# LAPORAN TUGAS PEMROGRAMAN WEB

## TUGAS KELIMA—*Praktikum Modul Kelima: Node.js*


Nama : Theo Jesen Naftali Sirait

NRP : 3125600106

Dosen Pengajar : Prof. M. Udin Harun Al Rasyid S. Kom, Ph. D

**PROGRAM STUDI S. Tr. TEKNIK INFORMATIKA**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA (PENS)**

**TAHUN 2026**

# BAB I

# PERCOBAAN

## 1.1. Percobaan Pertama—File System (fs) & Globals: Membaca File HTML.

### 1.1.1. *Source Code:*

app1.js > ...

```
import express from "express";
import path from "path";
import { fileURLToPath } from "url";
import routes from "./routes/index.js";

const app = express();
const port = 3000;

const __dirname = path.dirname(fileURLToPath(import.meta.url));

app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));
app.use("/", routes);
app.use((req, res) => {
  res.status(404).send("404 Tidak ada hhalamn");
});

app.listen(port, () => {
  console.log("Running on 3000");
});

```

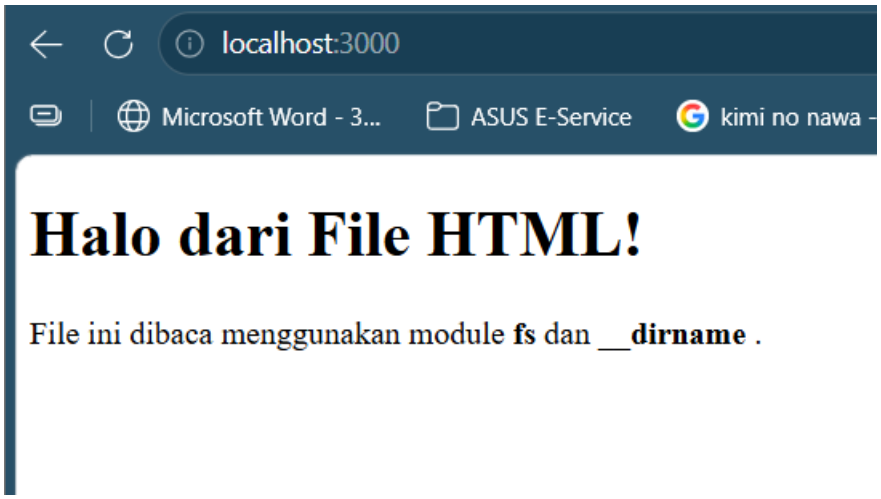
routes > index copy.js > ...

```
import express from "express";

const router = express.Router();

router.get("/", (req, res) => {
  console.log("acc", req.url);
  res.render("percobaan1");
});

export default router;
```

views > <% percobaan1.ejs > ...

```ejs
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>percbn 1</title>
  </head>
  <body>
    <h1>Halo dari File HTML!</h1>
    <p>File ini dibaca menggunakan module <strong>fs</strong> dan <strong>__dirname</s
  </body>
</html>

```

### 1.1.2. *Output:*




### 1.1.3. Penjelasan:

Pada percobaan ini, saya mempelajari bagaimana Node.js menggunakan modul fs dan variabel global __dirname untuk membaca file HTML secara asynchronous dan menampilkannya melalui server HTTP. Penggunaan __dirname memastikan path file bersifat absolut sehingga menghindari error lokasi file, sementara fs.readFile() memungkinkan proses pembacaan file dilakukan tanpa memblokir eksekusi program lain. Hasilnya menunjukkan bahwa Node.js mampu mengirimkan konten statis ke client dengan efisien melalui mekanisme non-blocking I/O.

## 1.2. Percobaan Kedua—Buffer & Console Module: Manipulasi Data Biner..

### 1.2.1. *Source Code:*

nodecoba > percobaan2.js > ...

```
1  const buf = Buffer.from("Belajar Node.js di Kuliah Pemrograman Web", "utf8");
2  console.time("WaktuProses"); // Console module untuk mengukur performa
3  console.log("--- Analisis Data ---");
4  console.dir(buf); // Menampilkan struktur objek buffer
5  console.log("Isi String:", buf.toString());
6  console.log("Ukuran Buffer:", buf.length, "bytes");
7  console.timeEnd("WaktuProses");
8
```

### 1.2.2. *Output:*

```
Node.js v22.17.1
PS D:\laragon\www\tugas5\nodecoba> node percobaan2.js
--- Analisis Data ---
Buffer(41) [Uint8Array] [
   66, 101, 108,  97, 106,  97, 114,  32,  78,
  111, 100, 101,  46, 106, 115,  32, 100, 105,
   32,  75, 117, 108, 105,  97, 104,  32,  80,
  101, 109, 114, 111, 103, 114,  97, 109,  97,
  110,  32,  87, 101,  98
]
Isi String: Belajar Node.js di Kuliah Pemrograman Web
Ukuran Buffer: 41 bytes
WaktuProses: 8.859ms
PS D:\laragon\www\tugas5\nodecoba>
```

### 1.2.3. Penjelasan:

Percobaan ini berfokus pada representasi data dalam bentuk buffer (biner) serta penggunaan modul console untuk analisis performa. Saya mengubah string menjadi buffer menggunakan Buffer.from() untuk memahami bagaimana data dikirim dalam bentuk byte di level rendah. Selain itu, penggunaan console.time() dan console.timeEnd() memberikan insight mengenai waktu eksekusi program, sehingga membantu dalam proses benchmarking sederhana. Ini menunjukkan pentingnya efisiensi dalam pengolahan data di Node.js.

## 1.3. Percobaan Ketiga—Crypto Module: Enkripsi Sederhana.

### 1.3.1. *Source Code:*

```
nodecoba > percobaan3.js > ...
1  import crypto from "crypto";
2  const algoritma = "sha256";
3  const rahasia = "kode-dosen-123";
4  // Membuat hash HMAC
5  const hash = crypto.createHmac(algoritma, rahasia).update("Data Mahasiswa A").digest("
6
7  console.log(`Algoritma: ${algoritma}`);
8  console.log(`Hasil Hash: ${hash}`);
9
```

### 1.3.2. *Output:*

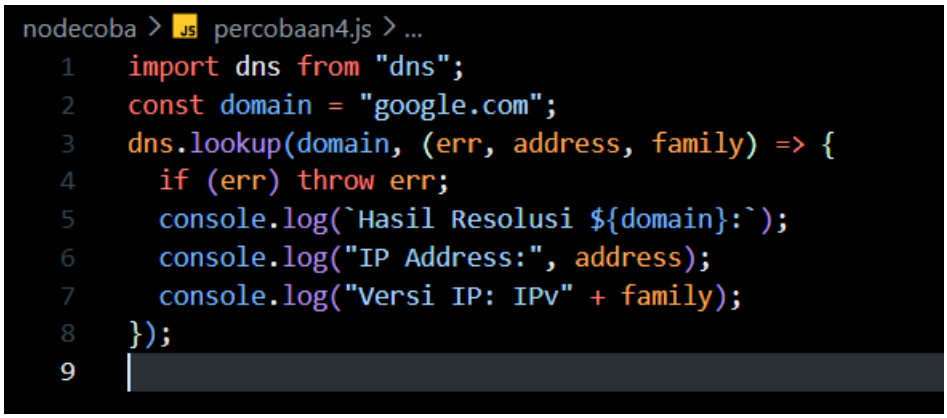
```
PS D:\laragon\www\tugas5\nodecoba> node percobaan3.js
Algoritma: sha256
Hasil Hash: 5457dba24626213b5dfcc4eaf3294ef2b5182adce518701714829e5
54158e759
```

### 1.3.3. Penjelasan:

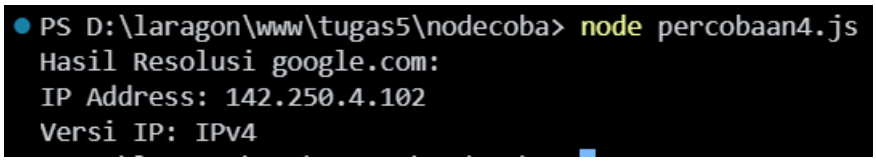
Pada percobaan ini, saya mempelajari konsep dasar keamanan data dengan menggunakan modul crypto untuk membuat hash berbasis algoritma SHA-256. Dengan fungsi crypto.createHmac(), data diubah menjadi hash yang bersifat irreversible, sehingga meningkatkan keamanan informasi. Proses ini menegaskan bahwa Node.js memiliki dukungan bawaan untuk implementasi kriptografi sederhana, yang sangat relevan dalam pengembangan sistem autentikasi maupun proteksi data.

## 1.4. Percobaan Keempat—DNS Module: Resolusi Domain.

### 1.4.1. *Source Code:*




### 1.4.2. *Output:*




### 1.4.3. Penjelasan:

Percobaan ini menunjukkan bagaimana Node.js berinteraksi dengan sistem jaringan melalui modul dns. Dengan menggunakan dns.lookup(), saya dapat menerjemahkan nama domain menjadi alamat IP, yang merupakan proses fundamental dalam komunikasi internet. Hasil percobaan memperlihatkan bahwa Node.js dapat memanfaatkan layanan DNS dari sistem operasi untuk melakukan resolusi domain secara efisien, sehingga penting dalam pengembangan aplikasi berbasis jaringan.

## 1.5. Percobaan Kelima—HTTP & Asset Management: Server Sederhana.

### 1.5.1. *Source Code:*

app1.js > ...

```
import express from "express";
import path from "path";
import { fileURLToPath } from "url";
import routes from "./routes/index.js";

const app = express();
const port = 3000;

const __dirname = path.dirname(fileURLToPath(import.meta.url));

app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));
app.use("/", routes);
app.use((req, res) => {
  res.status(404).send("404 Tidak ada hhalamn");
});

app.listen(port, () => {
  console.log("Running on 3000");
});

```

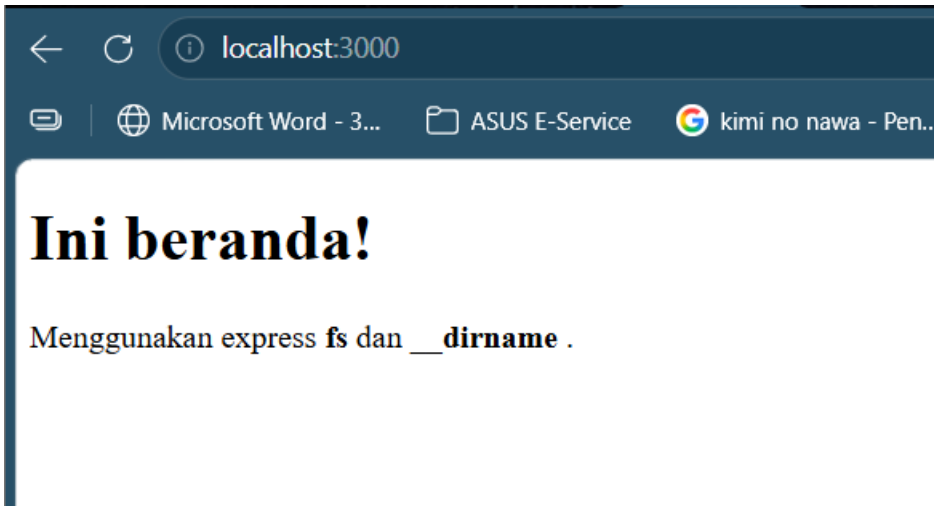
routes > index.js > ...

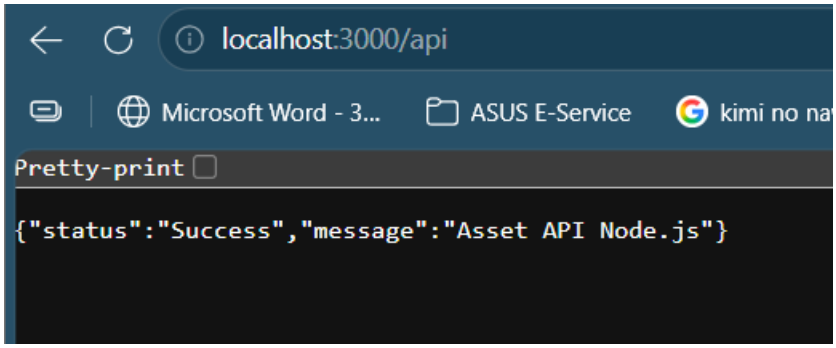
```
import express from "express";

const router = express.Router();

router.get("/", (req, res) => {
  console.log("acc", req.url);
  res.status(200).render("cekapi");
});

router.get("/api", (req, res) => {
  const data = { status: "Success", message: "Asset API Node.js" };
  res.writeHead(200, { "Content-Type": "application/json" });
  res.end(JSON.stringify(data));
});

export default router;

```

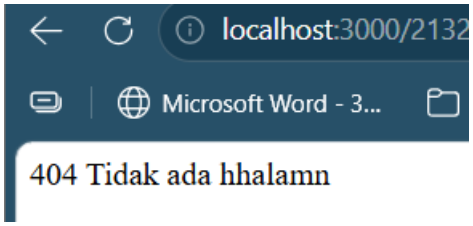
views > <% cekapi.ejs > ...

```ejs
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>percbn 1</title>
  </head>
  <body>
    <h1>Ini beranda!</h1>
    <p>Menggunakan express <strong>fs</strong> dan <strong>__dirname</strong> .</p>
  </body>
</html>
```

### 1.5.2. *Output:*



**1.5.3.** Penjelasan:

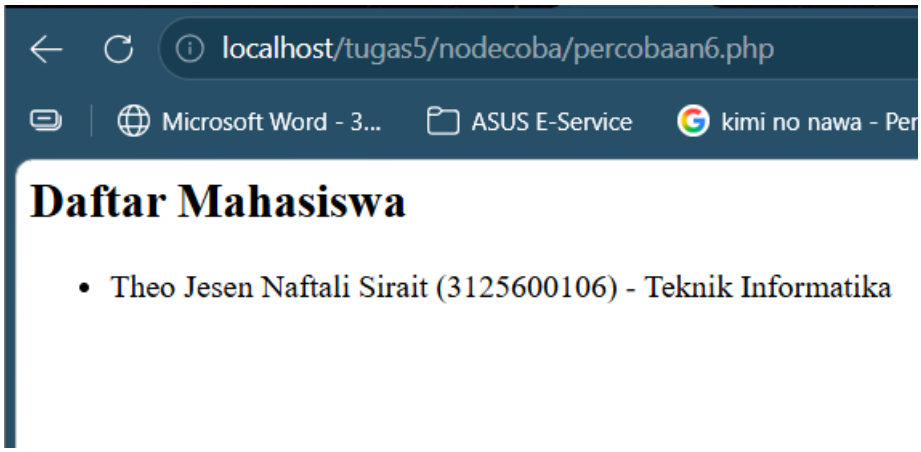
Pada percobaan ini, saya membuat server HTTP sederhana yang mampu menangani beberapa endpoint (routing dasar) berdasarkan URL yang diakses pengguna. Dengan memanfaatkan req.url, server dapat membedakan permintaan antara halaman utama dan endpoint API, serta memberikan respons dalam format HTML maupun JSON. Hal ini menunjukkan bagaimana Node.js dapat digunakan untuk membangun backend sederhana yang menangani request dan response secara dinamis.

## 1.6. Percobaan Keenam—Menampilkan Data Tabel MongoDB dengan PHP.

### 1.6.1. *Source Code:*

```php
<?php
require 'vendor/autoload.php';
$client = new MongoDB\Client("mongodb://localhost:27017");
$collection = $client->universitas->mahasiswa;
$cursor = $collection->find();
echo "<h2>Daftar Mahasiswa</h2>";
echo "<ul>";
foreach ($cursor as $dokumen) {
echo "<li>" . $dokumen['nama'] . " (" . $dokumen['nim'] . ") - " .
$dokumen['jurusan'] . "</li>";
}
echo "</ul>";
?>
```

### 1.6.2. *Output:*




### 1.6.3. Penjelasan:

Pada bagian ini, saya mengintegrasikan database MongoDB dengan PHP untuk menampilkan data dalam bentuk HTML. Menggunakan fungsi find(), seluruh dokumen dalam koleksi berhasil diambil dan ditampilkan melalui proses iterasi menggunakan cursor. Percobaan ini menunjukkan bagaimana data NoSQL dapat diakses dan disajikan ke pengguna secara dinamis, serta memperkenalkan konsep integrasi antara backend server dan database dalam aplikasi web.

# BAB II

# TUGAS PRAKTIKUM

## 2.1. The Secure Network Logger

### 2.1.1. *Struktur File:*


*Source Code:*

```
praktikum > logic.js > ip
1  import dns from "dns/promises";
2  import crypto from "crypto";
3  import fs from "fs/promises";
4
5  const { address: ip } = await dns.lookup("github.com");
6  const hash = crypto.createHash("sha256").update(ip).digest("hex");
7  const timestamp = new Date().toISOString();
8  await fs.appendFile("./log/security_log.txt", `[${timestamp}] IP: ${ip} SHA-256: ${hash}\n`);
9
```

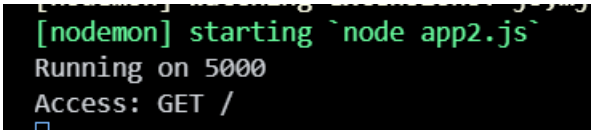
```
praktikum > app2.js > ...
1   import express from "express";
2   import routes from "./routes/index.js";
3   import "./logic.js";
4
5   const app = express();
6   const port = 5000;
7
8   app.use("/", routes);
9   app.use((req, res) => {
10    res.status(404).send("404 Tidak ada hhalamn");
11  });
12
13  app.listen(port, () => {
14    console.log(`Running on ${port}`);
15  });
16
```

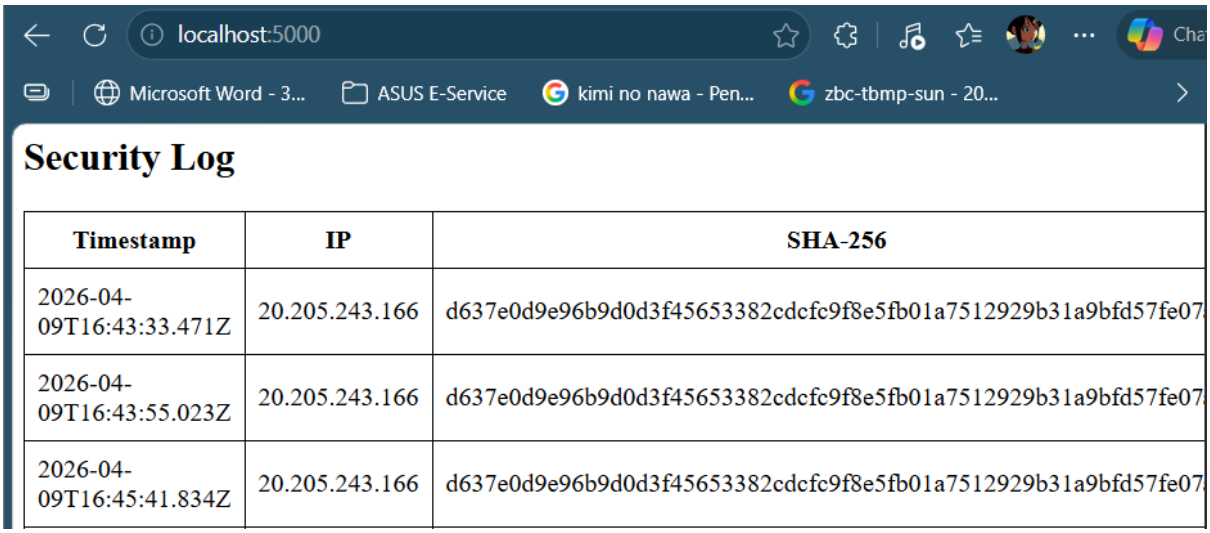
praktikum > routes > index.js > default

```
1   import express from "express";
2   import { handleResponse } from "../public/index.js";
3   const router = express.Router();
4
5   router.get(
6     "/",
7     (req, res, next) => {
8       console.info(`Access: ${req.method} ${req.url}`);
9       next();
10    },
11    handleResponse,
12  );
13
14  export default router;
15
```

```
import fs from "fs/promises";
import path from "path";
import { fileURLToPath } from "url";

const __dirname = path.dirname(fileURLToPath(import.meta.url));

export async function handleResponse(req, res) {
  try {
    const filePath = path.join(__dirname, "../log/security_log.txt");
    const data = await fs.readFile(filePath, "utf-8");
    const rows = data
      .trim()
      .split("\n")
      .map((line) => {
        const match = line.match(/\[(.*?)\] IP: (.*?) SHA-256: (.*)/);
        if (!match) return "";

        const [, time, ip, hash] = match;

        return `
            <tr>
              <td>${time}</td>
              <td>${ip}</td>
              <td>${hash}</td>
            </tr>
          `;
      })
      .join("");
    res.send(`
        <html>
          <head>
            <title>Security Log</title>
            <style>
              table { border-collapse: collapse; width: 100%; }
              th, td { border: 1px solid black; padding: 8px; }
            </style>
          </head>
          <body>
            <h2>Security Log</h2>
            <table>
              <tr>
                <th>Timestamp</th>
                <th>IP</th>
                <th>SHA-256</th>
              </tr>
              ${rows}
            </table>
          </body>
        </html>
      `);
  } catch (err) {
    res.status(500).send("file gaada");
  }
}
```

*Output:*




### 2.1.2. Penjelasan

Pada tugas praktikum ini, saya mengintegrasikan beberapa modul utama dalam Node.js yaitu dns, crypto, fs, http, dan console untuk membangun aplikasi logging berbasis jaringan. Proses dimulai dengan melakukan DNS lookup terhadap domain github.com untuk mendapatkan alamat IP, yang kemudian diamankan menggunakan hashing algoritma SHA-256 melalui modul crypto. Hasil hash tersebut disimpan ke dalam file lokal menggunakan modul file system dengan tambahan timestamp sebagai penanda waktu pencatatan. Selanjutnya, saya membangun server HTTP pada port 5000 yang berfungsi untuk membaca file log tersebut dan menampilkannya dalam bentuk tabel HTML saat endpoint utama diakses. Selain itu, setiap akses ke server dicatat menggunakan console.info sebagai bentuk monitoring aktivitas. Implementasi ini menunjukkan bagaimana Node.js dapat digunakan untuk membangun sistem logging sederhana yang menggabungkan aspek jaringan, keamanan, penyimpanan data, dan layanan web dalam satu alur terintegrasi.

# DOKUMENTASI

*naftali.it.student.pens.ac.id*