**Why do we need an API?**

As a producer – When we need our resource to be used by a third party application or developer e.g. provide current weather data of all areas in India. Without risking direct access to critical data or code which can be abused e.g. an application tried to update data from a different unreliable source into our Database which in turn affects all the other users of our application. To avoid this we provide them with an API which will still allow them to use our functionality or data without any access over it i.e., as a provider of an API you can dictate what the consumer can and cannot do with the API

As a Consumer – When we are in need of certain data e.g. details of the weather condition of a particular area in a city. In order to get this data, it’s easier to use an API to gain the required data which will be easier than setting up our own equipment and maintain which will be tedious and won’t be cost-effective. And also any future upgrades or optimization done to the API will automatically be applied without any change from our end. It also saves time taken to write and maintain the part of code so it can be used in our Application to optimize other parts and functionality

**What is API?**

We make use of an API to communicate between different applications and share services. Making use of an API reduces the developer’s efforts as we need not code all the features from the ground up. We can make use of the code from another application with the help of an API to access the features provided. In the same way, we need not exposed our code and data to a third-party application instead we provide them with an API which can be used to access our data or feature the third party application cannot make any changes to our codebase which in turn makes our application more secure at the same time we can provide public access without any vulnerability to our application. API can be used to achieve any one or more of the following from another application.

•            Access Data

•            Reduce complexity

•            Security

•            Extend functionality

API specification or Protocol is needed to standardize the exchange of data i.e. irrespective of different programming languages, operating systems, technology, or diverse system, there should be data exchange and communication seamlessly.

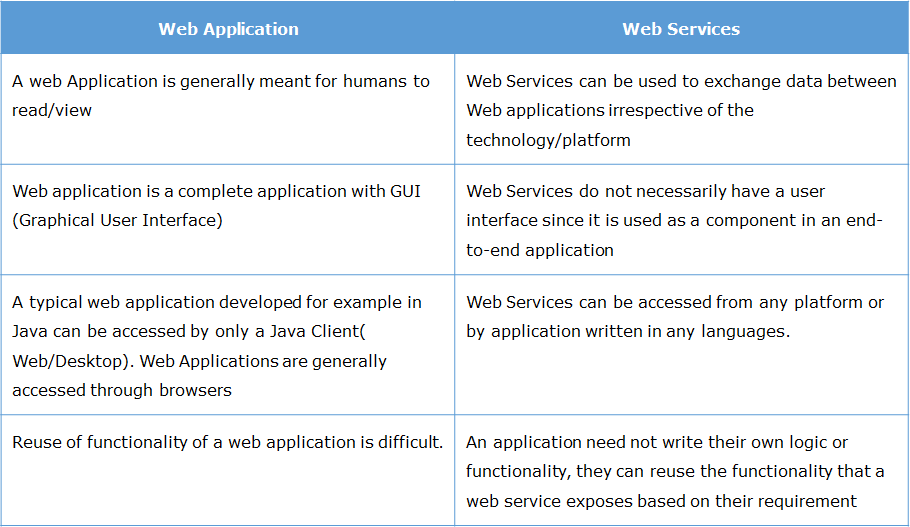
Let us look at the major types of protocols for APIs:

* **Remote Process Call**: It a protocol in which one program can use to request a service from a program located in another computer on a network without having to understand the network's details.
  + XML-RPC: It uses XML format to transfer/encode the data
  + JSON-RPC: It uses JSON format to transfer/encode the data
* **Simple Object Access Protocol (SOAP)**: SOAP developed by Microsoft uses the strict rules to be followed to transfer data. It uses XML for serving its purpose and ensure high security of transferred data. built-in ACID (Atomicity, Consistency, Isolation, Durability) compliance and retry logic for reliable messaging functionality which makes it more suitable among providers of payment gateways, identity management, and CRM solutions, as well as financial and telecommunication services. But being providing such feature SOAP became a heavy and very intensive resource, thus requiring more bandwidth.
* **Representational State Transfer (REST)**: It is an architectural style with six guiding principles (Stateless, Client-Server Architecture, Caching, Uniform interface, layered system, code on demand) for web API that works on HTTP. In REST every information is a resource and this resource can be represented in any format (JSON, XML, Text).
* **GraphQL**: It is the query language for API. It allows the client to send the exact data it needs and simplifies data fetching and aggregation. With mobile adoption and the need for faster feature development GraphQL comes to help in efficient data loading, thus being the future of API.

In this course we are going to learn more about RESTful API.

According to [W3C](https://en.wikipedia.org/wiki/W3C), web service is defined as:

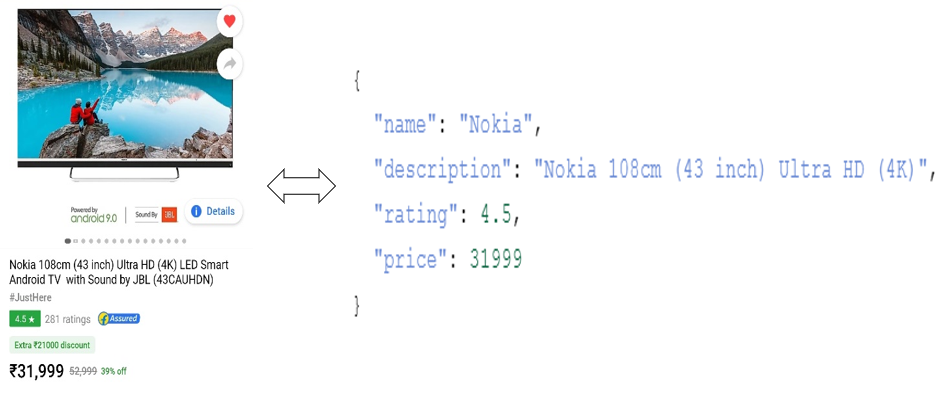
      "A software system designed to support interoperable machine-to-machine interaction over a network."

* To be more precise, server and client applications that communicate over the World Wide Web (WWW) using HTTP are called Web services.
* As described by the World Wide Web Consortium (W3C), web services enable communication between various applications running on different frameworks and platforms.
* The below table provides a comparison between a Web application and a Web service.
* 
* In this course, we will be creating web-services based on REST protocols. So let's understand what is REST in the next resource.

**Need for REST**

The existence of REST came into shape to restrict the ways that the server can process and respond to client requests by abiding certain set of principle (Which are the too guiding principle for REST as coined by Roy Fielding in 2000), so that the system gains desirable performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

**Guiding Principle for REST:**

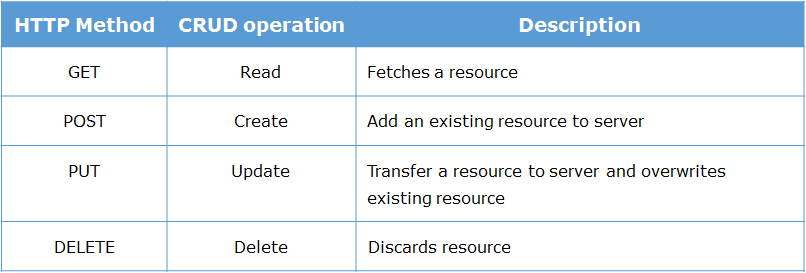
* **Client-server** – This states that we have a separate client and server. Client-Server architecture allows us to upgrade the server and client independently without altering the interface.
* **Stateless**– Statelessness constraint states that the server should not store any data about the client. So the client has to send all information about its state in each call.
* **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
* **Uniform interface**– By applying this principle of generality to the component interface, the overall system architecture is simplified, and the visibility of interactions is improved. REST is defined by four interface constraints:
  + identification of resources: Which tell Format of Data that can be JSON, XML or Text
  + manipulation of resources through representations: Metadata along with resources has enough information on how to modify the resource
  + self-descriptive messages: Each resource has the information on how to process the message ie.e use of HTTP method (GET, PUT, POST, Delete)
  + hypermedia as the engine of application state: Whenever relevant, a resource should contain links (HATEOAS) pointing to relative URIs to fetch related information.
* **Layered system** – REST allows you to use a layered system architecture where you deploy the APIs on server A and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way.
* **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. Code On Demand is the only Optional constraint. Most of the time, API will be sending the static representations of resources in the form of XML or JSON. But when it is needed, API is free to return executable code to support a part of the application
* REST  stands for  **"Representational State Transfer"**. It is an architectural style for the web which uses HTTP Protocol for communication, or you can call it a concept for the web which uses HTTP Protocol with a certain set principle on top of it for communication.
* This set of principles of REST states that any information in REST is a resource (call it data/functionality/web page/image/object) and this resource can be uniquely identified and is uniform across.
* The state(value) of resources at any timestamp is termed as resource representation.
* The communication in REST is made by exchanging representation of these resources (which can be JSON, XML, text format), while each request in communication being independent of the previous request, which makes REST to be Stateless approach that means each time client make HTTP request, it sends all necessary information to the server to fulfill it, sever never relies on information received from the previous request making both requests isolated from each other.
* Thus REST is the transfer of state of the resource representation.
* What does State Transfer mean?
* When we make a request to a resource using HTTP, we may be asking for the current state of this resource or its desired state. The server will understand the correct operation to be invoked based on the HTTP method that was used to make a call.
* REST takes advantage of the HTTP protocol and JSON format because of which it is fast.
* For better understanding let us consider an example of an eCommerce website selling TVs. Consider this TV, if you observe there is a card with an image and certain data along with it.
* 
* This data/resource can be represented as a JSON (representation of resource). Now at the current state price of this tv is 31999 bucks, but this state price might differ if you see it 2 days later. So, if the request made to fetch the state of this resource representation at any time in future the value might change for price and rating, that’s what REST all about, transfer of state of the resource representation.

What is RESTful API?

RESTful API is typically used to refer to web services implementing REST architecture.

In RESTful, resources are manipulated using GET, PUT, POST, and DELETE methods of HTTP to perform read, update, create, and delete operations on the resource.

You can use the HTTP Methods for the following operation on the resources



* **GET-** GET request is used only to fetch details/data from web service. It should not do any modification of data present on the server-side.
* **POST-** When a new entity (for e.g: detail of a new employee) has to stored then the POST request is used. Data can be sent to the server using an HTML form.
* **PUT-** When any modification or addition of new data to an existing entity is needed, the PUT request can be used.
* **DELETE-** DELETE method is used whenever data has to be removed in the server.

Some popular RESTFul Web Services are Google Maps API, PayPal Rest API, etc.

In this course, we will create a RESTful web service using Node and Express.

**Objective**

To send a request to the RESTful web service, we need a client that will trigger this request. In scenarios where our application is completely end-to-end, we use a browser to trigger the request either through form submission or through a link. In the scenario which we are in, we need to use a replica of a client in order to test the web service that we will be working on; thus we will be using a lightweight client application called **Rest Client.**

**How to use the Rest Client?**

We are going to take up two scenarios in the upcoming demo, where we will send the request with

* GET method
* POST method

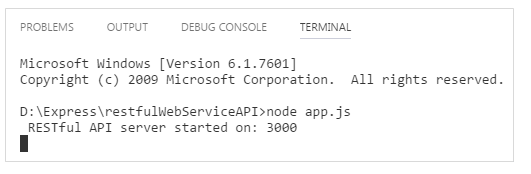
We will also access the relevant information coming from the client using the **request** object, whose syntax and meaning are discussed further in the course.

In the next page, we have given you an application to download and explore where you can trigger various request using GET and POST HTTP methods. Screenshots have been provided to give you clarity in using the Rest Client. You can use **RestClient.jar**file

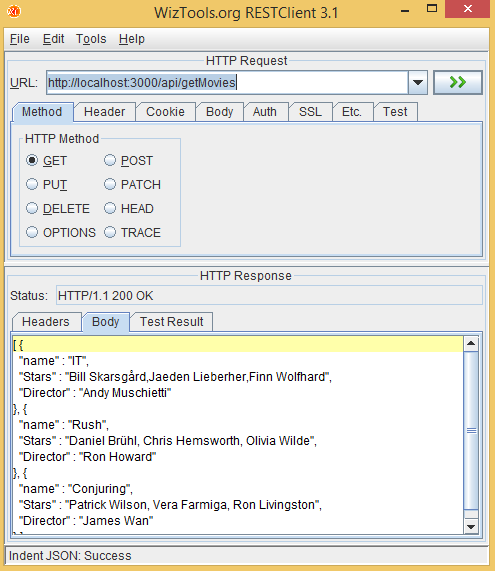
**Problem statement:**

**Step 1:** Download the [RESTful API Web Service](https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_24881105855761703000_shared/web-hosted/assets/RESTapi_Demo.zip) project.

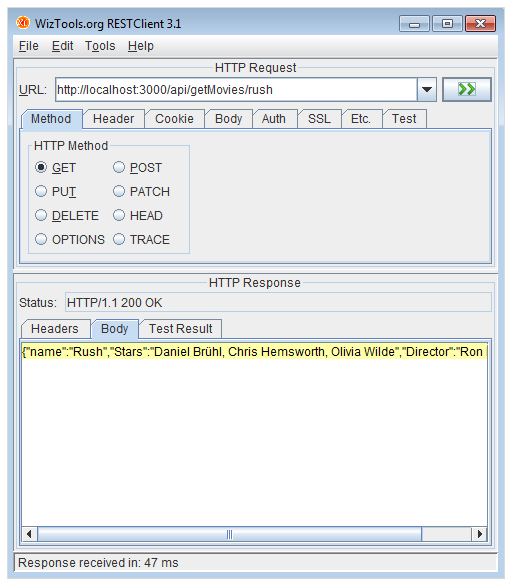
**Step 2:** Start the Server to be able to use the RESTful API Web Service for retrieving, adding, updating and deleting movies data.



**Step 3:** To trigger a request with the GET method to retrieve movie data using REST client, do it as shown below. Enter the URL, select the GET method, and click on the double arrow button to send the request. Observe the response message in the JSON format as shown in 'Body tab'

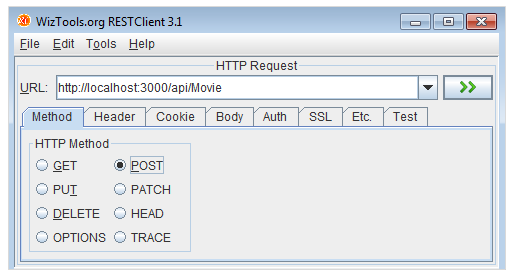


**Step 4:** To trigger a request with the GET method to get one particular movie data using REST client, do it as shown below. Enter the URL, select the GET method, and click on the double arrow button to send the request. Observe the response message in the JSON format as shown in 'Body tab'

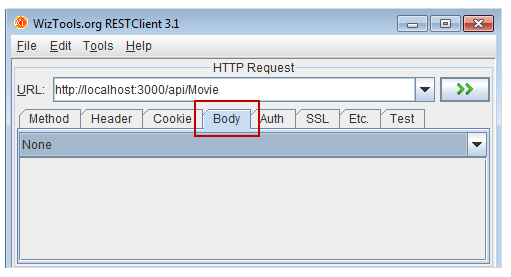


**Step 5:** To trigger a request with POST method to add movies data using REST client, follow the steps below

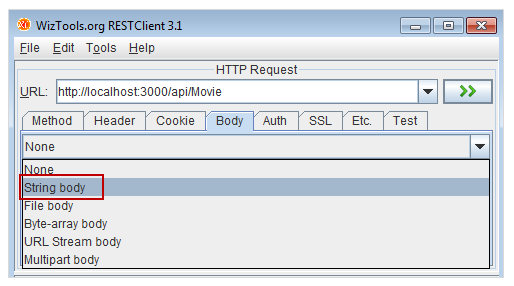
* Select POST method



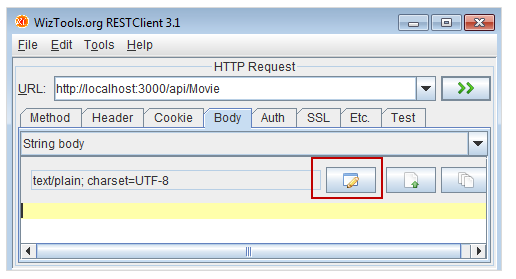
* Select Body tab



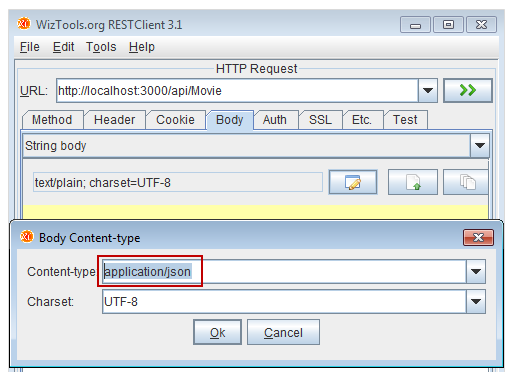
From the drop-down arrow select 'String body'



Click on the highlighted button in the below screenshot:



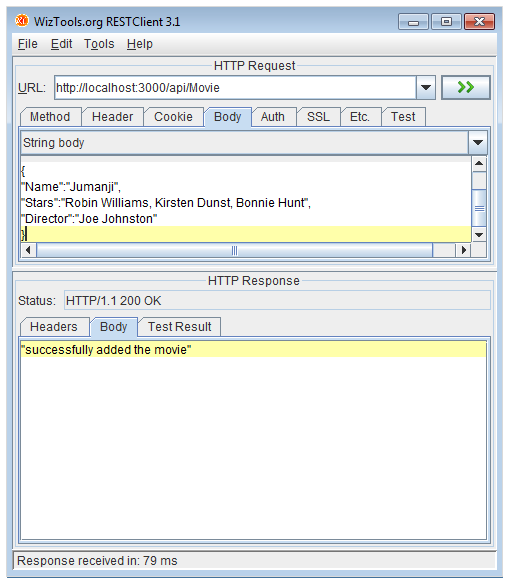
Select 'application/json' for content-type field:



Enter the following JSON data in the body tab as shown below:

1. {
2. "Name":"Jumanji",
3. "Stars":"Robin Williams, Kirsten Dunst, Bonnie Hunt",
4. "Director":"Joe Johnston"
5. }

On clicking double-arrow button you can see the response as shown below:



**Step 6:** Similarly, try out triggering the following requests with the PUT method to update movie data and DELETE method to remove movie data, using REST client

moviename, key, value are parameters sent via URL. For example, 'stars' can be assigned to a key, 'Chris Van Allsburg' can be assigned to value and 'Jumanji' can be assigned to movie name.

1. http://localhost:3000/api/Movie/:moviename/:key/:value
2. http://localhost:3000/api/Movie/:moviename

**Note:** In the above-given project, you just served the request on REST client and observed the output the code gave, this coded demo project was Restful Webservice built using Node & Express. You can also create a RESTful Webservice using SpringBoot, Django. But in this course, we will learn next to create web services using Node & Express.

* All the browsers use JavaScript engines to run these JavaScript codes.
* In order to execute a JavaScript program outside the browser environment, **Ryan Dahl** created a run-time environment called **Node.js**.
* Thus,**Node.js** can be used as the server-side technology as well.

Let us learn about the **features**of Node.js.

**Features of Node.js:**

**1. Reduced resources**:

Using Node.js as the backend technology makes the application single language dependent (**One Code Base**) i.e. JavaScript is used in both client-side as well as server-side. This helps in -

* Building faster application with less number of Human resources
* Building an application with fewer lines of code and less number of files

**2. Library support**:

NPM **(Node Package Manager)** is a default package manager provided by the Node.js. NPM offers the worlds largest software registry. You can also [explore other libraries](https://www.npmjs.com/).

**3. Asynchronous and Event-driven**:

The servers which are created using Node.js handles all the request asynchronously, which means it will never wait for a return value from the APIs

**4. Very Fast**:

Node.js is a run-time-environment which helps us code JavaScript, which is an interpreted language. Since Node.js is built on Google Chrome's V8 JavaScript Engine, it helps in executing the JavaScript code faster.

**5. Single threaded but highly scalable**:

A traditional web server creates limited threads for handling the incoming request. The usage of **Event handling mechanism**in Node.Js web server sends a response in a non-blocking way. Hence Node server can handle a large number of the request than the commonly used traditional servers.

To know more on non-blocking, refer [Node docs](https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/).

**6. No buffering**:

Node.js APIs' are asynchronous, hence they do not buffer the entire stream of data before processing it. These APIs' send the incoming stream of data in chunks for further processing.

**7. Streaming of data is easy**:

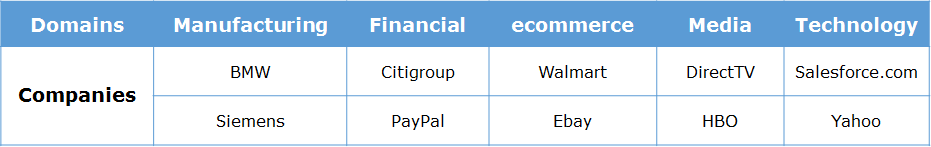
Node has built-in Stream API available, which can stream the data very fast. This API is useful in applications like a Twitter stream, video streaming etc.

**8. Wide client side and database connectivity :**

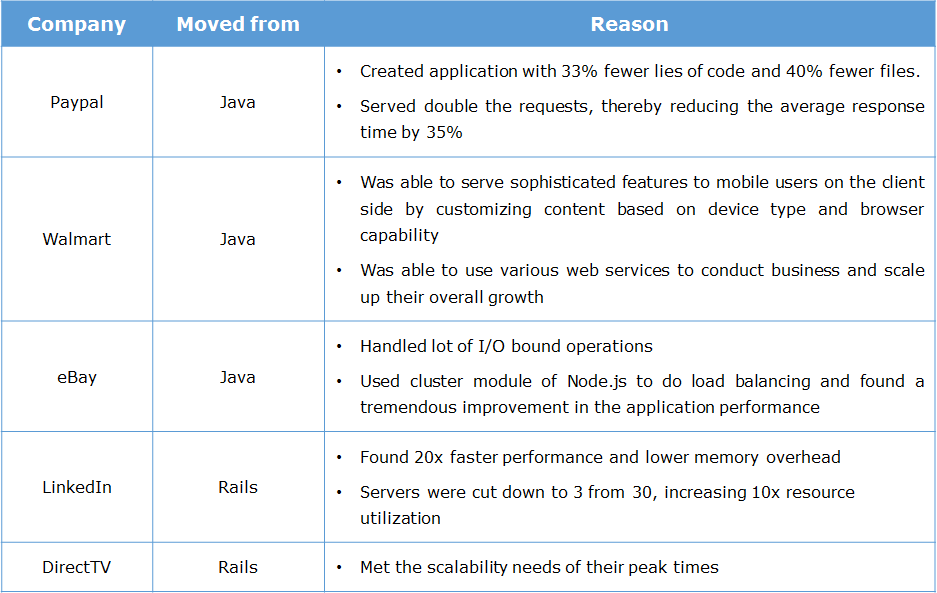
Node.js has absolutely no dependencies and also jells perfectly with any possible client-side technology like **Angular**, **Reactjs** etc. and any database like **MySQL** or **MongoDB**.

Because of all these features Node.js has grabbed attention of various organisations across different domains.

Node.js is used in applications developed for a wide range of domains. The below table lists out few domains and the companies that have migrated to Node.js.



The below table lists the reasons for moving from current technology to Node.js technology.



Now that we have seen the importance of Node.js, let us dive deep into the **Node.js** technology.

# ****Node.js :****

* Is an open-source server side run-time environment built on **Chrome's V8 JavaScript engine**.
* Has a model which is event-driven and non-blocking, which makes it effective and lightweight

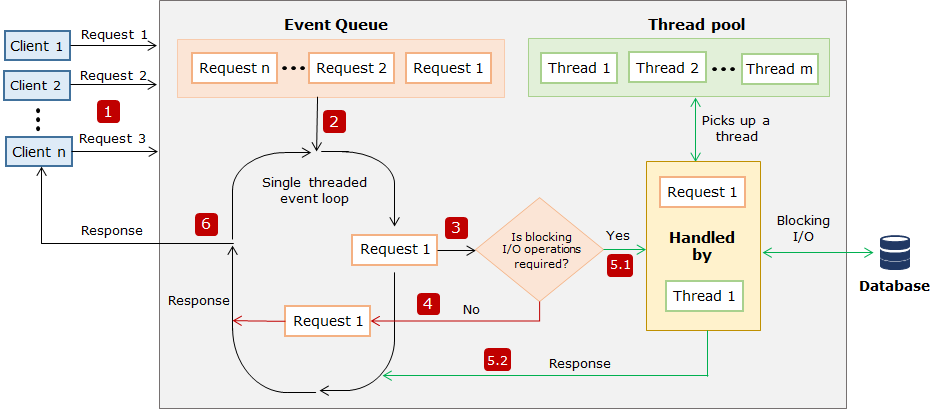
# ****Chrome's V8 JavaScript engine :****

* V8 is Google's open source high-performance JavaScript engine, written in C++ and used in Google Chrome, designed to improve the performance of the JavaScript execution in browsers.
* At the time of execution, it converts the JavaScript code into machine code using **Just-In-Time (JIT) Compiler**, to achieve high speed. Therefore byte code or intermediate code will not be created

Node.js is said to be highly scalable, because it handles the client request using **Single threaded model with event loop.**

Let us now take a look at this model structure.

Node.js environment is created based on **Single Threaded with Event Loop Model** which is built using JavaScript's callback mechanism.



**Single threaded event loop model processing steps:**

**Step 1:**   Assume **multiple** clients, send a request to access the web application **concurrently**from a **web server.** Node.js web server receives those requests from clients and places them into a queue known as **"Event Queue"**. The Node.js web server internally maintains a **limited thread pool** to provide service to the client. Let us assume **'**m**'** number of threads can be created and maintained.

**Step 2:**   The Node.js web server internally has a component, known as **"Event Loop"**. It receives a request and process them in an indefinite loop. But the Event loop component uses a **"Single Thread"** to process all the request.

**Step 3:**Event Loop component checks for any client request that is placed in Event Queue. If no requests are present, then it waits for incoming requests. If the requests are present, then it picks up one client request from Event Queue and starts processing that request.

**Step 4:**   If that client's request has no blocking I/O operations involved, then the request is processed till completion and the response is sent back to the client.

**Step 5.1:**If the client request requires some blocking I/O operations like file operations, database interactions, any external services then it checks the availability of threads from internal thread pool.

**Step 5.2:**One thread is assigned from the internal pool of threads and assigned to the client request. That thread is responsible for taking that request, processing it and performing blocking I/O operations.

**Step 6:** Once the request is processed, it prepares the response and sends it back to the Event Loop, which in turn, sends that response to the corresponding client.

Let us get started by coding in node Js

Let us test if the Node has been properly installed and configured in our system. To verify,

a. Open Visual Studio code IDE and press [Ctrl + `] keys to open the integrated console.

b. Type the following command.

1. node -v

If Node.js is installed correctly, we will observe the Node version in our terminal

Now that the required software is installed, let us go ahead and create our first node application.

**Note:** At the time of this course creation latest stable version for the nodejs is 12.18.3, you can always visit [nodejs.org](https://nodejs.org/en/) to get the stable and current version of nodejs.

**Highlights:**

* Creating a javascript file
* Executing a javascript file

Now As we know that Node environment is installed in our machine, let us create our first Node program.

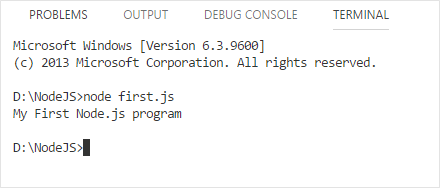
**Step 1:** Create a new JavaScript file, **first.js** and type the below code to be executed inside the file.

1. console.log("My First Node.js program");

**Step 2:** Execute the file using the node command.

1. node first.js

**Step 3:** After successful interpretation of the code, we can see the output in the terminal.

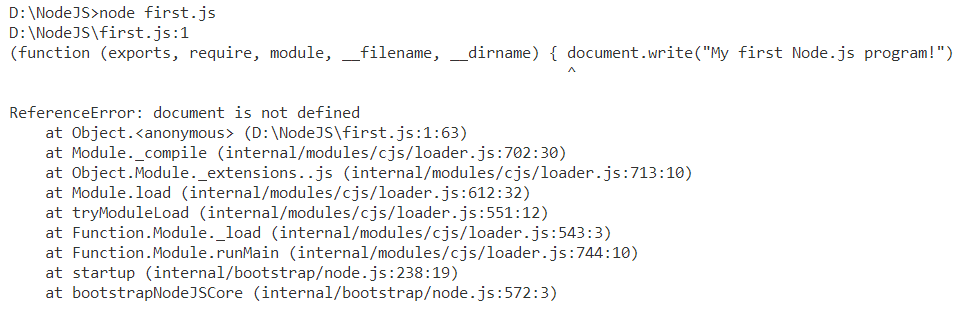


Let us see what happens when the JavaScript file which contain code for browser interaction is executed through Node.js

Let us make a small modification in our previous code.

1. document.write("My first Node.js program!");

We have used the **document.write()** function to display an output in the browser. But while interpreting the same in Node, we encounter an error as shown.



So any JavaScript file which doesn't contain codes for browser interaction will execute successfully.

# Highlights:

* Creating a Node application
* Executing first Node application

The below steps shows how to execute a **Node.js**application.

**Step 1:** Open VS Code and create a new JavaScript file named ***firstApp.js*** and type the following code.

1. let myFunction = (message) => {
2. console.log(message);
3. }
4. myFunction("I have begun my journey through Node.js");

**Step 2:** In the integrated terminal of VS code IDE, execute the file using the node command.

1. node firstApp.js

**Step 3:** Observe the output in the terminal.

https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_11742273089797390000_shared/web-hosted/assets/noderunres.PNG