LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT BERGERAK

MODUL VIII

RUNTIME CONFIGURATION DAN INTERNATIONALIZATION

Disusun Oleh :

Yogi Hafidh Maulana

S1SE-06-02

Asisten Praktikum :

Muhamad Taufiq Hidayat

Dosen Pengampu :

Riyan Dwi Yulian Prakoso, S.Kom., M.Kom.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

DIREKTORAT TELKOM KAMPUS PURWOKERTO

2025

BAB I
PENDAHULUAN

A. DASAR TEORI

1. Runtime Configuration

Runtime Configuration adalah pendekatan dalam pengembangan perangkat lunak yang memungkinkan konfigurasi aplikasi disimpan dalam file eksternal (seperti JSON), bukan dikodekan langsung (*hardcoded*) di dalam program. Dengan teknik ini, pengguna atau pengembang dapat mengubah perilaku aplikasi tanpa harus memodifikasi logika utama atau *source code*-nya. Dalam konteks aplikasi bank transfer ini, konfigurasi disimpan dalam file bank_transfer_config.json yang berisi:

- Bahasa aplikasi (lang),
- Batas biaya transfer dan besaran biaya (threshold, low_fee, high_fee),
- Metode transfer yang tersedia (methods),
- Kata konfirmasi dalam berbagai bahasa (confirmation).

2. Internationalization

Internationalization (i18n) adalah proses merancang dan membangun perangkat lunak agar dapat diadaptasi ke berbagai bahasa dan wilayah geografis tanpa perlu perubahan signifikan pada kode sumber. Dalam modul ini, aspek i18n ditangani secara sederhana dengan cara:

- Menyimpan kata/frasa penting dalam berbagai bahasa pada file konfigurasi JSON.
- Menggunakan nilai properti lang untuk menentukan bahasa yang digunakan aplikasi saat runtime.
- Menyesuaikan teks yang ditampilkan berdasarkan bahasa yang dipilih (misalnya, prompt, label, konfirmasi transaksi).

B. MAKSUD DAN TUJUAN

1. Maksud

Modul ini dibuat untuk memberikan pemahaman praktis kepada mahasiswa mengenai penerapan konsep Runtime Configuration dan Internationalization (i18n) dalam pengembangan aplikasi berbasis Node.js. Dengan pendekatan berbasis console, mahasiswa akan belajar bagaimana sebuah aplikasi dapat dibuat fleksibel dan mudah dikonfigurasi tanpa harus memodifikasi logika program utama, serta mampu beradaptasi dengan kebutuhan pengguna dari latar belakang bahasa yang berbeda.

2. Tujuan

Setelah mengikuti praktikum ini, mahasiswa diharapkan mampu:

1. Membuat aplikasi berbasis console dengan Node.js yang terstruktur dan modular.
2. Menerapkan runtime configuration menggunakan file konfigurasi eksternal (format JSON) untuk mengatur parameter aplikasi.
3. Mengimplementasikan fitur internationalization (i18n) sederhana agar aplikasi dapat menyesuaikan bahasa tampilan berdasarkan konfigurasi.
4. Membaca dan memuat konfigurasi menggunakan modul File System (fs) dalam Node.js.
5. Menghubungkan konfigurasi dengan logika aplikasi, seperti perhitungan biaya transfer, pilihan metode, dan konfirmasi transaksi.
6. Membiasakan diri dengan struktur project yang modular, memisahkan antara logika aplikasi, konfigurasi, dan data eksternal untuk kemudahan pengembangan lebih lanjut.

BAB II
IMPLEMENTASI (GUIDED)

Code

bank_transfer_config.json

```json
{
  "lang": "en",
  "transfer": {
    "threshold": 25000000,
    "low_fee": 6500,
    "high_fee": 15000
  },
  "methods": ["RTO (real-time)", "SKN", "RTGS", "BI FAST"],
  "confirmation": {
    "en": "yes",
    "id": "ya"
  }
}
```

BankTransferConfig.js

```js
const fs = require('fs');
const path = require('path');

class BankTransferConfig {
  constructor() {
    this.configPath = path.join(__dirname,
'../data/bank_transfer_config.json');
    this.defaultConfig = {
      lang: "en",
      transfer: {
        threshold: 25000000,
        low_fee: 6500,
        high_fee: 15000
      },
      methods: ["RTO (real-time)", "SKN", "RTGS", "BI FAST"],
      confirmation: {
        en: "yes",
        id: "ya"
      }
    };
    this.config = this.loadConfig();
  }

  loadConfig() {
    if (fs.existsSync(this.configPath)) {
      const data = fs.readFileSync(this.configPath, 'utf8');
      return JSON.parse(data);
    } else {
      this.saveConfig(this.defaultConfig);
      return this.defaultConfig;
    }
  }

  saveConfig(config) {
    fs.writeFileSync(this.configPath, JSON.stringify(config, null, 2));
  }
}

module.exports = BankTransferConfig;
```

BankTransferApp.js

```
const readline = require('readline');
const BankTransferConfig = require('../config/BankTransferConfig');

class BankTransferApp {
  constructor() {
    this.config = new BankTransferConfig().config;
    this.rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout
    });
  }

  askQuestion(query) {
    return new Promise(resolve => this.rl.question(query, resolve));
  }

  async run() {
    const lang = this.config.lang;

    const promptAmount = lang === "en" ?
      "Please insert the amount of money to transfer: " :
      "Masukkan jumlah uang yang akan di-transfer: ";

    const amountStr = await this.askQuestion(promptAmount);
    const amount = parseFloat(amountStr);

    const fee = amount <= this.config.transfer.threshold
      ? this.config.transfer.low_fee
      : this.config.transfer.high_fee;

    const total = amount + fee;

    if (lang === "en") {
      console.log(`Transfer fee = ${fee}`);
      console.log(`Total amount = ${total}`);
    } else {
      console.log(`Biaya transfer = ${fee}`);
      console.log(`Total biaya = ${total}`);
    }

    console.log(lang === "en" ? "Select transfer method:" : "Pilih metode
transfer:");
    this.config.methods.forEach((method, idx) => {
      console.log(`${idx + 1}. ${method}`);
    });

    await this.askQuestion(lang === "en" ? "Choose a method (press Enter after): " :
"Pilih metode (tekan Enter setelah): ");

    const confirmationPrompt = lang === "en" ?
      `Please type "${this.config.confirmation.en}" to confirm the transaction: ` :
      `Ketik "${this.config.confirmation.id}" untuk mengkonfirmasi transaksi: `;

    const confirmationInput = await this.askQuestion(confirmationPrompt);

    if (
      (lang === "en" && confirmationInput.trim().toLowerCase() ===
this.config.confirmation.en) ||
      (lang === "id" && confirmationInput.trim().toLowerCase() ===
this.config.confirmation.id)
    ) {
      console.log(lang === "en" ? "The transfer is completed" : "Proses transfer
berhasil");
    } else {
      console.log(lang === "en" ? "Transfer is cancelled" : "Transfer dibatalkan");
    }

    this.rl.close();
  }
}

module.exports = BankTransferApp;
```

index.js

```
const BankTransferApp = require('./app/BankTransferApp');

const app = new BankTransferApp();
app.run();
```

Output

```
D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime Configuration
e index.js
Please insert the amount of money to transfer: 1000000
Transfer fee = 6500
Total amount = 1006500
Select transfer method:
1. RTO (real-time)
2. SKN
3. RTGS
4. BI FAST
Choose a method (press Enter after): 4
Please type "yes" to confirm the transaction: yes
The transfer is completed

D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime Configuration
```

Deskripsi Code

Praktikum tadi pagi membuat aplikasi berbasis Node.js yang memanfaatkan runtime configuration dan internationalization (i18n) untuk membuat aplikasi transfer bank yang fleksibel dan mudah diubah tanpa menyentuh logika program utama. Konfigurasi aplikasi, seperti bahasa, batas minimal transfer, biaya transfer, metode transfer, dan kata konfirmasi, disimpan dalam file JSON eksternal (bank_transfer_config.json). File ini dibaca oleh class BankTransferConfig, yang juga akan otomatis membuat file konfigurasi default jika tidak ditemukan. Ketika aplikasi dijalankan melalui index.js, class BankTransferApp akan membaca konfigurasi tersebut dan menyesuaikan seluruh tampilan aplikasi—termasuk pertanyaan, label biaya, dan instruksi konfirmasi—berdasarkan bahasa yang dipilih (misalnya "en" untuk Inggris atau "id" untuk Indonesia). Ini menunjukkan bagaimana konfigurasi runtime memungkinkan perubahan cepat pada perilaku dan tampilan aplikasi tanpa perlu mengubah kode program, serta bagaimana internationalization memungkinkan aplikasi digunakan oleh pengguna dari berbagai bahasa.

BAB III
PENUGASAN (UNGUIDED)

Ubah bahasa default menjadi id lalu jalankan lagi programnya

Tambahkan opsi metode transfer baru di file JSON

```
{
  "lang": "id",
  "transfer": {
    "threshold": 25000000,
    "low_fee": 6500,
    "high_fee": 15000
  },
  "methods": ["RTO (real-time)", "SKN", "RTGS", "BI FAST", "QRIS"],
  "confirmation": {
    "en": "yes",
    "id": "ya"
  }
}
```

```
D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime
e index.js
Masukkan jumlah uang yang akan di-transfer: 200000
Biaya transfer = 6500
Total biaya = 206500
Pilih metode transfer:
1. RTO (real-time)
2. SKN
3. RTGS
4. BI FAST
5. QRIS
Pilih metode (tekan Enter setelah): 5
Ketik "ya" untuk mengkonfirmasi transaksi: YA
Proses transfer berhasil

D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime
```

Apa yang terjadi kalau input transfer amount bukan angka?

```
D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime
e index.js
Masukkan jumlah uang yang akan di-transfer: Yogi Hafidh Maulana
Biaya transfer = 15000
Total biaya = NaN
Pilih metode transfer:
1. RTO (real-time)
2. SKN
3. RTGS
4. BI FAST
5. QRIS
Pilih metode (tekan Enter setelah): 2
Ketik "ya" untuk mengkonfirmasi transaksi: ya
Proses transfer berhasil

D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\08_Runtime
```

Ketika pengguna diminta untuk memasukkan jumlah uang yang akan ditransfer, aplikasi menggunakan perintah readline untuk membaca input dari konsol. Nilai input tersebut disimpan dalam variabel string (amountStr), lalu diubah menjadi angka dengan fungsi parseFloat. Namun, jika pengguna memasukkan nilai yang bukan angka valid—seperti huruf (misalnya "abc"), karakter simbol (seperti "@#$"), atau membiarkannya kosong—fungsi parseFloat tidak dapat mengkonversi nilai tersebut menjadi angka yang valid. Hasil dari parseFloat dalam kasus seperti ini adalah NaN, yang merupakan singkatan dari "Not a Number".

Masalahnya muncul ketika nilai NaN tersebut digunakan dalam perhitungan matematis. Karena amount adalah NaN, maka operasi matematika apapun yang melibatkan NaN akan menghasilkan NaN juga. Akibatnya, output yang ditampilkan kepada pengguna menjadi tidak logis atau membingungkan. Secara teknis, aplikasi tidak akan mengalami crash atau menampilkan error secara eksplisit, namun hasil yang keluar tidak bisa diandalkan dan bisa menyesatkan pengguna yang tidak paham apa itu NaN.

Untuk mengatasi hal ini, sebaiknya program dilengkapi dengan validasi setelah parseFloat dijalankan. Dengan memeriksa apakah hasil parsing adalah angka valid menggunakan isNaN(), kita bisa menghentikan program secara elegan dan menampilkan pesan kesalahan yang lebih informatif.