Jurnal Pendidikan Teknologi Informasi (JUKANTI)
Volume (6) No (1) April 2023–eISSN : 2621-1467

PERANCANGAN *DOMAIN SPECIFIC LANGUAGE* DALAM *EMAIL TEMPLATE* DENGAN MENGGUNAKAN PYTHON

Andreas Gibran Nethanel Paat[a], Nina Setiyawati [b]

[ab]*Fakultas Teknologi Informasi Universitas Kristen Satya Wacana, Salatiga*

[a] 672019092@student.uksw.edu, [b] nina.setiyawati@uksw.edu

ABSTRAK

Domain Specific Language (DSL) merupakan bahasa pemrograman yang dirancang untuk suatu masalah yang spesifik. *Syntax* yang lebih sedikit dan keterbatasan pada suatu masalah tertentu, menjadikan DSL cocok untuk menyelesaikan pekerjaan yang membutuhkan ketepatan waktu. Salah satu pekerjaan yang sangat dibutuhkan dan perlu kecepatan serta ketepatan waktu adalah pengiriman *email*. Kebutuhan pengiriman *email* terus berkembang sehingga dibuatlah *template email* untuk lebih memudahkan pengiriman *email*. Akan tetapi muncul masalah: pengiriman *email* kepada banyak penerima dengan *template* yang berbeda ataupun menggunakan *template* yang sama tetapi dengan isi *email* yang berbeda, membutuhkan waktu yang lebih lama. Perlu dibuat suatu program yang dapat mempercepat dan meningkatkan efisiensi dalam pengiriman *email*. Pada penelitian ini, dirancang sebuah DSL untuk mengatasi permasalahan tersebut. DSL dibangun menggunakan Python SLY dengan metode Waterfall. Penelitian menghasilkan DSL yang dapat digunakan agar pengiriman *email* lebih cepat dan efisien.

Kata kunci *: Domain Specific Language, Python, Sly, Email*

ABSTRACT

A Domain-Specific Language (DSL) is a programming language designed for a specific problem. Less syntax and limitations on a particular issue, make DSL suitable for completing work that requires timeliness. One of the jobs that highly needs and requires speed and timeliness is sending emails. The need for sending e-mails is growing, so an e-mail template has been created to make it easier to send e-mails. However, a problem arises: sending emails to multiple recipients with different templates or using the same template but with other email contents, takes longer. It is necessary to create a program that can speed up and improve efficiency in sending e-mails. In this research, a DSL was designed to overcome this problem. DSL is built using Python SLY with the Waterfall method. Research produces a DSL that can be used to send emails faster and more efficiently.

Keywords: *Domain Specific Language, Python, Sly, Email*

1. PENDAHULUAN

Domain-Specific Language (DSL) adalah bahasa pemrograman yang dirancang untuk menyelesaikan suatu masalah spesifik. DSL memiliki *syntax* yang lebih sedikit untuk dipelajari dibanding dengan *general-purpose language* atau bahasa pemrograman pada umumnya seperti bahasa Python. DSL dibuat terbatas hanya untuk suatu masalah, agar dapat meningkatkan fokus terhadap permasalahan tersebut [1].

Dalam penggunaannya, DSL dapat mempercepat suatu pekerjaan sehingga membuat pekerjaan menjadi lebih efisien. Dalam sebuah penelitian yang dilakukan

oleh Kosar, dkk (2018), menunjukan tingkat efisiensi *user* meningkat ketika menggunakan DSL dibanding *general-purpose language* dalam pemahaman akan program dan saat menggunakan *Integrated Development Environment* (IDE) [2]. Hal ini membuat DSL sesuai untuk menyelesaikan pekerjaan yang membutuhkan kecepatan dan ketepatan waktu. Salah satu pekerjaan yang membutuhkan kecepatan dan ketepatan waktu adalah pengiriman *email template*.

Email template adalah suatu *template* atau format *email* yang telah ditentukan sebelumnya [3] [4]. *Template* biasanya adalah *file hypertext markup language* atau HTML. Penggunaan *template* bertujuan untuk meningkatkan efisiensi dari proses pengiriman *email*, dikarenakan pengirim tidak perlu membuat *email* baru ataupun desain baru tiap kali ingin mengirimkan *email*. *Email template* biasanya digunakan untuk mengirimkan *email* kepada banyak penerima dengan format yang sama. Masalah yang muncul dari pengiriman *email template* yaitu pengirim harus mengirimkan *email* dengan *template* yang berbeda kepada banyak penerima dalam waktu yang cepat. Selain itu pengirim juga harus mengirim *email* dengan *template* yang sama kepada banyak penerima tapi memiliki perbedaan dalam isi pesannya. Sedangkan pengiriman *email* dengan dua skenario tersebut membutuhkan waktu yang tidak sedikit. Oleh karena itu, perlu dibuat suatu program yang dapat mempercepat dan meningkatkan efisiensi dalam pengiriman *email.*

Berdasarkan latar belakang diatas, dilakukan penelitian berupa perancangan DSL sebagai bahasa pemrograman untuk konfigurasi *email*. Konfigurasi ditulis pada satu *config file* dengan bahasa yang dibuat dan dijalankan dengan Python. DSL akan dibangun menggunakan bahasa pemrograman Python dengan *library* SLY.

Adapun DSL adalah bahasa pemrograman dengan *syntax* yang terbatas yang didesain untuk suatu domain spesifik atau untuk menyelesaikan masalah khusus, contohnya seperti *Cascading Styles Sheets* (CSS) dan *Structured Query Language* (SQL) [1] [2].

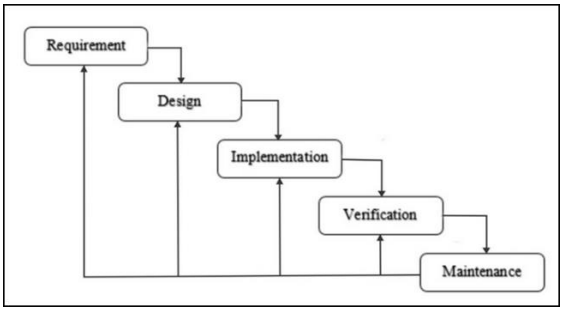
Python adalah salah satu bahasa pemrograman *high-level*, *general-purpose* yang telah digunakan secara luas. Bahasa pemrograman Python mendukung banyak paradigma pemrograman seperti *object-oriented*, *imperative* dan *functional programming* atau *procedural styles* [5].

SLY adalah *library* Python yang mengimplementasikan *lex* dan *yacc* untuk menulis *parsers* dan *compilers* [6]. SLY dipilih karena *library* tersebut dapat melakukan *parsing* dari format dan *grammar* bahasa yang dibuat oleh *user* sendiri. *User* dapat merancang format yang dipakai untuk bahasa yang dibuat dan aksi yang akan dilakukan untuk setiap *grammar*.

Penelitian ini bertujuan untuk merancang dan membangun sebuah DSL yang dapat digunakan untuk mengkonfigurasi pengiriman *email template* sehingga pengiriman *email* menjadi lebih efisien.

2. METODE PENELITIAN

Metode yang dipakai dalam perancangan sistem ini adalah Waterfall. Waterfall merupakan metode pengembangan perangkat lunak dengan pendekatan sistematis dan berurutan dimana setiap tahapan akan dilakukan setelah selesai melakukan tahapan sebelumnya [7].



Gambar 1. Tahapan Metode Waterfall

Tahapan penelitian pada Gambar 1, dijelaskan sebagai berikut:

1. Tahap *requirement* dilakukan dengan mengumpulkan segala informasi yang dibutuhkan terkait kebutuhan *email template*, semua fungsi yang diperlukan dan batasan-batasan pada program nantinya.
2. Tahap *design* adalah tahap dimana peneliti membuat desain program DSL yaitu alur program, *software* yang dibutuhkan dan arsitektur secara keseluruhan. Pada tahap ini peneliti membuat desain *file* konfigurasi untuk menulis DSL dan juga membuat desain *grammar* untuk DSL sehingga memenuhi kebutuhan yang telah dijabarkan.
3. Tahap *implementation* adalah tahap dimana peneliti melakukan pengembangan program berdasarkan desain yang telah dibuat. DSL dikembangkan menggunakan bahasa pemrograman Python dengan *library* SLY. Program akan dikembangkan per modulnya. Setelah itu akan diuji per modul terkait fungsionalitasnya.
4. Tahap *verification* adalah tahap dimana dilakukan pengujian keseluruhan terhadap program untuk memastikan program memenuhi persyaratan yang dibuat di awal. Program akan diuji untuk integrasi semua modul sehingga peneliti dapat melihat hasil dari integrasi modul dan mencatat kekurangan dari program secara utuh. Pengujian fungsionalitas program secara keseluruhan digunakan dengan metode *Black Box Testing*.
5. Tahap *maintenance* merupakan tahap akhir, dimana peneliti akan menjalankan program yang sudah jadi dan melakukan pemeliharaan. Pemeliharaan termasuk perbaikan akan kesalahan dari program yang tidak ditemukan pada tahap sebelumnya.

3. HASIL DAN PEMBAHASAN

Penelitian ini menghasilkan suatu DSL yang dapat digunakan untuk menulis konfigurasi pengiriman suatu *email template*. *File* konfigurasi berisi informasi yang dibutuhkan dalam pengiriman *email* termasuk *template* yang digunakan dan parameter yang akan dimasukkan ke dalam *template*. *Template* dari *email* ditulis dengan HTML. *File* konfigurasi yang sudah ditulis dijalankan menggunakan aplikasi *web* yang dibuat dengan Python.

Kode Program 1. *Grammar* DSL

```
script   : script statement
         | statement

statement: SOURCE_EXCEL ASSIGN word
         | FILENAME ASSIGN word
         | TEMPLATE ASSIGN word
         | RECEIVER ASSIGN recipient
         | RECEIVER_CC ASSIGN
recipient
         | RECEIVER_BCC ASSIGN
recipient
         | SUBJECT ASSIGN word
         | MULTI_RECIP_PARAM ASSIGN
words
         | ATTACHMENT ASSIGN words
         | PARAMETER expr
         | PARAMETER

expr     : expr term
         | term

term     : ID ASSIGN words

recipient: recipient ',' emails
         | emails

emails   : EMAIL STRING
         | EMAIL
         | word

words    : word ',' word
         | word

word     : STRING
         | ID
```

Kode Program 1 merupakan rancangan *grammar* untuk DSL yang akan dibuat. Perancangan DSL dimulai dengan merancang *grammar*, yang akan menjadi batasan dan juga aturan dalam pembuatan DSL [8] [9]. Untuk mengimplementasikan *grammar* yang telah dirancang, maka dibuat *parser* untuk melakukan *parsing*. *Parser* terdiri dari dua *class* yaitu *EmLexer* dan *EmParser*.

Kode Program 2. *Class EmLexer*

```
class EmLexer(Lexer):
    tokens = { STRING, EMAIL, ID,
ASSIGN, FILENAME, TEMPLATE,
PARAMETER,
               RECEIVER, RECEIVER_CC,
RECEIVER_BCC, MULTI_RECIP_PARAM,
SUBJECT,
               ATTACHMENT,
SOURCE_EXCEL }
 # Tokens
    literals = {',','[',']'}

    STRING =
r'"{3}([\s\S]*?)"{3}|""|"(.+?)"'
    EMAIL = r'\b[A-Za-z0-9._%+-]+@[A-
Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    FILENAME =r"filename"
    TEMPLATE =r"template"
    PARAMETER =r"PARAMETER"
    RECEIVER =r"receiver"
    RECEIVER_CC =r"receiver_cc"
    RECEIVER_BCC =r"receiver_bcc"
    MULTI_RECIP_PARAM
=r"multi_recip_param"
    SUBJECT =r"subject"
    ATTACHMENT = r"attachment"
    SOURCE_EXCEL = r"source_excel"
    ID = r'[a-zA-Z_][a-zA-Z0-9_-]*'

    # Special symbols
    ASSIGN = r'='
```

Pada Kode Program 2 terdapat *class EmLexer*. *Class* ini berfungsi untuk memecah teks masukan menjadi token-token, yang ditentukan menggunakan aturan *regular expression* [10] [11]. Token-token dan aturannya dibuat merujuk pada *grammar* yang telah dirancang sehingga dapat membatasi inputan dalam

DSL yang dirancang.

Kode Program 3. *Class EmParser*

```
class EmParser(Parser):
    tokens = EmLexer.tokens

    @_('script statement')
    def script(self, p):
        return p.script | p.statement

    @_('statement')
    def script(self, p):
        return p.statement

    @_('SOURCE_EXCEL ASSIGN words')
    def statement(self, p):
        dictr = {'source_excel' :
p.words}
        return dictr

    @_('FILENAME ASSIGN word')
    def statement(self, p):
        dictr = {'filename' : p.word}
        return dictr

    @_('TEMPLATE ASSIGN word')
    def statement(self, p):
        dictr = {'template' : p.word}
        return dictr
```

Kode Program 3 merupakan *class EmParser* yang berfungsi mengenali *syntax* dari DSL yang dibuat dan sesuai dengan *grammar*. Dalam *class* ini ditulis fungsi-fungsi yang mewakili tiap *grammar* yang telah dirancang beserta hasil atau tindakan jika sesuai [11]. *Class EmParser* menghasilkan sebuah *dictionary* yang menampung hasil terjemahan dari DSL.

Kode Program 4. Fungsi *one_recip*

```
def one_recip(dictr, data,
template_params):
    if "parameter" not in
dictr.keys():
        result = send(dictr, data)
        return result

    if dictr['name'] == True:
        dictr['parameter']['name'] =
dictr['receiver'][0][1]
        dictr['receiver'] =
dictr['receiver'][0][0]

    for param in dictr['parameter']:
        if param in template_params:

template_params.remove(param)
            if
type(dictr['parameter'][param]) ==
list:
                return ({'status' :
'danger',
                         'msg' : f'The
{param} parameter has two or more
value'})
            else:
                old = "{ "+param+" }"
                data =
data.replace(old,
dictr['parameter'][param])

    if len(template_params) > 0:
        return ({'status' : 'danger',
                 'msg' : f'Missing
parameter : {",
".join(template_params)}'})

    result = send(dictr, data)
    return result
```

Kode Program 4 merupakan fungsi yang digunakan untuk menulis hasil dari *parser* ke dalam teks *template*. Fungsi ini menerima tiga parameter yaitu *dictionary*

dari *parser*, teks dari *template* dan *param* yang ada dalam *template*. Fungsi ini mengembalikan teks *email*.

Kode Program 5. Fungsi *send*

```
def send(dictr, template):
        email = dbc.select_db("select email, password from sender limit 1;")
        sender_email = email[0][0]
        sender_pass = email[0][1]
        msg = MIMEMultipart('alternative')
        msg['Subject'] = dictr['subject']
        msg['From'] = sender_email
        msg['To'] = dictr['receiver']
        if 'receiver_cc' in dictr.keys():
            msg['Cc'] = dictr['receiver_cc']
        if 'receiver_bcc' in dictr.keys():
            msg['Bcc'] = dictr['receiver_bcc']

        html = MIMEText(template, 'html')

        msg.attach(html)
        if "attachment" in dictr:
            if dictr['attachment'] != "":
                if dictr['attachment'] != list:

dictr['attachment'] = [dictr['attachment']]

                for addr in dictr['attachment']:
                    filename = basename(addr)

attachment_package = MIMEApplication(Path(addr).read_bytes())

attachment_package.add_header('Content-Disposition', "attachment; filename= " + filename)

msg.attach(attachment_package)

        session = smtplib.SMTP('smtp.gmail.com', 587)
        session.starttls()
        session.login(sender_email, sender_pass)
        session.send_message(msg)
        session.quit()

        return ({'status' : 'success', 'msg' : 'Email Sent'})
```

Kode Program 5 merupakan fungsi yang berfungsi untuk mengirim *email*. Fungsi ini menerima dua parameter yaitu *dictionary* hasil *parser* dan hasil teks *email* dari fungsi pada Kode Program 4.

Kode Program 6. Model DSL

```
source_excel = "file.xlsx" [optional]
template = CodeTemplate

receiver =
    email1@domain.com "namaPertama",
    email2@domain.com "namaKedua",

subject = "Subect dari email"
multi_recip_param = param1

PARAMETER
    param1 = "valueParam1","value2Param1"
    param2 = "valueParam2"
    param3 = "valueParam3"

attachment = "alamat\attachment\file.txt"
```

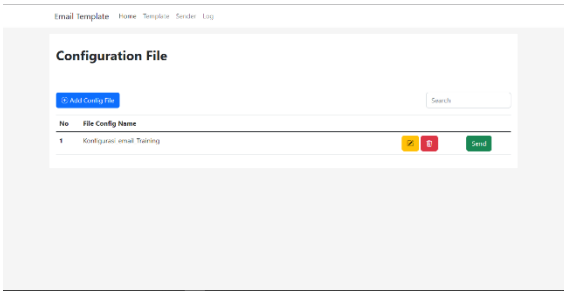
Kode Program 6 merupakan model DSL yang telah dirancang. DSL ini akan dipakai untuk menulis konfigurasi pengiriman *email* pada *file* CFG. Aturan penulisan DSL berupa pasangan antara *key* dan *value* yang dipisah dengan tanda '='. *Key* disini merupakan bagian-bagian yang diperlukan dalam pengiriman *email*, sedangkan *value* adalah nilai yang akan dimasukan pada bagian *email* tersebut. Pada baris ke-8 terdapat *multi_recip_param* berisi parameter yang *value*-nya akan dibedakan untuk tiap penerima. Parameter pada baris ke-10 hingga 14 merupakan *value* yang akan di subtitusi ke dalam badan *email*. Jumlah *value* untuk parameter yang ada pada *multi_recip_param* disesuaikan dengan jumlah *receiver*.

Kode Program 7. *Template Email*

```
1  <html>
2      <body>
3          <p>
4              Lorem ipsum dolor sit amet,
5  consectetur adipiscing elit. <br />
6          </p>
7          <p>
8              { parameter1 }<br />
9              { parameter2 }<br />
10             { parameter3 }<br />
11         </p>
12         <p>
13             Suspendisse accumsan nunc
14 volutpat, cursus eros sed, facilisis
15 turpis. Mauris scelerisque at velit id
16 tristique.<br />
17         </p>
18     </body>
19 </html>
```

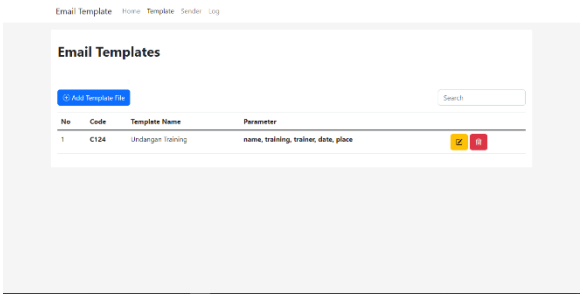
Kode Program 7 merupakan model penulisan *template* untuk *email*. Penulisan *file template* menggunakan HTML. Parameter dalam *template* ditulis dengan format '{ parameter }'.

File konfigurasi yang sudah dibuat menggunakan DSL dijalankan dengan melalui aplikasi *web*. Pembuatan aplikasi *web* bertujuan agar *user* memiliki antarmuka untuk menjalankan konfigurasi. Aplikasi *web* dibuat dengan menggunakan *framework* Python Flask. Flask merupakan *microframework* yang ditulis dengan bahasa Python untuk membuat *web* [12]. Flask dipilih karena bersifat ringan dan mudah modifikasi sesuai keinginan [13]. Tampilan dari aplikasi seperti pada Gambar 2, Gambar 3, Gambar 4 dan Gambar 5.



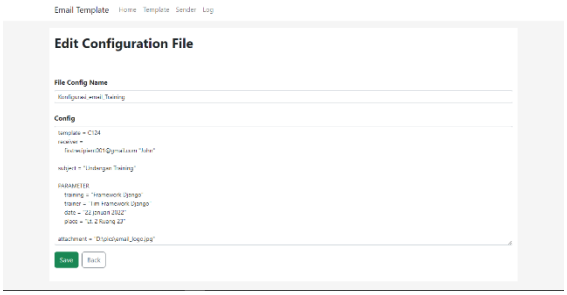
Gambar 2. Tampilan *List Configuration*

Gambar 2 merupakan tampilan dari *list file* konfigurasi yang telah dibuat. *File config* dapat dijalankan, ditambahkan dan dihapus pada tampilan ini. Pengguna dapat masuk ke tampilan edit dengan mengeklik tombol *edit*. Halaman *edit template* ditunjukan pada Gambar 4.

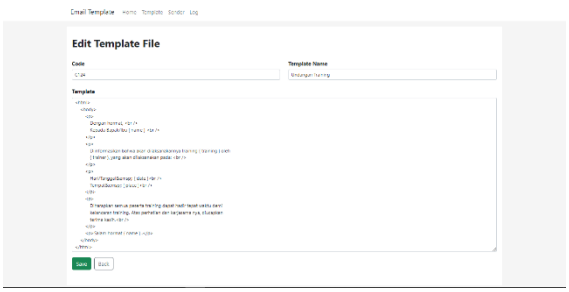


Gambar 3. Tampilan *List Template*

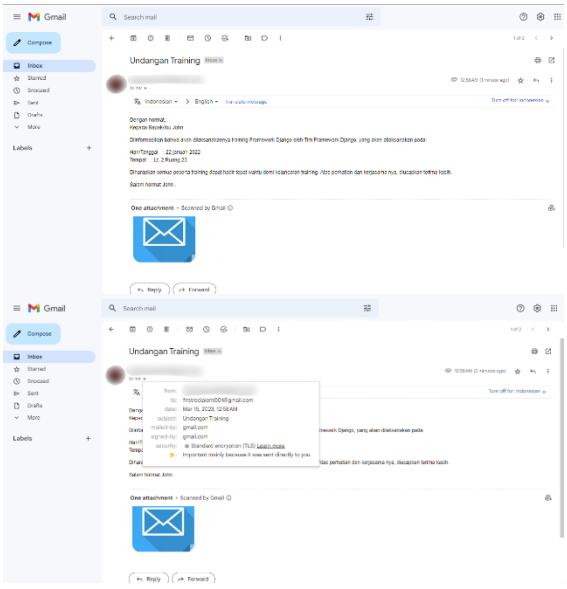
Gambar 3 adalah tampilan dari *list template* yang telah dibuat. *Template* dapat ditambahkan dan dihapus pada tampilan ini. Pengguna dapat masuk ke tampilan *edit* dengan mengeklik tombol *edit*. Halaman *edit template* ditunjukan pada Gambar 5.



Gambar 4. Tampilan *Edit Configuration*

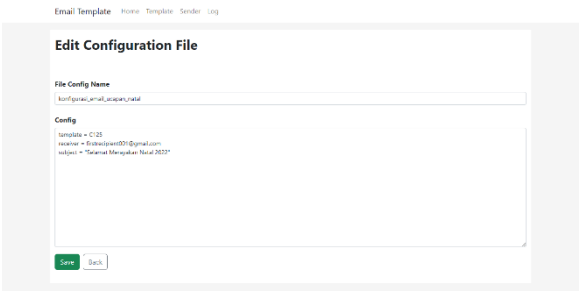


Gambar 5. Tampilan *Edit Template*

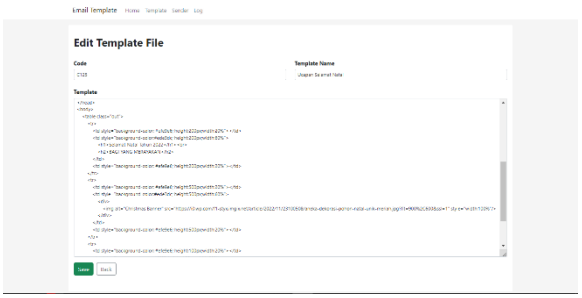


Gambar 6. Hasil Pengiriman *Email*

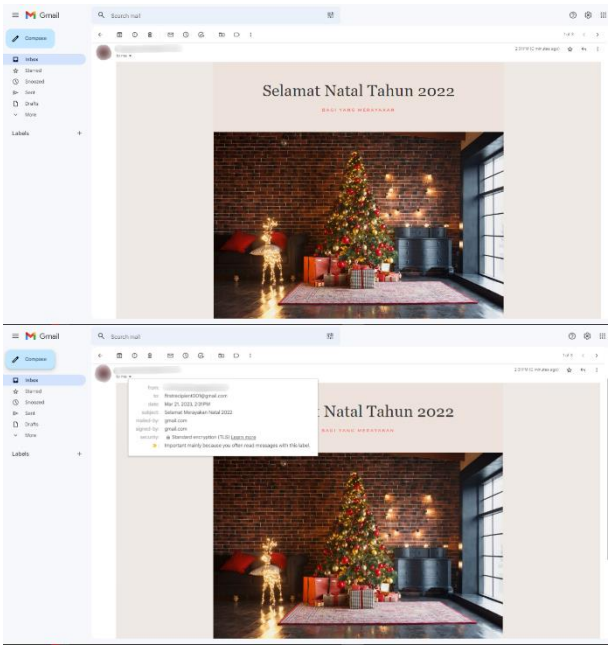
Gambar 6 merupakan tampilan hasil dari pengiriman *email* setelah menjalankan konfigurasi pada Gambar 4 dengan menggunakan *template* pada Gambar 5.



Gambar 7. Konfigurasi *Email* Gambar



Gambar 8. *Template Email* Gambar



Gambar 9. Hasil Pengiriman *Email* Gambar

Pada Gambar 7 terdapat konfigurasi untuk pengiriman *email* dengan *template* yang berupa gambar. *Template* ditunjukan pada Gambar 8. Hasil pengiriman dari *email* tersebut ditunjukan pada gambar 9. Kostumisasi *email* tersebut dapat dilakukan karena *template* menggunakan HTML *file*, sehingga lebih mudah memasukkan gambar dan merubah warna ataupun ukuran dari *font* didalamnya [14].

Dengan menggunakan DSL ini dalam penulisan konfigurasi pengiriman *email*, pengiriman *email* kepada banyak penerima dengan *template* berbeda maupun dengan *template* yang sama tetapi isi pesannya berbeda, tidak perlu dilakukan secara manual. Hal ini dikarenakan pengguna cukup menuliskan konfigurasinya untuk dijalankan. Untuk pengiriman kepada banyak penerima dengan *template* berbeda, pengguna dapat menulis beberapa konfigurasi untuk beberapa *template* dan dijalankan saat ingin dikirim. Untuk pengiriman dengan *template* yang sama tetapi memiliki isi *email* yang berbeda, pengguna cukup menulis satu *file* konfigurasi dengan menambahkan *multi_recip_param*. Dengan ini pengiriman *email* dapat dilakukan dengan lebih efisien.

Pengujian pada sistem dilakukan dengan *Black Box testing*. *Black box testing* merupakan pengujian yang hanya melibatkan *input* dan *output* tanpa melihat ke dalam kode program [15]. Pengujian ini dilakukan memvalidasi fungsi apakah dapat berfungsi dengan baik. Hasil pengujian dapat dilihat pada Tabel 1.

Tabel 1. Hasil Pengujian *Black Box*

Pengujian	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
Run file config dengan satu penerima	*Email* diterima oleh penerima	*Email* diterima oleh penerima	Valid
Run file config dengan lebih dari satu penerima	*Email* diterima oleh semua penerima	*Email* diterima oleh semua penerima	Valid
Run file config dengan parameter	Isi *email yang* diterima oleh penerima sesuai dengan parameter	Isi *email yang* diterima oleh penerima sesuai dengan parameter	Valid
Run file config dengan *multi_recip_param*	E*mail* diterima oleh semua penerima dengan isi yang berbeda	E*mail* diterima oleh semua penerima dengan isi yang berbeda	Valid
Run file config dengan *attachment*	E*mail* diterima oleh penerima dengan *attachment*	E*mail* diterima oleh penerima dengan *attachment*	Valid
Run file config dengan *source_excel*	E*mail* diterima oleh penerima dengan sesuai dengan data pada *file* excel	E*mail* diterima oleh penerima dengan sesuai dengan data pada *file* excel	Valid

Setelah melakukan pengujian *Black Box*, dilakukan pengujian waktu antara pengiriman *email* dengan *template* secara manual dan menggunakan konfigurasi dengan DSL. Pengujian dilakukan dengan melibatkan 10 orang penguji dengan skenario berikut:

1. Melakukan pengiriman *email* menggunkan *template* yang berbeda kepada beberapa penerima.
2. Melakukan pengiriman *email*

menggunakan *template* yang sama kepada beberapa penerima tetapi dengan nilai parameter yang berbeda untuk tiap penerima.

Pengiriman dilakukan kepada 5 penerima. Hasil pengujian berupa rata-rata waktu yang dibutuhkan melalui cara manual dan konfigurasi DSL. Hasil dapat dilihat pada Tabel 2.

Tabel 2. Hasil Pengujian Waktu

Skenario	Cara Manual	Konfigurasi DSL
Melakukan pengiriman *email* menggunkan *template* yang berbeda (4 *template*) kepada beberapa penerima	3 menit 49 detik	2 menit 10 detik
Melakukan pengiriman *email* menggunakan *template* yang sama kepada beberapa penerima tetapi dengan nilai parameter yang berbeda untuk tiap penerima	4 menit 28 detik	2 menit 16 detik

Dari Tabel 2 dapat dilihat bahwa waktu yang dibutuhkan saat menggunakan konfigurasi DSL lebih sedikit dibanding cara manual.

4. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan, dapat disimpulkan bahwa DSL untuk pengiriman *email* menggunakan *template* membantu pengiriman *email* lebih cepat dan efisien. Hal ini dikarenakan untuk pengiriman email ke banyak penerima dengan *template* berbeda ataupun *template* yang sama dengan isi yang berbeda, pengirim *email* cukup menulis konfigurasi untuk setiap *template* dan dijalankan untuk dikirim.

Saran untuk penelitian selanjutnya adalah untuk mengembangkan DSL *email* ini agar dapat menangani pengiriman *email* yang kompleks seperti menambahkan tabel dengan data dari *database*.

DAFTAR PUSTAKA

[1] M. A. Halomoan, A. P. Kharisma and Marji, "Pengembangan Domain Specific Language Untuk Aplikasi CRUD Berbasis Web," Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, vol. 5, pp. 34-41, 2021.

[2] T. Kosar, S. Gaberc and J. C. Carver, "Program comprehension of domain-specificand general-purpose languages: replication of a family of experiments using integrated development environments," Empirical software engineering, vol. 23, p. 2734–2763, 2018.

[3] Clarivate Analytics Company, Email Templates Guide, Web of Science Group, 2019.

[4] A. Qashqari, D. Alhbshi, A. Fatmah, G. Hadeel and A. Asia, "Electronic Mail Security," International J. of Comput. Science and Inf. Security (IJCSIS), pp. 46-53, 2020.

[5] S. A. Abdulkareem and A. J. Abboud, "Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics)," IOP Conference Series: Materials Science and Engineering, pp. 1-9, 2021.

[6] D. Beazley, "SLY (Sly Lex Yacc)," 24 November 2018. [Online]. Available:

https://sly.readthedocs.io/_/downloads/en/latest/pdf/. [Accessed 17 11 2022].

[7] A. A. Wahud, "Analisis Metode Waterfall Untuk Pengembangan Sistem Informasi," Jurnal Ilmu-ilmu Informatika dan Manajemen (STMIK), pp. 1-5, 2020.

[8] V. Natali and P. Alfandian, "Analisis dan Perancangan Domain Specific Language untuk Data Generator pada Relational Database," J. Masy. Inf. Unjani, vol. 3, no. 1, pp. 64-73, 2019.

[9] L. Bambaci, F. Boschetti and R. D. Gratta, "a Domain-specific Languagefor the Encoding of the critical Apparatus," International Journal of Information Science & Technology, vol. 3, no. 5, pp. 26-37, 2019.

[10] M. Harahap, "Perancangan Perangkat Lunak Teks Editor Bahasa C Menggunakan Metode Lexical Analyzer," Buletin Ilmiah Informatika Teknologi, vol. 1, no. 1, pp. 8-14, 2022.

[11] F. Rahman, A. Ramadhan and Y. Ikhwani, Rancangan dan Teknik Kompilasi, Banjarmasin: Universitas Islam Kalimantan Muhammad Arsyad Al Banjari, 2022.

[12] R. Irsyad, "Penggunaan Flask untuk Pemula," 2018. [Online]. Available: 10.31219/osf.io/t7u5r.

[13] D. F. Ningtyas and N. Setyawati, "Flask Framework Implementation in Development Purchasing Approval Request Application," J. Janitra Inform. dan Sist. Inf., vol. 1, no. 1, pp. 19-34, 2021.

[14] Wahyudi, Pemrograman WEb: HTML dan CSS, Purbalingga: EUREKA MEDIA AKSARA, 2022.

[15] R. Parlika, T. A. Nisaa, S. M. Ningrum and B. A. Haque, "Studi Literatur Kekurangan dan Kelebihan Pengujian Black Box," TEKNOMATIKA, vol. 10, no. 2, pp. 131-140, 2020.