

PIKSELDEN ÇİMENTOYA

Fiziksel Mimarlık İlkeleriyle Modüler Frontend Mimarisi İnşa Etmek

· Esra Sağın ·

“Bir arayüzü yalnızca ekranda görünen pikseller olarak değil, yük taşıyan, dolaşım kuran ve kullanıcıyı yönlendiren bir yapı olarak düşünürsek; frontend mimarisi çok daha anlaşılır hâle gelir.”

Özet

Bu çalışma, fiziksel mimarlık ile frontend geliştirme arasında bire bir eşitlik kurmak yerine, iki disiplinin karmaşıklığı yönetme biçimlerini karşılaştıran bir düşünme modeli sunar. Vitruvius’un firmitas, utilitas ve venustas üçlemesi; kod dayanıklılığı, kullanılabilirlik ve görsel bütünlük başlıkları üzerinden yeniden yorumlanır. Ardından Atomic Design, tasarım sistemleri, bileşen sözleşmeleri ve mikro ön yüzler; prefabrikasyon, malzeme katalogları, kat planları ve kampüs yerleşimleri gibi mimari kavramlarla ilişkilendirilir. Metin, benzetmelerin yararlı olduğu noktaları gösterirken sınırlarını da açıkça belirtir; böylece estetik bir metaforun ötesine geçerek uygulanabilir bir frontend mimarisi rehberine dönüşür.

1. Mimarlık Metaforu Neden İşe Yarar?

Bir bina, dışarıdan bakıldığında tek bir bütün gibi görünür. Oysa o bütün; temel, taşıyıcı sistem, dolaşım alanları, cephe, mekanik tesisat ve ince işlerin birbirine bağlanmasıyla oluşur. Modern bir web uygulaması da benzer biçimde tek bir ekran izlenimi verir; fakat arka planda veri modelleri, bileşenler, yönlendirme, durum yönetimi, erişilebilirlik kuralları ve görsel stiller birlikte çalışır.

Buradaki benzerlik, yazılımın gerçekten betonla inşa edildiği anlamına gelmez. Mimarlık metaforu; görünmeyen ilişkileri görünür kılmak, sorumluluk sınırlarını konuşmak ve “bu parça neden burada?” sorusunu sormak için yararlıdır. Bir geliştirici bileşen ağacını kurarken yalnız kod yazmaz; kullanıcıların dijital mekânda nasıl dolaşacağını, hangi yapıların tekrar kullanılacağını ve değişimin hangi bölgelere yayılacağını da tasarlar.

Bu nedenle frontend mimarisi, dosya klasörlerinin düzeninden daha geniş bir konudur. İyi bir mimari; değişiklik maliyetini düşürür, ekiplerin aynı dili konuşmasını sağlar ve kullanıcı deneyimini tesadüfe bırakmaz. Kötü bir mimari ise ilk gün çalışsa bile büyüdükçe kırılganlaşır: aynı butonun beş farklı sürümü oluşur, ekranlar birbirinden kopar ve küçük bir değişiklik beklenmedik alanları etkiler.

Temel düşünce

Mimari, parçaların yalnızca var olması değil; birbirleriyle hangi kurallar üzerinden ilişki kurduğudur.

2. Vitruvius Üçlemesini Arayüze Çevirmek

Vitruvius’un mimari kaliteyi firmitas, utilitas ve venustas başlıklarıyla ele alan üçlü yaklaşımı, yüzyıllar sonra bile tasarım tartışmalarında kullanılan güçlü bir çerçevedir [1]. Bu üçleme frontend için kesin bir ölçüm standardı değildir; fakat kaliteyi yalnız görselliğe indirgememek açısından işlevsel bir mercek sunar.



2.1. Firmitas: Kodun Ayakta Kalma Yeteneği

Bir yapının sağlamlığı yalnız kullanılan malzemenin sertliğine değil, yüklerin nasıl aktarıldığına ve hataların nasıl karşılandığına bağlıdır. Frontend tarafında firmitas; anlaşılır veri sözleşmeleri, otomatik testler, hata sınırları, izlenebilirlik ve kontrollü bağımlılıklarla kurulur. TypeScript, program çalışmadan önce bazı tür uyumsuzluklarını belirleyebilen statik bir denetleyici olarak bu yapıya katkı verir [2]. Ancak tip denetimi tek başına yeterli değildir; dış API'den gelen veriler çalışma zamanında ayrıca doğrulanmalı, ağ hataları ve boş durumlar tasarımın parçası olarak ele alınmalıdır.

- Bileşenin hangi veriyi kabul ettiğini açıkça tanımlamak.
- Kritik kullanıcı akışlarını otomatik testlerle korumak.
- Hata, yüklenme ve boş veri durumlarını sonradan eklenen ayrıntılar değil, temel senaryolar olarak tasarlamak.
- Bağımlılıkları güncellemek, ancak değişikliklerin etkisini ölçmeden sürüm yükseltmemek.

2.2. Utilitas: Kullanıcının Dijital Mekânda Yolunu Bulması

Fiziksel bir yapıda kapının yeri, koridorun genişliği ve yönlendirme levhaları nasıl kullanımı belirliyorsa; bir arayüzde bilgi mimarisi, odak sırası, form etiketleri ve geri bildirim mesajları da dijital dolaşımı belirler. Semantik HTML, tarayıcıların ve yardımcı teknolojilerin arayüzün anlamını kavramasına yardımcı olur. W3C'nin temel yaklaşımı, uygun bir yerel HTML ögesi varken aynı davranışı genel bir öğeye ARIA rolü ekleyerek taklit etmemektir [3]. Başka bir deyişle erişilebilirlik, son aşamada eklenen bir rampa değil; ilk plandan itibaren düşünülmesi gereken dolaşım sistemidir.

- Başlık seviyelerini görsel büyüklüğe göre değil, içerik hiyerarşisine göre kurmak.
- Form alanlarını görünür etiketlerle ilişkilendirmek.
- Klavye ile tüm etkileşimlere ulaşılabildiğini doğrulamak.
- Renk dışındaki ipuçlarını da kullanarak hata ve durum bilgisini iletme.

2.3. Venustas: Süsleme Değil, Algısal Bütünlük

Venustas yalnız “güzel renkler” anlamına gelmez. Tutarlı boşluklar, okunabilir tipografi, anlaşılır görsel hiyerarşi ve ölçülü mikro etkileşimler; kullanıcının sistemi daha hızlı anlamasını sağlar. Estetik burada işlevin rakibi değil, onun görünür hâle gelme biçimidir. Aynı eylemin farklı ekranlarda farklı renkte, farklı konumda ve farklı ifadeyle sunulması; binadaki her kapının başka bir yönde açılmasına benzer. Tasarım tokenları ve ortak bileşenler bu dağınıklığı azaltır.

İlke	Frontend karşılığı	Sorulacak soru	Örnek doğrulama
Firmitas	Kod dayanıklılığı	Bir hata tüm ekranı etkiliyor mu?	Tip denetimi, test, hata sınırı
Utilitas	Kullanılabilirlik ve erişilebilirlik	Kullanıcı görevi kolayca tamamlıyor mu?	Klavye testi, semantik kontrol, kullanıcı testi

İlke	Frontend karşılığı	Sorulacak soru	Örnek doğrulama
Venustas	Görsel ve etkileşimsel bütünlük	Arayüz kendi dilini tutarlı konuşuyor mu?	Tasarım tokenı, görsel regresyon, içerik tutarlılığı

3. Modülerlik: Atomik Tasarım ve Bileşen Sınırları

Prefabrikasyonun mimarlığa kazandırdığı asıl değer “hatasız üretim” değil; tekrarlanabilirlik, standardizasyon ve montaj hızıdır. Frontend bileşenleri de benzer bir avantaj sağlar: bir arayüz parçası açık bir sözleşmeyle tanımlandığında farklı ekranlarda yeniden kullanılabilir, test edilebilir ve merkezi olarak geliştirilebilir.

Brad Frost’un Atomic Design yaklaşımı arayüzü beş düzeyde ele alır: atomlar, moleküller, organizmalar, şablonlar ve sayfalar [4]. Bu beşli yapı katı bir klasör kuralı değildir. Amaç, küçük parçaların hangi bağlamlarda birleşerek daha büyük deneyimler oluşturduğunu görünür kılmaktır.



3.1. Yeniden Kullanılabilirlik Her Yerde Kullanmak Değildir

Bir pencere modülünün her cephede aynı biçimde kullanılabilmesi avantajdır; ancak iklim, yön ve kullanım koşulları göz ardı edilirse standart parça yanlış çözüme dönüşebilir. Bileşenlerde de benzer bir risk vardır. “Her ihtimali karşılayan” dev bir bileşen, onlarca seçenek ve koşullu davranış nedeniyle anlaşılmaz hâle gelebilir. İyi bir bileşen genel amaçlı olmaya çalışmak yerine tek bir sorumluluğu net biçimde üstlenir; farklılıkları kontrollü varyantlarla ifade eder.

3.2. Props Bir Bileşenin Teknik Şartnamesidir

React bileşenleri, üst bileşenden alt bileşene bilgi taşımak için props kullanır [5]. Mimari benzetmeyle props; bir prefabrik parçanın ölçü, malzeme ve bağlantı şartlarını belirleyen teknik şartnameye benzer. Şartname ne kadar açık olursa parça o kadar öngörülebilir kullanılır. Aşağıdaki örnek, erişilebilir etiket, hata durumu ve yerel input özelliklerini tek bir sözleşmede birleştirir:

```
import type { ComponentPropsWithoutRef } from 'react';

type FieldProps = ComponentPropsWithoutRef<'input'> & {
  id: string;
  label: string;
  errorMessage?: string;
};

export function TextField({
  id,
  label,
  errorMessage,
  ...inputProps
}: FieldProps) {
  const errorId = `${id}-error`;

  return (
    <div className="field">
      <label htmlFor={id}>{label}</label>
      <input
        id={id}
        aria-invalid={Boolean(errorMessage)}
        aria-describedby={errorMessage ? errorId : undefined}
        {...inputProps}
      />
      {errorMessage && (
        <p id={errorId} role="alert">{errorMessage}</p>
      )}
    </div>
  );
}
```

Bu örneğin modülerliği yalnız farklı renk varyantlarından gelmez. Bileşen, yerel HTML davranışını korur; etiket ve hata mesajı ilişkisini standartlaştırır; aynı zamanda çağrıldığı yere göre gerekli input özelliklerini kabul eder. Böylece yeniden kullanım, kopyala-yapıştır kolaylığından çok davranış tutarlılığı sağlar.

4. Tasarım Sistemi: Dijital Malzeme Kataloğu

Mimarlar yalnız tek tek yapı elemanlarıyla değil; malzeme katalogları, detay çizimleri, ölçülendirme kuralları ve uygulama standartlarıyla çalışır. Frontend’de tasarım sistemi de yalnız bir “buton klasörü” değildir. Tasarım tokenları, bileşenler, kullanım örüntüleri, dokümantasyon, sürümleme ve katkı süreçleri birlikte yönetildiğinde gerçek bir sistem ortaya çıkar.



Tasarım tokenları; renk, boşluk, tipografi ve hareket gibi kararları isimlendirilmiş değerlere dönüştürür. Örneğin tek tek ekranlara #9A4E3B yazmak yerine color-action-primary gibi anlam taşıyan bir token kullanmak, gelecekteki tema veya marka değişikliklerini daha kontrollü hâle getirir. Bileşenler bu tokenları kullanır; örüntüler bileşenlerin görev akışlarında nasıl birleşeceğini gösterir; yönetim ise sistemin kim tarafından ve hangi kurallarla değiştirileceğini tanımlar.

Önemli ayırım

Bir bileşen kütüphanesi “hangi parçalar var?” sorusunu yanıtlar. Tasarım sistemi buna ek olarak “bu parçalar neden böyle, nerede kullanılmalı ve nasıl değiştirilmeli?” sorularını da yanıtlar.

4.1. Merkezi Güncellemenin Gücü ve Riski

Fiziksel bir binadaki aynı tuğlalar, tek bir katalog değiştiğinde kendiliğinden dönüşmez. Dijital sistemlerde ise merkezi bir bileşenin yeni sürümü birçok ekrana yayılabilir. Bu büyük bir hız avantajıdır; aynı zamanda değişiklik yönetimi sorumluluğu doğurur. Tasarım sistemi bileşeni güncellenmeden önce görsel regresyon, erişilebilirlik, geriye dönük uyumluluk ve sürüm notları düşünülmelidir. Merkezi değişiklik, merkezi etki anlamına gelir.

5. Blueprint’ten Yayına: Frontend Yaşam Döngüsü

Kat planı inşaata başlamadan önce mekânsal kararların tartışılmasını sağlar. Wireframe ve prototipler de veri ayrıntıları ve görsel ciladan önce kullanıcı akışını görünür hâle getirir. Ancak

frontend geliştirme doğrusal bir “çiz ve bitir” süreci değildir. Kullanım verileri, erişilebilirlik testleri ve teknik gözlem sonuçları mimariyi sürekli geri besler.



İhtiyacı tanımla: Kullanıcının hangi görevi, hangi bağlamda ve hangi kısıtlarla tamamlayacağını açıkla.

Akışı kur: Sayfaları değil önce görev adımlarını ve olası hata yollarını tasarla.

Bileşen sınırlarını belirle: Tekrar eden görsel benzerliklerden önce tekrar eden davranışları bul.

Kodla ve belgele: Bileşenin kabul ettiği verileri, durumlarını ve kullanım dışı senaryolarını açıkça göster.

Test et: Fonksiyonel test, erişilebilirlik kontrolü ve farklı ekran boyutlarını birlikte değerlendir.

Yayınla ve ölç: Performans, hata oranı ve kullanıcı davranışını izleyerek kararları güncelle.

6. Fiziksel ve Dijital Mimarlık Karşılaştırma Matrisi

Aşağıdaki eşleşmeler bire bir teknik denklik değil, kararların rolünü açıklayan analogilerdir. Bu ayrım önemlidir: metafor, düşünmeyi kolaylaştırmalı; teknik gerçeğin yerini almamalıdır.

Fiziksel mimarlık	Frontend karşılığı	Ortak amaç	Benzetmenin sınırı
Kat planı / blueprint	Wireframe ve prototip	Uygulamadan önce yapıyı ve akışı tartışmak	Prototip, gerçek veri ve teknik kısıtları bütünüyle temsil etmez.
Taşıyıcı sistem	Bileşen sınırları, veri modeli ve hata yönetimi	Yükü ve sorumluluğu dengeli dağıtmak	Yazılım yapısı fiziksel yük taşımaz; değiştirilebilirliği çok daha yüksektir.
Prefabrik eleman	Yeniden kullanılabilir bileşen	Tekrarlanabilirlik ve standart üretim	Her bağlam için tek bileşen üretmek aşırı genelleştirmeye yol açabilir.
Malzeme kataloğu	Tasarım tokenları ve bileşen kütüphanesi	Kararları ortaklaştırmak ve tekrar kullanım sağlamak	Dijital değerler çok hızlı değişir; sürüm yönetimi gerekir.
İmar / uygulama standardı	Kod standartları, linter, erişilebilirlik kuralları	Asgari kalite ve uyumu korumak	ESLint veya Prettier hukuki yönetmelik değildir; ekip içi kalite araçlarıdır.
Tesisat ve dolaşım	Yönlendirme, veri akışı ve durum paylaşımı	Görünmeyen servisleri doğru noktalara ulaştırmak	Her state küresel olmamalıdır; yerel durum çoğu zaman daha güvenlidir.
Şantiye koordinasyonu	CI/CD, kod inceleme ve sürümleme	Farklı ekiplerin değişikliklerini kontrollü birleştirmek	Yazılım teslimi geri alınabilir ve çok daha sık tekrarlanabilir.
Kampüs yerleşimi	Mikro ön yüzler	Bağımsız birimleri ortak altyapıyla bir arada çalıştırmak	Küçük ekip ve ürünlerde ek karmaşıklık faydayı aşabilir.

7. Monolitten Mikro Ön Yüzlere Geçiş

Monolitik bir frontend, tüm ekranların aynı kod tabanında bulunması demektir. Bu yaklaşım kendiliğinden kötü değildir; tek ekipli veya orta ölçekli ürünlerde basitlik, ortak araç zinciri ve kolay yerel geliştirme sağlar. Sorun, ürün ve organizasyon büyüdükçe tek bir yayın sürecinin ekipleri birbirine aşırı bağımlı hâle getirmesiyle başlar.

Mikro ön yüz yaklaşımı, kullanıcı arayüzünün belirli iş alanlarını bağımsız ekiplerin yönettiği bölümlere ayırır. single-spa bu bölümleri farklı ekiplerce yönetilebilen arayüz parçaları olarak tanımlar [7]. Webpack Module Federation ise bağımsız derlemelerin çalışma zamanında modül sunmasına veya tüketmesine olanak verir [8]. Bu teknikler, mimarinin kendisi değil; seçilen mimariyi uygulamak için kullanılabilecek araçlardır.



7.1. Ne Zaman Anlamlıdır?

- Aynı ürün üzerinde çalışan birden fazla ekibin bağımsız yayın ihtiyacı varsa.
- İş alanları net sınırlarla ayrılabilir ve her alanın sahipliği belirlenebiliyorsa.
- Teknoloji göçü aşamalı yapılacaksa veya eski ve yeni arayüz bir süre birlikte yaşayacaksa.
- Ortak tasarım sistemi, gözleme ve güvenlik kurallarını yönetecek organizasyonel olgunluk varsa.

7.2. Hangi Maliyetleri Getirir?

- Birden fazla derleme ve yayın hattının işletilmesi.
- Ortak bağımlılıkların sürüm uyumsuzluğu ve gereksiz tekrar yüklenmesi.
- CSS çakışmaları, yönlendirme, kimlik doğrulama ve global hata yönetiminde ek koordinasyon.
- Uçtan uca test ve yerel geliştirme ortamının karmaşıklaşması.

Karar ilkesi

Mikro ön yüzler, “modern” görüldüğü için değil; ekip bağımsızlığı sorunu teknik ve organizasyonel olarak gerçekten mevcut olduğunda seçilmelidir.

8. Kısa Uygulama Örneği: E-Ticaret Ürün Sayfası

Bir ürün sayfasını dijital bir mağaza mekânı gibi düşünelim. Kullanıcı ürünü tanımak, seçenekleri karşılaştırmak, stok durumunu görmek ve sepete eklemek ister. Bu ekranı tek bir dev ProductPage bileşenine yığmak yerine görev ve davranış sınırlarına ayırmak daha sürdürülebilir bir yapı sağlar.

Bileşen	Sorumluluk	Veri sözleşmesi	Kritik durumlar
ProductGallery	Görsel seçimi ve yakınlaştırma	images[], activeImage	Eksik görsel, klavye gezinmesi, alternatif metin
ProductInfo	Başlık, fiyat ve kısa bilgi	title, price, discount	Fiyat değişimi, uzun başlık, para birimi
VariantSelector	Beden / renk seçimi	options[], selected, onChange	Stok dışı seçenek, seçim yapılmaması
AddToCart	Sepete ekleme eylemi	productId, quantity, disabled	Ağ gecikmesi, çift tıklama, başarısız istek
DeliveryEstimate	Teslimat bilgisini gösterme	location, stock, date	Konum bilinmiyor, veri gecikmesi

Bu planın değeri, ekranda beş kutu oluşturmaktan değil; her parçanın değişme nedenini ayrıştırmasından gelir. Örneğin teslimat hesaplama kuralı değiştiğinde görsel galeri etkilenmez. Sepete ekleme davranışı test edilirken varyant seçimi bağımsız olarak taklit edilebilir. Tasarım sistemi ise Button, Select, Price ve Alert gibi daha küçük yapı taşlarını bu iş alanı bileşenlerine sağlar.

8.1. Kalite Kontrol Listesi

- Ürün sayfası JavaScript yüklenirken anlamlı bir iskelet veya yüklenme durumu gösteriyor mu?
- Sepete ekleme butonu klavyeyle çalışıyor ve işlem sonucunu anlaşılır biçimde bildiriyor mu?
- Ürün görsellerinde anlamlı alternatif metinler var mı?
- Fiyat ve indirim bilgisi yalnız renk farkıyla mı anlatılıyor?
- Yavaş ağda kullanıcı aynı ürünü yanlışlıkla birden çok kez sepete ekleyebiliyor mu?
- Mobil ekranda eylem düğmesi içerik veya tarayıcı arayüzü tarafından örtülüyor mu?

9. Analoginin Sınırları

Mimarlık benzetmesi güçlüdür; fakat sınırsız kullanıldığında yanıltıcı olabilir. Fiziksel yapılarda değişiklik çoğu zaman pahalı, yavaş ve geri döndürülmesi zordur. Yazılım ise kopyalanabilir, sürümlenebilir ve kontrollü biçimde geri alınabilir. Bir bina kullanıma açıldıktan sonra temelini her hafta değiştirmek mantıklı değildir; yazılım sistemleri ise doğru test ve sürümleme ile sürekli evrilebilir.

İkinci fark, yazılımın davranış üretmesidir. Bir arayüz yalnız statik mekân düzenlemez; veri alır, karar verir, bekler, hata verir ve kullanıcı girdisine tepki gösterir. Bu nedenle frontend mimarisi, fiziksel mekân analogisinin yanında durum makineleri, veri akışları, ağ güvenilirliği ve performans gibi yazılıma özgü kavramlarla ele alınmalıdır.

Üçüncü fark organizasyoneldir. Mikro ön yüzleri kampüs binalarına benzetmek anlaşılırdır; fakat ekip sınırları yanlış tanımlanmışsa teknik ayrıştırma iletişim sorununu çözmez. Yazılım mimarisi, organizasyonun karar verme ve sahiplik yapısından bağımsız düşünülemez.

Analojiyi doğru kullanmak

Benzetme bir kararı açıklıyorsa yararlıdır. Kararın teknik gerekçesinin yerini alıyorsa yetersizdir.

Sonuç: Geleceğin Dijital Zanaatkârları

Fiziksel mimarlık ile frontend geliştirme aynı disiplin değildir; ancak ikisi de karmaşıklığı sınırlar, ilişkiler ve ortak kurallar üzerinden yönetir. Vitruvius'un sağlamlık, kullanışlılık ve estetik dengesi; modern arayüzlerde kod dayanıklılığı, erişilebilirlik ve görsel bütünlük olarak yeniden okunabilir. Atomic Design küçük parçaların sistematik biçimde birleşmesini, tasarım sistemleri ortak kararların sürdürülebilirliğini, mikro ön yüzler ise büyük organizasyonlarda ekip sınırlarının teknik yapıya yansımaları açıklar.

Başarılı bir frontend geliştirici yalnız ekrana piksel yerleştiren kişi değildir. Kullanıcının yolunu, değişikliğin etkisini ve ekiplerin birlikte çalışma biçimini tasarlar. Bu bakış açısıyla kod; tek tek dosyalardan oluşan bir yığın değil, sürekli yaşayan ve bakım gören bir dijital çevre hâline gelir. Geleceğin dijital zanaatkârı, estetiği teknik dayanıklılıktan; hızı erişilebilirlikten; modülerliği de bağlamdan ayrı düşünmeyen geliştiricidir.

Kaynakça

- [1] University of Chicago Library, "Firmness, Commodity, and Delight." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [2] Microsoft, "TypeScript Handbook: The Basics." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [3] W3C, "Using ARIA - First Rule of ARIA Use." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [4] Brad Frost, "Atomic Design Methodology." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [5] React, "Passing Props to a Component." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [6] MDN Web Docs, "HTML: A Good Basis for Accessibility." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [7] single-spa, "Microfrontends Overview." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).
- [8] webpack, "Module Federation." [Çevrim içi kaynak](#) (Erişim: 12 Temmuz 2026).