Nimbus: A Modern Solution For Server Management

Jyoti Mishra[a], Ashwini Kumar[b] and Devanshu Saroha [c]

[a]Department CSE, Galgotias University, Greater Noida, India, jyotimishramau@gmail.com
[b]Department CSE, Galgotias University, Greater Noida, India, ashwini34566@gmail.com
[c]Department CSE-CNCS, Galgotias University, Greater Noida, India, d.saroha1636@gmail.com

Article History: Received: 10 November 2020; Revised 12 January 2021 Accepted: 27 January 2021; Published online: 5 April 2021

Abstract: As NodeJS have taken a face in order to serve the client and server by using concept of API and REST API after the elixir and phoenix framework. As there is a large gap between these two services that are holding back the servers. So, we have come up with an solution to cover that gap by using some addons. As we all know that node have limitations like It can't handle multi-threaded tasks & also doesn't have support to handle very high computationally intensive task over servers. That gap has been filled by elixir but there are less developers and that framework need some more experience in that same framework. Our solution is simple we are using Rust that is newly introduced language with powerful concepts and can be used to fill the gap by using it as an addon with NodeJS. On top of that server, we are building an android application as well to show a proper server client model. Application Description: The era of mobile technology opens the windows to the android app and websites are vanishing and the mobile phones are emerging. It's the time to change from conventional websites to app, which has become the part of our daily routine. We are introducing an android application software which would help in time consuming and other things. It is type of app in which if any person is outside and want to get anything which will be available outside, with the help of this app the person will be able to get to know by seeing what I want to purchase from outside with the help of request-based features of this application.

Keywords:

1. Introduction

Sometimes a situation arises where an application that requires speed and productivity of developers has been incredibly important, but is the business needs changing rapidly? We also found ourselves in this situation where I was given the task of finding a solution and building a transporter-proof-concept transpiler for our team's content files. So we did some research and found a stack of technologies that would enable us to deliver all the components and, at the same time, be well integrated with our existing servers deployed via NodeJS. The following research paper decides to use Rust and Web Assembly or NodeJS servers and discusses why the combination of these two technologies is so powerful. This will be helpful as it covers maximum cons of NodeJS which makes more powerful and efficient to deploy and serve the client with better speed and management of users if load on server slightly increases and decreases in a short period of time. This can be used to manage these types of problems with little investment and more flexibility and control over the running server.

NodeJS is the most popular JavaScript application time for background applications. Its flexibility and flexibility have made it a premier choice for API implementation. As it is written language, JavaScript can be slow. But thanks to V8's performance, it's fast enough for running apps. That said, Node.js is not ready for a difficult promotion; as it has a single cord, it is dangerous to block a large cord for making long calculations. This is where the threads of the employee come in. Node.js has support for employee threads, so it can be used to perform longer calculations. No matter how big the staff threads are, JavaScript is still slow. In addition, an employee cable is not available for all Lode supported Node types. Fortunately, we can use Rust to create traditional Node.js inputs. FFI is another option, but slower than the add-on method. Rust burns quickly and there is a fearless financial deal. Since Rust has a very short operating time (or "no operating time"), our binary size should also be very small.

Rust:

Rust is the programming language of Mozilla. It can call the C library automatically and add support for the first phase of sending tasks to C. Rust gives you low level control and high ergonomics. Gives you control of memory management without the hassle associated with these controls. It also delivers zero cost discounts, so you only pay for what you use.

Rust can be called in the context of Node.js in a variety of ways. I have listed some of the most commonly used items below.

- You can use FFI from Node.js and Rust, but this is slow
- You can use Web Assembly to create a node module, but all Node.js functionality is not available

• You can use traditional adjectives

Native addon:

Node.js add-ons are shared items listed in C ++ powerfully connected. You can download them from Node.js using the require () function and use them as standard modules for Node. They basically provide a interface between JavaScript that works on Node.js libraries and C / C ++. The traditional addon provides a visual interface to work with another binary by uploading it to the V8 operating system. It is very fast and secure to make calls in all languages. Currently, Node.js supports two types of addon modes: C ++ addons and N-API C ++ / C addons.

C ++ addons:

The C ++ addon is something that is not charged by Node.js and is used during operation. Since C ++ is an integrated language, these add-ons are very fast. C ++ has a wide range of ready-to-use libraries that can be used to expand the Node environment. Many popular libraries use traditional add-ons to improve code performance and quality.

Additions of N-API C++/C:

The biggest problem with C ++ addons is that you need to integrate them with all the changes in the JavaScript performance time. It causes a problem with saving the addon. The N-API attempts to eliminate this by introducing a standard operating system (ABI) system. Header C is always backward. That means you can use an addon that is embedded in a particular type of Node.js with any version that is larger than the version it was embedded in. You can use this method to use your addon.

2. Litreature Review

First, let me explain more about the business needs I mentioned earlier. Our team is currently in the process of rewriting and redesigning our entire domain-based language program (DSL) with all of our content. We have thousands of these files, in total more than a gigabyte of content, and sometimes, we have multiple templates contained in the same file.

To rewrite successfully, we need to automate how we move content to the new system. We should also consider that hundreds of our content writers are trained on how to use this DSL, so any changes will require them to adapt to the new system. With those requirements, we had to decide between two options:

• A command line tool that we could use simultaneously to mass transfer our existing content. Content writers will learn a new program.

• A command line tool that we could use as a mass migration OR as a starting point for locally converted files for further construction so that authors can continue to use their existing system.

Since our team needed high flexibility and did not think it was possible to re-name all of our content writers, we decided to take a second option and create a tool that we could use once or as an additional build step.

Finally Rust come in:

Rust can mimic the functioning of a C library. In other words, it sends the function in a C format that you can understand and use. Corruption calls for a C function to access and use the APIs provided by Node.js. These APIs provide ways to build JavaScript cables, layouts, numbers, error, objects, tasks, and more. But we need to tell Rust whether these are outdoor activities, stakes, directions, etc. What it looks like. Rust puts down the channels in memory separately, so we need to tell it to use the C style it uses. It can be painful to create these tasks manually, so we will use a case called NodeJS-sys, which uses a bindgen to create a better N-API definition. bindgen automatically generates Rust FFI bonds in libraries C and C ++.

Powerful VS Solid Language:

While the comparison between consistent and dynamic typing is clear (types are known at the time of integration or not), strong and weak typing is even more boring as those types are similar to the scale at which languages can fall. That is because those words measure the extent to which a language allows for uninterrupted conversions between one type and another. If the language is solidly typed, then it allows fewer instantaneous conversions than a multilingual translation. Given that definition, Rust reaches a very high level on a solid scale as there is only one state of complete transformation, and this transformation can only be achieved under certain conditions. JavaScript, on the other hand, rises above the weak point as there are full conversions everywhere. In fact, whether you know it or not, every variation you offer has some sort of obvious transformation. That's because all the variations you announce are actually additional features given the JavaScript performance time.

Performance considerations:

Some of our initial performance considerations were related to speed: If we expect to use this every time a person makes his or her changes, the transfer process should be faster and happen in seconds. In some of our cases, we have considered the disclosure of a tool such as an API that converts content and provides a backlash to consumers. This solution increases the efficiency considerations, as some of these files can be more than a megabyte itself. In this case, the file would be sent, split, re-formatted, compressed, and sent to the client in a few seconds. Therefore, speed was much considered. The second major consideration was the ability to use the vehicle on the web. Many languages can process these files quickly on a local machine, but very few reach the same native speed as they are distributed on the web. Finally, we needed a language that we could easily and effectively associate with our Node-based environment. Our team would like to introduce the Node.js ecosystem, and the newly redesigned app will be a Vue-based client with an Express-based backend created as a REST API. Node tool, open source community, and the ability to write our client and server in the same language (for us, Typescript) is a great blessing in our team's agile delivery system. We knew we had to keep these amazing benefits in our operating systems, but we also knew that in order to get the performance we needed, we had to look elsewhere. Therefore, the excellent integration with our main Node applications has been a perfect requirement. Once we understand the needs of our business, then it is time to record evidence of certain ideas.

Performance:

• Rust struggles to compete with the likes of C and C ++ by working with features such as low-cost extraction which means you pay for what you use and not a lot of working time even if you use all the features.

• Binary size and memory is also compared to C and C ++ which is more noticeable and opens the way to use embedded cases, IoT, and other memory limitations.

• Application performance is not the only increase in performance available. Developer production is great with features taken from high-quality languages such as high-quality pattern matching, easy to use syntax, and excellent messaging error integration.

Security:

• Rust has many ways to help your app keep it in mind - and secure it as a memory aid aided. While memory management has never been easier in terms of developer production, compared to working with C and C ++ it is much easier for developers to keep up with it, and it is unlikely to happen unsafe without using an unsafe keyword. The unsafe keyword in Rust basically allows you to perform tasks that the compiler cannot guarantee safe to perform. Other examples include: removing the green index indicator, calling an unsafe job or method, or working with code from the C interface. This makes it easy to embed security in code updates and find out where the related problem is coming from. I would say the difficulty is between Go and C ++ in terms of dev’s effort in memory management, but it becomes much easier if you do it for a long time.

• Features such as a variety of options that eliminate null pointer variants (unless, again, using an unsafe keyword) also significantly reduce the number of errors that can be detected by the compiler and lead to crashes or worse, unspecified performance during operation.

• The compiler holds your hand when dealing with errors you find. This allows you to focus on your business objectives rather than detecting interruptions or clarifying confidential messages.

Tools

• Supports integration of all major operating systems.

• Package manager, Cargo, acting as npm from Node.js or pip from Python. This makes it very easy to integrate packages, either locally or remotely from your app or library.

• It is very easy to manage and install many types of Rust using rust up.

• In addition to the great tools around the language itself, it also has a very easy way to work with other languages using its set of Foreign Function Interface (FFI) tools.

The Community:

• Rust has a functionally valid Discord channel that is always ready and willing to help! I was often helped by various people as I worked on a delivery project.

• Excellent reading materials and books created and kept by the community greatly reduce the ch allenges of riding for new language engineers.

• Rust gains the support of many partners and large companies by donating money and code.

As we have searched over the internet about the limitations of the NodeJS Server, we have found the solution to the one of its limitation that NodeJS Server won't be able to process the large API Request on Single thread as a sync process are vulnerable to the data crash. So to overcome this problem we are going to use newly introduced language Rust which will work alongside with NodeJS server as an attached API server or Addon. which will increase the processing power of NodeJS CPU intensive with high computational tasks. A Server along with an addon and android app will be built over that server and deployed.

Existing System

- We have noticed that to increase the load of server every organisation uses load balancer as that is an easy solution but pays a cost in order to serve and increase the load of server.
- Instead of that they can use our solution to serve over a little if they have monitored a very low network increase. An asynchronous process to increase that much load so they can serve the client in less server power instead of increasing and paying for that extra load balancer. That's why we have chosen this project in which we are going to use Rust as an addon with NodeJS server to perform CPU intensive task and process for service between client and server.

Proposed System:

- Technically this project will help us to reduce the load on the server. As an example a company has server limit to serve 1 lakh user if one day the limit increases to 15000+ so to overcome this problem the company will definitely going to install a load balancer of a specific limit such as 20000, 50000, 1 lakh depending on the network traffic between client and server but the free space which is left over in the load balancer will be considered as a waste as low user traffic is generated between user client for a little amount of time.
- So to overcome that problem without an extension of hardware and paying the cost. They can use our solution to serve better to handle that little increase of network without waiting to install and configuration and putting servers on hold that will also become factor loss in business money. Our solution is cost effective as we increasing the limit using software without any extension of hardware to handle little limit.

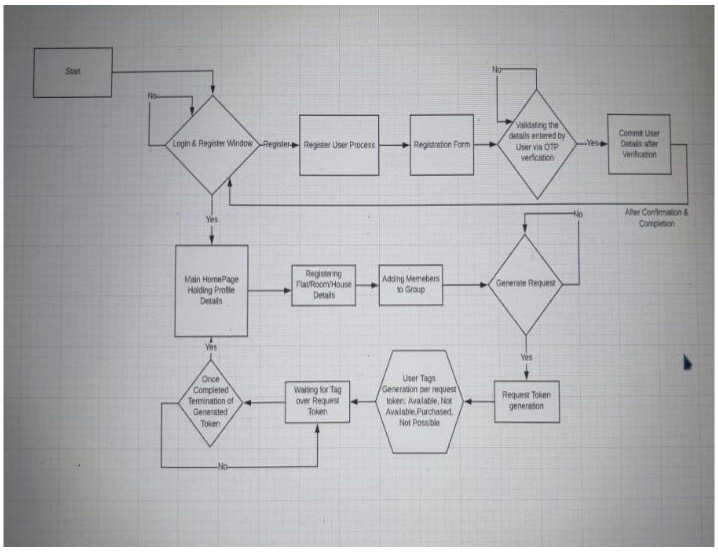
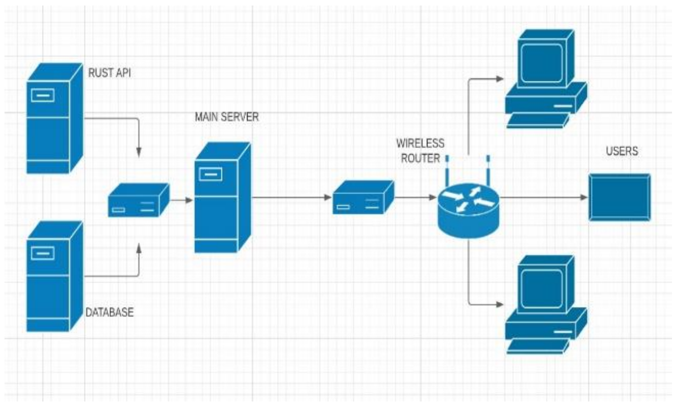

Start
No
Login & Register Window
Register
Register User Process
Registration Form
No
Validating the details entered by User via OTP verfication
Yes
Commit User Details after Verification
After Confirmation & Completion
Yes
Main HomePage Holding Profile Details
Registering Flat/Room/House Details
Adding Memebers to Group
No
Generate Request
Yes
Request Token generation
User Tags Generation per request token: Available, Not Available,Purchased, Not Possible
Waiting for Tag over Request Token
Once Completed Termination of Generated Token
Yes
No

Fig1. Working of our server.



3. Conclusion

When it comes to what you can do with the N-API, this is just a frozen point. We went through a few patterns and put together the basics, like extracting tasks, creating the most widely used JavaScript types like strings, numbers, layouts, objects, etc. Find the context of a task (e.g. this at work), etc. We also explored an in-depth example of how to use libuv threads and perform async work to perform heavy back-to-back calculations. Finally, we built and implemented the JavaScript commitment and learned to make mistakes in N-APIs. There are many libraries available if you do not want to write all the code by hand. This offers a good output, but the disadvantage is that it does not support all features. While there are many comparisons that can be made, I hope you now better understand how the features of Rust's basic language relate to JavaScript! I also hope you are not ashamed to try Rust. While it takes some familiarity, the reward is worth it, and you may find that your JavaScript code gets better as a result..

References

1. https://blog.logrocket.com/rust-and-node-js-a-match-made-in-heaven/
2. https://developer.ibm.com/articles/rust-for-NodeJS-developers/
3. https://developer.ibm.com/technologies/web-development/articles/why-webassembly-and-rust-together-improve-NodeJS-performance/
4. https://developer.ibm.com/articles/os-using-rust/
5. https://www.dotnettricks.com/learn/NodeJS/advantages-and-limitations-of-NodeJS
6. https://www.toptal.com/NodeJS/why-the-hell-would-i-use-node-js