Web Yazılım Geliştirme

Ders 06 - Web Servislere Async Çağrılar

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

Eğitmen: Ömür ŞAHİN

Hedefler

- SPA uygulamaları backend uygulamalarına AJAX kullanarak nasıl çağrı gerçekleştirebilir?
- JS'te olay döngüsü (event-loop) modeline giriş ve async/await ile asenkron davranışların kodlanması.
    - REST veya GraphQL gibi uzak servislere istekte bulunma

Tarayıcı-Sunucu İletişimi

- Genellikle TCP üzerinden HTTP ile gerçekleştirilir.
- Tarayıcının adres satırı: İlgili adresteki kaynakları indirmeyi sağlar (genellikle index.html)
- Ardından HTML içerisindeki diğer kaynaklar indirilir (CSS, görüntüler, JS dosyaları vs.)
- Kullanıcının sunucu ile etkileşimi genellikle <a> taglı bir linke tıklayarak veya bir <form>'u göndererek gerçekleşir.
  - Genellikle bu eylemler server tarafından yeni bir HTML sayfası dönmesi ile sonuçlanır.

AJAX (Asynchronous JavaScript and XML)

- JS koduna sunucu ile HTTP iletişimini başlatma yeteneği tanır.
- XML adı geçmişte XML kullanılıyor olması sebebiyleydi. Ana veri formatı artık JSON'dur.
- JSON formatında veri çekildikten sonra HTML güncellenir.

AJAX kullanımı

- Tarayıcıda JS için 2 temel yaklaşım bulunmaktadır.
- **XMLHttpRequest**: Callback kullanan eski bir yöntemdir.
  - XML adı AJAX olduğu gibi geçmişte XML'in yoğun kullanılıyor olması sebebiyledir. XML dışında JSON da kullanılabilir.
- fetch(): Promise yaklaşımı kullanan modern yaklaşımdır.

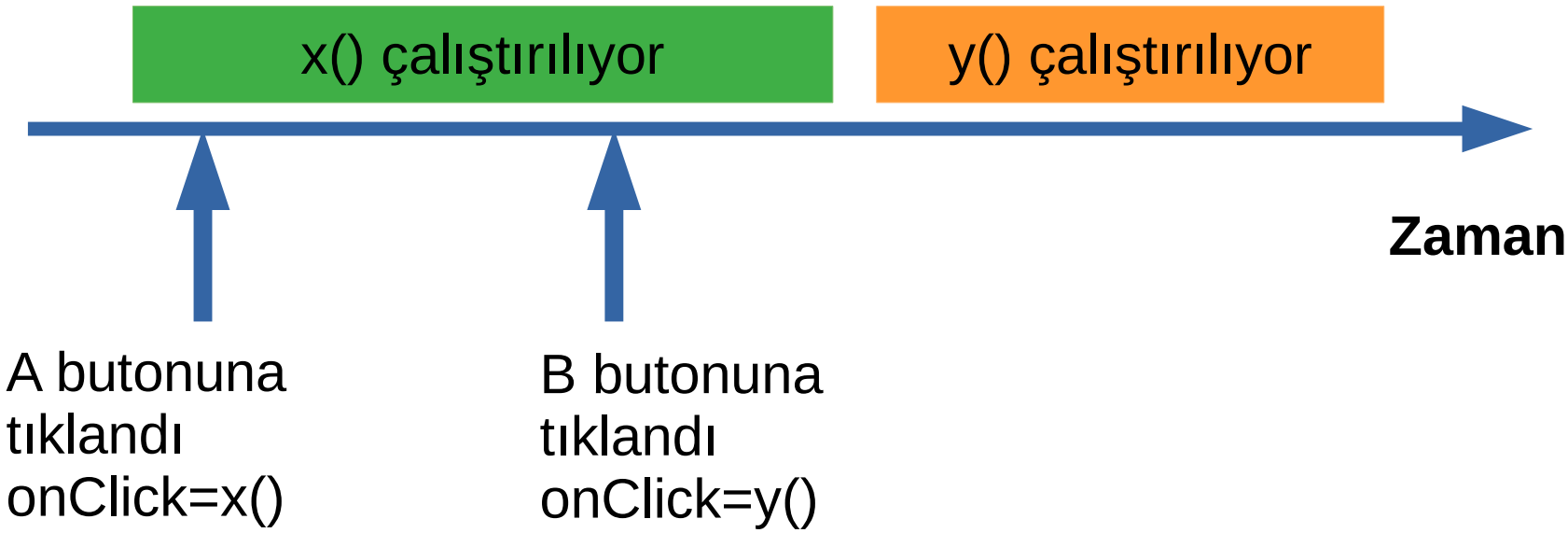
3. Parti Servis Kullanım Limitleri

- Ticari servisler onları test edebileceğiniz bazı ücretsiz özellikler sunmaktadır.
- Genellikle bir hesap oluşturmak gerekmektedir.
- Her bir HTTP isteği atıldığında bir anahtar (API KEY) göndermeniz gerekir.
- Çok fazla talep atılması durumunda erişim bloklanabilir veya ticari bir lisans satın almanız istenir.

AJAX ile ilgili sorunlar

- AJAX kullanarak TCP üzerinden HTTP yapmanızı sağlar.
- Bu türde bir çağrıya çok kısa bir süre içinde (ms içerisinde) cevap gelebileceği gibi saniyeler de sürebilmektedir.
- Bu türde bir çağrı çok kısa sürse bile mevcut sayfayı render etmeden önce birden fazla HTTP talebinde bulunmak zorunda olabilirsiniz.
- Böyle bir durumda sunucudan gelecek cevabı bekleyerek sayfanın donması istenmemektedir.
- Bu durum, JS'te çözümlenmesi gereken problemlerden biridir.

JS Olay Döngüsü - Örneği



Tamamlanma problemi

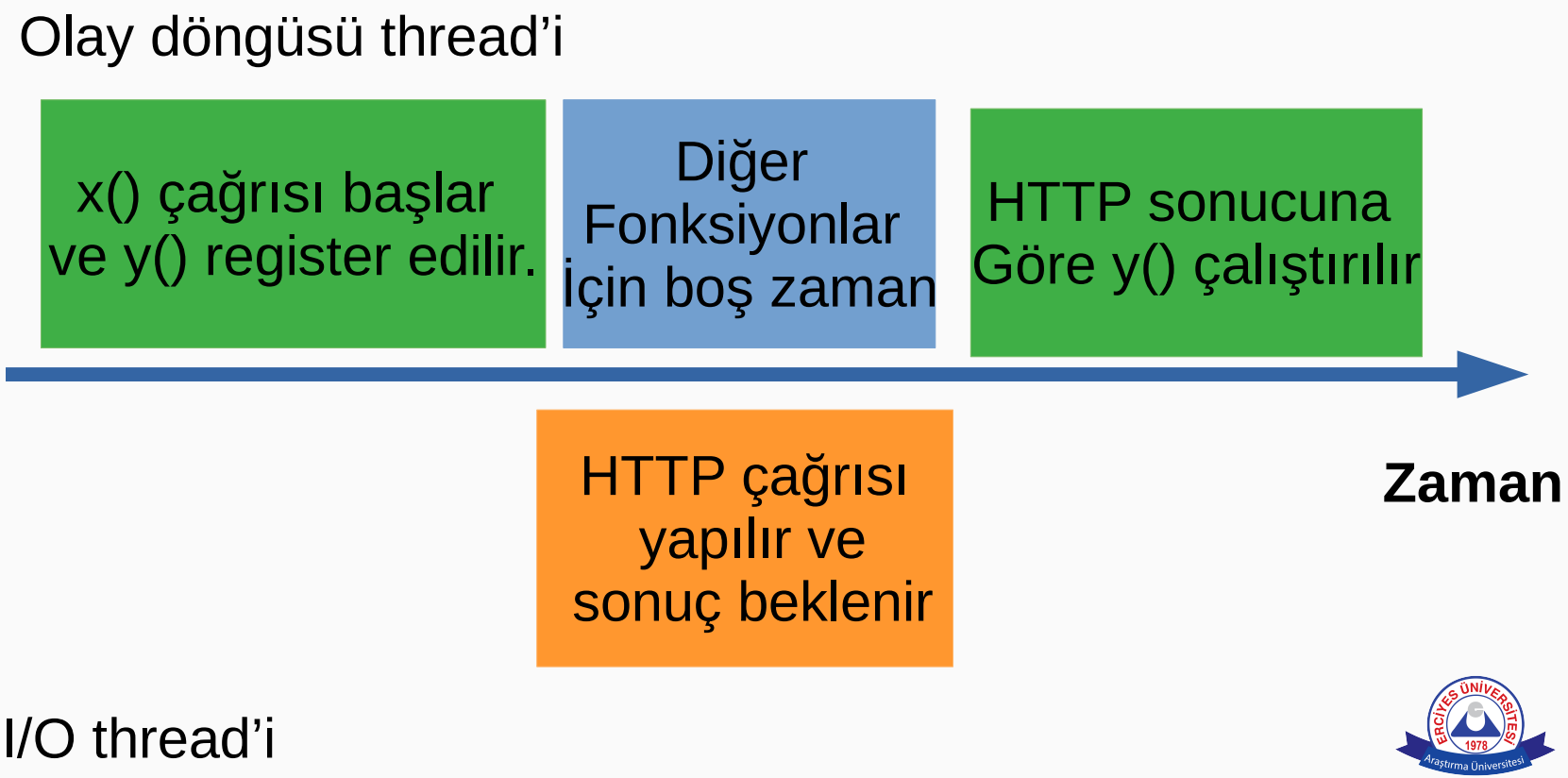
- **while(true)**{} böyle bir kod parçası sonsuz döngü olması se bebiyle uygulamayı tamamen dondurabilir.
  - Çeşitli buglar sonucu da sonsuz döngüler ortaya çıkabilir.
- JS kodlarındaki maliyetli CPU hesaplamaları uygulamanın cevap verme süresini artıracak ve uygulamanın oldukça hantal çalıştığını hissettirecektir.

AJAX ve Eylemlerin Tamamlanması

- HTTP çağrısı yapıldıktan sonra gelen cevaba göre işlemler yürütülmek istenebilir.
- Tekrar cevap gelene kadar aradan belli bir süre geçecektir.
- Cevap gelene kadar da uygulama bekleme yaparsa bu süreç içerisinde donacaktır. Bu türde bir donma istenmemektedir.
- Bu durumlarda 2 farklı çözüm bulunmaktadır: callback ve promise yaklaşımı (async/await)

Callback

- AJAX x() fonksiyonunu callback fonksiyon olarak register eder ve bundan glen cevaba göre y() fonksiyonunu çağırılacaktır.
- HTTP çağrısı bir I/O thread'i içerisinde gerçekleşecektir ve ardından y() çalıştırılacaktır.



Örnek

```
1   const ajax = new XMLHttpRequest();
2   /* callback olarak register edilir */
3   ajax.onreadystatechange = () => {
4   if(ajax.response){
5   const payload = JSON.parse(ajax.response)
6   console.log(payload);
7   }
8   }
9   let url =
        "http://api.openweathermap.org/data/2.5/forecast?
10  appid=5797bc3c4746e6fce9d9dc26addba2c7
11  &units=metric&q=Kayseri"
12  ajax.open("GET",url);
13  ajax.send();
```

- onreadystatechange sonuç geldiğinde çalışacaktır.
- send() fonksiyonu çalışmadan register edilmelidir.

Callback ile ilgili problemler

- Tek bir istek atılacağı zaman gayet iyi çalışmaktadır.
- Birden fazla asenkron iletişime ihtiyaç duyulan ve birbirine bağımlı olan isteklerde nelerin gerçekleştiğini ve hangi sıra ile çalıştığını takip etmek oldukça zordur.
- Bu yüzden genellikle Callback Hell olarak adlandırılır.

Promise yaklaşımı

- Promise'ler JavaScript nesneleridir.
- Promise'ler sonunda tamamlanan (veya hata ile sonuçlanan) ve işleme ait sonucu barındıran JavaScript nesnesi olarak tanımlanmaktadır.
- Bir promise en sonda asenkron işleme ait cevaba sahiptir ve bu cevaptan gelen sonuç erişilebilir olana kadar beklenebilir (await).
- fetch() metodu sonunda Promise dönen AJAX isteği atabilmektedir.

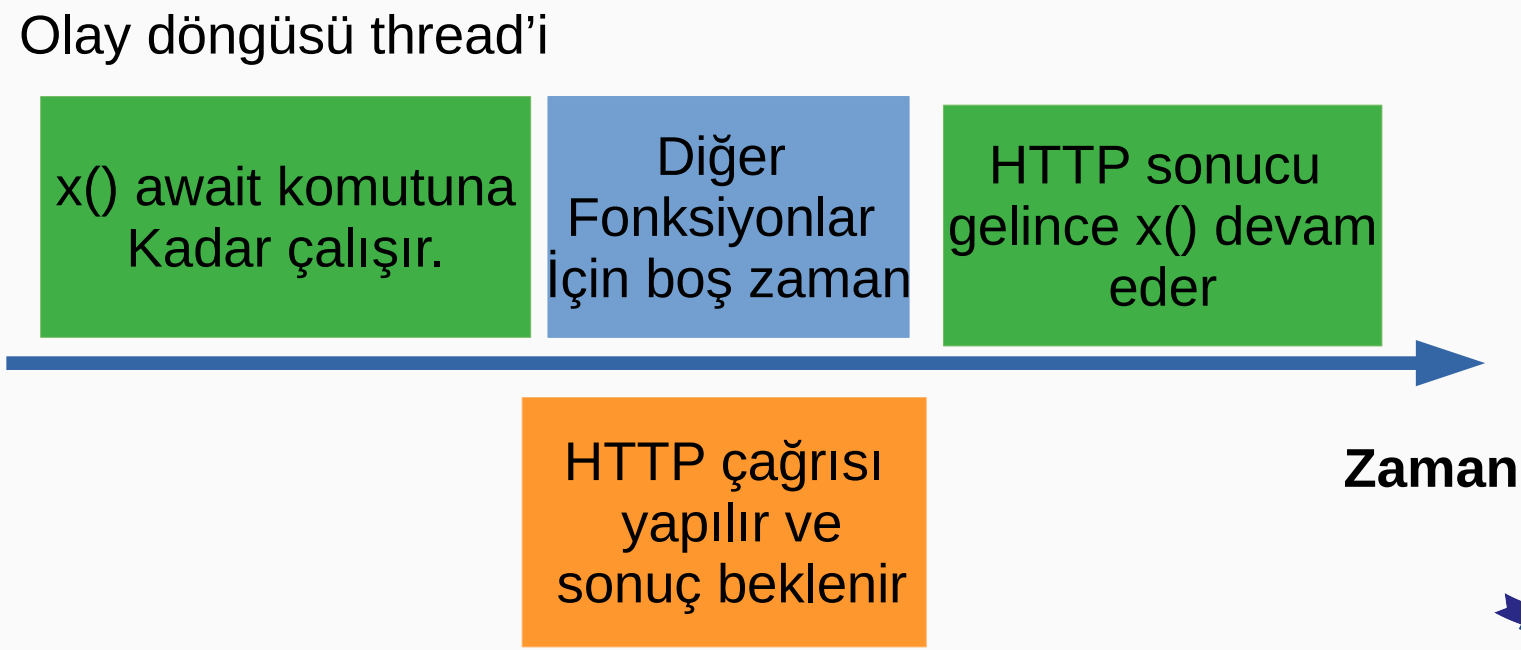
Örnek

```
1
2  let url =
       "https://api.openweathermap.org/data/2.5/forecast?
3  appid=5797bc3c4746e6fce9d9dc26addba2c7
4  &units=metric&q=Kayseri"
5
6  const doHttpFetch = async (url) =>  {
7  response = await fetch(url);
8  payload = response.json()
9  console.log(payload);
10 }
11
12 doHttpFetch(url);
```

- fetch işlemi bir async fonksiyon içerisinde çağırılmıştır.
- Sonuç geldikten sonra fonksiyonun devamı çalışır.

async/await

- async fonksiyonlar await komutu ile çalıştırma bloklarına bölünür.
  - Aynı async fonksiyon içerisinde birden fazla await komutu olabilir.
- I/O thread'i HTTP cevabı aldığında await'ten sonraki komutların çalışması için planlama yapılır.
- Olay döngüsü thread'i await boyunca çalışmaya devam eder.



async/await yapısının faydaları

- Kodun okunurluğunu oldukça fazla artırmaktadır. Ayrıca çalışma sırası da sıralı olarak görünmektedir.
  - Aynı fonksiyon içerisinde birden fazla asenkron işlem olduğu zamanlar bu durum doğrudur.
- Performans açısından büyük bir dezavantaj oluşturmamaktadır çünkü I/O thread'i çalışırken olay döngüsü thread'i çalışmaya devam etmektedir.
- İşletim sistemleri dersinden hatırlayacağınız üzere thread'ler arası geçiş maliyetli bir eylemdir.

Non-Blocking I/O

- Olay döngüsü içerisinde çalışan kod blok modeline genellikle Non-Blocking I/O adı verilir.
- Bu model NodeJS ile birlikte popüler olmuştur ancak Java, Kotlin, C# gibi diğer diller de bunu desteklemektedir.
- CRUD işlemlerinin yapıldığı web uygulamalarında oldukça iyidir.
  - CPU üzerinde gerçekleştirilen eylemlerin çoğunun maliyeti düşükken, veritabanında gerçekleşen I/O işlemlerinde darboğaz oluşmaktadır.
  - Birden fazla kullanıcıya thread değişikliği yapmadan hizmet sunmaya olanak sağlar.
- Ancak CPU bağımlı işlemlerde iyi bir yaklaşım değildir.
  - Yalnızca 1 çalıştırma thread'i bulunmaktadır.
  - Bu durumda load-balancer'lar kullanarak birden fazla uygulama çalıştırabilirsiniz.

Promise oluşturma

- Bu ders kapsamında fetch() çağrılarını bekleyeceğimiz (await) Promise yaklaşımı kullanılacaktır.
- Ancak kendi Promise'lerimizi oluşturabiliriz.
  - Test amacıyla bu türde yapılar oluşturacağız.
- Bir Promise girdi olarak fonksiyon almaktadır ve bu fonksiyon da iki girdiye sahiptir:
  - resolve (değer): Problemin başarıyla çözüldüğü ve bir değer dönmeye hazır olduğumuz durumlarda çağırılmaktadır. değer ne dönmek istiyor isek onu göstermektedir.
  - reject(): Promise'in başarısız olması durumunda ne olacağını belirler.
  - resolve(1) değeri anında dönüş yapar ve 1 çıktısını sonuç olarak döner.

```
1  new Promise((resolve,reject) => {/*kod*/});
```