DERS ADI İleri Web Programlama
ÜNİTE ADI Gelişmiş HTML ve CSS
ÜNİTE NO 1
YAZAR Arş.Gör. SATUK BUĞRAHAN ÖZTÜRK

HTML5 VE GELİŞMİŞ WEB TEKNOLOJİLERİ

HTML5, web geliştirme dünyasında önemli bir dönüm noktası olarak kabul edilmektedir. Bu teknoloji, geleneksel HTML sürümlerinden farklı olarak, daha güçlü ve kullanıcı dostu özellikler sunmaktadır. Yeni semantik etiketler (,,,) sayesinde web sayfalarının yapısı daha anlamlı hale gelirken, arama motorları ve yardımcı teknolojiler tarafından daha kolay anlaşılabilir bir yapı sunmaktadır. Ayrıca, HTML5 ile gelen ve etiketleri, multimedya içeriklerin üçüncü taraf eklentiler kullanılmadan doğrudan sayfalara entegre edilmesine olanak tanımaktadır.

HTML5'in bir diğer önemli yeniliği, web formlarında sunduğu gelişmiş özelliklerdir. Yeni form tipleri (email, url, date) ve placeholder, required gibi özellikler, kullanıcı deneyimini artırmakta ve doğrulama süreçlerini kolaylaştırmaktadır. Yerel depolama (Local Storage) ve oturum depolama (Session Storage) gibi veri yönetim araçları, kullanıcı verilerinin güvenli ve hızlı bir şekilde saklanmasını mümkün kılmaktadır. HTML5 ile gelen Canvas API ve WebGL desteği ise grafik ve görselleştirme ihtiyaçlarına güçlü çözümler sunarak, özellikle oyun ve veri görselleştirme uygulamalarında kullanılmaktadır.

GELİŞMİŞ CSS TEKNİKLERİ

CSS3 ile birlikte web tasarımında önemli yenilikler ve gelişmiş teknikler ortaya çıkmıştır. Yeni seçiciler (:nth-child(), [attribute^="value"]) sayesinde, karmaşık düzenlemeler daha kolay bir şekilde yapılabilmekte, kod okunabilirliği artırılmaktadır. Ayrıca, CSS3 animasyonları ve geçiş efektleri, web sayfalarına dinamik özellikler kazandırmakta, kullanıcı deneyimini zenginleştirmektedir. Örneğin, @ keyframes ile oluşturulan animasyonlar, bir öğenin belirli bir süre boyunca hareket etmesini sağlarken, transition özelliği, öğe değişikliklerinin daha akıcı olmasını sağlamaktadır.

Flexbox, CSS3 ile tanıtılan bir diğer önemli düzenleme aracı olarak, öğelerin hem yatay hem de dikey eksende kolayca hizalanmasını sağlar. Responsive tasarımlarda sıklıkla kullanılan Flexbox, düzenlerin esnek ve kullanıcı dostu olmasını sağlamaktadır. CSS Grid ise iki boyutlu düzenleme sistemleri için ideal bir çözüm sunar. Grid yapısı, karmaşık düzenlerin daha düzenli ve etkili bir şekilde oluşturulmasını mümkün kılar. Ayrıca, CSS değişkenleri sayesinde stil dosyaları daha modüler hale gelmekte ve kod yönetimi kolaylaşmaktadır.

RESPONSIVE TASARIM TEKNİKLERİ

Responsive tasarım, modern web geliştirme süreçlerinde cihaz bağımsız bir deneyim sunmayı mümkün kılan bir yaklaşımdır. CSS3 ile gelen media queries özelliği, farklı ekran boyutlarına ve cihaz türlerine göre özel stiller uygulanmasını sağlar. Bu teknik, web sayfalarının hem masaüstü hem de mobil cihazlarda sorunsuz bir şekilde görüntülenmesini mümkün kılar. Esnek düzenler, yüzde tabanlı genişlik ve yükseklik kullanımıyla ekran boyutlarına uygun bir yapı sunarken, esnek görseller ve videolar, içeriklerin herhangi bir ekrana kolayca uyum sağlamasını sağlar.

Mobil öncelikli tasarım (Mobile-First), responsive tasarımda önemli bir yaklaşımdır. Bu yöntemde, önce mobil cihazlar için optimize edilmiş bir tasarım oluşturulmakta ve ardından daha büyük ekranlar için genişletilmektedir. Ayrıca, Flexbox ve Grid gibi düzenleme araçları, responsive tasarımda karmaşık düzenlerin basit bir şekilde oluşturulmasını sağlamaktadır. Framework kullanımı (örneğin, Bootstrap), responsive tasarım süreçlerini hızlandıran etkili bir çözüm sunmaktadır.

MODERN WEB ARAÇLARI VE ÇALIŞMA YÖNTEMLERİ

Modern web geliştirme süreçlerinde kullanılan araçlar, iş akışını hızlandırmak ve kod kalitesini artırmak için önemli bir rol oynamaktadır. CSS ön işlemcileri (SASS/LESS), stil dosyalarının daha modüler ve yönetilebilir hale gelmesini sağlar. Bu araçlar, değişkenler, döngüler ve fonksiyonlar gibi özellikler sunarak, kod tekrarını azaltır ve daha esnek bir yapı oluşturur. PostCSS gibi optimize edici araçlar, CSS kodunu analiz eder, dönüştürür ve performansı artırır. Örneğin, Autoprefixer eklentisi, tarayıcı uyumluluğunu artırmak için gerekli ön eklemeleri otomatik olarak ekler.

Tarayıcı geliştirici araçları (Developer Tools), performans analizi ve hata ayıklama süreçlerinde kritik bir rol oynamaktadır. Bu araçlar, ağ trafiğini izleme, CSS ve DOM düzenlemeleri yapma ve performans sorunlarını tespit etme gibi işlevler sunmaktadır. Versiyon kontrol sistemleri (Git) ve kod işbirliği platformları (GitHub), ekip çalışmalarını kolaylaştırır ve kod değişikliklerinin yönetimini sağlar. Git, kodun farklı sürümlerini takip ederken, GitHub, projelerin paylaşılmasını ve ekip üyeleri arasında işbirliğini destekler.

SONUÇ

HTML5, CSS3 ve modern web araçlarının birlikte kullanımı, web geliştirme süreçlerinde hem işlevsellik hem de estetik gereksinimlerini karşılamaktadır. Bu teknolojiler, geliştiricilere daha esnek ve güçlü araçlar sunarken, kullanıcılar için daha iyi bir deneyim sağlamaktadır. Responsive tasarım teknikleri, cihaz bağımsız bir yapı sunarak kullanıcı memnuniyetini artırırken, modern araçlar iş akışını hızlandırmakta ve kod kalitesini artırmaktadır. Bu birleşim, web projelerinin başarılı bir şekilde hayata geçirilmesinde kritik bir rol oynamaktadır ve modern web geliştirme süreçlerinde vazgeçilmez bir temel oluşturmaktadır.

DERS ADI İleri Web Programlama
ÜNİTE ADI CSS Frameworkleri ve Tasarım
ÜNİTE NO 2
YAZAR MEHMET CANCAN

CSS FRAMEWORK NEDİR?

CSS frameworkler, web sayfalarının düzenini ve tasarımını hızlı bir şekilde oluşturmayı sağlayan araçlardır. İçerdikleri önceden tanımlı CSS sınıfları, stiller ve yapısal bileşenler sayesinde web geliştirme süreçlerini hem kolaylaştırır hem de standart bir yapıya oturtur. Geliştiriciler, bu frameworkler sayesinde sıfırdan kod yazma zorunluluğundan kurtulur ve projelerini daha hızlı tamamlayabilir. Örneğin, bir butonun rengi, boyutu ve şekli gibi özellikler framework içinde tanımlıdır. Bu sayede kullanıcılar, hazır sınıfları kullanarak hızlı, uyumlu ve estetik tasarımlar oluşturabilirler.

Neden CSS Framework Kullanılır?

CSS frameworkler, web geliştirme süreçlerini kolaylaştıran ve projelerde verimlilik sağlayan araçlardır. Bunların kullanılmasının başlıca avantajları arasında projelerin daha hızlı tamamlanmasını sağlaması, tekrarlayan işlemler için zaman kazandırması ve cihazlara uygun responsive tasarımlar oluşturmayı kolaylaştırması yer alır. Ayrıca, tüm projede standart bir tasarım dilini koruyarak tutarlılık sunar ve kapsamlı belgeleri sayesinde hem yeni başlayanlar hem de deneyimli kullanıcılar için kolay anlaşılır bir kullanım sağlar. Takım çalışmalarında, aynı frameworkün kullanımı kodun daha rahat anlaşılmasına ve sürdürülebilirliğine katkıda bulunur. Öne çıkan frameworkler arasında Bootstrap, Tailwind CSS ve Bulma bulunur; her biri farklı ihtiyaçlara yönelik çeşitli özellikler sunar.

POPÜLER CSS FRAMEWORKLER

CSS frameworkler, web geliştirme sürecini kolaylaştıran ve farklı ihtiyaçlara hitap eden araçlardır. Geliştiricilere sunduğu çeşitli özellikler sayesinde projelerde hız ve verimlilik sağlar. İşte öne çıkan ve yaygın olarak kullanılan bazı popüler CSS frameworkler:

Bootstrap

Bootstrap, en yaygın kullanılan frameworklerden biridir. Twitter geliştiricileri tarafından oluşturulan bu araç, güçlü bir grid sistemi ve çeşitli hazır bileşenlerle kullanıcıların hızlı ve özelleştirilebilir tasarımlar yapmasını sağlar. Butonlar, menüler, modal pencereler gibi birçok öğeyi kolayca oluşturmaya olanak tanır. Ayrıca, Bootstrap'in kapsamlı dökümantasyonu ve geniş bir topluluk desteği, öğrenme ve sorun giderme süreçlerini oldukça kolaylaştırır.

Tailwind CSS

Tailwind CSS, "utility-first" yaklaşımıyla dikkat çeken bir frameworktür. Her CSS özelliği için küçük sınıflar sunarak tasarımcılara tam kontrol sağlar. Esnek ve özelleştirilebilir yapısıyla, özellikle minimalist ve kullanıcı odaklı projeler için uygundur. Kullanıcıların kendi tasarımlarını hızlı bir şekilde oluşturmasına imkan tanıyan bu framework, projelerde maksimum esneklik sağlar.

Foundation

Foundation, profesyonel düzeyde bir framework olup, özellikle karmaşık ve performans odaklı projelerde tercih edilir. Esnek bir grid sistemi ve güçlü bileşenleri ile dikkat çeken Foundation, erişilebilirlik odaklı projelerde de yaygın olarak kullanılır. Büyük ölçekli ve detaylı projelerde sağlam bir altyapı sunar.

Bulma

Bulma, hafif ve modern yapısıyla öne çıkan bir CSS frameworküdür. Kullanımı oldukça kolay olan Bulma, özellikle mobil uyumluluğa odaklanır. Hızlı ve şık tasarımlar yapmak isteyenler için ideal bir seçimdir. Sade bir yapıya sahip olduğu için öğrenilmesi ve uygulanması oldukça basittir.

Bu frameworklerin her biri, farklı projeler ve gereksinimler için etkili çözümler sunar. Bootstrap kapsamlı projelerde tercih edilirken, Tailwind CSS esneklik gerektiren tasarımlar için uygundur. Foundation, profesyonel düzeyde karmaşık projelerde kullanılırken, Bulma daha hafif ve hızlı uygulamalar için tercih edilebilir.

CSS FRAMEWORKLERİN AVANTAJLARI VE DEZAVANTAJLARI

CSS frameworkler, web geliştirme sürecinde geliştiricilere önemli kolaylıklar sağlayan araçlardır. Bu araçlar, projelerde hız, standartlaşma ve kullanım kolaylığı sunarak sıklıkla tercih edilir. Frameworkler sayesinde, karmaşık düzenler hızlı bir şekilde oluşturulabilir ve manuel kodlama ihtiyacı azalır. Ayrıca, responsive tasarım özellikleri sayesinde farklı ekran boyutlarında tutarlı ve etkileyici tasarımlar oluşturmak kolaylaşır. Frameworklerin tüm projede standart bir tasarım dilini koruması, uyumlu ve düzenli bir yapı sunar. Bunun yanında, kapsamlı dokümantasyon ve geniş topluluk desteği, hem öğrenme hem de sorun çözme süreçlerini kolaylaştırır. Teknik bilgi seviyesi düşük

kullanıcılar bile, frameworklerin hazır bileşenlerini kullanarak şık ve işlevsel tasarımlar yapabilir. Özellikle ekip projelerinde, aynı frameworkün kullanılması kodun anlaşılabilirliğini artırır ve düzenli bir iş akışı sağlar.
Ancak, CSS frameworklerin bazı dezavantajları da bulunmaktadır. Çoğu framework, projenin ihtiyaç duymadığı birçok bileşen içerir ve bu durum sayfa boyutunu artırarak performans sorunlarına yol açabilir. Standart yapılar, özgün tasarımlar oluşturmayı zorlaştırabilir ve özelleştirme için ekstra çaba gerektirebilir. Ayrıca, bir frameworke bağımlı olmak, uzun vadede esneklik kaybına neden olabilir ve frameworkte yapılan güncellemeler projeyi etkileyebilir. Yeni başlayanlar için frameworklerin yapısını ve terminolojisini öğrenmek zaman alıcı olabilirken, gereksiz kodların yüklenmesi düşük hızlı internet bağlantılarında performansı olumsuz etkileyebilir.
Tüm bu avantajlar ve dezavantajlar göz önünde bulundurularak, bir CSS framework seçerken projenin ihtiyaçları dikkatlice değerlendirilmelidir. Frameworkler, hızlı ve düzenli bir geliştirme süreci sunmalarına rağmen, özelleştirme ve bağımlılık gibi zorluklar yaratabileceğinden, doğru seçim yapmak başarı için kritik öneme sahiptir.

CSS FRAMEWORK KULLANIMININ TEMEL ADIMLARI

CSS frameworklerin verimli bir şekilde kullanılması, doğru adımların dikkatlice uygulanmasıyla mümkündür. Bu süreç, frameworkün projeye dahil edilmesinden yapısını anlama ve özelleştirme aşamalarına kadar uzanır. İlk olarak, projenin gereksinimlerine uygun bir framework seçilmelidir; örneğin, Bootstrap kapsamlı projeler için uygunken, Tailwind CSS daha minimal ve özelleştirilebilir seçenekler sunar. Framework genellikle bir CDN bağlantısı, yerel dosyalar ya da npm gibi paket yöneticileri kullanılarak projeye entegre edilir. Daha sonra, framework dosyalarının düzenli bir yapıda organize edilmesi, projenin yönetimini kolaylaştırır. CSS frameworklerin sunduğu önceden tanımlı sınıflar, tasarım sürecini hızlandırır. Örneğin, Bootstrap'in .container sınıfı merkezi bir düzen sağlarken, Tailwind CSS'in bg-blue-500 sınıfı arka plan rengini hızlıca ayarlamaya olanak tanır. Sınıfların anlamını ve işlevini anlamak, frameworkün sunduğu avantajlardan tam anlamıyla yararlanmayı sağlar ve projelerde maksimum verimlilik elde edilmesine katkıda bulunur.

CSS FRAMEWORKLERİN TASARIM ÜZERİNDEKİ ETKİSİ

CSS frameworkler, web tasarım süreçlerinde hem görsel hem de işlevsel açıdan önemli katkılar sunar. Tasarımcıların iş yükünü azaltırken, projelerde tutarlılık ve hız sağlar. Hazır şablonlar ve bileşenler sayesinde, birkaç saat içinde web sayfalarının temel yapısı oluşturulabilir, bu da zamandan tasarruf sağlar. Responsive özellikleri, farklı cihazlarda tutarlı bir kullanıcı deneyimi sunarak memnuniyeti artırır. Ayrıca, standartlaştırılmış sınıflar ve yapılar, ekip içi uyumu kolaylaştırır ve projelerde düzen sağlar. Frameworkler, renkler ve düzenler gibi birçok tasarım öğesinin özelleştirilmesine imkan tanırken, gereksiz kod yükü nedeniyle performans sorunları yaratabilir. Ancak, iyi optimize edilmiş framework kullanımı bu sorunları minimize eder.

FRAMEWORK SEÇERKEN NELERE DİKKAT EDİLMELİDİR?

Bir CSS framework seçimi, web tasarım sürecinin kritik bir aşamasıdır ve doğru tercih, zaman ve kaynakların verimli kullanılmasını sağlar. Projenin gereksinimlerini net bir şekilde belirlemek bu süreçte önemlidir. Büyük projelerde Bootstrap gibi kapsamlı frameworkler tercih edilirken, minimalist tasarımlar için Tailwind CSS gibi hafif ve esnek frameworkler daha uygundur. Ekip üyelerinin seçilen framework konusunda bilgi ve deneyime sahip olması, öğrenme süresini kısaltır ve ekip uyumunu artırır.
Seçilen frameworkün responsive tasarımı desteklemesi, günümüz mobil odaklı dünyasında büyük bir gerekliliktir. Ayrıca, renkler, yazı tipleri gibi özelliklerin özelleştirilebilmesi, markaya özgü tasarımlar için büyük bir avantajdır. Performans açısından, gereksiz kod yükü barındırmayan optimize frameworklerin seçilmesi önemlidir.
Son olarak, kapsamlı belgeler ve geniş topluluk desteği sunan popüler frameworkler, öğrenme ve sorun giderme süreçlerini kolaylaştırır. Sık güncellemeler ve aktif destek, frameworkün uzun vadede sürdürülebilirliğini sağlar.

BOOTSTRAP İLE TASARIMA GİRİŞ

Bootstrap, günümüzde en popüler ve yaygın kullanılan CSS frameworklerinden biridir. Twitter tarafından geliştirilen bu framework, responsive ve mobil uyumlu tasarımlar oluşturmak için güçlü araçlar sunar. Tasarım sürecini hızlandırmak amacıyla hazır bir grid sistemi, bileşenler ve kolay özelleştirme seçenekleri sağlar. Bootstrap projelerde en hızlı şekilde, HTML dosyasına bir CDN bağlantısı eklenerek kullanılabilir.
Bootstrap'in en önemli özelliklerinden biri, 12 sütunlu grid sistemidir. Bu sistem, farklı ekran boyutlarına uyum sağlayarak esnek ve mobil uyumlu düzenler oluşturmayı kolaylaştırır. Grid sistemi, sütun genişliklerinin col-sm-, col-md- gibi sınıflarla özelleştirilebilmesine olanak tanır, böylece çeşitli ekran boyutlarına göre dinamik tasarımlar yapılabilir.
Ayrıca Bootstrap, tasarım sürecini hem estetik hem de işlevsellik açısından kolaylaştıran birçok hazır bileşen sunar. Bu bileşenler arasında butonlar, kartlar, navigasyon çubukları, modallar ve uyarılar gibi araçlar bulunur. Örneğin, butonlar farklı renk ve boyutlarda özelleştirilebilirken, kartlar metin, başlık

ve görselleri düzenli bir şekilde bir araya getirir. Navigasyon çubukları mobil uyumlu ve şık menüler oluşturmak için kullanılırken, modallar kullanıcı etkileşimlerini ve bilgi görüntülemeyi kolaylaştırır. Ayrıca sekmeler, ilerleme çubukları, listeler ve açılır menüler gibi araçlar, projelerin ihtiyaç duyduğu düzenleme ve etkileşimleri sağlar.
Bootstrap'in sunduğu bu araçlar ve bileşenler, kullanıcıların projelerini hızlı bir şekilde geliştirmesine ve işlevsel bir yapıya kavuşturmasına olanak tanır. Özellikle responsive tasarımlar için optimize edilmiş bu framework, modern web geliştirme süreçlerinde önemli bir yer tutar.

CSS3 Animasyonlarının Temel Yapısı

CSS3 animasyonları, modern web tasarımında kullanıcı deneyimini geliştirmek için kullanılan en etkili araçlardan biridir. Web sitelerinde daha dinamik ve etkileyici görseller oluşturmak için @ keyframes kuralları ve CSS özellikleri bir arada kullanılır. Bu animasyonlar sadece görsel açıdan etkileyici olmakla kalmaz, aynı zamanda kullanıcıların dikkatini çekmek, gezinmelerini kolaylaştırmak ve genel olarak daha sezgisel bir deneyim sunmak için önemli bir rol oynar. CSS3 animasyonlarının temel yapısı, iki ana bileşene dayanır. İlk olarak, @keyframes kuralları animasyonun başlangıç, bitiş ve ara karelerini tanımlar. Bu kurallar bir öğenin belirli bir zaman çizelgesindeki hareketini veya dönüşümünü kontrol eder. İkinci olarak, bu kuralları HTML öğelerine uygulamak için gerekli olan CSS özellikleri kullanılır. Bu yapı, hem basit hem de karmaşık animasyonların oluşturulmasına olanak tanır.

@keyframes Kuralları

CSS3 animasyonlarının temel bileşenlerinden biri olan @keyframes, bir öğenin zaman içinde nasıl bir değişim göstereceğini detaylı bir şekilde tanımlama imkânı sunar. Bu kurallar, animasyonun başlangıç noktası (0%), bitiş noktası (100%) ve ara noktalarını (örneğin, 50%) içerir. Her bir keyframe, bir öğenin görünümü, konumu, saydamlığı ya da diğer özellikleri üzerinde değişiklikler yapabilir. Örneğin, bir nesne önce tamamen saydam bir şekilde yukarıda başlayabilir, ardından ekrana doğru hareket ederken yavaşça görünür hale gelebilir. Bu süreç boyunca belirli hareket geçişleri eklenebilir. Karmaşık geçişler oluşturmak için birden fazla ara nokta tanımlanarak öğenin hareket yolu özelleştirilebilir. Böylece animasyon, dikkat çekici ve akıcı bir görsellik sunar.

Animasyon Özellikleri

@keyframes ile tanımlanan kurallar, CSS animasyon özellikleriyle HTML öğelerine uygulanır. Bu özellikler, animasyonun nasıl ve hangi hızla gerçekleşeceğini tanımlar. animation-name, uygulanacak @keyframes animasyonunun adını belirlerken; animation-duration, animasyonun ne kadar süreceğini ifade eder. Örneğin, bir öğe iki saniyede bir hareketi tamamlayabilir. animation-timing-function, animasyonun hız eğrisini kontrol eder; ease, linear, ease-in ve ease-out gibi değerlerle hız ayarlamaları yapılabilir. Ayrıca, animation-iteration-count ile bir animasyonun kaç kez tekrar edeceği belirlenir. Sonsuz tekrar için infinite değeri kullanılabilir. Bu özellikler bir arada kullanılarak görsel açıdan etkileyici ve kusursuz bir deneyim sağlanabilir.

CSS3 Animasyonlarının Avantajları

CSS3 animasyonları, web tasarımında yalnızca görselliği artırmakla kalmaz, aynı zamanda performans açısından da önemli avantajlar sunar. Tarayıcılar tarafından optimize edilen bu animasyonlar, GPU hızlandırması sayesinde çok daha akıcı bir deneyim sunar. JavaScript gibi alternatif yöntemlere kıyasla daha hafif ve kullanımı kolaydır. Kod okunabilirliği yüksek olduğundan, tasarım sürecini hızlandırır ve bakım maliyetlerini düşürür. Bu avantajlar, CSS3 animasyonlarını modern web tasarımında vazgeçilmez bir araç haline getirir.

İleri Düzey Kullanım

CSS3 animasyonları, basit geçişlerin ötesine geçerek oldukça karmaşık hareketlerin oluşturulmasına olanak tanır. Ara noktalar (keyframe) eklenerek bir animasyonun akışı hassas bir şekilde kontrol edilebilir. Örneğin, bir öğe önce hızlı bir şekilde büyüyebilir, ardından yavaşlayarak orijinal boyutuna dönebilir. Cubic-bezier fonksiyonu, hız eğrilerinin kişiselleştirilmesi için güçlü bir araç sunar. Bu sayede, animasyonlar yalnızca görsel açıdan etkileyici olmakla kalmaz, aynı zamanda daha doğal ve akıcı bir hareket sunar. Özelleştirme seçenekleri, animasyonları yalnızca bir araç değil, aynı zamanda tasarım sürecinin ayrılmaz bir parçası haline getirir.

Transform ve Transition Özellikleri

CSS3'ün transform ve transition özellikleri, dinamik ve akıcı animasyonların temelini oluşturur. Transform, bir öğenin konumunu, boyutunu, dönüş açısını veya eğimini değiştirmek için kullanılır. Örneğin, rotate özelliği bir öğeyi belirli bir açıyla döndürebilirken, scale özelliği büyütme veya küçültme işlemlerini gerçekleştirir. Transition, bu değişikliklerin akıcı bir şekilde gerçekleşmesini sağlar. Bir öğenin rengi, konumu veya boyutu, transition kullanılarak belirli bir süre boyunca yumuşak geçişlerle değiştirilebilir. Örneğin, bir düğmenin üzerine fareyle gelindiğinde, rengin aniden değil, birkaç saniye içinde değişmesi sağlanabilir. Bu özellikler, web sitelerinde kullanıcı etkileşimini güçlendirir.

3D Animasyonlar

CSS3, 3D animasyonlarla tasarımlara derinlik ve gerçekçilik katar. translate3d, scale3d ve rotateX gibi özellikler, bir nesneyi x, y ve z eksenlerinde hareket ettirme, döndürme veya ölçekleme imkânı sunar. Bu özelliklerle tasarımlar daha interaktif ve etkileyici hale getirilebilir. 3D animasyonlarda perspektif kullanımı da önemlidir; perspektif, bir nesnenin gözlemciden uzaklaştıkça küçülmesi gibi doğal bir etki yaratır. Bu özellikler, CSS3'ün modern web tasarımındaki gücünü ortaya koyar.

JavaScript ile Entegrasyon

CSS3 animasyonları, JavaScript ile entegre edildiğinde çok daha güçlü ve kullanıcı odaklı hale gelir. JavaScript, animasyonları belirli olaylarla (örneğin, tıklama, hover) tetikleyebilir, durdurabilir veya özelleştirebilir. Ayrıca, animationstart, animationend ve animationiteration gibi olaylar, animasyonun ne zaman başladığını, bittiğini veya tekrarladığını izlemek için kullanılabilir. JavaScript, animasyon süreçlerini kontrol etmek ve gerçek zamanlı kullanıcı etkileşimleri yaratmak için ideal bir araçtır.

Kullanıcı Etkileşimi

CSS3 animasyonları, kullanıcı etkileşimlerini güçlendirmek için etkili bir araçtır. Fareyle üzerine gelindiğinde (hover) ya da bir düğmeye tıklandığında animasyonların tetiklenmesi mümkündür. Örneğin, bir düğmeye tıklandığında düğme hareket edebilir, rengini değiştirebilir veya boyutunu büyütebilir. Bu etkileşimler, kullanıcıların bir web sitesine olan ilgisini artırır.

DOM MANİPÜLASYONU

DOM (Document Object Model), bir web sayfasının yapısını, stilini ve içeriğini temsil eden programlanabilir bir arayüzdür. Tarayıcı tarafından HTML veya XML belgesinden oluşturulur. DOM, belgeyi bir ağaç yapısı olarak modelleyerek geliştiricilerin sayfa içeriğini dinamik olarak değiştirmesine olanak tanır.

Öğeleri eklemek, güncellemek, silmek ve dinamik stiller uygulamak, modern web geliştirme sürecinin temel taşlarındandır.

DOM Ağacı Yapısı

DOM, düğümlerden oluşan bir ağaç yapısıdır. Bu yapı, sayfanın her bir elemanını birer düğüm (node) olarak temsil eder. Düğümler üç ana gruba ayrılır.

Eleman Düğümleri (Element Nodes): HTML etiketleri (örneğin veya). Bu düğümler, sayfadaki yapı taşlarını oluşturur ve diğer düğümleri içerebilir. Metin Düğümleri (Text Nodes): Etiketler içindeki metinleri temsil eder. Bir eleman düğümünün içeriği varsa, bu içerik metin düğümü olarak DOM ağacında

Öznitelik Düğümleri (Attribute Nodes): HTML etiketlerinin niteliklerini temsil eder.

DOM ağacı yapısını anlamak, elemanlar arasındaki ilişkileri ve her düğümün nasıl manipüle edilebileceğini kavramak açısından kritik öneme sahiptir. Alt üst ve aynı seviyede yer alan elemanlarını arasında bağlantılar göre işlemler yapılabilir.

DOM Elemanlarına Erişim Yöntemleri

DOM üzerindeki elemanlara erişmek için çeşitli yöntemler kullanılır. Bu yöntemler, elemanları benzersiz tanımlayıcılarla veya genel özelliklere göre seçmeyi sağlar.

“document” nesnesi, JavaScript'in DOM ile etkileşime geçmesini sağlayan temel bir nesnedir. Bu nesne, sayfanın içeriğini temsil eder ve DOM üzerinde işlem yapabilmek için kullanılan tüm yöntem ve özellikleri içerir. “document“ nesnesi global bir nesnedir ve tüm tarayıcılarda otomatik olarak sağlanır.

Id'ye göre seçim

getElementById() Id'ye göre seçim yapar. Bu yöntem, sayfa içerisinde benzersiz olan bir Id değeri üzerinden doğrudan ilgili HTML elemanına erişim ağlar. Id, bir HTML elemanını tanımlamak için kullanılır ve her sayfada yalnızca bir kez kullanılmalıdır.

Bu yöntem, oldukça fazla kullanılır çünkü doğrudan benzersiz bir eleman üzerinden işlem yapar. Projelerde sıkça kullanılmaktadır. Ancak Id'nin benzersiz olması gerektiği unutulmamalıdır.

Sınıf adına göre seçim

getElementsByClassName() sınıf adına göre seçim yapar. Sayfa içerisinde aynı sınıf adına sahip birden fazla eleman bulunabilir. Bu yöntem, tüm eşleşen elemanları bir HTMLCollection (HTML koleksiyonu) içinde döndürür. Bu koleksiyon, diziye benzer bir yapıya sahiptir ancak doğrudan dizi metodlarını desteklemez.

Etiket adına göre seçim

getElementsByTagName() etiket adına göre seçim yapar. Sayfadaki tüm belirli etiket türlerini seçmek için kullanılır.

CSS seçicileri

querySelector() ve querySelectorAll() CSS seçicileri kullanarak eleman seçer. querySelector(), ilk eşleşeni döndürürken, querySelectorAll() tüm eşleşenleri NodeList olarak döndürür.

CSS seçicileri sayesinde daha esnek ve güçlü seçim işlemleri gerçekleştirilmiştir.

CSS seçicileri, HTML öğelerini hedeflemek için kullanılır. Evrensel (*), tag (div), sınıf (.class), Id (#id) seçiciler temel seçeneklerdir. Alt öğe (div p), çocuk (div > p), komşu (h1 + p), genel kardeş (h1 ~ p), grup (h1, p), ve özellik seçicileri ([type="text"]) daha spesifik hedefleme sağlar.

Teknikler ve Araçlar

DOM manipülasyonu, web sayfalarının içerik ve yapısının dinamik olarak değiştirilmesine olanak tanır ve modern web geliştirme süreçlerinde kritik bir rol oynar.

İçerik eklemek

textContent, bir öğenin metin içeriğini ayarlamak veya almak için kullanılır. Bu yöntem, yalnızca düz metin eklemek amacıyla kullanılır ve eklenen metin, HTML etiketleri içeriyorsa bunları düz metin olarak gösterir.

innerHTML, bir öğenin içeriğini HTML olarak ayarlamak veya almak için kullanılır. Bu yöntem, HTML etiketlerini yorumlayarak sayfaya ekler.

Yeni öğe oluşturma

Yeni bir DOM öğesi oluşturmak için document.createElement yöntemi kullanılır.

appendChild() yöntemi, bir öğeyi belirtilen ebeveynin (parent node) sonuna eklemek için kullanılır. Eklenen öğe, ebeveynin son çocuğu (child) olur.

insertBefore() yöntemi, bir öğeyi, belirli bir referans öğeden öncesine veya sonrasına eklemek için kullanılır. Bu yöntem, öğeleri belirli sıralamalara göre yerleştirmek gerektiğinde tercih edilir.

append() yöntemi, bir öğeyi ebeveynin sonuna eklemek için kullanılır. appendChild() yönteminden farklı olarak, birden fazla öğe veya metin eklemeye de izin verir.

prepend() yöntemi, bir öğeyi ebeveynin başına eklemek için kullanılır. append()'in tersidir ve özellikle listeleme işlemlerinde öncelikli eklemeler için kullanılır.

Öğe Silme

remove() metodu, JavaScript'te bir DOM öğesini belge ağacından kaldırmanın en basit ve doğrudan yoludur.

removeChild() metodu, bir ebeveyn öğe üzerinden belirli bir alt öğeyi (child node) DOM'dan kaldırmak için kullanılır. Bu yöntem, ebeveyn-çocuk ilişkisi gerektirdiği için öncelikle ebeveyn öğeye ve ardından kaldırılacak alt öğeye erişilmesi gerekir.

Dinamik stil uygulamak

DOM manipülasyonu, HTML öğelerinin içeriğini, yapısını ve stillerini dinamik olarak değiştirme imkânı sunar. Özellikle JavaScript'in style özelliği, bir öğenin CSS özelliklerini doğrudan değiştirmek veya güncellemek için kullanılır.

JavaScript'in style özelliği, bir HTML öğesinin CSS stil özelliklerine doğrudan erişim sağlar. Bu özellik, bir öğeye inline (satır içi) stil eklemek veya var olan stilleri değiştirmek için kullanılır.

Sınıfları yönetmek

JavaScript'in classList özelliği, bir öğenin sınıflarını yönetmek için kullanılır. Bu yöntem, doğrudan stil eklemek yerine, daha yönetilebilir ve yeniden kullanılabilir CSS sınıflarıyla çalışmayı sağlar.

classList, bir öğenin class attribute (sınıf niteliği) üzerinde işlem yapmak için çeşitli metodlar sağlar ve öğeye birden fazla sınıf eklenmesine olanak tanır.

classList.add() –Sınıf Ekleme

Bu yöntem, öğeye bir veya daha fazla sınıf eklemek için kullanılır. Eğer eklenmek istenen sınıf zaten mevcutsa, tekrar eklenmez.

classList.remove() – Sınıf Kaldırma

Bu yöntem, öğeden bir veya daha fazla sınıfı kaldırmak için kullanılır.

classList.toggle() – Sınıf Ekleme/Kaldırma

Bu yöntem, belirtilen sınıf öğede mevcutsa kaldırır, yoksa ekler. Genellikle bir durumu açıp kapatmak için kullanılır

classList .contains() – Sınıf Kontrolü

Bu yöntem, belirtilen sınıfın öğede mevcut olup olmadığını kontrol eder ve true veya false döndürür.

classList.replace() – Sınıf Değiştirme

Bu yöntem, bir sınıfı başka bir sınıfla değiştirmek için kullanılır.

Attribute (nitelik) yönetmek

HTML öğelerine attribute (nitelik) eklemek, değiştirmek veya silmek, JavaScript ile dinamik olarak gerçekleştirilebilir.

Nitelik Ekleme veya Değiştirme

setAttribute(), bir öğeye yeni bir nitelik eklemek veya var olan bir niteliğin değerini değiştirmek için kullanılır. Eğer belirtilen nitelik zaten varsa, değeri güncellenir.

Adı ve değeri olarak parametre iki parametre alır.

Nitelik Değerini Okuma

getAttribute(),bir öğenin belirtilen niteliğinin değerini döndürür.

Nitelik Silme

removeAttribute(), bir öğeden belirli bir niteliği kaldırmak için kullanılır.

Nitelik Kontrolü

hasAttribute() ,bir öğenin belirli bir niteliğe sahip olup olmadığını kontrol eder ve true veya false döndürür.

Olay Yönetimi

JavaScript'te Event Handling (Olay Yönetimi), kullanıcı etkileşimlerini algılayarak belirli işlemleri gerçekleştirmek için kullanılır. Bir olay (event), fare tıklaması, klavye tuşuna basılması, bir formun gönderilmesi veya bir sayfanın yüklenmesi gibi tarayıcıda meydana gelen işlemleri ifade eder.

Event Listener Yöntemleri

Event Handling, bu olayları dinlemek (event listener) ve gerçekleştiğinde belirli bir işlevi tetiklemek için kullanılır

Dinleyicisi eklemek
addEventListener() yöntemi, JavaScript'te bir HTML elementine olay dinleyicisi eklemek için kullanılır. Bu yöntem, bir olay (örneğin tıklama, fare üzerine gelme) gerçekleştiğinde ne yapılacağını belirlemek için kullanılır.
element.addEventListener(olay, fonksiyon, useCapture);
event: Dinlenecek olay türü (örn. 'click', 'mouseover').
function: Olay gerçekleştiğinde çalıştırılacak işlev.
useCapture: Boolean (true/false) değer alır ve olayın hangi aşamada ele alınacağını kontrol eder. Varsayılan değeri false'tur.
Yakalama Aşaması useCapture: true
Olay, en dıştaki öğeden başlayarak hedef öğeye doğru ilerler.
Kabarcıklanma Aşaması useCapture: false
Olay, hedef öğeden başlayarak en dıştaki öğeye doğru geri döner.
Dinleyicisini kaldırmak
removeEventListener() yöntemi, Bir öğeye eklenmiş olan bir olay dinleyicisini kaldırmak için kullanılır.
Olay Türleri
Olayların geçekleşmesi için etkileşimlerin olması gerekir. Bu etkileşimler fare, klavye gibi giriş cihazlarında veya form, pencere olaylarının sonucunda olabilir.
Fare Olayları
JavaScript, fare ile yapılan etkileşimleri algılamak ve işlemek için bir dizi fare olayı (mouse event) sağlar. Bu olaylar, kullanıcıların fareyi kullanarak gerçekleştirdikleri hareketler veya tıklamalar sırasında tetiklenir.
Tek tıklama olayı
click olayı, bir öğeye fare ile tek tıklama yapıldığında tetiklenir. Bu, en sık kullanılan olaylardan biridir ve düğmeler, bağlantılar veya formlar gibi öğeler üzerinde yaygın olarak kullanılır.
Çift tıklama olayı
dblclick olayı, bir öğeye çift tıklama yapıldığında tetiklenir. İki tıklama arasında kısa bir süre olmalıdır; aksi takdirde olay tetiklenmez.
Üzerine gelme olayı
mouseover olayı, fare imleci bir öğenin üzerine geldiğinde tetiklenir. Ancak, alt öğelere geçildiğinde de tetiklenir.
Üzerinden ayrılma olayı
mouseout olayı, fare imleci bir öğenin üzerinden ayrıldığında tetiklenir. Alt öğelere geçildiğinde de tetiklenir.
Fare tuşuna basılma olayı
mousedown olayı, fare düğmesine basıldığı anda tetiklenir. Bu olay, fare düğmesi bırakılmadan önce işlenir.
Fare tuşu serbest bırakılma olayı
mouseup olayı, fare düğmesinin bırakıldığı anda tetiklenir. Genellikle mousedown olayıyla birlikte kullanılır.
Fare hareketi olayı
mousemove olayı, fare imleci hareket ettiğinde sürekli olarak tetiklenir.
Fare olaylarında event özellikleri
Her fare olayı bir event oluşturur ve bu nesne, olay hakkında bilgi taşır.
clientX ve clientY: Sayfa üzerindeki fare koordinatlarını döndürür.
pageX ve pageY: Tüm döküman içindeki koordinatları verir.
target: Olayın gerçekleştiği öğeyi döndürür.
button: Basılan fare düğmesini belirler (0: sol, 1: orta, 2: sağ).
altKey, ctrlKey, shiftKey: Basılı olan tuşları kontrol eder.
Klavye Olayları
Klavye olayları (keyboard events), kullanıcının klavye tuşlarına basmasıyla tetiklenir.
keydown olayı, bir tuşa basıldığı anda tetiklenir ve tuş serbest bırakılana kadar tetiklenmeye devam etmez. Tuşun fiziksel olarak basılması algılanır.
Tuş serbest bırakıldığında
keyup olayı, bir tuşun bırakıldığı anda tetiklenir. Tuşa ne kadar süreyle basılı tutulduğu önemli değildir; yalnızca serbest bırakma işlemi algılanır.
keypress olayı, bir tuşa basıldığında tetiklenir bir olaydır. Modern tarayıcılar bu olayı artık kullanmayı önermemektedir. Bunun yerine keydown veya keyup kullanılmalıdır. Bazı tarayıcılar hala destekliyor olsa da, ilgili web standartlarından kaldırılmış olabilir.
Klavye olaylarında event özellikleri
key: Tuşun gerçek adını döndürür. Harfler küçük harf olarak döndürülür (örn. 'a').

code: Tuşun fiziksel kodunu döndürür. Klavyedeki yerini belirtir (örn. 'KeyA', 'ArrowUp'). Digit1, Numpad1 rakamla "1" değerini karşılık gelir. Digit1 harflerin üzerinde yer alan "1" iken Numpad1 klavyenin sonunda yer alan numaralara karşılık gelir.
altKey, ctrlKey, shiftKey: Basılı olan özel tuşları kontrol eder. true veya false döndürür.

Form Olayları

Form olayları, HTML formlarıyla etkileşimde bulunulduğunda tetiklenen olaylardır. Kullanıcıların form alanlarına veri girmesi, değiştirmesi, odaklanması ve formu göndermesi gibi işlemler bu olaylarla takip edilebilir.

Form gönderme olayı

submit olayı, bir formun gönderilmesi sırasında tetiklenir. Form gönderimi, varsayılan olarak tarayıcı tarafından ele alınır ve sayfa yeniden yüklenir. Ancak JavaScript ile bu varsayılan davranış durdurulabilir ve özel işlemler gerçekleştirilebilir. JavaScript form olayları, kullanıcı etkileşimlerini algılayarak veri doğrulama, dinamik form kontrolleri ve geri bildirim mekanizmaları oluşturmak için güçlü bir araçtır.
event.preventDefault() ile sayfanın yeniden yüklenmesi engellenir.

Değer değiştirme olayı

change olayı, bir form elemanının değeri değiştirildiğinde tetiklenir. Değişiklik, kullanıcı öğeden çıkınca algılanır. Bu olay, özellikle öğeleri, onay kutuları ve radyo düğmeleri gibi kontrol elemanları için kullanışlıdır.

Odaklanma olayı

focus olayı, bir form alanına odaklanıldığında (seçildiğinde) tetiklenir. Kullanıcı bir metin kutusuna tıkladığında veya Tab tuşuyla bu kutuya geçildiğinde etkinleşir. Odak, genellikle kullanıcının form alanını düzenlemeye hazır olduğunu gösterir.

Odaktan çıkma olayı

blur olayı, bir form alanından odak kaybolduğunda (kullanıcı başka bir alana geçtiğinde) tetiklenir. Bu olay, çoğunlukla form doğrulama ve veri kontrolü yapmak için kullanılır.

Dinamik veri girişi olayı

input olayı, bir form alanında her değişiklik gerçekleştiğinde tetiklenir. Metin girildikçe veya silindikçe sürekli çalışır.

Seçim olayı

select olayı, bir form alanındaki metin seçildiğinde tetiklenir.

Form sıfırlama olayı

reset olayı, bir formun sıfırlanması sırasında tetiklenir.

Pencere Olayları

Pencere olayları (Window Events), bir web sayfasının tarayıcıda yüklenmesi, boyutunun değiştirilmesi, kaydırılması veya kapatılması gibi durumlarda tetiklenir. Bu olaylar, tarayıcıdaki genel sayfa durumlarını veya kullanıcı hareketlerini algılamak ve uygun tepkiler vermek için kullanılır.

Sayfa yüklenme olayı

load olayı, tüm sayfa içeriği (HTML, CSS, JavaScript, resimler ve diğer kaynaklar) tarayıcı tarafından tamamen yüklendiğinde tetiklenir. Bu olay, sayfanın tüm elemanlarının hazır olduğundan emin olmak için kullanılır.
DOMContentLoaded olayı, Sayfa içeriği (HTML) tamamen yüklendiğinde tetiklenir (CSS ve resimler henüz yüklenmemiş olabilir).
DOMContentLoaded olayı, yalnızca HTML'nin yüklenmesini beklerken, load tüm kaynakların yüklenmesini bekler.
Load özellikle resim, video veya büyük dosyaların tam yüklenmesini beklemek için kullanışlıdır.
Pencere boyutunun değiştirilmesi olayı
resize olayı, tarayıcı penceresi boyut değişikliği algılandığında tetiklenir.
window.innerWidth ve window.innerHeight ile pencerenin genişliği ve yüksekliği alınabilir.

Sayfa kaydırma olayı

scroll olayı, bir sayfa veya öğe kaydırıldığında tetiklenir. r.
window.scrollY ve window.scrollX, sayfanın dikey ve yatay kaydırma miktarlarını verir.

Sayfa kapatılma olayı

unload olayı, bir kullanıcı sayfadan ayrılmak üzereyken veya pencere kapatılırken tetiklenir. Bu olay, genellikle kullanıcıların sayfadan çıkmadan önceki durumlarını kaydetmek için kullanılır.

ECMAScript NEDİR
ECMAScript, JavaScript dilinin standartlaştırılmış versiyonudur ve web tarayıcıları arasında uyumluluğu sağlamak amacıyla geliştirilmiştir.
1996 yılında Netscape, JavaScript'in standartlaştırılması için Ecma International'a başvurmuş ve bu süreç sonucunda ECMAScript adı ortaya çıkmıştır.
ECMAScript 6 (ES6)
JavaScript diline önemli yenilikler ve iyileştirmeler getirerek, dilin modern uygulama geliştirme ihtiyaçlarına daha iyi cevap vermesini sağlamıştır. ES6, Haziran 2015'te yayımlanmış ve JavaScript dilinde büyük bir dönüm noktası olmuştur.
Değişkenler
JavaScript'te değişkenler, veri saklamak ve bu verilere daha sonra erişmek için kullanılan temel yapı taşlarıdır. Değişkenler, programlamada veriyi dinamik bir şekilde işlemek için gereklidir.
JavaScript'te değişkenler üç anahtar kelimeyle tanımlanır: var, let ve const.
let
let, blok düzeyinde kapsam sağlar ve yeniden tanımlamaya izin verir. Bu, genellikle değişkenin değerinin değişmesi gerektiğinde kullanılır.
const
const, sabit değişkenler için kullanılır. const ile tanımlanan bir değişkenin değeri sonradan değiştirilemez, bu nedenle sabit veri veya referanslar için idealdir. Ancak, bir const nesnesinin içeriği (örneğin, bir dizinin elemanları veya bir nesnenin özellikleri) değiştirilebilir.
Değişken fonksiyonları
Metin İşlemleri
includes() metodu, bir metinin başka bir metini içerip içermediğini kontrol eder. true veya false değerini döndürür.
startsWith() metodu, bir metinin belirli bir metinle başlayıp başlamadığını kontrol eder. true veya false değerini döndürür.
endsWith() metodu, bir metinin belirli bir metinle bitip bitmediğini kontrol eder. true veya false değerini döndürür.
Matematik (Math) İşlemleri
JavaScript'te Math nesnesi, matematiksel işlemleri gerçekleştirmek için çok çeşitli yerleşik metotlar ve sabitler sunar.
Math.trunc() metodu: Bir sayının ondalık kısmını keser ve sadece tam kısmını döndürür.
Math.sign() metodu, bir sayının pozitif, negatif veya sıfır olup olmadığını belirlemek için kullanılır.
Math.cbrt() metodu, bir sayının küp kökünü hesaplar. Negatif sayılar için de doğru sonuç döner.
Math.log2() metodu, bir sayının taban 2'ye (log2) göre logaritmasını hesaplar. Bu, "2'nin hangi kuvveti bu sayıyı verir?" sorusunun cevabını döner.
Math.log10() metodu, bir sayının taban 10'a göre logaritmasını hesaplar. Bu, "10'un hangi kuvveti bu sayıyı verir?" sorusunun cevabını döner.
isFinite() metodu, bir değerin sayı olup olmadığını kontrol eder. Sayılar için true, aksi halde false döner.
isNaN() metodu, bir değerin NaN (Not-a-Number) olup olmadığını kontrol eder. NaN için true, diğer durumlarda false döner.
Döngüler
JavaScript'te döngüler, belirli bir kod bloğunu birden çok kez çalıştırmak için kullanılır ve veri işleme işlemlerini kolaylaştırır.Modern JavaScript'te, for...of ve for...in gibi döngüler, diziler ve nesneler üzerinde gezinmek için idealdir.
for...of döngüsü
for...of döngüsü, JavaScript'te yineleyebilir yapılar üzerinde gezinmek için kullanılan modern ve sade bir döngü türüdür. Özellikle diziler, dizgeler (strings), maps ve kümeler (sets) gibi veri yapıları üzerinde çalışırken kullanışlıdır. Bu döngü, her bir öğeye sırasıyla erişim sağlar.
Fonksiyonlar
Fonksiyonlar, tekrar kullanılabilir kod blokları oluşturmak için kullanılan temel yapı taşlarıdır.
Arrow fonksiyonlar

Arrow fonksiyonlar, JavaScript'te daha kısa ve okunabilir bir sözdizimiyle fonksiyon tanımlamak için kullanılır. Geleneksel fonksiyonlara kıyasla, this bağlamını miras alır ve daha kısa bir yazımı vardır. Daha kısa bir sözdizimi sağlar. Basit işlemler ve kısa kod parçaları için idealdir.

Varsayılan Parametreli Fonksiyon

JavaScript'te varsayılan parametre değerleri, bir fonksiyon çağrılırken belirli bir parametre için bir değer verilmediğinde, bu parametreye otomatik olarak atanacak bir varsayılan değer tanımlamanızı sağlar.

Dinlenme (Rest) parametreli Fonksiyon

Dinlenme (Rest) parametreleri, bir fonksiyonun parametrelerine birden fazla değer geçildiğinde bu değerleri bir dizi içinde toplamak için kullanılan modern bir ES6 özelliğidir. Rest parametreleri, fonksiyonlarda dinamik sayıda argümanla çalışmayı kolaylaştırır.

Nesne ve Dizi

Nesneler, anahtar-değer (key-value) çiftleriyle veri saklamak için kullanılır. Bu yapı, ilişkili verileri organize etmek ve bir arada tutmak için idealdir.

Diziler, sıralı veri koleksiyonlarını saklamak için kullanılır ve her bir öğe bir indeksle (0'dan başlayan) tanımlanır.

Iterator, JavaScript'te, bir veri yapısının elemanlarını sırasıyla dolaşmayı (iterate etmeyi) sağlayan özel bir nesnedir. Iteratorlar, özellikle for...of döngüsü gibi yapılarla kullanılır ve her adımda veri yapısının bir sonraki elemanına erişmeyi mümkün kılar.

Object Destructuring

JavaScript'te bir nesnenin içindeki değerleri kolayca ayıklayıp değişkenlere atamak için kullanılan özelliğidir.

Array Destructuring

JavaScript'te bir dizinin içindeki değerleri kolayca ayıklayıp değişkenlere atamak için kullanılan özelliğidir.

Spread (...) Operatörü

JavaScript'te bir diziyi, nesneyi veya başka bir iterable (yineleyici) yapıyı açmak ve elemanlarına kolayca erişmek için kullanılan özelliğidir. Spread, özellikle kopyalama, birleştirme veya argümanları bir fonksiyona geçirme işlemlerinde oldukça kullanışlıdır.

Map

Map, JavaScript'te bir koleksiyon türüdür ve anahtar-değer çiftlerini saklamak için kullanılır. Map nesnesi, anahtarların türüne bakılmaksızın (string, number, object, hatta başka bir Map) herhangi bir veri türünü anahtar olarak kullanma yeteneğine sahiptir.

Map Özellikleri

Herhangi Bir Türde Anahtar: Map'te anahtar olarak hem nesneler hem de primitif değerler kullanılabilir.

Sıralı Saklama: Map, ekleme sırasını korur; yani elemanlar, eklendikleri sırayla geri alınır.

Tekrarlanan Anahtar Yok: Map, bir anahtar yalnızca bir kez kullanılabilir. Aynı anahtarı tekrar eklerseniz, önceki değer üzerine yazılır.

Özelleştirilmiş Boyut: map.size özelliği, Map'teki eleman sayısını verir.

Set

Set, bir veri yapısıdır. Bir Set, benzersiz değerlerden oluşan bir koleksiyon saklar. Yani bir Set içine aynı değeri birden fazla kez ekleyemezsiniz. Bu özellik, tekrarlayan değerlerin engellenmesi gerektiğinde Set'i ideal bir çözüm haline getirir.

Tekrarlayan Değer Yok: Bir Set'e aynı değer tekrar eklenmeye çalışılsa bile yalnızca bir kez saklanır.

Sırasız ve Benzersiz: Elemanlar, eklenme sırasına göre saklanır ancak benzersizdir.

Her Türde Değer Saklayabilir: Set, hem ilkel(string, number) hem de nesne türünde (object, array) değerleri saklayabilir.

Dizi fonksiyonları

entries() yöntemi, bir dizideki her elemanın [indeks, değer] çiftlerini içeren bir iterator (yineleyici) döndürür. İndeks ve değerlerin birlikte ele alınması gerektiği durumlarda kullanışlıdır.

Array.from() yöntemi, benzer dizi yapılarından (ör. bir dizi benzeri nesne veya iterable bir nesne) gerçek bir dizi oluşturur. Ayrıca, her bir eleman üzerinde dönüşüm uygulamak için bir haritalama işlevi kabul edebilir.

keys() yöntemi, bir dizinin sadece indekslerini içeren bir iterator döndürür. Dizinin elemanlarını değil, sadece indekslerini işlemek istediğiniz durumlarda faydalıdır.

find() yöntemi, bir dizideki belirli bir koşulu sağlayan ilk elemanı döndürür. Eğer hiçbir eleman koşulu sağlamıyorsa, undefined döner. Genellikle bir nesne veya değer ararken kullanılır.

findIndex() yöntemi, bir dizide belirli bir koşulu sağlayan ilk elemanın indeksini döndürür. Eğer hiçbir eleman koşulu sağlamıyorsa, -1 döner. find() ile benzer şekilde çalışır, ancak elemanın kendisini değil, indeksini döndürmesiyle ayrılır.

ECMAScript 6+ (ES6+)

ECMAScript (ES), JavaScript'in standartlaştırılmış bir sürümüdür ve her yeni sürümde dilin özelliklerini geliştiren yenilikler eklenir. ES6 (2015) sonrası sürümlerde birçok yeni özellik tanıtılmıştır.

ECMAScript 2016 (ES7)

Üst alma operatörü (**), JavaScript'te bir sayıyı belirli bir üsse yükseltmek için kullanılır. Bu operatör, matematikteki üs alma işlemini ifade eder ve Math.pow() metoduna alternatif olarak tanıtılmıştır. Operatör, tabanı sol tarafında, üssü sağ tarafında alır.

Üs alma ataması operatörü (**=), bir değişkenin mevcut değerini bir üsse yükseltmek için kullanılır.

Üs alma ataması operatörü (**=), bir değişkenin mevcut değerini bir üsse yükseltmek için kullanılır.

includes() metodu, bir dizide belirli bir değerin mevcut olup olmadığını kontrol etmek için kullanılır. Bu yöntem, boolean bir değer döndürür. Stringler ve diğer iterable yapılarda da kullanılabilir.

ECMAScript 2017 (ES8)

String padding, bir stringin başlangıcına veya sonuna belirli bir uzunluğu dolduracak şekilde karakterler eklemek için kullanılır.

padStart(targetLength, padString): Stringin başına belirtilen uzunluk kadar karakter ekler.

padEnd(targetLength, padString): Stringin sonuna belirtilen uzunluk kadar karakter ekler.

Eğer padString belirtilmezse varsayılan olarak boşluk (" ") kullanılır. Object.values(), bir nesnenin kendi sayılabilir değerlerini bir dizi olarak döndürür. Değerleri hızlıca işlemek için kullanışlıdır.

ECMAScript 2018 (ES9)

RegExp (Regular Expression) , belirli bir metin desenini tanımlamak ve bu desene uygun metni aramak, eşleştirmek veya değiştirmek için kullanılan bir araçtır. Regex, özellikle metin işlemede güçlü ve esnektir. Karakter grupları, tekrarlama, alternatifler ve özel sembollerle karmaşık desenler tanımlanabilir.

RegExp içinde gruplara isim vererek, sonuçlara daha anlamlı bir şekilde erişmeyi sağlar.

Bir ifadenin öncesini kontrol etmeyi sağlar.

ECMAScript 2019 (ES10)

trimStart() metodu, bir stringin başındaki boşluk karakterlerini kaldırır. trimLeft() ile aynı işlevselliğe sahiptir. Orijinal stringi değiştirmez, yeni bir string döndürür. Kullanıcı girdilerini işlerken baştaki gereksiz boşlukları temizlemek için idealdir.

trimEnd() metodu, bir stringin sonundaki boşluk karakterlerini kaldırır. trimRight() ile aynı işlevselliği sunar. Stringin sonunda gereksiz boşlukları kaldırmak için kullanılır.

try...catch bloğunda catch ifadesi için bir hata parametresi tanımlamak isteğe bağlı hale geldi. Bu, hatanın kullanılmadığı durumlarda daha temiz kod yazmayı sağlar.

flat() metodu, bir diziyi belirtilen derinliğe kadar düzleştirir. Varsayılan derinlik 1'dir.

flatMap() metodu, her eleman üzerinde bir dönüşüm gerçekleştirir ve ardından sonucu düzleştirir. map() ve flat() işlemlerini birleştirir.

sort() metodu artık kararlı sıralama (stable sort) sağlıyor. Bir anahtardaki öğeleri sıralarken, öğeler aynı anahtara sahip diğer nesnelere göreli konumlarını korumalıdır.

JSON.stringify() metodu, bir JavaScript değerini bir JSON dizesine dönüştürür; isteğe bağlı olarak bir değiştirici işlevi belirtilmişse değerleri değiştirir veya isteğe bağlı olarak bir değiştirici dizisi belirtilmişse yalnızca belirtilen özellikleri ekler.

ECMAScript 2020 (ES11)

Boş birleştirme operatörü (??), bir değişkenin değerini kontrol eder ve yalnızca null veya undefined olduğunda varsayılan bir değer döndürür. Bu operatör, boşluk veya false gibi değerleri null veya undefined ile karıştırmadan kontrol etmeyi kolaylaştırır.

İsteğe bağlı zincirleme operatörü (?.), bir nesne veya dizide derinlemesine erişim sırasında bir özellik mevcut değilse hata yerine undefined döndürür.

Mantıksal VE Atama (&&=), bir değişkenin mevcut değerini true olarak değerlendirirse belirli bir değeri atamak için kullanılır.

Mantıksal VEYA Atama (||=), bir değişkenin mevcut değeri false, null, veya undefined ise belirli bir değeri atar.

Boş Birleşme Ataması (??=), bir değişkenin mevcut değeri null veya undefined ise belirli bir değeri atar.

ECMAScript 2021 (ES12)

Sayısal ayırıcılar özelliği, büyük veya karmaşık sayıları daha okunabilir hale getirmek için sayıların içine alt çizgi (_) yerleştirilmesine olanak tanır.

ECMAScript 2022 (ES13)

Array.at(), bir dizide belirli bir konumdaki öğeyi döndürmek için kullanılan yeni bir yöntemdir. Negatif bir indeks verildiğinde, dizinin sonundan başlayarak eleman seçer.

String.at(), bir stringde belirli bir konumdaki karakteri döndürmek için kullanılan bir yöntemdir. Tıpkı Array.at() gibi, negatif bir indeksle stringin sonundan başlayarak karakter seçebilir.

Object.hasOwn(), bir nesnenin belirtilen özelliğe sahip olup olmadığını kontrol etmek için kullanılır.

Object.prototype.hasOwnProperty()'ye modern bir alternatiftir.

ECMAScript 2023 (ES14)

Array.findLast(), bir dizideki elemanları sağdan sola (sondan başa) kontrol eder ve belirtilen koşulu sağlayan ilk elemanı döndürür. Eğer koşulu sağlayan bir eleman bulunamazsa undefined döner.

Array.findLastIndex(), bir diziyi sağdan sola (sondan başa) kontrol ederek belirtilen koşulu sağlayan ilk elemanın indeksini döndürür. Eğer koşulu sağlayan bir eleman bulunamazsa -1 döner

Array.toReversed(), bir diziyi tersine çevirir, ancak orijinal diziyi değiştirmez. Bu yöntem, reverse() metoduna immutability (değişmezlik) eklenmiş bir alternatif olarak.

Array.toSorted(), bir diziyi sıralar ve sıralanmış yeni bir dizi döndürür. Orijinal dizi değişmeden kalır. sort() metodunun immutability özelliğiyle geliştirilmiş halidir.

Array.toSpliced(), bir dizide belirtilen pozisyonda eleman ekleme, silme veya değiştirme işlemi yapar ve bu işlemi yeni bir dizi döndürerek gerçekleştirir. Orijinal dizi değişmeden kalır.

Array.with(), bir dizideki belirli bir indeksin değerini değiştirerek yeni bir dizi döndürür. Orijinal dizi değişmeden kalır.

JavaScript ile Nesne Yönelimli Programlama (OOP)

Nesne Yönelimli Programlama (Object-Oriented Programming-OOP), modern yazılım geliştirme süreçlerinde kullanılan en etkili paradigmalar arasındadır. OOP, yazılımı modüler bir şekilde organize ederek kodun okunabilirliğini artırır, karmaşıklığı azaltır ve sürdürülebilir projeler geliştirilmesine olanak tanır. JavaScript, ES6 ile birlikte OOP prensiplerini sınıflar ve nesneler aracılığıyla güçlü bir şekilde desteklemektedir. Bu yaklaşım, yazılım geliştirme sırasında kodun daha yapılandırılmış ve tekrar kullanılabilir olmasını sağlar.

Sınıflar ve Nesneler

JavaScript'te sınıflar, class anahtar kelimesiyle tanımlanır ve özellikler ile davranışlar bu sınıflar aracılığıyla yapılandırılır. Sınıflar, yazılım geliştiricilerin nesne özelliklerini ve metotlarını bir arada düzenlemesine olanak tanır. ES6 öncesinde JavaScript, yalnızca prototip tabanlı bir yapıya sahipti. Ancak ES6 ile birlikte sınıf tabanlı tanımlar geliştiricilere daha kolay bir OOP deneyimi sunar. Sınıfların önemli bir parçası olan constructor metodu, bir sınıftan nesne oluşturulduğunda otomatik olarak çağrılır. Bu metot, nesneye ait başlangıç değerlerinin atanmasında kullanılır. Static özellikler ve metotlar ise belirli bir nesneye değil, sınıfa özgüdür ve tüm nesneler arasında paylaşılır.

Kalıtım (Inheritance)

Kalıtım, bir sınıfın başka bir sınıftan özellik ve metotları miras almasını sağlar. Bu yapı, alt sınıfların (subclass) üst sınıflardan (base class) faydalanarak kod tekrarını azaltmasını ve daha organize bir yapıya sahip olmasını mümkün kılar. Kalıtım sırasında extends ve super anahtar kelimeleri kullanılır. extends, alt sınıfın üst sınıftan türetildiğini belirtirken, super üst sınıfın metotlarına erişimi sağlar. Kalıtım işlemi sırasında, metotların alt sınıfta yeniden tanımlanması gerekebilir. Bu durumda, üst sınıfta tanımlı metot geçersiz kılınarak (overriding) alt sınıfa özel bir işlevsellik kazandırılır.

Prototip Tabanlı Yapılar

JavaScript'teki her nesne, kendisinden türetildiği bir başka nesneye referans verir. Bu mekanizma prototip zinciri olarak adlandırılır. Örneğin, bir alt sınıf olan Elektronik, üst sınıf olan Ürünün tüm özelliklerini miras alabilir. Bu yapı, bir nesne üzerinde aranan özelliğin önce kendi yapısında, sonra üst sınıflarda ve son olarak temel JavaScript sınıfı Object üzerinde aranmasını sağlar.

Kapsülleme (Encapsulation)

Kapsülleme, bir nesnenin iç verilerini dış etkilerden korur. JavaScript'te kapsülleme, verilerin yalnızca belirli metotlarla erişilebilir olmasını sağlar. ES6 ile gelen # sembolü, private (özel) özelliklerin tanımlanmasında kullanılır. Getter ve setter metotları, bu özel verilere erişimi kontrol eder.

Kapsülleme, yazılımın güvenilirliğini artırır ve kodun okunabilirliğini destekler. Örneğin, bir BankaHesabi sınıfında hesap numarası ve bakiye bilgisi gibi veriler dış erişime kapalı olabilir. Kullanıcı, yalnızca sınıfın sunduğu belirli metotlar aracılığıyla bu verilere erişebilir.

Asenkron Programlama ve JavaScript

Modern web uygulamalarında kullanıcı deneyimi ve performans, kritik bir öneme sahiptir. Asenkron programlama, uzun süren işlemleri bloklamadan yürütmeyi sağlar. Bu, özellikle API istekleri, dosya işlemleri ve zamanlayıcılar gibi işlemler sırasında uygulamanın diğer işlevlerini kesintiye uğratmadan çalışmasını mümkün kılar.

Senkron ve Asenkron İşlemler

Senkron işlemler, bir görev tamamlanmadan diğerine geçiş yapılmayan işlemlerdir. Bu durum, basit yapılar için uygun olsa da uzun süren görevlerde programın durmasına neden olabilir. Asenkron işlemler ise bir işlemin tamamlanmasını beklemeden diğer işlemlerin yürütülmesine olanak tanır. Bu sayede uygulamalar, kullanıcı etkileşimini kesintisiz şekilde sürdürebilir.

Callback Fonksiyonları

Callback fonksiyonları, bir işlemin tamamlanmasının ardından çağrılan fonksiyonlardır. Ancak, iç içe geçen çok sayıda callback fonksiyonu kod karmaşıklığına yol açabilir. Bu durum "Callback Hell" olarak adlandırılır ve asenkron programlamanın zorluklarından biri olarak görülür.

Promise Yapısı

Promise, asenkron işlemleri daha okunabilir ve yönetilebilir hale getiren bir yapıdır. Bir promise, üç farklı durumda olabilir:

Pending: İşlem tamamlanmamış.

Fulfilled: İşlem başarılı bir şekilde tamamlanmış.
Rejected: İşlem başarısız olmuş.
Promise zincirleme yöntemiyle birden fazla işlemi sıralı bir şekilde gerçekleştirmeyi sağlar. Promise, bir işlemin sonucuna göre then, catch ve finally metotları ile yönetilebilir.
Async/Await
Async/Await, ES8 ile JavaScript'e dahil edilen bir yapıdır ve promise yapısının daha sade bir sözdizimiyle kullanılmasını sağlar. Async/Await ile asenkron işlemler, tıpkı senkron işlemler gibi yazılabilir. Await anahtar kelimesi, bir işlemin tamamlanmasını bekler ve bu sayede kodun sıralı ve okunabilir bir şekilde yazılmasını sağlar.
OOP ve Asenkron İşlemlerin Avantajları
Bu bölümde ele alınan kavramlar, yazılım geliştirme sürecinde birçok avantaj sunar. OOP, yazılımın daha yapılandırılmış ve modüler bir şekilde geliştirilmesini sağlarken, asenkron programlama modern web uygulamalarında performansı artırır.
OOP'nin Avantajları:
Modülerlik ve okunabilirlik.
Kodun yeniden kullanılabilir olması.
Karmaşıklığın azaltılması.
Verilerin güvenliği ve yönetimi.
Asenkron Programlamanın Avantajları:
Bloklamayı önleme.
Uzun süren işlemler sırasında kullanıcı deneyimini kesintisiz sürdürme.
Performans optimizasyonu.

DERS ADI İleri Web Programlama
ÜNİTE ADI Formlar ve Validasyon
ÜNİTE NO 7
YAZAR Arş.Gör. SATUK BUĞRAHAN ÖZTÜRK

GİRİŞ

Web formları, kullanıcıların web uygulamalarıyla etkileşim kurmasını sağlayan en temel araçlardan biri olarak modern web teknolojilerinde vazgeçilmez bir yere sahiptir. Bu yapılar, kullanıcıların bilgi girişi yapmasını, veri göndermesini ve etkileşimli deneyimler yaşamasını mümkün kılar. HTML5 ile form elemanlarına yapılan eklemeler ve yerleşik doğrulama özellikleri, geliştiricilere hem işlevsellik hem de kullanıcı deneyimi açısından önemli avantajlar sunmaktadır. Bu yenilikler, yalnızca teknik geliştirmelere odaklanmakla kalmayıp, kullanıcı dostu ve erişilebilir bir tasarım anlayışını da teşvik etmektedir. Bu bağlamda, formların doğru ve etkili bir şekilde tasarlanması, bir web uygulamasının başarısında kritik bir rol oynamaktadır.

FORMLARIN TANIMI VE ÖNEMİ

Web formları, kullanıcıların sistemle etkileşim kurmasına olanak tanıyan temel araçlardır. Formlar, veri toplama, işleme ve sunma gibi süreçlerin merkezinde yer alır ve kullanıcı deneyimini şekillendiren bir yapı taşıdır. HTML5 ile gelen yenilikler, form tasarımı ve işlevselliğinde önemli iyileştirmeler sağlamış, böylece kullanıcıların hata yapma olasılığını azaltarak daha doğru veri girişini mümkün kılmıştır. Örneğin, e-posta, tarih ve sayı gibi yeni giriş türleri, formların yalnızca veri girişini kolaylaştırmasını değil, aynı zamanda veri doğruluğunu artırmasını sağlamıştır.

Formların kullanıcı etkileşimindeki yeri, yalnızca teknik işlevsellikleriyle sınırlı değildir. İyi tasarlanmış bir form, kullanıcıların bilgi girişini hızlandırırken memnuniyeti de artırır. Bunun yanı sıra, erişilebilirlik standartlarına uygun formlar, farklı ihtiyaçlara sahip kullanıcı gruplarına hitap ederek web uygulamalarının daha kapsayıcı olmasını sağlar. Özellikle erişilebilirlik açısından, etiketleme ve doğru düzenleme teknikleri büyük önem taşır. Bu yönüyle formlar, hem geliştirici hem de kullanıcı perspektifinden değerlendirilmesi gereken kritik bir unsurdur.

HTML5 FORM ELEMANLARI

HTML5, formlara yeni özellikler ve elemanlar ekleyerek web geliştirme süreçlerini önemli ölçüde geliştirmiştir. Form yapısının temel bileşenleri,
etiketi ile başlar ve bu yapı, kullanıcıdan bilgi toplamak ve sunucuya göndermek için çeşitli giriş elemanlarını içerir. HTML5 ile sunulan farklı türdeki giriş elemanları, formun işlevselliğini artırırken kullanıcı deneyimini de iyileştirir. Metin giriş türleri arasında, serbest metin girişi sağlayan text, şifrelerin gizlenmesini mümkün kılan password ve e-posta doğrulama işlemini destekleyen email yer almaktadır. Bunun yanı sıra, kullanıcıların tarih ve zaman bilgisi girmesini kolaylaştıran date ve time gibi elemanlar da bulunmaktadır.

Seçim elemanları, kullanıcılara birden fazla seçenek sunarak daha karmaşık form yapılarına olanak tanır. Örneğin, checkbox elemanları kullanıcıların birden fazla seçeneği işaretlemesine olanak tanırken, radio button yalnızca bir seçeneğin seçilmesini sağlar. Açılır listeler (dropdown) ise önceden tanımlanmış seçenekler arasından bir seçim yapılmasına imkan tanır. Dosya yükleme işlemleri için kullanılan file elemanı, kullanıcıların dosya yüklemelerini kolaylaştırırken, geliştiricilere bu dosyaların türlerini sınırlama yetkisi sunar. HTML5 ayrıca, form gönderim işlemleri için submit, sıfırlama işlemleri için reset ve özel işlevler için button gibi düğme türleri de sağlamaktadır.

FORM DÜZENİ VE TASARIMI

Formların düzeni ve tasarımı, kullanıcıların formları hızlı ve doğru bir şekilde doldurabilmesi için büyük önem taşır. Etkili bir form tasarımı, kullanıcı deneyimini iyileştirirken aynı zamanda veri doğruluğunu artırır. Label etiketleri, kullanıcıların her bir form elemanının ne tür bir bilgi beklediğini kolayca anlamasını sağlar. Placeholder metinleri ise, kullanıcıya giriş alanının nasıl doldurulacağı konusunda rehberlik eder. Ancak, placeholder metinlerinin yalnızca geçici bir rehberlik sunduğu ve label etiketlerinin yerini almaması gerektiği unutulmamalıdır.

HTML5, karmaşık formları düzenlemek için fieldset ve legend etiketlerini sunar. Bu etiketler, formların görsel ve mantıksal olarak gruplandırılmasını sağlar. Özellikle uzun ve karmaşık formlarda, bu yapıların kullanımı, kullanıcıların doğru bilgilere odaklanmasını kolaylaştırır. Form tasarımında erişilebilirlik, net talimatlar ve kullanıcı dostu bir düzen, kullanıcı deneyimini iyileştirmenin temel yollarıdır.

HTML5 DOĞRULAMA (VALIDASYON) TEKNİKLERİ

HTML5, form doğrulama süreçlerinde geliştiricilere yerleşik mekanizmalar sunarak harici doğrulama yöntemlerine olan bağımlılığı azaltmıştır. Required, pattern, minlength ve maxlength gibi doğrulama

öznitelikleri, form girişlerinin belirli standartlara uygun olmasını sağlar. Bu doğrulama özellikleri, formun sunucuya gönderilmeden önce kullanıcı girişlerini kontrol ederek hata yapma olasılığını en aza indirir. Gömülü doğrulama özellikleri sayesinde, tarayıcılar kullanıcıları anında uyararak daha iyi bir kullanıcı deneyimi sunar. Bunun yanı sıra, hata mesajlarının JavaScript kullanılarak özelleştirilmesi, kullanıcıların girdileri düzeltme sürecinde daha etkili geri bildirim almasını sağlar.

JAVASCRIPT İLE FORM DOĞRULAMA

HTML5 doğrulama teknikleri çoğu durumda yeterli olmakla birlikte, daha karmaşık iş süreçleri için JavaScript ile özelleştirilmiş doğrulama yöntemleri gereklidir. JavaScript, kullanıcı girdilerinin daha dinamik ve esnek bir şekilde kontrol edilmesine olanak tanır. Örneğin, event listener kullanarak form gönderim işlemleri sırasında belirli kuralların ihlal edilip edilmediği kontrol edilebilir. Ayrıca, JavaScript ile hata mesajlarının dinamik olarak gösterilmesi, kullanıcıların yaptıkları hataları daha hızlı anlamalarını sağlar ve böylece form doldurma sürecini iyileştirir.

FORM VERİLERİNİN GÖNDERİMİ VE GÜVENLİĞİ

Web formlarının temel amacı, kullanıcıdan toplanan verileri sunucuya iletmektir. HTML5 formları, veri gönderimi için GET ve POST yöntemlerini destekler. GET yöntemi, verileri URL parametreleri aracılığıyla gönderirken, POST yöntemi verileri HTTP isteğinin gövdesine ekler ve daha gizli bir aktarım sağlar. Ancak, form verilerinin güvenliği yalnızca veri aktarımıyla sınırlı değildir. CSRF ve XSS gibi güvenlik tehditlerine karşı önlem alınması gereklidir. CSRF token'ları, referer kontrolleri ve CAPTCHA gibi araçlar, bu tehditlere karşı etkili çözümler sunar. CAPTCHA ve reCAPTCHA gibi doğrulama yöntemleri, özellikle bot saldırılarına karşı koruma sağlarken, kullanıcı dostu bir deneyim sunmayı hedefler.

SONUÇ

HTML5 ile gelen form elemanları, doğrulama mekanizmaları ve güvenlik önlemleri, modern web uygulamalarının işlevselliğini ve kullanıcı deneyimini geliştiren temel bileşenlerdir. Doğru tasarlanan ve uygulanan formlar, yalnızca veri toplama aracı olmanın ötesinde, kullanıcılarla web uygulamaları arasında güçlü bir etkileşim köprüsü kurar. Kullanıcı dostu, erişilebilir ve güvenli bir form tasarımı, bir web uygulamasının başarısında belirleyici bir unsur olarak ön plana çıkar.

Web API'lerin Tanımı ve Önemi

API (Application Programming Interface), yazılımlar arasında iletişim kurmak için tasarlanmış bir arayüzdür. API'ler, özellikle farklı sistemlerin birbiriyle entegre çalışmasını kolaylaştırarak modern yazılım geliştirme süreçlerinde kritik bir rol oynar. API'ler, geliştiricilere yeniden kullanılabilir kodlar yazma ve sistemler arasındaki entegrasyonu sağlama olanağı sunar.

Web API'ler, özellikle istemci (tarayıcı veya mobil uygulama) ile sunucu arasında veri alışverişini sağlamak üzere tasarlanmıştır. Bu özellik, sosyal medya platformlarından e-ticaret sitelerine kadar geniş bir uygulama alanına sahiptir. Web API'ler, veri alışverişinde HTTP protokolü üzerinden çalışır ve çoğu zaman JSON formatında veri iletir.

RESTful API (Representational State Transfer), HTTP protokolü üzerinden veri alışverişi yapan bir mimari stildir. Basit, esnek ve ölçeklenebilir yapısı ile RESTful API'ler, modern uygulama geliştirme süreçlerinin temel bir bileseni haline gelmiştir. REST ilkeleri, istemci ve sunucu arasındaki iletişimde yalın ve etkili bir metodoloji sağlar.

RESTful API ve CRUD İşlemleri

CRUD (Create, Read, Update, Delete), veri manipülasyonunun temelini oluşturan dört önemli fonksiyonun kısaltmasıdır. RESTful API'ler, CRUD işlemlerini HTTP metotları aracılığıyla gerçekleştirir. Bu metotlar şu şekilde kullanılır:

GET: Veri almak için kullanılır.

POST: Yeni veri eklemek için kullanılır.

PUT/PATCH: Mevcut veriyi güncellemek için kullanılır.

DELETE: Veri silmek için kullanılır.

CRUD operasyonları, RESTful API'ler üzerinde veri ekleme, okuma, düzenleme ve silme gibi işlemleri kolayca gerçekleştirmenizi sağlar. RESTful API'lerin stateless (durumsuz) yapısı, her isteğin bağımsız olmasını gerektirir. Bu, sunucunun istemci hakkında herhangi bir oturum bilgisi saklamaması anlamına gelir ve sistemlerin daha kolay ölçeklenebilmesini sağlar.

Modern Veri İşleme Araçları

Fetch API, modern tarayıcılarda yerleşik olarak bulunan bir API'dir ve HTTP isteklerini yönetmek için kullanılır. Promise tabanlı yapısı sayesinde asenkron işlemleri yönetmek oldukça kolaylaşır. Fetch API, basit GET isteklerinden karmaşık POST işlemlerine kadar geniş bir yelpazede çözümler sunar. Ancak, hata yönetimi ve kod yapısı açısından sınırlıdır.

Axios, Fetch API'ye alternatif olarak geliştirilmiş popüler bir JavaScript kütüphanesidir. Daha kullanıcı dostu bir deneyim sunar ve ek özelliklerle geliştiricilere esneklik tanır. Axios, HTTP isteklerini daha kolay bir şekilde yönetmeyi sağlarken, interceptor mekanizması ve zaman aşımı kontrolü gibi özelliklerle öne çıkar.

Interceptor Mekanizması

Interceptor'ler, HTTP istekleri ve yanıtları üzerinde kontrol sağlamak için kullanılır. Bu mekanizma, yetkilendirme bilgileri ekleme, hata yönetimi veya özel veri işlemleri gibi durumlarda oldukça faydalıdır. Interceptor kullanımı, API entegrasyonlarında geliştiriciye daha fazla kontrol sağlar.

JSON ile Veri İşleme

JSON (JavaScript Object Notation), modern web geliştirme süreçlerinde veri alışverişinin standart formatıdır. Hafif, okunabilir ve dil bağımsız bir yapıya sahiptir. API'ler üzerinden alınan veriler genellikle JSON formatındadır ve bu verilerin manipüle edilmesi için şu teknikler kullanılabilir:

JSON.parse(): JSON string'ini JavaScript nesnesine dönüştürür.

JSON.stringify(): JavaScript nesnesini JSON string'ine dönüştürür.

Filter ve Map Metotları: JSON verilerini filtrelemek ve dönüştürmek için kullanılır. Array sınıfının altında tanımlı metotlardır.

Filter, JSON verileri arasından belirli kriterlere uyan verilerin seçilmesini sağlar. Map ise verilerde dönüşüm yaparak yeni veri yapıları oluşturur.

Sonuç

Bu ünite, RESTful API, Fetch API, Axios ve JSON gibi temel kavramlara odaklanarak modern web geliştirme dünyasında sıklıkla karşılaşılan teknolojilere yönelik kapsamı bir bakış sunar. API entegrasyonlarını anlamak ve uygulamalarda etkin kullanımını sağlamak, geliştiricilere çok yönlü bir beceri kazandırır. JSON'un sade ve hafif yapısı, veri alışverişini hızlı ve etkili hale getirirken Fetch

API ve Axios gibi araçlar, bu işlemleri daha kullanışlı hale getiren çözümler sunar. Bu özellikler, geliştiricilerin çalışma verimliliğini artırır.

REACT NEDİR?

React, bileşen tabanlı bir yaklaşımla geliştirilmiştir. Bu sayede uygulamaların görsel ve işlevsel parçaları daha kolay yönetilebilir. React'in öne çıkan özelliklerinden biri olan sanal DOM (Virtual DOM), geleneksel DOM manipülasyonlarına kıyasla performans avantajı sağlar. React'in sunduğu zengin eklenti ekosistemi ve topluluk desteği, geliştiricilere esneklik ve kapsamlı yardım sağlar. React'in sözdizimi ve yapısal tasarımı, geliştiricilerin arayüzü kolayca güncellemesine olanak tanır. Geniş kütüphane entegrasyonları ve modern JavaScript desteği ile React, mobil uygulama geliştirme dâhil çeşitli platformlarda çözümler sunar.

React ile Front-End Geliştirme

React, modern web geliştirme dünyasında önemli bir yer tutan ve büyük bir geliştirici topluluğu tarafından desteklenen bir JavaScript kütüphanesidir. 2013 yılında Facebook tarafından geliştirilen React, bileşen tabanlı yapısıyla öne çıkar ve kültürden büyük kurumsal projelere kadar geniş bir alanda kullanılır. Kullanıcı arayüzlerini daha modüler ve okunabilir hale getiren React, modern web uygulamaları geliştirme süreçlerini dönüştürmeyi başarır.

React'in Avantajları

React, aşağıdaki avantajları ile öne çıkmaktadır:

Bileşen Tabanlı Yapı: Uygulamaların daha modüler hale gelmesini ve bileşenlerin yeniden kullanılabilirliğini sağlar. Bu, projelerde geliştirme ve bakım süreçlerini kolaylaştırır.

Performans: Sanal DOM yapısı ile daha hızlı güncellemeler ve daha iyi bir kullanıcı deneyimi sağlar.

Genişletilebilirlik: Redux gibi durum yönetim aracı desteğiyle karmaşık uygulamaların yönetimini kolaylaştırır.

Dokümantasyon ve Topluluk Desteği: React, kapsamılı bir dokümantasyon ve aktif bir geliştirici topluluğu sunar.

React Native: Mobil uygulama geliştirme desteği sunarak React'i web ve mobil platformlar için çok yönlü bir aracı haline getirir.

React Kurulumu

React ile proje geliştirmek için uygun bir ortam kurmak önemlidir. Node.js ve npm aracı gereklidir. Kurulum aşamaları:

Node.js Kurulumu: Node.js'in en güncel sürümü resmi web sitesinden indirilebilir.

Create React App: Bu aracı kullanarak temel bir proje yapısı oluşturabilirsiniz.

Visual Studio Code: React projeleri için yaygın bir geliştirme aracıdır.

Yukarıdaki bahsedilen ortamların kurulumları tamamlandıktan sonra geliştirme ortamı hazır hale gelmiş olur ve geliştirmelere başlanabilir.

React'in Yapısı ve Temel Kavramları

React, modern bir JavaScript kütüphanesi olarak bileşen tabanlı mimariyi benimsemiştir. Bu yaklaşım, kullanıcı arayüzünün bağımsız parçalara bölünmesini sağlar.

JSX: React geliştirme sürecinde kullanılan özel bir sözdizimidir. HTML benzeri bir yapı sunarak bileşenlerin tanımlanmasını kolaylaştırır. JSX, JavaScript ifadeleriyle zenginleştirilmiş dinamik arayüzler oluşturmayı mümkün kılar.

Props (Properties): Bileşenler arası veri aktarımını sağlayan temel yapı taşıdır. Bir bileşenin dışarıdan alabileceği verileri ifade eder. Props, salt okunur niteliktedir ve bileşene dinamik özellikler kazandırır.

State: Bir bileşenin kendi içsel durumunu tanımlar. Kullanıcı etkileşimleri gibi durum değişiklikleri sonucunda bileşenin yeniden render edilmesini sağlar. State, bileşenin özelleştirilmesinde ve dinamikleştirilmesinde önemli bir rol oynar.

REACT BİLEŞENLERİ

Bileşenler, projeyi düzenli hale getirir ve yeniden kullanılabilir parçalar oluşturur. Props ve state ile bu bileşenlerin işlevselliği artırılabilir. React, iki ana bileşen yapısına sahiptir.

Fonksiyonel Bileşenler: Basit fonksiyonlar olarak tanımlanır ve genelde küçük arayüzler için kullanılır. Girdi alıp bir çıktı döndüren saf fonksiyonlar olarak işlev görürler. React Hook'ları sayesinde daha fazla işlevsellik kazanabilirler.

Sınıf Bileşenleri: Daha karmaşık işlevler için tercih edilir ve genellikle büyük projelerde kullanılır. Yaşam döngüsü metotları (componentDidMount, componentDidUpdate vb.) sayesinde detaylı kontrol sağlar. State ve props ile çalışırlar.

JSX: JavaScript ve HTML'in Buluşması

JSX, React uygulamalarında HTML ile JavaScript'i birleştirerek daha modüler ve dinamik kullanıcı arayüzleri oluşturmayı sağlar. HTML benzeri bir yapıya sahiptir ama aslında bir JavaScript ifadesidir. JSX, JavaScript değişkenleri ve fonksiyonlarıyla kolayca entegre olabilir. Tarayıcılar doğrudan JSX'i çalıştırmaz; bunun yerine Babel gibi bir derleyici JSX'i JavaScript'e dönüştürür. JSX, React bileşenleriyle uyumlu olup her bileşenin HTML yapısını ve JavaScript mantığını bir arada barındırmasını sağlar, böylece kod tekrar kullanılabilir ve daha okunabilir hale gelir.

Props Kavramı

React'te props, bileşenler arasında veri aktarımını sağlayan bir kavramdır. Props, bileşenlere dışarıdan aktarılan ve görünüm ya da davranışı özelleştiren verilerdir. Props, salt okunur olup bileşen içinde değiştirilemez. Fonksiyonel bileşenlerde props, parametre olarak kullanılırken, sınıf bileşenlerinde this.props ile erişilir. Props, bileşenlerin dinamik ve yeniden kullanılabilir olmasını sağlar. Varsayılan değerler ve tür kontrolü (prop-types) ile props yönetimi daha sağlam hale gelir. React'te veri akışı tek yönlüdür; üst bileşenden alt bileşene veri aktarılır, ancak alt bileşenler veriyi değiştiremez.

REACT'TE STATE YÖNETİMİ VE HOOK YAPISI

State Nedir ve Neden Kullanılır?

State, bir bileşenin içinde tanımlanarak zaman içinde değişebilen verileri saklayan bir nesnedir. Props'tan farklı olarak sadece bileşenin içinde kontrol edilir. State, bileşenin davranışını ve görünümünü etkileyen verileri yönetir. React, state değiştiğinde bileşeni yeniden render eder.

Sınıf Bileşenlerinde State Yönetimi

React'in erken sürümlerinde state yalnızca sınıf bileşenlerinde kullanılabiliyordu. Bu durumda state, constructor içinde tanımlanıp setState ile güncelleniyordu. Fonksiyonel Bileşenlerde State ve Hook Kullanımı

React 16.8 ile tanıtılan Hook'lar, fonksiyonel bileşenlerde state kullanımını sağlar. useState Hook'u, state ve onu güncelleyen fonksiyonları döndürerek fonksiyonel bileşenlerde daha basit ve okunabilir kod yazılmasını sağlar.

State ve Props Arasındaki Fark

Props dışarıdan aktarılır ve değiştirilemezken, state bileşenin içsel durumu olup değiştirilebilir. Props, bileşenler arası veri aktarımını sağlarken, state bileşenin iç işleyişini kontrol eder.

Hook'ların Avantajları

React Hook'ları, fonksiyonel bileşenlerde state ve yaşam döngüsü yönetimini sağlar, kodu sadeleştirir, yeniden kullanılabilirliği artırır ve geliştirici deneyimini iyileştirir. Hook'lar, modüler ve esnek yapı sunarak bileşenler arasında mantık paylaşımını kolaylaştırır.

REACT İLE STİLL YÖNETİMİ

Satır İçi Stil Ekleme (Inline Styling)

React, HTML'den farklı olarak, satır içi stil tanımlamaları için JavaScript nesnelerini kullanır. Bu yöntem, dinamik stiller uygulamayı kolaylaştırır. Küçük bileşenler için ideal olsa da büyük projelerde karmaşıklık yaratabilir.

CSS Modülleri

CSS Modülleri, her bileşenin kendi stil dosyasına sahip olmasını sağlayarak stil çakışmalarını engeller ve bileşen bazlı geliştirmeyi destekler. Bu yöntem, stil yönetimini daha kapsüllü hale getirir.

CSS Dosyaları

React, geleneksel CSS dosyalarının kullanımını destekler. Stiller ayrı bir .css dosyasında tanımlanır ve bileşene className ile uygulanır, bu yöntem büyük projelerde yaygın olarak tercih edilir.

CSS-in-JS Yaklaşımları

React, popüler stil kütüphaneleri (Bootstrap, Material-UI, Tailwind CSS) ile entegre çalışabilir. Bu kütüphaneler, esneklik ve hız sağlar, özellikle profesyonel kullanıcı arayüzleri oluşturmak isteyen projelerde kullanılır. React, esnek stil yönetimi çözümleri sunarak geliştirme sürecini kolaylaştırır.

REACT İLE OLAY YÖNETİMİ

React Olay Yönetimi

React, kullanıcı etkileşimlerini işlemek için olay yönetim sistemi sunar. Bu sistem, bileşenlerin kullanıcı girişlerine yanıt vermesini sağlar. React, olayları "SyntheticEvent" adı verilen bir sarmalayıcıyla işler ve tarayıcılar arası uyumluluğu sağlar. Ayrıca olay isimleri camelCase (örneğin, onClick, onChange) kullanılarak tanımlanır.

Fonksiyonel Bileşenlerde Olay Yönetimi

Fonksiyonel bileşenlerde olay yönetimi basittir. useState Hook'u ile olaylara bağlı değişiklikler yapılabilir. Örneğin, bir butona tıklanarak sayaç artırılabilir.

Sınıf Bileşenlerinde Olay Yönetimi

Sınıf bileşenlerinde olay yönetimi, this bağlamının doğru şekilde yönetilmesini gerektirir. Olay işleyicileri, bileşenin state'ini güncelleyerek etkileşim sağlar.

React, kullanıcı etkileşimlerini yönetmede esnek ve etkili bir çözüm sunar. Fonksiyonel bileşenlerde basit, sınıf bileşenlerinde ise daha detaylı kontrol imkanı sağlar.

REACT ROUTER İLE SAYFA GEÇİŞLERİ

React Router, tek sayfa uygulamalarında (SPA) kullanıcıların farklı URL'lere erişmesini ve sayfa geçişleri yapmasını sağlayan bir kütüphanedir. Geleneksel uygulamalarda her URL için tam sayfa yüklemesi yapılırken, React Router ile sadece gerekli bileşenler yeniden render edilir, bu da kullanıcı deneyimini iyileştirir ve performansı artırır.

React Router'ın Temel Yapısı React Router üç ana bileşen sunar:

BrowserRouter: Yönlendirme işlemlerini üstlenir.

Routes: Yönlendirme kurallarını tanımlar.

Route: Belirli bir yol ile ilgili bileşen eşlemesi yapar. React Router, sayfa yenilemesi olmadan hızlı bileşen güncellemeleri, dinamik parametre kullanımı, yönlendirme koruması ve geçiş efektleri gibi avantajlar sunar.

Modern web uygulamalarında, küçük ve büyük projelerde etkili yönlendirme çözümleri sağlar, kullanıcı deneyimini ve performansı geliştirir.

ANGULAR'A GİRİŞ

Angular, modern web geliştirme süreçlerinde kullanılan, Google tarafından geliştirilen ve açık kaynaklı bir ön yüz çerçevesidir. TypeScript tabanlı yapısıyla güçlü bir tip sistemi sunan Angular, özellikle tek sayfalık uygulamalar (Single Page Applications - SPA) geliştirmek için tasarlanmıştır. Dinamik kullanıcı arayüzleri oluşturmayı kolaylaştıran bu çerçeve, geliştiricilere modüler, yeniden kullanılabilir ve ölçeklenebilir bir yazılım mimarisi sağlar.

Angular'ın en temel özelliklerinden biri, bileşen tabanlı mimarisidir. Bu mimari, uygulamaları küçük ve bağımsız birimlere bölerek hem kodun daha anlaşılır olmasını sağlar hem de farklı bileşenlerin yeniden kullanılabilmesine olanak tanır.

ANGULAR KURULUMU VE PROJE YAPISI

Angular, modern web uygulamaları geliştirmek için güçlü araçlar sunan bir çerçevedir. Ancak bu çerçeveyi kullanmaya başlamadan önce, doğru bir kurulum sürecini takip etmek ve proje yapısını anlamak gereklidir.

Kurulum

Angular kurulum süreci, genellikle birkaç temel adımdan oluşur. İlk olarak, Angular CLI (Command Line Interface) aracı yüklenmelidir. Angular CLI'yi yüklemek için, Node.js ve npm (Node Package Manager) yüklü olmalıdır.

Proje Yapısı

Angular projeleri, modüler ve düzenli bir yapı sunmak için belirli bir dizin hiyerarşisi ile oluşturulur. Projenin kök dizininde klasör ve dosyalar bulunur:

src: Projenin ana dizinidir. Uygulamanın tüm bileşenleri, modülleri ve servisleri bu klasörde bulunur.
app: Uygulamanın ana modülü ve bileşenlerini içerir. Yeni bileşenler ve servisler burada organize edilir.
assets: Uygulamada kullanılan statik dosyalar (örneğin, görseller ve CSS dosyaları) burada saklanır.
angular.json: Projenin genel yapılandırma ayarlarını tanımlayan dosyadır.
package.json: Projede kullanılan bağımlılıkların listelendiği ve yönetildiği dosyadır.
node_modules/: npm tarafından indirilen ve projede kullanılan tüm üçüncü taraf kütüphanelerini içerir.

ANGULAR'IN TEMEL KAVRAMLARI

Angular, modern web uygulamaları geliştirmek için güçlü bir çerçeve sunar. Bu çerçeve, verimli bir geliştirme süreci sağlamak amacıyla bir dizi temel kavram ve mimari yapı üzerine inşa edilmiştir.

Bileşenler (Components)

Angular uygulamalarının temel yapı taşları bileşenlerdir. Her bileşen, bir kullanıcı arayüzü öğesini temsil eder ve uygulamanın belirli bir işlevselliğini yerine getirir. Bir bileşen üç ana öğeden oluşur: bir TypeScript sınıfı (iş mantığını tanımlar), bir HTML şablonu (görünümü oluşturur) ve bir CSS dosyası (stilleri tanımlar

Modüller

Angular, uygulamaları mantıksal bölümlere ayırmak için modül konseptini kullanır. Her Angular uygulaması, kök modül olan AppModule ile başlar. Bunun yanı sıra, özelliklere özgü modüller oluşturularak uygulama daha düzenli ve yönetilebilir hale getirilebilir.

Servisler ve Bağımlılık Enjeksiyonu

Servisler, uygulama genelinde paylaşılan iş mantığını barındıran yapılardır. Angular, servislerin kullanımı için güçlü bir bağımlılık enjeksiyonu mekanizması sunar. Bu mekanizma, bileşenlerin veya diğer servislerin ihtiyaç duyduğu bağımlılıkların otomatik olarak sağlanmasını mümkün kılar.

Veri Bağlama

Angular, kullanıcı arayüzü ile veri modeli arasında etkileşim sağlamak için veri bağlama yöntemlerini kullanır. İki tür veri bağlama öne çıkar:

Tek yönlü veri bağlama: Modelden görünüme ya da görünümlerden modele veri aktarımını sağlar.
Çift yönlü veri bağlama: Hem modelden görünüme hem de görünümlerden modele eşzamanlı veri aktarımı gerçekleştirir.

Yönlendirme

Angular, tek sayfalık uygulamalarda farklı görünümler arasında geçiş yapmak için güçlü bir yönlendirme sistemi sunar. RouterModule, kullanıcıların belirli URL'lere erişerek farklı bileşenlere yönlendirilmesini sağlar.

Şablonlar ve Direktifler

Angular'ın HTML şablon sistemi, dinamik ve etkileşimli kullanıcı arayüzleri oluşturmayı kolaylaştırır. Direktifler, HTML elemanlarının davranışını değiştirmek veya genişletmek için kullanılır. Örneğin, ngIf ve ngFor gibi yapısal direktifler, DOM manipülasyonunu etkili bir şekilde yönetir.

ANGULAR'DA BİLEŞENLER

Bileşenler (components), Angular frameworkünün temel yapı taşlarından biridir ve uygulamanın kullanıcı arayüzünü (UI) oluşturmak için kullanılır. Her bileşen, bir HTML şablonunu (template), CSS stil dosyasını ve iş mantığını içeren bir TypeScript sınıfını bir araya getirir

Bileşenlerin Tanımı ve Yapısı

Angular'da bir bileşen, @Component deyimi ile tanımlanır. Bu deyim, bileşenin meta verilerini belirler ve Angular'a bileşenin nasıl çalışacağını bildirir. Bir bileşen genel olarak şu unsurlardan oluşur:

Şablon (Template): Bileşenin HTML yapısını tanımlar. Bu yapı, kullanıcının gördüğü kullanıcı arayüzünü oluşturur.

Stil (Style): Bileşenin görsel düzenini belirler. CSS, SCSS veya diğer stil dilleri kullanılabilir.

Mantık (Logic): TypeScript ile yazılmış sınıf, bileşenin işlevselliğini ve dinamik davranışlarını yönetir.

Bileşenlerin Özellikleri ve İşlevleri

Modülerlik

Her bileşen, uygulamanın bağımsız bir parçasını temsil eder. Bu, kullanıcı arayüzünün bölümlere ayrılmasını ve yeniden kullanılabilir hale gelmesini sağlar.

Hiyerarşik Yapı

Angular uygulamaları, bir ana bileşen (root component) ve onun alt bileşenlerinden oluşan bir hiyerarşik yapı ile geliştirilir.

Veri Bağlama (Data Binding)

Bileşenler, şablon ile mantık arasındaki veri alışverişini sağlamak için veri bağlama mekanizmalarını kullanır. Bu mekanizmalar şunlardır:

Interpolation: Model verilerini şablona aktarmak için kullanılır.

Property Binding: HTML element özelliklerini dinamik olarak ayarlamak için kullanılır.

Event Binding: Kullanıcı etkileşimlerini yakalamak ve işlemek için kullanılır.

Yönergeler (Directives)

Bileşen şablonlarında, Angular'ın sunduğu yapısal ve öznitelik yönergeleri kullanılabilir. Örneğin: ngFor, listeleme işlemleri için kullanılır. ve ngIf, koşullu içerik göstermek için kullanılır.

Yaşam Döngüsü (Lifecycle)

Bileşenler, Angular'ın sağladığı yaşam döngüsü yöntemleriyle (lifecycle hooks) belirli olaylar sırasında özel işlevler çalıştırabilir..

SERVİSLER

Angular, modüler bir yapıya sahip olan ve güçlü bir bağımlılık enjeksiyonu (Dependency Injection) sistemi sunan bir frameworkdür. Bu yapıda servisler (services), iş mantığının ve uygulama genelinde kullanılan ortak verilerin yönetimini kolaylaştıran önemli bir bileşendir.

Servislerin Tanımı ve Temel Kullanımı

Servisler, genellikle uygulama genelinde kullanılabilecek olan iş mantığını kapsayan ve Angular'ın bağımlılık enjeksiyonu mekanizması ile bileşenlere sunulan TypeScript sınıflarıdır. Servisler, bir bileşen ya da modül tarafından ihtiyaç duyulan verileri ve işlevleri merkezi bir şekilde yönetir.

Servis Tanımlama

Bir Angular servis sınıfı, @Injectable deyimi ile işaretlenir. Bu deyim, servisin bağımlılık enjeksiyonu sistemi tarafından kullanılabilir olmasını sağlar.

Servislerin Bileşenlerde Kullanımı

Bir servis, bileşen içerisine bağımlılık enjeksiyonu yoluyla eklenir. Bu işlem, bileşenin yapıcı metodunda (constructor) gerçekleştirilir. Servis üzerinden alınan veriler şablonda görüntülenirken, yeni veriler servis aracılığıyla eklenmiştir.

Servislerin Sağlanabilirliği (Provider Configuration)

Angular servisleri, üç farklı bağlamda sağlanabilir:

Kök Sağlayıcı (Root Provider): providedIn: 'root' seçeneği kullanılarak servis, uygulama genelinde tek bir örnekle sağlanır. Bu yöntem, Angular'ın optimize edilmiş bağımlılık enjeksiyonu stratejisini kullanır ve uygulama boyunca servislerin paylaşılmasını sağlar.

Modül Sağlayıcı (Module Provider): Servisler, bir modül içerisinde providers dizisi aracılığıyla sağlanabilir. Bu yöntem, yalnızca belirli bir modüle özel servisler için uygundur.

Bileşen Sağlayıcı (Component Provider): Servisler, bir bileşene özgü olarak da sağlanabilir. Bu durumda, servis yalnızca ilgili bileşen ve onun alt bileşenleri tarafından kullanılabilir.

VERİ YÖNETİMİ

Angular, modern web uygulamalarında veri yönetimini kolaylaştırmak ve kullanıcı arayüzü ile veri modeli arasındaki etkileşimi güçlendirmek için etkili araçlar sunar

Veri Bağlama (Data Binding)

Veri bağlama, kullanıcı arayüzü ile veri modeli arasında bir bağlantı kurar. Angularda tek yönlü veri bağlama ve çift yönlü veri bağlama olmak üzere iki farklı veri bağlama yöntemi bulunmaktadır.

Tek Yönlü Veri Bağlama (One-Way Data Binding)

Tek yönlü veri bağlama (one-way data binding), Angular uygulamalarında modelden görünüme ya da görünümden modele veri aktarımını sağlayan bir mekanizmadır. Tek yönlü veri bağlama, iki ana başlık altında incelenebilir:

Modelden Görünüme Veri Bağlama (From Model to View): Modelden görünüme veri bağlama, bileşen sınıfındaki (TypeScript dosyası) değişken veya özelliklerin HTML şablonunda kullanıcıya aktarılmasını ifade eder.

Görünümden Modele Veri Bağlama (From View to Model): Görünümden modele veri bağlama, kullanıcının yaptığı bir etkileşim sonucunda verilerin bileşen sınıfına aktarılmasını sağlar. Bu bağlama türü, Angular'da event binding ile gerçekleştirilir. Event binding, HTML elementlerinde meydana gelen olayların (örneğin, "click", "input", "change") bileşen sınıfında tanımlı bir işlevi tetiklemesine olanak tanır.

Çift Yönlü Veri Bağlama (Two-Way Data Binding)

Çift yönlü veri bağlama, model ve görünüm arasında eşzamanlı bir veri akışını ifade eder. Angular'da çift yönlü veri bağlama için ngModel direktifi kullanılır.

Bileşen Tabanlı Veri Yönetimi

Veriler, bileşenlerin içerisinde saklanır ve yönetilir. Ancak, bu yaklaşım genellikle yalnızca küçük ve bağımsız veri kümeleri için uygundur.

Servis Tabanlı Veri Yönetimi

Angular servisleri, verilerin merkezi bir şekilde yönetilmesine olanak tanır. Servisler, bileşenler arasında veri paylaşımı ve yeniden kullanılabilir iş mantıkları için idealdir

Veri Bağlama ve Performans

Angular'ın değişiklik algılama mekanizması (Change Detection), veri bağlama sürecinde performansı artıran bir diğer önemli özelliktir. Herhangi bir veri değişikliği algılandığında, Angular DOM'u yeniden render ederek güncel bir kullanıcı arayüzü sağlar.

ANGULAR'DA YÖNLENDİRME

Angular, modern web uygulamaları için güçlü bir yönlendirme (routing) ve navigasyon sistemi sunar. Angular'ın yönlendirme sistemi, URL tabanlı bir mimariye dayanır ve kullanıcıların belirli URL'lere eriştiğinde farklı bileşenlerin yüklenmesine olanak tanır.

Angular Router'ın Temel Mimarisi

Angular Router, temel olarak aşağıdaki bileşenlerden oluşur:

RouterModule: Angular yönlendirme sisteminin ana modülüdür. Yönlendirme yapılandırmasını tanımlamak için RouterModule.forRoot() veya RouterModule.forChild() yöntemleri kullanılır.

Routes Dizisi: Uygulamanın yönlendirme yapılandırmasını tanımlayan bir JavaScript dizisidir. Bu dizi, her bir URL yolunu (route) bir bileşene eşler.

RouterOutlet: Yönlendirme ile ilişkili bileşenlerin dinamik olarak yüklendiği bir yer tutucudur.

RouterLink ve RouterLinkActive: Kullanıcıların belirli URL'lere yönlendirilmesini sağlayan yönergeler (directives) ve aktif yönlendirme durumunu belirleyen mekanizmalardır.

Yönlendirme Yapılandırması

Bir Angular uygulamasında yönlendirme, Routes dizisi kullanılarak yapılandırılır. Her bir yönlendirme, aşağıdaki özellikleri içerebilir:

path: URL yolunu tanımlar.

component: Belirtilen yolda yüklenecek bileşeni belirtir.

redirectTo: Kullanıcıları belirli bir yola yönlendirmek için kullanılır.

pathMatch: Yolun eşleştirme stratejisini belirtir (full veya prefix).

RouterOutlet ve Dinamik Bileşen Yükleme

RouterOutlet etiketi, yönlendirme sistemi tarafından dinamik olarak bileşen yüklemek için kullanılır. Bu etiket, HTML şablonu içerisinde yer alır ve aktif olan yolun karşılık geldiği bileşeni görüntüler.

Parametreli Yönlendirme

Angular yönlendirme sistemi, URL parametrelerini destekler ve bu parametreler, bileşenler arasında veri taşımak için kullanılır.

Yönlendirme ile Durum Yönetimi

Angular yönlendirme sistemi, durum yönetimi araçlarıyla entegre çalışabilir. Örneğin, NgRx Router Store kütüphanesi, yönlendirme durumunu merkezi bir şekilde yönetmek için kullanılabilir.

AUTHENTİCATİON VE AUTHORİZATİON

Modern web teknolojilerinde güvenlik, kullanıcı verilerinin korunması ve yetkisiz erişimlerin önlenmesi açısından kritik bir öneme sahiptir. Bu bağlamda authentication (kimlik doğrulama) ve authorization (yetkilendirme) kavramları, güvenliğin temel taşları olarak karşımıza çıkmaktadır. Authentication, bir kullanıcının veya uygulamanın kimliğinin doğrulanması sürecini ifade ederken, authorization, doğrulanan bir kullanıcının belirli kaynaklara veya işlemlere erişim yetkilerini belirler. Bu iki süreç, birbirinden farklı ancak birbiriyle entegre çalışması gereken mekanizmalardır ve güvenliğin sağlanması için merkezi bir role sahiptir.

Authentication

Authentication, bir kullanıcının veya uygulamanın kimliğinin doğrulanması sürecidir. Bu süreçte genellikle kullanıcı adı ve şifre gibi geleneksel yöntemlerin yanı sıra daha modern ve güvenli çözümler de kullanılmaktadır.

Authorization

Authorization, bir kullanıcının veya uygulamanın belirli bir kaynağa veya işleve erişme iznini belirleme sürecidir. Kimlik doğrulama işlemi tamamlandıktan sonra gelen bu adım, genellikle kullanıcının erişim seviyesini tanımlar. Yetkilendirme, güvenliğin önemli bir parçasıdır çünkü yalnızca yetkili kullanıcıların verilere veya sistemlere erişmesini sağlar.

Authentication ve Authorization Arasındaki Farklar

Authentication, bir kullanıcının sisteme kim olduğunu kanıtlamasıyla ilgilenirken, authorization, bu kullanıcının hangi kaynaklara erişebileceğini belirler. Örneğin, bir kullanıcı sisteme giriş yapabilir (authentication), ancak yalnızca kendi verilerini görüntüleme izni alabilir (authorization).

OAuth2

OAuth2 (Open Authorization 2.0), kullanıcıların kimlik bilgilerini paylaşmadan, üçüncü taraf uygulamalara sınırlı erişim izni vermesini sağlayan modern bir yetkilendirme protokolüdür. Örneğin, bir kullanıcı sosyal medya hesabını bir üçüncü taraf uygulamayla bağladığında, OAuth2 mekanizması devreye girer. Kullanıcı, sosyal medya kimlik bilgilerini üçüncü taraf uygulamayla paylaşmaz; bunun yerine bir erişim izni (access token) sağlar.

OAuth2, farklı kullanım senaryolarına uyum sağlayabilmek için birden fazla akış türü sunar. Her bir akış türü, belirli bir ihtiyacı karşılamak üzere tasarlanmıştır. Akış türleri:

Authorization Code Flow: Kullanıcı tarafından yetkilendirme işlemi tamamlandıktan sonra istemciye bir yetkilendirme kodu (authorization code) sağlanır. Bu kod, istemci tarafından yetkilendirme sunucusuna gönderilerek bir erişim token'ı (access token) alınır.

Implicit Flow: Özellikle tek sayfa uygulamaları (Single Page Applications - SPA) için tasarlanmıştır. Bu akışta, access token doğrudan istemciye döner. Yetkilendirme kodu kullanılmaz, bu da süreci hızlandırır.

Resource Owner Password Flow: Bu akış, kullanıcı kimlik bilgilerini doğrudan istemciye sağlayarak çalışır. Kullanıcının kimlik bilgileri istemci üzerinden yetkilendirme sunucusuna gönderilir ve access token alınır.

Client Credentials Flow: İstemci ile yetkilendirme sunucusu arasında sistemden sisteme iletişim (machine-to-machine) için kullanılır. Bu akışta bir kullanıcı rolü yoktur; yalnızca istemcinin kimlik doğrulaması yapılır.

Oauth2 ve Güvenlik

OAuth2'nin güvenliği, protokolün doğru uygulanmasına ve uygun yapılandırılmasına bağlıdır. Yanlış bir yapılandırma, erişim token'larının kötüye kullanılması veya veri sızıntıları gibi ciddi sorunlara yol açabili

Oauth2'nin Pratik Kullanım Alanları

OAuth2 protokolü, birçok farklı sektörde ve kullanım senaryosunda güvenlik, erişim kontrolü ve kullanıcı deneyimi açısından büyük avantajlar sunmaktadır. Protokolün esnekliği, hem kullanıcılar hem de geliştiriciler için kapsamlı bir çözüm haline gelmesini sağlamıştır. Başlıca kullanım alanları:

Sosyal Medya Entegrasyonu: Birçok sosyal medya platformu, OAuth2 protokolünü kullanarak üçüncü taraf uygulamaların kullanıcı verilerine erişmesine izin verir. Bu yöntem, kullanıcıların kimlik bilgilerini paylaşmadan güvenli bir şekilde giriş yapmalarını sağlar.

API Güvenliği: OAuth2, web servisleri ve API'lerde yaygın olarak kullanılır. Özellikle RESTful API'

ler, OAuth2'yi kullanarak erişim kontrolünü etkin bir şekilde yönetir.
Mobil ve IoT Cihazlarında Kullanım: OAuth2, mobil uygulamalar ve IoT cihazlarında kimlik doğrulama ve yetkilendirme süreçlerini kolaylaştırır. Örneğin, bir akıllı termostatın hava durumu API' sine erişimi OAuth2 ile sınırlandırılabilir.
Kurumsal Uygulamalar: Kurumsal uygulamalarda, kullanıcıların ve sistemlerin erişim kontrolü için OAuth2 geniş bir şekilde uygulanmaktadır. Bu durum, özellikle büyük ölçekli organizasyonlar için verimlilik ve güvenlik sağlar.
Sağlık ve Finans Uygulamaları: Bir fitness uygulamasının kullanıcı sağlık verilerini bir doktor uygulamasıyla paylaşması sağlık alanın dakullanıma bir örnektir. Aynı şekilde açık bankacılık çözümlerinde OAuth2, müşterilerin banka hesaplarını üçüncü taraf finansal uygulamalarla bağlamasına olanak tanır.
Eğitim ve Kamu Hizmetlerinde: Bir üniversite portalına Google hesabı ile giriş yapılması veya kamu hizmetlerinde kullanıcıların çeşitli devlet portallarına e-devlet şifresi ile giriş entegrasyonu örnek olarak verilebilir.

Oauth2 ve Alternatif Protokoller

OAuth2, modern uygulamalarda güvenli yetkilendirme için bir standart haline gelmiş olsa da, bazı durumlarda diğer protokoller de tercih edilebilir.
SAML: kimlik doğrulama ve yetkilendirme bilgilerini XML tabanlı mesajlar aracılığıyla taşıyan bir protokoldür. Daha çok kurumsal tek tıklamayla oturum açma (SSO) çözümlerinde kullanılır.
JWT: Kullanıcı kimlik doğrulama bilgilerini ve diğer verileri JSON formatında taşıyan bir token standardıdır. OAuth2'de token formatı olarak yaygın bir şekilde kullanılır. JWT, OAuth2'nin bir parçası olarak çalışabilir. OAuth2 daha geniş bir protokolken, JWT sadece veri taşımak için kullanılan bir standarttır.
OpenID Connect: OAuth2 protokolünün bir uzantısıdır ve kullanıcı kimlik doğrulama işlemleri için ek özellikler sağlar. OIDC, OAuth2 üzerine bir kimlik doğrulama katmanı ekleyerek kullanıcı bilgilerini doğrular ve iletir.
Kerberos: Özellikle yerel ağlardaki kimlik doğrulama süreçlerinde kullanılan bir protokoldür. Çoğunlukla Windows tabanlı sistemlerde kullanılır.
LDAP (Lightweight Directory Access Protocol): Merkezi bir dizin hizmeti üzerinden kimlik doğrulama sağlar. Kurumsal dizin hizmetlerinde (ör. Active Directory) kullanılır.

Gelecekteki Yönelimler

OAuth2 protokolü, modern uygulamalar için güçlü bir yetkilendirme ve erişim kontrol standardı haline gelmiş olsa da, teknoloji ve güvenlik tehditlerinin evrimi ile birlikte sürekli olarak geliştirilmesi gerekmektedir.
OAuth 2.1, protokolün karmaşıklığını azaltmayı ve güvenlik açıklarını gidermeyi amaçlamaktadır ve aşağıdaki yenilikleri içermesi beklenmektedir:
Güvenli Varsayılanlar: Eski protokoller (örn. Implicit Flow) kaldırılacak ve güvenli varsayılan ayarlar kullanılacaktır.
Herkese HTTPS Zorunluluğu: Token'ların ve iletişimin güvenliği için HTTPS zorunlu hale gelecektir.
Token Binding: Token'ın belirli bir istemciye bağlanması, ele geçirilme riskini azaltacaktır.
Zero Trust modeli, hiçbir kullanıcı veya cihazın varsayılan olarak güvenilir kabul edilmediği bir güvenlik yaklaşımıdır.
Sürekli Doğrulama: Kullanıcıların ve cihazların sürekli olarak doğrulanması.
Dinamik Yetkilendirme: OAuth2'nin scope özelliği, erişim izinlerinin dinamik olarak düzenlenmesini sağlar.
Minimum Yetki İlkesi: Kullanıcılara yalnızca işlerini yapmaları için gereken minimum erişim izinlerinin verilmesi.
OAuth2 ile çalışan sistemler, makine öğrenimi algoritmaları kullanılarak daha güvenli hale getirilebilir:
Anomali Tespiti: Makine öğrenimi, kullanıcı davranışlarını analiz ederek anormal veya kötü niyetli girişimleri tespit edebilir.
Proaktif Güvenlik Önlemleri: Örneğin, olağandışı bir erişim isteği geldiğinde, kullanıcıdan ek kimlik doğrulama talep edilebilir.
Blockchain teknolojisi, OAuth2'ye ek bir güvenlik katmanı sağlayabilir:
Dağıtık Yetkilendirme: Yetkilendirme kararlarının merkezi bir sunucu yerine, blockchain üzerinde alınmasıdır.
Token Güvenliği: Token bilgileri, blockchain üzerinde şifrelenmiş bir şekilde saklanabilir.

Node.js ve Temel Özellikleri

Node.js, 2009 yılında Ryan Dahl tarafından geliştirilen ve sunucu tarafında JavaScript çalıştırmak için kullanılan açık kaynaklı bir platformdur. Temelinde Google'ın V8 JavaScript motoru bulunur. Bu motor, JavaScript kodlarını doğrudan makine diline çevirerek performansı önemli ölçüde artırır. Node.js'nin en belirgin özelliklerinden biri, asenkron ve olay odaklı mimarisidir. Bu yapı, işlemlerin birbirini beklemeden yürütülmesini sağlar ve yüksek performanslı ağ uygulamaları oluşturmayı mümkün kılar. Non-blocking yapısı, geleneksel bloklama tabanlı sistemlerin performans sınırlamalarını aşarak aynı anda çok sayıda bağlantıyı etkin bir şekilde yönetir. Ayrıca, Node.js'nin NPM (Node Package Manager) desteği, dünya çapındaki geliştiriciler tarafından oluşturulan binlerce modülü projelere kolayca entegre etme olanağı sunar. Bu özellikler, Node.js'nin modern web uygulamalarında sıkça tercih edilmesini sağlar.

Kullanım Alanları

Node.js, geniş bir kullanım yelpazesine sahip olup özellikle gerçek zamanlı uygulamalarda öne çıkar. Sohbet uygulamaları, canlı veri akışı projeleri, oyun sunucuları ve RESTful API geliştirme gibi projeler için idealdir. Non-blocking yapısı, bu tür uygulamalarda kritik önem taşır, çünkü yüksek veri akışını ve çoklu kullanıcı taleplerini eş zamanlı olarak yönetir. Node.js, mikro hizmet mimarisi desteğiyle büyük ve karmaşık projelerde de kullanılabilir. Bu sayede projeler daha küçük, bağımsız birimlere ayrılarak yönetim ve ölçeklenebilirlik kolaylaştırılır. Ancak, CPU yoğun işlemler için uygun olmaması, hesaplama gerektiren projelerde sınırlamalar yaratabilir. Bu tür uygulamalarda performans sorunları yaşanabileceğinden alternatif teknolojiler tercih edilmelidir.

Avantajlar ve Dezavantajlar

Node.js'nin en önemli avantajlarından biri, JavaScript'in hem istemci hem de sunucu tarafında kullanılabilmesidir. Bu durum, tam yığın geliştirme süreçlerini hızlandırır ve geliştiricilerin öğrenme eğrisini azaltır. Asenkron programlama modeli, yüksek performanslı uygulamalar oluşturmayı kolaylaştırır ve sistem kaynaklarının etkin kullanımını sağlar. Ayrıca, NPM aracılığıyla geniş bir modül ve kütüphane ekosistemi sunar. Ancak, Node.js'nin bazı dezavantajları da vardır. Asenkron programlama, özellikle yeni başlayanlar için karmaşık bir yapı sunabilir ve hata ayıklama süreçlerini zorlaştırabilir. Tek iş parçacıklı yapısı, CPU yoğun işlemlerde performans kayıplarına neden olabilir. Bunun yanı sıra, açık kaynak ekosistemindeki bazı kütüphanelerin yetersiz test edilmesi, güvenlik açıkları ve uyumluluk sorunlarına yol açabilir. Büyük projelerde bellek tüketiminin yüksek olması, maliyetleri artırabilir ve yazılımın yönetimini karmaşıklaştırabilir.

Node.js Kurulumu ve Geliştirme Ortamı

Node.js'nin kurulumu oldukça basittir ve Windows, macOS, Linux gibi birçok işletim sisteminde çalışabilir. Node.js'nin resmi web sitesinden indirilen kurulum dosyası, birkaç adımda kolayca yüklenebilir. Kurulumun ardından geliştiriciler, terminal üzerinden JavaScript kodlarını çalıştırabilir ve HTTP sunucuları oluşturabilir. Node.js, "http" ve "fs" gibi yerleşik modülleriyle güçlü bir geliştirme ortamı sunar. Örneğin, "http" modülü ile bir sunucu oluşturulabilirken, "fs" modülü dosya işlemleri için kullanılabilir. Harici dosyalar ve yerel kütüphanelerle geliştirme süreçleri daha modüler ve esnek hale getirilebilir. Ayrıca, NPM üzerinden eklenen kütüphaneler, uygulamaların işlevselliğini artırır ve geliştirme süreçlerini hızlandırır.

Harici ve Yerel Kütüphane Kullanımı

Node.js, geniş bir standart modül kütüphanesi sunar ve aynı zamanda geliştiricilerin kendi kütüphanelerini oluşturmalarına olanak tanır. Örneğin, "http" modülü ile web sunucuları oluşturulabilirken, "fs" modülü dosya işlemleri için kullanılabilir. Bunun yanı sıra, geliştiriciler özel ihtiyaçlarına göre kendi kütüphanelerini oluşturabilir ve bunları projelerinde tekrar kullanabilir. Bu özellik, projelerin sürdürülebilirliğini artırır ve kod yönetimini kolaylaştırır. Ayrıca, NPM üzerinden indirilen kütüphanelerle uygulamalara yeni işlevler eklenebilir.

Express.js ile Güçlendirme

Express çerçevesi web ve mobil uygulamalar için özellik seti sağlamaktadır. Express çerçevesi isteklerin ve yanıtların işleme sürecini kolaylaştıran çeşitli araçlar ve ara yazılımlar sunarak web uygulamalarının ve API'lerin geliştirilmesini kolaylaştırmak üzere tasarlanmıştır.

Express çerçevesi kullanmanın temel avantajlarından biri, RESTful API'leri hızlı ve verimli bir şekilde oluşturma yeteneğidir. Çerçeve, geliştiricilerin farklı HTTP yöntemleri (GET, POST, PUT,

DELETE) için rotalar tanımlamasına ve istekleri son rota işleyicisine ulaşmadan önce işleyebilen ara yazılım işlevlerini kolayca yönetmesine olanak tanır.
Express çerçevesi JavaScript dilinde yazılmış olup asenkron olay odaklı bir çalışma zamanı olan Node.js üzerinde çalışır.
Dahası Express çerçevesi geliştiricilerin dinamik HTML sayfalarını kolayca oluşturmasına olanak tanıyan çok çeşitli şablonlama motorlarını destekler. Bu özellik, istemciye göndermeden önce sunucuda HTML içeriğinin oluşturulmasını sağladığı için sunucu tarafında oluşturma gerektiren uygulamalar için özellikle yararlıdır. Ek olarak Express çerçevesi kullanılarak MongoDB ve MySQL gibi çeşitli veritabanlarıyla sorunsuz bir şekilde entegre olur.

WebSockets ve Temel Özellikleri

WebSockets, istemciler ve sunucular arasında gerçek zamanlı, çift yönlü bir iletişim sağlayan modern bir protokoldür. İlk olarak 2011 yılında RFC 6455 standardıyla tanımlanmış ve İnternet Mühendisliği Görev Gücü (IETF) tarafından standartlaştırılmıştır. Geleneksel HTTP protokollerinin istek-yanıt modelinden farklı olarak WebSockets, sürekli açık bir bağlantı sağlar. Bu özellik, gerçek zamanlı veri akışı ve düşük gecikme gerektiren uygulamalar için büyük bir avantaj sunar. WebSockets, iletişim için bir TCP bağlantısını kullanır ve bu bağlantı, ilk el sıkışma sırasında HTTP üzerinden kurulur. Daha sonra protokol, WebSocket protokolüne geçerek veri iletiminde minimum ek yükle çalışır.

WebSockets, finansal ticaret platformları, çevrimiçi oyunlar ve canlı izleme sistemleri gibi uygulamalarda sıklıkla tercih edilir. Bu protokolün verimliliği, özellikle IoT cihazları gibi sınırlı kaynaklara sahip ortamlar için kritiktir. Sürekli açık bağlantısı sayesinde, istemci ve sunucu arasında tekrarlayan bağlantı talepleri gerekmez, bu da hem kaynak kullanımını optimize eder hem de bant genişliğinden tasarruf sağlar.

WebSockets'in Avantajları

WebSockets'in en büyük avantajlarından biri düşük gecikmeli iletişim sağlamasıdır. Geleneksel HTTP protokolünde her bir veri iletimi için bağlantı kurulur ve kapatılır. Ancak WebSockets, sürekli bir bağlantıyı koruyarak veri aktarım hızını artırır. Bu durum, özellikle finansal uygulamalarda veya canlı veri akışı gerektiren senaryolarda kritik öneme sahiptir. Örneğin, bir finansal ticaret platformunda WebSockets kullanılarak piyasa verileri anlık olarak kullanıcıya iletilebilir.

WebSockets ayrıca bant genişliği kullanımını optimize eder. HTTP protokolünün aksine, veri iletimi sırasında büyük başlık bilgileri gönderilmez. Bu da mobil cihazlar gibi sınırlı veri tüketimi gereken ortamlarda avantaj sağlar. WebSockets'in çift yönlü iletişim yapısı, hem istemcinin hem de sunucunun birbirlerine anlık olarak veri göndermesine olanak tanır. Bu özellik, oyunlar, sohbet uygulamaları ve canlı yayın hizmetleri gibi etkileşimli uygulamalarda kritik bir avantaj sağlar.

Güvenlik açısından, WebSockets protokolü Taşıma Katmanı Güvenliği (TLS) ile şifrelenebilir. Bu, özellikle finansal işlemler veya hassas verilerin iletildiği uygulamalarda gizliliği ve bütünlüğü garanti eder. WebSockets'in mevcut güvenlik protokolleriyle entegrasyonu, uygulamaların genel güvenlik durumunu iyileştirir.

WebSockets'in Dezavantajları

Avantajlarına rağmen, WebSockets'in bazı dezavantajları da bulunmaktadır. Bunlardan ilki, eski tarayıcıların WebSockets desteğinin olmamasıdır. Bu durum, bazı uygulamalar için uyumluluk sorunlarına yol açabilir. Ayrıca, WebSockets sunucusu her istemciyle sürekli bir bağlantı sürdürmek zorunda olduğundan, sunucu tarafında daha fazla kaynak tüketimine neden olabilir. Bu, büyük ölçekli uygulamalarda yönetim karmaşıklığını artırabilir. Güvenlik açısından, WebSockets standart bir protokol olmadığı için ek güvenlik önlemleri alınması gerekebilir. Özellikle kimlik doğrulama ve veri bütünlüğü gibi alanlarda ekstra dikkat edilmesi gerekir.

WebSockets Bağlantı Yaşam Döngüsü

WebSockets bağlantısı dört temel aşamada gerçekleşir:

1. El Sıkışması: Tarayıcı, sunucuya bir bağlantı isteği gönderir.
2. Bağlantı Kurulumu: Sunucu bu isteği kabul eder ve bağlantı kurulur.
3. Veri İletimi: Çift yönlü veri akışı başlar; her iki taraf da birbirine istediği zaman veri gönderebilir.
4. Bağlantıyı Kapatma: Bağlantı, tarayıcının veya sunucunun isteği üzerine kapatılır.

Bu yaşam döngüsü, sürekli veri iletimi gerektiren uygulamalarda basit ve etkili bir iletişim modeli sunar.

WebSockets ve HTTP Karşılaştırması

HTTP protokolü, istek-yanıt modeline dayanır ve her veri iletiminde yeni bir bağlantı açıp kapatır. Bu, yüksek gecikme ve fazla bant genişliği tüketimi ile sonuçlanabilir. Buna karşılık, WebSockets yalnızca başlangıçta bir HTTP el sıkışması gerektirir ve ardından sürekli bir bağlantı üzerinden veri iletimine olanak tanır. Bu özellik, WebSockets'i sık veri alışverişi gerektiren gerçek zamanlı uygulamalar için ideal hale getirir. Ancak, HTTP'nin daha basit yapısı ve yaygın desteği, gerçek zamanlı iletişim gerektirmeyen uygulamalar için tercih sebebi olmaya devam etmektedir.

Node.js ve WebSockets

Node.js, WebSocket programlama için ideal bir platformdur. Olay odaklı mimarisi, aynı anda birden fazla bağlantıyı etkin bir şekilde yönetmesine olanak tanır. Node.js ile WebSockets uygulamaları geliştirmek için genellikle "ws" ve "Socket.IO" gibi kütüphaneler kullanılır.

•"ws" Kütüphanesi: Bu kütüphane hafif ve yüksek performanslıdır. Gerçek zamanlı uygulamalar için ideal bir çözüm sunar.

• Socket.IO: WebSocket desteği olmayan durumlarda alternatif iletişim yöntemlerine geçiş yapabilen bir kütüphanedir. Esnek yapısı sayesinde geniş bir uyumluluk sağlar ve geliştiricilere kolay bir API sunar.

Node.js'nin Express.js gibi diğer çerçevelerle entegrasyonu, hem geleneksel HTTP işlemleri hem de WebSocket iletişimlerini aynı projede sorunsuz bir şekilde yönetmeyi mümkün kılar.

WEBRTC

Gerçek zamanlı iletişim teknolojileri, gelişmiş web teknolojilerinin önemli bir parçası olarak karşımıza çıkmaktadır. İnternet üzerinden sesli ve görüntülü iletişim, anlık mesajlaşma ve veri paylaşımı gibi ihtiyaçlar, kullanıcıların eklenti veya harici yazılımlar kullanmadan tarayıcı tabanlı çözümlere yönelmesine neden olmuştur. Bu noktada, WebRTC gibi yenilikçi teknolojiler, bireylerin ve işletmelerin iletişim süreçlerini köklü bir şekilde değiştirmektedir.

WebRTC Nedir?

WebRTC (Web Real-Time Communication [Web Gerçek Zamanlı İletişim]), internet tarayıcıları ve mobil uygulamalar üzerinden anlık ses, video ve veri iletişimi gerçekleştirilmesini sağlayan bir teknoloji standardıdır. Herhangi bir eklenti ya da üçüncü taraf yazılım gerektirmeksizin çalışan bu teknoloji, kullanıcı dostu ve esnek yapısıyla öne çıkar. WebRTC'nin amacı, düşük gecikme süreleriyle doğrudan ve güvenli iletişim sağlamaktır.

WebRTC'nin en dikkat çekici özelliklerinden biri, tarayıcılar arası uçtan uca (peer-to-peer) bağlantı kurarak çalışmasıdır. Bu durum, merkezi bir sunucuya olan bağımlılığı azaltır ve veri akışını daha hızlı hale getirir. Ayrıca, WebRTC kullanılarak hem medya (ses ve video) hem de diğer veri türleri doğrudan aktarılabilir.

WebRTC ilk olarak 2011 yılında Google tarafından geliştirildi ve açık kaynaklı bir proje olarak yayınlandı.

Neden WebRTC?

WebRTC'nin tercih edilmesindeki başlıca nedenler şunlardır:

- Eklenti Gerektirmemesi:
- Açık Kaynak ve Ücretsiz Olması
- Düşük Gecikme Süresi
- Çoklu Platform Desteği
- Güçlü Güvenlik Özellikleri

WebRTC'nin Teknik Kavramları

RTCPeerConnection: Ses ve video gibi medya akışlarının tarayıcılar arasında aktarılmasını sağlar.

RTCDataChannel: WebRTC'nin veri aktarımı için kullandığı bileşendir. Anlık veri paylaşımı (örneğin metin mesajları, dosya transferi) bu kanaldan yapılır.

MediaStream API: Kullanıcı tarayıcısı, medya kaynağını MediaStream API ile alır ve bu veri RTCPeerConnection üzerinden diğer tarayıcıya aktarılır.

WebRTC'nin Mimarisi

WebRTC mimarisi, şu bileşenleri içerir:

Sinyalleme (Signaling): İki tarayıcı arasındaki; IP adresi, bağlantı noktası, medya formatları gibi bilgilerin karşılıklı olarak paylaşımıdır.

Bağlantı Yönetimi: WebRTC, NAT gibi ağ engellerini aşmak ve bağlantıyı optimize etmek için ICE (Interactive Connectivity Establishment) protokolünü kullanır. ICE, STUN ve TURN sunucularından faydalanarak, kullanıcıların karmaşık ağ ortamlarında bile bağlantı kurmasını sağlar:

- Sunucuları: Kamuya açık bir IP adresi alır ve bağlantı için gerekli ağ bilgilerini sağlar.
- TURN Sunucuları: Tarayıcılar arasında doğrudan bağlantı mümkün olmadığında veri aktarımı için bir aracı görevi görür.

Güvenlik Protokolleri: WebRTC, hem veri hem de medya akışlarını şifrelemek için DTLS ve SRTP protokollerini kullanır.

WebRTC'de Güvenlik

P2P (peer-to-peer) Şifreleme: WebRTC, peer-to-peer (uçtan uca) şifreleme kullanarak üçüncü taraf erişimlerini engeller.

Kullanıcı İzinleri: WebRTC, kullanıcıların cihazlarına (mikrofon, kamera vb.) erişmeden önce mutlaka izin alır.

WebRTC Nasıl Çalışır?

Karşılıklı iletişim kurma süreci üç ana aşamaya ayrılır: sinyalleme, bağlantı kurulumu ve medya/veri aktarımı.

Sinyalleme

Sinyalleme iki taraf arasında gerekli bilgilerin karşılıklı değişimidir. Sinyalleme sunucuları, sadece

bağlantının ilk aşamasında devreye girer ve bağlantı kurulduktan sonra devre dışı kalır. Bu durum, bağlantının uçtan uca (peer-to-peer) kalmasını sağlar.

Bağlantı Kurulumu

Tarayıcılar arasında uçtan uca (peer-to-peer) bir bağlantı kurulur.

Uçtan Uca İletişim (Peer-to-Peer Communication)

Bağlantı kurulduktan sonra uçtan uca medya ve veri akışı gerçekleşir.

Medya Akışı (Media Stream): Kullanıcıların mikrofon ve kameralarından alınan medya, RTCPeerConnection üzerinden diğer tarafa iletilir.

Veri Akışı (Data Channel): RTCDataChannel, medya dışındaki verilerin (örneğin, dosyalar, metin mesajları) aktarımını sağlar. Veri aktarımı çift yönlüdür (full-duplex) ve düşük gecikme sürelerine sahiptir.

WebRTC'nin Çalışma Örneği

Aşağıdaki adımlar bir WebRTC uygulamasının nasıl çalıştığını anlamak için güzel bir örnektir:

- Kullanıcılar bir video konferans uygulamasına giriş yapar.
- Tarayıcılar, sinyalleme sunucusuna bağlanarak SDP ve aday bilgilerini paylaşır.
- ICE protokolü ile uçtan uca bir bağlantı kurulur.
- Mikrofon ve kameradan alınan medya akışı MediaStream API üzerinden karşı tarafa aktarılır.
- Görüşme tamamlandığında bağlantı otomatik olarak sonlandırılır.

WebRTC'nin Bileşenleri

WebRTC'nin işleyişi, ses, video ve veri aktarımını mümkün kılan bir dizi güçlü bileşene dayanır. Bu bileşenler, tarayıcılar arasında doğrudan iletişim sağlayan teknik altyapının temelini oluşturur. WebRTC'nin ana bileşenleri şunlardır:

RTCPeerConnection

RTCPeerConnection, WebRTC'nin temel taşıdır ve medya akışlarının (ses ve video) uçtan uca (peer -to-peer) bağlantı ile tarayıcılar arasında doğrudan aktarılmasını sağlar. RTCPeerConnection, özellikle video konferans ve canlı yayın gibi düşük gecikme gerektiren uygulamalarda kritik bir role sahiptir.

RTCDataChannel

RTCDataChannel, medya dışındaki veri türlerinin aktarımı için kullanılan bir bileşendir. Bu kanal, yüksek hızlı ve çift yönlü (full-duplex) veri iletişimi sağlar. Özellikle oyunlar, dosya paylaşımı ve gerçek zamanlı mesajlaşma gibi uygulamalarda yaygın olarak kullanılır.

MediaStream API

MediaStream API, kullanıcı cihazlarından (mikrofon, kamera veya ekran) medya verilerinin alınmasını ve aktarılmasını sağlar. MediaStream API, cihaz erişimi öncesinde mutlaka kullanıcıdan izin ister. Bu sayede gizliliğin korunmasını sağlanmış olur.

Güvenlik Protokolleri

WebRTC'nin bileşenleri, kullanıcı verilerini korumak için çeşitli güvenlik protokolleriyle entegre çalışır:

- SRTP (Secure Real-Time Transport Protocol): Ses ve video akışlarının şifrelenmesini sağlar.
- DTLS (Datagram Transport Layer Security): Veri kanalında güvenli veri aktarımı için uçtan uca şifreleme sağlar.
- ICE/STUN/TURN: Bağlantının güvenli ve kesintisiz bir şekilde kurulmasını destekler.

WebRTC'nin Avantajları

WebRTC, gerçek zamanlı iletişim için sunduğu yenilikçi özellikler sayesinde sağladığı avantajlar, hem bireysel kullanıcılar hem de işletmeler için WebRTC'nin tercih edilmesine sebep olur:

Eklenti Gerektirmemesi

WebRTC, herhangi bir tarayıcı eklentisi veya üçüncü taraf yazılım yüklenmesine gerek kalmadan çalışır. Kullanıcılar yalnızca uyumlu bir tarayıcıya ihtiyaç duyar.

Örneğin Google Meet doğrudan tarayıcı üzerinden çalışarak hızlı ve kolay erişim / iletişim sağlar.

Düşük Gecikme Süresi

WebRTC'nin uçtan uca (peer-to-peer) bağlantı mimarisi, medya ve verinin doğrudan aktarılmasını mümkün kılar. Bu sayede, aracı sunucuların neden olduğu gecikme süreleri ortadan kalkar. Özellikle gerçek zamanlı video konferanslar ve canlı yayınlar gibi uygulamalarda bu düşük gecikme avantajı kritik öneme sahiptir.

Güçlü Güvenlik Özellikleri

WebRTC, SRTP (Secure Real-Time Transport Protocol) ve DTLS (Datagram Transport Layer Security) gibi güvenlik protokollerini kullanarak medya ve veri akışlarını şifreler. Ayrıca, MediaStream API aracılığıyla cihazlara erişim sağlanmadan önce kullanıcı izni talep eder.

Çoklu Platform ve Tarayıcı Desteği

WebRTC, başlıca modern tarayıcılar (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) ve mobil platformlar tarafından desteklenir. Ayrıca WebRTC, cihaz bağımsız bir yapıya sahiptir; bu nedenle masaüstü, mobil ve tablet gibi farklı cihazlarda sorunsuz çalışabilir.

Açık Kaynak ve Ücretsiz Olması

WebRTC’nin açık kaynaklı ve ücretsiz olması, geliştiricilerin teknolojiyi kolayca incelemesine ve özelleştirmesine olanak tanır. Bu özellik, işletmelerin maliyetlerini düşürürken, yenilikçi çözümler geliştirme fırsatı da sunar.

Uygulama Geliştirme Kolaylığı

WebRTC, geliştiriciler için kolay entegre edilebilen bir teknoloji sunar. Sinyalleme ve bağlantı süreçlerinin standartlara uygun olması, uygulama geliştirme sürecini hızlandırır. Ayrıca, açık kaynaklı API’ler ile güçlü dokümantasyon ve topluluk desteği bulunmaktadır.

Düşük Maliyetli Çözümler Sunması

WebRTC’nin uçtan uca mimarisi, merkezi sunuculara olan bağımlılığı azaltır. Bu durum, sunucu maliyetlerini düşürür ve daha ölçeklenebilir çözümler sunar. WebRTC'nin ücretsiz bir teknoloji olması, özellikle küçük işletmeler ve bireysel kullanıcılar için büyük bir avantajdır.

Geniş Kullanım Alanı

WebRTC, video konferans, canlı yayın, anlık mesajlaşma, dosya paylaşımı, oyunlar ve eğitim uygulamaları gibi birçok farklı alanda kullanılabilir. Bu çeşitlilik, WebRTC teknolojisinin esnekliğini ve gücünü ortaya koyar.

WebRTC’nin Kullanım Alanları

WebRTC, geniş bir uygulama yelpazesiyle dijital iletişim teknolojilerinde devrim yaratmıştır. Sağladığı esneklik, düşük gecikme süreleri ve yüksek performans sayesinde, video konferans, canlı yayın, müşteri desteği, eğitim, sağlık ve daha birçok sektör ve uygulama alanında yaygın bir şekilde kullanılmaktadır.