Role of Node.js in Modern Web Application Development

Salil Bhatnagar[1] and Ashish Pandey[2]
[1] Chameli Devi Group of Institutions, Indore
Salilbhatnagar1@gmail.com
[2]Chameli Devi Group of Institutions, Indore
Ashishpandey0731@gmail.com

Abstract

Node.js is a cutting-edge open-source web platform that has gained significant popularity. Built on Google Chrome's V8 JavaScript runtime engine, Node.js enables developers to create network applications and servers with minimal code. It utilizes an asynchronous programming model based on non-blocking I/O and a single-threaded event loop, eliminating concerns about race conditions and synchronization issues common in concurrent multi-user programming. The research paper portrays the basics of the platform.

Key Words: **JavaScript, Node.js, event driven, single- threaded, non-blocking, asynchronous**

1. Introduction

Node.js is a versatile, open-source, and cross-platform JavaScript runtime environment that has revolutionized modern development. It empowers developers to use JavaScript not only in the browser but also on the server side, making it a powerful tool for a wide range of applications.

Powered by the V8 JavaScript engine—the same core used in Google Chrome—Node.js operates outside the browser, delivering exceptional performance. Unlike traditional models that spawn a new thread for every request, a Node.js application runs in a single process. It relies on asynchronous I/O primitives, enabling non-blocking operations by default. Blocking behavior is rare in Node.js, making it a highly efficient choice for modern applications.

When performing I/O tasks like reading from a network, database, or filesystem, Node.js avoids wasting CPU cycles by not blocking the thread. Instead, it resumes operations only when the response is ready. This approach allows Node.js to handle thousands of concurrent connections with a single server, eliminating the complexities of thread concurrency and reducing potential bugs.

A key advantage of Node.js is its seamless integration of frontend and backend development. JavaScript developers can now write both client-side and server-side code without learning an entirely new language, streamlining the development process.

Node.js supports the latest ECMAScript standards, allowing developers to take full advantage of modern JavaScript features. Since Node.js runs independently of browser updates, developers can decide which ECMAScript version to use by selecting the appropriate Node.js version or enabling experimental features with specific flags.

An Example Node.js Application

The most common example Hello World of Node.js is a web server:

```
const { createServer } = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

To execute this code snippet, save it as a `server.js` file and run it using the command `node server.js` in your terminal.

The script begins by including the Node.js `http` module, which is part of Node.js's robust standard library offering excellent support for networking.

The `createServer()` method from the `http` module creates and returns a new HTTP server instance. This server is then configured to listen on a specified hostname and port. Once the server starts successfully, a callback function is triggered to notify that the server is running.

When the server receives a request, the request event is fired, providing two key objects:

a) **request**: An http.IncomingMessage object containing details about the incoming HTTP request, such as headers and data (though these are not used in this simple example).
b) **response**: An http.ServerResponse object used to send data back to the client.

These two objects are crucial for managing the HTTP request-response cycle. The request object provides access to the incoming request's details, while the response object is used to craft and send the response to the caller.

In this case with:

res.statusCode = 200;

we set the statusCode property to 200, to indicate a successful response.

We set the Content-Type header:

res.setHeader('Content-Type', 'text/plain');

and we close the response, adding the content as an argument to end():

res.end('Hello World\n'); [1]

2. Node.js Internal Structure

V8: V8 is an open-source project developed by Google designed to execute JavaScript code outside the browser environment. It provides access to Node.js's underlying networking capabilities and plays a key role in managing concurrency—a core feature of Node.js. Approximately 70% of V8's codebase is written in C++, while the remaining 30% is written in JavaScript.

.

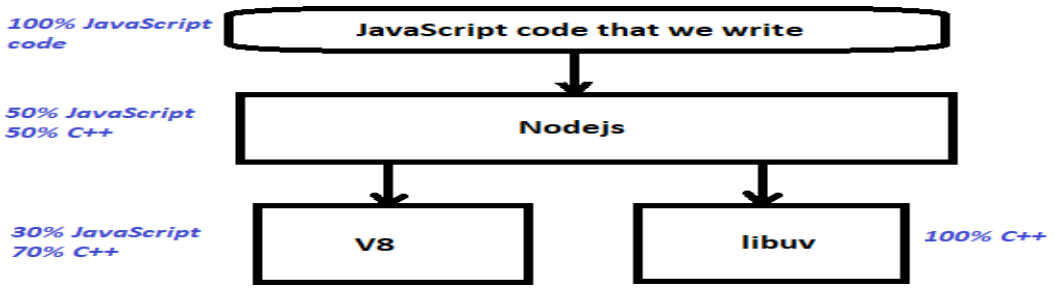

Fig-1:Node.jsinternalstructure-I

- **libuv:** libuv is an abstraction layer built on top of libraries like c-ares (for DNS), IOCP (for Windows asynchronous I/O), libeio, and libev. It handles and manages all input-output operations and events within the event loop. Simply put, libuv enables your JavaScript code to perform I/O operations such as networking, file handling, and more. It is the backbone for file system operations and TCP-level connectivity, and the library itself is entirely written in C++.

  As a JavaScript developer, you write your code in JavaScript, expecting it to compile and execute seamlessly. Node.js acts as the interface between your JavaScript code and the underlying open-source components (like V8 and libuv), which are implemented in JavaScript and C++. This abstraction means you don't need to directly interact with the C++ code. Instead, Node.js provides a unified API, allowing developers to use high-level JavaScript methods that internally relate to the C++ code running on your computer, ensuring smooth execution of your JavaScript programs. Refer to Fig-2 for a detailed overview.

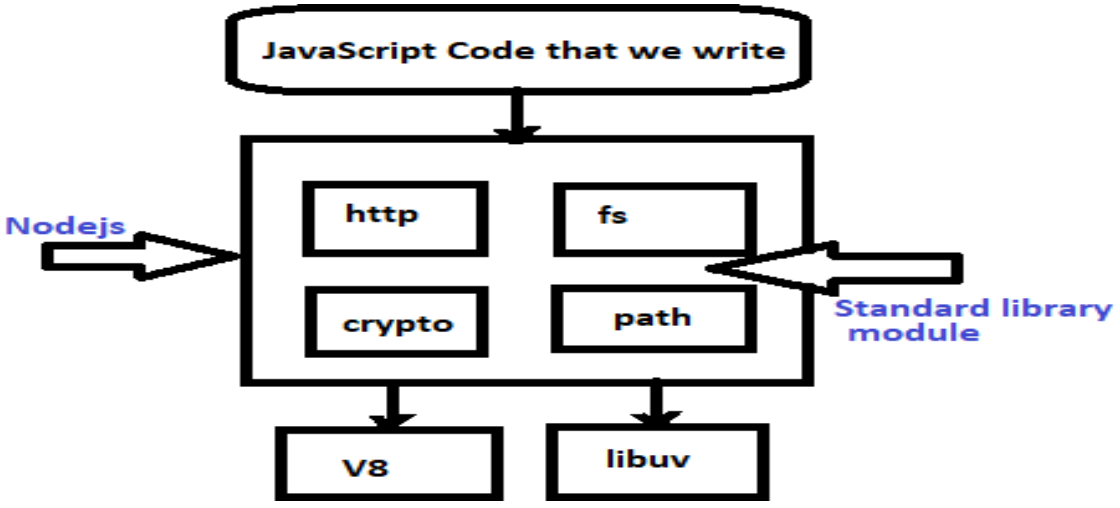

Fig-2:Node.jsinternalstructure-II

The library modules in Node.js, such as fs, http, path, and crypto, provide a consistent and easy-to-use API. These APIs ultimately interface with functionalities that are implemented within

the libuv project. As a developer, you interact with these modules using JavaScript functions, which internally invoke the underlying libuv project. You don't need to worry about the C++ code or the internal workings of libuv—Node.js and libuv handle those complexities for you.

3. Module System

JavaScript, by specification, does not provide a built-in API for managing module dependencies and isolation. Consequently, including multiple modules in a project traditionally involved exposing global variables. For instance, the jQuery module can be added to an HTML document by including the following line in the `<head>` tag:

`<script src="https://code.jquery.com/jquery-1.6.1.js"></script>`

The module is then accessed via the global `jQuery` object. However, this approach can lead to global namespace pollution and potential naming collisions.

To address this issue, Node.js introduced a modular system that avoids relying on global variables. Developers can create their own modules or utilize core and third-party modules. Node.js modules act as plugins, add-ons, and extensions to simplify the development process. Each Node module exposes a public API (Application Programming Interface) that can be accessed after importing the module into the current script. Node modules are categorized into **local modules, core modules, and third-party modules.**

NPM – The Node Package Manager

Node.js comes with built-in support for package management via **NPM**, a tool included by default with every Node.js installation. NPM modules function similarly to Ruby Gems, offering a collection of publicly available, reusable components. These modules can be easily installed from a web-based repository and include features for version and dependency management. A full list of NPM packages is available on the NPM website or through the NPM CLI, which is installed alongside Node.js. The ecosystem is open to everyone, allowing developers to publish their own modules to the repository.

Popular NPM Modules

Here are some of the most widely used NPM modules:

express: A Sinatra-inspired web development framework for Node.js, and the de facto standard for most Node.js applications.

connect: An extensible HTTP server framework for Node.js that provides a collection of high-performance middleware plugins.

socket.io and sockjs: Server-side components of two of the most popular WebSocket libraries

available today.

mongodb and mongojs: APIs for interacting with MongoDB databases in Node.js.

bluebird: A feature-rich Promises/A+ implementation with excellent performance.

moment: A JavaScript library for parsing, validating, manipulating, and formatting dates.

This list is by no means exhaustive. The NPM ecosystem hosts countless valuable packages, accessible to developers worldwide, and continues to grow daily.

4. KeyFeaturesofNode.js

Non-blocking I/O

The Node.js standard library provides asynchronous, non-blocking I/O methods that accept callback functions. Some methods also have synchronous (blocking) counterparts, whose names typically end with Sync.

Example of Blocking I/O:

```
const fs = require('fs');
const content = fs.readFileSync('/file.txt'); // Blocks here until the file is fully read
console.log(content);
moreWork(); // This runs only after console.log
```

Example of Non-blocking I/O:

```
const fs = require('fs');
fs.readFile('/file.txt', (err, content) =>
{
if (err) throw err;
console.log(content);
});
moreWork(); // This runs before console.log
```

In the first example, console.log executes before moreWork() because fs.readFileSync blocks execution. In contrast, in the second example, fs.readFile() is non-blocking, allowing JavaScript to continue executing moreWork() without waiting for the file to be read. This non-blocking approach is a fundamental design choice in Node.js that enables higher throughput.

Single-Threaded Event Loop

The Node.js platform does not follow the traditional Request/Response Multi-Threaded Stateless Model. Instead, it adopts a **Single-Threaded Event Loop Model** based on JavaScript's event-driven architecture and callback mechanisms.

Thanks to this architecture, Node.js efficiently handles multiple concurrent client requests.

The **Event Loop** is the core of the Node.js processing model. It enables the platform to manage non-blocking operations and asynchronous tasks, making it highly effective for applications requiring high concurrency and real-time processing.

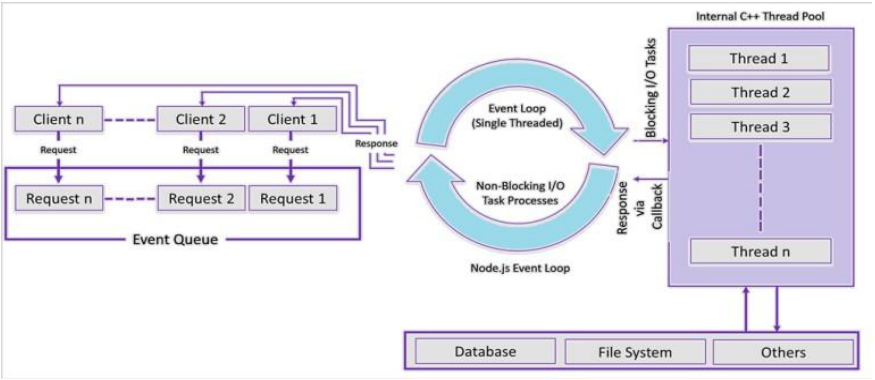

Fig-3:Node.jsApplication/Server

Single-Threaded Event Loop Model: Processing Steps

a) **User Request**:
   A user sends a request to the server.

b) **Thread Pool**:
   The Node.js web server maintains a limited thread pool to handle client requests requiring blocking operations.

c) **Event Queue**:
   Incoming requests are placed into an "Event Queue."

d) **Event Loop**:
   The Node.js server contains a core component called the "Event Loop." This loop continuously checks for new requests in the Event Queue and processes them. It is called an "Event Loop" because it uses an infinite loop to receive and handle requests.

e) **Pseudo-code for understanding:**

```
public class EventLoop {
   while (true) {
      if (EventQueue receives a JavaScript function call) {
         ClientRequest request = EventQueue.getClientRequest();
         if (request requires Blocking IO or extensive computation) {
            Assign request to Thread T1;
         } else {
            Process and prepare response;
         }
      }
   }
}
```

f) **Single Thread**:

The Event Loop operates on a single thread and is the core of the Node.js processing model.

g) **Waiting for Requests**:

- If no requests are in the Event Queue, the Event Loop waits indefinitely for incoming requests.
- If requests are present, the Event Loop retrieves one request and starts processing it.

h) **Non-blocking Requests**:

- If the request does **not** require any blocking I/O operations, the Event Loop processes it directly, prepares the response, and sends it back to the client.

i) **Blocking Requests**:

- For requests requiring blocking I/O operations (e.g., database access, file system interactions, or external services), the Event Loop follows a different approach:
  - **Check Thread Pool**: It checks for available threads in the internal thread pool.
  - **Assign Thread**: Assigns the request to an available thread.
  - **Thread Execution**: The thread processes the request, performs the blocking I/O operation, prepares the response, and returns it to the Event Loop.

j) **Response Handling**:

The Event Loop sends the final response back to the respective client.

- This efficient model allows Node.js to handle multiple concurrent client requests effectively, even with a single-threaded Event Loop.

5. Reasons for why Node.js used widely by Modern Web Developers:

Google V8 JavaScript Engine

Node.js uses the Google V8 engine to execute JavaScript code. Unlike other JavaScript interpreters, the V8 engine compiles JavaScript into native machine code. This enables the runtime environment to significantly enhance the performance of web server applications by executing JavaScript more quickly and efficiently.

Asynchronous I/O Operations

Node.js handles all I/O operations asynchronously using a single-threaded event loop. This advanced approach allows Node.js applications to send asynchronous tasks to the event loop with a callback function. While the async task is being processed, the application continues executing the remaining code. Once the operation completes, the event loop returns to the task and executes the callback function. This method not only reduces memory consumption but also allows Node.js to efficiently manage a large number of concurrent connections. Developers can use this runtime environment for common tasks like file system operations, network connections, and database read/write operations.

Robust Tooling

Node.js developers benefit from a reliable package manager like npm. npm is fast, consistent, and robust, and it simplifies the process of managing project dependencies while preventing version conflicts. Additionally, developers can leverage powerful file streaming tools like Broccoli, Gulp, and Brunch, as well as popular task runners like Grunt.

Real-Time and Multi-User Applications

In addition to supporting responsive web design, developers today often need to create real-time and multi-user web applications. Node.js makes it easier to develop complex applications such as gaming, chat, and communication tools. With WebSocket protocols, developers can build real-time applications where data is pushed from the server to the client more efficiently, without the overhead of HTTP. Additionally, the event loop feature of Node.js enables the creation of multi-user applications that handle numerous simultaneous connections.

Facilitating File Streaming

Web developers can take advantage of Node.js's efficient I/O capabilities to speed up file streaming from the file system. By using the runtime to manage read/write streams over HTTP and WebSockets, developers can reduce processing times for tasks like transcoding audio or video. For instance, a programmer can stream data directly from the web server to a browser via WebSockets, enabling real-time display of output.

Popularity of JavaScript

JavaScript has been an integral part of web development since the early days of the web. It gained further prominence with the advent of AJAX and continues to be the go-to language for client-side scripting. The familiarity of JavaScript and Node.js's adherence to the language for both client-side and server-side development has driven widespread adoption. By leveraging JavaScript's best features and nurturing a vibrant community, Node.js has grown into a popular platform with a continually expanding user base.

Single Codebase for Frontend and Backend

Node.js allows developers to write both client-side and server-side code in JavaScript, bridging the gap between frontend and backend development. This enables programmers to use a single language for building both the frontend and backend of web applications, streamlining development and reducing the complexity of managing separate codebases.

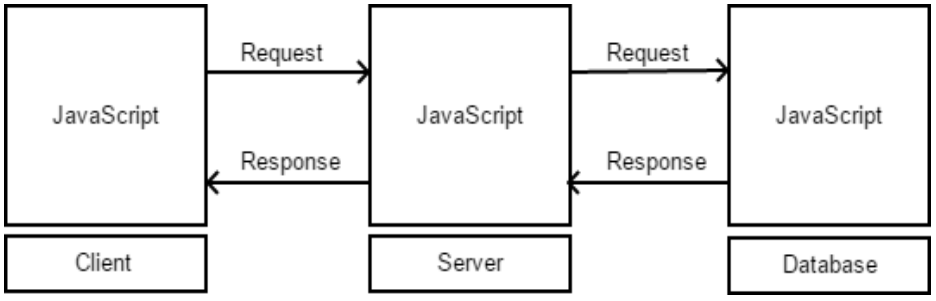

Fig-4: JavaScriptend-to-end

6.Why Are Major Companies Choosing Node.js?

PayPal

PayPal is a leading global platform for online transactions, allowing users to send money and conduct business in more than 100 currencies. By 2015, the company had over 184 million active customer accounts. PayPal leverages Node.js to develop the client-side of its web applications.

Why Node.js?

Jeff Harrel, Senior Director of Payments Products and Engineering at PayPal, explains that Node.js helps bridge the gap between the browser and server by enabling both to be built using JavaScript. This approach unifies development teams, improving their ability to meet client needs across the technology stack.

Results:

By adopting Node.js, PayPal developed its application twice as fast, using fewer developers. The new system required 33% fewer lines of code and 40% fewer files than its previous Java-based solution.

LinkedIn

LinkedIn, a professional networking platform founded in 2002 in Mountain View, California, serves over 400 million users across 200+ countries. The company utilizes Node.js to handle the server-side operations of its mobile application.

Why Node.js?

According to Kiran Prasad, LinkedIn's Mobile Development Lead, the decision to use Node.js was based on two key factors: scalability and its efficiency in interacting with other services.

Results:

The Node.js-based mobile app is up to 10 times faster than its predecessor built with Ruby on Rails. Additionally, it significantly reduces server resource usage, bringing it down from 30 servers to just 3. The development process also became much more efficient.

Yahoo

Yahoo, a global technology company, offers a web portal, search engine, and various online services, attracting more than 500 million users monthly in over 30 languages.

Why Node.js?

Eric Ferraiuolo, Principal Software Engineer at Yahoo, emphasizes that Node.js provides scalability, and every feature transitioned to Node.js has demonstrated improved performance.

Results:

The adoption of Node.js has revolutionized Yahoo's frontend development culture, contributing to the efficiency of multiple Yahoo websites.

Netflix

Netflix, the world's largest streaming service, provides movies and TV shows in over 190 countries. As of April 2016, it had more than 81 million subscribers, including 46 million in the U.S. Netflix has built its entire user interface using Node.js and intends to extend its use to other parts of its infrastructure.

Why Node.js?

The Netflix team chose Node.js to create applications that are lightweight, modular, and fast. This shift has reduced the startup time of their applications by 70%.

GoDaddy

GoDaddy, a publicly traded company specializing in domain registration and web hosting, manages over 61 million domains and serves more than 13 million customers as of January 2016. The company migrated its entire backend system to an open-source Node.js framework.

Why Node.js?

Stephen Commisso, Senior Software Developer at GoDaddy, cites the ability to build high-quality applications, deploy new features quickly, write efficient tests, and leverage NPM as the main advantages of using Node.js.

Results:

Antonio Silveira, Vice President of Engineering at GoDaddy, reports that the company now

operates with 10 times fewer servers while maintaining the same workload. Additionally, Time To First Byte (TTFB) has improved significantly, dropping from approximately 60ms to just 12ms. Performance has become a key differentiator, particularly in comparison to Google's search results.

According to Stack Overflow survey 2019 the Node.jsis the most commonly used and most wanted technology [4] give a look in Fig-5 and Fig-6.

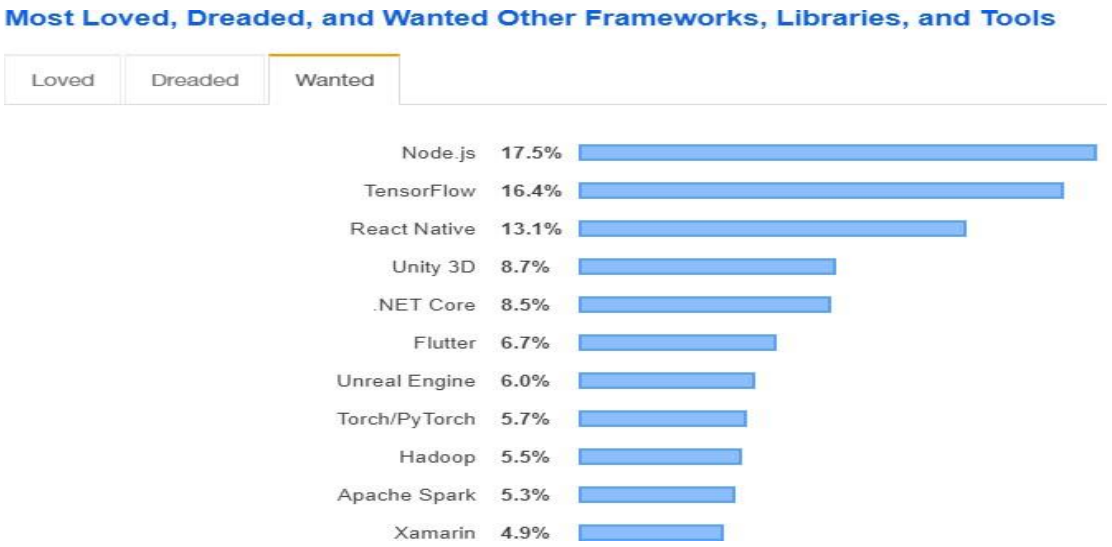

Fig-5: Applications of Node.js

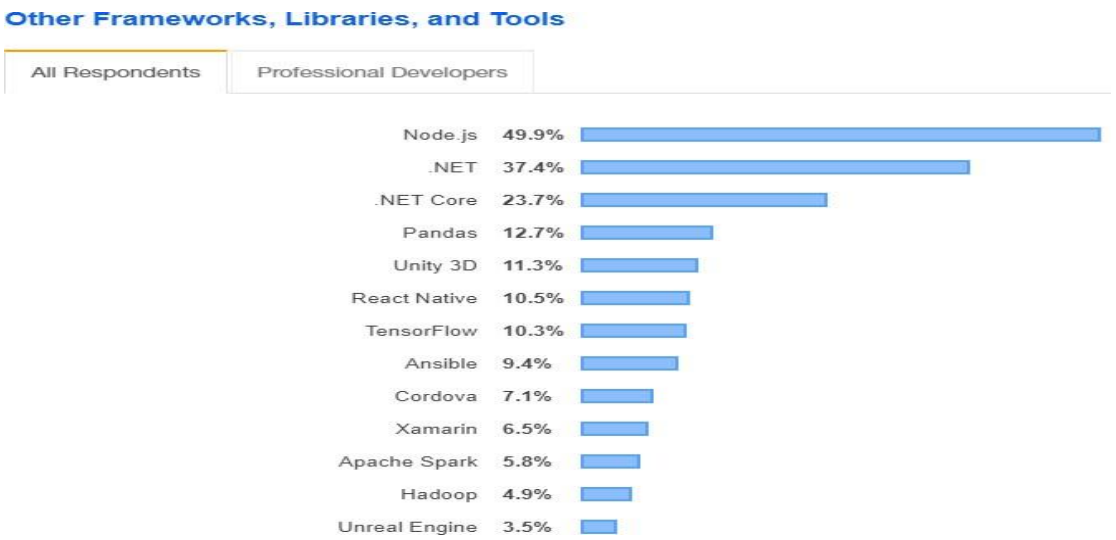

Fig-6: Market capture by Node.js

Potential application areas of Node.js

- Media
- Paymentgateways
- Ecommerce
- Socialmedia
- Enterprisewebapps
- Backend/API for mobile apps

7. Conclusion

Node.js has revolutionized the usability of JavaScript, turning it into a comprehensive programming language. It extends JavaScript beyond the browser to server-side scripting, offering a powerful runtime environment. With a vast library of free, useful modules accessible via the built-in tool NPM, Node.js provides seamless functionality. By leveraging event-driven I/O and non-blocking asynchronous programming, Node.js ensures lightweight and efficient performance. Businesses adopting Node.js can benefit from reduced server requirements, fewer developers, and faster page load times.

References

[1] https://Node.js.org/en/docs

[2] Node.jsinActionbyMikeCantelon,MarcHarter,T.J. Holowaychuk, Nathan Rajlich.

[3] https://brainhub.eu/blog/9-famous-apps-using-node-js

[4] https://insights.stackoverflow.com/survey/2019

[5] https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop

[6] https://www.toptal.com/nodejs/why-the-hell-would- i-use-node-js

[7] https://nodejs.org/api

[8] A Comparative Analysis of Node.js (Server-Side Java Script) Nimesh Chhetri.