Web Programlama I

Ders 03 - SPA Bileşenleri

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

Eğitmen: Ömür ŞAHİN

Sunum Planı

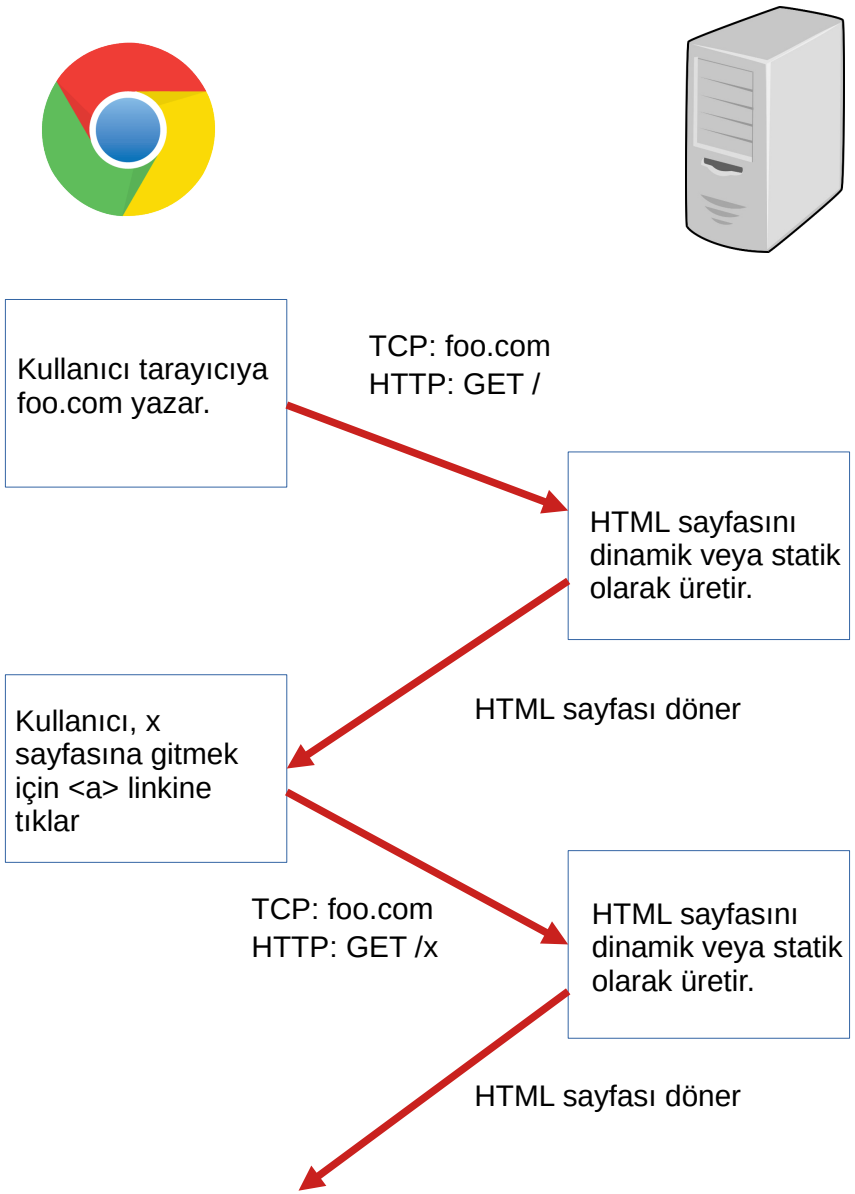
1-Genel Bakış

Genel Bakış

- Tek Sayfa Uygulamaları (Single Page Application, SPA) temel konsepti
- Neden doğrudan DOM manipülasyonu tavsiye edilmemektedir, neden kütüphane/framework kullanması daha iyidir?
- SPA componentlerini anlama
- Vite ile React projesi oluşturma
- Modern React (Hooks ve Functional Components)

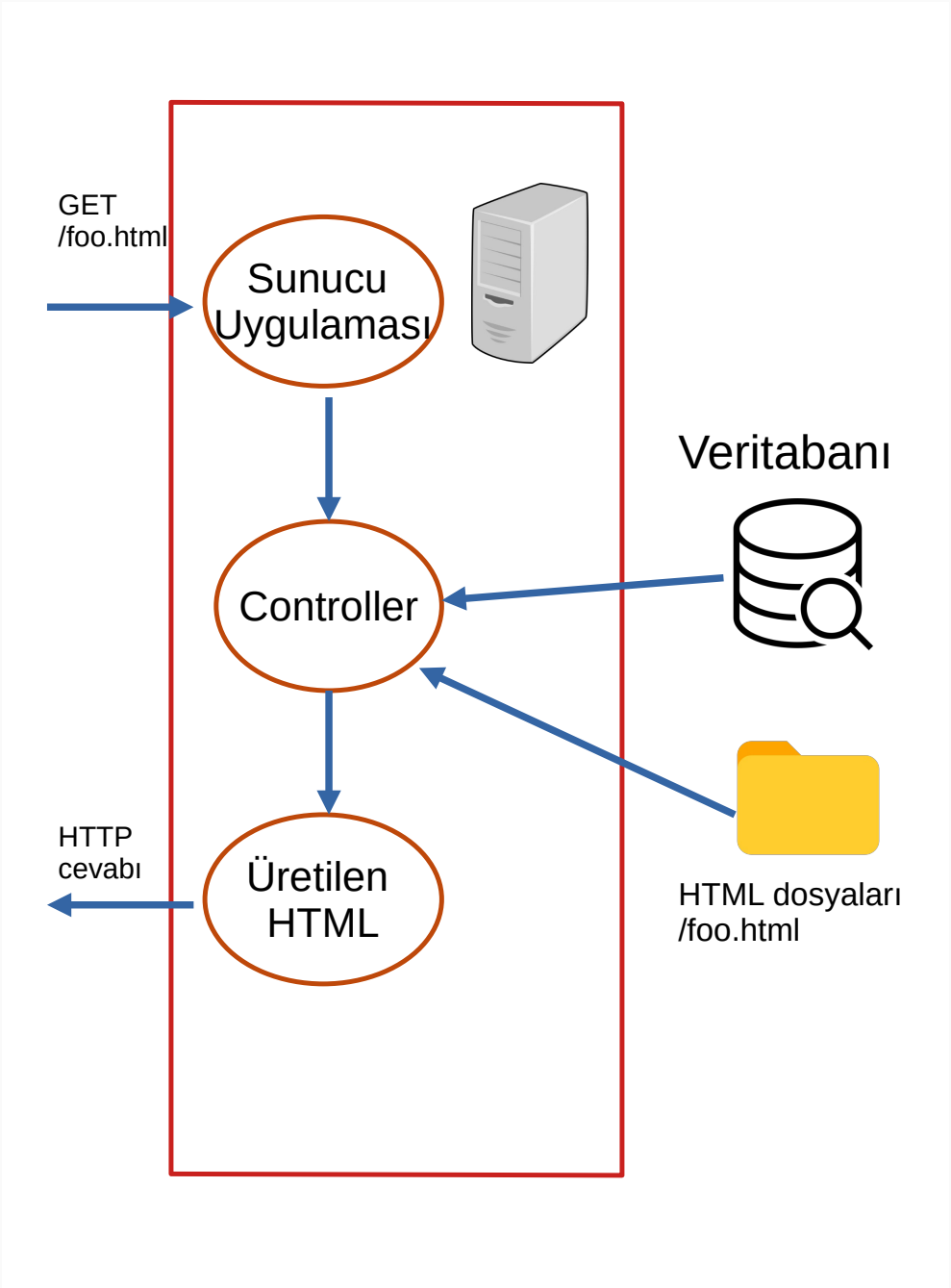
2-Geleneksel Web Uygulamaları

Geleneksel Web Uygulamaları



- Sayfalar arası etkileşim $< a >$ ve $< form >$ gibi taglarla gerçekleştirilir.
- Her bir istek GET, POST gibi metotlardan oluşmaktadır.
- HTML sayfasının tamamı sunucu tarafından üretilir.

Server Side Rendering



3-SPA

SPA

- Yalnızca tek bir HTML dosyasından oluşur.
- Bütün HTML içeriği JavaScript tarafından dinamik olarak üretilir.
- Sayfalar arası geçiş GUI'nin anında değiştirilmesi ile simüle edilir (Client-Side Routing).
- Modern build araçları (Vite) ile hızlı geliştirme deneyimi.

Veri çekme

- HTML dosyalar tarayıcı tarafından JS ile üretiliyor olsa da sunucu ile iletişime ihtiyaç duyulmaktadır.
  - Veri kaydetme/değiştirme
- Yeni bir HTML dosyası alınmaz.
- Veriler JavaScript Object Notation (JSON) formatındadır.
- JS DOM üzerindeki güncellemeleri JSON verilerine göre gerçekleştirir.
- Sunucular JSON veri sağlar.

SPA karmaşıklığı

- Frontend tarafında çok fazla JS kodu bulunmaktadır.
- Bütün tarayıcı eventlerinde veya durum değişikliklerinde DOM güncellemesini manuel yapmak çok ölçeklenebilir bir yaklaşım değildir.
- Bu türde karmaşık problemleri çözmek için tasarım paternlerine ve çeşitli araçlara ihtiyaç duyulmaktadır.

Kütüphane/Framework

- Frontend teknolojileri oldukça hızlı değişmektedir.
- **React**: Meta (Facebook) tarafından geliştirilmektedir.
  - En popüler frontend kütüphanesi
  - Hooks ve Functional Components ile modern yaklaşım
  - Geniş ekosistem (Next.js, Remix)
- **Vue**: Progressive framework, kolay öğrenilebilir
  - Composition API ile modern yaklaşım
  - Nuxt.js ile SSR desteği
- **Svelte**: Compiler-based yaklaşım
- **Solid.js**: React benzeri, ancak daha performanslı
- Angular hala enterprise projelerde kullanılmakta

4-React

React Bileşenleri (Component)

- Componentler bir **state** ile tanımlanır ve bu state kullanılarak HTML **render** edilir.
- Modern React'ta **Functional Components** ve **Hooks** kullanılır.
- Web sayfası bir root component ile temsil edilir ve ağaç yapısı halinde children component'lar barındırır.
- Doğrudan render işlemi gerçekleştirilmez. Herhangi bir state değişimi olduğunda React otomatik olarak gerekli yerleri render eder.

Render Optimizasyonu

- Tarayıcı üzerinde çok fazla event bulunabilir (kullanıcı tıklamaları, fare hareketi vs.)
- React gerekli noktaları tekrar render ederken otomatik olarak optimize eder.
- **Automatic Batching**: Birden fazla state güncellemesi tek render'da birleştirilir.
- Virtual-DOM
  - Component state'i değiştiğinde HTML'in oldukça küçük bir noktasını etkilemiş olabilir.
  - React tüm HTML'i yeniden oluşturmaz. Yalnızca değiştirilen kısımları oluşturur (Reconciliation).
  - Bir Virtual-DOM bellekte tutulur ve sadece mevcut arayüz VDOM'dan farklı olduğunda güncellenir.

JSX

- React component'leri **JSX** ile HTML benzeri syntax kullanarak içerik üretir.
- HTML içeriğini bir JS string'i içerisinde tutmak oldukça hataya açıktır.
- **JSX** ise JS ve HTML kodlarını bir arada tutabileceğiniz React dosya formatıdır.
- Modern React (17+) ile **import React** gerekmez (Automatic JSX Transform).
- Vite gibi modern araçlar JSX'i otomatik olarak JavaScript'e dönüştürür.
- Not: JSX dosyaları için ".jsx" uzantısını kullanacağız. Eğer typescript de bulunuyorsa "tsx" kullanılacaktır.

5-Vite ile React

Vite Nedir?

- Webpack ve Babel yerine modern **Vite** build aracı kullanılır.
- **Son derece hızlı**: ES modules ve esbuild kullanır
- Anında sunucu başlatma (instant server start)
- Hızlı Hot Module Replacement (HMR)
- Otomatik JSX dönüşümü ve TypeScript desteği
- Konfigürasyon gerektirmez (zero-config)

Vite ile React Projesi Oluşturma

```
# Proje olusturma
npm create vite@latest my-react-app -- --template
    react

# Dizine git
cd my-react-app

# Bagimliliklari yukle
npm install

# Gelistirme sunucusunu baslat
npm run dev
```

Vite Proje Yapısı

- **index.html**: Root HTML dosyası (public klasöründe değil!)
- **src/main.jsx**: Entry point
- **src/App.jsx**: Ana component
- **vite.config.js**: Konfigürasyon dosyası (opsiyonel)
- **package.json**: Bağımlılıklar ve scriptler

React.Component (Class-based) - LEGACY

- **Not**: Class-based componentler artık önerilmez!
- Modern React'ta **Functional Components + Hooks** kullanılır.
- Class componentler yalnızca eski projelerde görülebilir.
- Bu ders kapsamında **yalnızca Functional Components** kullanacağız.

6-React Hooks

React Lifecycle - Modern Yaklaşım

- Class componentlerin lifecycle metotları yerine **useEffect** hook kullanılır.
- **componentDidMount** → useEffect(() => {...}, [])
- **componentDidUpdate** → useEffect(() => {...}, [dependencies])
- **componentWillUnmount** → useEffect cleanup function

React Hook'lar

- Hook'lar 2019 yılında React'a dahil edilmiştir.
- Component'ları state kullanarak fonksiyon olarak tanımlanmasına olanak tanır.
- **2025 itibarıyla React'ın standart yaklaşımıdır**.
- Class componentler artık kullanılmamaktadır.
- Daha temiz, okunabilir ve test edilebilir kod.

useState Hook

```
1   import { useState } from 'react';
2
3   function Counter() {
4     const [count, setCount] = useState(0);
5
6     return (
7     <div>
8     <p>Count: {count}</p>
9     <button onClick={() => setCount(count + 1)}>
10    Increment
11    </button>
12    </div>
13    );
14  }
```

useEffect Hook

```
1   import { useState, useEffect } from 'react';
2
3   function DataFetcher() {
4     const [data, setData] = useState(null);
5
6     useEffect(() => {
7       // Component mount oldugunda calisir
8       fetch('/api/data')
9       .then(res => res.json())
10      .then(setData);
11
12      // Cleanup function (unmount)
13      return () => console.log('Cleanup');
14    }, []); // Bos array: sadece mount'ta calisir
15
16    return <div>{data?.name}</div>;
17  }
```

Sık Kullanılan React Hooks

- **useState**: Component state yönetimi
- **useEffect**: Side effects ve lifecycle
- **useContext**: Context API ile global state
- **useRef**: DOM referansları ve mutable değerler
- **useMemo**: Pahalı hesaplamaları cache'leme
- **useCallback**: Fonksiyonları memoize etme
- **useReducer**: Karmaşık state yönetimi

TypeScript ile React

- Vite TypeScript şablonu: **–template react-ts**
- Type safety ile daha az hata
- Daha iyi IDE desteği (autocomplete)
- Props ve state için tip tanımları

React Developer Tools

- Chrome/Firefox eklentisi
- Component ağacını görüntüleme
- Props ve state'i inceleme
- Performance profiling
- **Mutlaka yüklenmelidir!**