Web Applications

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 is Express.js?
2. Benefits of Express.js
3. Setting up an Express.js project
4. Routes in Express.js

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 install express’ to install Express in your project.
4. Create a new file, name it ‘index.js’ and import the express module.
5. 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.
- Example:

```
let logMiddleware = function(req, res, next){
        console.log(req);
        next();
}
app.get('/', logMiddleware, (req, res) => {
        res.send('Hello, Express!');
})
```

- Express supports routes of all types: get, post, put, delete…..

Middleware

- Functions that are invoked sequentially during the request-response cycle.

- 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!