ExpressJS and Restful APIs

Muhammad Zahir
Junejo

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- No question is daft or silly - **ask them!**
- Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- Should you have any questions after the lecture, please schedule a mentor session.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. What are RESTful APIs?
2. What is Express.js?
3. Benefits of Express.js
4. Setting up an Express.js project
5. Routes in Express.js

RESTful APIs

- RESTful API is an interface that two computer systems use to exchange information securely over the internet.
- For example, to generate monthly payslips, your internal accounts system has to share data with your customer's banking system to automate invoicing and communicate with an internal timesheet application. RESTful APIs support this information exchange.
- A gateway between clients and resources on the web.
- Clients are users who want to access information from the web.
- Resources are the information that different applications provide to their clients. Resources can be images, videos, text, numbers, or any type of data.

Yeah! But what is REST?

- Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work.
- Web services that implement REST architecture are called RESTful web services. The term RESTful API generally refers to RESTful web APIs.
- The client sends a request to the server after formatting the request according to the API documentation.
- The server authenticates the client and confirms that the client has the right to make that request.
- The server receives the request and processes it internally.
- The server returns a response to the client. Tells the client whether the request was successful and includes any information that the client requested.

What is Express?

- Fast, unopinionated, minimalistic and lightweight Node.js web application framework.
- Provides a solid foundation for building server side applications and APIs with ease.
- Advantages:
  - Flexibility: Express does not force a structure on you, allowing you to choose the components you need.
  - Routing: Express simplifies routing with its intuitive and expressive API.
  - Middleware: Middleware architecture allows you to extend and enhance your applications functionality.

Setting Up Express

1. Install Node.js and npm: Ensure you have node and npm installed.
2. Create a new directory.
3. npm init
4. ‘npm install express’ to install Express in your project.
5. Create a new file, name it ‘index.js’ and import the express module.
6. Create express application instance from the module you imported.

```
const express = require(‘express’);
const app = express()
```

Creating Routes

- Routes define the paths that users can navigate to on your web application.
- Routing refers to how an application's endpoints (URIs) respond to client requests.
- You define routing using methods of the Express app object that correspond to HTTP methods.
- Routing methods specify a callback function called when the application receives a request to the specified route (endpoint) and HTTP method. Application "listens" for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.
- Example:

```
app.get('/',(req, res) => {
        res.send('Hello, Express!');
})
```

- Format: app.METHOD(PATH, HANDLER)

Middleware

- Functions that are invoked sequentially during the request-response cycle.
- Functions that have access to the request object, the response object and the next function in the application's request-response cycle.
- The next function is a function in the Express router which, when called, executes the middleware succeeding the current middleware.
- Middleware function does not end the request-response cycle? Must call next() to pass control to the next middleware function. Or else the request will be left hanging.
- Common use cases:
  - Logging: Track incoming requests and server responses.
  - Authentication: Verify user identity before granting access.
  - Error handling: Catch and handle errors before they affect the application.

Handling Requests and Responses

- ‘req’ and ‘res’ are request and response objects respectively.
- Contain valuable information about the client’s request and the server’s response.
- Using Request Parameters:

```
app.get(‘/user/:id’, (req, res) => {
        const userId = req.params.id;
})
```

- Send responses:

```
app.get(‘/user/:id’, (req, res) => {
        const userId = req.params.id;
        res.status(200).json({data: data})
})
```

Templating

- Templating engines like EJS allow you to generate dynamic HTML content and inject data into your views.
- npm install ejs
- Set the view engine: app.set(‘view engine’, ‘ejs’)
- Create ‘views’ folder to store your templates.
- Inside the views folder create an ejs template (index.ejs).
- Use ejs tags to inject data: <%= username %>
- Render the template:

```
res.render( ‘index’, { username:  “Zahir” } )
```

References

- https://aws.amazon.com/what-is/restful-api/#:~:text=RESTful%20API%20is%20an%20interface,applications%20to%20perform%20various%20tasks.
- https://expressjs.com/en/starter/installing.html
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Questions and Answers

Thank You!