

Kadir ÇAPKIN 21360859023

NodeJS ile Web programlama Raporu Hafta – 10

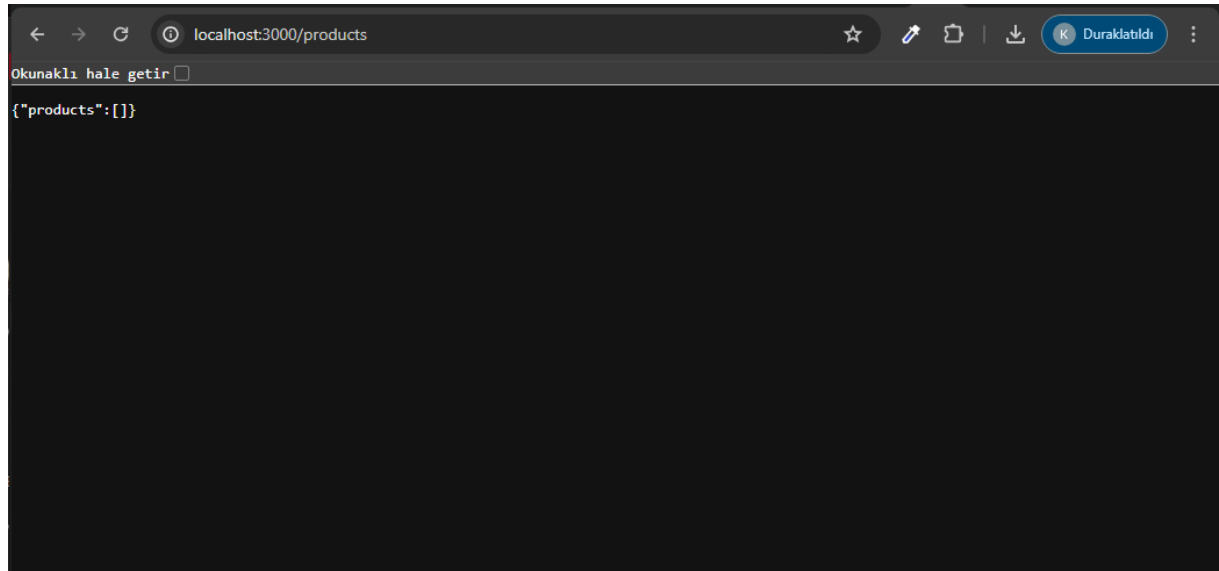
Weather app – 5

---

Şimdi amacımız daha önce yazdığımız hava durumu api isteklerimizi kendi sitemizde kullanıp mevcut hava durumunu kendi sitemizde görüntülemek olacak. Bunun için öncelikle app.js içerisinde yazdığımız weather router'ının altına test amaçlı bir router yazalım.

```
app.get('/products', (req, res) => {  
  res.send({  
    products: []  
  })  
})
```

Bu router sayfa içeriği olarak boş bir array içeren bir nesne yollar. Çıktısı şu şekildedir:



Bu sayfa şu an statik bir sayfadır sadece sabit boş bir array yollar. Eğer biz dinamik bir product kullanmak istersek query string kullanmamız gerekir. Bu query stringin çalışma mantığı şu şekilde işler. Örnek vermek gerekirse bir e ticaret sitesinde olduğunuzu düşünün televizyonlar linkine tıkladığınızda size satışta olan tüm televizyonlar listelenir. Eğer ki bu televizyonlardan birine tıklarsak televizyonun ayrıntılı bilgilerine ulaşırız ve bu ürünün linki koyulurken dinamik olarak konur. Bu sayede eğer bir ürüne tıkladığımız şöyle bir url yönlendirmesi olabilir localhost:3000/tv/products/3 -> 3 burada 3 id'ye sahip televizyondur. Şimdi bu ürünün ayrıntılı sayfasını hazırlamak istediğimizde ise bu ürüne id'si üzerinden dinamik olarak url üzerinden kolay bir şekilde ulaşabiliriz. Aynı zamanda ulaştıktan sonra da ürünle ilgili farklı sorgular yapabiliriz. Bu sorgular ise dinamik olarak query string sayesinde yapılabilir. Şimdi kendi kodumuz üzerinden bunu incelersek öncelikli olarak req.query'inin ne çıktısı verdiğine bakmak için console yazdıralım.

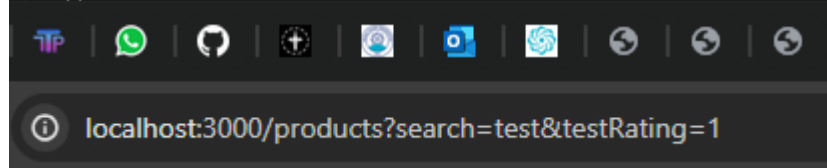
```
app.get('/products',(req,res)=>{
  console.log(req.query)
  res.send({
    products:[]
  })
})
```

Bunu yapıp sayfaya geldiğimizde şu sonuç gelir.

```
Listening on port 3000
{}
```

Şu an bu dizinin boş gelmesinin sebebi her products url'i üzerinde query stringe uygun bir sorgu yapmadığımız içindir. Şimdi aynı kodu çalıştırıp url'e bu sefer localhost:3000/products?search=test&testRating=1 konsolumuzda çıkan sonuca bakalım.

```
Listening on port 3000
{ search: 'test', testRating: '1' }
```



Görüldüğü gibi query sorgumuz consolda bastırıldı. Bu query sorguları anahtar,değer mantığına göre çalışır. Burada anahtar search ve testRating iken 'test' ve '1' ise değerlerdir. Şimdi konsolda bastırmış olduğumuz req.query değeri bir nesne döndürdüğü için bunun attribute'larına ulaşabiliriz.

```
app.get('/products',(req,res)=>{
  console.log(req.query);
  console.log("search value:",req.query.search);
  console.log("testRating value:",req.query.testRating);
  res.send({
    products:[]
  })
})
```

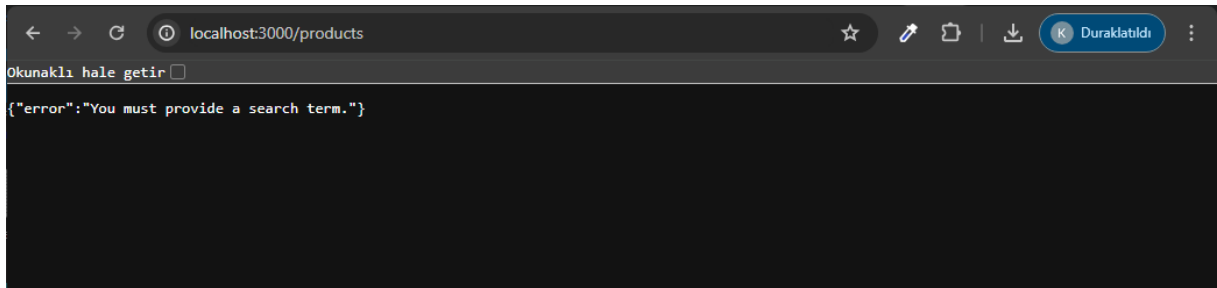
Bu kodu tekrar çalıştırsak karşımıza şu sonuç çıkar.

```
Listening on port 3000
{ search: 'test', testRating: '1' }
search value: test
testRating value: 1
[]
```

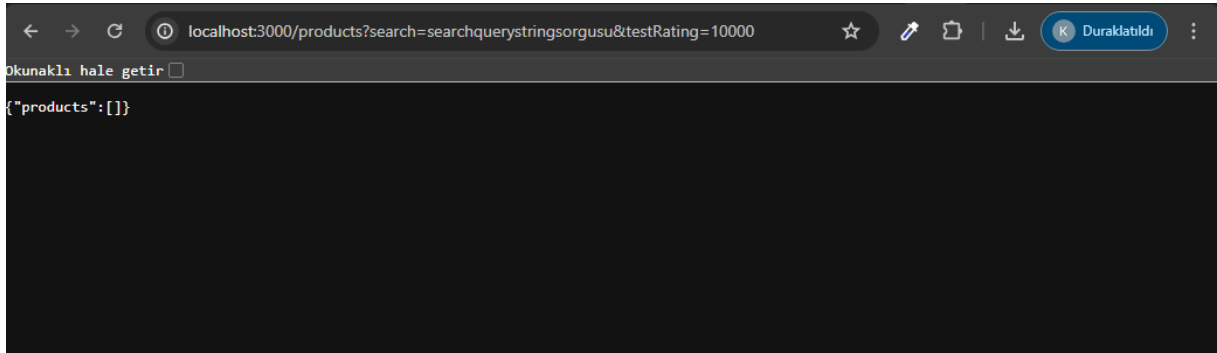
Bazı durumlarda query string kullanımını zorunlu hale getirebiliriz. Örneğin eğer bir url için query string kullanılmadıysa hata vermesi sağlarız. Kullanıldıysa zaten mevcut bilgiyi bize döndürür. Kodumuza koşul ekleyerek düzenlersek:

```
app.get('/products',(req,res)=>{
  /*
  console.log(req.query);*/
  if(!req.query.search){
    res.send({
      error:"You must provide a search term."
    })
  }
  console.log("search value:",req.query.search);
  console.log("testRating value:",req.query.testRating);
  res.send({
    products:[]
  })
})
```

Şimdi bunun testini yapalım. Öncelikle routerimize her hangi bir sorgu yapmadan (query string kullanmadan) ulaştığımızda karşımıza çıkacak ekran görüntüsü şu şekildedir.



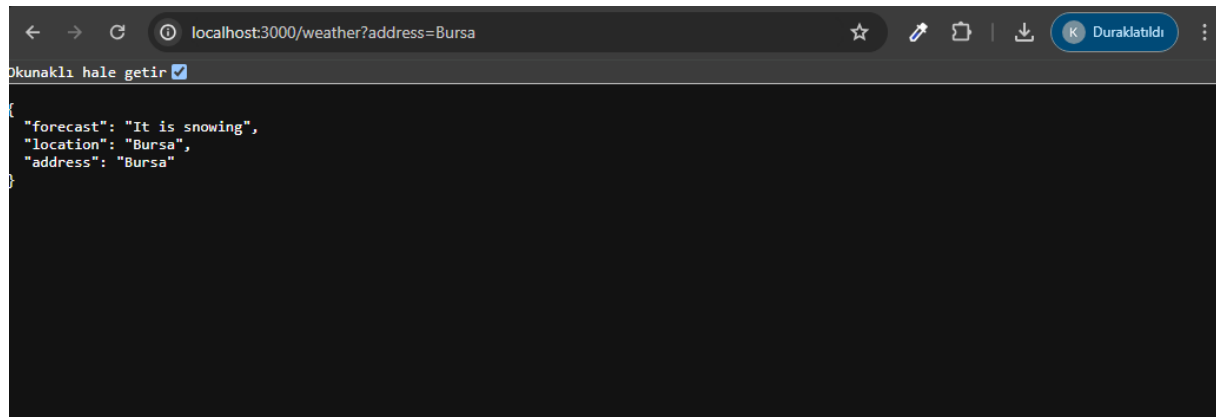
Şimdi ise rastgele bir sorgu kullandığımızda karşımıza gelecek ekran görüntüsü şu şekildedir.



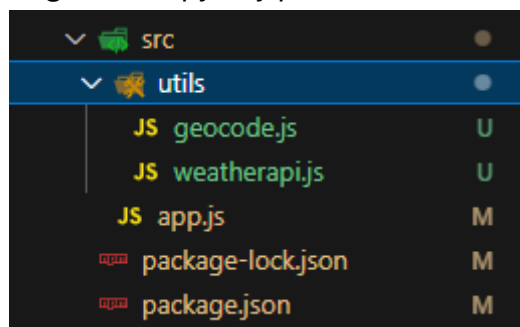
Şimdi products router'ı üzerinde yaptığımız işlemi /weather router'ı içerisinde kullanalım. Örnek olarak eğer query adresimiz varsa test amaçlı boş bir nesne yerine belirli bilgileri yollayalım.

```
app.get('/weather',(req,res)=>{
  if(!req.query.address){
    return res.send({
      error:"You must provide an adress"
    })
  }
  res.send({
    forecast:"It is snowing",
    location:"Bursa",
    address:req.query.address
  })
});
```

Şimdi bunu yaptıktan sonra localhost:3000/weather?address=Bursa yazıp çalıştıralım. Karşımıza yolladığımız bilgiler nesne gözükür. Ekran görüntüsü ise şu şekildedir.



Şimdi daha önce yazmış olduğumuz utils klasörü altındaki geocode.js ve weatherapi.js dosyalarını kullanacağız. Öncelikle bu dosyaları kullanabilmek için utils klasörünü doğrudan kopyalayıp web-server/src 'ye kopyalıyoruz.



Bu kodlar içerisindeki request modülü kullanabilmek için konsol üzerinde npm i request yazarak modülü yükleyelim. Daha sonrasında ise gerekli modülleri app.js içerisine bu şekilde dahil edelim.

```
const geocode = require('./utils/geocode');
const weatherapi = require('./utils/weatherapi');
```

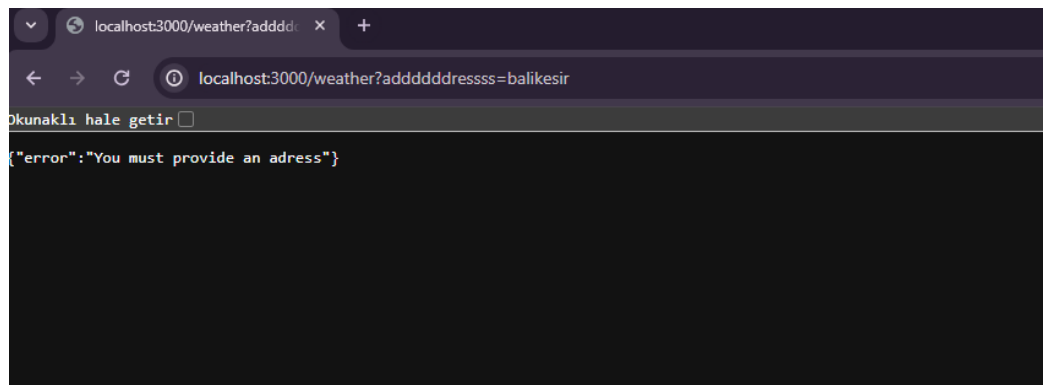
Modülleri dahil ettikten sonra yazdığımız “/weather” router’ini eklediğimiz modülle değiştirmemiz gerekiyor. Bunu da şu şekilde yapabiliriz.

```
if(!req.query.address){
  return res.send({
    error:"You must provide an adress"
  })
}

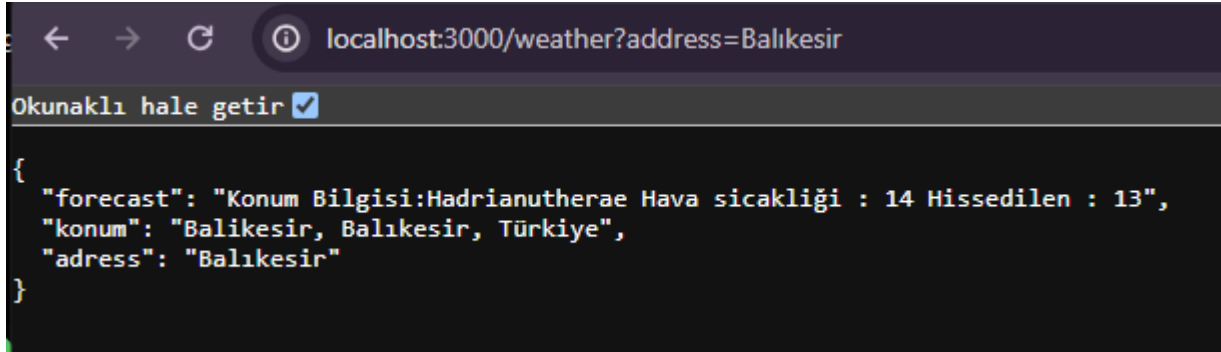
geocode(req.query.address,(error,{boylam,enlem,konum})=>{
  if(error){
    return res.send({error})
  }
  weatherapi(enlem,boylam,(error,forecastData)=>{
    if(error){
      return res.send({error})
    }

    res.send({
      forecast:forecastData,
      konum,
      adress:req.query.address
    })
  })
})
})
```

Bu kodu yazdığımızda /weather url i üzerinde öncelikle bizden bir query string ile sorgu yapmamızı yapmadığımız halde adres girmemiz gerektiğini söyleyen bir hata mesajı döndürür. Eğer bir sorgu girersek, bu sorgu daha önce yazdığımız geocode modülünden elde ettiğimiz fonksiyon kullanılır. Bu fonksiyon ilk parametre olarak bir konum bilgisi istiyor , ikinci parametre olarak ise bir callback fonksiyon ile bize girilen konumun enlem,boylam ve konum bilgisini döndürüyordu. Daha sonrasında ise bu callback fonksiyonundan aldığımız bilgiyi diğer bir yazdığımız modül olan weatherapi’ye göndererek enlem ve boylam bilgisi bilinen bir yerin hava durumu bilgisi,konum bilgisi ve adres bilgisini bize döndürüyor. Kodumuzu bu şekilde düzenledikten sonra çalıştırırsak ve denersek karşımıza sorguladığımız şehrin konum bilgisinin gelmesini bekleriz.Öncelikle hata durumunu test etmek için yanlış girelim. Çıktımız:



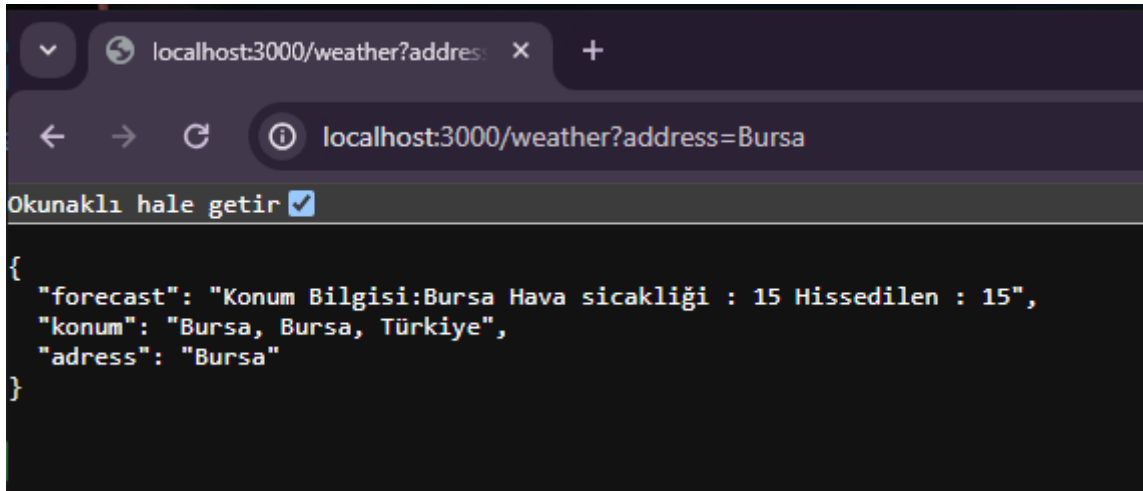
Ardından balıkesir ve bursa için sırasıyla doğru bir şekilde query girelim. Karşımıza çıkan sonuç şu şekildedir.



```
localhost:3000/weather?address=Balıkesir

Okunaklı hale getir ✓

{
  "forecast": "Konum Bilgisi:Hadrianutherae Hava sıcaklığı : 14 Hissedilen : 13",
  "konum": "Balıkesir, Balıkesir, Türkiye",
  "adress": "Balıkesir"
}
```



```
localhost:3000/weather?address=Bursa

Okunaklı hale getir ✓

{
  "forecast": "Konum Bilgisi:Bursa Hava sıcaklığı : 15 Hissedilen : 15",
  "konum": "Bursa, Bursa, Türkiye",
  "adress": "Bursa"
}
```

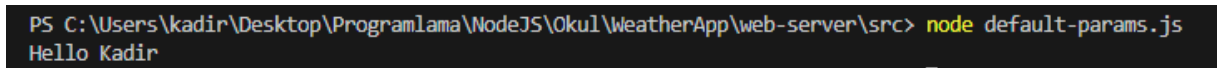
Başarılı bir şekilde çalışıyor.

Şimdi web-server altında default-params.js adlı yeni bir dosya açalım. Bu dosyanın içeriğine şunları ekleyelim.

```
const greeter = (name) =>{
  console.log("Hello "+name)
}

greeter("Kadir")
```

Bu fonksiyonu çalıştırdığımızda karşımıza şu sonuç çıkmaktadır.



```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Okul\WeatherApp\web-server\src> node default-params.js
Hello Kadir
```

Şimdi bu durumda eğer fonksiyonumuz çağırılırken bir parametre girilmeseydi, sonuç olarak undefined değer alırdık. Bunun önüne geçmek istersek doğrudan fonksiyonumuz tanımlanırken bir değişken için default olarak değer atayabiliriz. Kodumuzu bu şekilde değiştirirsek.

```
const greeter = (name='defaultValue',age) =>{
  console.log("Hello "+name)
}

greeter("Kadir")
greeter()
```

Şimdi bu kodu hem parametrelili hem de parametresiz çağırdığımızda çıktımız şu şekilde olur.

```
PS C:\Users\kadir\Desktop\Programlama\NodeJS\Okul\WeatherApp\web-server\src> node default-params.js
● Hello Kadir
Hello defaultValue
```

Görüldüğü gibi öncelikle parametre girdiğimizde default değer yerine öncelik girilen parametre olur. Eğer ki parametre girilmezse tanımladığımız default parametre değeri devreye girer.

Şimdi eğer tekrar app.js dosyamızı çalıştırıp url'de doğrudan bir adres girmek yerine şu adresi yazarsak localhost:3000/weather?address=! uygulamamız doğrudan çalışmayı durdurur. Bunun sebebi uygulamamız ! konumundan bir enlem ,boylam ve konum bilgisi bulmaya çalışıyor

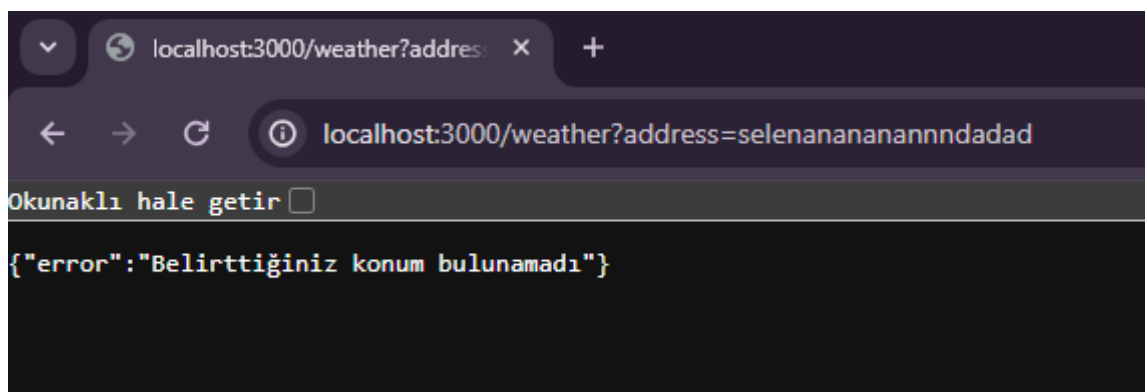
```
geocode(req.query.address, (error, {boylam, enlem, konum})
```

Bunun önüne geçebilmek için yukarıda yazdığımız kodu şu şekilde düzeltmemiz gerekir.

```
geocode(req.query.address, (error, {boylam, enlem, konum}={}))
```

Bunu yaptığımızda eğer ki mevcut query sonucu bir boylam, enlem ve konum bulunmadıysa doğrudan uygulamayı durdurmak yerine boş bir nesne döndürür. Bu sayede aşağıdaki geocode içerisinde kod bloğu çalışır hale gelir ve ekranda bunu alırız.

```
else if (body.features.length == 0) {
  callback("Belirttiğiniz konum bulunamadı", undefined)
} else {
```

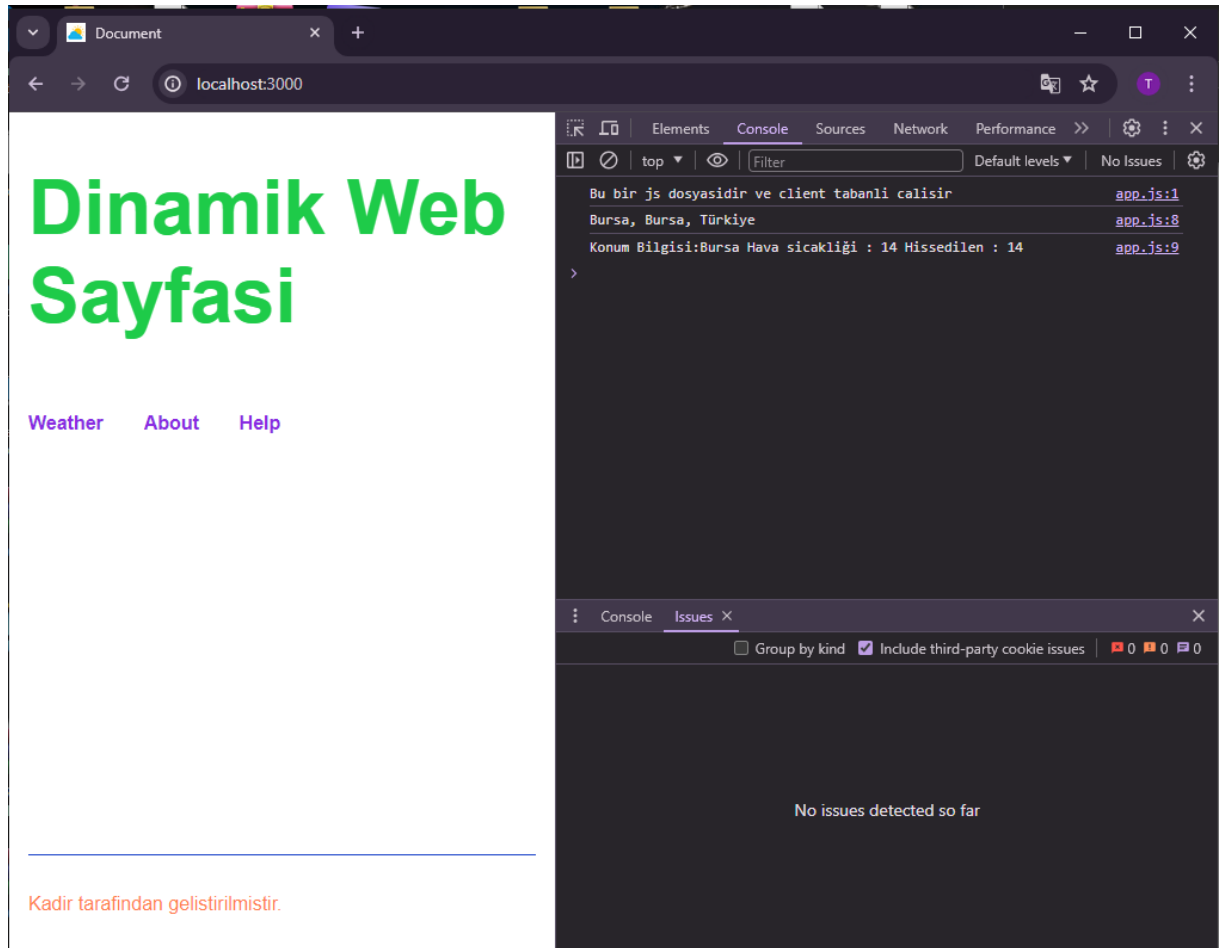


Şimdi yapacağımız işlem client tarafında bu bilgileri alıp kullanabilmek. Bu yüzden fetch API kullanacağız. Fetch verileri normalde node.js parçası değildir fakat modern tarayıcılar

bunu destekler. Şimdi public klasörü altında client tarafı için olan js dosyası altında app.js dosyamıza gelip şu kodu yazalım.

```
fetch("http://localhost:3000/weather?address=bursa").then((response)=>{
  response.json().then((data)=>{
    if(data.error){
      console.log(data.error)
    }else{
      console.log(data.konum);
      console.log(data.forecast)
    }
  })
})
```

Bu app.js(client) normalde views klasörü altında bütün .hbs uzantılı dosyalarda head kısmına eklendi bunu test etmek amaçlı localhost:3000 yazıp chrome üzerinde console bakarsak karşımıza şu çıkar.



Görüldüğü gibi burada yaptığımız işlem aslında sunucu tarafında yazmış olduğumuz veri çekme kodunu client tarafından çekme işlemiydi. Şimdi bu sunucudan veri alma işlevini ön yüzde biraz daha işlevli hale getirmek adına index.hbs dosyamızı düzenleyelim.



```
<div class = "main-content">
  <h1>{{>header}}</h1>
  <form id="weatherForm">

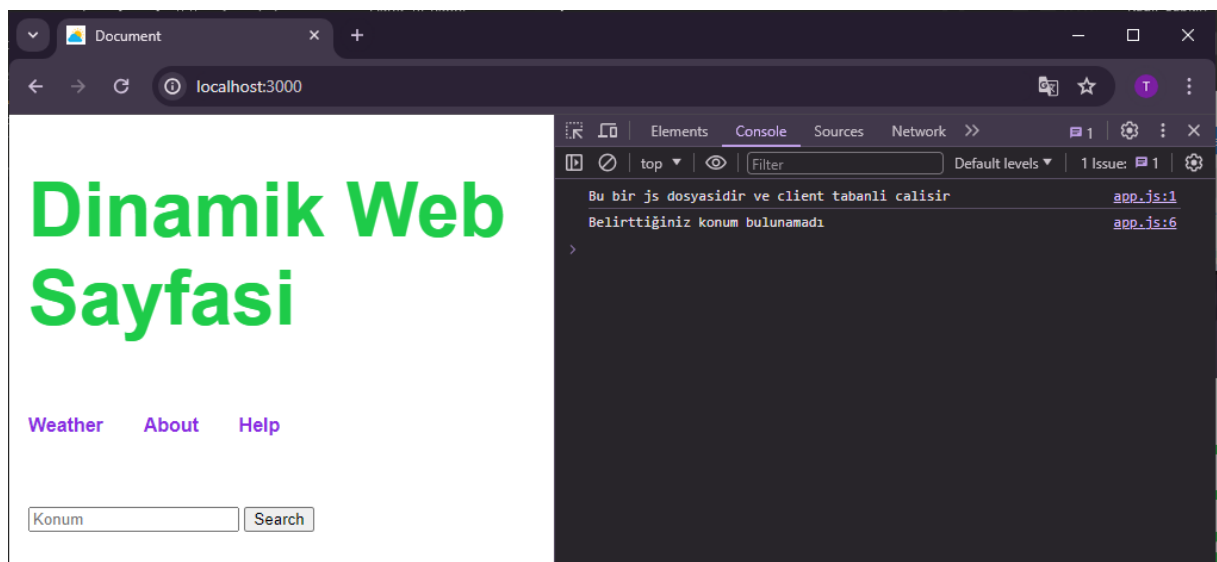
    <input placeholder="Konum">
    <button type="submit">Search</button>

  </form>
</div>
```

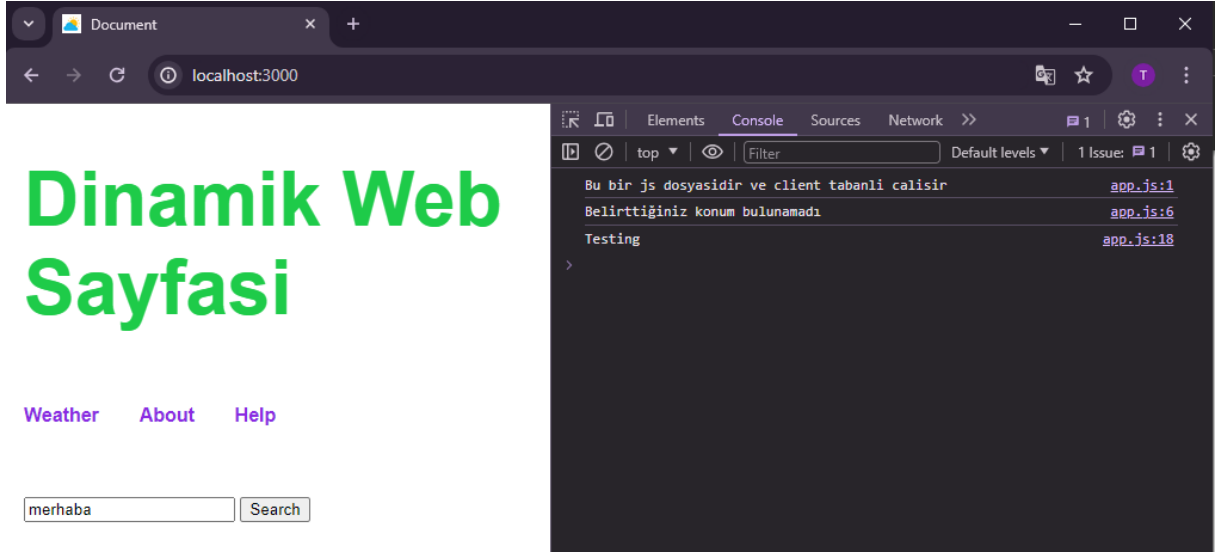
Bir form bilgisi ekledik bu form içerisinde bir buton ve input. Şimdi bu input bilgisini client tarafındaki app.js dosyasında alabilmek için tekrar app.js dosyamıza gidip şu kodları yazalım.

```
document.addEventListener("DOMContentLoaded", () => {
  const weatherForm = document.querySelector("#weatherForm");
  weatherForm.addEventListener("submit", (event) => {
    event.preventDefault(); // Formun varsayılan davranışını engeller
    console.log("Testing");
  });
});
```

Burada öncelikli olarak DOM eventinin yüklenip yüklenmediğini kontrol ettim. Aksi takdirde anlamadığım şekilde hata aldım. Ardından yazdığımız formun gönderilip gönderilmediği yani butona tıklanıp tıklanmadığını dinleyen bir event yazdım. Event.preventDefault() yazmamın sebebi formun varsayılan bir eventi vardır bunu engellemek içindir. Eğer bu butona tıklanırsa tarayıcı konsolunda testing yazar. Ekran görüntümüz şu şekilde olması gerekiyor.



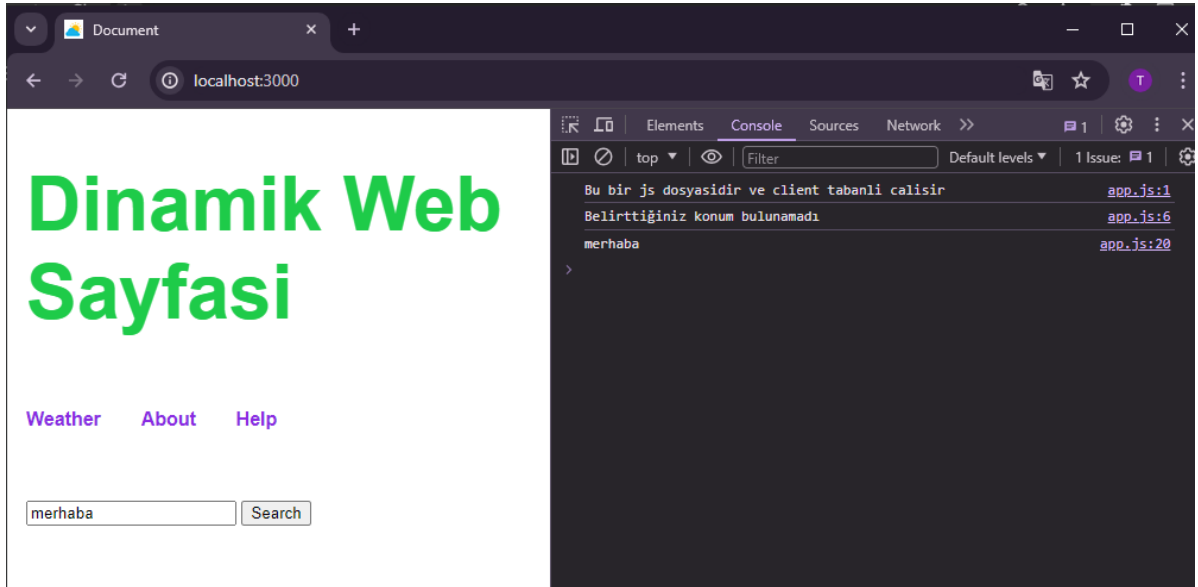
Şimdi tıklayalım.



Görüldüğü gibi submit olayı başarılı bir şekilde client tarafında dinleniyor. Şimdi input içerisine yazıp gönderdiğimiz stringin ön yüze gelip gelmediğine bakmak için kodumuza input elementinin değerini de ekleyelim.

```
document.addEventListener("DOMContentLoaded", () => {
  const weatherForm = document.querySelector("#weatherForm");
  const search = document.querySelector("input");
  weatherForm.addEventListener("submit", (event) => {
    event.preventDefault(); // Formun varsayılan davranışını engeller
    const location = search.value
    console.log(location);
  });
});
```

Kodumuzu test ettiğimiz karşımıza çıkacak sonuç şu şekildedir.



Bu da başarılı çalıştığının göre şunu yapabiliriz. Input içerisinden aldığımız konum bilgisini client tarafında fetch'in parametresi olarak yollayıp fetchapi sayesinde

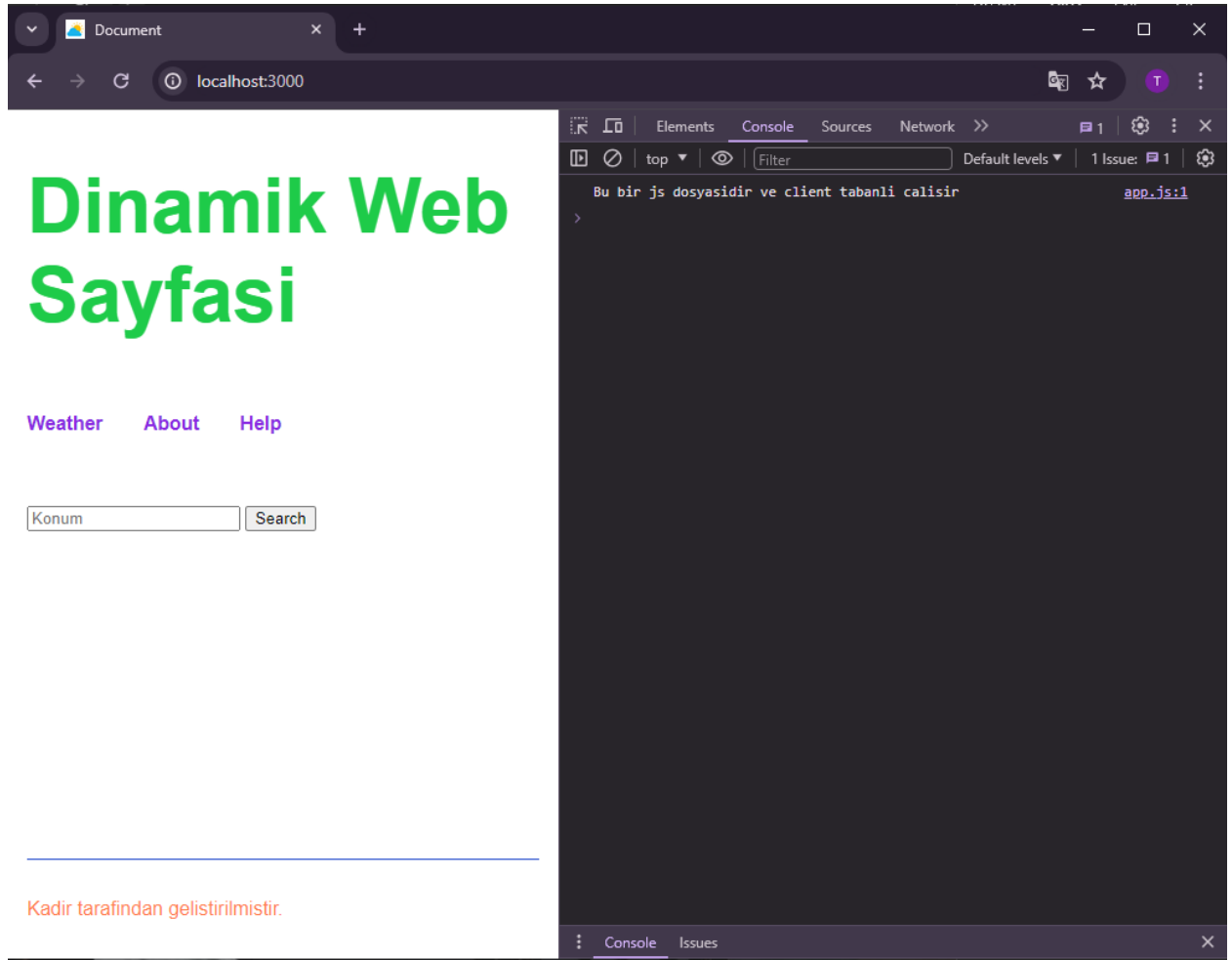
sunucudan input olarak girdiğimiz konumun bilgisine erişebiliriz. Bunu yapmak için kodumuzu şu şekilde güncelleyelim.

```
document.addEventListener("DOMContentLoaded", () => {
  const weatherForm = document.querySelector("#weatherForm");
  const search = document.querySelector("input");

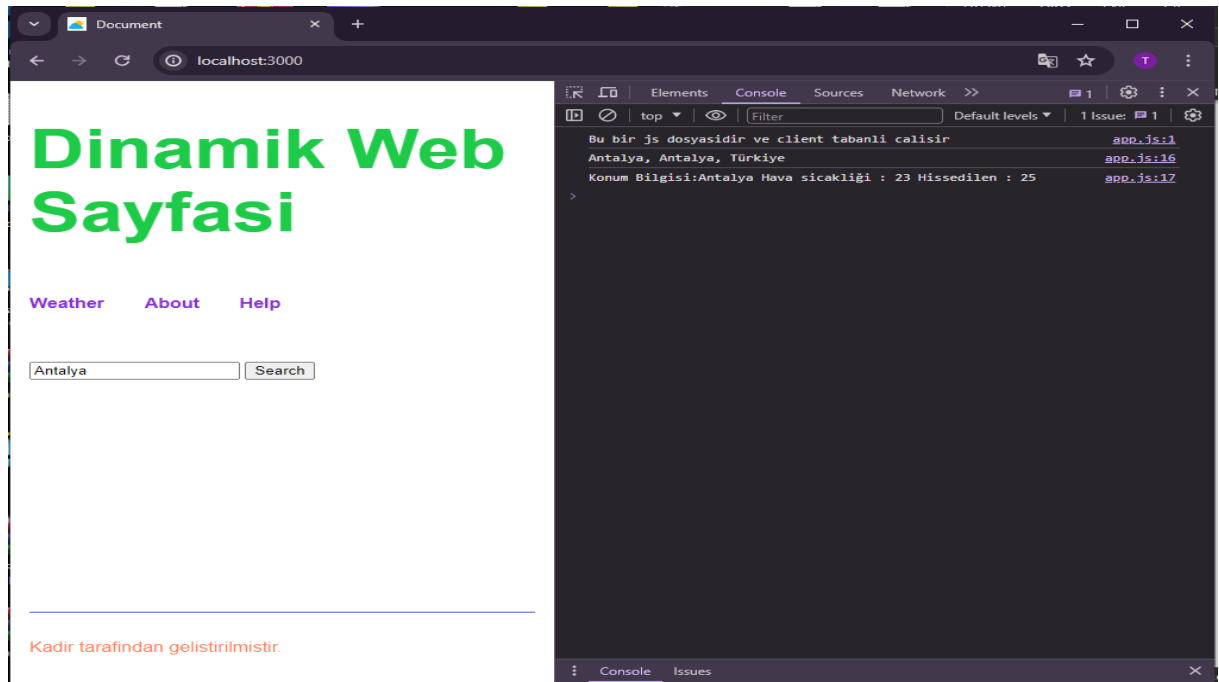
  weatherForm.addEventListener("submit", (event) => {
    event.preventDefault(); // Formun varsayılan davranışını engeller
    const location = search.value

    fetch("http://localhost:3000/weather?address="+location).then((response) => {
      response.json().then((data) => {
        if(data.error){
          console.log(data.error)
        }else{
          console.log(data.konum);
          console.log(data.forecast)
        }
      })
    })
  });
});
```

Aynı söylediğimiz gibi input içerisinde girilen bilgi fetch parametresindeki address'e dinamik olarak aktarılmış olacak. Bu sayede konsolda girdiğimiz konumun hava durumunu görüntülenebilir olacak. Testini yaptığımızda karşımıza çıkan ekran görüntüsü şudur.



Sorgumuzu yollayıp search butonuna tıkladığımızda:



Girdiğimiz input başarılı bir şekilde submit eventine ulaştı ve fetch fonksiyonuna dinamik olarak parametresine yollandı. Şimdi şu an bu kod görüldüğü gibi konsol üzerinde

çalışmakta çıkan sonucun ön yüzde kullanıcının görüntüleyebilmesi adına birkaç element ekleyelim ardından client tarafındaki js ile bu elemetlere manipule edip mevcut bilgileri ekleyelim.Hemen form elementinin altında iki tane p elementi tanımlayalım.

```
<p id="message-1"></p>
<p id="message-2"></p>
```

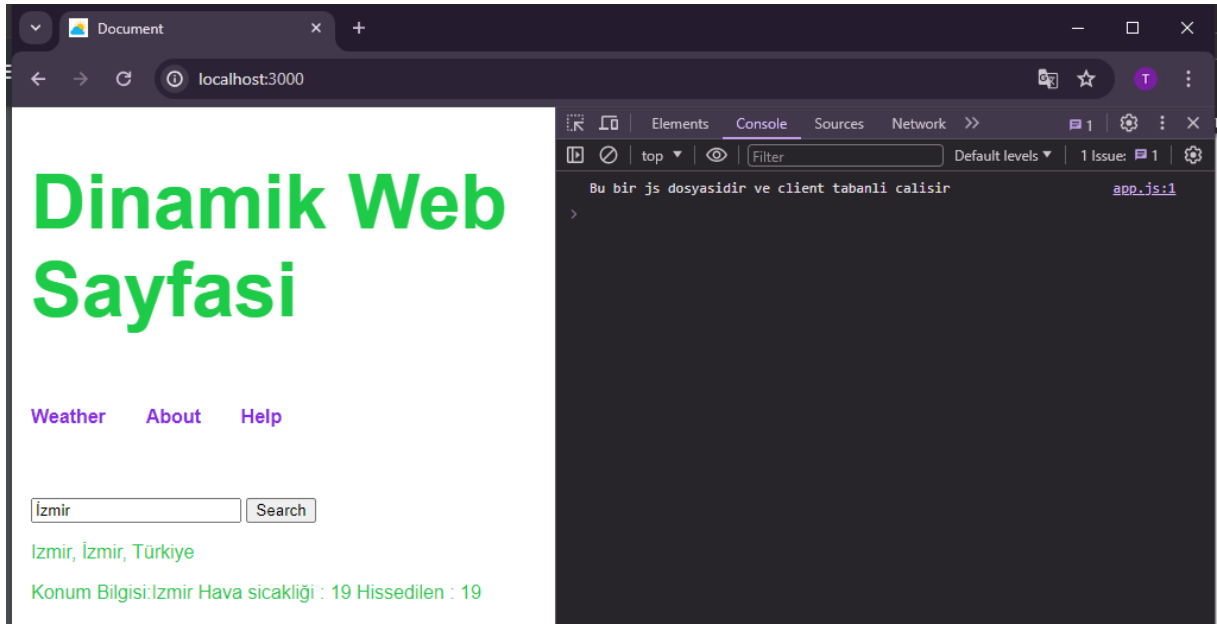
Ardından bu elementleri manipule etmek için js(client) tarafında elementleri seçelim.

```
const messageOne = document.querySelector("#message-1")
const messageTwo = document.querySelector("#message-2")
```

Bu elementleri seçtikten sonra geriye sadece sunucudan gelen bilgiyi bu elementin içeriğine yazmaktır. Bunu da şu kod ile yapıyoruz.

```
document.addEventListener("DOMContentLoaded", () => {
  const weatherForm = document.querySelector("#weatherForm");
  const search = document.querySelector("input");
  const messageOne = document.querySelector("#message-1")
  const messageTwo = document.querySelector("#message-2")
  weatherForm.addEventListener("submit", (event) => {
    event.preventDefault(); // Formun varsayılan davranışını engeller
    const location = search.value
    messageOne.textContent = "Loading";
    messageOne.textContent = "";
    fetch("http://localhost:3000/weather?address="+location).then((response) => {
      response.json().then((data) => {
        if(data.error){
          messageOne.textContent = data.error;
        }else{
          messageOne.textContent = data.konum;
          messageTwo.textContent = data.forecast;
        }
      })
    })
  });
});
```

Bu şekilde kodumuzu yazdıktan sonra test edersek karşımıza şu çıkar.



Birkaç css değişikliği yapıp kodumuzu son haline getirelim.

```
input {  
  border:1px solid #cccccc;  
  padding:8px;  
}  
  
button {  
  cursor:pointer;  
  border:1px solid #888888;  
  background:#888888;  
  color:white;  
  padding: 8px;  
}
```

Bu sayede kodumuz tamamlanmış oluyor.