React-formdata

// backend code

**// app.js**

// Import required modules

const express = require('express');

const multer = require('multer');

const path = require('path');

const mongoose = require('mongoose');

// Create an instance of Express

const app = express();

// Set up middleware for parsing JSON data

app.use(express.json());

// Set up multer for handling file uploads

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/');

},

filename: function (req, file, cb) {

const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1e9);

cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));

}

});

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

// Connect to MongoDB

mongoose.connect('mongodb://localhost/formdata', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Connected to MongoDB'))

.catch((err) => console.error('Failed to connect to MongoDB', err));

// Define a schema for the form data

const formDataSchema = new mongoose.Schema({

name: String,

dob: Date,

gender: String,

hobbies: [String],

state: String,

address: String,

resume: String

});

// Create a model based on the schema

const FormData = mongoose.model('FormData', formDataSchema);

// Define an API endpoint for form submission

app.post('/api/form', upload.single('resume'), (req, res) => {

// Extract form data from the request body

const { name, dob, gender, hobbies, state, address } = req.body;

const resume = req.file.filename;

// Validate the form data

if (!name || !dob || !gender || !hobbies || hobbies.length < 2 || !state || !address || !resume) {

return res.status(400).json({ error: 'Invalid form data' });

}

// Create a new form data instance

const formData = new FormData({

name,

dob,

gender,

hobbies,

state,

address,

resume

});

// Save the form data to the database

formData.save()

.then(() => res.status(200).json({ message: 'Form submitted successfully' }))

.catch((err) => res.status(500).json({ error: 'Failed to submit form' }));

});

// Start the server

const port = 3000;

app.listen(port, () => console.log(`Server running on port ${port}`));

// backend code to handle fetching the form data and serving it in a table format

// Define an API endpoint for fetching form data

app.get('/api/form', (req, res) => {

// Fetch all form data from the database

FormData.find()

.then((data) => res.status(200).json(data))

.catch((err) => res.status(500).json({ error: 'Failed to fetch form data' }));

});

// Serve the uploaded resume files

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

// Start the server

const port = 3000;

app.listen(port, () => console.log(`Server running on port ${port}`));

backend code to handle downloading the resume files:

// Define an API endpoint for downloading resume files

app.get('/api/form/:id/resume', (req, res) => {

// Fetch the form data by ID from the database

FormData.findById(req.params.id)

.then((data) => {

if (!data) {

return res.status(404).json({ error: 'Form data not found' });

}

// Set the appropriate headers for file download

res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document');

res.setHeader('Content-Disposition', `attachment; filename=${data.resume}`);

// Serve the resume file

res.sendFile(path.join(\_\_dirname, 'uploads', data.resume));

})

.catch((err) => res.status(500).json({ error: 'Failed to download resume' }));

});

// Start the server

const port = 3000;

app.listen(port, () => console.log(`Server running on port ${port}`));

To handle only .docx files being submitted on the backend, you can add a file filter to the multer middleware configuration. Here's an updated version of the backend code with the file filter implemented:

// Set up multer for handling file uploads

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/');

},

filename: function (req, file, cb) {

const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1e9);

cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));

}

});

const fileFilter = (req, file, cb) => {

if (file.mimetype === 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') {

cb(null, true);

} else {

cb(new Error('Only .docx files are allowed'));

}

};

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

**backend code to handle deleting form data:**

// Define an API endpoint for deleting form data

app.delete('/api/form/:id', (req, res) => {

// Find the form data by ID and delete it from the database

FormData.findByIdAndDelete(req.params.id)

.then((data) => {

if (!data) {

return res.status(404).json({ error: 'Form data not found' });

}

// Delete the associated resume file

fs.unlink(path.join(\_\_dirname, 'uploads', data.resume), (err) => {

if (err) {

console.error('Failed to delete resume file', err);

}

});

res.status(200).json({ message: 'Form data deleted successfully' });

})

.catch((err) => res.status(500).json({ error: 'Failed to delete form data' }));

});

// Start the server

const port = 3000;

app.listen(port, () => console.log(`Server running on port ${port}`));

**// Another way**

App.js

// server.js

const express = require('express');

const mongoose = require('mongoose');

const multer = require('multer');

const path = require('path');

const fs = require('fs');

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/formDB', { useNewUrlParser: true, useUnifiedTopology: true });

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function() {

console.log('Connected to MongoDB');

});

// Create a Mongoose schema for the form data

const formDataSchema = new mongoose.Schema({

name: String,

dob: Date,

gender: String,

hobbies: [String],

country: String,

state: String,

address: String,

resumePath: String

});

const FormData = mongoose.model('FormData', formDataSchema);

// Set up multer for file uploads

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/')

},

filename: function (req, file, cb) {

cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname))

}

});

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

// Handle form submissions

app.post('/submit-form', upload.single('resume'), (req, res) => {

const { name, dob, gender, hobbies, country, state, address } = req.body;

const resumePath = req.file.path;

const newFormData = new FormData({

name,

dob,

gender,

hobbies,

country,

state,

address,

resumePath

});

newFormData.save((err, formData) => {

if (err) {

console.error(err);

res.status(500).send('Error saving form data');

} else {

res.status(200).json(formData);

}

});

});

// Serve resume file for download

app.get('/download-resume/:id', (req, res) => {

FormData.findById(req.params.id, (err, formData) => {

if (err) {

console.error(err);

res.status(500).send('Error finding form data');

} else {

const resumeFilePath = formData.resumePath;

res.download(resumeFilePath);

}

});

});

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

// server.js

const express = require('express');

const mongoose = require('mongoose');

const multer = require('multer');

const path = require('path');

const fs = require('fs');

const app = express();

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/formDB', { useNewUrlParser: true, useUnifiedTopology: true });

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function() {

console.log('Connected to MongoDB');

});

// Create a Mongoose schema for the form data

const formDataSchema = new mongoose.Schema({

name: String,

dob: Date,

gender: String,

hobbies: [String],

country: String,

state: String,

address: String,

resumePath: String

});

const FormData = mongoose.model('FormData', formDataSchema);

// Set up multer for file uploads

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/')

},

filename: function (req, file, cb) {

cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname))

}

});

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

// Handle form submissions

app.post('/submit-form', upload.single('resume'), (req, res) => {

const { name, dob, gender, hobbies, country, state, address } = req.body;

const resumePath = req.file.path;

const newFormData = new FormData({

name,

dob,

gender,

hobbies,

country,

state,

address,

resumePath

});

newFormData.save((err, formData) => {

if (err) {

console.error(err);

res.status(500).send('Error saving form data');

} else {

res.status(200).json(formData);

}

});

});

// Serve resume file for download

app.get('/download-resume/:id', (req, res) => {

FormData.findById(req.params.id, (err, formData) => {

if (err) {

console.error(err);

res.status(500).send('Error finding form data');

} else {

const resumeFilePath = formData.resumePath;

res.download(resumeFilePath);

}

});

});

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

Got it! To ensure that the uploaded resume is only in the docx format, you can add a file type validation check using the fileFilter option provided by multer. Here's how you can modify the file upload setup to enforce the docx format requirement:

// server.js (continued)

// Set up multer for file uploads

const storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, 'uploads/')

},

filename: function (req, file, cb) {

cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname))

}

});

const fileFilter = (req, file, cb) => {

if (file.mimetype === 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') {

cb(null, true);

} else {

cb(new Error('File type not supported. Please upload a .docx file.'), false);

}

};

const upload = multer({

storage: storage,

fileFilter: fileFilter

});

// Handle form submissions

app.post('/submit-form', upload.single('resume'), (req, res) => {

// ... (previous code)

});

To enforce the requirement that at least two hobbies need to be checked, you can add a validation check in the backend before saving the form data. Here's how you can modify the form submission endpoint to include this validation:

// server.js (continued)

// Handle form submissions

app.post('/submit-form', upload.single('resume'), (req, res) => {

const { name, dob, gender, hobbies, country, state, address } = req.body;

const resumePath = req.file.path;

// Check if at least two hobbies are selected

if (hobbies && hobbies.length >= 2) {

const newFormData = new FormData({

name,

dob,

gender,

hobbies,

country,

state,

address,

resumePath

});

newFormData.save((err, formData) => {

if (err) {

console.error(err);

res.status(500).send('Error saving form data');

} else {

res.status(200).json(formData);

}

});

} else {

res.status(400).send('Please select at least two hobbies.');

}

});

**Certainly! Below is an example of how you might structure the frontend code for the project using React to create the user interface for the form and table display:**

// App.js

import React, { useState } from 'react';

import axios from 'axios';

function App() {

const [formData, setFormData] = useState({

name: '',

dob: '',

gender: '',

hobbies: [],

country: '',

state: '',

address: '',

resume: null

});

const handleInputChange = (e) => {

const { name, value } = e.target;

setFormData({ ...formData, [name]: value });

};

const handleCheckboxChange = (e) => {

const { value, checked } = e.target;

let updatedHobbies = [...formData.hobbies];

if (checked) {

updatedHobbies.push(value);

} else {

updatedHobbies = updatedHobbies.filter((hobby) => hobby !== value);

}

setFormData({ ...formData, hobbies: updatedHobbies });

};

const handleFileChange = (e) => {

setFormData({ ...formData, resume: e.target.files[0] });

};

const handleSubmit = async (e) => {

e.preventDefault();

try {

const response = await axios.post('/submit-form', formData, {

headers: {

'Content-Type': 'multipart/form-data'

}

});

console.log('Form submitted:', response.data);

// Handle success, e.g., show a success message to the user

} catch (error) {

console.error('Error submitting form:', error);

// Handle error, e.g., show an error message to the user

}

};

return (

<div>

<form onSubmit={handleSubmit}>

{/\* Add input fields, radio buttons, checkboxes, dropdown, and file input for the form \*/}

{/\* Use handleInputChange, handleCheckboxChange, and handleFileChange to update the form data \*/}

{/\* Add a submit button to trigger the form submission \*/}

</form>

{/\* Display the form data in a table format \*/}

{/\* Add a button in each row to download the resume \*/}

</div>

);

}

export default App;

Certainly! Here's an example of how you might structure the code inside the form element and for the table display using React:

// App.js

import React, { useState } from 'react';

import axios from 'axios';

function App() {

const [formData, setFormData] = useState({

name: '',

dob: '',

gender: '',

hobbies: [],

country: '',

state: '',

address: '',

resume: null

});

const handleInputChange = (e) => {

const { name, value } = e.target;

setFormData({ ...formData, [name]: value });

};

const handleCheckboxChange = (e) => {

const { value, checked } = e.target;

let updatedHobbies = [...formData.hobbies];

if (checked) {

updatedHobbies.push(value);

} else {

updatedHobbies = updatedHobbies.filter((hobby) => hobby !== value);

}

setFormData({ ...formData, hobbies: updatedHobbies });

};

const handleFileChange = (e) => {

setFormData({ ...formData, resume: e.target.files[0] });

};

const handleSubmit = async (e) => {

e.preventDefault();

try {

const form = new FormData();

form.append('name', formData.name);

form.append('dob', formData.dob);

form.append('gender', formData.gender);

formData.hobbies.forEach((hobby) => form.append('hobbies', hobby));

form.append('country', formData.country);

form.append('state', formData.state);

form.append('address', formData.address);

form.append('resume', formData.resume);

const response = await axios.post('/submit-form', form, {

headers: {

'Content-Type': 'multipart/form-data'

}

});

console.log('Form submitted:', response.data);

// Handle success, e.g., show a success message to the user

} catch (error) {

console.error('Error submitting form:', error);

// Handle error, e.g., show an error message to the user

}

};

return (

<div>

<form onSubmit={handleSubmit}>

<div>

<label>Name:</label>

<input type="text" name="name" value={formData.name} onChange={handleInputChange} />

</div>

<div>

<label>Date of Birth:</label>

<input type="date" name="dob" value={formData.dob} onChange={handleInputChange} />

</div>

<div>

<label>Gender:</label>

<label>

<input type="radio" name="gender" value="male" checked={formData.gender === 'male'} onChange={handleInputChange} />

Male

</label>

<label>

<input type="radio" name="gender" value="female" checked={formData.gender === 'female'} onChange={handleInputChange} />

Female

</label>

</div>

<div>

<label>Hobbies:</label>

<label>

<input type="checkbox" name="hobbies" value="reading" checked={formData.hobbies.includes('reading')} onChange={handleCheckboxChange} />

Reading

</label>

<label>

<input type="checkbox" name="hobbies" value="sports" checked={formData.hobbies.includes('sports')} onChange={handleCheckboxChange} />

Sports

</label>

<label>

<input type="checkbox" name="hobbies" value="music" checked={formData.hobbies.includes('music')} onChange={handleCheckboxChange} />

Music

</label>

</div>

<div>

<label>Country:</label>

<select name="country" value={formData.country} onChange={handleInputChange}>

<option value="usa">USA</option>

<option value="canada">Canada</option>

<option value="uk">UK</option>

</select>

</div>

<div>

<label>State:</label>

<select name="state" value={formData.state} onChange={handleInputChange}>

<option value="ny">New York</option>

<option value="ca">California</option>

<option value="tx">Texas</option>

</select>

</div>

<div>

<label>Address:</label>

<textarea name="address" value={formData.address} onChange={handleInputChange}></textarea>

</div>

<div>

<label>Upload Resume (docx only):</label>

<input type="file" accept=".docx" name="resume" onChange={handleFileChange} />

</div>

<button type="submit">Submit</button>

</form>

{/\* Display the form data in a table format \*/}

<table>

<thead>

<tr>

<th>Name</th>

<th>Date of Birth</th>

<th>Gender</th>

<th>Hobbies</th>

<th>Country</th>

<th>State</th>

<th>Address</th>

<th>Resume</th>

</tr>

</thead>

<tbody>

{/\* Map through the form data and display each entry in a table row \*/}

{/\* Add a button in each row to download the resume \*/}

</tbody>

</table>

</div>

);

}

export default App;

certainly! Below is an example of how you might create the table element and retrieve the form data to display in the table using React:

// App.js

import React, { useState } from 'react';

import axios from 'axios';

function App() {

const [formData, setFormData] = useState({

name: '',

dob: '',

gender: '',

hobbies: [],

country: '',

state: '',

address: '',

resume: null

});

// ... (handleInputChange, handleCheckboxChange, handleFileChange, and handleSubmit functions)

// Assume formSubmissionData is an array of form data objects retrieved from the backend

const formSubmissionData = [

{

name: 'John Doe',

dob: '1990-05-15',

gender: 'Male',

hobbies: ['Reading', 'Sports'],

country: 'USA',

state: 'NY',

address: '123 Main St, New York',

resume: 'resume1.docx'

},

{

name: 'Jane Smith',

dob: '1988-09-20',

gender: 'Female',

hobbies: ['Music', 'Sports'],

country: 'Canada',

state: 'ON',

address: '456 Oak St, Toronto',

resume: 'resume2.docx'

}

// Add more form data objects as needed

];

return (

<div>

{/\* Form element code here \*/}

<table>

<thead>

<tr>

<th>Name</th>

<th>Date of Birth</th>

<th>Gender</th>

<th>Hobbies</th>

<th>Country</th>

<th>State</th>

<th>Address</th>

<th>Resume</th>

</tr>

</thead>

<tbody>

{formSubmissionData.map((data, index) => (

<tr key={index}>

<td>{data.name}</td>

<td>{data.dob}</td>

<td>{data.gender}</td>

<td>{data.hobbies.join(', ')}</td>

<td>{data.country}</td>

<td>{data.state}</td>

<td>{data.address}</td>

<td>

<a href={`/download-resume/${index}`}>Download</a>

</td>

</tr>

))}

</tbody>

</table>

</div>

);

}

export default App;

Certainly! Below is an example of how you might implement the backend code to handle resume downloads using Node.js and Express:

// server.js

const express = require('express');

const app = express();

const path = require('path');

const fs = require('fs');

// ... (other imports and middleware setup)

// Serve resume file for download

app.get('/download-resume/:id', (req, res) => {

const { id } = req.params;

// Assume formSubmissionData is an array of form data objects with resume file paths

const formSubmissionData = [

{

id: '1',

resumePath: 'path/to/resume1.docx'

},

{

id: '2',

resumePath: 'path/to/resume2.docx'

}

// Add more form data objects as needed

];

const formData = formSubmissionData.find((data) => data.id === id);

if (!formData) {

return res.status(404).send('Form data not found');

}

const resumePath = formData.resumePath;

// Check if the file exists

if (fs.existsSync(resumePath)) {

// Set the appropriate content type for .docx files

res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document');

res.download(resumePath, path.basename(resumePath));

} else {

res.status(404).send('Resume file not found');

}

});

// ... (other routes and server setup)

// Start the server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});