List of Node.js interview questions covering various aspects:

1. **What is Node.js?**
2. **Explain the event-driven architecture of Node.js.**
3. **What is NPM? How do you use it?**
4. **What is the difference between require() and import in Node.js?**
5. **What is a callback function? Why are callbacks used in Node.js?**
6. **What is the role of the package.json file in Node.js projects?**
7. **Explain the concept of streams in Node.js.**
8. **What are the differences between synchronous and asynchronous programming in Node.js?**
9. **What is the purpose of process.argv in Node.js?**
10. **How does error handling work in Node.js?**
11. **Explain the concept of middleware in Express.js.**
12. **What is the difference between app.get() and app.post() in Express.js?**
13. **What is the significance of module.exports in Node.js?**
14. **How can you prevent callback hell in Node.js?**
15. **What are promises in Node.js? How do they differ from callbacks?**
16. **What is the purpose of the fs module in Node.js?**
17. **Explain the purpose of the path module in Node.js.**
18. **What is clustering in Node.js? Why is it useful?**
19. **Explain the concept of non-blocking I/O in Node.js.**
20. **What is the purpose of the util module in Node.js?**
21. **What are the differences between Node.js and traditional server-side languages like PHP or Ruby?**
22. **Explain the concept of the event loop in Node.js.**
23. **What are child processes in Node.js? When would you use them?**
24. **How can you debug a Node.js application?**
25. **Explain how you can handle file uploads in Node.js.**
26. **What is Express.js? How does it relate to Node.js?**
27. **Explain the difference between app.use() and app.get() in Express.js.**
28. **What is middleware and how does it work in Express.js?**
29. **What are routing parameters in Express.js?**
30. **How do you handle authentication in Node.js/Express.js applications?**
31. **Explain the concept of templating engines in Express.js.**
32. **What is JWT (JSON Web Token) and how is it used for authentication in Node.js?**
33. **What are the benefits of using a framework like Express.js compared to raw Node.js?**
34. **How do you handle errors globally in Express.js?**
35. **What is CORS (Cross-Origin Resource Sharing) and how do you enable it in an Express.js application?**
36. **Explain the concept of middleware chaining in Express.js.**
37. **What is the purpose of the body-parser middleware in Express.js?**
38. **How do you handle sessions in Express.js?**
39. **What is clustering in Node.js and how do you implement it?**
40. **Explain the difference between setTimeout() and setImmediate() in Node.js.**
41. **What is the purpose of the crypto module in Node.js?**
42. **Explain the concept of event emitters in Node.js.**
43. **How do you handle environment variables in Node.js applications?**
44. **What is RESTful routing and how is it implemented in Express.js?**
45. **What is WebSockets and how do you implement them in Node.js?**
46. **What are some security best practices to follow in Node.js applications?**
47. **Explain the purpose of the os module in Node.js.**
48. **How do you handle file uploads securely in Node.js?**
49. **What is the purpose of the http module in Node.js?**
50. **What is the difference between using == and === in JavaScript?**
51. **What is the purpose of the cluster module in Node.js? How does it facilitate load balancing?**
52. **Explain the concept of middleware error handling in Express.js. How can you create custom error handling middleware?**
53. **What is the purpose of the net module in Node.js? How do you create TCP servers and clients using this module?**
54. **What is a Promise in JavaScript? How do you handle Promise rejections?**
55. **What is async/await in Node.js? How does it simplify asynchronous code compared to using Promises?**
56. **Explain the purpose of the url module in Node.js. How do you parse and format URLs using this module?**
57. **What are the differences between HTTP and HTTPS? How do you set up an HTTPS server in Node.js?**
58. **What is GraphQL and how does it differ from RESTful APIs? How can you implement GraphQL servers in Node.js?**
59. **Explain the purpose of the querystring module in Node.js. How do you parse query strings in URLs?**
60. **What is the purpose of the cluster module in Node.js? How do you create a cluster of Node.js processes?**
61. **How do you handle authentication and authorization in a Node.js application?**
62. **What is the role of the assert module in Node.js? How do you use it for unit testing?**
63. **Explain the concept of middleware mounting in Express.js.**
64. **What is the purpose of the Buffer class in Node.js? How do you use it for handling binary data?**
65. **What is the purpose of the util.promisify function in Node.js? How do you use it to convert callback-based functions to Promise-based functions?**
66. **How do you deploy Node.js applications to production environments?**
67. **What is the purpose of the child\_process module in Node.js? How do you spawn child processes?**
68. **What are the differences between setTimeout() and setInterval() in Node.js?**
69. **How do you handle sessions in Express.js applications? What are some session storage options?**
70. **Explain the purpose of the os module in Node.js. How do you retrieve system information using this module?**

4.What is the difference between require() and import in Node.js?

Using require():

javascript

Copy code

// Importing a module using require()

const fs = require('fs');

// Using the imported module

fs.readFile('example.txt', 'utf8', (err, data) => {

  if (err) {

    console.error(err);

    return;

  }

  console.log(data);

});

Using import:

javascript

Copy code

// Importing a module using import

import fs from 'fs';

// Using the imported module

fs.readFile('example.txt', 'utf8', (err, data) => {

  if (err) {

    console.error(err);

    return;

  }

  console.log(data);

});

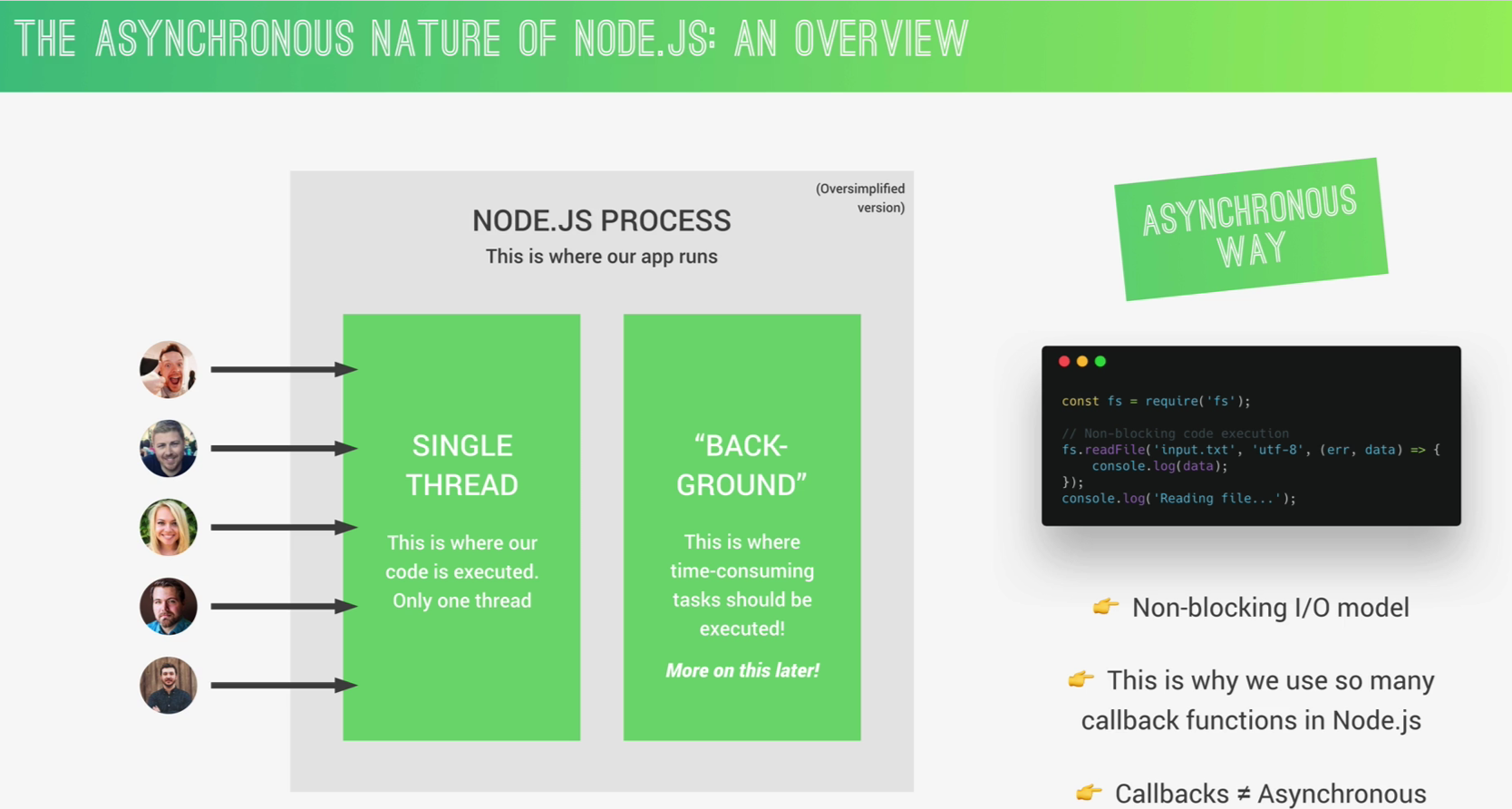
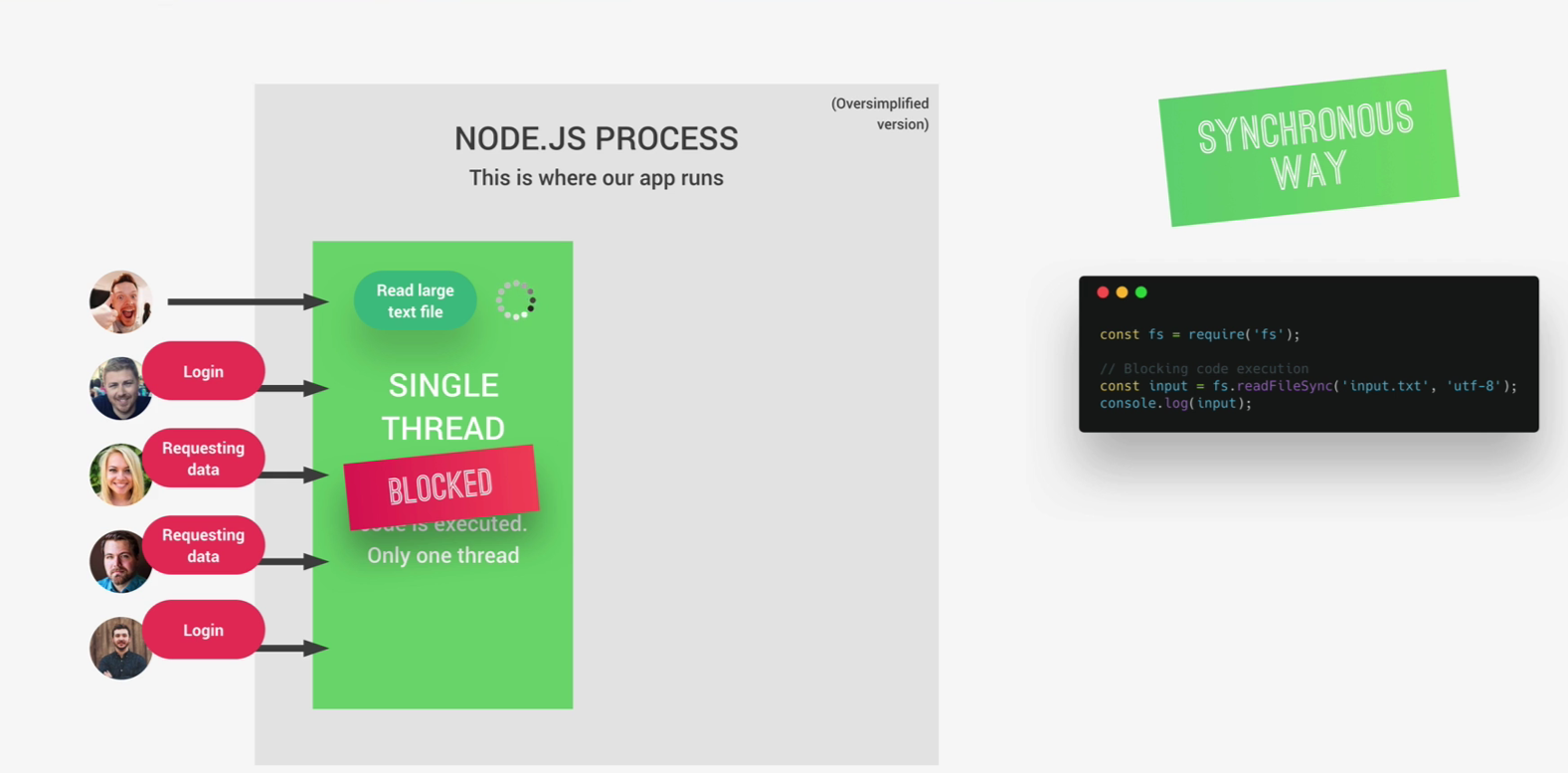
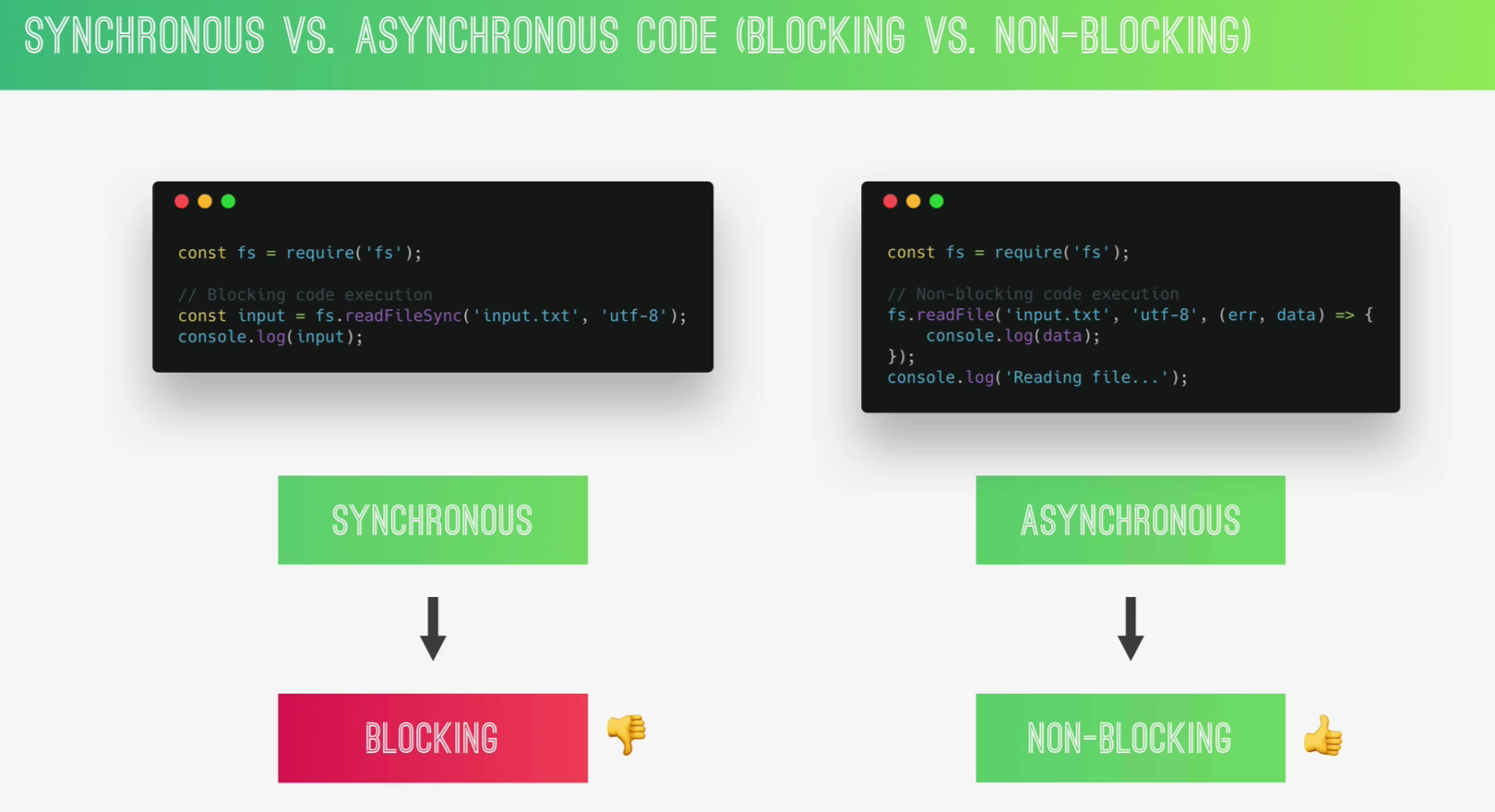
**7.What is the role of the package.json file in Node.js projects?**

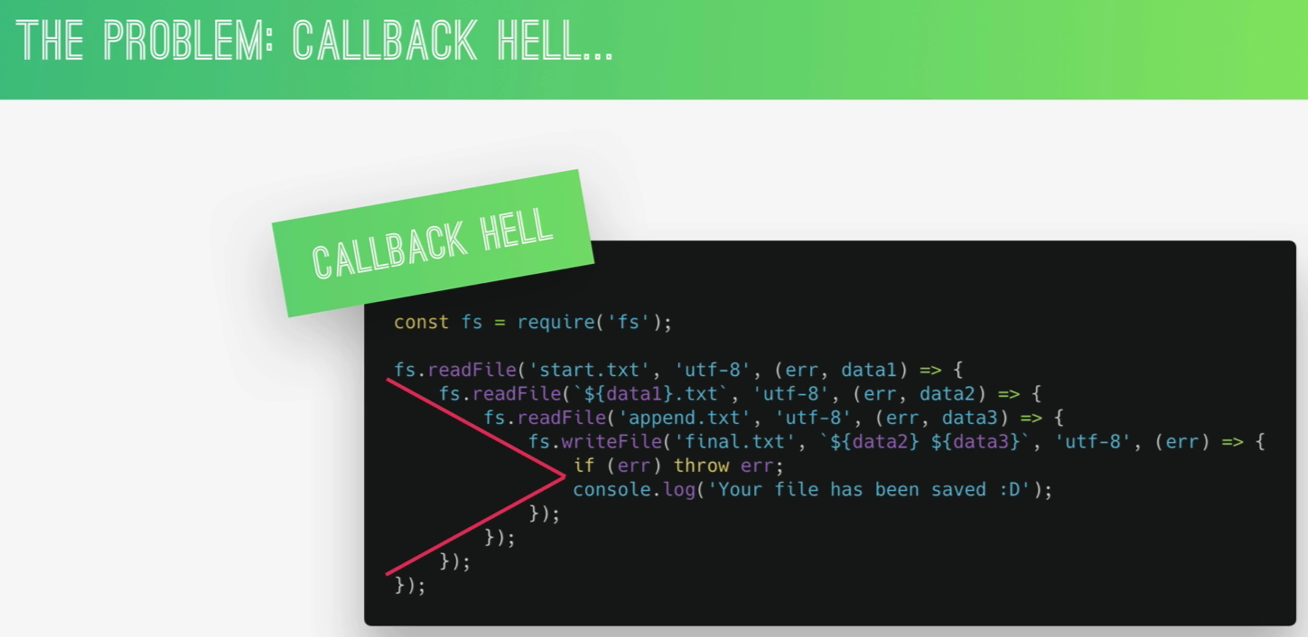
Overall, **package.json** serves as a central configuration and metadata file for Node.js projects, providing essential information about the project's dependencies, scripts, metadata, configuration, and versioning. It's a fundamental component of Node.js development and npm ecosystem.

**8.What are the differences between synchronous and asynchronous programming in Node.js?**

**9.What is the purpose of process.argv in Node.js?**

Images Learning





Callback hell ek programming term hai jo asynchronous JavaScript code ko describe karta hai jab code mein multiple nested callbacks hote hain, aur yeh nested callbacks unreadable aur difficult to manage code ko create karte hain. Callback hell ka upyog tab hota hai jab hum asynchronous operations jaise ki data fetching, file reading, ya network requests ko handle kar rahe hote hain.

Yeh kuch khaas characteristics hai callback hell ke:

1. **Nested Callbacks**: Callback functions ke andar aur andar aur callbacks ka chain bana rehta hai, jisse code ki readability aur maintainability khatre mein aa jaati hai.
2. **Pyramid of Doom**: Callbacks ka structure pyramid ke jaise hota hai, jismein har ek nested level ek aur indentation level badhata hai, jisse code padhna mushkil ho jaata hai.
3. **Error Handling Complexity**: Error handling bhi complicated ho jaata hai, kyunki har ek callback ke andar error ko handle karna padta hai, aur error propagation ke chances badh jaate hain.
4. **Debugging Challenges**: Callback hell mein code ko debug karna mushkil ho jaata hai, kyunki har ek asynchronous operation ke liye multiple levels of indentation aur callbacks hote hain.

Callback hell se bachne ke liye modern JavaScript mein Promise, async/await, aur libraries jaise ki RxJS, async.js ka upyog kiya jaata hai. Yeh approaches code ko readable, maintainable, aur error-prone banate hain.

Top of Form

Callback hell se bachne ke liye kuch upay hai:

1. **Promises**: Promises asynchronous JavaScript code ko handle karne ke liye ek better approach hai. Promises ko istemal karke hum multiple nested callbacks se bach sakte hain aur code ko readable aur maintainable bana sakte hain. Promises ke saath **.then()** aur **.catch()** ka upyog karke hum code ko sequential aur error handling ke saath likh sakte hain.
2. **Async/Await**: Async/Await JavaScript mein ek powerful feature hai jo Promises ke saath milta hai. Async/Await ka upyog karke hum asynchronous code ko synchronous code ki tarah likh sakte hain, jisse code ki readability aur maintainability improve hoti hai. Async/Await functions ko define karne ke liye **async** keyword ka upyog hota hai aur async functions ke andar **await** keyword ka upyog hota hai promises ko resolve karne ke liye.
3. **Modularization**: Code ko modularize karke, yaani ki chhote chhote functions mein divide karke likhne se callback hell se bacha ja sakta hai. Har ek function ek specific task ko handle kare aur agar possible ho toh callback functions ko alag se define kare.
4. **Libraries/Frameworks**: Kabhi kabhi libraries aur frameworks jaise ki RxJS, async.js, aur lodash ka upyog karke bhi callback hell se bacha ja sakta hai. In libraries mein functions jaise ki **map**, **reduce**, **filter** aadi uplabdh hote hain jo asynchronous operations ko handle karne ke liye use kiye ja sakte hain.
5. **Error Handling**: Sahi tareeke se error handling karna bhi callback hell se bachne ka ek tareeka hai. Error handling ko simple aur effective banane ke liye try-catch blocks ka upyog kiya ja sakta hai.
6. **ES6+ Features**: Modern JavaScript features jaise ki arrow functions, template literals, destructuring, spread/rest operators aadi ka upyog karke bhi code ki readability improve ki ja sakti hai aur callback hell se bacha ja sakta hai

# Server.listen() :

