Full Stack Developer Notes

Full Stack Developer Placement Preparation (2025)

Table of Contents

React Fundamentals

- Component-based UI library by Facebook
- JSX syntax (JavaScript XML) to write UI elements
- Components: Functional components (modern), Class components (older, lifecycle methods)
- Props & State
- Event handling (onClick, onChange)
- Conditional rendering (&&, ternary)
- Lists & keys (map function)
- Controlled vs uncontrolled components
- React Hooks: useState, useEffect, useContext
- React lifecycle (class components): componentDidMount, componentDidUpdate, componentWillUnmount
- React Router for SPA navigation
- Performance: React.memo, lazy loading

Sample React Functional Component:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

JavaScript Fundamentals
- Data types: string, number, boolean, null, undefined, symbol, object
- Variables: var, let, const
- Functions & arrow functions
- Scope & closures
- this keyword context
- Prototypal inheritance
- Template literals (`${var}`)
- ES6 modules (import / export)

Advanced JavaScript Concepts
- Destructuring objects & arrays
- Spread/rest operators (...)
- Default function parameters
- Classes & inheritance
- Promises & async/await
- Event loop, microtasks/macrotasks
- Debounce & throttle techniques
- Memory leaks & cleanup (important in React)

Asynchronous JavaScript
- Callbacks vs Promises vs Async/Await
- Event loop explained

- Fetch API and Axios for HTTP requests
- Handling asynchronous side effects in React (useEffect with async functions)

Node.js Basics
- Node.js is JavaScript runtime outside browser
- Event-driven, non-blocking I/O
- REPL (Read-Eval-Print Loop)
- Core modules: fs, path, http

Node.js Modules & Core APIs
- CommonJS modules: require(), module.exports
- File system operations: reading/writing files
- Creating HTTP servers
- Streams and buffers

NPM & Package Management
- What is NPM?
- package.json configuration
- Installing dependencies locally & globally
- Semantic versioning (^, ~)
- Popular packages: express, mongoose, cors, dotenv

Express.js Basics
- Minimal Node.js web framework
- Creating servers and defining routes
- Middleware functions
- Sending JSON responses

Express Middleware
- Types: application-level, router-level, error-handling
- Built-in middleware: express.json(), express.urlencoded()
- Custom middleware example:

```
function logger(req, res, next) {
  console.log(`${req.method} ${req.url}`);
  next();
}
app.use(logger);
```

Routing & REST APIs

- HTTP methods: GET, POST, PUT, DELETE
- Route parameters & query strings
- RESTful API principles
- Sending JSON responses
- Organizing routes with Express Router

Database Integration (MongoDB)

- NoSQL database, document-oriented
- Mongoose: ODM (Object Data Modeling) library
- Schema & models
- Connecting to MongoDB via mongoose.connect()
- CRUD operations example:

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

const user = new User({ name: 'John', email: 'john@example.com' });
await user.save();
```

Authentication & Security

- JWT (JSON Web Tokens) authentication flow
- Password hashing with bcrypt
- Environment variables with dotenv
- Security middleware: cors, helmet
- Input validation & sanitization

Error Handling & Debugging

- Try-catch with async-await
- Express error handling middleware:

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

- Debugging with console.log, Node Inspector, VSCode debugger
- Proper logging strategies

Interview Questions (React + JS + Node + Express)

React
1. What is JSX?
2. Difference between state and props?
3. Explain React hooks useState and useEffect.
4. How does React's virtual DOM work?
5. What is context API?
6. What are controlled components?
7. How do you optimize React app performance?
8. What is React Router?
9. Explain lifting state up.
10. How do you handle forms in React?

JavaScript
1. What is closure?
2. Difference between var, let, and const.
3. Explain prototypal inheritance.
4. What is event delegation?
5. Difference between synchronous and asynchronous code?
6. How does the event loop work?
7. Explain promises and async/await.
8. What is hoisting?

Node.js
1. What is Node.js?
2. Explain event loop in Node.js.
3. What are streams?
4. What is callback hell?
5. What is package.json used for?
6. How to create a simple web server?
7. How to handle errors in Node.js?

Express.js
1. What is middleware?

2. How do you create routes?
3. How do you parse JSON in Express?
4. How to handle errors in Express?
5. What is CORS and how to enable it?
6. How to secure an Express app?
7. How to implement authentication?

Sample Project Folder Structure

```
my-fullstack-app/
|
 client/                # React frontend
|    public/
|    src/
|        components/
|        pages/
|        App.js
|        index.js
|
 server/                # Node + Express backend
|    controllers/
|    models/
|    routes/
|    middleware/
|    app.js
|    server.js
|
 .env                   # Environment variables
 package.json
 README.md
```

Common Terminal Commands

React

```
npx create-react-app client
cd client
npm start
npm run build
```

Node.js + Express

```
npm init -y
npm install express mongoose dotenv cors
node server.js
```

Git

```
git init
git add .
git commit -m "Initial commit"
git push origin main
```

Final Tips

- Practice building full projects combining React frontend and Node/Express backend.
- Keep your GitHub active and clean.
- Deploy your projects on Vercel, Netlify (frontend) and Render, Heroku (backend).
- Prepare to explain your projects during interviews.
- Revise interview questions regularly.