Node.js

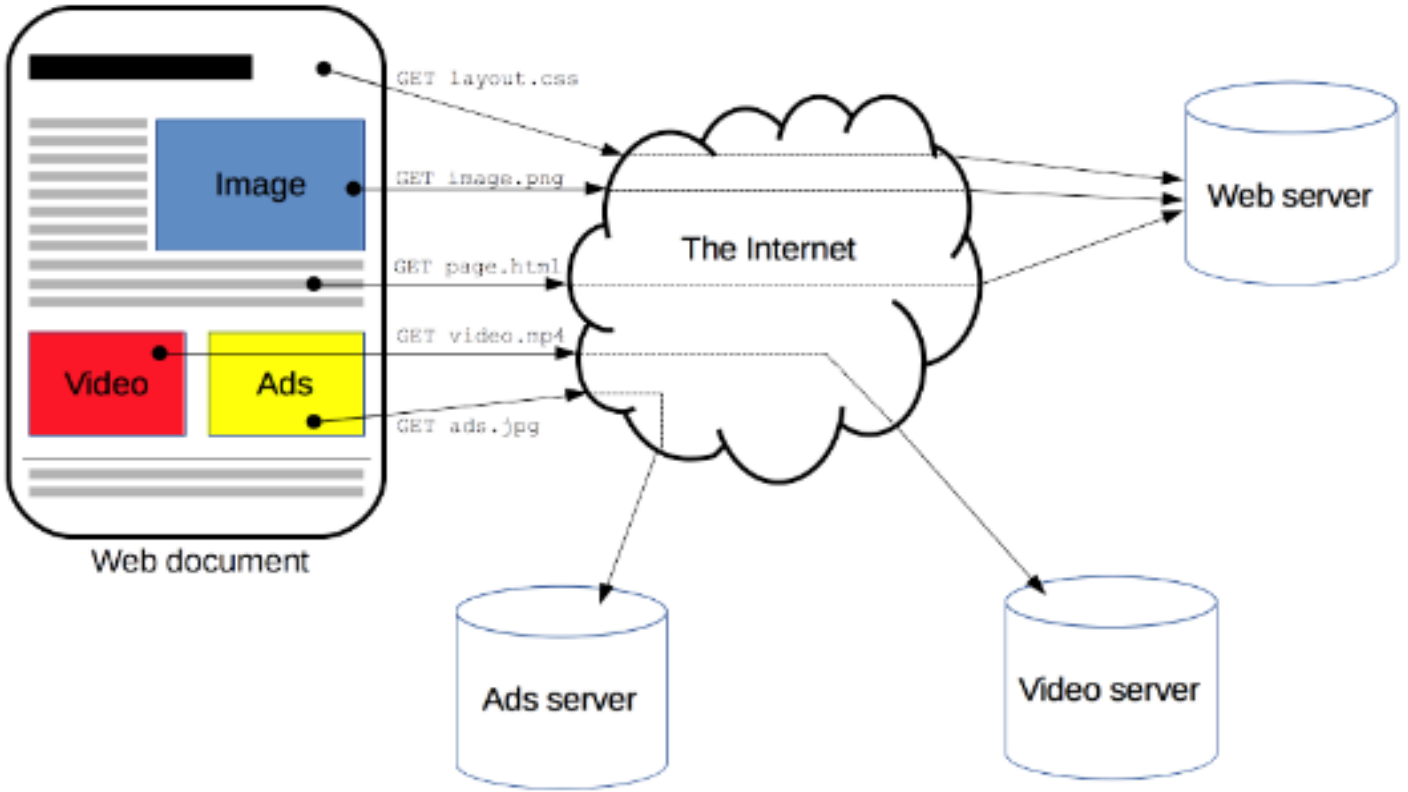
- Web Server and client
- Introduction to Node JS
- Node.js Console
- Node.js Module

Web Server and Client

- **What is Web server?**
  - A web server is a system that delivers content or services to end users over the internet. A web server consists of physical server, server operating system (OS) and software used to facilitate HTTP communication.
  - A server is not necessarily a single machine, but several servers can be hosted on the same machine.

- **What is Client?**
  - A client can be a simple application or a whole system that accesses services being provided by a server.
  - A client can connect to a server through different means like Internet protocols (HTTP).
  - Clients and servers communicate by exchanging individual messages.
  - The messages sent by the client, are called requests and the messages sent by the server as an answer are called responses.

Web Server and Client



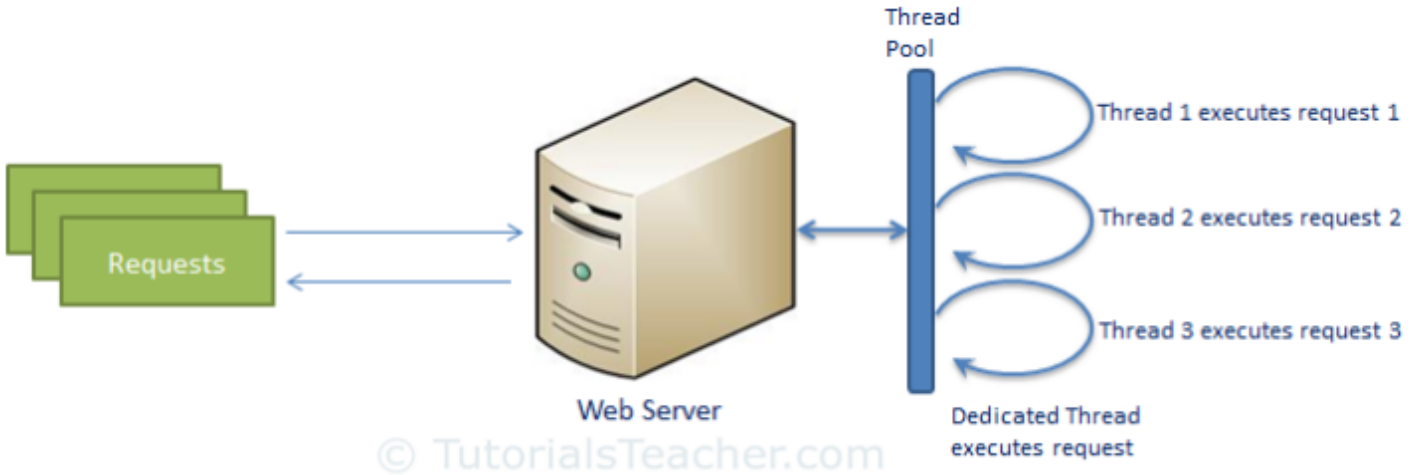
Node.js Introduction

Node.js is a server-side platform built on for easily building fast and scalable network applications.

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

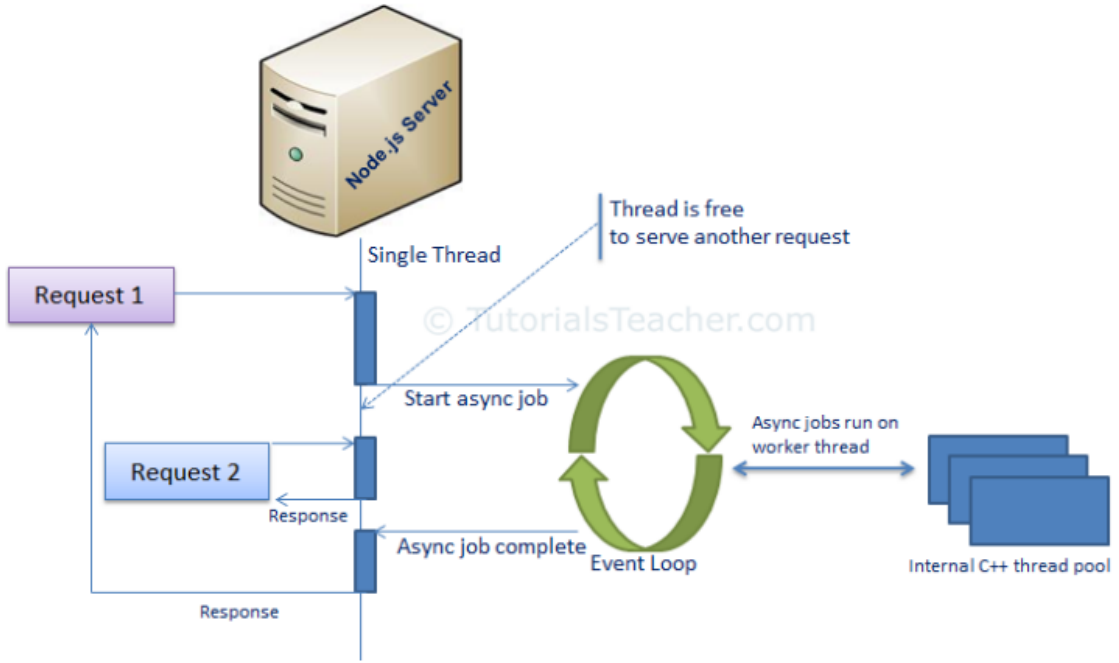
Traditional Web Server Model

- In the traditional web server model, each request is handled by a dedicated thread from the thread pool.
- If no thread is available in the thread pool at any point of time then the request waits till the next available thread.
- Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.



NodeJS Process Model cont.

- Node.js runs in a single process and the application code runs in a single thread.
- All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.
- When asynchronous I/O work completes then it processes the request further and sends the response.



Features of NodeJS

- **Extremely fast:** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
- **Single threaded:** Node.js follows a single threaded model with event looping.
- **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.

Node.js Console - REPL

- Node.js comes with virtual environment called REPL (Node shell).
- It is a quick and easy way to test simple Node.js/JavaScript code.
- To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type ***node*** as shown in the image.
- It will change the prompt to > in Windows and MAC.

Node.js Console – REPL cont.

- You can now test pretty much any Node.js/JavaScript expression in REPL.
- For example, if your write "10 + 20" then it will display result 30 immediately in new line.
- You can even concatenates strings using + operator as in browser's JavaScript.

```
> 10+20
30
>
```

```
> "hello world"
'hello world'
>
```

Node.js Console – REPL cont.

- You can execute an external JavaScript file by writing node “filename” command.
- For example, assume that node-example.js is on C drive of your PC with following code.
- Now, you can execute node-example.js from command prompt as shown here.

node-example.js

```
console.log("Hello World");
```

Node.js Console – REPL cont.

REPL Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

Node.js Module

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.
- It can be a function, can be a class, can be an object or even simple variables.
- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope.
- Each module can be placed in a separate .js file under a separate folder.
- Node.js includes three types of modules:
  - Core Modules
  - Local Modules
  - Third Party Modules

Node.js Core Modules

- The core modules include bare minimum functionalities of Node.js.
- These core modules are compiled into its binary distribution and load automatically when Node.js process starts.
- However, you need to import the core module first in order to use it in your application.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

- The following example demonstrates how to use Node.js http module to create a web server.
- In the given example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().

```
const http = require('http');   // 1 - Import Node.js core module

const port = 5000;

const server = http.createServer((req, res) => {    // 2 - creating server
    //handle incomming requests here..

});

server.listen(port, () => {   //3 - listen for any incoming requests
  console.log(`Server running at port ` + port);
});
```

Loading Core Modules

- In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.
- As per the syntax, specify the module name in the **require()** function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.
- Require function is globally available.

```
var module = require('module_name');
```

Node.js Local Module

- Local modules are modules created locally in your Node.js application.
- These modules include different functionalities of your application in separate files and folders.
- You can also package it and distribute it via NPM, so that Node.js community can use it.
- For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.
- In Node.js, module should be placed in a separate JavaScript file.

Create Local Module

- The ***module.exports*** is a special object which is included in every JS file in the Node.js application by default.
- Use **module.exports** or **exports** to expose a function, object or variable as a module in Node.js.
- The ***module.exports*** in the example exposes a Cat object as a module.

```
1  var Cat = {
2    legs: 4,
3    head: 2,
4    ears: 2,
5    sayHello: function() {
6      console.log('meow');
7    }
8  };
9
10 module.exports = cat;
```

Loading Local Module

```
1 | var cat = require('./cat');
2 |
3 | console.log(cat.legs)
4 | cat.sayHello();
```

```
C:\> node app.js
Info: Node.js started
```

Loading Local Module

- To use local modules in your application, you need to load it using require() function in the same way as core module and specify the path of JavaScript file of the module.
- First, it loads the Cat module using require() function and specified path (here the path './cat‘) where Cat module is stored.
- Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.
- The require() function returns a cat object because Cat module exposes an object in cat.js using module.exports. So now you can use Cat module as an object and call any of its function using dot notation e.g. cat.sayHello()
- Run the above example using command prompt as shown below.

```
1 | var cat = require('./cat');
2 |
3 | console.log(cat.legs)
4 | cat.sayHello();
```

```
C:\> node app.js
Info: Node.js started
```

Export Literals

message.js

```
1  module.exports = 'hello world'
2
```

app.js

```
var msg = require('./message.js');
console.log(msg)
```

```
C:\> node app.js
Hello World
```

Export Object

In the example, require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Message.js

```
exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';
```

app.js

```
var msg = require('./Messages.js');

console.log(msg.SimpleMessage);
```

```
C:\> node app.js
Hello World
```

Export Function

Log.js

```
module.exports = function (msg) {
    console.log(msg);
};
```

app.js

```
var msg = require('./Log.js');

msg('Hello World');
```

```
C:\> node app.js
Hello World
```

Module.exports VS export

- The exports object is just a reference to module.exports.

```
module.exports = {
}


exports = module.exports
```

Exercise

- Write a JavaScript code to display “Hello World” and run it in command prompt.
- Let's try out the Node console and execute a command. Drop into the node console by typing node. Now try the following:
  - Add 30 + 40.
  - Display “Hello World” by concatenating the strings “Hello” and “World” using + operator and console the output.
  - Find out how many seconds are there in a year.
  - How many seconds there are in a century.
- Create a file called app.js and add a line that uses console.log() to tell Node write "Hello World" to the console. Run your program from the console using the node command. You should see hello world.

Exercise

- Create a file app.js
- Require the 'http' core module in your app.js.
- Assign a port number of your choice.
- Create a server to and listen to the port.
- Console the message “server running on port: port #”.

Exercise

- Create a module that returns a date and time object using export function method.
- Create another file which has your create server code.
- Include the module in your server file and display the current date and time in the following format

    "The date and time are currently: "


Any query?