SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

Course Code and Name : 19TS601 FULL STACK DEVELOPMENT

Unit 3 : NODEJS AND EXPRESS

Topic : Serving static resource

Serving static resource

- In Node.js with Express, serving static resources means making files such as HTML, CSS, JavaScript, images, fonts, and videos available to clients (web browsers) so they can be accessed via HTTP requests.
- Express provides a built-in middleware function called express.static(), which allows developers to serve these static files efficiently.

What Are Static Resources?

- Static resources are files that do not change dynamically and remain the same for every user.

Examples include:

- HTML files (web pages)
- CSS files (styling of pages)
- JavaScript files (client-side scripts)
- Images, videos, and fonts (media resources)

These files are often placed in a directory like public/ or assets/ and are directly accessible via URLs.

Why Serve Static Resources?

Serving static resources in a web application has several advantages:

- **Improved Performance** – Files are served quickly without extra computation.
- **Better User Experience** – Faster load times due to cached static files.
- **Efficient Bandwidth Usage** – Reduces unnecessary server processing.
- **Caching Support** – Can leverage browser caching for faster load times.

Setting Up a Node.js Server with Express

Before serving static resources, install Express if it's not already installed:

sh

npm install express

Then, create a simple Express server:

```
const express = require('express');
const app = express();
// Start the server
app.listen(3000, () => {
    console.log('Server running on http://localhost:3000');
});
```

This sets up a basic Express server on **port 3000**.

Serving Static Files Using express.static()

Express provides a built-in middleware function called express.static() to serve static files.

Example: js

```
const express = require('express');
const app = express();
// Middleware to serve static files from "public" directory
app.use(express.static('public'));
app.listen(3000, () => {
    console.log('Server running on http://localhost:3000');
});
```

- Now, if you place a file like style.css inside the public/ folder, it can be accessed via:http://localhost:3000/style.css
- This method allows direct access to files inside the public/ folder.

Serving Static Files from Multiple Directories

- If your static resources are split into multiple folders, you can configure Express to serve files from multiple directories.

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

Files from both public/ and assets/ directories will be served.

Using a Virtual Path Prefix

- Sometimes, you may want to add a prefix to static files instead of exposing the directory structure.

Example:js

```
app.use('/static', express.static('public'));
```

Now, public/style.css can be accessed at:

http://localhost:3000/static/style.css

This prevents users from directly seeing the folder name.

- Restricting File AccessBy default, all files in the static folder are publicly accessible.
- However, you may want to restrict access to some files, such as private documents.
- Example:js

```
app.use('/secure', (req, res, next) => {
        if (!req.headers.authorization) {
            return res.status(403).send('Forbidden');
    }   next();
});
app.use('/secure', express.static('private'));
```

Now, only authorized users can access files in the private/ folder.

- Custom Error Handling for Static Files If a requested static file is missing, you can handle the error with custom middleware.
- Example:js

```
app.use((req, res, next) => {
res.status(404).send('File not found');
});
```

This sends a 404 error when the file is missing.

Optimizing Static File Delivery

- To improve performance, you can:
- **Enable caching**: Use cache headers to store files in the browser.
- **Use Gzip compression**: Compress files before sending them.
- **Use a Content Delivery Network** (CDN): Store files on a CDN for faster delivery.

Example of enabling caching:js

```
app.use(express.static('public', { maxAge: '1d' })); // Cache files for 1 day
```

- Serving static files in Express is essential for delivering web assets like CSS, JavaScript, and images.
- The express.static() middleware simplifies this process.
- It serve multiple directories, add virtual prefixes, and restrict access as needed.
- Performance optimizations like caching and CDNs further enhance efficiency.


Thank
You!