Penerapan Regular Expression dalam Konversi Hardcoded Text ke Struktur i18n pada Aplikasi Java Spring Boot

Suryani Mulia Utami – 13524042
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: 13524042@std.stei.itb.ac.id, suryaniutami449@gmail.com

***Abstract*—Dukungan multibahasa atau internasionalisasi memegang peran penting dalam meningkatkan jangkauan dan aksesibilitas perangkat lunak modern bagi pengguna global. Salah satu kendala yang umum dijumpai adalah banyaknya pesan antarmuka atau *user-facing text* yang dibuat secara *hardcoded* dan tersebar di *source code*. Makalah ini membahas penerapan *regular expression* sebagai pendekatan semi-otomatis untuk mendeteksi dan mengeksternalisasi *hardcoded text* pada aplikasi Java Spring Boot. Pendeteksian dilakukan melalui pencocokan pola string, filtering terhadap elemen teknis, serta pembangkitan *key* i18n secara otomatis. Hasil dari proses tersebut berupa berkas eksternal serta rekomendasi *rewrite* agar dapat mengikuti struktur i18n. Pendekatan ini menunjukkan bahwa *regular expression* dapat dimanfaatkan sebagai langkah awal dalam migrasi aplikasi menuju dukungan multibahasa, khususnya untuk mengurangi pekerjaan manual pada tahap deteksi dan eksternalisasi teks.**

***Kata Kunci*—, Regex, Multibahasa, Internasionalisasi, Java Spring Boot, i18n**

I. Pendahuluan

Perkembangan aplikasi modern membuat pengguna tidak lagi terbatas pada satu wilayah atau negara. Penggunaan bahasa yang sesuai dengan pengguna dapat meningkatkan *user experience* karena informasi yang ditampilkan menjadi lebih mudah dipahami. Oleh karena itu, dukungan multibahasa menjadi salah satu kebutuhan penting dalam pengembangan perangkat lunak. Selain teks pada antarmuka pengguna, dukungan multibahasa juga mencakup pesan kesalahan, respons API, pesan validasi, notifikasi, dan email yang diterima pengguna.

Dalam praktik pengembangan perangkat lunak, terdapat beberapa pendekatan untuk mendukung multibahasa. Pendekatan yang umum digunakan adalah *internationalization* (i18n), yaitu proses mempersiapkan struktur aplikasi agar dapat beradaptasi dengan berbagai bahasa tanpa perlu mengubah logika utama program [1]. Pendekatan ini dilakukan dengan memisahkan teks yang ditampilkan kepada pengguna dari *source code* dan menyimpannya pada berkas konfigurasi eksternal berdasarkan bahasa/wilayah *locale* tertentu [1].

Masalahnya adalah ketika aplikasi yang sudah berjalan belum dibangun dengan struktur i18n sejak awal tahap pengembangan [2]. Pada kondisi tersebut, teks yang seharusnya dilokalisasi belum memiliki penanda khusus dan masih ditulis secara langsung (*hardcoded user-facing text*) di berbagai *layer* kode. Akibatnya, pengembang harus melakukan identifikasi dan pemindahan teks secara manual, di mana proses ini terbukti sangat membosankan (*tedious*) serta rentan terhadap kesalahan manusia (*human error*) [2].

Salah satu pendekatan yang dapat digunakan untuk membantu proses tersebut adalah memanfaatkan *regular expression* untuk menganalisis *source code* secara statis [5]. Dengan mendefinisikan pola tertentu, *regular expression* dapat digunakan untuk mendeteksi string yang berpotensi ditampilkan kepada pengguna. Pada konteks aplikasi Java Spring Boot, pola tersebut dapat berupa teks pada `ResponseEntity`, pesan pada *annotation validation*, subjek email, maupun string sederhana yang mengandung parameter.

Berdasarkan hal tersebut, makalah ini membahas penerapan *regular expression* dalam perancangan program yang membantu proses deteksi dan eksternalisasi *hardcoded user-facing text* pada aplikasi Java Spring Boot. Program yang dirancang menghasilkan *key* i18n serta berkas `messages.properties` sebagai hasil eksternalisasi teks.

II. Landasan Teori

Teori utama yang digunakan dalam makalah ini adalah *Regular expression* (Regex) untuk melakukan deteksi dan *filtering source code* untuk mendapatkan daftar string yang perlu diinternasionalisasi. Untuk memahami pendekatan yang digunakan, diperlukan pemahaman mengenai Regex sebagai metode pencocokan string dan struktur i18n secara singkat.

A. Pencocokan String

Pencocokan string (*string matching*) adalah permasalahan untuk menemukan kemunculan suatu pola di dalam sebuah teks yang lebih panjang. Secara formal, komponen yang terlibat didefinisikan sebagai berikut:

1) T: Teks (*text*), yaitu *string* berukuran n karakter yang menjadi objek pencarian.
2) P: Pola (*pattern*), yaitu *string* berukuran m karakter yang dicari di dalam T (dengan asumsi $m \ll n$).

Tujuan utama dari algoritma ini adalah mengidentifikasi seluruh posisi indeks awal di mana P muncul sebagai *substring* dari T.

B. Regex

Regular Expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau *string*. Regex digunakan untuk pencocokan *string* (*string matching*) secara efisien. Sintaksis dasar dari regular expression umumnya terdiri dari beberapa elemen pencocokan berikut [4], [5]:

1) **String Literal**
   Digunakan untuk melakukan pencarian *string* yang identik secara persis (case-sensitive).
   Contoh: Pola regex `stima` hanya akan cocok dengan teks "stima".
2) **Metakarakter**
   Merupakan sejumlah karakter khusus yang memiliki makna sintaksis tersendiri dan mempengaruhi proses pencocokan pola. Karakter yang termasuk metakarakter dalam regex adalah: `<([{^-$!|])?*+.>`. Jika ingin mencocokkan karakter ini sebagai teks biasa, diperlukan karakter *escape* (\).
   Beberapa metakarakter utama meliputi:
   - **Wildcard** (`.`): Mencocokkan karakter tunggal apa pun kecuali baris baru (*newline*).
   - **Alternasi/Union** (`|`): Berfungsi sebagai operator logika OR untuk memilih antar-pola.
     Contoh: `aa|bb` akan mencocokkan "aa" atau "bb".
3) **Character Classes**
   Digunakan untuk mencocokkan satu karakter dari sekumpulan opsi karakter yang didefinisikan di dalam kurung siku.
   - **Simple Class**: Bentuk paling dasar di mana karakter opsi ditulis langsung secara berderet.
     Contoh: Regex `[bcr]at` akan cocok dengan "bat", "cat", dan "rat".
   - **Negasi**: Mencocokkan karakter apapun selain dari yang dituliskan dengan menambahkan simbol ^ di awal kurung siku.
     Contoh: Regex `[^bcr]at` berarti karakter pertama bukan "b", "c", atau "r". Teks "hat" akan cocok, sedangkan "cat" menjadi tidak cocok.
   - **Ranges**: Digunakan untuk menentukan rentang karakter atau angka menggunakan tanda hubung (-).
     Contoh: `[a-e]` mencocokkan huruf 'a' sampai 'e', dan `[1-5]` mencocokkan angka 1 sampai 5.
   - **Unions**: Gabungan atau kombinasi dari beberapa rentang kelas karakter sekaligus.
     Contoh: `[a-e[v-z]]` menyatakan pencocokan untuk huruf 'a' sampai 'e' digabung dengan 'v' sampai 'z'.
   - **Predefined Character Class**: Kelas karakter bawaan singkat yang merepresentasikan sekumpulan karakter umum:

Construct	Deskripsi	Contoh
`.`	Semua karakter (*kecuali newline*)	`a.c` cocok dengan "abc"
`\d`	Karakter digit (angka)	Ekivalen dengan `[0-9]`
`\D`	Karakter non-digit	Ekivalen dengan `[^0-9]`
`\s`	Karakter *whitespace*	Mencocokkan `[ \t\n\x0B\f\r]`
`\S`	Karakter non-*whitespace*	Ekivalen dengan `[^\s]`
`\w`	Karakter kata / *alphanumeric*	Ekivalen dengan `[a-zA-Z_0-9]`
`\W`	Karakter non-*word*	Ekivalen dengan `[^\w]`

4) **Quantifier (Greedy dan Lazy)**
   Quantifier mendefinisikan jumlah perulangan pola (*closure*). Secara standar, *quantifier* bersifat *greedy*. Jika ditambahkan karakter `?`, sifatnya berubah menjadi *lazy* atau mencocokkan sesedikit mungkin.

Greedy	Lazy	Deskripsi
`X?`	`X??`	Muncul 1 kali atau tidak sama sekali (0 atau 1)
`X*`	`X*?`	Muncul 0 kali atau lebih (banyak)
`X+`	`X+?`	Muncul 1 kali atau lebih (minimal 1 kali)
`X{n}`	`X{n}?`	Muncul tepat sebanyak n kali
`X{n,}`	`X{n,}?`	Muncul minimal sebanyak n kali
`X{n,m}`	`X{n,m}?`	Muncul antara n sampai m kali

5) **Pengelompokan (Grouping / Parentheses)**
   Menggunakan tanda kurung untuk mengelompokkan sub-pola agar dapat dikenai operasi *quantifier* atau disimpan dalam memori kelompok.
   - **Capturing Group** (`(expr)`): Mengelompokkan ekspresi dan menyimpan hasilnya untuk dipanggil kembali (misal dirujuk dengan `\1`).
   - **Non-Capturing Group** (`(?:expr)`): Mengelompokkan ekspresi hanya untuk kebutuhan struktur operasi tanpa menyimpannya dalam memori.
6) **Boundary Matchers dan Lookahead Assertions**
   Digunakan untuk membatasi posisi kecocokan pola pada tempat tertentu di dalam teks tanpa memakan (*consuming*) karakter teks itu sendiri.

Construct	Deskripsi	Contoh
^	Awal baris teks (*start of string*)	^Halo
$	Akhir baris teks (*end of string*)	ia$
\b	Batas kata (*word boundary*)	\bcat\b
\B	Batas bukan kata (*non-word boundary*)	\Bcat
\G	Akhir dari *match* sebelumnya	\G\d+
\Z	Akhir input untuk *final terminator*	test\Z
\z	Akhir dari keseluruhan input secara absolut	end\z
(?=expr)	*Positive Lookahead* (diikuti pola *expr*)	Jawa(?=barat)
(?!expr)	*Negative Lookahead* (tidak diikuti pola *expr*)	Jawa(?!timur)

C. Internationalization pada Java Spring Boot

Mekanisme *internationalization* (i18n) pada *framework* Java Spring Boot dikelola secara bawaan untuk menyediakan konten dalam berbagai bahasa tanpa mengubah logika utama program [3]. Pendekatan ini dilakukan dengan memisahkan teks yang ditampilkan kepada pengguna dari *source code* ke dalam berkas konfigurasi eksternal. Teks-teks tersebut kemudian direferensikan di dalam *source code* menggunakan sebuah kunci unik (*key*).

Melalui arsitektur ini, pesan yang semula ditulis secara langsung (*hardcoded*) seperti `"User not found"` atau `"Validation failed"` dapat dieksternalisasi menjadi bentuk *key* yang terstruktur, seperti `user.not.found` atau `error.validation.failed`. Ketika aplikasi menerima permintaan (*request*), Spring Boot secara otomatis mendeteksi bahasa yang diinginkan dan mengembalikan nilai teks yang sesuai.

D. Struktur Berkas Eksternal

Spring Boot menggunakan format berkas *property* (.properties) yang disimpan di bawah direktori `src/main/resources/` untuk menyimpan daftar terjemahan teks [3]. Ppada ekosistem Java, berkas-berkas eksternal ini dikelompokkan ke dalam satu kesatuan logis yang disebut sebagai *resource bundle*. *Resource bundle* bertindak sebagai wadah yang menyatukan kumpulan berkas konfigurasi bahasa yang memiliki nama dasar (*basename*) yang sama.

Dalam penerapannya, pengembang mendefinisikan satu berkas untuk setiap *locale* yang didukung di dalam *resource bundle* tersebut [3]. Berkas `messages.properties` bertindak sebagai *fallback* utama. Sementara itu, berkas spesifik bahasa ditandai dengan sufiks kode bahasa, seperti `messages_en.properties` untuk Bahasa Inggris atau `messages_id.properties` untuk Bahasa Indonesia [3].

Kunci fondasi dari skalabilitas sistem ini terletak pada konsistensi penulisan kunci, di mana setiap berkas wajib memiliki daftar *key* yang sama, sementara hanya nilai (*value*) dari *key* tersebut yang berubah sesuai bahasa target. Contoh struktur isi berkasnya adalah sebagai berikut:

```
# Berkas: messages_en.properties
user.not.found=User not found
product.added.success=Product added
successfully

# Berkas: messages_id.properties
user.not.found=Pengguna tidak ditemukan
product.added.success=Produk berhasil
ditambahkan
```

E. Pengambilan pesan

Untuk membaca berkas-berkas konfigurasi tersebut, Spring Boot menyediakan komponen `MessageSource`, umumnya melalui kelas implementasi `ResourceBundleMessageSource` [3]. Komponen ini dikonfigurasi dengan menentukan nama dasar berkas (*basename*), serta menetapkan *default encoding* ke format UTF-8 agar karakter non-Latin dapat dirender dengan benar [3].

Secara teknis, `MessageSource` menyediakan metode utama untuk mengambil pesan dengan cetak biru pemanggilan sebagai berikut [3]:

```
messageSource.getMessage(code, args, locale)
```

Keterangan parameter dari metode tersebut meliputi [3]:

- `code`: *String* yang merepresentasikan kunci (*key*) pesan pada berkas *properties*.
- `args`: Daftar argumen objek (*array of objects*) opsional untuk mengisi nilai dinamis pada teks.
- `locale`: Objek `Locale` yang menentukan bahasa aktif pada *request* saat ini.

Agar penentuan *locale* berjalan dinamis pada setiap *request*, Spring memanfaatkan objek konteks global yang disebut `LocaleContextHolder` [3]. Melalui pemanggilan metode statis `LocaleContextHolder.getLocale()`, aplikasi dapat mengekstrak objek *locale* aktif yang dikirim oleh pengguna melalui *request header* [3].

Mekanisme tersebut yang mendasari (*rewrite source code*) secara statis dari bentuk *string literal* menjadi kode yang adaptif terhadap multibahasa.

F. Pesan Berparameter

Pada aplikasi riil, pesan sering kali membutuhkan data dinamis yang diperoleh saat aplikasi berjalan (*runtime*), seperti nama pengguna atau variabel angka [3]. Dalam arsitektur i18n Spring, nilai dinamis tersebut tidak digabungkan menggunakan teknik konkatenasi *string* biasa (+), melainkan dinyatakan sebagai nomor indeks di dalam kurung kurawal yang berfungsi sebagai penanda posisi (*placeholder*).

Sebagai contoh, penulisan pesan berparameter didefinisikan pada berkas *properties* sebagai berikut:

```
welcome.message=Welcome, {0}!
```

Ketika metode pemanggilan pesan dijalankan, argumen variabel dilewatkan ke dalam metode komponen:

`messageSource.getMessage("welcome.message", new Object[]{"Kunal"}, locale);`

Sistem secara otomatis akan menggantikan indeks {0} dengan argumen pertama tersebut, menghasilkan keluaran `"Welcome, Kunal!"`. Mekanisme *placeholder* ini mendukung fleksibilitas lokalisasi karena setiap bahasa memiliki struktur tata bahasa yang berbeda.

III. Analisis

A. Metode Eksperimen

Eksperimen ini dirancang untuk mendeteksi dan mengeksternalisasi *hardcoded user-facing text* melalui analisis *source code* secara statis. Dalam prosesnya, *regular expression* diterapkan secara spesifik untuk mendeteksi pola teks dan melakukan penyaringan (*filtering*) terhadap kandidat *string* yang valid.

Berikut adalah parameter masukan dan luaran eksperimen:

Tabel I: Komponen Masukan dan Luaran Eksperimen

Komponen	Nama Variabel	Deskripsi
Masukan	`project_path`	Jalur direktori proyek Spring Boot
Luaran	`messages.properties`	Hasil eksternalisasi teks

Alur logis deteksi hingga generasi berkas dilakukan melalui langkah-langkah berikut:

Inisialisasi *peta_i18n* kosong (struktur *key-value*)
Pindai semua berkas `.java` pada `project_path`
For each berkas *B* **do**
  Baca isi *source code B*
  Ekstrak teks menggunakan *regular expression*
  For each teks *T* yang ditemukan **do**
    If *T* lolos tahap *filtering* **then**
      Hasilkan *key* i18n
      If *key* belum ada di *peta_i18n* **then**
        Simpan pasangan *key* dan teks *T* ke *peta_i18n*
      End If
    End If
  End For
End For
Generasi berkas `messages.properties` dari *peta_i18n*

B. Implementasi

1) Definisikan Regex untuk Deteksi
   Pada program ini, definisi *hardcoded user-facing text* pada aplikasi Java Spring Boot dibatasi pada pesan `ResponseEntity`, pesan *annotation validation*, subjek email, serta *string* sederhana berparameter. String-string ini yang akan menjadi target eksternalisasi.
   a) Pola deteksi teks pada `ResponseEntity`
      Pola ini digunakan untuk mendeteksi pesan respons yang dikembalikan oleh *backend* melalui `ResponseEntity`. Implementasi mendukung dua bentuk utama, yaitu `ResponseEntity` dengan dua argumen dan `ResponseEntity` dengan tiga argumen yang menyertakan variabel *headers*. Pada kedua bentuk tersebut, argumen pertama harus berupa string literal agar dapat dianggap sebagai *hardcoded user-facing text*.
      Pola untuk `ResponseEntity` dengan dua argumen adalah sebagai berikut.

```
new\s+ResponseEntity\s*(?:<[^>]*>)?\s*(\s
    *"((?:[^"\]|\.)*)"(?!\s*+)\s*,\s*
    HttpStatus.([A-Z_][A-Za-z0-9_]*)\s*)
```

Kode Program 1: Pola ResponseEntity dengan 2 argumen

Pola tersebut akan cocok dengan format berikut.

```
new ResponseEntity<>(STRING_LITERAL,
HttpStatus.SOMETHING)
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
new ResponseEntity<>(
"Success", HttpStatus.OK)

new ResponseEntity<>(
"Invalid OTP",
HttpStatus.NOT_ACCEPTABLE
)

new ResponseEntity<String>(
"User "admin" not found",
HttpStatus.NOT_FOUND
)
```

Pola untuk `ResponseEntity` dengan tiga argumen adalah sebagai berikut.

```
new\s+ResponseEntity\s*(?:<[^>]*>)?\s*(\s
    *"((?:[^"\]|\.)*)"(?!\s*+)\s*,\s*([A-
    Za-z_][A-Za-z0-9_]*)\s*,\s*HttpStatus
    .([A-Z_][A-Za-z0-9_]*)\s*)
```

Kode Program 2: Pola ResponseEntity dengan 3 argumen

Pola tersebut akan cocok dengan format berikut.

```
new ResponseEntity<>(
STRING_LITERAL, headers,
HttpStatus.SOMETHING)
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
new ResponseEntity<>(
"Successfully loggedOut",
headers,
HttpStatus.OK
)

new ResponseEntity<Map>(
"Data fetched",
responseHeaders,
HttpStatus.OK
)
```

Pada kedua pola tersebut, bagian `(?!\s*\+)` digunakan agar string yang mengandung operator concatenation tidak ditangkap oleh pola `ResponseEntity` biasa. String berparameter seperti `"User " + username` ditangani oleh pola concatenation sederhana.

b) Pola deteksi pesan pada *annotation validation*
Pola ini digunakan untuk mendeteksi pesan validasi yang ditulis pada atribut `message` di annotation seperti `@NotBlank`, `@NotNull`, `@Email`, atau `@Size`. Deteksi dilakukan dalam dua tahap. Tahap pertama mencari annotation validation beserta isi parameternya, sedangkan tahap kedua mengambil nilai dari atribut `message`.
Pola utama untuk mendeteksi annotation validation adalah sebagai berikut.

```
@(NotBlank|NotNull|NotEmpty|Email|Size|Min
    |Max|Pattern)\s*(((?:[^()]|([^()]*))*)
    )
```

Kode Program 3: Pola Annotation Validation

Selanjutnya, isi annotation diperiksa menggunakan pola atribut pesan berikut.

```
message\s*=\s*"((?:[^"\]|\.)*)"
```

Kode Program 4: Pola Atribut Message

Pola tersebut akan cocok dengan format berikut.

```
@AnnotationName(
..., message = "STRING_LITERAL", ...)
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
@NotBlank(
message = "username must not be empty"
)

@Email(
message = "email format is invalid"
)

@Size(
min = 3,
max = 20,
message = "name length invalid"
)
```

Apabila nilai `message` masih berupa string literal biasa, teks tersebut akan diambil sebagai kandidat i18n. Namun, apabila nilai `message` sudah menggunakan format key seperti `{msg.username.required}`, program tidak memprosesnya kembali karena pesan tersebut dianggap sudah mengikuti struktur i18n.

c) Pola deteksi subjek email
Pola ini digunakan untuk mendeteksi teks pada pemanggilan method `setSubject`. Subjek email termasuk *user-facing text* karena akan terlihat langsung oleh pengguna melalui email.
Pola dasar untuk mendeteksi subjek email berupa string literal adalah sebagai berikut.

```
.setSubject\s*(\s*"((?:[^"\]|\.)*)"(?!\s
    *+)\s*)
```

Kode Program 5: Pola Subjek Email

Pola tersebut akan cocok dengan format berikut.

```
object.setSubject("STRING_LITERAL")
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
sendSmtpEmail.setSubject(
"Email verification OTP");

mimeMessageHelper.setSubject(
"Your invoice is ready");
```

Selain bentuk string literal biasa, subjek email yang menggunakan concatenation sederhana juga dapat diproses menggunakan pola pesan berparameter. Dengan demikian, subjek seperti `"Welcome, " + name` dapat dikonversi menjadi template pesan dengan placeholder.

d) Pola deteksi concatenation sederhana
Pola ini digunakan untuk mendeteksi string yang mengandung nilai dinamis melalui operator +. Deteksi dibatasi pada concatenation sederhana yang diawali oleh string literal dan diikuti oleh variabel atau method call sederhana tanpa argumen kompleks.
Pola dasar concatenation sederhana adalah sebagai berikut.

```
"(?:[^"\]|\.)*"
(?:\s*+\s*(?:
"(?:[^"\]|\.)*"|
[A-Za-z_][A-Za-z0-9_]*
(?:.[A-Za-z_][A-Za-z0-9_]*
(?:(\s*))?)*))+
```

Kode Program 6: Pola Concatenation Sederhana

Pola tersebut akan cocok dengan format berikut.

```
"STRING_LITERAL" + variable

"STRING_LITERAL" + object.getSomething()
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
"Success created for " +
newUser.getUsername()

"Hello " + firstName + " " + lastName

"Order #" + orderId + " has been shipped"
```

Hasil deteksi diubah menjadi template pesan berparameter dengan format `{0}`, `{1}`, dan seterusnya. Sebagai contoh, kode berikut:

```
"Success created for "
+ newUser.getUsername()
```

akan dikonversi menjadi template berikut.

`Success created for {0}`

Bagian dinamis seperti `newUser.getUsername()` disimpan sebagai argumen yang nantinya digunakan pada pemanggilan i18n. Pada implementasi ini, expression yang lebih kompleks seperti ternary, operasi aritmetika, atau method call dengan argumen tidak menjadi target deteksi.

e) Pola deteksi `String.format`

Pola ini digunakan untuk mendeteksi pesan yang dibentuk menggunakan method `String.format`. Placeholder bergaya *printf* kemudian dikonversi menjadi placeholder i18n berbentuk {0}, {1}, dan seterusnya.

Pola dasar untuk mendeteksi `String.format` adalah sebagai berikut.

```
String.format\s*(\s*"((?:[^"\]|\.)*)"\s
    *(,\s*(?:[^()]|([^()]*))*)?)
```

Kode Program 7: Pola String.format

Pola tersebut akan cocok dengan format berikut.

```
String.format("FORMAT_STRING",
arg1, arg2, ...)
```

Contoh kode yang memenuhi pola tersebut adalah sebagai berikut.

```
String.format("Halo %s", name)

String.format("Total: %d item(s)",
count)

String.format("Harga %.2f", price)
```

Placeholder seperti `%s`, `%d`, dan `%f` akan dikonversi menjadi placeholder i18n. Sebagai contoh, kode berikut:

```
String.format("Halo %s", name)
```

akan dikonversi menjadi:

```
Halo {0}
```

Program hanya memproses `String.format` apabila format string ditulis langsung sebagai string literal. Apabila format string disimpan terlebih dahulu ke dalam variabel, isi format tersebut tidak menjadi target deteksi karena tidak dapat dianalisis secara langsung melalui pencocokan regex pada source code.

2) Terapkan Penyaringan Kandidat

Setelah kandidat teks ditemukan menggunakan regex, program melakukan tahap penyaringan untuk memastikan bahwa teks tersebut benar-benar layak diekternalisasi ke dalam berkas i18n. Tahap ini diperlukan karena tidak semua string literal yang tertangkap oleh regex merupakan *user-facing text*. Beberapa string dapat berupa endpoint, konfigurasi teknis, URL, MIME type, format tanggal, atau prompt internal yang tidak perlu diterjemahkan.

Pada implementasi ini, penyaringan dibuat sederhana. Kandidat yang cocok dengan pola string teknis akan diabaikan, sedangkan kandidat yang tidak cocok dengan pola penyaringan akan diteruskan ke tahap pembangkitan *key* i18n dan rekomendasi rewrite source code.

a) Pola penyaringan endpoint dan resource path

Pola ini digunakan untuk mengabaikan string yang merepresentasikan endpoint, route, atau path resource statis. String seperti ini tidak termasuk *user-facing text* karena digunakan sebagai alamat akses, bukan pesan yang ditampilkan kepada pengguna.

```
^/.*$

^(classpath:|file:|static/|public/|
    templates/|META-INF/).*$

(?i).*.(html|css|js|png|jpg|jpeg|gif|svg|
    ico|woff2?|ttf|map)$
```

Kode Program 8: Pola Endpoint dan Resource Path

Contoh string yang diabaikan:

```
/login
/resumeAnalyser/entry/v1/**
/assets/**
/index.html
classpath:/templates/
```

Dengan penyaringan ini, string yang berfungsi sebagai route atau lokasi file tidak dimasukkan ke dalam `messages.properties`.

b) Pola penyaringan konstanta teknis dan keamanan

Pola ini digunakan untuk mengabaikan string yang berkaitan dengan HTTP method, header, skema autentikasi, role, token, cookie, atau encoding. String tersebut termasuk string teknis karena digunakan oleh mekanisme komunikasi HTTP atau keamanan aplikasi.

```
^(ROLE|SCOPE|AUTHORITY)*[A-Z0-9*]+$

(?i)^(entrypasstoken|accesstoken|
    refreshtoken|
access_token|refresh_token|id_token|jwt|
    jti|
sub|iss|aud|exp|iat|nbf|scope|sessionid)$
```

Kode Program 9: Pola Konstanta Teknis dan Keamanan

Selain pola regex, beberapa nilai juga diperiksa menggunakan daftar konstanta, misalnya:

```
GET
POST
PUT
DELETE
Authorization
Set-Cookie
Bearer
Content-Type
UTF-8
```

Kandidat yang cocok dengan pola tersebut diabaikan karena tidak perlu diterjemahkan.

c) Pola penyaringan URL, MIME type, dan format tanggal
Pola ini digunakan untuk mengabaikan string yang berupa URL, MIME type, atau format tanggal dan waktu. Ketiga jenis string tersebut tidak dipindahkan ke berkas i18n karena penggunaannya lebih bersifat teknis.

```
^(https?|ftp|wss?)://\S+$

^[a-zA-Z0-9.+-]+/[a-zA-Z0-9.+-]+
(?:\s*;\s*[a-zA-Z0-9-]+=[a-zA-Z0-9-]+)?$

^(?:'[^']*'|[yMdHhmsSEazZX]|[-/:., ])+$
```

Kode Program 10: Pola URL, MIME Type, dan Format Tanggal

Contoh string yang diabaikan:

`https://api.example.com/data`
`application/json`
`multipart/form-data`
`yyyy-MM-dd`
`yyyy-MM-dd'T'HH:mm:ss`

Format tanggal dan waktu tidak diproses sebagai pesan i18n karena termasuk masalah *locale-sensitive formatting*, bukan eksternalisasi teks pesan.

d) Pola penyaringan prompt panjang dan pesan log
Pola ini digunakan untuk mengabaikan string panjang yang kemungkinan besar merupakan prompt internal atau pesan debugging. Pada project uji coba, terdapat string panjang yang digunakan sebagai prompt AI sehingga tidak perlu dimasukkan ke dalam berkas bahasa.

```
(?is).*(Analyze this resume|Response
    structure|
\bpros\b|\bcons\b|suggestions|system
    prompt|
respond (?:only |strictly )?(?:in|with)
    json).*

(?:System.(?:out|err).print(?:ln)?|
log(?:ger)?.(?:trace|debug|info|warn|error
    )|
e.printStackTrace)\s*(
```

Kode Program 11: Pola Prompt Panjang dan Log

Selain pencocokan keyword, program juga mengabaikan string yang panjangnya melebihi batas tertentu, misalnya lebih dari 250 karakter. Contoh string yang diabaikan adalah prompt analisis resume, instruksi output JSON untuk AI, atau pesan yang berada di dalam `System.out.println` dan logging.

C. Hasil Program

Pengujian dilakukan pada project Java Spring Boot yang memiliki beberapa *hardcoded user-facing text*. Program dijalankan dengan masukan berupa project_path, kemudian memindai seluruh berkas `.java`, menjalankan proses deteksi, filtering, pembangkitan key, dan generasi berkas properties.

Hasil eksekusi program ditampilkan pada terminal, meliputi jumlah berkas Java yang dipindai, jumlah kandidat yang ditemukan, jumlah kandidat yang lolos filtering, serta jumlah key i18n yang berhasil dibangkitkan.

```
jar target/i18n-regex-converter-1.0.0.jar examples/resume-analyzer-sa
mple output en,id en <DIR>          ..
[INFO] Project path  : examples/resume-analyzer-samplemple
[INFO] Output dir    : output       sample-project
[INFO] Locales       : [en, id]       0 bytes
[INFO] Source locale : en228.224.684.032 bytes free
[INFO] Java files found     : 5
[INFO] Raw candidates found : 37r 4\stima\i18n-regex-converter>
[INFO] Eligible candidates  : 37
[INFO] Ignored candidates   : 0
[INFO] Generated keys       : 30
[INFO] Properties generated successfully.
[INFO] Properties generated successfully.
[INFO] Report generated successfully.
```

Gambar 1: Hasil eksekusi program melalui terminal

Program juga menghasilkan laporan konversi dalam berkas `conversion-report.md`. Laporan tersebut memuat lokasi file, nomor baris, tipe kandidat, teks asli, template hasil konversi, serta key i18n yang dihasilkan.

```
output > conversion-report.txt
 1  REGEX-BASED I18N CONVERTER REPORT
 2  ================================
 3
 4  SUMMARY
 5  Total candidates : 37
 6  Eligible         : 37
 7  Ignored          : 0
 8
 9  DETECTED USER-FACING TEXT
10  ========================
11
12  [1]
13  File      : src/main/java/com/example/resumeanalyser/model/userLogin.java
14  Line      : 17
15  Type      : ANNOTATION_MESSAGE
16  Pattern   : Annotation validation message
17  Original  : Email must nto be empty
18  Template  : Email must nto be empty
19  Key       : msg.email.invalid
20  --------------------------------
21
22  [2]
23  File      : src/main/java/com/example/resumeanalyser/model/userLogin.java
24  Line      : 20
25  Type      : ANNOTATION_MESSAGE
26  Pattern   : Annotation validation message
27  Original  : password has atLeast 6 characters and atMax 16 characters
28  Template  : password has atLeast 6 characters and atMax 16 characters
29  Key       : msg.password.invalid
30  --------------------------------
```

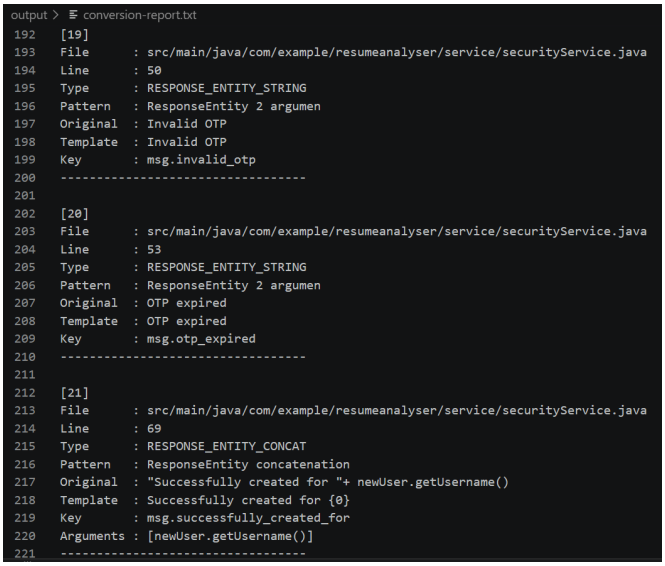
Gambar 2: Contoh laporan kandidat hasil deteksi

Hasil akhir eksternalisasi teks ditulis ke dalam berkas `messages.properties`. Untuk pesan berparameter, bagian dinamis diubah menjadi placeholder seperti `0` dan `1`, sehingga dapat digunakan oleh mekanisme i18n Spring Boot.

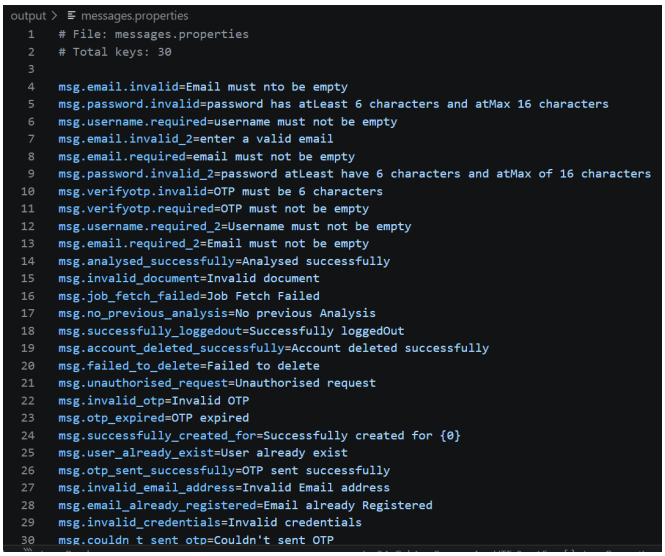
Berdasarkan hasil tersebut, program telah berhasil menjalankan alur utama, yaitu memindai source code, mendeteksi *hardcoded user-facing text*, menyaring string teknis, membangkitkan key i18n, dan menghasilkan berkas properties.

IV. KESIMPULAN

Penerapan *regular expression* untuk membantu proses konversi aplikasi Java Spring Boot menuju struktur multibahasa telah berhasil dirancang dan diuji pada proyek *Resume*



Gambar 3: Contoh laporan kandidat hasil deteksi



Gambar 4: Hasil pembangkitan berkas `messages.properties`

Analyser. Hasil pengujian menunjukkan bahwa regex mampu mendeteksi beberapa pola *hardcoded user-facing text*, seperti pesan pada `ResponseEntity`, pesan validasi pada annotation, serta pesan yang berkaitan dengan email. Teks yang berhasil terdeteksi dapat dieksternalisasi ke dalam berkas `messages.properties` melalui pembangkitan *key* i18n, sehingga *source code* menjadi lebih siap untuk menggunakan mekanisme internasionalisasi.

Meskipun begitu, pendekatan yang digunakan masih memiliki beberapa keterbatasan. Program telah menerapkan filtering dasar, namun belum dapat menjamin seluruh string terklasifikasi secara tepat. Pada kondisi tertentu, masih terdapat kemungkinan string teknis ikut terdeteksi sebagai teks yang perlu diterjemahkan. Sebaliknya, beberapa *hardcoded user-facing text* juga dapat tidak terdeteksi apabila pola penulisannya berbeda dari aturan regex yang telah didefinisikan.

Keakuratan dan kelengkapan hasil deteksi dapat ditingkatkan dengan memperketat mekanisme filtering dan menambahkan pola regex untuk menangani lebih banyak variasi penulisan pesan. Selain itu, berkas `messages.properties` yang dihasilkan dapat dikembangkan lebih lanjut dengan mengisi terjemahan untuk setiap *locale*, baik secara manual maupun melalui bantuan translation API. Selain itu, dapat ditambahkan rekomendasi *rewrite source code* yang diterapkan melalui proses *code review* atau mode *dry-run* agar perubahan yang dilakukan tetap aman dan tidak merusak logika aplikasi.

V. Ucapan Terima Kasih

Penulis menyampaikan puji syukur kepada Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga makalah ini dapat disusun dan diselesaikan dengan baik sebagai bagian dari tugas akhir mata kuliah IF2211 Strategi Algoritma Semester II Tahun 2025/2026. Penulis juga menyampaikan rasa terima kasih yang sebesar-besarnya kepada Prof. Dr. Ir. Rinaldi, M.T., Dr. Eng. Ir. Rila Mandala, M.Eng., Dr. Techn. M. Catur Zuhri Chandra, S.T, M.T., Dr. Fazat Nur Azizah, S.T, M.Sc., dan Tricya Esterina Widagdo, S.T, M.Sc. selaku dosen pengampu mata kuliah ini, atas ilmu, arahan, serta bimbingan yang telah diberikan selama perkuliahan. Selain itu, penulis berterima kasih kepada seluruh penulis jurnal, artikel, serta sumber referensi lainnya yang telah menjadi rujukan dalam penyusunan makalah ini. Dalam proses penyusunan makalah ini, penulis memanfaatkan teknologi kecerdasan buatan sebagai kakas untuk *brainstorming* ide serta perbaikan ejaan dan tata bahasa guna meningkatkan kejelasan penyampaian informasi. Akhir kata, penulis memohon maaf apabila terdapat kekeliruan dalam penulisan maupun penyampaian isi makalah ini.

Referensi

[1] Obregon, A., "How Spring Boot Configures Internationalization (i18n)," Medium, 2026. [Online]. Available: https://medium.com/@AlexanderObregon/how-spring-boot-configures-internationalization-i18n-6440bd597edc

[2] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun, "TranStrL: An Automatic Need-to-Translate String Locator for Software Internationalization," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, Cape Town, South Africa, 2010, pp. 555–558.

[3] K. Rathod, "How to Implement Internationalization (i18n) in Spring Boot: A Beginner-Friendly Guide to Multilingual Applications," *Medium*, 2026. [Online]. Available: https://medium.com/@kunal.rathod/spring-boot-i18n-build-multilingual-apps-with-messagesource-and-localeresolver-1172eb3fa152.

[4] Oracle, "Lesson: Regular Expressions," *The Java™ Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/regex/

[5] D. Jurafsky and J. H. Martin, "Regular Expressions, Text Normalization, Edit Distance," in *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2009, ch. 2.

[6] Spring Framework Documentation, "MessageSource," Spring Framework API Documentation. [Online]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/MessageSource.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Suryani Mulia Utami
13524042

LAMPIRAN

Dokumentasi program:
https://github.com/suryaniutaami/internationalization-java-springboot.git