Full-Stack Web Development

Full-Stack viva/interview questions.

What is Node.js?

Node.js is an open-source, cross-platform **JavaScript runtime environment**. This means it allows you to run JavaScript code on the server, outside of a web browser. It's built on Google's V8 JavaScript engine, which is the same engine that powers Google Chrome, making it very fast. Node.js is known for its **event-driven, non-blocking I/O model**, which makes it highly efficient and suitable for building real-time, scalable applications.

What is Express.js and how does it relate to Node.js?

Express.js is a minimal and flexible **web application framework** for Node.js. While Node.js gives you the raw tools to build a server, Express.js provides a robust set of features to make web and mobile application development much easier. It simplifies common tasks like routing, managing requests and responses, and implementing middleware. Think of Node.js as the foundation and Express.js as the blueprint and tools you use to quickly build a house on that foundation.

Key Concepts and Questions

1. What are Node.js modules?

A module is a reusable block of code in a separate file. Modules help you organize your code, prevent variable conflicts, and make your application more manageable. There are three types of modules:

- **Core Modules:** Built into Node.js, like http or fs (file system).
- **Local Modules:** Custom modules you create in your own project.
- **Third-party Modules:** Installed using a package manager like **npm** (Node Package Manager).

2. What is npm?

NPM is the default package manager for Node.js. It's used to install, share, and manage project dependencies and packages. When you install a library like Express.js, you use npm.

3. What is middleware in Express.js?

Middleware is a function that has access to the request (req) object, the response (res) object, and the next() function in the application's request-response cycle. It acts like a "middleman" that can execute code, modify the request and response objects, and even end the request-response cycle.

- **Common uses of middleware:** logging, authentication, data parsing, and handling errors.
- **Example:** When a user logs in, a middleware function can check their credentials before allowing the request to proceed to the final route handler.

4. What is the purpose of the next() function?

The next() function is a crucial part of middleware. It passes control to the next middleware function in the stack. If you don't call next(), the request will be left hanging, and the user won't receive a response.

5. How do you handle routing in Express.js?

Routing is how an application responds to a client's request for a specific endpoint (a URI and HTTP method). Express.js makes routing simple using methods like app.get(), app.post(), app.put(), and app.delete(). Each method defines a different route handler for a specific HTTP method.

- **Example:**

JavaScript

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

This code defines a route for a GET request to the root URL (/) and sends back "Hello World!".