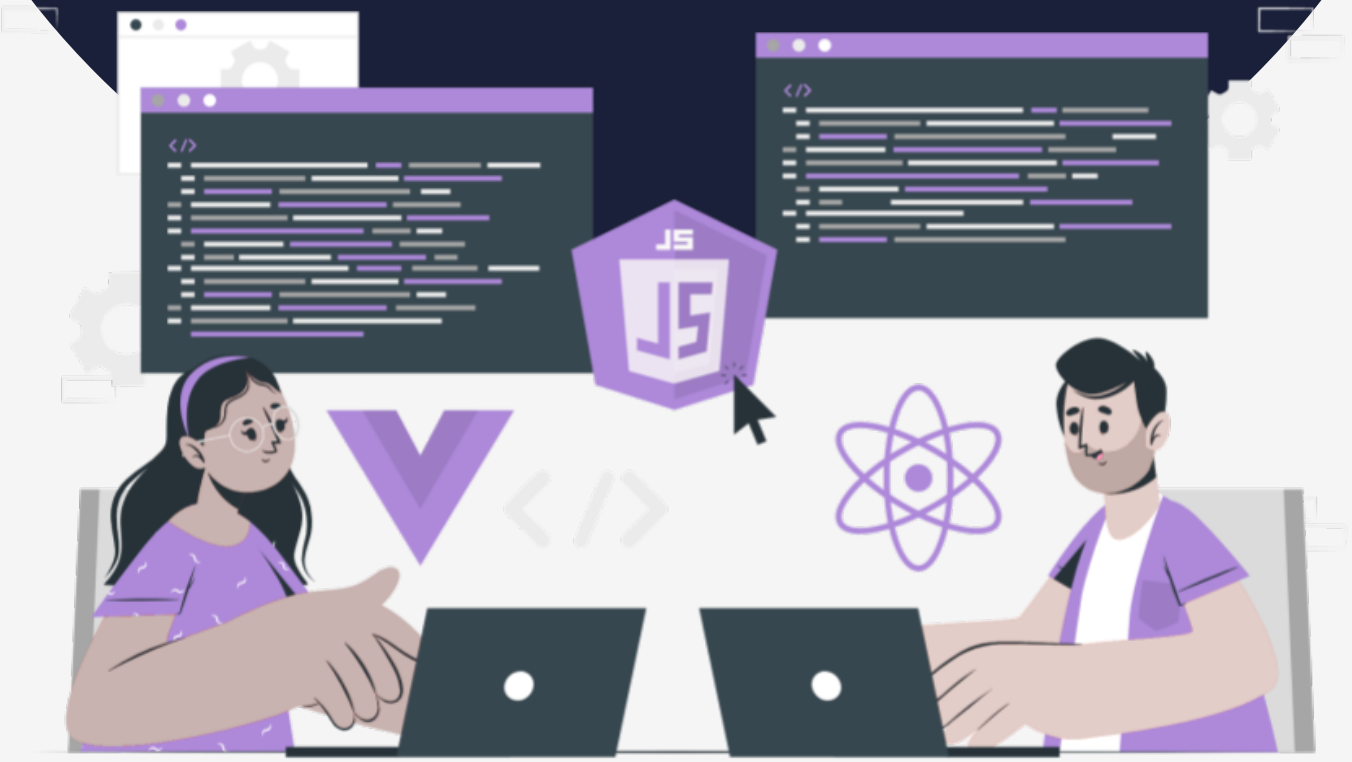
Lesson:

Basic of Routes with Express



PW SKILLS

Topics to be covered

- Definition
- Syntax
- Request and Response object in Express
- Routing
- Handling success and error
- Example of an E-commerce application

Definition

In an Express application, routes define how the application responds to client requests for specific URLs and HTTP methods. A route consists of a combination of an HTTP method (such as GET, POST, PUT, DELETE, etc.) and a URL pattern, which together determine which handler function should be executed to generate a response.

In Express, we can define routes using the **app.METHOD()** functions, where METHOD is the HTTP method name (e.g., get, post, put, delete, etc.). Each of these functions takes two arguments: the URL pattern to match and the handler function to execute.

Syntax

Each route can have one or more handler functions, which are executed when the route is matched.

Routes definition takes the following structure:

```
app.METHOD(PATH, HANDLER)
```

where -

- `app` is an instance of express (assume instance of express name as `app`
- METHOD is an HTTP request method, in lowercase i.e get, post, delete, put, patch, and so on.
- PATH is a URL path on the server
- HANDLER is the callback function executed when the route is matched

Request and Response object in Express

Request object

The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

In Express.js, the request method is used to retrieve information from an HTTP request sent to the server. It is represented by the `req` object that is available in the callback function of an Express route handler.

Some of the most frequently used request properties and methods are as follows -

- **`req.params` -** This property is an object that contains properties mapped to the named route parameters. For example, if you have a route `/users/:userId`, you can access the userId parameter using req.params.userId.

- **`req.query` -** This property is an object that contains properties mapped to the query string parameters in the URL. For example, if you have a URL **`/search?q=nodejs`**, you can access the `q` query parameter using **`req.query.q`**

- **`req.body`** - This property contains the parsed request body sent by the client in the HTTP request. This is typically used for POST, PUT, and PATCH requests where data is sent in the request body. For example, you can access the 'name' pass in the body by **`const {name} = req.body`**

- **`req.headers`** - This property is an object that contains the HTTP headers sent by the client in the request. Example - **`console.log(req.headers)`**, this will log the **`req.headers`** which is a key value map where each is a header name and each value corresponds to the value of the header. Some common headers include **'user-agent', 'accept`,** and **'content-type'**

- **`req.cookies`** - This property is an object that contains the cookies sent by the client in the request. Cookies object can be accessed using **`req.cookies`** Example - to access the value of the **"myCookie"** cookie **"req.cookies.myCookie"** can be used to access it.

- **`req.get(header)`** - This method is used to retrieve the value of a specific HTTP header in the request. Example, to retrieve the value of the User-Agent header, you can use **`req.get('User-Agent')`**

- **`req.ip`** - This property contains the IP address of the client that sent the request.
  Example -

```
console.log(req.ip)
// ⇒ '127.0.0.1
```

- **`req.path` -** This property contains the path part of the URL of the request.
  Example -

```
// example.com/users?sort=desc
console.dir(req.path)
// output ⇒ '/users'
```

- **`req.method` -** This property contains the HTTP method used by the request, such as GET, POST, PUT, etc.
  Example

```
// http GET request --
console.log(req.method)
// ⇒ GET
```

Response object

In Express.js, the response object (res) is used to send a response to the client that sent the HTTP request. The res object provides a variety of methods that can be used to send different types of responses, such as HTML pages, JSON data, and error messages.

Here are some commonly used res object:

- **res.send() –** This method sends a string, buffer, JSON object, or an HTML file as the response.
  Example

```
// send
res.send("hello world")
// ⇒ hello world
```

- **res.json() –** This method sends a JSON response.
  Example

```
res.json({
 success: true,
 message: "request successful"
})
```

- **res.render() –** This method renders an HTML view using a template engine like EJS or Handlebars.
  Example

```
// send the render view to the client
res.render("index")
```

- **res.status() –** This method sets the status code of the response.
  Example, res.status(404) sets the response status to 404 (Not Found).

- **res.redirect() –** This method redirects the client to a different URL.
  Example

```
// redirect to google.com
res.redirect('http://google.com')
```

- **res.download() –** This method sends a file as a download to the client.
  Example

```
 const filePath = '/path/to/file.pdf';
  const fileName = 'file.pdf';
  res.download(filePath, fileName)
```

- **res.sendFile() –** This method sends a file as a response to the client.
  Example

```
const filePath = '/path/to/file.pdf';
res.sendFile(filePath);
```

- **res.cookie() –** This method sets a cookie in the response.
  Example

```
// cookie with name myCookie with maxm,httpOnly & secure options
res.cookie('name', 'myCookie', {
 maxAge: 60 * 60 * 1000, // 1 hour
     httpOnly: true,
     secure: true
})
```

- **res.clearCookie() –** This method clears a cookie in the response.
  Example

```
// clear the cookie "myCookie"
res.clearCookie(myCookie)
```

Routing

Routing in Express.js involves several parts that work together to handle incoming HTTP requests and generate appropriate responses. Here are the main parts of routing in Express.js:

1. **Route methods –** Express.js supports several HTTP methods, such as GET, POST, PUT, DELETE, and others, which are used to define routes that handle requests of a specific type. You define these methods using the app.METHOD() methods, such as `app.get()`, `app.post()`, `app.put()`, and`app.delete()`

Example –

```
// GET method
app.get("/", (req, res) ⇒ {
 res.send('Hello world')
})
```

2. **Route paths –** Route paths, in combination with a request method, define the endpoints at which requests can be made. Route paths can be strings, string patterns, or regular expressions.

Example –

```
// path or endpoints
app.get('/about', (req, res) ⇒ {
  res.send('about')
})
```

3. **Route Handlers –** The route handler is a callback function that executes when the server receives a request matching the HTTP method and route path. This function can perform various tasks, such as sending a response, rendering a view, or forwarding the request to other middleware.

Example –

```
// A single callback function handler
app.get('/example', (req, res) => {
  res.send('Hello from A!')
})
```

4. **Route Parameters -** Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the `req.params` object, with the name of the route parameter specified in the path as their respective keys.

Example -

```
// Routes with route parameters
app.get('/users/:userId', (req, res) => {
  console.log(req.params) // params object
  console.log(req.params.userId // specifi userId object value
  res.send(req.params)
})
```

Handling Success and Error in Routing

There are several ways of handling error and success in an Express.js route:

1. Using the `res.status()` method to set the HTTP status code the `res.send()` or `res.json()` method to send the response data.

Example -

```
app.get('/users', (req, res) => {
  const users = [{ id: 1, name: 'John' }, { id: 2, name: 'Jane' }]

 if (users.length === 0) {
    // handling error
    res.status(200).json([])
  } else {
    // handle success
    res.status(200).json({users})
  }
})
/**
The curly braces around `users` are shorthand for `{users:users}`, which creates
an object with a property named `users` whose value is the `users` array.
**/
```

2. Using `try-catch` statement to handle the errors.

Example -

```
const fetchProfiles = async (req, res) => {
  try {
    // mock data user data query
    const user = await User.find();
    if (user) {
 // handle success
      return res.status(200).json({
        success: true,
        message: "profile fetch successfully",
        user,
      });
    }
  } catch (error) {
// handling error
res.status(500).json({
      success: false,
      message: "Unable to fetch profile",
    });
  }
};
```

Example of a News application.

```
// get all the newsfeeds
app.get("/newsfeeds", (req, res) => {

 // mock data query to news database
 // and store in news

  // respond with json news object
  res.json(news);
})
```

The above code is the example of an Express route that responds to an HTTP GET request to the URL (“/newsfeeds”).

When a GET request is made to the URL(“/newsfeeds”), the callback function specified in the second argument will be called. In this case, the function simply sends

The app instance is used to define the route, which is created by calling the express() function. The app.get() method is used to define a route for the HTTP GET method and the URL ("/newsfeeds"). The second argument to app.get() is a callback function that is called when a GET request is made to the URL. The req object contains information about the incoming request, while the res object sends a response object back to the client.

Some more examples are -
Respond to a PUT request to the /newsfeeds/:id route.

```
app.put('/newsfeeds/:id', (req, res) => {
  const newId = req.params.id
  const update = req.body
  // perform logic...
  // perform update logic
  res.send('news updated successfully!')
})
```

Respond to a DELETE request to the /newsfeed/:id route

```
app.delete('/newsfeed/:id', (req, res) => {
 const newsId = req.params.id
   // perform logic for delete of the particular id
 res.send("the particular newsfeed is deleted!")
})
```

Respond to a POST request on the route (/newsfeed), the application's home page.

```
app.post('/newsfeed', (req, res) => {
  const newsData = req.body
  // perform update to database logic
  res.send('news item created successfully')
})
```