

BURSA TEKNİK ÜNİVERSİTESİ

**Mühendislik ve Doğa Bilimleri Fakültesi – Bilgisayar
Mühendisliği Bölümü**



BLM0470 NodeJS İle Web Programlama

2023-2024 Bahar Dönemi

1.Hafta Raporu

Berkan SERBES - 22360859353

NodeJS Tarihçesi

NodeJS, Ryan Dahl tarafından 2009 yılında geliştirilmeye başlanmış bir JavaScript çalışma zamanı ortamıdır. Geliştirilme süreci, web uygulamalarının daha verimli ve ölçeklenebilir hale getirilmesine yönelik ihtiyaca cevap vermek üzere ortaya çıkmıştır.

Geleneksel web sunucuları, her istek için ayrı bir işlem oluşturarak çoklu kullanıcı taleplerini işlerken, NodeJS, olay tabanlı(event-driven) ve asenkron bir yapıya sahiptir. Bu da tek bir işlemde birden fazla kullanıcının talebini işleyebilme yeteneği sağlar. Bu özellik, ağ tabanlı uygulamaların performansını artırır ve daha iyi ölçeklenebilirlik sunar.

NodeJS'in temeli, V8 JavaScript motoruna dayanır. V8 motoru, C/C++ ile geliştirilmiş Google Chrome'un içinde bulunan açık kaynaklı bir JavaScript motorudur ve JavaScript komutlarını makine diline çevirmek için kullanılan bir ara yazılımdır. Ryan Dahl, NodeJS'i oluştururken, bu güçlü motoru sunucu tarafı uygulamaları için kullanmaya karar verdi.

Günümüzde, NodeJS, web geliştirme alanında en popüler ve güçlü araçlardan biri olarak kabul edilmektedir.

NodeJS'in Popülerliğinin Sebepleri

NodeJS popüler olmasının sebebi hızlı ve performanslı olmasının yanında JavaScript komutlarının esnek oluşu, komutların bloklanmadan işlenmesi ve olay tabanlı çalışması ayrıca diğer sunucu tarafı çalışan programlama dilleri gibi ek bir web sunucusuna (Apache HTTP, IIS, Nginx vb.) ihtiyaç duymamasıdır.

Diğer sunucu tarafı çalışan programlama dilleri (PHP, ASP.NET vb.) ile yazılmış uygulamalar web sunucu (Apache HTTP, IIS, Nginx vb.) denilen istemci ve sunucu arasında bağlantıyı kuran ek yazılımlara ihtiyaç duyar.

Web sunucusuna gelen istekler daha sonra sunucu tarafı çalışan programlama dillerine iletilir ve istenilen komutlar çalıştırılır.

NodeJS içerisinde gelen çekirdek modülleri sayesinde ek bir web sunucusuna ihtiyaç duymadan komutların çalıştırılmasını sağlar.

Diğer sunucu tarafı çalışan programlama dillerine herhangi bir kullanıcı istekte bulunduğu anda sunucu sadece o isteğe cevap verir ve diğer istekler kuyruğa alınır.

Bir isteğin uzun sürmesi diğer kullanıcıları etkiler ancak NodeJS komutları bloklamadan işlediğinden işlemi uzun süren komut sistemi yavaşlatmaz ve NodeJS diğer kullanıcılara da cevap verir.

Bununla ilgili en güzel örnek yemek sipariş sistemi diyebiliriz.

Klasik sunucu tarafı programlama dilleri bir yemek siparişi geldiğinde sırada duran diğer müşteriler siparişin hazırlanmasını bekler.

NodeJS ise herhangi bir yemek siparişi geldiğinde siparişi arka tarafa bildirir ve not alır ve daha sonra sıradaki müşterinin siparişini alır.

Verilen yemek siparişlerinin hangisi daha önce hazırlanırsa o yemek siparişine cevap verir.

Böylece müşteri daha önce ve hızlı hazırlanacak bir yemek için sırada fazladan beklemesiz.

Bloklamadan işlem yapılmasına non-blocking I/O, ölçeklenebilir uygulama geliştirme gibi çeşitli isimler verilmiştir.

Bu yapı sayesinde anlık mesajlaşma, oyun sistemleri gibi gerçek zamanlı uygulamalar kolaylıkla ve daha az maliyetle yapılır.

NodeJS Kurulumu

NodeJS'i bilgisayarınıza kurmak oldukça basittir ve çeşitli işletim sistemlerinde desteklenir. İşte NodeJS'in yaygın işletim sistemlerine nasıl kurulacağına dair adımlar:

1. Windows İşletim Sistemi İçin:

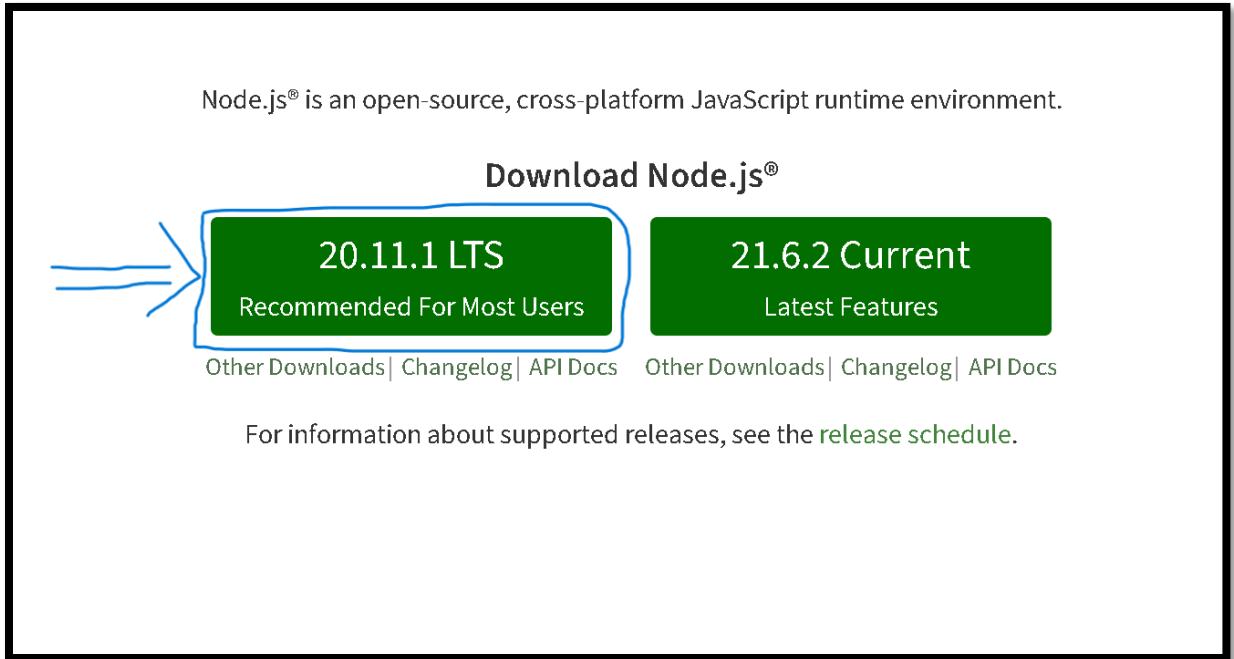
NodeJS web sitesine (<https://nodejs.org/en>) gidin.

Ana sayfada, "LTS" olarak etiketlenmiş olan "Recommended for most users" seçeneğini indirin.

İndirilen kurulum dosyasını çift tıklayarak çalıştırın.

Kurulum sihirbazını takip ederek varsayılan ayarları kabul edin.

Kurulum tamamlandığında, NodeJS ve NPM (Node Package Manager) bilgisayarınıza başarıyla yüklenecektir.



Windows x64 tabanlı sistem için görselde belirtilen 20.11.1 LTS versiyonu çoğu kullanıcı için tercih edilen daha sorunsuz ve stabil versiyonudur. 21.6.2 Current ise mevcut en güncel versiyondur. İşletim sistemi farklı olan kullanıcılar için her iki versiyon için de Other Downloads seçeneğiyle diğer işletim sistemleri ve sürümleri için kurulum yapabileceğiniz indirme bağlantılarına ulaşabiliyorsunuz.

Kurulumun başarılı olup olmadığını anlamak için komut satırına aşağıdaki komutları yazarak anlayabilirsiniz. Eğer versiyon bilgisi yüklediğiniz versiyon ile aynıysa kurulumunuz başarıyla gerçekleşmiş demektir.

```
Komut İstemi
Microsoft Windows [Version 10.0.22621.3155]
(c) Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\berka>node -v
v20.11.1

C:\Users\berka>node --version
v20.11.1
```

2. Mac İşletim Sistemi İçin:

Mac OS işletim sistemi NodeJS kurulumu Windows işletim sistemine benzer şekilde gerekli olan kurulum dosyası indirilerek yapılır. İndirmenin başarılı olup olmadığını yine **node -v** veya **node --version** komutlarını kullanarak anlayabilirsiniz.

3. Linux Tabanlı İşletim Sistemleri için:

Linux'ta NodeJS kurulumu genellikle paket yöneticisi aracılığıyla yapılır. Ubuntu veya Debian gibi Debian tabanlı dağıtımlarda aşağıdaki komutları kullanarak kurulumu gerçekleştirebilirsiniz:

```
sudo apt update
sudo apt install nodejs
sudo apt install npm
```

Kurulum tamamlandıktan sonra, terminal veya komut istemcisinde **node -v** ve **npm -v** komutlarını kullanarak NodeJS ve NPM'in sırasıyla yüklü olup olmadığını kontrol edebilirsiniz. Bu komutlar, yüklü sürümlerin bilgilerini gösterecektir.

JavaScript ve NodeJS Arasındaki Farklar

JavaScript ve NodeJS, her ikisi de JavaScript tabanlı olmalarına rağmen, farklı kullanım alanlarına ve çalışma ortamlarına sahip olan teknolojilerdir.

JavaScript, ilk olarak web tarayıcılarında istemci tarafı betik dili olarak kullanılmak üzere geliştirilmiştir. Genellikle, web sayfalarının etkileşimli ve dinamik olmasını sağlamak için kullanılır. İstemci tarafı olarak bilinen bu kullanım, web tarayıcılarında kullanıcı

etkileşimlerini, form doğrulamalarını, animasyonları ve sayfa içi dinamik içerikleri yönetmek için kullanılır. JavaScript, tarayıcı tarafında çalışır ve tarayıcı tarafından yorumlanır.

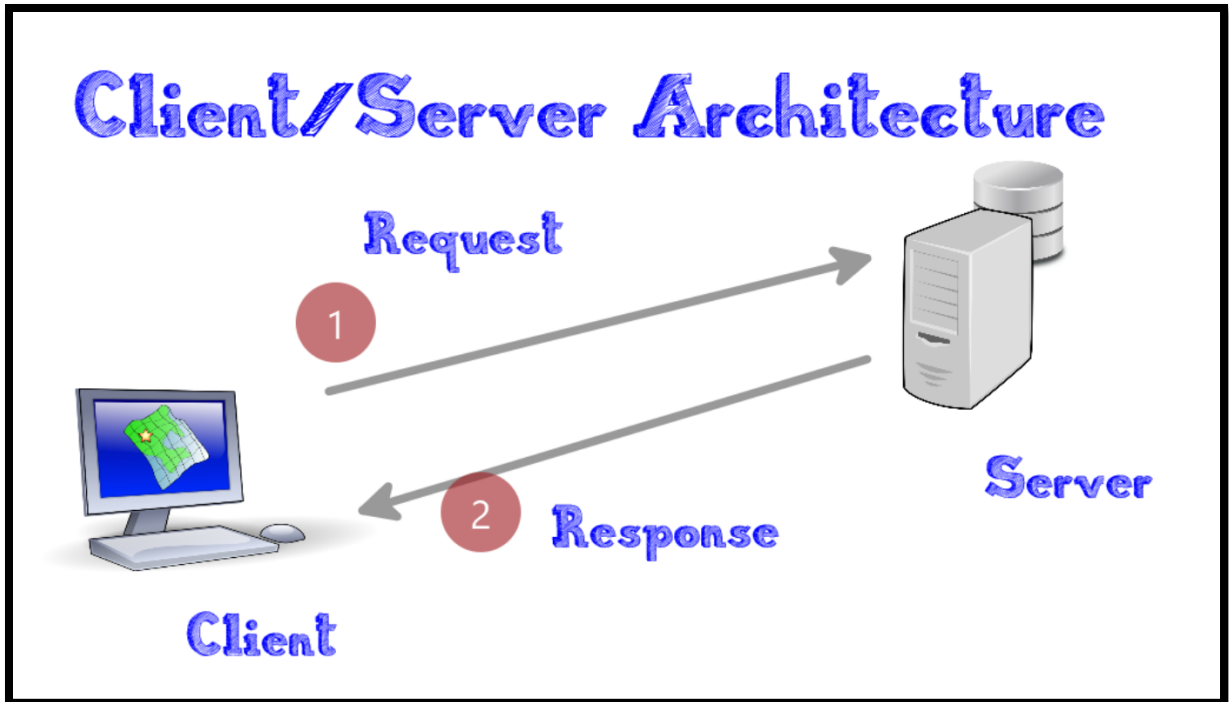
NodeJS, JavaScript'in sadece istemci tarafında değil, aynı zamanda sunucu tarafında da çalışabilmesini sağlar. Bu, JavaScript'in web sunucuları, API sunucuları ve diğer sunucu taraflı uygulamalarda kullanılmasını mümkün kılar. NodeJS, Google'ın Chrome tarayıcısında kullanılan V8 JavaScript motorunu temel alır ve sunucu tarafında JavaScript kodunu hızlı bir şekilde yorumlar.

Bu iki teknoloji arasındaki önemli farklardan biri de modül sistemidir. JavaScript'te, tarayıcı tarafında kullanılan modül sistemi genellikle ECMAScript 6 (ES6) modül sistemi olarak bilinirken, NodeJS'te ise CommonJS modül sistemi yaygın olarak kullanılır. Bu farklı modül sistemleri, her iki ortamda da farklı şekillerde modül ve paket yönetimini destekler.

Son olarak, API'ler de JavaScript ve NodeJS arasında farklılık gösterir. JavaScript'in tarayıcı tarafında, DOM (Document Object Model) API'leri gibi web tarayıcısının sunduğu API'ler kullanılırken, NodeJS'in sunucu tarafında, dosya sistemi API'leri, ağ API'leri ve diğer sunucu taraflı API'ler kullanılır.

İstemci - Sunucu Mimarisi (Client – Server Architecture)

Client - Server mimarisi, bilgisayar ağlarında bilgi alışverişini sağlayan temel bir yapıdır. Bu mimari, iki ana bileşenden oluşur: istemci (client) ve sunucu (server). İstemci, bilgiyi talep eden ve sunucuya istekte bulunan cihaz veya yazılımdır. Sunucu ise, istemcilerden gelen talepleri işleyen ve istemcilere yanıt veren cihaz veya yazılımdır. İstemci ve sunucu arasındaki etkileşim, belirli bir protokol üzerinden gerçekleşir.



İstemci (Client):

İstemci, kullanıcı tarafındaki cihaz veya yazılımdır. Web tarayıcıları, akıllı telefon uygulamaları veya masaüstü uygulamaları gibi çeşitli platformlarda bulunabilir. İstemci, sunucuya belirli bir hizmet veya bilgi talebinde bulunur. Örneğin, bir web tarayıcısı, bir web sitesinin içeriğini sunucudan talep eder ve sunucudan gelen yanıtı görüntüler.

İstemci, sunucudan gelen verileri kullanıcı arayüzünde göstermek veya işlemek için gereken görevleri yerine getirir. Kullanıcı etkileşimlerini algılar ve bu etkileşimlere yanıt olarak sunucuya istekler gönderebilir.

Sunucu (Server):

Sunucu, istemcilerden gelen istekleri işleyen ve istemcilere yanıt veren bir bilgisayar veya yazılımdır. Sunucu, istemcinin taleplerini kabul eder, gerekli işlemleri yapar ve istemciye yanıt olarak veri gönderir.

Sunucular, genellikle yüksek performans, güvenlik ve sürekli erişilebilirlik gerektiren uygulamalar için özel olarak yapılandırılır. Web sunucuları, e-posta sunucuları, veritabanı sunucuları gibi çeşitli türlerde sunucular bulunabilir.

İstemci - Sunucu Etkileşimi:

İstemci ve sunucu arasındaki etkileşim, belirli bir iletişim protokolü üzerinden gerçekleşir. İstemci, sunucuya istekte bulunmak için belirli bir protokolü kullanır ve sunucudan gelen yanıtları alır.

En yaygın kullanılan iletişim protokolleri arasında HTTP (Hypertext Transfer Protocol) ve HTTPS (HTTP Secure) bulunur. Web tabanlı uygulamalar genellikle bu protokoller üzerinden çalışır ve istemci, web tarayıcısı üzerinden sunucuya HTTP istekleri gönderir.

Client - Server mimarisi, dağıtık sistemlerin geliştirilmesi ve yönetilmesi için temel bir modeldir. Bu mimari, güvenilir, ölçeklenebilir ve verimli uygulamaların geliştirilmesine imkan tanır. İstemci ve sunucu arasındaki etkileşim, modern internet tabanlı uygulamaların temelini oluşturur ve birçok alanda kullanılır, örneğin web siteleri, e-posta hizmetleri, bulut depolama ve daha fazlası.

Editör ve IDE Nedir?

IDE (Integrated Development Environment - Entegre Geliştirme Ortamı) ve editör, yazılım geliştirme sürecinde kullanılan iki farklı araçtır. Her ikisi de kod yazma, düzenleme ve geliştirme için kullanılabilir, ancak farklı özelliklere ve işlevselliklere sahiptirler. Bir IDE, kod geliştirmeyi kolaylaştıran ve hızlandıran bir yazılım aracıdır. Kod yazma, hata ayıklama, derleme, sürüm kontrolü ve diğer geliştirme işlemlerini tek bir arayüzde birleştirir. Bir IDE, genellikle projeleri düzenlemek, oluşturmak ve yönetmek için gelişmiş araçlar sağlar. Proje dosyalarını ve klasörlerini düzenlemek, bağımlılıkları yönetmek gibi işlevlere sahiptirler. Ayrıca, IDE'ler genellikle entegre hata ayıklama araçlarına sahiptir ve yazılan kodu derleme ve yürütme işlemlerini kolaylaştıran entegre araçlar sunabilirler. Öte yandan, bir metin editörü temel olarak kod yazma ve düzenleme işlevlerini sağlayan bir yazılım aracıdır. Editörler, daha hafif ve esnek araçlardır ve genellikle daha hızlı başlatılır ve daha az sistem kaynağı kullanır. Temel olarak kod odaklıdır ve genellikle derleme, hata ayıklama gibi daha gelişmiş geliştirme işlevlerine sahip değildirler. Editörler, genellikle eklenti sistemlerine sahiptir, bu

sayede kullanıcılar, ihtiyaçlarına uygun eklentileri yükleyerek editörün yeteneklerini genişletebilirler.

IDE (Entegre Geliştirme Ortamı) Örnekleri

Visual Studio: Microsoft tarafından geliştirilen kapsamlı bir IDE'dir. C#, .NET, Java, Python gibi çeşitli programlama dilleri için geliştirme ortamı sağlar.

Eclipse: Açık kaynaklı bir IDE olan Eclipse, Java geliştirme için çok popülerdir. Ancak, çeşitli eklentiler aracılığıyla diğer diller için de genişletilebilir.

IntelliJ IDEA: JetBrains tarafından geliştirilen IntelliJ IDEA, Java, Kotlin, Groovy gibi diller için geliştirme yapmak için kullanılan güçlü bir IDE'dir.

PyCharm: JetBrains tarafından geliştirilen Python geliştirme için özel olarak tasarlanmış bir IDE'dir. Python projelerini geliştirmek için kapsamlı bir araç seti sunar.

Editör Örnekleri

Visual Studio Code: Microsoft tarafından geliştirilen, hafif ancak güçlü bir metin editörüdür. Geniş eklenti desteği ve çoklu dil desteği ile öne çıkar. Hem küçük düzeltmeler yapmak için hem de büyük projeler geliştirmek için kullanılabilir.

Sublime Text: Hızlı ve esnek bir metin editörü olan Sublime Text, çeşitli programlama dilleri için eklenti desteği sunar. Hafif olması ve hızlı başlatılmasıyla bilinir.

Atom: GitHub tarafından geliştirilen Atom, açık kaynaklı bir metin editörüdür. Geniş eklenti desteği ve kolay özelleştirme seçenekleri sunar.

Notepad++: Windows için popüler bir metin editörü olan Notepad++, hafif olması ve kullanımının kolaylığı ile bilinir. Temel kod düzenleme işlevleri için idealdir.

Sonuç olarak, IDE'ler genellikle büyük ve karmaşık projeler için tercih edilirken, editörler daha basit projeler veya hızlı kod düzenlemeleri için daha yaygın olarak kullanılır.

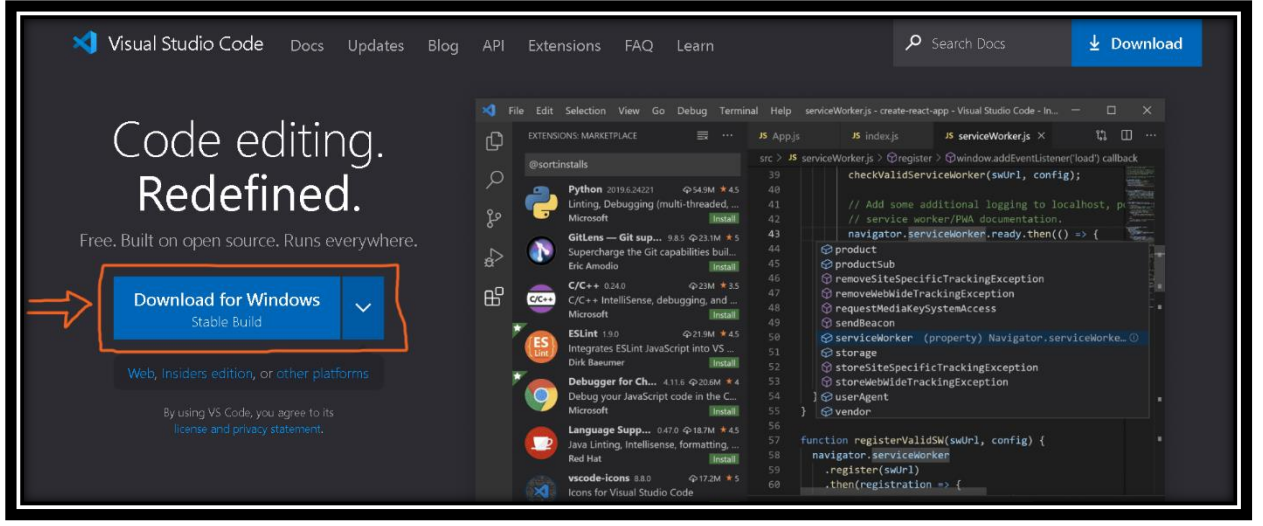
Tüm bu araçlar, NodeJS geliştirme sürecini desteklemek için çeşitli özelliklere ve esnekliklere sahiptir. Hangi editörün veya IDE'nin kullanılacağı, kişisel tercihler, proje gereksinimleri ve geliştirme ortamıyla ilgili faktörlere bağlı olarak değişebilir.

Bu rapor boyunca, NodeJS'in detaylı bir şekilde ele alınması ve geliştirme sürecinin açıklanması için Visual Studio Code'un tercih edilerek, bu editörün özelliklerine ve NodeJS ile uyumlu çalışma biçimine odaklanılacaktır. Siz derseniz istediğiniz herhangi bir editör ya da IDE'yi kullanabilirsiniz.

Visual Studio Code Kurulumu

Windows için Kurulum:

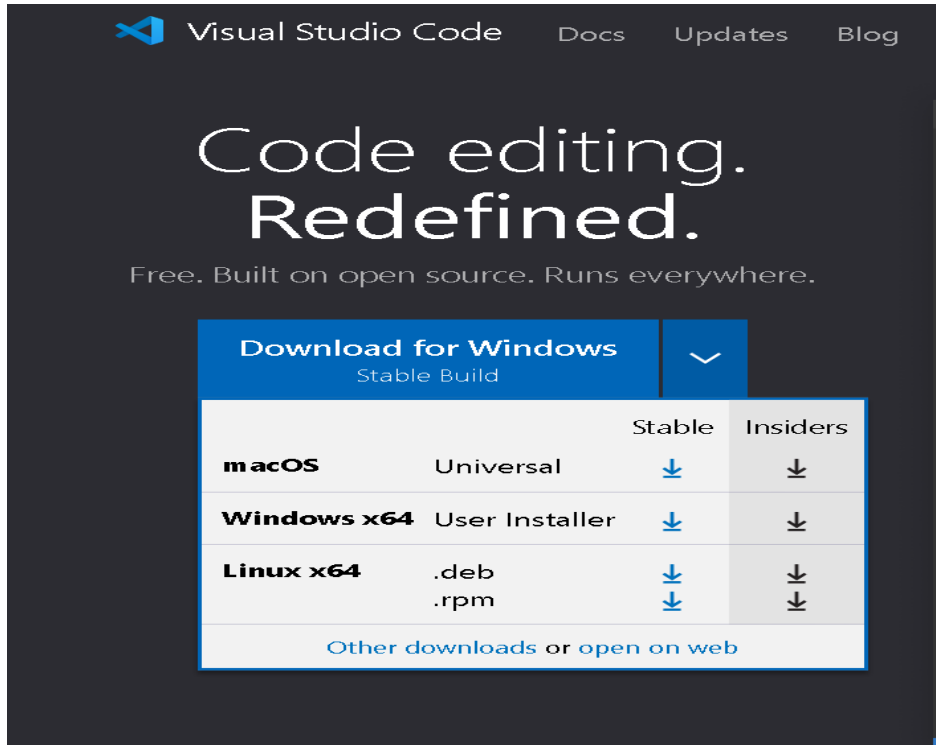
1. İlk adım olarak, Visual Studio Code'un resmi web sitesini ziyaret edin:
<https://code.visualstudio.com/>



2. Yukarıdaki görselde yer alan “Download for Windows” alanına tıklayın.
3. İndirme tamamlandıktan sonra, indirilen setup dosyasını açın.
4. Kurulum sihirbazındaki adımları takip edin. Genellikle varsayılan seçenekleri kabul ederseniz yüklemeniz başarıyla tamamlanır.
5. Kurulum tamamlandığında, Visual Studio Code'un kısayolu masaüstünüzde ve Başlat menüsünde oluşturulacaktır.
6. Visual Studio Code'u açın ve kullanmaya başlayın.

macOS için Kurulum:

1. İlk olarak, Visual Studio Code'un resmi web sitesini ziyaret edin:
<https://code.visualstudio.com/>
2. Aşağıdaki görselde yer alan macOS Stable sürümünü seçin. Bu, macOS için Visual Studio Code'un en son sürümünü indirecektir.



3. İndirme tamamlandıktan sonra, indirilen .dmg dosyasını açın (genellikle İndirilenler klasöründe bulunur).
4. Açılan penceredeki Visual Studio Code simgesini sürükleyip "Applications" (Uygulamalar) klasörüne bırakın.
5. Visual Studio Code'u başlatmak için Launchpad veya Finder üzerinden "Applications" (Uygulamalar) klasörüne gidin, Visual Studio Code simgesini bulun ve tıklayın.
6. Visual Studio Code'u açın ve kullanmaya başlayın.

Linux için Kurulum:

1. İlk olarak, Visual Studio Code'un resmi web sitesini ziyaret edin:
<https://code.visualstudio.com/>
2. MacOS kurulum adımlarında yer alan görseldeki Linux x64 .deb veya .rpm dosyasını yükleyin. Ubuntu veya Debian tabanlı bir dağıtım kullanıyorsanız .deb dosyasını, CentOS veya Fedora gibi bir dağıtım kullanıyorsanız .rpm dosyasını yüklemeniz tavsiye edilir.
3. İndirme tamamlandıktan sonra, terminali açın. Genellikle indirilen dosyalar "İndirilenler" klasöründe bulunur. İndirme konumu farklıysa, o konuma gidin.
4. İndirilen dosyayı yüklemek için aşağıdaki komutları kullanın. <dosya-adı> kısmını, indirdiğiniz dosyanın adıyla değiştirin.

Debian/Ubuntu tabanlı dağıtımlar için:

```
bash  
  
sudo dpkg -i <dosya-adı>.deb
```

Red Hat/Fedora tabanlı dağıtımlar için:

```
bash  
  
sudo rpm -i <dosya-adı>.rpm
```

5. Yükleme tamamlandığında, uygulamalar menüsünden Visual Studio Code'u bulun ve kullanmaya başlayın.

NodeJS ile Programlamaya Başlangıç

İlk NodeJS Kodumuzu Yazalım:

İlk NodeJS kodunu yazmaya başlamadan önce bir klasör açıp o klasörün içine **hello.js** adında bir dosya oluşturalım ve bu klasörü Visual Studio Code editöründe açalım.

Ardından aşağıdaki görseldeki kodları yazıp editörde yer alan terminale **node hello.js** komutunu yazıp kodumuzu çalıştıralım.



```
1 console.log("Hello NodeJS");
2
3 let myName = "Berkan";
4 let department = "Computer Engineering";
5 let school = "Bursa Technical University";
6
7 console.log(
8   `Hi, I am ${myName} and I am studying ${department} at the ${school}.`
9 );
10
```

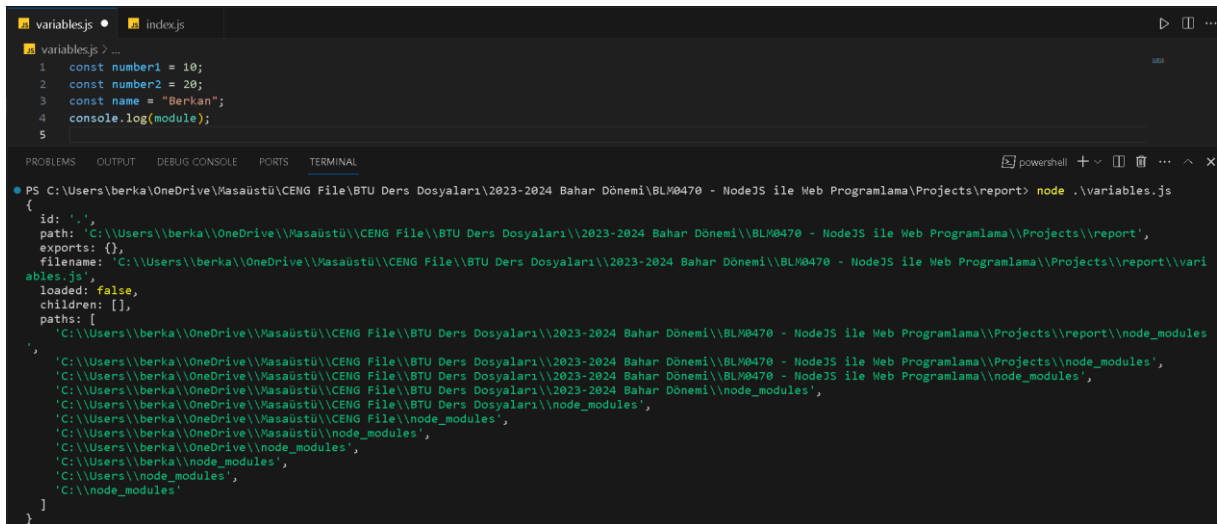
```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node hello.js
Hello NodeJS
Hi, I am Berkan and I am studying Computer Engineering at the Bursa Technical University.
```

Yukarıdaki kod kısaca konsol ekranına mesaj yazdırmaktadır.

Verileri Dışarıya Aktarma (export) İşlemi:

Eğer bir dosyada yazdığımız verileri başka bir dosyada kullanmak istersek **module.exports** ile istediğimiz verileri dışarıya aktarabiliyoruz. Örneğe geçmeden önce **module** nesnesinin ne olduğu hakkında bilgi edinelim


Module nesnesi: Module nesnesi, her NodeJS dosyasının içinde bulunan ve dosyanın kendisini temsil eden özel bir nesnedir ve dosyanın dışarı aktarılacak öğelerini tanımlamak için kullanır.



```
1 const number1 = 10;
2 const number2 = 20;
3 const name = "Berkan";
4 console.log(module);
5
```

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node .\variables.js
{
  id: '.',
  path: 'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\2023-2024 Bahar Dönemi\\BLM0470 - NodeJS ile Web Programlama\\Projects\\report',
  exports: {},
  filename: 'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\2023-2024 Bahar Dönemi\\BLM0470 - NodeJS ile Web Programlama\\Projects\\report\\variables.js',
  loaded: false,
  children: [],
  paths: [
    'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\2023-2024 Bahar Dönemi\\BLM0470 - NodeJS ile Web Programlama\\Projects\\report\\node_modules',
    'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\2023-2024 Bahar Dönemi\\BLM0470 - NodeJS ile Web Programlama\\Projects\\node_modules',
    'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\2023-2024 Bahar Dönemi\\node_modules',
    'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\BTU Ders Dosyaları\\node_modules',
    'C:\\Users\\berka\\OneDrive\\Masaüstü\\CENG File\\node_modules',
    'C:\\Users\\berka\\OneDrive\\node_modules',
    'C:\\Users\\node_modules',
    'C:\\node_modules'
  ]
}
```

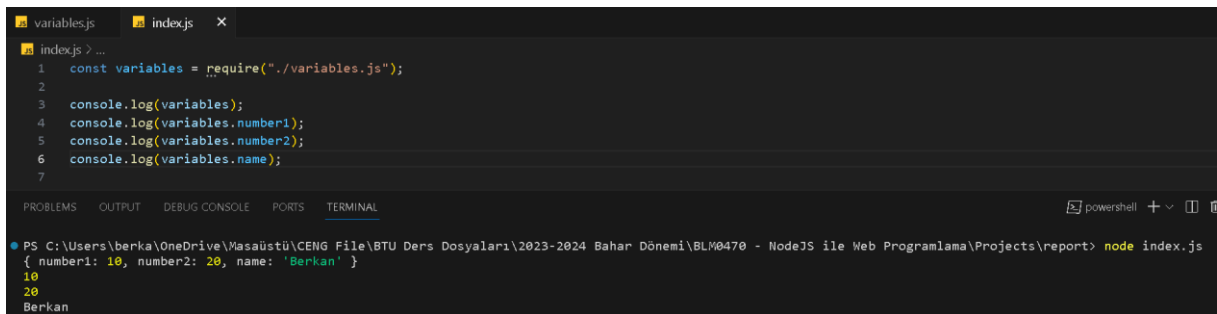
variables.js adında bir dosya açıp yukarıdaki kodları yazalım. Konsol ekranında çıktı olarak module nesnesinin hangi özelliklere sahip olduğu görülmektedir. Şu ana kadar herhangi bir veriyi dışı aktarmadığımız için **'exports'** nesnesi boş olarak görülmektedir. Şimdi de yukarıda yazdığımız üç değişkeni de **module.exports** özelliğini kullanarak dışarıya aktaralım.



```
variables.js X index.js
variables.js > ...
1  const number1 = 10;
2  const number2 = 20;
3  const name = "Berkan";
4
5  module.exports = { number1, number2, name };
6
```

Yukarıdaki gibi verilerimizi bir nesne biçiminde çoklu bir şekilde dışarıya aktardık. Şimdi de dışarıya aktardığımız verileri içeriye alalım yani **import** işlemini yapalım.

İlk olarak **index.js** adında bir dosya oluşturalım ve aşağıdaki görseldeki kodları yazalım ve **node index.js** komutu ile çalıştıralım.



```
variables.js X index.js
index.js > ...
1  const variables = require("./variables.js");
2
3  console.log(variables);
4  console.log(variables.number1);
5  console.log(variables.number2);
6  console.log(variables.name);
7
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node index.js
{ number1: 10, number2: 20, name: 'Berkan' }
10
20
Berkan
```

Direkt olarak değişkenlerin kendisine ulaşmak istiyorsak eğer JavaScript'in bir özelliği olan Destructring yöntemini kullanarak nesnenin içeriğini açalım. Bu özelliği kullanırken dikkat etmemiz gereken unsur, aktardığımız değişkenlerin adıyla içeri aldığımız değişkenlerin adının aynı olması gerekmektedir.



```
variables.js X index.js
index.js > ...
1  const { number1, number2, name } = require("./variables.js");
2
3  console.log(number1);
4  console.log(number2);
5  console.log(name);
6
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node index.js
10
20
Berkan
```

Yukarıda yer alan görseldeki gibi aktardığımız verilere kolay bir biçimde ulaşabiliriz. Kodda görüldüğü üzere ilk satırda atanmanın sağ tarafında **require** fonksiyonu kullanılmış ve parametre olarak dosya yolu gönderilmiş. Şimdi de **require** fonksiyonunun ne olduğunu açıklayalım.

require: NodeJS'te modül tabanlı bir programlama modeli sağlayan ve modüller arasında bağımlılıkları yöneten bir özelliktir. Bir dosyadaki modülü başka bir dosyada kullanmak için 'require' kullanılır. Bu, NodeJS projelerinde modüler bir yapı oluşturmanın temel taşlarından biridir. Require, aynı zamanda bir fonksiyondur ve bir modül dosyasının yolunu alır ve bu dosyadaki içeriği yükler. Bu işlem sırasında, dosya yüklenir, çalıştırılır ve içinde tanımlanan

öğeler exports veya module.exports aracılığıyla dışarıya aktarılır. require işlemi sonucunda, içe aktarılan modülün dışa aktardığı öğeler bir nesne olarak döner.

Modül içe aktarma işlemi, tam dosya yolu veya dosyanın ismi ile yapılabilir.

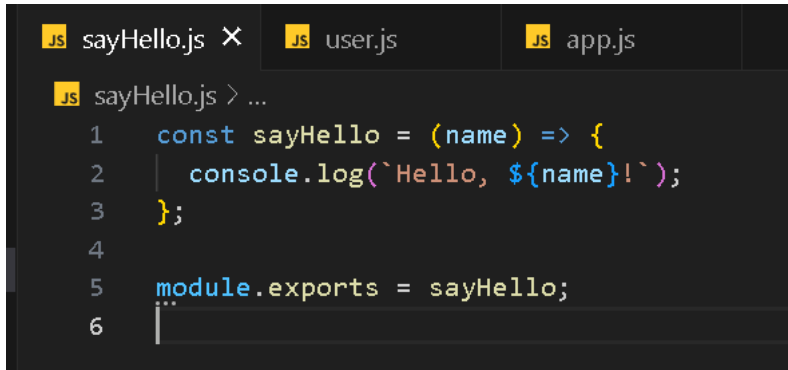
NodeJS projelerinde require işlemi, kodun daha modüler hale getirilmesini sağlar. Bu sayede, kodun farklı parçaları ayrı dosyalara bölünerek, daha yönetilebilir ve tekrar kullanılabilir hale gelir. Ayrıca, dışa aktarılan modüller arasında bağımlılıklar yönetilir ve bu sayede karmaşık projeler daha iyi organize edilir.

Import & Export Örnekleri:

Şimdiye kadar sadece string ve number tipinde değişkenleri kullandık. Şimdi yapacağımız örnekte array, fonksiyon, nesne ve diğer ilkel veri tipleri kullanarak import & export konusunu iyice pekiştirelim.

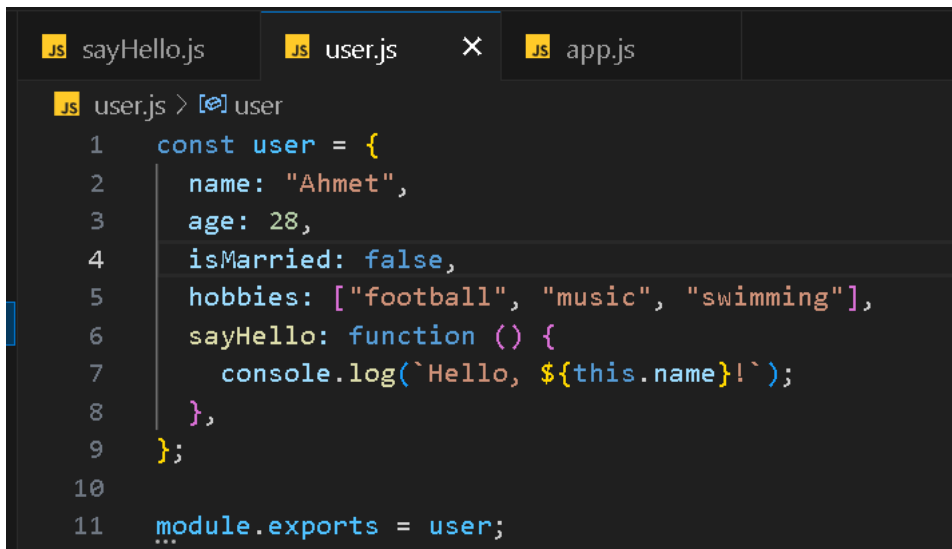
Öncelikle **app.js**, **user.js** ve **sayHello.js** adında üç farklı dosya oluşturalım.

sayHello.js adlı dosyamızın içeriği aşağıdaki gibi olsun.



```
JS sayHello.js X JS user.js JS app.js
JS sayHello.js > ...
1  const sayHello = (name) => {
2    console.log(`Hello, ${name}!`);
3  };
4
5  module.exports = sayHello;
6
```

user.js adlı dosyamızın içeriği aşağıdaki gibi olsun.



```
JS sayHello.js JS user.js X JS app.js
JS user.js > [E] user
1  const user = {
2    name: "Ahmet",
3    age: 28,
4    isMarried: false,
5    hobbies: ["football", "music", "swimming"],
6    sayHello: function () {
7      console.log(`Hello, ${this.name}!`);
8    },
9  };
10
11 module.exports = user;
```

app.js adlı dosyamızın içeriği de aşağıdaki gibi olsun. Bu dosya diğer dosyaların içe aktarıldığı ana dosyamızdır. Bu dosyanın içeriğini aşağıdaki gibi yazdıktan sonra komut satırına **node app.js** komutunu yazarak dosyamızı çalıştıralım.



```
1 const user = require("./user");
2 const sayHello = require("./sayHello");
3
4 sayHello(user.name);
5
6 user.sayHello();
7 user.hobbies.forEach((hobby) => {
8   console.log(hobby);
9 });
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

PS C:\Users\berka\OneDrive\ Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node app

Hello, Ahmet!
Hello, Ahmet!
football
music
swimming

Bu dosyada önceden oluşturduğumuz **user.js** ve **sayHello.js** adlı dosyaları içe aktardık. 4. satırda **sayHello.js** dosyamızda yer alan sayHello adlı fonksiyona **user.js** dosyasında yer alan user nesnesindeki name property'sini parametre olarak gönderdik ve çıktı olarak **Hello, Ahmet!** mesajını aldık. 6. satırda ise user nesnesinde yer alan sayHello fonksiyonunu çalıştırdık ve çıktı olarak yine **Hello, Ahmet!** mesajını aldık. 6 – 9. satırlar arasında yazdığımız kod ise user nesnesinde yer alan hobbies adlı dizide yer alan elemanlarını teker teker forEach döngüsüyle dolaşarak her bir elemanı konsola yazdırdık.

NPM (Node Package Manager)

NPM (Node Package Manager), NodeJS projelerinde bağımlılıkları yönetmek ve dış paketlerin kolayca entegrasyonunu sağlamak için kullanılan bir paket yöneticisidir. NPM, genellikle NodeJS'in yüklenmesiyle birlikte gelir. NPM'in önemli işlevleri arasında paket yönetimi, bağımlılık yönetimi, proje yönetimi ve paket yayınlama/paylaşma gibi bir dizi kritik görev bulunur. Bu görevleri kısaca bahsedecek olursak:

- 1. Paket Yönetimi:** NPM, NodeJS projelerinde kullanılacak dış paketleri aramanıza, yüklemenize ve güncellemenize olanak tanır. Projelerinizin ihtiyaç duyduğu her türlü modülü NPM üzerinden bulabilir ve tek bir komutla projenize dahil edebilirsiniz. Bu, işlevsellik genişletme, veritabanı bağlantıları, web sunucuları ve daha fazlası gibi birçok farklı alanda kullanılabilen kütüphaneleri içerir.
- 2. Bağımlılık Yönetimi:** NodeJS projeleri genellikle dış paketlere bağımlıdır. NPM, projenizdeki dış paketlerin uyumlu ve tutarlı bir şekilde çalışmasını sağlamak için bir bağımlılık çözümleyiciye sahiptir. Bu, projenizin ihtiyaç duyduğu farklı paketler arasındaki uyumluluğu garanti eder ve geliştirme sürecini daha sorunsuz hale getirir.
- 3. Proje Yönetimi:** NPM, projenizin paket bağımlılıklarını ve yapılandırmasını bir **package.json** dosyasında saklar. Bu dosya, projenizin gereksinimlerini ve bağımlılıklarını açıkça tanımlar ve projenin yönetilmesini ve dağıtılmasını kolaylaştırır. Ayrıca, projenin sürüm kontrol sistemlerinde (örneğin: Git) saklanmasını da destekler.
- 4. Paket Yayınlama ve Paylaşma:** NPM, projelerinizde kendi yazdığınız paketleri NPM Deposu'na (NPM Registry) yayınlamanıza ve başkalarıyla paylaşmanıza olanak tanır. Bu, projenizde geliştirdiğiniz özel kodları ve kütüphaneleri diğer geliştiricilerle

paylaşmanızı sağlar. Aynı şekilde, başkalarının yayınladığı paketleri de projenize entegre edebilirsiniz.

NPM'in bu özellikleri, NodeJS projelerinin geliştirilmesi ve yönetilmesi süreçlerini önemli ölçüde kolaylaştırır. NodeJS kullanımının popülerleşmesinde NPM önemli bir rol oynamaktadır.

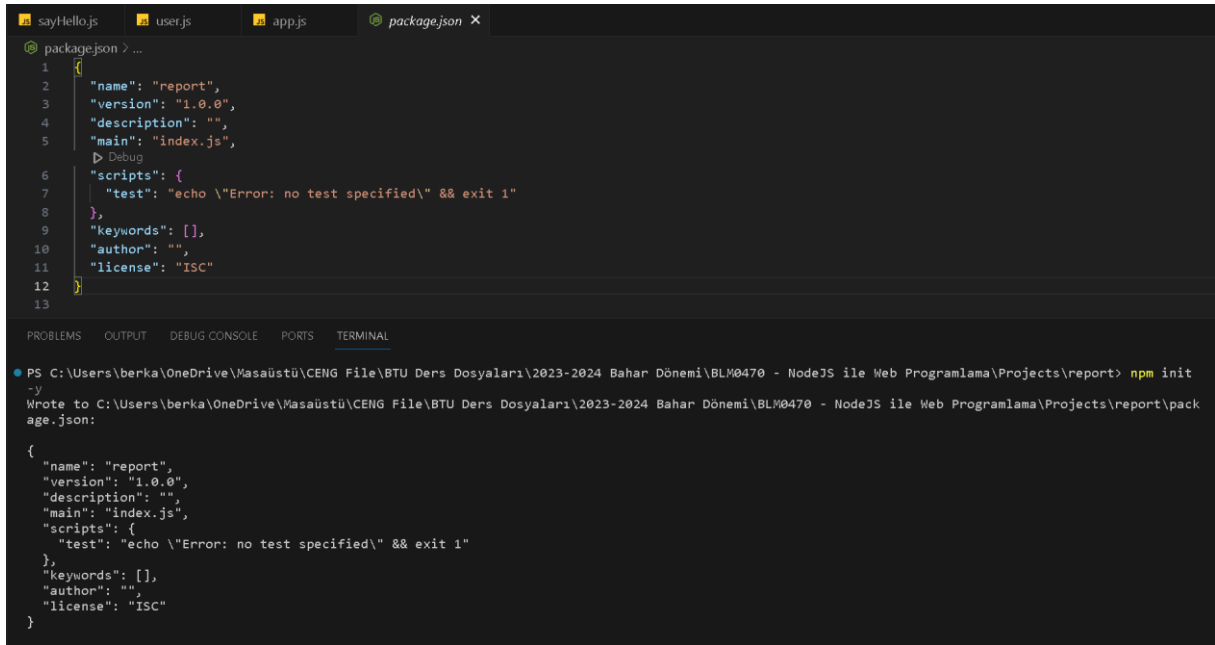
NPM komutları

npm init: Bu komut, NodeJS projeniz için **package.json** dosyasını oluşturmaya yarar. Bu dosya, projeniz hakkında önemli bilgileri içerir, Örneğin,

- Projenizin adı
- Projenizin sürümü
- Projenizin yazarları
- Projenizin açıklaması
- Projenizin bağımlılıkları.

npm init komutunu çalıştırdığınızda, size bir dizi soru sorulacaktır. Bu sorulara verdiğiniz cevaplar, package.json dosyasında saklanacaktır. Sorulara cevap vermek istemezseniz, Enter tuşuna basarak varsayılan değerleri kabul edebilirsiniz veya **npm init -y** komutunu kullanarak varsayılan değerleri otomatik olarak kabul edip dosyayı oluşturabilirsiniz.

Editörümüzdeki komut satırını açıp **npm init -y** komutunu çalıştıralım. Aşağıdaki görselde görüldüğü üzere bu komutu çalıştırdıktan sonra **package.json** dosyamız oluşturuldu.



```
package.json > ...
1 {
2   "name": "report",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
12 }
13

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
• PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> npm init
-y
Wrote to C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report\pack
age.json:

{
  "name": "report",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

npm install: Proje bağımlılıklarını yüklemek için kullanılır. **npm install** komutu, package.json dosyasındaki bağımlılıkları kontrol eder ve bunları yükler. Bağımlılıklar, dependencies ve devDependencies olarak iki kategori altında tanımlanabilir. **dependencies**, projenizin çalışması için gerekli olan ve üretim (production) ortamında kullanılan dış paketleri belirtmek için kullanılır. Yani, projenizin çalışması için gerçekten gerekli olan paketler burada belirtilir.

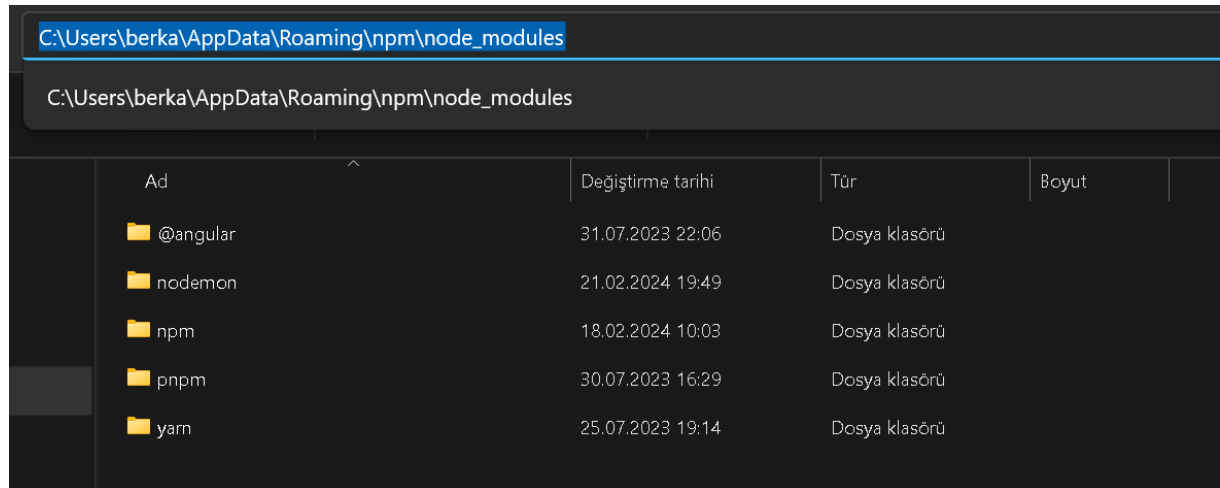
devDependencies ise projenin geliştirme sürecinde gerekli olan ve sadece geliştirme ortamında kullanılan dış paketleri belirtmek için kullanılır. Bu, projenin derlenmesi, test edilmesi, belgelenmesi veya düzenlenmesi gibi işlemlerde kullanılan yardımcı araçlar veya kütüphaneler içindir. Bu paketler, projenin kullanıcıları tarafından kullanılmayacağını sadece geliştiriciler tarafından geliştirme sürecinde kullanılacağını belirtir.

npm install komutuyla beraber kullanılan birkaç tane flag da vardır.

--save: Bu flag, bir paketin dependencies bölümüne eklenmesi gerektiğini belirtir. Yani, bu seçenek kullanıldığında, yüklenen paket package.json dosyasına **dependencies** bölümü altına eklenir.

--save-dev: Bu flag, bir paketin devDependencies bölümüne eklenmesi gerektiğini belirtir. Bu seçenek kullanıldığında, yüklenen paket package.json dosyasına **devDependencies** bölümü altına eklenir. Bu komut yerine kısa yazım olarak **-D** komutu da kullanılabilir.

-g: Bu flag, bir paketin global olarak yüklenmesini sağlar. Yani, paket proje dizininin dışında, kullanıcı hesabının altında veya sistem genelinde kullanılabilir hale gelir. Genellikle, projeye bağlantılı olmayan araçlar veya yardımcı programlar için kullanılır. Global olarak yüklenen paketler, package.json dosyasına eklenmezler. Bu komut kullanıldığında ilgili paket Windows işletim sistemlerinde **C:\Users\<userName> \AppData\Roaming\npm\node_modules** dizini altına eklenir.



Ad	Değiştirme tarihi	Tür	Boyut
@angular	31.07.2023 22:06	Dosya klasörü	
nodemon	21.02.2024 19:49	Dosya klasörü	
npm	18.02.2024 10:03	Dosya klasörü	
pnpm	30.07.2023 16:29	Dosya klasörü	
yarn	25.07.2023 19:14	Dosya klasörü	

npm uninstall: kurulu bir paketi kaldırmak için kullanılır. **npm uninstall <paket-adi>** komutu, belirtilen paketi projeden kaldırır. Bu, package.json dosyasındaki bağımlılıkları günceller.

npm update: Paketlerin güncellenmesi için kullanılır. npm update komutu, mevcut paketleri en son sürümlere günceller. **npm update <paket-adi>** şeklinde kullanılarak belirli bir paketin güncellenmesi sağlanabilir.

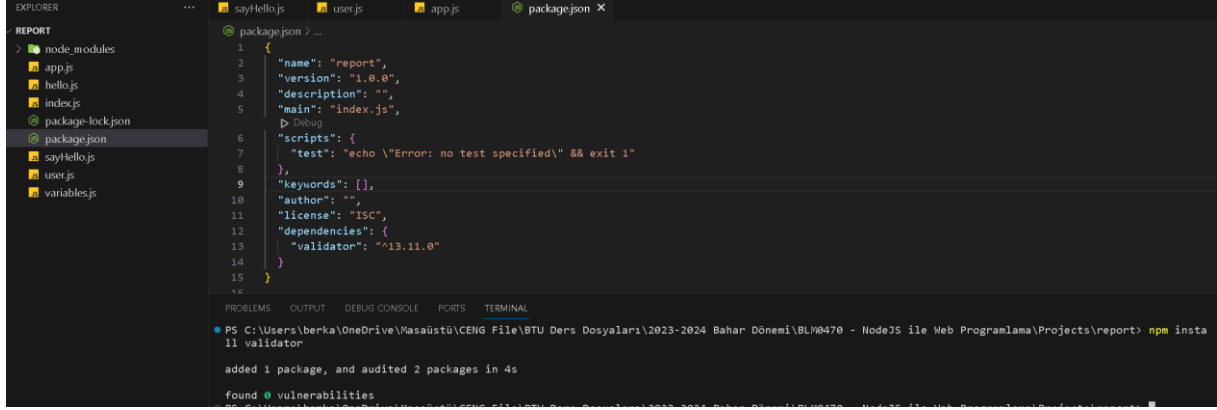
npm run: package.json dosyasında tanımlı özel scriptleri çalıştırmak için kullanılır. Örneğin, npm run build komutu, projenizin yapısını oluşturmak için belirtilen bir derleme betiğini çalıştırır.

npm start: package.json dosyasında belirtilen başlangıç betiğini çalıştırmak için kullanılır. Genellikle, bir NodeJS sunucusunu başlatmak veya uygulamayı çalıştırmak için kullanılır.

İlk Paketimizi Yükleme

NPM'in ne olduğu hakkında fikir edindiğimize göre projemizi açıp ilk paketimizi entegre edelim.

Komut satırına gelip **npm install validator** komutunu çalıştıralım. Görselde görüldüğü üzere 'validator' adlı paketimiz dependencies nesnesi altına eklendiğini görmekteyiz.



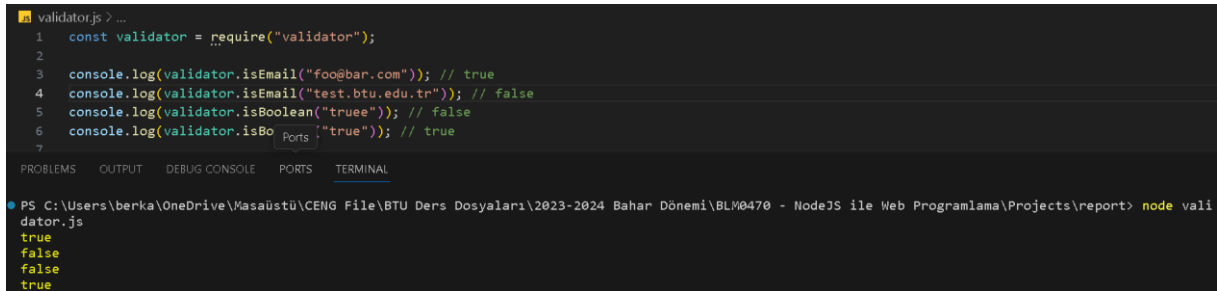
```
package.json
1 {
2   "name": "report",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "validator": "^13.11.0"
14  }
15 }
```

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> npm install validator
added 1 package, and audited 2 packages in 4s
found 0 vulnerabilities
```

Sol tarafta yer alan proje dosyalarının olduğu kısımda paketi yükledikten sonra package.json'a ek olarak **package-lock.json** adlı dosya oluşturuldu. Bu dosya, projenin bağımlılıklarının belirlenmiş sürümlerini içerir. Her bir bağımlılığın yüklenmiş olan kesin sürümü ve bu sürümün bağımlı olduğu diğer paketlerin sürümleri **package-lock.json** dosyasında listelenir. Bu sayede, projenin farklı ortamlarda tekrar oluşturulması veya başka bir geliştirici tarafından çalıştırılması durumunda, her seferinde aynı bağımlılıkların aynı sürümlerinin yüklendiğinden emin olunabilir. Ayrıca, **package-lock.json** dosyası, projenin bağımlılıklarının güvenliğini de sağlar. Her bir bağımlılığın yüklenmiş olan paketin **hash** değeri dosyada bulunur. Bu, paketlerin orijinalliğini kontrol etmek için kullanılır ve projenin güvenilirliğini artırır.

'**validator**' paketi, NodeJS ortamında kullanılan bir JavaScript kütüphanesidir. Bu kütüphane, genellikle web uygulamalarında, özellikle form doğrulama işlemlerinde kullanılır. validator paketi, çeşitli veri doğrulama işlemleri için hazır fonksiyonlar içerir ve girdi verilerini kontrol etmek için kullanıcıya kolaylık sağlar. Şimdi ise yüklediğimiz validator paketiyle ilgili örnekler yazalım.

validator.js adlı bir dosya oluşturup içerisine aşağıdaki görseldeki kodları yazalım ve **node validator.js** komutu ile dosyamızı çalıştıralım.



```
validator.js
1 const validator = require("validator");
2
3 console.log(validator.isEmail("foo@bar.com")); // true
4 console.log(validator.isEmail("test.btu.edu.tr")); // false
5 console.log(validator.isBoolean("truee")); // false
6 console.log(validator.isBoolean("true")); // true
7
```

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node validator.js
true
false
false
true
```


validator.isEmail() fonksiyonu parametre olarak string bir deęişken alır ve bu deęişkeni kontrol edip mail formatına uygun olup olmadığını kontrol eder ve geriye true veya false deęerini döndürür. isBoolean() fonksiyonu da parametre olarak string bir deęişken alır ve bu deęişkenin boolean deęerinde yani true veya false mi gönderilmiş olduğunu kontrol eder ve benzer şekilde geriye true veya false deęerini döndürür.

Dięer kullanacağımız paket olan **chalk** paketini uygulamamıza indirelim. Bu paket konsol çıktılarını renklendirmek için kullanılır. Bu paketi yüklerken dikkat etmemiz gereken bir nokta bulunmaktadır. Bu paketin 5 ve sonraki sürümleri CommonJS modülünü desteklememektedir. Bu yüzden paketimizi indirirken versiyon numarasını da belirtmemiz gerekir.

Paketi **npm install chalk@4.1.2** komutu ile yükleyelim ve **validator.js** adlı dosyamızı aşığıdaki görseldeki gibi yeniden düzenleyelim.



The screenshot shows a code editor with two tabs: 'validator.js' and 'package.json'. The 'validator.js' tab is active, displaying the following code:

```
1 const validator = require("validator");
2 const chalk = require("chalk");
3
4 console.log(chalk.blue.bgRed(validator.isEmail("foo@bar.com"))); // true
5 console.log(chalk.underline.green(validator.isEmail("test.btu.edu.tr"))); // false
6 console.log(chalk.red(validator.isBoolean("truee"))); // false
7 console.log(chalk.blue.underline.bold(validator.isBoolean("true"))); // true
```

Below the code editor, the 'TERMINAL' tab is active, showing the command and output:

```
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node .\validator.js
true
false
false
true
```

Görüldüğü üzere çıktılarımız renklendirilmiş bir biçimde görülmektedir.

Çekirdek Modüller (Core Modules)

NodeJS'in çekirdek (core) modülleri, NodeJS'in kendisiyle birlikte gelen ve genellikle temel işlevleri gerçekleştiren modüllerdir. Core modüller, ek bir kurulum gerektirmez ve NodeJS yüklendiğinde otomatik olarak kullanılabilir hale gelirler. Bu modüller, NodeJS'in temel işlevlerini gerçekleştirmek için gereken API'leri sağlarlar ve genellikle dosya sistem işlemleri, ağ işlemleri, işletim sistemi işlemleri gibi temel operasyonlar için kullanılırlar.

NodeJS'in bazı çekirdek modüllerinin ne olduğu ve ne işe yaradığı aşığıda açıklanmıştır.

1. **fs (File System):** Dosya sistemi işlemleri için kullanılır. Dosya oluşturma, okuma, yazma, güncelleme, silme gibi işlemleri gerçekleştirmek için bu modülü kullanabilirsiniz.
2. **http:** HTTP sunucu ve istemci işlemleri için kullanılır. Web sunucuları oluşturmak ve HTTP istekleri yapmak için bu modülü kullanabilirsiniz.
3. **path:** Dosya yollarını işlemek ve düzenlemek için kullanılır. Dosya yollarını birleştirme, ayrıştırma, mutlak veya göreceli yola dönüştürme gibi işlemler bu modül ile gerçekleştirilir.

4. **os (Operating System):** İşletim sistemi ile ilgili bilgilere erişim sağlar. Bilgisayarın işletim sistemini, CPU mimarisini, bellek miktarını, ağ arayüzlerini vb. özellikleri bu modül aracılığıyla alabilirsiniz.
5. **events:** Olaylar (events) ve olay dinleyicileri (event listeners) oluşturmak için kullanılır. NodeJS'in olay tabanlı yapılarını oluşturmak ve kullanmak için bu modülü kullanabilirsiniz.
6. **util:** Yardımcı işlevler ve hata ayıklama araçları sağlar. Bu modül, nesne işlemleri, formatlama, hata ayıklama ve diğer yardımcı işlevler için kullanılır.
7. **crypto:** Kriptografi işlemleri için kullanılır. Şifreleme, çözme, hash oluşturma gibi güvenlik işlemleri için bu modülü kullanabilirsiniz.
8. **stream:** Akış (stream) işlemleri için kullanılır. Dosya okuma/yazma, ağ verisi okuma/yazma gibi işlemleri akışlar aracılığıyla gerçekleştirmek için bu modülü kullanabilirsiniz.
9. **url:** URL işlemleri için kullanılır. URL'leri ayrıştırma, oluşturma, çözme gibi işlemler için bu modülü kullanabilirsiniz.

Fs (File System) Modülünün Kullanımı

Bu modül, asenkron ve senkron olmak üzere iki farklı API sunar. Asenkron API, işlemlerin bir geri arama (callback) fonksiyonu kullanarak asenkron olarak gerçekleştirilmesini sağlar. Senkron API ise işlemlerin senkron olarak gerçekleştirilmesini sağlar. **fs** modülünün bazı işlevleri aşağıdaki gibidir.

1. **Dosya Okuma ve Yazma:** **fs.readFile()** ve **fs.writeFile()** gibi fonksiyonlar aracılığıyla dosya okuma ve yazma işlemleri gerçekleştirilebilir.
2. **Dosya Oluşturma ve Silme:** **fs.createFile()** ve **fs.unlink()** gibi fonksiyonlar aracılığıyla dosya oluşturma ve silme işlemleri gerçekleştirilebilir.
3. **Klasör Oluşturma ve Silme:** **fs.mkdir()** ve **fs.rmdir()** gibi fonksiyonlar aracılığıyla klasör oluşturma ve silme işlemleri gerçekleştirilebilir.
4. **Dosya ve Klasör İçeriğini Listeleme:** **fs.readdir()** fonksiyonu aracılığıyla bir dizinin içeriği listelenebilir.
5. **Dosya ve Klasör İsimlerini Değiştirme:** **fs.rename()** fonksiyonu aracılığıyla dosya ve klasör isimlerini değiştirme işlemleri gerçekleştirilebilir.

fs modülünün kullanımına geçmeden önce **callback fonksiyon**, **asenkron** ve **senkron** işlem hakkında bilgi edinelim.

Callback fonksiyonu: Callback fonksiyonlar, JavaScript programlama dilinde oldukça yaygın olarak kullanılan ve özellikle asenkron işlemleri yönetmek için kullanılan önemli olan

bir programlama kavramıdır. Bir callback fonksiyonu, **başka bir fonksiyona parametre olarak iletilen ve o fonksiyonun tamamlandığında çağırılması gereken bir fonksiyondur**. Callback fonksiyonları, genellikle bir işlemin sonucunu veya hata durumunu ele almak için kullanılır. Bir işlem başarılı bir şekilde tamamlandığında veya bir hata meydana geldiğinde, callback fonksiyonu bu sonucu işleyebilir veya uygun bir şekilde hata yanıtı verebilir.

Örneğin, NodeJS'te `fs.readFile()` fonksiyonu dosya okuma işlemi için kullanılır ve bu işlem asenkron olarak gerçekleşir. Bu fonksiyon, dosya okunduğunda çağırılacak bir callback fonksiyonunu kabul eder.

```
javascript

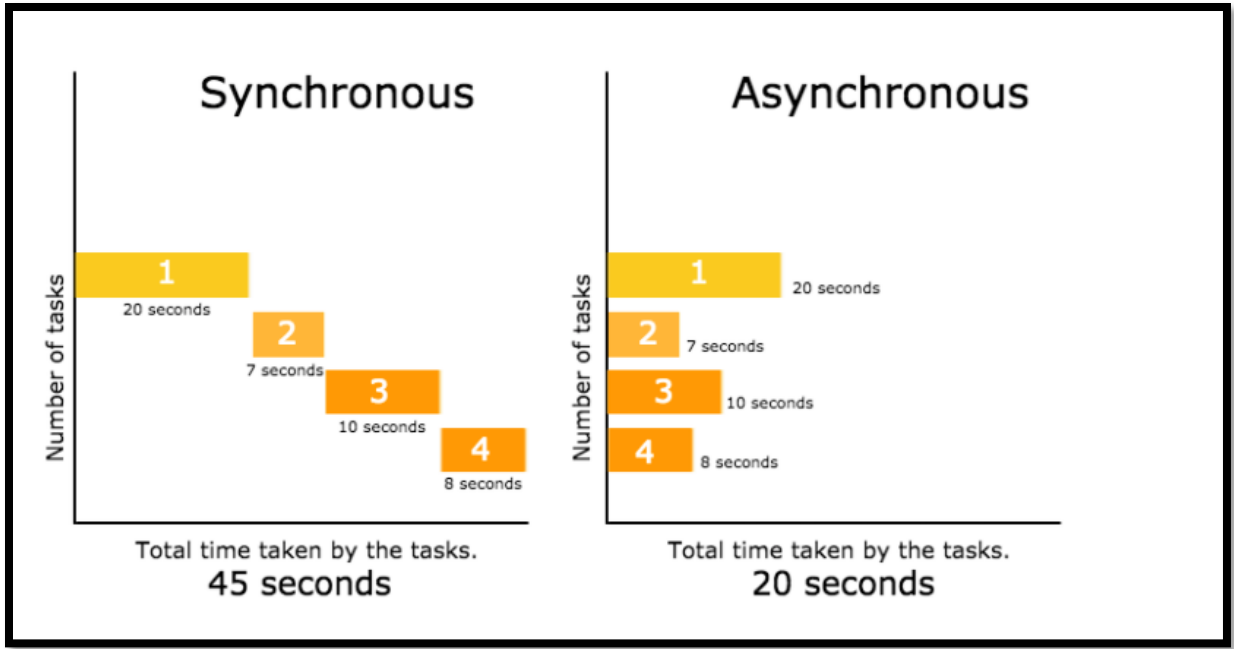
const fs = require('fs');

fs.readFile('dosya.txt', 'utf8', function(err, data) {
  if (err) {
    console.error('Dosya okuma hatası:', err);
  } else {
    console.log('Dosya içeriği:', data);
  }
});
```

Bu örnekte, `fs.readFile()` fonksiyonu dosya okuma işlemini gerçekleştirir ve işlem tamamlandığında veri veya hata durumunu ele almak için bir callback fonksiyonu kullanır. Callback fonksiyonu, işlem başarılı olduğunda dosya içeriğini veya hata durumunu konsola yazdırır.

Senkron işlem: Senkron biçiminde çalışan programda her görev sırasıyla birbirini takip edecek şekilde yürütülür. Eğer bir görev tamamlanması zaman alıyorsa, program veya süreç, bir sonraki göreve geçmeden önce o görevin tamamlanmasını bekler. Örnek olarak senkron olarak bir dosyayı okuyan bir fonksiyon düşünün. Eğer okunan dosyanın boyutu büyükse, program dosya tam okunana kadar bekler ve ilgili işlem bitmeden diğer bir görevin işlenmesine izin vermez. Senkron işlem ayrıca blocking kod işleme olarak da bilinir.

Asenkron işlem: Asenkron işlemler, programın normal akışından farklı olarak eşzamansız bir şekilde gerçekleştirilen işlemlerdir. Bu tür işlemler, işlem tamamlanana kadar diğer işlemlerin beklemesini gerektirmez ve programın diğer kısımlarının aynı anda çalışmasına olanak tanır. JavaScript gibi dillerde, özellikle NodeJS gibi platformlarda, asenkron işlemler sıkça kullanılır. Örneğin, asenkron programlamada bir dosya okuma işlemi başlatılabilir ve dosya arka planda okunurken program diğer görevlere devam edebilir. Geri çağrılar (callback fonksiyonları), promises ve `async/await` asenkron kodu işleme yöntemlerindendir. Asenkron işlem ayrıca non-blocking kod işleme olarak da bilinir.



Senkron ve Asenkron işlemlerinin çalışma mantığı

Şimdi bu öğrendiklerimizi fs modülü üzerinde pratik yapalım.

İlk olarak **file-module.js** adında bir dosya oluşturalım ve içeriğini aşağıdaki görseldeki gibi yazıp **node file-module.js** komutu ile kodumuzu çalıştıralım.

```
file-module.js X file.txt
file-module.js > ...
1 const fs = require("fs");
2
3 const message = 'Deneme yazısı 123\n';
4
5 fs.writeFileSync("./file.txt", message, { flag: "a", encoding: "utf-8" });
6 console.log(fs.readFileSync("./file.txt", "utf8"));
-
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node file-module.js
Deneme yazısı 123
```

`writeFileSync()` fonksiyonu senkron bir biçimde dosyaya yazma işlemlerini gerçekleştiren bir fonksiyondur. Bu fonksiyon, dosyaya veri yazmak için kullanılır ve işlem tamamlandığında geri dönüş değeri olarak bir hata (error) nesnesi döndürür. Eğer işlem başarılı bir şekilde gerçekleştirilirse, hata değeri null olur. Yukarıdaki kod örneğinde bu fonksiyona üç parametre gönderildi. İlk parametre mesajın yazılacağı dosyanın yolu, ikinci parametre yazacağımız mesajın kendisi, üçüncü parametre ise dosyaya hangi kurallar ile yazılacağını belirten opsiyonlar nesnesi gönderdik.

flag: "a" ==> Dosyaya yazma işlemi yaparken yazılan metnin silinmeden sonuna ekleme yapmak için kullanılır.

encoding: "utf-8" ==> Dosyanın hangi karakter kodlamasıyla yazılacağını belirlemek için kullanılır. En yaygın kullanılan karakter kodlamalarından biri UTF-8'dir. UTF-8, geniş bir karakter yelpazesini destekleyen ve dünya çapında en yaygın olarak kullanılan karakter kodlamalarından biridir.

Yukarıdaki kod parçası aşağıdaki sonucu verecektir.

```
file-module.js  file.txt  X
file.txt
1  Deneme yazısı 123
2
```

Senkron kod örneği:

```
file-module.js  file.txt
file-module.js > ...
1  const fs = require("fs");
2
3  const message = `Senkron yazma\n`;
4
5  console.log("İşlem başladı");
6  fs.writeFileSync("./file.txt", message, { flag: "a", encoding: "utf-8" });
7  console.log(fs.readFileSync("./file.txt", "utf8"));
8  console.log("İşlem tamamlandı");

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node file
-module.js
İşlem başladı
Deneme yazısı 123
Senkron yazma
İşlem tamamlandı
```

Yukarıda görüldüğü üzere çıktıya dikkatlice bakılırsa her bir işlem sıra sıra işlenmiş.

Asenkron kod örneği:

```
file-module.js  file.txt
file-module.js > [?] asyncProcess
1  const fs = require("fs");
2
3  const message = `Senkron yazma\n`;
4
5  const asyncProcess = () => {
6    fs.writeFile(
7      "./file1.txt",
8      message,
9      { flag: "a", encoding: "utf-8" },
10     (error) => {
11       if (error) {
12         console.log("Hata oluştu", error.message);
13       }
14       console.log("Dosya yazma işlemi başarılı");
15     }
16   );
17   fs.readFile("./file1.txt", "utf8", (err, data) => {
18     if (err) {
19       console.log("Dosya okuma sırasında hata oluştu", err.message);
20     }
21     console.log("Dosya okuma işlemi başarılı: ", data);
22   });
23 };
24
25 console.log("İşlem başladı");
26 asyncProcess();
27 console.log("İşlem tamamlandı");
28

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  TERMINAL
PS C:\Users\berka\OneDrive\Masaüstü\CENG File\BTU Ders Dosyaları\2023-2024 Bahar Dönemi\BLM0470 - NodeJS ile Web Programlama\Projects\report> node file
-module.js
İşlem başladı
İşlem tamamlandı
Dosya yazma işlemi başarılı
Dosya okuma işlemi başarılı: Senkron yazma
```

Bu örnekte ise çıktıya bakılırsa senkron işlemden farklı olarak bir işlem diğer işlemleri beklemeden işlenmiş ve yukarıdaki gibi bir çıktı oluşmuştur.

Kaynakça

<https://medium.com/@mehmetemin.nacarkahya/senkron-ve-asenkron-59f37eebc6c6>

<https://chat.openai.com/>