13. File Uploads and Static Assets

o Handling file uploads in Express.js with Multer

o Serving static files in Express.js

o Implementing basic image upload and display functionality

1. Handling File Uploads in Express.js with Multer

Overview:

- File uploads in Express.js are commonly handled using the Multer middleware, designed to handle multipart/form-data (form encoding used for file uploads).
- Multer enables you to save files to the server's disk or in memory.

Steps to Implement:

1. **Install Multer**:

```
npm install multer
```

Configure Multer: Multer requires a configuration for storing files. You can specify where and how files are stored.

- **Disk Storage**: Saves files to a specific folder on the server's disk.

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

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/'); // Folder where files will be stored
  },
  filename: (req, file, cb) => {
    const uniqueName = Date.now() + '-' + file.originalname;
    cb(null, uniqueName); // Save file with a unique name
  }
});

const upload = multer({ storage: storage });
```

Memory Storage: Stores files in memory as a Buffer object.

```
const memoryStorage = multer.memoryStorage();
const upload = multer({ storage: memoryStorage });
```

Create an Upload Route: Use Multer's middleware in your Express routes to handle file uploads.

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

app.post('/upload', upload.single('file'), (req, res) => {
  res.send(`File uploaded: ${req.file.filename}`);
});

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

Test Your File Upload: Use tools like Postman to send a POST request to /upload with a file in the body.

Handling Multiple File Uploads in Express.js with Multer

Overview:

Multer can handle multiple file uploads using the upload.array() method. This is useful when you need to allow users to upload multiple files at once (e.g., uploading multiple images or documents).

Steps to Handle Multiple Files:

1. **Configure Multer for Multiple File Uploads**: Multer's array() method allows you to specify the field name in the form and the maximum number of files to accept.

   Example configuration:

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

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/'); // Directory for saving files
  },
  filename: (req, file, cb) => {
    const uniqueName = Date.now() + '-' + file.originalname;
    cb(null, uniqueName); // Save each file with a unique name
  }
});

const upload = multer({ storage: storage });
```

Set Up a Route for Multiple Files: Use upload.array(fieldname, maxCount) to handle multiple file uploads.

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

app.post('/upload-multiple', upload.array('files', 5), (req, res) => {
  const fileNames = req.files.map(file => file.filename);
  res.send(`Files uploaded successfully: ${fileNames.join(', ')}`);
});

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

- fieldname: Name of the input field used in the form for file uploads (e.g., <input type="file" name="files" multiple>).
- maxCount: Maximum number of files allowed for this field.

Testing Multiple File Uploads:

- Use tools like **Postman** to send a POST request to /upload-multiple.
- Use the form-data option to upload multiple files under the field name files

2. Serving Static Files in Express.js

Refer to the earlier notes for setting up express.static() middleware to serve static files like uploaded files, images, CSS, and JavaScript files.

When handling multiple file uploads, ensure your upload directory (e.g., uploads/) is served using

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

This allows users to access uploaded files via a URL.

3. Implementing Multiple Image Upload and Display Functionality

Uploading Multiple Images:

1. **File Filter for Images**: To ensure only image files are uploaded, use a file filter:

```
let upload = multer({
   storage:storage,
   fileFilter:function(req,file,cb){
      console.log(file)
      if(file.mimetype === 'image/png'){
         cb(null,true)
      } else {
         console.log('Only PNG files are supported!');}
```

```
    }

})
```

Create a Route for Multiple Image Uploads: Handle multiple images in a single request.

```
app.post('/upload-images', upload.array('images', 10), (req, res) => {
    const imagePaths = req.files.map(file => `/uploads/${file.filename}`);
    res.send(`
      <h1>Images Uploaded Successfully!</h1>
      ${imagePaths.map(path => `<img src="${path}" alt="Uploaded Image" style="max-
width: 200px; margin: 10px;">`).join('')}
    `);
  });
```

images: Field name in the form for uploading multiple images.
10: Maximum number of files allowed.

interview questions and answers

Q1: What is Multer, and why is it used in Express.js?

- **Answer**:
  Multer is a middleware for handling multipart/form-data, primarily used for uploading files in Express.js. It processes files sent via forms and makes them available in the req.file or req.files object, depending on whether a single or multiple files are uploaded.

Q2: What are the storage options provided by Multer? Explain their use cases.

- **Answer**:
  Multer provides two main storage options:
  - **DiskStorage**: Files are stored directly on the disk in a specified directory.
    - Use Case: Suitable when the application has sufficient disk space and file access requirements are straightforward.
  - **MemoryStorage**: Files are stored in memory as Buffer objects.
    - Use Case: Useful when files need to be processed immediately (e.g., resized or uploaded to a cloud service).

Q3: How would you implement file type validation in Multer?

- **Answer**:
  File type validation can be implemented using the fileFilter option in Multer.

```
const fileFilter = (req, file, cb) => {
   if (file.mimetype.startsWith('image/')) {
     cb(null, true); // Accept the file
   } else {
     cb(new Error('Not an image!'), false); // Reject the file
   }
 };

 const upload = multer({ storage, fileFilter });
```

Q4: How do you handle single and multiple file uploads using Multer?

- **Answer**:
  - **Single File Upload**: Use upload.single('fieldname').
    Example:

```
app.post('/upload', upload.single('file'), (req, res) => {
   res.send(`File uploaded: ${req.file.filename}`);
 });
```

Multiple File Uploads: Use upload.array('fieldname', maxCount).
Example:

```
app.post('/upload-multiple', upload.array('files', 5), (req, res) => {
   res.send(`Files uploaded: ${req.files.map(file => file.filename).join(', ')}`);
 });
```

Q5: How can you limit the size of uploaded files in Multer?

- **Answer**:
  Use the limits option in Multer to specify the maximum file size (in bytes).

```
const upload = multer({
   storage,
   limits: { fileSize: 5 * 1024 * 1024 } // Limit to 5MB
 });
```

Q6: What is the purpose of express.static() middleware in Express.js?

- **Answer**:
  The express.static() middleware is used to serve static files such as HTML, CSS,

JavaScript, images, and fonts. It allows clients to access files directly via HTTP requests.

Q7: How do you serve static files from a specific directory in Express.js?

- **Answer**:
  Use app.use() with express.static() to serve files from a directory.
  Example:

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

This serves files in the public/ directory, accessible via URLs like http://localhost:3000/file.jpg.

Q8: How can you serve static files from a custom URL path?

- **Answer**:
  You can specify a custom URL path for serving static files.

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

Files in the public/ folder are accessible under the /static path (e.g., http://localhost:3000/static/file.jpg).

Q9: What are common security concerns when serving static files?

- **Answer**:
  - **Directory Traversal Attacks**: Ensure the express.static() middleware is configured properly to avoid exposing unintended files.
  - **Cache Control**: Set proper headers to manage caching behavior.
  - **File Injection**: Avoid serving user-uploaded files directly without validation.

Q10: How would you implement a basic image upload functionality in Express.js?

- **Answer**:
  Steps:
  1. Configure Multer with diskStorage for saving images

```
const multer = require('multer');
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    const uniqueName = Date.now() + '-' + file.originalname;
```

```
    cb(null, uniqueName);
  }
});
const upload = multer({ storage });
```

Create a route to handle image uploads

```
app.post('/upload-image', upload.single('image'), (req, res) => {
    res.send(`Image uploaded: ${req.file.filename}`);
  });
```

Q11: How do you display uploaded images to the client?

- **Answer**:
  Steps:
  1. Serve the upload directory using express.static()

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

2. Use the image URL in the front-end to display the image

```
<img src="/uploads/your-uploaded-image.jpg" alt="Uploaded Image">
```

Q12: How can you handle multiple image uploads and display them dynamically?

- **Answer**:
  - Handle multiple image uploads using upload.array()

```
app.post('/upload-images', upload.array('images', 10), (req, res) => {
    const imagePaths = req.files.map(file => `/uploads/${file.filename}`);
    res.send(imagePaths);
  });
```

Dynamically display the images:

```
<script>
  fetch('/upload-images').then(res => res.json()).then(imagePaths => {
    imagePaths.forEach(path => {
      const img = document.createElement('img');
      img.src = path;
      document.body.appendChild(img);
    });
  });
</script>
```

Q13: What are best practices for handling user-uploaded files in an Express.js application?

- **Answer**:
    - **Validation**: Validate file type and size using Multer's fileFilter and limits options.
    - **Directory Management**: Use a dedicated directory for user uploads and ensure it is secured.
    - **Serve Files Safely**: Avoid serving uploaded files directly without validation. Use express.static() carefully.
    - **Error Handling**: Implement proper error handling for invalid file uploads.
    - **Cloud Storage**: For scalability, consider uploading files to cloud storage (e.g., AWS S3, Google Cloud Storage).

Q14: Write an Express.js program that allows unlimited image uploads and displays all uploaded images dynamically on the client.

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

const app = express();

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    const uniqueName = Date.now() + '-' + file.originalname;
    cb(null, uniqueName);
  }
});

const upload = multer({ storage });

app.use('/uploads', express.static('uploads'));

app.post('/upload-images', upload.array('images'), (req, res) => {
  const imagePaths = req.files.map(file => `/uploads/${file.filename}`);
  res.send(imagePaths);
});

app.get('/', (req, res) => {
  res.send(`
    <h1>Upload Images</h1>
    <form action="/upload-images" method="post" enctype="multipart/form-data">
      <label for="images">Choose Images:</label>
      <input type="file" id="images" name="images" multiple>
      <button type="submit">Upload</button>
    </form>
  `);
```

```
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```