LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT BERGERAK

MODUL VIII

RUNTIME CONFIGURATION DAN INTERNATIONALIZATION

Disusun Oleh :

Satria Ariq Adelard Dompas

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 Runtime Configuration merupakan suatu pendekatan dalam pengembangan perangkat lunak yang memungkinkan penyimpanan konfigurasi aplikasi di luar kode sumber, misalnya dalam bentuk file eksternal seperti JSON. Pendekatan ini memberikan fleksibilitas kepada pengguna maupun pengembang untuk menyesuaikan perilaku aplikasi tanpa perlu melakukan perubahan langsung pada logika utama atau kode program. Dalam konteks aplikasi transfer bank, konfigurasi disimpan dalam file bernama bank_transfer_config.json, yang memuat informasi sebagai berikut:

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

2. Internationalization

Internationalization Internationalization (i18n) merupakan proses perancangan dan pengembangan perangkat lunak agar dapat dengan mudah disesuaikan untuk berbagai bahasa dan wilayah geografis tanpa memerlukan perubahan signifikan pada kode sumber. Dalam modul ini, penerapan i18n dilakukan secara sederhana melalui langkah-langkah berikut:

- Penyimpanan kata-kata atau frasa penting dalam berbagai bahasa ke dalam file konfigurasi berformat JSON,
- Pemanfaatan properti lang untuk menentukan bahasa yang akan digunakan oleh aplikasi saat dijalankan (runtime),
- Penyesuaian tampilan teks, seperti prompt, label, dan pesan konfirmasi transaksi, sesuai dengan bahasa yang dipilih.).

B. MAKSUD DAN TUJUAN

1. Maksud

Modul ini dirancang untuk memberikan pemahaman praktis kepada mahasiswa terkait implementasi konsep *Runtime Configuration* dan *Internationalization (i18n)* dalam pengembangan aplikasi berbasis Node.js. Melalui pendekatan berbasis antarmuka konsol, mahasiswa akan mempelajari bagaimana sebuah aplikasi dapat dirancang agar bersifat fleksibel dan mudah dikonfigurasi tanpa perlu melakukan perubahan pada logika utama program, serta mampu menyesuaikan diri dengan preferensi pengguna dari berbagai latar belakang bahasa.

2. Tujuan

Setelah mengikuti praktikum ini, mahasiswa diharapkan memiliki kemampuan untuk:

1. Mengembangkan aplikasi berbasis konsol menggunakan Node.js dengan struktur yang terorganisir dan modular,
2. Menerapkan runtime configuration melalui penggunaan file konfigurasi eksternal berformat JSON untuk mengatur berbagai parameter aplikasi,
3. Mengimplementasikan fitur internationalization (i18n) secara sederhana agar tampilan bahasa aplikasi dapat disesuaikan berdasarkan pengaturan dalam konfigurasi,
4. Mengakses dan memuat file konfigurasi menggunakan modul File System (fs) yang tersedia di Node.js,
5. Mengintegrasikan konfigurasi dengan logika aplikasi, mencakup proses perhitungan biaya transfer, pemilihan metode transfer, serta konfirmasi transaksi,
6. Membiasakan diri dengan struktur proyek yang modular, dengan pemisahan yang jelas antara logika aplikasi, konfigurasi, dan data eksternal, guna mendukung pengembangan dan pemeliharaan yang lebih efisien.

BAB II
IMPLEMENTASI (GUIDED)

Code

bank_transfer_config.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

```
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;
```

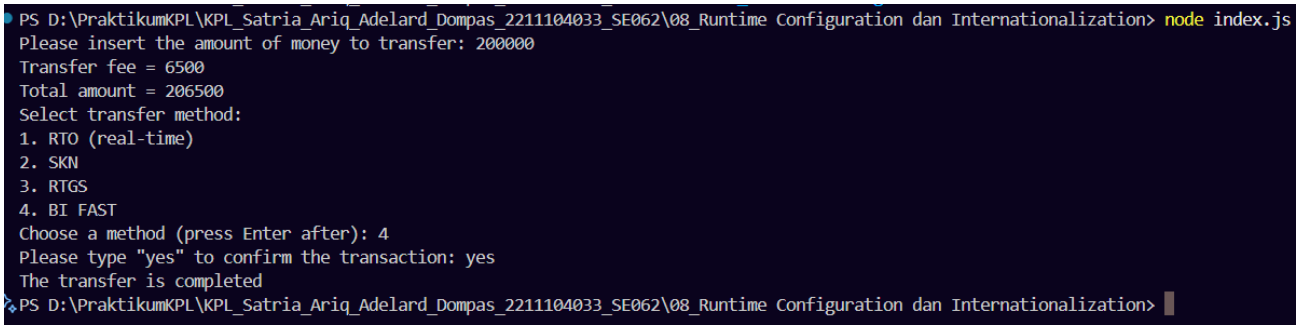
BankTransferApp.js

```
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



Deskripsi Code

Pada praktikum yang dilaksanakan pagi tadi, telah dibuat sebuah aplikasi berbasis Node.js yang menerapkan konsep *runtime configuration* dan *internationalization* (i18n). Aplikasi tersebut dirancang untuk melakukan proses transfer bank dengan fleksibilitas tinggi, sehingga memungkinkan perubahan konfigurasi tanpa perlu memodifikasi logika utama program.

Seluruh konfigurasi aplikasi—seperti pilihan bahasa, batas minimum transfer, biaya transfer, metode transfer, serta kata konfirmasi—disimpan dalam sebuah file eksternal berformat JSON dengan nama *bank_transfer_config.json*. File ini dibaca oleh kelas `BankTransferConfig`, yang juga memiliki kemampuan untuk secara otomatis menghasilkan file konfigurasi default apabila file yang dimaksud tidak ditemukan.

Pada saat aplikasi dijalankan melalui berkas *index.js*, kelas `BankTransferApp` akan mengambil konfigurasi tersebut dan menyesuaikan seluruh antarmuka aplikasi—termasuk pertanyaan, label biaya, serta instruksi konfirmasi—berdasarkan bahasa yang telah ditentukan, misalnya "en" untuk bahasa Inggris atau "id" untuk bahasa Indonesia.

Penerapan *runtime configuration* ini menunjukkan bagaimana perilaku dan tampilan aplikasi dapat diubah dengan cepat tanpa intervensi langsung terhadap kode sumber. Selain itu, penggunaan *internationalization* memungkinkan aplikasi untuk menjangkau dan digunakan oleh pengguna dengan latar belakang bahasa yang berbeda.

.

BAB III
PENUGASAN (UNGUIDED)

Ubah bahasa default menjadi id lalu jalankan lagi programnya

Tambahkan opsi metode transfer baru di file JSON

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

```
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\08_Runtime Configuration dan Internationalization> node index.js
Please insert the amount of money to transfer: 400000
Transfer fee = 6500
Total amount = 406500
Select transfer method:
1. RTO (real-time)
2. SKN
3. RTGS
4. BI FAST
5. QRIS
Choose a method (press Enter after): 5
Please type "yes" to confirm the transaction: yes
The transfer is completed
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\08_Runtime Configuration dan Internationalization>
```

Apa yang terjadi kalau input transfer amount bukan angka?

```
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\08_Runtime Configuration dan Internationalization> node index.js
Please insert the amount of money to transfer: Satria Ariq Adelard Dompas
Transfer fee = 15000
Total amount = NaN
Select transfer method:
1. RTO (real-time)
2. SKN
3. RTGS
4. BI FAST
5. QRIS
Choose a method (press Enter after): 5
Please type "yes" to confirm the transaction: yes
The transfer is completed
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\08_Runtime Configuration dan Internationalization>
```

Pada saat pengguna diminta untuk memasukkan jumlah uang yang akan ditransfer, aplikasi memanfaatkan modul readline untuk membaca input dari konsol. Nilai yang diperoleh disimpan dalam variabel bertipe string (amountStr) dan kemudian dikonversi menjadi angka menggunakan fungsi parseFloat. Namun demikian, apabila input yang diberikan oleh pengguna bukan merupakan angka yang valid—seperti huruf (contohnya "abc"), karakter khusus (seperti "@#$"), atau bahkan dibiarkan kosong—maka fungsi parseFloat tidak akan mampu mengubahnya menjadi nilai numerik yang sah. Dalam situasi seperti ini, hasil dari parseFloat adalah NaN (*Not a Number*).

Permasalahan timbul ketika nilai NaN tersebut digunakan dalam operasi matematika. Karena amount memiliki nilai NaN, maka seluruh perhitungan yang melibatkan variabel ini juga akan menghasilkan NaN. Hal ini menyebabkan keluaran program menjadi tidak sesuai atau membingungkan bagi pengguna. Meskipun aplikasi tidak mengalami *crash* atau menampilkan pesan kesalahan secara eksplisit, hasil akhir yang diberikan tidak dapat diandalkan dan berpotensi menyesatkan, terutama bagi pengguna yang tidak memahami makna dari NaN.

Untuk menghindari situasi tersebut, sebaiknya aplikasi dilengkapi dengan mekanisme validasi setelah proses konversi menggunakan parseFloat. Dengan memanfaatkan fungsi isNaN() untuk memeriksa apakah hasil parsing merupakan angka yang valid, aplikasi dapat menghentikan proses secara terkontrol dan menampilkan pesan kesalahan yang informatif, sehingga pengalaman pengguna tetap terjaga dan kesalahan dapat ditangani secara elegan.

.