11. File Sharing System

Creating a secure file-sharing system using the MEAN stack (MongoDB, Express.js, Angular, Node.js) is a substantial project. Here's a high-level overview and code snippets to guide you:

Project Setup and Structure

Set up a new project folder and structure for your File Sharing system. Install the required Node.js packages and create a basic Angular application.

```
# Create a new Angular application
ng new file-sharing-app
```

- Backend (Node.js & Express.js)

Create the backend of your File Sharing system using Node.js and Express.js.

Installation of Packages

Install the necessary packages for Express.js, Mongoose (for MongoDB), and other dependencies.

```
npm install express mongoose cors multer bcryptjs jsonwebtoken
```

Setting up Express.js

Create your Express.js server, set up middleware, and handle routes.

- **javascript**

```
// server.js

const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const multer = require('multer');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const app = express();

// Middleware
app.use(express.json());
app.use(cors());

// Database connection
mongoose.connect('mongodb://localhost/file-sharing-app', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
  useCreateIndex: true,
});

// Define Mongoose models for User and File data
const User = mongoose.model('User', {
  username: String,
  password: String,
  role: String, // "user" or "admin" for example
});

const File = mongoose.model('File', {
  filename: String,
  path: String,
  owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
});

// Multer storage for file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
```

```
  filename: (req, file, cb) => {
    cb(null, Date.now() + '-' + file.originalname);
  },
});

const upload = multer({ storage });

// Routes for user authentication and file upload/download
app.post('/api/register', async (req, res) => {
  // Register a new user
  // Store hashed password in the database
});

app.post('/api/login', async (req, res) => {
  // Authenticate user, generate JWT token
});

app.post('/api/upload', upload.single('file'), async (req, res) => {
  // Upload and store the file in the database
});
```

```
app.get('/api/files', (req, res) => {
  // Retrieve a list of files with user permissions
});

// Create similar routes for user permission management, file deletion, etc.
```

- ## Frontend (Angular)

Create the frontend of your File Sharing system using Angular. Design the user interface, implement file upload/download functionality, and user management.

v

Design and UI

Design the user interface for your File Sharing system using Angular components, templates, and styles.

File Upload/Download

Create components and forms for users to upload and download files. Ensure proper security measures, such as user authentication and role-based access.

- **typescript**

```
// file-upload.component.ts
```

```
import { Component } from '@angular/core';
import { FileService } from './file.service';

@Component({
  selector: 'app-file-upload',
  templateUrl: './file-upload.component.html',
})
export class FileUploadComponent {
  file: File;
  filename: string;

  constructor(private fileService: FileService) {}

  uploadFile() {
    this.fileService.uploadFile(this.file).subscribe((data) => {
      // Handle success response
    });
  }
}
```

MongoDB

Create a MongoDB database to store user profiles, file metadata, and permissions.

Authentication and Permissions

Implement user authentication and authorization, ensuring that users can only access files and perform actions for which they have permission.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle file uploads, downloads, and user permissions properly.

Building a secure file-sharing system with user permissions is a complex project. Consider adding features like encryption, auditing, and access control lists (ACLs) for advanced security and control over shared files.