T.C.
BEYKENT ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

# WEB UYGULAMALARI İÇİN YENİLİKÇİ YAKLAŞIMLARI KULLANAN UYGULAMA ÇATISI

Yüksek Lisans Tezi

Tezi Hazırlayan :

**Güner Kaan ALKIM**

İstanbul, 2020

T.C.
BEYKENT ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

# WEB UYGULAMALARI İÇİN YENİLİKÇİ YAKLAŞIMLARI KULLANAN UYGULAMA ÇATISI

Yüksek Lisans Tezi

Tezi Hazırlayan :

**Güner Kaan ALKIM**

Öğrenci No:

17080200023

Danışman:

Dr. Öğr. Üyesi Ediz ŞAYKOL

İstanbul, 2020

## YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum "Web Uygulamaları İçin Yenilikçi Yaklaşımları Kullanan Uygulama Çatısı" başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 03/02/2020

Aday : **Güner Kaan ALKIM**

**T.C.**
**BEYKENT ÜNİVERSİTESİ**
**LİSANÜSTÜ EĞİTİM ENSTİTÜSÜ MÜDÜRLÜĞÜ**
**TEZLİ YÜKSEK LİSANS SINAV TUTANAĞI**

3 / 2 / 2020

Enstitümüz ***Bilgisayar Mühendisliği*** Anabilim Dalı ***Bilgisayar Mühendisliği*** Programı yüksek lisans öğrencilerinden ***17080200023*** numaralı ***Güner Kaan ALKIM***'ın *"Beykent Üniversitesi Lisansüstü Eğitim – Öğretim Yönetmeliği"*nin ilgili maddesine göre hazırlayarak, Enstitümüze teslim ettiği ***"Web Uygulamaları İçin Yenilikçi Yaklaşımları Kullanan Uygulama Çatısı"*** konulu tezini, Yönetim Kurulumuzun 28/01/2020 tarih ve 2020/04 sayılı toplantısında seçilen ve Taksim Yerleşkesinde toplanan biz jüri üyeleri huzurunda, Beykent Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 29. maddesinin 3. fıkrası gereğince (45) dakika süre ile aday tarafından savunulmuş ve sonuçta adayın tezi hakkında ~~***oyçokluğu/***~~***oybirliği*** ile ***Kabul/~~Red~~ veya ~~Düzeltme~~*** kararı verilmiştir.

İşbu tutanak, 4 nüsha olarak hazırlanmış ve Enstitü Müdürlüğü'ne sunulmak üzere tarafımızdan düzenlenmiştir.

**DANIŞMAN**
Dr. Öğr. Üyesi Ediz ŞAYKOL
**(Beykent Üniversitesi)**

**ÜYE**
Dr. Öğr. Üyesi Atınç YILMAZ
**(Beykent Üniversitesi)**

**ÜYE**
Dr. Öğr. Üyesi Ömer ÇETİN
**(Milli Savunma Üniversitesi)**

Adı ve Soyadı : Güner Kaan ALKIM
Danışmanı : Dr. Öğr. Üyesi Ediz ŞAYKOL
Türü ve Tarihi : Yüksek Lisans, 2020
Alanı : Bilgisayar Mühendisliği
Anahtar Kelimeler : Virtual DOM, React, Frontend Development, State Management

## ÖZ

## WEB UYGULAMALARI İÇİN YENİLİKÇİ YAKLAŞIMLARI KULLANAN UYGULAMA ÇATISI

Bileşen tabanlı web uygulamaları son yıllarda yazılım dünyasında hızla kendine yer edilmeye başlamıştır. Öyle ki dünyaca ünlü yazılım şirketleri(Facebook, Google, Ebay vs…) bu konsepti uygulayan kütüphane ya da uygulama çatılarını piyasa sürmüşler ve kendi uygulamalarını bu tabanda geliştirmişlerdir. Günümüzde yeni başlayan ve devam eden çoğu proje bileşen tabanında geliştirilmektedir. Tezin amacı, web uygulamalarının geliştirilmesinde bileşen adı verilen bu yapıların uygulama geliştirme süreçlerine etkilerini incelemek ve gelecekteki gidişatına bir projeksiyon tutmaktır. Bu konuda, birden fazla test aşaması oluşturulmuştur. Bununla birlikte tez konusunun birebir uygulandığı bir kullanıcı ara yüz kütüphanesi olan Anatolia geliştirilmiştir. Bileşen yapısı, eklenti ve pure js tekniklerine nazaran hem geliştirme süresinde, hem de bileşenin içerisinde kullanmış olduğu algoritmalar bağlamında performansa ve yüksek etki gözlemlenmiştir. Bileşen yapısının geliştirme süreçlerinden kullanılması, uygulama geliştirilmesini ve bakımının kolaylaştırıldığını göstermiştir. Pure Javascript‘ e nazaran daha anlaşılır bir kod yapısı sunduğu, tekrar kullanım olasılığını yükselttiği ve bu sayede yazılan kodun daha işlevsel hale geldiği gözlemlenmiştir. Bileşen tabanlı geliştirme yaklaşımı, ön yüz geliştirme dünyasında kodun nasıl yapılandırılması gerektiği algısını değiştirmiştir. Bu teknik aynı zamanda birden fazla platform üzerinde de uygulanabilir olmuş ve Javascript topluluğu tarafından kabul edilmiştir.

Name and Surname : Güner Kaan ALKIM
Thesis Advisor : Dr. Lecturer Ediz ŞAYKOL
Thesis Type ve Date : Yüksek Lisans, 2020
Study Field : Bilgisayar Mühendisliği
Keywords : Virtual DOM, React, Frontend Development, State Management

## ABSTRACT

## APPLICATION FRAMEWORK USING INNOVATIVE APPROACHES FOR WEB APPLICATIONS

In recent years, component-based web applications have quickly become a part of the software world. So much so that the world-renowned software companies (Facebook, Google, Ebay, etc…) have applied the concept of the library or application roofs market and developed their own applications on this basis. Nowadays, most of the new and ongoing projects are developed on the component base. The aim of the thesis is to examine the effects of these structures, which are called components, in the development of web applications on application development processes and to keep a projection on their future course. In this regard, more than one test phase has been established. However, Anatolia, a user interface library where the thesis topic is applied one by one, has been developed. Compared to the component structure, plug-in and pure js techniques, performance and high impact were observed both in development time and in the context of the algorithms used within the component. The use of component structure from development processes has shown that application development and maintenance are facilitated. It has been observed that it offers a more understandable code structure compared to Pure Javascript, increases the likelihood of reuse, and thus makes the written code more functional. The component-based development approach has changed the perception of how code should be structured in the frontend development world. This technique has also been implemented on multiple platforms and has been accepted by the Javascript community.

## İÇİNDEKİLER

**Sayfa No.**

**ÖZ** ..... **i**
**ABSTRACT** ..... **ii**
**TABLOLAR LİSTESİ** ..... **v**
**ŞEKİLLER LİSTESİ** ..... **vi**
**KISALTMALAR** ..... **viii**
**GİRİŞ** ..... **1**

**BİRİNCİ BÖLÜM**
**ÖN YÜZ GELİŞTİRME METODOLOJİLERİ**

**1. JAVASCRIPT MV* TASARIM ŞABLONLARI** ..... **5**
1.1. MVC ..... 6
1.2. MVP ..... 8
1.3. MVVM ..... 9
**2. BİLEŞEN TABANLI GELİŞTİRME** ..... **10**
2.1. Yaşam Döngüsü ..... 11
2.2. Render Süreci ..... 11
2.3. Veri Yönetimi ..... 12
2.4. Yeniden Kullanılabilirlik ..... 12
**3. DECLARATIVE / IMPERATIVE PROGRAMLAMA** ..... **13**
**4. FONKSİYONEL PROGRAMLAMA** ..... **14**

**İKİNCİ BÖLÜM**
**BİLEŞEN VE VERİ İLİŞKİSİ**

**1. BİLEŞEN VE VERİ BAĞLAMI (DATA BINDING)** ..... **15**
1.1. Tek Yönlü Veri Bağlama (One Way Data Bındıng) ..... 15
1.2. İki Yönlü Veri Bağlama (Two Way Data Bındıng) ..... 16
**2. VERİ YÖNETİMİ (STATE MANAGEMENT)** ..... **16**
**3. JAVASCRIP 'İN DERLENMESİ VE HTML OLUŞTURULMASI** ..... **17**
3.1. Virtual DOM ..... 17
3.2. Memoization ..... 18

## ÜÇÜNCÜ BÖLÜM

## ANATOLIA JAVASCRIPT KÜTÜPHANESİ

**1. ANATOLIA 'NIN BİLEŞENLERE YAKLAŞIMI ..... 20**

1.1. DOM Oluşturma Stratejisi ..... 21

**2. PUBLISHER / SUBSCRIBER İLE VERİ YÖNETİMİ ..... 24**

**3. TEKRAR KULLANILABİLİRLİK ..... 28**

## DÖRDÜNCÜ BÖLÜM

## ARAŞTIRMA METOTLARI

**1. KULLANILAN METOTLARI ..... 30**

**2. PERFORMANS TESTLERİ ..... 30**

2.1. Geçerlilik ..... 30

2.2. Uygulama ..... 31

2.3. Test Durumları ..... 31

2.4. Testlerin Uygulanması ..... 31

2.5. Test Araçları ..... 32

## BEŞİNCİ BÖLÜM

## PERFORMANS TEST SONUÇLARI

**1. NORMAL DURUM MOBİL CİHAZ PERFORMANS TESTİ ..... 33**

**2. YAVAŞLATILMIŞ DURUM MOBİL CİHAZ PERFORMANS TESTİ ..... 34**

**3. NORMAL DURUM MASAÜSTÜ CİHAZ PERFORMANS TESTİ ..... 35**

**4. YAVAŞLATILMIŞ DURUM MASAÜSTÜ CİHAZ PERFORMANS TESTİ ..... 36**

**5. UYGULAMA DEĞİŞİKLİKLERİNİN PERFORMANS TESTİ ..... 37**

**6. BENCHMARK.JS PERFORMANS TEST SONUÇLARI ..... 38**

**SONUÇ ..... 39**

**KAYNAKÇA ..... 41**

**EKLER ..... 42**

EK 1 - Benchmark.js Test Kodu ..... 42

# TABLOLAR LİSTESİ

**Sayfa No.**

**Tablo 1** Normal Durum Mobil Cihaz Performans Test Bilgileri ..... 33
**Tablo 2** Yavaşlatılmış Durum Mobil Cihaz Performans Test Bilgileri ..... 34
**Tablo 3** Yavaşlatılmış Durum Masaüstü Cihaz Performans Test Bilgileri..... 36

# ŞEKİLLER LİSTESİ

**Sayfa No.**

**Şekil 1** Javascript Dilinin 2019 Yılı İçerisindeki Google Popülaritesi ..... 2
**Şekil 2** StackOverFlow.com Sitesinde Javascript Etiketine Sahip Sorular ..... 3
**Şekil 3** StackOverFlow.com Sitesinin 2019 Geliştirici Anket Sonucu ..... 4
**Şekil 4** MVC – MVP – MVVM ..... 5
**Şekil 5** MVC – Model Nesnesi ..... 7
**Şekil 6** MVC – View Nesnesi ..... 8
**Şekil 7** MVC – Controller Nesnesi ..... 8
**Şekil 8** Declarative SQL Betiği ..... 13
**Şekil 9** Saf ve Saf Olmayan Fonksiyonlar ..... 14
**Şekil 10** Virtual DOM Çalışma Prensibi ..... 18
**Şekil 11** Anatolia ‘da Bileşen Yapısı ..... 20
**Şekil 12** Anatolia ‘da Bileşen Kullanımı ..... 21
**Şekil 13** State Değişikliklerini Dinleyen Metot 1 ..... 22
**Şekil 14** State Değişikliklerini Dinleyen Metot 2 ..... 22
**Şekil 15** Big O Notasyonu Kompleksite Çizelgesi ..... 23
**Şekil 16** Observer Tasarım Şablonu ..... 24
**Şekil 17** Pub/Sub Tasarım Şablonu ..... 25
**Şekil 18** Anatolia Publisher Uygulaması ..... 26
**Şekil 19** Publisher Kaydından Sonra Eventbus‘ ın Son Durumu ..... 26
**Şekil 20** Anatolia Subscriber Uygulaması ..... 26
**Şekil 21** Subscriber Kaydından Sonra Eventbus‘ ın Son Durumu ..... 27
**Şekil 22** Anatolia ‘da Bileşen Şablonları ..... 28
**Şekil 23** Anatolia ‘da Yeniden Kullanılabilirlik ..... 28
**Şekil 24** Anatolia ‘da Üretilen Bileşenler ..... 29
**Şekil 25** Anatolia Normal Durum Testi ..... 33
**Şekil 26** React Normal Durum Testi ..... 34
**Şekil 27** Anatolia Yavaşlatılmış Durum Testi ..... 35
**Şekil 28** React Yavaşlatılmış Durum Testi ..... 35
**Şekil 29** Anatolia Yavaşlatılmış Durum Testi ..... 36

**Şekil 30** React Yavaşlatılmış Durum Testi..... 36
**Şekil 31** Anatolia Yürütme Süreci..... 37
**Şekil 32** React Yürütme Süreci ..... 37
**Şekil 33** Benchmark.js Performans Testi Sonuçları ..... 38

## KISALTMALAR

| | | |
|---|---|---|
| **API** | **:** | Application Programming Interface |
| **DOM** | **:** | Document Object Model |
| **GOF** | **:** | Gang Of Four |
| **HTML** | **:** | Hypertext Markup Language |
| **MVC** | **:** | Model – View – Controller |
| **MVP** | **:** | Model – View – Presenter |
| **MVVM** | **:** | Model – View – ViewModel |
| **PUB / SUB** | **:** | Publisher / Subscriber |
| **PWA** | **:** | Progressive Web Apps |
| **SOC** | **:** | Seperation of Concern |
| **VDOM** | **:** | Virtual DOM |
| **WS** | **:** | Web Socket |
| **XHR** | **:** | XMLHttpRequest |

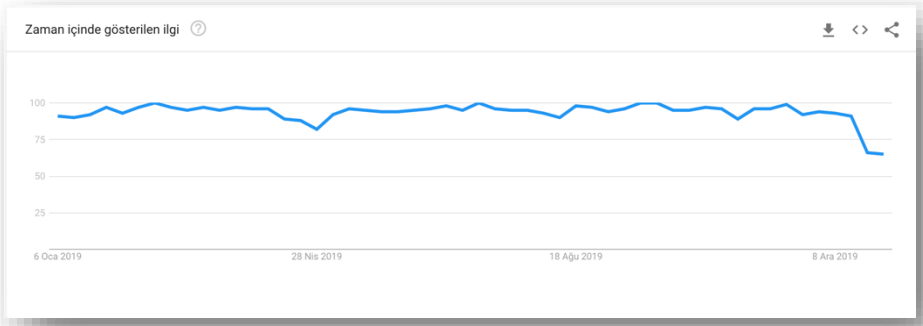
# GİRİŞ

90'ların başından günümüze kadar web sayfaları dramatik değişikliklere uğramıştır. Önceleri statik ve etkileşimden uzak olan web sayfaları, artan son kullanıcı gereksinimlerinde dolayı yıllar içinde hızla gelişmiş ve günümüzde web uygulaması olarak tanımlanır olmuştur. Web sayfalarının ilk hizmet amacı yalnızca bilimsel dokümanların çevrimiçi paylaşımını sağlamakken, günümüzde ise finans sektöründen, gıda sektörüne; iletişimden, hastane bilgi yönetim sistemlerine varana kadar hizmet sağlamaktır.

Web sayfalarının gereksinimlerinin ve sağladıklarının yanında, web geliştirme de bu hızlı değişiklikten oldukça etkilenmiştir. Web siteleri ilk zamanlarda, yalnızca sunucu tarafında yapılan değişiklikler ile güncellenebiliyorlardı. Bu durum zamanla, web tarayıcılarının yeteneklerinin artması, son kullanıcı ve sektörel gelişmeler, web sayfalarının doğrudan internet tarayıcısı içerisinden değiştirilmesini sağlamıştır. Bu değişiklikleri ve yeni yetenekleri kazandırabilmek içinse yeni bir dil geliştirilmiştir, Javascript.

Javascript istemci tarafında çalışan, yorumlamalı bir dil olarak, Netscape firması için 1995 Mayıs ayında yalnızca on gün içerisinde Brendan Eich tarafından geliştirilen bir dildir. Dil, ilk tasarlandığında adı Mocha idi. Daha sonra Livescript adını alan bu dil, nihai olarak Javascript olarak anılmaktadır.

Javascript bugün esnek geliştirme seçeneklerinden dolayı tarayıcılarda, sunucu tarafında, mobil uygulamalarda ve hatta gömülü devre programlamada kullanılmaktadır.




**Şekil 1** Javascript Dilinin 2019 Yılı İçerisindeki Google Popülaritesi

Artan donanım gücü ve internetin artık taşınabilir noktaya gelmesi ise web sitelerini ve web uygulamalarını doğrudan mobil cihazlarda görüntülememizi sağlamıştır. Bununla birlikte web uygulamaları; lisans, dağıtım, güncelleme kolaylığı gibi nedenler ile masaüstü uygulamalara tercih edilir olmuştur. Son yıllarda ise web uygulamaları artık PWA( Progressive Web App ) denilen bir yaklaşım ile geliştirilmektedirler. Bu yaklaşımda, kullanılan teknoloji ve yazılım standartları web uygulamasını çevrimdışı da erişilebilir yaparak sanki bir native mobil uygulama kullanıyormuş hissiyatı yaratmaktadır.

Javascript ‘in hızla artan kütüphaneleri, onu doğrudan mobil uygulama geliştirmede de kullanmamızı sağlamıştır. Ör; Javascript ile geliştirilen bir web uygulaması, PhoneGap ya da Cordova kütüphaneleri yardımıyla doğrudan bir mobil uygulama haline getirilip hızlıca yayına alınabilmektedir. Bununla birlikte ReactNative ya da Weex kullanarak ise doğrudan platforma bağımlı ve daha performanslı mobil uygulamaları Javascript ile geliştirmemiz mümkündür.

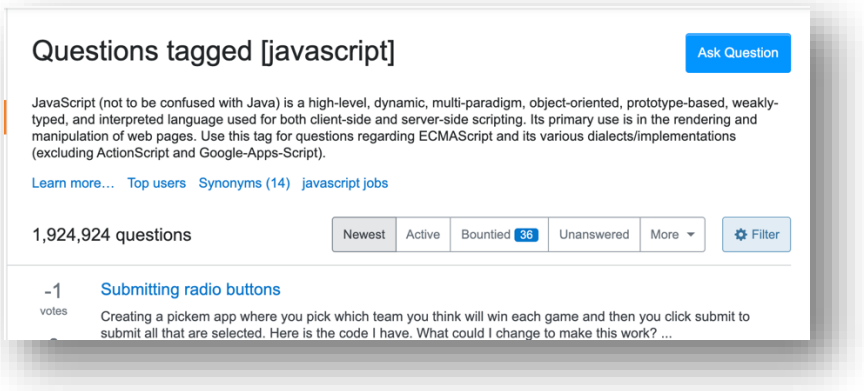
Hızla geçen bu süreç, programcıların kodlarının bakım ve yönetiminin daha kolay ve optimistik olmasını sağlaması açısından, tekrar kullanılabilir ve geliştirici dostu bir yazılım paradigması olan Bileşen(Component) tabanlı geliştirmeyi doğurmuştur. Bileşen tabanlı geliştirmede, kendi başına bağlamı ayrı olmak üzere iş yapan tüm alanların geliştirmesini birbirinden izole ve çeşitli veri yolları ile iletişimde olacak şekilde geliştirilmektedir. Bu bağlamda, tasarlanan bir bileşen, birden fazla üst

bileşen tarafından da kullanılabilir ya da farklı bileşenlerle ortak çalışabilir olmaktadır. Bileşenler kendi kod yapılarına, kendi metotlarına ve kendi API ‘larına sahiptirler. Bileşenlerin birbirlerinden izole olan yapısı, aynı lego parçaları gibi, bir çok farklı hareketli yapıya sahip bir kullanıcı ara yüzü oluşturmalarına olanak sağlar.

Bileşenler, istemci tarafından tetiklenen istekleri, diğer bileşenlerden bağımsız olarak XHR, web socket vb bir teknoloji yardımıyla sunucu tarafına iletir. Her bir bileşenin kendine özgü bir ara yüzü olmasından dolayı doğrudan tüm kullanıcı ara yüzünü bütün olarak etkilemeden görünüm katmanını güncelleyebilirler.

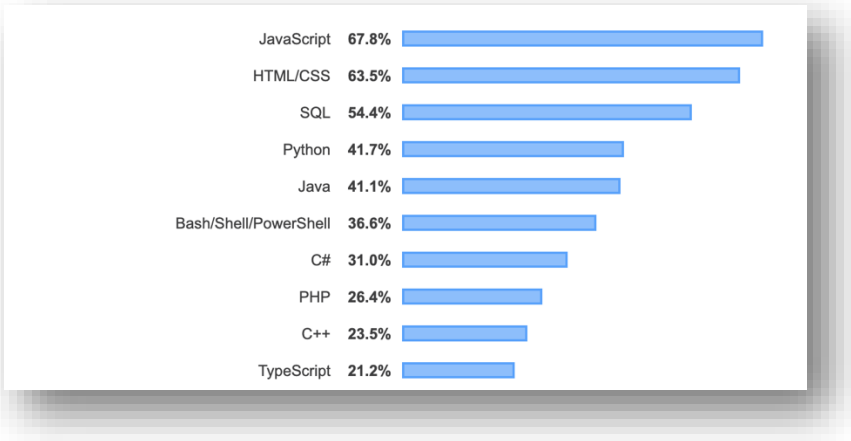
Bileşen tabanlı uygulamalar, bir bileşene ait tüm metot ve API ‘ların o bileşen içerisinde olmasını gerektirir. Örneğin; MVC yapısında metotlar üç farklı konsept içerisinde tanımlanırken, bileşenlerde bu özelliklerin tümü tek sınıfta bulunur.

Yazılım geliştiriciler arasında en popüler site olan StackOverFlow.com‘ da 2020 Ocak ayı itibariyle Javascript etiketine sahip iki milyona yakın soru bulunmaktadır.




**Şekil 2** StackOverFlow.com Sitesinde Javascript Etiketine Sahip Sorular

Bununla birlikte yine StackOverFlow.com adlı site dünya genelinde yazılım geliştiricilere yönelik yapmış olduğu ankette Javascript dili en popüler dil olarak seçilmiştir.




**Şekil 3** StackOverFlow.com Sitesinin 2019 Geliştirici Anket Sonucu

Tezin amacı, ön yüz geliştirme metodolojilerinden bileşen tabanlı geliştirme yaklaşımını yeni bir kullanıcı arayüzü kütüphanesi ile uygulamak, diğer kütüphanelere göre hız bakımından daha performanslı bir kütüphane oluşturmak ve bileşen tabanlı geliştirmenin ön yüz geliştirme performansına olan etkilerini göstermektir. Bu bağlamda, Anatolia adında bir kullanıcı arayüzü kütüphanesi geliştirilmiştir. Bu kütüphane ile, bileşen yaklaşımının faydaları uygulanarak, bileşen yaklaşımını uygulayan diğer kütüphanelerden hız bakımından daha performanslı sonuçlar alınması hedeflenmiştir.

Tez bölümleri oluşturulurken ön yüz geliştirme teknolojilerinin neler olduğu, hangi gereksinimlere göre geliştirme yapıldığı, geliştirme metodolojilerinin nereye doğru evrildiği, tez kapsamında geliştirilen kütüphanenin püf noktaları ve performans test sonuçları göz önüne alınmıştır.

# BİRİNCİ BÖLÜM

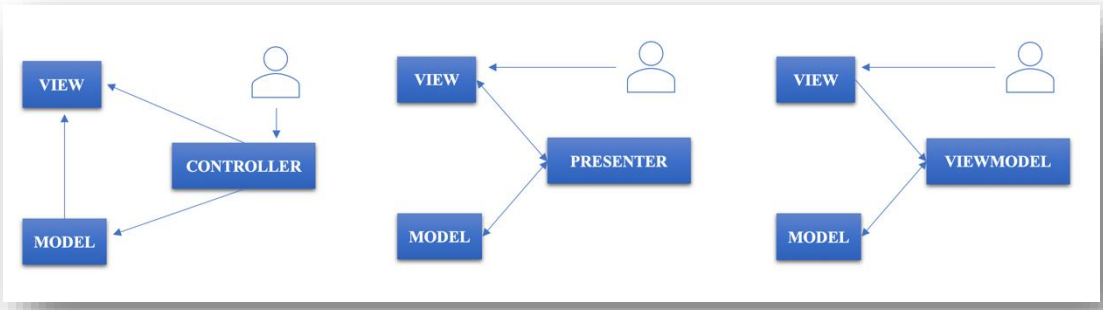
# ÖN YÜZ GELİŞTİRME METODOLOJİLERİ

## 1. JAVASCRIPT MV* TASARIM ŞABLONLARI

Tasarım şablonları, bir yazılım projesinde karşılaşılan temel ve ortak problemlerin çözümü noktasında geliştirilen tekniklerdir. Yazılım esnasında tekrar eden sorunları çözmek için kullanılan ve tekrar kullanılabilir tipte kod yazılımını destekleyen bir yada birden fazla sınıftan oluşmuş modül ve program parçalarına **tasarım şablonu** denir (Acar, 2012, 29).

Tasarım şablonları bir yazılım projesinin tüm yapısını / kurgusunu değiştirebileceği gibi, proje esnasında karşılaşılan bir problemin çözümü için de kullanılabilirler. Şablonlar, her dilin kendi diyalektiği içerisinde farklı şekilde uygulanabilirler.

MV* adı altında toplanan MVC, MVP ve MVVM desenleri, projelerin mimarisine etki ettikleri için, mimari tasarım şablonları olarak adlandırılırlar. Bu tasarım şablonu; veri, verinin elde ediliş ya da işleniş biçimi ve verinin nasıl biçimleneceği konuları üzerinde dururlar.




**Şekil 4** MVC – MVP – MVVM

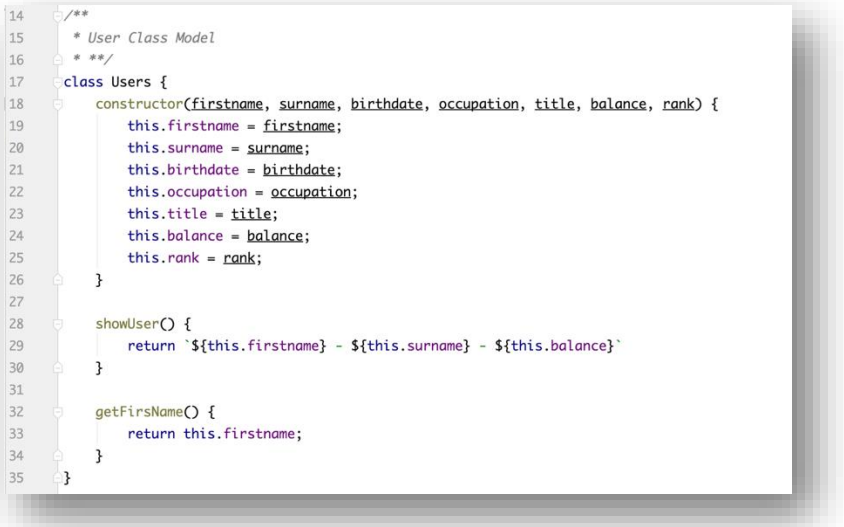
## 1.1. MVC

MVC, yazılım mühendisliğinde **SOC** adlı prensibi uygulayarak gelişmiş uygulama mimarisi öneren bir tasarım desenidir. Bu desen gereği; veri işlemleri, arayüz işlemleri ve iş mantığı işlemleri birbirinden ayrı katmanlarda ele alınmakta ve uygulanmaktadır.

İlk tasarlandığında Model-View-Controller-Editor olarak isimlendirilen bu desen, Smalltalk-80(1979) üzerinde çalışan Trygve Reenskaug tarafından ortaya atılmıştır. MVC, 1995‘ te “Design Patterns: Elements of Reusable Object-Oriented Software”(GoF) adlı kitapta derinlemesine anlatılmış ve kullanımının popülerleşmesinde bu kitap rol oynamıştır (Addy Osmani., **Learning Javascript Design Patterns**, https://addyosmani.com/resources/essentialjsdesignpatterns/book/, 18/01/2020).

MVC, üç ana bileşenden oluşmakta ve bir web uygulamasında aşağıda açıklanan kavramlara değinmektedir.

**Model :** Modeller, uygulamanın veri katmanını yönetmek için tasarlanmışlardır. Model katmanı, kullanıcı ara yüzü katmanı ile doğrudan herhangi bir etkileşim kurmamaktadır. Modeller üzerindeki değişikliklerin kullanıcı ara yüzüne yansıtılması, model katmanını gözlemleyen başka arabirimler tarafından gerçekleştirilmektedir. Model katmanının daha iyi anlaşılabilmesi için, bir web uygulamasında Javascript dili ile hazırlanmış aşağıdaki örneği inceleyebiliriz.



```
/**
 * User Class Model
 * **/
class Users {
    constructor(firstname, surname, birthdate, occupation, title, balance, rank) {
        this.firstname = firstname;
        this.surname = surname;
        this.birthdate = birthdate;
        this.occupation = occupation;
        this.title = title;
        this.balance = balance;
        this.rank = rank;
    }

    showUser() {
        return `${this.firstname} - ${this.surname} - ${this.balance}`
    }

    getFirsName() {
        return this.firstname;
    }
}
```

**Şekil 5** MVC – Model Nesnesi

Model nesneleri bir veritabanında oluşturulup tutulabileceği gibi, doğrudan istemci tarafında localStorage gibi tarayıcı belleğinde oluşturulup tutulabilir.

Çeşitli Javascript kütüphaneleri model nesneleri üzerinde çalışma yapabilmek için bir takım araç setleri sağlarlar. Bunlar model nesnelerinin doğrulanması (e-posta'nın regex ile kontrolü vs.) gibi durumlarda kullanılabilirler. Model nesneleri kullanıcı ara yüzü katmanından tamamen ayrı ele alındığı için model üzerindeki değişikliklerin de gözlemlenmesi gerekmektedir. Bu nedenle, **observer(gözlemci)** yada **publisher/subscriber(yayıncı/abone)** tasarım kalıpları kullanılarak hazırlanmış kütüphaneler yardımıyla model değişikliklerini dinleyebilmekteyiz. Bir modelin tamamı ya da içerisindeki herhangi bir özellik değiştiğinde bundan haberdar olup, modele bağlı işlemlerimizi gerçekleştirebiliriz. Nihayetinde, model nesneleri uygulamanın veri katmanı ile ilgilenmektedir.

**View:** Model katmanında tanımlanan verilerin görsel temsilini oluşturan bölüm view(görünüm/kullanıcı arayüzü) katmanında ele alınmakmadır. Görünüm aslında model katmanının tamamının ya da bir bölümünün filtrelenerek kullanıcı arayüzüne bağlanmasını sağlamaktadır. Bu bölümde kullanıcı, görselleştirme yardımıyla veri ile iletişime girer ve model verisine göre hareket eden iş mantığı ortaya çıkar (Yener, Theedom, 2015, 184).

View katmanının tam olarak nasıl çalıştığını anlayabilmek için bir web uygulamasında Javascript dili ile hazırlanmış aşağıdaki örneği inceleyebiliriz.

```
import {newUser} from '../model/Users';
import {lengthController} from '../controller/UserInfoController';

const UserInfoComponent = () => {
    let userName = document.createElement( tagName: "p");

    try {
        userName.textContent = lengthController(newUser.getFirsName());
    } catch (e) {
        alert(e.message);
    }

    return (
        userName
    )
};

export default UserInfoComponent;
```

**Şekil 6** MVC – View Nesnesi

**Controller :** Bu katmanda, view ve model katmanları arasında aracılık yapan ve gerekli iş mantığı kodlarının bulunur. Kullanıcı ara yüzünden aldığı verileri gerekli iş mantığı kodlarını işlettikten sonra modele, modelden aldığı verileri ise yine iş mantığı kodlarını işlettikten sonra ise view katmanına aktarmaktadır.

Controller katmanının tam olarak nasıl çalıştığını anlayabilmek için bir web uygulamasında Javascript dili ile hazırlanmış aşağıdaki örneği inceleyebiliriz.

```
const MAX_PROPERTY_LENGTH = 100;
const MIN_PROPERTY_LENGTH = 1;

export const lengthController = (property) => {
    let propertLength = property.length;

    if (propertLength > MIN_PROPERTY_LENGTH && propertLength < MAX_PROPERTY_LENGTH) {
        return property;
    }

    throw "Property length should be between MAX PROPERTY LENGTH && MIN_PROPERTY LENGTH";
};
```

**Şekil 7** MVC – Controller Nesnesi

## 1.2. MVP

MVP tasarım deseni de yine **SOC** prensibine dayalı olarak, view katmanı ile model katmanı arasındaki doğrudan ilişki kurmamak için tasarlanmıştır ve MVC

tasarım kalıbından türemiştir. Bu tasarım deseninde, kullanıcı ara yüzü ile iletişime giren kullanıcı doğrudan controller' lar ile iletişim kurmak yerine view katmanı ile iletişim kurmaktadır.

View katmanı kullanıcıdan almış olduğu aktiviteleri presenter katmanı üzerinden modele iletir. View 'da yapılacak bir değişiklik, presenter 'daki bir metodu tetikler ve model üzerinde güncelleme yapar. Modelde yapılan bu değişikliğin view katmanına yansıtılması ise yine presenter üzerindeki başka bir metot üzerinden gerçekleşir. Bu mimari desen sayesinde özellikle birim testlerin yazılması kolaylaşmaktadır (Ankit Sinhal, *MVC, MVP, MVVM Design Pattern*, https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad , 18/01/2020).

### 1.3. MVVM

MVVM modeli, görünüm ve görünüm modeli arasında ki yönlü data bağlanmasını(two-way data binding) sağlamak için tasarlanmıştır. Görünüm modeli, model üzerinde yapılan değişiklikleri herhangi bir controller yada presenter katmanına ihtiyaç duymadan otomatik olarak yansıtmaktadır. Görünüm modeli, view katmanındaki değişiklikleri model katmanına yansıtabilmek için observer(gözlemci) tasarım desenini kullanmaktadır.

Görünüm modeli(ViewModel) katmanı, view 'ın durumunu korumaya, modeli görünümdeki eylemlerin sonucu olarak manipüle etmeye ve görünümün içindeki olayları tetiklemeye yardımcı olan yöntemleri, komutları ve diğer özellikleri göstermekten sorumludur. View katmanının, ViewModel katmanına erişimi vardır fakat, ViewModel katmanı view hakkında herhangi bir referansa ya da bilgiye sahip değildir. View ve ViewModel arasındaki iki yönlü veri bağlantısı, ViewModel 'daki verilerin View ile anlık senkronize olmasını sağlamaktadır.

MVVM tasarım deseni, özellikle çift yönlü veri bağlama özelliğine ihtiyaç duyan uygulamalar için çok uygun bir seçimdir.

## 2. BİLEŞEN TABANLI GELİŞTİRME

Angular ‘ın JavaScript uygulamalarına en büyük katkısı muhtemelen her iş parçacığının ayrı bir namespace içerisinde yazılmasını getirmek olmuştur diyebiliriz. Angular bize gösterdi ki, kod bloklarını yapılandırırken önceden biraz efor sarf ederek, birbiriyle izole olarak çalışabilecek kod parçaların directive denilen özel alt parçalara ayrılması oldukça efektif bir yaklaşımdı.

Bugün bir çok Javascript uygulama çatısı birbirinden izole ve yeniden kullanılabilir kod blokları geliştirmek için çeşitli çözümler sunmaktadır. Bu yazının konusu, bugün React ve Vue gibi bir çok projede kullanılan kütüphanelerde bulunan Component yapısıdır.

Uzun zamandır geliştirmekte olduğumuz Javascript metotlarımız çoğunlukla bir View ‘ın render etmeye ya da render edilmesinde gerekli işlemleri yapmaya yaramaktalar. Bu render işleminde çoğunlukla herhangi bir kaynaktan gelen veriyi işlemektedirler.

Önceleri, geliştirdiğimiz Javascript metotları parametre aldığı veri nesnesi üzerinde gerekli işlemleri yaparak akışını tamamlarlardı. Almış olduğu parametre tekrar kullanılmayacaksa, veri nesnesi metodun bağlamından çıktıktan sonra yok edilirdi. Örneğin bir DOM parçası render ediyorsak, metot her çağırıldığında bu parça yeni baştan inşa edilirdi. Zaman ilerledikçe, metodun iş yapmak için veri nesnesine tam bağımlılık sergilediği, veri üzerinde çeşitli manipülasyonlar yapabilmek için bu verinin saklanması gerektiği sonucu ortaya çıktı.

İşte uzun zamandır her yerde karşımıza çıkan state(veri durumu) kavramı, bir Javascript metodunun veri bağlamından başka bir şey değildir. Tabi veriye state adı verildiğinden onun salt bir Javascript objesi olmaktan çıktığını ve sürekli kontrol altında tutulması gerektiğini unutmamamız gerekmektedir. Çünkü state, geliştirmiş olduğumuz Javascript bileşenine doğrudan bağımlıdır ve state üzerindeki her değişiklik View ‘ı doğrudan etkilemektedir. Yani artık mühim olan konu kodun kendisi değil, kodun üzerinde koştuğu verinin kendisidir.

State dediğimiz konu artık kodun kendisinden daha önemli bir konuma geldiği için, kod parçacıklarımız verinin nasıl saklanacağı, nasıl işleneceği gibi konulara çözüm bulma odaklı olmaktadır.

Ön yüz geliştirme dünyasına daha çok AngularJS ile girmiş olan bileşen tabanlı geliştirme yaklaşımı, bir web uygulamasının ayrı çalışabilen her bir alt parçasının bileşen olarak adlandırılan bölümler halinde geliştirilmesini önermektedir. Her bir bileşenin kendi yaşam döngüsü, kendi veri bağlamı ve kullanıcı ile etkileşim özellikleri vardır.

Bileşenler ve bileşen tabanlı yaklaşımlar, e-iş devriminin itici gücü ve internet çağı için kurumsal ölçekte çözümlerin geliştirilmesinin yoludur (Brown, 2000, 70). Bu nedenle bileşen yapısının ön yüz süreçlerinde kullanılması, her kütüphane ve uygulama çatısı özelinde farklılık ve çeşitlilik göstermektedir. Bu tez kapsamında ise bileşen tabanlı geliştirme, React ve Anatolia kütüphanesi temel alınarak incelenmektedir.

Bileşen tabanlı geliştirme yaklaşımını ön yüz özelinde incelerken, bir bileşenin dört özelliğine dikkat çekeceğiz. Bunlar;

## 2.1. Yaşam Döngüsü

Bileşenler temelde dört yaşam döngüsü aşamasına sahiptir. Bunlar, initialization, mounting, updation, unmounting olarak adlandırılmaktadır. Çoğu Javascript kütüphanesi bu aşamaları sağlamakla birlikte, bunların haricinde birçok yaşam döngüsü elemanı da sağlamaktadırlar (W3Path, *React Component Lifecycle*, https://w3path.com/react-component-lifecycle/, 18/01/2020).

## 2.2. Render Süreci

Ön yüz kütüphaneleri özelinde yaklaşırsak, render süreci, bir bileşein tarayıcı vasıtası ile DOM ‘e eklenilmesi anlamına gelmektedir. Bu aşamada yine her kütüphane kendi özel yaklaşımını kullanmaktadır. Genellikle bileşemn üzerinde bulunan render metodu yardımıyla, belirtilen direktiflere uygun olarak HTML node oluşturulur ve yeni bir node olarak DOM yapısına eklenir (Krunal Lathiya, *React*

*Lifecycle Methods Render And ComponentDidMount*, https://www.codingame.com/playgrounds/8747/react-lifecycle-methods-render-and-componentdidmount, 18/01/2020).

### 2.3. Veri Yönetimi

Bileşen yapısı gereği veri ile kuvvetli bağ içermektedir. Önceleri bileşen yapısı olmadan geliştirdiğimiz metotlar HTML render işlemini gerçekleştirdikten sonra HTML 'i veri bağlamında koparmaktaydı. Günümüzde ise bir bileşenin davranışsal yapısı, bileşenin veri ile ilişkisi üzerinden belirlenmektedir. Bu durumda ise bileşeni veri bağlamı ile birlikte değerlendirmek ve geliştirmek gerekmektedir (ReactJS, *Component State*, https://reactjs.org/docs/faq-state.html, 18/01/2020).

### 2.4. Yeniden Kullanılabilirlik

Bileşenler bir kez yazılıp ya da tasarlanıp bir den fazla yerde kullanılmak üzere geliştirilmektedirler. Bu nedenle davranışsal yapıları değiştirilmez. Bileşenin nasıl davranacağına property denilen özellikleri üzerinden erişim sağlamaktayız (ReactJS, *Components And Props*, https://reactjs.org/docs/components-and-props.html, 18/01/2020).

## 3. DECLARATIVE / IMPERATIVE PROGRAMLAMA

Declaratif programlama paradigması nasıl yapılacağı bilgisinden çok ne yapılacağı bilgisi ile ilgilenmektedir (Lloyd, 1994, 1). Declarative programlama bir işlemin ne olduğu ve nasıl olmadığı yazılarak gerçekleştirilir. İşlemin nasıl olması gerektiği, kod parçasını işleten derleyiciye ya da yorumlayıcıya bırakılmıştır. Declaratif yaklaşım bir programlama dili özelinde tümden uygulanmış olabileceği gibi(ör; SQL, XQuery vb.), yeni ön yüz kütüphanelerinin kullanımında da görülebilir. Aşağıdaki SQL sorgu örneğini incelediğimizde, bir veri tabanı tablosundan birtakım verilerin getirilmesini göstere SQL sorgusunu görebiliriz. Bu sorgu, neyin yapılması gerektiğini gösterdiği gibi nasıl yapılması gerektiğini gizlemektedir.

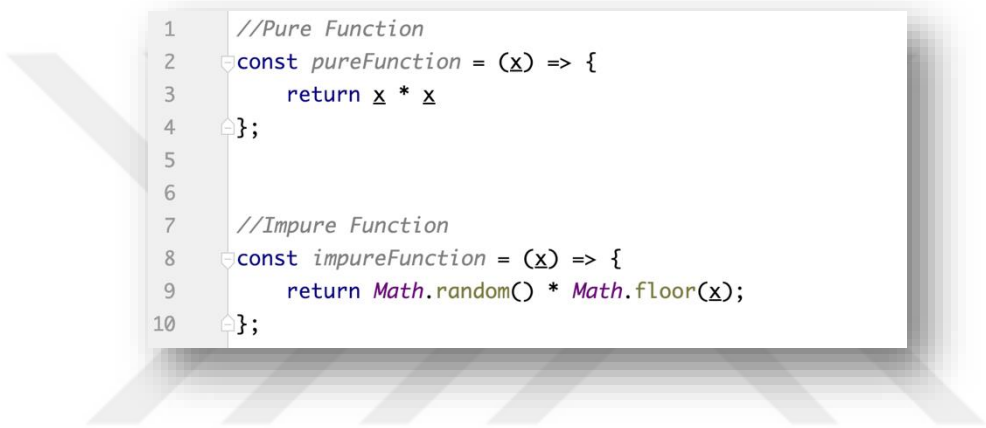
```
4
5   SELECT *
6   FROM USER_PRIVILEGES;
7
```

**Şekil 8** Declarative SQL Betiği

Imperatif programlama ise, bir programın nasıl çalıştığını anlatmaya odaklanır. Bu yaklaşımda program bir ya da birden fazla prosedürden ve komutlardan oluşmaktadır. Prosedürler, program state‘ ini lokal olarak değiştirebilmekte ve state değişiklikleri bu prosedürlerinin dönüş değerleri ile elde edilmektedir. Prosedürel programlamayı, deklaratif programlamaya doğru bir evrilme olarak düşünebiliriz. Bir prosedürün adına, parametrelerine ve geri dönüş değerine bakılarak detaylara girmeden işlemin nasıl gerçekleştiği konusunda bilgi sahibi olunabilir. Nesne yönelimli programlama, imperatif programlamanın bir uygulanma biçimidir.

## 4. FONKSİYONEL PROGRAMLAMA

İşlevsel programlama, matematiksel işlevleri tanımlama şeklimiz gibi, nasıl hesaplandığı yerine ne hesaplandığına odaklanır (Hu, Wang, Hughes, 2015, 1). Daha açık bir dille ifade etmek gerekirse fonksiyonel programlama; fonksiyonların yan etkilerini azaltmak ve ortak veri paylaşmaktan kaçınarak, saf fonksiyonlar ile programlama yapmak anlamına gelmektedir. Bunun için bir program, mümkün olan en yüksek efor ile saf(pure) fonksiyonlar kullanılarak geliştirilmektedir. Burada saf fonksiyonlar ile anlatılmak istenen, bir fonksiyonun her aynı giriş ifadesine karşılık hep aynı sonucu üretmesidir. Bir örnek vermek gerekirse;




**Şekil 9** Saf ve Saf Olmayan Fonksiyonlar

Burada Javascript dili ile geliştirilmiş ve ***pureFunction*** olarak isimlendirilen fonksiyon, parametre olarak almış olduğu her değer ***x*** değeri için, onun karesi döndürür ve her zaman aynı giriş değeri için için aynı çıkış değeri üretilir. Fakat yine Javascript dili ile geliştirilmiş olan ***impureFunction*** adlı fonksiyon, anlık olarak rastgele üretilen bir sayı ile çarpılarak döndürülür. Burada metodun çıkış değerini tahmin etmek imkansıza yakındır. Bu nedenle ikinci örnek saf fonksiyon standartlarını bozmaktadır.

Ayrıca fonksiyonel programlama, state bağlamını saf fonksiyonlar ile yönetmektedir. Global olarak erişilip değiştirilebilen state değerleri yerine bir state akışını modifiye eden saf fonksiyonlar kullanılmasını salık vermektedir. Bu sayede ön yüz geliştirmede en önemli parçalardan birisi olan verinin tam olarak nerede modifiye edildiğini tespit etmek ve yan etkilerden korunmak mümkün olmaktadır.

# İKİNCİ BÖLÜM

# BİLEŞEN VE VERİ İLİŞKİSİ

## 1. BİLEŞEN VE VERİ BAĞLAMI (DATA BINDING)

Veri bağlama konsepti kısa olarak, veriler ile kullanıcı arayüzü nesnelerinin programatik olarak ilişkilendirilmesidir. Ön yüz geliştirme süreçlerinde veri bağlama konusunda öncelikle verinin bağlanacağı HTML nesnesi ve verinin bir property 'si birbiri ile ilişkilendirilir. Bu sayede HTML elemanından veriye doğru ya da tam tersi bir şekilde veri akışı sağlanabilir.

İki tür veri yönetiminin de kullanılmasını gerektirecek durumlar mevcuttur. Örneklemek gerekirse, sınav sonuç bilgilerinin listelendiği bir uygulama olduğunu varsayalım. Bu durumda kullanıcı tarafından herhangi bir veri girişi olmayacağı için arayüz kısmında tek yönlü veri bağlama seçeneği uygun olacaktır. Fakat örneğin bir restoran tarafından kullanılan ve kullanıcı değerlendirmelerinin alındığı bir sistem olduğunu varsayarsak, bu durumda ise veri akışının kullanıcı arayüzünden veri kaynağına doğru güncellenmesi işleminin de yapılması gerekmektedir.

Ön yüz geliştirme süreçlerinde veri bağlama konusu bir bileşenin tüm anatomisini ve geliştirme süreçlerini etkileyen en önemli konudur. Bununla birlikte, ön yüz geliştirme dünyasında en çok kullanılan iki yöntem olan, tek yönlü ve çift yönlü veri bağlama yöntemleri üzerinde durulacaktır.

### 1.1. Tek Yönlü Veri Bağlama (One Way Data Binding)

Tek yönlü veri bağlama, verinin tek yönde, tipik olarak veri kaynağından kontrol mekanizmasına doğru aktığı durumdur. Bu durumda veri kontrol mekanizmasında olacağı ve kullanıcı tarafından değiştirilemeyeceği için verinin salt okunur olması durumu sağlanmış olur. Verinin değiştirilmesi gereken durumlarda veriyi güncelleyecek olan metotlar vasıtası ile ***mutation*** gerçekleşir.

### 1.2. İki Yönlü Veri Bağlama (Two Way Data Bındıng)

İki yönlü veri bağlama, verinin iki yönde, veri kaynağından görünüm/kontrol mekanizmasına ya da görünüm/kontrol mekanizmasından veri kaynağına doğru akışın olduğu durumdur. Bu durum, verinin kullanıcı arayüzünden(ör; bir input vasıtası ile) değiştirildiğinde veri kaynağında da değişeceğini garanti eder. Kullanıcının kontrol/görünüm katmanında yapmış olduğu tüm hareketler veri kaynağını doğrudan etkilemektedir.

## 2. VERİ YÖNETİMİ (STATE MANAGEMENT)

Veri yönetimi konusu, ön yüz geliştirme dünyasında, veri kaynağının nasıl kontrol edileceğini açıklamakta ve incelemektedir. Javascript özelinde veri yönetimi, veri kaynağı ön yüz elemanları ile etkileşimde olmasından kaynaklı olarak, uygulama kullanılmaya başlandığı andan itibaren kullanıcının veri üzerinde yapmış olduğu tüm işlemleri kapsamaktadır. Verinin dinlenilmesi ve gerekli güncellemelerin kullanıcı arayüzünde yapılması ya da çift yönlü veri bağlamada olduğu gibi kullanıcı tarafından yapılan değişikliklerin veri nesnesine yansıtılması veri yönetimi konusu kapsamına girmektedir.

Birçok web uygulamasında veriler bileşenler arasında ve uygulama genelinde dağıtılmış durumdadır. Bileşenlerin birbirleri ile olan etkileşimleri, yeni gereksinimler ya da bir verinin birden fazla kaynak tarafından üretiliyor ya da tüketiliyor olması, veri bağlamının karmaşıklaşmasına yol açmaktadır.

Verinin daha kolay yönetilebilmesi ve tek yönlü ya da çift yönlü olarak HTML elemanlarına bağlanabilmesi için veri yönetim araçları ve yaklaşımları geliştirilmiştir. Bu bağlamda redux, mobx gibi veri yönetim kütüphaneleri, verinin daha efektif yönetilebilmesi için günümüz arayüz kütüphanelerine entegre edilmişlerdir.

Bu kütüphaneler daha önce bahsetmiş olduğumuz programlama paradigmalarından bir ya da birkaçını kullanmaktadırlar. Bunlar, declaratif programlama ve fonksiyonel programlamadır.

Redux kütüphanesini örnek almamız gerekirse; bugün Redux, Facebook şirketinin bir veri yönetim biçimi olan Flux mimarisini uygulayan JS kütüphanesidir. Bu kütüphane, veri yönetimini React bileşenleri için optimize etmektedir.

Kütüphane, verileri store adı verilen global bir değişken içerisinde saklamaktadır. Veri üzerinde değişiklik sağlayacak her türden işlemi action olarak tanımlamaktadır. Bu bağlamda action‘ lar gerçekleştikçe ya da tetiklendikçe, önceki veri yeni gelen/oluşan bir veri ile değişmektedir. Veri dispatch işlemi ile Reducer adındaki fonksiyonlara aktarılmaktadır. Reducer‘ lar, veri üzerinde ne tür bir değişiklik işlemi yapılacaksa bunları gerçekleştirmekte ve Store objesine yansıtmaktadırlar. Yansıtılan objeler kullanıcı arayüzü katmanına tek yönlü veri bağlama sayesinde yansıtılmaktadırlar. Bununla birlikte, eğer kullanıcı tarafından yapılan bir işlem state nesnesini mutate edecekse de, yine bileşene ait olan setState metodu yardımıyla, yukarıda bahsetmiş olduğumuz adımlar, Redux kütüphanesi yardımıyla tekrarlanmakta ve kontrol katmanından veri katmanına veri akışı sağlanılmaktadır.

Teze konu olan iyileştirmeleri içeren Anatolia‘ da ise, state management mevcut web kütüphanelerinden farklı olarak, pub/sub tasarım şablonu kullanarak hazırlanmış bir veri yönetim kütüphanesi kullanmaktadır. Bu kütüphanenin en büyük farkı her bir bileşenin asenkron olarak birbiriyle ve herhangi bir veri kaynağıyla haberleşebilmesidir.

# 3. JAVASCRIP ‘İN DERLENMESİ VE HTML OLUŞTURULMASI

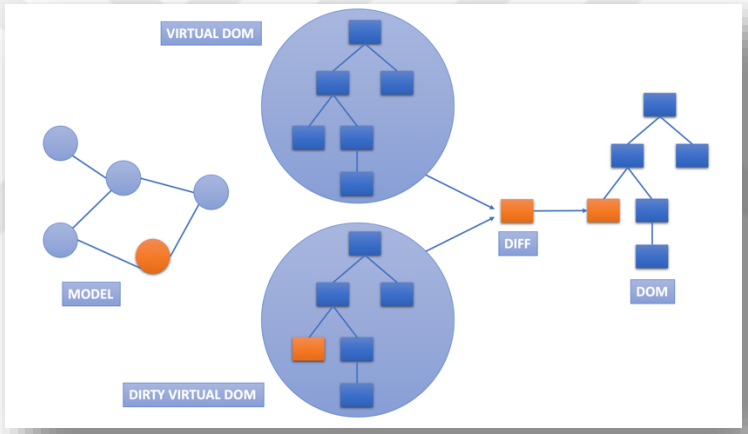
Bu bölümde kısaca, popüler web uygulama çatılarının bir web bileşenini nasıl işlediğini ve ekranda nasıl gösterdiğini göreceğiz. Şu anda React, Vue ve Angular gibi kütüphaneler aşağıdaki yaklaşımları kullanmaktadırlar.

## 3.1. Virtual DOM

Virtual DOM, Facebook ‘un React kütüphanesinin benimsediği, veri farklarını kıyaslayarak bileşenlerin oluşturmasını temel alan bir yaklaşımdır. Bu yaklaşıma göre, bir bileşen verilen bir state nesnesine göre render edilir. Bu render aşamasından sonra,

bileşen içerisinde vDOM adı ile bir nesne tutulmaktadır. Bu nesne, bileşenin tuttuğu state nesnesine göre bileşenin HTML görüntüsünün JS nesnesine dönüştürülmüş halidir. Bir t anında state değişip bileşen kendini tekrar render ettiğinde, prosedür tekrarlanarak yeni bir vDOM nesnesi oluşmaktadır. Bu nesneye özel olarak dirty vDOM denilmektedir. Bileşen ekrana çizilmeden hemen önce, ilk oluşan vDOM nesnesi ile dirty vDOM, sahip oldukları her node bakımından kıyaslanır ve dirty vDOM üzerindeki değişen kısımlar, ilk vDOM 'a, iki DOM nesnesi eşit olana kadar yansıtılır. Son olarak, vDOM üzerinde değişen kısımlar mevcut tarayıcı DOM 'u üzerine de yansıtılır ve proses böylece son bulur.

Bu yaklaşımın performansa en büyük etkisi, verinin her değişiminde bileşenin tamamının render edilmesi yerine, yalnızca değişen yerleri yansıtılmasıdır. Bu sayede daha az render işlemi yapılmaktadır.




**Şekil 10** Virtual DOM Çalışma Prensibi

## 3.2. Memoization

'Memoization' terimi Donald Michie tarafından 1968 yılında "Memo" Fonksiyonlar ve Makine Öğrenimi [Mic68] adlı makalesinde yer aldı. Çalışması, bir fonksiyonun sonucunu kaydetmenin yararını gösteren ilk çalışmadır ve dolayısıyla Michie, memoization 'ın babası olarak kabul edilir (Suresh, 2016, 21).

Memoization yaklaşımı, frontend dünyasında yine Facebook 'un React kütüphanesinde ve Imba adlı web uygulama çatısında kullanılmaktadır. Bu yaklaşım, bir fonksiyonun her zaman aynı giriş değerlerine karşılık aynı çıkış değerleri üretmesi fikrine dayanır. Bu yaklaşımda, örneğin render işleminden sorumlu olan metoda

verilen parametreler bir orta katman yardımıyla takip edilir, giriş değerleri orta katmanda yer alan herhangi bir giriş değeri ile eşleşiyorsa, bu metot çalıştırılıp sonuç üretmesi beklenmeden, metodun bu giriş değerine karşılık daha önce üretmiş olduğu sonuç döndürülür. Bu sayede metodun yürütülme süresinden tasarruf edilmiş olur.

# ÜÇÜNCÜ BÖLÜM

# ANATOLIA JAVASCRIPT KÜTÜPHANESİ

## 1. ANATOLIA 'NIN BİLEŞENLERE YAKLAŞIMI

Anatolia, veri ile iç içe bir tasarım sergilemesi açısından kodu bileşen denen daha alt birimlere bölüyor ve veri nesnelerimizi bileşenlerin içerisinde saklıyor.

Bir bileşen sınıfı geliştirirken göz önünde bulundurmamız gereken dört konu vardır.

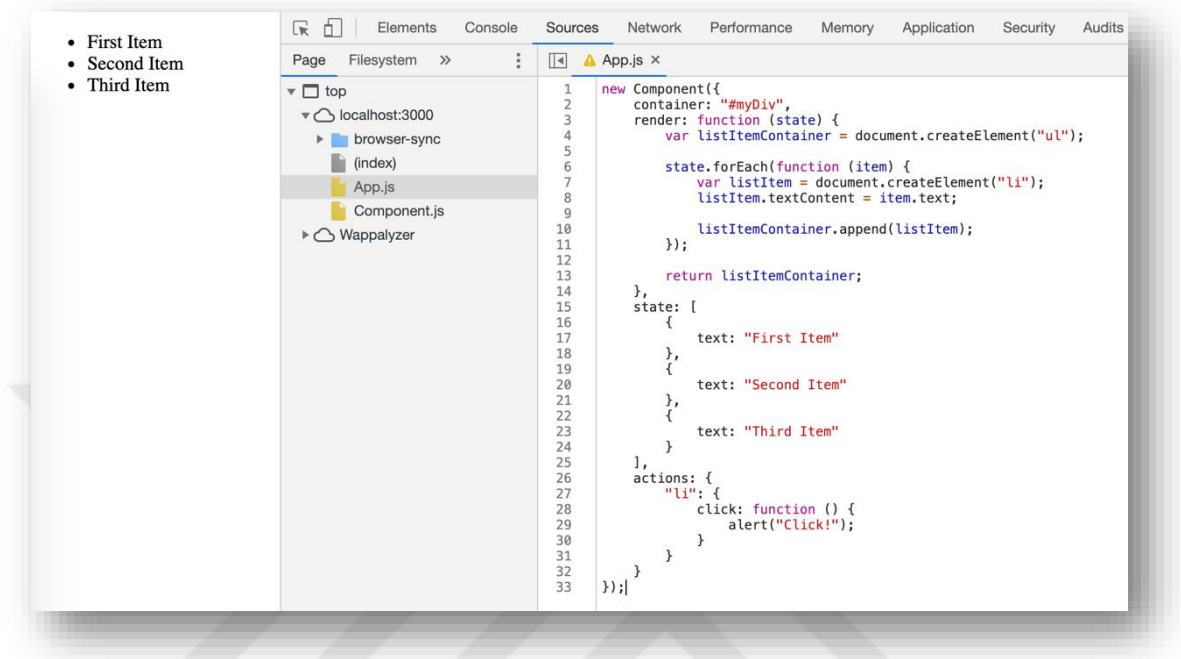
a) Bileşenin HTML de nereye render edileceği bilgisi
b) Bileşenin nasıl render edileceği bilgisi
c) Bileşenin üzerinde koşacağı ve saklayacağı state bilgisi
d) Bileşenin etkileşimli olması için gerekli Javascript event bilgileri

Bu dört madde bileşenin ana unsurlarıdır. Bize bir bileşen nesnesi oluşturacak olan Javascript metodumuz kullanımı temel olarak aşağıdakine benzeyecektir.

```
1  new Component( options: {
2      container: "",
3      state: {},
4      render: function (state) {
5
6      },
7      actions: {}
8  });
```

**Şekil 11** Anatolia 'da Bileşen Yapısı

Bu yapı yardımıyla, bir liste yapısı bileşen yardımıyla uygulandığında aşağıdaki gibi görünmektedir.

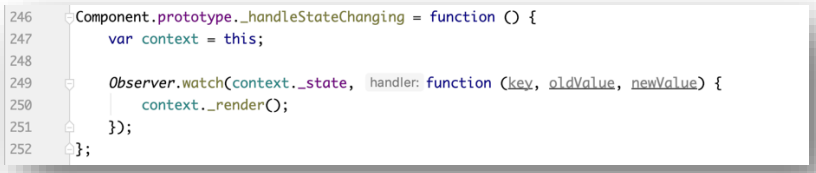



**Şekil 12** Anatolia ‘da Bileşen Kullanımı

Anatolia, bileşen yapısı içerisinde fonksiyonel programlama tekniği kullanılmıştır. Bu teknik kullanılarak ile mümkün olduğunca az işlem ile sonuca ulaşılmaya çalışılmıştır. Metotlar üzerindeki soyutlama işlemleri ve render mekanizması olabildiğince sonuç odaklı ve en az işlemle çıktı verecek şekilde tasarlanmıştır. Nihayetinde HTML DOM ‘unun oluşabilmesi işin soyutlamadan uzak olarak doğrudan DOM kütüphanesi kullanılmıştır.

### 1.1. DOM Oluşturma Stratejisi

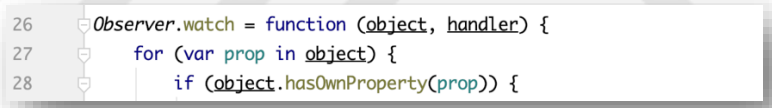
Anatolia Component yaklaşımı içerisinde HTML oluşturmak için gereken en az eforu sarf etmek üzere tasarlanmıştır. Bunun için mümkün olan en az kodla en fazla işlemi yapmayı ön planda tutar. Aşağıda DOM oluşturma stratejisinin önemli noktalarına değinilmiştir




**Şekil 13** State Değişikliklerini Dinleyen Metot 1

Bileşen içerisindeki state nesnesi her yeniden set edildiğinde, ***_handleStateChanging*** metodunu tetiklemektedir. Bu metot Observer sınıfı içerisindeki watch metodunu çalıştırmaktadır. ***handleStateChanging*** metodu O(n) kompleksitesine sahiptir. O(n) kompleksitesi -ya da doğrusal çalışma zamanı-, giriş değerindeki -state nesnesi- değişiklik gösteren her bir property için boyutla orantılı olarak büyüyecektir. Eğer state nesnesi içerisinde bir önceki state ile farklılık gösteren üç property varsa, **Observer.watch** metodu üç kez tetiklenecektir.

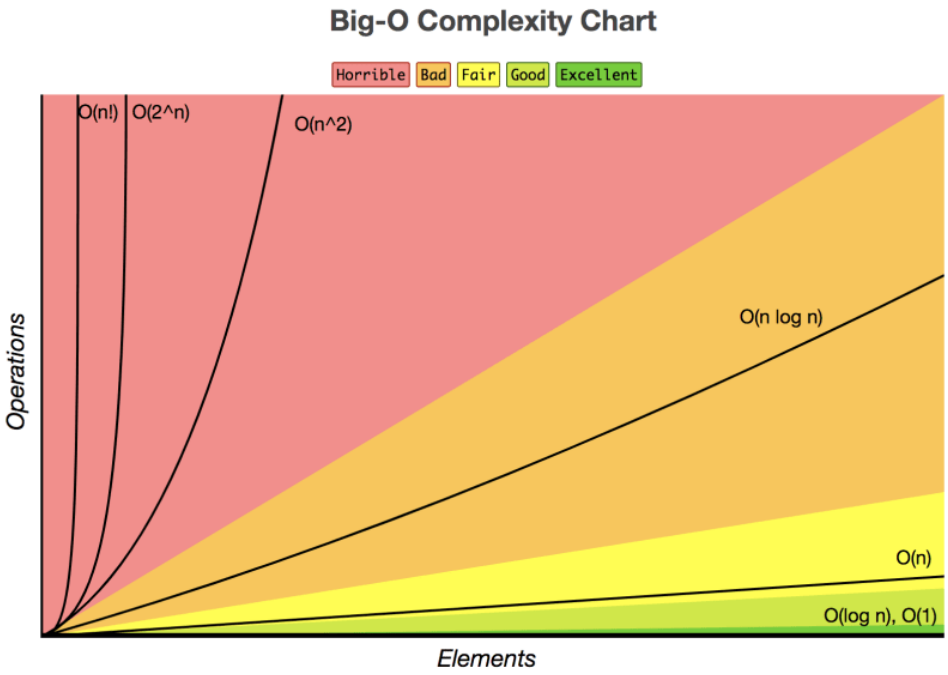
Observer.watch metodu aşağıdaki gibi bir yapıya sahiptir.




**Şekil 14** State Değişikliklerini Dinleyen Metot 2

Bu metot ise, değişen state içerisindeki her bir nesne elemanını tek tek ele alarak, state nesnesi içerisinde bu nesne elemanlarıyla aynı isimli fakat gizli nesne elemanları tanımlar. Bu gizli nesne elemanları ise, watch metodunun içerisinde oldValue değerine eş gelmekte ve nesne elemanının set edilmeden önceki değerini tutmaktadırlar. Observer.watch metodu yine O(n) kompleksitesine sahiptir. Bu nedenle doğrusal çalışma zamanında çalıştır.

O(n) karmaşıklığı daha iyi görmek için aşağıdaki çizelgeyi inceleyebiliriz.

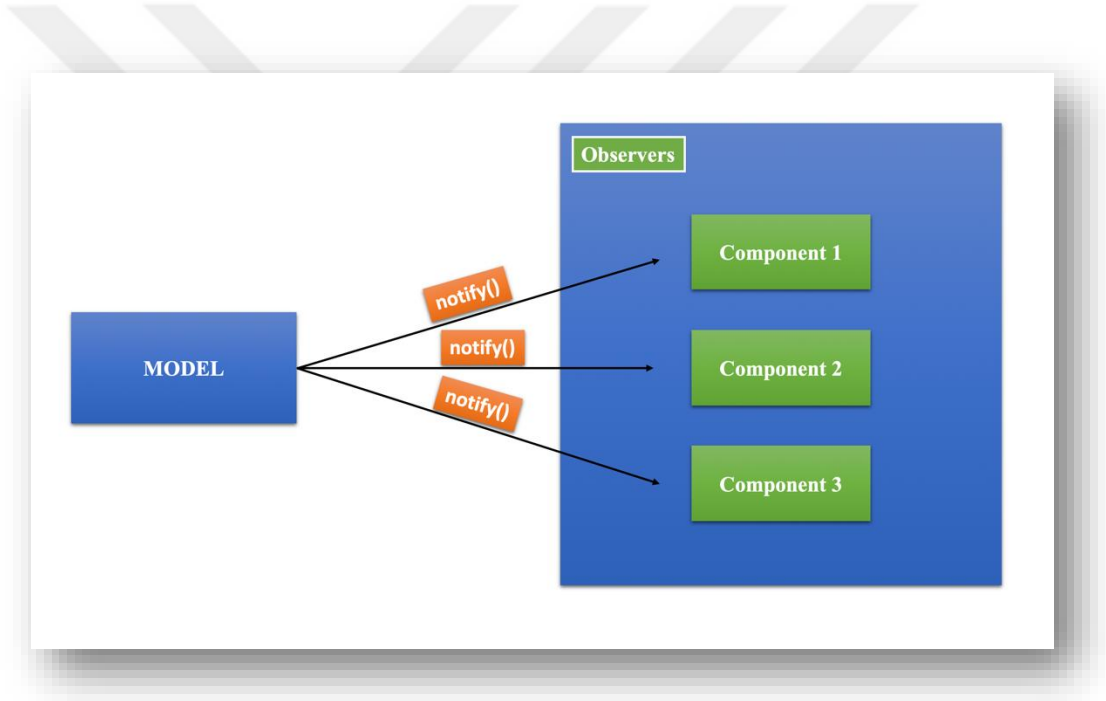



**Şekil 15** Big O Notasyonu Kompleksite Çizelgesi (Big-O Cheat Sheet, *Know Thy Complexities*, https://www.bigocheatsheet.com/, 18/01/2020)

## 2. PUBLISHER / SUBSCRIBER İLE VERİ YÖNETİMİ

Anatolia, bileşenlerin kendi aralarında ya da bir veri kaynağı ile -lokal dataset, RS, WS vb...- haberleşebilmesi için Publisher/Subsciber tasarım kalıbını kullanan bir veri yönetim mekanizması kullanmaktadır. Bu veri yönetim mekanizmasının en önemli özelliği, verileri global nesneler içerisinde soyutlaması ve Mutation adlı nesneler yardımıyla state değişimlerini yönetebilmesini sağlamaktır.

Veri değişimleri genellikle Observer tasarım kalıbı ile yapmaktayız. Observer tasarım kalıbı bize veri değişimlerini aşağıdaki görselde açıklandığı gibi yönetmemizi salık verir.

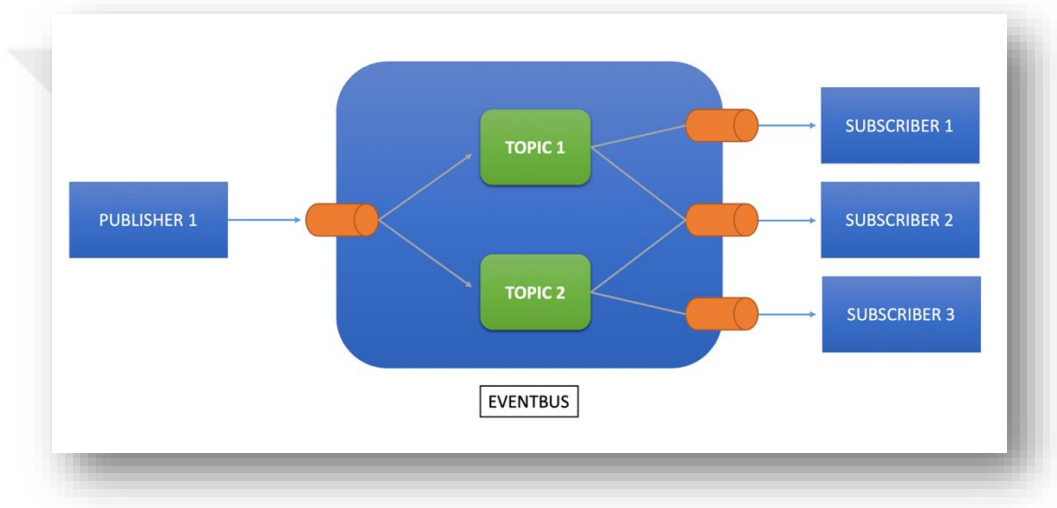



**Şekil 16** Observer Tasarım Şablonu

Observer tasarım kalıbında gözlemciler, veri kaynağı ile doğrudan haberleşmektedirler. Observer yapı ile takip edilen veri kaynağı herhangi bir değişiklik durumunda, gözlemcileri kendileri haberdar etmektedirler. Bunun yapılabilmesi için, gözlemciler veri kaynağını takip eden metotlarını, kaynağa vermektedirler. Veri kaynağı değiştiği takdirde ise, bu metotları değişen state nesnesi

ile çalıştırmaktadır. Bu durumda veri kaynağı ile gözlemciler birbirlerinden doğrudan haberdarlardır.

Anatolia ise, Publisher/Subscriber temelli bir veri yönetim mekanizması kullanarak, gözlemci ile veri kaynağını birbirlerinden doğrudan haberdar olmalarının önüne geçerek, daha geniş imkanlara sahip bir geliştirme ortamı sağlamaktadır. Aşağıdaki şekil üzerinden ilerlersek, Publisher/Subscriber 'ın çalışma mantığını tam olarak anlayabiliriz.

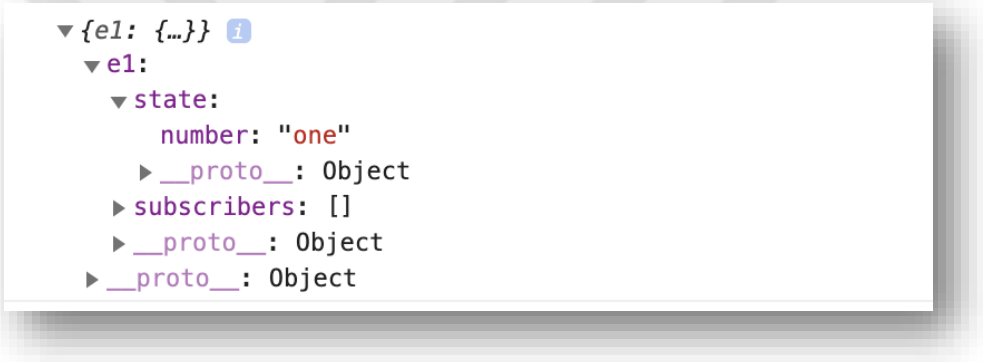



**Şekil 17** Pub/Sub Tasarım Şablonu

Şekilde görüldüğü gibi, Publisher/Subscriber yapısında, veri kaynağı(publisher) ve aboneler(subscriber) birbirlerine bir orta katman vasıtası ile bağlıdır. Bu orta katmana Eventbus denilmektedir. Publisher, bir veri kaynağından almış oldukları veri setini, bir anahtar - değer ikilisi şekilde Eventbus yapısı üzerinde tutmaktadır. Aşağıdaki şekil bu yapının Anatolia üzerindeki uygulamasını göstermektedir.


**Şekil 18** Anatolia Publisher Uygulaması

Publisher, Eventbus üzerine kaydedildikten, Eventbus‘ ın son durumu aşağıdaki gibi olmaktadır.




**Şekil 19** Publisher Kaydından Sonra Eventbus‘ ın Son Durumu

Subscriber‘ lar ise, bu anahtarın değerinin tutmuş olduğu değeri gözlemleyebilmek için, Eventbus üzerinde kendilerini kaydederler.


**Şekil 20** Anatolia Subscriber Uygulaması

Subscriber, Eventbus üzerine register edildiğinde, Eventbus‘ ın son durumu aşağıdaki gibi olmaktadır.

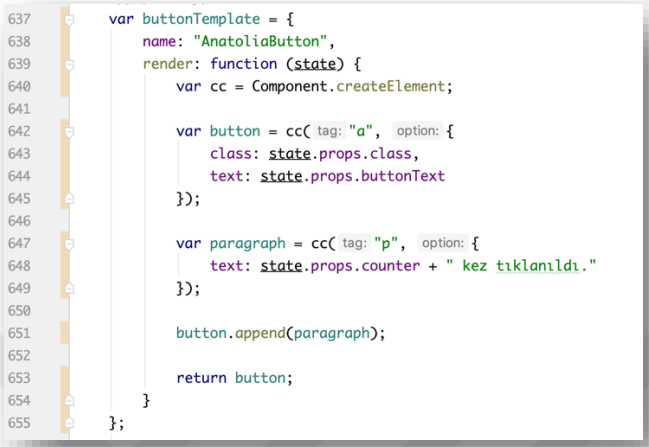

**Şekil 21** Subscriber Kaydından Sonra Eventbus‘ ın Son Durumu

Burada Eventbus‘ ın görevi, Publisher ‘dan gelen veri nesnesini verilen anahtar değeri ile saklamak ve Subscriber ‘dan gelen abonelik talebini de, bu anahtar değer ile eşleştirmektir.

Observer tasarım kalıbından temel fark olarak, veri hangi kaynaktan, hangi yöntemle gelirse gelsin, Subscriber yapılarında herhangi bir değişiklik yapmadan gözlemleme imkanı sağlanmaktadır. Ayrıca çok daha az kod ile diğer veri yönetim mekanizmalarından daha efektif çalışmakta ve kod yapısı daha anlaşılır olmaktadır..

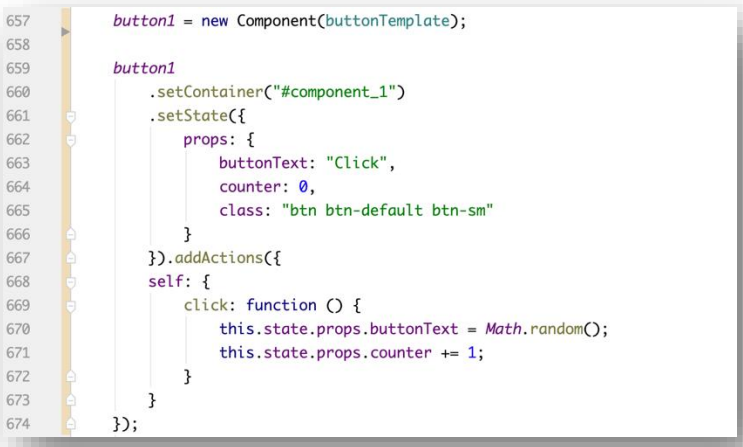
## 3. TEKRAR KULLANILABİLİRLİK

Anatolia, bileşenlerin tekrar kullanılabilirliğinin sağlanabilmesi için Javascript nesnelerini kullanır. Aşağıdaki örnek, render metodunun bir JS nesne şablonuna dönüştürerek nasıl tekrar kullanabileceğini göstermektedir.



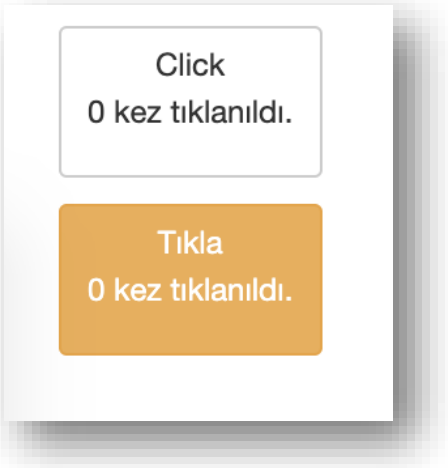

**Şekil 22** Anatolia 'da Bileşen Şablonları

Bu şablon yapısı, bileşenlerin aynı yapı üzerinden türetilmesini sağlamaktadır. Burada render metodu, veri bağımsız olarak yalnızca bir buton elemanının nasıl HTML yapısına sahip olması gerektiğini tanımlamaktadır. Aşağıdaki şekil, aynı şablon üzerinden farklı verilere sahip butonların nasıl türetildiğini göstermektedir.




**Şekil 23** Anatolia 'da Yeniden Kullanılabilirlik

Bu yapı ile türetilen iki butonun tarayıcıdaki son halleri aşağıdaki gibidir.




**Şekil 24** Anatolia ‘da Üretilen Bileşenler

# DÖRDÜNCÜ BÖLÜM

# ARAŞTIRMA METOTLARI

## 1. KULLANILAN METOTLARI

Gerek bileşen yapısı gerekse diğer kütüphanelerin kullanmış olduğu yaklaşımlar olsun, Javascript dünyasında bu konular yeterince tartışılmamıştır ve oldukça az sayıda çalışma yapılmıştır. Bu nedenle bu konuda literatür oldukça kısıtlıdır. Fakat buna karşılık, konu Javascript geliştiricileri arasında oldukça ilgi çekici duruma gelmiş ve geliştiriciler tarafından makale, video ya da benzer kaynaklar üretilmiştir.

Araştırmanın yapılış süreci boyunca hem sayısal ifadelere dayalı araştırmalar yapılmıştır. Bununla birlikte performans kantitatif yöntemler kullanılarak incelenmiştir.

## 2. PERFORMANS TESTLERİ

Anatolia‘ nın diğer (React, Vue vs…) kütüphanelere nazaran daha iyi performans gösterdiğini dair biz dizi test yapıldı. Testlerin amacı, daha hızlı render sürecini göstermek için performans sayılarını göstermektir.

### 2.1. Geçerlilik

Performans testleri platformlara bağımlıdır, çevresel değişkenlere(işlemci hızı, RAM hızı vs…) bağlı olarak değişir. Bu nedenle performansı ölçmek ve sayısallaştırmak kolay değildir. Bu tez kapsamında hazırlanan testler mümkün olduğunca basit tutulmuşlardır. Testlerde farklılıkların ortaya çıkmaması için aynı durumlar kodlanmıştır. Ayrıca hepsine eşit şartların sağlanabilmesi için tüm testler aynı makineler üzerinde koşturulmuşlardır. Performans ölçümünde sayılar kesin ve nihai değildir, sayıların göstermiş oldukları eğilimler gösterilmiştir ve üzerinde durulmuştur.

## 2.2. Uygulama

Test durumlarının uygulandığı kodlar GitHub üzerinden servis edilmektedir. Ayrıca kodlar üzerinde daha fazla bilgi alabilmek için EK 1 e bakabilirsiniz.

## 2.3. Test Durumları

Temelde iki durum test edilmiştir. Bunlar;

- İlk yüklenme
- Buton Tıklanmaları
- Benchmark.js Performans Testleri

Bu durumlardan ilk yüklenme adımı, uygulama çatısının uygulamanın ilk oluşturulmasının performans sayılarına erişmek için uygulanmaktadır. Buton tıklamaları adımı ise, butonun üzerindeki veri nesnesinin değişim performansını ve değişen veri ile birlikte tekrar render olan HTML' in oluşma süresini ölçmek için kullanılmaktadır.

## 2.4. Testlerin Uygulanması

Testler aşağıdaki özelliklere sahip bir bilgisayar üzerinde koşturulmuşlardır. Ayrıca Chrome tarayıcısı gizli modda çalıştırılmış olup, tarayıcı eklentilerinin sayfa çalışmasına olabilecek olası müdahalesi engellenmek istenmiştir.

- **Bilgisayar**

İşletim Sistemi : macOS Mojave Version 10.14.16
Bellek : 16GB 2133 MHz
İşlemci : 2.9 GHz Intel Core i7
Grafik İşlemci : Intel HD Graphics 630 1536 MB
İşletim Sistemi Tipi : 64-Bit

- **Web Tarayıcı**

Tarayıcı Adı : Google Chrome
Versiyon : 79.0.3945.88 (Official Build) (64-Bit)

## 2.5. Test Araçları

Performans ölçümlerinin yapılabilmesi için Google Chrome tarayıcısına ait Performans paneli kullanılmıştır. Bu panel, uygulamanın ya da sayfanın yürütme süresini ve bellek kullanımını ölçümlemeye yaramaktadır. Bu panelde iki ölçümleme aracı bulunur;

- CPU Profiler : Bu araç Javascript‘ in toplam yürütme süresinin hangi metotlarda harcandığını göstermektedir.
- Heap Profiler : Bu araç ise Javascript nesnelerinin bellek dağılımlarını göstermektedir.

# BEŞİNCİ BÖLÜM

# PERFORMANS TEST SONUÇLARI

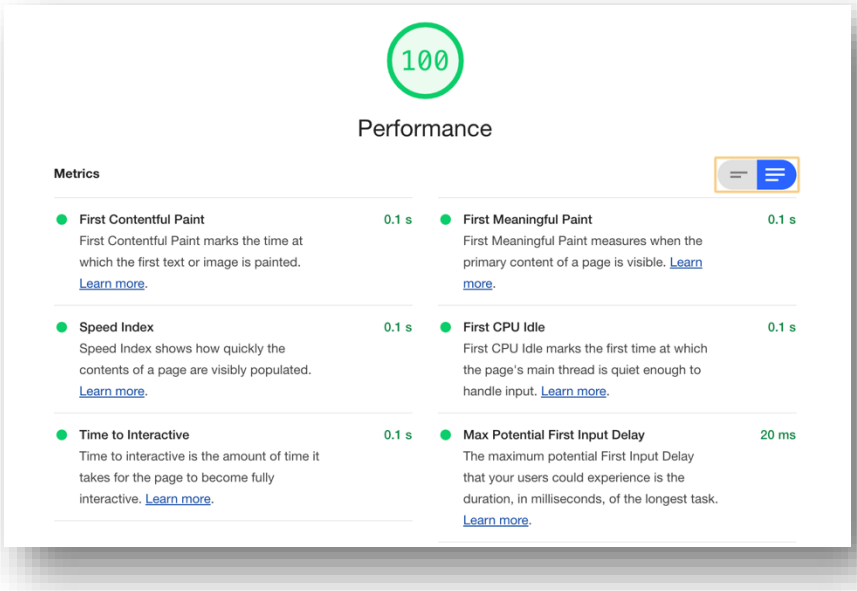
Performans testleri iki ana başlık altında incelenmiştir. İlk yükleme sonuçları, her iki uygulamada da en sade hali ile hazırlanmış "Yapılacak Listesi" uygulamasının elli kalem yapılacaklar listesi üzerinden hesaplanmıştır. Chrome tarayıcısı üzerinde Lighthouse aracı kullanılmıştır. Buton tıklama testleri ise yine Chrome tarayıcısına ait Performans aracı yardımıyla yapılmıştır.

## 1. NORMAL DURUM MOBİL CİHAZ PERFORMANS TESTİ

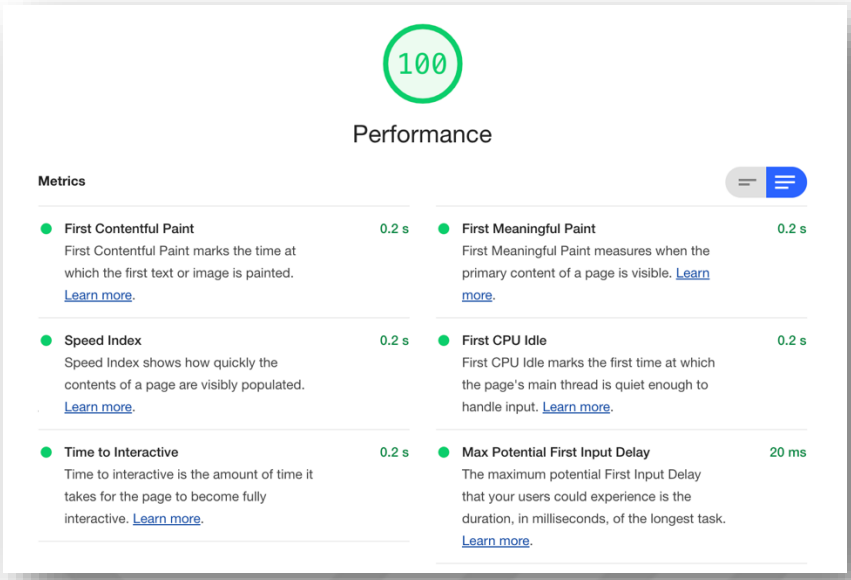
Performans testine başlanılmadan önce aşağıdaki seçenekler işaretlenmiştir.

**Tablo 1** Normal Durum Mobil Cihaz Performans Test Bilgileri

| SEÇENEK | DEĞER |
|---|---|
| **Device** | Mobile |
| **Audits** | Performance |
| **Throttling** | No Throttling |




**Şekil 25** Anatolia Normal Durum Testi




**Şekil 26** React Normal Durum Testi

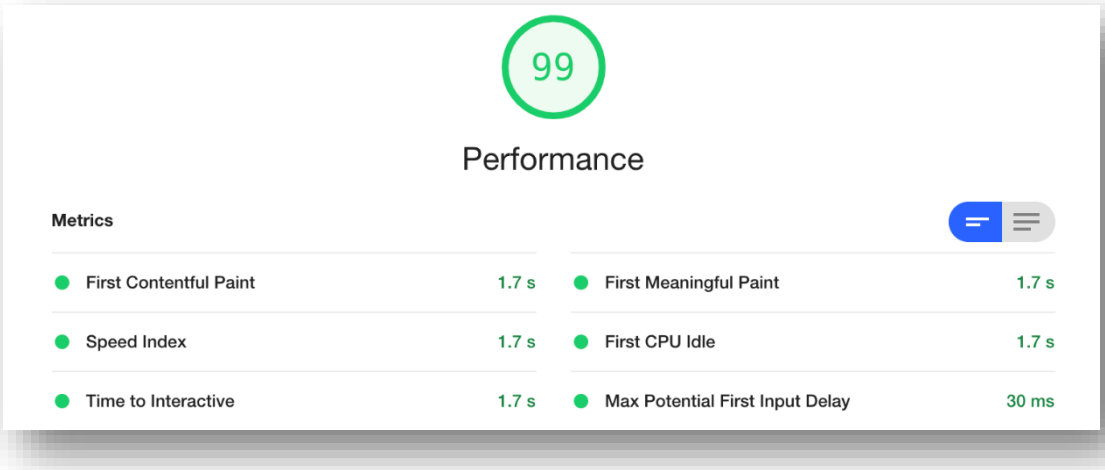
Normal durum mobil cihaz performans testine göre, Anatolia ilk anlamlı uygulama görüntüsünü, React ‘a nazaran daha hızlı oluşturmaktadır.

## 2. YAVAŞLATILMIŞ DURUM MOBİL CİHAZ PERFORMANS TESTİ

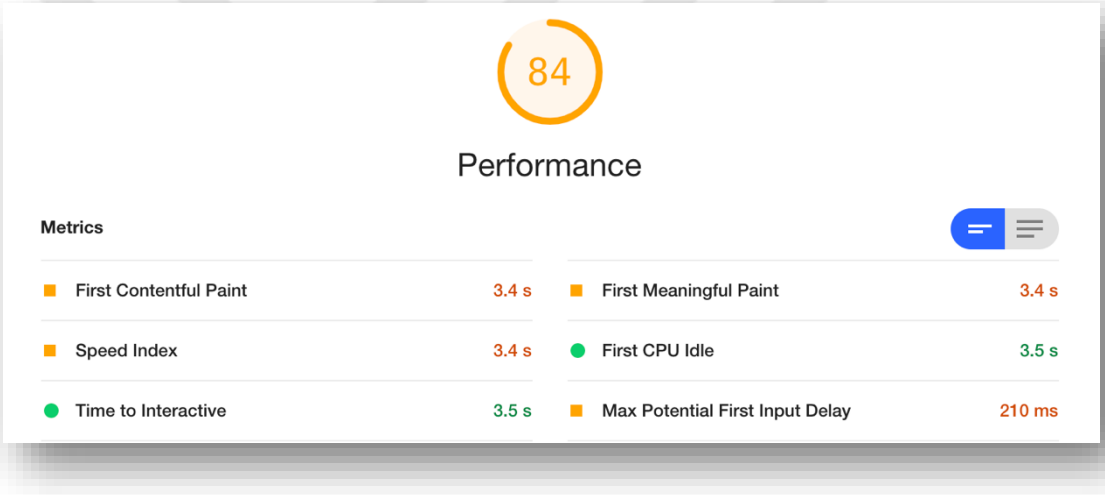
Bu test durumunda yavaşlatılmış bir internet hızı ve dört kez yavaşlatılmış işlemci hızı söz konusudur.

**Tablo 2** Yavaşlatılmış Durum Mobil Cihaz Performans Test Bilgileri

| SEÇENEK | DEĞER |
|---|---|
| **Device** | Mobile |
| **Audits** | Performance |
| **Throttling** | Simulated Slow 4G, 4x CPU Slowdown |




**Şekil 27** Anatolia Yavaşlatılmış Durum Testi




**Şekil 28** React Yavaşlatılmış Durum Testi

Yavaşlatılmış durum mobil cihaz performans testine göre, Anatolia ilk anlamlı uygulama görüntüsünü, React‘ a nazaran iki kat hızlı oluşturmaktadır.

## 3. NORMAL DURUM MASAÜSTÜ CİHAZ PERFORMANS TESTİ

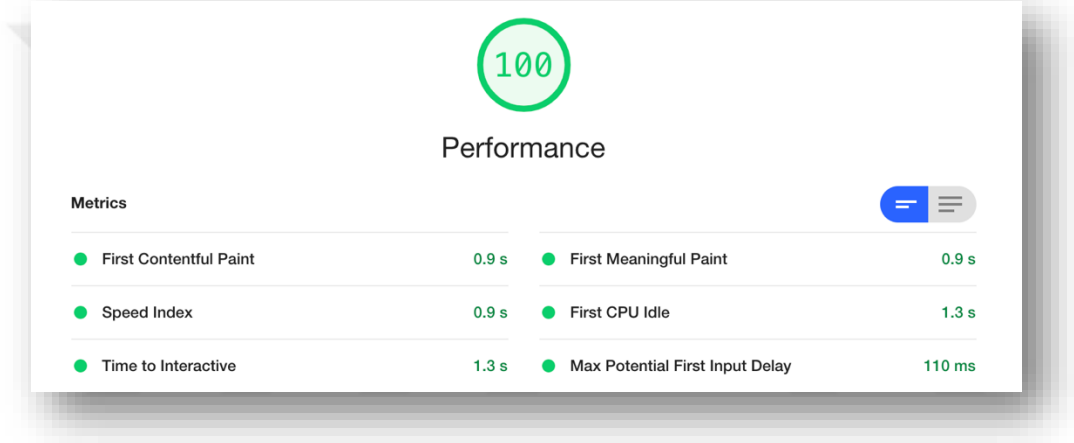
Bu test durumunda birinci adımla neredeyse bire bir aynı sonuçlar alındığı için şekilsel gösterimleri bilerek yapılmamıştır.

## 4. YAVAŞLATILMIŞ DURUM MASAÜSTÜ CİHAZ PERFORMANS TESTİ

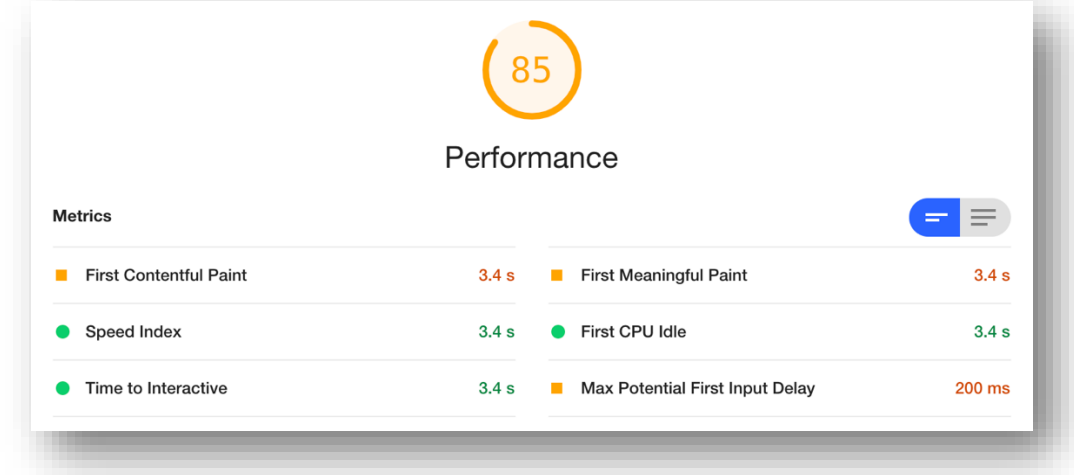
Bu test durumunda yavaşlatılmış bir internet hızı ve dört kez yavaşlatılmış işlemci hızı söz konusudur.

**Tablo 3** Yavaşlatılmış Durum Masaüstü Cihaz Performans Test Bilgileri

| SEÇENEK | DEĞER |
|---|---|
| **Device** | Desktop |
| **Audits** | Performance |
| **Throttling** | Simulated Slow 4G, 4x CPU Slowdown |




**Şekil 29** Anatolia Yavaşlatılmış Durum Testi

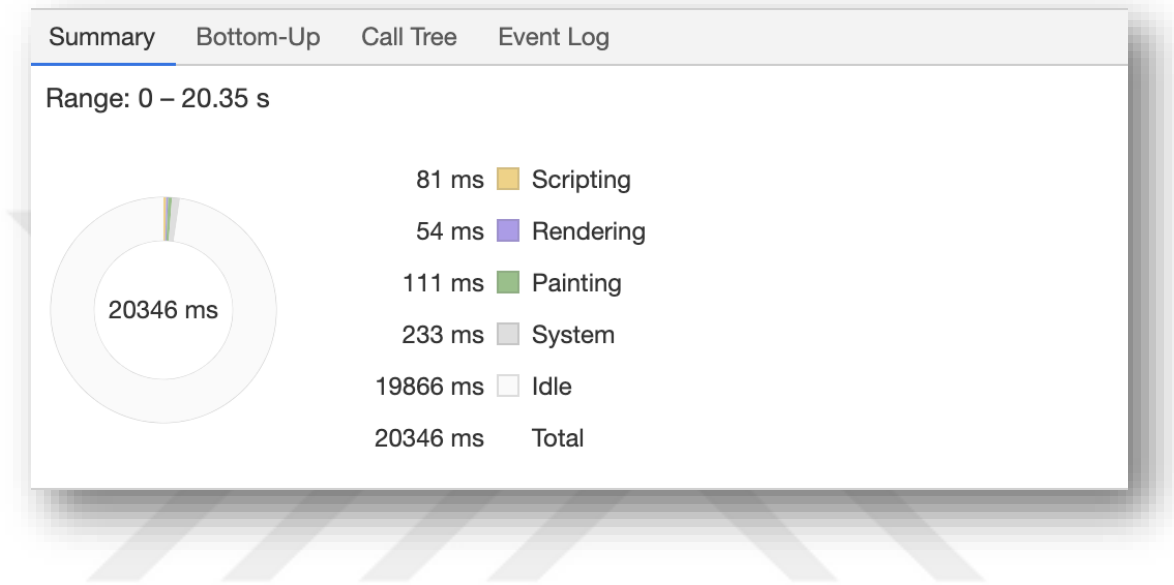



**Şekil 30** React Yavaşlatılmış Durum Testi

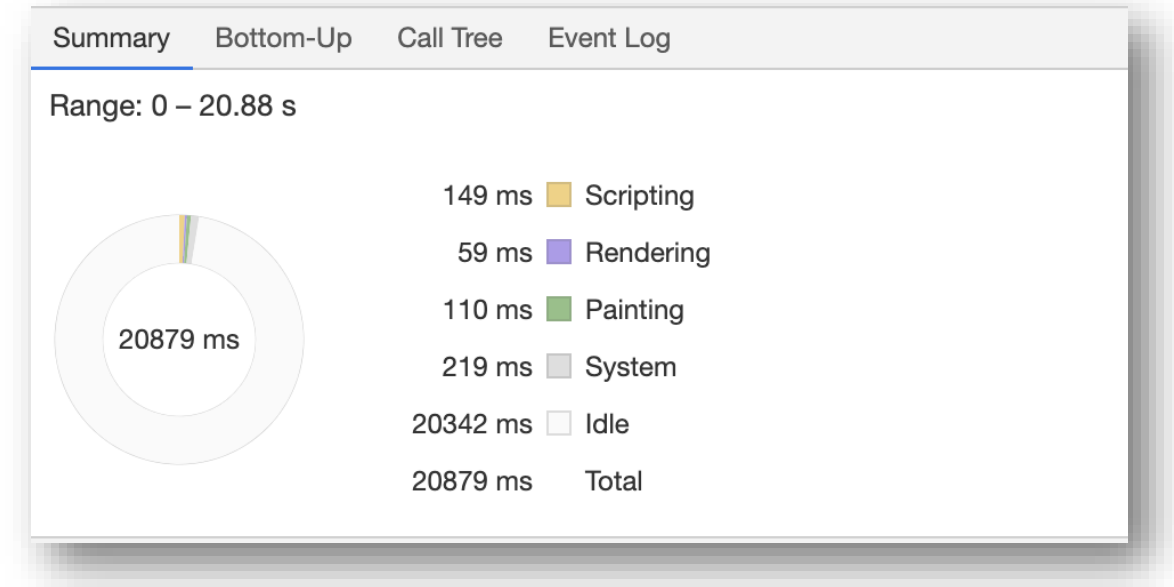
Yavaşlatılmış durum masaüstü cihaz performans testine göre, Anatolia ilk anlamlı uygulama görüntüsünü, React‘ a nazaran dört kat hızlı oluşturmaktadır.

## 5. UYGULAMA DEĞİŞİKLİKLERİNİN PERFORMANS TESTİ

Bu test adımında buton tıklama vasıtasıyla yapılacaklar listesine on kalem madde eklenmiştir ve tıklamalar ölçümlenmiştir.




**Şekil 31** Anatolia Yürütme Süreci




**Şekil 32** React Yürütme Süreci

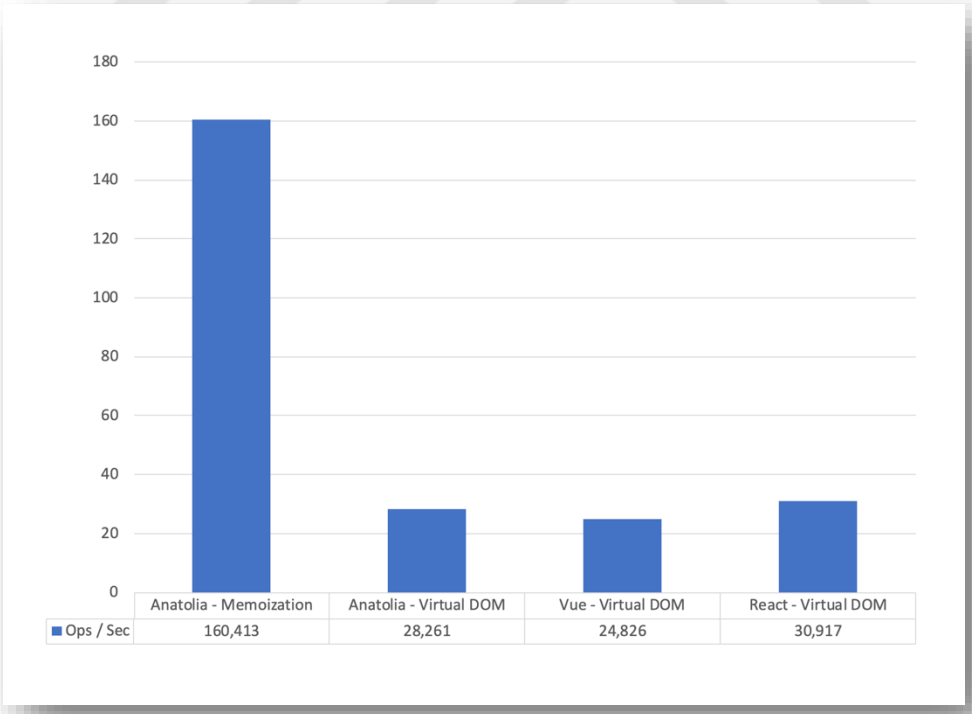
Bu test sonucundan anlaşılması gereken, Anatolia' nın aynı işlemi yapabilmek için Javascript' i yürütme süresi, React kütüphanesine nazaran neredeyse yarı süresidir. Ölçümleme süresindeki sıfırdan sonraki kısımlar dikkate alınmamış, yaklaşık süre üzerinde durulmuş olup, uygulamanın çalışma süresi boyunca göstermiş olduğu eğilime dikkat çekilmek istenilmiştir.

## 6. BENCHMARK.JS PERFORMANS TEST SONUÇLARI

Benchmarj.js, Javascript fonksiyonlarının performans metriklerini ölçümlemek için kullanılan bir kütüphanedir. Benchmark.js ile yapılmış testlerde, Memoization kullanan Anatolia sürümü, Virtual DOM kullanan Anatolia sürümü, React ve Vue kütüphaneleri aynı bileşeni uygulamıştır. Testte yürütülen kod parçacığını EK 1 de bulabilirsiniz.

Her bir test döngüsünde hemen hemen aynı sayıda örnekleme yapılmış olup, hata payları da test sonuçlarının yanlarında yer almaktadır.

Test sonuçları aşağıdaki gibidir.




**Şekil 33** Benchmark.js Performans Testi Sonuçları

# SONUÇ

Yukarıda bahsedilen test adımlarının sonuçlarına göre, Anatolia' nın bileşen yaklaşımına getirmiş olduğu uygulama çözümü, diğer Javascript kütüphanelerine göre HTML' in daha hızlı oluşturulması ve yorumlanmasını sağlamıştır. Bu performans iyileştirilmesindeki temel etken algoritma kompleksitesinin daha düşük olması ve mümkün olan en az adımı kullanarak sonuca ulaşmasıdır.

Anatolia' nın hedefi, diğer Javascript kütüphanelerindeki gibi popüler olan yaklaşımları barındırmak ve alternatif olarak daha performanslı uygulamalarını sağlamaktır.

Herhangi bir ön derleyici ya da paketleyici yazılımına ihtiyaç duymaması legacy -miras, eski sistemler- diye tabir edilen projelere de hızlıca entegre edilebilmesini sağlamaktadır. Bu bakımdan Anatolia' nın geliştirme süresinede olumlu yönde etki edeceğini düşünülmektedir.

Anatolia' nın gerek hızlı yüklenme ve gerekse daha sonra gerçekleşecek olan kullanıcı etkileşimlerinin bileşen ve veri seti üzerinde de hızlı olacağı önermesi, yapılan test sonuçları ile doğrulanmıştır. Bu test sonuçlarından ilkine göre mobil cihazlarda herhangi bir donanım kısıtlamasına maruz kalınmadan mobil cihazlarda ilk yükleme, React kütüphanesine nazaran %100 daha hızlıdır. İkinci test durumu olan mobil cihazlarda kısıtlanmış internet ve işlemci hızında ise ilk yükleme yine React'a göre yaklaşık %100 daha hızlıdır. Üçüncü test durumu sonuçları neredeyse birinci test durumu ile birebir aynı olup dördüncü test durumu ise yine kısıtlanmış internet ve işlemci hızında masaüstü cihazlarda ilk yükleme React'a göre %270 daha hızlıdır. Beşinci test durumu olan kullanıcı etkileşiminde ise Anatolia ile yazılmış uygulamanın React kütüphanesi ile yazılmış versiyonuna göre Javascript kodunun tarayıcı tarafından yorumlanmasında %80 daha hızlı sonuç alınmıştır.

Yazılım şirketlerinde bir projede teknoloji seçimi; güvenilirlik, esneklik, kullanım kolaylığı, adaptasyon süresi, eklenti ve komünite desteği gibi bir çok kriteri barındırmaktadır. Anatolia tarafında ise bu kriterlere mümkün olan yüksek seviyede

özen gösterilmiştir. Bu kriterlerden komünite desteğinin ise, Anatolia' nın kısa ve orta vadede kullanımının yaygınlaşması ile oluşmasını beklemekteyiz.

Bu çalışmaya özgü olan algoritmalar, bahsedilen kriterlere göre gelişme ve değişiklik gösterebilir, uygulama metodolojisi yazılım geliştirme süreçlerinin farklı aşamalarında kullanılabilir.

## KAYNAKÇA


Acar Ö., **"Java Tasarım Şablonları ve Yazılım Mimarileri**, Pusula Yayıncılık, Küçükçekmece/İstanbul, 2012, sy. 29-30

Big-O Cheat Sheet, Know Thy Complexities, https://www.bigocheatsheet.com/, (erişim tarihi, 18/01/2020)

Brown A. W.. "**Large Scale, Component-Based Development"**, Prentice Hall, New Jersey/USA, 2000, sy.70

Hu Z., Wang M., Hughes J., **"How Functional Programming Mattered",** National Science Review, 2015, sy. 1-2

Lathiya K., "**React Lifecycle Methods Render And ComponentDidMount"**, https://www.codingame.com/playgrounds/8747/react-lifecycle-methods-render-and-componentdidmount (erişim tarihi, 18/01/2020)

Lloyd J.W. , "**Practical Advantages of Declarative Programming"**, Joint Conference on Declarative Programming, 1994, sy 1

Osmani A., **Learning Javascript Design Patterns**, https://addyosmani.com/resources/essentialjsdesignpatterns/book/, (Erişim tarihi: 18/01/2020)

ReactJS, **"Component State",** https://reactjs.org/docs/faq-state.html, (erişim tarihi, 18/01/2020)

ReactJS, "**Components And Props",** https://reactjs.org/docs/components-and-props.html, (erişim tarihi, 18/01/2020)

Sinhal A., **"MVC, MVP, MVVM Design Pattern**", https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad , (erişim tarihi, 18/01/2020)

Suresh A. "**Intercepting Functions for Memoization**", Universite de Rennes, 2016, sy. 21

W3Path, **"React Component Lifecycle"**, https://w3path.com/react-component-lifecycle/, (erişim tarihi, 18/01/2020)

Yener M., Theedom A., "**Professional Java EE Design Patterns"**, John Wiley & Sons, Indianapolis/Indiana, 2015, sy. 184

# EKLER

## EK 1 - Benchmark.js Test Kodu

```
function benchmark() {
    var suite = new Benchmark.Suite;

    suite
        .add('Anatolia Component with Memoization', function () {
            var state1 = {
                value: Math.random()
            };

            new MemoizeComponent({
                container: "#component_1",
                state: state1,
                render: function (state) {
                    var button = Component.createElement("a", {
                        class: "btn btn-warning",
                        text: state.value
                    });

                    return button;
                }
            }).render();
        })
        .add('Anatolia Component with Virtual DOM', function () {
            var state2 = {
                value: Math.random()
            };

            new Component({
                container: "#component_2",
                state: state2,
                render: function (state) {
                    var button = Component.createElement("a", {
                        class: "btn btn-warning",
                        text: state.value
                    });

                    return button;
                }
            }).render();
        })
        .add('Vue Component with Virtual DOM', function () {
            var value = Math.random();

            Vue.component('my_component', {
                data: function () {
                    return {
                        value: value
                    }
                },
                template: '<a class="btn btn-default">{{value}}</a>'
            });

            new Vue({el: "#component_3"})
        })
        .add('React Component with Virtual DOM', function () {
            class Sample extends React.Component {
                render() {
                    return React.createElement("a", null, Math.random())
                }
            }
```

```
            ReactDOM.render(
                React.createElement(Sample, null, null),
                document.getElementById('component_4')
            );
        })
        // // add listeners
        .on('cycle', function (event) {
            console.log(String(event.target));
        })
        .on('complete', function () {
            console.log('Fastest is ' + this.filter('fastest').map('name'));
        })
        // run async
        .run({'async': true});
}

benchmark();
```

# ÖZGEÇMİŞ

7 Nisan 1991 Amasya/Merzifon doğumluyum. İlk okulun bir kısmını Amasya ‘da, orta okul, lise ve üniversite eğitimimi İstanbul‘da tamamladım. Fatih Üniversitesi Bilgisayar Mühendisliği bölümünden 2013 yılında mezun oldum.

Yedi yıldan fazla bir süredir sektörde uzman yazılım geliştirici olarak çalışmaktayım.

Evliyim.

Yabancı dilim İngilizce’dir.

**Güner Kaan ALKIM**