

Backend Media Handling

Gold - Chapter 6 - Topic 1

Selamat datang di **Chapter 6 Topic 1**
online course Back End Javascript dari
Binar Academy!



Di topic sebelumnya kita udah belajar tentang Authentication.

Di topic kedua ini, kita bakal bahas materi **Media Handling**. Mulai dari bikin feature buat video upload, image upload, document upload, hingga melakukan generate dan download QR code.

Cuss kita kepoin materinya~



Detailnya, kita bakal bahas hal-hal berikut ini:

- Handling Image Uploads
- The Multer Module
- Uploading and Managing Files (Filetype: e.g., PDF)
- Generating and Downloading QR Codes
- Integrating Cloud Storage Services (Imagekit.io)



Kamu sadar, nggak?

Di abad ke-21 ini, tampilan website cenderung lebih fokus pada media ketimbang tulisan, lho.

Nggak cuma supaya kelihatan lebih menarik aja, tapi sekaligus juga bisa dibuat fungsional.



Tapi, semakin banyaknya jumlah media yang bertambah setiap hari, nggak jarang juga nih bikin kita kesulitan buat mengelolanya.

So, yuk kita belajar cara handling media lebih dalam di chapter ini!



Eits, ada yang perlu kamu pahami dulu nih sebelum bikin fitur buat picture, video, dan document upload, yaitu **Handling upload image**.

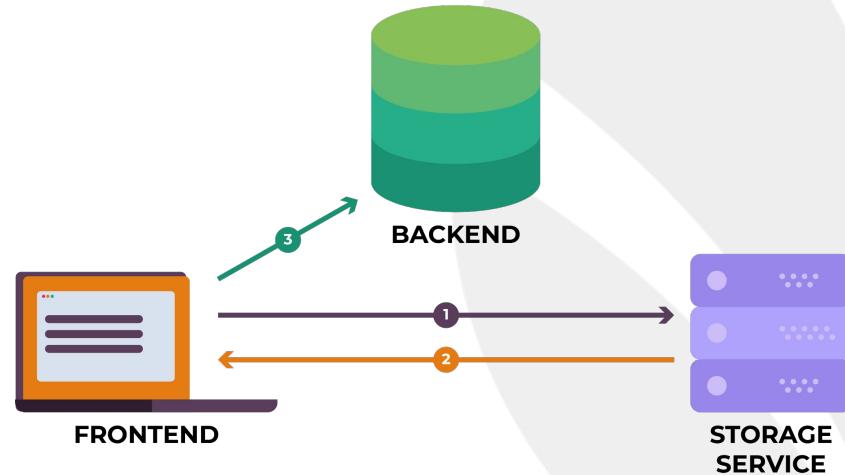
Langsung kita cari tahu, yuk!



Flow Upload Image yang pertama~

Coba perhatikan langkah-langkahnya ya, bestie~

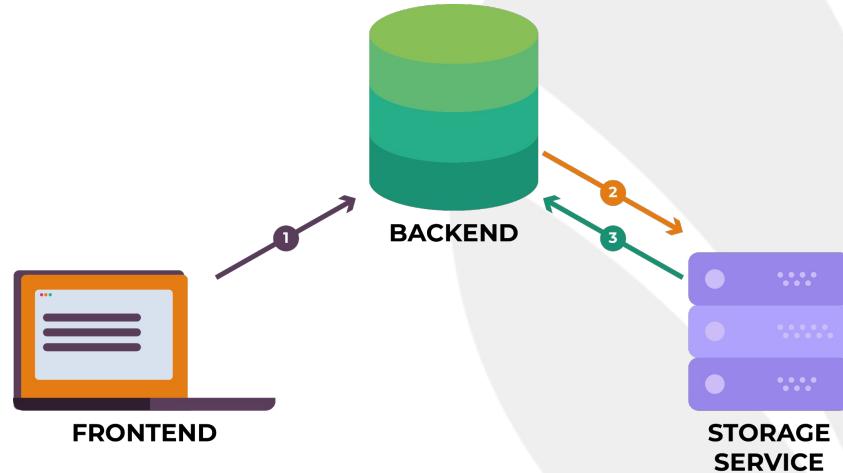
1. Frontend melakukan Upload ke storage service (AWS/cloudinary) yang nanti bakal mendapatkan sebuah data URL dari imagennya.
2. URL yang didapat, ditambahkan ke dalam request body upload image.
3. Backend bakal menyimpan data URL image dari frontend ke database.



Flow Upload Image kedua~

Sedikit berbeda dari cara yang pertama, sekarang kita coba simak cara yang kedua ini, ya!

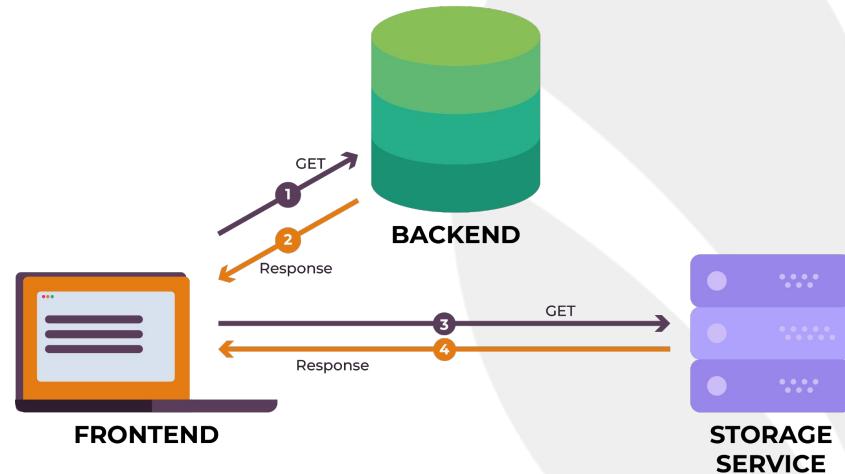
1. Frontend melakukan upload ke backend (API).
2. Backend bakal meng-upload ke storage service (AWS/cloudinary) yang nanti bakal mendapatkan data URL dari imagennya.
3. Simpan data URL tersebut ke database.



Next! Flow GET user profile~

Jangan lupa sambil dilihat gambar disampingnya, yaa~

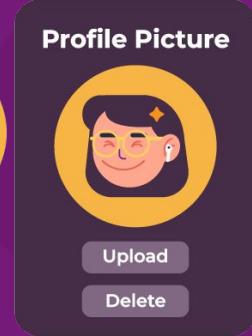
1. Frontend melakukan GET request /user/profile ke backend.
2. Backend bakal memberikan Response, yaitu mengambil data dari DB.
3. IMG URL dari response backend, bakalan GET request ke storage service.
4. Selanjutnya storage service bakal mengembalikan data imagnya.



Kita pasti sering banget gonta-ganti profile picture di sosmed kayak Facebook, Twitter, atau Instagram, kan?

Tapi pernah kepikiran nggak, gimana sih cara bikin fitur kayak gitu?

Salah satu caranya ternyata dengan menggunakan middleware kayak **Multer** lho, sob.



“Multer itu apa, ya?”

Multer adalah **middleware Node Js buat menangani multipart/form-data**, terutama yang dipakai buat mengunggah file, gengs.

Itu ditulis di atas busboy buat efisiensi maksimum.

Tapi perlu diingat ya, kalau Multer nggak bakal memproses formulir apa pun yang bukan multipart (multipart/form-data).



Tadi kita udah belajar teori-teorinya,
sekarang waktunya kita cari tahu tentang
**penggunaan Multer buat upload dan
manage files.**

Kamu udah penasaran banget, kan?

Kalau gitu, markibas alias mari kita
bahassss~



Berikut adalah contoh penggunaan dasar multer buat upload gambar!

Eits, nggak usah dibayangkan sendiri, gengs.

Langsung aja yuk kita coba caranya kayak di bawah ini!



```
<form action="/profile" method="post" enctype="multipart/form-data">
  <input type="file" name="avatar" />
</form>
```

Pertama, kita init project yang akan kita kerjakan 

```
▶ npm init -y

Wrote to /Users/tatangdev/example/media-handling/package.json:

{

  "name": "media-handling",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Lanjut, install package yang kita butuhkan.

```
> npm i dotenv express multer

added 82 packages, and audited 83 packages in 3s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Oke, kita mulai dari setup app.js

Simak baik-baik, ya! 😊

```
// app.js

require('dotenv').config();

const express = require('express');

const app = express();


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

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


const mediaRouter = require('./routes/media.routes');

app.use('/api/v1', mediaRouter);


const { PORT = 3000 } = process.env;

app.listen(PORT, () => console.log('listening on port', PORT));
```

Selanjutnya, kita setup middleware multer didalam file **libs/multer.js**.

```
// libs/multer.js

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

const filename = (req, file, callback) => {
    const fileName = Date.now() + path.extname(file.originalname);
    callback(null, fileName);
};

const generateStorage = (destination) => {
    return multer.diskStorage({
        destination: (req, file, callback) => {
            callback(null, destination);
        },
        filename

    });
};
```

Ini lanjutan **libs/multer.js**.

```
// libs/multer.js

module.exports = {

  image: multer({

    storage: generateStorage('../public/images'),

    fileFilter: (req, file, callback) => {

      const allowedMimeTypes = ['image/png', 'image/jpg', 'image/jpeg'];

      if (allowedMimeTypes.includes(file.mimetype)) {
        callback(null, true);
      } else {
        const err = new Error(`Only ${allowedMimeTypes.join(', ')} allowed to upload!`);
        callback(err, false);
      }
    },
    onError: (err, next) => {
      next(err);
    }
  })
};
```

Lanjut, kita buatkan handler upload image nya.

```
// controllers/media.controllers.js

module.exports = {

    storageImage: (req, res) => {
        const imageUrl = `${req.protocol}://${req.get('host')}/images/${req.file.filename}`;

        return res.status(200).json({
            status: true,
            message: 'success',
            data: {
                image_url: imageUrl
            }
        });
    }
};
```

Kalau udah, kita tambahkan routing untuk handler yang sudah kita buat. Selesai deh~

```
// routers/media.routes.js

const router = require('express').Router();
const storage = require('../libs/multer');

const { strogeImage } = require('../controllers/media.controllers');

router.post('/images', storage.image.single('image'), strogeImage);

module.exports = router;
```

**Kalau Image upload udah bisa,
lanjut ke video upload~**

Oke! Sekarang kita lanjut ke penggunaan Multer
buat Upload Video.

Caranya sama aja kok kaya upload gambar,
bedanya cuma validasinya aja.

Cek slide selanjutnya ya!



“Emang gimana sih cara upload video-nya?”

Tenang, tenang~

Caranya, kita bisa menambahkan validasi file khusus buat format/ file type video, gengs.

Biar makin paham, coba perhatikan gambar yang disamping, yaaa~

```
// libs/multer.js

video: multer({
    storage: generateStorage('./public/videos'),
    fileFilter: (req, file, callback) => {
        const allowedMimeTypes = ['video/mp4', 'video/x-msvideo', 'video/quicktime'];

        if (allowedMimeTypes.includes(file.mimetype)) {
            callback(null, true);
        } else {
            const err = new Error(`Only ${allowedMimeTypes.join(', ')} allowed to upload!`);
            callback(err, false);
        }
    },
    onError: (err, next) => {
        next(err);
    }
})
```

Jangan lupa tambahkan controller beserta dengan routes nya. Selesai deh. Mudah banget kan? 😎

```
// controllers/media.controllers.js

storageVideo: (req, res) => {
    const videoUrl = `${req.protocol}://${req.get('host')}/videos/${req.file.filename}`;

    return res.status(200).json({
        status: true,
        message: 'success',
        data: {
            video_url: videoUrl
        }
    });
}
```

```
// routers/media.routers.js

const router = require('express').Router();
const storage = require('../libs/multer');

const { strogeImage, storageVideo } = require(
    '../controllers/media.controllers'
);

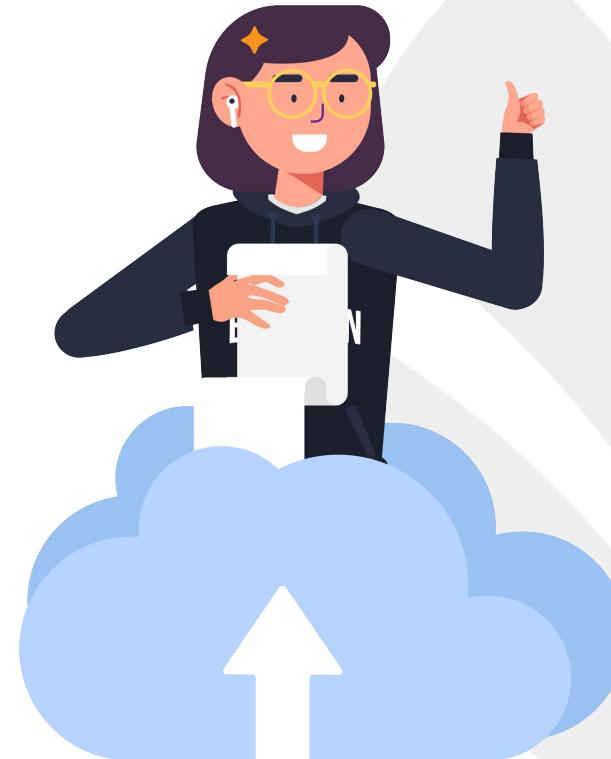
router.post('/images', storage.image.single('image'), strogeImage);
router.post('/videos', storage.image.single('video'), storageVideo);

module.exports = router;
```

Oke, selanjutnya document upload~

Secara garis besar, caranya sama seperti upload image ataupun video. Nah karena tipe dokumen berbeda-beda, kita bisa banget nambahin validasi.

Langsung aja cek toko sebelah, eh slide sebelah maksudnya 😅



Langsung aja! Begini cara buat upload document~

Jadi, kita bisa menambahkan validasi file khusus buat format atau file type document kayak pdf, gengs.

Boleh dilirik dulu nih gambar di samping biar makin paham 😊

```
// libs/multer.js

file: multer({
  storage: generateStorage('./public/files'),
  fileFilter: (req, file, callback) => {
    const allowedMimeTypes = ['application/pdf'];

    if (allowedMimeTypes.includes(file.mimetype)) {
      callback(null, true);
    } else {
      const err = new Error(`Only ${allowedMimeTypes.join(', ')} allowed to upload!`);
      callback(err, false);
    }
  },
  onError: (err, next) => {
    next(err);
  }
})
```

Jangan lupa tambahkan controller beserta dengan routes nya. Selesai deh~

```
// controllers/media.controllers.js

storageFile: (req, res) => {
    const fileUrl = `${req.protocol}://${req.get('host')}/files/${req.file.filename}`;

    return res.status(200).json({
        status: true,
        message: 'success',
        data: {
            file_url: fileUrl
        }
    });
}
```

```
// routers/media.routers.js

const router = require('express').Router();
const storage = require('../libs/multer');

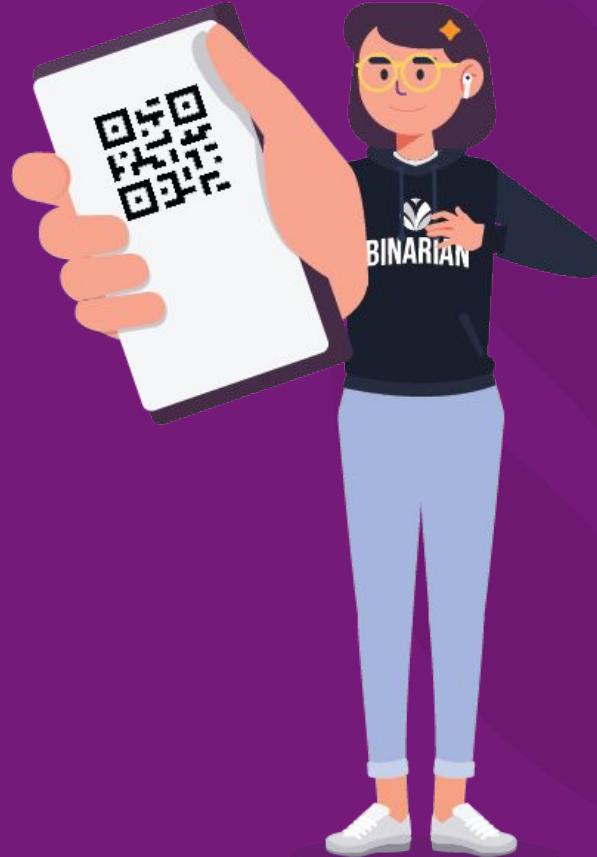
const { strogeImage, storageVideo, storageFile } = require(
    '../controllers/media.controllers'
);

router.post('/images', storage.image.single('image'), strogeImage);
router.post('/videos', storage.image.single('video'), storageVideo);
router.post('/files', storage.image.single('file'), storageFile);

module.exports = router;
```

Generate & Download QR Code

Sekarang kita udah tau gimana caranya untuk upload image, video hingga document. Sekarang kita lanjut dengan **Generate & Download QR Code**. Lesgooo~



Generate & Download QR Code

QR code dapat di-generate ke dalam format **.png**, **.svg**, **.eps** ataupun **.pdf**. Semua bisa kamu pilih sesuai kebutuhan.

Mode data QR code ini terdiri dari mode **numerik** dan **alfanumerik**. Apa sih bedanya? Kalau mode numerik itu datanya berupa angka saja (0-9). Sedangkan mode alfanumerik terdiri dari angka (0-9), huruf alfabet (A-Z, a-z) dan juga simbol (@#\$%*).



Yuk kita install Node QR Image!

Untuk melakukan instalasi Node QR Image cukup dengan ketik perintah berikut:

```
npm install node-qr-image
```



Cara penggunaannya gimana sih?

Berikut perintah untuk penggunaan Node QR Image

```
var qr = require('node-qrcode');

var qr_svg = qr.image('KODETIKET001', { type: 'png' });
qr_svg.pipe(require('fs').createWriteStream('kodetiket.png'));

var svg_string = qr.imageSync('KODETIKET001', { type: 'png' });
```

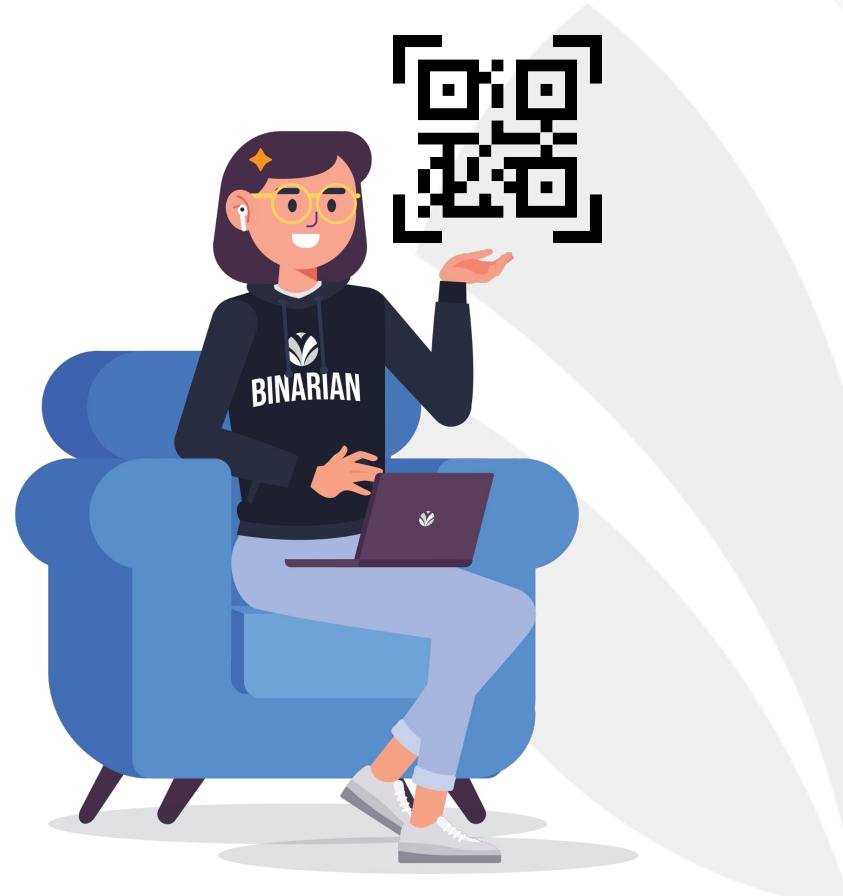
Apa aja sih metode yang dipakai untuk membuat QR Code?

1. `qr.image(text, [ec_level | options])`

Aliran yang dapat dibaca dengan data gambar.
Metode ini untuk mengubah teks menjadi QR
Code.

2. `qr.imageSync(text, [ec_level | options])`

Kalau ini untuk string dengan data gambar
(buffer untuk png).



3. `qr.svgObject(text, [ec_level | options])`

Objek dengan jalur dan ukuran SVG.

4. `qr.matrix(teks, [ec_level])`

Kode terakhir ini untuk Array 2D.



Options

Dari metode-metode tadi, ada beberapa opsi seperti berikut::

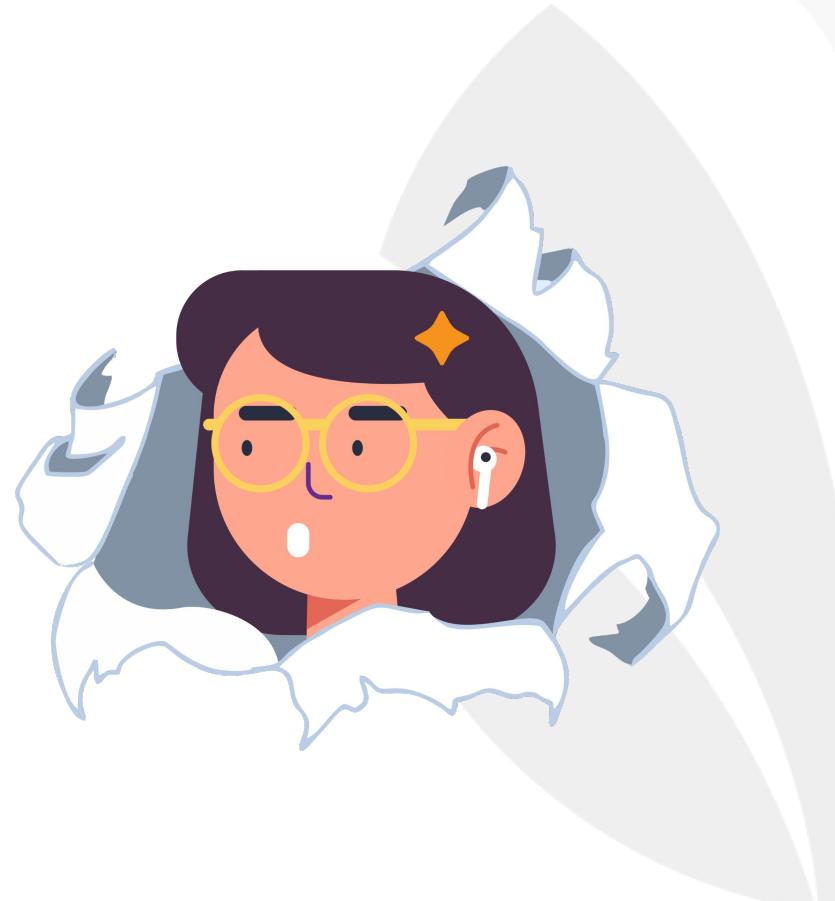
1. **Text**

Karakter string yang harus di-encoding menjadi pola QR code.

2. **ec_level**

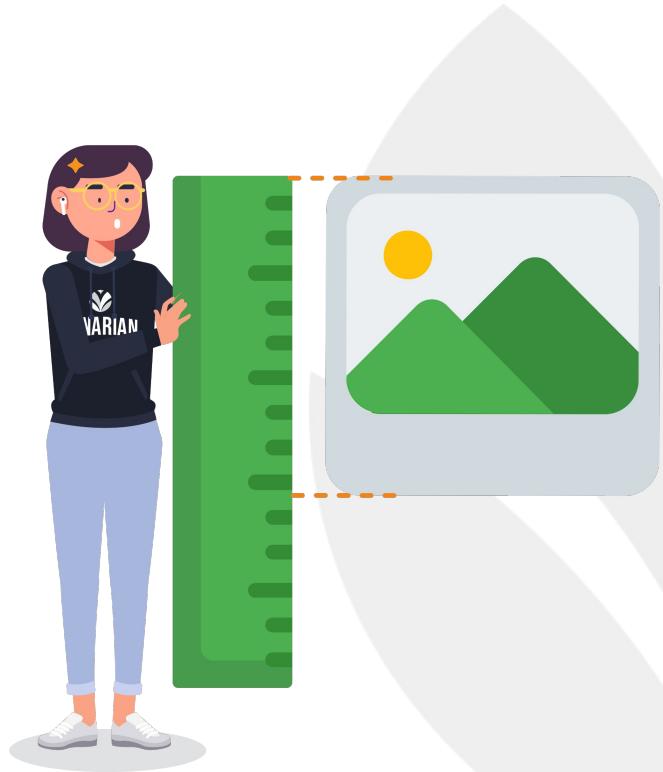
Tingkat koreksi kesalahan. Dibedakan dari ukurannya yaitu L (Large), M (Medium), Q (Quarter), H (High).

Default atau yang paling sering dipakai yaitu M.



3. Options — objek opsi gambar:

- **ec_level** — Default M.
- **type** — Jenis gambar. Defaultnya png. Bisa juga loh dibuat format svg, pdf dan eps.
- **size** (hanya png dan svg) — ukuran lebar atau tinggi image dalam piksel. Defaultnya 100 untuk png dan tidak ditentukan untuk svg.



- **margin** — ruang putih di sekitar gambar QR dalam modul. Defaultnya 4 untuk png dan 1 untuk lainnya.
- **customize** (hanya png) — berfungsi untuk menyesuaikan bitmap qr sebelum menyandikan ke PNG.
- **parse_url** (eksperimental, default salah) — coba optimalkan kode QR untuk URL.

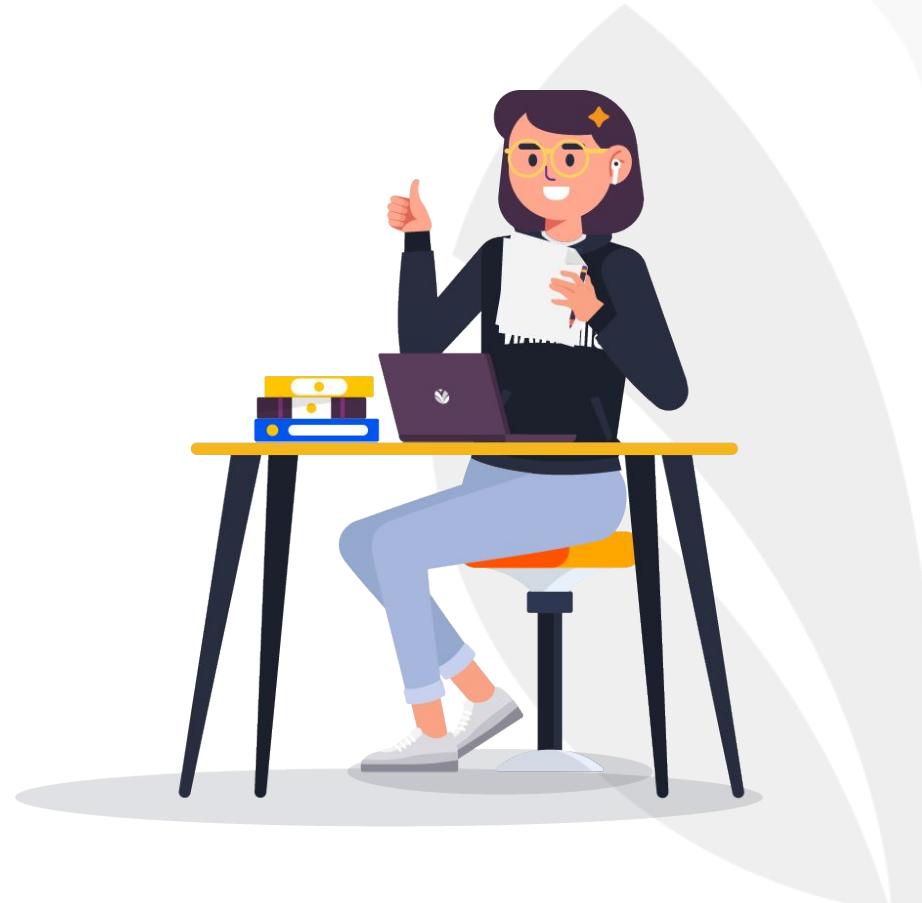


Generate & Download QR Code

Biar lebih paham, kamu dapat melihat contoh lebih lanjutnya pada repository berikut ya sob 👇

<https://github.com/wanming/qr-image>

Jika ada bagian-bagian yang belum kamu pahami, dapat didiskusikan bareng teman-teman dan Facilitator di kelasmu 🌟



Nah, pada bagian ini, kita akan lanjut mempelajari tentang bagaimana mengintegrasikan Cloud Storage Services menggunakan Imagekit.io 😍

Seperti apa ya pembahasannya? Yuk, kita simak bareng pembahasannya di slide berikut ➡



Apa itu Cloud Storage Services? 🌐

Cloud Storage Services merupakan layanan penyimpanan data yang memungkinkan kamu untuk menyimpan data kamu di internet.

Dengan layanan ini, kamu bisa mengakses data kamu dari mana saja dan dari perangkat apa saja yang terhubung ke internet.



Beberapa manfaat dari adanya Cloud Storage Services adalah seperti berikut:

- **Kapasitas yang besar**👉 Cloud Storage Services menawarkan kapasitas penyimpanan yang besar, sehingga kamu bisa menyimpan semua data kamu dengan aman.
- **Akses yang mudah**👉 Kamu bisa mengakses data kamu dari mana saja dan dari perangkat apa saja yang terhubung ke internet.



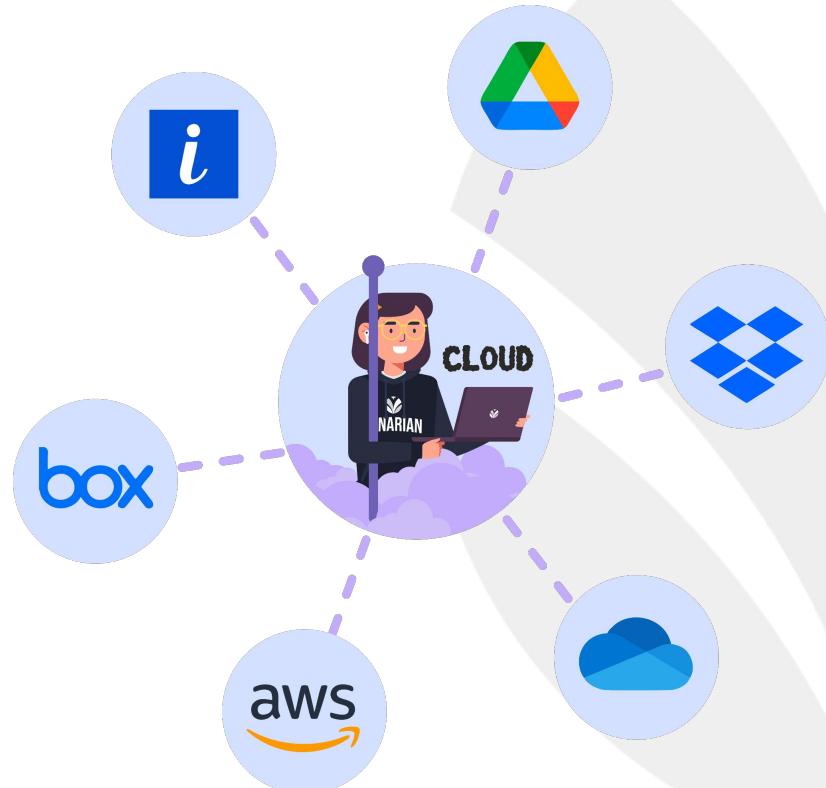
- **Keamanan yang terjamin**👉 Cloud Storage Services menggunakan teknologi keamanan yang canggih untuk melindungi data kamu dari serangan cyber.
- **Ketersediaan yang tinggi**👉 Cloud Storage Services beroperasi 24/7, sehingga kamu bisa mengakses data kamu kapan saja.



Nah, beberapa penyedia jasa penyimpanan cloud yang populer antara lain:

- [Google Drive](#)
- [Dropbox](#)
- [OneDrive](#)
- [Amazon S3](#)
- [Box](#)
- [Imagekit](#)

Khusus dalam materi ini, pembahasan kita akan meliputi Imagekit aja ya. Cuss ke slide berikutnya 



ImageKit.io

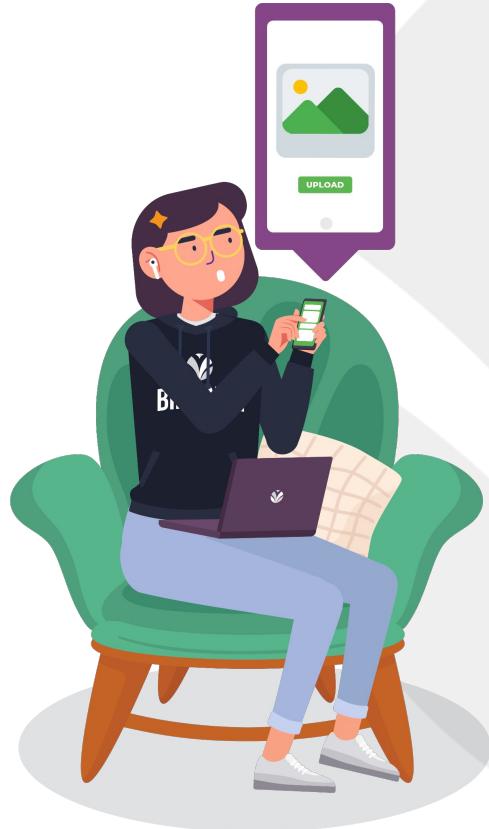
ImageKit.io adalah layanan manajemen gambar yang memungkinkan kamu untuk menyimpan, mengelola, dan mengoptimalkan gambarmu secara online.

Layanan ini menawarkan berbagai fitur dan manfaat yang dapat membantu kamu menghemat waktu dan uang, serta meningkatkan kinerja situs web kamu.

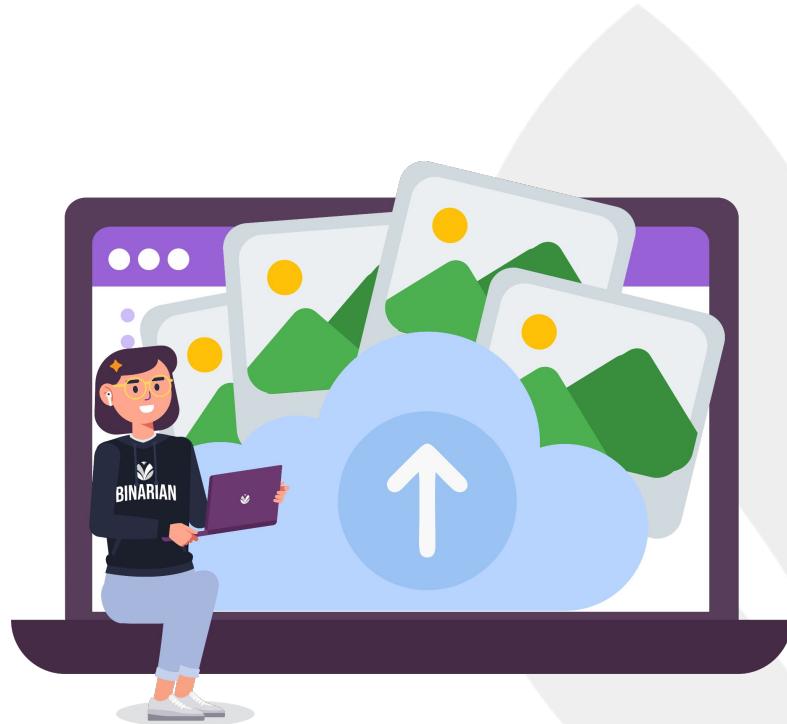


ImageKit.io menawarkan berbagai manfaat, termasuk:

- **Penyimpanan dan manajemen gambar yang mudah:** ImageKit.io menyediakan penyimpanan cloud yang andal dan aman untuk gambarmu. Kamu bisa dengan mudah mengunggah, mengelola, dan berbagi gambarmu dari mana saja.



- **Optimisasi gambar yang cepat:** ImageKit.io menawarkan berbagai alat untuk mengoptimalkan gambarmu untuk ukuran, kualitas, dan kecepatan. Ini bisa membantumu mengurangi ukuran gambarmu hingga 80% tanpa mengurangi kualitasnya.
- **Peningkatan kinerja situs web:** Gambar yang dioptimalkan dapat membantu meningkatkan kinerja situs web kamu. Ini bisa membuat situs web kamu lebih cepat dan responsif bagi pengguna.



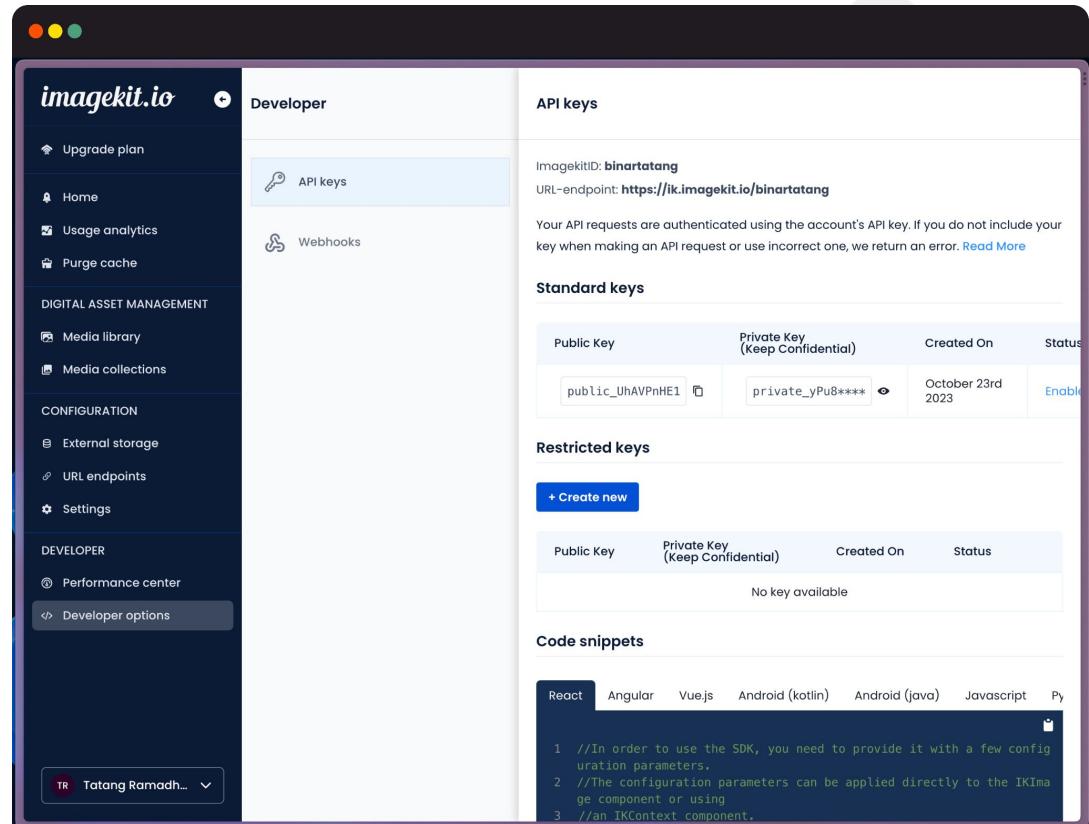
Yuk, coba implementasi Imagekit.io

Sebelum kita mulai, kita buat dulu akun imagekitnya untuk mendapatkan credentials yang kita butuhkan seperti **public_key**, **private_key** dan **endpoint_url** nya.



Setelah membuat akun baru, kamu bisa mendapatkan credentials yang kita butuhkan dengan mengarahkan navigasi ke **Developer options -> API keys**.

Salin dan sisipkan credentials tersebut kedalam file .env kita.



The screenshot shows the imagekit.io developer dashboard. On the left sidebar, under the 'DEVELOPER' section, 'Developer options' is selected. The main content area is titled 'API keys'. It displays a table for 'Standard keys' with one row:

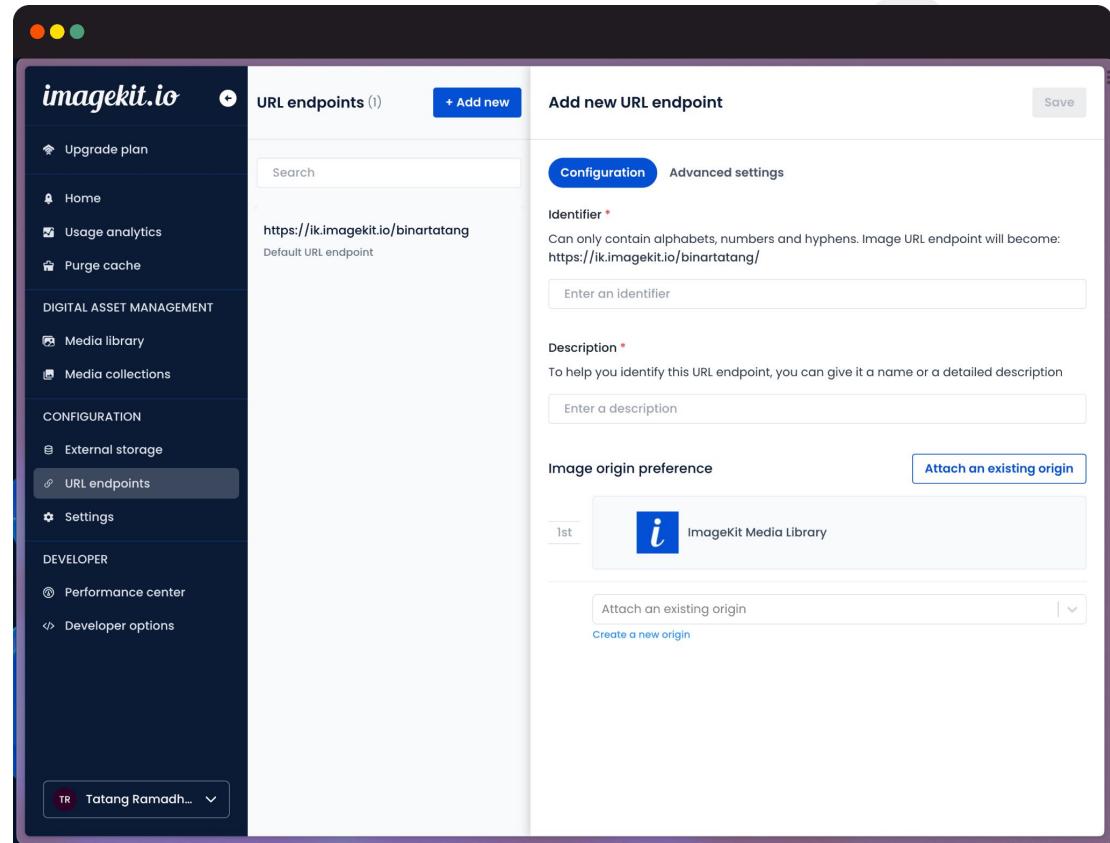
Public Key	Private Key (Keep Confidential)	Created On	Status
public_UhAVPnHE1	private_yPu8***	October 23rd 2023	Enabled

Below this is a section for 'Restricted keys' with a note: 'No key available'. At the bottom, there's a 'Code snippets' section with tabs for React, Angular, Vue.js, Android (kotlin), Android (java), Javascript, and Python. The Python tab is currently selected, showing the following code snippet:

```
1 //In order to use the SDK, you need to provide it with a few configuration parameters.  
2 //The configuration parameters can be applied directly to the IKImage component or using  
3 //an IKContext component.
```

Oh ya, kalau kamu tidak menemukan URL endpoint, artinya kamu perlu membuatnya terlebih dahulu.

Arahkan navigasi ke **URL endpoints** lalu klik **+Add new**



Nah, ini file env dengan credentials imagekit yang sudah kita tambahkan ya.

Gunakan imagekit credentials kamu sendiri ya gengs~

```
# .env file  
  
# imagekit credentials  
  
IMAGEKIT_PUBLIC_KEY="public_UhAVPnHE1AhisQEUGZcPgWAcg+s="  
  
IMAGEKIT_SECRET_KEY="private_yPu8LzzaNJuNDdur03cuVQc0q9k="  
  
IMAGEKIT_URL_ENDPOINT="https://ik.imagekit.io/binartatang"
```

Setelah itu jangan lupa install imagekitnya terlebih dahulu ya. Caranya gampang, cek aja kode berikut 👉

```
// libs/imagekit.js

var ImageKit = require('imagekit');

const {
  IMAGEKIT_PUBLIC_KEY,
  IMAGEKIT_SECRET_KEY,
  IMAGEKIT_URL_ENDPOINT
} = process.env;

module.exports = new ImageKit({
  publicKey: IMAGEKIT_PUBLIC_KEY,
  privateKey: IMAGEKIT_SECRET_KEY,
  urlEndpoint: IMAGEKIT_URL_ENDPOINT
});
```

Buat controller

```
// controllers/media.controllers.js
```

```
const imagekit = require('../libs/imagekit');
```

```
imagekitUpload: async (req, res) => {
```

```
    try {
```

```
        const stringBuffer = req.file.buffer.toString('base64');
```

```
        const uploadFile = await imagekit.upload({
```

```
            fileName: req.file.originalname,
```

```
            file: stringBuffer
```

```
    ...
```

```
    return res.json({
```

```
        status: true,
```

```
        message: 'success',
```

```
        data: {
```

```
            name: uploadFile.name,
```

```
            url: uploadFile.url,
```

```
            type: uploadFile.fileType
```

```
        }
```

```
    }); } catch (err) {
```

```
        throw err;
```

```
    }
```

```
}
```

Yang terakhir, daftarkan router. Udah deh~

Untuk lebih lengkapnya bisa lihat di sini ya gengs:

[Media Handling](#)

```
// routers/media.routers.js

const router = require('express').Router();

const storage = require('../libs/multer');

const { strogeImage, storageVideo, storageFile, imagekitUpload } = require(
  '../controllers/media.controllers');

router.post('/images', storage.image.single('image'), strogeImage);

router.post('/videos', storage.image.single('video'), storageVideo);

router.post('/files', storage.image.single('file'), storageFile);

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

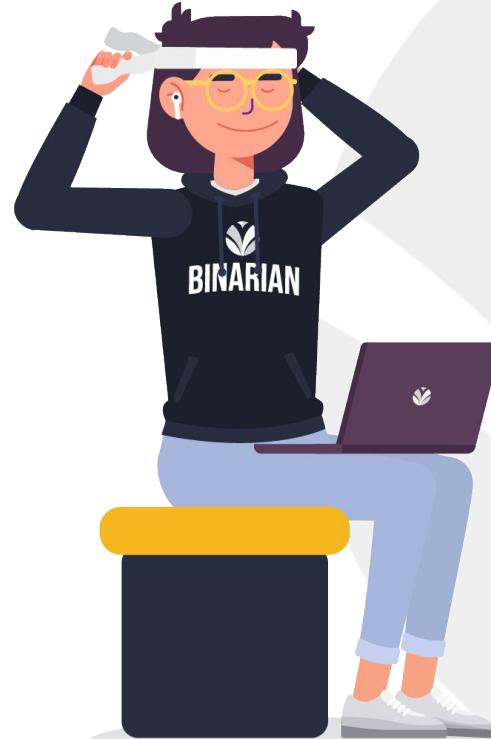
router.post('/imagekit', multer.single('image'), imagekitUpload);

module.exports = router;
```

Latihan dulu yuk!

Mengimplementasikan media handling:

1. Buat utilitas untuk mengupload media ke imagekit.
2. Buat middleware untuk membaca file menggunakan multer.
3. Buat api put /profiles/{userId} untuk memperbarui foto avatar.
4. Integrasikan seluruh komponen agar user dapat memperbarui avatar.



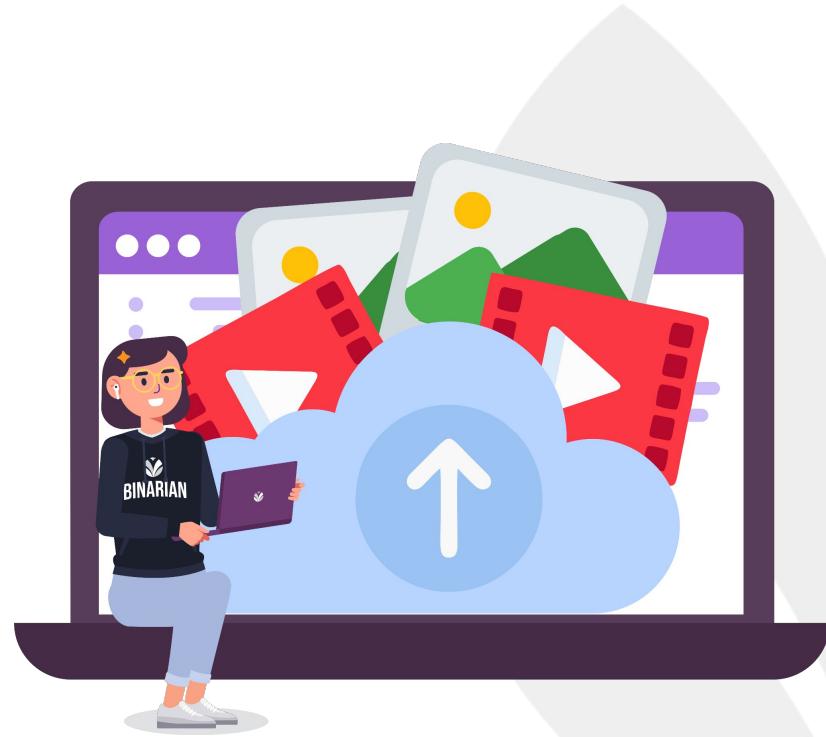
Backend Media Handling

Image Upload

Frontend melakukan Upload ke storage service dan akan mendapatkan URL > tambahkan URL tersebut ke dalam request body > URL image akan disimpan.

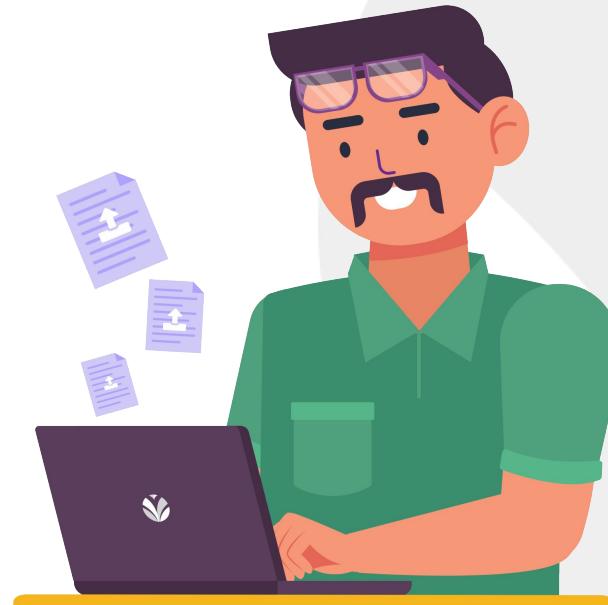
Video Upload

Caranya dengan menambahkan validasi file khusus untuk format/ file type video.



Document Upload

Caranya dengan menambahkan validasi file khusus buat format atau file type document kayak pdf.

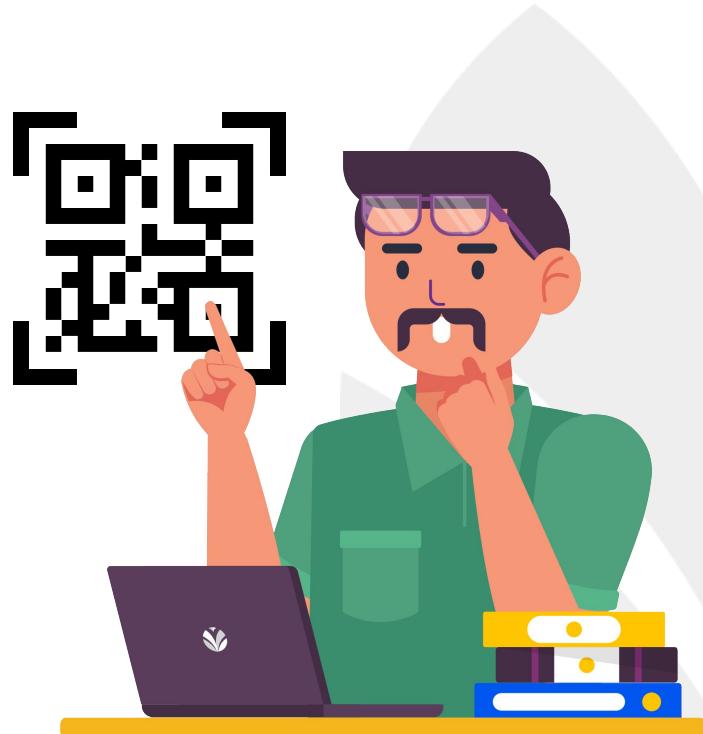


Generate & Download QR Code

QR code dapat di-generate ke dalam format .png, .svg, .eps ataupun .pdf. Semua bisa kamu pilih sesuai kebutuhan.

Jangan lupa install dulu Node QR Imagenya ya:

```
npm install node-qr-image
```



Integrating Cloud Storage Services

Cloud Storage Services merupakan layanan penyimpanan data yang memungkinkan kamu untuk menyimpan data di internet. Dengan layanan ini, kamu bisa mengakses data kamu dari mana saja dan dari perangkat apa saja yang terhubung ke internet.



Yuhuuu! Kamu udah berhasil
menamatkan **Chapter 6 Topic 1** 😊

Cusss kita lanjut ke **Topic 2** yuk, learners!



**Selamat
kamu 😊
BERHASIL**