**Express.js Tutorial**

* Express is a web application framework that is built on the top of Node.js
* It is designed to built APIs and make connections to databases.
* It simplifies the development of server-side applications by offering an easy-to-use API for routing, middleware, and HTTP utilities.
* Express.js supports asynchronous routing, which allows you to handle non-blocking I/O operations, such as database queries or external API requests, without freezing the server.

**Installing Express:**

1. Download and Install Node.js
2. Creating a directory for our project and make that our working directory.

mkdir dir\_name

cd dir\_name

1. Using npm init command to create a package.json file for our project.

npm init

1. Installing Express: npm install express
2. Verify installation: npm --version express
3. Create file “server.js” and run using command: node server.js

**Creating an Express.js Server:**

1. Use **require(‘express’)** to import the Express module.
2. Call **express()** to create an Express application instance.
3. Define the **port** for the application, typically 3000.
4. Set up a basic **GET route** with app.get(‘/’, (req, res) => res.send(‘Hello World!’)).
5. Use **app.listen()** method to listen to your desired PORT and to start the server.

**Example:**

const express = require("express");

const app = express();

const port = 3000;

// Routing

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

    res.send("Home page"); // sends message to client

});

// start the server

app.listen(port, ()=>{

    console.log(`Server is running on port: ${port}`);

});

**Note:** Routes are the endpoints of the server, which are configured on our backend server and whenever someone tries to access those endpoints they respond accordingly to their definition at the backend.

**Syntax**: app.httpMethod(path, callback) // httpMethods = get, post, put, delete

**Response Methods:**

1. **res.send():** It sends the HTTP response. The body parameter can be a String or a Buffer object or an object or an Array.
2. **res.sendFile():** transfers the file at the given path and it sets the Content-Type response HTTP header field based on the filename extension.
3. **res.sendStatus()** function is used to set the response HTTP status code to statusCode and send its string representation as the response body.
4. **res.status()** function set the HTTP status for the response. It also allows to add message using above functions.
5. **res.download()** function transfers the file at the path as an ‘attachment’. Typically, browsers will prompt the user to download.
6. **res.end()** function is used to end the response process. This method actually comes from the Node core, specifically the response.end() method of HTTP.ServerResponse.
7. **res.get()** function returns the HTTP response header specified by the field. The match is case-insensitive.
8. **res.json()** function sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using the JSON.stringify() method.
9. **res.redirect()** function redirects to the URL derived from the specified path, with specified status, an integer (positive) which corresponds to an HTTP status code.
10. **res.render()** function is used to render views using a template engine (such as EJS, Handlebars, etc.), allowing **dynamic content** to be injected into the template before sending it to the client.

Syntax: res.render(view [, locals] [, callback])

1. **response.cookie()** function in ExpressJS 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.

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

**EJS Template Engine:**

* Template engine is used to create dynamic HTML pages from server.
* Popular Template Engines are EJS and PUG.
* EJS stands for Embedded JavaScript, which is a templating engine used to generate HTML with JavaScript. It allows you to embed JavaScript code into your HTML templates.

**Installation:** npm install ejs

* In order to set ejs in our project we write: app.set(‘view engine’, ‘ejs’);
* Create a folder named: “**views**” which has .ejs files (similar to html files).
* To render any file in our project, use app.render(“filename”, {name: “hanni”});
* Essential feature of Template Engine is that they allow us to create reusable components called as “**Partials**”.
* The **<% %>** tags in EJS are used to execute JavaScript code within an EJS template.
* **<%= %>** is used to output the value of a variable to the HTML.
* **'<%- %>'**is majorly used to directly insert HTML markup into your templates.

Data

Template Engine

HTML

Document

<p>Hello, <%= name%></p>

**Middleware in Express:**

Middleware in Express refers to functions that are used to perform tasks during the request-response cycle. These functions can modify request and response objects, end the request-response cycle, or call the next middleware function.

* They can perform tasks like authentication, logging, or error handling.
* Middleware helps separate concerns and manage complex routes efficiently.

**Syntax:**

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

console.log('Middleware executed');

next();

});

**Working of Middleware in Express.js:**

* In Express.js, middleware functions are executed sequentially in the order they are added to the application.
* When a request is modified, then modified request is passed through the middleware functions in the order they were defined.
* Each middleware can perform a task and either send a response or call the**next()**function to pass control to the next middleware function.

**Types of Middleware:**

1. **Application-level middleware:**It effects the entire application and executes for all routes. It is used by **app.use()** or **app.METHOD()** where METHOD = get, post, put, delete.
2. **Router-level middleware**: Associated with specific routes and executes for routes defined within that router. It is used by using**router.use()** or **router.METHOD()**.

For example:

router.get(“/user”, (req,res)=>{ console.log(“Middlewae”) });

1. **Error-handling middleware:**It is also an Application level middlewarebut it only handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
2. **Built-in middleware:**Pre-written middleware, provided by Express which we can use directly. (e.g., express.static(), Express.json(), etc.).
3. **Third-party middleware**: Developed by external packages (e.g., body-parser, CORS, etc.).

**Routing with Express.js**

* Express routing is responsible for defining how the server responds to different types of HTTP requests (e.g., GET, POST, PUT, DELETE) made to specific routes.
* The **express.Router()** function is used to create a new router object.
* Steps to setup Routes:

1. Create a folder named: ‘routes’
2. Inside ‘routes’ folder create logIn.js file which will handle ‘/login’ URL.
3. Create router object:

const express = require("express");

const router = express.Router();

1. Create a route:

router.get("/login",(req,res,next)=>{

res.send("This is the login request")

})

1. Export the Router: module.exports = router
2. Import all create routes in index.js file

const loginroute=require("./routes/login")

1. Handle route request: app.use("/",loginroute)

* Route parameters are dynamic values in the URL that can be accessed using req.params in Express. These are typically used to capture values that are part of the route, such as an ID or slug.

**Interview Questions:**

1. **What are some popular alternatives to ExpressJS?**

* Koa.js
* Hapi.js
* Sails.js
* Fastify

1. **Which major tools can be integrated with ExpressJS?**

* **Database tools:** [MongoDB](https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/), MySQL, PostgreSQL.
* **Template Engines:** EJS, Pug, Mustache.
* **Authentication libraries:** Passport.js.
* **Logging libraries:** Morgan, Winston.
* **Validation libraries**: Joi, express-validator.
* **ORM libraries:** Sequelize, Mongoose.

1. **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.

1. **What are JWT?**

[JSON Web Tokens](https://www.geeksforgeeks.org/json-web-token-jwt/) 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](https://www.geeksforgeeks.org/json-web-token-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 3 part namely- Header, Payload and Signature.

1. **What is Bcrypt used for?**

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.

1. **Why should you separate the Express app and server?**

In ExpressJS, it is recommended to separate the Express App and the server setup. This provides the modularity and flexibility and makes the codebase more easier to maintain and test.

1. **What is CORS in ExpressJS?**

[CORS (Cross-Origin Resource Sharing)](https://www.geeksforgeeks.org/cross-origin-resource-sharing-cors/) 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 ExpressJS, CORS refers to a middleware that enables Cross-Origin Resource Sharing for your application.

1. **How would you configure properties in ExpressJS?**

In ExpressJS, you can configure properties using the app.set() method. This method allows you to set various properties and options which affects the behavior of the Express application.

app.set(name, value);

1. **Elaborate on the various methods of debugging an express.js application?**

* **Console.log:** The simplest way to debug an ExpressJS application is by using console.log(). You can output messages to the console which can be viewed in the terminal.
* **Node Inspector:** This is a powerful tool that allows you to debug your applications using Chrome Developer Tools. It supports features like setting breakpoints, stepping over functions, and inspecting variables.
* **Visual Studio Code Debugger:** VS Code provides a built-in debugger that works on both Linux and Windows. It supports advanced features like conditional breakpoints, function breakpoints, and logpoints.
* **Utilizing debug module:** The debug module is a small NodeJS debugging utility that allows you to create debugging scopes.

1. **What is meant by the sanitizing input process in ExpressJS?**

Sanitizing input in ExpressJS 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.

1. **What is meant by Scaffolding in ExpressJS?**

[Scaffolding](https://www.geeksforgeeks.org/scaffolding-expressjs-app-scratch/) in ExpressJS refers to the process of generating a basic project structure automatically. This can speed up the initial setup and help maintain consistency in the way projects are structured, especially in large teams.

1. **How to generating a skeleton ExpressJS app using terminal command?**

To generate a skeleton for an ExpressJS application using the terminal, you can use the Express application generator which is a command-line tool provided by the ExpressJS framework. This generator will setup a basic directory structure which includes necessary files, and installs essential dependencies.

**Steps to generate:**

* + 1. Open your terminal and install the Express application generator globally using the following command: npm install -g express-generator
    2. After that you can use the express command to generate your ExpressJS app.

express my-express-app

* + 1. Now go to the app directory and install the dependencies and start the app by running-

npm install

npm start