

# Lab7 File Upload

## Learning Outcomes

- Add Post method to upload file at server side
- Update tutorial model to include image file
- Call API to upload image at client side
- Add tutorial with image file
- Update tutorial with image file

POST uploadFile

No Environment

[HTTP](#) learning / uploadFile
 

Save

POST

{{baseUrl}}/file/upload

Send

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	file	pexels-somben-chea-1289363.jpg			
	Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK 10 ms 296 B

Save as Example

Pretty

Raw

Preview

Visualize

JSON

```

1  {
2    "filename": "KKm91brk9L.jpg"
3  }

```

## Add Tutorial

Title

My favorite color

Description

I like blue ❤️

UPLOAD IMAGE



ADD

## Tutorials

Search



ADD



### My favorite color

Li Haiyun

19 Jul 2023 15:45

I like blue ❤️



### My Story

Wong Wei Xuan

19 Jul 2023 14:24

Your Song

## Edit Tutorial

Title

My favorite color

Description

I like blue 

UPLOAD IMAGE



UPDATE

DELETE

---

## Activity 1: Add Post method to upload file at server side

1. Use VS Code to open the folder **learning-app**.
2. Under the folder server, add a new folder “**public**”
3. Under the folder public, add a new folder “**uploads**”
4. Open file server/index.js, add code as below:

```
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
app.use(express.static('public'));
```

The method `express.urlencoded()` allows the request to use form data. We will use it for file upload.

The method `express.static()` sets the static folder, which can be accessed by the public. We will upload the files in the folder `\public\uploads`.

5. Open a new terminal and change directory to the **server** folder.
6. Run command to install package multer:  
**npm i multer**  
Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.
7. Run command to install package nanoid:  
**npm i nanoid@3**  
Nanoid is a tiny, secure, URL-friendly, unique string ID generator for JavaScript.  
Note: The newer version 4 or 5 only supports ES6 syntax “import”, not ES5 syntax “require”.  
We need to use version 3 for ES5.

8. Under the folder middlewares, add a new file upload.js:

```
const multer = require('multer');
const { nanoid } = require('nanoid');
const path = require('path');

const storage = multer.diskStorage({
  destination: (req, file, callback) => {
    callback(null, './public/uploads/');
  },
  filename: (req, file, callback) => {
    callback(null, nanoid(10) + path.extname(file.originalname));
  }
});

const upload = multer({
  storage: storage,
  limits: { fileSize: 1024 * 1024 }
}).single('file'); // file input name

module.exports = { upload };
```

The middleware for file upload is created:

- Set the destination to \public\uploads folder
- Replace the original file name to a new unique id
- Limit the file size to 1MB
- The file input name in the form must be 'file'

9. Under the folder route, add a new file file.js:

```
const express = require('express');
const router = express.Router();
const { validateToken } = require('../middlewares/auth');
const { upload } = require('../middlewares/upload');

router.post('/upload', validateToken, (req, res) => {
  upload(req, res, (err) => {
    if (err) {
      res.status(400).json(err);
    }
    else if (req.file == undefined) {
      res.status(400).json({ message: "No file uploaded" });
    }
    else {
      res.json({ filename: req.file.filename });
    }
  })
});

module.exports = router;
```

The post method is added to upload file:

- **validateToken** middleware ensures the request is from a valid user
- **upload** function processes the request using multer and sets the uploaded file to the request

10. In server/index.js, add the file route:

```
const userRoute = require('./routes/user');
app.use("/user", userRoute);
const fileRoute = require('./routes/file');
app.use("/file", fileRoute);
```

11. Go to the free photo website Pexels to download images for testing. You can choose the size when you click on the Free download button.

<https://www.pexels.com/search/wallpaper/?orientation=landscape>

- Download some images with large size, usually the size is smaller than 1MB
- Download one image with original size, usually the size is larger than 1MB

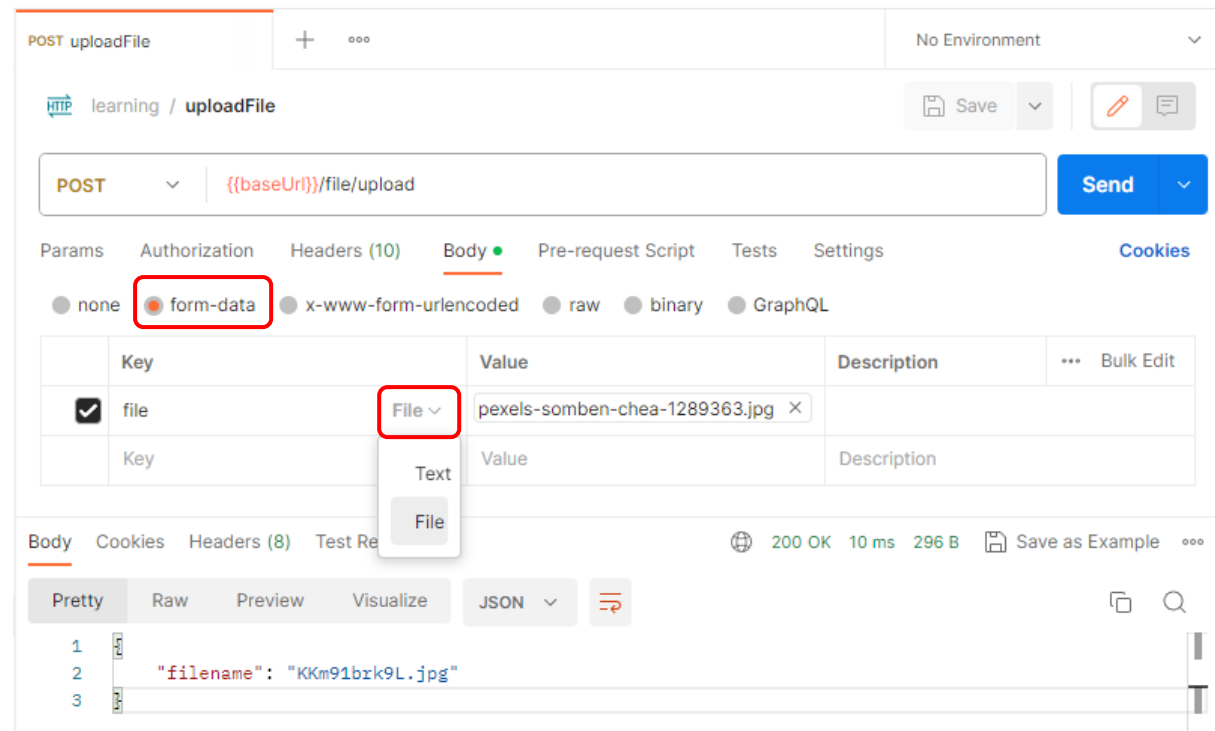
## 12. Use Postman to test upload file.

Method: POST

URL: {{baseUrl}}/file/upload

Body: (select form-data)

- file (File type): select one image file from your computer (size smaller than 1MB)



The response body is the new filename.

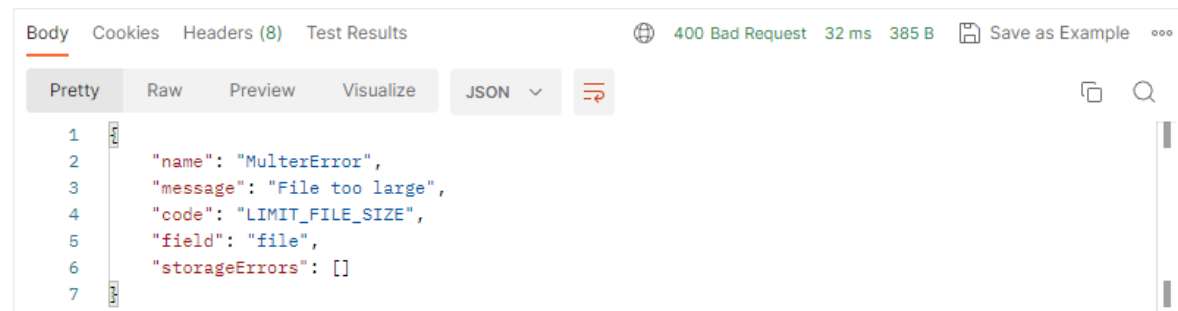
## 13. In VS Code, check the new file is uploaded to the folder server/public/uploads.

## 14. The url of the file is http://localhost:<port\_number>/uploads/<file\_name>

e.g. <http://localhost:3001/uploads/9tyejHUulo.jpg>

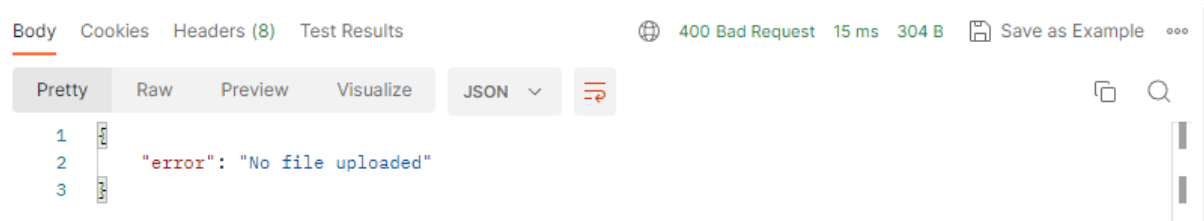
You can test the url of your newly uploaded file in browser.

## 15. Select one image file with size larger than 1MB and test again.



The response status is 400 Bad Request and the multer error is returned in the response body.

16. Uncheck the file input in the request body and test again.



The response status is 400 Bad Request and the error message is returned in the response body.

## Activity 2: Update tutorial model to include image file

1. In VS Code, open models/Tutorial.jsx, add new attribute **imageFile**:

```

description: {
  type: DataTypes.TEXT,
  allowNull: false
},
imageFile: {
  type: DataTypes.STRING(20)
}

```

2. Start the server.
3. Open MySQL Workbench. Observe that in the table tutorials, the new column imageFile is added. The default value is null.

### Information

**Table: tutorials**

**Columns:**

<b>id</b>	int AI PK
title	varchar(100)
description	text
createdAt	datetime
updatedAt	datetime
<b>userId</b>	int
imageFile	varchar(20)



## Activity 3: Call API to upload image at client side

17. In VS Code, under the folder client, open src/pages/AddTutorial.jsx, update code to use grid layout:

```
import { Box, Typography, TextField, Button, Grid } from '@mui/material';
...

<Box component="form" onSubmit={formik.handleSubmit}>
  <Grid container spacing={2}>
    <Grid item xs={12} md={6} lg={8}>
      <TextField
        ...
      />
    <Grid item xs={12} md={6} lg={8}>
      <TextField
        ...
      />
    <Grid item xs={12} md={6} lg={4}>
      <Button variant="contained" type="submit">
        Add
      </Button>
    </Grid>
  </Grid>
</Box>
</Box>
```

In the grid container, the first item includes the 2 text fields title and description. We will add the file input to the 2<sup>nd</sup> item.

18. Add button for the file input:

```
<Grid item xs={12} md={6} lg={4}>
  <Box sx={{ textAlign: 'center', mt: 2 }} >
    <Button variant="contained" component="label">
      Upload Image
      <input hidden accept="image/*" multiple type="file" />
    </Button>
  </Box>
</Grid>
```

The file input accepts image types only.

19. Split the terminal. Change directory to the **client** folder and start the client.

20. Go to the browser, navigate to the add tutorial page. Observe that the upload image button is displayed.

### Add Tutorial

21. In VS Code, file AddTutorial.jsx, add on change handler for the file input:

```
const onFileChange = (e) => {
  let file = e.target.files[0];
  if (file) {
    let formData = new FormData();
    formData.append('file', file);
    http.post('/file/upload', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })
      .then((res) => {
        console.log(res.data);
      })
      .catch(function (error) {
        console.log(error.response);
      });
  }
};
...
<Button variant="contained" component="label">
  Upload Image
  <input hidden accept="image/*" multiple type="file"
    onChange={onFileChange} />
</Button>
```

When the file is selected, call api file upload using form data.

22. Add code to check file size:

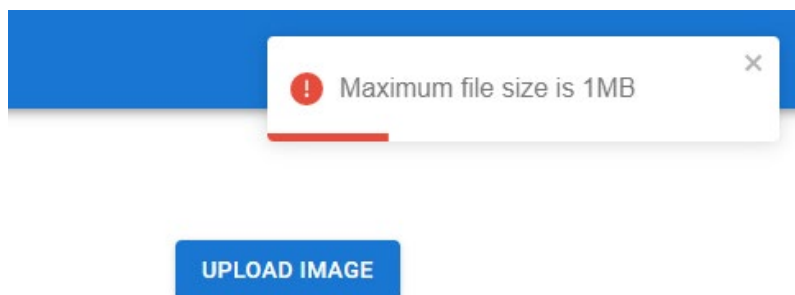
```
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

...
const onFileChange = (e) => {
  let file = e.target.files[0];
  if (file) {
    if (file.size > 1024 * 1024) {
      toast.error('Maximum file size is 1MB');
      return;
    }
    ...
  }
};

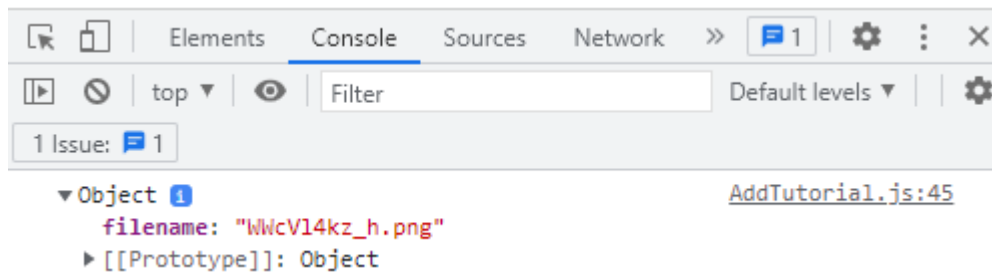
...
return (
  <Box>
    ...
    <ToastContainer />
  </Box>
);
```

If file size is larger than 1MB, show the error toast.

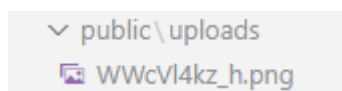
23. Go to the browser, upload the image with size larger than 1MB. Observe that the toast is displayed.



24. Upload an image with size smaller than 1MB. Open the browser console, observe that the api response is logged.



25. Go to VS Code explorer. Observer that the file is uploaded in the folder server/public/uploads.



26. Open file ".env" under client, add new variable:

```
VITE_API_BASE_URL = "http://localhost:3001"
VITE_FILE_BASE_URL = "http://localhost:3001/uploads/"
```

The file base URL will be used to form image URL.

27. Open file "index.css", add code for css class *aspect-ratio-container*:

```
.aspect-ratio-container {
  position: relative;
  padding-bottom: 56.25%;
  /* 16:9 */
}

.aspect-ratio-container img {
  position: absolute;
  width: 100%;
  height: 100%;
  left: 0;
  top: 0;
  object-fit: cover;
}
```

This class with be used for image container to keep aspect ratio 16:9.

28. In file AddTutorial.jsx, add state for image file:

```
import React, { useState } from 'react';
...
const navigate = useNavigate();
const [imageFile, setImageFile] = useState(null);
```

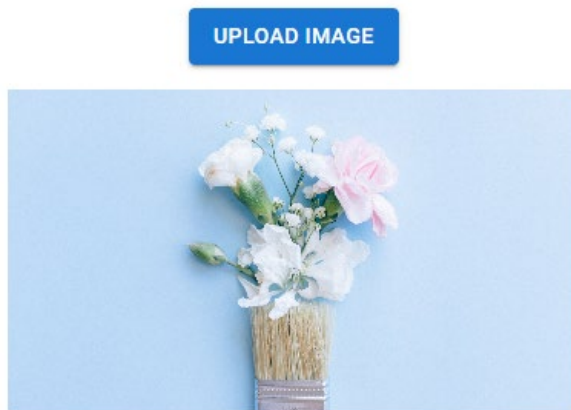
29. After getting the upload response, set the value of image file state.

```
    http.post('/file/upload', formData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })
    .then((res) => {
      setImageFile(res.data.filename);
    })
    .catch(function (error) {
      console.log(error.response);
    });
```

30. Use the image file state to display image preview, below the upload image button.

```
<Button variant="contained" component="label">
  Upload Image
  <input hidden accept="image/*" multiple type="file"
    onChange={onFileChange} />
</Button>
{
  imageFile && (
    <Box className="aspect-ratio-container" sx={{ mt: 2 }}>
      <img alt="tutorial"
        src={`${import.meta.env.VITE_FILE_BASE_URL}${imageFile}`} />
      </img>
    </Box>
  )
}
```

31. Go to the browser. Upload an image, observe that the uploaded image is displayed.



#### Activity 4: Add tutorial with image file

1. In VS Code, file AddTutorial.jsx, add code when submit the form:

```
onSubmit: (data) => {
  if (imageFile) {
    data.imageFile = imageFile;
  }
  ...
}
```

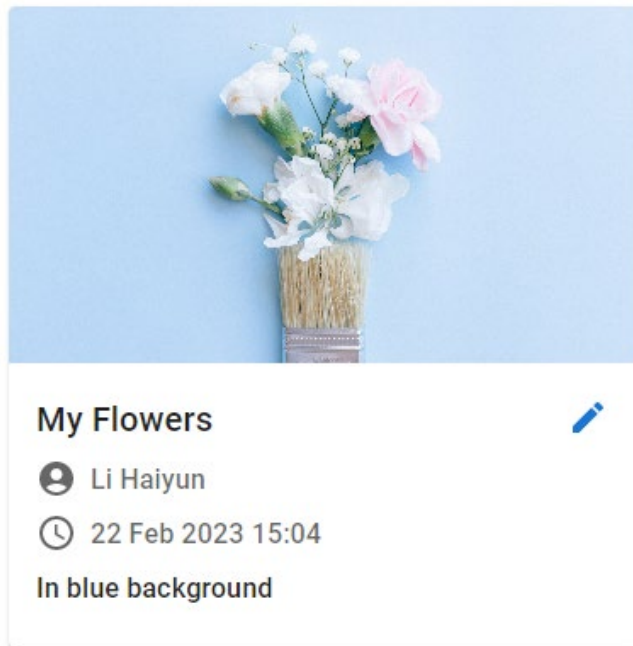
If the image file is uploaded, add the image file to the data, which is sent in the request body.

2. Open file Tutorials.jsx, add code to display image:

```
<Card>
  {
    tutorial.imageFile && (
      <Box className="aspect-ratio-container">
        <img alt="tutorial"
          src={` ${import.meta.env.VITE_FILE_BASE_URL}${tutorial.imageFile}`} >
        </img>
      </Box>
    )
  }
  <CardContent>
    ...
  </CardContent>
</Card>
```

If the tutorial image file is available, display the image at the top of the card.

3. Go to the browser, add a tutorial with an image. Observe that in the tutorials page, the image is displayed for the new tutorial.



## Activity 5: Update tutorial with image file

1. In VS Code, refer to add tutorial, try yourself to upload image in edit tutorial page:
  - Add upload image button with grid layout
  - Add on change handler for file input
  - Add file size validation with toast
  - Add state for image file
  - Set image file state after getting upload response
  - Use image file state to display the uploaded image
2. In EditTutorial.jsx, set value of image file state after getting the tutorial object from api:

```
http.get(`/tutorial/${id}`).then((res) => {
  setTutorial(res.data);
  setImageFile(res.data.imageFile);
});
```

3. Refer to add tutorial, try yourself to include image file when updating tutorial.



4. Go to the browser. Test update image.
  - In the tutorial list, edit one tutorial, the existing image is displayed in the edit tutorial page.
  - Update tutorial with a new image, the tutorial with the updated image is displayed in the tutorial list.

Learning
Tutorials
Li Haiyun
LOGOUT

### Edit Tutorial


Title  
My Story

Description  
Your Song...

UPDATE

DELETE


UPLOAD IMAGE




Learning
Tutorials
Li Haiyun
LOGOUT

### Tutorials

Search
ADD



My Flowers
Li Haiyun
22 Feb 2023 15:04
In blue background



My Story
Li Haiyun
20 Feb 2023 17:33
Your Song...