Topic progress: 16%

devinterview

Sign in with Google

★★★★★

"Huge timesaver. Worth the money"

★★★★★

"It's an excellent tool"

★★★★★

"Fantastic catalogue of questions"

your next tech interview with

Explore our carefully curated catalog of interview essentials covering full-stack, data structures and algorithms, system design, data science, and machine learning interview questions

Start preparing now

Web & Mobile Dev | Data Structures & Algorithms | System Design | Machine Learning & Data Science

ADO.NET · Agile & Scrum · Android · Angular · AngularJS · ASP.NET · ASP.NET MVC · ASP.NET Web API · AWS · Azure · C++ · C# · Cosmos DB · CSS · Dependency Injection · DevOps · Django · Entity Framework · Express.js · Flutter · GIT · Golang · GraphQL · HTML5 · Ionic · Java · JavaScript · jQuery · Kotlin · Laravel · LINQ · MongoDB · .NET Core · Next.js · Node.js · Objective-C · OOP · PHP · PWA · Python · React · React Native · Reactive Programming · Reactive Systems · Redis · Redux

Go up

58 Express.js interview questions

Only coding challenges ○ Reset progress Search...

Express.js Fundamentals

1. What is **Express.js**, and how does it relate to **Node.js**? Question ✓

Answer:

Express.js is a web application framework that runs on **Node.js**. It simplifies the process of building web applications and APIs by providing a range of powerful features, including robust routing, middleware support, and HTTP utility methods. Thanks to its modular design, you can expand its functionality through additional libraries and Node.js modules.

Key Features

- **Middleware**: Express.js makes use of middleware functions that have access to the request-response cycle. This allows for a variety of operations such as logging, authentication, and data parsing.
- **Routing**: The framework offers a flexible and intuitive routing system, making it easy to handle different HTTP request methods on various URLs.
- **Templates**: Integrated support for template engines enables the dynamic rendering of HTML content.
- **HTTP Methods**: It provides built-in methods for all HTTP requests, such as `get`, `post`, `put`, `delete`, simplifying request handling.
- **Error Handling**: Express streamlines error management, and its middleware functions can specifically handle errors.
- **RESTful APIs**: Its features such as request and response object chaining, along with HTTP method support, make it ideal for creating RESTful APIs.

Go up

Relationship with Node.js

of Node.js for web development. Node.js, on the other hand, is a cross-platform JavaScript runtime environment that allows developers to build server-side and networking applications.

Express.js accomplishes this through a layer of abstractions and a more structured approach, which Node.js, by itself, doesn't provide out of the box.

Code Example: Basic Express Server

Here is the Node.js code:

```
// Import required modules
const express = require('express');

// Create an Express application
const app = express();
const port = 3000;

// Define a route and its callback function
app.get('/', (req, res) => {
```

Click to expand

2. Explain the concept of **middleware** in **Express.js**. Question ✓
3. How would you set up a basic **Express.js** application? Question ✓
4. What is the purpose of the `app.use()` function? Question ✓
5. How do you serve **static files** using **Express.js**? Question ✓
6. Discuss the difference between `app.get()` and `app.post()` in **Express.js**. Question ○
7. How do you retrieve the **URL parameters** from a **GET request** in **Express.js**? Question ✓
8. What are **route handlers**, and how would you implement them? Question ○
9. How do you enable **CORS** in an **Express.js** application? Question ○
10. Explain the use of `next()` in **Express.js middleware**. Question ✓

Routing and Requests

11. What is the role of the `express.Router` class? Question ○
12. How do you handle **404 errors** in **Express.js**? Question ✓
13. What are the differences between `req.query` and `req.params`? Question ○

Go up

Topic progress: 16%

15. How do you create a **middleware** that logs the **request method** and **URL** for every request? Question ✓

16. Explain how you would implement **nested routes** in **Express.js**. 🔒 Question ○

17. How can you **capture** and **respond** to **URL parameters** in a route? 🔒 Question ○

18. How do you serve different **content types** (e.g., JSON, HTML) with **Express.js responses**? 🔒 Question ○

19. What are best practices for structuring a large **Express.js** application with multiple routes? 🔒 Question ○

Middleware and Error Handling

20. Explain the concept and use of **built-in middleware** in **Express.js**. 🔒 Question ○

21. How do you write **custom middleware functions** in **Express.js**? 🔒 Question ○

22. How do you handle **file uploads** in **Express.js**? 🔒 Question ○

23. What is `express-session`, and how would you use it? 🔒 Question ○

24. Discuss **error handling** in an **Express.js** application. How do you define an **error-handling middleware**? 🔒 Question ○

25. Provide an example of using **third-party middleware**, such as `body-parser` or `morgan`. 🔒 Question ○

26. How do you protect against **SQL injection** or other **security threats** in **Express.js**? 🔒 Question ○

Response and Performance

27. How would you implement **caching** in an **Express.js** application? 🔒 Question ○

28. How do you **set cookies** and **get cookies** in an **Express.js** application? 🔒 Question ○

29. What are ways to **improve the performance** of **Express.js** applications? 🔒 Question ○

30. How do you **configure** an Express.js app for a **reverse proxy**, like **Nginx**? 🔒 Question ○

31. Explain the purpose of **template engines**. How would you integrate one with **Express.js**? 🔒 Question ○

Testing and Debugging

32. How do you test an **Express.js** application? 🔒 Question ○

33. Discuss common **debugging techniques** for **Express.js** applications. 🔒 Question ○

Go up

Topic progress: 16%

35. How do you use a **debugger** with an **Express.js** app running in **Node.js**? Question ○

Express.js with Databases

36. How would you connect a **MongoDB database** with an **Express.js** application? Question ○
37. Explain how to integrate an **ORM** like **Sequelize** with **Express.js**. Question ○
38. How do you handle **database errors** in **Express.js**? Question ○
39. What are the advantages of using a **database pooling mechanism** in an **Express.js** app? Question ○

Authentication and Authorization

40. Describe how you would implement **user authentication** in **Express.js**. Question ○
41. Explain how **sessions** are managed in **Express.js**. Question ○
42. Discuss how you would handle **user roles and permissions** in an **Express.js** application. Question ○
43. How can you secure **passwords** and **sensitive information** in your **Express.js** app? Question ○

Integration and Configuration

44. How do you integrate a **third-party API** in an **Express.js** application? Question ○
45. Explain the steps to **deploy** an **Express.js** application to a cloud provider like **AWS** or **Heroku**. Question ○
46. How can you ensure that your **Express.js** application is scalable_? Question ○
47. What is the **Twelve-Factor App methodology**, and how does it apply to **Express.js**? Question ○

Coding Challenges

48. Write an **Express.js middleware** function that limits requests to 100 per hour per **IP address**. Coding Challenge
49. Create an **Express.js route** that accepts a **JSON payload** and responds with the same payload in reverse order. Coding Challenge
50. Develop a simple **REST API** with **Express.js** that includes **CRUD operations** for managing books. Include **route definitions** and **handler functions** that interact with a placeholder data store. Coding Challenge

Go up

Topic progress: 16%

51. Discuss strategies for building a **real-time application** with **Express.js**. Question ○

52. What are some common **optimization techniques** for **Express.js** applications? Question ○

53. Explain the concept and benefits of **server-side rendering** with **Express.js** and a **templating engine**. Question ○

54. How do you ensure that your **Express.js code is maintainable** and follows **best practices**? Question ○

Case Studies and Scenario-Based Questions

55. How would you structure an **Express.js application** for a large-scale **e-commerce platform**? Question ○

56. Describe the key considerations for building a **secure API** with **Express.js**. Question ○

57. Discuss how to handle **session management** in a distributed **Express.js application**. Question ○

58. What are the implications of **microservices architecture** for an **Express.js application**? Question ○

Go up

Topic progress: 16%