**31 OCAK 2023 DERSİ (JS-21): weather application Project solution**

LMS Link <https://lms.clarusway.com/mod/page/view.php?id=39425>

****

Instructor – Ryan Daniel;

**ÖZET:**

Hava durumu bilgilerini API vasıtasıyla internetten çeken (<https://openweathermap.org/> ) ve basit bir “input” “button” kombinasyonuna sahip form yapısıyla istenen şehrin verilerini ekrana “card” yapısıyla etiren bir web aplikasyon tasarlıyoruz. HTML ve CSS dosyaları önceden tasarlanmıştır.

Bireysel veri olan API anahtarımızı localStorage altında şifreli muhafaza ediyoruz. Bunu sağlamak için AES-128 şifreleme algoritması (aes.js) ve bunu çağıran fonksiyonları (Extension.js) HTML koduna script olarak bağlıyoruz. Ayrıca diğer bir alternatif çözümün JQuery ile yapılabilmesi için “axios” kütüphanesi de bağlandı.

Aynı isimle tekrar arama yapıldığında, şehir ismi yanlış yazıldığında ve boş giriş yapıldığında ekranda küçük ikaz yazıları görüntülenecek.

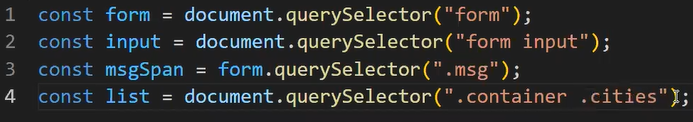
Kodun son bölümünde JS DOM events ile ilgili bazı ilave bilgiler mevcuttur, bunların aplikasyonun çalışmasına bir etkisi yoktur.

Hava durumu verisi çekeceğimiz API adresi olarak openweather.org tercih edildi. Kullanıcıya ait bireysel veri olduğu için JS kodu içinde API anahtarı silindi.  bu siteye email adresi ile üye olup kendi anahtarınızı oluşturun.

API için URL’de yazabileceğimiz parametreler ve API ile çekebileceğimiz veri çeşitlerini <https://openweathermap.org/current#geo> sitenin footer bölümünde “how to make an API call” başlığında bulabiliriz. Ayrıca API anahtarını incelemek için POSTMAN ve VScode-Thunder uzantısından faydalanabiliriz.

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

Değişkenlerin JS’de yakalanması, formun kendisi // input-veri giriş alanı // ikaz mesajının yazılacağı alanı // hava durumu kartlarının basılacağı alan yakalandı. FORM yapısı değişken olarak tümden seçildiği için burada “button” seçilmedi.



“msgSpan” değişkeni yakalanırken document yerine == > form seçicisi kullanıldı.

“.container.cities” şeklinde BİTİŞİK yazarsak tam bir class adı // “.container .cities” BOŞLUKLA yazarsak parent-child ilişkisi tanımlanmış olur ve container içindeki cities tarif edilir.

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

localStorage kullanımı;



Bireysel veri olan API KEY localStorage’a “apiKey” key adıyla AES-128 bit şifrelenerek gönderilmesi için “EncryptStringAES” fonksiyonunu çağırdık. Parantez içerisine tırnak içinde (“ ”) bireysel anahtarımızı yazacağız.

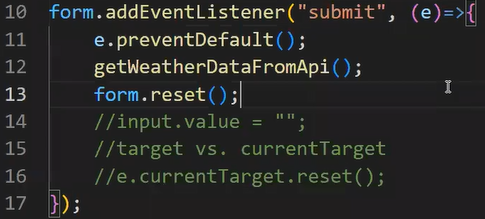
ALTERNATİF YÖNTEM (Prof. Noah);

API anahtarımızı uzantısı (.env) olan bir environment dosyasına yazar ve bunu “gitIgnore” yapabiliriz. Anahtar kullanılacağı kod bloğunda ise  şeklinde çağırabiliriz.

Sanal server hizmeti veren Netlify, Heroku, Vercel gibi platformlarda “environment” bulunuyor. Buralarda muhafaza ederek anahtarımızı kullanıcıların gözünden uzak tutabiliriz.

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

FORM yapısının avantajlarını kullanıyoruz. Görselde event “click”seçilmiş ama en DOĞRU yaklaşım “submit” seçimi



Form değişkenine “submit” event tanımlıyoruz. Formun back-end tarafına veri gönderme işlevini burada kullanmadığımız için “preventDefault()” ile bunu engelliyoruz. “input” alanının temizlenmesi için “reset() fonksiyonunu kullanıyoruz. Seçilen işlem “submit” olduğu için currentTarget ve target aynı işlevi görüyor ama aslında “click” işlevi tanımlansaydı target seçimi reset yapamazdı çünkü seçilen alan “form” alanıdır, “input” alanına hükmedemez, currentTarget ise parent’ iseçtiği için reset yapabilirdi. FORM.RESET() en doğru YAKLAŞIM. Son olarak API’den veri çekmemize yarayan “getWeatherDataFromApi()” adını verdiğimiz fonksiyonumuzu çağırıyoruz.

Bir butona 4 farklı şekilde event tanımlanabilir (HTML inline assign, addEventListener, onclick, setAttribute(“submit”))

HTML’de form etiketi içine action=”” tanımlayarak form ile veri transferi işlevleri yapılabilir. (back-end konusu)

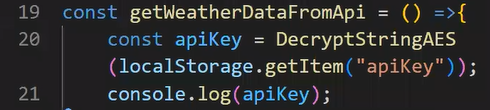
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

Taslak fonksiyonun yazımı; bu aşamada “openweather/current” sayfasından ”API CALL” için URL’de hangi parametreleri (http request - endpoint) yazacağımızı belirliyoruz. Postman’den bu API hakkında incelemeler yaparak front-end için hangi bilgileri çekeceğiz, gelen veri obj. ise destructure ile hangi verileri çekeceğiz gibi konulara karar veriyoruz. Postman’de istediğimiz verileri içeren JSON dosyası oluşturabilir ve bunu başka kullanıcılara gönderebiliriz. “import” butonu ile bize gönderilen bir JSON’u alabiliriz.

URL’de ? işaretinden sonrası parametre bölümüdür ,bu bölümde sol taraf key = sağ taraf value olur // parametreler arasına ise BOŞLUK bırakmadan & işareti koyulur.

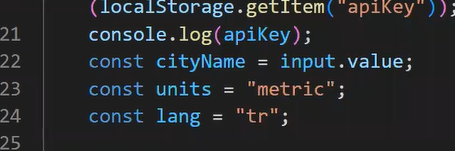
URL’de en sona &mode=xml // &mode=html yazarak çekeceğimiz verinin formatını değiştirebiliriz. Default olarak bu JSON

Status: 400 ve üzeri “client” kullanıcı tarafı hataları // 405 method not allowed // Status:200 OK

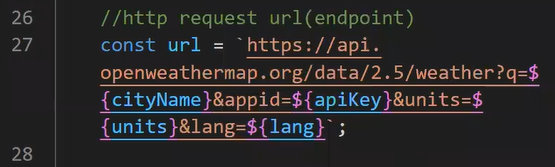


Taslak fonksiyon içinde ilk önce localStorage’de şifreli olan API anahtarımızı şifresini çözerek (DecryptStringAES fonksiyonu ile) değişkene aktarıyoruz. (apiKey)

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----



http request -URL içine yazacağımız parametreleri fonksiyon içinde değişken olarak tanımladık. CityName, Units, Lang

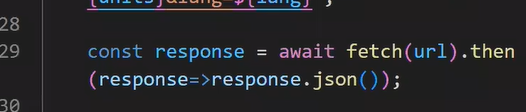


URL’i değişken olarak tanımladık, value değerlerini ${ } ile değişken olarak veriyoruz.

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

FETCH metoduyla verilerimizi alacağız. Buradaki ana yapı ASYNC – AWAIT olacak. “async” ifadesi yukarıda fonksiyonun başına yazılır. “await” ifadesi ise bekleme yapmamız gereken fetch() işleminin yanına yazılır. Talep ettiğimiz “response” cevabını json’laştırıyoruz.





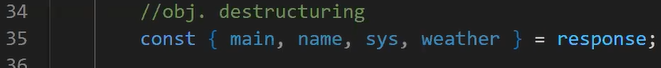
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

TRY – CATCH yapısı; hata yakalayabilmek için fetch işlemimizi bu yapının içine gömüyoruz. Hata için bir taslak hata mesajını catch içine koyduk.



---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

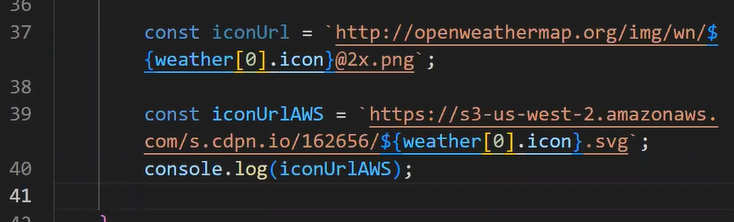
OBJECT DESTRUCTURE aşaması; gelen veri içerisinden işimize yarayacak olanları ayrıştırma aşaması



---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

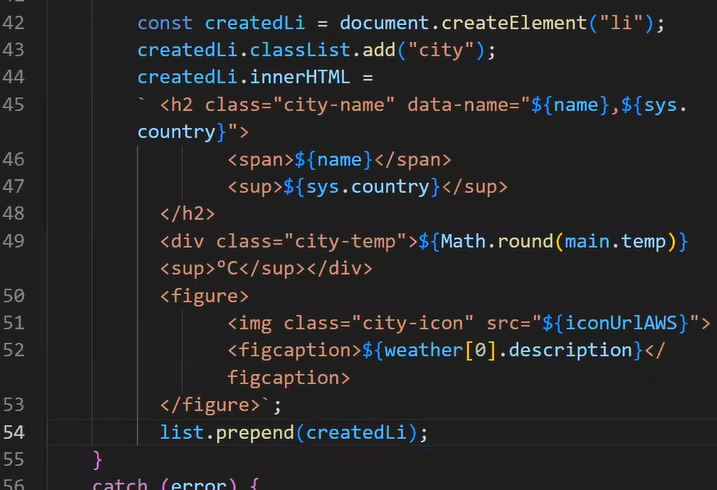
Weather ICON kullanımı; gelen veride “weather” isimli DİZİ de durum “icon” kodu mevcut. Bunu nasıl çağıracağımız sitede açıklanmış. Aşağıdaki görselde “10d” yerine destructure edilmiş icon adresini (weather dizisinin 0 indeks numaralı elemanının .icon elemanı) koyarız. Bundan daha canlı görseli olan icon için ikinci bir tanımlama daha yapıldı.





---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

Dinamik HTML elementlerini JS’de oluşturuyoruz.

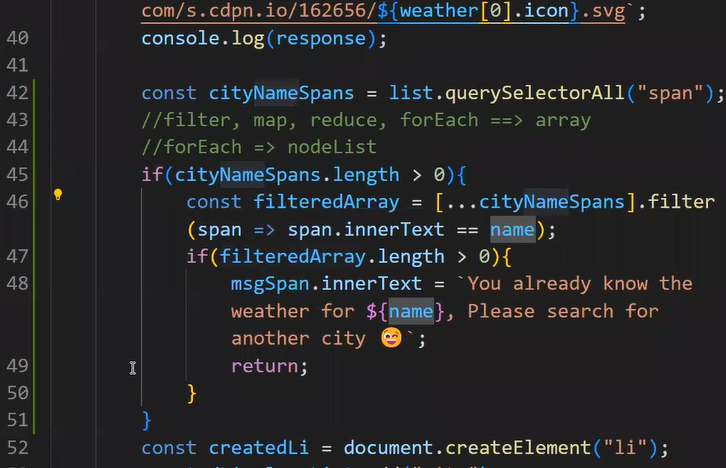


<li> elementi oluştur => buna “.city” class ata => CSS düzenlemesi yapılmış card yapısını innerHTML ile “li” içine yazdır => Bu card yapısındaki ilgili yerleri önceden belirlediğimiz değişken adlarıyla düzenle => sıcaklık değerini yuvarla => icon kullan => bunu “list” değişkenine PREPEND ile bağla => böylece en son aranan sol tarafta kalacak.

Figcaption kullanımı SEO açısından siteye fayda sağlamaktadır.

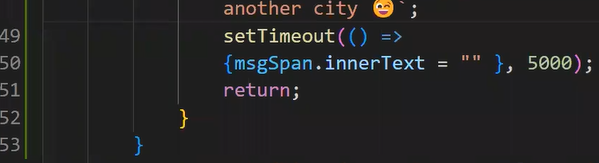
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

AYNI ŞEHİR adını tekrar tekrar aramaya engel olmak; bunu sağlayacak kontrolü “card” oluşturmadan önce yapılması gerekiyor.



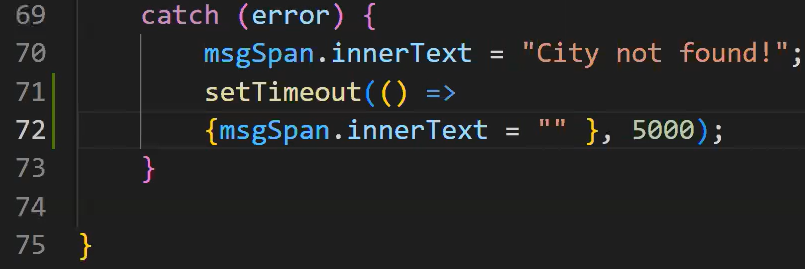
queryselectorALL ile UL altındali Lİ lerdeki bütün <span> ları seçtik => if yapısıyla eğer burada kart varsa (> 0) bunu spread operatörü ile (…) bir diziye aktar => filter metoduyla <span> innerText içeriği == API’den gelen “name” verisiyle aynı ise uyarı mesajı yazdığımız yapıya başka bir ikaz mesajı yazdır.

INPUT içine yazılan şehir adını değil, API’dan gelen arama sonuçlarını kıyaslayarak, küçük büyük harf farkı gibi hataları ayrıştırdık.



Bu uyarı mesajını 5 saniye sonra ekrandan SİL

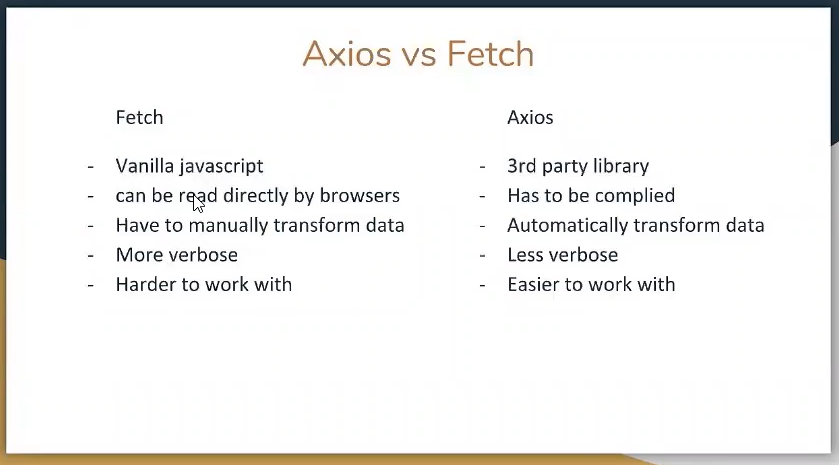
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----



Aynı HATA mesajı süre sınırlamasını CATCH içine de koyduk.

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

AXIOS;



Vanilla JS = Pure JS için fetch() kullanılır.

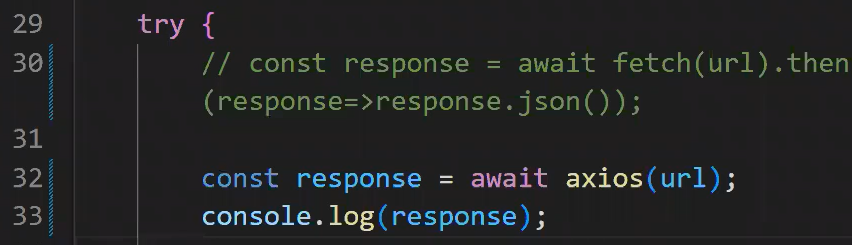
Axios react gibi kütüphanelerde kullanılır.

Axios içinde veri JSON gider ve aynı şekilde JSON gelir. Bu HTML’de axios cdn ile çağrıldı. Bizde NODE bilgisayarda bulunduğu için npm ile de alabilirdik.

Bunu CDN ile ya da bir “package manager” vasıtasıyla tanımlayıp kullanırız.

Bizim HTML kodumuzda da tanımlı.





Fetch () yapısının axios() ile değiştirmek 👆