

**EKSPLORASI MANDIRI
PEMOGRAMAN JARINGAN
EMAIL PROTOKOL**



OLEH:
RENO NILAM SARI
22343068

DOSEN PENGAMPU:
RANDI PROSKA SANDRA, M.Sc

**PRODI INFORMATIKA
DEPARTEMEN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2024**

A. LATAR BELAKANG APLIKASI

Dalam era modern yang serba terkoneksi, email telah menjadi salah satu alat komunikasi utama yang mendukung berbagai kebutuhan, baik pribadi maupun profesional. Selain digunakan untuk bertukar pesan, email juga berfungsi dalam berbagai hal seperti pengiriman notifikasi, verifikasi akun, pengaturan ulang kata sandi, promosi bisnis, hingga penyampaian laporan otomatis.

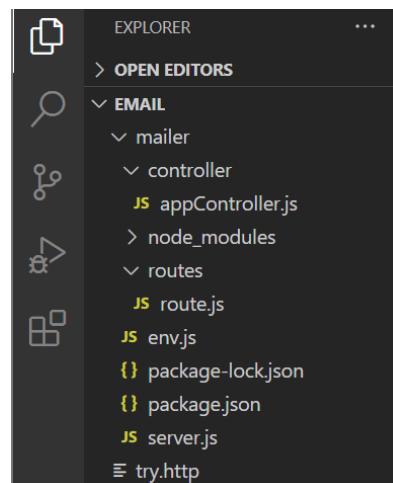
Namun, proses pengiriman email secara manual, terutama dalam jumlah besar atau untuk tugas yang berulang, seringkali menjadi tantangan karena tidak efisien dan memakan waktu. Oleh karena itu, diperlukan solusi otomatisasi yang memungkinkan pengiriman email secara cepat, efisien, dan dapat diintegrasikan dengan berbagai sistem.

Node.js Email Sender adalah aplikasi yang dirancang menggunakan Node.js dengan library seperti NodeMailer untuk mengotomasi pengiriman email melalui server. Aplikasi ini dapat memenuhi kebutuhan pengiriman, mulai dari notifikasi hingga email massal untuk promosi atau laporan berkala.

NodeMailer, sebagai salah satu library andal dalam Node.js, mendukung pengiriman email melalui protokol SMTP, API layanan email, atau server lokal. Dengan fleksibilitas tinggi dan kemudahan konfigurasi, NodeMailer menjadi pilihan ideal untuk membangun sistem pengiriman email yang efektif dan terintegrasi.

B. LANGKAH-LANGKAH PEMBUATAN APLIKASI

1. Membuat folder dan file



Membuat folder mailer dan file-file yang diperlukan sesuai tutorial, untuk package-lock.json, package.json, dan node_modules dibuat secara otomatis menggunakan perintah `npm init -y`

2. Mengisi kode kedalam file

server.js :

```
const express = require('express');
const appRoute = require('./routes/route.js')
```

```

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());

/** routes */
app.use('/api', appRoute);

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`)
})

```

appController.js

```

const nodemailer = require('nodemailer');
const Mailgen = require('mailgen');

const { EMAIL, PASSWORD } = require('../env.js')

/** send mail from testing account */
const signup = async (req, res) => {

    /** testing account */
    let testAccount = await nodemailer.createTestAccount();

    // create reusable transporter object using the default SMTP transport
    let transporter = nodemailer.createTransport({
        host: "smtp.ethereal.email",
        port: 587,
        secure: false, // true for 465, false for other ports
        auth: {
            user: testAccount.user, // generated ethereal user
            pass: testAccount.pass, // generated ethereal password
        },
    });

    let message = {
        from: '"Fred Foo 🧑" <foo@example.com>', // sender address
        to: "bar@example.com, baz@example.com", // list of receivers
        subject: "Hello ✓", // Subject line
        text: "Successfully Register with us.", // plain text body
        html: "<b>Successfully Register with us.</b>", // html body
    }

    transporter.sendMail(message).then((info) => {
        return res.status(201)
            .json({
                msg: "you should receive an email",
                info : info.messageId,
                preview: nodemailer.getTestMessageUrl(info)
            })
    }).catch(error => {
        return res.status(500).json({ error })
    })

    // res.status(201).json("Signup Successfully...!");
}

```

```

/** send mail from real gmail account */
const getbill = (req, res) => {

    const { userEmail } = req.body;

    let config = {
        service : 'gmail',
        auth : {
            user: EMAIL,
            pass: PASSWORD
        }
    }

    let transporter = nodemailer.createTransport(config);

    let MailGenerator = new Mailgen({
        theme: "default",
        product : {
            name: "Mailgen",
            link : 'https://mailgen.js/'
        }
    })

    let response = {
        body: {
            name : "Daily Tuition",
            intro: "Your bill has arrived!",
            table : {
                data : [
                    {
                        item : "Nodemailer Stack Book",
                        description: "A Backend application",
                        price : "$10.99",
                    }
                ]
            },
            outro: "Looking forward to do more business"
        }
    }

    let mail = MailGenerator.generate(response)

    let message = {
        from : EMAIL,
        to : userEmail,
        subject: "Place Order",
        html: mail
    }

    transporter.sendMail(message).then(() => {
        return res.status(201).json({
            msg: "you should receive an email"
        })
    }).catch(error => {
        return res.status(500).json({ error })
    })

    // res.status(201).json("getBill Successfully...!");
}

```

```
module.exports = {
  signup,
  getbill
}
```

route.js

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

const { signup, getbill } = require('../controller/appController.js')

/** HTTP Request */
router.post('/user/signup', signup);
router.post('/product/getbill', getbill);

module.exports = router;
```

package.json

```
{
  "name": "mailer",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.21.1",
    "mailgen": "^2.0.28",
    "nodemailer": "^6.9.16",
    "nodemon": "^3.1.7"
  }
}
```

catt : mengubah script awal menjadi nodemon server.js

3. Menghubungkan ke API

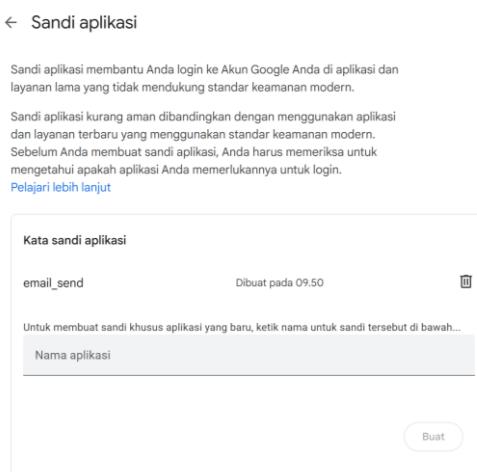
App Controller berfungsi sebagai pengelola logika utama aplikasi, mengatur bagaimana data diproses dan dikirimkan ke klien. Ketika dihubungkan dengan **routes**, controller ini bertindak sebagai "jembatan" antara permintaan yang diterima melalui API dan tanggapan yang dikirimkan kembali.

Dalam konteks pengujian menggunakan **Thunder Client**, rute-rute tersebut memungkinkan pengguna untuk mengakses endpoint API, yang diatur oleh logika dalam App Controller, seperti menambahkan, membaca, mengubah, atau menghapus data.

4. Membuat file env.js

File .env dibuat untuk menyimpan konfigurasi sensitif atau variabel lingkungan dalam sebuah proyek. File ini penting dalam pengembangan aplikasi karena membantu menjaga keamanan dan mempermudah pengelolaan konfigurasi.

Dalam konteks ini, kita membuatnya untuk menyimpan email dan password aplikasi yang didapatkan dari akun google.



berikut source code nya:

```
module.exports = {
  EMAIL: 'renonilamsari2806@gmail.com',
  PASSWORD: '',
};
```

5. Melakukan pengiriman email

Untuk mencoba fungsi pengiriman email, kita dapat menggunakan email acak yang disediakan oleh layanan seperti **TempMail**. TempMail adalah layanan email sementara yang memungkinkan kita menerima email tanpa perlu mendaftar atau menggunakan alamat email pribadi.

Berikut langkah-langkahnya:

1. **Akses TempMail:**

Kunjungi situs **TempMail** (<https://temp-mail.org>) untuk mendapatkan alamat email acak yang dapat digunakan.

2. **Salin Email:**

Salin alamat email yang diberikan oleh TempMail.

3. **Kirim Email dari Aplikasi:**

Gunakan alamat email TempMail sebagai penerima di aplikasi email sender Anda. Misalnya, saat mengirim permintaan API menggunakan Thunder Client:

```
{  
    "userEmail" : "yowev26650@kimasoft.com"  
}
```

4. **Periksa Inbox TempMail:**

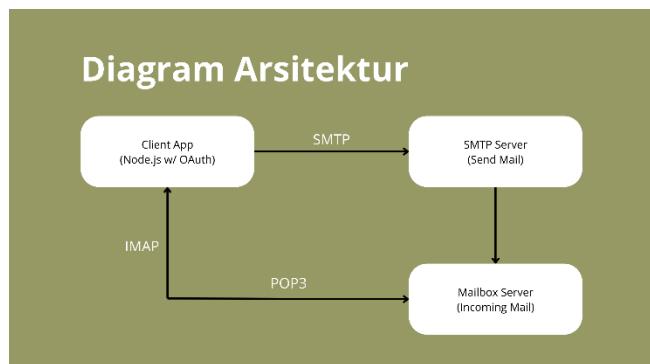
Kembali ke situs TempMail dan periksa inbox untuk melihat apakah email telah diterima.

C. DIAGRAM APLIKASI

Diagram Arsitektur

Komponen utama:

1. **Client (Aplikasi Node.js dengan Nodemailer)** - Mengirim email melalui server SMTP dan menerima email melalui IMAP atau POP3.
2. **Server Email** - Mengelola pengiriman, penerimaan, dan autentikasi email.



a. IMAP (Internet Message Access Protocol)

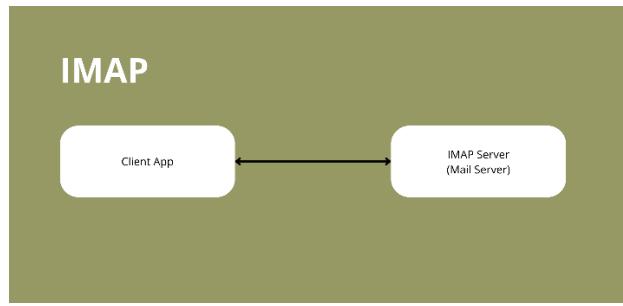
Penjelasan:

IMAP memungkinkan pengguna membaca email langsung dari server tanpa mengunduhnya sepenuhnya, sehingga cocok untuk sinkronisasi email di berbagai perangkat. Perubahan (seperti membaca, menghapus) disinkronkan di server.

Diagram Alur IMAP:

1. Client (Laptop, Ponsel) ↔ IMAP Server

- Client meminta daftar email.
- Email disinkronkan dan hanya diunduh saat dibuka.



b. POP3 (Post Office Protocol 3)

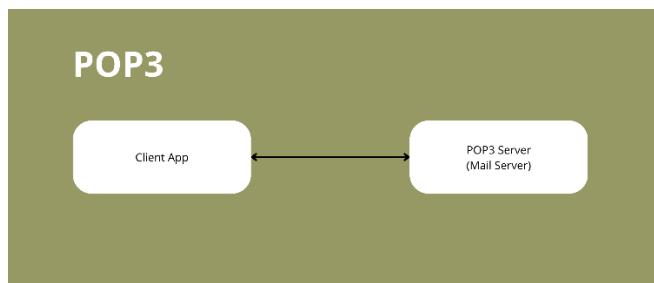
Penjelasan:

POP3 mengunduh email dari server ke perangkat lokal, biasanya menghapus email dari server setelah pengunduhan. Cocok untuk penggunaan satu perangkat dengan koneksi offline.

Diagram Alur POP3:

1. Client ← POP3 Server

- Email diunduh penuh ke perangkat lokal.
- Salinan email sering dihapus dari server.



c. SMTP (Simple Mail Transfer Protocol)

Penjelasan:

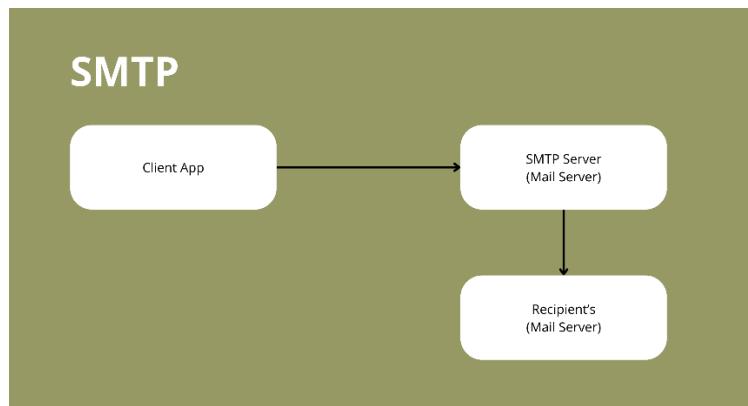
SMTP digunakan untuk mengirim email dari client ke server, atau antar server. Protokol ini menangani pengiriman keluar, sedangkan IMAP/POP3 menangani penerimaan.

Diagram Alur SMTP:

1. Client → SMTP Server → Penerima

- Email dikirimkan dari client ke server pengirim.

- Server SMTP mengarahkan email ke penerima melalui jaringan.



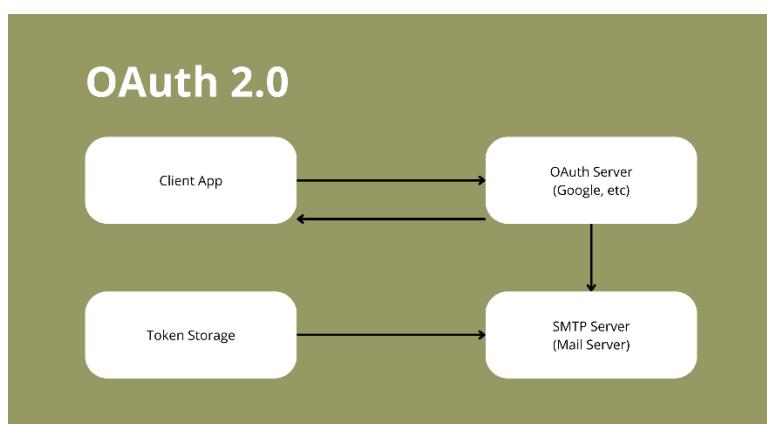
d. OAuth 2.0

Penjelasan:

OAuth 2.0 adalah protokol autentikasi yang memungkinkan aplikasi mendapatkan akses aman ke sumber daya tanpa menyimpan kredensial pengguna, menggunakan token akses.

Diagram Alur OAuth 2.0 dalam Nodemailer:

1. **Client App → OAuth Server (Google, Microsoft, dll.)**
 - Client meminta token akses.
2. **OAuth Server → Client App**
 - OAuth Server mengembalikan token akses.
3. **Client App → SMTP Server**
 - Token digunakan untuk mengautentikasi pengiriman email.



D. REFERENSI

<https://nodemailer.com/>

<https://www.indoworx.com/pengertian-imap-pop3-dan-smtp/>