- Nodejs Architecture
- Multithreading
- Node version manager
- Global Object
    - Module
        - Builtin module
            - fs module
            - Query String
            - path
            - Process
            - Child Process
        - Custom Modules
        - Third Party Modules
        - Multiple imports and combined export
- Home Work
- Request Methods
- Http Server
    - Basic Server app
    - Html page using http server
- Template Engines

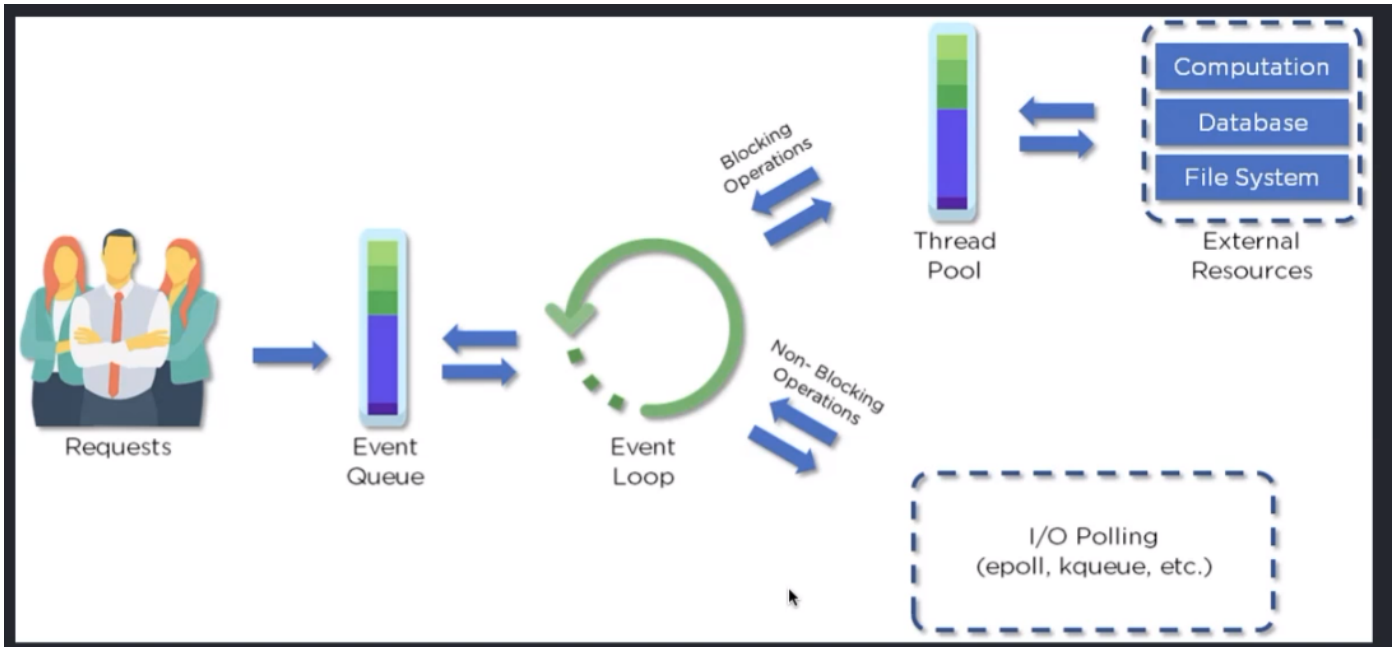
A runtime environment developed using C/C++ that contents google chrome v8 engine and used to execute javascript code outside the browser. V8 is really the fast engine to execute JS code.

JS Conf Youtube channel

Nodejs Architecture

JS app is always a single threaded application. It is still very fast due to asynchronous behavior.

Article



The Node.js runtime architecture follows an event-driven, non-blocking I/O model, which allows it to handle concurrent requests efficiently. Here's a high-level overview of the Node.js runtime architecture workflow:

1. Event Loop:
    - The Event Loop is the heart of the Node.js architecture. It manages and coordinates all incoming and outgoing events.
    - It constantly checks for new events or tasks in the event queue and executes them in a loop.
    - The event loop is responsible for handling I/O operations, timers, callbacks, and other asynchronous tasks.
2. Event-driven I/O:
    - Node.js utilizes non-blocking I/O operations to handle requests efficiently.
    - When a request is received, Node.js initiates the I/O operation (e.g., reading from a file, making a network request) and registers a callback function.
    - While waiting for the I/O operation to complete, Node.js moves on to handle other requests without blocking the execution of subsequent code.
3. Callbacks and Event Emitters:
    - When an I/O operation or a task completes, Node.js triggers the associated callback function or emits an event.
    - Callback functions are provided as arguments to asynchronous functions and are called once the operation is finished.

- Event Emitters allow the creation and handling of custom events. They emit events that can be listened to by registered event listeners.

4. Event Queue:

   - The Event Queue holds events and callbacks to be processed by the Event Loop.
   - When an event or callback is triggered, it is placed in the event queue for the Event Loop to process.

5. Microtask Queue:

   - The Microtask Queue holds microtasks, which are usually promises or certain types of callbacks with higher priority than regular events.
   - Microtasks are executed before the next cycle of the event loop begins, ensuring timely execution of certain operations.

6. Timers:

   - Node.js includes a timer module that allows scheduling tasks to be executed after a specified time delay or at regular intervals.
   - Timers are managed by the Event Loop and trigger the associated callbacks when the specified time elapses.

7. Worker Threads (optional):

   - Node.js also provides a Worker Threads module that allows for the creation of additional threads to handle computationally intensive tasks in parallel.
   - Worker Threads enable multi-threaded execution, but developers need to explicitly manage the communication and synchronization between threads.

Overall, the event-driven, non-blocking I/O model, along with the event loop, allows Node.js to handle multiple requests concurrently and efficiently utilize system resources, resulting in high performance and scalability.

Multithreading

We can achieve this using either child processes or clusters. It is not recommended.

For Streaming like apps, Nodejs is the best choice. For CRUD apps, you may choose any tool available.

Node version manager

Node version manager commands

```
npm i -g nvm
nvm ls
```

Global Object

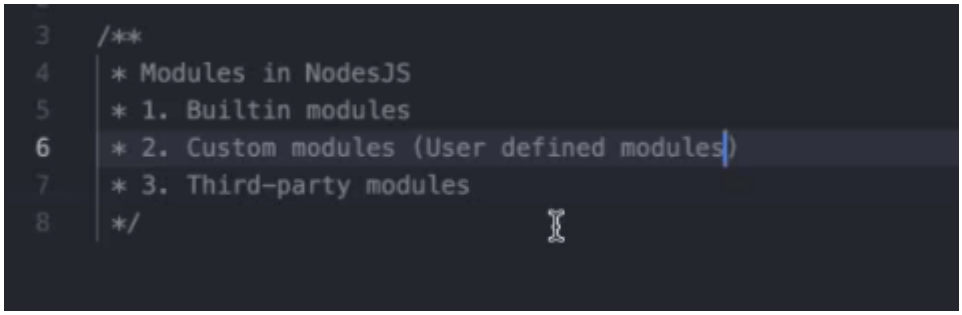
```
console.log("Hello World");

/**
 * Global Object
 *  - console
 *  - setTimeout
 *  - clearTimeout
 *  - setInterval
 *  - clearInterval
 *  - ...
 */
console.log(global);

global.setTimeout(() => {
  console.log("Hello World");
}, 1000);

/**
 * Module Object
 *  - exports
 *  - require
 *  - module
 *  - __filename
 *  - __dirname
 *  - ...
 */
console.log(module);
```

Module



Builtin module

fs module

```
const fs = require("fs");
/**
 * fs is a core module
 *  1. Core Module
 *  2. File or Folder
 *
 */
console.log(fs);

fs.writeFileSync("test.txt", "Hello World");

const data = fs.readFileSync("test.txt");
console.log(data); // <Buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64>
console.log(data.toString());
```

Query String

```
const queryString = require("querystring");
const obj = {
  name: "Sajjad Ali",
  age: 23,
};
console.log("Original object " + JSON.stringify(obj)); // { name: 'Sajjad Ali',
age: 23 }
const str = queryString.stringify(obj);
console.log("query string from object " + str); // name=Sajjad%20Ali&age=23
const res = queryString.parse(str);
console.log("Parsed: " + JSON.stringify(res)); // { name: 'Sajjad Ali', age: '23' }
```

path

```
const path = require("path");

const pathObj = path.parse(__filename);

console.log(pathObj);

const pathObj2 = path.join("api", "users", "login");
console.log(pathObj2); // api\users\login

const pathObj3 = path.join(__dirname, "api", "users", "login");
console.log(pathObj3); // d:\BSCS VIII\EAD\practice\Lecture
09\modules\api\users\login

const normalizedPath = path.normalize("api//users/login");
console.log(normalizedPath); // api\users\login

const absolutePath = path.isAbsolute("api//users/login");
console.log(absolutePath); // false

const absolutePath2 = path.isAbsolute(__filename);
console.log(absolutePath2); // true

const absolutePath3 = path.isAbsolute(
  "D:\\BSCS VIII\\EAD\\practice\\Lecture 09\\modules\\api\\users\\login"
);
console.log(absolutePath3); // true

const ext = path.extname(__filename);
console.log(ext); // .js
```

Process

```
const process = require("process");

console.log(process.pid);
console.log(process.versions.node);
console.log(process.argv);

const num = process.argv[2];

switch (num) {
  case "1":
    console.log("One");
    break;
  case "2":
    console.log("Two");
}

process.kill(process.pid);
```

Child Process

```
const child_process = require("child_process");
const res = child_process.spawn("notepad", ["processMod.js"]);
```

Custom Modules

`lib.js` file

```
module.exports = {
  name: "Sajjad Ali",
  age: 23,
};
```

`index.js` file

```
const lib = require("./lib");
console.log(JSON.stringify(lib));
```

Third Party Modules

First Install the module via npm:

```
npm install express
```

You can access it via following:

```
const express = require("express");
```

Multiple imports and combined export

```
module.exports = {
  name: "Sajjad Ali",
```

```
  age: 23,

  A: require("./A"),
  B: require("./B"),
  C: require("./C"),
};
```

Home Work

- Profiler Apis
- Redux
- Reducers
- Custom Hooks
- npm package vs library

Library is not executable. However, the packages are.

Request Methods

Docs

- PUT
- POST
- GET
- PATCH
- PUT
- DELETE

Http Server

Basic Server app

Run the following js code and then open browser.

Launch

localhost:3000/

localhost:3000/about

localhost:3000/dummy

```
const http = require("http");

const server = http.createServer((request, response) => {
  console.log(request.url);
  switch (request.url) {
    case "/":
      response.write("Hello World");
      break;
    case "/about":
      response.write("About Page");
      break;
    default:
      response.write("Not Found");
      break;
  }

  response.end();
});

server.listen(3000, () => {
  console.log("Server is running on port 3000");
});
```

Html page using http server

home.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>About</title>
  </head>

  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

about.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>

  <body>
    <h1>About Page</h1>
  </body>
</html>
```

notFound.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>

  <body>
    <h1>Page Not Found</h1>
  </body>
</html>
```

http.js

```
const http = require("http");
const fs = require("fs");
const path = require("path");

const server = http.createServer((request, response) => {
  console.log(request.url);
  switch (request.url) {
    case "/":
      sendResponse(response, "home.html");
      break;
    case "/about":
      sendResponse(response, "about.html");
      break;
    default:
      sendResponse(response, "notFound.html");
      break;
```

```
  }
});

function sendResponse(response, filename) {
  const joinedPath = path.join(__dirname, filename);
  const data = fs.readFileSync(joinedPath).toString();
  response.end(data);
}

server.listen(3000, () => {
  console.log("Server is running on port 3000");
});
```

Template Engines