

MERN Stack

Basic Level (1-25)

1. What is the MERN stack?

- MERN stands for **MongoDB, Express.js, React.js, and Node.js**, a full-stack JavaScript technology stack used for web development.

2. What is MongoDB?

- MongoDB is a **NoSQL database** that stores data in **JSON-like BSON format** for flexibility and scalability.

3. What is Express.js?

- Express.js is a **backend web framework for Node.js** that simplifies server-side development.

4. What is React.js?

- React.js is a JavaScript **library** used for building **user interfaces**, primarily for single-page applications (SPA).

5. What is Node.js?

- Node.js is a JavaScript runtime environment that allows executing JavaScript **on the server-side**.

6. How does the MERN stack work together?

- **MongoDB** stores data → **Express.js** handles backend APIs → **React.js** manages UI → **Node.js** executes backend logic.

7. Why is the MERN stack popular?

- Uses **JavaScript** for both frontend & backend, has a strong **developer community**, and offers **high performance**.

8. What is the difference between SQL and NoSQL?

- SQL databases are **structured** (e.g., MySQL), while NoSQL databases (e.g., MongoDB) store **unstructured or semi-structured** data.

9. What is JSX in React?

- JSX allows writing **HTML inside JavaScript**, making UI development more intuitive.

10. What is a REST API?

- A REST API is an **architectural style** using HTTP methods (GET, POST, PUT, DELETE) for data communication.

11. How do you create a basic Express server?

javascript

CopyEdit

```
const express = require('express');
const app = express();
app.listen(3000, () => console.log('Server running on
port 3000'));
```

12. What is npm?

- npm (Node Package Manager) is a tool to manage JavaScript **packages and dependencies**.

13. What is **useState** in React?

A **React Hook** used to manage component-level state:

javascript

CopyEdit

```
const [count, setCount] = useState(0);
```

-

14. What is a functional component in React?

- A simple JavaScript function that returns JSX and doesn't have lifecycle methods like class components.

15. How do you install MongoDB?

- Download from [MongoDB official website](#) and run `mongod` to start the server.

16. What is middleware in Express.js?

- Middleware functions process requests **before they reach the final handler**.

17. What is `useEffect` in React?

- A React Hook for managing **side effects** like API calls or event listeners.

18. What is the difference between class and functional components?

- Class components use `this.state`, while functional components use **Hooks** like `useState`.

19. What is `props` in React?

- **Props (short for properties)** allow data to be passed from **parent to child** components.

20. What is the difference between React and React Native?

- React is for web applications, while React Native is for **mobile app development**.

21. How do you define a schema in Mongoose?

javascript

CopyEdit

```
const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({ name: String,
email: String });
const User = mongoose.model('User', UserSchema);
```

22. What is CORS in Express.js?

- Cross-Origin Resource Sharing (CORS) allows requests **from different domains**.

23. How do you handle form submission in React?

- Using the `onSubmit` event and `useState` to manage form data.

24. What is the difference between state and props in React?

- **State** is mutable (internal), while **props** are immutable (passed from parent to child).

25. How do you start a new React project?

- Run: `npx create-react-app my-app`

Intermediate Level (26-50)

26. What is an arrow function in JavaScript?

javascript

CopyEdit

```
const sum = (a, b) => a + b;
```

27. What is the purpose of MongoDB indexes?

- Indexes **improve query performance** by optimizing searches.

28. How do you fetch data in React using fetch API?

javascript

CopyEdit

```
useEffect(() => {  
  fetch('https://api.example.com/data')  
    .then(res => res.json())  
    .then(data => setData(data));  
}, []);
```

29. What is a higher-order component (HOC) in React?

- A function that takes a component and returns an enhanced component.



GUIDE'S FOR PERFECT CAREER PATHWAY

30. What is **map()** in JavaScript?

- An array method used to iterate and transform elements.

31. How do you use Express Router to structure your backend?

- Express Router helps in **modularizing routes** for better code management.

javascript

CopyEdit

```
const express = require('express');  
const router = express.Router();  
router.get('/users', (req, res) => res.send('User  
list'));  
module.exports = router;
```

32. How do you handle errors in Express.js?

- Using an error-handling middleware:

javascript

CopyEdit

```
app.use((err, req, res, next) => {  
  res.status(500).json({ error: err.message });  
});
```

33. What is the purpose of **useRef** in React?

- **useRef** creates a **persistent mutable object** without re-rendering the component.

javascript

CopyEdit

```
const inputRef = useRef();
```



34. What is **useMemo** in React, and why is it used?

- **useMemo** **caches computations** to optimize performance.

javascript

CopyEdit

```
const memoizedValue = useMemo(() =>  
  computeExpensiveValue(data), [data]);
```

35. What is Redux, and why is it used in React?

- Redux is a **state management library** that provides a **centralized store** for managing app-wide state.

36. What are the main components of Redux?

- **Store** (holds state), **Actions** (define events), **Reducers** (modify state), **Dispatch** (sends actions).

37. How do you create a Redux store?

javascript

CopyEdit

```
import { createStore } from 'redux';  
const store = createStore(reducer);
```

38. How do you update state in Redux?

- Using `dispatch()`:

javascript

CopyEdit

```
store.dispatch({ type: 'INCREMENT' });
```

39. What is `useDispatch` and `useSelector` in React-Redux?

- `useDispatch()` sends actions to Redux, `useSelector()` gets state from the store.

40. What is Context API in React, and how does it compare to Redux?

- Context API provides a way to **pass data down the component tree** without props drilling. It is **simpler than Redux** but less powerful for complex state management.

41. What is a `Promise` in JavaScript?

- A `Promise` represents an asynchronous operation that may succeed (`resolve()`) or fail (`reject()`).

javascript

CopyEdit

```
let myPromise = new Promise((resolve, reject) => {
  resolve("Success"); });
```

42. What is async/await in JavaScript?

- It is used to **handle asynchronous code** in a synchronous-like manner.

javascript

CopyEdit

```
async function fetchData() {
  let response = await
  fetch('https://api.example.com/data');
  let data = await response.json();
  console.log(data);
}
```

43. How do you use JWT for authentication in MERN?

- Generate JWT token:

javascript

CopyEdit

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ userId: user._id }, 'secretKey',
{ expiresIn: '1h' });
```

44. How do you verify a JWT token in Express.js?

- Middleware to protect routes:

javascript

CopyEdit

```
app.use('/protected', (req, res, next) => {
```

```

const token = req.headers['authorization'];
jwt.verify(token, 'secretKey', (err, decoded) => {
  if (err) return res.status(401).send('Unauthorized');
  req.user = decoded;
  next();
});
});

```

45. How do you create a protected route in React?

- Using React Router:

javascript

CopyEdit

```

const PrivateRoute = ({ component: Component, ...rest })
=> {
  const isAuthenticated =
!!localStorage.getItem('token');
  return (
    <Route {...rest} render={(props) => (
      isAuthenticated ? <Component {...props} /> :
<Redirect to="/login" />
    )} />
  );
};

```

46. What is Axios, and why use it over fetch()?

- Axios is a **Promise-based HTTP client** with features like request canceling and automatic JSON parsing.

javascript

CopyEdit

```

axios.get('/api/data').then(response =>
console.log(response.data));

```

47. What is the difference between `sessionStorage` and `localStorage`?

- `sessionStorage` stores data **for the session**, while `localStorage` persists **until manually cleared**.

48. How do you optimize MongoDB queries?

- Use indexes, limit fields returned, avoid unnecessary queries, and use aggregation pipelines.

49. What is Mongoose `populate()` method?

- It is used to **fetch referenced documents** in MongoDB.

javascript
CopyEdit

```
User.find().populate('posts').exec((err, users) =>
console.log(users));
```

50. How do you set up Redux in a React project?

- Install dependencies:

bash

CopyEdit

```
npm install redux react-redux
```

- Create a Redux store:

javascript
CopyEdit

```
import { createStore } from 'redux';
```

```
const store = createStore(reducer);
```

- Wrap the app with `<Provider>`:

javascript

CopyEdit

```
import { Provider } from 'react-redux';  
<Provider store={store}>  
  <App />  
</Provider>
```

Advanced Level (51-75)

51. How do you manage authentication in MERN Stack?

- Using **JWT (JSON Web Token)** for user authentication.

52. How does React Virtual DOM work?

- It updates only the **changed parts of the UI** efficiently.

53. What is lazy loading in React?

javascript

CopyEdit

```
const LazyComponent = React.lazy(() =>  
import('./Component'));
```

54. What are WebSockets, and how are they used?

- WebSockets enable **real-time communication** between the server and client.

55. What is server-side rendering (SSR) in React?

- **SSR** renders React components on the server instead of the browser, improving performance and SEO.

56. How do you implement SSR in React with Next.js?

- Using **getServerSideProps()** in Next.js:

```
export async function getServerSideProps() {  
  const res = await  
  fetch('https://api.example.com/data');  
  const data = await res.json();  
  return { props: { data } };  
}
```

- This fetches data **before rendering the page**.

57. What is code splitting in React?

- It **loads JavaScript code only when needed**, improving performance.

```
const LazyComponent = React.lazy(() =>  
  import('./Component'));
```

58. How does React Suspense work?

- It handles **lazy loading components** and displays a fallback UI while loading.

```
<Suspense fallback={<div>Loading...</div>}>
```

```
<LazyComponent />
</Suspense>
```

59. What are React Portals?

- **Portals render components outside their parent DOM hierarchy, useful for modals.**

```
ReactDOM.createPortal(<Modal />,
document.getElementById('modal-root'));
```

60. How do you optimize React performance?

- **Use React.memo(), useCallback(), useMemo(), lazy loading, and optimize re-renders.**

61. What is GraphQL, and how does it compare to REST APIs?

- **GraphQL allows fetching only required data with a single query, unlike REST.**

62. How do you implement GraphQL in a MERN stack app?

- **Install GraphQL dependencies:**

```
npm install express-graphql graphql
```

- **Define a schema:**

```
const { GraphQLObjectType, GraphQLSchema, GraphQLString }
= require('graphql');
const RootQuery = new GraphQLObjectType({
  name: 'RootQueryType',
  fields: { message: { type: GraphQLString, resolve() {
return "Hello World"; } } }
});
module.exports = new GraphQLSchema({ query: RootQuery });
```

63. What is WebSocket, and how is it used in a MERN app?

- **WebSockets enable real-time communication** using bidirectional connections.

```
const socket = new WebSocket('ws://localhost:5000');
```

64. What is MongoDB Aggregation?

- A framework for **processing and transforming data** in MongoDB.

```
db.users.aggregate([ { $group: { _id: "$role", count: {
$sum: 1 } } } ] );
```

65. How do you handle transactions in MongoDB?

- Using the **session object** with transactions:

```
const session = await mongoose.startSession();
session.startTransaction();
try {
```

```
    await User.updateOne({ _id: id }, { balance: newBalance
}, { session });
    await session.commitTransaction();
} catch (error) {
    await session.abortTransaction();
}
```

66. What is Redux Thunk, and why is it used?

- **Redux Thunk handles async operations** inside Redux actions.

```
export const fetchData = () => async (dispatch) => {
  const response = await fetch('/api/data');
  const data = await response.json();
  dispatch({ type: 'FETCH_SUCCESS', payload: data });
};
```



IDM TECHPARK
GUIDE'S FOR PERFECT CAREER PATHWAY

67. How does **useReducer** work in React?

- A **React Hook alternative to Redux** for managing state with a reducer function.

```
const reducer = (state, action) => action.type ===
"increment" ? state + 1 : state;
const [count, dispatch] = useReducer(reducer, 0);
```

68. How do you secure Express.js APIs?

- **Use JWT authentication, input validation, CORS, HTTPS, and rate limiting.**

69. How do you prevent SQL/NoSQL injection attacks?

- Sanitize inputs, use parameterized queries, and limit database access.

70. What are Microservices, and how can they be used in MERN stack?

- Microservices split an app into small, independent services communicating via APIs.
- Example: Separate auth, payments, and user services as independent apps.

71. How do you implement role-based access control (RBAC) in MERN?

- Middleware checks user roles before granting access.

```
const authorize = (roles) => (req, res, next) => {  
  if (!roles.includes(req.user.role)) return  
  res.status(403).send("Forbidden");  
  next();  
};
```

72. What is Docker, and how can you deploy a MERN stack app using it?

- Docker containerizes the app for easy deployment.
- Sample Dockerfile for Node.js backend:

FROM node:14

```
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
CMD ["node", "server.js"]
```

73. How do you use PM2 to manage Node.js processes?

- Install PM2 and start the server:

```
npm install -g pm2
pm2 start server.js
```

74. How do you integrate Redis with a MERN stack app?

- Install Redis and use it as a **cache layer**:

```
const redis = require('redis');
const client = redis.createClient();
client.set("key", "value");
```

75. How do you implement OAuth authentication in MERN stack?

- Use **Google OAuth or Facebook Login** with Passport.js.

```
passport.use(new GoogleStrategy({ clientID, clientSecret,
  callbackURL },
  (accessToken, refreshToken, profile, done) => {
    User.findOrCreate({ googleId: profile.id }, done);
  })
);
```

));

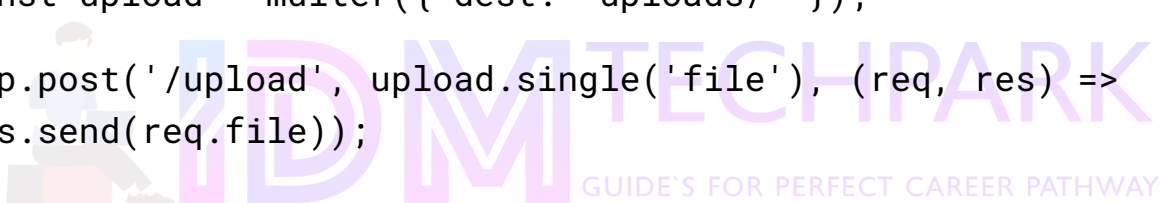
Here are **20 technical-level MERN Stack interview questions and answers**, continuing from 76 to 95.

Technical-Level (76-95)

76. How do you handle file uploads in a MERN app?

- Using `multer` in Express.js:

```
const multer = require('multer');  
  
const upload = multer({ dest: 'uploads/' });  
  
app.post('/upload', upload.single('file'), (req, res) =>  
  res.send(req.file));
```



77. How do you implement pagination in MongoDB?

- Using `.skip()` and `.limit()`:

```
const page = 2;  
  
const limit = 10;  
  
const users = await User.find().skip((page - 1) *  
  limit).limit(limit);
```

78. How do you improve the performance of MongoDB queries?

- **Index fields, use projections, limit the number of documents returned, and optimize aggregation pipelines.**

79. How do you configure CORS in an Express.js app?

- Using `cors` middleware:

```
const cors = require('cors');  
  
app.use(cors({ origin: 'http://example.com' }));
```

80. How do you optimize React rendering performance?

- Use `React.memo()`, `useCallback()`, `useMemo()`, `shouldComponentUpdate()`, and `React Profiler`.

81. What is HOC (Higher-Order Component) in React?

- A function that **takes a component and returns a new component** with added functionality.

```
const withLogger = (Component) => (props) => {  
  console.log("Rendered");  
  return <Component {...props} />;  
};
```

82. What is Debouncing in JavaScript, and how do you implement it?

- Delays function execution until after a certain time passes.

```
function debounce(fn, delay) {  
  let timer;  
  return (...args) => {  
    clearTimeout(timer);  
    timer = setTimeout(() => fn(...args), delay);  
  };  
}
```

83. What is Throttling in JavaScript, and how does it differ from Debouncing?

- **Throttling limits function execution at a fixed interval.**

```
function throttle(fn, limit) {  
  let lastCall = 0;  
  return (...args) => {  
    const now = Date.now();  
    if (now - lastCall >= limit) {  
      lastCall = now;  
      fn(...args);  
    }  
  }  
}
```

```
};  
}
```

84. How do you create a custom React Hook?

- Example: `useFetch` hook for fetching data.

```
function useFetch(url) {  
  const [data, setData] = useState(null);  
  useEffect(() => {  
    fetch(url).then(res => res.json()).then(setData);  
  }, [url]);  
  return data;  
}
```



85. How do you configure environment variables in a MERN stack app?

- **Backend:** Use `.env` file with `dotenv`:

```
require('dotenv').config();  
  
const PORT = process.env.PORT;
```

- **Frontend:** Store variables in `.env` with `REACT_APP_` prefix.

86. How do you implement WebSockets in MERN for real-time updates?

- Use `socket.io`:

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

87. How do you implement authentication middleware in Express.js?

```
function authMiddleware(req, res, next) {  
  const token = req.headers.authorization;  
  
  if (!token) return  
  res.status(401).send('Unauthorized');  
  
  jwt.verify(token, 'secretKey', (err, user) => {  
    if (err) return res.status(403).send('Forbidden');  
  
    req.user = user;  
  
    next();  
  });  
}
```

88. How do you use React Context API for global state management?

```
const MyContext = createContext();

function MyProvider({ children }) {

  const [state, setState] = useState("value");

  return <MyContext.Provider
value={state}>{children}</MyContext.Provider>;

}
```

89. How do you test a React component using Jest and React Testing Library?

```
import { render, screen } from '@testing-library/react';
import MyComponent from './MyComponent';

test('renders component', () => {

  render(<MyComponent />);

  expect(screen.getByText(/hello/i)).toBeInTheDocument();

});
```

90. How do you test an Express.js API using Jest and Supertest?

```
const request = require('supertest');

const app = require('../server');

test('GET /api/data', async () => {
```

```
const res = await request(app).get('/api/data');  
  
expect(res.statusCode).toBe(200);  
  
});
```

91. How do you secure MongoDB connections in production?

- Use environment variables for credentials, enable authentication, whitelist IPs, and disable public access.

92. What are the differences between `useState` and `useReducer`?

- `useState` is simpler for small state changes, while `useReducer` is better for complex state logic with multiple updates.

93. How do you handle large amounts of data in React?

- Virtualize lists with `react-window` or `react-virtualized` to improve performance.

94. How do you implement email verification in a MERN app?

- Send a verification email with a token link, store the token in the DB, and verify it upon user click.

95. How do you implement a multi-step form in React?

- Manage the **step state** and render different components based on the step index.

```
const [step, setStep] = useState(1);  
  
return step === 1 ? <StepOne /> : <StepTwo />;
```

96. How do you handle background jobs in a MERN stack application?

- Use **Bull.js** (Redis-based job queue) for handling background tasks like email sending.

```
const Queue = require('bull');  
  
const emailQueue = new Queue('emailQueue');  
  
emailQueue.process(async (job) => {  
  console.log(`Sending email to ${job.data.email}`);  
});  
  
emailQueue.add({ email: 'user@example.com' });
```

97. What is the best way to deploy a MERN stack app on AWS?

- **Frontend:** Deploy React on **S3 + CloudFront**.
- **Backend:** Use **EC2 (PM2)**, **Elastic Beanstalk**, or **Lambda (serverless)**.
- **Database:** Use **MongoDB Atlas (managed)** or **self-hosted MongoDB on EC2**.

- **Authentication:** Use **Cognito**, **OAuth**, or **Firebase Auth**.
- **CI/CD:** Use **GitHub Actions**, **AWS CodePipeline**, or **Jenkins**.

98. How do you handle database schema migrations in MongoDB?

- Use **Mongoose versioning** or **MongoDB migrations libraries like migrate-mongo**.

```
npm install -g migrate-mongo
```

```
migrate-mongo init
```

```
migrate-mongo create add_new_field
```

```
migrate-mongo up
```



99. How do you integrate third-party payment gateways like Stripe in a MERN app?

- Install Stripe SDK and create a backend API:

```
const stripe = require('stripe')('your_secret_key');  
app.post('/checkout', async (req, res) => {  
  const session = await stripe.checkout.sessions.create({  
    payment_method_types: ['card'],  
    line_items: [{ price_data: { currency: 'usd',  
unit_amount: 1000, product_data: { name: 'Course' } } },  
quantity: 1 }],
```

```
mode: 'payment',  
  
success_url: 'https://yourapp.com/success',  
  
cancel_url: 'https://yourapp.com/cancel',  
  
});  
  
res.json({ id: session.id });  
  
});
```

100. What are the key considerations for making a MERN stack application production-ready?

- **Security:** Use JWT, HTTPS, helmet.js, CORS restrictions.
 - **Performance:** Optimize MongoDB queries, React rendering, and API response times.
 - **Scalability:** Use Docker, Kubernetes, or AWS services.
 - **Logging & Monitoring:** Use Winston, Morgan, Sentry, or Datadog.
 - **CI/CD:** Automate deployment with GitHub Actions, Jenkins, or AWS CodePipeline.
-