1. Technologies and Toolset

1.1. Node

The official website (http://www.nodejs.org) defines Node as "a platform built on Chrome's JavaScript runtime for easily building fast, 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 [1]."

Regardless, JavaScript is the world's most famous programming languages. If you have done any programming for the web, it's unavoidable. JavaScript, in view of the sheer reach of the web, has satisfied the "compose once, run anywhere" dream that Java had back in the 1990s. Around the season of the Ajax insurgency in 2005, JavaScript went from being a "toy" language to something individuals wrote genuine and noteworthy projects with. A portion of the eminent firsts were Google Maps and Gmail, yet today there are a large group of web applications from Twitter to Facebook to GitHub.

JavaScript has for some time been the true standard for frontend side web development. While about all frontend code is composed in JavaScript, server-side development is a variety of choices between PHP, Java, and various different technologies. Life as a web engineer would be substantially more straightforward if a single language was utilized all around. Since JavaScript overwhelms in the browser, it bodes well to utilize it on the server too.

The idea of server-side JavaScript is not a new one. Netscape initially introduced JavaScript into the server world in 1994. Since that time, a lot of projects have endeavoured, and failed, to advance JavaScript as a server-side language. Execution, or scarcity thereof, restricted JavaScript from picking up a genuine a dependable balance in the server space.

Throughout the years, JavaScript has seen gigantic upgrades in performance. Because of its pertinence in the program, enormous players like Google have contributed a considerable measure of time and cash to make JavaScript as fast as possible. In 2009, Ryan Dahl of Joyent, put the large part of that recently discovered execution to great use on the server when he made the Node.js structure. Dahl assembled Node.js over Google's V8 JavaScript engine. V8 is a similar engine that has given Google Chrome its astounding JavaScript performance, and helped it turn into the most well-known browser on the planet.

1.1.1. Technical Details

— **Threading**

Node.js works in a single threaded environment, utilizing non-blocking I/O calls, enabling it to support a huge number of simultaneous connections without incurring the cost of thread context switching [17]. The idea of sharing a single thread among every one of the requests that utilization the observer design is for building exceptionally concurrent applications, where any function performing I/O must utilize a callback. With a specific end goal to accommodate the single-threaded event loop, Node.js uses the libuv library that, utilizes a fixed-sized thread pool that is in charge of a portion of the non-blocking async I/O operations [18].

A drawback of this single-threaded approach is that Node.js does not permit vertical scaling by increasing the amount of CPU cores of the machine it's running on while not using a further module, like cluster, StrongLoop process Manager, or pm2. However, developers can increase the default range of threads within the libuv thread pool; these threads are probably to be distributed across multiple cores by the server software system [19].

Execution of concurrent tasks in Node.js is handled by a thread pool. the main thread decision functions post tasks to the shared task queue that threads within the thread pool pull and execute. Inherently non-blocking system functions like networking interprets to kernel-side non-blocking sockets, whereas inherently blocking system functions like file I/O run in an exceedingly block method on its own thread When a thread in the thread pool completes a task, it informs the main thread of this, which wakes up and execute the callback. As callbacks are handled synchronously on the main thread, long lasting computations and other CPU-bound tasks will freeze the whole event-loop till completion.

— **V8 Engine**

V8 is the JavaScript execution engine built for Google Chrome and open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code instead of interpreting it in real time [18].

Node.js makes use of libuv to handle asynchronous events. Libuv is an abstraction layer file system and network functionality on each Windows and POSIX-based systems like UNIX operating system, macOS, OSS on NonStop, and Unix.

The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written in C++, connect these technologies to each other and to the operating system.

— **Package Management**

npm is an in-built package manager for the Node.js servers. It is utilized to install Node.js programs from the npm registry, sorting out the installation and

administration of third-party Node.js programs. npm isn't to be mistaken for the CommonJS âĂŸrequire()âĂŹ statement. It is not utilized to load code; rather, it is utilized to install code and manage dependencies from the command line. The bundles found in the npm registry can extend from basic helper libraries, for example, Lodash to task runners suck as Gulp.

— **Unified API**

Node.js may be combined with a browser, a database supporting JSON information (such as Postgres, MongoDB, or CouchDB) and JSON for a unified JavaScript development stack. With the difference of what were basically server-side development patterns like MVC, MVP, MVVM, etc., Node.js permits the reuse of the same model and service interface between client-side and server-side.

— **Event Loop**

Node.js registers itself with the software system so as to be notified once a connection is created, and also the OS can issue a callback. inside the Node.js runtime, every connection could be a small heap allocation. historically, comparatively heavyweight OS processes or threads handled every connection. Node.js uses an event loop for scalability, rather than processes or threads [74]. In distinction to different event-driven servers, Node.js's event loop doesn't need to be known as explicitly. Instead callbacks are outlined, and also the server mechanically enters the event loop at the end of the callback definition. Node.js exits the event loop once there are not any any callbacks to be performed.

— **Non-blocking**

The question of whether an operation is blocking or non-blocking refers to the fact that it must finish before the next operation begins. Non-blocking operations are said to be asynchronous and blocking operations are said to be synchronous. Node is non-blocking i.e. operations don't have to happen consecutively.

— **Scalability**

Node.js utilizes a single threaded model with event callbacks. Events encourage the server to react in a non-blocking way and makes the server scalable compared to traditional servers which give restricted access to threads to deal with requests. Node utilizes a single threaded program to provide services to a substantially bigger number of requests than traditional servers like Apache HTTP Server.

— **Memory Management**

Since Node is single-threaded, that implies that every one of your users will be sharing the same memory allocation. At the end of the day, unlike to in the browser, you must be mindful so as not to store user-specific information in closures where different connections can affect it.

1.1.2. Middleware

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides a thin layer of fundamental web application features, without obscuring Node.js features [17].

Express is the most prominent framework for Node applications, and it highlights middleware utilizing continuation passing. When you need to run a similar code for conceivably a wide range of routes, the perfect place for that code is likely middleware. Middleware is a function that gets passed the request and response objects, alongside a continuation function to call, called next(). Envision that you need to add a requestId to each request/response pair with the goal that you can follow them back to the individual request when you're troubleshooting or debugging your logs for something.

You can write some middleware like this:

```
require('dotenv').config();
const express = require('express');
const cuid = require('cuid');

const app = express();

// request id middleware
const requestId = (req, res, next) => {
  const requestId = cuid();
  req.id = requestId;
  res.id = requestId;

  // pass continuation to next middleware
  next();
};

app.use(requestId);

app.get('/', (req, res) => {
  res.send('\n\nHello, world!\n\n');
});

module.exports = app;
```

1.2. PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML [18].

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something" (in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow you to jump into and out of "PHP mode."

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve [18].

PHP began as a small open source venture that advanced as an ever-increasing number of people discovered how valuable it was. Rasmus Lerdorf released the primary rendition of PHP back in 1994.