ISSN: 2230-9926

Available online at http://www.journalijdr.com

IJDR

International Journal of Development Research

Vol. 10, Issue, 05, pp. 35539-35550, May, 2020

https://doi.org/10.37118/ijdr.18695.05.2020

RESEARCH ARTICLE OPEN ACCESS

PERFORMANCE ANALYSIS AND MONITORING OF COMPUTATIONAL RESOURCES WITH A COMPARISON BETWEEN NODE.JS. AND AS PHP AND .NET CORE PLATFORMS

[1]Luiz Felipe Carvalho Libertino, [1]Aldemir Carvalho Uchôa Neto, [1]Bruno Pereira Gonçalves, [1]Jean Mark Lobo de Oliveira, [1]Rilmar Pereira Gomes and *,[2] David Barbosa de Alencar

[1]Academic Department, University Center FAMETRO, Amazon-Brazil; [2]Institute of Technology and Education Galileo of Amazon (ITEGAM), Brazil

ARTICLE INFO

Article History:
Received 02nd February, 2020
Received in revised form 06th March, 2020
Accepted 28th April, 2020
Published online 25th May, 2020

Key Words:

Node.js; JavaScript; Non-blocking; Event-Loop; V8.

Corresponding author: David Barbosa de Alencar

ABSTRACT

This article aims to compare the performance of the Node.js development platform. with the PHP and .Net Core platforms, in addition to demonstrating the internal workings of the Node.js platform, which uses V8 engineering, the same used in the Google browser, to compile the Javascript language, whose compilation time is translated into the machine language with the JIT engine. Three applications were developed to compare the speed and operation of requests for each of them, in addition to the time for each request and the total time spent. The survey showed that the vast majority of respondents know and recommend the knowledge and use of Node.js, as its ease and performance are quite high, in addition to the possibility of developing in a single language, Javascript, both mobile devices, servers and Applications from the web.

Citation: Luiz Felipe Carvalho Libertino, Aldemir Carvalho Uchôa Neto, Bruno Pereira Gonçalves, Jean Mark Lobo de Oliveira, Rilmar Pereira Gomes and David Barbosa de Alencar, 2020. "Performance analysis and monitoring of computational resources with a comparison between node.js. and as php and .net core platforms*", International Journal of Development Research*, 10, (05), xxxxxxxxxxxxxx.

INTRODUCTION

Web development has been improving more and more with the addition of new technologies. Everything about solving problems of data consumption, production content, etc. Thus, increasing the need for interaction between users with these services on the web, without delay to receive your data. These applications require a highly performative and fast processing power, so that it gives the sensation of real-time interaction between client and server. And more, it must exactly happen on a massive scale, supporting hundreds to millions of customers. created natively on .NET, Java, PHP, Ruby or Python platforms present one thing in common: they stop processing while using I/O on server. This shutdown is known as the Blocking-Thread model. To resolve this issue, Node.js was created in late 2009, Ryan Dahl with the help of of 14 employees developed this platform that has a model innovative: totally non-blocking thread architecture. Node.js is a powerful application platform to easily and quickly build scalable network, and that uses the high performance open source JavaScript engine from Google Chrome browser, the V8 engine, which is written in C++ (William Moraes, 2015). Applications that use a lot of file processing and / or perform a lot of I/O, adopting this type of architecture will result in good performance in relation to the consumption of memory. With the advantages of using Javascript as a programming language, an untyped, multiplatform language, same language for development web, backend and recently being interpreted natively for mobile. Node.js deals single-thread only (single thread per process), using an agent to listen and broadcast events in the system with Event-Loop. Being a highly scalable and low level thanks to the Javascript V8 engine, the same used in Google Chrome browser. V8 is an extremely optimized JavaScript engine that works like a JIT compiler (Just-In-Time) - concept introduced by McCarthy (1960) - and turns high-level code into machine code. Your advantages are: working in real time, flexibility, lightness, team productivity, largest repository in the world, innovation environment, asynchronous and event-based. Its disadvantages are: having low object orientation, weak typing, being new thus losing credibility for bigger and heavier projects. For better understand how technology can support thousands of connections and maintain its high efficiency, we do a detailed study in this article on Node.js and the optimization techniques exercised inside the platform, in addition to some test's comparative with other programming languages.

Bibliographic Reference: Presenting the theoretical foundation aboutmaterials,technologies, languages programming presented in the development of the article.

Node.Js: Platform that allows you to run Javascript outside the browser domain. Being one set of libraries that run on the V8 engine, allowing you to run the JavaScript code on the server (Ryan Dahl, 2009). The entire origin of the runtime (run time execution) starts with the limited possibilities of the Apache HTTP Server - the server of the most popular web at the time - to handle many simultaneous connections. Node.js came to combat this problem, with a type of non-blocking architecture, because the Most codes were sequential, leading to an entire process block or multiple execution stacks in the case of multiple simultaneous connections.

Javascript: High-level interpreted scripting language (Brendan Eich, 1995). Javascript was created exclusively to meet Marc Andreessen's idea of having "language glue" between HTML and web designers, which must be easy to use to assemble components like images and plug-ins, so that the code was written directly in the markup of the web page.

Google Forms: Free service to create online forms, created by Google in 2008. The user can produce multiple-choice surveys, ask discursive questions, request numerical scale assessments, among other options. The tool is ideal for who needs to request feedback on something, organize event registrations, invitations or ask for ratings.

Visual Studio Code: Source code editor developed by Microsoft in 2015 for Windows, Linux and macOS. Includes debugging support, built-in Git control, syntax highlighting, smart code completion, snippets and code refactoring. Supporting numerous programming languages and a set of features that may or may not be available for the given language.

PHP**:** PHP is a programming language that was created in 1994, source code was released in 1995 as a CGI problem pack created by Rasmus Lerdorf, with name (Personal Home Page Tools), we are currently at version 7.4.2, PHP issued by developers to build dynamic websites, is often used more on the server side, it can be used as application integration, in the development part it streamlines processes, is known language and one of the most used because it is easy to learn, to handle, as is compatible with all operating systems, makes your cost less.

.Net Core: The .Net Core was primarily developed by Microsoft and released under the license MIT, launched in 2016, is currently 3.1, version 5 is scheduled for November2020, .Net Core is a free and open source framework for Windows systems, Linux and macOS, is a successor to the .Net Framework, basically it's a leaner version, is fully supported in C # and F # and partially Visual Basic .Net.
s
Threads**:** The thread is a subsystem being a way for a process to split into two or more tasks, these tasks can be performed simultaneously to perform more program quickly, during the rapid change of thread, apparently it is like that each one was running in parallel, in node.js, it was never a better option for heavy CPU computing, and because of that it is difficult to use in machine learning, AI and data Science projects.

Callback**:** Callback is nothing more than a piece of executable code that is passed as a parameter or method, the method is expected to execute the code at some point, the execution of the segment can be immediate as in a synchronous callback, or in another moment, as an asynchronous callback, it is very common in Javascript on the client side and server (Node.js), call-back are supported in different languages, but are implemented with subroutines, expressions, blocks of code.

MATERIALS AND METHODS

Javascript: Programming language used for the application, we use its syntax in the examples along with the application in Node.js;

Visual Studio Code*:* we use the text editor to create the source code of the applications: Node.js, PHP and .Net Core.Node.js - platform where the algorithms will run on the server side along with JavaScript, being able to see the way it acts in the systems, and observing the improvement of responses and performance compared to other competitors;

Google Forms: We use the platform to create the 105 questionnaires, are both multiple choice and dichotomous, questions are closed and can only be choose one of them, we share via WhatsApp and E-mail to the programmers from Brazil.

METHODOLOGY**:** To obtain knowledge and develop research, we use bibliographic research, in which we select books and articles that on the Node.js platform and Javascript developed on the server side. News focused on the operation of the V8 engine to interpret Javascript code, advantages of using the Javascript programming language on the server side, your single-threaded event-oriented I / O model, non-blocking and asynchronous. Was descriptive research was used in order to expose the frequency with which Node.js structure and function, aiming at its identification, registration, factors or variables that associate with the case. Quantitative research was also used, applying a questionnaire of 10 closed, dichotomous and multiple choice questions, using the Google Forms platform, applied with programmers, in communities of development on the social network Facebook.

RESULTS AND DISCUSSION

Web systems created natively on .NET, Java, PHP, Ruby or Python have one thing in common: they stop processing while use an I / O on the server. This shutdown is known as a blocking model (Blocking-Thread). In a web system we can observe it in a broad and functional way. Considering that each process is a request made by the user. As the application progresses, new users' access, generating new requests to the server. For each request it is created a new thread, to proceed it. Generating a demand for resources computational (such as RAM). Since these resources are limited, threads will not be created infinitely, and reaching that limit, new requests they will have to wait for the dispersion of these allocated resources to be treated. A blocking system queues calls and then processes them, one by one, not allowing multiple processing. While a call is being processed, others are idle, maintaining a queue of requests waiting for a period of time. This is a classic architecture, existing in countless systems and that has a unproductive design.With the increase in requests in the system, the frequency of bottlenecks will be greater, expanding the need to update the server hardware. But as updating the machines is very complicated, the ideal would be to look for new technologies, making good use of existing hardware and making the most of the power current processor, not keeping you waiting when tasks like blocking. To resolve this issue, Node.js was created in late 2009, Ryan Dahl with initial help from 14 employees developed this platform that has a model innovative: totally non-blocking architecture. The start of the runtime starts with the limited capabilities of the Apache HTTP Server - the web server most famous at the time - to handle many simultaneous connections. In addition, Dahl criticized the way of writing the code, which was sequential, this could cause an entire process block or multiple execution stacks in case of multiple connections simultaneous.

Node.js is a powerful platform for applications, to build easily and scalable network applications quickly and using the open source JavaScript engine high-performance Google Chrome browser, the V8 engine, which is written in C ++ (William Moraes, 2015). Applications that use a lot of file processing and / or performs a lot of I / O, adopting this type of architecture will result in good performance in memory consumption, making the processing capacity of servers. With the advantages of using Javascript as a programming language, an untyped, multiplatform language, same language for development web, backend and recently being interpreted natively for mobile. The fundamental feature that differentiates Node.js from other technologies, such as PHP, Java, C #, is the fact that its execution is single-thread. In short, just one thread is responsible for executing the application's Javascript code, as in the other languages the execution is multi-thread. As explained earlier, in the blocking model, the requisitions created generate a new thread to handle it, however, within each one, the waiting time of the response is not used, leaving this part of the CPU idle, locked to other actions, so for each new thread consuming computational resources, reaching at a time when resources run out and they are idle for completion of a request, see Figure 1.

In Node.js, only one thread is responsible for handling requests. This thread is called an Event Loop, taking this name because each request is treated as an event. It is running waiting for new events to be dealt with, and for each request, a new event is created, see Figure 2. This means that the operations of input and output (eg database access and reading system files) are asynchronous and do not block the thread. Such events appear exclusively in this queue when they are broadcast during emissions of events in the application (Caio Pereira, 2013). The Event Emitter, is the module responsible for issuing events, and most Node.js libraries inherit from this its functionality to broadcast and listen to events. When a defined code emits an event, it is sent to the event queue so that the Event Loop execute and then return your result in a callback. A callback is a listener, a function passed by parameter to another function, to be executed after a certain event. Javascript is completely asynchronous and Node.js takes full advantage of this. While the Event Loop handles an I / O operation for a system thread asynchronously and progresses by handling other requests that arise in its stack of events, the threads of the traditional model await the completion of operations I/O, consuming computational resources throughout this waiting period. Although Node.js is single-threaded, its architecture facilitates a greater number of competing requests are handled in comparison to the traditional model, which is restricted due to high computational consumption due to the formation and continuity of threads to each request.Node.js has the advantage of not suffering from deadlocks, simply because work in single-thread.

Comparisons: Node.js, PHP and .Net Core are popular platforms for websites, APIs and others types of web content. Obviously, they have similarities, but their differences outweigh these affinities.

Types of application: Here we will see where each case is most used, where the programmer finds more ease to develop an application, solution, system, API, etc.

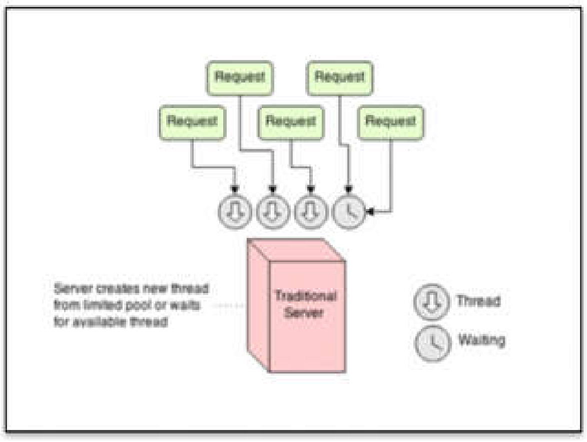

Request
Request
Request
Request
Request
Server creates new thread from limited pool or waits for available thread
Traditional Server
Thread
Waiting

Fig. 1: Blocking Model (Source: Opus-Software)

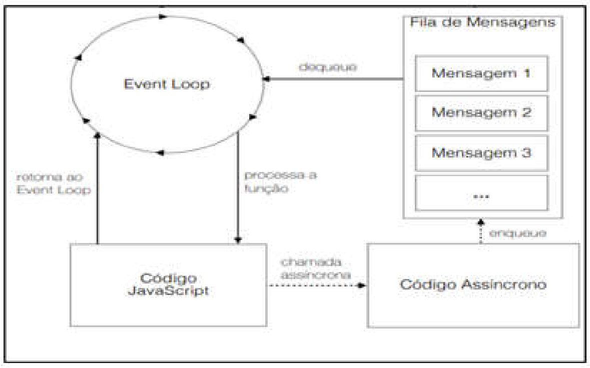


Fig. 2: How the Event Loop works (Source: Authors, (2020).

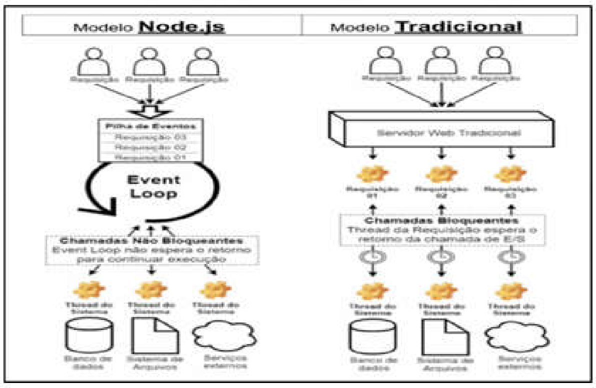


Fig. 3: Comparison of blocking and non-blocking models (Source: Opus-Software)

Node.js: Node.js has very common use cases in: Real Time Applications-One of its usability is in chat applications. Demanding very little processing and usually consists of transferring messages from one to another; Scalable Environments-With its non-blocking architecture, it is totally suitable for applications with a large number of concurrent connections, due to the potential for support this demand compared to other traditional servers; Layer of Server Entry - Doing little data processing and just passing forward requests, communicating with backend services, such as (Files, etc.);

Mocks and Prototypes - Easy creation of APIs and backend services with agility, thus being able to simulate the communication with an external service, for example; API with NoSQL behind - Because NoSQL is based on JSON (JavaScript Object Notation), your communication with Node.js, is very automatic. Not being necessary to do data conversions, for example, because the same objects stored in the database can be used in the front-end without the lack of no type of treatment or conversion; Others - Can be used for systems of the type IoT (Internet of Things), game server, recently Desktops applications.

.Net Core: .Net Core has quite common applications in: Web UI and Web APIs - For having enough resources that helps in the construction of this type of application, with ease in integrate with client-side tools; Building IoT apps - Easy integration, features and robust; Automated systems - Applications that require to be run from time to time. Because .net core can be compiled into an windows or Linux, thus integrating with operating system tools; Desktop Systems - As part of Microsoft, your tools come standard in their libraries, thus facilitating the construction of this type of system, in addition to accessible integration with the SQL Server database.

***PHP*:** PHP has common use in: Full Web Development - with its ease of integrate with the web, database and server. The integration of the system as a whole is simplified; CMS- Content Management System - These are systems that help manage content, such as: blogs, forums, online stores. Because of community there are several ready-made structures on the web, leaving the developer to make changes to use. Facilitating the construction of this type of site, the most currently used is the famous WordPress.

Conclusion on application types: Node.js, PHP and .NET Core are widely used to handle web requests. Being executed by a server and handling calls that are routed by then. Being able to use them to display static content, dynamic web pages, real-time web applications and data requests. However, Node.js and .Net Core, both can be used for embedded systems, IoT, Desktops and games. Like this having wide usability for application developments. Node.js is a very popular alternative when it comes to serving web sockets - the initial implementation is the socket.io library. However, PHP has sockets available since 2003, using the Ratchet library, however its architecture blocking, does not meet the demand well. In another way, the .NET Core has SignalR for socket procedures, since .Net Core 1.1, which was launched in 2016, helping a lot the development of programmers in the C # language, with results satisfactory to the demand for requisitions.

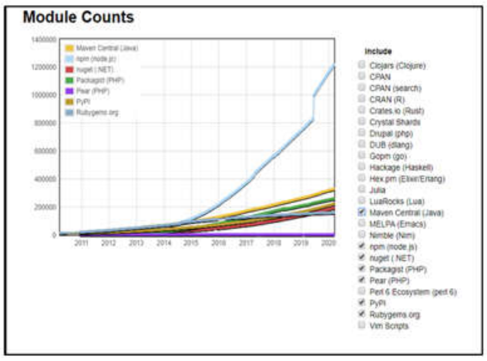


Fig. 4: Module Count by Package Manager (Source: Module Counts)

***Extensibility*:** These technologies can be extended to meet the needs of each programmer. Being able to fork the source code, change it and compile it to extend. There are add-ons and package managers for both platforms.

***Node*.js:** NPM (Node Package Manager) is the package manager for Node.js and is also the largest software repository in the world, as seen in Figure 4. Making theNode.js platform a potential to be applied in any situation and implementation as appropriate. One of the most popular packages is called Express.js Thus providing an enormous amount with reusable code packages and probably that integration you need with another system or database that have already been implemented. However, this number is totally outside the curve of the others. Managers of other languages, is justified by the fact that the class that develops with JavaScript, loves to create a package for him to wrap another package, whose objective is call a function that you have natively on the platform.

.Net Core: .Net Core was produced by a series of independent components and interfaces and with well-defined particularities. Native components generate an interface or inherit from abstract classes. It is possible to change the main components or extend its procedures by others of its own implementation.

- You can use the standard implementation of the component, as is. (Sufficient in most cases);
- It is possible to derive a subclass of the standard implementation to adjust your behavior.
- Replace the component entirely with a new implementation of abstract basic interface or class.

Having the Nugget package as a package manager, with a stable and respectable, meeting the needs of each developer.

***PHP*:** PHP has two popular package managers: PEAR and Packgist. Having more time of existence, having the community more time to create means for solutions. So, resolving programmer issues from the beginning of language. With 80% of internet applications created in PHP, a myth has been created about it, like the “PHP curse”. Because of its easy implementation for the development, made it possible for anyone to publish a web system. These systems due to amateurism, often lacked validations, responsiveness or good HTML rendering. So, creating the myth about language be bad, bad, not robust, etc. Because of every 10 projects carried out in PHP, 3 of them were of quality. There is a very good English proverb which is “One man's trash is another man's treasure ”, which means that trash for some can be a treasure for other, in summary these 3 successful projects were enough to help implementations of other applications, helping the community to grow and conquer the web.

Conclusion on extensibility: Both platforms mentioned in this article, have excellent extensibility, theNode.js community just growing and growing, stable PHP community and .Net Moderate core, but a little more closed, opening gradually. They all have excellent documentation, videos, blogs for learning. All these being open-source platforms, with open source, enabling the creation of extensions, libraries, tools, etc.

Performance - advantages and disadvantages: We mention performance here, knowing it is a loaded topic. After all, the final performance of an application depends on many factors such as: Programming; the environment; the performance of dependencies such as databases and usage affecting the performance. Like Node.js, PHP and .Net Core are generally not used as similarly, it will be difficult to compare performance directly.

Node.js: Node.js will normally be used to handle requests for API, web applications that use a lot of I / O, video streaming, real-time applications, so it’s usually just a thin layer between a web client and a database. You should definitely not use Node. js for CPU intensive operations, using it for heavy computing will nullify almost all of its advantages as it performs only on a single thread. Where it really is used is in building applications scalable and fast networks, due to the ability to support a large number of simultaneous connections with high throughput, equivalent to high scalability. This fact is because the requests are not continuous but asynchronous, making the execution very fast and perfect for a high number of requisitions. Using the event scheduling model, thus not blocking the executions while waiting for an I/O So not losing performance or consuming lot of hardware resources. Allowing web developers to program with a JavaScript is the only language, without needing another one for the development backend. Having the disadvantages of being new compared to other platforms, thus not passing confidence to larger projects. Its asynchronous aspect cans bea complicator, for an application with a lot of coding.

.Net Core: Used extensively in building APIs that consume databases relational, web applications, IoT applications, real-time applications. Because of your test-driven architecture, high performance, native dependency injection, code openness and focus on the community are huge differentials that help with growth of his popularity. Taking advantage of asynchronous resources, with the creation of threads (multi-threading) for each request, while maintaining low consumption computational process, quickly processing the requested requests, thus leaving the application not block able for a low time. Vast APIs natively on the platform, with complement of APIs developed by the community. The disadvantages to this platform are: The difficulty for beginners to enter this world, because their implementation is verbose and often strenuous; Medium and / or closed community; Labor market below, most vacancies are for senior developers.

PHP**:** Acting strongly in web development, due to its ease to construction, publication and database integration. Incredible language fora beginner who wants to enter the development world, with a strong friendly community to assist them. With the disadvantage of not executing the CPU level, as it runs through the Zend VM, but the community has already wait for the PHP 8.0 update that will already come with JIT - Just in Time implementation. Another disadvantage is that the community itself does not want to migrate to the latest updates.80% of the internet developed in PHP, is basically 51% using PHP5.5 and some 40% with the current PHP 7.4, with an incompatibility between versions, leaving a low standardization, for the development of the same, as for example, a command that works in one version and doesn't work in another. This version of PHP 5.5 has even been discontinued, without new security updates including, making data browsing dangerous, the creator of PHP itself - RasmusLerdorf asks to stop using that version and use the latest ones.

Conclusion on performance-advantages and disadvantages: In order to act, each platform contemplates its requested need, such as example Node.js, create a scalable application that has many requests simultaneous, it is a good request, creation of APIs with object database NoSQL and or real-time systems like chat and games, Node.js is an excellent requested. Just like PHP is a great demand for those who want to develop web rendering HTML directly from the server, such as blogs, e-commerce. Increasing your SEO (Search Engine Optimization), which translates is approximately like this “Search engine optimization”, improving digital marketing, reaching good rankings on the results pages. .Net Core follows the same demand, creating API, Web UI Applications, easily deployed on servers, the same you can create IoT applications. So, it depends on the need and the workforce of the developer, if you are going to work with relational banks and APIs, or live chat, games, etc. Depending on the demand these platforms supply the need. However, all programming has its limitations, just as PHP is limited to Web Systems orNode.js does not work with heavy CPU or .Net Core processing with difficult deployment to servers other than Windows. To avoid comparing apples with oranges, we have to consider many factors. It may not even be worth the effort. After all, your environment behaves differently from any benchmark. The best way to understand performance is to monitor it with something. That way, you can maintain your Node.js application. or PHP or .Net Core running under circumstances where performance really counts.

Comparing platform performance**:** We developed the same application for the runtime comparison, performance, memory expenditure, synchronicity. This application consumes an APIexternal, which generates data about the characters, planets, races, etc. These data will be shown on the Windows PowerShell console for viewing. Concerning the fictional cinematic universe of George Lucas, the Star Wars saga. Was developed also on the .Net Core platform, an application for monitoring performance in the CPU of the machine used, to monitor the applications developed for this test, that generated the executables, unfortunately PHP does not generate an executable, the same being from outside for CPU monitoring.

Characteristics of the environment: We execute the requests from our Windows machine, which executes the monitoring application, being is running and monitoring CPU resources. We understand that application performance is affected by competition for resources machine and also internet bandwidth, due to requests to external APIs.All applications have been used recent versions of the platforms, the applications are console type, thus facilitating the implementation and publication in the system.

The framework versions used were as follows:

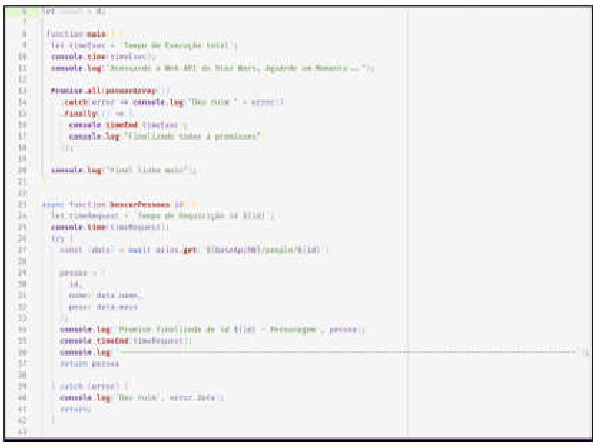
- .Net Core 3.1.0
- Node.js 13.5.0
- Axios ^ 0.19.2
- PHP 7.3.9
- ReactPHP
- Visual Studio Code

The machine resources used are as follows:

- Windows machine;
- Processor: Intel® Core (TM) i7-2600 CPU @ 3.40GHz 3.40 GHz
- Memory: 16.0 GB
- Disk: 500 GB
- Operating System: Windows 10 Pro

Application and Results: In summary, applications were developed to receive external data from the API, created a list of functions to store the requests sent, each one having a timed time and the value of the data returned, observing the oscillation between them, at the end of executing all the requests, it will show how long it took to run the program as a whole.

Development with Node.js



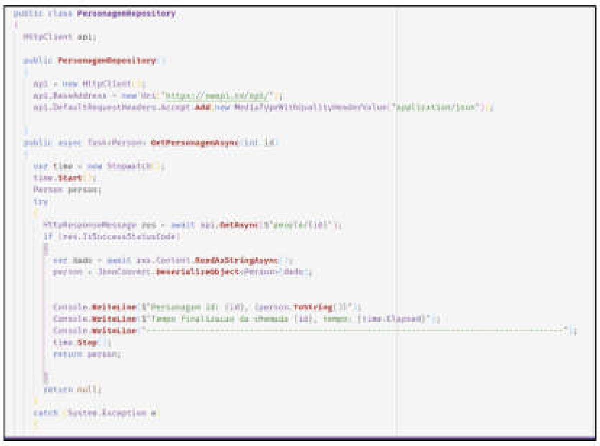
```
6   let count = 0;
7
8   function main() {
9     let timeExec = 'Tempo de Execução total';
10    console.time(timeExec);
11    console.log('Acessando a Web API do Star Wars, Aguarde um Momento ... ');
12
13    Promise.all(povoarArray())
14      .catch(error => console.log('Deu ruim ' + error))
15      .finally(() => {
16        console.timeEnd(timeExec);
17        console.log("Finalizado todas a promisses")
18      });
19
20    console.log("Final linha main");
21  }
22
23  async function buscarPessoas(id) {
24    let timeRequest = `Tempo de Requisição id ${id}`;
25    console.time(timeRequest);
26    try {
27      const {data} = await axios.get(`${baseApi}/people/${id}`);
28
29      pessoa = {
30        id,
31        nome: data.name,
32        peso: data.mass
33      };
34      console.log(`Promise finalizada de id ${id} - Personagem`, pessoa);
35      console.timeEnd(timeRequest);
36      console.log('--------------------------------------------------------------------------');
37      return pessoa;
38
39    } catch (error) {
40      console.log('Deu ruim', error.data);
41      return;
42    }
43
```

Fig. 5: Application with Node.js. Source: Authors, (2020).

Figure 5 represents the source code of the API consumer application in Node.js, composed of 3 functions:

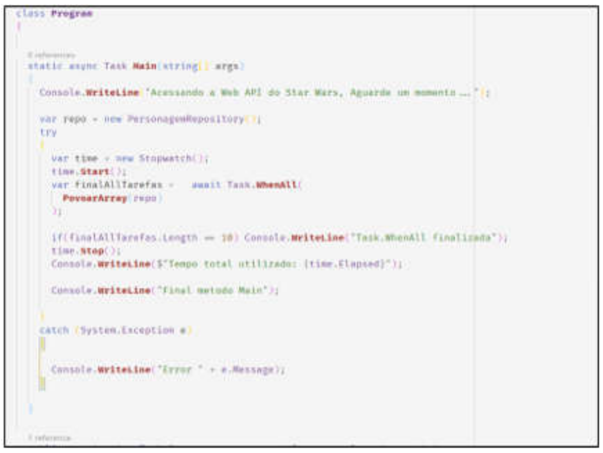
1. to populate Array () - Whose objective is to populate an Array with Promises of requests to the external API, it has an internal repetition structure from 0 to 88 for populate the array and passing the index + 1 as if it were the ID for each request, after finalizing the repetition structure returns this array with promises.
2. seekPeople (id) - Objective is to execute a Promisse, making a request via Axios library for the external Star Wars API, receiving as parameter one id to search for the character with the referring id, in addition to timing the start of the request at the end of it and returns the data about the character.
3. Main () - It is the main function that times your start of execution until the end of the last promise processed, using the native Promises' function to execute the array returned from the populateArray () function that internally calls the function seekPeople in their repetition structure.

Development with .Net Core: The application developed in .Net Core was divided into two connected classes, being CharacterRepository and Program (main class), the class CharacterRepository serves as an entry, to start requests and handle them. While the main class starts and shows the data output. The class in figure 6, starts with the constructor instantiating the native HTTP Client Class, which will do the connection to the external API. There is also a Get Personagem Async method that receives in its parameters the id of the requested character, this function uses the async / await mechanics of .Net Core leaving calls asynchronous. To do this differently from the mechanics of Node.js V8, the calls async / await in .Net Core separates the process into tasks, creating the so-called mult-threading, for each request is a thread within the system, thus consuming computational resources but running with performance and speed. In figure 7 right at the beginning instance the CharacterRepository class, to use this unique method, right after an instance of the platform's native class, called Stopwatch, which times, as soon as started. Soon after another native Task class that calls its function WhenAll which aims to perform several simultaneous tasks, in its parameter we call the function, PovoarArray, whose purpose is to create a task list with repetition control. Used async / await to wait for the completion of all tasks and not close the program directly, without even finishing them. Other lines are the stopwatch and screen exit.



```
public class PersonagemRepository
{
  HttpClient api;

  public PersonagemRepository()
  {
    api = new HttpClient();
    api.BaseAddress = new Uri("https://swapi.co/api/");
    api.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

  }
  public async Task<Person> GetPersonagemAsync(int id)
  {
    var time = new Stopwatch();
    time.Start();
    Person person;
    try
    {
      HttpResponseMessage res = await api.GetAsync($"people/{id}");
      if (res.IsSuccessStatusCode)
      {
        var dado = await res.Content.ReadAsStringAsync();
        person = JsonConvert.DeserializeObject<Person>(dado);


        Console.WriteLine($"Personagem id: {id}, {person.ToString()}");
        Console.WriteLine($"Tempo finalizacao da chamada {id}, tempo: {time.Elapsed}");
        Console.WriteLine("-------------------------------------------------------------");
        time.Stop();
        return person;

      }
      return null;
    }
    catch (System.Exception e)
    {
```

Fig. 6: Character Repository Class (Source: Authors, (2020).

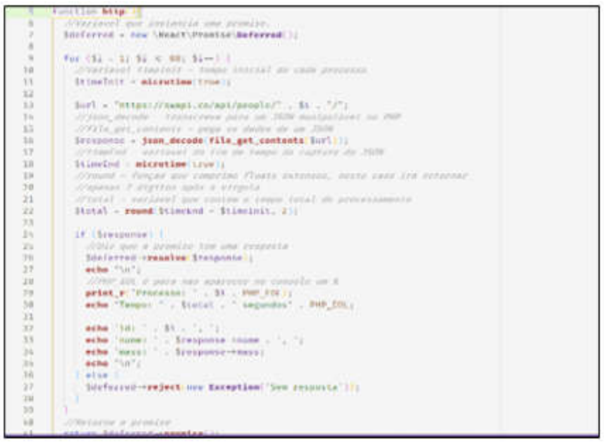


```
class Program
{

  0 referencias
  static async Task Main(string[] args)
  {
    Console.WriteLine("Acessando a Web API do Star Wars, Aguarde um momento ... ");

    var repo = new PersonagemRepository();
    try
    {
      var time = new Stopwatch();
      time.Start();
      var finalAllTarefas =   await Task.WhenAll(
        PovoarArray(repo)
      );

      if(finalAllTarefas.Length == 10) Console.WriteLine("Task.WhenAll finalizada");
      time.Stop();
      Console.WriteLine($"Tempo total utilizado: {time.Elapsed}");

      Console.WriteLine("Final metodo Main");

    }
    catch (System.Exception e)
    {

      Console.WriteLine("Error " + e.Message);

    }
  }

  0 referencia
```

Fig. 7: Main Class. Source: Authors, (2020).

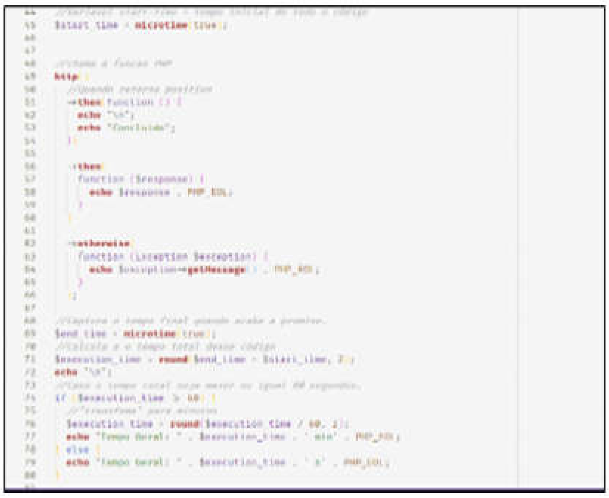
Development with PHP: The application starts according to figure 8, with the http function, whose objective is performing a repetition control to request the external API, timing each calling and showing the generated data. A library was used to compose the called API, library called ReactPHP, whose objective is to leave requests asynchronous, but it does not have a function that performs several requests and the wait. In figure 9, time the beginning of the http function until its end, in the continuation of the script more data output.

Execution time result: The following figure 10 shows the results of each application request in Node.js and .Net Core, in figure 10 the initial request time result for API of Star Wars, it is observed that Node.js started 1 second earlier to return your first request, bringing the returns asynchronously, we can see that the each return does not follow an order, as if it were lined up, but rather asynchronous. Even his initial return was ID - 71, then 54, 41, 11 ... N. You can note that before the first return we have a message "Final main line", as the one shown on the console says, this message is at the end of the main () function in Node.js figure 5 line 20 of the image. "But how? If all requests to the API have not been finalized and is this message at the end of the function?".



```
function http() {
    //Variavel que instancia uma promise.
    $deferred = new \React\Promise\Deferred();

    for ($i = 1; $i < 88; $i++) {
        //variavel $timeInit - tempo inicial de cada processo
        $timeInit = microtime(true);

        $url = "https://swapi.co/api/people/" . $i . "/";
        //json_decode - transforma para um JSON manipulavel no PHP
        //file_get_contents - pega os dados de um JSON
        $response = json_decode(file_get_contents($url));
        //timeEnd - variavel do fim do tempo da captura do JSON
        $timeEnd = microtime(true);
        //round - função que comprime floats extensos, neste caso irá retornar
        //apenas 2 digitos após a virgula
        //total - variavel que contem o tempo total do processamento
        $total = round($timeEnd - $timeInit, 2);

        if ($response) {
            //Diz que a promise tem uma resposta
            $deferred->resolve($response);
            echo "\n";
            //PHP_EOL é para não aparecer no console um %
            print_r("Processo: " . $i . PHP_EOL);
            echo "Tempo: " . $total . " segundos" . PHP_EOL;

            echo 'id: ' . $i . ', ';
            echo 'nome: ' . $response->nome . ', ';
            echo 'mass: ' . $response->mass;
            echo "\n";
        } else {
            $deferred->reject(new Exception('Sem resposta'));
        }
    }
    //Retorna a promise
    return $deferred->promise();
```

Fig. 8: HTTP Function (Source: Authors, (2020).



```
//variavel start_time = tempo inicial de todo o código
$start_time = microtime(true);


//Chama a função PHP
http()
    //Quando retorna positivo
    ->then(function () {
        echo "\n";
        echo "Concluido";
    })

    ->then(
        function ($response) {
            echo $response . PHP_EOL;
        }
    )

    ->otherwise(
        function (\Exception $exception) {
            echo $exception->getMessage() . PHP_EOL;
        }
    );

//Captura o tempo final quando acaba a promise.
$end_time = microtime(true);
//Calcula a o tempo total desse código
$execution_time = round($end_time - $start_time, 2);
echo "\n";
//Caso o tempo total seja maior ou igual 60 segundos.
if ($execution_time >= 60) {
    //"transforma" para minutos
    $execution_time = round($execution_time / 60, 2);
    echo "Tempo Geral: " . $execution_time . " min" . PHP_EOL;
} else {
    echo "Tempo Geral: " . $execution_time . " s" . PHP_EOL;
}
```

Fig. 9: Execution of the function and rest of the script Source: Authors, (2020).

Thanks to its engineering and event methodology, Node.js can continue a function, even if it is not finished, because the Event Loop monitors each function that has a "callback" to the famous Callback, so long as a function is not finished, he continues reading his Call Stack and the pending function stays in the Callback Stack, after the function is finished, he places his callback in CallStack. The .Net Core started 1 second late, as explained in the application code of the .Net, it natively has the functions of async / await, a means similar to that of Node.js, however .Net works with mult-thread, leaving the application asynchronous, as we can see in the result, the data returns came without ordering, ending by time. Even though they started late, their requests ended almost the same instant, causing some lines to return at the same time on the console screen, returning some data outside the context of your ID. This fact happens, due to the application process to split each request as if it were a task, that is, requesting a thread for each call, when two tasks end together at the at the same time, or in a matter of a few thousandths, the data end up together. Analyzing the first exits from .Net there were 23 returns in less than 1 second, while in Node.js the first outputs were 10 returns, however some return oscillations happened in .Net, like, from one request to another it took about 4 seconds and then continued to increasingly follow the timing. At the end of the Node.js line, you ended your requests with thousandths to least, see figure 11.

In comparison in the PHP application, the results according to figure 12, the beginning of calls, happen in ascending order, on average 1.5 seconds out of 88 created calls, however it is blocking, locking the system, waiting for the completion of each, therefore returning in order, leaving the rest of the system idle. At the end of the test, see figure 13, until it ended all the calls it took almost 3 minutes, but observing each request, it took on average to finish 1.5 seconds, which compared to the other apps, is quite a high-speed request completion. However, blocking the rest of the system.

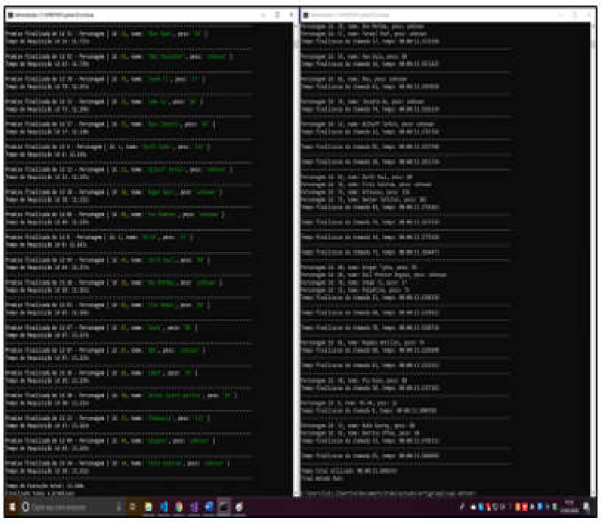


Fig. 10: Result between Node.js and .Net Core. Source: Authors, (2020).

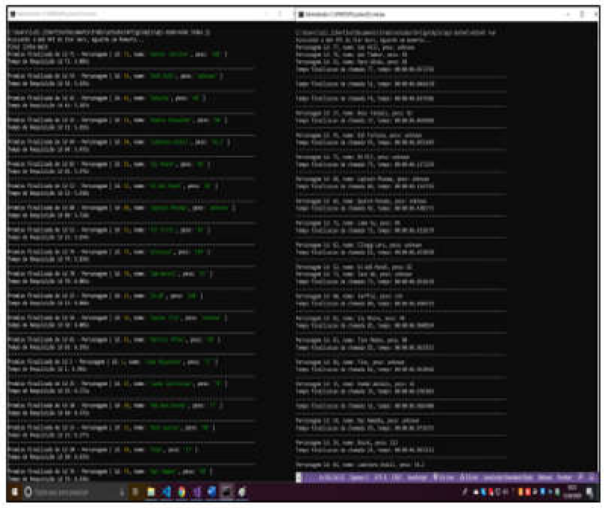


Fig. 11: Final result between Node.js and .Net Core. Source: Authors, (2020).

Monitoring application: The resource monitoring application, from the Node.js and .Net Core applications, was built with the .Net Core platform, it has excellent libraries for processing diagnostics, control, status, etc. The algorithm starts a process, through the executable of each application and monitors each processing, memory consumption and time spent running until its completion. Showing the final result of peak memory spent and sum of memory virtual spend. Remembering that it was not possible to generate an executable in PHP, then the even stayed out of resource monitoring.

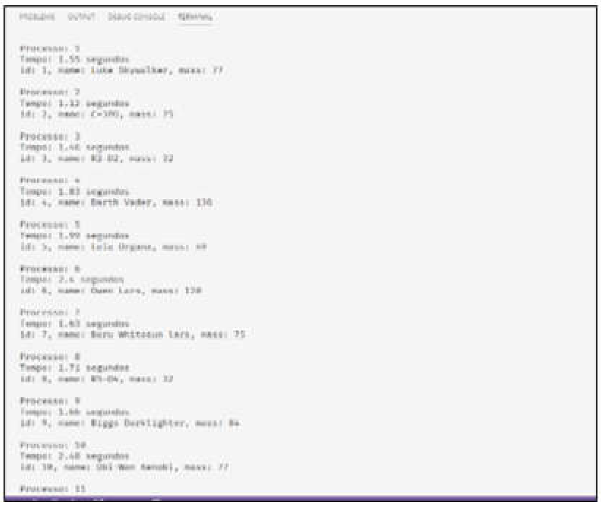
Monitoring result: For monitoring the application in Node.js, see figure 14 on the left, started using a 3-megabyte memory paging, right after that, we observed that it started consuming

42 megabytes of physical memory, with a time process time 4 milliseconds and memory paging at 30 megabytes. Keeping this average consumption. Upon completion, see figure 15 on the left side the data generated are:

- Peak physical memory used - 48.00MB
- Peak memory paging used - 35.00MB
- Peak virtual memory used - 4432.00MB

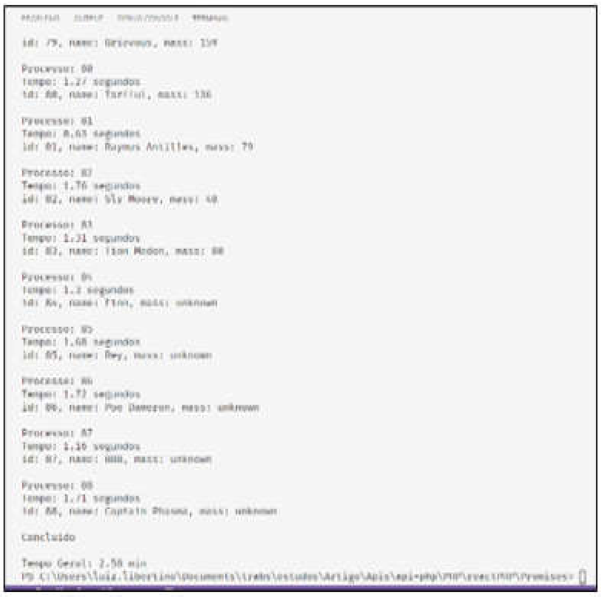
In the monitoring for the .Net Core application, see figure 14 on the right, started with all statuses 0, in the second call we have a consumption of 35 megabytes, using 14 millisecond processor time and paging memory of 21 megabytes. Having an average consumption of: 50.87 MB of physical memory and 34.00MB of memory paging. Upon completion, see figure 15 on the right side the data generated are:

- Peak physical memory used - 60.00MB
- Peak memory paging used - 43.00MB
- Peak virtual memory used - 2102004.00MB



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Processo: 1
Tempo: 1.55 segundos
id: 1, name: Luke Skywalker, mass: 77

Processo: 2
Tempo: 1.13 segundos
id: 2, name: C-3PO, mass: 75

Processo: 3
Tempo: 1.46 segundos
id: 3, name: R2-D2, mass: 32

Processo: 4
Tempo: 1.83 segundos
id: 4, name: Darth Vader, mass: 136

Processo: 5
Tempo: 1.99 segundos
id: 5, name: Leia Organa, mass: 49

Processo: 6
Tempo: 2.4 segundos
id: 6, name: Owen Lars, mass: 120

Processo: 7
Tempo: 1.63 segundos
id: 7, name: Beru Whitesun lars, mass: 75

Processo: 8
Tempo: 1.71 segundos
id: 8, name: R5-D4, mass: 32

Processo: 9
Tempo: 1.66 segundos
id: 9, name: Biggs Darklighter, mass: 84

Processo: 10
Tempo: 2.48 segundos
id: 10, name: Obi-Wan Kenobi, mass: 77

Processo: 11
```

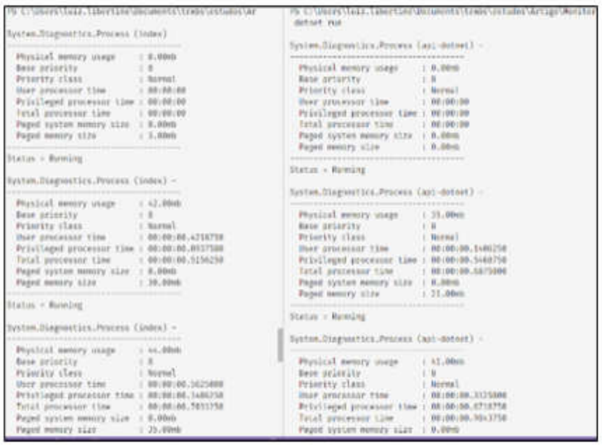
Fig. 12: PHP Result (Source: Authors, (2020).



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

id: 79, name: Grievous, mass: 159

Processo: 80
Tempo: 1.27 segundos
id: 80, name: Tarfful, mass: 136

Processo: 81
Tempo: 0.63 segundos
id: 81, name: Raymus Antilles, mass: 79

Processo: 82
Tempo: 1.76 segundos
id: 82, name: Sly Moore, mass: 48

Processo: 83
Tempo: 1.31 segundos
id: 83, name: Tion Medon, mass: 80

Processo: 84
Tempo: 1.2 segundos
id: 84, name: Finn, mass: unknown

Processo: 85
Tempo: 1.68 segundos
id: 85, name: Rey, mass: unknown

Processo: 86
Tempo: 1.72 segundos
id: 86, name: Poe Dameron, mass: unknown

Processo: 87
Tempo: 1.16 segundos
id: 87, name: BB8, mass: unknown

Processo: 88
Tempo: 1.71 segundos
id: 88, name: Captain Phasma, mass: unknown

Concluido

Tempo Geral: 2.58 min
PS C:\Users\luiz.libertino\Documents\trabs\estudos\Artigo\Apis\api-php\PHP\exec\PHP\Promises>
```

Fig. 13: Final PHP Result (Source: Authors, (2020).

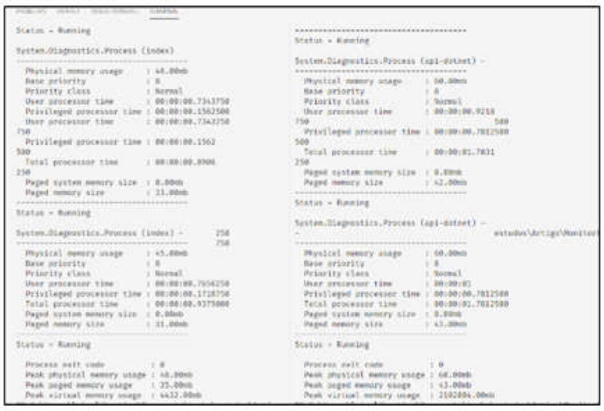
The question "Do you know node.js?", Got 321 answers, we can verify that node.js is not an unknown technology and the vast majority of professionals in the field programmers know it.



```
PS C:\Users\luiz.libertino\Documents\trabs\estudos\Ar

System.Diagnostics.Process (index)
------------------------------------------
 Physical memory usage     : 0.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00
 Privileged processor time : 00:00:00
 Total processor time      : 00:00:00
 Paged system memory size  : 0.00mb
 Paged memory size         : 3.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (index) -
------------------------------------------
 Physical memory usage     : 42.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.4218750
 Privileged processor time : 00:00:00.0937500
 Total processor time      : 00:00:00.5156250
 Paged system memory size  : 0.00mb
 Paged memory size         : 30.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (index) -
------------------------------------------
 Physical memory usage     : 44.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.3625000
 Privileged processor time : 00:00:00.1406250
 Total processor time      : 00:00:00.5031250
 Paged system memory size  : 0.00mb
 Paged memory size         : 35.00mb
```

```
PS C:\Users\luiz.libertino\Documents\trabs\estudos\Artigo\Monitor
 dotnet run

System.Diagnostics.Process (api-dotnet) -
------------------------------------------
 Physical memory usage     : 0.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00
 Privileged processor time : 00:00:00
 Total processor time      : 00:00:00
 Paged system memory size  : 0.00mb
 Paged memory size         : 0.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (api-dotnet) -
------------------------------------------
 Physical memory usage     : 35.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.1406250
 Privileged processor time : 00:00:00.5468750
 Total processor time      : 00:00:00.6875000
 Paged system memory size  : 0.00mb
 Paged memory size         : 21.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (api-dotnet) -
------------------------------------------
 Physical memory usage     : 41.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.3125000
 Privileged processor time : 00:00:00.6718750
 Total processor time      : 00:00:00.9843750
 Paged system memory size  : 0.00mb
```

Fig. 14: Result of the initial monitoring. Source: Authors, (2020).



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Status - Running

System.Diagnostics.Process (index)
------------------------------------------
 Physical memory usage     : 48.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.7343750
 Privileged processor time : 00:00:00.1562500
 User processor time       : 00:00:00.7343750
750
 Privileged processor time : 00:00:00.1562
500
 Total processor time      : 00:00:00.8906
250
 Paged system memory size  : 0.00mb
 Paged memory size         : 31.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (index) -       250
------------------------------------------ 750
 Physical memory usage     : 45.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.7656250
 Privileged processor time : 00:00:00.1718750
 Total processor time      : 00:00:00.9375000
 Paged system memory size  : 0.00mb
 Paged memory size         : 31.00mb
------------------------------------------
Status - Running

 Process exit code           : 0
 Peak physical memory usage  : 48.00mb
 Peak paged memory usage     : 35.00mb
 Peak virtual memory usage   : 4432.00mb
```

```
------------------------------------------
Status - Running

System.Diagnostics.Process (api-dotnet) -
------------------------------------------
 Physical memory usage     : 50.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:00.9218
750                                       500
 Privileged processor time : 00:00:00.7812500
500
 Total processor time      : 00:00:01.7031
250
 Paged system memory size  : 0.00mb
 Paged memory size         : 42.00mb
------------------------------------------
Status - Running

System.Diagnostics.Process (api-dotnet) -
-                                  estudos\Artigo\Monitor
------------------------------------------
 Physical memory usage     : 60.00mb
 Base priority             : 8
 Priority class            : Normal
 User processor time       : 00:00:01
 Privileged processor time : 00:00:00.7812500
 Total processor time      : 00:00:01.7812500
 Paged system memory size  : 0.00mb
 Paged memory size         : 43.00mb
------------------------------------------
Status - Running

 Process exit code           : 0
 Peak physical memory usage  : 60.00mb
 Peak paged memory usage     : 43.00mb
 Peak virtual memory usage   : 2102004.00mb
```

Fig. 15: Result of final monitoring. Source: Authors, (2020).

4.4 Search on node.js

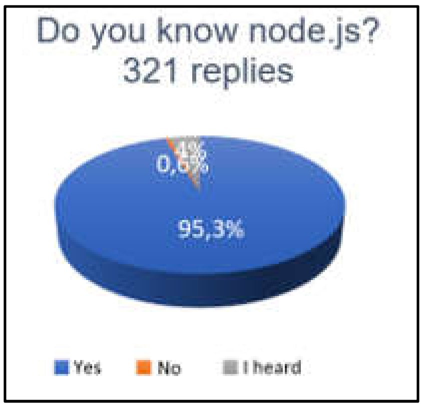


Fig. 16: Question graph 1. Source: Authors, (2020).

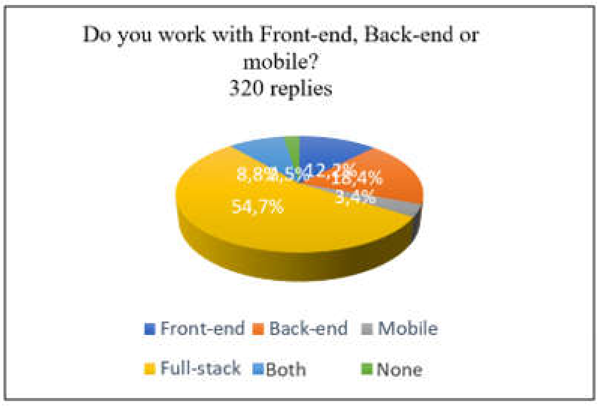


Fig. 17: Question graph 2. Source: Authors, (2020).

Question "Do you work with Front-end, Back-end or mobile", got 320 responses, with this question we can verify which functions the programmers are specialized, we found that more than 50% of programmers work as a full-stack, if Javascript, programmers can be full-stack with just a single language, whose Node.js works on the server side, the client side works a lot with current Javascript frameworks, in addition to the possibility to program for mobile devices.

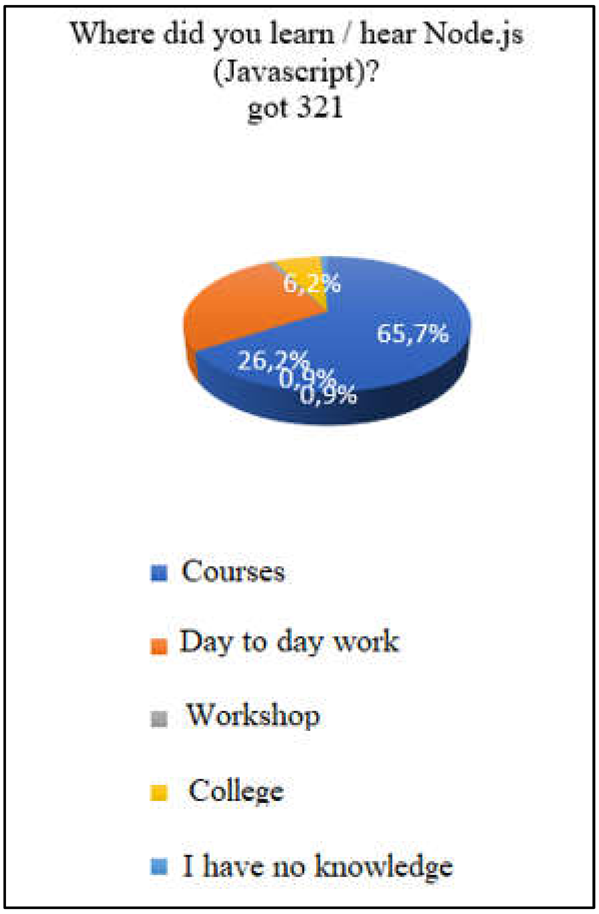


Fig. 18: Question graph 3. Source: Authors, (2020).

The question "Where did you learn / hear Node.js (Javascript)?", We note that the Most programmers who want to learn about the technology have taken courses and 26% of them obtain knowledge at work.

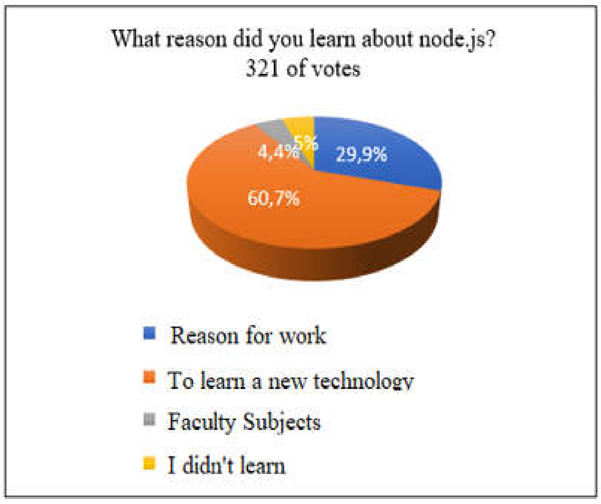


Fig. 19: Question graph 4. Source: Authors, (2020).

The question "What reason did you learn about node.js?", Obtained 321 of the votes, that most professionals obtained the knowledge for the reason of learning a new technology, and only 29% of them were for work reasons.

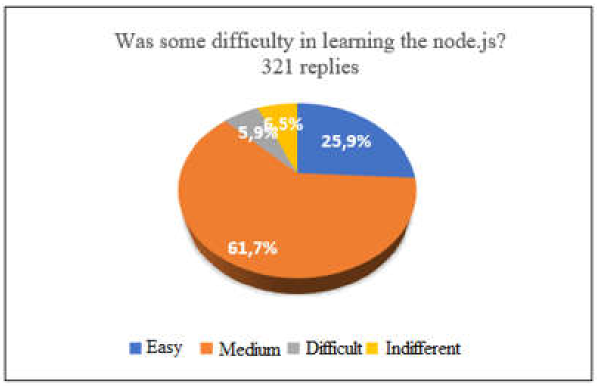


Fig. 20: Question graph 5. Source: Authors, (2020).

The question "Was there any difficulty in learning node.js?", We obtained 321 answers, we can verify that for the learning of this technology, the medium difficulty, having to know other technologies like Javascript, paradigm functional, etc. 25% reported that learning was easy, a fact due to reuse of language, which is widely used on the client side.

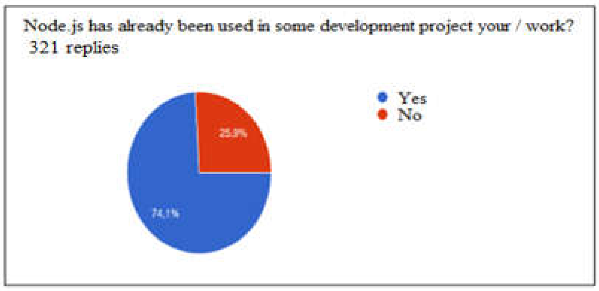


Fig. 21: Question graph 6. Source: Authors, (2020).

The question "Node.js has already been used in some development project your / work?", we get 321 responses, we can see that most have already used the node.js in some project, being small to the big company. Only 25% used.

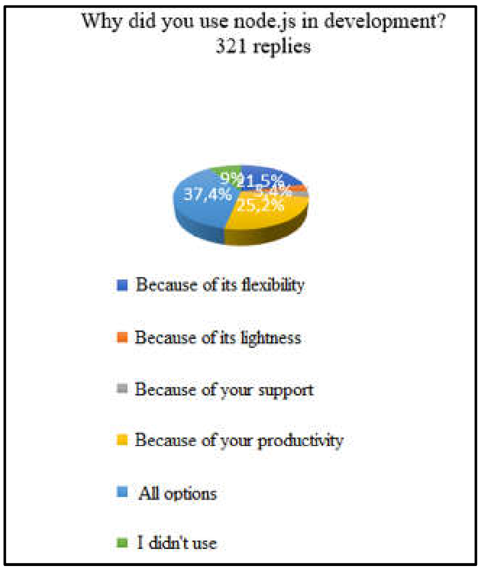


Fig. 22: Question graph 7. Source: Authors, (2020).

The question “Why did you use node.js in development?”, Obtained 321 responses, it was observed that 37% selected all options, and more than 25% per story technology productivity.

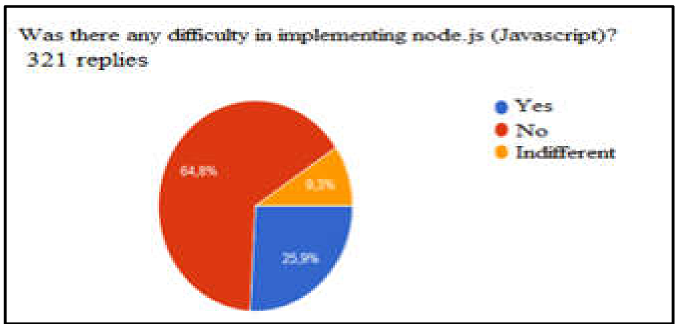


Fig. 23: Question graph 8. Source: Authors, (2020).

The question “Was there any difficulty in implementing node.js (Javascript)?”, obtained 321 responses, we found that the majority did not have any difficulty in implementation, 25% already had some difficulty.

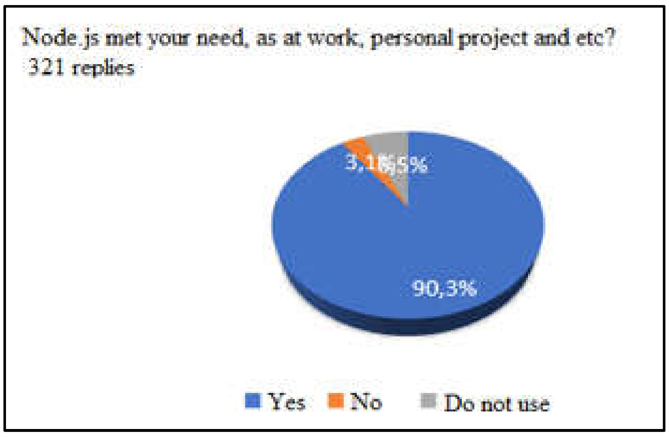


Fig. 24: Question graph 9. Source: Authors, (2020).

The question “Node.js met your need, as at work, personal project and etc? ”, got 321 responses, we can see that node.js meets the need proposal to the programmer, a small part claims that it did not meet the need.

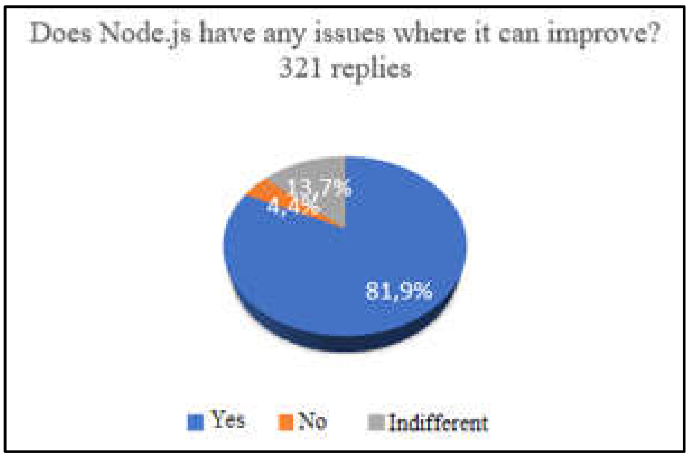


Fig. 25: Question graph 10. Source: Authors, (2020).

The question “Does Node.js have any issues where it can improve?”, Obtained 321 answers, we can watch node.js has many issues to improve, few do not know, and small.

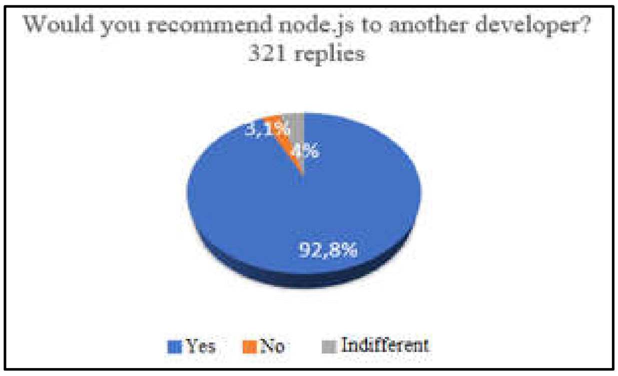


Fig. 26: Question graph 11. Source: Authors, (2020).

The question “Would you recommend node.js to another developer?”, Obtained 321 answers, we note that most programmers recommend node.js for other programmers, a small part says no.

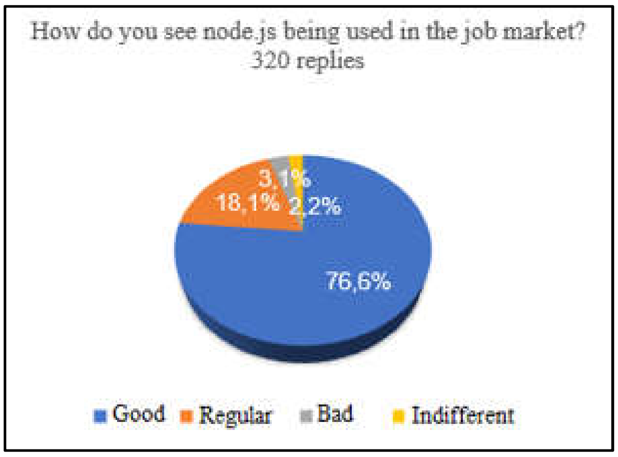


Fig. 27: Question graph 12. Source: Authors, (2020).

The question “How do you see node.js being used in the job market?”, Had 320 answers, we can see that in the job market node.js has a good result, a small part sees it as regular, and a very small part as bad and indifferent.

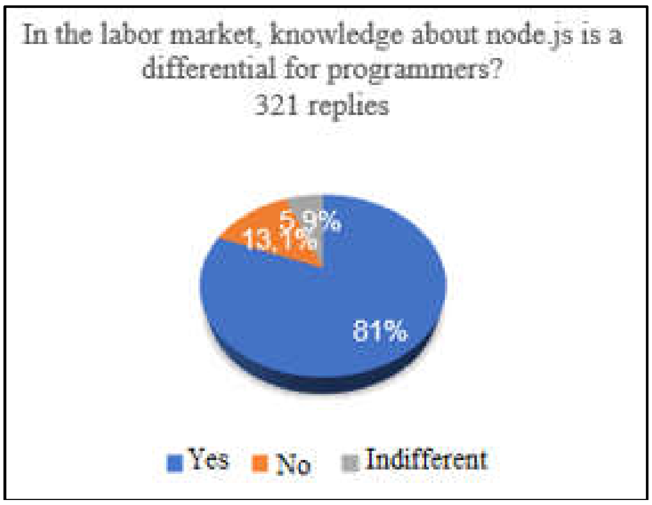


Fig. 28: Question graph 13. Source: Authors, (2020).

The question “In the labor market, knowledge about node.js is a differential for programmers? ”, had 321 responses, we can verify that having knowledge about node.js is a differentiator for programmers, and a small part says no.

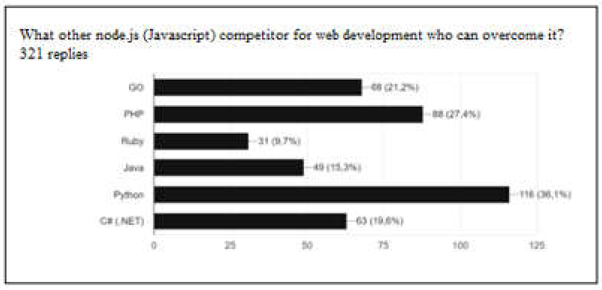


Fig. 29: Question graph 14. Source: Authors, (2020).

The question “What other node.js (Javascript) competitor for web development who can overcome it?”, got 321 responses, we can verify that the competitor who can overcome, according to respondents is Python, followed by PHP.

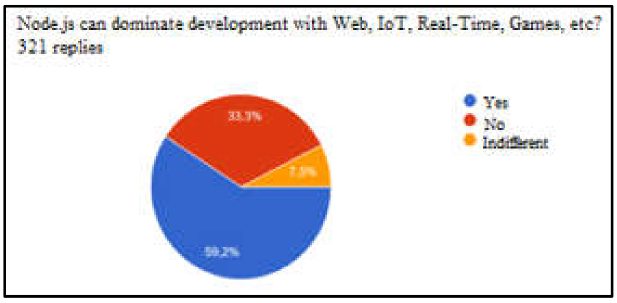


Fig. 30: Question graph 15. Source: Authors, (2020).

The question “Node.js can dominate development with Web, IoT, Real-Time, Games, etc?”, Got 321 responses, we can see that half of the programmers think that node.js can dominate, more than 33% think not and more than 7% indifferent.

CONCLUSION

The platform is very flexible, making it easy to implement standards, object-oriented and function-oriented programming. Easy to implement tests, little code writing, for great functionality. With the advantage of integrate more teams that work with the server side and the client side, because using a single language for the development of the system, now also with development of mobile device applications, as there are libraries that use Javascript for this type of development, as the mobile device interprets Javascript natively. However, not everything is flowers, with the growth of the system developed inNode.js platform, increases the difficulty of understanding, due to its engineering such as: Event Loop operation, Callback handling, and asynchronism, weak typing. Because of this, developers have no confidence for a complete development and recommends it to other developers.

It was collected that the vast majority of those surveyed, know or use Node.js to development, some think Node.js has a lot to improve as a platform of development. A large number would recommend it, to others programmers, because its ease for those who already use Javascript, for client-side development with dynamic sites, greatly facilitates migration of these types of developers. The vast majority of respondents consider themselves to be full-stack developers, justifying the large number of people who know or use Node.js, because who knows the Javascript language, can use it so much in the development for the server side, as well as the client side and now with development for mobile devices, with Javascript frameworks. It was observed, for those surveyed that the major competitors for the platform, are: 1st place Python, becoming very popular due to science programming data, artificial intelligence, automations and big data. It is a major competitor of platform; in 2nd place comes PHP, due to its popularity for building-commerce sites and web development in general, with a major contribution to the web world. In terms of who can dominate the web, the vast majority of respondents believe that the platform can dominate development in general, due to JavaScript, that can be interpreted on any device.

In monitoring we noticed that Node.js compared to other platforms, has a performance for fast input and output, in addition to remaining asynchronous, making requests according to the Event Loop call stack, but without wait for the calls to end, placing them on hold in the Callback stack, completed will be on hold on the call stack. In computational resources, Node.js used little, compared to .Net Core it used resources a little more, but significant if compared to larger applications, maintaining execution speed compared to Node.js We conclude that Node.js has a lot to evolve and that it is going down the road sure to be a very popular development platform. With a great community that tends to grow more and more and strengthen the development of platform with feedbacks and suggestions. Knowledge of the programming language JavaScript, helps the migration of frontend developers, to become fullstack,which means a developer who works so much with the business rule, the logic of programming, as for the interface for the end customer, in addition to interfaces for devices providing advantages for this type of professional in the job market, since the integration for teams that know this language is natural.

Acknowledgements

To God for having given health and strength to overcome diversity and difficulties. This university, its faculty, direction and administration that provided the window that today I see a superior horizon, to our supervisor Prof. Bruno Pereira Gonçalves, for the support, for his corrections and incentives. To members of our family, for the love, encouragement and unconditional support, and to all who directly or indirectly they were part of my training, thank you very much.

REFERENCES

[1] Delfino, P., Node.js: entenda o que é e como funciona essa tecnologia, e-TiNet. Consultado em fevereiro 19, 2020 em: https://e-tinet.com/linux/node-js/.
[2] Duarte, L. Node.js para iniciantes (pp. 6-9). Umbler.
[3] Introductionto Node.js, Node.js. Disponível em: https://nodejs.dev/introduction-to-nodejsf.
[4] Lenon, Node.js – O que é, como funciona e quais as vantagens, Opus Software. Consultado em fevereiro 20, 2020 em: https://www.opus-software.com. br/node-js/.
[5] Maccune, R. R., Node.js paradigms and benchmarks. UniversityofNotreDame. C. Disponível em: https://pdfs.semanticscholar.org/301b/45bb8e795f83774c920b942c0dba7e290b53.pdf.
[6] Moares, W. B. (2015). Construindo aplicações com nodejs. São Paulo: Novatec.
[7] Pereira, C. R. (2014). Node.js: Aplicações web real-time com Node.js. São Paulo: Casa do código.
[8] Pereira, C. R. (2016). Construindo APIs Rest com Node.js (pp.1-11). São Paulo: Casa do código.

[9] Powers, S. (2017). Aprendendo node: usando javascript no servidor. São Paulo: Novatec.

[10] Rubens, J. (2017). Primeiros passos com Node.js. São Paulo: Casa do código.

[11] Stefano, B. Como melhorar a performance de aplicações Node.js utilizando o módulo cluster, InfoQ. Consultado em fevereiro 19, 2020 em: https://www.infoq.com/br/articles/nodejs-utilizando-modulo-de-cluster/.

[12] Hota, A. K. Prabhu, D. M. (2014) Node.js: Lightweiht, Event driven I/O web development. Informatics.nic.in. Disponível em: https://informatics.nic.in/uploads/pdfs/26b47a73_node.js.pdf. Acessadoemfevereiro 20, 2020.

****