1. Which of the following is a popular Node.js framework?

- A) Django

- B) Laravel

- C) Express.js

- D) Ruby on Rails

Answer: C) Express.js

2. Which Node.js framework is known for its modular architecture and TypeScript support?

- A) Koa.js

- B) Sails.js

- C) Hapi.js

- D) NestJS

Answer: D) NestJS

3. What does Koa.js primarily use for handling asynchronous code?

- A) Promises

- B) Callbacks

- C) Async/Await

- D) Generators

Answer: C) Async/Await

4. Which Node.js framework is designed for real-time applications and follows the MVC pattern?

- A) Express.js

- B) Sails.js

- C) Koa.js

- D) NestJS

Answer: B) Sails.js

5. Which of the following Node.js frameworks provides a robust plugin system?

- A) Hapi.js

- B) Express.js

- C) Koa.js

- D) NestJS

Answer: A) Hapi.js

6. What is Express.js primarily used for?

- A) Building desktop applications

- B) Building web and mobile applications

- C) Building operating systems

- D) Building database management systems

Answer: B) Building web and mobile applications

7. Which object in Express.js is used to handle HTTP requests?

- A) res

- B) req

- C) app

- D) server

Answer: B) req

8. Which method in Express.js is used to listen for incoming requests?

- A) app.get()

- B) app.listen()

- C) app.on()

- D) app.start()

Answer: B) app.listen()

9. How do you install Express.js using npm?

- A) npm install express.js

- B) npm express install

- C) npm install express

- D) npm express.js install

Answer: C) npm install express

10. What is the command to create a new Express.js application using the Express Generator?

- A) express-app

- B) express-generator

- C) npx express-generator

- D) npm express

Answer: C) npx express-generator

11. Which middleware in Express.js is used to serve static files?

- A) express.static

- B) express.files

- C) express.serve

- D) express.middleware

Answer: A) express.static

12. Where do you typically place static files in an Express.js project?

- A) In the views directory

- B) In the public directory

- C) In the routes directory

- D) In the controllers directory

Answer: B) In the public directory

13. How do you serve static files from a folder named "assets"?

- A) app.use('/static', express.assets('assets'));

- B) app.use('/static', express.serve('assets'));

- C) app.use('/static', express.static('assets'));

- D) app.use('/assets', express.static('public'));

Answer: C) app.use('/static', express.static('assets'))

14. Which of the following is true about static files in Express.js?

- A) They are dynamically generated on each request

- B) They are served as they are from the server

- C) They cannot be cached

- D) They are always served from the root directory

Answer: B) They are served as they are from the server

15. How would you serve multiple static directories in Express.js?

- A) app.use(express.static('dir1', 'dir2'));

- B) app.use([express.static('dir1'), express.static('dir2')]);

- C) app.use(express.static(['dir1', 'dir2']));

- D) app.use(express.static('dir1')); app.use(express.static('dir2'));

Answer: D) app.use(express.static('dir1')); app.use(express.static('dir2'));

16. Which method is used to define a route in Express.js that handles GET requests?

- A) app.get()

- B) app.post()

- C) app.put()

- D) app.route()

Answer: A) app.get()

17. How do you define a route in Express.js that handles POST requests?

- A) app.send()

- B) app.fetch()

- C) app.post()

- D) app.push()

Answer: C) app.post()

18. What does `req.params` represent in Express.js?

- A) Query parameters

- B) Body parameters

- C) Route parameters

- D) Header parameters

Answer: C) Route parameters

19. Which method is used to redirect a request to another route in Express.js?

- A) res.send()

- B) res.redirect()

- C) res.route()

- D) res.forward()

Answer: B) res.redirect()

20. How would you define a route in Express.js that responds to all HTTP methods?

- A) app.all()

- B) app.any()

- C) app.every()

- D) app.route()

Answer: A) app.all()

21. Which method is used to define a route that matches any request path?

- A) app.all('', handler)

- B) app.route('', handler)

- C) app.use('', handler)

- D) app.get('', handler)

Answer: D) app.get('', handler)

22. How do you define a route with a parameter in Express.js?

- A) app.get('/user/id', handler)

- B) app.get('/user/:id', handler)

- C) app.get('/user@id', handler)

- D) app.get('/user=id', handler)

Answer: B) app.get('/user/:id', handler)

23. Which Express.js method allows you to define multiple routes that share the same path?

- A) app.route()

- B) app.group()

- C) app.paths()

- D) app.use()

Answer: A) app.route()

24. How do you define a route that handles PUT requests to update a resource?

- A) app.patch()

- B) app.update()

- C) app.put()

- D) app.modify()

Answer: C) app.put()

25. Which method allows you to chain route handlers for a specific route in Express.js?

- A) app.use()

- B) app.next()

- C) app.chain()

- D) app.route()

Answer: D) app.route()

### Route Paths

26. How would you define a route that matches any subpath of a given route in Express.js?

- A) app.use('/path/', handler)

- B) app.get('/path/', handler)

- C) app.all('/path/', handler)

- D) All of the above

Answer: D) All of the above

27. What is the correct way to define a route that matches a URL path starting with `/api` and ending with `/users`?

- A) app.get('/api//users', handler)

- B) app.get('/api/users', handler)

- C) app.get('/api/:param/users', handler)

- D) app.get('/api/users', handler)

Answer: C) app.get('/api/:param/users', handler)

28. In Express.js, how would you define a route that matches `/user123`, where `123` is a dynamic value?

- A) app.get('/user:id', handler)

- B) app.get('/user/:id', handler)

- C) app.get('/user/{id}', handler)

- D) app.get('/user(@id)', handler)

Answer: B) app.get('/user/:id', handler)

29. Which wildcard character is used in route paths to match any single character in Express.js?

- A)

- B) ?

- C) +

- D) .

Answer: B) ?

30. How do you define a route that matches all HTTP methods for a specific path?

- A) app.use(path, handler)

- B) app.route(path, handler)

- C) app.all(path, handler)

- D) app.get(path, handler)

Answer: C) app.all(path, handler)

31. Which method is used to retrieve route parameters in Express.js?

- A) req.get()

- B) req.params()

- C) req.params

- D) req.route()

Answer: C) req.params

32. How do you define an optional route parameter in Express.js?

- A) app.get('/user/:id?', handler)

- B) app.get('/user/:id', handler)

- C) app.get('/user/[:id]', handler)

- D) app.get('/user/@id', handler)

Answer: A) app.get('/user/:id?', handler)

33. What does `req.params` contain in Express.js?

- A) Route parameters as key-value pairs

- B) Query string parameters as key-value pairs

- C) HTTP headers as key-value pairs

- D) Body parameters as key-value pairs

Answer: A) Route parameters as key-value pairs

34. How would you define a route in Express.js that captures two parameters, `id` and `name`?

- A) app.get('/user/:id/:name', handler)

- B) app.get('/user/:id-:name', handler)

- C) app.get('/user/@id/@name', handler)

- D) app.get('/user/:id@name', handler)

Answer: A) app.get('/user/:id/:name', handler)

35. Which of the following is true about route parameters in Express.js?

- A) They are case-sensitive

- B) They must always be numeric

- C) They are optional by default

- D) They must always be alphabetic

Answer: A) They are case-sensitive

36. How do you define a route handler in Express.js that sends a JSON response?

- A) res.sendJSON()

- B) res.json()

- C) res.send()

- D) res.response()

Answer: B) res.json()

37. Which of the following is true about route handlers in Express.js?

- A) They must always send a response

- B) They can be asynchronous functions

- C) They cannot access the `req` object

- D) They must always use the `next()` function

Answer: B) They can be asynchronous functions

38. How would you handle multiple route handlers for a single route in Express.js?

- A) app.get(path, [handler1, handler2])

- B) app.get(path, handler1).next(handler2)

- C) app.get(path, handler1, handler2)

- D) app.get(path).use(handler1, handler2)

Answer: C) app.get(path, handler1, handler2)

39. What does the `next()` function do in a route handler?

- A) It sends the response to the client

- B) It passes control to the next middleware or route handler

- C) It ends the request-response cycle

- D) It logs the request details

Answer: B) It passes control to the next middleware or route handler

40. How do you define a route handler that returns an HTML response in Express.js?

- A) res.html()

- B) res.sendHTML()

- C) res.send()

- D) res.render()

Answer: C) res.send()

41. Which method is used to send a plain text response in Express.js?

- A) res.json()

- B) res.send()

- C) res.text()

- D) res.write()

Answer: B) res.send()

42. How do you set the status code for a response in Express.js?

- A) res.statusCode()

- B) res.setStatus()

- C) res.status()

- D) res.code()

Answer: C) res.status()

43. Which method is used to send a JSON response with a specific status code in Express.js?

- A) res.json(status, data)

- B) res.status(data).json()

- C) res.send(status, data)

- D) res.status().json(data)

Answer: B) res.status(status).json(data)

44. How do you send an HTTP response with a 404 status code in Express.js?

- A) res.notFound()

- B) res.status(404).send()

- C) res.statusCode(404).send()

- D) res.status(404).send("Not Found")

Answer: D) res.status(404).send("Not Found")

45. Which method is used to send a file as a response in Express.js?

- A) res.file()

- B) res.sendFile()

- C) res.download()

- D) res.send()

Answer: B) res.sendFile()

46. How do you set a custom HTTP header in the response in Express.js?

- A) res.header()

- B) res.setHeader()

- C) res.headers()

- D) res.set()

Answer: D) res.set()

47. Which method is used to send a JSON object as a response in Express.js?

- A) res.sendJSON()

- B) res.json()

- C) res.sendObject()

- D) res.send()

Answer: B) res.json()

48. How do you send an HTTP response with a 500 status code in Express.js?

- A) res.error()

- B) res.sendError()

- C) res.status(500).send()

- D) res.status(500).json()

Answer: C) res.status(500).send()

49. Which method is used to end the response process in Express.js?

- A) res.end()

- B) res.finish()

- C) res.stop()

- D) res.close()

Answer: A) res.end()

50. How would you send a response with an attachment in Express.js?

- A) res.attach()

- B) res.attachment()

- C) res.sendAttachment()

- D) res.download()

Answer: D) res.download()

51. What is middleware in Express.js?

- A) Functions that handle HTTP requests

- B) Functions that process requests between the client and the server

- C) Functions that serve static files

- D) Functions that only handle errors

Answer: B) Functions that process requests between the client and the server

52. Which of the following is an example of application-level middleware?

- A) app.use('/route', handler)

- B) app.get('/route', handler)

- C) router.use('/route', handler)

- D) app.all('/route', handler)

Answer: A) app.use('/route', handler)

53. How do you create router-level middleware in Express.js?

- A) app.router.use()

- B) router.use()

- C) app.use(router())

- D) router.route()

Answer: B) router.use()

54. What is the primary purpose of error-handling middleware in Express.js?

- A) To process requests after the response has been sent

- B) To catch and handle errors in the application

- C) To handle static files

- D) To route requests

Answer: B) To catch and handle errors in the application

55. Which of the following is required to define error-handling middleware in Express.js?

- A) app.error()

- B) app.catch()

- C) A function with four parameters: err, req, res, next

- D) A function with two parameters: req, res

Answer: C) A function with four parameters: err, req, res, next

56. What does the `next()` function do in middleware?

- A) It sends the response to the client

- B) It passes control to the next middleware function

- C) It terminates the request-response cycle

- D) It logs the request details

Answer: B) It passes control to the next middleware function

57. How do you define third-party middleware in Express.js?

- A) Using app.use(thirdPartyMiddleware)

- B) Using app.thirdParty(thirdPartyMiddleware)

- C) Using app.middleware(thirdPartyMiddleware)

- D) Using router.use(thirdPartyMiddleware)

Answer: A) Using app.use(thirdPartyMiddleware)

58. Which of the following is an example of third-party middleware in Express.js?

- A) express.static()

- B) body-parser

- C) app.use()

- D) res.json()

Answer: B) body-parser

59. How do you define middleware that runs for every request to the server?

- A) app.all()

- B) app.every()

- C) app.use()

- D) app.middleware()

Answer: C) app.use()

60. In Express.js, where is middleware typically defined?

- A) After defining routes

- B) Before defining routes

- C) Inside route handlers

- D) In a separate file from the application

Answer: B) Before defining routes

61. What is the correct order of the middleware lifecycle in Express.js?

- A) Initialization → Middleware execution → Route matching → Response generation

- B) Middleware execution → Initialization → Route matching → Response generation

- C) Initialization → Route matching → Middleware execution → Response generation

- D) Route matching → Middleware execution → Initialization → Response generation

Answer: C) Initialization → Route matching → Middleware execution → Response generation

62. How can middleware be conditionally executed in Express.js?

- A) Using if statements inside the middleware function

- B) By passing a condition as the first argument to app.use()

- C) By using app.conditionalMiddleware()

- D) Middleware cannot be conditionally executed

Answer: A) Using if statements inside the middleware function

63. What happens if `next()` is not called in middleware?

- A) The request is immediately terminated

- B) The next middleware or route handler is automatically executed

- C) The request will hang and not receive a response

- D) The response is sent with a default message

Answer: C) The request will hang and not receive a response

64. How do you skip the remaining middleware and go straight to the error-handling middleware?

- A) By passing an error object to `next()`

- B) By calling `next(false)`

- C) By using `app.error()`

- D) By calling `res.end()`

Answer: A) By passing an error object to `next()`

65. Which of the following middleware is used to handle CORS in Express.js?

- A) express.json()

- B) express.static()

- C) cors()

- D) body-parser

Answer: C) cors()

66. What is application-level middleware?

- A) Middleware that is applied globally to all requests in the application

- B) Middleware that is specific to a single route

- C) Middleware that handles static files

- D) Middleware that handles errors

Answer: A) Middleware that is applied globally to all requests in the application

67. How do you apply middleware to a specific route?

- A) app.use(middleware).route(path)

- B) app.use(path, middleware)

- C) router.use(path, middleware)

- D) app.route(path, middleware)

Answer: B) app.use(path, middleware)

68. Which of the following is an example of application-level middleware in Express.js?

- A) app.use(express.json())

- B) app.use('/api', router)

- C) router.use(middleware)

- D) app.error(middleware)

Answer: A) app.use(express.json())

69. How do you apply multiple middleware functions to a single route?

- A) app.use(middleware1).use(middleware2).route(path)

- B) app.use([middleware1, middleware2], path)

- C) app.use(path, middleware1, middleware2)

- D) app.get(path, middleware1, middleware2)

Answer: D) app.get(path, middleware1, middleware2)

70. How can you define middleware that runs only for a specific HTTP method and route?

- A) app.use(method, path, middleware)

- B) app.method(path, middleware)

- C) app.use(path, middleware)

- D) router.method(path, middleware)

Answer: B) app.method(path, middleware)

71. What is router-level middleware in Express.js?

- A) Middleware that is specific to a single route

- B) Middleware that is applied globally to all requests in the application

- C) Middleware that is applied to all routes in a specific router instance

- D) Middleware that handles errors

Answer: C) Middleware that is applied to all routes in a specific router instance

72. How do you create a router instance in Express.js?

- A) const router = new express.Router()

- B) const router = express.Router()

- C) const router = require('express').Router()

- D) const router = express.createRouter()

Answer: B) const router = express.Router()

73. How do you apply middleware to a router instance?

- A) app.use(middleware)

- B) router.use(middleware)

- C) app.router(middleware)

- D) router.middleware(middleware)

Answer: B) router.use(middleware)

74. Which of the following is true about router-level middleware?

- A) It can only be applied to a single route

- B) It is applied to all routes in the application

- C) It is applied to all routes in a specific router instance

- D) It cannot be applied to a specific route

Answer: C) It is applied to all routes in a specific router instance

75. How do you apply router-level middleware to a specific route in Express.js?

- A) router.use(path, middleware)

- B) router.route(path).use(middleware)

- C) app.use(path, router)

- D) router.get(path, middleware)

Answer: D) router.get(path, middleware)

76. What is the purpose of error-handling middleware in Express.js?

- A) To handle errors that occur during the request-response cycle

- B) To catch and handle all HTTP requests

- C) To log requests and responses

- D) To serve static files

Answer: A) To handle errors that occur during the request-response cycle

77. Which of the following is required to define error-handling middleware?

- A) A function with two parameters: req, res

- B) A function with three parameters: err, req, res

- C) A function with four parameters: err, req, res, next

- D) A function with one parameter: err

Answer: C) A function with four parameters: err, req, res, next

78. How do you pass an error to the error-handling middleware in Express.js?

- A) next(err)

- B) app.error(err)

- C) res.error(err)

- D) router.error(err)

Answer: A) next(err)

79. Where should error-handling middleware be placed in the middleware stack?

- A) At the beginning of the stack

- B) After all other middleware and routes

- C) In the middle of the stack

- D) Before all other middleware and routes

Answer: B) After all other middleware and routes

80. How can you send a custom error response from the error-handling middleware?

- A) res.status(500).send('Custom Error')

- B) next('Custom Error')

- C) app.error('Custom Error')

- D) res.error('Custom Error')

Answer: A) res.status(500).send('Custom Error')

81. What is third-party middleware in Express.js?

- A) Middleware that is built into Express.js

- B) Middleware that is created by the application developer

- C) Middleware that is provided by external libraries

- D) Middleware that handles HTTP requests

Answer: C) Middleware that is provided by external libraries

82. Which of the following is an example of third-party middleware?

- A) app.use(express.static())

- B) app.use(express.json())

- C) app.use(bodyParser.json())

- D) app.use(router())

Answer: C) app.use(bodyParser.json())

83. How do you install third-party middleware in Express.js?

- A) Using npm install

- B) Using express install

- C) Using middleware install

- D) Using app.use()

Answer: A) Using npm install

84. Which third-party middleware is commonly used for parsing

JSON bodies in Express.js?

- A) body-parser

- B) morgan

- C) cors

- D) express.json()

Answer: A) body-parser

85. How do you apply third-party middleware to an Express.js application?

- A) app.use(thirdPartyMiddleware())

- B) app.use(thirdPartyMiddleware)

- C) app.middleware(thirdPartyMiddleware())

- D) app.thirdParty(thirdPartyMiddleware)

Answer: A) app.use(thirdPartyMiddleware())

86. Which third-party middleware is commonly used for logging HTTP requests in Express.js?

- A) body-parser

- B) morgan

- C) cors

- D) express.json()

Answer: B) morgan

87. How do you configure third-party middleware in Express.js?

- A) Pass configuration options as the second argument to app.use()

- B) Pass configuration options as the first argument to app.use()

- C) Pass configuration options to the middleware function

- D) Third-party middleware cannot be configured

Answer: C) Pass configuration options to the middleware function

88. Which third-party middleware is commonly used for enabling CORS in Express.js?

- A) body-parser

- B) morgan

- C) cors

- D) express.json()

Answer: C) cors

89. How do you apply third-party middleware to specific routes in Express.js?

- A) app.use(thirdPartyMiddleware).route(path)

- B) app.use(path, thirdPartyMiddleware)

- C) router.use(path, thirdPartyMiddleware)

- D) app.route(path).use(thirdPartyMiddleware)

Answer: B) app.use(path, thirdPartyMiddleware)

90. Which third-party middleware is commonly used for serving static files in Express.js?

- A) express.static()

- B) morgan

- C) cors

- D) body-parser

Answer: A) express.static()

91. Which HTTP method is used to create a new resource in Express.js?

- A) GET

- B) POST

- C) PUT

- D) DELETE

Answer: B) POST

92. What does `app.get('/path', handler)` define in Express.js?

- A) A route that handles POST requests

- B) A route that handles PUT requests

- C) A route that handles GET requests

- D) A route that handles DELETE requests

Answer: C) A route that handles GET requests

93. How do you define a route that responds to both GET and POST requests?

- A) app.all('/path', handler)

- B) app.use('/path', handler)

- C) app.get('/path', handler).post('/path', handler)

- D) app.get('/path', handler).post('/path', handler)

Answer: A) app.all('/path', handler)

94. Which method is used to update an existing resource in Express.js?

- A) GET

- B) POST

- C) PUT

- D) DELETE

Answer: C) PUT

95. What is the purpose of `app.delete('/path', handler)` in Express.js?

- A) To update a resource at `/path`

- B) To delete a resource at `/path`

- C) To create a resource at `/path`

- D) To retrieve a resource at `/path`

Answer: B) To delete a resource at `/path`

96. Which method is used to handle multiple HTTP methods at the same route?

- A) app.route('/path').get(handler).post(handler)

- B) app.all('/path', handler)

- C) app.use('/path', handler)

- D) app.route('/path').all(handler)

Answer: A) app.route('/path').get(handler).post(handler)

97. How do you define a route for a specific HTTP method in Express.js?

- A) app.method('/path', handler)

- B) app.use('/path', handler)

- C) app.route('/path').method(handler)

- D) app.method('/path', handler)

Answer: A) app.method('/path', handler)

98. Which HTTP method is used to retrieve data from the server?

- A) POST

- B) PUT

- C) DELETE

- D) GET

Answer: D) GET

99. How do you define a route for handling PUT requests in Express.js?

- A) app.put('/path', handler)

- B) app.update('/path', handler)

- C) app.modify('/path', handler)

- D) app.patch('/path', handler)

Answer: A) app.put('/path', handler)

100. What does `app.all('/path', handler)` do?

- A) Handles all types of HTTP requests to `/path`

- B) Handles only GET requests to `/path`

- C) Handles only POST requests to `/path`

- D) Handles only PUT requests to `/path`

Answer: A) Handles all types of HTTP requests to `/path`

101. How would you define a route that matches `/user/:id` where `:id` is a route parameter?

- A) app.get('/user/:id', handler)

- B) app.get('/user/@id', handler)

- C) app.get('/user/[id]', handler)

- D) app.get('/user/:id?', handler)

Answer: A) app.get('/user/:id', handler)

102. How do you define a route that matches `/user` and `/user/:id`?

- A) app.get('/user', handler).get('/user/:id', handler)

- B) app.get('/user/:id?', handler)

- C) app.route('/user').get(handler).get('/user/:id', handler)

- D) app.all('/user/:id?', handler)

Answer: B) app.get('/user/:id?', handler)

103. What is the correct syntax for defining a route that matches `/user/123`?

- A) app.get('/user/:id', handler)

- B) app.get('/user/[123]', handler)

- C) app.get('/user/@123', handler)

- D) app.get('/user/123', handler)

Answer: A) app.get('/user/:id', handler)

104. How do you define a route that will match `/user` or `/user/123`?

- A) app.get('/user/:id?', handler)

- B) app.get('/user(/:id)?', handler)

- C) app.get('/user/:id([0-9]+)?', handler)

- D) app.get('/user/[id]', handler)

Answer: A) app.get('/user/:id?', handler)

105. How would you define a route that captures an optional query string parameter?

- A) app.get('/search/:query?', handler)

- B) app.get('/search/?query', handler)

- C) app.get('/search/:query', handler)

- D) app.get('/search/:query', handler)

Answer: A) app.get('/search/:query?', handler)

106. Which route path syntax captures a numeric route parameter?

- A) app.get('/item/:id([0-9]+)', handler)

- B) app.get('/item/:id', handler)

- C) app.get('/item/:id?', handler)

- D) app.get('/item/:id([a-z]+)', handler)

Answer: A) app.get('/item/:id([0-9]+)', handler)

107. How do you capture a route parameter that is a string with letters and numbers?

- A) app.get('/item/:param([a-zA-Z0-9]+)', handler)

- B) app.get('/item/:param', handler)

- C) app.get('/item/:param', handler)

- D) app.get('/item/:param?', handler)

Answer: A) app.get('/item/:param([a-zA-Z0-9]+)', handler)

108. What does `app.get('/user/:id(\\d+)', handler)` do?

- A) Matches `/user/:id` where `id` is a number

- B) Matches `/user/:id` where `id` is a string

- C) Matches `/user/` with optional `id`

- D) Matches `/user/` with `id` as any character

Answer: A) Matches `/user/:id` where `id` is a number

109. How do you define a route with a query string parameter in Express.js?

- A) app.get('/search', handler)

- B) app.get('/search/:query', handler)

- C) app.get('/search', handler)

- D) app.get('/search?query=value', handler)

Answer: C) app.get('/search', handler)

110. What does `app.get('/user/:id(\\w+)', handler)` do?

- A) Matches `/user/:id` where `id` is a word character

- B) Matches `/user/:id` where `id` is a digit

- C) Matches `/user` with optional `id`

- D) Matches `/user/:id` where `id` is a string

Answer: A) Matches `/user/:id` where `id` is a word character

111. How do you access route parameters in Express.js?

- A) req.params

- B) req.query

- C) req.body

- D) req.headers

Answer: A) req.params

112. How do you define a route parameter in Express.js?

- A) Using a colon (:) before the parameter name

- B) Using a dollar sign ($) before the parameter name

- C) Using curly braces ({}) around the parameter name

- D) Using square brackets ([]) around the parameter name

Answer: A) Using a colon (:) before the parameter name

113. How do you access a route parameter named `userId` in Express.js?

- A) req.params.userId

- B) req.query.userId

- C) req.body.userId

- D) req.headers.userId

Answer: A) req.params.userId

114. How do you define a route with multiple parameters?

- A) app.get('/user/:id/:name', handler)

- B) app.get('/user/:id-name', handler)

- C) app.get('/user/:id/name', handler)

- D) app.get('/user/:id/name?', handler)

Answer: A) app.get('/user/:id/:name', handler)

115. What is the purpose of `req.params` in Express.js?

- A) To access route parameters in the URL

- B) To access query parameters in the URL

- C) To access body data in the request

- D) To access headers in the request

Answer: A) To access route parameters in the URL

116. How do you access a route parameter named `postId` in an Express route handler?

- A) req.params.postId

- B) req.query.postId

- C) req.body.postId

- D) req.headers.postId

Answer: A) req.params.postId

117. How do you specify a route parameter that should be an integer?

- A) app.get('/post/:id(\\d+)', handler)

- B) app.get('/post/:id', handler)

- C) app.get('/post/:id([a-zA-Z]+)', handler)

- D) app.get('/post/:id', handler)

Answer: A) app.get('/post/:id(\\d+)', handler)

118. What does `req.params` contain in a route handler?

- A) An object of route parameters

- B) An object of query parameters

- C) An object of body data

- D) An object of headers

Answer: A) An object of route parameters

119. How do you define a route with an optional parameter?

- A) app.get('/user/:id?', handler)

- B) app.get('/user/:id', handler)

- C) app.get('/user/:id+', handler)

- D) app.get('/user/:id()', handler)

Answer: A) app.get('/user/:id?', handler)

120. How would you access multiple parameters named `id` and `name` in an Express route handler?

- A) req.params.id and req.params.name

- B) req.query.id and req.query.name

- C) req.body.id and req.body.name

- D) req.headers.id and req.headers.name

Answer: A) req.params.id and req.params.name

121. What is a route handler in Express.js?

- A) A function that processes incoming requests to a route

- B) A function that defines a route

- C) A function that serves static files

- D) A function that initializes middleware

Answer: A) A function that processes incoming requests to a route

122. How do you define a basic route handler in Express.js?

- A) app.get('/path', (req, res) => { / handler code / })

- B) app.use('/path', (req, res) => { / handler code / })

- C) app.route('/path').handler((req, res) => { / handler code / })

- D) app.route('/path').get((req, res) => { / handler code / })

Answer: A) app.get('/path', (req, res) => { / handler code / })

123. How do you send a JSON response from a route handler?

- A) res.json({ key: 'value' })

- B) res.send({ key: 'value' })

- C) res.write({ key: 'value' })

- D) res.end({ key: 'value' })

Answer: A) res.json({ key: 'value' })

124. How do you send a text response from a route handler?

- A) res.send('Text response')

- B) res.json('Text response')

- C) res.write('Text response')

- D) res.end('Text response')

Answer: A) res.send('Text response')

125. How do you handle errors within a route handler?

- A) By passing an error to `next()`

- B) By calling `res.error()`

- C) By logging the error and sending a response

- D) By using `app.error()`

Answer: A) By passing an error to `next()`

126. How can you return a status code from a route handler?

- A) res.status(404).send('Not Found')

- B) res.sendStatus(404)

- C) res.send('Not Found', 404)

- D) res.statusCode(404).send('Not Found')

Answer: A) res.status(404).send('Not Found')

127. How do you chain multiple response methods in Express.js?

- A) res.status(200).json({})

- B) res.send().json({})

- C) res.json().status(200)

- D) res.send().status(200)

Answer: A) res.status(200).json({})

128. How can you send a file as a response from a route handler?

- A) res.sendFile('path/to/file')

- B) res.file('path/to/file')

- C) res.download('path/to/file')

- D) res.attachment('path/to/file')

Answer: A) res.sendFile('path/to/file')

129. How do you set a custom header in the response?

- A) res.set('Custom-Header', 'value')

- B) res.header('Custom-Header', 'value')

- C) res.addHeader('Custom-Header', 'value')

- D) res.customHeader('Custom-Header', 'value')

Answer: A) res.set('Custom-Header', 'value')

130. What is the purpose of `res.redirect('/path')`?

- A) To redirect the client to a different URL

- B) To update the URL in the browser

- C) To handle a new route

- D) To log a response message

Answer: A) To redirect the client to a different URL

131. What happens if you do not call `next()` in middleware?

- A) The request will hang and no response will be sent

- B) The request will automatically proceed to the next middleware

- C) The response will be sent with a default message

- D) The middleware will be skipped

Answer: A) The request will hang and no response will be sent

132. How do you skip the remaining middleware in the stack?

- A) By calling `next()` with an error

- B) By calling `res.end()`

- C) By using `return` in the middleware function

- D) By using `app.stop()`

Answer: C) By using `return` in the middleware function

133. What is the purpose of the `next()` function in middleware?

- A) To pass control to the next middleware or route handler

- B) To terminate the request

- C) To start a new request

- D) To send a response

Answer: A) To pass control to the next middleware or route handler

134. How do you handle asynchronous operations in middleware?

- A) By using `async` and `await`

- B) By using `next()` with a delay

- C) By using `res.end()`

- D) By using `setTimeout()`

Answer: A) By using `async` and `await`

135. How do you create a middleware function that runs only for specific routes?

- A) By passing the route path as the first argument to `app.use()`

- B) By specifying the route path as an argument to `router.use()`

- C) By adding the route path directly inside the middleware function

- D) By defining multiple middleware functions for each route

Answer: B) By specifying the route path as an argument to `router.use()`

136. What is the order of execution for middleware in Express.js?

- A) Middleware is executed in the order it is defined

- B) Middleware is executed in reverse order of definition

- C) Middleware is executed randomly

- D) Middleware is executed based on HTTP method

Answer: A) Middleware is executed in the order it is defined

137. How do you pass data between middleware functions?

- A) By attaching data to the `req` object

- B) By using global variables

- C) By returning data from middleware functions

- D) By using `res.locals`

Answer: A) By attaching data to the `req` object

138. How can you define middleware that only runs for a specific HTTP method?

- A) By using `app.method(path, middleware)`

- B) By using `app.use(path, method, middleware)`

- C) By specifying the method in the route definition

- D) By adding method-specific middleware in the route handler

Answer: C) By specifying the method in the route definition

139. What is the purpose of error-handling middleware in Express.js?

- A) To catch and handle errors that occur during request processing

- B) To log requests and responses

- C) To serve static files

- D) To parse incoming request bodies

Answer: A) To catch and handle errors that occur during request processing

140. How do you define error-handling middleware in Express.js?

- A) By defining a function with four arguments: `err`, `req`, `res`, `next`

- B) By defining a function with three arguments: `req`, `res`, `next`

- C) By using `app.use(errorHandler)`

- D) By specifying an error code in the middleware function

Answer: A) By defining a function with four arguments: `err`, `req`, `res`, `next`

141. How do you apply application-level middleware in Express.js?

- A) By using `app.use(middleware)`

- B) By using `app.middleware(middleware)`

- C) By using `router.use(middleware)`

- D) By using `app.middleware.use(middleware)`

Answer: A) By using `app.use(middleware)`

142. How can you use application-level middleware to parse JSON request bodies?

- A) app.use(express.json())

- B) app.use(bodyParser.json())

- C) app.use(express.urlencoded())

- D) app.use(express.raw())

Answer: A) app.use(express.json())

143. How do you apply middleware to all routes in an Express.js application?

- A) By using `app.use(middleware)`

- B) By using `router.use(middleware)`

- C) By using `app.route(middleware)`

- D) By using `app.all(middleware)`

Answer: A) By using `app.use(middleware)`

144. How can you limit the application of middleware to specific routes?

- A) By specifying the route path as the first argument to `app.use()`

- B) By adding middleware directly in the route definition

- C) By defining middleware in route handlers

- D) By using route-specific middleware functions

Answer: A) By specifying the route path as the first argument to `app.use()`

145. How do you apply middleware that executes before route handlers?

- A) By defining middleware before route definitions

- B) By defining middleware after route definitions

- C) By using `app.all()` for middleware

- D) By using middleware in route handlers

Answer: A) By defining middleware before route definitions

146. What is the purpose of `app.use(express.urlencoded({ extended: true }))`?

- A) To parse URL-encoded request bodies

- B) To parse JSON request bodies

- C) To parse raw request bodies

- D) To parse query strings

Answer: A) To parse URL-encoded request bodies

147. How can you apply middleware to specific HTTP methods only?

- A) By using `app.method('/path', middleware)`

- B) By using `app.use('/path', middleware)`

- C) By using `app.all('/path', middleware)`

- D) By using `router.use('/path', middleware)`

Answer: A) By using `app.method('/path', middleware)`

148. How can you ensure that middleware runs only once per request?

- A) By placing it before other middleware and route handlers

- B) By using `app.once('/path', middleware)`

- C) By using `app.middlewareOnce('/path', middleware)`

- D) By using `app.useOnce(middleware)`

Answer: A) By placing it before other middleware and route handlers

149. How can you create middleware that logs the request method and URL?

- A) app.use((req, res, next) => { console.log(`${req.method} ${req.url}`); next(); })

- B) app.use((req, res, next) => { console.log(req.method); console.log(req.url); next(); })

- C) app.use((req, res, next) => { res.log(req.method + req.url); next(); })

- D) app.use((req, res, next) => { res.write(`${req.method} ${req.url}`); next(); })

Answer: A) app.use((req, res, next) => { console.log(`${req.method} ${req.url}`); next(); })

150. How do you use middleware to handle static files in Express.js?

- A) app.use(express.static('public'))

- B) app.use(express.files('public'))

- C) app.use(express.serve('public'))

- D) app.use(express.staticFiles('public'))

Answer: A) app.use(express.static('public'))

151. What is router-level middleware in Express.js?

- A) Middleware that is applied to a specific router instance

- B) Middleware that is applied globally to the app

- C) Middleware that is used for error handling

- D) Middleware that handles static files

Answer: A) Middleware that is applied to a specific router instance

152. How do you define router-level middleware in Express.js?

- A) By using `router.use(middleware)`

- B) By using `app.use(middleware)`

- C) By using `router.middleware(middleware)`

- D) By using `router.all(middleware)`

Answer: A) By using `router.use(middleware)`

153. How do you apply router-level middleware to a specific route?

- A) By using `router.use('/path', middleware)`

- B) By using `router.route('/path').use(middleware)`

- C) By adding middleware directly in route definitions

- D) By specifying middleware in route handlers

Answer: A) By using `router.use('/path', middleware)`

154. How do you define a router instance in Express.js?

- A) const router = express.Router()

- B) const router = new express.Router()

- C) const router = express.createRouter()

- D) const router = express.router()

Answer: A) const router = express.Router()

155. How can you apply router-level middleware to a subset of routes?

- A) By defining middleware on a specific router and mounting it to a route

- B) By using `router.use` directly within route definitions

- C) By using `app.use(router)` with middleware

- D) By specifying middleware in route handlers

Answer: A) By defining middleware on a specific router and mounting it to a route

156. How do you define a route within a router instance?

- A) router.get('/path', handler)

- B) router.route('/path').get(handler)

- C) router.use('/path', handler)

- D) router.all('/path', handler)

Answer: A) router.get('/path', handler)

157. How do you mount a router instance to a specific path in the application?

- A) app.use('/path', router)

- B) app.mount('/path', router)

- C) app.use(router).at('/path')

- D) app.route('/path').use(router)

Answer: A) app.use('/path', router)

158. What is the benefit of using router-level middleware?

- A) To organize middleware and routes into modular, reusable components

- B) To apply middleware globally to the application

- C) To handle static files

- D) To handle error responses

Answer: A) To organize middleware and routes into modular, reusable components

159. How do you apply middleware to specific methods within a router?

- A) By using `router.use(method, middleware)`

- B) By adding middleware in route definitions for specific methods

- C) By using `router.all(method, middleware)`

- D) By specifying middleware in `router.route()`

Answer: B) By adding middleware in route definitions for specific methods

160. How can you combine multiple router instances in Express.js?

- A) By mounting each router instance to a different path using `app.use()`

- B) By using `app.combine(router1, router2)`

- C) By using `app.route().use(router1, router2)`

- D) By defining a master router that includes other routers

Answer: A) By mounting each router instance to a different path using `app.use()`

161. What is the primary purpose of error-handling middleware?

- A) To catch and handle errors that occur during request processing

- B) To log requests and responses

- C) To parse request bodies

- D) To handle static files

Answer: A) To catch and handle errors that occur during request processing

162. How do you define an error-handling middleware function?

- A) By using a function with four arguments: `err`, `req`, `res`, `next`

- B) By using a function with three arguments: `req`, `res`, `next`

- C) By using a function with two arguments: `req`, `res`

- D) By using a function with one argument: `err`

Answer: A) By using a function with four arguments: `err`, `req`, `res`, `next`

163. How do you ensure that error-handling middleware is executed for all errors?

- A) By defining it after all other middleware and route handlers

- B) By using `app.use(errorHandler)` at the beginning of the middleware stack

- C) By calling `errorHandler()` in every route

- D) By specifying error codes in middleware functions

Answer: A) By defining it after all other middleware and route handlers

164. How do you handle a specific type of error in error-handling middleware?

- A) By checking the `err` object for specific properties or types

- B) By using `if` statements in the error-handling function

- C) By using different error-handling middleware functions for different errors

- D) By logging the error and sending a generic response

Answer: A) By checking the `err` object for specific properties or types

165. What is the purpose of `next(err)` in error-handling middleware?

- A) To pass the error to the next error-handling middleware function

- B) To terminate the request-response cycle

- C) To skip to the next middleware function

- D) To handle errors in the request body

Answer: A) To pass the error to the next error-handling middleware function

Certainly! Here are the next 35 MCQs:

166. What should error-handling middleware do if it cannot handle a specific error?

- A) Call `next(err)` to pass the error to the next error-handling middleware

- B) Send a generic error response

- C) Log the error and terminate the request

- D) Return a default status code

Answer: A) Call `next(err)` to pass the error to the next error-handling middleware

167. How do you create a custom error class in JavaScript for use with Express error-handling?

- A) By extending the built-in `Error` class

- B) By using `Object.create(Error)`

- C) By defining a function with `Error` as a prototype

- D) By using a factory function to create error instances

Answer: A) By extending the built-in `Error` class

168. How can you send a custom error message from error-handling middleware?

- A) By using `res.status(statusCode).send('Custom error message')`

- B) By using `res.json({ error: 'Custom error message' })`

- C) By using `res.write('Custom error message')`

- D) By using `res.end('Custom error message')`

Answer: A) By using `res.status(statusCode).send('Custom error message')`

169. What method should be used in error-handling middleware to respond with an HTML error page?

- A) `res.sendFile('path/to/error.html')`

- B) `res.render('error')`

- C) `res.html('path/to/error.html')`

- D) `res.send('error page')`

Answer: B) `res.render('error')`

170. What is the sequence of events for handling a request in Express.js?

- A) Middleware execution → Route matching → Route handler execution → Response

- B) Route matching → Middleware execution → Route handler execution → Response

- C) Route handler execution → Middleware execution → Route matching → Response

- D) Middleware execution → Route handler execution → Route matching → Response

Answer: A) Middleware execution → Route matching → Route handler execution → Response

171. What happens if a middleware function does not call `next()` in Express.js?

- A) The request-response cycle is halted and no response is sent

- B) The next middleware or route handler is automatically executed

- C) The server sends a default response

- D) The request is sent back to the client

Answer: A) The request-response cycle is halted and no response is sent

172. How can middleware modify the request object in Express.js?

- A) By adding or altering properties on `req`

- B) By sending a modified request to the next middleware

- C) By returning a new request object from the middleware

- D) By setting request headers directly

Answer: A) By adding or altering properties on `req`

173. What is the purpose of the `req` object in Express.js?

- A) To represent the HTTP request and provide access to request data

- B) To represent the HTTP response and provide access to response methods

- C) To handle errors during request processing

- D) To manage routing and middleware

Answer: A) To represent the HTTP request and provide access to request data

174. How does Express.js determine which route to use for a request?

- A) By matching the request URL and HTTP method with defined routes

- B) By examining the request headers

- C) By analyzing the request body

- D) By using the request's IP address

Answer: A) By matching the request URL and HTTP method with defined routes

175. How are dynamic route parameters processed in Express.js?

- A) They are extracted from the request URL and made available via `req.params`

- B) They are included in the query string and accessed via `req.query`

- C) They are added to the request headers and accessed via `req.headers`

- D) They are parsed from the request body and accessed via `req.body`

Answer: A) They are extracted from the request URL and made available via `req.params`

176. What is the difference between blocking and non-blocking code in Node.js?

- A) Blocking code waits for an operation to complete before moving on, while non-blocking code does not

- B) Blocking code executes asynchronously, while non-blocking code executes synchronously

- C) Blocking code executes in a different thread, while non-blocking code executes in the main thread

- D) Blocking code handles network requests, while non-blocking code handles file I/O

Answer: A) Blocking code waits for an operation to complete before moving on, while non-blocking code does not

177. Which of the following is an example of blocking code in Node.js?

- A) Synchronous file system operations

- B) Asynchronous database queries

- C) Event-driven callbacks

- D) Promises

Answer: A) Synchronous file system operations

178. How does non-blocking code improve performance in Node.js?

- A) By allowing other operations to continue while waiting for I/O operations to complete

- B) By using multiple threads for execution

- C) By blocking the main thread until operations complete

- D) By prioritizing CPU-bound tasks

Answer: A) By allowing other operations to continue while waiting for I/O operations to complete

179. How do you write non-blocking code in Node.js?

- A) By using callbacks, promises, or async/await

- B) By using synchronous functions and methods

- C) By executing tasks in a separate process

- D) By limiting the number of simultaneous requests

Answer: A) By using callbacks, promises, or async/await

180. What is an advantage of using non-blocking I/O in Node.js?

- A) It increases the application's ability to handle multiple concurrent operations

- B) It reduces the memory footprint of the application

- C) It simplifies the application code

- D) It ensures operations are executed sequentially

Answer: A) It increases the application's ability to handle multiple concurrent operations

181. What is the purpose of `body-parser` middleware in Express.js?

- A) To parse incoming request bodies and make the data available in `req.body`

- B) To parse query strings and make the data available in `req.query`

- C) To parse cookies and make them available in `req.cookies`

- D) To parse request headers and make them available in `req.headers`

Answer: A) To parse incoming request bodies and make the data available in `req.body`

182. How do you use `body-parser` to parse JSON request bodies?

- A) `app.use(bodyParser.json())`

- B) `app.use(express.json())`

- C) `app.use(bodyParser.urlencoded({ extended: true }))`

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

Answer: A) `app.use(bodyParser.json())`

183. How do you use `body-parser` to parse URL-encoded request bodies?

- A) `app.use(bodyParser.urlencoded({ extended: true }))`

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

- C) `app.use(bodyParser.json())`

- D) `app.use(express.json())`

Answer: A) `app.use(bodyParser.urlencoded({ extended: true }))`

184. What is the `extended` option in `body-parser.urlencoded()` used for?

- A) To determine whether to use the `querystring` library or the `qs` library for parsing

- B) To specify the maximum request body size

- C) To indicate whether to parse JSON data

- D) To set the encoding for the request body

Answer: A) To determine whether to use the `querystring` library or the `qs` library for parsing

185. Why might you use `body-parser` middleware in your Express application?

- A) To handle and parse incoming request bodies for easier access to form data

- B) To manage routing and route handling

- C) To serve static files from a directory

- D) To handle authentication and authorization

Answer: A) To handle and parse incoming request bodies for easier access to form data

186. How do you define a route in Express.js?

- A) By using `app.method('/path', handler)`

- B) By using `app.route('/path').method(handler)`

- C) By using `app.use('/path', handler)`

- D) By using `app.all('/path', handler)`

Answer: A) By using `app.method('/path', handler)`

187. How do you define a route parameter in Express.js?

- A) By using a colon `:` followed by the parameter name in the route path

- B) By using a dollar sign `$` followed by the parameter name

- C) By using curly braces `{}` around the parameter name

- D) By using square brackets `[]` around the parameter name

Answer: A) By using a colon `:` followed by the parameter name in the route path

188. How do you access route parameters in a route handler?

- A) Through `req.params`

- B) Through `req.query`

- C) Through `req.body`

- D) Through `req.headers`

Answer: A) Through `req.params`

189. What is the purpose of `router.route('/path')` in Express.js?

- A) To define a route with multiple HTTP methods

- B) To mount a router instance on a specific path

- C) To handle all routes in a router instance

- D) To define a middleware function for a specific path

Answer: A) To define a route with multiple HTTP methods

190. How can you define multiple route handlers for different HTTP methods on the same path?

- A) By chaining `get`, `post`, `put`, and `delete` methods on the route

- B) By using `app.route('/path').all(method1, method2)`

- C) By defining separate routes for each HTTP method

- D) By using `app.use('/path', method1, method2)`

Answer: A) By chaining `get`, `post`, `put`, and `delete` methods on the route

191. What does `app.all('/path', handler)` do in Express.js?

- A) It registers a handler for all HTTP methods on the specified path

- B) It registers a handler only for the `GET` method on the specified path

- C) It registers a handler only for the `POST` method on the specified path

- D) It registers a handler for a specific HTTP method on all paths

Answer: A) It registers a handler for all HTTP methods on the specified path

192. How do you handle optional route parameters in Express.js?

- A) By using a question mark `?` after the parameter name in the route path

- B) By using square brackets `[]` around the parameter name

- C) By defining two separate routes, one with and one without the parameter

- D) By using a default value in the route handler function

Answer: A) By using a question mark `?` after the parameter name in the route path

193. How do you handle dynamic route parameters in a route handler?

- A) By using `req.params.paramName`

- B) By using `req.query.paramName`

- C) By using `req.body.paramName`

- D) By using `req.headers.paramName`

Answer: A) By using `req.params.paramName`

194. What is the purpose of `next()` in middleware functions?

- A) To pass control to the next middleware function or route handler

- B) To terminate the request-response cycle

- C) To restart the middleware stack

- D) To handle errors in the middleware function

Answer: A) To pass control to the next middleware function or route handler

195. What happens if `next()` is not called in a middleware function?

- A) The request-response cycle is halted and no response is sent

- B) The request is sent to the next route handler automatically

- C) The middleware stack continues to the next middleware

- D) An error is thrown and the request is terminated

Answer: A) The request-response cycle is halted and no response is sent

196. How do you ensure that a middleware function is executed only once for a request?

- A) By placing it at the beginning of the middleware stack

- B) By using the `once` method on the middleware

- C) By specifying `maxRequests: 1` in the middleware configuration

- D) By calling `next()` multiple times in the middleware function

Answer: A) By placing it at the beginning of the middleware stack

197. How do you chain multiple middleware functions together?

- A) By defining them one after another in `app.use()` or `router.use()`

- B) By using a single function that calls all middleware functions

- C) By using `app.chain()` method

- D) By defining middleware functions in an array and passing it to `app.use()`

Answer: A) By defining them one after another in `app.use()` or `router.use()`

198. What is the typical order of execution for middleware in Express.js?

- A) Middleware is executed in the order it is defined

- B) Middleware is executed in reverse order of definition

- C) Middleware is executed randomly

- D) Middleware is executed based on priority settings

Answer: A) Middleware is executed in the order it is defined

199. What does `app.listen(port, callback)` do in Express.js?

- A) It starts the server and listens for incoming connections on the specified port

- B) It defines the route handlers for the application

- C) It parses incoming request bodies

- D) It mounts middleware to the application

Answer: A) It starts the server and listens for incoming connections on the specified port

200. How do you set a default response for all unmatched routes in Express.js?

- A) By defining a catch-all route handler at the end of the middleware stack

- B) By using `app.use('', handler)`

- C) By using `app.all('', handler)`

- D) By specifying a default handler in `app.listen()`

Answer: A) By defining a catch-all route handler at the end of the middleware stack