SECTION 1: Core Node.js (Basic to Intermediate)

1. **What is Node.js?**
    - Node.js is a runtime environment that executes JavaScript code outside a browser. It's built on Chrome's V8 engine and uses an event-driven, non-blocking I/O model.
2. **What are the features of Node.js?**
    - Asynchronous and Event-Driven
    - Very Fast
    - Single-threaded
    - No Buffering
    - Cross-platform
3. **What is the difference between Node.js and JavaScript?**
    - JavaScript runs in browsers, while Node.js is a server-side runtime for running JS code on servers.
4. **Explain Event Loop in Node.js.**
    - The event loop allows Node.js to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible.
5. **What is the use of `process` object in Node.js?**
    - `process` provides information about and control over the current Node.js process. It can access environment variables, handle signals, exit codes, etc.
6. **What is a callback in Node.js?**
    - A function passed as an argument to another function, executed after the parent function completes.
7. **What is a Promise in Node.js?**
    - An object representing the eventual completion or failure of an asynchronous operation.
8. **How do you create a simple server in Node.js?**

```
const http = require('http');
http.createServer((req, res) => {
  res.write('Hello, World!');
  res.end();
}).listen(3000);
```

9. **What is the difference between `setImmediate()` and `setTimeout()`?**
    - `setImmediate()` executes a callback after the current poll phase, while `setTimeout()` schedules a callback after a delay.
10. **Explain the difference between synchronous and asynchronous methods.**

- Synchronous blocks the thread until completion; asynchronous continues executing and invokes a callback upon completion.

11. **What is the role of `require` in Node.js?**

- It’s used to import modules or files.

12. **What are modules in Node.js?**

- Reusable blocks of code whose existence does not impact other code.

13. **What is NPM?**

- Node Package Manager used to install, update, and manage dependencies in Node.js projects.

14. **What is the difference between dependencies and devDependencies?**

- dependencies are required for the app to run. devDependencies are used only in development.

15. **What is the purpose of package.json?**

- Metadata file that holds project information and dependencies.

16. **Explain middleware in Node.js.**

- Functions that execute during the lifecycle of a request to the server.

17. **What is the use of the cluster module in Node.js?**

- It allows Node.js to take advantage of multi-core systems.

18. **What are streams in Node.js?**

- Objects used to read or write data piece by piece.

19. **Types of streams in Node.js?**

- Readable, Writable, Duplex, Transform

20. **How to handle exceptions in Node.js?**

- Using try/catch blocks, error events, or .catch() for promises.

SECTION 2: Express.js in Node.js

21. **What is Express.js?**

- A fast, minimalist web framework for Node.js.

22. **How to install and use Express in Node.js?**

```
npm install express
const express = require('express');
const app = express();
app.listen(3000);
```

23. **How to define routes in Express?**

```
app.get('/', (req, res) => res.send('Home'));
```

24. **What is a middleware in Express?**

- Functions that execute in the middle of request/response cycle.

25. **Explain routing in Express.js.**

- It refers to determining how an application responds to client requests.

26. **How to handle 404 errors in Express?**

```
app.use((req, res) => res.status(404).send('Not Found'));
```

27. **What is `req` and `res` in Express?**

- `req`: request object, `res`: response object

28. **How to handle JSON requests in Express?**

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

29. **What is `next()` function in middleware?**

- It passes control to the next middleware function.

30. **How to implement CORS in Express.js?**

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

31. **What is body-parser?**

- Middleware to parse incoming request bodies. Now built into Express.

32. **How to serve static files in Express?**

```
app.use(express.static('public'));
```

33. **What is app.use() used for?**

- Mounts the specified middleware function(s) at the path.

34. **How to connect MongoDB with Express.js?**

- Using Mongoose or native MongoDB driver.

35. **How to handle form data in Express?**

```
app.use(express.urlencoded({ extended: true }));
```

SECTION 3: MongoDB and Mongoose

36. **What is MongoDB?**

- A NoSQL database that stores data in JSON-like documents.

37. **What is Mongoose?**

- An ODM (Object Data Modeling) library for MongoDB and Node.js.

38. **How to define a schema in Mongoose?**

```
const schema = new mongoose.Schema({ name: String });
```

39. **How to create a model in Mongoose?**

```
const User = mongoose.model('User', schema);
```

40. **How to connect to MongoDB?**

```
mongoose.connect('mongodb://localhost:27017/mydb');
```

41. **Difference between findOne and find in Mongoose?**

- findOne returns one document; find returns an array of documents.

42. **How to update a document in Mongoose?**

```
User.updateOne({ _id }, { $set: { name: 'New' } });
```

43. **How to delete a document in Mongoose?**

```
User.deleteOne({ _id });
```

44. **What is population in Mongoose?**

- A way to join documents across collections.

45. **How to use Mongoose validation?**

- Define validations in the schema (e.g., `required: true`).

SECTION 4: Authentication & Security

46. **How to implement JWT in Node.js?**

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ id }, 'secret', { expiresIn: '1h' });
```

47. **What is middleware-based authentication?**

- Using middleware to verify tokens before accessing protected routes.

48. **How to hash passwords in Node.js?**

```
const bcrypt = require('bcrypt');
const hash = await bcrypt.hash(password, 10);
```

49. **How to verify hashed passwords?**

```
await bcrypt.compare(plainText, hash);
```

50. **How to store session in Express?**

- Use `express-session` middleware.

51. **What is CORS and why is it important?**

- CORS is a security feature to restrict resource access from different origins.

52. **How to prevent SQL/NoSQL injection in Node.js?**

- Use ORMs/ODMs like Mongoose and sanitize input.

SECTION 5: Advanced Topics

53. **What is an event emitter?**

- Node.js class that handles named events.

54. **How to use event emitter?**

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
emitter.on('start', () => console.log('Started'));
emitter.emit('start');
```

55. **What are environment variables in Node.js?**

- Configuration variables stored outside the code using .env.

56. **How to use dotenv in Node.js?**

```
require('dotenv').config();
```

57. **How to handle file uploads in Node.js?**

- Use multer middleware.

58. **How to send emails from Node.js?**

- Use nodemailer package.

59. **What is throttling and rate-limiting?**

- Mechanisms to control API usage.

60. **What are WebSockets in Node.js?**

- Protocol for real-time two-way communication.

SECTION 6: Real-time and Deployment

61. **What is Socket.IO and how is it used in Node.js?**

- Socket.IO is a library for real-time web applications, enabling bi-directional communication between client and server over WebSockets.

62. **Example of using Socket.IO in Node.js?**

```
const io = require('socket.io')(server);
io.on('connection', socket => {
  console.log('User connected');
  socket.on('message', msg => io.emit('message', msg));
});
```

63. **What is the role of res.send() vs res.json() in Express?**

- res.send() sends any type of response. res.json() specifically sends a JSON response.

64. **How to implement file downloads in Node.js?**

```
res.download('./files/report.pdf');
```

65. **What is PM2 in Node.js?**

- PM2 is a process manager for Node.js apps that enables load balancing, monitoring, and restarting on failure.

66. **How to use PM2 to start a Node app?**

```
pm2 start app.js
```

67. **What is Helmet.js and how does it help?**

- Helmet is middleware that sets HTTP headers for app security (e.g., XSS, CSP, etc.).

68. **How do you deploy a MERN app to production?**

- Host frontend on Netlify/Vercel or static server; backend on services like Heroku, Render, AWS, or DigitalOcean; connect to MongoDB Atlas or self-hosted MongoDB.

69. **How do you connect React with a Node.js backend?**

- Use fetch() or axios to call RESTful APIs defined in Node/Express from React components.

70. **How to serve a React frontend from a Node.js Express backend?**

```
app.use(express.static(path.join(__dirname, 'client/build')));
```

SECTION 7: Testing and Debugging

71. **How do you debug a Node.js app?**

- Use `console.log()`, Node Inspector, or `node --inspect`.

72. **What are some popular testing libraries in Node.js?**

- Jest, Mocha, Chai, Supertest

73. **What is unit testing?**

- Testing individual units/components in isolation.

74. **What is integration testing?**

- Testing how different parts of the application work together.

75. **Example of a unit test using Jest:**

```
test('adds 1 + 2 = 3', () => {
  expect(1 + 2).toBe(3);
});
```

76. **What is mocking in tests?**

- Replacing real functions or modules with simulated versions for isolated testing.

77. **How to test an Express API endpoint?**

- Use `supertest` to make HTTP requests to your Express routes.

78. **What is TDD (Test-Driven Development)?**

- Writing tests before writing the code to implement functionality.

79. **How do you handle logging in Node.js?**

- Use built-in `console` or libraries like `winston`, `pino`.

80. **How to handle global error logging in Express?**

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

SECTION 8: REST API Design

81. **What is REST API?**

- REST stands for Representational State Transfer. It's an architecture for designing networked applications using HTTP methods.

82. **Common HTTP methods and their purpose:**

- GET: Retrieve data
- POST: Create data
- PUT/PATCH: Update data
- DELETE: Remove data

83. **What is RESTful routing in Express?**

- Structuring routes according to REST principles, like /users, /users/:id.

84. **How to send status codes in Express response?**

```
res.status(201).json({ success: true });
```

85. **How to create REST API with CRUD operations?**

```
app.post('/items', createItem);
app.get('/items', getItems);
app.put('/items/:id', updateItem);
app.delete('/items/:id', deleteItem);
```

86. **What is Postman and how is it used?**

- Postman is a tool to test APIs by sending HTTP requests and inspecting responses.

87. **How to handle query parameters in Express?**

```
app.get('/search', (req, res) => {
  const query = req.query.q;
});
```

88. **How to handle route parameters in Express?**

```
app.get('/user/:id', (req, res) => {
  const id = req.params.id;
});
```

89. **What is API versioning and why is it important?**

- Including a version (e.g., /api/v1/) helps manage backward compatibility of APIs.

90. **How to document REST APIs?**

- Use tools like Swagger/OpenAPI, Postman docs, or manually via README.

SECTION 9: Miscellaneous & Best Practices

91. **How to improve performance in Node.js?**

- Use clustering, async patterns, caching, database indexing, and avoid blocking operations.

92. **Difference between require and import in Node.js?**

- require is CommonJS (default in Node). import is ES Module syntax (modern JS with "type": "module" in package.json).

93. **What is the __dirname variable in Node.js?**

- Absolute path of the directory containing the currently executing file.

94. **How do you structure a large Node.js project?**

- Use folders like routes, controllers, models, services, and middlewares.

95. **How to use async/await in Node.js?**

```
const getUser = async () => {
  const user = await User.findById(id);
};
```

96. **What is the purpose of .gitignore in a Node.js project?**

- To prevent certain files (e.g., node_modules, .env) from being committed to Git.

97. **What is the role of .env file?**

- Stores environment variables like DB_URI, PORT, API_KEY.

98. **How to secure a Node.js API?**

- Use HTTPS, input validation, rate limiting, auth mechanisms like JWT, and avoid exposing sensitive info.

99. **Difference between local and global installation in npm?**

- Local: installed in project (node_modules), Global: accessible from anywhere via CLI.