Node-js

1. **What is Node.js?**
   Node.js is a runtime environment that allows you to run JavaScript code on the server side using the V8 engine.

2. **What are the main features of Node.js?**
   - Asynchronous and Event-Driven
   - Fast Execution (thanks to V8 engine)
   - Single-Threaded but handles concurrency using Event Loop
   - Built-in modules

3. **What is npm?**
   npm (Node Package Manager) is the default package manager for Node.js. It is used to install, share, and manage dependencies.

4. **What is the difference between `require()` and `import`?**
   - `require()` is used in CommonJS modules.
   - `import` is used in ES6 modules (with `"type": "module"` in `package.json`).
   -

5. **What is the purpose of `package.json`?**
   It manages the metadata of a Node.js project, including dependencies, scripts, version, and more.
6. **What is the Event Loop in Node.js?**
   It's a mechanism that allows Node.js to perform non-blocking I/O operations despite being single-threaded.

7. **Explain asynchronous programming in Node.js.**
   Node.js uses callbacks, Promises, or `async/await` to handle asynchronous operations, allowing it to process other tasks while waiting for I/O.

8. **What are streams in Node.js?**
   Streams are objects that let you read data from a source or write data to a destination in a continuous fashion. Types: Readable, Writable, Duplex, Transform.

9. **What are buffers in Node.js?**
   Buffers are used to handle binary data, especially in file streams and network protocols.

10. **Difference between `process.nextTick()`, `setImmediate()`, and `setTimeout()`?**

- `process.nextTick()`: Runs after the current operation but before the next event loop tick.
- `setImmediate()`: Executes in the next iteration of the event loop.
- `setTimeout()`: Delays execution for a given time.

11. **What is middleware in Express.js?**
    Functions that execute during the lifecycle of a request to the Express server. They can modify `req`, `res`, or end the request-response cycle.

12. **How does Node.js handle concurrency with a single thread?**
    Through non-blocking I/O and the Event Loop + Worker Threads (for CPU-heavy tasks).

13. **What is the role of the cluster module in Node.js?**
    It allows you to create child processes (workers) to take advantage of multi-core systems.

14. **How to handle uncaught exceptions in Node.js?**

- `process.on('uncaughtException', handler)`
- Best practice: log and restart the process, don’t continue execution.

15. **What is the difference between synchronous and asynchronous file operations in Node.js?**
- Synchronous blocks the event loop.
- Asynchronous does not block and uses callbacks or Promises.

16. **How can you secure a Node.js application?**
- Use `helmet`, input validation, rate limiting, authentication, HTTPS, environment variables, sanitize inputs, avoid `eval`.

17. **What is the use of `dotenv` in Node.js?**
It loads environment variables from a `.env` file into `process.env`.

18. **What is the use of `util.promisify()`?**
It converts callback-based functions to return Promises for use with `async/await`

19. **What is the difference between `spawn()` and `fork()` in Node.js?**
- `spawn()` launches a new process.
- `fork()` is a special case of `spawn()` that creates a new Node.js instance with IPC channel.

20. **What are some best practices for building a Node.js application?**
- Use async/await
- Avoid blocking the event loop
- Use environment variables
- Modularize code
- Handle errors gracefully
- Use logging and monitoring tools (like Winston, PM2)

Express

1. **What is Express.js?**
Express.js is a minimal and flexible Node.js web application framework that provides robust features to build web and mobile applications.

2. **What are the key features of Express.js?**
    - Routing
    - Middleware support
    - Fast server-side development
    - Integration with databases
    - Simplified HTTP request handling

3. **How do you create a basic Express server?**

```
const express = require('express');
const app = express();
app.get('/', (req, res) => res.send('Hello World!'));
app.listen(3000, () => console.log('Server running on port 3000'));
```

4. **What is middleware in Express.js?**
Middleware is a function that executes during the request-response cycle. It has access to req, res, and next().

5. **How do you serve static files in Express?**
app.use(express.static('public'));

6. **What are the different types of middleware in Express?**

   There are five types of middleware
   - Application-level middleware
   - Router-level middleware
   - Error-handling middleware
   - Built-in middleware
   - Third-party middleware

7. **How do you handle routing in Express.js?**

```
app.get('/route', (req, res) => { ... });
app.post('/route', (req, res) => { ... });
app.put('/route', (req, res) => { ... });
app.delete('/route', (req, res) => { ... });
```

8. **How do you handle form data in Express.js?**

```
app.use(express.urlencoded({ extended: true }));
app.post('/submit', (req, res) => {
  const { name } = req.body;
  res.send(`Name: ${name}`);
});
```

9. **How do you handle JSON data in Express?**

```
app.use(express.json());
```

10. **How do you create routes in a separate file using Express Router?**

```
// routes/user.js
const express = require('express');
const router = express.Router();
router.get('/', (req, res) => res.send('User home'));
module.exports = router;

// app.js
const userRouter = require('./routes/user');
app.use('/user', userRouter);
```

11. **How do you handle errors in Express?**

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

12. **What is next() used for?**

    It's used to pass control to the next middleware or route handler.

13. **How do you use async/await with Express routes?**

```
app.get('/data', async (req, res, next) => {
  try {
    const result = await someAsyncFunction();
    res.json(result);
```

```
  } catch (error) {
    next(error); // pass to error handler
  }
});
```

14. **How do you secure your Express app?**
- Use helmet for setting secure HTTP headers
- Use express-rate-limit for rate limiting
- Validate inputs
- Sanitize user data
- Enable CORS carefully

15. **How can you handle 404 errors in Express?**

```
app.use((req, res) => {
  res.status(404).send('Page not found');
});
```

16. **What is CORS and how do you enable it in Express?**
- CORS (Cross-Origin Resource Sharing) allows restricted resources to be accessed across domains.
-

```
const cors = require('cors');
app.use(cors());
```

17. **How do you connect Express with a MySQL database?**
    Using the mysql2 or sequelize package:

```
const mysql = require('mysql2');
const db = mysql.createConnection({ host, user, password, database });
db.connect();
```

18. **What are some common third-party middleware used in Express?**
- morgan – for logging
- helmet – for security headers
- cors – for enabling CORS
- body-parser (mostly now built-in)
- express-session – for session handling

19. Why Use Middleware?

Middleware functions can:

- Execute any code
- Modify req and res objects
- End the request-response cycle
- Call the next middleware in the stack

20. Application-level Middleware

Used at the application level using app.use() or route methods like app.get().

```
app.use((req, res, next) => {
  console.log('Time:', Date.now());
  next();
});
```

21. Router-level Middleware

Works the same as application-level middleware, but bound to an instance of express.Router().

```
const router = express.Router();
router.use((req, res, next) => {
  console.log('Router-level middleware triggered');
  next();
});

app.use('/api', router);
```

22. Error-handling Middleware

Defined with **four** arguments: (err, req, res, next). It catches errors in the app.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

23. Built-in Middleware

Express provides built-in middleware:

```
app.use(express.json()); // Parse JSON body
app.use(express.urlencoded({ extended: true })); // Parse form data
app.use(express.static('public')); // Serve static files
```

24. Third-party Middleware

Installed from npm for added functionality.

```
const morgan = require('morgan');
const helmet = require('helmet');
const cors = require('cors');

app.use(morgan('dev'));
app.use(helmet());
app.use(cors());
```

25. Custom Middleware Example

```
function logger(req, res, next) {
  console.log(`${req.method} ${req.url}`);
  next(); // Move to next middleware or route
}

app.use(logger);
```

26. Chaining Multiple Middleware

```
app.use('/products', middleware1, middleware2, (req, res) => {
  res.send('Final handler');
});
```

27. Real-World Use Cases of Middleware

- **Logging requests** (e.g. morgan)
- **Validating tokens** (e.g. JWT auth middleware)
- **Checking permissions/roles**
- **Sanitizing input**
- **Handling file uploads**
- **Rate limiting**

EJS

28. **What is EJS?**
    EJS (Embedded JavaScript) is a simple templating language that lets you generate HTML markup with plain JavaScript.

29. **How do you install and set up EJS in an Express app?**

npm install ejs

In app.js:

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

30. **How do you render an EJS template in Express?**

res.render('home', { name: 'Mahesh' });

In home.ejs:

<h1>Hello <%= name %></h1>

31. **What's the difference between <%= %> and <%- %> in EJS?**

- <%= value %>: Escapes HTML (prevents XSS)
- <%- value %>: Outputs raw HTML (used for rendering HTML directly)

32. **Where should EJS files be stored by default in Express?**
    Inside a views/ folder (Express looks here by default).

33. **How do you include partials in EJS?**
    Use `<%- include('partials/header') %>` to reuse code like headers or footers.

34. **How do you pass multiple variables to an EJS file?**

```
res.render('dashboard', {
     user: 'Mahesh',
     age: 25,
     hobbies: ['Gym', 'Code']
   });
```

35. **How do you loop through an array in EJS?**

```
   <ul>
   <% hobbies.forEach(function(hobby) { %>
<li><%= hobby %></li>
<% }); %>
</ul>
```

36. **How can you send EJS variables via a route and display them dynamically?**

- Backend:

```
app.get('/profile', (req, res) => {
  res.render('profile', { name: 'Mahesh', loggedIn: true });
});
```

- Frontend (EJS):

```
<% if (loggedIn) { %>
  <h2>Welcome, <%= name %>!</h2>
<% } %>
```

28**GET Request Example**

```
app.get('/student', (req, res) => {
  const name = req.query.name;
  res.send(`Student name is ${name}`);
});
```

21.