

React Hook Form

Az kod yazarak
hızlı etkili formlar oluşturabilmemizi sağlayan
başarılı bir kütüphanedir

<https://www.npmjs.com/package/react-hook-form>

<https://react-hook-form.com/>

React Hook Form

Hook odaklı mimariye sahiptir

`useForm` hooku form için gerekli tüm fonksiyonları ve state tanımlarını içerir

Manuel Form Yönetimini hatırlayalım

Form datası için **state** tanımlamalıyız

Form Input nesnelerine **change handler** fonksiyonları yazmalıyız

State değerini form input'una **value** olarak atamalıyız.



Form'a **submit handler** fonksiyonu eklemeliyiz.

Örnek projemizdeki iletişim formunu inceleyelim...

React Hook Form'da ise

useForm

- + State tanımlarını: **formData, errors**
- + **changeHandler** fonksiyonlarını
- + **Validasyon** ön tanımlarını ve fazlasını içermektedir



React Hook Form Kurulumu

External Library - Harici Kütüphane

> **npm install react-hook-form**

Form sayfamıza dahil etmek için ise

> **import { useForm } from "react-hook-form";**

Haydi örnek projemizdeki iletişim formuna dahil edelim...

React Hook Form Yapısı

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```



useForm içinden çektiğimiz fonksiyon & state'ler



useForm'a verdığımız ayarlar & konfigürasyonlar

React Hook Form: defaultValues

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

defaultValues

Form elemanlarının alacağı varsayılan - başlangıç değerlerini belirler

defaultValues konfigürasyonunu projemize uygulayalım

React Hook Form: register

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

register

Form alanı ile Input nesnesini ilişkilendirir.
Input'a state ve change handler fonksiyonu ekler.

```
<input  
  type="email"  
  {...register("email")}>  
</input>
```

React Hook Form: register

Manuel Form Yönetimi

```
<input  
  className="form-control"  
  type="email"  
  name="email"  
  value={formData.email}  
  onChange={inputChangeHandler}  
/>
```



React Hook Form

```
<input  
  className="form-control"  
  type="email"  
  { ...register("email") }  
/>
```

React Hook Form: register

```
register("email") => ({  
  name: "email",  
  onChange: changeHandler  
});
```



```
emailInputProps = {  
  name: "email",  
  onChange: changeHandler  
};
```

```
<input  
  type="email"  
  {...emailInputProps}  
/>
```



```
<input  
  type="email"  
  name="email"  
  onChange={changeHandler}  
/>
```

React Hook Form: register

```
<input  
  className="form-control"  
  type="email"  
  { ...register("email")}  
/>
```



```
<input  
  className="form-control"  
  type="email"  
  name="email"  
  onChange={inputChangeHandler}  
/>
```

React Hook Form: handleSubmit

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm{  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

handleSubmit

Formu submit etmemizi sağlar.

```
<form  
  onSubmit={handleSubmit(submitForm)}  
>  
  ...  
</form>
```

React Hook Form: handleSubmit

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

```
const submitForm = (formData) => {  
  console.log("Submitted: ", formData)  
}  
...  
<form  
onSubmit={handleSubmit(submitForm)}>  
...  
</form>
```

Form Validasyonunun Önemi

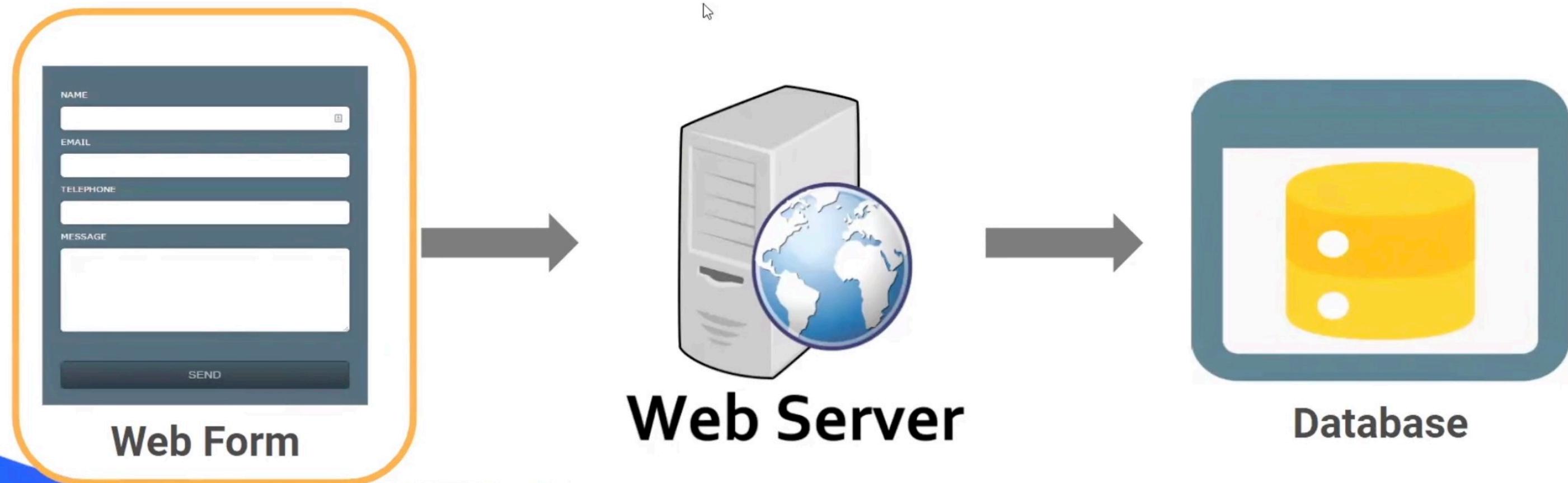
Öncelikle bilgisayar ve yazılımın temel işlevini tekrarlayalım



Datanın tutarlı ve geçerli (istedigimiz formatta) olması gerekmektedir.

Form Validasyonunun Önemi

Form Validasyonları kullanıcıyı geçerli data girmesi için yönlendirir. Böylece **hatalı** datanın **backend**'e gitmesini engellemiştir. Gereksiz data trafigini de engellemiştir.



Form Validasyonunun Önemi

Mesela kullanıcının e-posta bilgisi gerekiyorsa, girilen datanın içerisinde

[metin] @ [metin] . [metin]

Veya telefon numarası bilgisi girmesi gerekiyorsa 10 rakamdan oluşan bir data
girmelidir

[xxx xxx xxxx]

React Hook Form: Validasyonlar

React hook form da ise validasyon kuralları oluşturmak oldukça basittir.

register methodu 2. parametre olarak validasyon objesi alır.

```
register( [FormAlanAdı] , { Validasyon_Kuralları } )
```



React Hook Form: Validasyonlar

Örneğin bir form elemanına data girişini **zorunlu (* required)** yapmak istersek

```
<input  
    type="text"  
    { ...register( "isim", {  
        required: "İsim alanı boş bırakılamaz"  
    })  
}
```

React Hook Form: Validasyonlar

Hook form içerisinde birçok ön tanımlı validasyon bulunur.

required, min, max, minLength, maxLength gibi...



Dökümantasyonu inceleyelim:

<https://react-hook-form.com/get-started#Applyvalidation>

React Hook Form: Validasyonlar

Bazı validasyonlar sadece **kural** ve **hata mesajı** çifti formatında yazılırken:

required: "İsim alanı boş bırakılamaz."

Bazı validasyonlar ise obje olarak **kural: { value, message }** değerlerini alır.

```
minLength: {  
    value: 3,  
    message: "İsim alanı 3 karakterden az olamaz."  
}
```

React Hook Form: Validasyonlar

Bu öğrendiğimiz bilgileri kullanarak aşağıdaki validasyon kurallarını formumuza uygulayalım:

isim: zorunlu ve minimum 3 karakter uzunluğunda olabilir

e-posta: zorunlu

başlık: zorunlu ve maximum 20 karakter uzunluğunda olabilir

mesaj: zorunlu ve minimum 10 karakter uzunluğunda olabilir



React Hook Form: pattern

Bir diğer validasyon çeşidi: **pattern**

pattern ile **REGEX** ifadeleri kullanarak data formatı uyumluluğu test edilebilir.

```
{ pattern: {  
    value: /^[A-Za-z]+$/i,  
    message: "İsim alanına rakam veya özel karakter giremezsiniz"  
}  
}
```

REGEX ifadelerini test etmek ve öğrenmek için: <https://regexr.com/>

React Hook Form: pattern

Pattern'i kullanarak aşağıdaki validasyon kurallarını formumuza uygulayalım

isim: Sadece büyük, küçük harf ve boşluk karakteri kullanılabilir

e-posta: Geçerli bir e-posta formatı olmalıdır.

E-posta data formatını test eden **REGEX** kodu için internette arama yapabiliriz.

React Hook Form: validate

validate ile özel (**custom**) validasyonlar oluşturabiliriz.

```
validate: (age) => {
  if (age < 18) {
    return "Bu program maalesef yaş grubunuza uygun değil!"
  }
  return false;
}
```

React Hook Form: errors

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

errors

Validasyon kurallarından oluşan hata mesajı **errors** objesi içerisinde tutulur.

Örneğin **name** alanı ile ilgili hata mesajı

> **errors.name.message**

İçerisinden çekilebilir.

React Hook Form: errors

errors nesnesinin form içinde kullanımı

```
<input ... />

{ errors.name && (
  <div className="form-error">
    { errors.name.message }
  </div>
)
}
```

React Hook Form: `isValid`

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

isValid

Tüm form alanlarına girilen dataların geçerli olup olmadığı bilgisini verir.

Boolean { true || false }

React Hook Form: **isValid**

isValid değeri genellikle Submit butonunu disabled etmek için kullanılır

```
<button type="submit" disabled={ !isValid } >  
    Gönder  
</button>
```

React Hook Form: mode

```
const {  
  register,  
  handleSubmit,  
  formState: { errors, isValid },  
} = useForm({  
  defaultValues: {  
    name: "",  
    ...  
  },  
  mode: "all"  
});
```

mode

Mode konfigürasyonu ile validasyon işleminin ne zaman tetikleneceğini belirleriz.

- **onSubmit**: form submit edildiğinde
- **onBlur**: input pasif olduğunda
- **onChange**: her değişimde
- **onTouched**: mobil cihazda dokunma
- **all**: Hepsinde

Toastify

Toastify uygulama esnasında gerçekleşen işlemler hakkında kullanıcıyı bilgilendirmek için **bildirim (notification) popup** gösterimi sağlar.

npm > <https://www.npmjs.com/package/react-toastify>

Toastify Kurulumu

<https://fkhadra.github.io/react-toastify/installation>

> npm install react-toastify

Toastify Kurulumu

<https://fkhadra.github.io/react-toastify/installation>

> npm install react-toastify

```
import React from 'react';
import { ToastContainer, toast } from
'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

function App(){
  const notify = () => toast("Wow so easy !");

  return (
    <div>
      <button onClick={notify}>Notify !</button>
      <ToastContainer />
    </div>
  );
}
```

React-Toastify

GitHub  Search  

Getting Started 

Introduction

Installation

Migrate to v10 

Migrate to v9

Migrate to v8

Release notes

Usage 

Addons 

API Reference 

The playground

Toast Container

```
<ToastContainer
  position="top-right"
  autoClose={5000}
  hideProgressBar={false}
  newestOnTop={false}
  closeOnClick
  rtl={false}
  pauseOnFocusLoss
  draggable
  pauseOnHover
  theme="light"
  transition: Bounce,
  />
/* Same as */
<ToastContainer />
```

Toast Emitter

```
toast.success('🦄 Wow so easy!', {
  position: "top-right",
  autoClose: 5000,
  hideProgressBar: false,
  closeOnClick: true,
  pauseOnHover: true,
  draggable: true,
  progress: undefined,
  theme: "light",
  transition: Bounce,
});
```

Position

top-left top-right top-center bottom-left bottom-right

The playground

Features

Contribute

Contributors

Code Contributors

Financial Contributors

License

CSS Utility Class Yaklaşımını Kullanan Popüler Kütüphaneler [Top CSS Frameworks](#)

- TailwindCSS
- Bootstrap
- Bulma
- Materialize
- Foundation
- vb...

CSS Utility Class'ları

Tek bir css özelliğini kullanarak, tek bir amaca hizmet eden class'lar oluşturmayı baz alan yaklaşımındır.

- Class isimlendirme için harcanan zaman kaybını engeller.
- CSS dosyasının zamanla büyümeyi engeller.
- Özellik ekleme ve çıkarma sonrası -zamanla- oluşan css çöplüğünü engeller.
- Proje üretim hızını ve prototip yapma hızını artırır.
- Değişikliklerin daha güvenli yapılmasına yardımcı olur.

CSS Utility Class'ları

Geleneksel Yöntem

```
.header_title {  
    font-size: 1.75rem;  
    font-weight: bold;  
    color: white;  
}
```

```
.page_title {  
    font-size: 1.75rem;  
    font-weight: bold;  
    color: black;  
}
```

Utility Class

```
.bold {  
    font-weight: bold;  
}
```

```
.fs-1 {  
    font-size: 1.75rem;  
}
```

```
.text-white {  
    color: white;  
}
```

```
.text-black {  
    color: black;  
}
```



TailwindCSS Library <https://tailwindcss.com/>

- İçerisinde her bir CSS kuralı için class tanımı vardır.
- HTML'de, sadece class isimleri ile tüm sayfa tasarıımı yapılabilmektedir.
- Özelleştirme (customization) yapmak çok daha kolaydır.
- Flexible: diğer css kütüphanelerine göre daha esnektir.

TailwindCSS Library kullanımı

1. Projeye tailwindcss kütüphanesi yüklenir.

```
› npm install -D tailwindcss
```

2. Initilaze edilir.

```
› npx tailwindcss init
```

TailwindCSS Library kullanımı

- Aşağıdaki sarı kutudakileri `tailwindcss.config` dosyasına eklenir.

```
module.exports = {  
  content: ["./src/**/*.{js,jsx,ts,tsx}"],  
  theme: {  
    extend: {},  
  },  
  plugins: [],  
};
```

- Tailwind katmanlarını `@tailwind directive`'ini kullanarak projemize `index.css` dosyasına eklenir.

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

TailwindCSS Library kullanımı

- tailwindcss'de tanımlı class'ları projede kullan.

Vanilla CSS	TailwindCSS
<pre><h1 className="title">Selam Dünya!</h1> .title { font-size: 1.25rem; color: #3245A5; font-weight: bold; text-align: center; }</pre>	<pre><h1 className="text-xl text-blue-600 font-bold text-center"> Selam Dünya! </h1></pre>

TailwindCSS Class'ları

```
Padding: p{t|r|b|l|x|y}-{size}
p-0.5 => padding: 0.125rem /* 2px */
pt-1 => padding-top: 0.25rem /* 4px */
px-2 => padding-left: 0.5rem; padding-right: 0.5rem /* 8px */
```

```
Border-radius:
rounded-none => border-radius: 0px;
rounded-sm => border-radius: 0.125rem; /* 2px */
rounded => border-radius: 0.25rem; /* 4px */
```

```
color:
text-red-500 => color: rgb(239 68 68); Aa
text-red-600 => color: rgb(220 38 38); Aa
text-red-900 => color: rgb(127 29 29); Aa
```

TailwindCSS Arbitrary Values

```
p-[3px] => padding: 3px
```

```
pt-[3px] => padding-top: 3px
```

```
rounded-[12px] => border-radius: 12px
```

```
text-[#dddddd] => color: #dddddd;
```

TailwindCSS Customization

```
module.exports = {  
  content: ["./src/**/*.{js,jsx,ts,tsx}"],  
  theme: {  
    extend: {  
      colors: {  
        'wit-blue': '#243c5a',  
      },  
    },  
    plugins: [],  
  },  
};
```

Kullanım:

Font rengi için: [text-wit-blue](#)
Background için: [bg-wit-blue](#)

```
module.exports = {  
  content: ["./src/**/*.{js,jsx,ts,tsx}"],  
  theme: {  
    extend: {  
      padding: {  
        'lg': '5px',  
      },  
    },  
    plugins: [],  
  },  
};
```

Kullanım:

padding-top için: [pt-lg](#)

TailwindCSS **@Apply** Directive

@apply directive'i ile uzun tailwind classlarını tekrar tekrar yazmak yerine bir class içinde tanımlayabiliriz.

```
/* Input */
.btn {
  @apply font-bold py-2 px-4 rounded !important;
}
```

(4) Date-Fns Library <https://date-fns.org/>

Genellikle JS uygulamalarına, içerisinde birçok yardımcı method barındıran bir **date kütüphanesi** eklenir. Date-Fns de moment kütüphanesi gibi popüler date kütüphanelerinden biridir.

Date kütüphanesi kullanırız çünkü;

- **Date**, JS'de bulunan non-primitive bir veri tipidir.
- İçinde **tarih** ve **saat** bilgisini saklar: Tue Mar 21 2023 01:42:29 GMT+0300 (GMT+03:00)
- Year, Month, Day, Hour, Minute, Second, Milliseconds, Timezone gibi bilgiler vardır.
- Ama, **Date objesi ile çalışmak zordur.**
- Yeterli yardımcı metodlar olmadığı için, ayrı ayrı bir çok metodlar yazmak gereklidir.
- Date kütüphanelerinde bu metodlar hazır gelir.

Date-Fns Library Kullanımı

1. Projeye date-fns kütüphanesi yüklenir.

```
> npm install date-fns
```

2. Kullanılmak istenen metodlar import edilir ve kullanılır.

```
import { differenceInDays, formatDistanceToNow } from 'date-fns';
import { tr } from 'date-fns/locale';
```

Date-Fns Library Faydalı Metodlar

formatDistanceToNow(date, options?): verilen tarihten şu ana kadar olan farkı metin olarak verir:

- 1 dakika
- Yaklaşık 1 ay v.b.

Distance to now	Result
0 secs ... 5 secs	less than 5 seconds
5 secs ... 10 secs	less than 10 seconds
10 secs ... 20 secs	less than 20 seconds

```
//Bugünün 1 Ocak 2001 olduğunu var sayalım. 2 Temmuz 2000 arasındaki fark?  
const result = formatDistanceToNow( new Date(2000, 6, 2) ) //=> '6 months'
```

Suffix(son ek) ekleyerek gösterim:

```
//Bugünün 1 Ocak 2015 olduğunu var sayalım. 1 Ocak 2016 arasındaki fark?  
const result = formatDistanceToNow( new Date(2016, 0, 1), {addSuffix: true} )  
//=> 'in about 1 year'
```

Türkçe gösterelim:

```
//Bugünün 1 Ocak 2001 olduğunu var sayalım. 2 Temmuz 2000 arasındaki fark?  
const result = formatDistanceToNow( new Date(2000, 6, 2), {locale: tr} )  
//=> '6 months'
```

Date-Fns Library Faydalı Metodlar

format(date, formatString, options?): verilen tarihi istenen formata dönüştürür.

```
const result = format(new Date(2014, 1, 11), 'MM/dd/yyyy') //=> '02/11/2014'
```

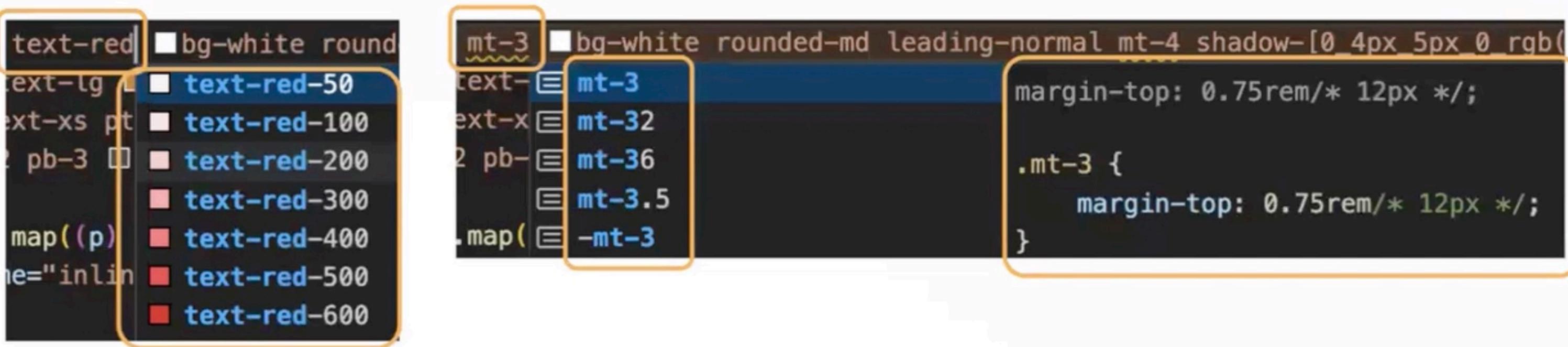
Türkçe gösterelim:

```
//localization bilgisini import etmeyi unutmayalım!
import { tr } from 'date-fns/locale'
const result = format(new Date(2024, 1, 20), "MMMM yyyy EEEE", { locale: tr })
//=> 'Şubat 2024 Salı'
```

Extra: TailwindCSS VS Code eklentisi

VSCode üzerine tailwind classlarını kullanırken bize öneri listesi (**intellisense**) getirerek yardımcı olacak bir eklenti kullanabiliriz.

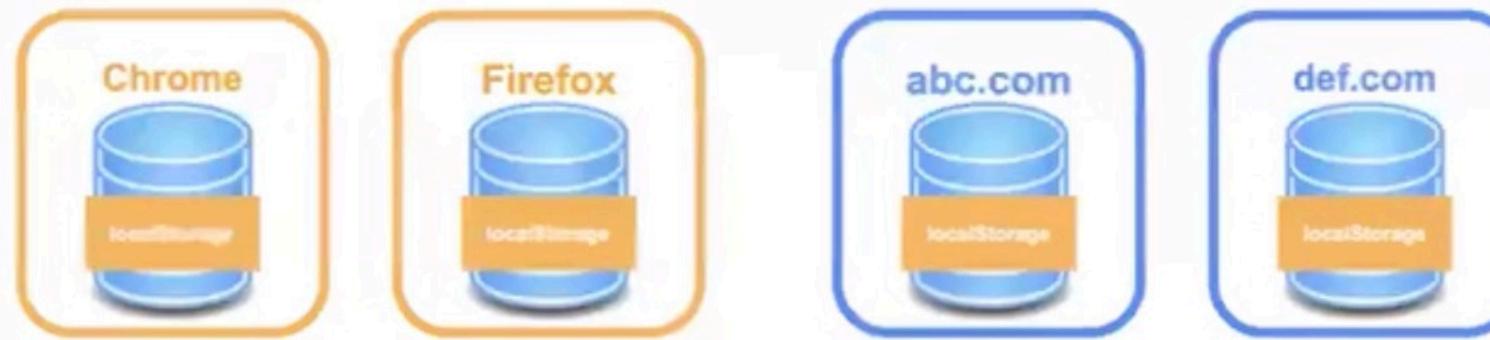
<https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss>



LocalStorage Nedir?

Web uygulamalarının **browser**'da (Chrome, Firefox, Safari, ..) saklanan veri tabanıdır.

- Veri Tabanı browser kapatılsa da silinmez.
- Her **browser** ve **domain** (URL)'in kendine has DB'i vardır.



- Temel görevi kullanıcının tercihlerini (datalarını) hatırlamaktır:
 - **Dil, tema, son giriş yapan kullanıcı bilgileri** v.b.
 - **Şifre saklanmaz**: güvenlik sorunu oluşturur!

LocalStorage Nedir?

- Datalar (**key** : **value**) çifti formatında saklanır
- localStorage datayı metin (**string**) formatında saklar.
- Kapatisitesi 5 mb'dır. Browser'dan browser'a farklılık gösterebilir.
- **DevTools** içinden görüntülenebilir

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar under 'Storage', 'Local storage' is expanded, and the entry for 'https://docs.google.com' is selected. This selection highlights the row in the main table below. The table lists various key-value pairs stored in the local storage of this origin. The columns are 'Key' and 'Value'. Some values are truncated with an ellipsis (...).

Key	Value
kx-e-n	1
punch-e-v	1
ritz-e-n	1
punchv-e-f	1
kx-e-v	1
PeopleStackExperiments:[[34,1],"113894550444353168526"]]	{"keyPath": "[[34,1], \"113894550444353168526\"]", "ln...
punchv-e-v	1
PeopleStackExperiments:[["N",N,"N",1],1]	{"keyPath": "[\"N\",N,\"N\",1],1", "ln..."}
ritz-e-f	1
PeopleStackExperiments:[["N",N,"N",1],4]	{"keyPath": "[\"N\",N,\"N\",1],4", "ln..."}
docsOfflineFrameApi-n	1

LocalStorage Metodları

1. `localStorage.setItem(key,value)` : Bir veriyi Key&value ikilisi olarak kaydetmek için kullanılır.
2. `localStorage.getItem(key)` : Key'e karşılık gelen veriyi almak için kullanılır.
3. `localStorage.removeItem(key)` : Key'i -ve bağlı olan veriyi- localStorage'dan silmek için kullanılır.
4. `localStorage.clear()` : Tüm kayıtlı verileri silmek için kullanılır.

Dikkat: localStorage tüm datayı **string** olarak kaydeder:

- `localStorage.setItem("sayi", 7);` => "7"
- `localStorage.setItem("bool", true);` => "true"
- `localStorage.setItem("obje", { name: "ali" });` => [Object object]

Local Storage'da Complex Veri Tiplerini Saklama

Complex veri tiplerini string'e çevirerek saklayabiliriz.

JSON kütüphanesinindeki .stringfy(complexVeri) ve .parse(stringVeri) metodlarını kullanabiliriz.

- **JSON.stringfy(complexVeri)** : complex veri tipini string formatına dönüştürür.

```
const user = { name: "Ali", surname: "Metin" };
JSON.stringify(user) // '{"name":"Ali","surname":"metin"}'
```

- **JSON.parse(kayıtlıVeri)** : String olarak kaydettiğimiz veriyi tekrar complex veri tipine dönüştürmek için kullanılır.

```
const stringData = localStorage.getItem(key);
JSON.parse(stringData) // { name: "Ali", surname: "Metin" }
```

Stateful Logic

- React uygulamaları içinde iş akışları genellikle **state** değerlerine bağlıdır.
 - Örn: butona basınca ekranın görünür/görünmez olması
 - Veya butona basınca sayfanın temasının dark / light olması.
- Bunun gibi state değerine bağlı iş akışlarına **stateful logic** denir.

Custom Hooks

- React uygulaması içinde tekrar tekrar kullanılan bir **stateful logic**, ayrı bir yerde **React hook** yapısında yazılır.
- Böylece her ihtiyacımız olduğunda bu **custom hook**'u çağrıarak kullanabiliriz.
- **Bu kullanım kod tekrarı yapmamızın önüne geçer.**
- Custom hook içerisinde React hookları kullanılabılır: **useState, useEffect**
- Custom hook içerisinde başka custom hook'lar da kullanılabilir.

Custom Hooks

- Custom hook bir **fonksiyondur** ve React hookları gibi (useState, useEffect) **use** prefix'i ile başlar.

```
const useCounter = (start, min = 0, max = 100) => {
  const [counter, setCounter] = useState(start);

  const arttir = () => setCounter(counter + 1);
  const azalt = () => setCounter(counter - 1);

  useEffect(() => {
    if (counter < min) {
      setCounter(min);
    } else if (counter > max) {
      setCounter(max);
    }
  }, [counter]);

  return [counter, arttir, azalt];
};
```

return değerine dikkat!

- Component içerisinde React hookları gibi kullanılır.

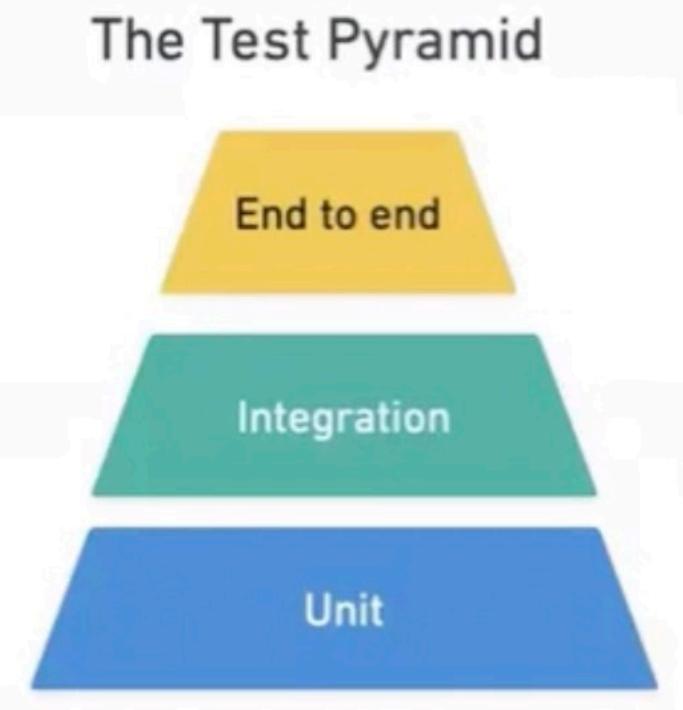
```
const CounterDisplay = (props) => {
  const {initialCount} = props;
  const [counter, fnArtir, fnAzalt] = useCounter(initialCount, 0, 100);

  return (
    <div></div>
  )
};
```

Test Türleri

4 farklı test vardır:

1. Statik testler ([ESLint](#))
2. Unit testler
3. **Entegrasyon testleri**
4. E2E testleri ([Cypress.io](#))





React Testing Library <https://testing-library.com/>

React uygulamaları içinde kolaylıkla **Integration Test**ler yazabilmemizi sağlayan bir test kütüphaneleri paketidir.

- **DOM Testing Library, Jest, Jest-dom** gibi kütüphaneler uyarlanarak React uygulamalarında etkili testler hazırlanması sağlanmıştır.
- **Create React App** ile oluşturulan projelerde kurulu ve bazı konfigürasyonlar yapılmış olarak gelir.
- **Vite** gibi başka toollar ile oluşturulan React uygulamalarında ise sonradan kurulması gereklidir.

React Testing Library **Kullanımı**

1. Projeye yüklenir.

```
▶ npm i -D @testing-library/react@13
```

2. Konfigurasyon dosyası oluşturulur.
 - a. src klasörünün altında “**setupTest.js**” dosyası oluşturulur.
 - b. dosyaya “**import '@testing-library/jest-dom';**” satırı eklenir.
 3. Testleri yazacağımız test dosyası oluşturulur.
 - a. srx klasörünün altında “**__tests__**” adında bir klasör oluştururuz.
 - b. test dosyamızı bu klasörde “**{component}.test.js**” adıyla oluştururuz.
- Örn: App.test.js

React Testing Library Kullanımı

4. Testlerimizi yazarız.

```
import { render } from "@testing-library/react";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

test("Uygulama başarıyla başlatıldı!", () => {
  render(
    <BrowserRouter>
      <App />
    </BrowserRouter>
  );
});
```

5. Testleri çalıştırırız.

```
[> npm run test
```

Extra: Axios Problemi Çözümü

Axios kütüphanesi test aşamasında bir hataya sebep olabilir.

Bunu engellemek için **package.json** içerisinde aşağıdaki ayar eklenmeli:

```
"jest": {  
  "moduleNameMapper": {  
    "axios": "axios/dist/node/axios.cjs"  
  }  
},
```

Component Testi: render

`render(...)` method'u ile bir component, ekrana yerleştirildiği gibi çalıştırılır. Bu sırada test arayüzünde ekran outputunu göremeyiz.

1. Öncelikle, render metodunu test dosyamıza import ederiz.

```
import { render } from "@testing-library/react";
```

2. component'imizi render metodunun içine ekleriz. (Eğer, component prop bekliyor ise onları da ekeleriz)

```
test("Ana Sayfa Component Testi!", () => {
  render(
    <AnaSayfa />
  );
});
```

Component Testi: render

`render(...)` method'u ile bir component, ekrana yerleştirildiği gibi çalıştırılır. Bu sırada test arayüzünde ekran outputunu göremeyiz.

1. Öncelikle, render metodunu test dosyamıza import ederiz.

```
import { render } from "@testing-library/react";
```

2. component'imizi render metodunun içine ekleriz. (Eğer, component prop bekliyor ise onları da ekeleriz)

```
test("Ana Sayfa Component Testi!", () => {
  render(
    <AnaSayfa />
  );
});
```

Component Testi: screen

screen ile render olan componentimizdeki nesnelere erişip onları seçebilir, çeşitli özelliklerini ve değerlerini kontrol edebiliriz.

1. Öncelikle, screen'i test dosyamıza import ederiz.

```
import { render, screen } from "@testing-library/react";
```

2. istediğimiz nesneyi yakalarız.

```
<h2 data-testid="anasayfa-title">{title}</h2>
```

```
render(<AnaSayfa />);
```

```
const titleH2 = screen.getByTestId("anasayfa-title");
```

Testin 3 Aşaması: **Arrange-Act-Assert**

Her test case'i 3A kuralına göre yazılmalıdır:

- **Arrange** aşamasında, testin çalışması için gerekli tüm ayarları yapıyoruz.
- **Act** aşamasında, test edilecek işlemi gerçekleştiriyoruz.
- **Assert** aşamasında ise, işlemin sonucunun beklediğimiz gibi olup olmadığını kontrol ediyoruz.

```
test("Ana Sayfa Component Testi!", () => {
  // Arrange
  render(
    <AnaSayfa />
  );

  // Act
  const titleH2 = screen.getByTestId("anasayfa-title");

  // Assert
  expect(titleH2).toHaveTextContent("ReactJS Kütüphanesi");
});
```

Component Testi: queries

<https://testing-library.com/docs/react-testing-library/cheatsheet>

Query Types

	No Match	1 Match	1+ Match	Await?
getBy	throw	return	throw	No
findBy	throw	return	throw	Yes
queryBy	null	return	throw	No
getAllBy	throw	array	array	No
findAllBy	throw	array	array	Yes
queryAllBy	[]	array	array	No

Queries

- ByRole
- ByLabelText
- ByPlaceholderText
- ByText
- ByDisplayValue
- ByAltText
- ByTitle
- ByTestId

Examples

- **ByLabelText** find by label or aria-label text content
 - getByLabelText
 - queryByLabelText
 - getAllByLabelText
 - queryAllByLabelText
 - findByLabelText
 - findAllByLabelText
- **ByPlaceholderText** find by input placeholder value
 - getByPlaceholderText
 - queryByPlaceholderText
 - getAllByPlaceholderText
 - queryAllByPlaceholderText
 - findByPlaceholderText
 - findAllByPlaceholderText

Component Testi: assertionlar

Assertionlar için **jest** ve **jest-dom** kütüphaneleri kullanılır. Yapısı:

expect(SOMETHING).method(SOMEVALUE);

```
// Assert
expect(titleH2).toHaveTextContent("ReactJS Kütüphanesi");
```

Tüm assertion methodları için kaynak: <https://github.com/testing-library/jest-dom#tobevisible>

Sık kullanılan assertion ifadeleri:

```
// Assertions
expect(counterVal).toHaveTextContent("9");
expect(submitBtn).toBeDisabled();
expect(imgLogo).not.toBeVisible();
expect(containerDiv).toHaveClass("dark");
expect(nameInput).toHaveValue("ali");
```

Extra: screen.debug()

render edilen component'in yapısını **console**'da görmek için **screen.debug()** komutunu kullanabiliriz.

```
// ...
expect(titleH2).toHaveTextContent("ReactJS Kütüphanesi");
screen.debug();
});
```



```
<body>
<div>
<div
```

Kullanıcı Etkileşimleri Testi: fireEvent

fireEvent ile JS Event'lerini([click](#), [change](#), [submit](#), [select](#), [scroll](#), [copy](#), [paste](#) v.b.) tetikleyebiliriz.

Tüm listeye buradan bakabiliriz:

<https://github.com/testing-library/dom-testing-library/blob/main/src/event-map.js>

1. Önce test dosyamıza import ediyoruz:

```
import { fireEvent, render, screen } from "@testing-library/react";
```

2. sonra istediğimiz event'i -yakaladığımız nesne üzerinde- tetikleriz.

```
const titleInput = screen.getByTestId("title-input");
const submitBtn = screen.getByTestId("submit-btn");

fireEvent.change(titleInput, { target: { value: "Yeni başlık" } });
fireEvent.click(submitBtn);
```

Kullanıcı Etkileşimleri Testi: **userEvent**

fireEvent birebir JS event tetiklerken **userEvent** daha çok kullanıcının browser deneyimine yakın bir şekilde olayları([click](#), [dblClick](#), [type](#), [keyboard](#), [select v.b.](#)) tetikler.

Tüm listeye buradan bakabiliriz:

<https://testing-library.com/docs/user-event/v13/>

1. Önce test dosyamıza import ediyoruz:

```
import userEvent from "@testing-library/user-event";
```

2. sonra istediğimiz event'i -yakaladığımız nesne üzerinde- tetikleriz.

```
const titleInput = screen.getByTestId("title-input");
const submitBtn = screen.getByTestId("submit-btn");

userEvent.type(titleInput, "Yeni başlık");
userEvent.click(submitBtn);
```