Download PDF

Back End Technologies - Subjective Interview Questions

Edunet Foundation

Interview Questions – Backend Technologies

Question 1: What is Express.js, and what is it used for?

Answer: Express.js is a fast and minimal web application framework for Node.js. It is used to build web applications and RESTful APIs, offering robust features for handling HTTP requests, middleware, and routing.

Question 2: What are middleware functions in Express.js?

Answer: Middleware functions in Express.js are functions that have access to the request object (req), response object (res), and the next middleware function in the application's request-response cycle. Middleware can execute code, modify req and res objects, end the request-response cycle, or call the next middleware in the stack.

Question 3: How do you set up a basic server with Express.js?

Answer: To set up a basic server with Express.js:

1. Import Express using const express = require('express');.
2. Create an instance with const app = express();.
3. Define a route, e.g., app.get('/', (req, res) => res.send('Hello World!'));.
4. Start the server using app.listen(PORT, () => console.log('Server running on port ' + PORT));.

Question 4: What is the purpose of app.use() in Express.js?

Answer: app.use() is used to apply middleware to an application. This middleware can handle specific routes or the entire application and is used for functions like logging, body parsing, authentication, etc.

Question 5: What is routing in Express.js, and how is it implemented?

Answer: Routing in Express.js refers to defining the various URL endpoints (routes) and handling client requests for those endpoints. It is implemented using methods such as app.get(), app.post(), app.put(), app.delete(), etc., which specify the type of request and the callback function to handle it. Example:

```
app.get('/about', (req, res) => {
  res.send('About page');
});
```

Question 6: How do you handle errors in Express.js?

Answer: Error handling in Express.js is done using middleware functions with four parameters: (err, req, res, next). This special middleware handles errors and can be defined as:

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

Answer: req.params is an object that contains route parameters, which are dynamic parts of a route. For example, in a route defined as app.get('/user/:id', ...), req.params.id would contain the value passed in the URL at /user/:id.

Question 8: What is the difference between app.get() and app.post() in Express.js?

Answer: app.get() is used to handle GET requests, typically used for retrieving data. app.post() handles POST requests, used for sending data to the server, such as form submissions.

Question 9: What is req.query in Express.js?

Answer: req.query is an object that contains key-value pairs of query parameters from the URL. For example, in the URL /search?term=node, req.query.term would return 'node'.

Question 10: What is req.body in Express.js, and how do you use it?

Answer: req.body is an object containing data submitted in the body of a request. To parse the body, middleware such as express.json() or express.urlencoded() needs to be used:

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

Question 11: How can you serve static files in Express.js?

Answer: Static files such as images, CSS, and JavaScript can be served using the built-in express.static middleware:

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

Question 12: What is next() in Express.js, and when is it used?

Answer: next() is a function that, when called, passes control to the next middleware function in the stack. It is used to move to the next step in the request-response cycle.

Question 13: What is the role of the app.listen() method?

Answer: The app.listen() method binds and listens for connections on a specified host and port. It starts the server and makes it ready to handle incoming requests.

Question 14: How do you create a route that handles all HTTP methods?

Answer: You can use app.all() to handle all HTTP methods for a particular route:

```
app.all('/example', (req, res) => {

  res.send('This handles all HTTP methods');

});
```

Question 15: How do you implement URL parameters in routes?

Answer: URL parameters are implemented by defining a colon followed by the parameter name in the route:

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

  res.send(`User ID: ${req.params.id}`);

});
```

Question 16: What is res.send() used for?

Answer: res.send() is used to send a response to the client. It can send strings, objects, arrays, or buffers, and automatically sets the Content-Type header based on the type of content.

Question 17: What is res.json() in Express.js?

Answer: res.json() is a method used to send a JSON response. It converts the response object to a JSON string and sets the Content-Type to application/json.

```
app.use((req, res, next) => {
  console.log('Custom middleware executed');
  next();
});
```

Question 19: How do you enable CORS in an Express.js app?

Answer: CORS (Cross-Origin Resource Sharing) can be enabled using the cors package:

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

Question 20: What is the purpose of the express.Router()?

Answer: express.Router() is used to create modular, mountable route handlers. It allows you to organize routes into separate files or sections:

```
const router = express.Router();
router.get('/path', (req, res) => {
  res.send('Router path');
});
app.use('/base', router);
```

Question 21: What is Node.js, and what is it used for?

Answer: Node.js is an open-source, cross-platform JavaScript runtime environment that runs on the server side. It is built on the V8 JavaScript engine and is used for building scalable network applications and server-side scripting.

Question 22: What are the main features of Node.js?

Answer: Key features of Node.js include:

Asynchronous and Event-Driven: Node.js operates on a non-blocking I/O model.

Fast Execution: Uses the V8 engine for fast execution of JavaScript.

Single-Threaded: Uses a single-threaded model with event looping.

Cross-Platform: Compatible with Windows, Linux, and macOS.

Question 23: What is npm, and what is it used for?

Answer: npm (Node Package Manager) is a package manager for Node.js. It is used to install, update, and manage JavaScript packages and libraries in Node.js projects.

Question 24: How do you install a package using npm?

Answer: To install a package, use the following command:

```
npm install package-name
```

Question 25: What is the package.json file, and why is it important?

Question 26: What is the purpose of require() in Node.js?

Answer: The require() function is used to import modules in Node.js. For example:

```
const fs = require('fs');
```

Question 27: What are core modules in Node.js?

Answer: Core modules are built-in modules that come with Node.js, such as http, fs, path, and os. These modules provide essential functionalities without needing external libraries.

Question 28: How do you create an HTTP server in Node.js?

Answer:

```
const http = require('http');

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/plain');

  res.end('Hello, World!\n');

});

server.listen(3000, () => {

  console.log('Server running at http://localhost:3000/');

});
```

Question 29: What is event-driven programming in Node.js?

Answer: Event-driven programming in Node.js is a programming paradigm where the flow of the program is determined by events such as user actions, messages, or sensor outputs. Node.js uses the EventEmitter class to handle such events.

Question 30: What is the purpose of process in Node.js?

Answer: The process object provides information about the current Node.js process, and it can be used to interact with the environment, handle arguments, and exit the application.

Question 31: What is the Event Loop in Node.js?

Answer: The Event Loop is a mechanism in Node.js that handles and processes events, callbacks, and asynchronous operations. It allows Node.js to perform non-blocking I/O operations despite being single-threaded.

Question 32: How do you handle errors in Node.js?

Answer: Errors in Node.js can be handled using try-catch blocks, or by attaching error event listeners to streams and event emitters:

```
} catch (error) {
  console.error(error);
}
```

Question 33: What is a callback function in Node.js?

Answer: A callback function is a function passed as an argument to another function, to be executed after the completion of that function. Node.js makes extensive use of callbacks to handle asynchronous operations.

Question 34: What is Buffer in Node.js?

Answer: Buffer is a global object in Node.js used to handle binary data directly, especially when dealing with streams like reading or writing files and handling network packets.

Question 35: What is the difference between readFileSync and readFile in Node.js?

Answer: readFileSync reads a file synchronously, blocking the event loop until the file is read, while readFile reads a file asynchronously and does not block the event loop.

Question 36: What is middleware in Node.js?

Answer: Middleware refers to functions that are executed in the request-response cycle in a Node.js/Express.js application to process requests, responses, or invoke the next middleware.

Question 37: What are Streams in Node.js?

Answer: Streams are objects that enable reading or writing data piece by piece. They are useful for handling large amounts of data without consuming a lot of memory.

Question 38: What is setImmediate() in Node.js?

Answer: setImmediate() is a method that schedules the immediate execution of a callback after the current event loop completes its current cycle.

Question 39: What is the purpose of path module in Node.js?

Answer: The path module provides utilities for working with file and directory paths. Example:

```
const path = require('path');

console.log(path.join('/users', 'john', 'docs'));
```

Question 40: What is the fs module in Node.js, and what is it used for?

Question 41: What is MongoDB?

Answer: MongoDB is a document-oriented NoSQL database used for high volume data storage. It is an open-source database written in C++.

Question 42: What is NoSQL?

Answer: NoSQL stands for “Not Only SQL.” It refers to a variety of database systems that are designed to handle large volumes of data and are not based on traditional relational database principles.

Question 43: What are the types of NoSQL databases?

Answer: Document-based (e.g., MongoDB), Key-Value (e.g., Redis), Column-based (e.g., Cassandra), and Graph-based (e.g., Neo4j).

Question 44: What are the advantages of MongoDB?

Answer: High performance, high availability, easy scalability, and flexibility in handling large volumes of data.

Question 45: What is a Document in MongoDB?

Answer: A document is a set of key-value pairs. Documents have a dynamic schema, meaning that documents in the same collection do not need to have the same set of fields.

Question 46: What is a Collection in MongoDB?

Answer: A collection is a group of MongoDB documents. It is the equivalent of an RDBMS table.

Question 47: What are Dynamic Schemas?

Answer: Dynamic schemas mean that documents in a collection do not need to have the same structure. Fields can vary from document to document.

Question 48: What is Mongo Shell?

Answer: MongoDB Shell is a JavaScript interface to MongoDB, used to perform administrative tasks and interact with the database.

Question 49: What are some features of MongoDB?

Answer: Indexing, Aggregation, Special collection types, File storage, and Sharding.

Question 50: Where can MongoDB be used?

Answer: Content management, mobile apps, data management, and big data applications.

Question 51: Which languages does MongoDB support?

Answer: MongoDB supports multiple languages including C++, Java, Python, PHP, and Node.js.

Answer: String, Integer, Boolean, Double, Arrays, Objects, Null, and Date.

Question 53: What is ObjectId in MongoDB?

Answer: ObjectId is a unique identifier for documents in a MongoDB collection. It is a 12-byte identifier that ensures uniqueness.

Question 54: What is Sharding in MongoDB?

Answer: Sharding is the process of distributing data across multiple machines to support large datasets and high throughput operations.

Question 55: What is Replication in MongoDB?

Answer: Replication is the process of synchronizing data across multiple servers to ensure high availability and redundancy.

Question 56: What is an Aggregation Framework?

Answer: The aggregation framework is used for processing data and returning computed results. It involves operations like grouping, sorting, and transforming data.

Question 57: What is a Replica Set?

Answer: A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and high availability.

Question 58: What is a Primary and Secondary Replica?

Answer: In a replica set, the primary replica receives all write operations, while secondary replicas replicate the primary’s data and can serve read operations.

Question 59: What is a MongoDB Index?

Answer: Indexes support the efficient execution of queries. Without indexes, MongoDB must perform a collection scan to find documents.

Question 60: What is a Covered Query?

Answer: A covered query is a query in which all the fields in the query are part of an index, and the index contains all the fields returned in the query.

Question 61: What is a Geospatial Index?

Answer: Geospatial indexes support queries for geospatial shapes and data, such as finding points within a certain distance from a location.

Question 62: What is the Aggregation Pipeline?

Answer: The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines.

for processing.

Question 64: What is the $match stage in Aggregation?

Answer: The $match stage filters documents to pass only those that match the specified condition(s) to the next pipeline stage.

Question 65: What is the $group stage in Aggregation?

Answer: The $group stage groups input documents by a specified identifier expression and applies the accumulator expressions to each group.

Question 66: What is the $project stage in Aggregation?

Answer: The $project stage reshapes each document in the stream, such as by adding new fields or removing existing fields.

Question 67: What is the $sort stage in Aggregation?

Answer: The $sort stage sorts all input documents and returns them to the pipeline in sorted order.

Question 68: What is the $limit stage in Aggregation?

Answer: The $limit stage limits the number of documents passed to the next stage in the pipeline.

Question 69: What is the $skip stage in Aggregation?

The $skip stage skips over a specified number of documents and passes the remaining documents to the next stage in the pipeline.

Question 70: What is the $unwind stage in Aggregation?

Answer: The $unwind stage deconstructs an array field from the input documents to output a document for each element.