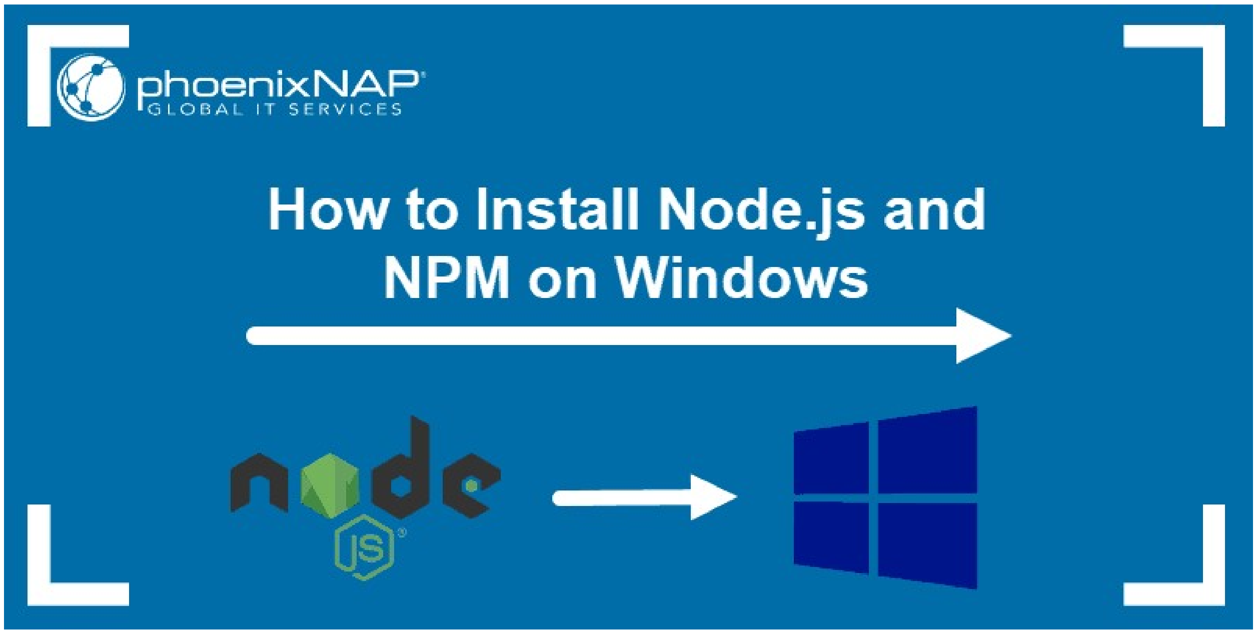
WEB DEVELOPMENT USING MEAN STACK

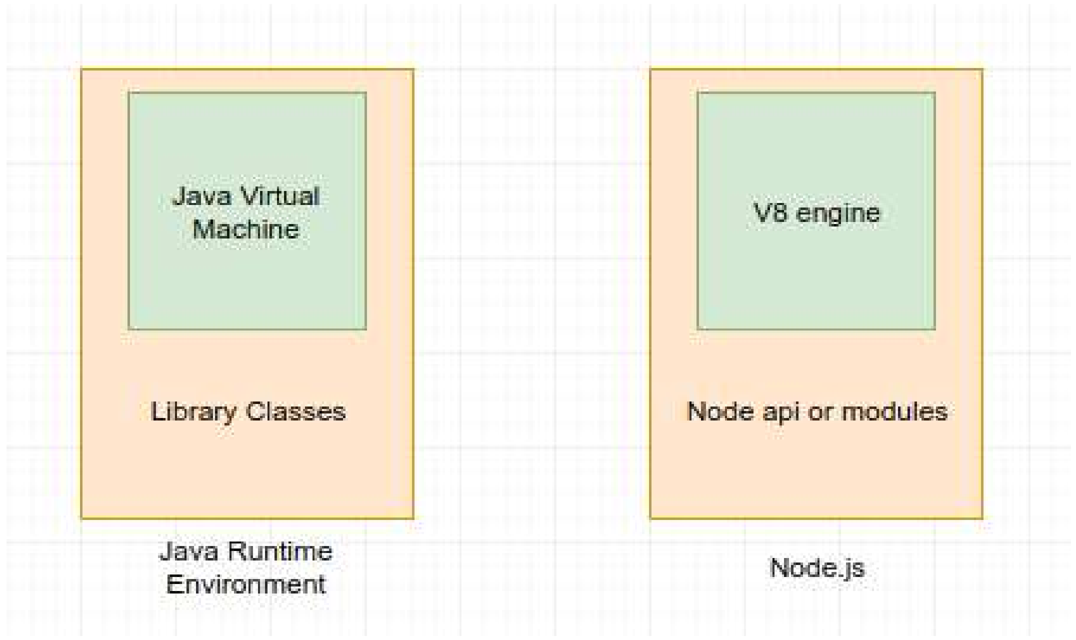
Unit 1 - Introduction to Nodejs



1. What exactly is Node.js?

“Node.js is a JavaScript runtime environment. Sounds great, but what does that mean? How does that work?”

1. The Node.js run-time environment includes everything you need to execute a program written in JavaScript.
2. Node.js came into existence when the original developers of JavaScript extended it from something you could only run in the browser to something you could run on your machine as a standalone application.



3. Now you can do much more with JavaScript than just making websites interactive. JavaScript now has the capability to do things that other scripting languages like Python can do.

4. Both your browser JavaScript and Node.js run on the V8 JavaScript runtime engine. This engine takes your JavaScript code and converts it into a faster machine code. Machine code is low-level code which the computer can run without needing to first interpret it.

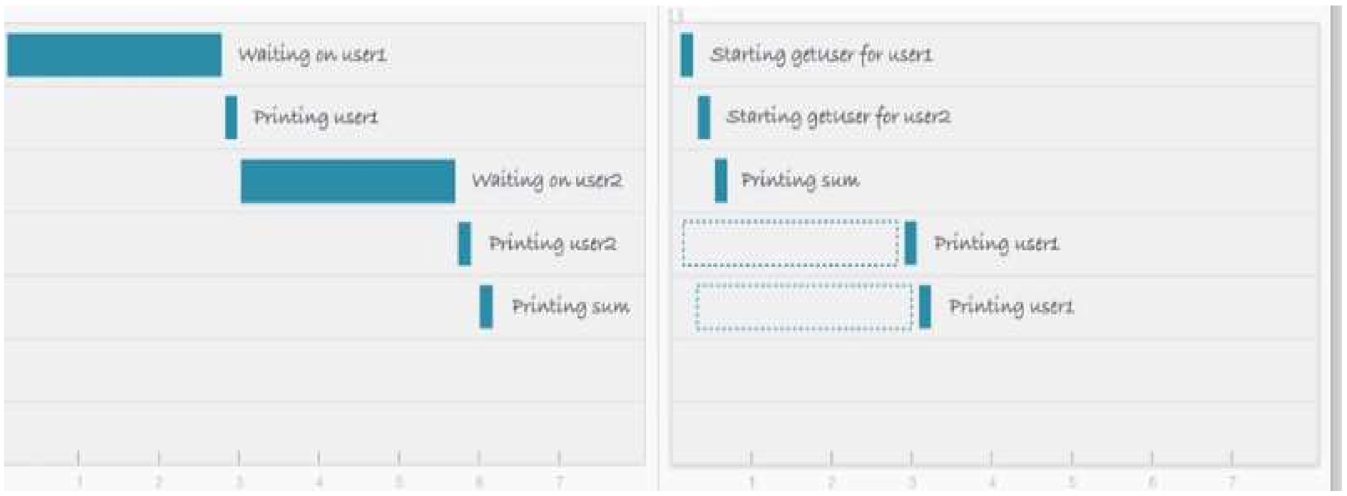
2. Why Node.js?

Here's a formal definition as given on the official Node.js website

1. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.

2. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

3. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.



Blocking I/O (left) vs Non-Blocking I/O (right)

Consider a scenario where we request a backend database for the details of user1 and user2 and then print them on the screen/console. The response to this request takes time, but both of the user data requests can be carried out independently and at the same time.

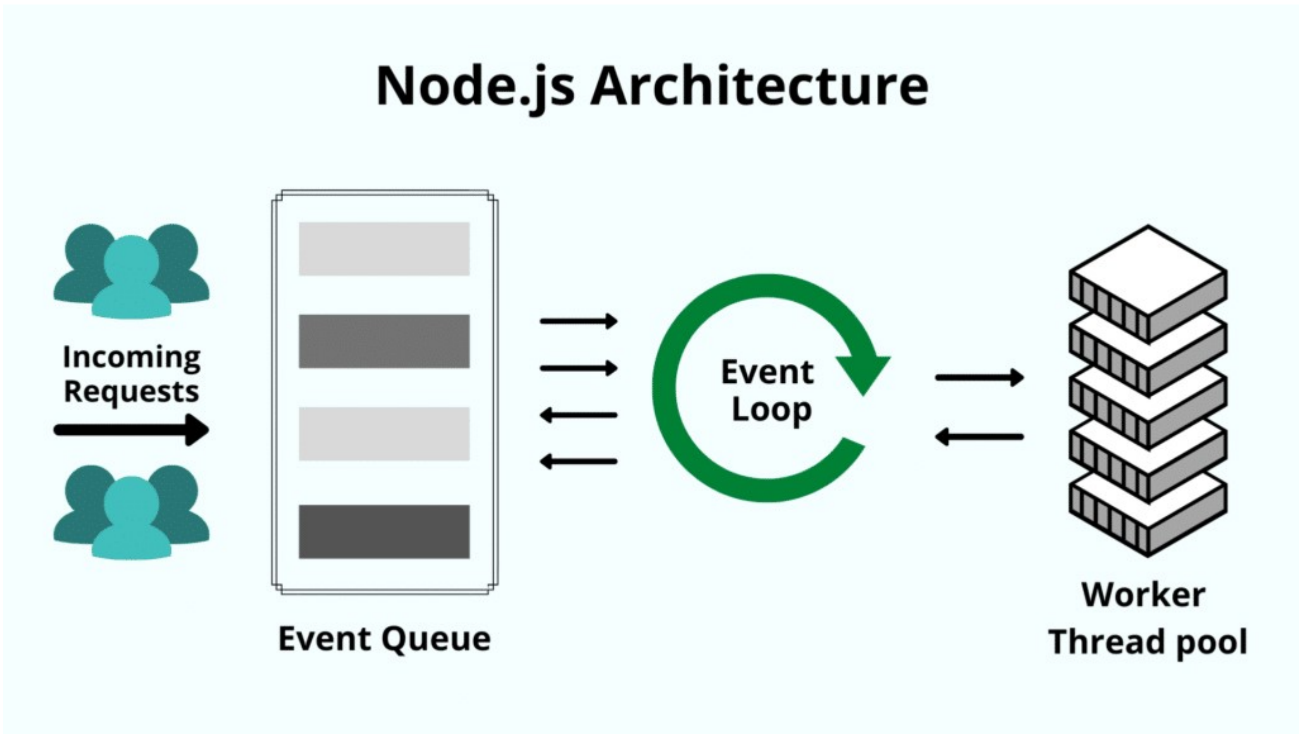
In the blocking method, user2's data request is not initiated until user1's data is printed to the screen.

On the other hand, using a non-blocking request, you can initiate a data request for user2 without waiting for the response to the request for user1. You can initiate both requests in parallel.

3. Node.js Architecture and How It Works?

1. Node.js uses the “Single Threaded Event Loop” architecture to handle multiple clients at the same time. To understand how this is different from other runtimes, we need to understand how multi-threaded concurrent clients are handled in languages like Java.

2. In a multi-threaded request-response model, multiple clients send a request, and the server processes each one before sending the response back. However, multiple threads are used to process concurrent calls. These threads are defined in a thread pool, and each time a request comes in, an individual thread is assigned to handle it.



Node.js works differently. Let’s take a look at each step it goes through:

1. Node.js maintains a limited thread pool to serve requests.
2. Whenever a request comes, Node.js places it into a queue.
3. Now, the single-threaded "Event loop"—the core component—comes into the picture. This event loop waits for requests indefinitely.
4. When a request comes in, the loop picks it up from the queue and checks whether it requires a blocking input/output (I/O) operation. If not, it processes the request and sends a response.
5. If the request has a blocking operation to perform, the event loop assigns a thread from the internal thread pool to process the request. There are limited internal threads available. This group of auxiliary threads is called the worker group.
6. The event loop tracks blocking requests and places them in the queue once the blocking task is processed. This is how it maintains its non-blocking nature.

Since Node.js uses fewer threads, it utilizes fewer resources/memory, resulting in faster task execution. So for our purposes, this single-threaded architecture is equivalent to multi-threaded architecture. When one needs to process data-intensive tasks, then using multi-threaded languages like Java makes much more sense. But for real-time applications, Node.js is the obvious choice.

4. Features of Node.js?

1. **Easy**—Node.js is quite easy to start with. It's a go-to choice for web development beginners. With a lot of tutorials and a large community—getting started is very easy.

2. **Scalable**—It provides vast scalability for applications. Node.js, being single-threaded, is capable of handling a huge number of simultaneous connections with high throughput.

3. **Speed**—Non-blocking thread execution makes Node.js even faster and more efficient.

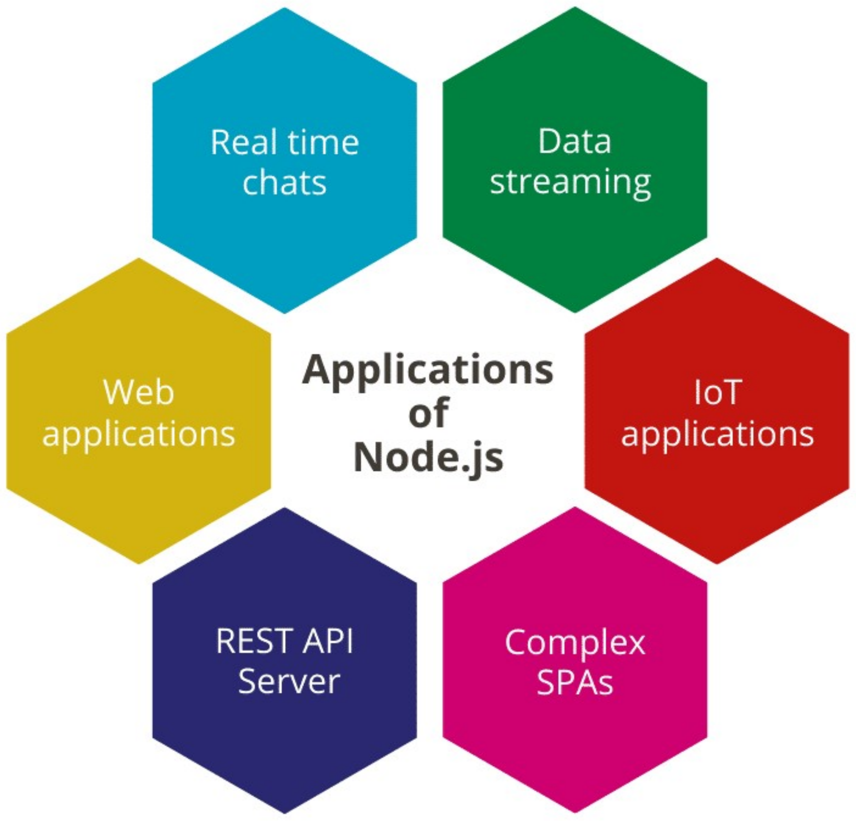
4. **Packages**—A vast set of open-source Node.js packages is available that can simplify your work. There are more than one million packages in the NPM ecosystem today.

5. **Strong backend**—Node.js is written in C and C++, which makes it speedy and adds features like networking support.

6. **Multi-platform**—Cross-platform support allows you to create SaaS websites, desktop apps, and even mobile apps, all using Node.js.

7. **Maintainable**—Node.js is an easy choice for developers since both the frontend and backend can be managed with JavaScript as a single language.

5. Applications of Node.js?



Node.js is used for a wide variety of applications. Let’s explore some popular use cases where Node.js is a good choice:

1. **Real-time chats**—Due to its single-threaded asynchronous nature, Node.js is well-suited to processing real-time communication. It can easily scale and is often used in building chatbots. Node.js also makes it simple to build additional chat features like multi-person chat and push notifications.

2. **Internet of Things**—IoT applications usually comprise multiple sensors, as they frequently send small chunks of data that can pile into a large number of requests. Node.js is a good choice since it's able to handle these concurrent requests quickly.

3. **Data streaming**—Companies like Netflix use Node.js for streaming purposes. This is mainly due to Node.js being lightweight and fast, besides which Node.js provides a native streaming API. These streams allow users to pipe requests to each other, resulting in data being streamed directly to its final destination.

4. **Complex single-page applications (SPAs)**—In SPAs, the whole application is loaded in a single page. This usually means there are a couple of requests made in the background for specific components. Node.js's event loop comes to the rescue here, as it processes requests in a non-blocking fashion.

5. **REST API-based applications**—JavaScript is used both in the frontend and backend of sites. Thus, a server can easily communicate with the frontend via REST APIs using Node.js. Node.js also provides packages like Express.js and Koa that make it even easier to build web applications.

6. Node js Modules?

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. Node.js has a set of built-in modules which you can use without any further installation.

7. Types of node modules

There are three types of node modules:

1. Core Modules
2. User-defined Modules
3. Node Package Manager

1- Core Modules: The modules which are in-built and can be used without any installation. You can use the core modules just by installing node's.

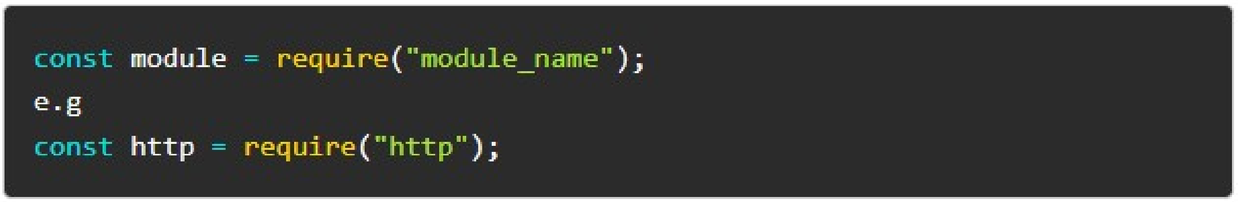
2- User-defined Modules: As the name suggests, a user-defined, basically these are the modules that are created by you so that you can reuse them in a node's application.

3- Node Package Manager: Node Package Manager (NPM) is a directory where the other developers develop the module and publish it. There are thousands of packages available on NPM's official site. You can use any packages just by installing them.

8. Syntax to import a node module?

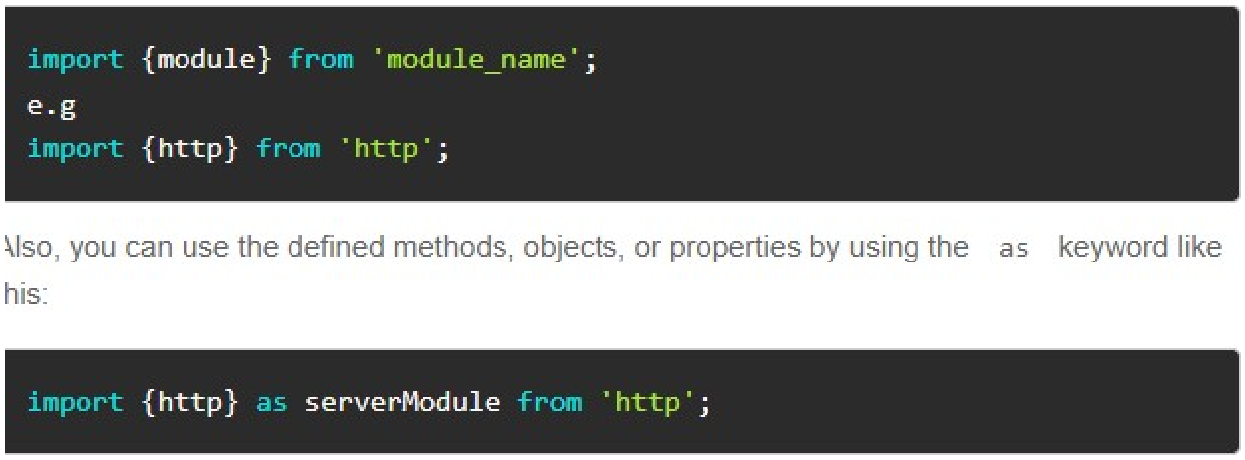
We can import a node module in two ways:

1. by using a require keyword



```
const module = require("module_name");
e.g
const http = require("http");
```

2. using import keyword



```
import {module} from 'module_name';
e.g
import {http} from 'http';
```

lso, you can use the defined methods, objects, or properties by using the `as` keyword like his:

```
import {http} as serverModule from 'http';
```

9. Out of 30+ prebuilt (core modules), the most used modules are?

1. **fs (file system)** → It handles the file systems, using this module you can create, read, and open files, etc.
2. **path** → This module is used to handle the path system.
3. **buffer** → It is used to handle the binary data
4. **http** → It is a mostly used module that makes NodeJS, an http server. Using this module, we will create a server to print the message.
5. **url** → It parses the url string
6. **util** → Used to handle the utility function in NodeJS

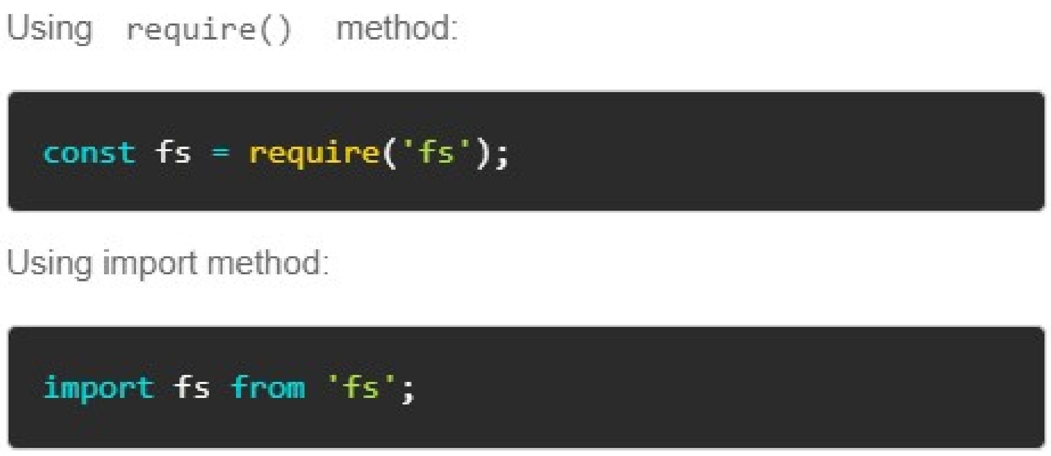
Last but not least,

7. **OS** → handles the Operating System Information.

10. File System Module?

The file system is a method of retrieving and organizing files from a storage device. Basically, the file system module is used to create, read, update or delete a file.

The file system module is a core module in NodeJS. And, it is used to perform various tasks on the files. Since this is a prebuilt module, there is no need to install it. Just import it to use this module.



Using `require()` method:

```
const fs = require('fs');
```

Using import method:

```
import fs from 'fs';
```

In the file system module, there are various operations can be performed. We'll explain each one by one with the help of an example.

The following(s) are the operations in the File System in Node:

1. Open
2. Get
3. Write
4. Read

5. Append
6. Rename
7. Delete

1. Open

This method is used to open the file and perform various operations.

Syntax:

```
fs.open(path, flags, callback());
```

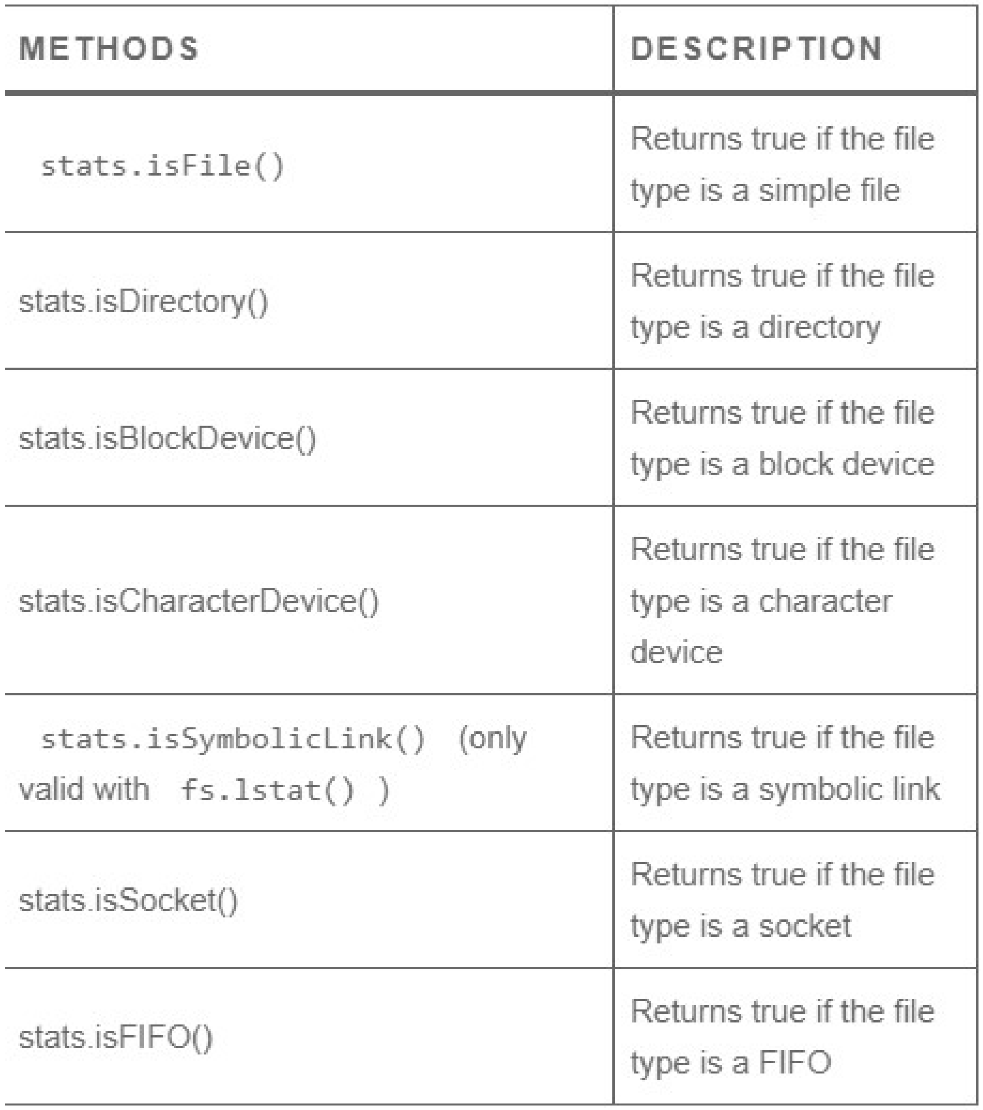
2. Get

This method is used to get useful information related to the file. In Node, we use the stat method to get the information about the file. The syntax of the stat method is:

```
fs.stat(path, callback);
```

In the above code, the first parameter is for the path of a file and the second parameter is a callback function which takes two arguments(err, stats);

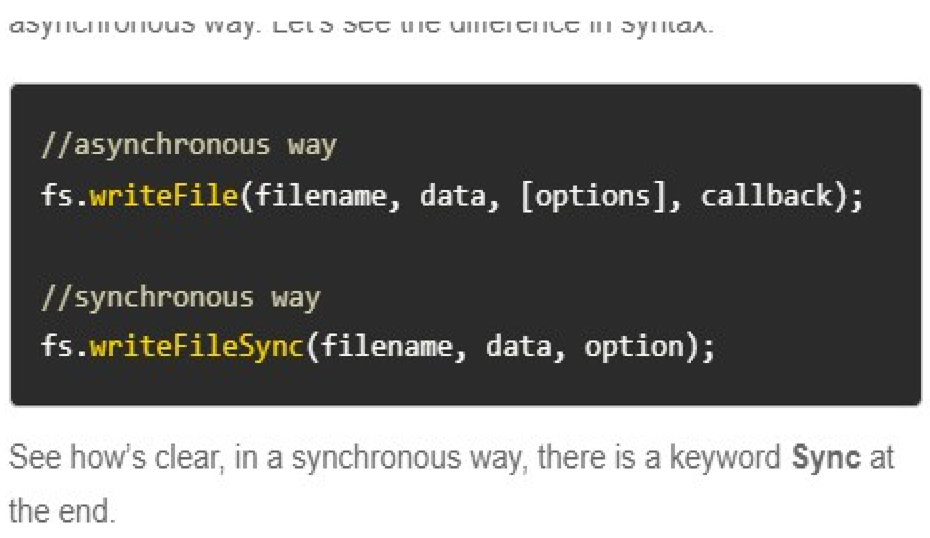
There are various methods available in the `fs.stat` class. Each method is used to check the various file type:



METHODS	DESCRIPTION
`stats.isFile()`	Returns true if the file type is a simple file
stats.isDirectory()	Returns true if the file type is a directory
stats.isBlockDevice()	Returns true if the file type is a block device
stats.isCharacterDevice()	Returns true if the file type is a character device
`stats.isSymbolicLink()` (only valid with `fs.lstat()`)	Returns true if the file type is a symbolic link
stats.isSocket()	Returns true if the file type is a socket
stats.isFIFO()	Returns true if the file type is a FIFO

3. Write

This method is used to write the data into the specified file. You can write the data using two methods both synchronously and asynchronously.



asynchronous way. Let's see the difference in syntax.

```
//asynchronous way
fs.writeFile(filename, data, [options], callback);

//synchronous way
fs.writeFileSync(filename, data, option);
```

See how's clear, in a synchronous way, there is a keyword **Sync** at the end.

Read Files

The fs.readFile () method is used to read files on your computer.

Delete Files

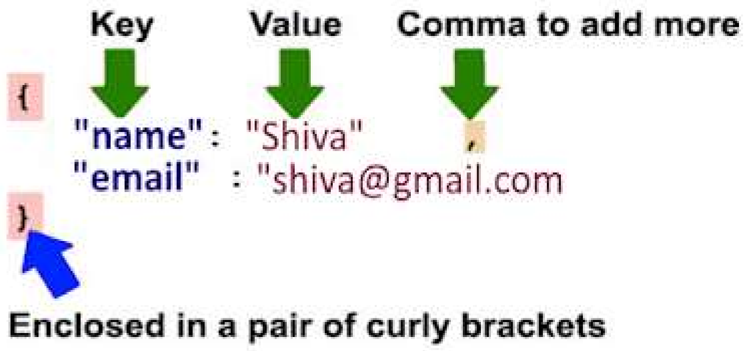
To delete a file with the File System module, use the fs.unlink() method.

Rename Files

To rename a file with the File System module, use the fs.rename() method.

11. Json data?

JSON means JavaScript Object Notation. JSON is an open standard lightweight format for data interchange. JSON is a format for storing and transporting data consisting of comma-separated attribute-value pairs and array data types. JSON is commonly used to send the information from a server to a web page.

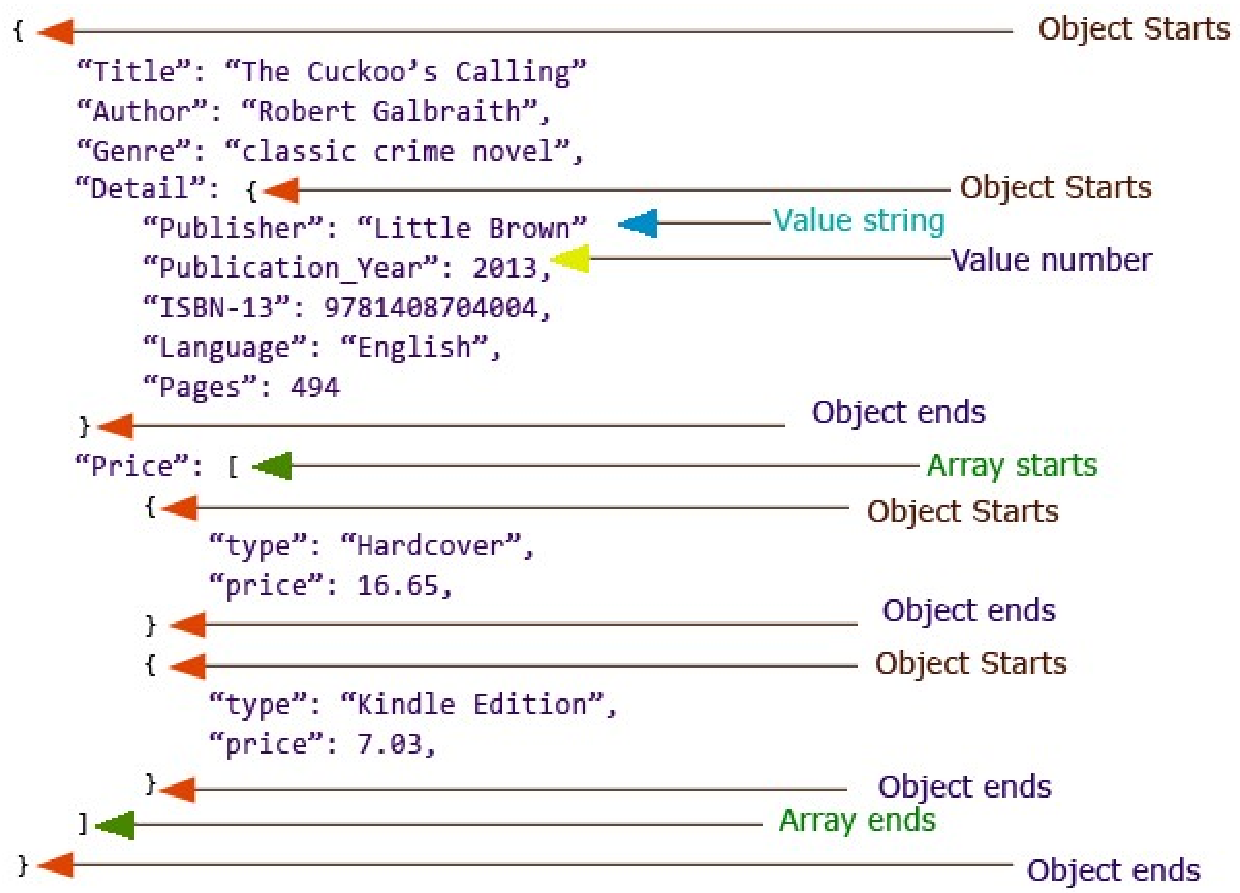

Key
Value
Comma to add more
{
"name" : "Shiva" ,
"email" : "shiva@gmail.com
}
Enclosed in a pair of curly brackets

12. Basic Constructs of json?

- There four basic and built-in data types in JSON. They are strings, numbers, Booleans (i.e. true and false) and null. Besides, there are two data types which are structured - objects and arrays.

- Objects are wrapped within '{' and '}'. Arrays are enclosed by '[' and ']'. Objects are a list of label-value pairs. Arrays are list of values.
- Both objects and arrays can be nested.

- Strings, numbers, Booleans (i.e true and false) and null can be used as values.



```
{
    "Title": "The Cuckoo's Calling"
    "Author": "Robert Galbraith",
    "Genre": "classic crime novel",
    "Detail": {
        "Publisher": "Little Brown"
        "Publication_Year": 2013,
        "ISBN-13": 9781408704004,
        "Language": "English",
        "Pages": 494
    }
    "Price": [
        {
            "type": "Hardcover",
            "price": 16.65,
        }
        {
            "type": "Kindle Edition",
            "price": 7.03,
        }
    ]
}
```

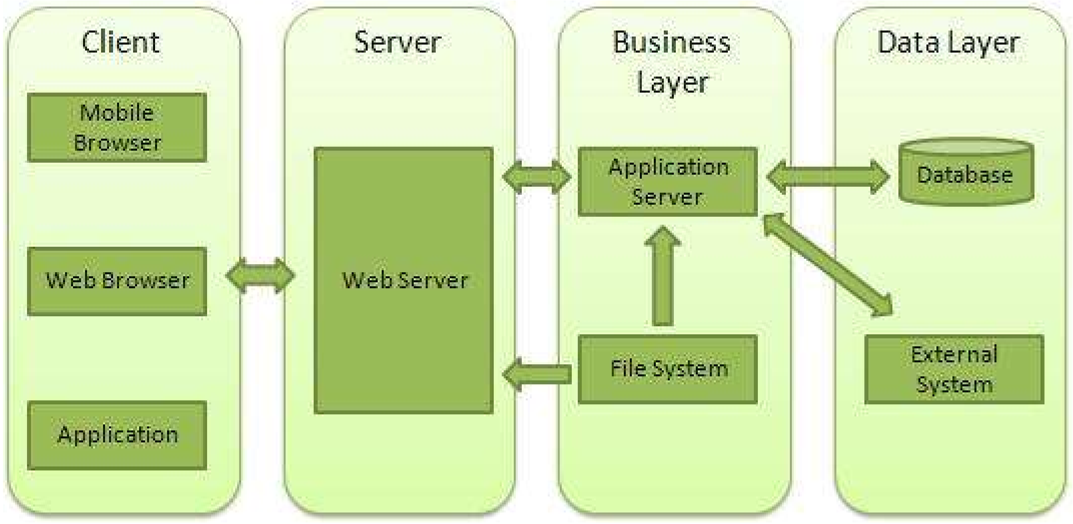
13. Http Server and Client?

A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients. Web servers usually deliver html documents along with images, style sheets, and scripts.

Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.

Web Application Architecture

A Web application is usually divided into four layers −



Client − This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.

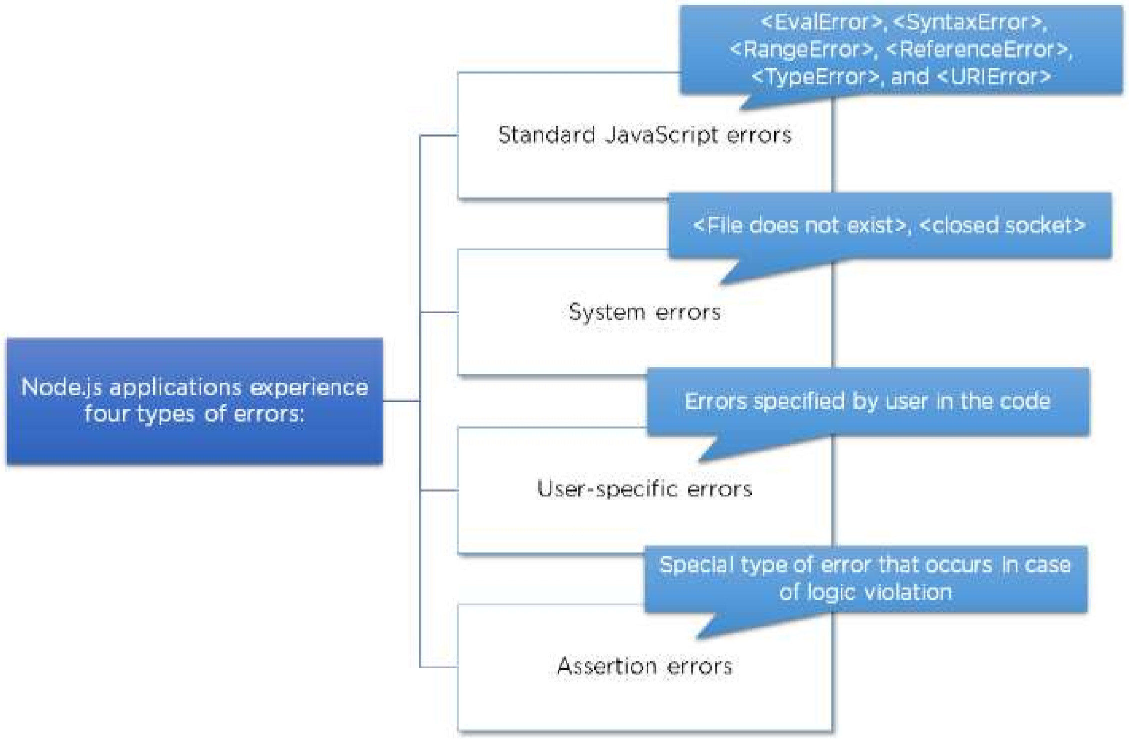
Server − This layer has the Web server which can intercept the requests made by the clients and pass them the response.

Business − This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.

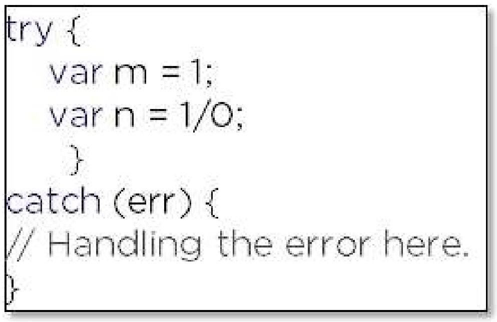
Data − This layer contains the databases or any other source of data.

14. Error Handling in Nodejs?

Node.js applications experience four types of errors.


Node.js applications experience four types of errors:
Standard JavaScript errors
<EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <TypeError>, and <URIError>
System errors
<File does not exist>, <closed socket>
User-specific errors
Errors specified by user in the code
Assertion errors
Special type of error that occurs in case of logic violation

Errors in Node.js are handled through exceptions. For example, let's handle the error that would occur when we divide a number by zero. This error would crash the Node.js application, so we should handle this error to continue with the normal execution of the application.



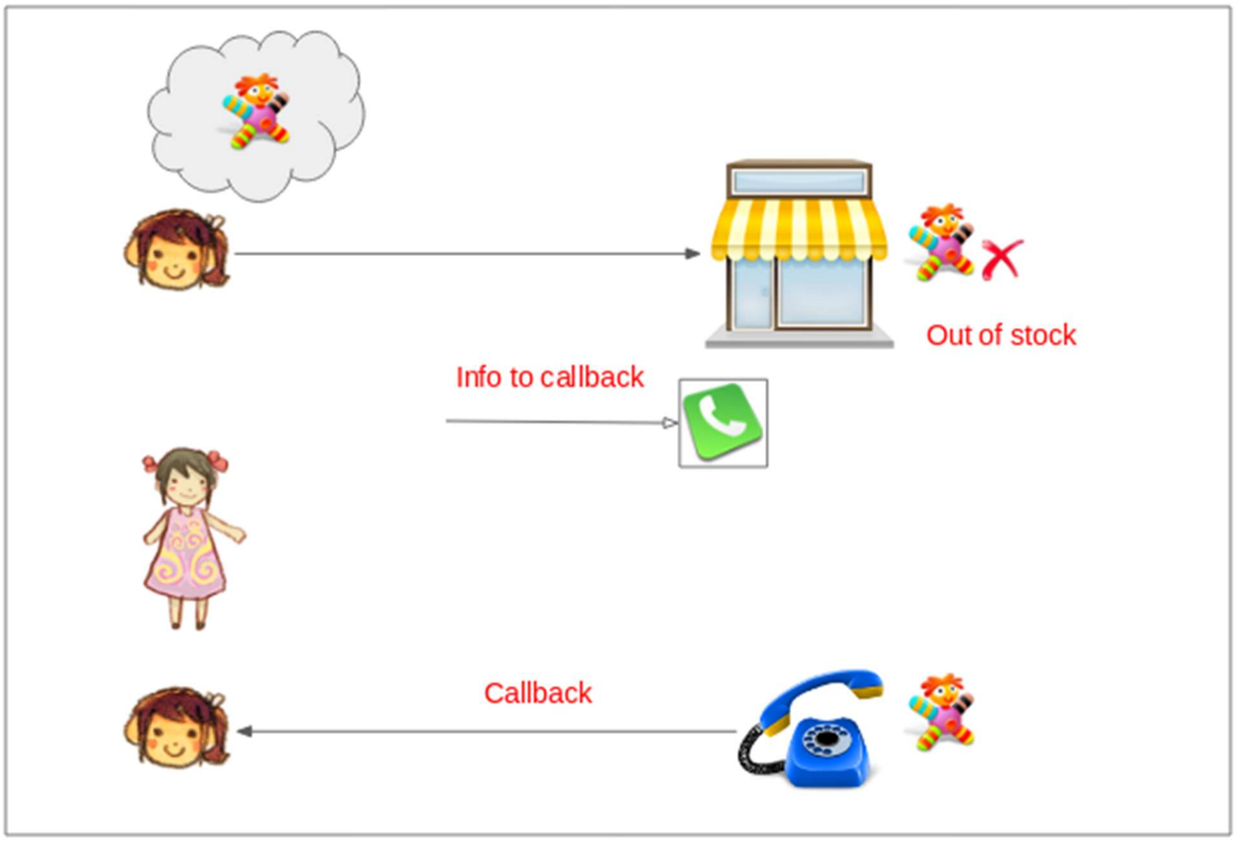
```
try {
    var m = 1;
    var n = 1/0;
     }
catch (err) {
// Handling the error here.
}
```

Fig: Node.js error handling

15. Callback function?

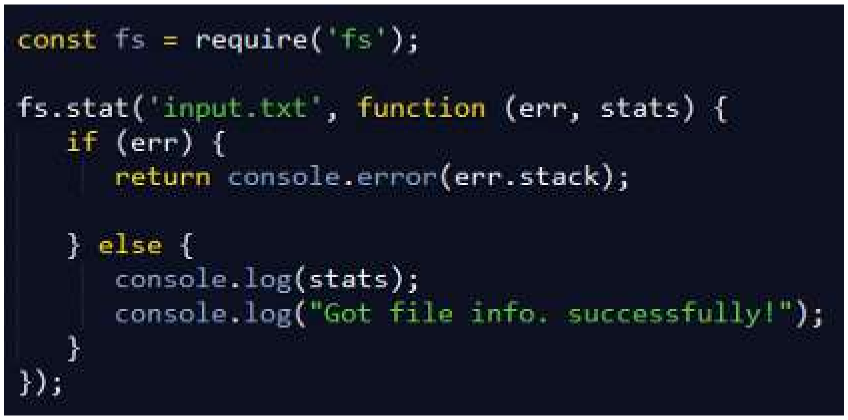
To explain what the Callback is, let's see a situation as follows:

“You come to a shop to buy a something you like. The shop staff tells you that the item is out of stock now. You leave the phone number and ask them to call back immediately after the goods is available. Then you can go out or do a job and do not need to care about the shop anymore, until you get a phone call notifying of the item asked by you from the shop.”



NodeJS server can receive many requests from many users. Therefore to improve serviceability, all APIs of the NodeJS are designed to support Callback. **The "callback" is a function that will be called when the NodeJs completes a specific task.**

Callback functions help us in the situation where, the function may accept some Param's and it may return a result after some time, not immediately. For example, when we call external third API services, these APIs may take some time to return the data. For handling this kind of situation the JavaScript provide the callback function approach to manage this problem easily.

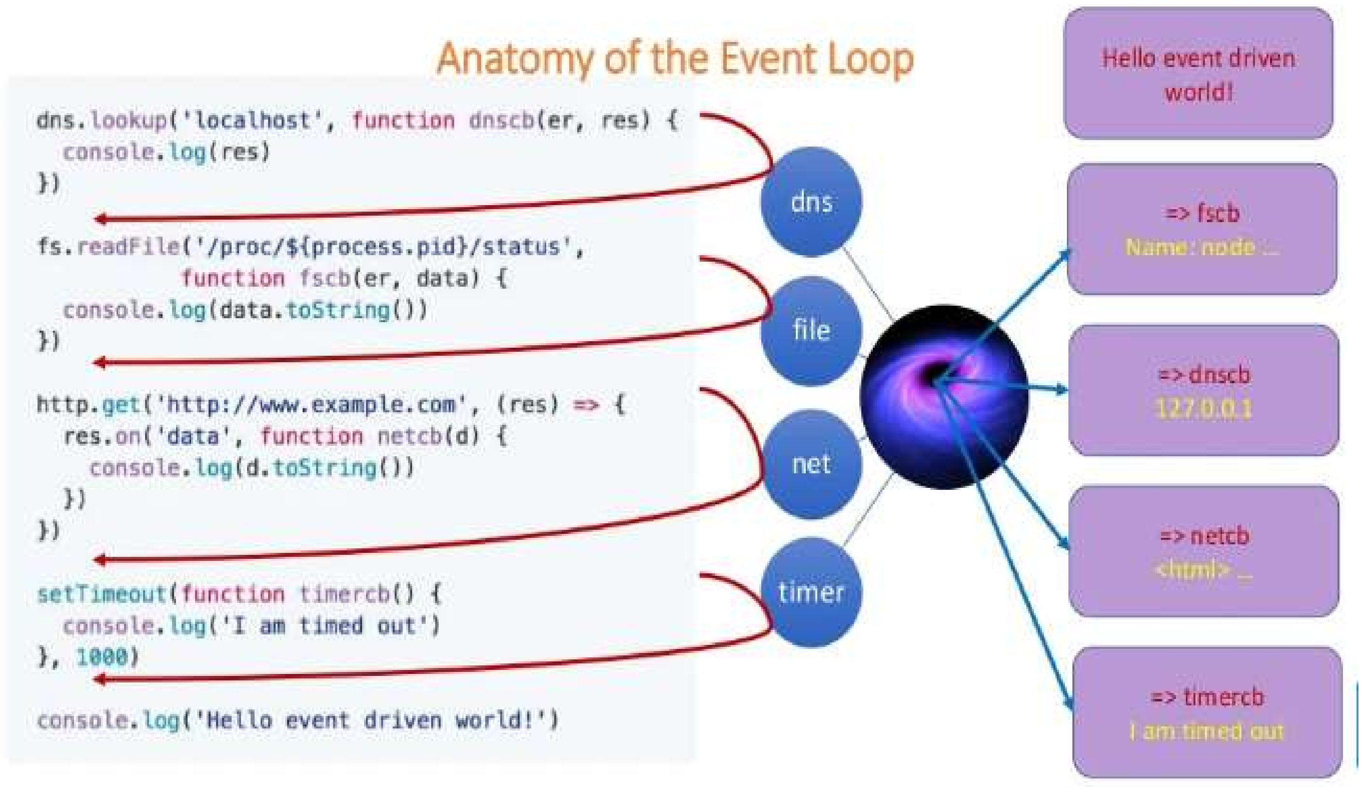


```
const fs = require('fs');

fs.stat('input.txt', function (err, stats) {
    if (err) {
        return console.error(err.stack);

    } else {
        console.log(stats);
        console.log("Got file info. successfully!");
    }
});
```

16. Asynchronous programming in Node.js?

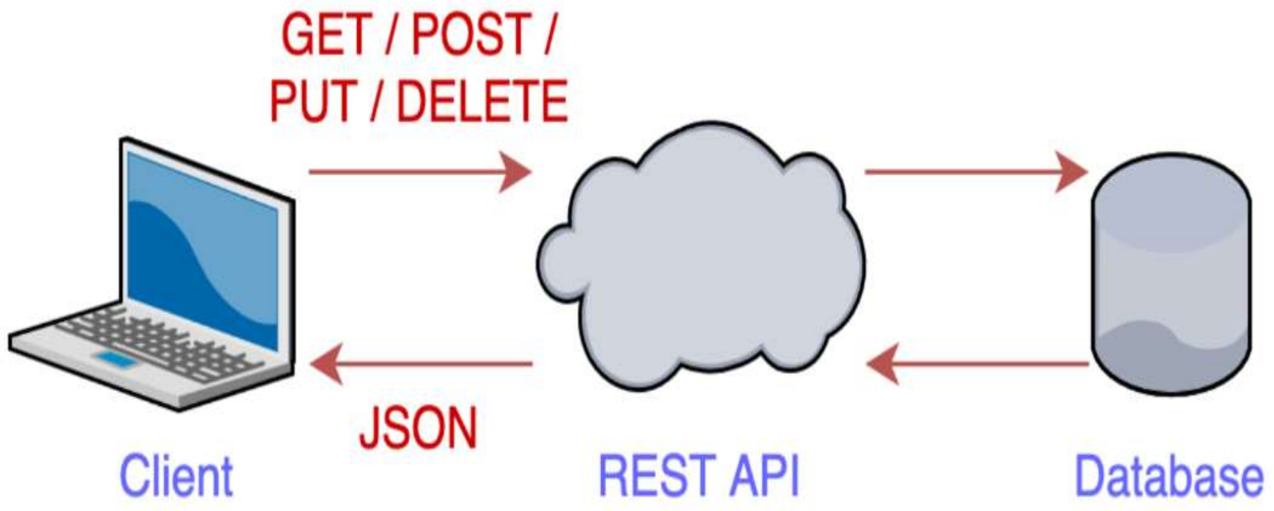
1. Asynchronous programming is a design pattern which ensures the non-blocking code execution.

2. Asynchronous I/O is a form of input/output processing that permits other processing to continue before the transmission has finished.



```
dns.lookup('localhost', function dnscb(er, res) {
  console.log(res)
})

fs.readFile('/proc/${process.pid}/status',
           function fscb(er, data) {
  console.log(data.toString())
})

http.get('http://www.example.com', (res) => {
  res.on('data', function netcb(d) {
    console.log(d.toString())
  })
})

setTimeout(function timercb() {
  console.log('I am timed out')
}, 1000)

console.log('Hello event driven world!')
```

17. REST API's(GET, POST, PUT, DELETE ,UPDATE)?



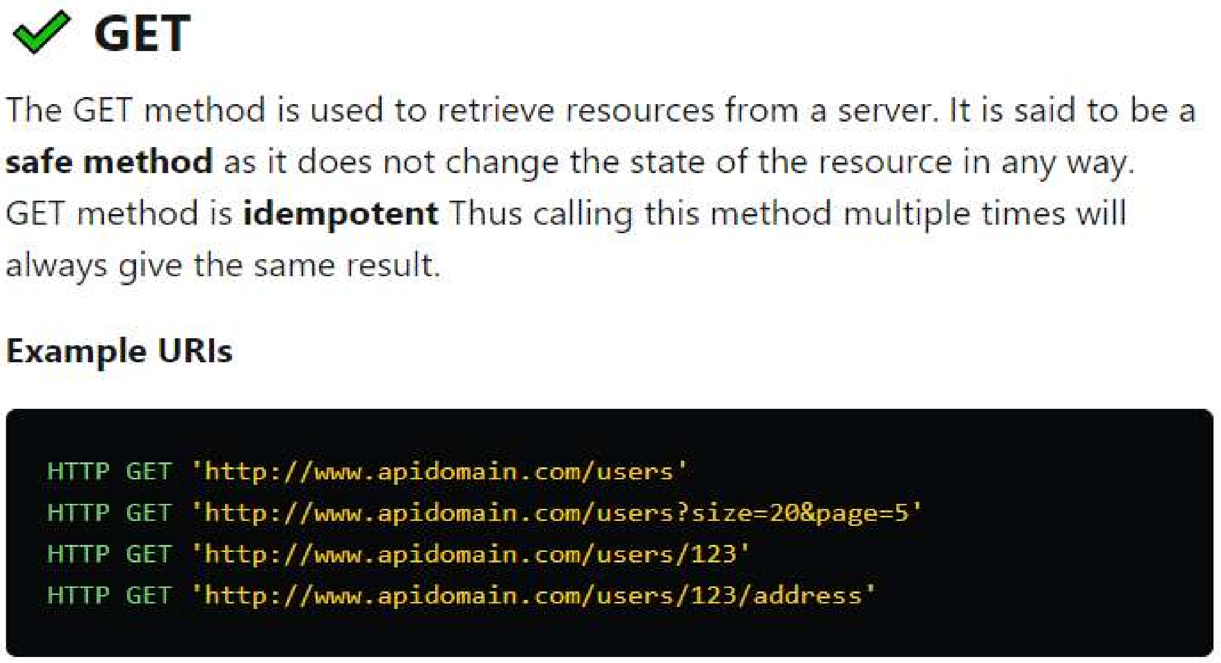
REST API stands for REpresentational State Transfer , which defines a set of principles on how Web standards are supposed to be used. This is a very common standards that a huge majority of the web / mobile applications run on.

The goals of REST are:

- Generality interfaces
- Reduce latency and thus improve performance
- Scalability of interactions between components
- Reliable

To work with REST API efficiently, It’s important to know the basic HTTP methods and what each of them does. Out of 39 http methods, developers typically use the **"GET", "POST", "PUT", "PATCH" and "DELETE"** methods.

Now, I will be showing you the difference between the 5 methods mentioned above and when to use them.



✓ GET

The GET method is used to retrieve resources from a server. It is said to be a **safe method** as it does not change the state of the resource in any way. GET method is **idempotent** Thus calling this method multiple times will always give the same result.

Example URIs

```
HTTP GET 'http://www.apidomain.com/users'
HTTP GET 'http://www.apidomain.com/users?size=20&page=5'
HTTP GET 'http://www.apidomain.com/users/123'
HTTP GET 'http://www.apidomain.com/users/123/address'
```

✓ POST

POST method is used to create a new resource into the collection of resources on a server.

> It is important to note that POST is Non-idempotent. Thus invoking two identical POST requests will result in duplicate information being created on the server.

Example URIs

```
HTTP POST 'http://www.apidomain.com/users'
HTTP POST 'http://www.apidomain.com/users/123/accounts'
```

✓ PUT

PUT is used to update the existing resource on the server and it updates the full resource.
If the resource does not exist, PUT may decide to create a new resource.
PUT method is **idempotent** Thus calling this method multiple times will always update the same resource multiple times.

Example URIs

```
HTTP PUT 'http://www.apidomain.com/users/123'
HTTP PUT 'http://www.apidomain.com/users/123/accounts/456'
```

✅ PATCH

PATCH is used to update the existing resource on the server and it updates a portion of the resource.
If the resource does not exist, PUT may decide to create a new resource.
Just as the PUT method, PATCH is also **idempotent**

Example URIs

```
HTTP PATCH 'http://www.apidomain.com/users/123'
HTTP PATCH 'http://www.apidomain.com/users/123/accounts/456'
```

✨PATCH vrs PUT

PUT method primarily fully replaces an entire existing resource but PATCH partially updates an existing resource.

> The PATCH method is not a substitute to the PUT method. It applies a delta (diff) rather than replacing the entire resource.

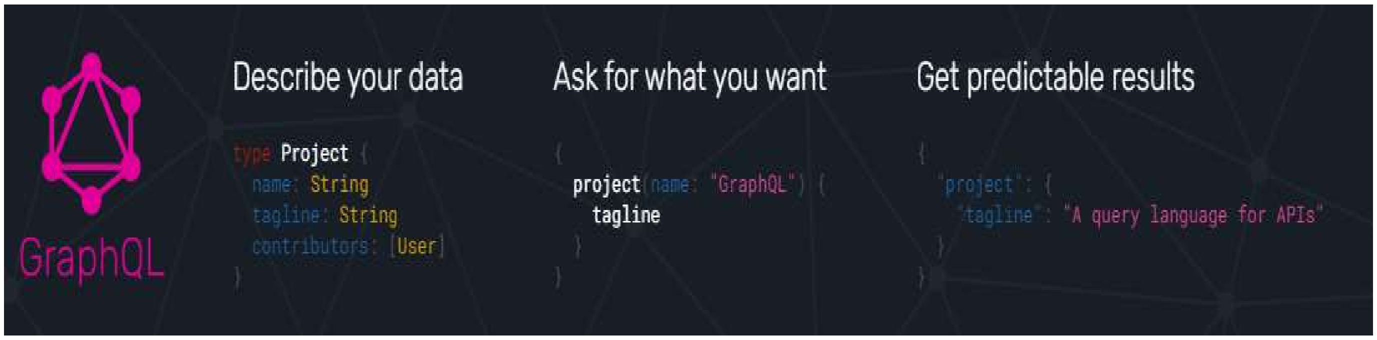
✅ DELETE

DELETE Method is used to delete the resources from a server. It deletes resource identified by the Request-URI.

DELETE method are idempotent.

18. GraphQL?

- GraphQL is a query language and server-side runtime for application programming interfaces (APIs) that prioritizes giving clients exactly the data they request and no more.
- GraphQL is designed to make APIs fast, flexible, and developer-friendly.
- GraphQL was developed by Facebook, which first began using it for mobile applications in 2012. The GraphQL specification was open sourced in 2015. It is now overseen by the GraphQL Foundation.
- The most common GraphQL operations are likely to be **queries, schema, and mutations.** If we were to think about them in terms of create, read, update and delete (CRUD) model, a query would be equivalent to read. All the others (create, update, and delete) are handled by mutations.



```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

19. Promises?

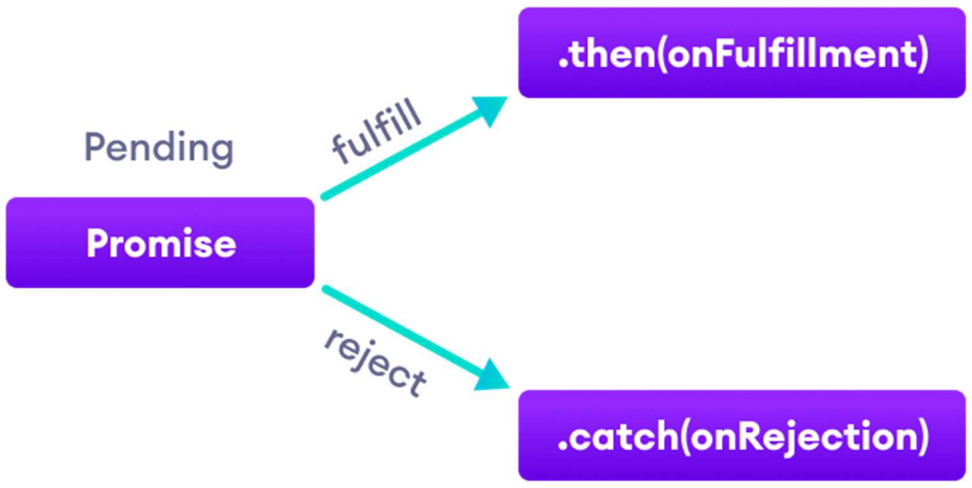
Promise is a good way to handle asynchronous operations. It is used to find out if the asynchronous operation is successfully completed or not.

A promise may have one of three states.

- Pending
- Fulfilled
- Rejected

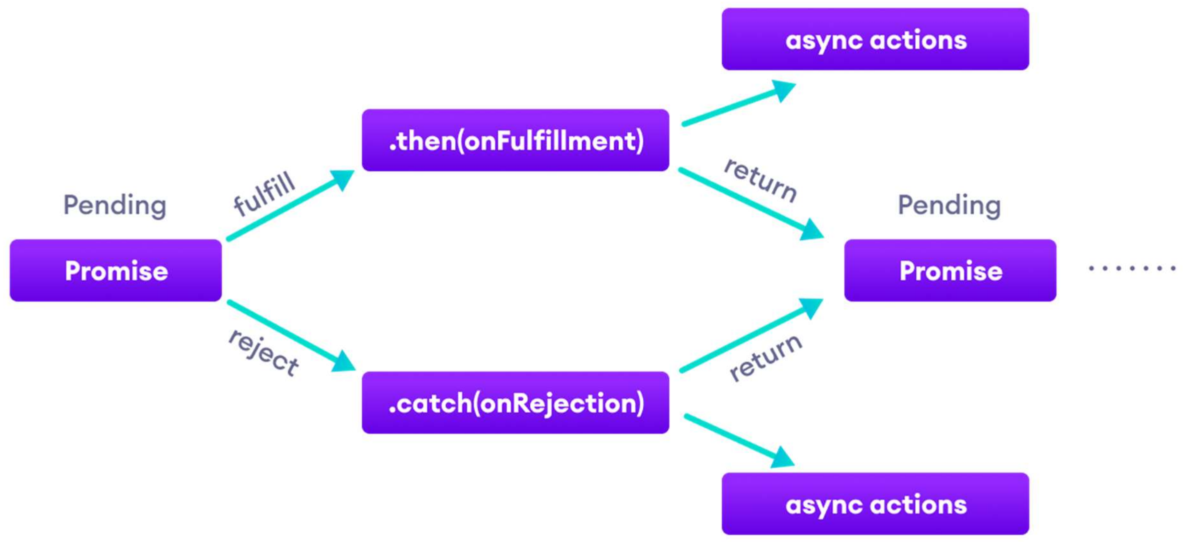
A promise starts in a pending state. That means the process is not complete. If the operation is successful, the process ends in a fulfilled state. And, if an error occurs, the process ends in a rejected state.

For example, when you request data from the server by using a promise, it will be in a pending state. When the data arrives successfully, it will be in a fulfilled state. If an error occurs, then it will be in a rejected state.



20. Promise Chaining?

Promises are useful when you have to handle more than one asynchronous task, one after another. For that, we use promise chaining. You can perform an operation after a promise is resolved using methods then(), catch() and finally().



JavaScript Promise versus Callback?

Promises are similar to callback functions in a sense that they both can be used to handle asynchronous tasks. JavaScript callback functions can also be used to perform synchronous tasks.

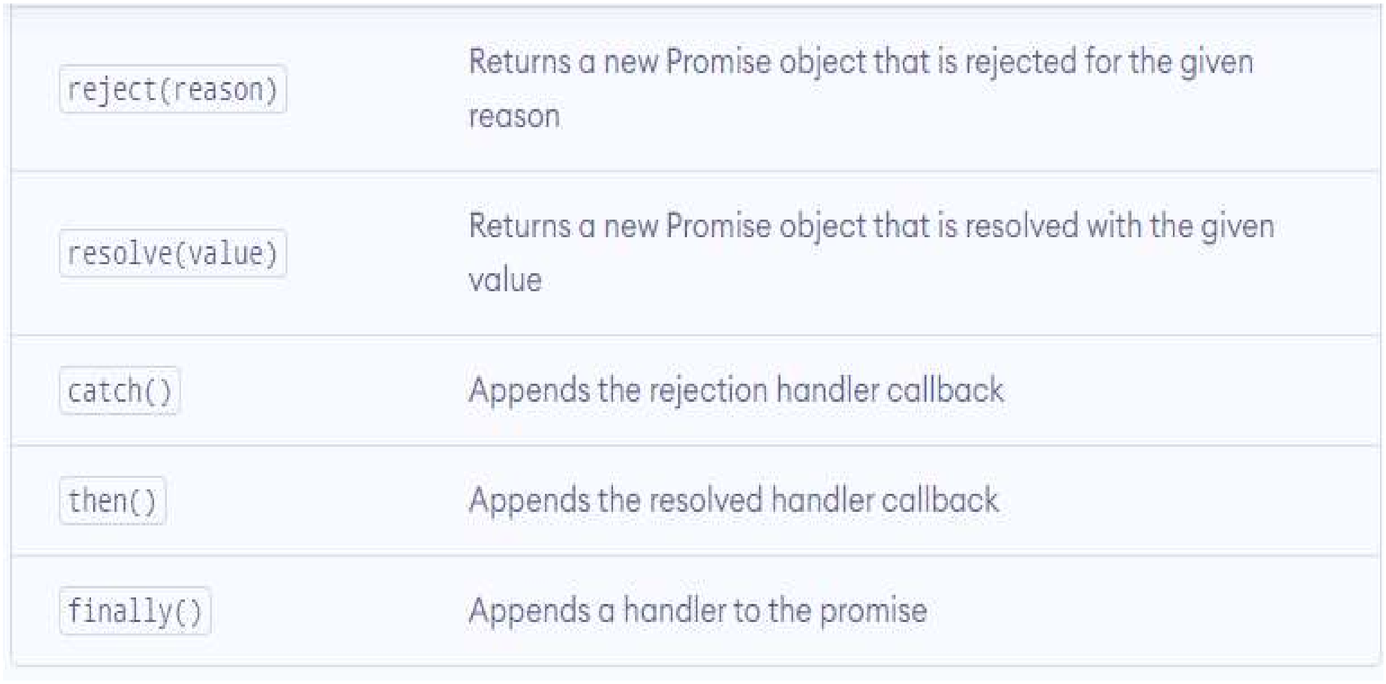
JavaScript Promise

- The syntax is user-friendly and easy to read.
- Error handling is easier to manage.

JavaScript Callback

- The syntax is difficult to understand.
- Error handling may be hard to manage.

JavaScript Promise Methods?



`reject(reason)`	Returns a new Promise object that is rejected for the given reason
`resolve(value)`	Returns a new Promise object that is resolved with the given value
`catch()`	Appends the rejection handler callback
`then()`	Appends the resolved handler callback
`finally()`	Appends a handler to the promise

Introduction to template engine (EJS)?

- EJS or Embedded Javascript Templating is a templating engine used by Node.js. Template engine helps to create an HTML template with minimal code. Also, it can inject data into HTML template at the client side and produce the final HTML.
- EJS is a simple templating language which is used to generate HTML markup with plain JavaScript.

- EJS is very simple, light and fast. It allow us to create HTML Markup with Plain JavaScript. Using EJS as templating engine we need to install EJS

EJS Features?

- Very Simple Syntax
- Use Plain JavaScript
- Very Fast Compilation and Execution
- Compiles with Express View system
- Caching for template and intermediate JavaScript
- Includes feature available
- Easy Debugging