**Görev (1)**

Facebook, Twitter, vb. sosyal ağ platformlarına benzer, yüksek hit alması hedeflenen bir projeyi geliştirmek amacıyla bir proje mimarisi hazırlamanız gerekmektedir. Bu projede, kullanıcılar içerik paylaşmalı, etkileşimde bulunmalı ve platformu büyütmeli. Aşağıda, bu projenin sunucu mimarisi ve performans yönetimi için kullanılacak bileşenler hakkında ayrıntılar yer almaktadır.

Aşağıdaki görevleri yerine getirerek projenin sunucu mimarisi ve performans yönetimi ile ilgili bir tasarım ve açıklama sunmanız beklenmektedir:

**Sunucu Mimarisi Tasarımı:** Projenin temel sunucu mimarisi hakkında bir açıklama sunun.

Sunucular arasındaki rol ve sorumlulukları açıklayan bir diyagram oluşturun.

**Veritabanı Yönetimi:** Veri tabanı seçiminizi ve bu seçimin nedenlerini açıklayın. Veritabanı sunucusu ve veri tabanı kümesi (cluster) kurulumunu açıklayın.

**Ölçeklenebilirlik Stratejileri:** Projenin ölçeklenebilirliği nasıl yönetilecek? Yük arttıkça nasıl ölçeklendirilecek? Yük dengeleme stratejilerini açıklayın.

**Önbellekleme ve Hızlandırma:** Hangi önbellek (cache) mekanizmalarını kullanmayı düşünüyorsunuz? Sunucu ve veri tabanı erişimini hızlandırmak için hangi yöntemleri düşünüyorsunuz?

**Güvenlik Stratejileri:** Kullanıcı kimlik doğrulama ve yetkilendirme nasıl yönetilecek? Proje güvenliği için hangi önlemleri almayı düşünüyorsunuz?

**Performans İzleme ve Analiz:** Sunucu performansını izlemek ve sorunları tanımlamak için kullanılacak araçları ve yöntemleri açıklayın.

## SUNUCU MIMARISI

Web Sunucuları

Kullanıcı isteklerini alır, işler ve sonuçları kullanıcılara geri gönderecek. Bu sunucular genellikle yük dengeleyicilerle birlikte çalışmalı ve talep arttıkça ölçeklenebilir olmalıdır

Apache tercih edilebilir.

Uygulama Sunucuları

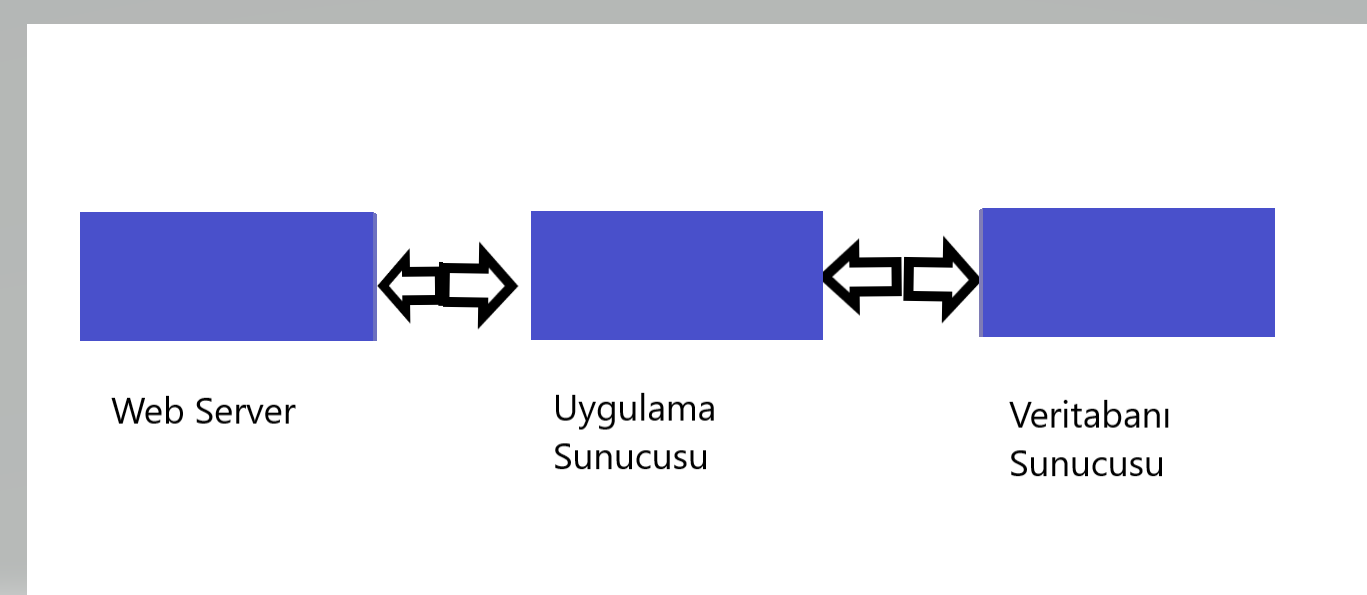
İstemci isteklerini işleyip, veritabanına erişecek. Genellikle web sunucularının arkasında bulunurlar ve uygulama kodunu çalıştırırlar.

Node.js tercih edilebilir.

Veritabanı

Ayrcıa kullanıcıların verilerini depolamak için bir veritabanına ihtiyacımız var.

MongoDB tercih edilebilir.



## Veritabanı Yönetimi

Veri Tabanı Seçimi: MongoDB

MongoDB, JSON benzeri belgelere dayalı bir veri modeline sahip. Bu da , karmaşık ve değişken veri yapılarını daha kolay ve esnek bir şekilde yönetmeyi sağlar.

MongoDB, yüksek performanslı bir veritabanı sistemidir. Sorguların hızlı ve verimli bir şekilde çalışmasını sağlar.

Geleneksel sunuculara göre ölçeklenebilirlik konusunda daha iyi olduğunu düşünüyorum. Son yıllarda community ve dökümanlarının da alternatiflerine göre bir adım önde görüyorum.

Veritabanı Sunucusu Seçimi : AWS Amazon Web Services

AWS, MongoDB'nin servisleri olan Amazon DocumentDB ve Amazon DocumentDB gibi servisleri sunar. Bu servisler, MongoDB veritabanını yönetmek için gerekli altyapıyı sağlar ve kullanıcıların MongoDB'yi kolayca AWS ortamında çalıştırmasını sağlar.

DocumentDB otomatik yedekleme , very çoğaltma ve hata toleransı gibi özelliklerle gelir.

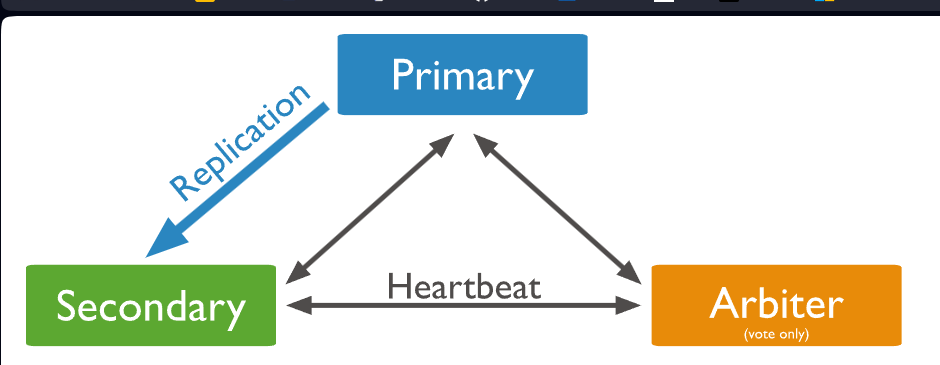
AWS’nin ayrıca dökümantasyon ve kullanıcı forumları konusunda rakiplerine nazaran daha iyi bir seçenek olduğunu düşnüüyorum

Cluster Yapısı

Primary Sunucuda yazma işlemleri gerçekleşir ve kümedeki diğer sunuculara veri çoğaltır. Ana veritabanı yazma ve okuma işlemlerini burada gerçekleşir.

İkincil Sunucu birinci sunucudan veri çoğaltıcak yani yedek gibi çalışacak. Okuma işlemleri daha çok bu sunucudan gerçekleşecek.

Son olarak Aracı Sunucu yapısını kullanacağız . Bu da aslında işlemlerin hangi sunucular arasında olacağını yönetmek için kullanılacak. Extra bir işlem yapmayacak



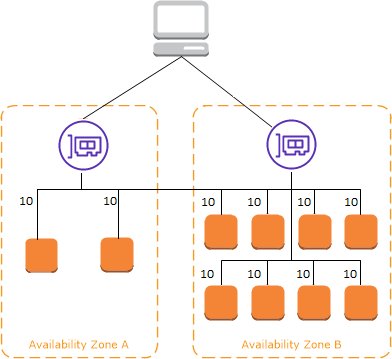
## Ölçeklenebilirlik Stratejileri

MongoDB'nin sharding özelliğini kullanarak, veriyi parçalayabilir ve farklı sunuculara dağıtarak yükü dengeleyebiliriz.

AWS Auto Scalling teknolojisini entegre ederek kullanıcı data ve işlemleri arttıkça otomatik yeni sunucu eklenmesini sağyalabiliriz.

## Yük Dengeleme Stratejileri

AWS'de Elastic Load Balancer kullanarak, gelen istekleri dağıtarak yükü dengeleyebiliriz.



MongoDB’deki Sharding özelliğindeki Shard Keyler ile yükleri dengeli bir şekilde dağıtabiliriz.

## Ön Bellekleme ve Hızlandırma

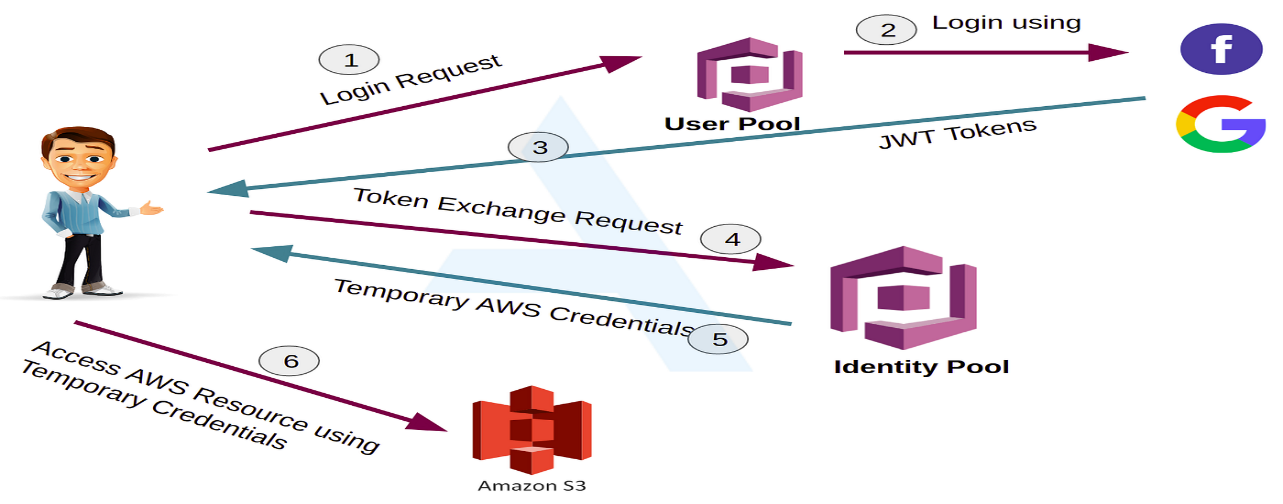
Elastic Load Balancer burada da işimizi kolaylaştıracak

Amazon’un CDN çözümü olan ClodFront kullanılarak static içeriklerin daha hızlı aktarılması sağlanır.

MongoDb sıkça erişilen bilgiler olan (nickname ,profil fotoğrafı, hastagler ) verileri tespit edip cacheleyebiliriz .

## Güvenlik Stratejileri

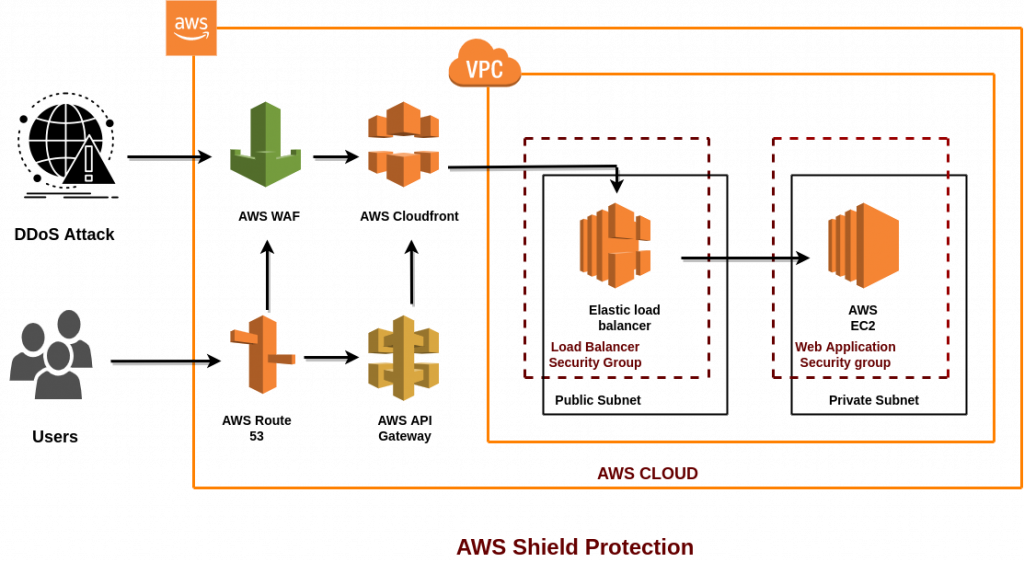
Kullanıcıların kimlik doğrulama ve yetiklendirme işleriini yönetmek için AWS Cognito kullanılabilir.



Mongo Db’de proje ekibinde yer alacak insanların yetkilerine göre veri tabanına erişimi için yetkilendirme yapılabilir.

Veritabanı ile iletişimde SSL yapısı kullanılmalı. Aksi takdirde veri tabanına gönderdiğimiz paketlere dışardan erişim sağlanabilir.

DDOS atakları için AWS Shield kullanılabilir.



Kullancııların sistemde girdiği şifrelerin kolay kırılamayan şifreler olmasını sağlamalıyız.

Ayrıca internette sıkça kullanılan ve ya sızmış şifreleri kullanıcıların almamalarını Cognito üzerinden sağlamalıyız.

Son olarak KVKK’ye uygunluğunu bir hukuk danışmanıyla birlikte test etmeliyiz.

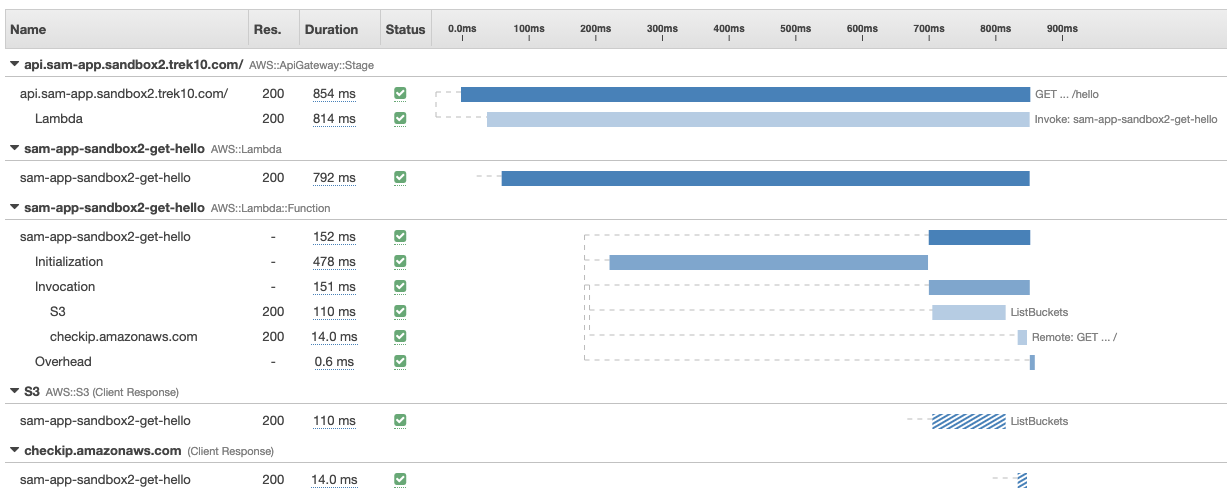
## Performans İzleme

Amazon CloudWatch ile AWS Kaynaklarının performans verilerini , ağ trafiğini , depolama kullanımını izleyebilir ve aynı zamanda sistemdeki log kayıtlarına ulaşabiliriz.

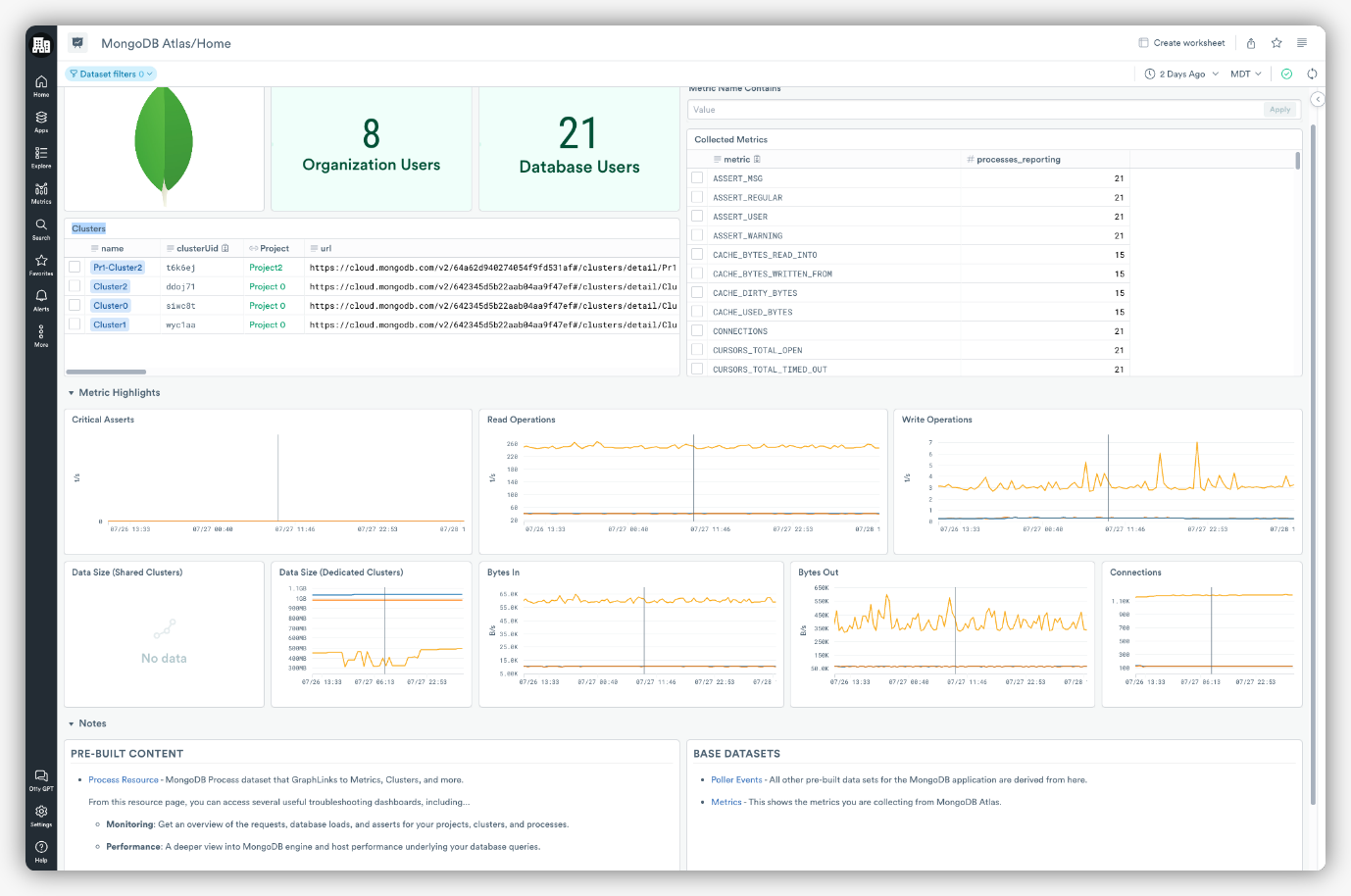


AWS Cloud Trail ile AWS servislerinin kim tarafından ne zaman ve neden kullanabildiğini inceleyebiliriz.

AWS X-Rayi ise Hizmetler arası etkileşimleri ve gecikmeleri görmek için kullanabiliriz.



Mongo Db performansını Atlas Cloud Monitoring ile monitor edebiliriz.



Mongo DBde Profiller oluşturup bazı sorguların performansını inceleyip performansı düşükse index ve ya cache çözümlerine başvurarak performansı arttırabiliriz.

**Görev (2)**

Fibonacci Dizisi ve Asal Sayılar ile ilgili bir görev aşağıda verilmiştir. Bu görev, algoritmaları

anlama ve uygulama yeteneğini değerlendirmeyi amaçlamaktadır.

Görev Açıklaması:

İlk olarak, 500.000'den küçük en büyük asal sayıyı ve 500.000'den büyük en küçük asal sayıyı

belirlemek ve bulunan iki asal sayı arasındaki farkı hesaplamak için bir Java programı yazınız.

Bu program, asal sayıları belirlerken etkili bir algoritma kullanmalıdır.

Notlar:

1. İşlem süresini minimize etmek için verimli bir algoritma kullanmanız beklenmektedir.

2. İşlem sonucunu kullanıcıya açık ve anlaşılır bir şekilde sunmanız gerekmektedir.

İkinci görevin implementasyonu aşağıdaki şekildedir :

function isPrime(number) {

    if (number <= 1) {

        return false;

    }

    for (let i = 2; i <= Math.sqrt(number); i++) {

        if (number % i === 0) {

            return false;

        }

    }

    return true;

}

function findLargestPrimeSmallerThan(limit) {

    for (let number = limit; number >= 2; number--) {

        if (isPrime(number)) {

            return number;

        }

    }

    return 2;

}

function findSmallestPrimeGreaterThan(limit) {

    for (let number = limit;; number++) {

        if (isPrime(number)) {

            return number;

        }

    }

}

function calculateDifference() {

    const smallPrineNum = findLargestPrimeSmallerThan(500000);

    const largePrimeNum = findSmallestPrimeGreaterThan(500000);

    const difference = largePrimeNum - smallPrineNum;

    console.log("500.000'den küçük en büyük asal sayı: " + smallPrineNum);

    console.log("500.000'den büyük en küçük asal sayı: " + largePrimeNum);

    console.log("İki asal sayı arasındaki fark: " + difference);

}

calculateDifference();

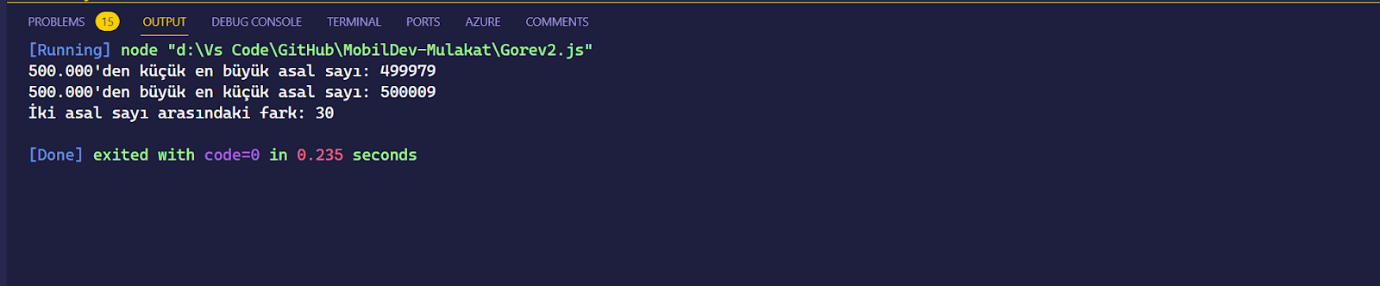
isPrime() fonksiyonu verilen girdinin asal olup olmadığını kontrol etti.

findLargestPrimeSmallerThan() fonksiyonunda parametredeki limitten düşük olan en büyük asal sayıyı döndürdük

findSmallestPrimeGreaterThan() fonksiyonunda parametredeki limitten büyük olan en küçük asal sayıyı döndürdük

Difference değişkeni ise bu iki sayının arasındaki farkı verdi.

Programın çıktısı aşağıdaki ekran görüntüsündedir :



**Görev (3)**

Basit Bir Web Uygulaması Geliştirme: Bu görevde, bir web uygulaması geliştirmeniz

beklenmektedir. Bu uygulama, kullanıcıların çeşitli bilgileri girip bu verilerin bir veritabanına

kaydedilmesini ve aynı sayfa üzerinde listelenmesini içermelidir. Görev, kullanıcı arayüzü (UI)

tasarımı, web servis entegrasyonu ve veritabanı işlemleri gibi farklı beceri alanlarını içerir.

Görev Açıklaması:

UI Tasarımı ve Form Oluşturma: Bootstrap kullanarak basit bir web sayfası tasarlayın. Sayfa,

kullanıcıların verileri girebileceği bir form içermelidir.

jQuery ve Ajax Kullanarak Web Servis İsteği: Kullanıcının formdaki verileri girdikten sonra,

jQuery ve Ajax kullanarak bu verileri bir web servisine gönderin. Web servisi, verileri kabul edip

bir RESTful API aracılığıyla sunmalıdır.

Veri Tabanına Kaydetme: Web servis, aldığı verileri bir veritabanına kaydetmelidir. Veritabanı

işlemleri için gerekli SQL sorgularını oluşturun.

Verilerin Listelenmesi: Web servis, verileri alabilir ve aynı sayfa üzerinde bir tablo şeklinde

listelenebilir. jQuery ve Ajax kullanarak verileri alın ve tablo içinde görüntüleyin.

Hata İşleme ve Geri Bildirim: Kullanıcı dostu bir şekilde hataları işleyin ve kullanıcıya geri

bildirim sağlayın.

Notlar:

1. Proje için Core Java kullanmalısınız.

2. Kullanıcı deneyimini düşünerek kullanıcı dostu bir arayüz tasarımı yapmalısınız.

3. Bootstrap, jQuery ve Ajax gibi önemli teknolojileri etkili bir şekilde kullanmalısınız.

4. Proje, veritabanıyla etkileşimli bir web uygulaması olmalıdır.

Ben bu soru için SQLite veri tabanını ve nodejs frameworkünü kullandım yani JavaScript kullanarak uygulamayı geliştirdim.

Öncelikle terminale gelip sırasıyla aşağıdaki komutları yazdım :

1)mkdir web\_uygulamasi

2)cd web\_uygulamasi

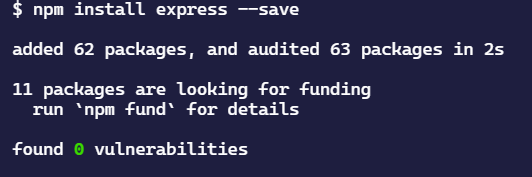
3)touch app.js index.html style.css

Bu komutlar projede dosya yapısını oluşturmamızı sağladı.

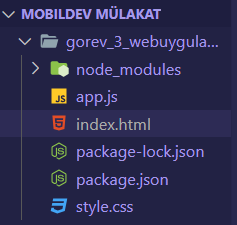
Ardından "npm init -y" komutu ile npm paketini kurdum

Bundan sonra da expressjs modülünü yüklüyorum :

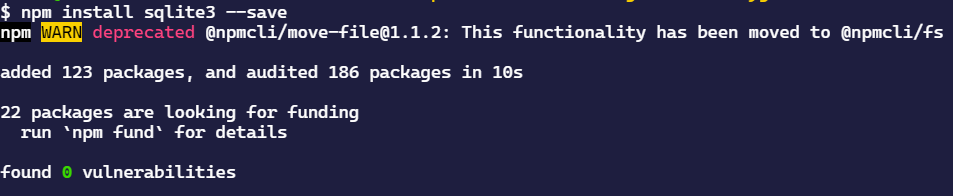
npm install express --save



Paketler aşağıdaki ekran görüntüsündeki gibi projemin içine yüklendi :



Ardından veri tabanı bağlantımı aktifleştiriyorum bunun için “terminalde npm install sqlite3 “--save komutunu yazdım.



Şimdi arayüz tasarımına geldik. HTML5 ,CSS ve jQuery kullanmayı seçtik. Uygulamamın frontend iskeletini aşağıdaki gibi oluşturuyorum:

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Web Uygulaması</title>

    <link *rel*="stylesheet" *href*="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

    <link *rel*="stylesheet" *href*="style.css">

    <script *src*="https://code.jquery.com/jquery-3.5.1.min.js"></script>

</head>

<body>

    <div *class*="container">

        <h1>Web Uygulaması</h1>

        <form *id*="myForm">

            <div *class*="form-group">

                <label *for*="name">Ad:</label>

                <input *type*="text" *class*="form-control" *id*="name" *name*="name">

            </div>

            <div *class*="form-group">

                <label *for*="email">E-posta:</label>

                <input *type*="email" *class*="form-control" *id*="email" *name*="email">

            </div>

            <button *type*="submit" *class*="btn btn-primary">Gönder</button>

        </form>

        <div *id*="feedback"></div>

        <table *class*="table mt-4">

            <thead>

                <tr>

                    <th>Ad</th>

                    <th>E-posta</th>

                </tr>

            </thead>

            <tbody *id*="data">

            </tbody>

        </table>

    </div>

    <script *src*="app.js"></script>

</body>

</html>

Bu şekilde basit bir HTML ve  Boostrap aracılığıyla form oluşturdum sırada veritabanı ve webservis işlemleri var.Bunun için uygulama portunu 3000 olarak belirledim. Ana dizine ('/') yapılan isteklerde, index.html dosyası sunuluyor:

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const app = express();

const port = 3000;

app.use(express.urlencoded({ extended: true }));

app.use(express.static('public'));

let db = new sqlite3.Database(':memory:');

db.serialize(() => {

    db.run("CREATE TABLE users (name TEXT, email TEXT)");

});

app.get('/', (req, res) => {

    res.sendFile(\_\_dirname + '/index.html');

});

app.post('/submit', (req, res) => {

    const { name, email } = req.body;

    db.run(`INSERT INTO users (name, email) VALUES (?, ?)`, [name, email], (err) => {

        if (err) {

            res.status(500).send('Hata: Veritabanına kaydedilemedi.');

        } else {

            res.send('Veri başarıyla eklendi.');

        }

    });

});

app.get('/data', (req, res) => {

    db.all("SELECT \* FROM users", (err, rows) => {

        if (err) {

            res.status(500).send('Hata: Veriler alınamadı.');

        } else {

            res.json(rows);

        }

    });

});

app.listen(port, () => {

    console.log(`Uygulama ${port} portunda çalışıyor.`);

});

Ardından ajax isteği ile form işlerini yöneteceğimiz script’i yazıyoruz. Bunun için index.html dosyasında script etiketi oluşturdum :

  <script>$(document).ready(function() {

        $('#myForm').submit(function(e) {

            e.preventDefault();

            var email = $('#email').val();

            if (!isValidEmail(email)) {

                $('#feedback').text('Geçersiz e-posta adresi!');

                return;

            }

            $.ajax({

                type: 'POST',

                url: '/submit',

                data: $(this).serialize(),

                success: function(response) {

                    $('#feedback').text(response);

                    $('#myForm')[0].reset();

                    fetchData();

                },

                error: function(err) {

                    $('#feedback').text('Hata: ' + err.responseText);

                }

            });

        });

        function fetchData() {

            $.get('/data', function(data) {

                $('#data').empty();

                data.forEach(function(row) {

                    $('#data').append(`<tr><td>${row.name}</td><td>${row.email}</td></tr>`);

                });

            });

        }

        fetchData();

        function isValidEmail(email) {

            var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

            return emailRegex.test(email);

        }

    });

    </script>

jQuery ve AJAX teknolojileri kullandığım bu fonksiyonlarrda iki koşul kontrolü yaptım

  function isValidEmail(email) {

            var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

            return emailRegex.test(email);

        }

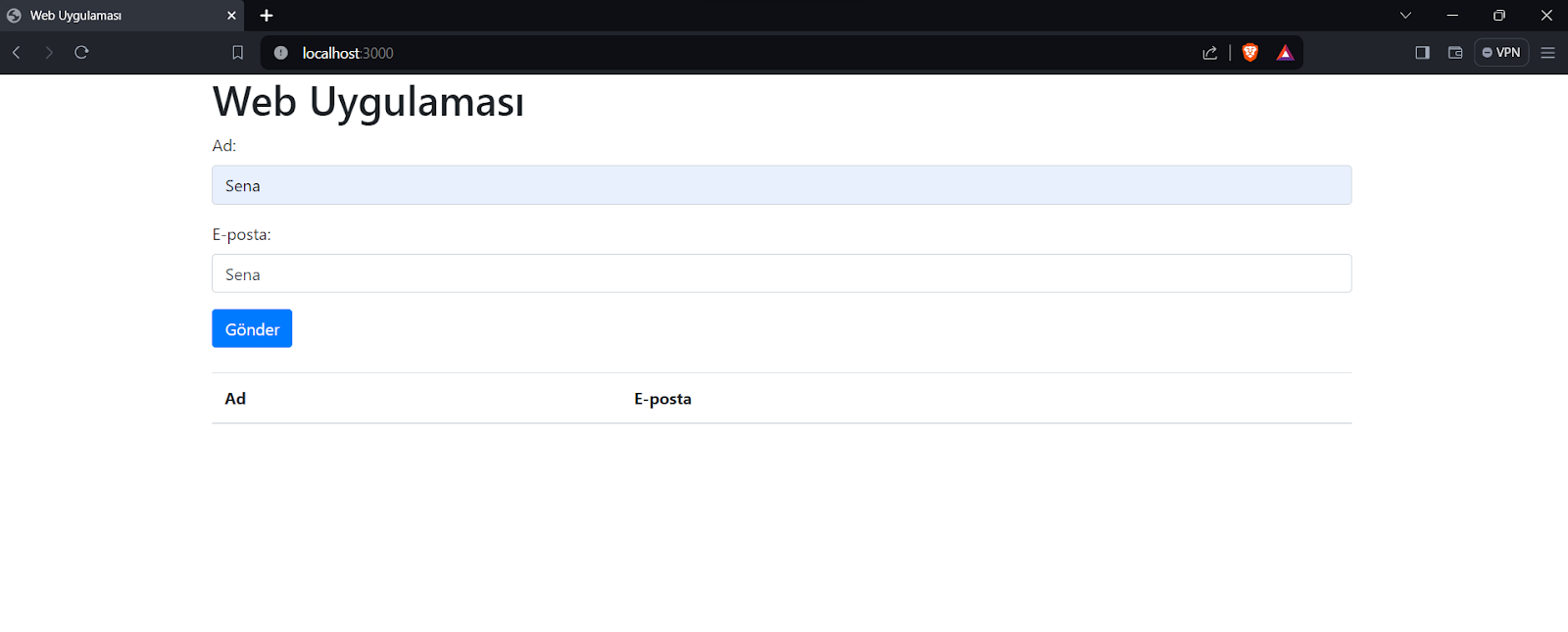
    });

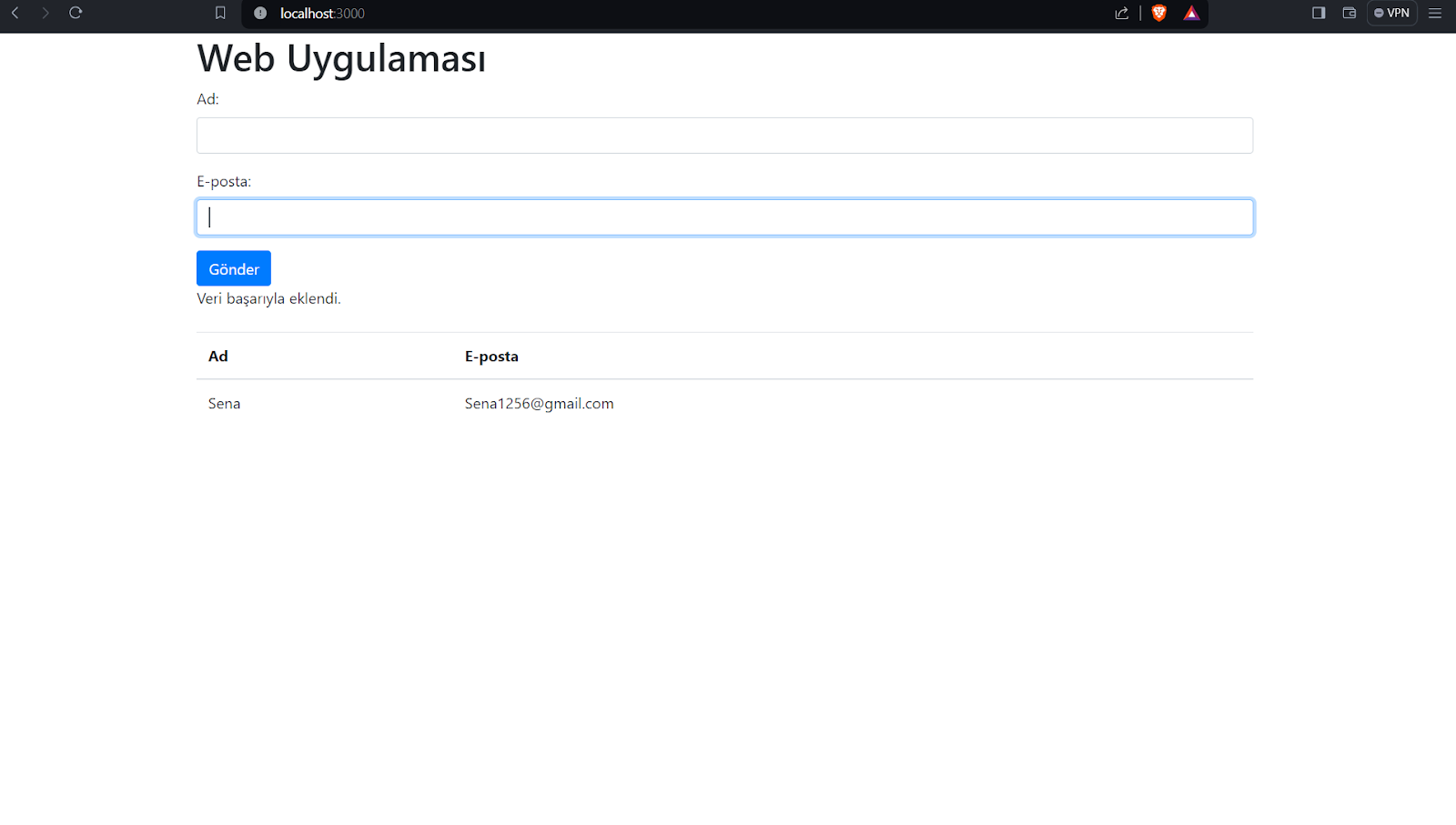
Bu fonksiyonda biçimsel olarak mail formatını kontrol ettik.

$('#myForm').submit() fonksiyonu, form gönderilme olayını tetikliyor.

fetchData() işlevi, sayfa yüklendiğinde otomatik olarak çağrılır,

böylece sayfa yüklendiğinde veriler kaydolan database tablosuna hemen ekleniyor.





**Görev (4)**

Veri Normalizasyonu ve Algoritma Tasarımı: Bu görevde, rasgele sayılar içeren bir dizinin

verilerini normalleştiren ve her yeni eleman eklediğinde dizinin minimum ve maksimum

değerlerini güncelleyerek normalizasyon hesaplayan bir algoritma tasarlamanız ve kodlamayı

sunmanız beklenmektedir.

Görev Açıklaması:

1 ile 10.000 arası sayılardan rastgele sayı üreten ve farklı elemanlardan oluşan 500 elemanlı

bir dizi oluşturun. Diziye her yeni eleman eklediğinizde, minimum ve maksimum değerleri

güncelleyen ve verileri aşağıda verilen normalizasyon formülüne göre normalleştiren bir

algoritma tasarlayın ve uygulayın.

N = (xi – xmin) / (xmax – xmin)

Örnek Çıktı: sadece 3 veri için

Veri seti (tek elemanlı) : {110}

Normalizasyon : [0]

Veri Seti (iki elemanlı) : {1382, 110}

Normalizasyon : [1.0, 0.0]

Veri Seti (üç elemanlı) : {1382, 7357, 110}

Normalizasyon : [0.1755209052021526, 1.0, 0.0]

Notlar:

Diziye her eleman eklediğinizde minimum ve maksimum değerlerini güncellemelisiniz.

Normalizasyon hesaplama adımlarını anlaşılır bir şekilde ekrana yazdırmalısınız.

Dizinin başlangıçta nasıl oluşturulacağını ve nasıl güncelleneceğini düşünmelisiniz.

Dördüncü görevin implementasyonu aşağıdaki şekildedir :

*//diziler , dizi boyutu gibi değişkenler burada tanımlanıyor.*

let data = [];

const dataSize = 500;

const minValue = 1;

const maxValue = 10000;

for (let i = 0; i < dataSize; i++) {

    data.push(Math.floor(Math.random() \* (maxValue - minValue + 1)) + minValue);

}

*// bu kısımda normalizasyonu hesaplıyoruz*

function normalizeData(newElement) {

*// Yeni eleman ekleme*

    data.push(newElement);

    const currentMin = Math.min(...data);

    const currentMax = Math.max(...data);

*//normalizasyon*

    const normalizedData = data.map(x => (x - currentMin) / (currentMax - currentMin));

*//fonksiyon çıktısı*

    return normalizedData;

}

*//test senaryosu*

const newElement = Math.floor(Math.random() \* (maxValue - minValue + 1)) + minValue;

const normalizedData = normalizeData(newElement);

console.log("Dizi uzunluğu:", normalizedData.length);

console.log("Normalleştirilmiş veriler:", normalizedData);

Belirlenen limitleri yukarıda tanımladım. Bütün dizideki özellikler yukarıdaki değerlere göre şekillenecek. Bir fonksiyon daha oluşturup onun içinde de bu parametreelri girebilirdik ancak ben bu şekilde yapmayı seçtim.

For döngüsünün içinde rastgele değerleri dizinin içine attım.

Ardından verinin içinde max ve min değerleri bularak soruda verilen normalizasyon formünü bu değerler üstünde uyguladım.

Ardından testi yazdım ve çıktıyı ekranda şu şekilde sonuçlandırdım :



Görev 5 )

Aşağıda verilen veritabanı tabloları kullanılarak, her departmandaki çalışan sayısını hesaplayan ve sonucu departman adıyla birlikte listeleyen bir SQL sorgusu yazmanız gerekmektedir.

**Veritabanı Tabloları:**

"Emp" tablosu (Alanlar: Id, Name, DeptId) – 10 kayıt içerir.

"Dept" tablosu (Alanlar: Id, Name) – 5 kayıt içerir.

**Görev Açıklaması:**

1. SQL sorgusu yazarak, her departmandaki çalışan sayısını hesaplayın.
2. Sonucu departman adıyla birlikte listeleyin.
3. Sonuç olarak kaç satır listeleneceğini açıklayın.

SELECT dpt.Name AS Department, COUNT(e.Id) AS EmployeeCount

FROM Dept dpt

INNER JOIN Emp e ON dpt.Id = e.DeptId

GROUP BY dpt.Name

**Bonus Görev :**

Mobildev XML SMS API metotlarından SMSMultisender metodunu kendi geliştirmekte olduğunuz bir uygulama içerisine kullanmak için bir kütüphane oluşturunuz. Not: user / pass / alfanumerik gibi bilgiler, bir ayar dosyasından alınacak şekilde ayarlanmalıdır.

<https://postman.mobildev.com/?version=latest#e7eeba09-7b99-466a-8fb2-95972d5c2386>

Bonus görev ile ilgili node.js’de ufak bir kütüphane geliştirdim.

Önce dosya yapısını oluşturdum , bunun için sırasıyla aşağıdaki komutları giriyorum:

1)mkdir mobildev-sms-api

2)cd mobildev-sms-api

3)npm init -y (mobildev-sms-api klasörünün içine npm paketlerini yüklediğimiz kısım)

4) mkdir lib

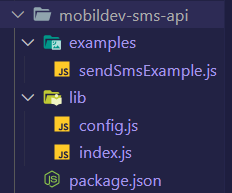
5) mkdir examples

6) touch lib/config.js

7) touch lib/index.js

8) touch examples/sendSmsExample.js

Bu şekilde dosya yapısını oluşturuyorum , aşağıdaki gibi:



İkinci adımda ayar dosyamızı yazalım :

module.exports = {

    username: 'api\_key',

    password: 'api\_password',

    alphanumeric: 'your\_alphanumeric\_sender\_name'

};

Görevde belirtildiği üzere üç parametreyi export ettik.

Şimdi kütüphanemizin bulunduğu kısmı yazıyoruz:

const axios = require('axios');

const config = require('./config');

const sendSms = *async* (numbers, message) => {

    try {

        const xmlData = `

            <MainmsgBody>

                <UserName>${config.username}</UserName>

                <PassWord>${config.password}</PassWord>

                <Action>1</Action>

                <Messages>

                    <Message>

                        <Mesgbody>${message}</Mesgbody>

                        <Number>${numbers}</Number>

                    </Message>

                </Messages>

                <AccountId>1</AccountId>

                <Originator>${config.alphanumeric}</Originator>

                <SDate></SDate>

                <EDate></EDate>

                <Encoding>1</Encoding>

                <MessageType>N</MessageType>

                <RecipientType></RecipientType>

            </MainmsgBody>

        `;

        const response = await axios.request({

            url: 'https://xmlapi.mobildev.com',

            method: 'post',

            headers: { 'Content-Type': 'application/xml' },

            data: xmlData

        });

        return response.data;

    } catch (error) {

        throw new Error(error.response.data);

    }

};

module.exports = { sendSms };

HTTP işlemlerini kolay bir şekilde yapabilmemizi sağladığı için axios kütüphanesini seçtim.

API referansında belirtildiği üzere fonksiyon içerisinde xml verisini temsil eden bir değişken tanımladım ve fonksiyon parametrelerindeki numara ve mesajı xml’in içine temsil edecek biçimde gömdüm. Sadece fonksiyon parametreleri için değil , config dosyasındaki değişkenleri de xml verisi içine dahil etmeyi ihmal etmedim.

Xml verisini tanımlamamın ardından axios’un request metodunu istek atılacak link , metod , header parametreleriyle birlikte isteği yolluyorum.

Ve bu işlem sonucundan gelen cevabı try catch döngüsü olarak ayırdım.

Son işlem olarak test senaryosunun olduğu sendSmsExample.js dosyasını yazıyorum:

const { sendSms } = require('../lib');

(*async* () => {

    try {

        const response = await sendSms(['3525334265'], 'Test message from Mobildev SMS API');

        console.log(response);

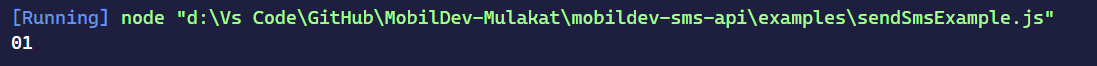
    } catch (error) {

        console.error(error.message);

    }

})();

İndex.js’de export ettiğimiz şeyi import ederek fonksiyonu değişkene atıyorum ve yazdırıyorum :



id çıktısı ekrandaki gibidir.