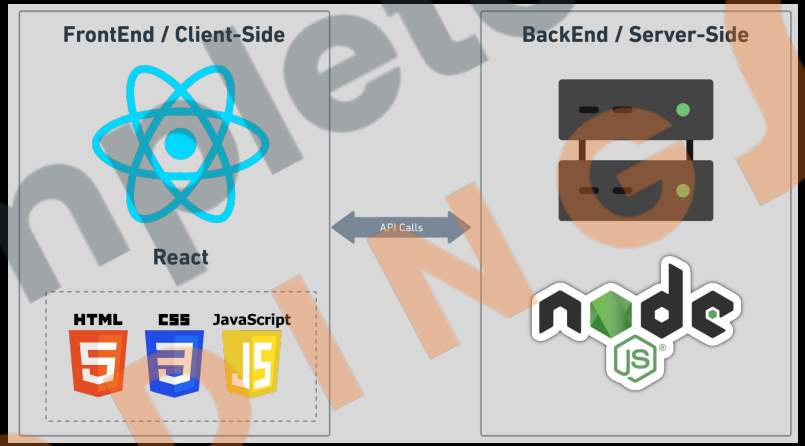
COMPLETE

YouTube Playlist Link

NodeJS

Complete Course



Introduction to NodeJS

1. Pre-requisites
2. What is NodeJS
3. NodeJs Features
4. JavaScript on Client
5. JavaScript on Server
6. Client Code vs Server Code
7. Other uses of NodeJs
8. Server architecture with NodeJs

JS is required for NodeJS

COMPLETE JAVASCRIPT

14 HOURS

MYNTRA PROJECT

+

CERTIFICATE + NOTES

React is recommended before NodeJS

COMPLETE

React + REDUX

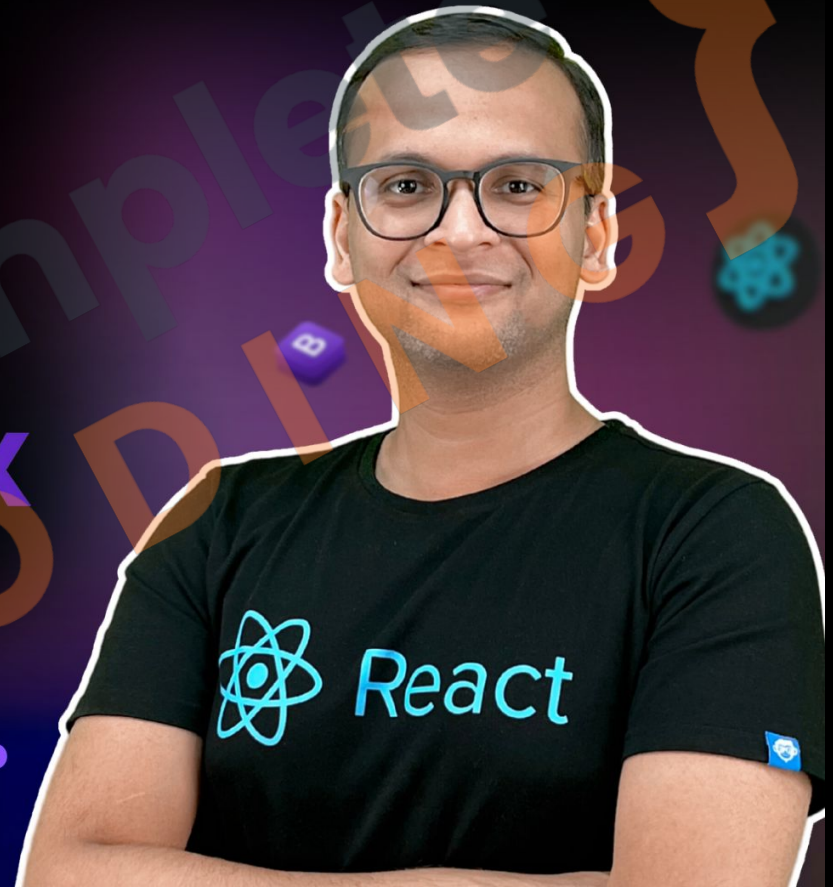
20 HOURS

6 PROJECTS

using Bootstrap

CERTIFICATE

NOTES



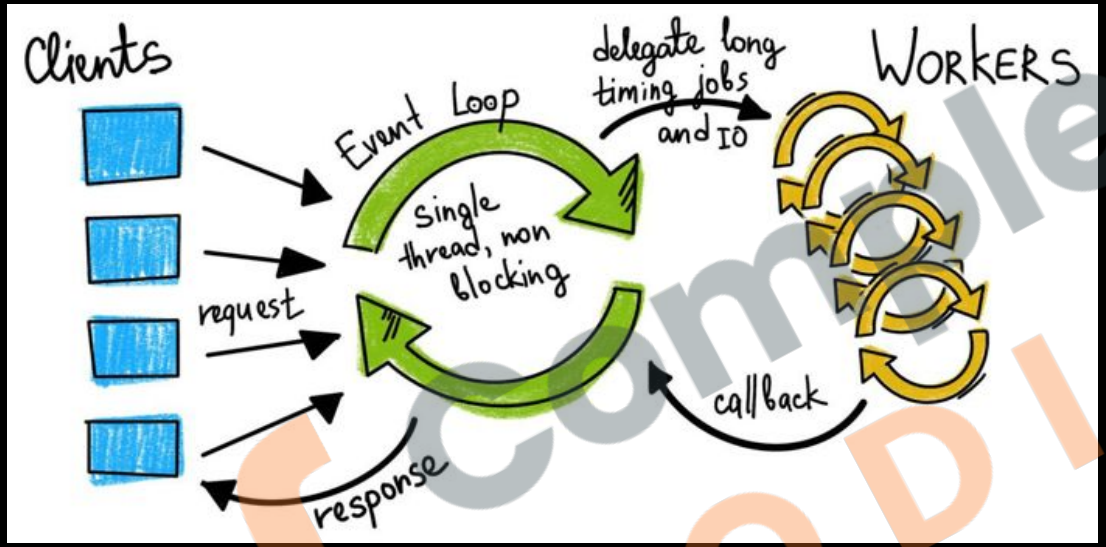
2.What is NodeJS

1. JavaScript Runtime: Node.js is an open-source, cross-platform runtime environment for executing JavaScript code outside of a browser.
2. NodeJs is a JavaScript in a different environment means Running JS on the server or any computer.
3. Built on Chrome's V8 Engine: It runs on the V8 engine, which compiles JavaScript directly to native machine code, enhancing performance.
4. V8 is written in C++ for speed.
5. V8 + Backend Features = NodeJs

2.What is NodeJS

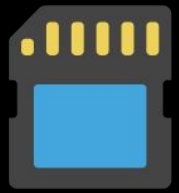
1. Design: Features an event-driven, non-blocking I/O model for efficiency.
2. Full-Stack JavaScript: Allows using JavaScript on both server and client sides.
3. Scalability: Ideal for scalable network applications due to its architecture.
4. Versatility: Suitable for web, real-time chat, and REST API servers.

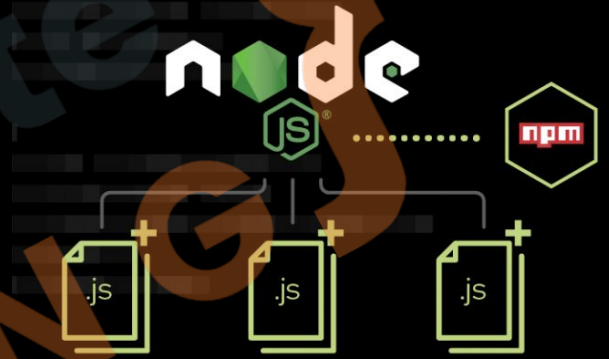
3. NodeJs Features
(Added)



1. Non-blocking I/O: Designed to perform non-blocking operations by default, making it suitable for I/O-heavy operations.
2. Networking Support: Supports TCP/UDP sockets, which are crucial for building lower-level network applications that browsers can't handle.

3. NodeJs Features
(Added)

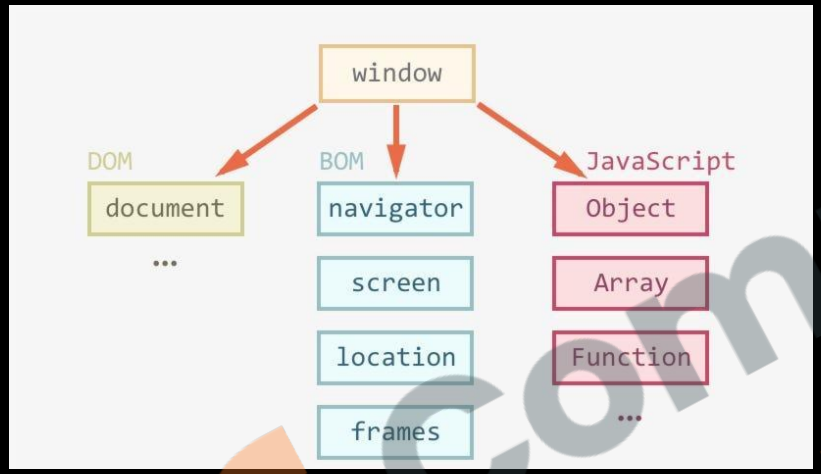


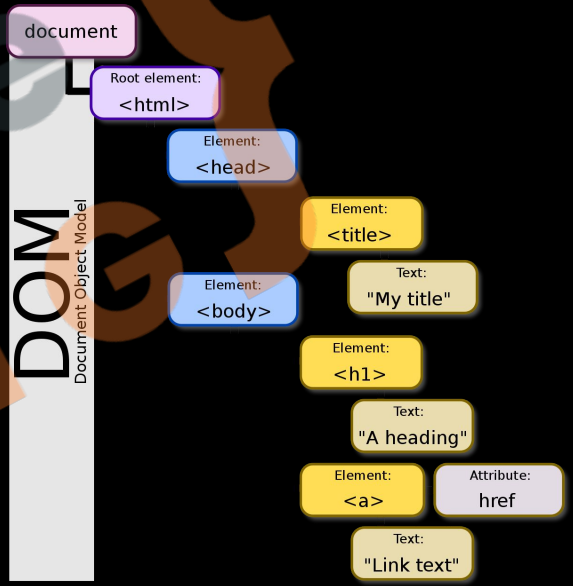


1. File System Access: Provides APIs to read and write files directly, which is not possible in browser environments for security reasons.
2. Server-Side Capabilities: Node.js enables JavaScript to run on the server, handling HTTP requests, file operations, and other server-side functionalities.
3. Modules: Organize code into reusable modules using require().

3. NodeJs Features

(Removed)

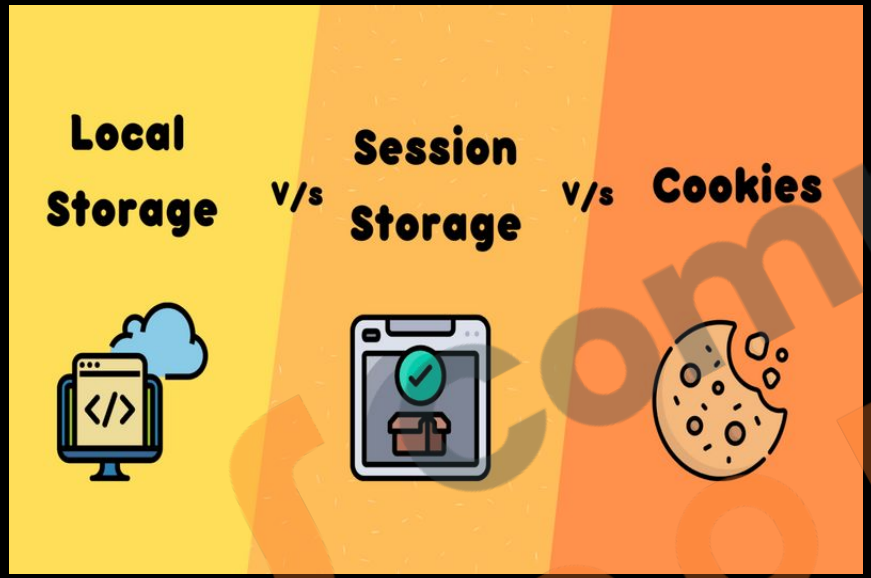


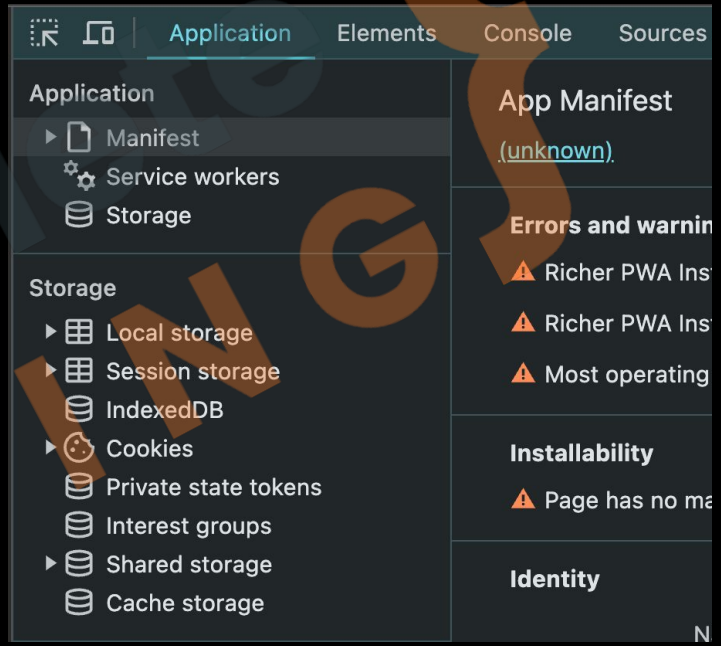


1. Window Object: The global window object, which is part of web browsers, is absent in Node.js.
2. DOM Manipulation: Node.js does not have a built-in Document Object Model (DOM), as it is not intended to interact with a webpage's content.
3. BOM (Browser Object Model): No direct interaction with things like navigator or screen which are part of BOM in browsers.

3. NodeJs Features

(Removed)





Web-Specific APIs: APIs like localStorage, sessionStorage, and browser-based fetch are not available in Node.js.

4. JavaScript on Client

1. Displays Web Page: Turns HTML code into what you see on screen.
2. User Clicks: Helps you interact with the web page.
3. Updates Content: Allows changes to the page using JavaScript.
4. Loads Files: Gets HTML, images, etc., from the server.

5. JavaScript on Server

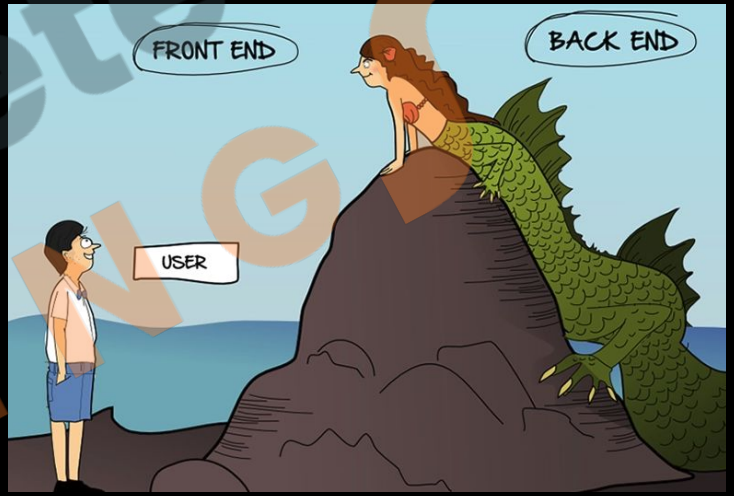
1. Database Management: Stores, retrieves, and manages data efficiently through operations like CRUD (Create, Read, Update, Delete).
2. Authentication: Verifies user identities to control access to the system, ensuring that users are who they claim to be.
3. Authorization: Determines what authenticated users are allowed to do by managing permissions and access controls.
4. Input Validation: Checks incoming data for correctness, completeness, and security to prevent malicious data entry and errors.
5. Session Management: Tracks user activity across various requests to maintain state and manage user-specific settings.

5. JavaScript on Server

6. API Management: Provides and handles interfaces for applications to interact, ensuring smooth data exchange and integration.
7. Error Handling: Manages and responds to errors effectively to maintain system stability and provide useful error messages.
8. Security Measures: Implements protocols to protect data from unauthorized access and attacks, such as SQL injection and cross-site scripting (XSS).
9. Data Encryption: Secures sensitive information by encrypting data stored in databases and during transmission.
10. Logging and Monitoring: Keeps records of system activity to diagnose issues and monitor system health and security.

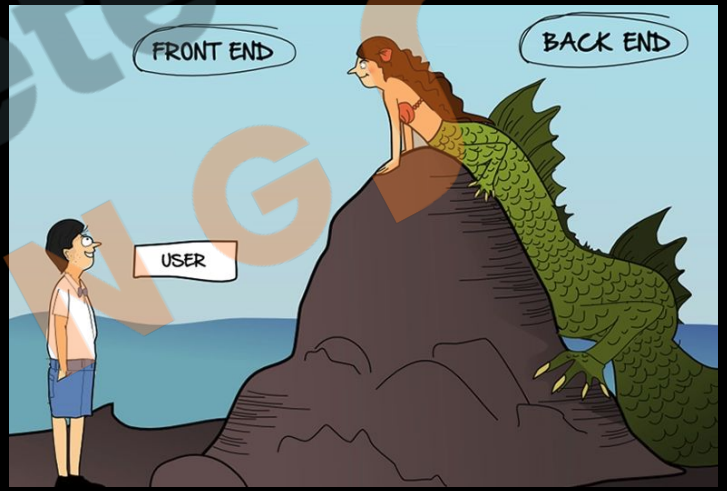
6. Client Code vs Server Code

1. User/client can't access server code directly.
2. Client must raise requests for particular APIs to access certain features or data.
3. Environment Access: Server-side JavaScript accesses server features like file systems and databases.
4. Security: Server-side code can handle sensitive operations securely, while client-side code is exposed and must manage security risks.
5. Performance: Heavy computations are better performed on the server to avoid slowing down the client.



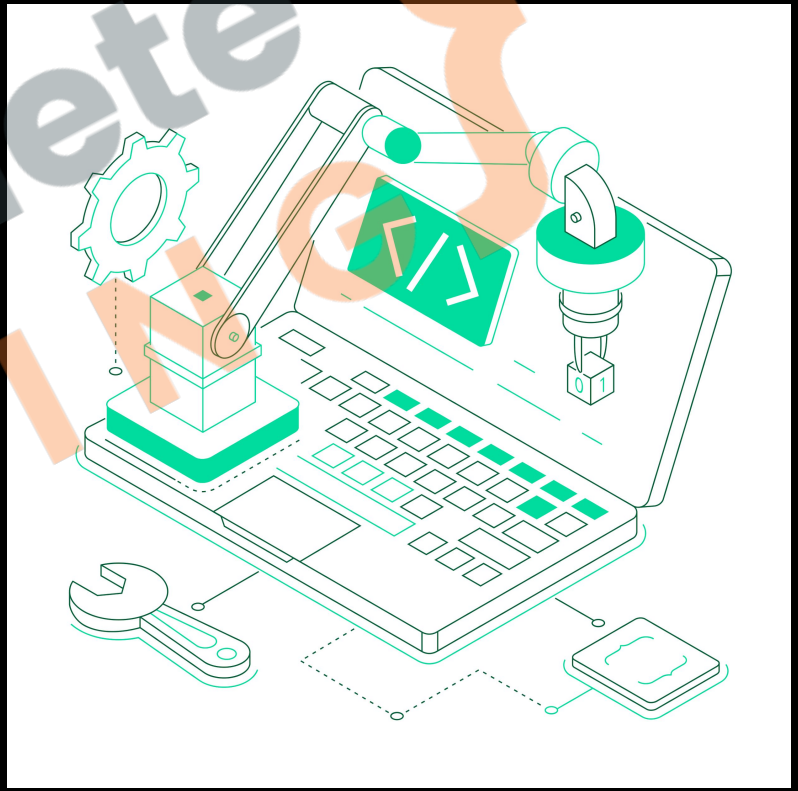
6. Client Code vs Server Code

6. Resource Utilization: Servers generally offer more powerful processing capabilities than client devices.
7. Data Handling: Server-side can directly manage large data sets and database interactions, unlike client-side JavaScript.
8. Asynchronous Operations: Server-side JavaScript is optimized for non-blocking I/O to efficiently manage multiple requests.
9. Session Management: Servers handle sessions and user states more comprehensively.
10. Scalability: Server-side code is designed to scale and handle requests from multiple clients simultaneously.

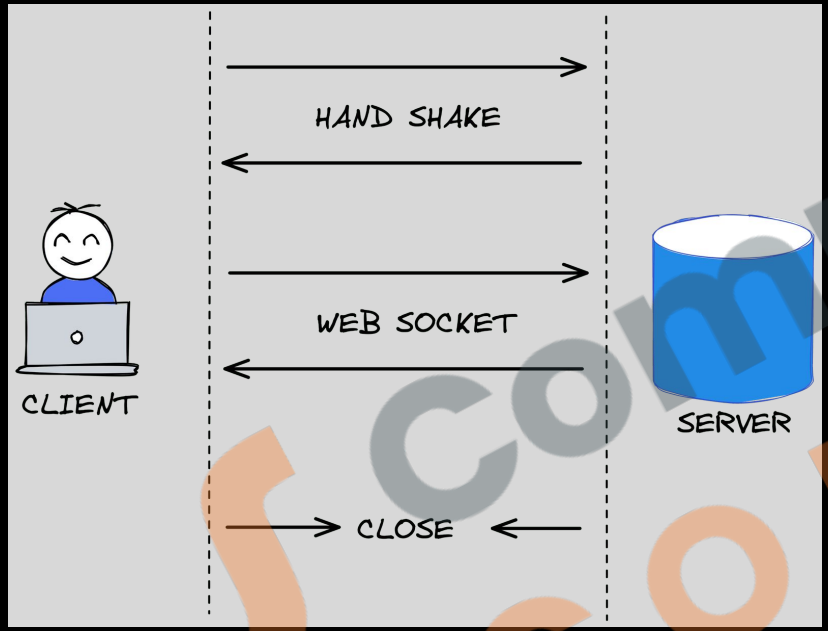


7. Other uses of NodeJs

1. Local Utility Scripts: Automates tasks and processes files locally, like using shell scripts but with JavaScript.
2. Internet of Things (IoT): Develops server-side applications for IoT devices, managing communications and data processing.
3. Scripting for Automation: Automates repetitive tasks in software development processes, such as testing and deployment.



7. Other uses of NodeJs



Real-Time Applications: Efficiently manages real-time data applications, such as chat apps and live updates, using WebSockets.

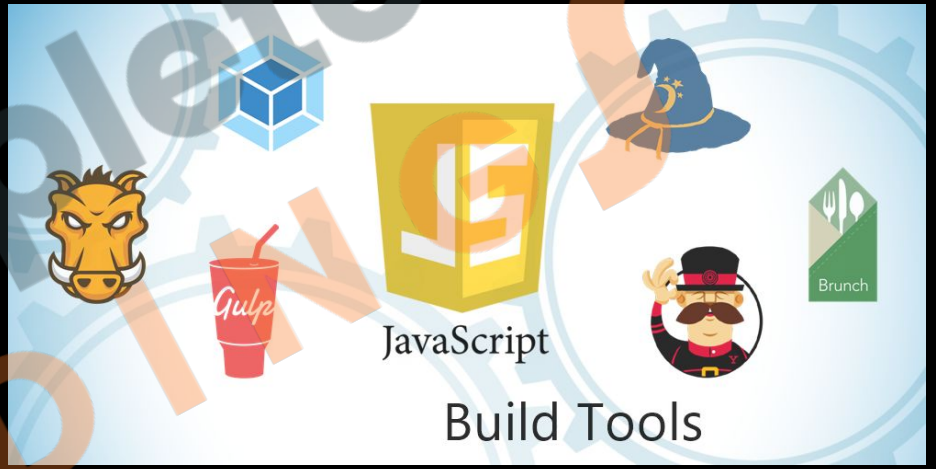
7. Other uses of NodeJs

Desktop Applications: Creates cross-platform desktop applications using frameworks like Electron.

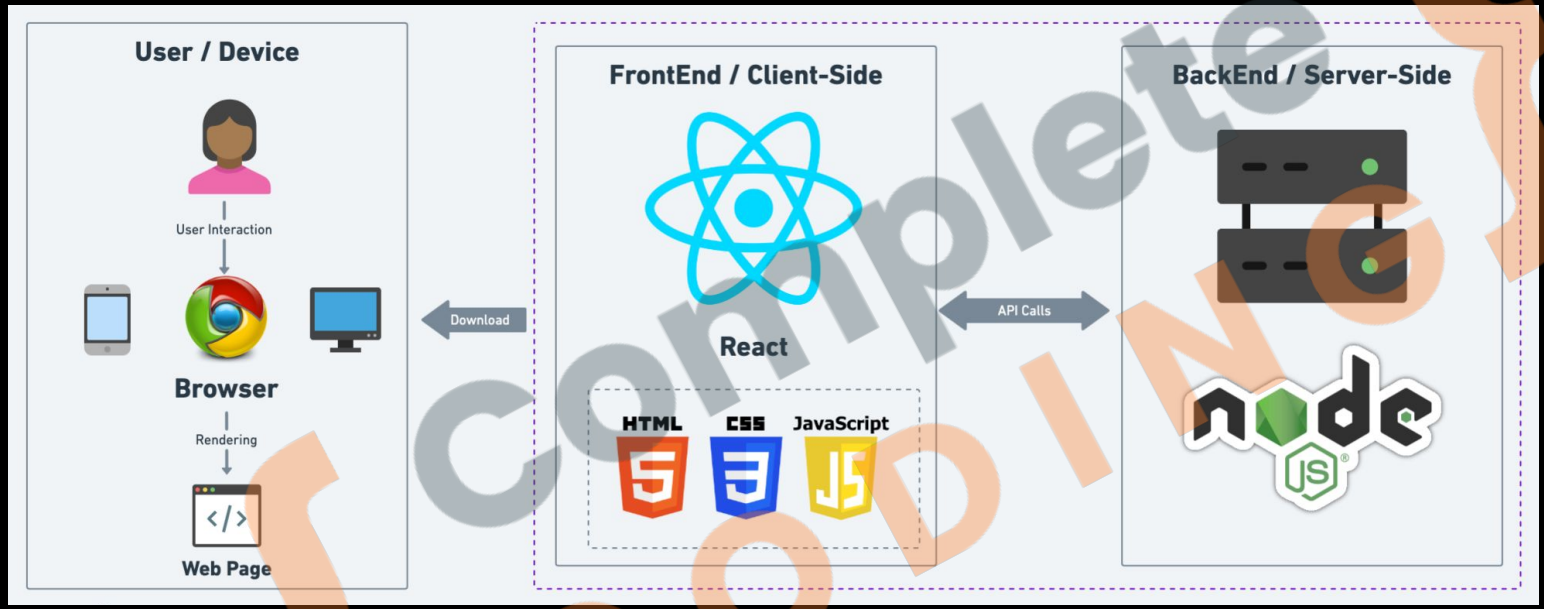
7. Other uses of NodeJs

Build Tools: Powers build processes for front-end technologies using tools like:

- Webpack
- Grunt
- Gulp
- Browserify
- Brunch
- Yeoman



8. Server architecture with NodeJs



Nodejs server will:

1. Create server and listen to incoming requests
2. Business logic: validation, connect to db, actual processing of data
3. Return response HTML, JSON, CSS, JS

Revision

1. Pre-requisites
2. What is NodeJS
3. NodeJs Features
4. JavaScript on Client
5. JavaScript on Server
6. Client Code vs Server Code
7. Other uses of NodeJs
8. Server architecture with NodeJs


node
JS


Complete
{CODING}

2. Installation of NodeJS

1. What is IDE
2. Need of IDE
3. MAC Setup
   - Install latest Node & VsCode
4. Windows Setup
   - Install latest Node & VsCode
5. Linux Setup
   - Install latest Node & VsCode
6. VsCode (Extensions and Settings)
7. Executing first .js file
8. What is REPL
9. Executing Code via REPL

2.1 What is IDE

1. IDE stands for Integrated Development Environment.
2. Software suite that consolidates basic tools required for software development.
3. Central hub for coding, finding problems, and testing.
4. Designed to improve developer efficiency.

2.2 Need of IDE

1. Streamlines development.
2. Increases productivity.
3. Simplifies complex tasks.
4. Offers a unified workspace.
5. IDE Features
    1. Code Autocomplete
    2. Syntax Highlighting
    3. Version Control
    4. Error Checking

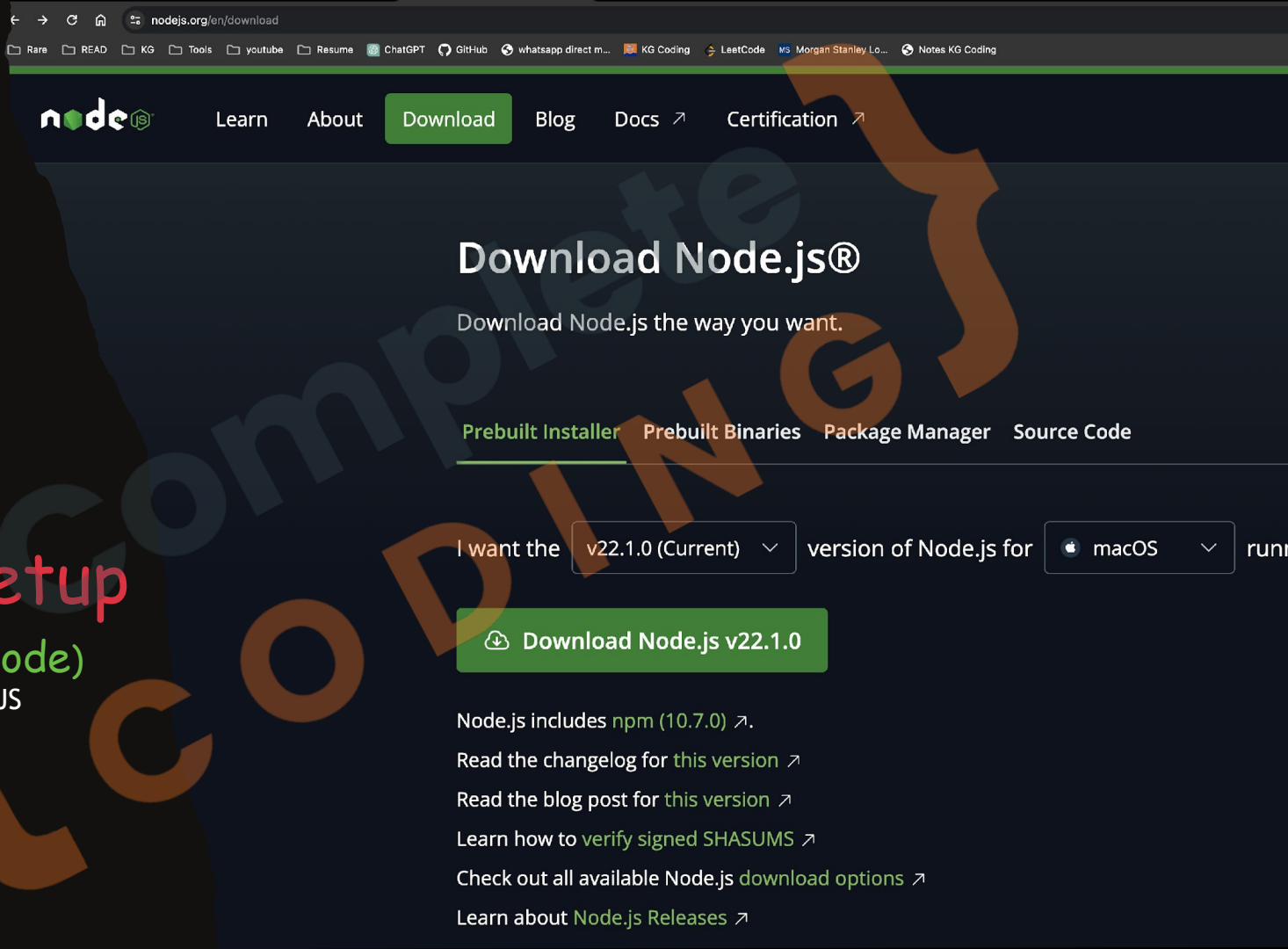
2.3 MAC Setup

mac OS

2.3 MAC Setup

(Install latest Node)

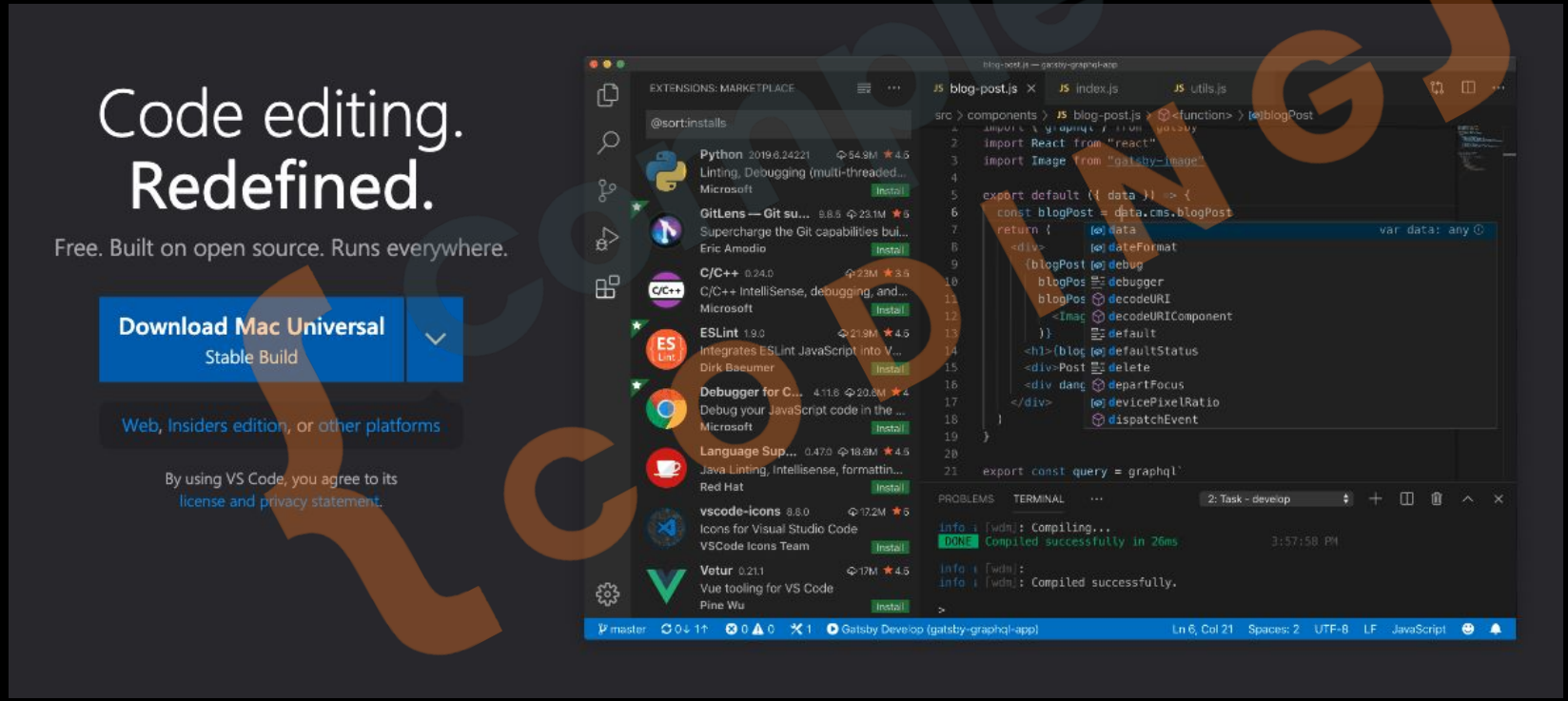
Search Download NodeJS



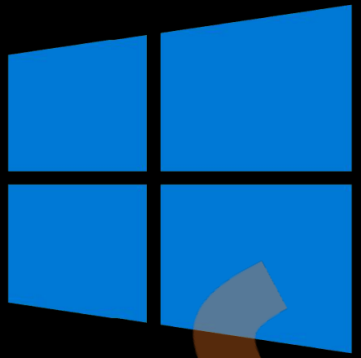
2.3 MAC Setup

(Install VsCode)

Search VS Code on Google



2.4 Windows Setup



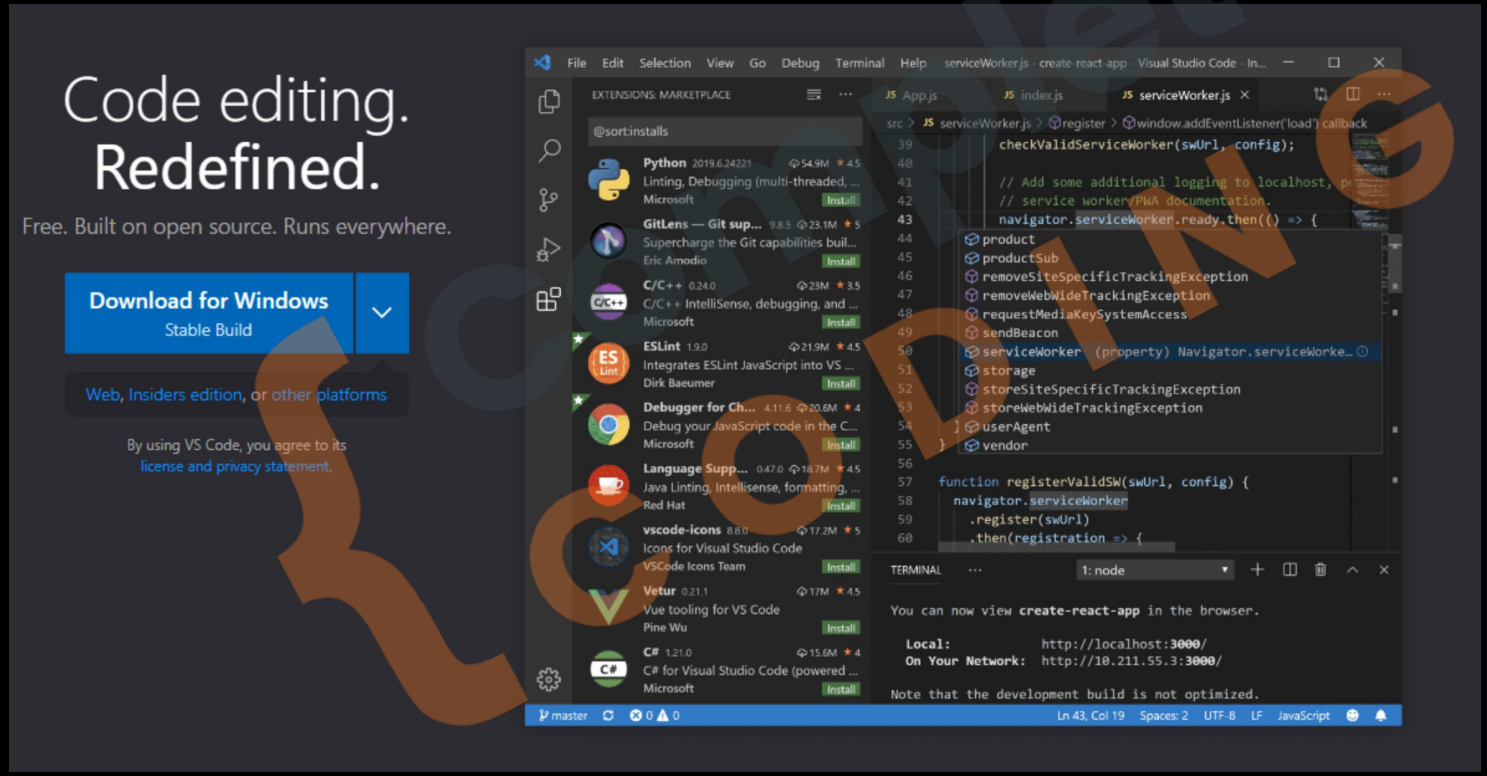
Windows


nodejs.org/en/download
KG
Tools
youtube
Resume
ChatGPT
GitHub
whatsapp direct m...
KG Coding
LeetCode
Morgan Stanley Lo...
Notes KG Coding
Learn
About
Download
Blog
Docs
Certification
Download Node.js®
Download Node.js the way you want.
Prebuilt Installer
Prebuilt Binaries
Package Manager
Source Code
I want the
v22.1.0 (Current)
version of Node.js for
Windows
running
x64
Download Node.js v22.1.0
Node.js includes npm (10.7.0).
Read the changelog for this version
Read the blog post for this version
Learn how to verify signed SHASUMS
Check out all available Node.js download options
Learn about Node.js Releases
2.4 Windows Setup
(Install latest Node)
Search Download NodeJS
Complete
CODING

2.4 Windows Setup

(Install VsCode)

Search VS Code on Google



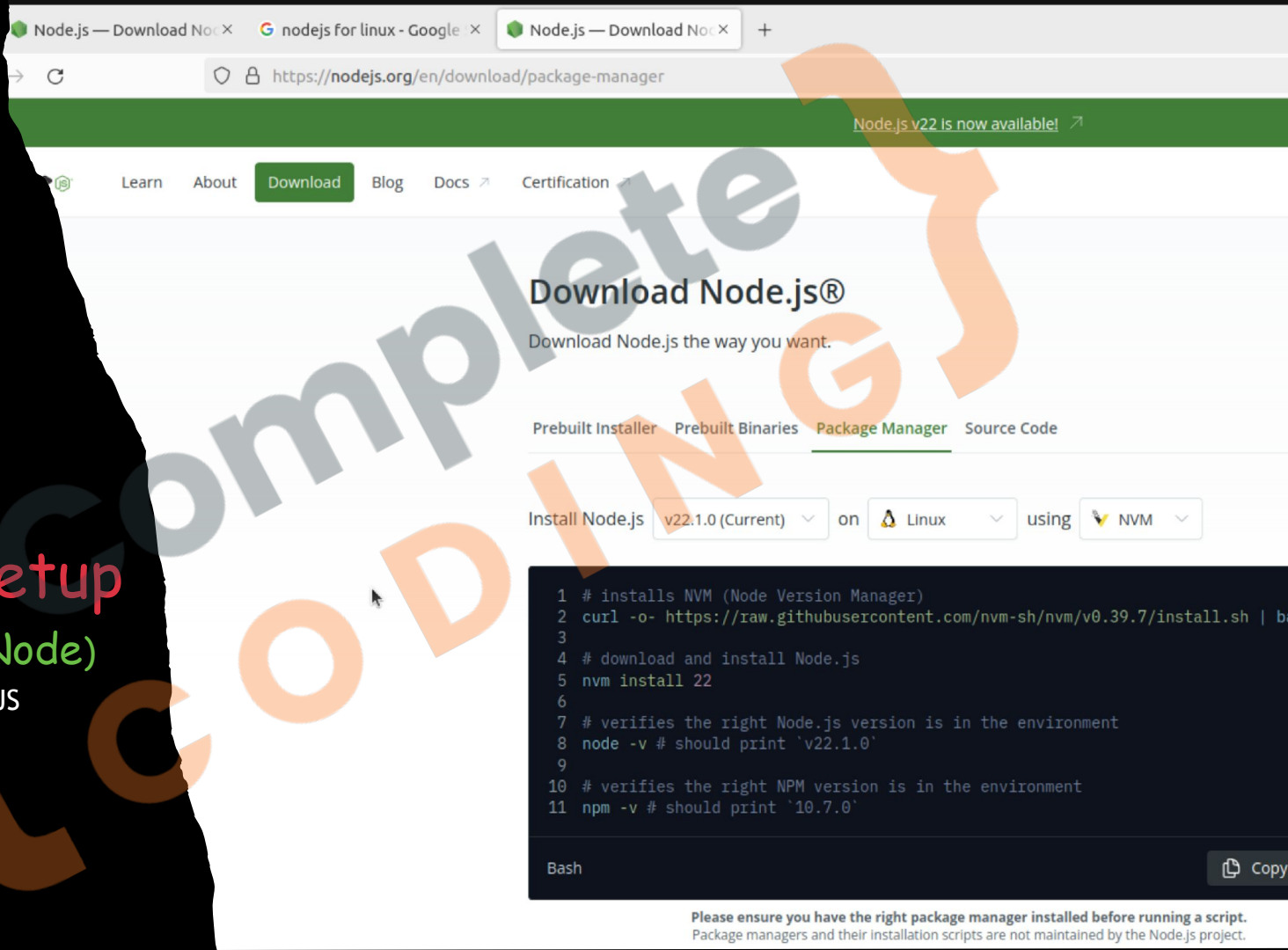
2.5 Linux Setup

Linux

2.5 Linux Setup

(Install latest Node)

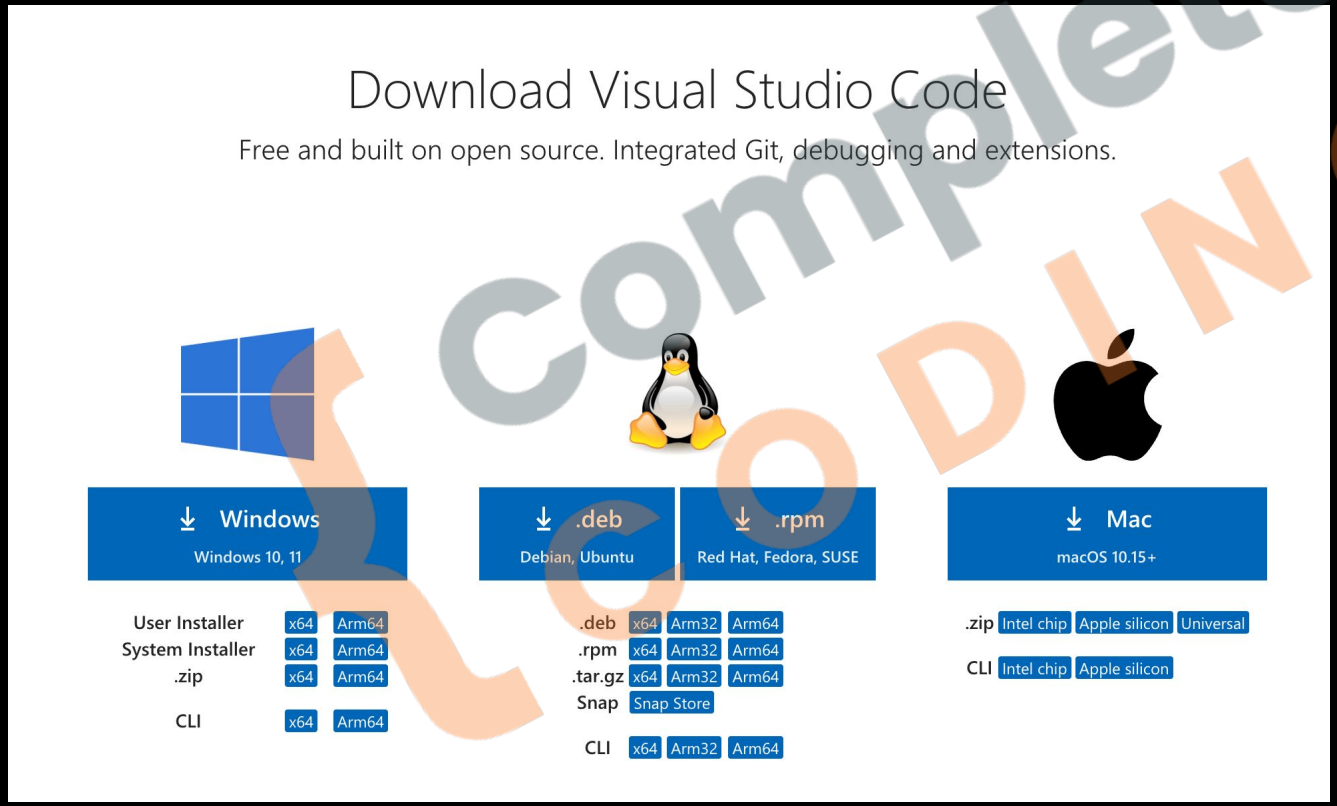
Search Download NodeJS



2.5 Linux Setup

(Install VsCode)

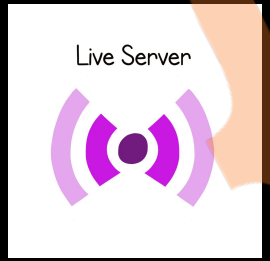
Search VS Code on Google



2.6 VsCode

(Extensions and Settings)

1. Prettier (Format on Save)
2. Line Wrap
3. Tab Size from 4 to 2



2.7 Executing first .js file

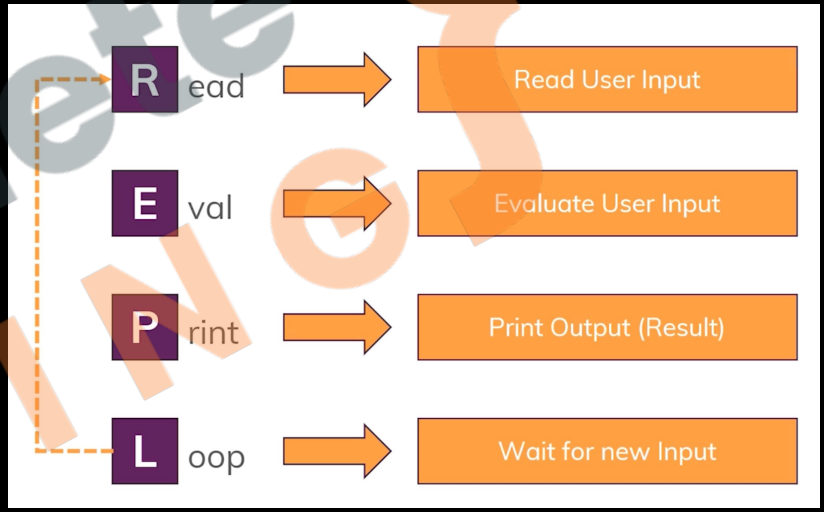
```
const fs = require('fs');

// Define two variables
let a = 10;
let b = 5;

// Basic arithmetic operations
let sum = a + b;
let product = a * b;

// Prepare data to write
let data = `Sum: ${sum}\nProduct: ${product}`;
console.log(data);

// Write data to a local file
fs.writeFile('output.txt', data, (err) => {
    if (err) throw err;
    console.log('Data written to file');
});
```

```
prashantjain@Mac-mini Desktop % node test.js
Sum: 15
Product: 50
Data written to file
```

1. Streamlines Node Command: Use node filename.js to execute a JavaScript file in the Node.js environment.
2. Require Syntax: Use require('module') to include built-in or external modules, or other JavaScript files in your code.
3. Modular Code: require helps organize code into reusable modules, separating concerns and improving maintainability.
4. Caching: Modules loaded with require are cached, meaning the file is executed only once even if included multiple times.

2.8 What is REPL

1. Streamlines Interactive Shell: Executes JavaScript code interactively.
2. Quick Testing: Ideal for testing and debugging code snippets on the fly.
3. Built-in Help: Offers help commands via .help.
4. Session Management: Supports saving (.save) and loading (.load) code sessions.
5. Node.js API Access: Provides direct access to Node.js APIs for experimentation.
6. Customizable: Allows customization of prompt and behaviour settings.



2.9 Executing Code via REPL

```
prashantjain@Mac-mini Desktop % node
Welcome to Node.js v20.9.0.
Type ".help" for more information.
> 5 + 6
11
> console.log('KG Coding is the best');
KG Coding is the best
undefined
> fs.writeFile('output.txt', 'Writing to file', (err) => {
...     if (err) throw err;
...     console.log('Data written to file');
... });
undefined
> Data written to file
```

Revision

1. What is IDE
2. Need of IDE
3. MAC Setup
   - Install latest Node & VsCode
4. Windows Setup
   - Install latest Node & VsCode
5. Linux Setup
   - Install latest Node & VsCode
6. VsCode (Extensions and Settings)
7. Executing first .js file
8. What is REPL
9. Executing Code via REPL

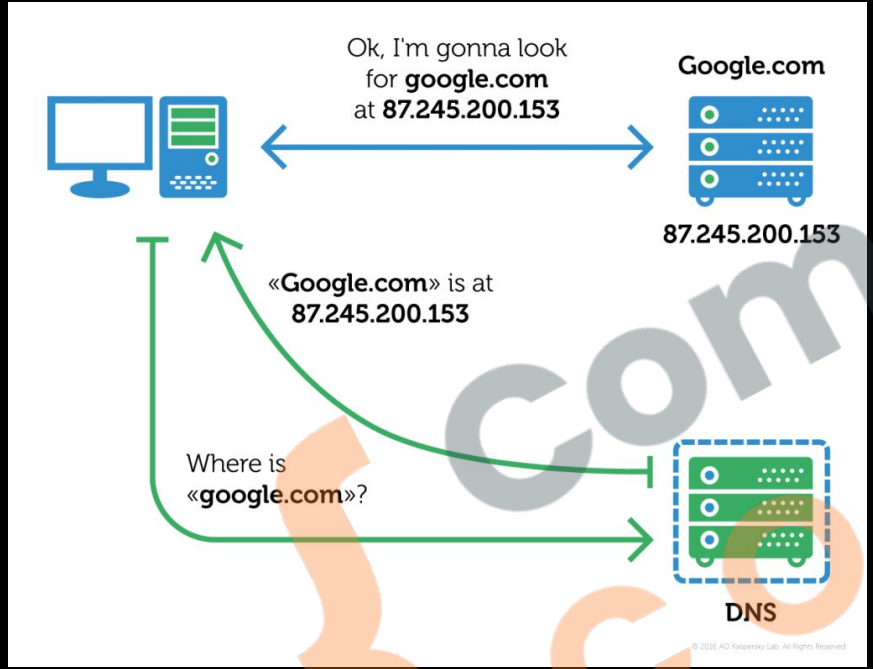

node
JS


Complete
{CODING}

3. First Node Server

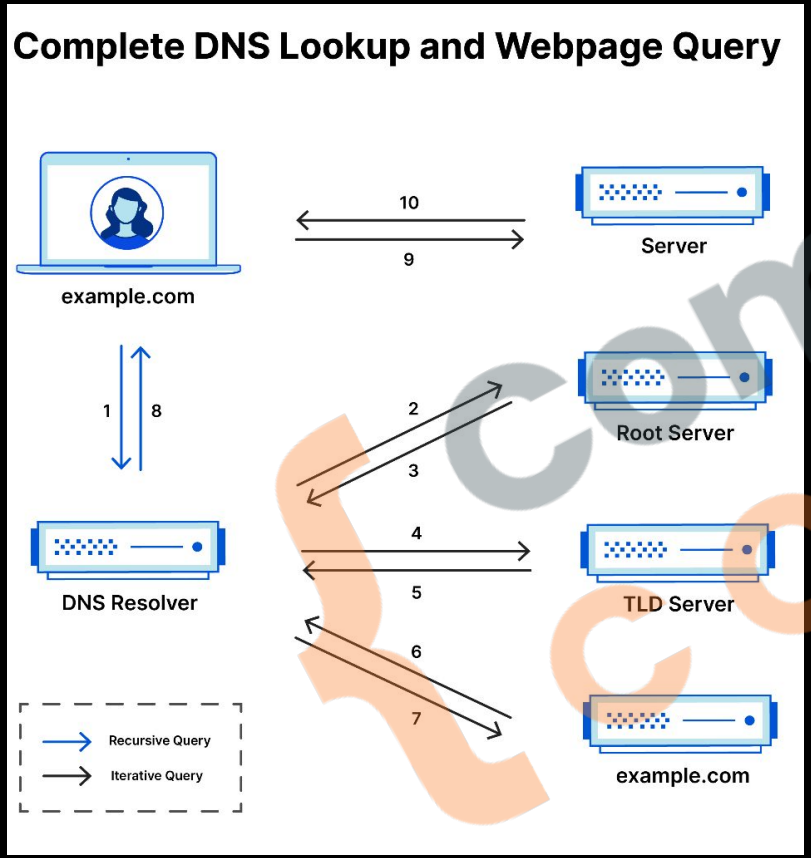
1. How DNS Works?
2. How Web Works?
3. What are Protocols?
4. Node Core Modules
5. Require Keyword
6. Creating first Node Server

3.1 How DNS Works?



1. Domain Name Entry: User types a domain (e.g., www.example.com) into the browser.
2. DNS Query: The browser sends a DNS query to resolve the domain into an IP address.
3. DNS Server: Provides the correct IP address for the domain.
4. Browser Connects: The browser uses the IP to connect to the web server and loads the website.

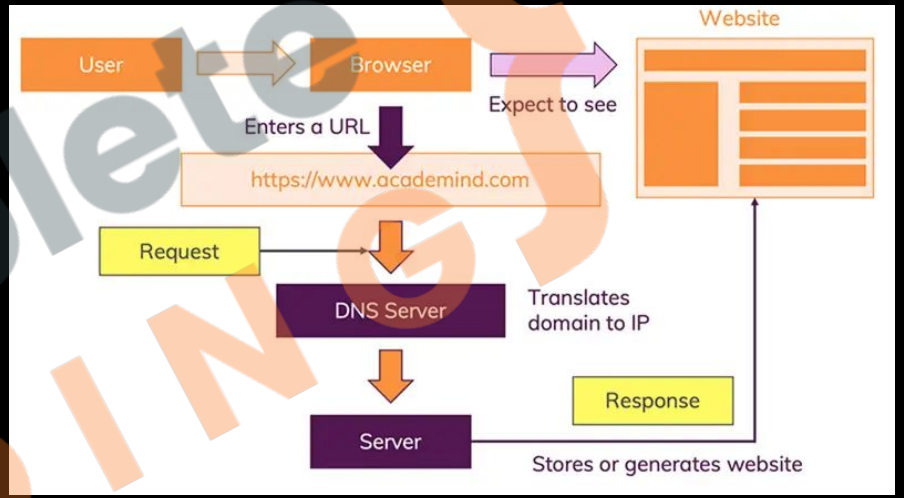
3.1 How DNS Actually Works?



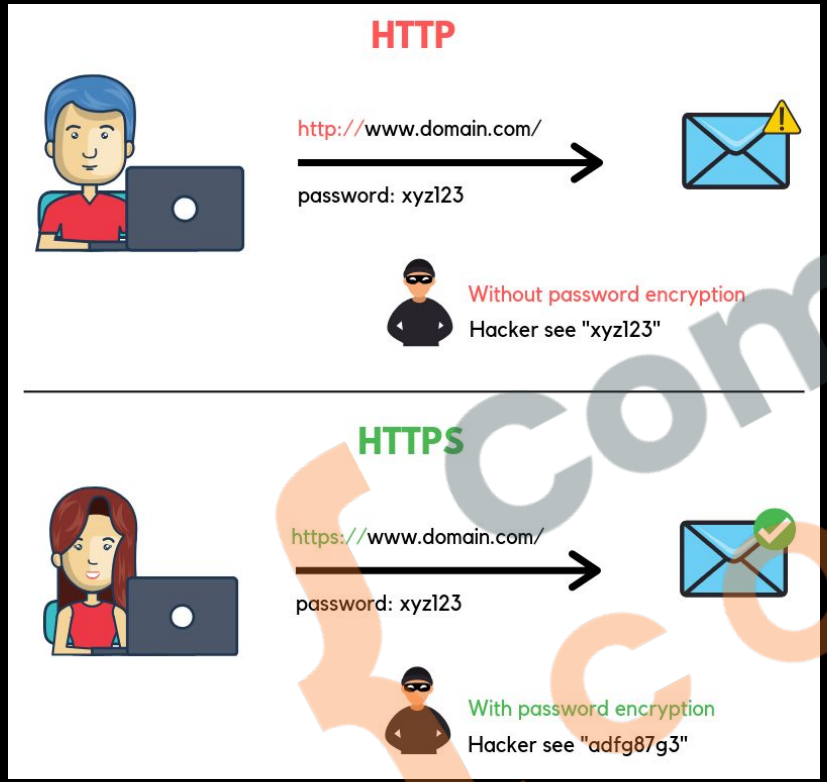
1. Root DNS: Acts as the starting point for DNS resolution. It directs queries to the correct TLD server (e.g., .com, .org).
2. TLD (Top-Level Domain) DNS: Handles queries for specific top-level domains (e.g., .com, .net) and directs them to the authoritative DNS server (e.g., Verisign for .com, PIR for .org)
3. Authoritative DNS: Contains the actual IP address of the domain and answers DNS queries with this information.(e.g., Cloudflare, Google DNS).

3.2 How Web Works?

1. Client Request Initiation: The client (browser) initiates a network call by entering a URL.
2. DNS Resolution: The browser contacts a DNS server to get the IP address of the domain.
3. TCP Connection: The browser establishes a TCP connection with the server's IP address.
4. HTTP Request: The browser sends an HTTP request to the server.
5. Server Processing: The server processes the request and prepares a response.
6. HTTP Response: The server sends an HTTP response back to the client.
7. Network Transmission: The response travels back to the client over the network.
8. Client Receives Response: The browser receives and interprets the response.
9. Rendering: The browser renders the content of the response and displays it to the user.



3.3 What are Protocols?



Http (HyperText Transfer Protocol):

- Facilitates communication between a web browser and a server to transfer web pages.
- Sends data in plain text (no encryption).
- Used for basic website browsing without security.

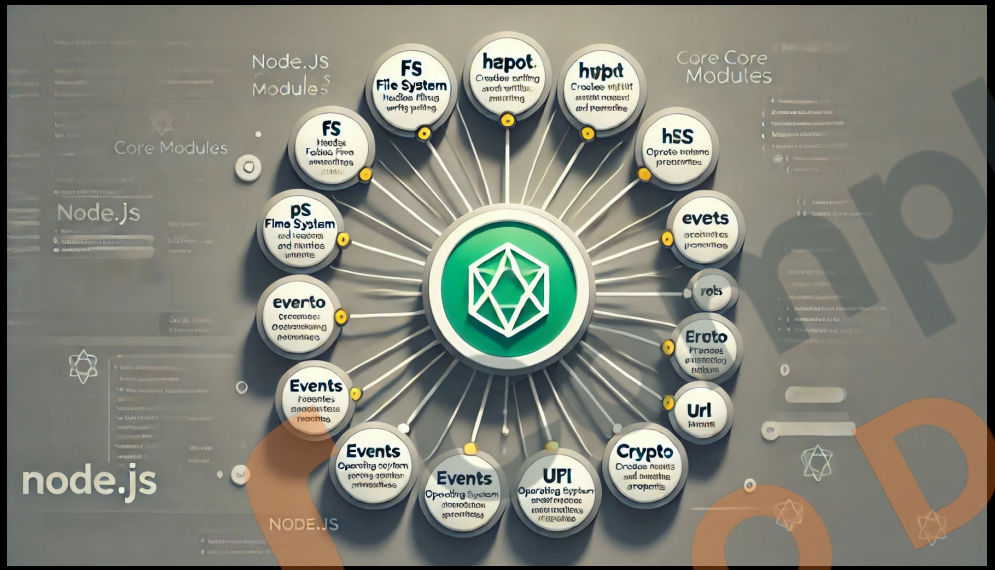
HTTPS (HyperText Transfer Protocol Secure):

- Secure version of HTTP, encrypts data for secure communication.
- Uses SSL/TLS to encrypt data.
- Used in online banking, e-commerce.

TCP (Transmission Control Protocol):

- Ensures reliable, ordered, and error-checked data delivery over the internet.
- Establishes a connection before data is transferred.

3.4 Node Core Modules



1. Built-in: Core modules are included with Node.js installation.
2. No Installation Needed: Directly available for use without npm install.
3. Performance: Highly optimized for performance.

3.4 Node Core Modules

1. fs (File System): Handles file operations like reading and writing files.
2. http: Creates HTTP servers and makes HTTP requests.
3. https: Launch a SSL Server.
4. path: Provides utilities for handling and transforming file
5. paths.os: Provides operating system-related utility methods and properties.
6. events: Handles events and event-driven programming.
7. crypto: Provides cryptographic functionalities like hashing and encryption.
8. url: Parses and formats URL strings.

3.5 Require Keyword

1. Purpose: Imports modules in Node.js.
2. Caching: Modules are cached after the first require call.
3. .js is added automatically and is not needed to at the end of module name.
4. Path Resolution: Node.js searches for modules in core, node_modules, and file paths.

Syntax:

```
const moduleName = require('module');
```

```
// Load the built-in http module
const http = require('http');

// Load the third party express module
const express = require('express');

// Load the custom myModule module
const myModule = require('./myModule');
```

3.6 Creating first Node Server

```
1  // Simple Node.js server
2  const http = require('http');
3
4  function requestListener(req, res) {
5      console.log(req);
6  }
7
8  http.createServer(requestListener);
```

3.6 Creating first Node Server

```
1  // Simple Node.js server
2  const http = require('http');
3
4  http.createServer(function (req, res) {
5    console.log(req);
6  });
```

3.6 Creating first Node Server

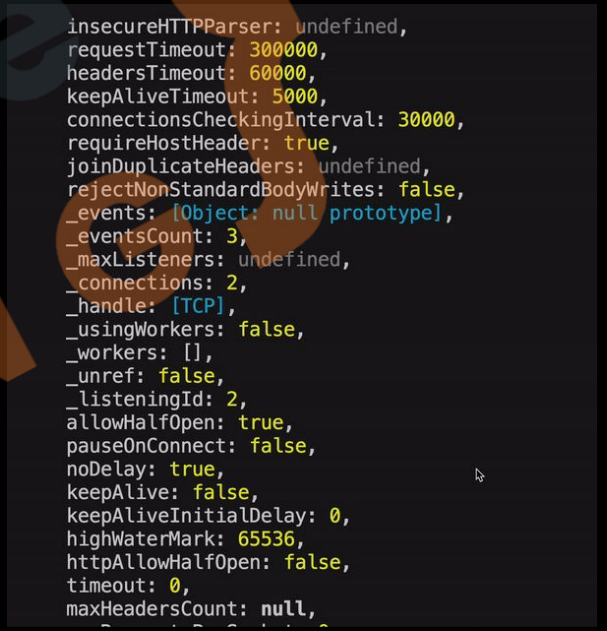
```
1  // Simple Node.js server
2  const http = require('http');
3
4  http.createServer((req, res) => {
5    console.log(req);
6  });
```

Run the code with:

node app.js

3.6 Creating first Node Server

```
1  // Simple Node.js server
2  const http = require('http');
3
4  const server = http.createServer((req, res) => {
5    console.log(req);
6  });
7
8  server.listen(3000);
```



3.6 Creating first Node Server

```
1  // Simple NodeJS server
2  const http = require('http');
3
4  const server = http.createServer((req, res) => {
5    console.log(req);
6  });
7
8  const PORT = 3000;
9  server.listen(PORT, () => {
10   console.log(`Server running at http://localhost:${PORT}/`);
11 });
```

Revision

1. How DNS Works?
2. How Web Works?
3. What are Protocols?
4. Node Core Modules
5. Require Keyword
6. Creating first Node Server

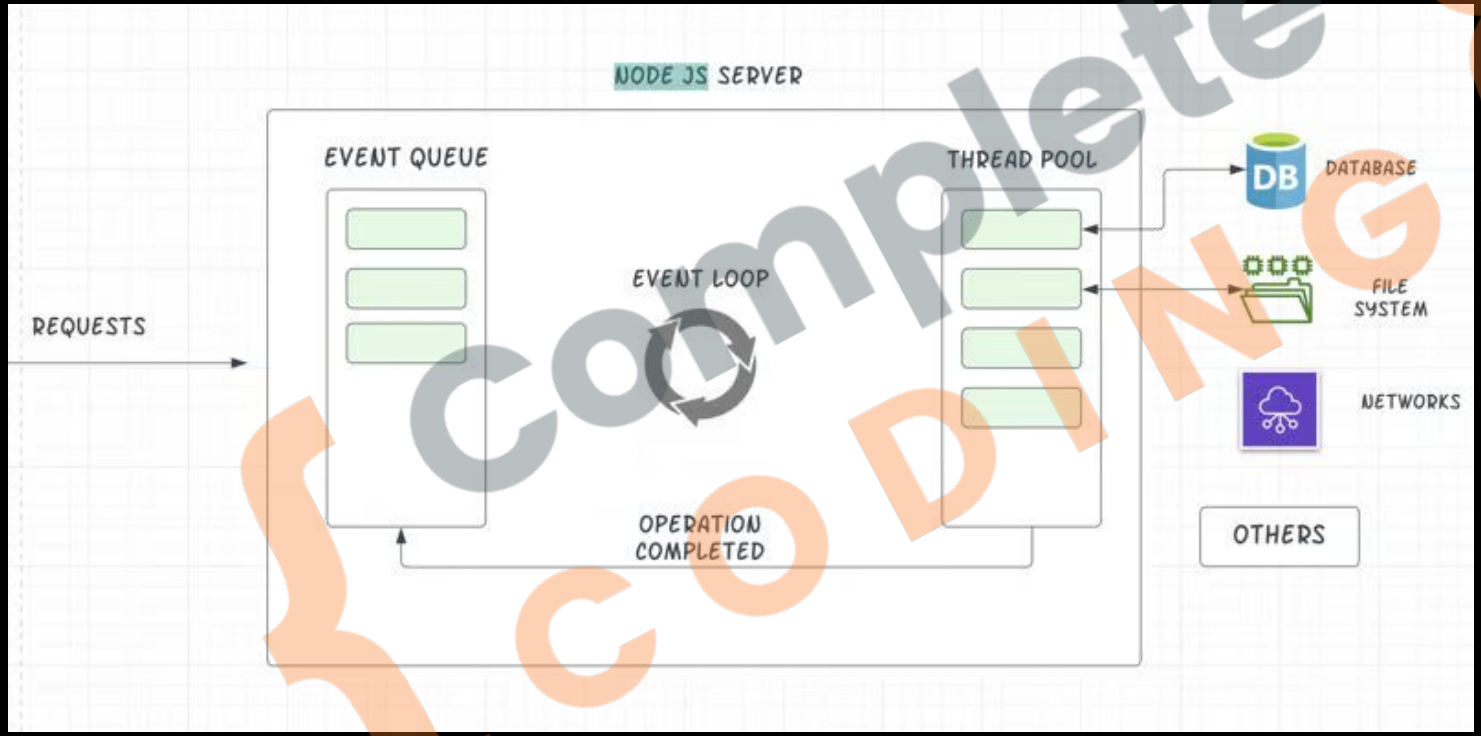

node
JS


Complete
{CODING}

4. Request & Response

1. Node Lifecycle & Event Loop
2. How to exit Event Loop
3. Understand Request Object
4. Sending Response
5. Routing Requests
6. Taking User Input
7. Redirecting Requests

4.1 Node Lifecycle & Event Loop



4.2 How to exit Event Loop

```
1  // Simple Node.js server
2  const http = require('http');
3
4  const server = http.createServer((req, res) => {
5    console.log(req);
6    process.exit(); // Stops event loop
7  });
8
9  const PORT = 3000;
10 server.listen(PORT, () => {
11   console.log(`Server running at http://localhost:${PORT}/`);
12 });
```

4.3 Understand Request Object

```
  [Symbol(kHeaders)]: {
    host: 'localhost:3000',
    connection: 'keep-alive',
    'cache-control': 'max-age=0',
    'sec-ch-ua': '"Chromium";v="128", "Not;A=Brand";v="24", "Google Chrome";v="128"',
    'sec-ch-ua-mobile': '?0',
    'sec-ch-ua-platform': '"macOS"',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (
36',
    accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
nge;v=b3;q=0.7',
    'sec-fetch-site': 'none',
    'sec-fetch-mode': 'navigate',
    'sec-fetch-user': '?1',
    'sec-fetch-dest': 'document',
    'accept-encoding': 'gzip, deflate, br, zstd',
    'accept-language': 'en-US,en-IN;q=0.9,en;q=0.8,hi-IN;q=0.7,hi;q=0.6',
    cookie: 'token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImM3M2Y0MzNjLTFlYzYtNDl
qvUSrMLqbP1uy5bTjnJEQHXc1c'
  },
  [Symbol(kHeadersCount)]: 32,
  [Symbol(kTrailers)]: null,
  [Symbol(kTrailersCount)]: 0
}
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

4.3 Understand Request Object

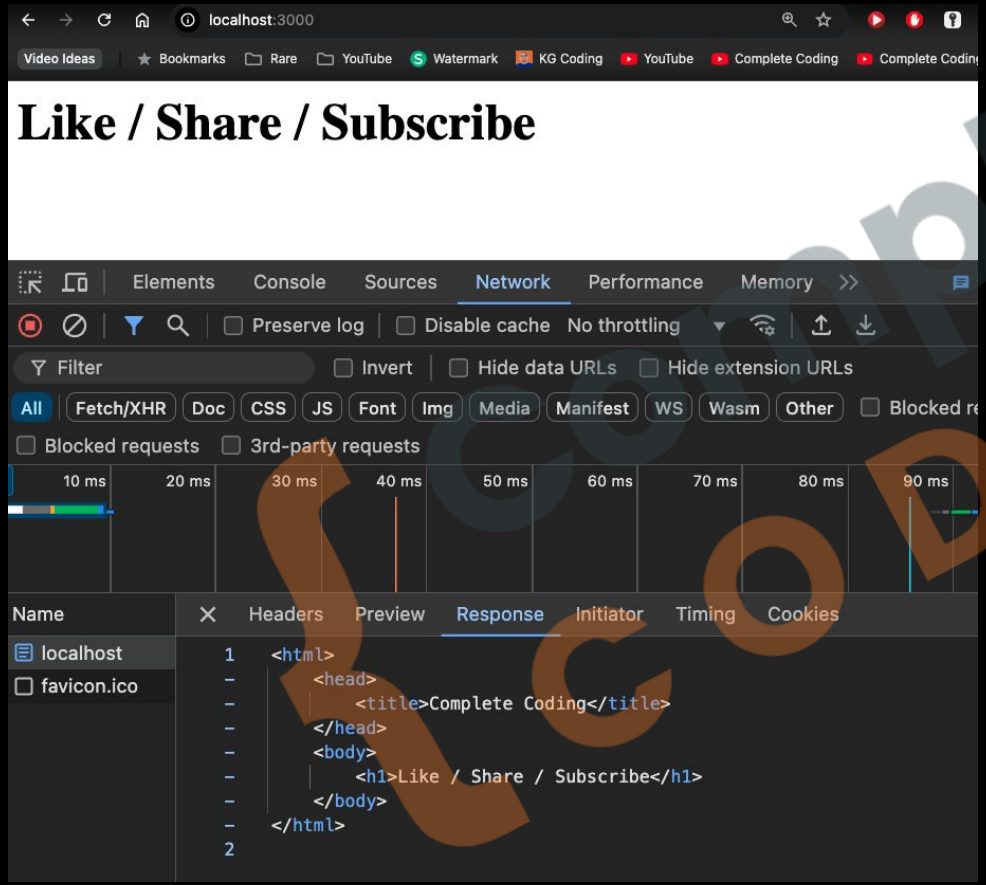
```
// Simple NodeJS server
const http = require('http');

const server = http.createServer((req, res) => {
  console.log(req.url, req.method, req.headers);
});

const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}/`);
});
```

Use the browser to access:
http://localhost:3000/
http://localhost:3000/products

4.4 Sending Response

```
1  // Simple NodeJS server
2  const http = require('http');
3
4  const server = http.createServer((req, res) => {
5    //res.setHeader('Content-Type', 'json');
6    res.setHeader('Content-Type', 'text/html');
7    res.write('<html>');
8    res.write('<head><title>Complete Coding</title></head>');
9    res.write('<body><h1>Like / Share / Subscribe</h1></body>');
10   res.write('</html>');
11   res.end();
12 });
13
14 const PORT = 3000;
15 server.listen(PORT, () => {
16   console.log(`Server running at http://localhost:${PORT}/`);
17 });
```

4.4 Sending Response



4.5 Routing Requests

```
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>Complete Coding</title></head>');

  if (req.url === '/') {
    res.write('<h1>Welcome to Home page</h1>');
    res.end();
  } else if (req.url.toLowerCase() === '/products') {
    res.write('<h1>Products</h1>');
    res.end();
  }

  res.write('<body><h1>Like / Share / Subscribe</h1></body>');
  res.write('</html>');
  res.end();
});
```

4.5 Routing Requests

```
prashantjain@Prashants-Mac-mini node % node app.js
Server running at http://localhost:3000/
node:_http_outgoing:699
    throw new ERR_HTTP_HEADERS_SENT('set');
    ^

Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (node:_http_outgoing:699:11)
    at Server.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:14:7)
    at Server.emit (node:events:520:28)
    at parserOnIncoming (node:_http_server:1146:12)
    at HTTPParser.parserOnHeadersComplete (node:_http_common:118:17) {
  code: 'ERR_HTTP_HEADERS_SENT'
}

Node.js v22.5.1
```

4.5 Routing Requests

```
const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>Complete Coding</title></head>');

  if (req.url === '/') {
    res.write('<h1>Welcome to Home page</h1>');
    return res.end();
  } else if (req.url.toLowerCase() === '/products') {
    res.write('<h1>Products</h1>');
    return res.end();
  }

  res.write('<body><h1>Like / Share / Subscribe</h1></body>');
  res.write('</html>');
  return res.end();
});
```

4.6 Taking User Input

```
if (req.url === '/') {
  res.write('<h1>Welcome to Home page</h1>');
  res.write('<form action="/submit-details" method="POST">');
  res.write('<input type="text" id="name" name="name" placeholder="Enter your
  name"><br><br>');
  res.write('<label for="gender">Gender:</label>');
  res.write('<input type="radio" id="male" name="gender" value="male">');
  res.write('<label for="male">Male</label>');
  res.write('<input type="radio" id="female" name="gender" value="female">');
  res.write('<label for="female">Female</label><br><br>');
  res.write('<button type="submit">Submit</button>');
  res.write('</form>');
  return res.end();
}
```

4.7 Redirecting Requests

```
} else if (req.method == 'POST' &&
           req.url.toLowerCase() === '/submit-details') {
  fs.writeFileSync('user-details.txt', 'Prashant Jain');
  res.statusCode = 302;
  res.setHeader('Location', '/');
  return res.end();
}
```

node > user-details.txt

```
1  Prashant Jain
```

Practise Set

Create a page that shows a navigation bar of Myntra with the following links:

A. Home
B. Men
C. Women
D. Kids
E. Cart

Clicking on each link page should navigate to that page and a welcome to section text is shown there.

Revision

1. Node Lifecycle & Event Loop
2. How to exit Event Loop
3. Understand Request Object
4. Sending Response
5. Routing Requests
6. Taking User Input
7. Redirecting Requests

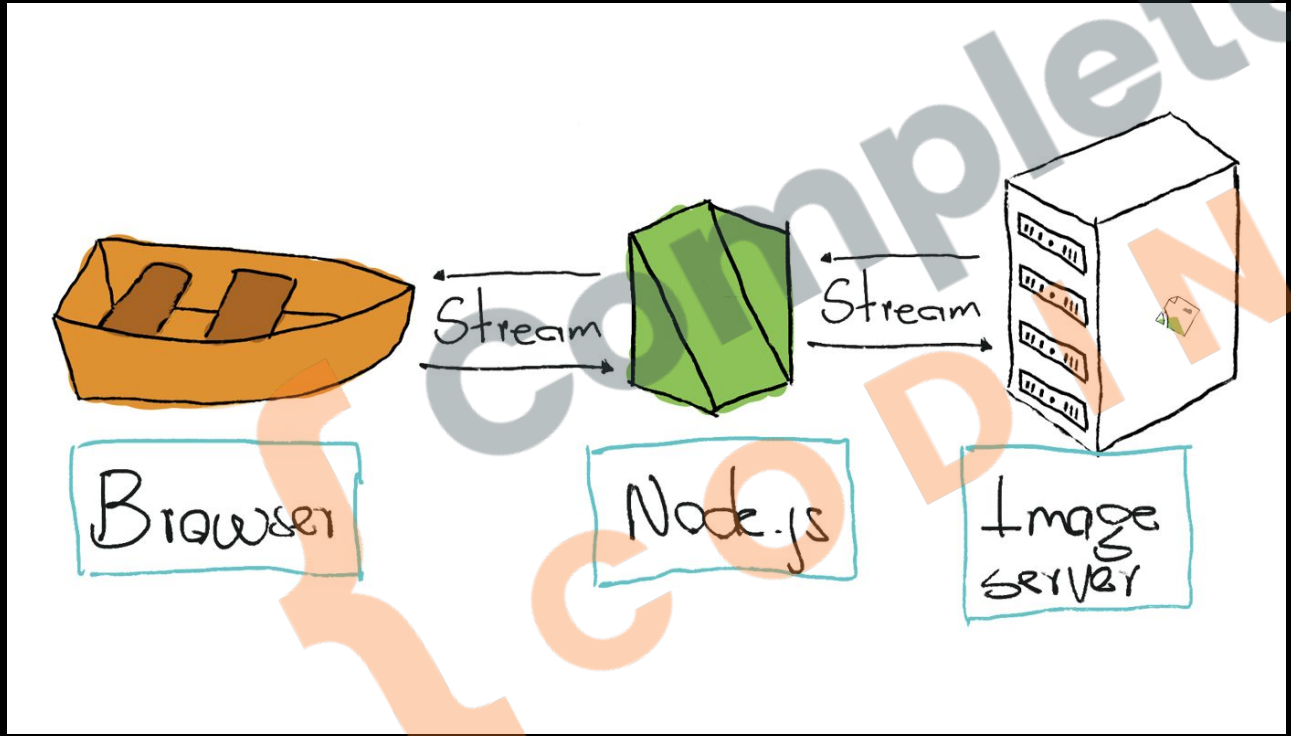

node
JS


Complete
{CODING}

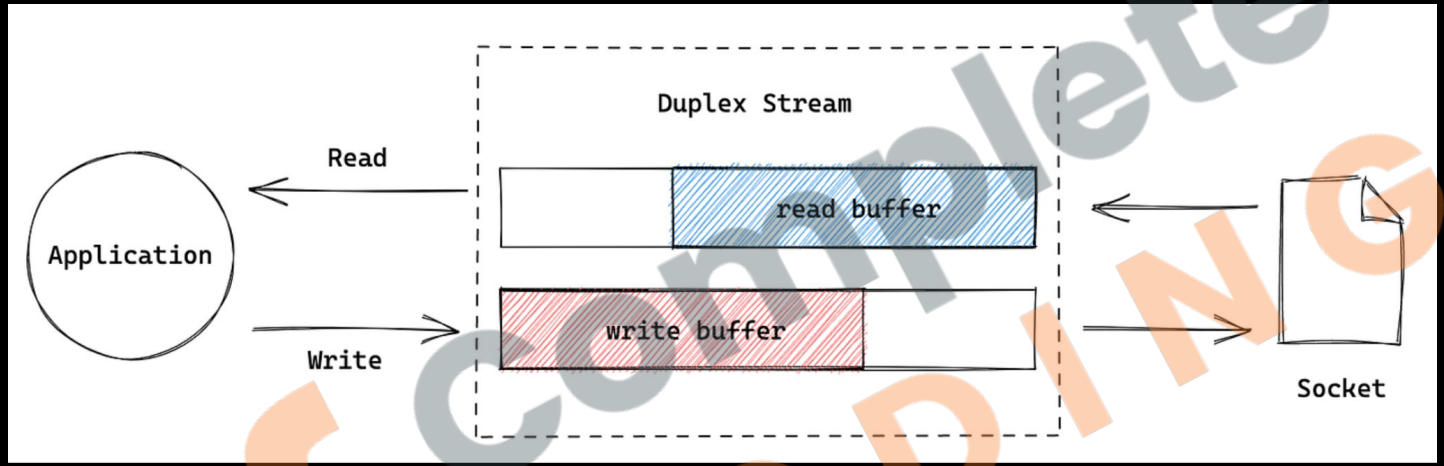
5. Parsing Request

1. Streams
2. Chunks
3. Buffers
4. Reading Chunk
5. Buffering Chunks
6. Parsing Request
7. Using Modules

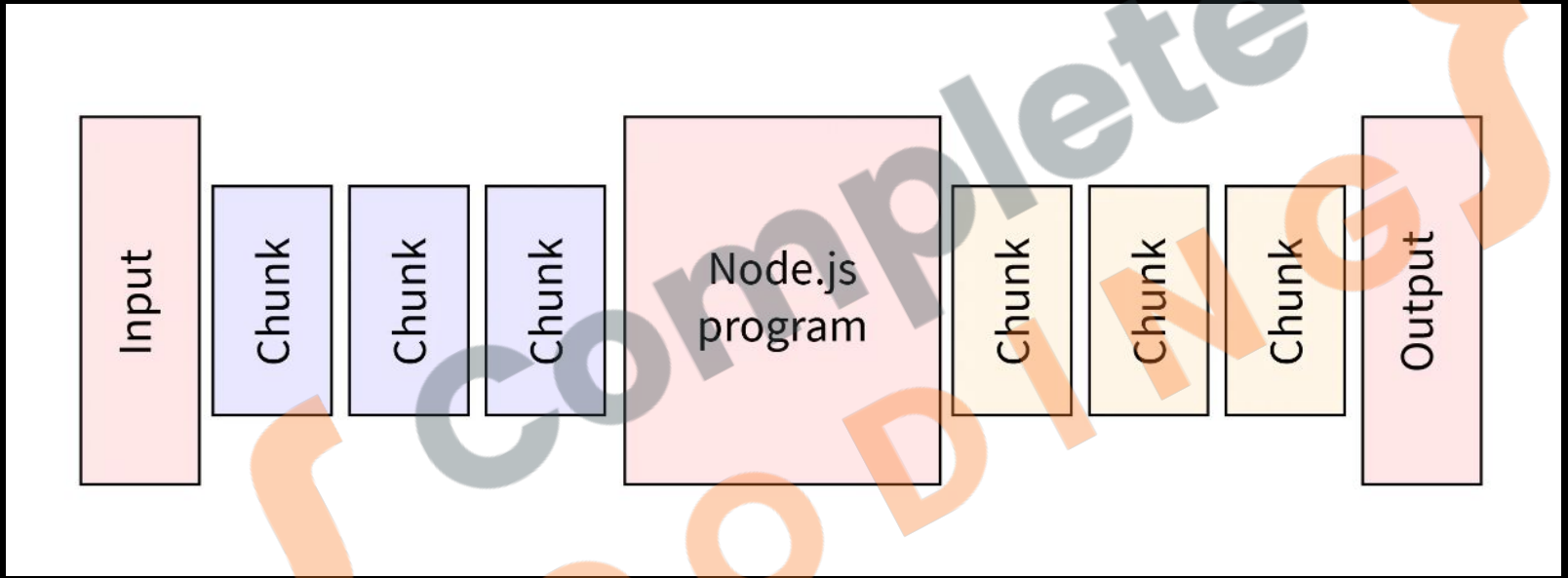
5.1 Streams



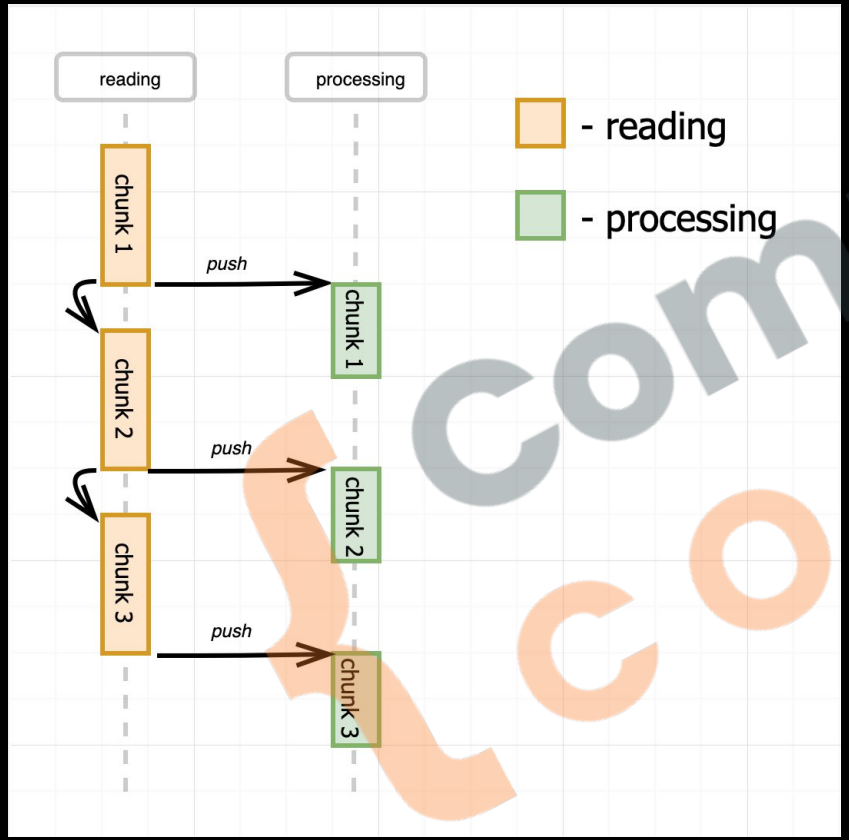
5.1 Streams



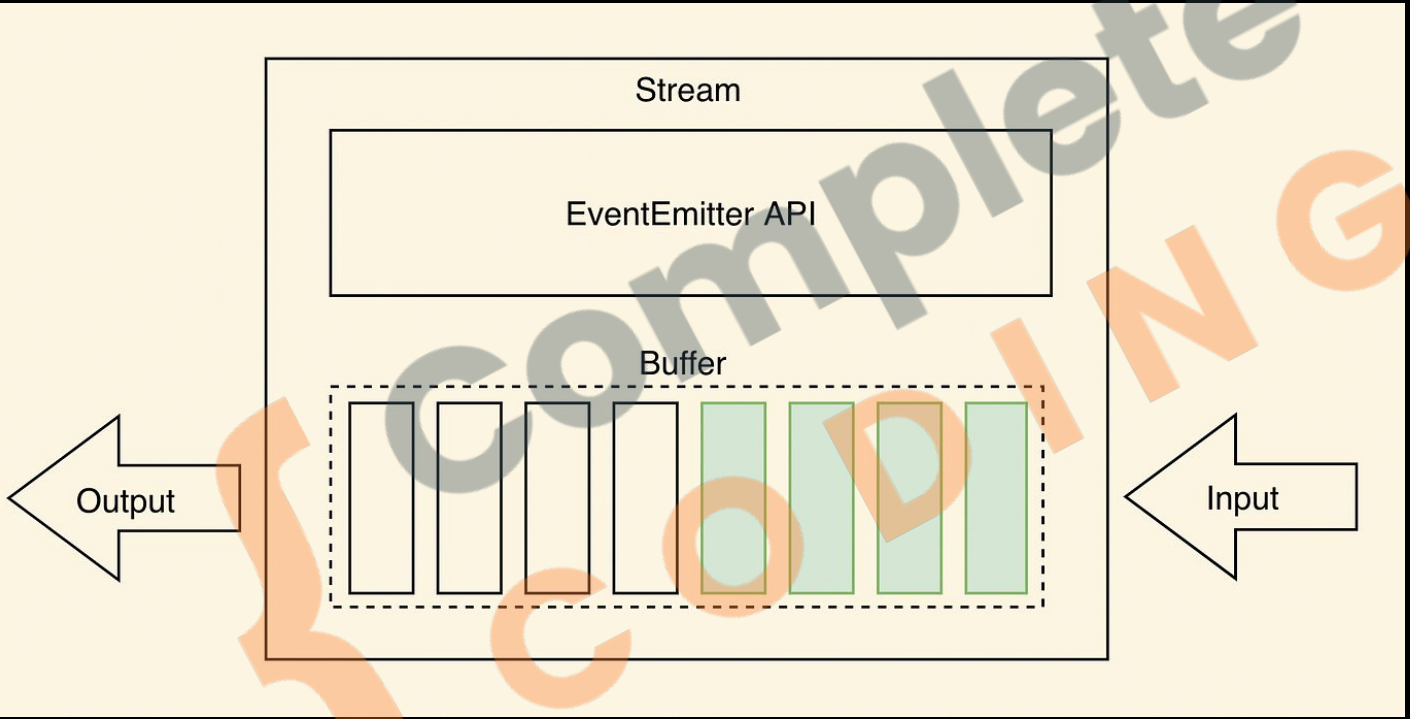
5.2 Chunks



5.2 Chunks


reading
processing
- reading
- processing
chunk 1
push
chunk 1
chunk 2
push
chunk 2
chunk 3
push
chunk 3

5.3 Buffers



5.4 Reading Chunk

```
} else if (
  req.method == "POST" &&
  req.url.toLowerCase() === "/submit-details"
) {
  req.on("data", (chunk) => {
    console.log(chunk);
  });
  fs.writeFileSync("user-details.txt", "Prashant Jain");
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
}
```

```
prashantjain@Prashants-Mac-mini node % node app.js
Server running at http://localhost:3000/
<Buffer 6e 61 6d 65 3d 50 72 61 73 68 61 6e 74 26 67 65 6e 64 65 72 3d 6d 61 6c 65>
```

5.5 Buffering Chunks

```
const body = [];
req.on("data", (chunk) => {
  console.log(chunk);
  body.push(chunk);
});
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
});
fs.writeFileSync("user-details.txt", "Prashant Jain");
```

```
prashantjain@Prashants-Mac-mini node % node app.js
Server running at http://localhost:3000/
<Buffer 6e 61 6d 65 3d 50 72 61 73 68 61 6e 74 26 67 65 6e 64 65 72 3d 6d 61 6c 65>
name=Prashant&gender=male
```

5.6 Parsing Request

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  console.log(jsonObject);
  // Output: { name: 'Prashant', gender: 'male' }
});
fs.writeFileSync("user-details.txt", "Prashant Jain");
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```

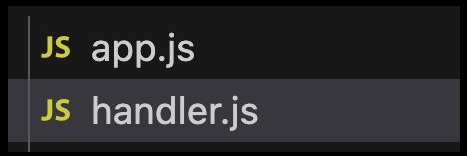
5.6 Parsing Request

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
});
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```

node > user-details.txt

```
1  {"name":"Prashant","gender":"male"}
```

5.7 Using Modules



handler.js > ...

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

const requestHandler = (req, res) => {
  if (req.url === "/") {
    res.setHeader("Content-Type", "text/html");
```

```
module.exports = requestHandler
```

app.js > ...

```
// Simple NodeJS server
const http = require('http');
const requestHandler = require('./handler');

const server = http.createServer(requestHandler);
```

5.7 Using Modules

```
// Method 1: Multiple exports using object
module.exports = {
  handler: requestHandler,
  extra: "Extra"
};

// Method 2: Setting multiple properties
module.exports.handler = requestHandler;
module.exports.extra = "Extra";

// Method 3: Shortcut using exports
exports.handler = requestHandler;
exports.extra = "Extra";
```

Practise Set

Create a Calculator

1. Create a new Node.js project named “Calculator”.
2. On the home page (route “/”), show a welcome message and a link to the calculator page.
3. On the “/calculator” page, display a form with two input fields and a “Sum” button.
4. When the user clicks the “Sum” button, they should be taken to the “/calculate-result” page, which shows the sum of the two numbers.
    - Make sure the request goes to the server.
    - Create a separate module for the addition function.
    - Create another module to handle incoming requests.
    - On the “/calculate-result” page, parse the user input, use the addition module to calculate the sum, and display the result on a new HTML page.

Revision

1. Streams
2. Chunks
3. Buffers
4. Reading Chunk
5. Buffering Chunks
6. Parsing Request
7. Using Modules

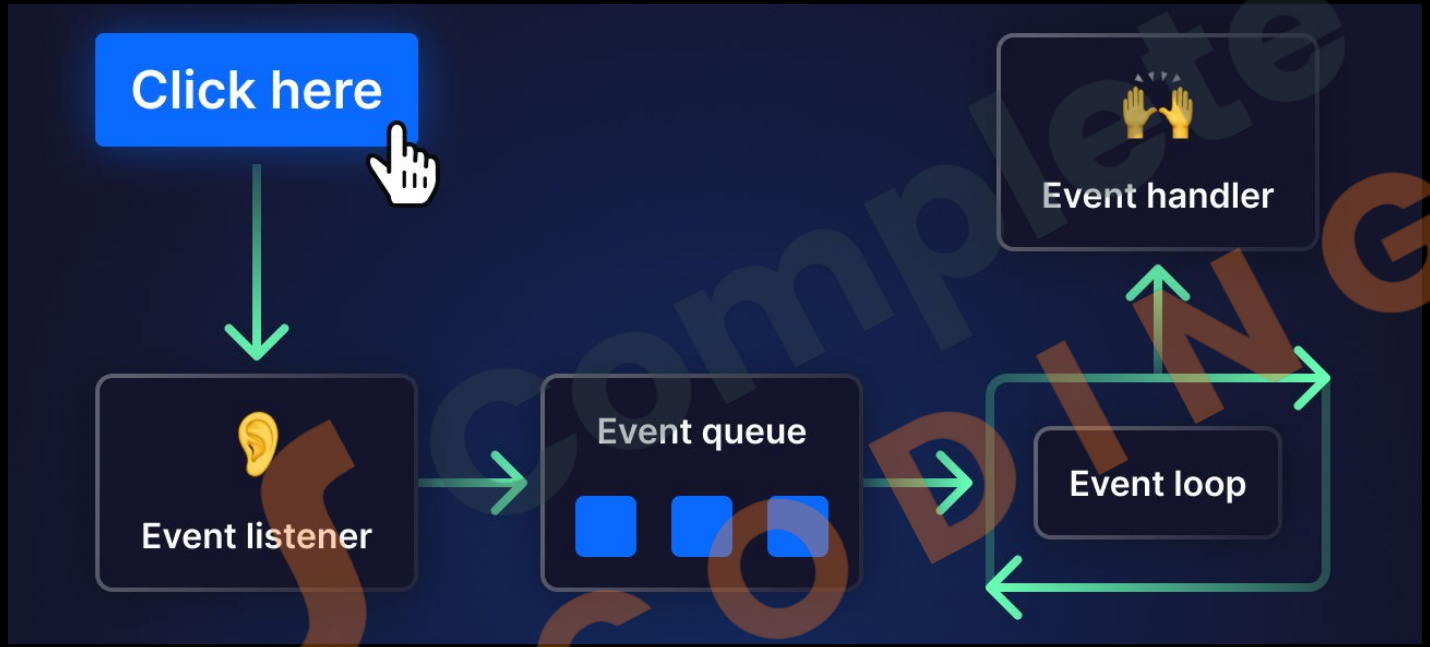

node
JS


Complete
{CODING}

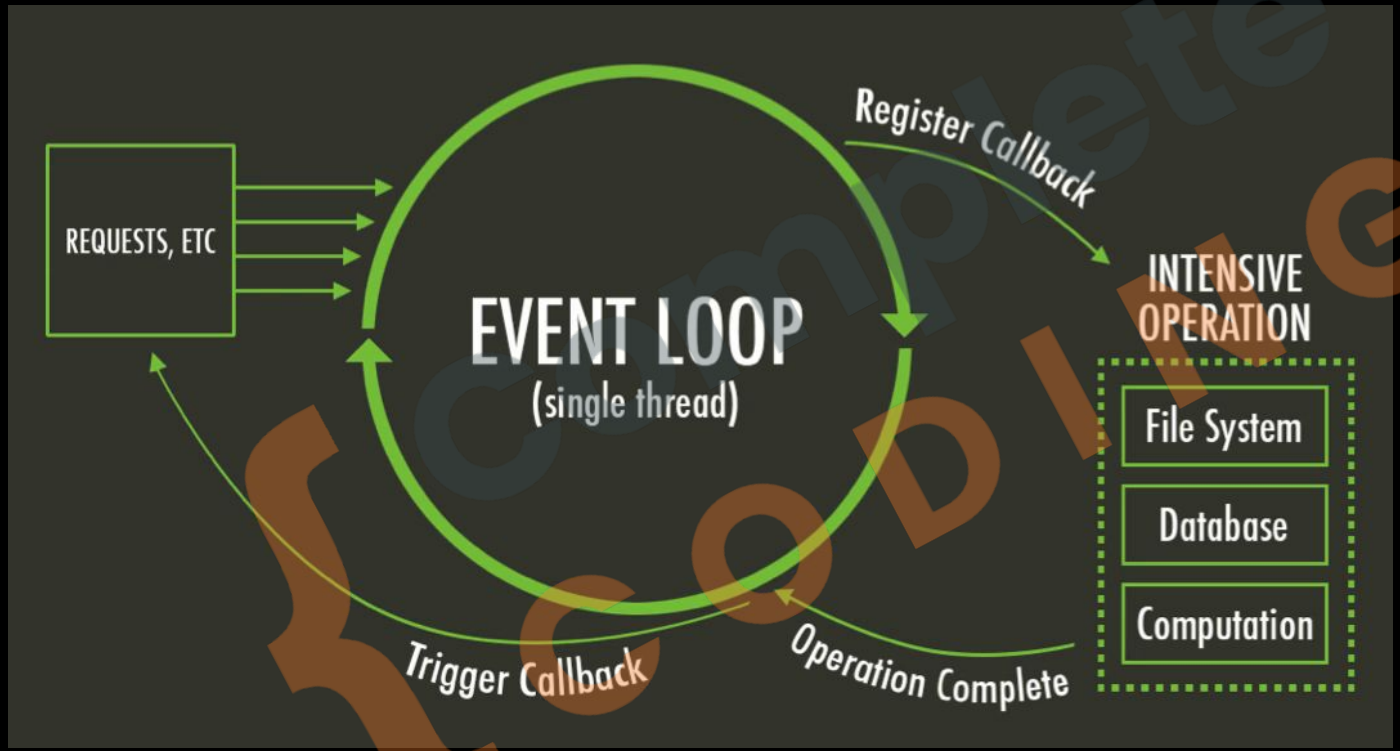
6. Event Loop

1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code

6.1 Event Driven



6.2 Single Threaded



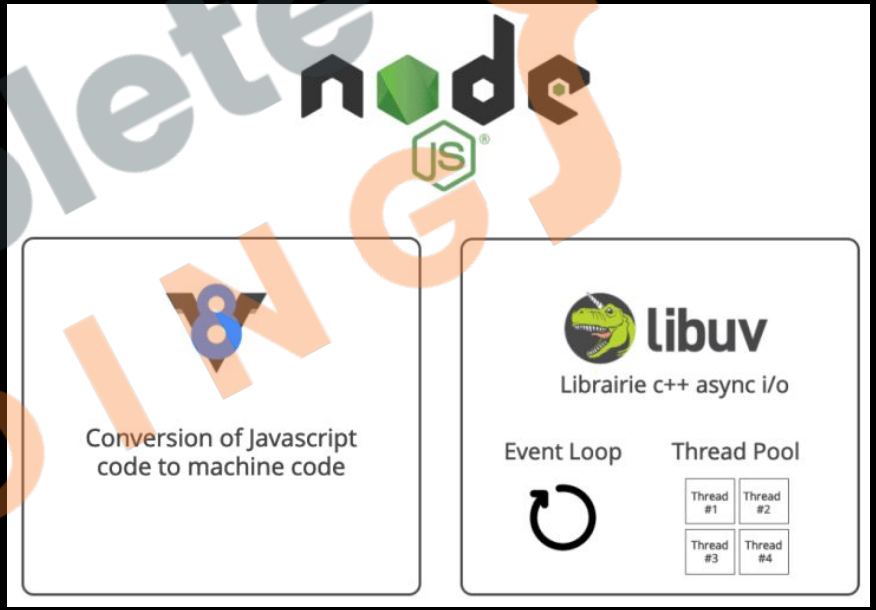
6.3 V8 vs libuv

V8:

1. Open-source JavaScript engine by Google.
2. Used in Chrome and Node.js.
3. Compiles JavaScript to native machine code.
4. Ensures high-performance JavaScript execution.

libuv:

1. Multi-platform support library for Node.js.
2. Handles asynchronous I/O operations.
3. Provides event-driven architecture.
4. Manages file system, networking, and timers non-blockingly across platforms.

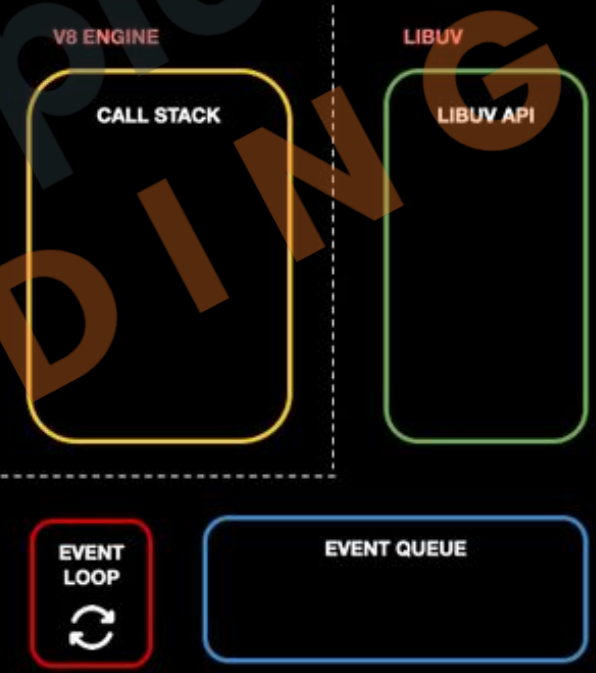


6.4 Node Runtime

An invoked function is added to the call stack. Once it returns a value, it is popped off.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res) {
  console.log("Query executed");
});

console.log("Before query result");
```

OUTPUT



6.4 Node Runtime

Database queries or other I/O ops do not block Node.js single thread because Libuv API handles them.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res) {
  console.log("Query executed");
});

console.log("Before query result");
```

OUTPUT

```
Starting Node.js
```

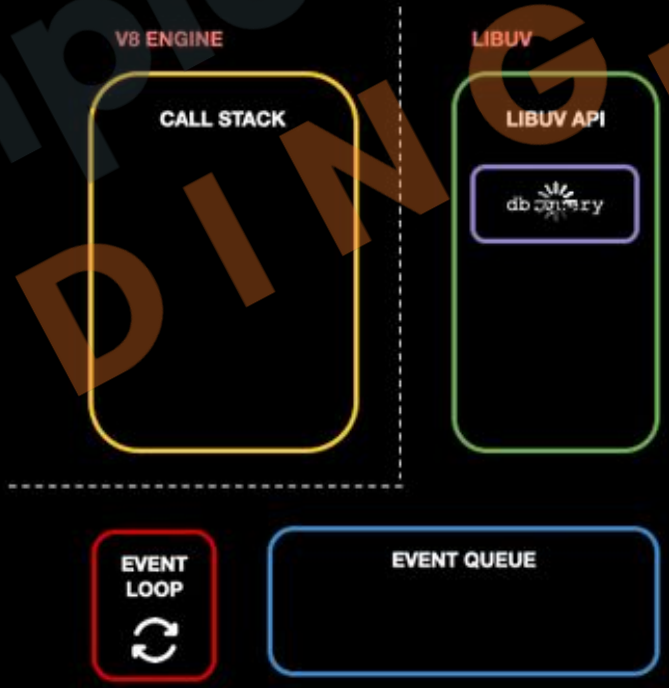
6.4 Node Runtime

While Libuv asynchronously handles I/O operations, Node.js single thread keeps running code.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res) {
  console.log("Query executed");
});

console.log("Before query result");
```

OUTPUT

```
Starting Node.js
```



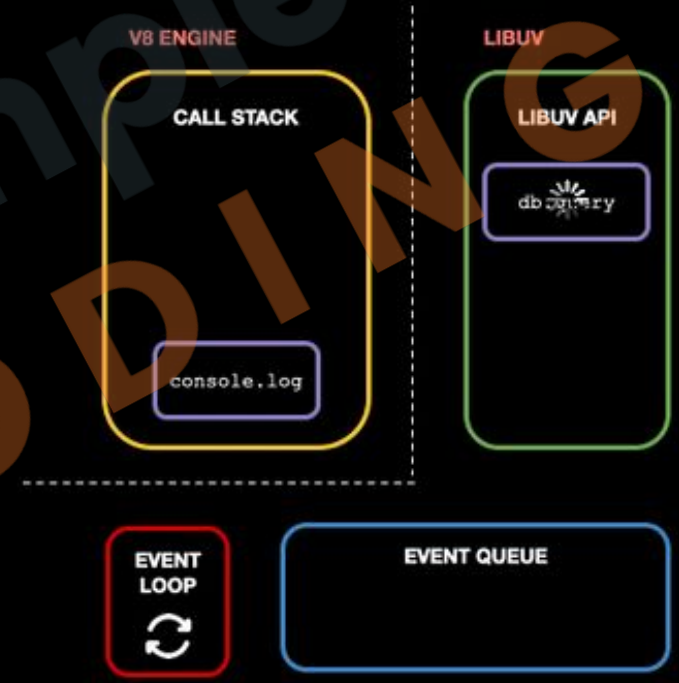
6.4 Node Runtime

Callbacks of completed queries are moved to the event queue. If the call stack is empty, the event loop checks for callbacks and transfers the first.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```

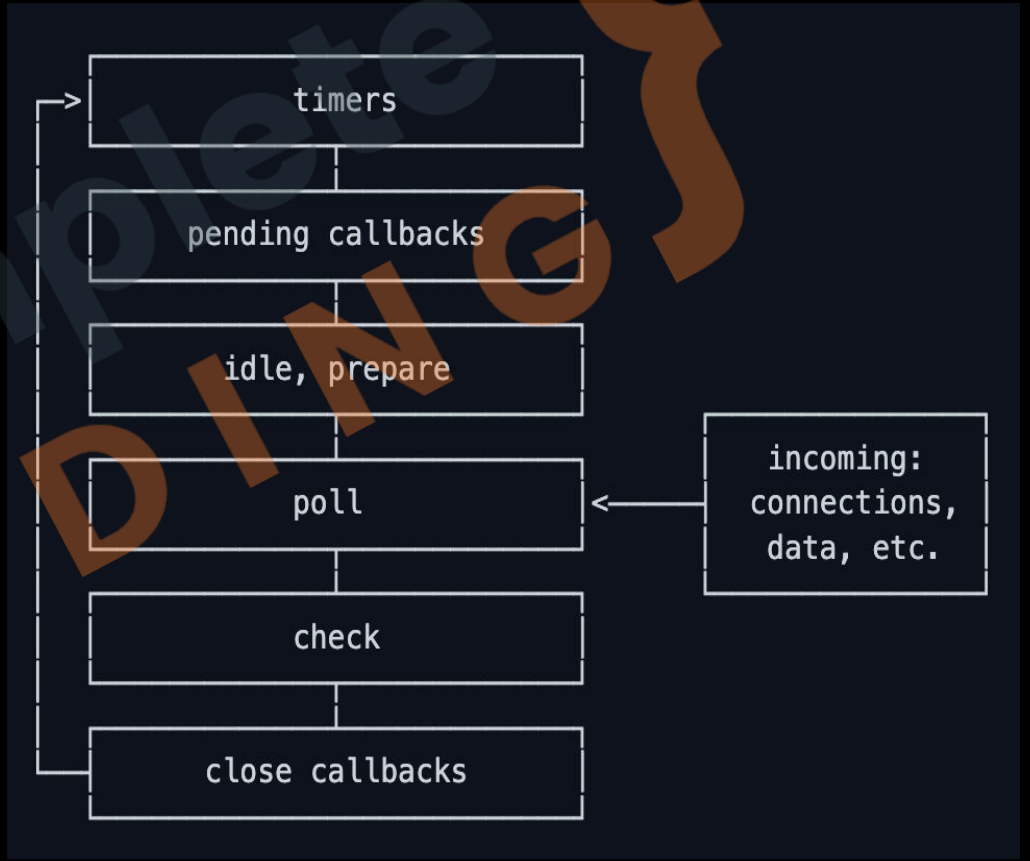
OUTPUT

```
Starting Node.js

Before query result
```

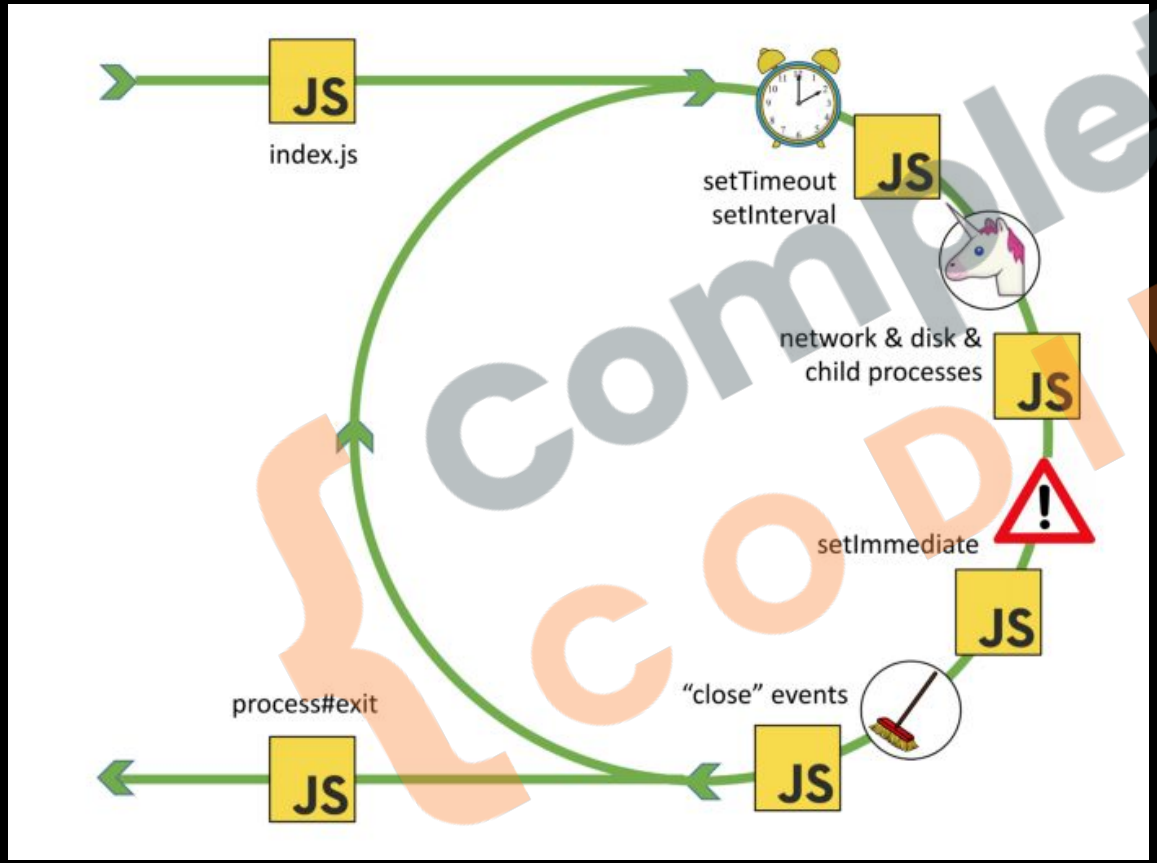


6.5 Event Loop

- timers: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- pending callbacks: executes I/O callbacks deferred to the next loop iteration.
- idle, prepare: only used internally.
- poll: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and `setImmediate()`); node will block here when appropriate.
- check: `setImmediate()` callbacks are invoked here.
- close callbacks: some close callbacks, e.g. `socket.on('close', ...)`.



6.5 Event Loop



6.6 Async Code

```
    const jsonString = JSON.stringify(jsonObject);
    console.log(jsonString);
    fs.writeFileSync("user-details.txt", jsonString);
    res.setHeader("Location", "/");
    res.statusCode = 302;
    res.end();
  });
}

res.write('<body><h1>Like / Share / Subscribe</h1></
body>');
res.write('</html>');
return res.end();
```

```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (node:_http_outgoing:699:11)
    at IncomingMessage.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:44:11)
    at IncomingMessage.emit (node:events:532:35)
    at endReadableNT (node:internal/streams/readable:1696:12)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  code: 'ERR_HTTP_HEADERS_SENT'
}
```

6.6 Async Code

```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```

6.7 Blocking Code

```
const jsonString = JSON.stringify(jsonObject);
console.log(jsonString);
// BLOCKING EVERTHING
fs.writeFileSync("user-details.txt", jsonString);
res.setHeader("Location", "/");
```

6.7 Blocking Code

```
console.log(jsonString);
// Async Operation
fs.writeFile("user-details.txt", jsonString, error => {
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```

Run & Observe

Blocking vs Async

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

console.log('1. Start of script');

// Synchronous (blocking) operation
console.log('2. Reading file synchronously');
const dataSync = fs.readFileSync('user-details.txt', 'utf8');
console.log('3. Synchronous read complete');

// Asynchronous (non-blocking) operation
console.log('4. Reading file asynchronously');
fs.readFile('user-details.txt', 'utf8', (err, dataAsync) => {
 if (err) throw err;
 console.log('6. Asynchronous read complete');
});

console.log('5. End of script');
```

```
1. Start of script
2. Reading file synchronously
3. Synchronous read complete
4. Reading file asynchronously
5. End of script
6. Asynchronous read complete
```

Run & Observe

Event Loop Sequence

```
console.log('1. Start of script');

// Microtask queue (Promise)
Promise.resolve().then(() => console.log('2. Microtask 1'));

// Timer queue
setTimeout(() => console.log('3. Timer 1'), 0);

// I/O queue
const fs = require('fs');
fs.readFile('user-details.txt', () => console.log('4. I/O operation'));

// Check queue
setImmediate(() => console.log('5. Immediate 1'));

// Close queue
process.on('exit', (code) => {
 console.log('6. Exit event');
});

console.log('7. End of script');
```

```
1. Start of script
7. End of script
2. Microtask 1
3. Timer 1
5. Immediate 1
4. I/O operation
6. Exit event
```

Revision

1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code