Assignment:

Make A note on Web Back-End Development

1. Rest API Best Practices

**Best Practices for Naming REST API Endpoints**

* Use Nouns to Name URIs. ...
* Use Clear, Unabridged Names That Are Intuitive. ...
* Use Forward Slashes to Denote URI Hierarchy. ...
* Separate Words with Hyphens. ...
* Use Lowercase Letters. ...
* Avoid Special Characters. ...
* Avoid File Extensions. ...
* Be Consistent with Naming REST API Endpoints.

**REST API Design Best Practices**

* Use JSON as the Format for Sending and Receiving Data. ...
* Use Nouns Instead of Verbs in Endpoints. ...
* Name Collections with Plural Nouns. ...
* Use Status Codes in Error Handling. ...
* Use Nesting on Endpoints to Show Relationships. ...
* Use Filtering, Sorting, and Pagination to Retrieve the Data Requested.

Response Body:

The response body can contain information about the output and additional output ports. If the output type was file, the response body contains a path to the output. If the output type was buffer, the response body contains the buffer contents. The response body can also contain a success or failure code and message.

Response Cookies:

The response cookie are **the cookies that you want to place in the browser**.

**Cookies should never be used to store secure data, such as passwords. Cookies are transmitted as clear text. If a malicious user taps an Internet connection, then they can take cookie data to impersonate a client and gain access to their data. If you must transmit sensitive data, do so on a Secure Sockets Layer (SSL) connection.**

GET REQUEST:

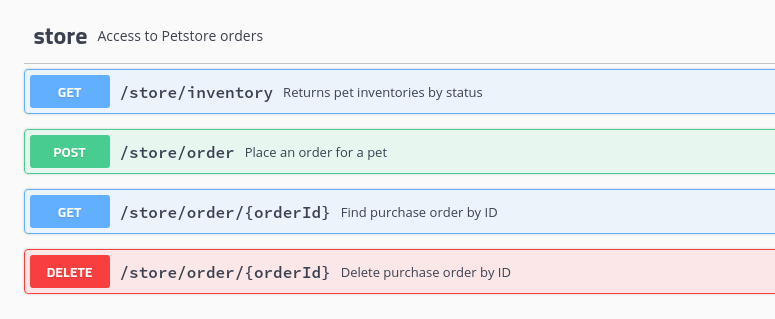
A GET request **gets the information from the server**. When you make the GET request on the server, then the server responds to the request. GET request will not affect any data on the server. Means, there is no creation, updation, addition, or deletion of data on the server when you are making a GET request.

POST REQUEST:

Postman POST request allows appending data to the endpoint. This is a method used **to add information within the request body in the server**. It is commonly used for passing delicate information. Once we send some the request body via POST method, the API in turn yields certain information to us in Response.

2. Http methods Best Practices

Ever wondered what the difference is between GET and POST requests, or when to use PUT? You're not alone. Having a basic understanding of the different HTTP methods, or *verbs*, an API supports is an helpful knowledge when **exploring and testing APIs**.



In this post, I'll discuss how each HTTP method is used and how to **incorporate them in your API testing**.

**HTTP Methods**

* [GET](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#get)
* [POST](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#post)
* [PUT](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#put)
* [HEAD](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#head)
* [DELETE](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#delete)
* [PATCH](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#patch)
* [OPTIONS](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#options)

GET

GET requests are the most common and widely used methods in APIs and websites. Simply put, the GET method is used to **retreive data from a server at the specified resource**. For example, say you have an API with a /users endpoint. Making a GET request to that endpoint should return a list of all available users.

Since a GET request is only requesting data and not modifying any resources, it's considered [a safe and idempotent method](http://restcookbook.com/HTTP%20Methods/idempotency/).

Testing an API with GET requests

When you're creating tests for an API, the GET method will likely be the most frequent type of request made by consumers of the service, so it's important to **check every known endpoint with a GET request**.

At a basic level, these things should be validated:

* Check that a valid GET request returns a 200 status code.
* Ensure that a GET request to a specific resource returns the correct data. For example, GET /users returns a list of users.

GET is often the **default method in HTTP clients**, so creating tests for these resources should be simple with any tool you choose.

POST

In web services, POST requests are used to **send data to the API server** to create or update a resource. The data sent to the server is stored in the [request body](https://stackoverflow.com/questions/22034144/what-does-it-mean-http-request-body) of the HTTP request.

The simplest example is [a contact form](https://assertible.com/contact) on a website. When you fill out the inputs in a form and hit *Send*, that data is put in the **response body** of the request and sent to the server. This may be JSON, XML, or query parameters (there's plenty of other formats, but these are the most common).

It's worth noting that a POST request is **non-idempotent**. It mutates data on the backend server (by creating or updating a resource), as opposed to a GET request which does not change any data. [Here is a great explanation of idempotentcy](https://www.infoq.com/news/2013/04/idempotent).

Testing an API with POST requests

The second most common HTTP method you'll encounter in your API tests is POST. As [mentioned above](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#post), POST requests are used to **send data to the API server** and create or update a resource. Since POST requests modify data, it's important to **have API tests for all of your POST methods**.

Here are some tips for testing POST requests:

* Create a resource with a POST request and ensure a 200 status code is returned.
* Next, make a GET request for that resource, and ensure the data was saved correctly.
* Add tests that ensure POST requests **fail** with incorrect or ill-formatted data.

For some more ideas on common API testing scenarios, [check out this post](https://assertible.com/blog/4-common-api-errors-and-how-to-test-them).

PUT

Simlar to POST, PUT requests are used to send data to the API to **update or create a resource**. The difference is that [**PUT requests are idempotent**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT). That is, calling the same PUT request multiple times **will always produce the same result**. In contrast, calling a POST request repeatedly make have side effects of creating the same resource multiple times.

Generally, when a PUT request *creates* a resource the server will respond with a 201 (Created), and if the request *modifies* existing resource the server will return a 200 (OK) or 204 (No Content).

Testing an API with PUT requests

Testing an APIs PUT methods is very similar to testing POST requests. But now that we know the difference between the two (idempotency), we can **create API tests to confirm this behavior**.

Check for these things when testing PUT requests:

* Repeatedly calling a PUT request always returns the same result ([idempotent](https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them#put)).
* The proper status code is returned when creating and updating a resource (eg, 201 or 200/204).
* After updating a resource with a PUT request, a GET request for that resource should return the correct data.
* PUT requests should fail if invalid data is supplied in the request -- **nothing should be updated**.

PATCH

A PATCH request is one of the lesser-known HTTP methods, but I'm including it this high in the list since it is similar to POST and PUT. The difference with PATCH is that you **only apply partial modifications to the resource**.

The difference between PATCH and PUT, is that a [**PATCH request is non-idempotent**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH) (like a POST request).

To expand on partial modification, say you're API has a /users/{{userid}} endpoint, and a user has a *username*. With a PATCH request, **you may only need to send the updated username** in the request body - as opposed to POST and PUT which require the full user entity.

Testing an API with PATCH requests

Since the PATCH method is so simlar to POST and PUT, many of **the same testing techniques apply.** It's still important to validate the behavior of any API endpoints that accept this method.

What to look for when testing PATCH requests:

* A successful PATCH request should return a 2xx status code.
* PATCH requests should fail if invalid data is supplied in the request -- **nothing should be updated**.

*The semantics of PATCH requests will largely depend on the specific API you're testing.*

DELETE

The DELETE method is exactly as it sounds: **delete the resource at the specified URL**. This method is one of the more common in RESTful APIs so it's good to know how it works.

If a new user is created with a POST request to /users, and it can be retrieved with a GET request to /users/{{userid}}, then making a DELETE request to /users/{{userid}} will completely remove that user.

Testing an API with DELETE requests

DELETE requests should be heavily tested since they generally remove data from a database. Be careful when testing DELETE methods, make sure you're using the correct credentials and not testing with real user data.

A **typical test case for a DELETE request** would look like this:

1. Create a new user with a POST request to /users
2. With the user id returned from the POST, make a DELETE request to /users/{{userid}}
3. A subsequent GET request to /users/{{userid}} should return a 404 not found status code.

In addition, sending a DELETE request to an unknown resource should return [a non-200 status code](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

HEAD

The HEAD method is almost identical to GET, **except without the response body**. In other words, if GET /users returns a list of users, then HEAD /users will make the same request but won't get back the list of users.

HEAD requests are **useful for checking what a GET request will return** before actually making a GET request -- like before downloading a large file or response body. [Learn more about HEAD requests on MDN](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD).

*It's worth pointing out that not every endpoint that supports GET will support HEAD - it completely depends on the API you're testing.*

Testing an API with HEAD requests

Making API requests with HEAD methods is actually an effective way of simply **verifying that a resource is available**. It is good practice to have a test for HEAD requests everywhere you have a test for GET requests (as long as the API supports it).

Check these things when testing an API with HEAD requests:

* Verify and check [HTTP headers](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) returned from a HEAD request
* Make assertions against the status code of HEAD requests
* Test requests with various query parametesr to ensure the API responds

Another useful case for HEAD requests is [API smoke testing](https://blog.codeship.com/add-post-deploy-smoke-tests-to-any-codeship-pipeline/) - **make a HEAD request against every API endpoint** to ensure they're available.

OPTIONS

Last but not least we have OPTIONS requests. OPTIONS requests are one of my favorites, though not as widely used as the other HTTP methods. In a nutshell, an OPTIONS request should **return data describing what *other* methods and operations the server supports** at the given URL.

OPTIONS requests are more loosely defined and used than the others, making them a good candidate to **test for fatal API errors**. If an API isn't expecting an OPTIONS request, it's good to put a test case in place that verifies failing behavior.

Testing an API with OPTIONS requests

Testing an OPTIONS request is dependent on the web service; whether or not it supports that and what is supposed to return will **define how you should test it**.

How to validate an endpoint using OPTIONS:

* Primarily, check the response headers and status code of the request
* Test endpoints that don't support OPTIONS, and ensure they fail appropriately

3.JSON Best Practices

**JSON Structure, Syntax, and Usage**

JSON’s simplicity is part of its appeal. It’s easy to write, easy to read, and easy to translate between the data structures used by most languages. Let’s look at what makes up a JSON object, the data types that JSON supports, and other specifics with the syntax of this popular data format.

Chances are you’ve seen JSON as you’ve looked through data or API documentation. The criteria for valid JSON is rather elementary, though it can be used to describe complex data. The structure of a JSON object is as follows:

* Curly braces {} hold objects
* The data are in key, value pairs
* Square brackets [] hold arrays
* Each data element is enclosed with quotes if it‘s a character, or without quotes if it is a numeric value
* Commas are used to separate pieces of data

Here’s a basic example:

**{** "name":"Katherine Johnson" **}**

The key is “name” and the value is “Katherine Johnson” in the above example. However, JSON can hold more than one key:value pair. This second example adds an “age” key, which includes a number and a second string value, assigned to the “city” key:

**{** "name":"Katherine Johnson"**,** "age":101**,** "city":"Newport News" **}**

It’s common to encounter nested JSON structures, like this:

**{**"inventors":**[**

**{** "name":"Katherine Johnson"**,** "age":101**,** "city":"Newport News" **},**

**{** "name":"Dorothy Vaughan"**,** "age":98**,** "city":"Hampton" **},**

**{** "name":"Henry Ford"**,** "age":83**,** "city":"Detroit" **}**

**]}**

In this final example, you see a primary object with a single key (“inventors”) that has an array as its value. Within that array, each item is itself an object, similar to the earlier simple example. Objects and arrays are values that can hold other values, so there’s an unlimited nesting that could happen with JSON data. That allows JSON to describe most data types, from tabular to even more complex.

**JSON Data Types**

Now that you’ve seen the structure of JSON, you’ve been introduced to several of its data types. There are only a couple others to introduce. Here is the complete list of JSON data types:

* string – Literal text that’s enclosed in quotes.
* number – Positive or negative integers or floating point numbers.
* object – A key, value pair enclosed in curly braces
* array – A collection of one or more JSON objects.
* boolean – A value of either true or false with no quotes.
* null – Indicates the absence of data for a key value pair, represented as “null” with no quotes.

Here’s an example of a JSON object that includes all of these data types:

**{**

"name":"Katherine Johnson"**,**

"age":101**,**

"orbital\_mechanics": **[**"trajectories"**,**"launch windows"**,**"emergency return paths"**],**

"mathmatician": true**,**

"last\_location": null

**}**

**JSON Syntax**

We’ve already discussed the structure of JSON, which provides the basics of the syntax. In this section, we’ll suggest some best practices to avoid common JSON errors:

**Always enclose the key, value pair within double quotes.** Most JSON parsers don’t like to parse JSON objects with single quotes.

**{** "name": "Katherine Johnson" **}**

**Never use hyphens in your key fields.** Use underscores  ( \_ ), all lower case, or camel case.

**{** "first\_name":"Katherine"**,** "last\_name":"Johnson" **}**

**Use a JSON linter to confirm valid JSON.**Install a command line linter or use an online tool like [JSONLint](https://jsonlint.com/" \t "_blank). If you copy this next example into a JSON linter, you should get a parse error for the pesky single quotes around the value for last\_name.

**{** "first\_name":"Katherine"**,** "last\_name":'Johnson' **}**

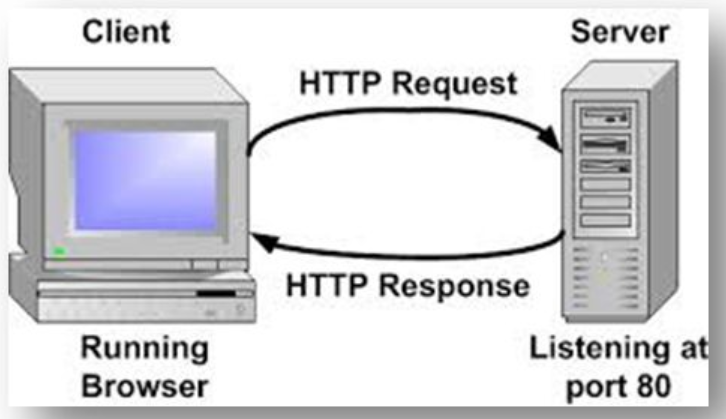
**What Are the Benefits of JSON?**

The rise in JSON’s popularity coincides with a need for websites and mobile apps to more easily and efficiently transfer data from one system to another. But there are many ways that JSON is used to share data, store settings, and interact with systems. Its simplicity and flexibility make it applicable to a number of different situations.

While you’ll find JSON files on servers around the web, the most common usage is to exchange serialized data over a network connection. Some other common uses for JSON include public, front-end, or internal APIs, NoSQL databases, schema descriptions, configuration files, public data, or data exports.  
The **benefits** of JSON include:

* **Compact, efficient format**: JSON syntax offers easy parsing of data and even faster implementation
* **Easily readable:** Both humans and computers can quickly interpret the syntax with minimal errors
* **Broadly supported:** Most languages, operating systems, and browsers can consume JSON out of the box, which allows JSON to be used without compatibility concerns
* **Self-describing**: It’s easy to distinguish between data types and makes it easier to interpret the data without knowing what to expect ahead of time
* **Flexible format**: JSON supports a wide range of data types that can be combined to express the structure of most data

1. **Request - Response Best Practices**



**Server** - the abstract “thing” where the browser’s requests arrive. We don’t care about that detail to much here. It’s enough to know that the backend code is running on the server as well as other services. Request for the backend arrive at the server and are eventually passed on to your backend code.

**Backend** - the part of your web app which is not directly visible to the user. It receives requests and prepares data which is transmitted back to the user’s browser. Backend code is built to be running on a server and it’s never running on the user’s machine.

**Frontend** - the parts of your web application which are intended to be used directly by the the user’s browser. Code which is executed inside the browser, or markup which is interpreted while rendering a page. HTML, CSS and in-browser JavaScript are good examples for what I would consider to be part of the frontend concept. But only in their finished form. While the backend code can be *assembling* a HTML response, the final HTML arriving in the browser is meant here.

**Browser** - an application running on the user’s device. It sends out HTTP requests, receives responses, processes the received data, and uses it to render a viewable page. All of communication from the user’s side goes through their browser.

# **HTTP Request Type**

All requests are sent using an "HTTP method". The method designates the type of request being made to the web server. The most common types of request methods are GET and POST but there are many others, including HEAD, PUT, DELETE, CONNECT, and OPTIONS. GET and POST are widely supported while support for other methods is sometimes limited but expanding.

#### GET requests

* Sent when URL is submitted in the browser location bar or a user clicks a link
* Send data in the URL and query string
* Reloadable
* Can bookmark
* Used for read-only operations
  + View, search, sort, or filter data
  + Data does not change

#### POST requests

* Sent when web form is submitted
* Send data in the URL and as an attachment
* Semi-reloadable (prompt to send data again)
* Can not bookmark
* Used for write operations
  + Create, update, or delete data
  + Data does change

What is request **throttling**?

**Throttling** is **the process of limiting the number of requests you (or your authorized developer) can submit to a given operation in a given amount of time**.

## Throttling Values

* **Request quota** - The number of requests that you can submit at one time without throttling. The request quota decreases with each request you submit, and increases at the restore rate.
* **Restore rate** (also called the **recovery rate**) - The rate at which your request quota increases over time, up to the maximum request quota.
* **Maximum request quota** (also called the **burst rate**) - The maximum size that the request quota can reach.
* **Hourly request quota** - The maximum number of requests you can submit per hour.

**5.Web Security Practices**

**Best practices of Web security:**

### 1. Maintain Security During Web App Development

Before you run out and hire a team of security consultants, realize that you can maintain security in your web applications during the actual development of those tools.

### 2. Be Paranoid: Require Injection & Input Validation (User Input Is Not Your Friend)

A good rule of thumb is to consider all input to be hostile until proven otherwise. Input validation is done so that only properly-formed data passes through the workflow in a web application. This prevents bad or possibly corrupted data from being processed and possibly triggering the malfunction of downstream components.

Some types of input validation are as follows:

* Data type validation (ensures that parameters are of the correct type: numeric, text, et cetera).
* Data format validation (ensures data meets the proper format guidelines for schemas such as JSON or XML).
* Data value validation (ensures parameters meet expectations for accepted value ranges or lengths).

There is a whole lot more to input validation and injection prevention, however, the basic thing to keep in mind is that you want to validate inputs with both a syntactical as well as a semantic approach. Syntactic validation should enforce correct syntax of information (SSN, birth date, currency or whole numbers) while semantic validation should enforce the correctness of their values within a very specific business context (end date is greater than the start date, low price is less than high price).

### 3. Encrypt your data

Encryption is the basic process of encoding information to protect it from anyone who is not authorized to access it. Encryption itself does not prevent interference in transmit of the data but obfuscates the intelligible content to those who are not authorized to access it.

Not only is encryption the most common form of protecting sensitive information across transit, but it can also be used to secure data “at rest” such as information that is stored in databases or other storage devices.

When using Web Services and APIs you should not only implement an authentication plan for entities accessing them, but the data across those services should be encrypted in some fashion. An open, unsecured web service is a hacker’s best friend (and they have shown increasingly smarter algorithms that can find these services rather painlessly).

### 4. Use Exception Management

Another development-focused security measure is proper exception management. You would never want to display anything more than just a generic error message in case of a failure. Including the actual system messages verbatim does not do the end-user any good, and instead works as valuable clues for potentially threatening entities.

When developing, consider that there are generally only three possible outcomes from a security standpoint:

1. Allow the operation.
2. Reject the operation.
3. Handle an exception.

Usually, in the case of an exception or error, you will revert to rejecting the operation. An application that fails securely will prevent operations from unintentionally being allowed. For example, if an ATM failed you would prefer it to display a simple, friendly message to the user (not spill money out onto the ground).

### 5. Apply Authentication, Role Management & Access Control

Implementing effective account management practices such as strong password enforcement, secure password recovery mechanisms and multi-factor authentication are some strong steps to take when building a web application. You can even force re-authentication for users when accessing more sensitive features.

When designing a web application, one very basic goal should be to give each and every user as little privileges as possible for them to get what they need from the system. Using this principle of minimal privilege, you will vastly reduce the chance of an intruder performing operations that could crash the application or even the entire platform in some cases (thus adversely affecting other applications running on that same platform or system).

Other considerations for authentication and access control include things such as password expiration, account lock-outs where applicable, and of course SSL to prevent passwords and other account-related information being sent in plain view.

### 6. Don't Forget Hosting/Service-Focused Measures

Equally important as development-focused security mechanisms, proper configuration management at the service level is necessary to keep your web applications safe.

Is your site vulnerable? [**Read how the LRS web solutions team recovered and secured the Macon County Circuit Clerk’s website after hackers attacked it.**](https://www.lrswebsolutions.com/Blog/Posts/77/Case-Studies/2019/11/Hackers-Attack-Macon-County-Circuit-Clerks-Website/blog-post/)

### 7. Avoid Security Misconfigurations

Given the endless amount of options that contemporary web server management software provides, this also means that there are endless ways to really muck things up:

* Not protecting files/directories from being served
* Not removing default, temporary, or guest accounts from the webserver
* Unnecessarily having ports open on the webserver
* Using old/defunct software libraries
* Using outdated security level protocols
* Allowing digital certificates to expire



Have a well-documented process for not only setting up new websites but also for setting up the web servers and the software used to serve those websites.

The modular nature of web server features allows for more granular control over resources and security. Although, this can make your applications less secure if you are not careful when using them. Be extremely cautious and careful when managing more high-risk security options and features.

### 8. Implement HTTPS (and Redirect All HTTP Traffic to HTTPS)

We had discussed encryption previously with development-focused approaches. Encryption at the service level is also extremely helpful (and sometimes necessary) preventative measure that can be taken to safeguard information. This is typically done by using HTTPS (SSL or Secure Sockets Layer).

SSL is a technology used to establish an encrypted link between a web server and a browser. This ensures that the information passed between the browser and the webserver remains private. SSL is used by millions of websites and is the industry standard for protecting online transactions.

In addition, blanket use of SSL is advised not only because it simply will then protect your entire website, but also because many issues can crop up with resources like stylesheets, JavaScript or other files if they aren’t referenced via HTTPS over an SSL.

### 9. Include Auditing & Logging

We are also concerned with auditing and logging at the server level. Thankfully, much of this is built into the content serving software applications such as IIS (Internet Information Services) and is readily accessible should you need to review various activity-related information.

Not only are logs often the only record that suspicious activity is taking place, but they also provide individual accountability by tracking a user’s actions.

Different from Error Logging, Activity or Audit Logging should not require really much setup at all since it is generally built into the webserver software. Be sure to leverage it to spot unwanted activities, track end user’s actions, and to review application errors not caught at code-level.

In extremely rare cases, logs may be needed in legal proceedings. As I am sure you well know, in these cases the handling of the log data is critical.

### 10. Use Rigorous Quality Assurance and Testing

If your situation at all allows you to, utilizing a third-party service that specializes in penetration testing or vulnerability scanning as an addition to your own testing efforts is a great idea. Many of these specialized services are very affordable.

It is better to be overly cautious when possible, and not rely on only your own in-house quality assurance process to uncover every little hole in every little web application you are using. Adding another layer of testing to catch a few holes here and there that were perhaps not identified by other means of testing is never a bad thing.

To make security upgrades and routine testing efforts go more smoothly, have a well-defined and easily replicable process in place, as well as a thorough inventory of all web applications and where they exist. Nothing is more frustrating than trying to fix security bugs with a specific code library, but to only then have no idea which web applications are even using it!

Your web applications should also be free of any vulnerabilities or breaches that would fail any PCI or HIPAA guidelines. To be certain of this, you should be diligent in all these areas with your approach and design. Whenever possible, you should consult with a party that specializes in adherence to these guidelines so that you can be fully confident that you have everything in place to not only thwart attacks but to simply follow the rules put forth by governing agencies as well.

### 11. Be Proactive to Keep Up With the Bad Guys

When I talk to people about cybersecurity I often use military analogies and phraseology, since cybersecurity seems to me like an arms race. Threats are constantly evolving and developing new attacks and tactics are constantly being developed. Businesses with an online presence must counter these threats to keep up with the ‘bad guys’ out there.

Like a good military strategy, the key to cybersecurity is proactivity.

You should have a well-defined blueprint for a security plan for all your sensitive web applications. This means prioritizing your more high-risk applications. It can be easier to identify if you have an inventory or repository of all the web applications that your business uses or provides to its end users.

As security threats evolve, so should your approach and plan for handling them. Increasingly sophisticated adversaries and ever-expanding soft spots as we turn to web applications to solve more and more of even our most tenable business needs is a concern that requires a full-time effort.

The current reality is that while you cannot exactly expect to avert all attacks, you should certainly aim to meet the challenge by building your own intel as a force multiplier. Get your leadership fully engaged and make sure you have ample resources applied to build an active defense to detect and respond to emerging security risks and hazards.

The web security landscape is changing constantly, and so must your strategy to traverse it.

6.Collect 50 Interview Question on Web Back-End Development

### 1. How do you install an express application generator for scaffolding?

Express application generator is used for quickly creating an application skeleton. The given command is used for installing the express application generator.

npm install expr

ess-generator –g

express myApp

It will create the project "myApp" with some files. Then we install all the dependencies stated in package.json using the given command:

cd myApp

npm install

### 2. How do you install a yeoman for scaffolding?

Generators are used by yeoman for scaffolding the applications. And we can use the following command to install yeoman.

npm install –g yeoman

### 3. Mention the arguments that are available in an Express JS route handler function.

The arguments that are available in the route handler function of Express JS are given below:

* **Res** - It is the response object.
* **Req** - It is the request object
* **Next (optional)** - This argument is used for passing the management to any of the above-given route handlers.

**[ Learn Complete**[**Express JS Tutorial**](https://mindmajix.com/express-js-tutorial)**]**

### 4. Mention the ways of debugging on Linux as well as Windows.

Debugging on Windows can be done as follows:

set DEBUG = express:\*

node app.js

And debugging on [Linux](https://mindmajix.com/linux-tutorial) can be done as follows:

DEBUG = express:\*

node app.js

### 5. List the built-in middleware functions provided by Express.

Express JS provides the following built-in middleware functions:

1. **Static**: We use it for serving static assets like images, HTML files, etc.
2. **JSON**: This is available in Express 4.16.0+. And we use it for passing the incoming requests with JSON payloads.
3. **URL encoded**: This is also available in Express 4.16.0+. And we use it for passing the incoming requests with URL-encoded payloads.

### 6. Mention some third-party middleware provided by Express JS.

Some of the many third-party middlewares that the Express JS provides are:

* Cookie-parser
* Body-parser
* Cors
* Mongoose
* Express-validator
* Sequelize.

[](https://bit.ly/3if9dmk)

### 7. When is application-level Middleware used?

We use the application-level Middleware for binding the app object with the help of the app.use() method. It can be applied on all routes. The syntax is given below:

// This Middleware executes for each route.

App.use(function (req, res, next) {

console.log(‘Current Time:’, Date.now())

next()

})

### 8. Tell us about Router-level Middleware and Built-in Middleware.

**Router-level Middleware**- We use the router-level Middleware for binding with a particular instance of Express.Router().

**Built-in Middleware**- The version 4.x of Express introduced the built-in Middleware. The dependency on connecting gets removed by use of this Middleware.

### 9. Mention some of the databases with which Express JS is integrated.

A myriad of NoSQL and RDBMS databases are supported by Express JS, such as:

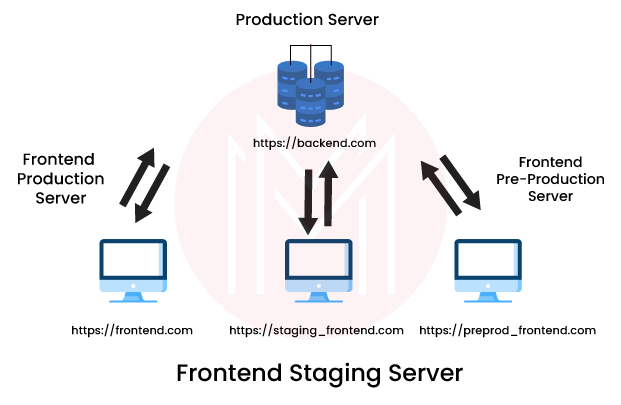
* MySQL
* [MongoDB](https://mindmajix.com/what-is-mongodb)
* PostgreSQL
* SQLite
* Oracle
* SQLite.

|  |
| --- |
| **Related Article**[**MongoDB vs PostgreSQL**](https://mindmajix.com/mongodb-vs-postgresql) |

### 10. What is meant by CORS in Express JS? And what are the ways by which it can be achieved?

CORS is the acronym for Cross-origin resource sharing. We can request the restricted resources from another server or domain using this mechanism. And we can do this mainly in the following three ways:

1. Express cors module
2. Res.header() (or res.set()): Multiple headers can be set using this way.
3. Res.setHeader(): Only a single header can be set in this way.



### 11. What ways are provided by Express JS to configure the properties?

Express JS provides us with two ways for configuring the properties, which are given below:

* With process.ENV
* With require.JS.

### 12. How are the properties configured with process.ENV?

The properties are configured by the given steps:

* We create a file within the project folder and name it ".env."
* All other properties are let to be separate within the ".env" file.
* We can employ any of the properties in server.js.

### 13. How are the properties configured with require.JS?

The properties are configured by the given steps:

* We create a file within the config folder of the project folder and name it "config.json."
* All the config properties are present there within the config.json file.

### 14. How can the Express JS application be structured?

There is no specific answer to this question. The dimensions of our application and hence the concerned team define the solution in different situations. The express logic in the Routes and alternative applications can board as many files as we want in any directory structure. The given examples can be read for further inspiration:

* Route map
* Route listings
* MVC vogue controllers.

### 15. How is the plain HTML rendered?

We don’t need to render HTML with the function- res.render(). Instead, we can use the res.sendFile() function if we have a specific file. And we can use the Express.static() middleware function if we serve several assets from a directory.

## Express JS Interview Questions For Experienced

### 16. Write the code for "Hello world" using Express.

Create a new file by the name- index.js and type the following commands:

var express = require (‘express’);

var app = express ();

app.get (‘/’, function (req, res){

res.send (“Hello world”);

});

app.listen (3000);

Now go to the terminal after saving it and type:

nodemon index.js

### 17. What are the most used HTTP methods in Express JS?

The following HTTP methods are the most used ones:

* **GET**- A specified resource's representation is requested by the GET method. These requests can only retrieve data.
* **POST**- Posting of the data enclosed in the request as a new entity is done using the POST method. The entity is identified by the URI.
* **PUT**- Modification in the existing entity is done with the data enclosed in the request identified by the URI.
* **DELETE**- The request for deleting the specified source is made by the DELETE method.

### 18. How can the cookies be manipulated using ‘Response.cookie()’?

We use the “res.cookie(‘username’, ‘Flavio’)” command is used for manipulating. But it accepts a third parameter conaining various options as specified below:

res.cookie (‘username’, ‘Flavio’, { domain: ‘flaviocopes.com’, path: ‘/administrator’, secure: true })

res.cookie (‘username’, ‘Flavio’ , { expires: new Date(Date.now() + 90000), httpOnly: true})

### 19. When does a Cross-Origin resource get failed in Express JS?

A cross-Origin can fail in the following scenarios-

* If it's to a different domain
* If it's to a different port
* If it's to a different subdomain
* If it's to a different protocol.

### 20. How can you use a Pug template engine inside Express?

We will first install it using the given command:

npm install pug

Then we will set it as following when initializing the Express app:

const express = require ('express')

const app = express()

app.set ('view engine', 'pug')

### 21. What do you mean by the sanitizing input process?

People can always enter weird things via the client-side code. They use the tools to POST things directly to our endpoints. For this, the Express provides various sanitizing methods to prevent these happenings.

### 22. Mention some methods for sanitizing.

Consider the following sanitizing methods:

* Trim() will trim the characters at the beginning as well as the ending of a string.
* Escape() will replace ‘, “, <, >, &, / with the corresponding HTML entities.
* NormalizeEmail() will canonicalize an email address.
* Blacklist() will remove the characters appearing on the blacklist.

### 23. Give an example of HTML form code allowing user to upload a file.

Consider the given example of HTML form code that allows a user to upload file.

<form method = “POST” action = “/submit-form”>

<input type = “file” name = “document” />

<input type = “submit” />

</form>

### 24. What are the methods that you can call when the Formidable.File objects arise giving the information about the uploaded file?

We can call the following methods in such cases-

* **File.name**- the name of the file
* **File.path**- the path to which the file is written
* **File.size**- the size of the file in bytes
* **File. type**- the file's MIME-type.

### 25. What steps will you follow to set up HTTP for Express with the help of Let's Encrypt and Certbot?

We will follow the given steps to set up HTTP:

1. Installing certbot
2. Generating the SSL certificate by Certbot
3. Allowing Express to serve the static files
4. Confirming the domain
5. Obtaining the certificate
6. Setting up the renewal.

### 26. What options are available in the terminal command for generating a skeleton Express JS app?

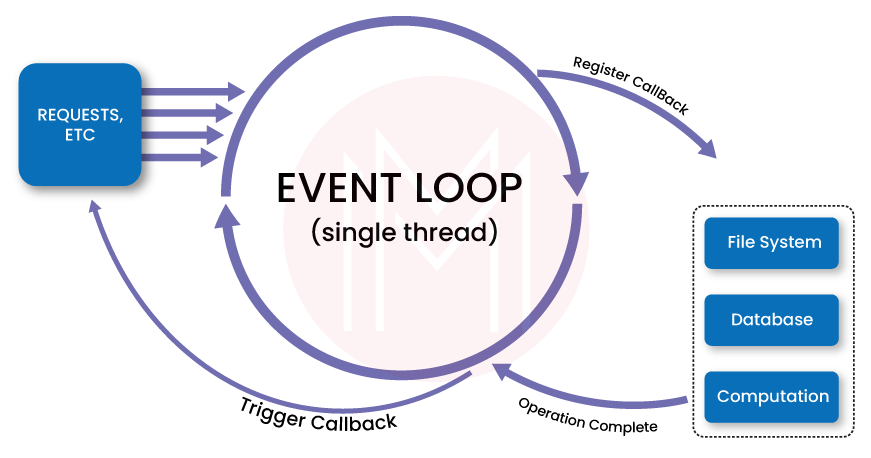
The options available are given below:

* --sessions or –s for adding session report
* --hogan or –H for adding Hogan.js engine support
* --ejs or –e for adding EJS engine support
* --css <engine> or –c <engine> for adding style sheet support
* -jshtml or –J for adding JSHTML engine support
* --force or –f for forcing app generation on the directory which is non-empty.

|  |
| --- |
| **Related Article**[**Basic Examples of Node JS**](https://mindmajix.com/node-js/node-basic-examples) |

### 27. What is meant by an event-loop in Node JS?

The event-loop manages the async content using a listener and queue. The main thread sends the async function to a different thread whenever it requires to be executed. Alongside, v8 is allowed to execute the main code. The event loop has different stages including pending callbacks, timers, check, poll, close callbacks, etc. with different FIFO queues.



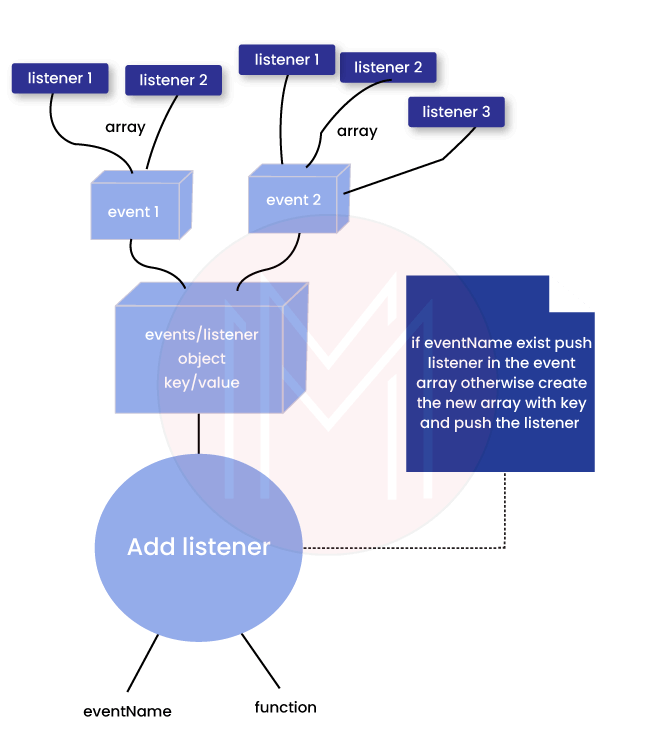
### 28. Why should the Express server and app be separated?

The server initializes the Middleware, routes, and other application logic. On the other hand, the app contains all the business logic that the server-initiated routes will serve. This enables the encapsulation of the business logic from the application logic for smooth functioning.

|  |
| --- |
| **Check out**[**Node JS Interview Questions and Answers**](https://mindmajix.com/node-js-interview-questions-answers) |

### 29. What is meant by an Event emitter in Node JS?

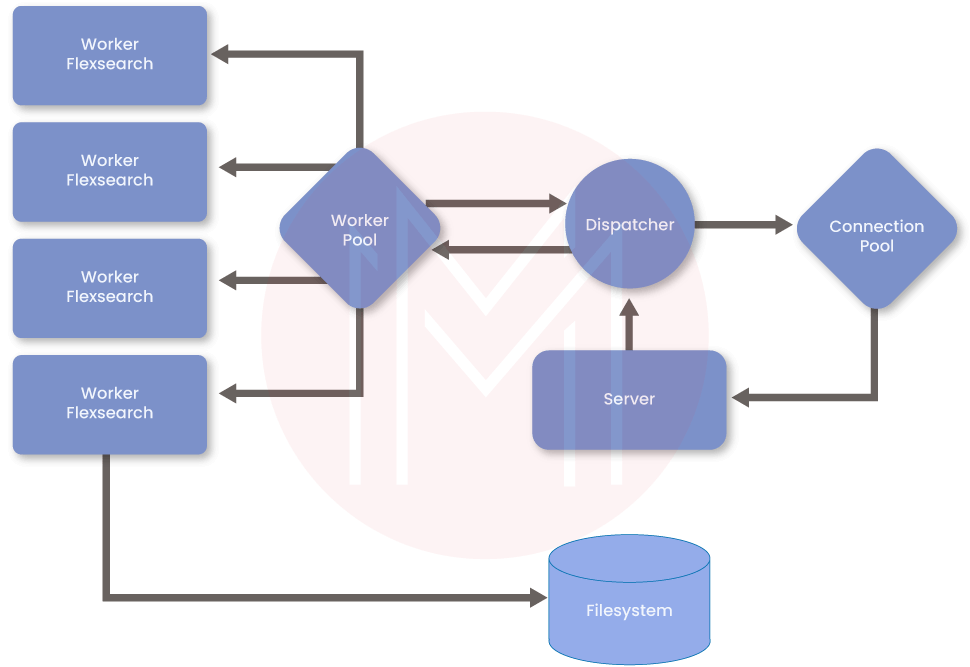
It's a class of Node JS capable of emitting events. We do this by attaching the named events emitted by the object by using the function- eventEmitter.on().



### 30. Differentiate between worker threads and clusters in Node JS.

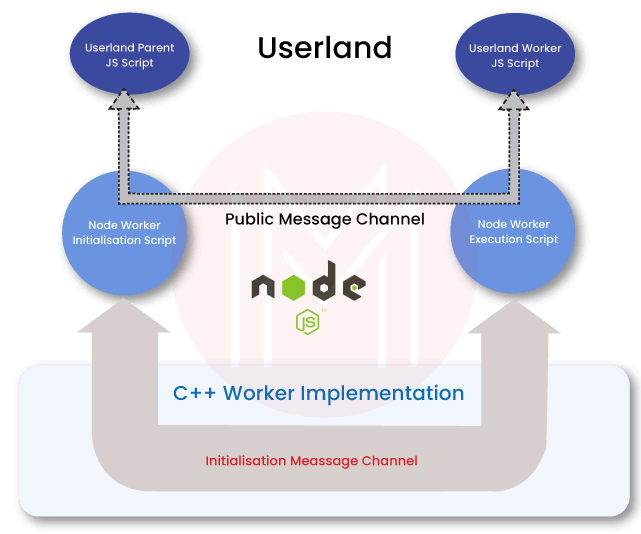
**Cluster**:

* Each CPU has one process with an IPC to communicate.
* Clusters help when multiple servers are required to accept HTTP requests through a single port.
* The processes have separate memory because of being spanned in different CPUs leading to memory issues.



**Worker Threads**:

* Only a single process is there with multiple threads.
* Each Node contains one Node having most APIs accessible.
* The memory is shared with other threads.
* We can use this for CPU-intensive tasks.



#### 31. What do you mean by Express JS and what is its use?

**Answer:**  
Express JS is an application framework that is light-weighted node JS. Variety of versatile, helpful and vital options are provided by this [JavaScript](https://www.educba.com/careers-in-javascript/) framework for the event of mobile additionally as internet applications with the assistance of node JS.  
Express JS Use – Express.js could be a light-weight internet application that helps in organizing the net application into MVC design on the server aspect.

#### 32. Write the steps for setting up an Express JS application?

**Answer:**  
Following are the steps accustomed for An Express JS application:

* A folder with a constant name because the project name is made.
* A file named package.json is made within the folder created.
* “npm install” command is run on the electronic communication. It installs all the libraries gift in package.json.
* A file named server.js is made.
* “Router” file is made within the package that consists of a folder named index.js.
* “App” is made within the package that has the index.html file.

Let us move on to the next Express JS Interview Questions

#### 33. What function are arguments available to Express JS route handlers?

**Answer:**  
The arguments which are available to an Express JS route handler-function are-

* **Req:** the request object
* **Res:** the response object
* **Next (optional):** a function that is employed to pass management to 1 of the following route handlers.

The third argument is optional and should be omitted, however, in some cases, it’s helpful wherever there’s a series of handlers and management will be passed to 1 of the following route handlers skipping this one.

#### 34. How to Config properties in Express JS?

**Answer:**  
In Express JS, there are 2 ways that for configuring the properties:

With process.ENV:

* A file with the name “.env” is to be created within the project folder.
* All the properties are to be other within the “.env” file.
* Any of the properties will be employed in server.js.

With require.JS:

* A file with the name “config.json” is to be created within the config folder within the project folder.
* The config properties are to be present within the config.json file.
* Now, ought to be accustomed access the config.json file.

#### 35. How Should I Structure my Express JS Application?

**Answer:**  
This is the basic Express JS Interview Questions asked in an interview. There is no definitive answer to the current question. The solution depends on the dimensions of your application and therefore the team that’s concerned.  
Routes and alternative application-Express logic will board as several files as you would like, in any directory structure you favor. Read the subsequent examples for inspiration:

* Route listings
* Route map
* MVC vogue controllers

Also, there are third-party extensions for Express JS applications that modify a number of these patterns:

* Resourceful routing

#### 36. How to allow CORS in Express JS? Explain with an example?

**Answer:**  
In order to permit CORS in Express.js, add the subsequent code in server.js:

For Example:

app.all('\*', function(req, res, next) {  
res.set('Access-Control-Allow-Origin', '\*');  
res.set('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT');  
res.set('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type');  
if ('OPTIONS' == req.method) return res.send(200);  
next();  
});

#### 37. How to enable debugging in express app?

**Answer:**  
In different operative Systems, we’ve got following commands:

On UNIX operating system the command would be as follows:

$ DEBUG=express:\* node index.js

On Windows the command would be:

set DEBUG=express:\* & node index.js

From Webstrome IDE

C:\Program Files (x86)\JetBrains\WebStorm 2016.2.4\bin\runnerw.exe" "C:\Program Files\nodejs\node.exe" --debug-brk=61081 --expose\_debug\_as=v8debug E:\Development\nodejd\librarey\bin\www

Let us move on to the next advanced Express JS Interview Questions.

#### 38. Explain Error Handling In Express.js Using An Example?

**Answer:**  
From Express 4.0 Error handling is easier. The steps are as following:

Create a middleware as following:

// error handler  
app.use(function(err, req, res, next) solely providing error in development  
res.locals.message = err.message;  
res.locals.error = req.app.get('env') === 'development' ? err : ;  
// render the error page  
res.status(err.status || 500);  
res.render('error');  
});

Install Error Handler Middleware:

* Install errorhandler

npm install errorhandler --save

* Create a variable

var e errorhandler = require('errorhandler')

* Use the middleware as following:

if (process.env.NODE\_ENV === 'development') solely use in development  
app.use(errorhandler())  
}  
function errorNotification(err, str, req) power unit title = 'Error in ' + req.method + ' ' + req.url  
notifier.notify()  
}

#### 39. What is the use of next in Express JS?

**Answer:**  
Next -It passes management to a consecutive matching route. OR a operate to pass management to 1 of the following route handlers.  
The argument could also be omitted, however, is beneficial in cases wherever you have got a series of handlers and you’d wish to pass management to 1 of the following route handlers, and skip this one.

app.get('/user details/:id?', function(req, res, next));

* **Req and Res:** It represents the request and response objects
* **Next:** It passes management to a consecutive matching route.

#### 40. How to Redirect 404 Errors to A Page In ExpressJS?

**Answer:**  
In server.js add the subsequent code to send 404 errors back to a page in our ExpressJS App:

/\* Define fallback route \*/  
app.use(function(req, res, next) {  
res.status(404).json({errorCode: 404, errorMsg: "route not found"});  
});

### 41. What is Node.js? Where can you use it?

[Node.js is an open-source](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs), cross-platform [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript) runtime environment and library to run web applications outside the client’s browser**.**It is used to create server-side web applications.

Node.js is perfect for data-intensive applications as it uses an asynchronous, event-driven model. You can use  I/O intensive web applications like video streaming sites. You can also use it for developing: Real-time web applications, Network applications, General-purpose applications, and Distributed systems.

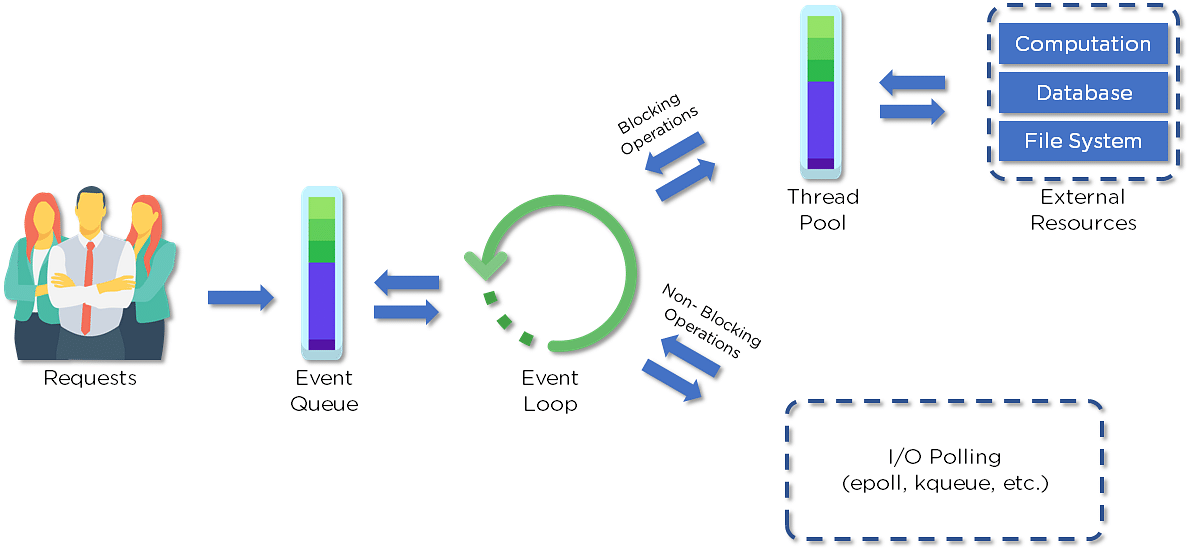
### 42. Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

* It is generally fast
* It rarely blocks
* It offers a unified programming language and data type
* Everything is asynchronous
* It yields great concurrency

### 43. How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.



* Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:
* Querying for data
* Deleting data
* Updating the data
* Node.js retrieves the incoming requests and adds those to the Event Queue
* The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough not to require any external resources
* The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.

Once the task is carried out completely, the response is sent to the Event Loop that sends that response back to the client.**e:**

### 44. Why is Node.js Single-threaded?

Node.js is single-threaded for async processing. By doing async processing on a single-thread under typical web loads, more performance and scalability can be achieved instead of the typical thread-based implementation.

### 45. If Node.js is single-threaded, then how does it handle concurrency?

The Multi-Threaded Request/Response Stateless Model is not followed by the Node JS Platform, and it adheres to the Single-Threaded Event Loop Model. The Node JS Processing paradigm is heavily influenced by the JavaScript Event-based model and the JavaScript callback system. As a result, Node.js can easily manage more concurrent client requests. The event loop is the processing model's beating heart in Node.js.

### 46. Explain callback in Node.js.

A callback function is called after a given task. It allows other code to be run in the meantime and prevents any blocking.  Being an asynchronous platform, Node.js heavily relies on callback. All APIs of Node are written to support callbacks.

### 47. What are the advantages of using promises instead of callbacks?

* The control flow of asynchronous logic is more specified and structured.
* The coupling is low.
* We've built-in error handling.
* Improved readability.

### 48. How would you define the term I/O?

* The term I/O is used to describe any program, operation, or device that transfers data to or from a medium and to or from another medium
* Every transfer is an output from one medium and an input into another. The medium can be a physical device, network, or files within a system



### 49. How is Node.js most frequently used?

Node.js is widely used in the following applications:

1. Real-time chats
2. Internet of Things
3. Complex SPAs (Single-Page Applications)
4. Real-time collaboration tools
5. Streaming applications
6. Microservices architecture

### 50. Explain the difference between frontend and backend development?

|  |  |
| --- | --- |
| **Front-end** | **Back-end** |
| Frontend refers to the client-side of an application | Backend refers to the server-side of an application |
| It is the part of a web application that users can see and interact with | It constitutes everything that happens behind the scenes |
| It typically includes everything that attributes to the visual aspects of a web application | It generally includes a web server that communicates with a database to serve requests |
| HTML, CSS, JavaScript, AngularJS, and ReactJS are some of the essentials of frontend development | Java, PHP, Python, and Node.js are some of the backend development technologies |

### 51.What is NPM?

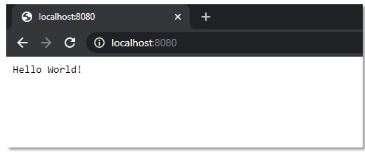
NPM stands for Node Package Manager, responsible for managing all the packages and modules for Node.js.

Node Package Manager provides two main functionalities:

* Provides online repositories for node.js packages/modules, which are searchable on search.nodejs.org
* Provides command-line utility to install Node.js packages and also manages Node.js versions and dependencies

### 52. What are the modules in Node.js?

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. To include a module in a Node.js application, use the **require()** function with the parentheses containing the module's name.



Node.js has many modules to provide the basic functionality needed for a web application. Some of them include:

|  |  |
| --- | --- |
| **Core Modules** | **Description** |
| HTTP | Includes classes, methods, and events to create a Node.js HTTP server |
| Util | Includes utility functions useful for developers |
| Fs | Includes events, classes, and methods to deal with file I/O operations |
| url | Includes methods for URL parsing |
| query string | Includes methods to work with query string |
| Stream | Includes methods to handle streaming data |
| Zlib | Includes methods to compress or decompress files |

### 53. What is the purpose of the module .Exports?

In Node.js, a module encapsulates all related codes into a single unit of code that can be parsed by moving all relevant functions into a single file. You may export a module with the module and export the function, which lets it be imported into another file with a needed keyword.

### 54. Why is Node.js preferred over other backend technologies like Java and PHP?

Some of the reasons why Node.js is preferred include:

* Node.js is very fast
* Node Package Manager has over 50,000 bundles available at the developer’s disposal
* Perfect for data-intensive, real-time web applications, as Node.js never waits for an API to return data
* Better synchronization of code between server and client due to same code base
* Easy for web developers to start using Node.js in their projects as it is a JavaScript library

### 55. What is the difference between Angular and Node.js?

|  |  |
| --- | --- |
| **Angular** | **Node.js** |
| It is a[frontend development](https://www.simplilearn.com/how-to-become-a-front-end-developer-article) framework | It is a server-side environment |
| It is written in[TypeScript](https://www.simplilearn.com/tutorials/programming-tutorial/advanced-typescript) | It is written in [C, C++ languages](https://www.simplilearn.com/tutorials/cpp-tutorial/difference-between-c-and-cpp) |
| Used for building single-page, client-side web applications | Used for building fast and scalable server-side networking applications |
| Splits a web application into MVC components | Generates database queries |

### 56. Which database is more popularly used with Node.js?

[MongoDB](https://www.simplilearn.com/tutorials/mongodb-tutorial) is the most common database used with Node.js. [It is a NoSQL](https://www.simplilearn.com/rise-of-nosql-and-why-it-should-matter-to-you-article), cross-platform, document-oriented database that provides high performance, high availability, and easy scalability.

### 57. What are some of the most commonly used libraries in Node.js?

There are two commonly used libraries in Node.js:

* [**ExpressJS**](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js)- Express is a flexible Node.js web application framework that provides a wide set of features to develop web and mobile applications.
* **Mongoose** - [Mongoose](https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-mongodb) is also a Node.js web application framework that makes it easy to connect an application to a database.

### 58. What are the pros and cons of Node.js?

|  |  |
| --- | --- |
| **Node.js Pros** | **Node.js Cons** |
| Fast processing and an event-based model | **Not suitable for heavy computational tasks** |
| Uses JavaScript, which is well-known amongst developers | Using callback is complex since you end up with several nested callbacks |
| Node Package Manager has over 50,000 packages that provide the functionality to an application | Dealing with relational databases is not a good option for Node.js |
| Best suited for streaming huge amounts of data and I/O intensive operations | Since Node.js is single-threaded, CPU intensive tasks are not its strong suit |