Getting Started with Express

What is Express?

Express is a fast, unopinionated, and minimalist web framework for Node.js.
It simplifies the process of building web applications and APIs by providing a lightweight layer on top of Node.js.

Problems Solved by Express

- **Routing**: Express makes it easy to define routes and handle different types of HTTP requests.
- **Middleware**: It allows adding middleware functions for tasks like request/response processing and error handling.
- **Template Engines:** Express enables the integration of various template engines like Pug, EJS, and Handlebars for dynamic HTML pages.

Advantages of Express

- **Easy to learn and use:** Familiarity with Node.js and JavaScript makes it easy to start with Express.
- **High performance:** Built on top of Node.js, Express inherits its high-performance capabilities.
- **Extensibility:** Express is highly customizable and can be extended with middleware, plugins, and modules.

Use Cases for Express

Social media platforms, e-commerce websites, and online learning platforms can benefit from using Express as their backend framework.

Creating a Server using Express

Installing Express

To start working with Express, we must install it first. To do that, open your terminal or command prompt and navigate to your project folder. Then, type the following command: `npm install express`

Creating Server using Express and Using GET Method

Now that we have Express installed create a server using it. First, create a new file called **app.js** in your project folder. Then, add the following code:

1. Require the Express module in your file

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

2. Create an instance of the Express server

```
const server = express();
```

3. In your app.js file, define a route that handles default GET requests to the root URL (/)

```
server.get('/', (req, res) => {
  res.send('Hello World! This is Express server')
});
```

4. Start the server on specific port:

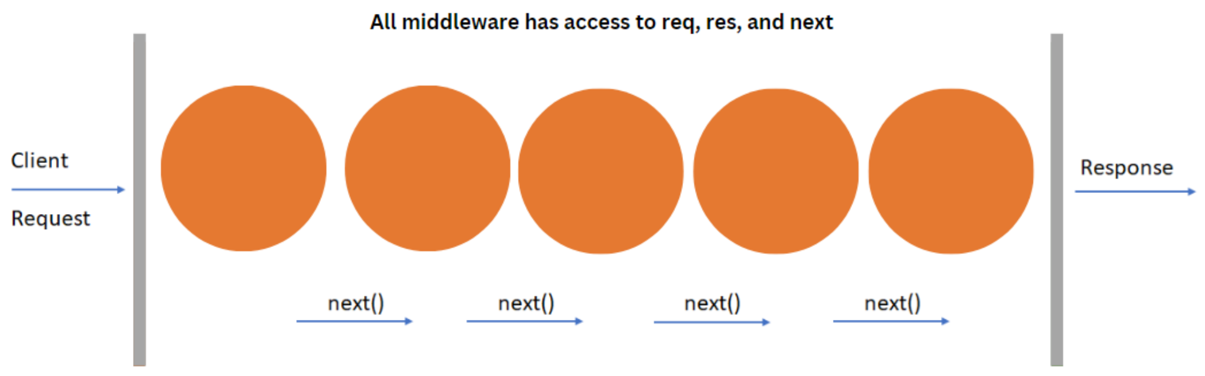
```
server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

5. In your terminal, run the `node app.js` command to start the server.
6. Visit http://localhost:3000 in your web browser to see the response from the server.

Explanation:

- The server.get() method defines a route that handles GET requests to a specific URL.
- Inside the route handler function, the req parameter represents the client's request, and the res parameter represents the server's response.
- To send a response back to the client, use the res.send() method.
- The server.listen() method is used to start the server and listen for incoming requests on a specific port.

Express Middleware and its Functionality



Middleware functions are a series of functions with access to the request object (req), the response object (res), and the next function in the application’s request-response cycle. These functions can execute code, modify request and response objects, or end the request-response cycle.

Examples of Middleware Functions

- Logging requests: Middleware function to log information about incoming requests.
- Parsing request bodies: Middleware function to parse and extract data from the request body.
- Handling authentication: Middleware function to authenticate and authorize user requests.

Example of Using Middleware

Handling GET Requests with Middleware:

```
const express = require('express');
const app = new express();

app.get('/', (req, res, next) => {
  console.log("Middleware 1 executed");
  next();
});

app.get('/', (req, res, next) => {
    res.send('GET request to the homepage from middleware 2');
});

app.listen(3000, () => console.log('Listening on port 3000'));
```

Explanation

- Two route handlers are defined using app.get() both match the root URL (/).
- The first route handler includes a middleware function that logs the message "Middleware 1 executed" to the console and then calls next() to pass control to the next middleware or route handler.
- The second route handler includes a middleware function that sends the response "GET request to the homepage from middleware 2" back to the client.

Different Types of Requests in Express

Express is a powerful framework for handling web requests and responses in Node.js. To build a functional web application, we need to handle different types of HTTP requests.

Common HTTP Request Methods

- **GET**: Used to retrieve data from the server.
```
app.get('/', (req, res) => {
  res.send('GET request received')
});
```

- **POST**: Used to send data to the server to create new resources.
```
app.post('/', (req, res) => {
  res.send('POST request received')
});
```

- **PUT**: Used to update existing resources on the server.
```
app.put('/', (req, res) => {
  res.send('PUT request received')
});
```

- **DELETE**: Used to remove resources from the server.
```
app.delete('/', (req, res) => {
  res.send('DELETE request received')
});
```

Note: Use Postman to test all these request methods.

HTTP Headers

HTTP headers are key-value pairs that carry additional information between the client and the server in an HTTP request or response. They help define how the data should be processed and provide metadata about the request or response.

Why We Need HTTP Headers

HTTP headers are necessary to:

- Provide metadata about the request or response, such as content type or length.
- Set cookies to store user-specific data.
- Control caching behavior.
- Communicate server-specific information to the client.

How to Use HTTP Headers in Express

In Express, you can set headers using the res.set() method.

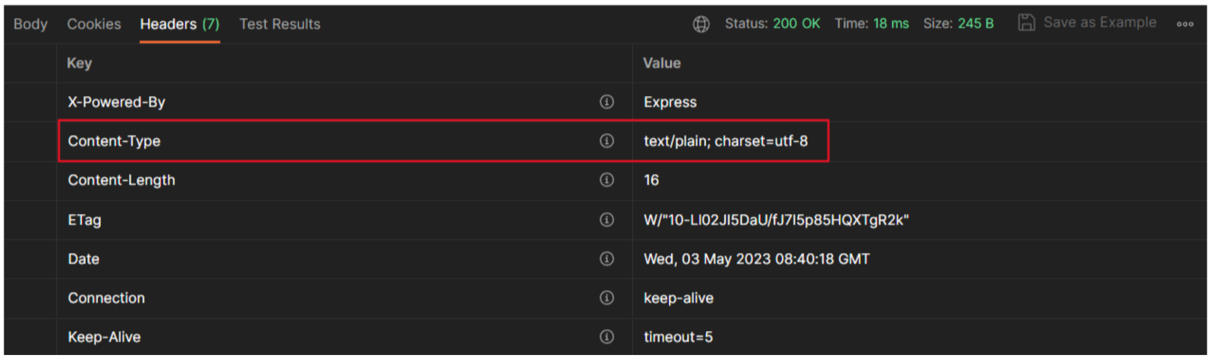
```
const express = require('express');
const app = new express();

app.get('/', (req, res) => {
   // Setting a response header
   res.set('Content-Type', 'text/plain')
   res.send('Request Received')
});
app.listen(3000, () => console.log('Listening on port 3000'));
```

In Postman, you can view the Content-Type header in the response details.

- Send a GET request to `http://localhost:3000` using Postman.
- Once you receive the response, you will see a " Headers " section in the response details.

- Expand the "Headers" section, and you should be able to see the Content-Type header along with its value.



Key	Value
X-Powered-By	Express
Content-Type	text/plain; charset=utf-8
Content-Length	16
ETag	W/"10-LI02JI5DaU/fJ7I5p85HQXTgR2k"
Date	Wed, 03 May 2023 08:40:18 GMT
Connection	keep-alive
Keep-Alive	timeout=5

HTTP response status codes

These are used to indicate the completion status of an HTTP request. These status codes are classified into five classes, each representing a different category of response:

1. **Informational responses (100 – 199)**: These status codes indicate that the server has received the request and is continuing to process it.
2. **Successful responses (200 – 299)**: Status codes in this range indicate that the request was successfully received, understood, and processed by the server.
3. **Redirection messages (300 – 399)**: Redirection status codes inform the client that further action is needed to complete the request.
4. **Client error responses (400 – 499)**: Status codes in this range indicate that there was an error on the client side, and the server cannot fulfill the request.
5. **Server error responses (500 – 599)**: Server error status codes indicate that the server encountered an error while processing the request.

How to Use Status Codes in Express

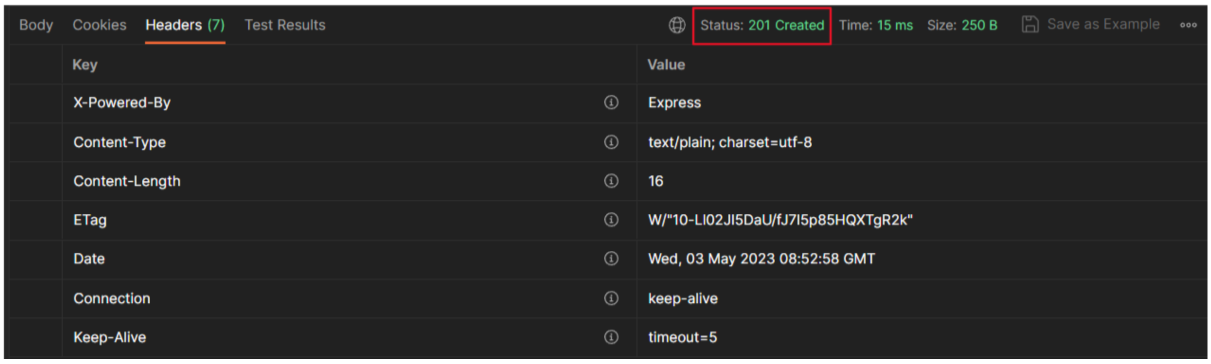
In Express, you can set the status code of a response using the res.status() method. By default, Express sets the status code to 200 for successful requests and 404 for unsuccessful ones.

```
const express = require('express');
const app = new express();

app.get('/', (req, res) => {
   // Setting a response header
   res.set('Content-Type', 'text/plain')

   // Setting status code
   res.status(201).send('Request Received')
});
app.listen(3000, () => console.log('Listening on port 3000'));
```

In Postman, you can see the status code in the response details. Send a GET request to `http://localhost:3000` using Postman.



Status: 201 Created   Time: 15 ms   Size: 250 B

Key	Value
X-Powered-By	Express
Content-Type	text/plain; charset=utf-8
Content-Length	16
ETag	W/"10-LI02JI5DaU/fJ7I5p85HQXTgR2k"
Date	Wed, 03 May 2023 08:52:58 GMT
Connection	keep-alive
Keep-Alive	timeout=5

Serving HTML Files

Serving HTML files allows to create more complex and dynamic web applications. Instead of sending just plain text responses, we can serve complete web pages that include styling, scripts, and other resources.

How to serve HTML files using Express?

1. Create a folder named "public" in the same directory as your Express app. This folder will contain your HTML file.

2. Put your index.html file to the "public" folder. This is the file that you want to serve.
3. Use express.static() middleware middleware to serve static files like HTML, CSS, and JavaScript. This middleware serves files from a specified directory.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Request Received');
});

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

app.listen(3000, () => {
  console.log('Server is listening on port 3000');
});
```

4. Potential issues: Serving files from the current directory may unintentionally expose sensitive files. By serving files from a specific folder like "public," we can avoid this issue.

5. Start your server and visit `http://localhost:3000/index.html` in your browser. You should see the content of your index.html file displayed.

Summarising it

Let's summarise what we have learned in this module:

- Learned about express and how to install it.
- Learned how to install and use Express to create a server for handling HTTP requests.
- Explored the concept of middleware and how to use it in Express to add additional functionality to our server.

- Discussed different types of HTTP requests, such as GET, POST, PUT, and DELETE, and how to handle them in Express.
- Learned about HTTP headers and their role in providing additional information in requests and responses. We also discussed HTTP status codes and their significance in indicating the outcome of a request.
- Discovered how to serve HTML files in Express by using the express.static() middleware and serving files from a specific directory.

Some Additional Resources:

- **How To Get Started with Node.js and Express?**
- **Behind the Scenes of ExpressJS**
- **HTTP request methods**
- **HTTP headers**
- **HTTP response status codes**
- **Serving static files in Express**