BINAR ACADEMY

Media Handling

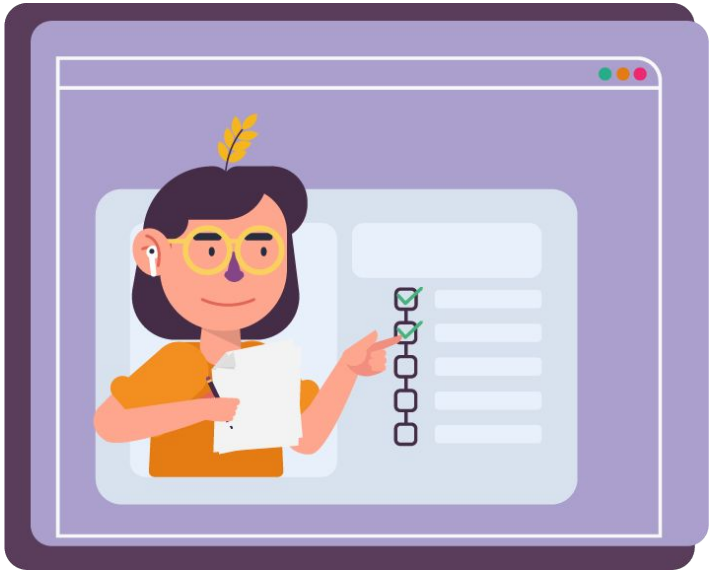
Gold - Chapter 8 - Topic 3

Selamat datang di Chapter 8 Topik 3 online course Fullstack Web dari Binar Academy!

Nahhh... di topik ketiga ini kita akan belajar bagaimana cara **mengelola media pada aplikasi web kita**. Banyak sekali media yang bisa kelola seperti video, image, dan pdf. Cusss mari kita coba ~

Detailnya, kita bakal bahas hal-hal berikut ini:

- Memahami cara melakukan upload dan penyimpanan file pada website dari back-end
- Memahami cara melakukan upload dan penyimpanan file pada website dari front-end

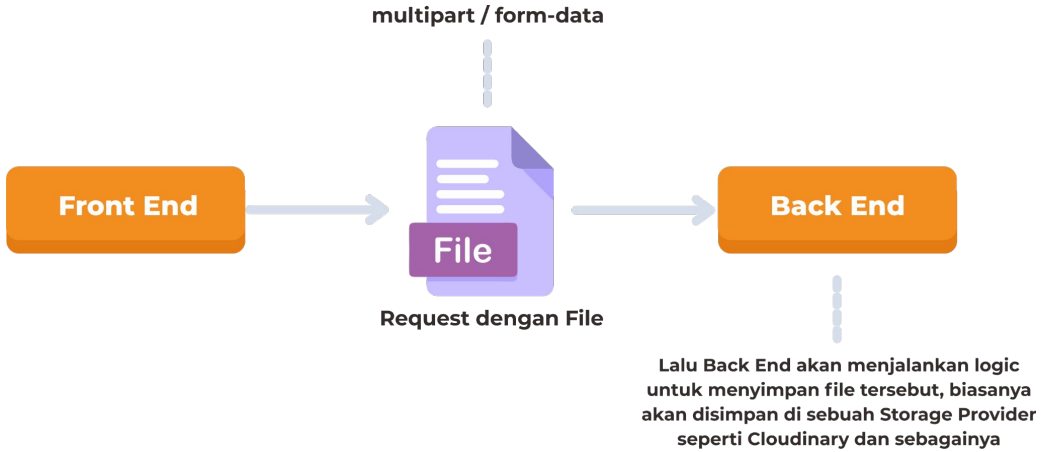


Nah, mungkin kalian penasaran kan gimana cara website nyimpen file yang kalian upload, kayak Profile Picture kalian di Whatsapp atau Facebook.

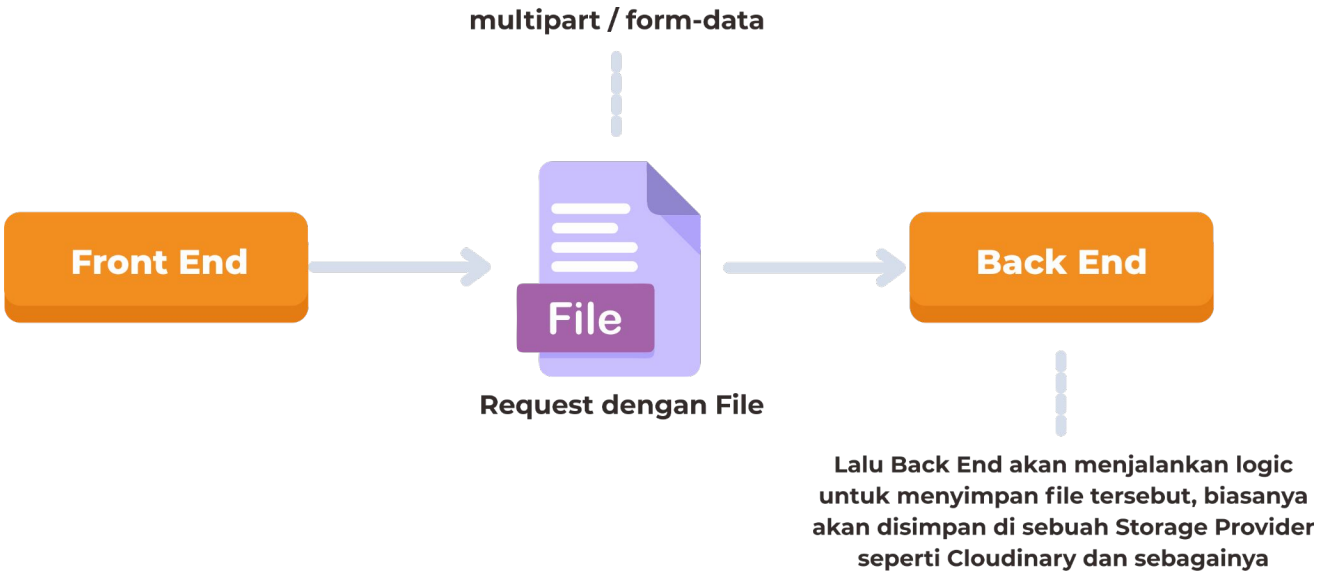
Untuk memahami gimana cara kerjanya, kita akan bahas dari 2 sisi, yaitu backend dan frontend. Pembahasan pertama akan kita mulai bahas mengenai **bagaimana back-end ngehandle ini**, yuk kita cari tau!

Cara handle file upload di Back-end

Seperti yang kalian tahu, kalau orang-orang ingin menyimpan data di sebuah website pasti lewat back-end. Tentu saja back-end juga ga bakal lepas dari penyimpanan image, video dan sebagainya.



Nah untuk melakukan handle file upload di sisi back-end, biasanya back-end hanya akan meminta request dengan content-type **multipart/form-data**. Dimana dalam request tersebut terdapat binary dari file yang ingin kita upload.



Melakukan middleware upload

Contoh di samping adalah sebuah contoh definisi multer yang digunakan untuk mendeklarasikan bagaimana cara kita meng-handle dan menyimpan file upload.

Nah, code di samping menggunakan library bernama **multer**, yang berfungsi untuk mengurai request yang tipe kontennya **multipart/form-data**.

```
const multer = require("multer");
const path = require("path");

// Menentukan tempat penyimpanan file
const publicDirectory = path.join(__dirname, "public");
const uploadDirectory = path.join(publicDirectory, "uploads")

// Mendefinisikan gimana cara nyimpen file-nya
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, uploadDirectory)
  },

  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
    cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));
  }
})

// Membuat upload middleware
module.exports = multer({ storage })
```

Memasangkan middleware ke Endpoint

Nah, middleware upload ini biasanya dipasang ke endpoint yang membutuhkan file upload. Biasanya itu endpoint buat upload file hanya ada satu. Tapi pada kasus ini, kita akan coba membuat endpoint untuk upload PP dari sebuah aplikasi.

Sampai tahap ini, kamu udah berhasil menghandle file upload request dan menyimpannya dengan aman.

```
const express = require("express");
const upload = require("./upload")
const app = express();

app.use(express.static("public"));
app.use(express.json());

app.put("/api/v1/profiles/:id/picture", upload.single("picture"), (req, res) => {
  const url = `/uploads/${req.file.filename}`;
  // TODO: Simpen URL-nya ke tabel profiles, atau users
  res.status(200).json({ message: "Foto berhasil di-upload, silahkan cek URL", url });
})

app.listen(8080);
```

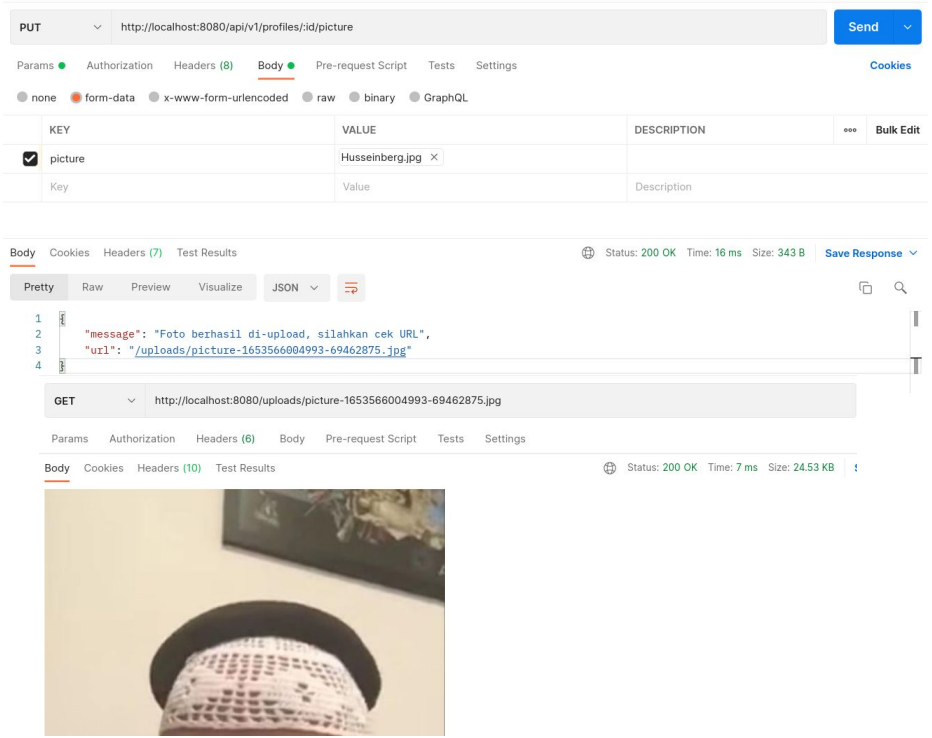
Kita coba di Postman Yuk!

Nah untuk nyobain fitur upload file di back-end kamu bisa pake Postman. Jangan lupa, request yang kamu kirim harus berupa **multipart/form-data** khusus untuk upload file doang.

Media Handling : Back-End

Kalo kamu lihat dari gambar di samping, ada URL di dalam response. Nah si URL itulah yang nantinya ketika dibuka, akan memberi response file yang sudah kamu upload.

Kode tadi bisa kamu download [disini](#).



Eits, tapi ada kekurangannya lho kalo kamu nyimpen file di server-mu sendiri!

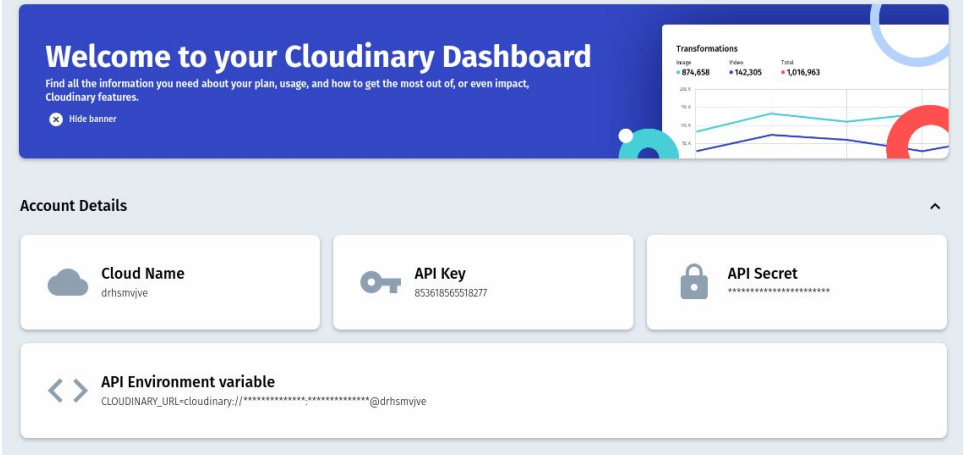
Kalo kamu nyimpen file di server-mu, lama-lama bakalan penuh, dan untuk ngehandle delivery file ke user juga belum teroptimalisasi seperti storage facility di luar sana kayak Google Cloud Storage dan lain-lain.

Maka dari itu sangat dianjurkan untuk memakai Cloud Storage yang sudah mengimplementasikan CDN, sebagai contoh **cloudinary**.

Implementasi Cloudinary

Untuk mengimplementasikan Cloudinary, kamu perlu mendaftar dulu, nih kamu bisa buka link ini.

Setelah kamu berhasil mendaftar di Cloudinary, kamu akan mendapatkan yang namanya **API Key** dan **API Secret**, yang mana dua hal ini bisa dipakai untuk menghubungkan back-end-mu dengan Cloudinary.



```
// Require the Cloudinary library
const cloudinary = require('cloudinary').v2

cloudinary.config({
  cloud_name: 'drhsmvjve', // TODO: Ganti dengan cloudname-mu
  api_key: '853618565518277', // TODO: Ganti dengan API Key-mu
  api_secret: 'RpCRK0as_DNhSPRbIKJLcCzMRjg', // TODO: Ganti dengan API Secret-mu
  secure: true
});

module.exports = cloudinary;
```

Implementasi Cloudinary di Express

Nah untuk mengimplementasikan Cloudinary di express kamu perlu library bernama **cloudinary**. Detail implementasi bisa kalian lihat disini, tapi overall di-summarize sama kode berikut.

Pertama kamu perlu setup cloudinary-nya dulu, kek kode di samping yah!

Setup Multer agar menyimpan file yang di-upload ke Memory dulu!

Untuk upload file ke Cloudinary, kamu perlu mengirim file tersebut dalam bentuk **data URI**. Nah maka dari itu, kamu perlu akses ke **buffer** dari file tersebut dan mengkonversi-nya jadi **base64**. Untuk melakukannya cukup sederhana kan! Tinggal pake aja multer dengan **MemoryStorage**.

```
const multer = require("multer");
const path = require("path");

// Mendefinisikan gimana cara nyimpen file-nya
const storage = multer.memoryStorage();

// Membuat upload middleware
module.exports = multer({ storage })
```

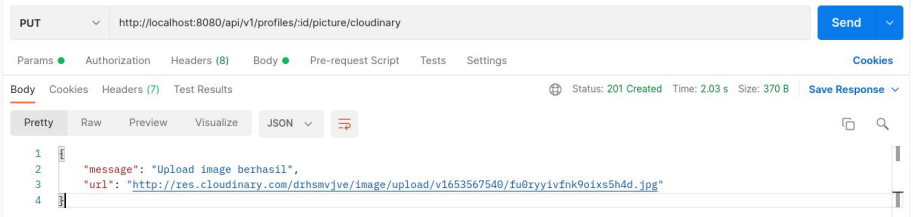
```
app.put("/api/v1/profiles/:id/picture/cloudinary",
uploadOnMemory.single("picture"), (req, res) => {
  const fileBase64 = req.file.buffer.toString("base64")
  const file = `data:${req.file.mimetype};base64,${fileBase64}`

  cloudinary.uploader.upload(file, function(err, result) {
    if (!!err) {
      console.log(err)
      return res.status(400).json({
        message: "Gagal upload file!"
      })
    }

    res.status(201).json({
      message: "Upload image berhasil",
      url: result.url,
    })
  })
})
```

Memasangkan ke endpoint-nya

Nah, setelah kamu punya multer dan cloudinary yang sudah di-setup, kamu bisa langsung pakai dua benda itu di endpoint yang mau kamu pakai untuk upload file.

Kita coba di Postman yuk!

Cara-nya sama kok, hanya saja response-nya akan sedikit berbeda. Nah repo tadi, bisa kamu liat langsung disini.



Oke! Sekarang kamu udah tahu kan gimana cara implementasinya di Back-end?

Kita coba cari tahu yuk, cara implementasinya di Front-end itu seperti apa?

Implementasinya simpel gaes!

Kamu cuma perlu bikin **input** dan kasih **type**-nya jadi **file**. Nah nantinya file itu akan kamu masukin sebagai atribut dari object **FormData**.

Coba deh cek kode gambar di samping!

```
import { useState, Fragment } from "react";
import axios from "axios";

function App() {
  const [file, setFile] = useState(null);
  const [uploadedFileURL, setUploadedFileURL] = useState(null);

  async function handleSubmit(e) {}

  return (
    <Fragment>
      {uploadedFileURL && <img src={uploadedFileURL} alt="Uploaded Image URL" />}

      <form onSubmit={handleSubmit}>
        <input type="file" onChange={(e) => setFile(e.target.files[0])}/>
        <input type="submit" value="Upload" />
      </form>
    </Fragment>
  );
}

export default App;
```

```
async function handleSubmit(e) {
  e.preventDefault();

  const form = new FormData();

  form.append("picture", file);

  try {
    const response = await
axios.put("http://localhost:8080/api/v1/profiles/1/picture",
form, {
      headers: {
        "Content-Type": "multipart/form-data"
      },
    });

    // Kalo di upload langsung di-server
    setUploadedFileURL("http://localhost:8080/" +
response.data.url);
  }

  catch(err) {
    console.log(err);
    console.log(err?.responses?.data);
  }
}
```

Implementasi submit form

Nah ketika user melakukan submit form-nya, kamu cuma perlu lakuin HTTP Request ke server dengan content type **multipart/form-data** aja kok, abis itu kamu bebas mau ngelakuin apa.

Outputnya gimana ya?

Nah, setelah kamu selesai implemen kode tadi, output dari kode-mu bakalan kayak gini. Jangan lupa nyalain server back-end nya yaaa!

Kode dari implementasi diatas bisa kamu liat disini yak!

Referensi dan bacaan lebih lanjut~

- https://www.youtube.com/watch?v=9Qzmri1WaaE
- https://www.npmjs.com/package/multer
- https://www.npmjs.com/package/cloudinary#installation
- https://developer.mozilla.org/en-US/docs/Web/API/FormData/FormData

Nah, selesai sudah pembahasan kita di Chapter 8 Topic 3 ini.

Selanjutnya, kita bakal bahas tentang gimana caranya melakukan unit testing dan TDD pada aplikasi web kita. Yukkk! Gas! Ngeng~

Terima Kasih!

Next Topic

loading...

BINAR