Form Validation (old style)

```
<Form onSubmit={handleSubmit}>
  <Input name="firstName"
    placeholder="Adınız"
    value={formData.firstName}
    onChange={(e) =>
      handleFormInput(e)}
  />
  <Input
    name="lastName"
    placeholder="Soyadınız"
    value={formData.lastName}
    onChange={(e) =>
      handleFormInput(e)}
  />
  <Button>Save</Button>
</Form>
```

```
const handleFormInput = (e) => {
  setFormData((prev) => {
    return {
      ...prev,
      [e.target.name]: e.target.value
    };
  });
}

const handleSubmit = (e) => {
  e.preventDefault();

  if( !formData.firstName){
    alert("Lütfen...");
    return;
  }
  ...
  alert("Tebrikler");
}
```



Formik & Yup

Form validation için kullanılan en popüler kütüphane **Formik** tir. Formik ile beraber validation scheme oluşturmak için **Yup** kütüphanesinden faydalанılır.



FORMIK

formik.org

YUP

github.com/jquense/yup



Formik & Yup

Formik yapısı 3 bölümden oluşur

initial values

Validation scheme

onSubmit



Formik & Yup

initial values

Form elemanlarına girilecek bilgilerin saklanacağı ve ilk değerlerinin ayarlandığı bölümdür.

```
const initialValues = {  
  email: "",  
  password: "",  
};
```



Formik & Yup

Validation scheme

Yup aracılığı ile form elemanları ile ilgili validation kuralları belirlenir.

```
const validationSchema = Yup.object({
  email: Yup.string().email().required("Please enter your email"),
  password: Yup.string().required("Please enter your password"),
});
```



Formik & Yup

onSubmit

Form da submit işlemi gerçekleştiğinde, eğer validation başarılı olursa bu fonksiyon çalıştırılır. Fonksiyona form elemanlarının değerleri bir object içinde gönderilir.

```
const onSubmit = (values) => {  
  ...  
}
```



Formik & Yup

Tanımlanan bu 3 bölüm **useFormik** hook u aracılığı ile bir araya getirilerek, formik validation ayarlanır.

```
const formik = useFormik({  
    initialValues,  
    validationSchema,  
    onSubmit,  
});
```



Formik & Yup

Form elemanları aşağıdaki şekilde tanımlanmalıdır.

Html5
validation
devre dışı
bırakılır

Name, value,
onChange
gibi gerekli
attribute lar
oluşturulur.

```
<Form noValidate onSubmit={formik.handleSubmit}>  
  <FormGroup>  
    <Input type="email"  
          {...formik.getFieldProps("email")}  
          invalid={!formik.errors.email}>  
    />  
    <FormFeedback>  
      {formik.errors.email}  
    </FormFeedback>  
  </FormGroup>  
</Form>
```

Formik
validation
devreye alınır

Hata
durumuna
göre input
renklenir

Hata mesajı
gösterilir.

PRACTISE



Formlar

Önceki yaptığınız
uygulamaya formik &
yup ile validation
ekleyiniz.

First Name !

Please enter your firstname

Last Name !

Please enter your lastname

Email !

Please enter your email

Phone !

Please enter your phone

Save



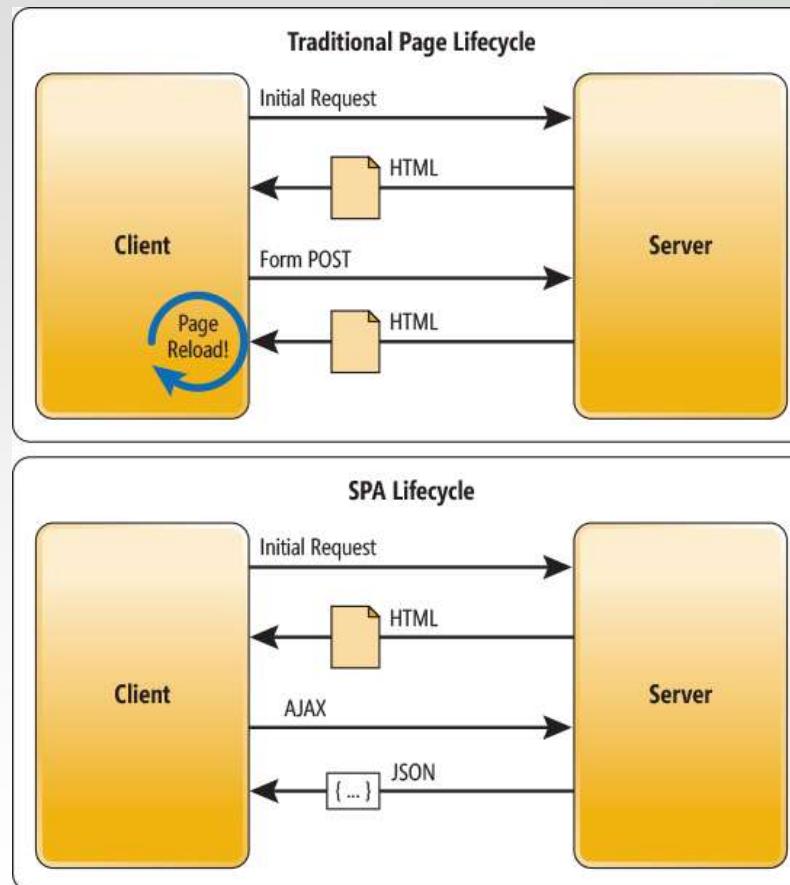
REACT

React Router DOM

- › MPA vs SPA
- › React Router DOM



MPA vs SPA





Router

Single Page Application (SPA) larda sanal sayfalar arasında geçişler yapmak için Router kullanılmalıdır. En popüler router kütüphanesi **react-router-dom** dur.

```
npm i react-router-dom
```

reactrouter.com



Router

```
import {BrowserRouter, Routes, Route}  
from "react-router-dom";
```

```
const App = () => {  
  return (  
    <BrowserRouter>  
      <Header/>  
      <Navbar/>  
      <Routes>  
        <Route path="/about" element={<About/>}/>  
        <Route path="/contact" element={<Contact/>}/>  
        <Route path="/" element={<Home/>}/>  
      </Routes>  
      <Footer/>  
    </BrowserRouter>  
  );  
};
```

Gerekli importlar yapılır

Tüm uygulama
BrowserRouter ile
sarmallanır

Her bir sayfa için Routes
içinde bir Route
tanımlanır. Her sayfada
sabit olan header, footer
gibi alanlar Routes
dışına konulur.



Router

```
import {Link} from "react-router-dom";  
  
const Test = () => {  
  return (  
    <nav>  
      <Link to="/">Home</Link>  
      <Link to="/about">About</Link>  
    </nav>  
  );  
};
```

Sayfalar arasında
bağlantı kurmak için a
tagı yerine **Link**
component kullanılır



Router

Bugüne kadar yapmış olduğumuz uygulamaları bir router aracılığı ile menü üzerinden ulaşılabilir hale getiriniz.



REACT

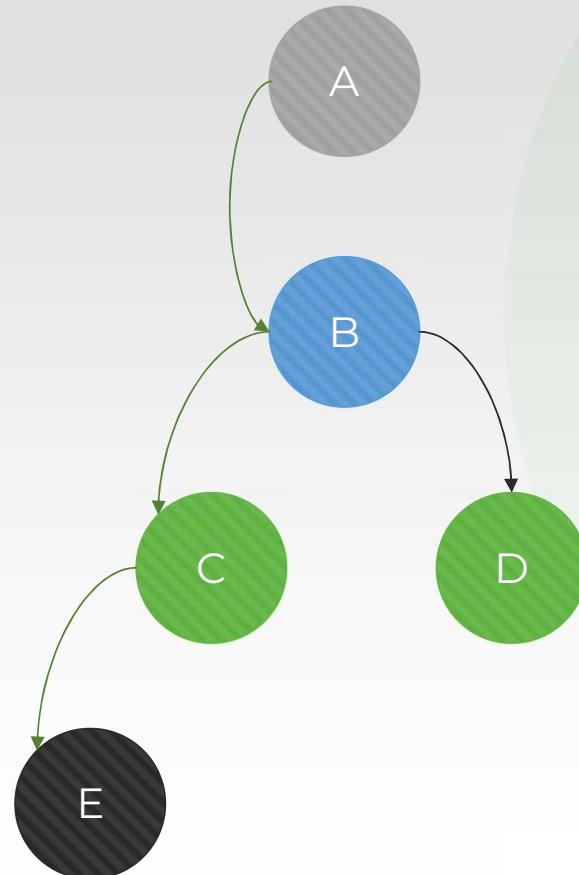
Contex API

- › Contex API Nedir?
- › Yapılandırma
- › Reducers



Context API

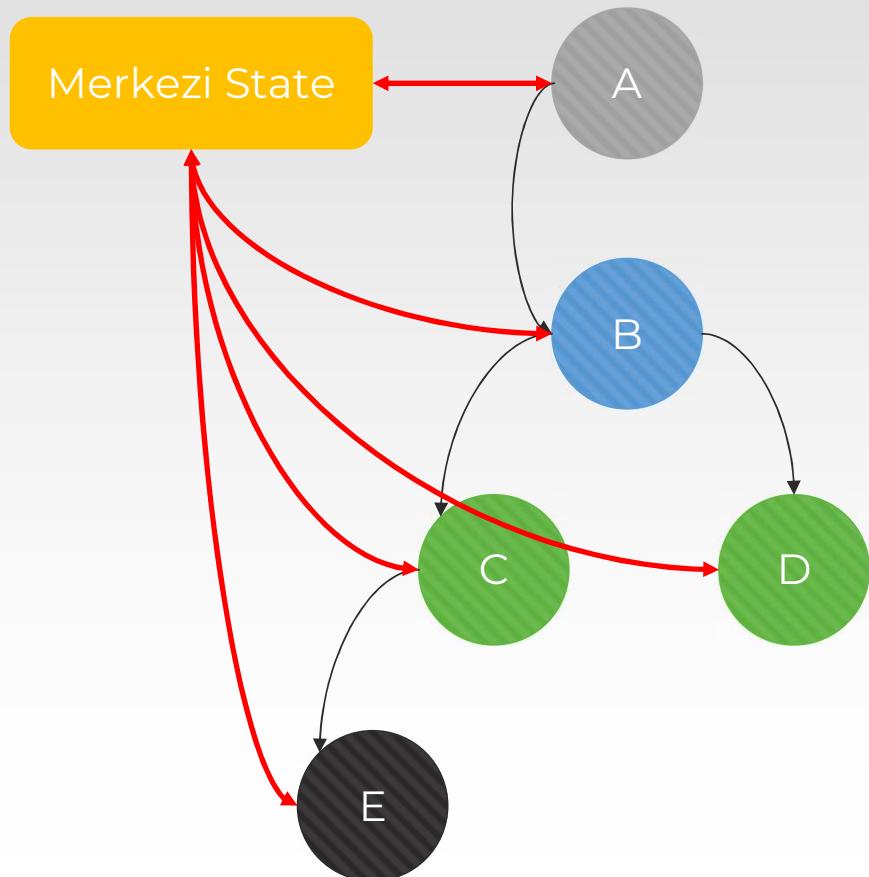
Prop
Drilling



Context
API



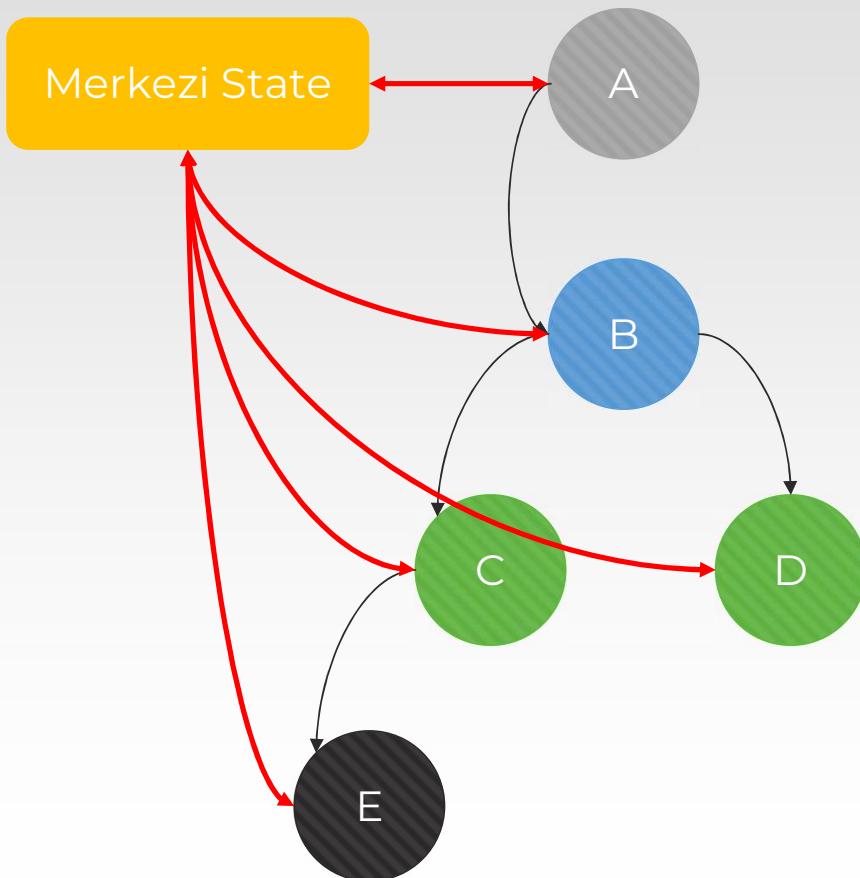
Context API



Aynı datanın birçok component tarafından ortak kullanıldığı durumlarda veya birbirine uzak componentler arasında kolaylıkla data transferi yapabilmek için merkezi bir state kullanılarak bu durum çözülür.



Context API



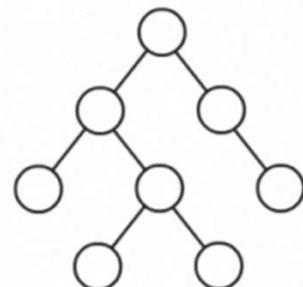
- Merkezi state için useContext hook u kullanılır.
- Alternatif olarak çok yaygın olarak harici **Redux** paketi bulunmaktadır.
- Redux paketi, react in context hook undan sonra redux-toolkit ile tekrar güçlenmiştir.

Context API



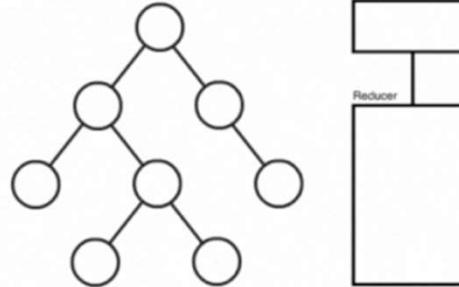
State Change In React vs State Change In React-Redux 😱

React 



- State change initiated
- State change

React With Redux 





Context API

```
import { createContext } from "react";
export const StoreContext = createContext();
```

```
<StoreContext.Provider value={{changeColor}}>
  <div className="App">
    ...
  </div>
</StoreContext.Provider>
```

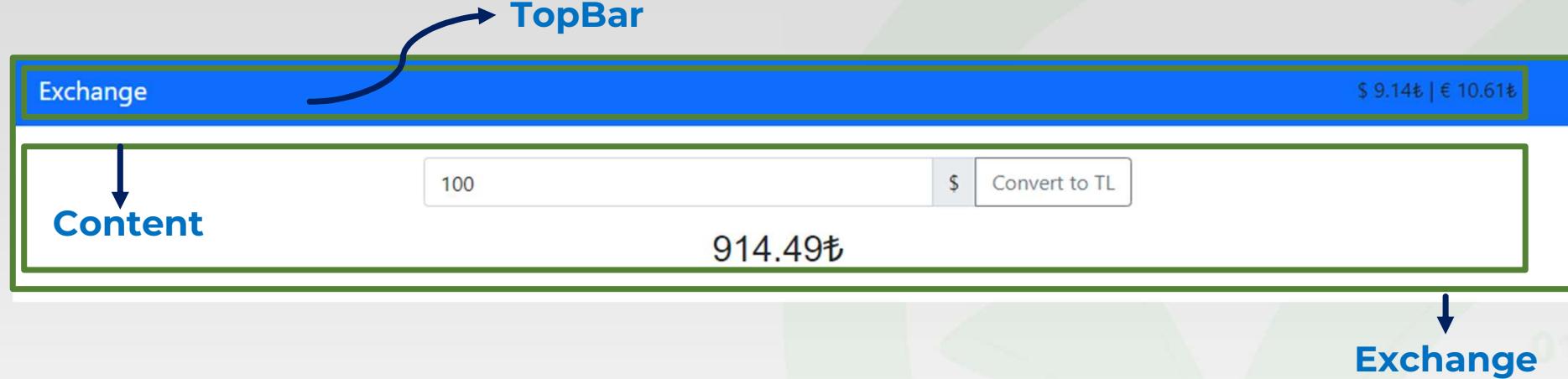
```
import {useContext} from "react";
const context = useContext(StoreContext);
context.changeColor();
```

- Öncelikle src içinde bir klasör (store) oluşturularak içine createContext ile içinde merkezi state i barındıracak context tanımlanır.
- Daha sonra App.js içinde oluşturulan context ile tüm uygulama sarmalanır ve value değeri ile her yerde kullanılacak state ler tanımlanır.
- Herhangi bir componentin içinde kullanırken useContext hook u kullanılır

PRACTISE



Merkezi State



- <https://api.frankfurter.app/latest?from=try> API dan exchange oranları çekilecek
- Bu oranlar tüm componentlerden erişebilir olacak
- TopBar da oranlar gösterilecek
- Content te ise USD – TL dönüştürücü yapılacak



Reducer Kullanımı

Fazla sayıda state in olduğu büyük projelerde bu state leri organize etmek, yönetilebilirliğini artırmak ve daha güvenli setter lar oluşturmak için **reducer** yapısı kullanılır.



Reducer Kullanımı

Reducer, 4 bölümden oluşur

Initial State

State te saklanacak alanlar ve ilk değerleri belirlenir.

Reducer

State i güncelleyen fonksiyonlardır.

Actions

Güncelleme işlemlerinde hangi reducer in çağrılacağını ve varsa state e gönderilecek datayı belirleyen nesnedir.

Types

Actions ve reducer bölgelerinde kullanılacak olan action types lar tanımlanır



Get State

Good morning
Set Message

useStore

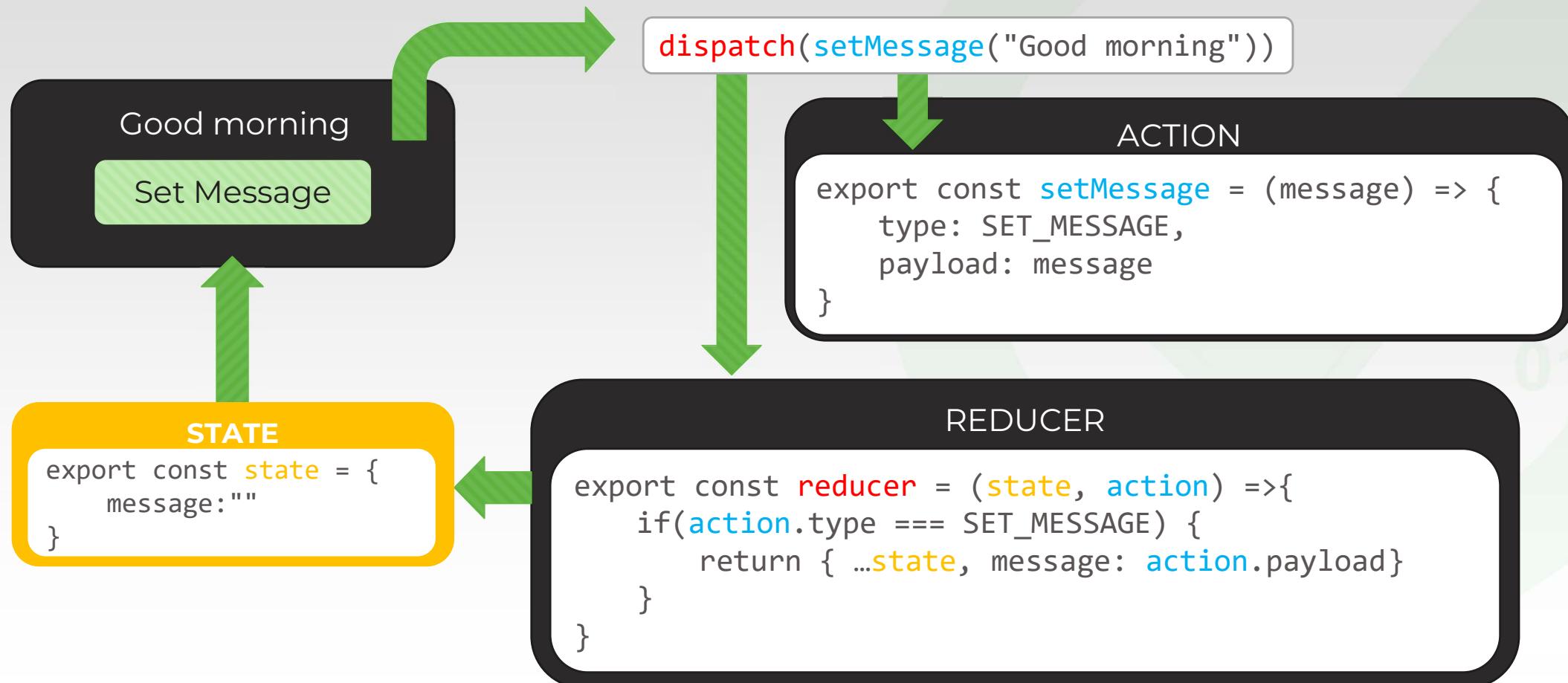
```
const state = useContext(StoreContext);  
const { message } = state;
```

STATE

```
export const state = {  
  message:  
}
```



Set State





UseReducer

Types: action ve reducer bölgümlerinde kullanılacak türler burada tanımlanır.

Initial State: Oluşturulacak merkezi state'in ilk değerlerinin verildiği bölümdür.

```
// types
export const ADD_NOTE = 'ADD_NOTE';
export const REMOVE_NOTE = 'REMOVE_NOTE';
```

```
//initial state
export const initialState = {
  notes: []
};
```



UseReducer

Actions:

Merkezi state i değiştirmek için Reducer' a gönderilecek olan object lerin oluşturulduğu bölümdür.

Bu object içinde genelde iki tane eleman olur. Bunlar:

- **type**: reducer da hangi bölümün çalıştırılacağını belirler.
- **payload**: state e yerleştirilecek veri

```
//actions
export const addNote = (note) => ({
  type: ADD_NOTE,
  payload: note
});

export const removeNote = (id) => ({
  type: REMOVE_NOTE,
  payload: id
});
```



UseReducer

Reducer:

Dispatch yoluyla gönderilen action lara göre ilgili bölümü çalıştırarak merkezi state güncellenir.

```
export const noteReducer = (state=initialState, action) => {
  if(action.type === ADD_NOTE){
    return{
      ...state, notes: [...state.notes, action.payload]
    }
  }
  else if(action.type === REMOVE_NOTE){
    return{
      ...state, notes: state.notes.filter( (note) =>{
        return note.id !== action.payload
      })
    }
  }
  return state;
}
```



UseReducer

- Projenin farklı component leri içinden merkezi state ten veri çekmek için merkezi state import edilir.
- useStore ile state çekilir ve state içindeki data kullanılır.

```
import { useStore } from "./store";

const Notes = () => {
  const [state] = useStore();
  const { notes } = state;

  return (
    <>
      {notes.map((note) => {
        return ( <Note note={note} /> );
      })}
    </>
  );
};
```

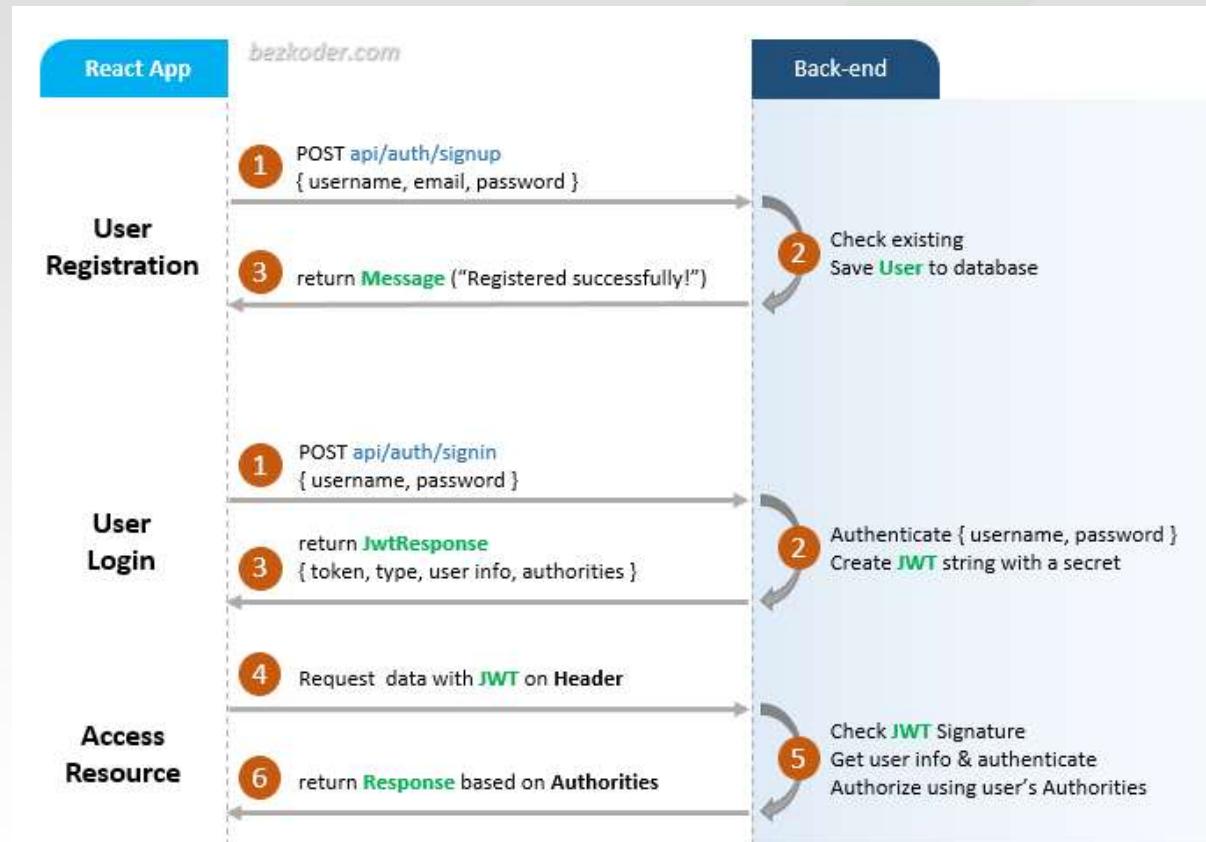


UseReducer

- Projenin farklı component lerinden merkezi state i değiştirmek için merkezi state ve ilgili action lar import edilir.
- Dispatch komutu ile action üzerinden state güncellenir.

```
import { removeNote } from './store/notes/notes-actions';
import { useStore } from './store';
const Note = (props) => {
  const {id, note} = props.note;
  const [state, dispatch] = useStore();
  const handleClick = (id) =>{
    dispatch(removeNote(id));
  }
  return (
    <div className="card-footer">
      <button onClick={e=> handleClick(id)}>Sil</button>
    </div>
  )
}
```

User Authentication

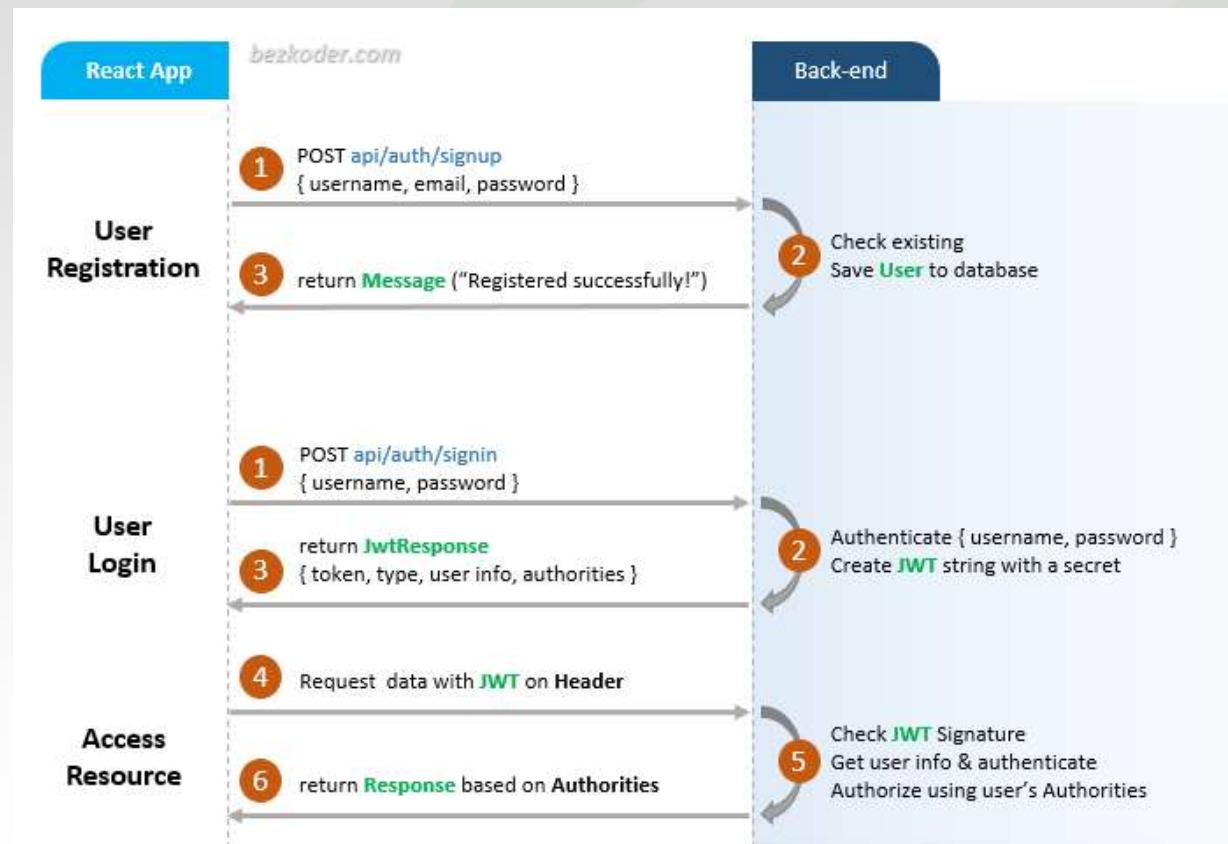


PRACTISE



Login

User Login işlemi yapılacak.
Kullanıcı giriş yaptıktan sonra kullanıcının bilgileri merkezi bir state te saklanarak diğer sayfalardan ulaşımı sağlanacak.





REACT

Redux-Toolkit

- Redux nedir?
- Redux-Toolkit nedir?
- Kurulum
- Slices
- Store



Redux nedir?

Uygulama state ini merkezileştirmek için kullanılan açık kaynak bir Javascript kütüphanesidir.



redux.js.org



Redux-Toolkit nedir?

Redux'ın, redux yapısını oluşturma konusunda kurumsal olarak kullanılmasını tavsiye ettiği yeni bir yaklaşım sunan en yeni kütüphanesidir. React ve Redux'ın merkezi state oluşturma konusundaki dağınık yaklaşımını baitleştirmiştir.

Redux Toolkit

Redux Toolkit is our official recommended approach for writing Redux logic. It wraps around the Redux core, and contains packages and functions that we think are essential for building a Redux app. Redux Toolkit builds in our suggested best practices, simplifies most Redux tasks, prevents common mistakes, and makes it easier to write Redux applications.

redux-toolkit.js.org

react-redux.js.org



Kurulum

```
npm install @reduxjs/toolkit react-redux
```

Redux'in yeni konsepti ile ilgili yapıları içerir

Redux'in, React ile birlikte çalışmasını sağlayan yapıları içerir



Slices

Slice, initial state, action ve reducerların tek bir yerden oluşturuldukları yapılardır. Action lar otomatik oluşturulur.

```
import { createSlice } from '@reduxjs/toolkit'

export const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0,
  },
  reducers: {
    increment: (state) => { state.value += 1 },
    decrement: (state) => { state.value -= 1 },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

export const { increment, decrement, incrementByAmount } = counterSlice.actions
export default counterSlice.reducer
```

Initial state

Reducers

Exporting actions
and reducers



Redux Store

Tüm uygulamaya açılacak olan merkezi state burada oluşturulur.

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './slices/counter-slice';

export default configureStore({
  reducer: {
    counter: counterReducer
  },
})
```

Store oluşturulur

Reducer lar eklenir

store.js



Redux Store

Redux store ile tüm uygulama sarmallanır

```
import App from './App'  
import store from './app/store'  
import { Provider } from 'react-redux'  
  
const root =  
ReactDOM.createRoot(document.getElementById('root'))  
  
root.render(  
  <Provider store={store}>  
    <App />  
  </Provider>  
)
```

index.js

Store import edilir

Provider import edilir

Provider ve store kullanılarak tüm uygulama sarmallanır



Redux Store

Oluşturulan store a erişim ve manipülasyon için **useSelector** ve **useDispatch** kullanılır

```
import { useSelector, useDispatch } from 'react-redux'  
import { decrement, increment } from './counterSlice'  
  
export function Counter() {  
  const count = useSelector((state) => state.counter.value)  
  const dispatch = useDispatch()  
  
  return (  
    <div>  
      <button onClick={() => dispatch(increment())}> + </button>  
      <span>{count}</span>  
      <button onClick={() => dispatch(decrement())}> - </button>  
    </div>  
  )  
}
```

PRACTISE



Redux-Toolkit

Darkmode ve dil seçeneklerinin olduğu bir uygulamayı, redux-toolkit ile yapınız.



REACT

Deployment

- Deployment nedir?
- Manuel deployment
- Automated deployment



Deployment

Geliştirilen uygulamaların hedef kitlenin ulaşabilmesi için internet üzerinden ulaşılabilir hale getirme süreçlerine denir

Manuel Deployment

Automated Deployment



Manuel Deployment

1

Hosting & Domain

2

npm run build

3

Transfer files

4

.htaccess for **Linux** servers
web.config for **Windows** servers



Automated Deployment

Proje source kodlarını bir otomasyon aracı ile build edip projeyi deploy eden sistemlerdir.



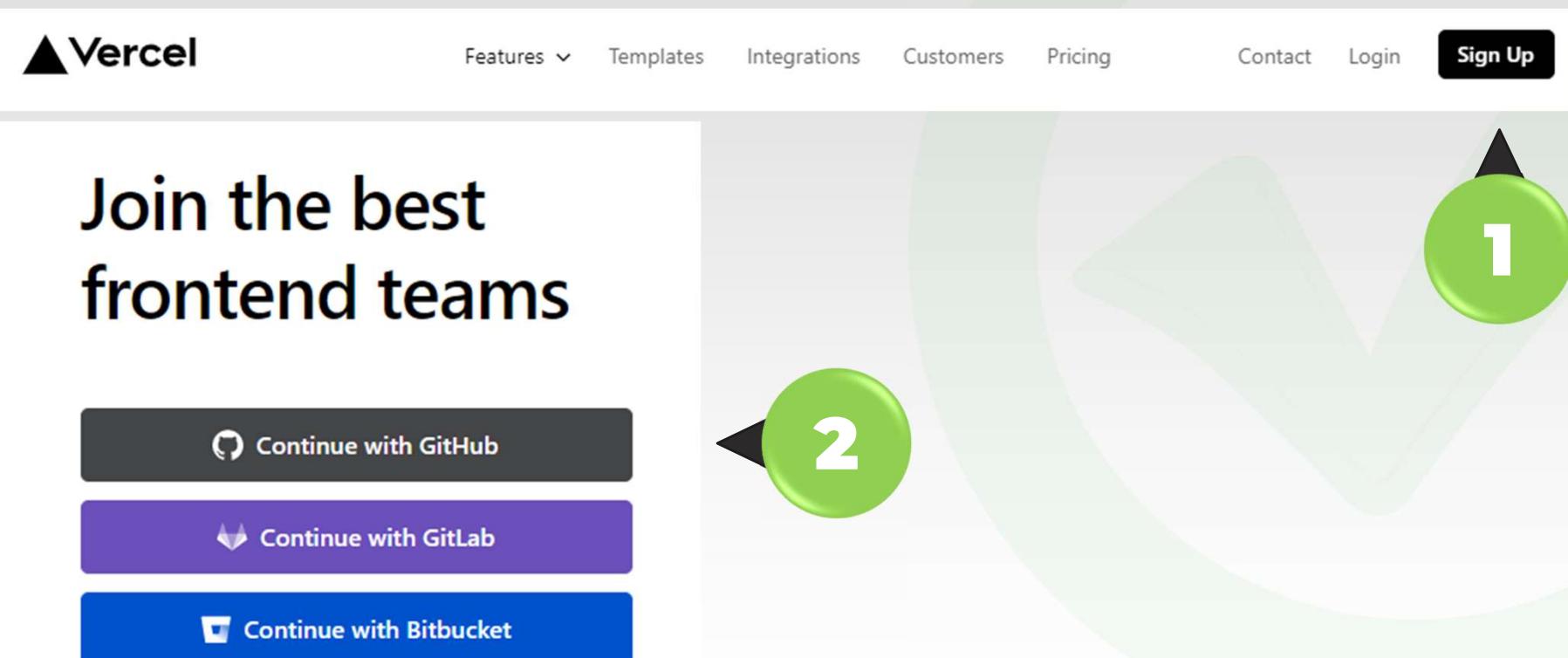
Github
Pages



SURGE



Deployment with Vercel



The image shows the Vercel sign-up page. At the top, there is a navigation bar with the Vercel logo, a search bar, and links for Features, Templates, Integrations, Customers, Pricing, Contact, Login, and Sign Up. The main content area features a large call-to-action button with the text "Join the best frontend teams". Below this are three buttons for connecting via GitHub, GitLab, or Bitbucket. A blue "Continue with Email →" link is located at the bottom. Two large green circular numbers are overlaid on the page: a "1" in the upper right corner pointing upwards, and a "2" in the lower left corner pointing towards the "Continue with Email" link.

▲Vercel

Features ▾ Templates Integrations Customers Pricing Contact Login Sign Up

Join the best frontend teams

Continue with GitHub

Continue with GitLab

Continue with Bitbucket

Continue with Email →

Deployment with Vercel



Overview Integrations Activity Domains Usage Settings

Search... Create a Team + New Project

Import Git Repository

ziya3435 Search...
longwrentalcars-fron... · 31d ago Import
bluerentalcars-frontend · 58d ago Import
utdbank-frontend · 58d ago Import
ziya3435 · 97d ago Import

Import Third-Party Git Repository →

Configure Project

PROJECT NAME: utdbank-frontend
FRAMEWORK PRESET: Create React App
ROOT DIRECTORY: ./

▶ Build and Output Settings
▶ Environment Variables

Deploy

3

4

5

```
graph TD; 1[Import Git Repository] --> 2[Configure Project]; 2 --> 3[+ New Project]; 3 --> 4[Import]; 4 --> 5[Deploy];
```