**1.What is Express JS?**

Express.js, or simply Express, is a free, open-source, lightweight, and fast backend web application framework for Node.js. It is released as open-source software under the MIT License. It is designed for building single-page, multi-page, and hybrid web applications and APIs.

**2.Provide some features of Express JS.**

It can be used to design single-page, multi-page, and hybrid web applications and APIs.

It allows to set up middleware to respond to HTTP/RESTful Requests.

It defines a routing table to perform different HTTP operations (method and URL).

It allows to dynamically rendering HTML Pages based on passing arguments to templates.

It provides high performance because of its ultra-fast I/O. It prepares a thin layer; therefore, the performance is adequate.

Its MVC-like structure makes it organize the web application into MVC architecture.

It provides good database support. It supports RDBMS as well as NoSQL databases.

It is asynchronous and single-threaded.

Its robust API makes routing easy.

**3. Write the differences between Express JS and Node JS.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Express.js** | **Node.js** |
| Definition | Express.js is a lightweight and fast backend web application framework for Node.js. | Node.js is an open-source and cross-platform that is used to execute JavaScript code outside of a browser. |
| Usage | Express.js is used to develop complete web applications such as single-page, multi-page, and hybrid web applications and APIs. It uses approaches and principles of Node.js. | Node.js is used to build server-side, input-output, event-driven apps. |
| Features | Express has more features than Node.js. | Node.js has fewer features as compared to Express.js. |
| Building Block | Express.js is built on Node.js. | Node.js is built on Google's V8 engine. |
| Written in | Express.js is written in JavaScript only. | Node.js is written in C, C++, and JavaScript language. |
| Framework/Platform | Express.js is a framework of Node.js based on its functionalities. | Node.js is a run-time platform or environment designed for server-side execution of JavaScript. |
| Controllers | Express.js is assigned with controllers. | Node.js is assigned with controllers. |
| Routing | Routing is provided in Express.js. | Routing is not provided in Node.js. |
| Middleware | Express.js uses middleware to arrange the functions systematically on the server-side. | Node.js doesn't use any such provision of middleware. |
| Coding | Express is easy to code and requires less coding time. | Node.js requires more coding time as compare to Express.js. |

**4.Write a simple expression of Express.JS.**

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Welcome to myAPI!');

});

var server = app.listen(8000, function () {

var host = server.address().address;

var port = server.address().port;

console.log('Example app listening at http://%s:%s', host, port);

});

**5.Write the code which is responsible for fetching post request body.**

var bodyParser = require('body-parser')

app.use( bodyParser.json() ); // to support JSON-encoded

app.use(bodyParser.urlencoded({ // to support URL-encoded

extended: true

}));

**6. What do you understand by Scaffolding in Express.js?**

Scaffolding is a technique used for creating the skeleton structure of an application. It facilitates users to easily create their public directories, routes, views, etc., or a web application skeleton. Generally, users manually create their public directory, add middleware, create separate route files, etc. Using a scaffolding tool, they can set up all these things to directly get started with building their application.

Example: Express application generator : Express

**7.Explain the default folder and file generated by Express Command.**

**Bin**: The bin folder contains one file called www is the main configuration file of the app.

**Public**: The public folder contains JavaScript, CSS, and images, etc.

**Routes**: This folder contains the routing files.

**Views**: The view folder contains the view files of the application.

**js**: The app.js file is the main file of the application.

**json**: The package.json file is the manifest file. It contains all metadata of the project, such as the **packages** used in the app (called dependencies) etc.

**8. Which are the arguments available to an Express JS route handler function?**

Following are the arguments that are available to an Express.js route handler-function:

**Req**: the request object

**Res**: the response object

**Next** (optional): It is a function employed to pass management to one of the above route handlers.

**9. How can you enable debugging in Express.js app?**

Use the following command on Windows:

set DEBUG=express:\*

node app.js

**10. How can you allow CORS in Express.js?**

We can allow CORS in Express.js, by adding the following code in server.js:

app.all('\*', function(req, res, next) {

res.set('Access-Control-Allow-Origin', '\*');

res.set('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT');

res.set('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type');

if ('OPTIONS' == req.method) return res.send(200);

next();

});

**11. Write the code to start serving static files in Express.js.**

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

app.use('/static', express.static(path.join(\_\_dirname, 'public')))

**12. What is Middleware in Express.js?**

Middleware is a function invoked by the Express routing layer before the final request handler.

Middleware functions are used to perform the following tasks:

* It is used to execute any code.
* It is also used to make changes to the request and the response objects.
* It is responsible for ending the request-response cycle.
* It can call the next middleware function in the stack.

**13. What are the different types of Middleware?**

Following are the main types of Middleware:

1. Application-level Middleware
2. Router-level Middleware
3. Error-handling Middleware
4. Built-in Middleware
5. Third-party Middleware

**Application-level middleware:**

The application-level middleware method is used to bind to the app object using app.use() method. It applies on all routes.

//This middleware will execute for each route.

app.use(function (req, res, next) {

console.log('Current Time:', Date.now())

next()

})

**Router-level Middleware:**   
The router-level Middleware is used to bind to a specific instance of express.Router().Built-in Middleware: The built-in Middleware was introduced with version 4.x. It ends the dependency on Connect.

There are the following built-in middleware functions in Express.js:

static: It is used to serve static assets such as HTML files, images, etc.

json: It is used to parse the incoming requests with JSON payloads. It is available with Express 4.16.0+

urlencoded: It is used to parse the incoming requests with URL-encoded payloads. It is available with Express 4.16.0+

**Third-party Middleware:**   
There are many third-party middleware available such as:

*Body-parser, Cookie-parser, Mongoose, Sequelize, Cors, Express-validator*

To handle HTTP POST requests in Express.js version 4 and above, we have to install a middleware module called body-parser. Body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body, The Middleware was a part of Express.js earlier, but now you have to install it separately. You can install it by using the following command:

npm install MODULE\_NAME

You can load it by using requires and used later:

See the Example:

var bodyParser = require('body-parser');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false }))

**14.Which template engines do Express support?**

Express.js supports any template engine that follows the (path, locals, callback) signature.

**15.How can we render a pain HTML?**

There is no need to "render" HTML with the res.render() function. If you have a specific file, you can use the res.sendFile() function, but you should use the express if you serve many assets from a directory.static() middleware function.

**16.What is the use of app.use() in Express.js?**

app.use() is used to add middleware functions to an Express application. It can be used to add global middleware functions or to add middleware functions to specific routes.

**17.What is the purpose of the next() function in Express.js?**

The next() function is used to pass control from one middleware function to the next function. It is used to execute the next middleware function in the chain

**18.What is the difference between res.send() and res.json() in Express.js?**

res.send() is used to send a response with any type of data (string, object, buffer, etc.). While res.json() is used to send a JSON response. res.json() also sets the Content-Type header to application or JSON.

**19.How does Express.js handle file uploads?**

Express.js provides support for file uploads through middleware functions and the request object. Developers can use middleware functions like multer or busboy to handle file uploads. It can access the uploaded files through the request object.

**20.What is a template engine, and how does Express.js use it?**

A template engine is a tool used to generate HTML or other output based on dynamic data. Express.js supports several template engines, such as EJS and Handlebars. These engines can dynamically render HTML pages based on data stored in the application.

**21.What is the difference between app.route() and app.use() in Express.js?**

app.route() defines multiple route handlers for a single route. While the app.use() function is used to add middleware functions to an application.

**22.What is the purpose of the req.params object in Express.js?**

The req.params object is used to access route parameters in Express.js. Route parameters capture values from the URL and pass them to the request handler.

**Req.param:**

Abc.com/route/1  
app.get('/:id', function (req, res) {

console.log(req.params['id']);

res.send();

});

**Req.query:**

Abc.com/route?name=rakesh

app.get('/profile', function (req, res) {

console.log(req.query.name);

res.send();

});

**23.What is the difference between req.query and req.params in Express.js?**

req.query is used to access the query parameters in a URL. While req.params is used to access route parameters in a URL.

**24.What is the purpose of the app.locals object in Express.js?**

The app.locals object stores application-level data in an Express.js application. This data is available to all templates and routes.

**25.How can you deal with error handling in Express.js? Explain with an example.**

Error handling is much easier in the Express versions over Express 4.0. Use the following steps to do the error handling:

Create an Express.js application. There is no built-in middleware like error handler in express 4.0, so you have to either install a middleware or create a custom one.

Create a Middleware:

Create a middleware as following:

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

**Install Error Handler Middleware:**

Install the errorhandler as following:

npm install errorhandler --save

**Create a variable:**

var errorhandler = require('errorhandler')

**Use the middleware as following:**

if (process.env.NODE\_ENV === 'development') {

// only use in development

app.use(errorhandler({log: errorNotification}))

}

function errorNotification(err, str, req) {

var title = 'Error in ' + req.method + ' ' + req.url

notifier.notify({

title: title,

message: str

})

}