**What is Express.js and what are its key features?**

Express.js is a popular web application framework for Node.js. It provides a minimalistic and flexible set of features for building web applications and APIs. Some key features of Express.js include routing, middleware support, template engine integration, and simplified handling of HTTP requests and responses.

**How do you create a basic Express.js application?**

To create a basic Express.js application, you need to install Express.js using npm and then create an instance of the Express application.

// Import the Express module  
const express = require('express');

// Create an instance of Express application   
const app = express();// Start the server  
app.listen(3000, () => {  
 console.log('Server is running on port 3000');  
});

**How do you define routes in Express.js?**

In Express.js, routes are defined using the app.get(), app.post(), app.put(), app.delete(), or app.use() methods. Here's an example of

//defining a GET route:   
app.get('/users', (req, res) => {  
 res.send('Get all users');  
});

**How do you handle URL parameters in Express.js?**

URL parameters can be accessed using **req.params**. Here's an example:

app.get('/users/:id', (req, res) => {  
 const userId = req.params.id;  
 res.send(`User ID: ${userId}`);  
});

**What is middleware in Express.js and how is it used?**

Middleware functions in Express.js are functions that have access to the **req**, **res**, and **next** objects. They can modify request and response objects, execute code, and invoke the next middleware function in the stack. Middleware is used for tasks such as logging, authentication, error handling, etc. Here's an example:

// Middleware function  
const logger = (req, res, next) => {  
 console.log(`Request URL: ${req.url}`);  
 next();  
};

// Register middleware  
app.use(logger);

**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. It is called the de facto standard server framework for Node.js. It was founded and developed by **TJ Holowaychuk** in 2010 and written in JavaScript.

**What are some distinctive features of Express?**

As Express is a lightweight, minimal and flexible Node.js web application framework, it provides a robust set of features for web and mobile applications. Following is the list of some distinctive features of this framework:

* js 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.

**Is Express.js front-end or backend framework?**

Express.js or Express is a JavaScript backend framework. It is mainly designed to develop complete web applications (single-page, multi-page, and hybrid web applications) and APIs. Express is the backend component of the **MEAN stack** where **M stands for MongoDB**, which handles database; **E stands for Express,** which handles backend; **A stands for AngularJS**, which is for the front-end, and **N stands for Node**.

**Why do we use Express.js?**

Express.js is an automatically prebuilt Node.js framework that facilitates us to create server-side web applications faster and smarter. The main reason for choosing Express is its simplicity, minimalism, flexibility, and scalability characteristics.

**What is the difference between Express.js and Node.js?**

Node.js is an open-source, cross-platform run-time environment used for executing JavaScript code outside of a browser. Node.js is not a framework or a programming language; it is a platform that acts as a web server. Many big companies such as Paypal, Uber, Netflix, Wallmart, etc., are using this. On the other hand, Express is a small framework based on the functionality of Node.js.

**Some key 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 runtime environment 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. |

**How does an Express code look like?**

The express.js program is saved with ".js" extension.

**See the example:**

var express = require('express');

var app = express();

app.get('/', function (req, res) {

  res.send('Welcome to JavaTpoint!');

});

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);

});

When you run the Node.js command prompt, the app will listen at the specified server address and give the following output.

**Output:**

Welcome to JavaTpoint!

**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.

There are two ways to install Scaffolding and use it in your application.

* Express application generator
* Yeoman

**Express application generator :-**

This is used to create an application skeleton quickly. Use the following command to install the Express application generator.

npm install express-generator -g

express myApp

By using the above command, a project named "myApp" will be created along with following the files/folders in the project.

**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.

Now, we have to install all the dependencies mentioned in the package.json file by using the following command:

cd myApp

npm install

**Yeoman :-**

Use the following command in your terminal to install Yeoman:

npm install -g yeoman

Yeoman uses generators to scaffold out applications.

**Do Other MVC frameworks also support scaffolding?**

The Scaffolding technique is also supported by other MVC frameworks other than Express. The following frameworks mainly support it: Ruby on Rails, OutSystems Platform, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, YII, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET, etc.

Click here for more information on Scaffolding: <https://www.javatpoint.com/expressjs-scaffolding>

**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.

Note: The third argument is optional and should be omitted; however, in some cases, it is helpful.

**What is the difference between Express and Django?**

Django is a standalone and lightweight web server for testing and development. On the other hand, Express.js is a Node.js framework that can set the middleware to reply to any HTTP request.

Following is a list of some key differences between Express.js and Django:

|  |  |  |
| --- | --- | --- |
| **Aspects** | **Express.js** | **Django** |
| Architecture | Express follows the MVC architecture. | Django supports the MTV (Model Template View) design. It uses managing data for interacting and validating. |
| Framework | Express is a free, open-source, lightweight, and fast backend web application framework for Node.js to build single-page, multi-page, and hybrid web applications and APIs. | This is a Python-based framework used to develop computer apps in a specified time frame. |
| Efficiency | It is best for developing web applications rapidly on Node.js. | It is more efficient and delivers at a fast speed so, it is cost-effective. |
| Programming language | The Express framework is programmed in Node.js. | Django is programmed in Python programming language. |
| Complexity | Express.js is less complex than Django. | Django is more complex than Express.js |
| Scalability | It provides better scalability. | It is less scalable. |
| Flexibility | Express is a flexible, minimal API-developing Node.js tool. | It provides limited flexibility. |
| Full-stack development | It provides a full-stack development that reduces the cost as you don't need to hire several developers to administer a web application's backend and frontend. | It does not deliver full-stack development. |
| Companies using this technology | Companies such as PayPal, IBM, Fox Sports, etc., are using this technology. | Companies such as Instagram, Mozilla, Bitbucket, etc., are using this technology. |

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

There are different ways to enable debugging in Express.js app in different Operating Systems

**Use the following command on Linux:**

DEBUG=express:\*

node app.js

**Use the following command on Windows:**

set DEBUG=express:\*

node app.js

**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();

});

**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

 })

}

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

See the Example:

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

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

**What is Middleware in Express.js? What are the different types of Middleware?**

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.

Note: If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

**Type of Middleware**

* Application-level Middleware
* Router-level Middleware
* Error-handling Middleware
* Built-in Middleware
* 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** }))

Note: You can use multiple middleware as an array on a single route.

**See the Example:**

var middlewareArray = [middleware1, middleware2]

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

  //Code snippets

})

**Which template engines do Express support?**

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

**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.

**Write a ‘Hello World’ Express.js application?**

To create a simple Express.Js application first we need to install Express in our NodeJs application.

npm install express

After that in the app.js file write the code

**const** express = require('express');

**const** app = express();

**const** PORT = ;

app.get('/', (req, res) => {

res.send('Hello World!');

});

app.listen(PORT, () => {

console.log(`Server is listening at port :**${**PORT**}**`);

});

**Mentions few features of Express.js.**

Few features of the Express.js includes

* **Routing:** Express provides a simple way to define routes for handling HTTP requests. Routes are used to map different URLs to specific pieces of code, making it easy to organize your application’s logic.
* **Middleware:** Express uses middleware functions to perform tasks during the request-response cycle. Middleware functions have access to the request, response, and the next middleware function.
* **HTTP Utility Methods:** Express mainly used for handling HTTP methods like GET, POST, PUT, and DELETE. This makes it easy to define how the application should respond to different types of HTTP requests.
* **Static File Serving:** It can also serve static files, such as images, CSS, and JavaScript, with the help of built-in express.static middleware.
* **Security:** It includes features and middleware to strengthen the security of your web applications, such as the helmet middleware to secure your app.

[**Explain the structure of an Express JS application?**](https://www.geeksforgeeks.org/how-to-structure-my-application-in-express-js/)

The structure of an Express JS application can vary greatly depending on its complexity and the specific needs of the project. However, here is a basic approach that is commonly used:

* **Entry point**: This is the starting point of the application where you set up your server, connect to your database, add middleware, and define the main routes.
* **Routes directory**: This directory contains files for the app’s routes.
* **Controllers directory**: This directory contains files that define the logic to handle requests for a specific route.
* **Models directory**: This directory is used for creating the schema models for the different data.
* **Middleware directory**: This directory contains custom middleware functions that you can use in your routes.
* **Views directory**: If you’re using a templating engine, this directory contains your view templates.
* **Public directory**: This directory contains static files that are served directly by the server such as images, CSS files, and JavaScript files.

This structure separates concerns in a logical way, making the application easier to understand and maintain.

**What are some popular alternatives to Express JS?**

There are several popular alternatives to Express.js which includes: Koa.js, Hapi.js, Sails.js, Fastify etc.

**Which major tools can be integrated with Express JS?**

There are many tools and libraries that can be integrated with Express.js such as:

* **Database tools**: MongoDB, MySQL, PostgreSQL.
* **Template Engines**: EJS, Pug, Mustache.
* **Authentication libraries**: Passport.js.
* **Logging libraries**: Morgan, Winston.
* **Validation libraries**: Joi, express-validator.
* **ORM libraries**: Sequelize, Mongoose.

**What is .env file used for?**

The .env file is used for storing sensitive information in a web application which we don’t want to expose to others like password, database connection string etc. It is a simple text file where each line represents a key-value pair, and these pairs are used to configure various aspects of the application.

[**What are JWT?**](https://www.geeksforgeeks.org/json-web-token-jwt/)

Json Web Tokens are mainly a token which is used for authentication and information exchange. When a user signs in to an application, the application then assigns JWT to that user. Subsequent requests by the user will include the assigned JWT. This token tells the server what routes, services, and resources the user is allowed to access. Json Web Token includes part namely- Header, Payload and Signature.

**Create a simple middleware for validating user.**

*// Simple user validation middleware*

**const** validateUser = (req, res, next) => {

**const** user = req.user;

*// Check if the user object is present*

**if** (!user) {

**return** res.status().json({ error: 'Unauthorized - User not found' });

}

*// If the user is valid, move to the next middleware or route handler*

next();

};

*// Example of using the middleware in an Express route*

app.get('/profile', validateUser, (req, res) => {

**const** user = req.user;

res.json({ message: 'Profile page', username: user.username });

});

[**What is Bcrypt used for?**](https://www.geeksforgeeks.org/password-verification-in-node-js/)

Bcrypt is a password hashing function which is used to securely hash and store user passwords. It is designed to be slow and computationally intensive, making it resistant to brute-force attacks and rainbow table attacks. Bcrypt is a key component in enhancing the security of user authentication systems.

[**What do you understand about ESLint?**](https://www.geeksforgeeks.org/eslint-pluggable-javascript-linter/)

EsLint is a JavaScript linting tool which is used for automatically detecting incorrect patterns found in ECMAScript/JavaScript code. It is used with the purpose of improving code quality, making code more consistent, and avoiding bugs. ESLint is written using Node.js to provide a fast runtime environment and easy installation via npm.

**Define the concept of the test pyramid.**

The Test Pyramid is a concept in software testing that represents the distribution of different types of tests. It was introduced by Mike Cohn, and it suggests that a testing strategy should be shaped like a pyramid, with the majority of tests at the base and fewer tests as you move up. The Test Pyramid consists of three levels: Unit Tests, Integration Tests, and End-to-End (EE) Tests.

[**Differentiate between res.send() and res.json().**](https://www.geeksforgeeks.org/difference-between-res-send-and-res-json-in-express-js/)

Both res.send() and res.json() serves similar purposes with some difference. So it depends on the data type which we are working with. Choose res.json() when you are specifically working with JSON data. Use res.send() when you need versatility and control over the content type or when dealing with various data types in your responses.

[**Explain what CORS is in Express JS?**](https://www.geeksforgeeks.org/how-to-allow-cors-in-express/)

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to control how web pages in one domain can request and interact with resources hosted on another domain.

In the context of Express.js, CORS refers to a middleware that enables Cross-Origin Resource Sharing for your application. This allows the application to control which domains can access your resources by setting HTTP headers.

**What are Built-in Middlewares?**

Express.js, includes a set of built-in middlewares that provide common functionality. These built-in middlewares are included by default when you create an Express application and can be used to handle various tasks. Here are some of the built-in middlewares in Express:

* express.json(): This middleware is used to parse incoming JSON requests. It automatically parses the request body if the Content-Type header is set to application/json.
* express.Router(): The express.Router() function is often used to create modular route handlers. It allows you to group route handlers together and then use them as a middleware.
* express.static(): This middleware is used to serve static files, such as images, CSS, and JavaScript files, from a specified directory.

[**How would you render plain HTML using Express JS?**](https://www.geeksforgeeks.org/how-to-render-plain-text-of-html-in-node-js/)

In Express.js, you can render plain HTML using the res.send() method or res.sendFile() method.

**Sample code:**

*//using res.send*

**const** express = require('express');

**const** app = express();

**const** port = ;

app.get('/', (req, res) => {

**const** htmlContent = '<html><body><h>Hello, World!</h></body></html>';

res.send(htmlContent);

});

app.listen(port, () => {

console.log(`Server is listening on port **${**port**}**`);

});

**What is the use of ‘Response.cookie()’ function?**

The response.cookie() function in Express.js is used to set cookies in the HTTP response. Cookies are small pieces of data sent from a server and stored on the client’s browser. They are commonly used to store information about the user or to maintain session data.

Basic syntax of response.cookie():

res.cookie(name, value, [options]);

[**Under what circumstances does a Cross-Origin resource fail in Express JS?**](https://www.geeksforgeeks.org/how-to-deal-with-cors-error-in-express-node-js-project/)

When a Cross-Origin Resource Sharing request is made, the browser enforces certain security checks, and the request may fail under various circumstances:

* No CORS Headers: The server doesn’t include the necessary CORS headers in its response.
* Mismatched Origin: The requesting origin does not match the origin specified in the Access-Control-Allow-Origin header.
* Restricted HTTP Methods: The browser enforces restrictions on which HTTP methods are allowed in cross-origin requests.
* No Credentials: The browser makes restrictions on requests that include credentials (such as cookies or HTTP authentication).

[**What is Pug template engine in Express JS?**](https://www.geeksforgeeks.org/how-to-add-pug-engine-in-express-js/)

Pug is a popular template engine for Express.js and other Node.js frameworks. You can use Pug to render dynamic HTML pages on the server side. It allows you to write templates using a syntax that relies on indentation and concise tags.

**What is meant by the sanitizing input process in Express JS?**

Sanitizing input in Express.js application is an important security practice to prevent various types of attacks, such as Cross-Site Scripting (XSS) and SQL injection. It involves cleaning and validating user input before using it in your application so that it does not contain malicious code or can be a security risk.

**Mention some third-party middleware provided by Express JS.**

Express.js allows you to use third-party middleware to extend and enhance the functionality of your web application. Here are some commonly used third-party middleware in Express.js:

* **body-parser**: This middleware is used to parse incoming request bodies, allowing you to access form data or JSON payloads on req.body.
* **cors**: This module provides middleware to enable Cross-Origin Resource Sharing (CORS) in your Express application.
* **morgan**: Morgan is a middleware module that provides request logging functionality.
* **helmet**: Helmet helps to secure Express apps by setting various HTTP headers.
* **express-session**: This middleware is used for managing user sessions in your Express application.
* **passport**: This middleware is used for implementing authentication and authorization in Express applications.

**When application-level Middleware is used?**

Application-level middlewares are bound to an instance of the Express application and are executed for every incoming request. These middlewares are defined using the app.use() method, and they can perform tasks such as logging, authentication, setting global variables, and more.

**Explain Router-level Middleware.**

Router-level middlewares are specific to a particular router instance. This type of middleware is bound to an instance of express.Router(). Router-level middleware works similarly to application-level middleware, but it’s only invoked for the routes that are handled by that router instance. This allows you to apply middleware to specific subsets of your routes, keeping your application organized and manageable.

[**How to secure Express.Js application?**](https://www.geeksforgeeks.org/node-js-securing-apps-with-helmet-js/)

It is very important to secure your application to protect it against various security threats. We can follow few best practices in our Express.js app to enhance the security of our application.

* **Keep Dependencies Updated**: Regularly update your project dependencies, including Express.js and other npm packages.
* **Use Helmet Middleware**: The helmet middleware helps secure your application by setting various HTTP headers. It helps prevent common web vulnerabilities.
* **Set Secure HTTP Headers**: Configure your application to include secure HTTP headers, such as Content Security Policy (CSP), Strict-Transport-Security (HSTS), and others.
* **Use HTTPS**: Always use HTTPS to encrypt data in transit. Obtain an SSL certificate for your domain and configure your server to use HTTPS.
* **Secure Database Access**: Use parameterized queries or prepared statements to prevent SQL injection attacks. Ensure that your database credentials are secure and not exposed in configuration files.

[**What is Express router() function?**](https://www.geeksforgeeks.org/express-js-express-router-function/)

The express.Router() function is used to create a new router object. This function is used when you want to create a new router object in your program to handle requests.

**Syntax:**

express.Router( [options] )

[**What are the different types of HTTP requests?**](https://www.geeksforgeeks.org/explain-different-types-of-http-request/)

The primary HTTP methods are commonly referred to as CRUD operations, representing Create, Read, Update, and Delete. Here are the main HTTP methods:

**GET**: The GET method is used to request data from a specified resource.

**POST**: The POST method is used to submit data to be processed to a specified resource.

**PUT**: The PUT method is used to update a resource or create a new resource if it does not exist.

**PATCH**: The PATCH method is used to apply partial modifications to a resource.

**DELETE**: The DELETE method is used to request that a specified resource be removed.

[**Which are the arguments available to an Express JS route handler function?**](https://www.geeksforgeeks.org/express-js-router-param-function/)

In Express JS route handler function, there are mainlyarguments available that provide useful information and functionality.

* req: This represents the HTTP request object which holds information about the incoming request. It allows you to access and manipulate the request data.
* res: This represents the HTTP response object which is used to send the response back to the client. It provides methods and properties to set response headers, status codes, and send the response body.
* next: This is a callback function that is used to pass control to the next middleware function in the request-response cycle.

[**How can you deal with error handling in Express.js?**](https://www.geeksforgeeks.org/explain-error-handling-in-express-js-using-an-example/)

Express.js provides built-in error-handling mechanism with the help of the next() function. When an error occurs, you can pass it to the next middleware or route handler using the next() function. You can also add an error-handling middleware to your application that will be executed whenever an error occurs.

**What is the difference between a traditional server and an Express.js server?**

A traditional server is a server that is built and managed independently. Traditional server may provide a basic foundation for handling HTTP requests and responses. While an Express.js server is built using the Express.js framework. It runs on top of Node.js. Express.js provides a simple and efficient way to create and manage web applications. It offers a wide range of features and tools for handling routing, middleware, and request or response objects.

[**What is the purpose of the next() function in Express.js?**](https://www.geeksforgeeks.org/what-is-the-use-of-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. If there are no next middleware function in the chain then it will give control to router or other functions in the app. If you don’t call next() in a middleware function, the request-response cycle can be terminated, and subsequent middleware functions won’t be executed.

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

app.route() is more specific to route handling and allows you to define a sequence of handlers for a particular route, on the other hand app.use() is a more general-purpose method for applying middleware globally or to specific routes.

**Explain what dynamic routing is in Express.js.**

Dynamic routing in Express.js include parameters, which allows you to create flexible and dynamic routes in your web application. This parameters are used in your route handlers to customize the behaviour based on the data provided.

In Express, dynamic routing is achieved by using route parameters, denoted by a colon (:) followed by the parameter name.

**const** express = require('express');

**const** app = express();

*// Dynamic route with a parameter*

app.get('/users/:userId', (req, res) => {

**const** userId = req.params.userId;

res.send(`User ID: **${**userId**}**`);

});

*// Start the server*

**const** port = ;

app.listen(port, () => {

console.log(`Server is listening on port **${**port**}**`);

});

[**How to serve static files in Express.Js?**](https://www.geeksforgeeks.org/express-js-express-static-function/)

In Express.js, you can serve static files using the built-in express.static middleware. This middleware function takes the root directory of your static files as an argument and serves them automatically.

[**What is the use of app.use() in Express.js?**](https://www.geeksforgeeks.org/express-js-app-use-function/)

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

### ****How does Express.js handle middleware?****

Express.js has a middleware system that allows developers to define and manage middleware functions. These functions can perform tasks such as authentication, validation, or modification of request and response objects. Middleware functions are executed in the order they are defined. They can be added to the application using the app.use() method.

### ****How does Express.js handle request and response objects?****

Express.js has a request and response object system that provides access to information about incoming requests and outgoing responses. The request object contains information about the incoming request, such as the URL, method, and headers. The response object is used to send a response back to the client. Developers can use methods such as res.send(), res.json(), and res.render() to send responses to the client.

### ****What is the difference between a traditional server and an Express.js server?****

A traditional server is a standalone server that is built and managed independently. While an Express.js server is built using the Express.js framework. It runs on top of Node.js. Express.js provides a simple and efficient way to create and manage web applications. It offers a wide range of features and tools for handling routing, middleware, and request or response objects.

### ****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.

### ****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.

### ****How does Express.js differ from other Node.js frameworks?****

Express.js is a flexible framework that provides only the essential features required for web application development. On the other hand, other Node.js frameworks, such as Meteor, Sails.js, and Koa.js, offer more features but may not be required for smaller projects. Express.js is a good choice for simple, fast, and scalable web applications.

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

app.use() is a method in Express.js used to mount middleware functions at a specified path. Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. These functions can execute any code, modify the request and response objects, end the request-response cycle, and call the next middleware in the stack.

const express = require('express');

const app = express();

app.use((req, res, next) => {

console.log('Middleware executed');

next();

});

app.get('/', (req, res) => {

res.send('Hello World');

});

app.listen(3000);

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

The next() function in Express.js is used to pass control to the next middleware function in the stack. If the current middleware does not end the request-response cycle, it must call next() to pass control to the next middleware. This is essential for creating a chain of middleware functions that handle various aspects of the request.

app.use((req, res, next) => {

console.log('First middleware');

next();

});

app.use((req, res, next) => {

console.log('Second middleware');

res.send('Hello World');

});

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

**app.route()** is used to create a chainable route handler for a specific path. It allows for the definition of multiple HTTP methods for the same route using a cleaner and more concise syntax.

app.route('/user')

.get((req, res) => {

res.send('Get a user');

})

.post((req, res) => {

res.send('Add a user');

})

.put((req, res) => {

res.send('Update a user');

});

**app.use()** is used to mount middleware functions or a group of routes at a specific path. It applies the middleware to all routes under the specified path.

app.use('/user', (req, res, next) => {

console.log('Request Type:', req.method);

next();

});

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

The req.params object in Express.js contains route parameters, which are named URL segments used to capture the values specified at their position in the URL. Route parameters are defined in the route path by prefixing a colon (:) before the parameter name.

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

const userId = req.params.id;

res.send(`User ID: ${userId}`);

});

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

**req.params**: Contains route parameters defined in the URL path. These are used for capturing values from named segments of the URL. Example:

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

console.log(req.params.id); // Accesses the value of :id in the URL

});

**req.query**: Contains query string parameters appended to the URL. These are used for capturing values specified in the query string. Example:

app.get('/search', (req, res) => {

console.log(req.query.q); // Accesses the value of the query parameter 'q'

});

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

The app.locals object in Express.js is used to store local variables that can be accessed throughout the lifetime of the application. These variables are available to the application’s views and can be used to pass data and configuration settings.

app.locals.siteTitle = 'My Express App';

app.get('/', (req, res) => {

res.render('index', {

title: app.locals.siteTitle

});

});

By using app.locals, you can maintain a global state or configuration that can be easily accessed and used across different parts of your application.

**Can you explain the role of middleware in Express JS?**

Middleware in Express JS serves as a function that has access to the request and response objects. Middleware can modify requests and responses, end the request-response cycle, or call the next middleware in the stack. Middleware enables powerful request processing capabilities in Express JS applications.

**How do you configure a view engine in Express JS?**

Configuring a view engine in Express JS involves setting the 'view engine' property of the Express application. This process enables Express JS to render views using the specified template engine. The view engine facilitates dynamic HTML rendering.

**What is the significance of the 'req' and 'res' objects in Express JS?**

The 'req' (request) and 'res' (response) objects in Express JS are central entities to handle HTTP requests and responses. The 'req' object provides details about the HTTP request. The 'res' object is used to return data to the client.

**How do you implement and use route parameters in Express JS?**

Implementing and using route parameters in Express JS involves defining variables within a route's path. Express JS then populates these parameters as properties of the req.params object, allowing the application to access and use them for various functionalities such as retrieving specific data.

**How do you create and manage sessions in Express JS?**

Creating and managing sessions in Express JS requires session middleware like express-session. The express-session middleware stores session data on the server and sends a session ID to the client. The session ID is usually a cookie. The session ID enables the application to maintain stateful interactions with users.

**How do you integrate Express JS with a database like MongoDB or MySQL?**

Integrating Express JS with a database like MongoDB or MySQL involves using respective Node.js drivers or ORM (Object Relational Mapping) libraries. Mongoose for MongoDB and Sequelize for MySQL provides a way to interact with the database. This allows Express JS applications to perform CRUD operations and manage data effectively.

**How do you handle cookies in Express JS?**

Handling cookies in Express JS requires cookie-parser middleware. The cookie-parser middleware parses cookies attached to the client request and makes them available in the req.cookies object. The cookie-parser middleware facilitates the management of session data and user preferences.