Date - / /
Page No. -

- Why Node.js is Fast ?

- B'coz of its asynchronous, non-blocking I/O model, which allows handle multiple concurrent requests without waiting for each one to finish.

- Express.JS.

- It is node.js web app^n framework that provides broad features to create / build web app^n
- It provides minimal interface to build our app^n.
- It is flexible as there are numerous modules available on npm, which can be directly plugged into Express.

- How to Run ? (IntroBack)

- npm init -y - Create JS file
- npm install express
- node index.js.

- framework :
- It is a set of pre-written code that provides a structure for building an application.

- Library :
- It is a collection of pre-built tools that can be used for specific tasks.

Date:- / /
Page No.:-

- Request / Response / Testing / Routing in Express JS
(Routing)

① npm init -y

② npm install ~~nodemon~~ express

③ npm install nodemon // helps to speedy dev. of node js app

④ Add dev scripts in package.json

```
"scripts": {
    "dev": "nodemon index.js",
    "test":
}
```

⑤ index.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello')
})

app.listen(port, () => {
    console.log("App")
})
```

Date:- / /
Page No.:-

- Require
  - Get - used to send only limited amt of data
  - Post - send large amt of data.
  - put - update data
  - delete - When fn is ran it is going straight into throw error line.

(vi) Add require in index.js

```
app.get ('/', (req, res) => {
    res.send ("Get method")
});

app.post ('/', (req, res) => {
    res.send ("post")
});

app.delete ('/items/:id', () => {
});

app.put (() => {
});
```

- Use postman for testing request response routing.

Date:- / /
Page No.:-

(VII) npm run dev.

(VIII) Best way.
- folder - routes — file - item.js.

```
const express = require ('express')
const router = express.Router()

router.get ('/', (req, res) => {
    res.send ("Got a GET req")
    // res.sendFile ("./ tp.html', {root: __dirname});
});

router.post (() => {
});

router.put (() => {
});

router.delete (() => {
});

module.exports = router
```

(IX) import router in index.js

```
const port = 3000

const item = require ('./routes/item');
// import item to router file
```

Date:- / /
Page No.:-

// load into appⁿ
app.use ('/api', item);

- Routing :

- It refers to determining how an application responds to a client request to a particular endpoint

- It is about showing your app how to respond diffⁿ URL

Methods	Description
res.download()	Prompt file to be downloaded.
res.end()	End the response process
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send response
res.sendFile()	Send file
res.sendStatus()	Set the response status code & send its ~~it~~ the representⁿ ~~by~~ as response body.

Middlewar fn access to req. obj, the response obj. & the next middlewar fn in appn req-res cycle.

Date: / /
Page No.:-

- Middleware in Express JS (Middlewar)

① to ⑥ step of previous.

⑥. index.js

```
// looding middlewar into th app
// inbuilt middlewar

app.use (express.json());

app.get ('/', (req, res) => {
    console.log (req.body);
    res.send ("Hello")
})
```

in postman: Create body to send

http://localhost 3000

Body - raw - { "name": "AP", "cgz": "20" }

⑦ Create middleware - loggins, auth, . validation

```
// Creation of middlewar
Const loggingMiddlewar = function (req, res, next) {
    Console.log ("LOGING");
    next();
};

// loading middlewar into appn.
app.use (loggingMiddlewar);
```

Date:- / /
Page No.:-

- Types of middleware

(i) Application - level middleware
(ii) Router - level
(iii) Error - handling
(iv) Built - in
(v) Third - party

- Route - Specific Middleware

/student → student ✓
admin ✗
/admin → admin ✓
Jta ✗

(i) routes → route.js

```
const express = require('express')
const router = express.Router()

const auth = fn(req, res, next) {
    console.log("I am inside auth wali middleware");
    // tumhari sahayta ke liye ek dummy user add
    kr raha hu
    req.user = { userId: 1, role: 'student' };
```

Date:- / /

Date:- / /
Page No.:-

```
if (req.user) {
    next();          // if valid user
}

else {               // not valid user
    res.json({
        success: false,
        message: "Not valid user",
    })
}

const isStudent = fn(req, res, next) {
    console.log("I am inside std wali mw");

    if (req.user.role === "student") {
        next();
    }
    else {
        res.json({
            success: false,
            msg: "Access Denied",
        })
    }
}

// Same for admin

module.exports = router;
```

Date:-
Page No.:-

Date:- / /
Page No.:-

. index . js .

```
const route = require ('./routes/route')
// mounting the routes
app.use ('/api', route)
```

- import path from "path";

let mypath = "/ any url path";
console.log (mypath)
Console.log (path.extname (mypath))
.dirname
basename

Console.log (path.join (url)).

Express JS.

- It is used to build single page, multi page, hybrid web app^n.

- Why Nodemon ?

- helps with speedy devdpement of Nodejs app^n
- It monitors your project directory & automatically restarts your nodejs app^n

npm i expres
npm i expres@4

- req params : capture dynamic values from URL path

- Request . Respons & Routes

- ~~Serving Html file~~ : • (testing port with html)

- public folder → mypage.html) (Create)

- <Script>

async fn testPost () {

let a = await fetch ("/", { method: 'Post' })

let b = await a.text()

console.log (b)

}

testPost()

</Script>

- Serving HTML file

- templates → index.html)

app.get ('/index', () => {

Console.log (index)

~~res.sendFile ("Hello index")~~

res.sendFile ('templates/index.html', {root, __dirname})

Date:- / /
Page No.:-

- Express Router.

- An Express router is a way to organize routes in a nodejs app using Express.
- It helps break the app into smaller, cleanie parts:

documentation → Code
- routes → blog.js
- router.get ('/blogpost/:slug', (req, res) => {
  res.send ('Hitted the blogpost for ' + req.params.slug)
})

- main.js.

```
const blogs = require ('./routes/blogs')
app.use ('/blogs', blogs)
```

- EJS Template Engine in Express

- Embedded JavaScript templating
- It is a simple templating language that lets you generate HTML markup with plain JS.

- npm init -y
- npm i express
- index.js → express basic code
- Template folder created - basic code add in index.html (bootstrap code)

- index.js

```
app.get('/', (req, res) => {
    let siteName = "Adidas"
    let searchTxt = "search now"
    res.sendFile("templates/index.html", {root: __dirname})
})
```

```
app.get('/blog/:slug', (req, res) => {
    let blogTitle = "Adidas"
    let blogcont = "Its good brand"
    res.sendFile("templat/index.html", {root: __dirname})
})
```

- npm install ejs.
- add in index.js (document, ejs)

```
app.set('view engine', 'ejs');
```

- app.get('/blog/:slug'

```
res.render("templates/blogpost.html",
{ blogTitle: blogtitle, blog content: blogcontent })
```

- app.get('/'

```
res.sende("templates/__", { siteName: siteNam
    , searchTxt: searchtxt })
})
```

Date:- / /
Page No.:-

- change name index.html to index.ejs
- templates - views
- code मध्ये पण change करा.

- index.ejs - change anywhe

<%= siteName %>

- Extension - EJS lang support
- snippet - ejs

- See documentation ejs.

ExpressJS Interview Questions and Answers - Beginners Level

1. What is ExpressJS?

Express is a small framework that sits on top of NodeJS's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to NodeJS's HTTP objects and provides the rendering of dynamic HTTP objects.

Express is a part of MEAN stack, a full-stack JavaScript solution for building fast, robust, and maintainable production web applications.

2. Why use ExpressJS?

ExpressJS is a lightweight NodeJS framework that allows us to create server-side web applications faster and smarter. The main reason for choosing Express is its simplicity, minimalism, flexibility, and scalability characteristics. It provides an easy setup for middleware and routing.

3. Write a 'Hello World' ExpressJS application.

To create a simple ExpressJS application, first, we need to install Express in our NodeJS application.

Initialize a NodeJS project


```
npm init -y
```

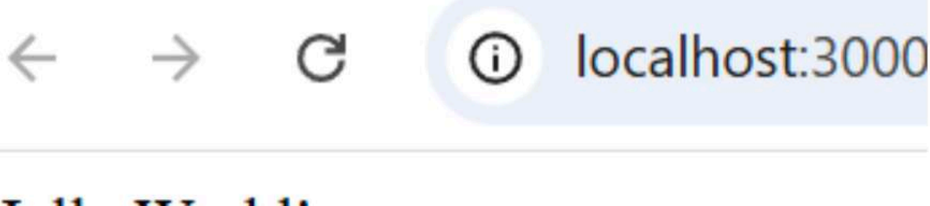
Install ExpressJS

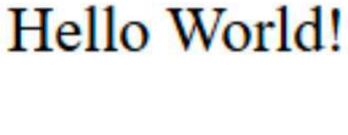
```
npm install express
```

Create the index.js file and write the below code

```
// Import the express module
const express = require('express');

const app = express();

app.get('/', (req, res) => {
    res.send('Hello World!');
});

const PORT = 3000;

// Start the server
app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Output





Hello world in Express.JS

4. Differentiate between NodeJS and ExpressJS?

Feature	NodeJS	ExpressJS
Definition	A runtime environment that allows JavaScript to be executed outside the browser.	A web framework built on top of NodeJS to simplify server-side development.
Type	A server-side runtime environment	A backend web framework based on NodeJS

Hello world in Express.JS

4. Differentiate between NodeJS and ExpressJS?

Feature	NodeJS	ExpressJS
Definition	A runtime environment that allows JavaScript to be executed outside the browser.	A web framework built on top of NodeJS to simplify server-side development.
Type	A server-side JavaScript runtime.	A backend web framework based on NodeJS.
API Development	Can be used to create APIs but requires extra effort.	Provides an easy and structured way to develop RESTful APIs.
Performance	Slightly faster as it's minimal and doesn't include extra abstractions.	Adds a slight overhead due to additional features but is still highly efficient.
Use Case	Ideal for low-level operations, real-time applications, microservices, and command-line tools.	Ideal for web applications, APIs, RESTful services, and middleware-based projects.

5. Is ExpressJS a front-end or a back-end framework?

ExpressJS is a JavaScript backend framework. It is mainly designed to develop complete web applications and APIs. Express is the backend component of the MERN stack which stands for MongoDB, ExpressJS, React.js, NodeJS.

6. Mentions few features of ExpressJS.

Few features of the ExpressJS includes

- **Routing:** Express provides a simple way to define routes for handling HTTP requests. Routes are used to map different URLs to specific pieces of code, making it easy to organize your application's logic.
- **Middleware:** Express uses middleware functions to perform tasks during the request-response cycle. Middleware functions have access to the request, response, and the next middleware function.
- **HTTP Utility Methods:** Express mainly used for handling HTTP methods like GET, POST, PUT, and DELETE. This makes it easy to define how the application should respond to different types of HTTP requests.
- **Static File Serving:** It can also serve static files, such as images, CSS, and JavaScript, with the help of built-in express.static middleware.
- **Security:** It includes features and middleware to strengthen the security of your web applications, such as the helmet middleware to secure your app.

7. Explain the structure of an ExpressJS application?

The structure of an ExpressJS application can vary depending on its complexity and the specific needs of the project. However, here is a basic approach that is commonly used:

- **Entry point:** This is the starting point of the application where you set up your server, connect to your database, add middleware, and define the main routes.
- **Routes directory:** This directory contains files for the app's routes.
- **Controllers directory:** This directory contains files that define the logic to handle requests for a specific route.
- **Models directory:** This directory is used for creating the schema models for the different data.
- **Middleware directory:** This directory contains custom middleware functions that you can use in your routes.
- **Views directory:** If you're using a templating engine, this directory contains your view templates.
- **Public directory:** This directory contains static files that are served directly by the server such as images, CSS files, and JavaScript files.

8. What are some popular alternatives to ExpressJS?

There are several popular alternatives to ExpressJS which includes:

- Koa.js
- Hapi.js

images, CSS files, and JavaScript files.

8. What are some popular alternatives to ExpressJS?

There are several popular alternatives to ExpressJS which includes:

- Koa.js
- Hapi.js
- Sails.js
- Fastify

9. Which major tools can be integrated with ExpressJS?

There are many tools and libraries that can be integrated with ExpressJS such as:

- **Database tools:** MongoDB, MySQL, PostgreSQL.
- **Template Engines:** EJS, Pug, Mustache.
- **Authentication libraries:** Passport.js.
- **Logging libraries:** Morgan, Winston.
- **Validation libraries**: Joi, express-validator.
- **ORM libraries:** Sequelize, Mongoose.

10. What is .env file used for?

The .env file is used for storing sensitive information in a web application which we don't want to expose to others like password, database connection string etc. It is a simple text file where each line represents a key-value pair, and these pairs are used to configure various aspects of the application.

11. What are JWT?

JSON Web Tokens are mainly a token which is used for authentication and information exchange. When a user signs in to an application, the application then assigns JWT to that user. Subsequent requests by the user will include the assigned JWT. This token tells the server what routes, services, and resources the user is allowed to access. Json Web Token includes 3 part namely- Header, Payload and Signature.

12. Create a simple middleware for validating user.

```
// Simple user validation middleware
const validateUser = (req, res, next) => {
    const user = req.user;

    if (!user) {
        return res.status(401).json({ error: 'Unauthorized - User not found' });
    }

    next();
};

app.get('/profile', validateUser, (req, res) => {
    const user = req.user;
    res.json({ message: 'Profile page', username: user.username });
});
```

13. What is Bcrypt used for?

Bcrypt is a password hashing function which is used to securely hash and store user passwords. It is designed to be slow and computationally intensive, making it resistant to brute-force attacks and rainbow table attacks. Bcrypt is a key component in enhancing the security of user authentication systems.

14. Why should you separate the Express app and server?

In ExpressJS, it is recommended to separate the Express App and the server setup. This provides the modularity and flexibility and makes the codebase more easier to maintain and test. Here are some reasons why you should separate the Express app and server:

- **Modularity:** You can define routes, middleware, and other components in the Express app independently of the server configuration.
- **Ease of Testing:** Separation makes it easier to write unit tests for the Express app without starting an actual server. You can test routes, middleware, and other components in isolation.
- **Reusability:** You can reuse the same Express app in different server configurations.
- **Configuration Management:** Separating the app and server allows for cleaner configuration management.
- **Scalability:** It provides a foundation for a scalable code structure. As your application grows, it will easier to maintain the code.

15. What do you understand about ESLint

management.

- **Scalability:** It provides a foundation for a scalable code structure. As your application grows, it will easier to maintain the code.

15. What do you understand about ESLint?

EsLint is a JavaScript linting tool which is used for automatically detecting incorrect patterns found in ECMAScript/JavaScript code. It is used with the purpose of improving code quality, making code more consistent, and avoiding bugs. ESLint is written using NodeJS to provide a fast runtime environment and easy installation via npm.

16. Define the concept of the test pyramid.

The Test Pyramid is a concept in software testing that represents the distribution of different types of tests. It was introduced by Mike Cohn, and it suggests that a testing strategy should be shaped like a pyramid, with the majority of tests at the base and fewer tests as you move up. The Test Pyramid consists of three levels: Unit Tests, Integration Tests, and End-to-End (E2E) Tests.

17. Differentiate between res.send() and res.json().

Feature	res.send()	res.json()
Purpose	Sends a response of any type (string, object, array, buffer, etc.).	Specifically sends a JSON response.
Data Handling	Converts objects/arrays into JSON automatically, but also supports sending other data formats.	Converts objects/arrays into JSON format explicitly.
Response Type	Can send text, HTML, JSON, or any other data type.	Only sends JSON-formatted responses.
Use Case	Used when sending various types of responses, including HTML pages, strings, or JSON data.	Used specifically for sending JSON responses in APIs.
Example Usage	res.send('Hello World!')	res.json({ message: 'Success' })

ExpressJS Interview Questions and Answers - Intermediate Level

18. What is meant by Scaffolding in ExpressJS?

Scaffolding in ExpressJS refers to the process of generating a basic project structure automatically. This can speed up the initial setup and help maintain consistency in the way projects are structured, especially in large teams.

19. How would you install an Express application generator for scaffolding?

Express application generator are used for quickly setting up a new Express application with some basic structure. You can install it using Node Package Manager (npm), which comes with NodeJS.

To install it globally:

```
npm install -g express-generator
```

20. What is Yeoman and how to install Yeoman for scaffolding?

Yeoman is a scaffolding tool for web applications that helps developers to create new projects by providing a generator-based workflow.

To install Yeoman run the following command:

```
npm install -g yo
```

Yeoman works with generators, which are packages that define the structure and configuration of a project. You can install a generator like this:

```
npm install -g generator-express
```

Once installed, you can use Yeoman to create a new application:

```
yo express
```

```
npm install -g express-generator
```

20. What is Yeoman and how to install Yeoman for scaffolding?

Yeoman is a scaffolding tool for web applications that helps developers to create new projects by providing a generator-based workflow.

To install Yeoman run the following command:

```
npm install -g yo
```

Yeoman works with generators, which are packages that define the structure and configuration of a project. You can install a generator like this:

```
npm install -g generator-express
```

Once installed, you can use Yeoman to create a new application:

```
yo appname
```

21. What is CORS in ExpressJS?

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to control how web pages in one domain can request and interact with resources hosted on another domain.

In the context of ExpressJS, CORS refers to a middleware that enables Cross-Origin Resource Sharing for your application. This allows the application to control which domains can access your resources by setting HTTP headers.

22. What are Built-in Middlewares?

ExpressJS, includes a set of built-in middlewares that provide common functionality. These built-in middlewares are included by default when you create an Express application and can be used to handle various tasks. Here are some of the built-in middlewares in Express:

- **ExpressJSon():** This middleware is used to parse incoming JSON requests. It automatically parses the request body if the Content-Type header is set to application/json.
- **express.Router():** The express.Router() function is often used to create modular route handlers. It allows you to group route handlers together and then use them as a middleware.
- **express.static():** This middleware is used to serve static files, such as images, CSS, and JavaScript files, from a specified directory.

23. How would you configure properties in ExpressJS?

In ExpressJS, you can configure properties using the app.set() method. This method allows you to set various properties and options which affects the behavior of the Express application.

```
app.set(name, value);
```

Here, name represents the name of the property you want to configure, and value is the value you want to assign to that property. Express provides a wide range of properties that you can configure based on your application's requirements.

24. Which template engines do Express support?

ExpressJS supports any template engine that follows the (path, locals, callback) signature.

25. Elaborate on the various methods of debugging on both Linux and Windows systems?

The debugging is the vital need at the time of software development to identifying issues in the application's logic, handling of HTTP requests, middleware execution, and other aspects specific to web development. Here are some methods commonly used for debugging an ExpressJS application on both Linux and Windows:

- **Console.log:** The simplest way to debug an ExpressJS application is by using console.log(). You can output messages to the console which can be viewed in the terminal.
- **Node Inspector:** This is a powerful tool that allows you to debug your applications using Chrome Developer Tools. It supports features like setting breakpoints, stepping over functions, and inspecting variables.
- **Visual Studio Code Debugger:** VS Code provides a built-in debugger that works on both Linux and Windows. It supports advanced features like conditional breakpoints, function breakpoints, and logpoints.

ExpressJS supports any template engine that follows the (path, locals, callback) signature.

25. Elaborate on the various methods of debugging on both Linux and Windows systems?

The debugging is the vital need at the time of software development to identifying issues in the application's logic, handling of HTTP requests, middleware execution, and other aspects specific to web development. Here are some methods commonly used for debugging an ExpressJS application on both Linux and Windows:

- **Console.log:** The simplest way to debug an ExpressJS application is by using console.log(). You can output messages to the console which can be viewed in the terminal.
- **Node Inspector:** This is a powerful tool that allows you to debug your applications using Chrome Developer Tools. It supports features like setting breakpoints, stepping over functions, and inspecting variables.
- **Visual Studio Code Debugger:** VS Code provides a built-in debugger that works on both Linux and Windows. It supports advanced features like conditional breakpoints, function breakpoints, and logpoints.
- **Utilizing debug module:** The debug module is a small NodeJS debugging utility that allows you to create debugging scopes.

26. Name some databases that integrate with ExpressJS?

ExpressJS can support a variety of the databases which includes:

- MySQL
- MongoDB
- PostgreSQL
- SQLite
- Oracle

27. How would you render plain HTML using ExpressJS?

In ExpressJS, you can render plain HTML using the res.send() method or res.sendFile() method.

Sample code:

JavaScript JavaScript

```javascript
//using res.send

const express = require('express');
const app = express();
const port = 8000;

app.get('/', (req, res) => {
    const htmlContent = '<html><body><h1>Hello, World!</h1></body></html>';
    res.send(htmlContent);
});


app.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

28. What is the use of 'Response.cookie()' function?

The response.cookie() function in ExpressJS is used to set cookies in the HTTP response. Cookies are small pieces of data sent from a server and stored on the client's browser. They are commonly used to store information about the user or to maintain session data.

```
res.cookie(name, value, [options]);
```

29. Under what circumstances does a Cross-Origin resource fail in ExpressJS?

When a Cross-Origin Resource Sharing request is made, the browser enforces certain security checks, and the request may fail under various circumstances:

- **No CORS Headers:** The server doesn't include the necessary CORS headers in its response.
- **Mismatched Origin:** The requesting origin does not match the origin specified in the Access-Control-Allow-Origin header.
- **Restricted HTTP Methods:** The browser enforces restrictions on which HTTP methods are allowed in cross-origin requests.
- **No Credentials:** The browser makes restrictions on requests that include credentials (such as cookies or HTTP authentication).

30. What is Pug template engine in ExpressJS?

Pug is a popular template engine for ExpressJS and other NodeJS frameworks. You can use Pug to

cookies or HTTP authentication).

30. What is Pug template engine in ExpressJS?

Pug is a popular template engine for ExpressJS and other NodeJS frameworks. You can use Pug to render dynamic HTML pages on the server side. It allows you to write templates using a syntax that relies on indentation and concise tags.

31. What is meant by the sanitizing input process in ExpressJS?

Sanitizing input in ExpressJS application is an important security practice to prevent various types of attacks, such as Cross-Site Scripting (XSS) and SQL injection. It involves cleaning and validating user input before using it in your application so that it does not contain malicious code or can be a security risk.

32. How to generating a skeleton ExpressJS app using terminal command?

To generate a skeleton for an ExpressJS application using the terminal, you can use the Express application generator which is a command-line tool provided by the ExpressJS framework. This generator will setup a basic directory structure which includes necessary files, and installs essential dependencies.

Steps to generate:

Step 1: Open your terminal and install the Express application generator globally using the following command:

```
npm install -g express-generator
```

Step 2: After that you can use the express command to generate your ExpressJS app.

```
express my-express-app
```

Step 3: Now go to the app directory and install the dependencies and start the app by running-

```
npm install
npm start
```

33. What are middlewares in ExpressJS?

Middleware functions are those functions that have the access to request and response object and the next middleware or function. They can add functionality to an application, such as logging, authentication, and error handling.

ExpressJS Interview Questions and Answers - Advanced Level

34. What are the types of middlewares?

There are mainly five types of Middleware in ExpressJS:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

35. List the built-in middleware functions provided by Express.

ExpressJS comes with several built-in middleware functions. Few of them are:

- **ExpressJSON:** This is used for parsing incoming requests with JSON payloads.
- **express.static:** This is used to serve static files like images, CSS files, and JavaScript files.
- **express.urlencoded:** This is used for parsing incoming requests with URL-encoded payloads.
- **express.raw**: This is used for parse incoming requests with a raw body.
- **express.text**: This is used for parse incoming requests with a text body.

36. Mention some third-party middleware provided by ExpressJS.

ExpressJS allows you to use third-party middleware to extend and enhance the functionality of your web application.

Here are some commonly used third-party middleware in ExpressJS

- **body-parser:** This middleware is used to parse incoming request bodies, allowing you to access form data or JSON payloads on req.body.
- **cors:** This module provides middleware to enable Cross-Origin Resource Sharing (CORS) in your Express application.

36. Mention some third-party middleware provided by ExpressJS.

ExpressJS allows you to use third-party middleware to extend and enhance the functionality of your web application.

Here are some commonly used third-party middleware in ExpressJS

- **body-parser:** This middleware is used to parse incoming request bodies, allowing you to access form data or JSON payloads on req.body.
- **cors:** This module provides middleware to enable Cross-Origin Resource Sharing (CORS) in your Express application.
- **morgan:** Morgan is a middleware module that provides request logging functionality.
- **helmet:** Helmet helps to secure Express apps by setting various HTTP headers.
- **express-session:** This middleware is used for managing user sessions in your Express application.
- **passport:** This middleware is used for implementing authentication and authorization in Express applications.

37. When application-level Middleware is used?

Application-level middlewares are bound to an instance of the Express application and are executed for every incoming request. These middlewares are defined using the app.use() method, and they can perform tasks such as logging, authentication, setting global variables, and more.

38. Explain Router-level Middleware.

Router-level middlewares are specific to a particular router instance. This type of middleware is bound to an instance of express.Router(). Router-level middleware works similarly to application-level middleware, but it's only invoked for the routes that are handled by that router instance. This allows you to apply middleware to specific subsets of your routes, keeping your application organized and manageable.

39. How to secure ExpressJS application?

It is very important to secure your application to protect it against various security threats. We can follow few best practices in our ExpressJS app to enhance the security of our application.

- Keep Dependencies Updated: Regularly update your project dependencies, including ExpressJS and other npm packages.
- Use Helmet Middleware: The helmet middleware helps secure your application by setting various HTTP headers. It helps prevent common web vulnerabilities.
- Set Secure HTTP Headers: Configure your application to include secure HTTP headers, such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), and others.
- Use HTTPS: Always use HTTPS to encrypt data in transit. Obtain an SSL certificate for your domain and configure your server to use HTTPS.
- Secure Database Access: Use parameterized queries or prepared statements to prevent SQL injection attacks. Ensure that your database credentials are secure and not exposed in configuration files.

40. What is Express router() function?

The express.Router() function is used to create a new router object. This function is used when you want to create a new router object in your program to handle requests.

```
express.Router( [options] )
```

41. What are the different types of HTTP requests?

The primary HTTP methods are commonly referred to as CRUD operations, representing Create, Read, Update, and Delete. Here are the main HTTP methods:

- **GET:** The GET method is used to request data from a specified resource.
- **POST**: The POST method is used to submit data to be processed to a specified resource.
- **PUT:** The PUT method is used to update a resource or create a new resource if it does not exist.
- **PATCH**: The PATCH method is used to apply partial modifications to a resource.
- **DELETE:** The DELETE method is used to request that a specified resource be removed.

42. Do Other MVC frameworks also support scaffolding?

The Scaffolding technique is supported by other MVC frameworks also which includes- Ruby on Rails, OutSystems Platform, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, YII, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET, etc.

- **POST**: The POST method is used to submit data to be processed to a specified resource.
- **PUT:** The PUT method is used to update a resource or create a new resource if it does not exist.
- **PATCH**: The PATCH method is used to apply partial modifications to a resource.
- **DELETE:** The DELETE method is used to request that a specified resource be removed.

42. Do Other MVC frameworks also support scaffolding?

The Scaffolding technique is supported by other MVC frameworks also which includes- Ruby on Rails, OutSystems Platform, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, YII, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET, etc.

43. Which are the arguments available to an ExpressJS route handler function?

In ExpressJS route handler function, there are mainly3 arguments available that provide useful information and functionality.

- **req:** This represents the HTTP request object which holds information about the incoming request. It allows you to access and manipulate the request data.
- **res:** This represents the HTTP response object which is used to send the response back to the client. It provides methods and properties to set response headers, status codes, and send the response body.
- **next**: This is a callback function that is used to pass control to the next middleware function in the request-response cycle.

44. How can you deal with error handling in ExpressJS?

ExpressJS provides built-in error-handling mechanism with the help of the next() function. When an error occurs, you can pass it to the next middleware or route handler using the next() function. You can also add an error-handling middleware to your application that will be executed whenever an error occurs.

45. What is the difference between a traditional server and an ExpressJS server?

Feature	Traditional Server (PHP, Java, .NET)	ExpressJS Server (NodeJS)
Language	Uses languages like PHP, Java, C#, Python.	Uses JavaScript (NodeJS).
Architecture	Multi-threaded, blocking I/O.	Single-threaded, non-blocking I/O.
Performance	Slower due to thread-based handling.	Faster due to event-driven, async processing.
Routing Mechanism	Routing is predefined and handled differently for each language.	ExpressJS provides a built-in and flexible routing system.
Used For	Enterprise applications, legacy systems, large-scale applications.	APIs, SPAs, microservices, real-time applications.

46. What is the purpose of the next() function in ExpressJS?

The next() function is used to pass control from one middleware function to the next function. It is used to execute the next middleware function in the chain. If there are no next middleware function in the chain then it will give control to router or other functions in the app. If you don't call next() in a middleware function, the request-response cycle can be terminated, and subsequent middleware functions won't be executed.

47. What is the difference between app.route() and app.use() in ExpressJS?

Feature	app.route()	app.use()
Purpose	Defines multiple HTTP methods (GET, POST, PUT, etc.) for a single route.	Mounts middleware or routers to handle requests.
Middleware Support	Does not apply middleware; only handles route-specific logic.	Used to apply middleware functions like authentication, logging, or parsing request bodies.
Routing Scope	Specific to a single route.	Can apply to multiple routes or all requests.

a middleware function, the request-response cycle can be terminated, and subsequent middleware functions won't be executed.

47. What is the difference between app.route() and app.use() in ExpressJS?

Feature	app.route()	app.use()
Purpose	Defines multiple HTTP methods (GET, POST, PUT, etc.) for a single route.	Mounts middleware or routers to handle requests.
Middleware Support	Does not apply middleware; only handles route-specific logic.	Used to apply middleware functions like authentication, logging, or parsing request bodies.
Routing Scope	Specific to a single route.	Can apply to multiple routes or all requests.
Example Usage	javascript app.route('/user') .get((req, res) => res.send('GET User')) .post((req, res) => res.send('POST User')) .put((req, res) => res.send('PUT User'));	javascript app.use('/user', (req, res, next) => { console.log('Middleware for /user'); next(); });
Used For	When multiple HTTP methods need to be handled for the same path.	When applying middleware globally or to a group of routes.

48. Explain what dynamic routing is in ExpressJS.

Dynamic routing in ExpressJS include parameters, which allows you to create flexible and dynamic routes in your web application. This parameters are used in your route handlers to customize the behaviour based on the data provided.

In Express, dynamic routing is achieved by using route parameters, denoted by a colon (:) followed by the parameter name.

Here's a simple example:

```
const express = require('express');
const app = express();

// Dynamic route with a parameter
app.get('/users/:userId', (req, res) => {
    const userId = req.params.userId;
    res.send(`User ID: ${userId}`);
});

// Start the server
const port = 8000;
app.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

49. How to serve static files in ExpressJS?

In ExpressJS, you can serve static files using the built-in express.static middleware. This middleware function takes the root directory of your static files as an argument and serves them automatically.

50. What is the use of app.use() in ExpressJS?

app.use() is used to add middleware functions in an Express application. It can be used to add global middleware functions or to add middleware functions to specific routes.

48. Explain what dynamic routing is in ExpressJS.

Dynamic routing in ExpressJS include parameters, which allows you to create flexible and dynamic routes in your web application. This parameters are used in your route handlers to customize the behaviour based on the data provided.

In Express, dynamic routing is achieved by using route parameters, denoted by a colon (:) followed by the parameter name.

Here's a simple example:

```
const express = require('express');
const app = express();

// Dynamic route with a parameter
app.get('/users/:userId', (req, res) => {
    const userId = req.params.userId;
    res.send(`User ID: ${userId}`);
});

// Start the server
const port = 8000;
app.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

49. How to serve static files in ExpressJS?

In ExpressJS, you can serve static files using the built-in express.static middleware. This middleware function takes the root directory of your static files as an argument and serves them automatically.

50. What is the use of app.use() in ExpressJS?

app.use() is used to add middleware functions in an Express application. It can be used to add global middleware functions or to add middleware functions to specific routes.

Next Article
Introduction to Express

Similar Reads

1. NodeJS Interview Questions and Answers
2. Express JS Exercises, Practice Questions and Solutions
3. Backend Developer Interview Questions
4. 10 Most Asked ES6 Interview Questions & Answers For Developers
5. Node Interview Questions and Answers (2025) - Advanced Level
6. MERN Stack Interview Questions
7. NodeJS Interview Experience
8. Fasal Interview Experience for Product Engineering Internship (6 Months) | Off-Campus 2021
9. Inara Consultancy Services Interview Experience for SDE
10. Why Express Is Used For Enterprise App Development ?

to this source.

Beginner Node.js Interview Questions

1. What is Node.js and how it works?

Node.js is a virtual machine that uses JavaScript as its scripting language and runs Chrome's V8 JavaScript engine. Basically, Node.js is based on an event-driven architecture where I/O runs asynchronously making it lightweight and efficient. It is being used in developing desktop applications as well with a popular framework called electron as it provides API to access OS-level features such as file system, network, etc.

Here is a Free course on Node.js for beginners to master the fundamentals of Node.js.

2. What tools can be used to assure consistent code style?

ESLint can be used with any IDE to ensure a consistent coding style which further helps in maintaining the codebase.

3. What is a first class function in Javascript?

When functions can be treated like any other variable then those functions are first-class functions. There are many other programming languages, for example, scala, Haskell, etc which follow this including JS. Now because of this function can be passed as a param to another function(callback) or a function can return another function(higher-order function). map() and filter() are higher-order functions that are popularly used.

4. How do you manage packages in your node.js project?

It can be managed by a number of package installers and their configuration file accordingly. Out of them mostly use npm or yarn. Both provide almost all libraries of javascript with extended features of controlling environment-specific configurations. To maintain versions of libs being installed in

4. How do you manage packages in your node.js project?

It can be managed by a number of package installers and their configuration file accordingly. Out of them mostly use npm or yarn. Both provide almost all libraries of javascript with extended features of controlling environment-specific configurations. To maintain versions of libs being installed in a project we use package.json and package-lock.json so that there is no issue in porting that app to a different environment.

5. How is Node.js better than other frameworks most popularly used?

- Node.js provides simplicity in development because of its non-blocking I/O and event-based model results in short response time and concurrent processing, unlike other frameworks where developers have to use thread management.
- It runs on a chrome v8 engine which is written in c++ and is highly performant with constant improvement.
- Also since we will use Javascript in both the frontend and backend the development will be much faster.
- And at last, there are sample libraries so that we don't need to reinvent the wheel.

6. Explain the steps how "Control Flow" controls the functions calls?

- Control the order of execution
- Collect data
- Limit concurrency
- Call the following step in the program.

7. What are some commonly used timing features of Node.js?

- **setTimeout/clearTimeout** – This is used to implement delays in code execution.
- **setInterval/clearInterval** – This is used to run a code block multiple times.
- **setImmediate/clearImmediate** – Any function passed as the setImmediate() argument is a callback that's executed in the next iteration of the event loop.
- **process.nextTicl**

- Call the following step in the program.

7. What are some commonly used timing features of Node.js?

- **setTimeout/clearTimeout** – This is used to implement delays in code execution.
- **setInterval/clearInterval** – This is used to run a code block multiple times.
- **setImmediate/clearImmediate** – Any function passed as the setImmediate() argument is a callback that's executed in the next iteration of the event loop.
- **process.nextTick** – Both setImmediate and process.nextTick appear to be doing the same thing; however, you may prefer one over the other depending on your callback's urgency.

8. What are the advantages of using promises instead of callbacks?

The main advantage of using promise is you get an object to decide the action that needs to be taken after the async task completes. This gives more manageable code and avoids callback hell.

9. What is fork in node JS?

A fork in general is used to spawn child processes. In node it is used to create a new instance of v8 engine to run multiple workers to execute the code.

10. Why is Node.js single-threaded?

Node.js was created explicitly as an experiment in async processing. This was to try a new theory of doing async processing on a single thread over the existing thread-based implementation of scaling via different frameworks.

11. How do you create a simple server in Node.js that returns Hello World?

```
var http = require("http");
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);
```

12. How many types of API functions are there in Node.js?

There are two types of API functions:

- **Asynchronous, r**
  the main loop.

```
  response.end('Hello World\n');
}).listen(3000);
```

12. How many types of API functions are there in Node.js?

There are two types of API functions:

- **Asynchronous, non-blocking functions** - mostly I/O operations which can be fork out of the main loop.
- **Synchronous, blocking functions** - mostly operations that influence the process running in the main loop.

13. What is REPL?

PL in Node.js stands for **R**ead, **E**val, **P**rint, and **L**oop, which further means evaluating code on the go.

14. List down the two arguments that async.queue takes as input?

- Task Function
- Concurrency Value

15. What is the purpose of module.exports?

This is used to expose functions of a particular module or file to be used elsewhere in the project. This can be used to encapsulate all similar functions in a file which further improves the project structure.

For example, you have a file for all utils functions with util to get solutions in a different programming language of a problem statement.

```
const getSolutionInJavaScript = async ({
 problem_id
}) => {
...
};
const getSolutionInPython = async ({
 problem_id
}) => {
...
};
module.exports = { getSolutionInJavaScript, getSolutionInPython }
```

Thus using module.exports we can use these functions in some other file:

```
const { getSolutionInJavaScript, getSolutionInPython} = require("./utils")
```

Intermediate Node.js Interview Questions

1. Explain the concept of stub in Node.js?

Stubs are used in writing tests which are an important part of development. It replaces the whole function whic

Intermediate Node.js Interview Questions

1. Explain the concept of stub in Node.js?

Stubs are used in writing tests which are an important part of development. It replaces the whole function which is getting tested.

This helps in scenarios where we need to test:

- External calls which make tests slow and difficult to write (e.g HTTP calls/ DB calls)
- Triggering different outcomes for a piece of code (e.g. what happens if an error is thrown/ if it passes)

For example, this is the function:

```
const request = require('request');
const getPhotosByAlbumId = (id) => {
const requestUrl = `https://jsonplaceholder.typicode.com/albums/${id}/photos?_limit=3`;
return new Promise((resolve, reject) => {
  request.get(requestUrl, (err, res, body) => {
    if (err) {
      return reject(err);
    }
    resolve(JSON.parse(body));
  });
});
};
module.exports = getPhotosByAlbumId;
To test this function this is the stub
const expect = require('chai').expect;
const request = require('request');
const sinon = require('sinon');
const getPhotosByAlbumId = require('./index');
describe('with Stub: getPhotosByAlbumId', () => {
before(() => {
  sinon.stub(request, 'get')
    .yields(null, null, JSON.stringify([
      {
        "albumId": 1,
        "id": 1,
        "title": "A real photo 1",
        "url": "https://via.placeholder.com/600/92c952",
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
      },
      {
        "albumId": 1,
        "id": 2,
        "title": "A real photo 2",
        "url": "https://via.placeholder.com/600/771796",
        "thumbnailUrl": "https://via.placeholder.com/150/771796"
      },
      {
        "albumId": 1,
        "id": 3,
        "title": "A real photo 3",
        "url": "https://via.placeholder.com/600/24f355",
        "thumbnailUrl": "https://via.placeholder.com/150/24f355"
      }
    ]));
});
after(() => {
  request.get.restore();
});
it('should getPhotosByAlbumId', (done) => {
  getPhotosByAlbumId(1).then((photos) => {
    expect(photos.length).to.equal(3);
    photos.forEach(photo => {
      expect(photo).to.have.property('id');
      expect(photo).to.have.property('title');
      expect(photo).t
    });
```

```
            "id": 2,
            "title": "A real photo 2",
            "url": "https://via.placeholder.com/600/771796",
            "thumbnailUrl": "https://via.placeholder.com/150/771796"
          },
          {
            "albumId": 1,
            "id": 3,
            "title": "A real photo 3",
            "url": "https://via.placeholder.com/600/24f355",
            "thumbnailUrl": "https://via.placeholder.com/150/24f355"
          }
        ]));
});
after(() => {
  request.get.restore();
});
it('should getPhotosByAlbumId', (done) => {
  getPhotosByAlbumId(1).then((photos) => {
    expect(photos.length).to.equal(3);
    photos.forEach(photo => {
      expect(photo).to.have.property('id');
      expect(photo).to.have.property('title');
      expect(photo).to.have.property('url');
    });
    done();
  });
});
});
```

2. Describe the exit codes of Node.js?

Exit codes give us an idea of how a process got terminated/the reason behind termination.

A few of them are:

- Uncaught fatal exception - (code - 1) - There has been an exception that is not handled
- Unused - (code - 2) - This is reserved by bash
- Fatal Error - (code - 5) - There has been an error in V8 with stderr output of the description
- Internal Exception handler Run-time failure - (code - 7) - There has been an exception when bootstrapping function was called
- Internal JavaScript Evaluation Failure - (code - 4) - There has been an exception when the bootstrapping process failed to return function value when evaluated.

3. For Node.js, why Google uses V8 engine?

Well, are there any other options available? Yes, of course, we have Spidermonkey from Firefox, Chakra from Edge but Google's v8 is the most evolved(since it's open-source so there's a huge community helping in developing features and fixing bugs) and fastest(since it's written in c++) we got till now as a JavaScript and WebAssembly engine. And it is portable to almost every machine known.

4. Why should you separate Express app and server?

The server is responsible for initializing the routes, middleware, and other application logic whereas the app has all the business logic which will be served by the routes initiated by the server. This ensures that the business logic is encapsulated and decoupled from the application logic which makes the project more readable and maintainable.

5. Explain what a Reactor Pattern in Node.js?

Reactor pattern again a pattern for nonblocking I/O operations. But in general, this is used in any event-driven architecture.

There are two compo

server. This ensures that the business logic is encapsulated and decoupled from the application logic which makes the project more readable and maintainable.

5. Explain what a Reactor Pattern in Node.js?

Reactor pattern again a pattern for nonblocking I/O operations. But in general, this is used in any event-driven architecture.

There are two components in this: 1. Reactor 2. Handler.

Reactor: Its job is to dispatch the I/O event to appropriate handlers
Handler: Its job is to actually work on those events

6. What is middleware?

Middleware comes in between your request and business logic. It is mainly used to capture logs and enable rate limit, routing, authentication, basically whatever that is not a part of business logic. There are third-party middleware also such as body-parser and you can write your own middleware for a specific use case.

7. What are node.js buffers?

In general, buffers is a temporary memory that is mainly used by stream to hold on to some data until consumed. Buffers are introduced with additional use cases than JavaScript's Unit8Array and are mainly used to represent a fixed-length sequence of bytes. This also supports legacy encodings like ASCII, utf-8, etc. It is a fixed(non-resizable) allocated memory outside the v8.

8. What is node.js streams?

Streams are instances of EventEmitter which can be used to work with streaming data in Node.js. They can be used for handling and manipulating streaming large files(videos, mp3, etc) over the network. They use buffers as their temporary storage.

There are mainly four types of the stream:

- **Writable:** streams to which data can be written (for example, fs.createWriteStream()).
- **Readable:** streams from which data can be read (for example, fs.createReadStream()).
- **Duplex:** streams that are both Readable and Writable (for example, net.Socket).
- **Transform:** Duplex streams that can modify or transform the data as it is written and read (for example, zlib.createDeflate()).

9. How can we use async await in node.js?

Here is an example of using async-await pattern:

```
// this code is to retry with exponential backoff
function wait (timeout) {
 return new Promise((resolve) => {
setTimeout(() => {
  resolve()
}, timeout);
 });
}
async function requestWithRetry (url) {
 const MAX_RETRIES = 10;
 for (let i = 0; i <= MAX_RETRIES; i++) {
try {
  return await request(url);
} catch (err) {
  const timeout = Math.pow(2, i);
  console.log('Waiting', timeout, 'ms');
  await wait(timeout);
  console.log('Retrying', err.message, i);
}
```

9. How can we use async await in node.js?

Here is an example of using async-await pattern:

```
// this code is to retry with exponential backoff
function wait (timeout) {
 return new Promise((resolve) => {
setTimeout(() => {
  resolve()
}, timeout);
 });
}
async function requestWithRetry (url) {
 const MAX_RETRIES = 10;
 for (let i = 0; i <= MAX_RETRIES; i++) {
try {
  return await request(url);
} catch (err) {
  const timeout = Math.pow(2, i);
  console.log('Waiting', timeout, 'ms');
  await wait(timeout);
  console.log('Retrying', err.message, i);
}
 }
}
```

10. How does Node.js overcome the problem of blocking of I/O operations?

Since the node has an event loop that can be used to handle all the I/O operations in an asynchronous manner without blocking the main function.

So for example, if some network call needs to happen it will be scheduled in the event loop instead of the main thread(single thread). And if there are multiple such I/O calls each one will be queued accordingly to be executed separately(other than the main thread).

Thus even though we have single-threaded JS, I/O ops are handled in a nonblocking way.

11. Differentiate between process.nextTick() and setImmediate()?

Both can be used to switch to an asynchronous mode of operation by listener functions.

process.nextTick() sets the callback to execute but setImmediate pushes the callback in the queue to be executed. So the event loop runs in the following manner

timers->pending callbacks->idle,prepare->connections(poll,data,etc)->check->close callbacks

In this process.nextTick() method adds the callback function to the start of the next event queue and setImmediate() method to place the function in the check phase of the next event queue.

12. If Node.js is single threaded then how does it handle concurrency?

The main loop is single-threaded and all async calls are managed by libuv library.

For example:

```
const crypto = require("crypto");
const start = Date.now();
function logHashTime() {
 crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
console.log("Hash: ", Date.now() - start);
 });
}
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

Thus even though we have single-threaded JS, I/O ops are handled in a nonblocking way.

11. Differentiate between process.nextTick() and setImmediate()?

Both can be used to switch to an asynchronous mode of operation by listener functions.

process.nextTick() sets the callback to execute but setImmediate pushes the callback in the queue to be executed. So the event loop runs in the following manner

timers->pending callbacks->idle,prepare->connections(poll,data,etc)->check->close callbacks

In this process.nextTick() method adds the callback function to the start of the next event queue and setImmediate() method to place the function in the check phase of the next event queue.

12. If Node.js is single threaded then how does it handle concurrency?

The main loop is single-threaded and all async calls are managed by libuv library.

For example:

```
const crypto = require("crypto");
const start = Date.now();
function logHashTime() {
 crypto.pbkdf2("a", "b", 100000, 512, "sha512", () => {
console.log("Hash: ", Date.now() - start);
 });
}
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

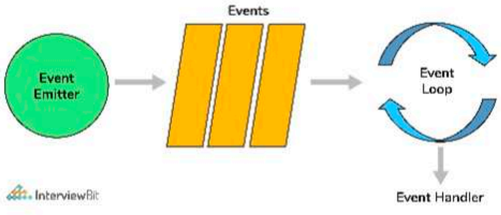
This gives the output:

```
Hash: 1213
Hash: 1225
Hash: 1212
Hash: 1222
```

This is because libuv sets up a thread pool to handle such concurrency. How many threads will be there in the thread pool depends upon the number of cores but you can override this.

13. What is an event-loop in Node JS?

Whatever that is async is managed by event-loop using a queue and listener. We can get the idea using the following diagram:



Node.js Event Loop

So when an async function needs to be executed(or I/O) the main thread sends it to a different thread allowing v8 to keep executing the main code. Event loop involves different phases with specific tasks such as timers, pending callbacks, idle or prepare, poll, check, close callbacks with different FIFO queues. Also in between iterations it checks for async I/O or timers and shuts down cleanly if there aren't any.

14. What do you

thread allowing v8 to keep executing the main code. Event loop involves different phases with specific tasks such as timers, pending callbacks, idle or prepare, poll, check, close callbacks with different FIFO queues. Also in between iterations it checks for async I/O or timers and shuts down cleanly if there aren't any.

14. What do you understand by callback hell?

```
async_A(function(){
  async_B(function(){
    async_C(function(){
      async_D(function(){
      ....
      });
    });
  });
});
```

For the above example, we are passing callback functions and it makes the code unreadable and not maintainable, thus we should change the async logic to avoid this.

Advanced Node.js Interview Questions

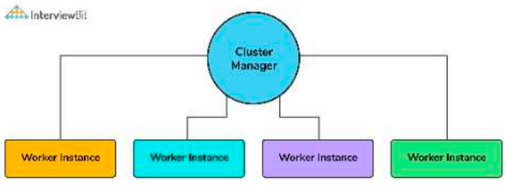
1. What is an Event Emitter in Node.js?

EventEmitter is a Node.js class that includes all the objects that are basically capable of emitting events. This can be done by attaching named events that are emitted by the object using an eventEmitter.on() function. Thus whenever this object throws an even the attached functions are invoked synchronously.

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
 console.log('an event occurred!');
});
myEmitter.emit('event');
```

2. Enhancing Node.js performance through clustering.

Node.js applications run on a single processor, which means that by default they don't take advantage of a multiple-core system. Cluster mode is used to start up multiple node.js processes thereby having multiple instances of the event loop. When we start using cluster in a nodejs app behind the scene multiple node.js processes are created but there is also a parent process called the **cluster manager** which is responsible for monitoring the health of the individual instances of our application.



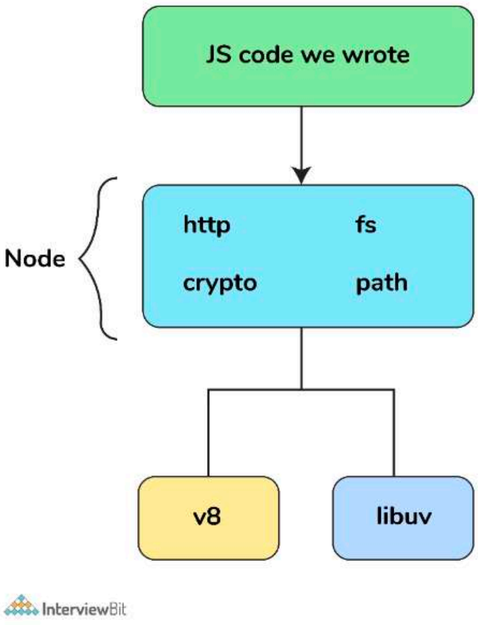
Clustering in Node.js

3. What is a thread pool and which library handles it in Node.js

The Thread pool is handled by the libuv library. libuv is a multi-platform C library that provides support for asynchronous I/O-based operations such as file systems, networking, and concurrency.

3. What is a thread pool and which library handles it in Node.js

The Thread pool is handled by the libuv library. libuv is a multi-platform C library that provides support for asynchronous I/O-based operations such as file systems, networking, and concurrency.



Thread Pool

4. What is WASI and why is it being introduced?

Web assembly provides an implementation of WebAssembly System Interface specification through WASI API in node.js implemented using WASI class. The introduction of WASI was done by keeping in mind its possible to use the underlying operating system via a collection of POSIX-like functions thus further enabling the application to use resources more efficiently and features that require system-level access.

5. How are worker threads different from clusters?

Cluster:

- There is one process on each CPU with an IPC to communicate.
- In case we want to have multiple servers accepting HTTP requests via a single port, clusters can be helpful.
- The processes are spawned in each CPU thus will have separate memory and node instance which further will lead to memory issues.

Worker threads:

- There is only one process in total with multiple threads.
- Each thread has one Node instance (one event loop, one JS engine) with most of the APIs accessible.
- Shares memory with other threads (e.g. SharedArrayBuffer)
- This can be used for CPU-intensive tasks like processing data or accessing the file system since NodeJS is single-threaded, synchronous tasks can be made more efficient leveraging the worker's threads.

6. How to measure the duration of async operations?

Performance API provides us with tools to figure out the necessary performance metrics. A simple example woul

since NodeJS is single-threaded, synchronous tasks can be made more efficient leveraging the worker's threads.

6. How to measure the duration of async operations?

Performance API provides us with tools to figure out the necessary performance metrics. A simple example would be using async_hooks and perf_hooks

```
'use strict';
const async_hooks = require('async_hooks');
const {
 performance,
 PerformanceObserver
} = require('perf_hooks');
const set = new Set();
const hook = async_hooks.createHook({
 init(id, type) {
if (type === 'Timeout') {
  performance.mark(`Timeout-${id}-Init`);
  set.add(id);
}
 },
 destroy(id) {
if (set.has(id)) {
  set.delete(id);
  performance.mark(`Timeout-${id}-Destroy`);
  performance.measure(`Timeout-${id}`,
                      `Timeout-${id}-Init`,
                      `Timeout-${id}-Destroy`);
}
 }
});
hook.enable();
const obs = new PerformanceObserver((list, observer) => {
 console.log(list.getEntries()[0]);
 performance.clearMarks();
 observer.disconnect();
});
obs.observe({ entryTypes: ['measure'], buffered: true });
setTimeout(() => {}, 1000);
```

This would give us the exact time it took to execute the callback.

7. How to measure the performance of async operations?

Performance API provides us with tools to figure out the necessary performance metrics.

A simple example would be:

```
const { PerformanceObserver, performance } = require('perf_hooks');
const obs = new PerformanceObserver((items) => {
 console.log(items.getEntries()[0].duration);
 performance.clearMarks();
});
obs.observe({ entryTypes: ['measure'] });
performance.measure('Start to Now');
performance.mark('A');
doSomeLongRunningProcess(() => {
 performance.measure('A to Now', 'A');
 performance.mark('B');
 performance.measure('A to B', 'A', 'B');
});
```

Additional Useful Resources

1. NodeJS MCQ
2. Top 10 Node JS