

SKRIPSI

PERKAKAS COMMAND LINE KIRI



Alfred Aprianto Liaunardi

NPM: 6181801014

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2022

DAFTAR ISI

DAFTAR ISI	iii
DAFTAR GAMBAR	v
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 <i>Command Line</i>	5
2.1.1 <i>Command Line Interface</i> dan <i>Graphical User Interface</i>	5
2.1.2 <i>Command Line</i> di Linux	6
2.1.3 <i>Command Line</i> di Windows	7
2.2 KIRI	10
2.2.1 Tampilan	11
2.2.2 API ¹	12
2.3 Fungsi dan <i>Library</i> Bahasa C	17
2.3.1 getopt [1]	18
2.3.2 libcurl [2]	21
2.3.3 cJSON ²	24
2.3.4 CMake [3]	29
3 ANALISIS	35
3.1 Analisis Aplikasi Sejenis	35
3.1.1 Chrome Web Store Item Property CLI	35
3.1.2 iTunes Search API	37
3.1.3 Uber CLI	38
3.1.4 Google Maps Direction CLI	40
3.2 Analisis API KIRI	41
3.2.1 Search Place	41
3.2.2 Routing	42
3.3 Analisis Perkakas yang Akan Dibuat	43
3.3.1 Analisis Fitur Perkakas	43
3.3.2 Analisis Use Case	45
4 PERANCANGAN	49
4.1 Rancangan Diagram	49
4.1.1 Mencari lokasi menggunakan kata kunci pencarian	49
4.1.2 Mencari rute dengan angkot menggunakan <i>latitude</i> dan <i>longitude</i> lokasi	50

4.1.3	Mencari rute dengan angkot menggunakan kata kunci pencarian lokasi . . .	51
5	PENGUJIAN DAN EKSPERIMENT	55
5.1	Lingkungan Implementasi	55
5.1.1	Lingkungan Perangkat Keras	55
5.1.2	Lingkungan Perangkat Lunak	55
5.2	Implementasi	55
5.2.1	Instalasi	56
5.2.2	Implementasi kode	56
5.2.3	Pengujian	56
6	KESIMPULAN	57
DAFTAR REFERENSI		59
A	KODE PROGRAM	61

DAFTAR GAMBAR

1.1	Tampilan halaman web KIRI	1
2.1	Dua jenis tampilan perangkat lunak	5
2.2	Baris <i>shell prompt</i> terminal di sistem operasi Linux.	6
2.3	Tampang kedua antarmuka <i>command line</i> bawaan di sistem operasi Windows.	8
2.4	Tampilan awal halaman web KIRI	11
2.5	Tampilan akhir halaman web KIRI	12
2.6	Halaman web <i>API Keys</i> KIRI.	13
2.7	Penggunaan API KIRI untuk layanan pencarian lokasi	14
2.8	Penggunaan API KIRI untuk layanan pencarian rute	17
2.9	Penggunaan API KIRI untuk layanan <i>smart direction</i>	18
2.10	Tampilan aplikasi cmake-gui	31
2.11	Tampilan aplikasi ccmake	31
3.1	Contoh penggunaan perkakas Chrome <i>Web Store Item Property CLI</i>	36
3.2	Contoh penggunaan perkakas <i>iTunes Search API</i>	38
3.3	Contoh penggunaan perkakas Uber CLI (<i>time</i>)	39
3.4	Contoh penggunaan perkakas Uber CLI (<i>price</i>)	39
3.5	Contoh penggunaan perkakas Google <i>Maps Direction CLI</i>	41
3.6	Diagram <i>use case</i> perkakas yang akan dibangun	45
4.1	Diagram <i>use case</i> perkakas yang akan dibangun	49
4.2	Diagram <i>use case</i> perkakas yang akan dibangun	50
4.3	Diagram <i>use case</i> perkakas yang akan dibangun	51
4.4	Diagram <i>use case</i> perkakas yang akan dibangun	52
4.5	Diagram <i>use case</i> perkakas yang akan dibangun	53
4.6	Diagram <i>use case</i> perkakas yang akan dibangun	53

1

BAB 1

2

PENDAHULUAN

3 1.1 Latar Belakang

- 4 Project KIRI¹ (akan disingkat sebagai KIRI dalam dokumen ini) adalah sebuah perangkat lunak ber-
5 basis web yang dibuat untuk membantu mengurangi efek dari kemacetan. KIRI mengurangi dampak
6 kemacetan dengan membantu penggunanya, baik masyarakat maupun turis, dalam menggunakan
7 salah satu sarana transportasi umum yang ada di Indonesia, yaitu angkutan kota (angkot). Cara
8 KIRI mempermudah penggunaan angkot adalah dengan menunjukkan rute yang akan ditempuh,
9 beserta langkah-langkah yang harus dilakukan oleh pengguna yang ingin berpergian dari satu
10 titik ke titik lain, mulai dari seberapa jauh pengguna harus berjalan untuk menaiki angkot yang
11 bersangkutan, di mana pengguna harus naik atau turun, seberapa jauh lagi pengguna harus berjalan
12 sampai ke titik tujuan, dan seberapa lama estimasi waktu perjalanan yang akan ditempuh. Untuk
13 kebutuhan pembuatan perangkat lunak yang memanfaatkan fitur dari KIRI, tersedia juga REST
14 API KIRI yang dapat digunakan secara praktis. Adapun tampilan dari halaman web ini dapat
15 dilihat di gambar 1.1.



Gambar 1.1: Tampilan halaman web KIRI, yang menunjukkan rute dari Cihampelas Walk ke Universitas Katolik Parahyangan.

- 16 Sementara itu, dalam komputer, salah satu dari sekian banyak tipe perangkat lunak adalah
17 perangkat lunak *command line*. *Command line (command line interpreter*, atau *command line*

¹<https://projectkiri.id>

1 *interface*) adalah sebuah perangkat lunak berupa sebuah kotak/*window* yang memuat teks berupa
2 perintah-perintah,² yang menerima masukan dari pengguna dan menjalankannya.[4] Perintah-
3 perintah ini hanya berupa gabungan dari teks and simbol-simbol berupa karakter, tanpa ada
4 tambahan gambar grafis apapun. Singkatnya, tipe perangkat lunak ini bukan merupakan tipe yang
5 paling indah untuk dilihat oleh para pengguna, tetapi jika digunakan dengan tepat, maka jenis
6 perangkat lunak ini bisa menyuruh komputer untuk melakukan banyak sekali perintah-perintah
7 dengan sangat cepat dan sangat efektif.

8 Pada skripsi ini akan dibuat sebuah perangkat lunak berupa perkakas *command line* (*command*
9 *line tool*) yang dapat menjalankan fungsi-fungsi API dari KIRI. Perangkat lunak ini, seperti jenisnya,
10 akan dibuat murni sebagai perkakas yang dijalankan dari *command line* (terminal, cmd, PowerShell,
11 dll.), dan tampilan akhir dari perangkat lunak akan berupa *command line interface* tanpa tambahan
12 *graphical user interface*. Keseluruhan dari perangkat lunak ini akan dibangun dalam bahasa C.

13 1.2 Rumusan Masalah

- 14 1. Bagaimana membangun perkakas *command line* yang dapat mengimplementasikan fitur-fitur
15 API KIRI dalam bahasa C?
16 2. Bagaimana integrasi perkakas *command line* KIRI dapat dilakukan dengan perkakas-perkakas
17 *command line* lainnya di Linux?

18 1.3 Tujuan

- 19 Batasan masalah dalam skripsi ini adalah sebagai berikut:
20 1. Membangun perkakas *command line* yang dapat mengimplementasikan fitur-fitur API KIRI
21 dalam bahasa C.
22 2. Melakukan integrasi perkakas *command line* KIRI dengan perkakas-perkakas *command line*
23 lainnya di Linux.

24 1.4 Batasan Masalah

- 25 Batasan masalah dalam skripsi ini adalah sebagai berikut:
26 1. Perangkat lunak dibuat murni dalam bentuk CLI, tanpa tambahan GUI.
27 2. Perangkat lunak yang dibuat tidak menyelesaikan batasan (lokasi tidak terdeteksi, rute tidak
28 berhasil ditemukan, dsb.) yang sudah sejak awal terdapat dalam KIRI.

29 1.5 Metodologi

- 30 Metodologi yang akan diikuti dalam skripsi ini adalah sebagai berikut:
31 1. Melakukan studi dan eksplorasi terhadap fungsi-fungsi yang dimiliki perangkat lunak KIRI
32 serta cara implementasi API KIRI.
33 2. Melakukan analisis dan desain perangkat lunak yang akan dibangun.

²Ubuntu Tutorials - The Linux command line for beginners: 3. Opening a Terminal

- 1 3. Melakukan studi dan eksplorasi terhadap seluruh kemungkinan *library-library* yang memenuhi
2 spesifikasi dalam pembuatan perangkat lunak, berdasarkan analisis dan desain yang telah
3 dilakukan sebelumnya.
- 4 4. Melakukan analisis kebutuhan fitur-fitur perangkat lunak dan melakukan eksplorasi *library*
5 yang dapat digunakan dan memenuhi spesifikasi dalam pembuatan perangkat lunak.
- 6 5. Membangun perangkat lunak berdasarkan rancangan yang sudah dibuat, dengan megimple-
7 mentasikan seluruh modul dan *library* yang telah ditentukan di tahap sebelumnya dalam
8 bahasa C.
- 9 6. Melakukan pengujian fungsional, perbaikan *bug*, serta rekomendasi perbaikan berdasarkan
10 pengujian yang sudah dilakukan.
- 11 7. Menyelesaikan pembuatan dokumen-dokumen yang berkaitan, seperti dokumen skripsi dan
12 dokumentasi perangkat lunak.

13 **1.6 Sistematika Pembahasan**

- 14 Setiap bab dalam skripsi ini memiliki sistematika pembahasan dalam poin-poin sebagai berikut:
- 15 1. Bab 1: Pendahuluan
16 Bab ini berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi peneli-
17 tian, dan sistematika pembahasan.
 - 18 2. Bab 2: Dasar Teori
19 Bab ini berisi pembahasan-pembahasan teoritis mengenai aspek-aspek yang akan dirujuk di
20 dalam skripsi ini, seperti *command line*, bahasa C, dan juga KIRI.
 - 21 3. Bab 3: Analisis dan Perancangan
22 Bab ini berisi pembahasan mengenai rancangan perangkat lunak serta seluruh analisis yang
23 dilakukan terhadap kebutuhan fitur perangkat lunak.
 - 24 4. Bab 4: Implementasi dan Pengujian
25 Bab ini berisi pembahasan mengenai pembuatan perangkat lunak, implementasi seluruh
26 modul-modul yang telah ditentukan di bab 3, serta pengujian fitur-fitur dari perangkat lunak.
 - 27 5. Bab 5: Kesimpulan dan Saran
28 Bab ini berisi kesimpulan hasil pembuatan perangkat lunak dan saran-saran terhadap hasil
29 perangkat lunak yang diberikan selama pengeraaan skripsi.

1

BAB 2

2

LANDASAN TEORI

3 2.1 *Command Line*

4 *Command line* (atau *command line interface*) dapat diartikan sebagai tampilan antarmuka/*interface*
5 yang memproses perintah dari pengguna dan meneruskannya langsung ke sistem operasi untuk
6 dijalankan.^[5] Seluruh sistem operasi komputer yang ada memiliki sebuah *command line interface*
7 dalam bentuk *shell*, yang dapat digunakan oleh penggunanya untuk langsung mengakses fungsi
8 atau servis yang disediakan oleh sistem operasi.^[6]

9 2.1.1 *Command Line Interface* dan *Graphical User Interface*

10 Ada beberapa dari tipe antarmuka yang masih banyak digunakan di zaman sekarang, tetapi dua tipe
11 yang paling banyak muncul adalah *command line interface* dan *graphical user interface*. Perangkat
12 lunak berbasis *command line* sendiri bisa memiliki berbagai macam tampilan, tetapi semuanya
13 selalu mengikuti satu bentuk antarmuka umum. Bentuk yang dimaksud adalah sebuah area/*window*
14 yang memuat teks berupa perintah-perintah dari user untuk dilakukan oleh komputer, beserta
15 keluarannya yang juga berupa teks, seperti dapat dilihat pada gambar 2.1a. Jenis perangkat lunak
16 seperti ini disebut memiliki antarmuka jenis *command line interface* (CLI). Adapun dekorasi visual
17 yang dimiliki oleh jenis tampilan ini hanya berupa warna pada teks-teks yang ada, tanpa tambahan
18 gambar apapun. Jika perangkat lunak tersebut memiliki dekorasi dan/atau tombol interaktif berupa
19 gambar grafis, seperti pada gambar 2.1b, maka perangkat lunak tersebut dikategorikan sebagai
20 perangkat lunak berbasis *graphical user interface*.

```
devasc@labvm: ~/labs/personal/samples
File Edit View Search Terminal Help
devasc@labvm:~$ ls -l
total 20
drwxr-xr-x 3 devasc devasc 4096 Dec 14 22:44 Desktop
drwxr-xr-x 2 devasc devasc 4096 Sep 18 2021 documents
drwxr-xr-x 3 devasc devasc 4096 Sep 18 2021 downloads
drwxr-xr-x 4 devasc devasc 4096 Dec 14 22:44 Labs
drwxr-xr-x 5 devasc devasc 4096 Jun 17 2020 snap
devasc@labvm:~$ cd "labs/personal/samples"
devasc@labvm:~/labs/personal/samples$ ls
txtsample.txt
devasc@labvm:~/labs/personal/samples$ cat txtsample.txt
Utique latet et non videntur. Non videntur et non
Loquuntur enim dolor sit amet, consectetur adipisciing elit. Collatio ligitur ista te nihil
tuvat. Honesta oratio, Socratica, Platonis etiam. Primum in nostrane potestate est, qui
d meminerimus? Duo Reges: constructio interrete. Quid, si etiam lucunda memoria est pra
eteritorum malorum? Si quidem, inquit, tollerem, sed relinqui. An nisi populari fama?
Quamquam id quidem licetit lis existimare, qui legerint. Summum a vobis bonum voluptas
dictur. At hoc in eo M. Refert tamen, quo modo. Quid sequatur, quid repugnet, vident.
Iam id ipsum absurdum, maximum malum neglegit.
devasc@labvm:~/labs/personal/samples$
```



(a) Antarmuka perangkat lunak berbasis *command line interface*.

(b) Antarmuka perangkat lunak berbasis *graphical user interface*.

Gambar 2.1: Contoh dua jenis antarmuka (*interface*) perangkat lunak.

Selain dari tampilannya sendiri, ada beberapa perbedaan utama lain antara perangkat-perangkat lunak berbasis *command line interface* dengan perangkat lunak berbasis *graphical user interface*.

Adapun perbedaan-perbedaan utama dari kedua jenis antarmuka ini adalah sebagai berikut.[6]

- Penggunaan sumber daya sistem untuk menjalankan perangkat lunak berbasis *command line interface* lebih rendah dibandingkan dengan perangkat lunak berbasis *graphical user interface*.
- Bagi pengguna pemula (atau pengguna awam pada umumnya), perangkat lunak berbasis *command line interface* akan lebih sulit digunakan karena tidak adanya bantuan apapun dalam bentuk visual, sehingga satu-satunya cara untuk tahu bagaimana cara menggunakan fitur-fiturnya adalah melalui dokumentasi perangkat lunak yang ada. Karena alasan yang sama pula, perangkat lunak berbasis *command line interface* lebih sulit untuk dibiasakan penggunaannya.
- Automasi perintah yang bersifat berulang-ulang jauh lebih mudah dilakukan pada perangkat lunak berbasis *command line interface*. Hal ini dikarenakan perangkat lunak berbasis *command line interface* tidak hanya lebih mudah untuk dibuat *script*-nya, tetapi juga lebih efisien untuk digunakan ketika ada banyak sekali perintah yang harus dilakukan pada suatu saat tertentu.

2.1.2 *Command Line* di Linux

Linux merupakan sebuah sistem operasi yang sangat modular, jadi ada banyak sekali *shell* yang dapat dijalankan dan digunakan di dalamnya. Walaupun begitu, ada satu *shell* yang selalu datang ter-*install* di dalam semua sistem operasi Linux, yaitu “*bash*” (GNU *Bourne Again Shell*).[7]

Tampilan

Ketika terminal di Linux dijalankan, akan keluar kotak dialog, beserta sebuah baris. Baris ini biasanya berisi sebuah teks dengan format sebagai berikut.

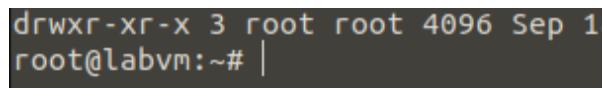
```
<nama pengguna>@<nama perangkat>:<direktori yang sedang diproses>$
```

Tanda dolar di ujung baris ini menandakan bahwa baris tersebut merupakan baris *shell prompt*, yang merupakan waktu di mana terminal sudah siap menerima masukan dari pengguna untuk diproses. Perlu diingat bahwa di posisi tanda dolar ini, terkadang justru terdapat tanda pagar (#). Tanda pagar di akhir baris *shell prompt* menandakan bahwa terminal tersebut dijalankan dengan tingkat akses *superuser*, yang berarti bahwa entah pengguna masuk ke sistem sebagai user *root*, atau terminal memiliki izin tingkat *superuser/administrator*.[5]



```
drwx----- 5 devasc devasc 4096 Sep 1
devasc@labvm:~$ |
```

(a) *Shell prompt* terminal dengan tingkat izin normal.



```
drwxr-xr-x 3 root root 4096 Sep 1
root@labvm:~# |
```

(b) *Shell prompt* terminal dengan tingkat izin *superuser*.

Gambar 2.2: Baris *shell prompt* terminal di sistem operasi Linux.

1 Navigasi [5]

2 Sama seperti di Windows, Linux menyimpan file-filenya di sebuah struktur direktori yang bersifat
3 hierarkial. Hal ini berarti bahwa file-file tersebut disimpan dalam direktori-direktori (atau *folder-*
4 *folder*) yang tersusun seperti sebuah pohon. dalam arti bahwa satu *folder* bisa jadi berada di dalam
5 satu *folder* lain, atau berisi beberapa *folder* lainnya.

6 Untuk navigasi, terminal Linux memiliki beberapa perintah utama. Adapun perintah-perintah
7 tersebut adalah sebagai berikut.

- 8 • `pwd`

9 `pwd` merupakan singkatan dari *print working directory*, yang berarti bahwa perintah ini akan
10 mengeluarkan *working directory*, atau direktori tempat terminal sekarang sedang bekerja/ber-
11 jalan, sebagai keluaran dari perintah tersebut. Ketika pengguna pertama kali menjalankan
12 terminal, *working directory*-nya selalu merupakan direktori *home* dari perangkat.

- 13 • `ls`

14 `ls` digunakan untuk menghasilkan keluaran berupa isi dari folder yang dispesifikasi. Biasanya
15 digunakan ketika pengguna sudah memasuki folder yang diinginkan, walaupun dengan perintah
16 ini, pengguna bisa saja mengintip isi dari folder manapun di direktori manapun, dengan
17 mengikutkan direktori yang diinginkan sebagai parameter dari perintah tersebut. Adapun
18 Isi dari folder yang diikutkan sebagai parameter tidak hanya berupa folder lain, tetapi juga
19 seluruh file-file yang ada, walaupun untuk file-file yang disembunyikan (nama file diawali
20 dengan tanda titik), perlu ditambahkan opsi `-a` agar file-file tersebut muncul pula dalam
21 keluarannya.

- 22 • `cd`

23 `cd` adalah perintah yang berfungsi untuk mengganti *working directory* dari terminal. Untuk
24 melakukan hal tersebut, perintah yang perlu dimasukkan adalah sebagai berikut:

25 `cd <direktori yang diinginkan>`

26 Direktori yang diinginkan dapat berupa direktori absolut, atau direktori relatif. Perbedaannya
27 adalah direktori absolut selalu dimulai dari folder *root*, mengikuti folder-folder apapun yang
28 ada di antara *root* sampai ke folder yang diinginkan.

29 Sedangkan, direktori relatif selalu dimulai dari *working directory*. Untuk penggunaan direktori
30 relatif, diperlukan dua buah notasi spesial, yaitu titik `(.)`, yang merepresentasikan *working*
31 *directory* sekarang itu sendiri, dan dua titik `(..)`, yang merepresentasikan *parent folder* dari
32 *working directory*.

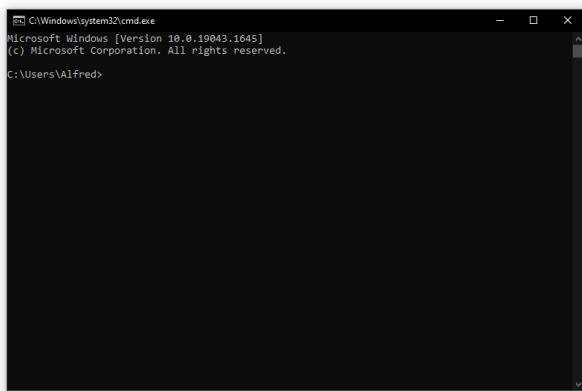
33 2.1.3 Command Line di Windows

34 Cara kerja *command line* di Windows serupa dengan cara kerja *command line* di Linux, dalam
35 arti bahwa untuk bekerja dengan *command line* di Windows, penggunanya juga akan langsung
36 berinteraksi dengan utilitas yang disediakan oleh sistem operasi. *Command line* di Windows juga
37 dapat digunakan untuk hal-hal yang serupa dengan *command line* di Linux, seperti menulis (dan
38 menjalankan) *script*, menjalankan perintah yang diinginkan pengguna secara otomatis, atau melihat
39 status dari sistem operasi.[6]

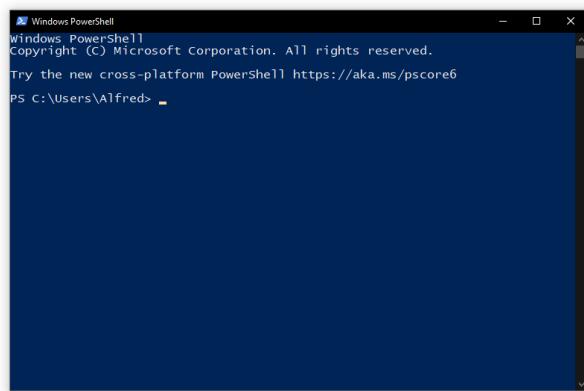
1 Masih sama dengan Linux, ada banyak sekali command yang bisa digunakan, sehingga susah
 2 untuk menghafal seluruh command-command yang ada—termasuk masukan, parameter-parameter
 3 yang dibutuhkan, serta keluarannya. Untuk melihat dokumentasi, atau penjelasan detail untuk
 4 masukan, keluaran, parameter, serta opsi-opsi dari perintah tertentu, pengguna dapat memasukkan
 5 perintah tersebut, diikuti dengan /?.[6]

6 Tampilan

7 Di sistem operasi Windows, ada dua jenis antarmuka *command line*, yaitu cmd (*Command Prompt*)
 8 dan *PowerShell*. Keduanya memiliki tampilan yang kurang lebih sama—hanya saja awalnya cmd
 9 memiliki latar belakang hitam, sedangkan *PowerShell* memiliki latar belakang biru tua, seperti
 10 terlihat di gambar 2.3.



(a) Antarmuka Windows *Command Prompt* (*cmd*)



(b) Antarmuka Winodws *PowerShell*

Gambar 2.3: Tampang kedua antarmuka *command line* bawaan di sistem operasi Windows.

11 Navigasi

12 Untuk navigasi di antarmuka *command line* Windows, ada dua perintah penting yang dipakai ketika
 13 pengguna sedang berurusan dengan file-file dan navigasi dalam direktori sistem. Kedua perintah
 14 tersebut adalah **cd** dan **dir**.

- 15 • **cd** (**chdir**) [8]

16 **cd** merupakan sebuah perintah yang memiliki tiga fungsi utama, yaitu menampilkan *drive*
 17 tempat sedang *command line* berada (jika pengguna hanya memasukkan **cd** tanpa parameter
 18 apapun), menampilkan direktori tempat *command line* sedang berada (jika pengguna hanya
 19 memasukkan *drive* sebagai parameter, atau fungsi yang paling umumnya, untuk mengganti
 20 *working directory* dari *command line*.

21
 22 Adapun format dari perintah **dir** adalah sebagai berikut.

23 **cd** [/d] [<drive>:] [<path>]

1 Dengan fungsi dari semua opsi dan parameter yang ada sebagai berikut.

2 – /d

3 Opsi yang menandakan bahwa pengguna ingin mengganti *drive* (partisi) dan juga *working*
4 *directory* dari *command line*.

5 – <drive>:

6 Kode huruf dari partisi yang akan diproses.

7 – <path>

8 Direktori yang akan diproses. Parameter ini harus diikutkan beserta kode huruf partisi
9 (tidak dapat berdiri sendiri.)

10 • **dir**

11 **dir** merupakan sebuah perintah yang mengeluarkan/menampilkan sebuah daftar berisi file-file
12 yang ada di suatu direktori, termasuk subdirektori. Jika tidak disertai parameter apapun,
13 perintah ini akan menampilkan label volume dan nomor serial *disk*, dilanjutkan dengan daftar
14 direktori dan file di dalamnya. Untuk file, akan ditampilkan nama beserta ukurannya. Perintah
15 ini juga akan menampilkan jumlah direktori dan file yang didaftarkan, ukuran kumulatifnya,
16 dan sisa dari *disk* yang tidak terpakai (dalam *bytes*).[8]

17 Adapun format dari perintah **dir** adalah sebagai berikut.[6]

19 **dir** [<drive:>] [<path>] [<filename>] [/A[:<attributes>]] [/B]
20 [/C] [/D] [/L] [/N] [/O[:<sortorder>]] [/P] [/Q] [/R] [/S]
21 [/T[:<timefield>]] [/W] [/X] [/4]

22 Untuk perintah ini, seperti terlihat di atas, memiliki banyak sekali opsi dan parameter.
23 Tiap-tiap dari parameter tersebut memiliki fungsi tersendiri, yaitu:

24 – /A[:<attributes>]

25 Menampilkan file-file dengan atribut tertentu, seperti file yang disembunyikan, file sistem,
26 file *read-only*, dan sebagainya.

27 – /B

28 Menghilangkan *heading* dan ringkasan informasi dari keluaran, atau dengan kata lain,
29 hanya menampilkan file-file dan direktori, tanpa informasi tambahan apapun.

30 – /C

31 Menggunakan separator koma untuk tiap angka ribuan di ukuran file. Jika opsi yang
32 dimasukkan adalah /-C, separator koma justru akan dihilangkan.

33 – /D

34 Menampilkan keluaran dengan format yang lebih lebar. Jika opsi ini diikutkan, keluaran
35 akan ditampilkan dengan urutan berdasarkan kolom.

36 – /L

37 Seluruh teks dalam keluaran akan menggunakan huruf kecil. Jika opsi ini tidak digunakan,
38 keluaran akan mengandung huruf besar dan huruf kecil *mixed case*.

39 – /N

40 Menampilkan daftar dengan format panjang, dengan nama file berada di ujung paling
41 kanan.

- /O[[:]<sortorder>]
Menampilkan daftar direktori yang terurut berdasarkan urutan tertentu, seperti berdasarkan ekstensi file, berdasarkan tanggal dibuat, berdasarkan nama, dan sebagainya. Jika tidak diikutkan tanda minus (-) sebelum huruf O pada perintah, daftar yang muncul akan terurut secara menaik.
- /P
Memberhentikan keluaran selama beberapa waktu singkat (memberi jeda kecil) setelah setiap halaman informasi.
- /Q
Menambahkan informasi mengenai pemilik file dalam keluaran.
- /R
Menampilkan *data stream* alternatif, jika ada.
- /S
Mendaftarkan seluruh file di direktori dan subdirektori yang diproses. Tiap-tiap direktori akan memiliki *header* tersendiri dalam keluarannya.
- /T[[:]<timefield>]
Menspesifikasi *time field* mana yang akan tampil dan digunakan sebagai urutan, jika aturan pengurutan lain tidak ditentukan. *Time field* yang dapat digunakan adalah waktu pembuatan file, kapan terakhir file diakses, dan kapan file terakhir dimodifikasi. Jika parameter ini tidak dispesifikasi, *time field* yang digunakan adalah kapan file terakhir dimodifikasi.
- /W
Menampilkan keluaran dengan format yang lebih lebar. Opsi ini hampir sama dengan /D, hanya saja untuk /W, jika opsi ini diikutkan, keluaran akan ditampilkan dengan urutan berdasarkan baris, dan bukan kolom.
- /X
Menampilkan nama pendek yang dibuat untuk nama-nama file non-8.3. Opsi ini memiliki format tampilan yang sama seperti opsi /N, hanya saja nama pendeknya ditampilkan di keluaran sebelum nama panjangnya.
- 4
Menampilkan angka tahun dengan format angka empat digit.

32 2.2 KIRI

33 KIRI merupakan sebuah perangkat lunak berbasis web yang berfungsi untuk menyelesaikan (atau
34 setidaknya mengurangi) dampak dari masalah-masalah yang dapat diselesaikan oleh transportasi
35 umum/publik di Indonesia, seperti pemanasan global, kemacetan, atau peningkatan harga bensin.
36 Selain itu, turis mancanegara juga memilih untuk menaiki transportasi umum, karena jenis sarana
37 transportasi tersebut tidak hanya jauh lebih murah, tetapi juga memberikan kesempatan yang
38 mudah kepada mereka untuk melihat seluk-beluk dari kota-kota yang mereka kunjungi. Walaupun
39 begitu, banyak masyarakat lokal sendiri yang seringkali masih segan untuk menaiki transportasi
40 publik, umumnya karena transportasi publik dianggap lebih rumit persiapannya dibandingkan
41 dengan metode-metode transportasi privat, seperti menaiki kendaraan pribadi.¹

¹<https://projectkiri.github.io/#about-kiri>

Di halaman web KIRI, pengguna dapat memasukkan input berupa lokasi awal dan lokasi tujuan dan KIRI akan menghasilkan seluruh langkah yang harus ditempuh oleh pengguna untuk sampai ke lokasi tujuan, dengan menggunakan angkot. Keluaran ini sudah meliputi kode angkot mana saja yang harus dinaiki, dan juga seberapa jauh pengguna harus berjalan kaki untuk sampai ke lokasi rute angkot berikutnya.

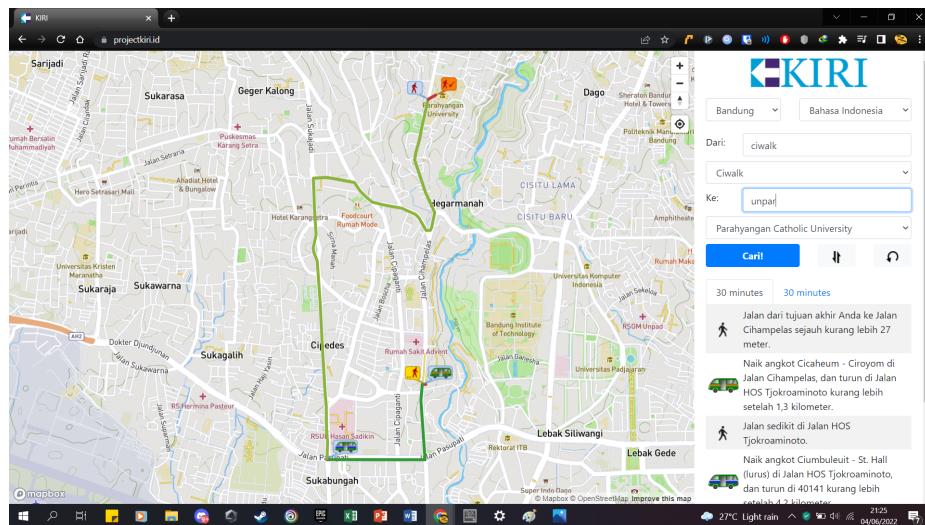
2.2.1 Tampilan

Pada saat pertama kali dibuka, hal pertama yang paling mencolok di halaman awal web KIRI adalah sebuah peta besar di sebelah kiri yang dapat diperbesar ataupun diperkecil. Sedangkan, bagian kanan dari halamannya terdiri atas beberapa bagian. Di bagian paling atas terdapat logo KIRI, beserta sepasang menu *dropdown*—yang pertama merupakan pilihan kota tempat pengguna berada (untuk sekarang hanya tersedia pilihan kota Jakarta dan Bandung), dan yang kedua merupakan pilihan bahasa, entah bahasa Indonesia atau Inggris. Di bawahnya merupakan sepasang menu *dropdown* yang merupakan tempat di mana pengguna memasukkan lokasi awal dan tujuan yang akan diproses oleh KIRI. Terakhir, di bawahnya ada sebuah bagian kosong, yang nantinya akan menjadi tempat di mana KIRI akan meletakkan keluaran dari prosesnya. Adapun tampilan awal dari halaman web ini dapat dilihat di gambar 2.4.



Gambar 2.4: Tampilan awal halaman web KIRI.

Ada dua area yang memiliki perbedaan yang signifikan ketika pengguna sudah memasukkan masukan dan menyuruh KIRI untuk memprosesnya. Bagian yang pertama adalah bagian peta, yang setelah pemrosesan masukan, akan memiliki garis-garis berwarna yang menandakan rute angkot maupun tujuan perjalanan kaki yang harus ditempuh oleh pengguna. Bagian kedua adalah bagian keluaran, yang tadinya kosong, sekarang akan berisi langkah-langkah yang harus ditempuh oleh penggunanya untuk pergi dari lokasi awal ke lokasi tujuan. Spesifiknya, perbedaan-perbedaan ini dapat dilihat di gambar 2.5.



Gambar 2.5: Tampilan halaman web KIRI setelah pemrosesan masukan dari pengguna selesai.

2.2.2 API²

KIRI juga memiliki sebuah API yang dapat digunakan untuk keperluan pengembangan perangkat lunak. API ini menyediakan tiga buah jenis layanan web (*webservice*), yang ketiganya dapat dilakukan dengan mengirim permintaan (*request*) tipe GET melalui API tersebut. Isi dari permintaan yang perlu dikirimkan serta respon dari API yang akan dikembalikan berbeda tergantung dari jenis layanan yang digunakan. Adapun ketiga jenis layanan tersebut adalah pencarian tempat (*search place*), pencarian rute (*routing*), dan *smart direction*.

Search Place

Layanan pencarian lokasi (*search place*) adalah layanan web pada API KIRI yang berfungsi untuk mencari suatu lokasi berdasarkan kata kunci yang diberikan oleh pengguna. Untuk menggunakan layanan ini, pengguna harus mengirim permintaan GET ke alamat <https://projectkiri.id/api>. Adapun permintaan tersebut harus memiliki parameter-parameter seperti terlihat di bawah ini.

- **version**

Kemungkinan nilai: 2

Parameter ini merupakan tanda bagi API untuk menggunakan protokol versi 2.

- **mode**

Kemungkinan nilai: searchplace

Parameter ini merupakan mode dari servis/jasa API yang akan digunakan oleh pengguna. Untuk penggunaan layanan pencarian lokasi, variabel ini harus diisi dengan **searchplace**.

- **region**

Kemungkinan nilai: cgk, bdo, mlg, atau sub

Parameter ini merupakan kode bandara IATA tiga huruf yang merepresentasikan daerah mana tempat lokasi yang ingin dicari berada. Kode yang dapat diproses oleh API ini meliputi **cgk** (Cengkareng/Jakarta), **bdo** (Bandung), **mlg** (Malang), dan **sub** (Surabaya).

²<https://github.com/projectkiri/Tirtayasa/wiki/KIRI-API-v2>



Gambar 2.6: Halaman web *API Keys* KIRI.

1 • **querystring**

2 **Kemungkinan nilai:** *string* apapun dengan panjang minimal satu karakter

3 Parameter ini berisi kata kunci yang akan digunakan untuk menentukan lokasi yang ingin
4 dicari pengguna.

5 • **apikey**

6 **Kemungkinan nilai:** angka heksadesimal 16-digit

7 Parameter ini berisi kunci API pribadi yang harus digenerasi terlebih dahulu sebelum API
8 dapat digunakan.

9 Perlu diperhatikan bahwa salah satu dari parameter yang harus diikutkan dalam pesan tersebut
10 merupakan parameter yang meminta kunci API. Kunci tersebut harus digenerasikan terlebih dahulu
11 sebelum API KIRI dapat digunakan, melalui halaman *API Keys* KIRI,³ yang dapat dilihat di
12 gambar 2.6.

13 Untuk mengakses halaman tersebut, pengguna harus membuat sebuah akun terlebih dahulu.
14 Ketika akun sudah dibuat, maka pengguna baru akan dapat membuat kunci API yang dibutuhkan,
15 sekaligus membuat filter *domain*, yang membatasi di *domain* mana saja kunci tersebut dapat
16 digunakan, serta memberikan deskripsi untuk kunci API tersebut. Kunci ini kemudian dapat
17 digunakan sebagai nilai dari parameter **apikey** yang diperlukan dalam permintaan tadi.

18 Sebelum membahas keluaran dari layanan API ini, perlu ditegaskan dulu apa definisi dari nilai
19 *latitude* dan *longitude* suatu lokasi. *Latitude* merupakan berapa derajat sebuah tempat berada dari
20 garis ekuator, dengan lokasi-lokasi yang berada maksimum 90 derajat di atas ekuator memiliki
21 nilai *latitude* positif, sedangkan lokasi-lokasi yang berada maksimum 90 derajat di bawah ekuator
22 memiliki nilai *latitude* negatif. Sedangkan, *longitude* merupakan berapa derajat lokasi sebuah
23 tempat berada dari garis meridian (bujur) utama Bumi, dengan rentang nilai dari -180 derajat di
24 sisi kiri (barat) meridian utama, hingga 180 derajat di kanan (timur) bujur tersebut.⁴ Kedua nilai
25 ini merupakan salah satu dari dua variabel yang dikembalikan dalam respon API untuk layanan ini,

³<https://projectkiri.id/dev/apikeys>

⁴https://gsp.humboldt.edu/olm/Lessons/GIS/01%20SphericalCoordinates/Latitude_and_Longitude.html



Gambar 2.7: Penggunaan API KIRI untuk layanan pencarian lokasi menggunakan Postman. Gambar ini menunjukkan hasil pencarian lokasi “unpar” di daerah Bandung.

1 dengan variabel lainnya berupa nama dari lokasi yang ditemukan itu sendiri. Adapun respon yang
2 diberikan oleh API akan berupa sebuah objek JSON yang selalu memiliki setidaknya dua variabel,
3 yaitu:

- 4 • **status**

5 **Kemungkinan nilai:** `ok` atau `error`

6 Variabel ini manandakan apakah permintaan berhasil diproses atau tidak. Jika permintaan
7 berhasil diproses, variabel ini akan bernilai `ok`, dan jika tidak, variabel ini akan bernilai `error`.

- 8 • **searchresult**

9 Variabel ini merupakan hasil pencarian yang ditemukan oleh layanan API ini. Isi dari variabel
10 ini merupakan objek *array* yang masing-masing memiliki variabel berikut:

- 11 – **placename**

12 Variabel ini berisi nama lokasi yang ditemukan berdasarkan kata kunci yang diberikan
13 oleh pengguna.

- 14 – **location**

15 Variabel ini berisi nilai *latitude* dan *longitude* dari lokasi yang ditemukan dalam pencarian.

16 Contoh dari penggunaan API KIRI untuk layanan ini dapat dilihat di gambar 2.7.

17 ***Routing***

18 Layanan pencarian rute (*routing*) adalah layanan web pada API KIRI yang memiliki fungsi yang
19 sama dengan fungsi utama dari perangkat lunak KIRI sendiri, yaitu menunjukkan rute serta
20 langkah-langkah yang harus ditempuh untuk pergi dari satu lokasi ke lokasi lainnya, dengan
21 menggunakan angkot yang tersedia. Untuk menggunakan layanan ini, pengguna harus mengirim
22 permintaan GET ke alamat <https://projectkiri.id/api>. Adapun permintaan tersebut harus memiliki
23 parameter-parameter seperti terlihat di bawah ini.

- 1 • **version**

2 **Kemungkinan nilai:** 2

3 Parameter ini merupakan tanda bagi API untuk menggunakan protokol versi 2.

- 4 • **mode**

5 **Kemungkinan nilai:** **findroute**

6 Parameter ini merupakan mode dari servis/jasa API yang akan digunakan oleh pengguna.

7 Untuk penggunaan layanan pencarian rute dengan angkot, variabel ini diisi dengan **findroute**.

- 8 • **locale**

9 **Kemungkinan nilai:** **en** atau **id**

10 Parameter ini mengatur bahasa apa yang akan digunakan dalam keluaran API nantinya—**en**

11 berarti keluaran akan menggunakan bahasa Inggris, dan **id** berarti keluaran akan menggunakan

12 bahasa Indonesia.

- 13 • **start**

14 **Kemungkinan nilai:** **lat, lng;** dalam bentuk desimal

15 Parameter ini merupakan nilai *latitude* dan *longitude* dari titik awal perjalanan pengguna.

- 16 • **finish**

17 **Kemungkinan nilai:** **lat, lng;** dalam bentuk desimal

18 Parameter ini berisi nilai *latitude* dan *longitude* dari titik akhir/tujuan perjalanan pengguna.

- 19 • **presentation** (opsional)

20 **Kemungkinan nilai:** **desktop**

21 Parameter ini hanya digunakan untuk fitur *backwards compatibility*.

- 22 • **apikey**

23 **Kemungkinan nilai:** angka heksadesimal 16-digit

24 Parameter ini berisi kunci API pribadi yang harus digenerasi terlebih dahulu sebelum API
25 dapat digunakan.

26 Sedangkan, respon yang diberikan oleh API akan berupa sebuah objek JSON yang selalu memiliki
27 setidaknya dua variabel, yaitu:

- 28 • **status**

29 **Kemungkinan nilai:** **ok** atau **error**

30 Variabel ini manandakan apakah permintaan berhasil diproses atau tidak. Jika permintaan
31 berhasil diproses, variabel ini akan bernilai **ok**, dan jika tidak, variabel ini akan bernilai **error**.

- 32 • **message**

33 Variabel ini bisa berisi dua macam objek. Jika permintaan dari user tidak berhasil diproses,
34 atau dalam kata lain, terjadi sebuah *error*, maka variabel ini akan berisi string yang merupakan
35 pesan *error* serta alasan spesifik mengapa *error* tersebut terjadi. Di lain sisi, jika permintaan
36 dari user berhasil diproses, variabel ini akan mengalami dua perubahan utama. Pertama,
37 nama variabel ini akan berubah menjadi **routingresults**, dan kedua, isi dari variabel ini
38 akan menjadi sebuah *array* JSON yang merupakan respon dari API KIRI berupa keluaran
39 yang akan dilihat oleh pengguna. *Array* JSON ini sendiri terbagi menjadi beberapa variabel
40 lainnya, yang dapat dilihat di daftar di bawah ini.

- 41 – **steps**

1 **Tipe:** *array*

2 Variabel ini merepresentasikan satu buah langkah yang harus ditempuh oleh pengguna.
3 Adapun *array* ini sendiri berisi variabel-variabel berikut:

- 4 * Tipe transportasi

5 Tipe sarana transportasi yang harus dipakai oleh pengguna. Jika pengguna harus
6 berjalan kaki, variabel ini akan berisi **walk**. Jika pengguna harus menaiki angkot,
7 variabel ini akan berisi **angkot**.

- 8 * Kode angkot

9 Variabel ini menunjukkan angkot mana yang harus dinaiki oleh pengguna di langkah
10 tersebut. Jika penggunaan angkot tidak dimungkinkan pada langkah ini (pengguna
11 harus berjalan kaki), variabel ini akan berisi **walk**.

- 12 * *Array latitude dan longitude* lokasi

13 *Array* nilai-nilai desimal *latitude* dan *longitude* dari berbagai titik lokasi yang terdapat
14 dalam rute.

- 15 * Deskripsi langkah

16 Deskripsi langkah yang harus ditempuh, dalam bahasa natural. Bahasa yang digu-
17 nakan tergantung parameter **locale** yang diatur dalam masukan.

- 18 * URL untuk mendapatkan tiket kendaraan

19 Tautan untuk mendapatkan tiket angkutan umum, jika diperlukan. Jika transportasi
20 pada langkah tersebut tidak memerlukan tiket, variabel ini akan berisi **null**.

- 21 * URL editor rute

22 Tautan untuk melakukan modifikasi rute, jika dimungkinkan. Jika rute tidak bisa
23 dimodifikasi, variabel ini akan berisi **null**.

24 – **traveltime**

25 **Tipe:** *string*

26 Variabel ini berisi estimasi jangka waktu yang diperlukan untuk menyelesaikan langkah
27 tersebut.

28 Adapun gambar 2.8 menunjukkan penggunaan API KIRI untuk layanan pencarian rute dari
29 Cihampelas Walk ke Universitas Katolik Parahyangan.

30 **Smart Direction**

31 Layanan terakhir dari API ini adalah layanan *smart direction*, yang merupakan gabungan dari
32 kedua layanan sebelumnya. Berbeda dengan kedua layanan tadi, yang harus mengakses API secara
33 manual (dengan mengirimkan permintaan GET), layanan ini tidak memerlukan pengguna untuk
34 mengirim permintaan tersebut secara manual—layanan ini sudah otomatis mengirimkan permintaan
35 tersebut. Berbeda dengan kedua layanan sebelumnya pula, layanan ini tidak memerlukan dibuatnya
36 kunci API terlebih dahulu.

37 Layanan ini bekerja dengan menyuruh peramban pengguna untuk langsung mengakses halaman
38 web KIRI yang sudah langsung menunjukkan rute yang perlu ditempuh untuk pergi dari lokasi satu
39 ke lokasi lainnya. Untuk melakukan hal ini, layanan ini memerlukan sebuah URL, yang memiliki
40 format sebagai berikut:



Gambar 2.8: Penggunaan API KIRI untuk layanan pencarian rute menggunakan Postman. Gambar ini menunjukkan hasil pencarian rute dari Cihampelas Walk ke UNPAR.

- 1 <https://projectkiri.id?start=<lokasi awal>&finish=<lokasi akhir>&locale=<locale>>
 - 2 Dapat dilihat bahwa URL tersebut memiliki tiga buah parameter, yaitu:
 - **start**
Kemungkinan nilai: Nilai *latitude* dan *longitude* lokasi, atau nama lokasi tersebut
Parameter ini berisi lokasi yang ingin digunakan sebagai lokasi mulainya pencarian rute.
 - **finish**
Kemungkinan nilai: Nilai *latitude* dan *longitude* lokasi, atau nama lokasi tersebut
Parameter ini berisi lokasi yang merupakan tujuan akhir yang ingin dicapai dalam pencarian rute.
 - **locale** (opsional)
Kemungkinan nilai: id atau en
Menentukan dalam bahasa apa hasil pencarian rutenya akan ditampilkan (bahasa Indonesia atau bahasa Inggris). Jika parameter ini tidak diberikan oleh pengguna, maka bahasa yang akan digunakan adalah bahasa yang terakhir dipakai di halaman web KIRI sendiri.
 - 15 Misalkan pengguna ingin mencari rute dari Cihampelas Walk ke Universitas Katolik Parahyangan, dan menampilkan langkah-langkah yang harus ditempuh dalam rutenya dalam bahasa Indonesia.
 - 17 Pengguna dapat memasukkan URL berikut ke peramban mereka.
 - 18 <https://projectkiri.id?start=ciwalk&finish=unpar&locale=id>
 - 19 Jika URL tersebut sudah dimasukkan ke kotak *link* pada peramban, halaman yang akan ditampilkan akan terlihat seperti pada gambar 2.9.
- ## 21 2.3 Fungsi dan *Library* Bahasa C
- 22 Di bagian ini akan dilakukan studi literatur terhadap seluruh fungsi bawaan serta *library-library* bahasa pemrograman C yang akan digunakan dalam pebuatan perkakas ini.



Gambar 2.9: Penggunaan API KIRI untuk layanan *smart direction*, dari Cihampelas Walk ke UNPAR.

1 2.3.1 getopt [1]

2 **getopt** merupakan sebuah fungsi yang dapat mengautomasi pekerjaan-pekerjaan yang berhubungan
3 dengan penerimaan opsi-opsi untuk *command line* berbasis UNIX.

4

5 Fungsi **getopt** dapat dipanggil dengan format sebagai berikut.

6 **getopt (argc, argv, <options>)**

7 Seluruh kode ini dapat dimasukkan ke suatu variabel berupa sebuah karakter yang merepre-
8 sentasikan opsi yang ingin digunakan. **argc** merupakan jumlah argumen yang terdapat dalam
9 masukan, sedangkan **argv** merupakan sebuah *array* yang berisi argumen-argumen tersebut.

10
11 Selain itu, penggunaan **getopt** juga akan memakai variabel-variabel tertentu, yang nilainya akan
12 diisi oleh fungsi **getopt** tersebut sendiri. Variabel-variabel ini beserta penjelasannya dapat dilihat
13 di daftar berikut.⁵

- 14 • **opterr**

15 Isi dari variabel ini akan memberi sinyal ke perangkat lunak/perkakas yang menentukan
16 apakah **getopt** akan mengirim pesan ke *error stream* atau tidak. Jika variabel ini bukan
17 bernilai 0, maka pesan *error* akan dikirim. Sebaliknya, jika variabel ini bernilai 0, **getopt**
18 tidak akan mengirim pesan *error* apapun, tetapi tetap akan mengembalikan sebuah karakter
19 tanda tanya (?) sebagai tanda bahwa sebuah *error* telah terjadi.

- 20 • **optopt**

21 Ketika **getopt** menemukan sebuah karakter yang tidak didefinisikan dalam kumpulan opsi,
22 atau sebuah opsi yang tidak disertai argumen yang diperlukan, karakter tersebut akan disimpan
23 di variabel ini.

- 24 • **optind**

25 Variabel ini digunakan oleh **getopt** sebagai indeks untuk *array argv*. Jika seluruh argumen

⁵https://www.gnu.org/software/libc/manual/html_node/Using-Getopt.html

1 sudah diproses, nilai variabel ini dapat digunakan untuk menentukan argumen mana yang
 2 merupakan arguman tambahan yang tidak terpakai. Nilai dari variabel ini dimulai dari 1.
 3 • **optarg**
 4 Jika opsi yang sedang diproses memerlukan argumen, variabel ini adalah tempat dimana
 5 argumen tersebut akan disimpan.
 6 • **<options>**
 7 Variabel ini merupakan salah satu variabel yang tertera di format pemanggilan `getopt` diatas.
 8 Variabel ini berupa *string* yang menandakan karakter-karakter apa saja yang menjadi opsi
 9 yang mungkin dalam perkakas tersebut, beserta tipenya. Jika karakter opsi:
 10 – Diikuti dengan titik dua (:), maka opsi tersebut memiliki argumen yang bersifat wajib.
 11 – Diikuti dengan titik dua ganda (::), maka opsi tersebut memiliki argumen yang bersifat
 12 opsional.
 13 – Tidak diikuti apa-apa, maka opsi tersebut merupakan opsi tidak berarguman.
 14 Untuk memberikan gambaran yang lebih baik mengenai penggunaan fungsi `getopt`, berikut meru-
 15 pakan contoh perangkat lunak berbasis *command line* sederhana yang menggunakan fungsi tersebut.
 16 Perangkat lunak ini akan menerima masukan berupa opsi tidak berparameter `-a` dan/atau opsi
 17 berparameter `-b`. Untuk opsi `-a`, perangkat lunak ini akan mengeluarkan nilai 0 jika opsi `-a` tidak
 18 dipakai, dan 1 jika opsi tersebut dipakai. Sebagai keluaran keduanya, perangkat lunak ini akan
 19 mengeluarkan isi parameter dari opsi `-b`, atau `NULL` jika opsi `-b` tidak dipakai.

Kode 2.1: Contoh sederhana penggunaan `getopt`

```

21 1 #include <ctype.h>
22 2 #include <stdio.h>
23 3 #include <stdlib.h>
24 4 #include <unistd.h>
25 5
26 6 int main(int argc, char **argv) {
27 7     int aflag = 0;
28 8     char *bvalue = NULL;
29 9     int index;
30 0     int args;
31 1
32 2     opterr = 0;
33 3
34 4     while ((args = getopt(argc, argv, ":ab:")) != -1)
35 5         switch (args) {
36 6             case 'a':
37 7                 aflag = 1;
38 8                 break;
39 9             case 'b':
40 0                 bvalue = optarg;
41 1                 break;
42 2             case ':':
43 3                 if (optopt == 'b') {
44 4                     fprintf(stderr, "Option_-%c_requires_an_argument.\n", optopt);
45 5                 }
46 6                 return 1;
47 7             case '?':
48 8                 if (isprint(optopt)) {
49 9                     fprintf(stderr, "Unknown_option_-%c.\n", optopt);
50 0                 }
51 1                 else fprintf(stderr, "Unknown_option_character_\\x%x.\n", optopt);
52 2                 return 1;
53 3             default:
54 4                 abort();
55 5         }
56 6
57 7     printf("aflag=%d,bvalue=%s\n", aflag, bvalue);
58 8     for (index = optind; index < argc; index++)
59 9         printf("Non-option_argument_%s\n", argv[index]);
60 0
61 1     return 0;
62 2 }
```

1 **getopt-long**

2 Ada pula versi getopt yang memungkinkan perangkat lunak untuk menerima dua jenis opsi—opsi
 3 versi pendek berupa sebuah karakter singular, seperti pada getopt biasa, dan/atau opsi panjang
 4 bergaya GNU, berupa sebuah kata.

5 getopt-long juga memiliki seluruh variabel-variabel yang dimiliki oleh getopt, hanya saja
 6 getopt-long memiliki sebuah variabel tambahan berupa struktur, yaitu `long_options`. Variabel
 7 ini merupakan sebuah struktur berupa *array* yang berisi beberapa *array* lainnya, di mana *array-array*
 8 lain ini merupakan masing-masing opsi dari fungsi getopt-long tersebut. Tiap-tiap *array* tersebut
 9 memiliki variabel-variabel berikut:

10 • `name`

11 Variabel ini merupakan nama panjang dari opsi.

12 • `has_arg`

13 Variabel ini merupakan penanda apakah opsi memerlukan argumen atau tidak. Nilai yang
 14 mungkin dalam variabel ini adalah `no_argument`, `required_argument`, atau `optional_argument`.

15 • `flag & val`

16 Kedua variabel ini menandakan bagaimana sebuah opsi akan diberlakukan ketika diterima
 17 oleh getopt-long. Variabel `flag` dapat diisi dengan penunjuk (*pointer*) ke suatu variabel
 18 lain yang akan diisi dengan isi dari variabel `val` untuk menandakan bahwa getopt-long telah
 19 berhasil memroses opsi tersebut. Di lain sisi, jika variabel ini berisi *null pointer*, maka fungsi
 20 getopt-long akan mengembalikan isi dari variabel `val`.

21 Struktur ini harus diakhiri dengan sebuah *array* tambahan yang seluruh variabelnya bernilai 0.

22

23 Untuk menjelaskan lebih lanjut mengenai cara penggunaan getopt-long, berikut merupakan
 24 contoh perangkat lunak berbasis *command line* sederhana yang menggunakan fungsi tersebut.

25 Adapun perangkat lunak ini memiliki spesifikasi sebagai berikut:

- 26 • Perangkat lunak ini akan menerima masukan dari penggunaan opsi-opsi serta parameternya.
- 27 • Keluaran dari perangkat lunak ini adalah opsi apa saja yang dipilih, serta parameter yang
 28 diberikan (jika ada).
- 29 • Opsi pertama yang disediakan adalah `-a` atau `--args` yang merupakan opsi tidak berargumen.
- 30 • Opsi kedua yang disediakan adalah `-n` atau `--noargs`, yang merupakan opsi yang membu-
 31 tuhkan sebuah arguman.
- 32 • Opsi ketiga dan keempat merupakan penanda mode keluaran, yaitu `--short` dan `--long`.
 33 Jika opsi `--long` digunakan, maka perangkat lunak ini akan mengeluarkan versi panjang dari
 34 keluaran, sedangkan jika opsi `--short` digunakan, maka perangkat akan mengeluarkan versi
 35 pendek dari keluaran.

Kode 2.2: Contoh sederhana penggunaan getopt-long

```
37.1 #include <stdio.h>
38.2 #include <stdlib.h>
39.3 #include <getopt.h>
40.4
41.5 int main(int argc, char **argv) {
42.6     int option;
43.7     static int verbose;
44.8
45.9     while (1) {
46.0         static struct option long_options[] = {
```

```

11         {"long", 0, &verbose, 1},
12         {"short", 0, &verbose, 0},
13         {"noargs", 0, 0, 'n'},
14         {"args", 1, 0, 'a'},
15         {0, 0, 0, 0}};
16     int option_index = 0;
17     option = getopt_long(argc, argv, "na:", long_options, &option_index);
18
19     if (option == -1)
20         break;
21     switch (option) {
22     case 0:
23         if (verbose) {
24             printf("Print_mode_is_set_to:_%s", long_options[option_index].name);
25         }
26         else
27             printf("Print_mode_is_set_to:_%s", long_options[option_index].name);
28         putchar('\n');
29         break;
30
31     case 'n':
32         if (verbose == 1) {
33             printf("Option_%'%s'_was_picked._This_option_does_not_require_any_arguments.", long_options[option_index].name)
34             ;
35         }
36         else
37             printf("Argumentless_option_%'%s'_was_picked.", long_options[option_index].name);
38         putchar('\n');
39         break;
40
41     case 'a':
42         if (verbose == 1) {
43             printf("Option_%'%s'_was_picked_with_argument_%'s'.", long_options[option_index].name, optarg);
44         }
45         else
46             printf("a=_%s", optarg);
47         putchar('\n');
48         break;
49
50     case '?':
51         break;
52
53     default:
54         abort();
55     }
56
57     if (optind < argc)
58     {
59         printf("Arguments_passed_without_a_corresponding_option_(argv):_");
60         while (optind < argc) {
61             printf("%s_", argv[optind++]);
62         }
63         putchar('\n');
64     }
65
66     exit(0);
67 }
```

60 2.3.2 libcurl [2]

61 libcurl merupakan sebuah *library* yang berisi fungsi-fungsi yang disediakan dalam bentuk API bahasa
62 C, untuk digunakan oleh aplikasi-aplikasi bahasa C. Libcurl didesain dengan berorientasi transfer
63 (biasanya transfer berkas), tanpa memerlukan para penggunaanya untuk mengerti protokol-protokol
64 yang digunakan dalam proses pentransferan tersebut. Fitur transfer ini dapat dibuat sesederhana
65 mungkin, dan seluruh aturan dan opsi seputar pemindahan berkas tersebut dapat diatur nantinya
66 secara manual melalui opsi-opsi yang ada.

67 Untuk mulai melakukan transfer berkas, diperlukan juga sebuah *handle* yang perlu diinisialisasi
68 terlebih dahulu. Adapun cURL memiliki dua jenis *handle*, yaitu *easy handle* dan *multi handle*.

1 *Easy Handle*

2 *Easy handle* dari cURL dapat diinisialisasi dengan memanggil fungsi `curl_easy_init()`. Setelah
 3 itu, untuk mengatur opsi-opsi yang perlu diatur sesuai kebutuhan pengguna, seperti URL yang
 4 dituju, protokol yang ingin dipakai, koneksi ke port spesifik, dan sebagainya,⁶ pengguna harus
 5 mengurnya dengan fungsi `curl_easy_setopt()`. *Handle* ini berhasil diatur opsinya apabila
 6 fungsi `curl_easy_setopt()` tadi mengembalikan CURLE_OK. Terakhir, untuk menjalankan transfer-
 7 nya, fungsi yang perlu dipanggil adalah `curl_easy_perform(<easy handle>)`, dengan variabel
 8 `<easy handle>` diisi dengan nama dari *easy handle* yang ingin dimulai transfernya.

9 *Handle* yang telah diatur ini dapat digunakan berulang kali dengan konfigurasi yang sama,
 10 sampai entah pengguna mengganti konfigurasi opsi-opsinya kembali, atau atau *handle*-nya direset
 11 dengan pemanggilan fungsi `curl_easy_reset()`.

12 *Multi Handle*

13 *Multi handle* merupakan sebuah *handle* yang dapat memfasilitasi beberapa transfer yang dilakukan
 14 secara paralel. Metode pentransferan filenya masih sama dengan *easy handle*, hanya saja untuk
 15 *multi handle*, diperlukan sebuah *handle* tambahan yang dapat menampung seluruh *easy handle*
 16 yang akan digunakan. Adapun *handle* tipe ini dapat diinisialisasi dengan memanggil fungsi
 17 `curl_multi_init()`, dan untuk mengatur opsi-opsi seputar *multi handle* tersebut, pengguna dapat
 18 memanggil fungsi `curl_multi_setopt()`.

19 Untuk memulai transfer paralel, tentunya perlu diinisialisasi dulu masing-masing *easy handle*-nya.
 20 Setelah *handle-handle* tersebut diinisialisasi, *handle* tersebut dapat dimasukkan ke dalam sebuah
 21 *multi handle* dengan memanggil fungsi berikut.

22 `curl_multi_add_handle(<multi handle>, <easy handle>);`

23 Dengan `<easy handle>` merupakan nama dari *easy handle* yang ingin dimasukkan ke dalam *multi*
 24 *handle* tertentu dan `<multi handle>` merupakan nama dari *multi handle*-nya sendiri. Sedangkan,
 25 untuk menghapus sebuah *easy handle* dari dari dalam *multi handle*, dapat dipanggil fungsi berikut.

26 `curl_multi_remove_handle(<multi handle>, <easy handle>);`

27 Setelah seluruh *easy handle* yang ingin dijalankan dimasukkan ke dalam *multi handle*, *multi*
 28 *handle* tersebut dapat dijalankan dengan menggunakan sebuah *loop* transfer. Adapun isi dari loop
 29 ini meliputi tiga langkah utama, yaitu:

30 1. Inisialisasi variabel `transfers_running`

31 `transfers_running` merupakan sebuah variabel `integer` yang menjadi penanda bagi pengguna
 32 mengenai berapa banyak *handle* yang sedang melakukan proses transfer di suatu waktu. Selama
 33 nilai variabel ini bukan 0, artinya ada *easy handle* yang belum selesai melakukan proses transfer
 34 berkas.

35 2. Menjalankan transfer dalam *multi handle*

36 Langkah selanjutnya adalah menjalankan transfer dalam *mutli handle*, dengan memanggil
 37 fungsi `curl_multi_perform`. Adapun fungsi tersebut harus dipanggil dengan format berikut,
 38 yaitu:

⁶https://curl.se/libcurl/c/curl_easy_setopt.html

```
1     curl_multi_perform(<multi handle>, <transfers_running>)
```

2 3. Menunggu transfer untuk selesai sebelum data diekstraksi

3 Tentunya sebelum data hasil transfer dapat diekstraksi untuk dipakai, proses transfernya sendiri
 4 harus selesai terlebih dahulu—untuk kasus dalam *multi handle* libcurl, seluruh *handle* di dalam
 5 *multi handle* harus selesai melakukan transfer terlebih dahulu, atau sampai terjadi *timeout*.
 6 Adapun langkah ini dapat diselesaikan entah dengan menggunakan `curl_multi_wait()`
 7 atau `curl_multi_poll()`, atau dengan cara manual, yaitu dengan memasukkan deskriptor-
 8 deskriptor file serta nilai *timeout* secara manual, dan kemudian menggunakan `select()`,
 9 walaupun metode ini tidak dianjurkan karena keterbatasan jumlah deskriptor yang dapat
 10 digunakan.⁷

11 Implementasi singkat dari ketiga langkah ini dapat dilihat di potongan kode berikut.

Kode 2.3: Loop sederhana dari penggunaan *multi handle* curl

```
12 do {  
13     curl_multi_wait (multi_handle, NULL, 0, 1000, NULL);  
14     curl_multi_perform (multi_handle, &transfers_running);  
15 } while (transfers_running);  
16 }
```

18 **Multi Socket Handle**

19 Ada mode lain dari *multi handle*, yaitu *multi socket handle*. Bedanya dengan *multi handle* biasa
 20 adalah *multi socket handle* memperhatikan *socket-socket* yang ada dan memberitahu *handle-handle*
 21 jika ada *socket* yang sudah siap untuk digunakan dalam proses *read/write*. Cara operasinya hampir
 22 sama dengan *multi handle*—hal utama yang berbeda adalah dengan *multi socket handle*, diperlukan
 23 satu buah parameter tambahan, yaitu kumpulan *socket* yang ingin diperhatikan.

24 *Socket* ini, beserta apa aksi yang ingin ditunggu dalam *socket* tersebut, diperhatikan dengan
 25 implementasi sebuah fungsi *callback*, yaitu `socket_callback()`. Handle mana yang ingin digunakan,
 26 *socket* mana yang ingin diperhatikan, serta aksi apa yang ditunggu diatur dalam fungsi ini. Jika
 27 ada beberapa *socket* yang ingin diperhatikan, fungsi ini harus dipanggil lagi untuk setiap *socket*-nya.
 28 Selain itu, perlu juga dipanggil fungsi `timer_callback()`, yang berfungsi untuk mengatur seberapa
 29 lama aplikasi akan menunggu *socket*, sebelum terjadi *timeout*. Kedua *callback* ini dapat diatur ke
 30 dalam suatu *multi handle* dengan menggunakan fungsi `curl_multi_setopt()` biasa—untuk *callback*
 31 *socket*, fungsi ini ditandai dengan menggunakan parameter `CURLOPT_TIMERFUNCTION`, sedangkan
 32 untuk *callback timer*, fungsi tersebut ditandai dengan parameter `CURLOPT_TIMERFUNCTION`.

33 Adapun seluruh *handle* ini dapat dipanggil dengan memanggil fungsi `curl_multi_socket_action()`,
 34 dan cara untuk melihat apakah seluruh transfer sudah selesai atau masih ada transfer yang berlangs-
 35 sung pada suatu waktu sama dengan *multi transfer* biasa.

36 Contoh implementasi singkat dari seluruh fungsi-fungsi tersebut dapat dilihat di potongan kode
 37 berikut.

Kode 2.4: Kumpulan implementasi penggunaan *multi socket handle* curl

```
38 /* socket callback */  
39 int socket_callback(CURL *easy, /* easy handle */  
40                     curl_socket_t s, /* socket */  
41                     int what, /* aksi apa yang ditunggu */  
42 }
```

⁷https://curl.se/libcurl/c/curl_multi_poll.html

```

15     void *userp,      /* penunjuk callback privat */
16     void *socketp)   /* penunjuk socket privat */
17
18 curl_multi_setopt(multi_handle, CURLMOPT_SOCKETFUNCTION, socket_callback);
19
20 /* timer callback */
21 int timer_callback(multi_handle, /* multi handle */
22                     timeout_ms,    /* waktu tunggu dalam milidetik */
23                     userp)         /* penunjuk callback privat */
24
25 curl_multi_setopt(multi_handle, CURLMOPT_TIMERFUNCTION, timer_callback);
26
27 /* multi socket action */
28 curl_multi_socket_action(multi, CURL_SOCKET_TIMEOUT, 0, &running);

```

2.3.3 cJSON⁸

cJSON merupakan sebuah *library* yang berfungsi sebagai *parser* JSON untuk perangkat-perangkat lunak bahasa C. *Library* ini sendiri terdiri atas sebuah file C dan sebuah file header.

Instalasi

cJSON dapat diinstal dengan beberapa cara, yaitu:

- Manual

Instalasi manual hanya membutuhkan pengembang perangkat lunak untuk menyalin kedua file *library* cJSON ke dalam direktori perangkat lunak tersebut.

- CMake

Untuk penggunaan cJSON dengan CMake, perlu dibuat sebuah direktori bernama `build`, dan kemudian CMake harus dijalankan di dalam direktori tersebut, dengan mengeksekusi perintah `cmake`. Dengan melakukan ini, Makefile akan dibuat di dalam direktori tersebut, yang nantinya akan dapat di-*compile* dan diinstal dengan perintah `make install`.

Proses ini akan menginstal file-file *header* ke direktori `/usr/local/include/cjson`, dan file-file *library* ke dalam direktori `usr/local/lib`. Selain itu, file-file untuk `pkg-config` juga akan diinstal untuk memudahkan deteksi instalasi CMake sebelumnya.

Proses pembangunan cJSON juga memiliki beberapa opsi yang dapat diatur sedemikian rupa sesuai dengan kebutuhan pembuat perangkat lunak. Adapun opsi-opsi yang dapat diatur dapat dilihat di daftar berikut.

- `-DENABLE_CJSON_TEST`

Nilai awal: On

Jika opsi ini dinyalakan (diberi nilai “On”), maka tes-tes cJSON akan dibuat dan dijalankan bersamaan dengan instalasi.

- `-DENABLE_CJSON_UTILS`

Nilai awal: Off

Jika opsi ini dinyalakan (diberi nilai “On”), maka file-file utilitas cJSON akan diinstal bersamaan dengan instalasi.

- `-DENABLE_TARGET_EXPORT`

Nilai awal: On

Mengekspor target-target ekspor cJSON. Opsi ini dapat dimatikan jika terjadi masalah saat instalasi.

⁸<https://github.com/DaveGamble/cJSON>

- 1 – `-DENABLE_CUSTOM_COMPILER_FLAGS`
 - 2 **Nilai awal:** On
 - 3 Mengaktifkan properti-properti untuk *compiler* non-standar (Clang, GCC, MSVC). Opsi
 - 4 ini dapat dimatikan jika terjadi masalah saat instalasi.
 - 5 – `-DENABLE_VALGRIND`
 - 6 **Nilai awal:** Off
 - 7 Menjalankan tes-tes yang ada menggunakan Valgrind.
 - 8 – `-DENABLE_SANITIZERS`
 - 9 **Nilai awal:** Off
 - 10 Meng-compile cJSON dengan menyalakan AddressSanitizer dan UndefinedBehaviorSanitizer,
 - 11 jika mungkin.
 - 12 – `-DENABLE_SAFE_STACK`
 - 13 **Nilai awal:** Off
 - 14 Menyalakan SafeStack. Pada saat skripsi ini dibuat, fitur ini hanya didukung untuk
 - 15 *compiler* Clang.
 - 16 – `-DBUILD_SHARED_LIBS`
 - 17 **Nilai awal:** Off
 - 18 Membangun *library-library* umum yang tersedia.
 - 19 – `-DBUILD_SHARED_AND_STATIC_LIBS`
 - 20 **Nilai awal:** On
 - 21 Membangun *library* umum dan *library* statik yang tersedia.
 - 22 – `-DCMAKE_INSTALL_PREFIX`
 - 23 **Nilai awal:** -
 - 24 Mengatur *prefix* direktori tempat instalasi cJSON.
 - 25 – `-DENABLE_LOCALES`
 - 26 **Nilai awal:** On
 - 27 Memungkinkan penggunaan metode `localeconv`.
 - 28 – `-DCJSON_OVERRIDE_BUILD_SHARED_LIBS`
 - 29 **Nilai awal:** On
 - 30 Memungkinkan penimpaan nilai dari opsi `-BUILD_SHARED_LIBS` menggunakan nilai dari
 - 31 opsi `-DCJSON_BUILD_SHARED_LIBS`.
 - 32 – `-DENABLE_CJSON_VERSION_SO`
 - 33 **Nilai awal:** On
 - 34 Menyalakan versi so dari cJSON.
- 35 • Makefile
 - 36 Jika CMake tidak tersedia, cJSON juga dapat dibangun dengan menggunakan GNU Make,
 - 37 dengan menggunakan perintah `make all`, dan menginstal *library-library* yang sudah ter-
 - 38 *compile* dengan perintah `make install`. Akan tetapi, perlu diingat bahwa metode instalasi
 - 39 ini sudah tidak lagi diperbarui (*deprecated*), dan dukungannya hanya sebatas pembetulan *bug*.
 - 40 • vcpkg
 - 41 Melalui vcpkg, cJSON dapat diunduh dan diinstal secara langsung. *Port* dari cJSON di vcpkg
 - 42 terus diperbarui oleh tim Microsoft dan kontributor-kontributor dari komunitas publik, jadi

versi dari cJSON yang diinstal melalui vcpkg kemungkinan besar akan selalu versi terbarunya.

2 Penggunaan

3 Jika cJSON diinstal melalui CMake atau Makefile, cJSON dapat digunakan dengan mengikutkan
4 baris ini dalam kode program:

```
5 #include <cjson/cJSON.h>
```

6 Struktur Data

7 cJSON merepresentasikan sebuah nilai JSON dengan struktur data cJSON, yang dapat dilihat di
8 potongan kode 2.5.

Kode 2.5: Struktur data cJSON

```
9
10 typedef struct cJSON
11 {
12     struct cJSON *next;
13     struct cJSON *prev;
14     struct cJSON *child;
15     int type;
16     char *valuestring;
17     int valueint;
18     double valuedouble;
19     char *string;
20 } cJSON;
```

22 Dengan variabel-variabel dalam struktur tersebut sebagai berikut:

- 23 • **next** dan **prev**

24 Variabel ini merupakan penunjuk ke struktur cJSON lainnya yang merupakan nilai JSON
25 selanjutnya (untuk **next**) dan sebelumnya (untuk **prev**).

- 26 • **child**

27 Variabel ini merupakan penunjuk ke struktur cJSON lainnya yang merupakan elemen JSON
28 di dalam struktur cJSON tersebut.

- 29 • **type**

30 Variabel ini menandakan tipe dari nilai JSON yang terdapat di dalam struktur cJSON tersebut.
31 Adapun tipe-tipe yang mungkin adalah sebagai berikut.

- 32 – **cJSON_Invalid**

33 Merepresentasikan sebuah objek yang tidak valid dan tidak bernilai. Jika seluruh *field*
34 diatur sehingga berisi 0 byte, maka variabel **type** akan otomatis berisi tipe ini.

- 35 – **cJSON_True**

36 Merepresentasikan nilai boolean *true*.

- 37 – **cJSON_False**

38 Merepresentasikan nilai boolean *false*.

- 39 – **cJSON_Number**

40 Merepresentasikan nilai numerik apapun. Jika nilainya merupakan nilai *double*, maka
41 nilai tersebut akan disimpan di dalam variabel **valuedouble**. Sedangkan jika nilainya
42 merupakan nilai *integer*, maka nilai tersebut akan disimpan di dalam variabel **valueint**.

- 43 – **cJSON_String**

44 Merepresentasikan nilai *string* apapun. Nilainya disimpan sebagai *string* yang dipisah
45 dengan *null terminator* ('\0' atau \u0000).

1 – **cJSON_Array**

2 Merepresentasikan nilai *array*. *Array* dalam cJSON diimplementasikan dengan menunjukkan isi dari variabel **child** tadi ke sebuah *linked list* dari objek-objek cJSON. Jika objek cJSON ini merupakan elemen dalam sebuah *array*, maka isi dari variabel **prev** dan **next** merupakan salah satu dari elemen-elemen lain dalam *array* yang menjadi elemen sebelum dan sesudah elemen ini.

7 – **cJSON_Object**

8 Merepresentasikan nilai objek general. Implementasinya sama dengan *array*, hanya saja 9 untuk objek general, kuncinya diletakkan di dalam variabel **string**.

10 – **cJSON_Raw**

11 Merepresentasikan objek JSON apapun yang disimpan dalam bentuk *array* yang diterminasi/dipisah oleh *null terminator*.

13 – **cJSON_NULL**

14 Merepresentasikan sebuah nilai *null*.

15 • **valuestring/valuestring/valuestring/string**

16 Merupakan variabel-variabel yang menyimpan nilai dari struktur cJSON. Variabel apa yang 17 diisi dan apa isinya tergantung dari tipe data yang disimpan.

18 **Fungsi-Fungsi Dasar**

19 • Tipe-tipe data dasar

20 Untuk setiap tipe data cJSON, ada sebuah fungsi **cJSON_Create.....** Semua fungsi tersebut akan mengalokasikan sebuah struktur cJSON yang nantinya dapat dihapus dengan **cJSON_Delete**. Perlu diingat juga bahwa seluruh struktur yang dibuat harus dihapus agar tidak terjadi kebocoran memori. Adapun fungsi-fungsi **cJSON_Create** yang dapat digunakan untuk tipe-tipe data yang tersedia adalah sebagai berikut.

25 – *null* dibuat dengan **cJSON_CreateNull**.

26 – *boolean* dibuat dengan **cJSON_CreateBool**, atau jika ingin membuat data *boolean* langsung dengan nilainya, dapat menggunakan **cJSON_CreateTrue** atau **cJSON_CreateFalse**.

28 – Bilangan apapun dibuat dengan **cJSON_CreateNumber**. Penggunaan fungsi ini akan langsung mengisi nilai variabel **valueint** dan **valuedouble**.

30 – *string* apapun dibuat dengan **cJSON_CreateString** (membuat sebuah string langsung sebagai nilai data JSON), atau **cJSON_CreateStringReference** (mereferensikan *string* yang sudah ada sebagai nilai data JSON).

33 • *Array*

34 Sebuah *array* kosong dapat dibuat dengan menggunakan fungsi **cJSON_CreateArray**. Selain fungsi tersebut, pembuatan *array* juga dapat dilakukan dengan memanggil sebuah fungsi alternatif, yaitu **cJSON_CreateArrayReference**. Bedanya adalah dengan fungsi alternatif ini, *array* yang dibuat tidak secara langsung “mengandung” nilai-nilainya, jadi ketika *array* tersebut dihapus dengan **cJSON_Delete**, nilai-nilai di dalamnya tidak terhapus juga. Hal yang sama juga dapat dilakukan dengan objek-objek di dalam *array* tersebut, dimana objek ini dapat ditambahkan dengan fungsi **cJSON.AddItemToArray**, yang akan langsung menambahkan objek tersebut ke dalam *array*-nya, atau dengan menggunakan

`cJSON_AddItemToArray`, yang hanya akan menambahkan referensi dari objek yang ingin ditambahkan ke dalam *array*. Jika ada sebuah objek yang ingin dihapus dari *array* tertentu, perangkat lunak dapat memanggil fungsi `cJSON_DetachItemFromArray`. Fungsi ini akan mengembalikan objek yang dihapus dari *array* tadi, jadi objek ini harus diberikan ke sebuah penunjuk lainnya (dimasukkan ke dalam variabel lain) agar tidak terjadi kebocoran memori. Selain fungsi tersebut, fungsi `cJSON_DeleteItemFromArray` juga dapat digunakan—bedanya adalah objek yang dihapus dari *array* tadi, jika dihapus menggunakan fungsi ini, akan langsung dihapus, seakan-akan fungsi `cJSON_Delete` dipanggil untuk objek tersebut. Untuk menimpa/mengganti nilai suatu objek di dalam *array*, fungsi `cJSON_ReplaceItemInArray` dapat dipanggil dengan indeks dari objek yang ingin diganti sebagai parameternya. Selain fungsi tersebut, fungsi `cJSON_ReplaceItemViaPointer` juga dapat digunakan—fungsi ini bekerja dengan memutuskan (*detach*) objek lama yang ingin diganti, menghapus objek tersebut, dan menyisipkan objek yang baru ke tempat objek lama yang sudah dihapus tadi. Terakhir, untuk mendapatkan objek tertentu dalam *array* berdasarkan indeksnya, perangkat lunak dapat memanggil fungsi `cJSON_GetArrayItem`. Ukuran dari *array*-nya sendiri juga dapat dilihat dengan fungsi `cJSON_GetArraySize`.

- Objek

Sama seperti *array*, ada dua jenis fungsi pembuatan objek. Yang pertama adalah `cJSON_CreateObject` untuk membuat objek biasa, dan `cJSON_CreateObjectReference` untuk membuat objek yang nilai-nilainya terdiri atas kumpulan referensi ke variabel-variabel lain. Untuk menambahkan sebuah objek ke dalam suatu objek lainnya, pengguna dapat memanggil fungsi `cJSON.AddItemToObject`. Jika objek ingin ditambahkan ke suatu objek lain yang memiliki sebuah variabel konstan sebagai nama, atau sebuah referensi, prosesnya harus menggunakan fungsi `cJSON.AddItemToObjectCS`. Fungsi `cJSON_DetachItemFromObjectCaseSensitive` dapat dipanggil ketika ada sebuah objek ingin dibuang dari objek lain. Sama seperti fungsi *detach* di *array* tadi, fungsi ini akan mengembalikan objek yang dibuang tadi, dan objek tersebut harus dimasukkan ke variabel lain untuk menghindari kebocoran memori. Selain itu, ada juga fungsi `cJSON_DeleteItemFromObjectCaseSensitive`, yang bekerja dengan cara yang sama seperti fungsi penghapusan objek untuk *array*. Untuk pengeditan/penggantian nilai objek, lagi-lagi cara kerjanya sama dengan *array*. Untuk penggantian nilai objek berdasarkan kuncinya, fungsi `cJSON_ReplaceItemInObjectCaseSensitive` dapat digunakan, sedangkan untuk penggantian objek langsung dengan penunjuk ke elemen lainnya, fungsi `cJSON_ReplaceItemViaPointer` dapat digunakan. Terakhir, untuk mengakses sebuah benda di dalam objek, pengguna dapat memanggil fungsi `cJSON_GetObjectItemCaseSensitive`, dan untuk mengetahui ukuran dari objek tersebut, fungsi yang dapat digunakan sama dengan fungsi yang dapat digunakan untuk *array*, yaitu `cJSON_GetArraySize`.

- *Parsing*

Objek JSON yang dibatasi/diterminasi dengan *null terminator* dapat di-parse dengan menggunakan fungsi berikut.

```
40      cJSON_Parse(<JSON>)
```

Sedangkan, jika objek JSON tersebut tidak (atau belum tentu) dibatasi oleh *null terminator*, objek tersebut dapat di-parse dengan fungsi berikut.

1 cJSON_Parse(<JSON>, <ukuran JSON yang ingin di-parse>)

2 Kedua fungsi ini akan membuat sebuah struktur hierarkial dari objek-objek cJSON yang
3 merepresentasikan keseluruhan dari objek JSON tersebut. Setelah objek ini selesai digunakan,
4 penggunanya harus mendealokasikan objek tersebut dengan fungsi cJSON_Delete.

5 **2.3.4 CMake [3]**

6 CMake merupakan sebuah perkakas generator *build system* yang memungkinkan pengembang
7 perangkat lunak untuk menentukan parameter-parameter pembangunan perangkat di sebuah file
8 yang berupa teks biasa. File ini kemudian dipakai oleh CMake untuk membuat perkakas-perkakas
9 pembangunan perangkat lunak yang dapat dibaca oleh IDE-IDE tertentu, seperti Microsoft Visual
10 Studio, Apple Xcode, Linux, dan sebagainya. Selain itu, CMake juga menangani aspek-aspek
11 rumit dari pembangunan perangkat lunak, seperti pembangunan perangkat lunak *cross-platform*,
12 introspeksi sistem, dan juga pembangunan perangkat lunak yang dapat disesuaikan berdasarkan
13 penggunaanya.

14 Untuk proyek-proyek yang dibangun di satu platform, CMake memiliki fungsi sebagai berikut.

- 15 • Memberikan kemampuan untuk melakukan pencarian seluruh perangkat lunak, file-file *library*,
16 dan file-file *header* yang dibutuhkan untuk proses pembangunan perangkat lunak. Pencarian
17 ini juga dapat dilakukan sampai ke dalam *registry* sistem.
- 18 • Memungkinkan pembangunan perangkat lunak diluar folder tempat *source code* perangkat
19 lunak tersebut sendiri.
- 20 • Memberikan kemampuan untuk membuat perintah-perintah khusus untuk file-file yang dibuat
21 secara otomatis, seperti `moc()` dari Qt, atau generator pembungkus dari SWIG. Perintah-
22 perintah ini dapat mengatur pembuatan file *source* baru yang nantinya akan diintegrasikan
23 langsung ke dalam perangkat lunak akhirnya.
- 24 • Memberikan kemampuan untuk memilihkan file-file opsional dari *library* yang digunakan.
- 25 • Memungkinkan pengaturan pembuatan proyek dari sebuah file `.txt` sederhana.
- 26 • Kemampuan untuk dengan mudah beralih dari *static build* dan *shared build*.
- 27 • Membuat ketentuan keperluan (*dependency*) secara otomatis.

28 Sedangkan, untuk perangkat-perangkat lunak yang dibuat *cross-platform*, CMake memberikan
29 fungsi-fungsi tambahan sebagai berikut.

- 30 • Memberikan kemampuan mengetes urutan *machine byte* dan karakteristik-karakteristik lainnya
31 yang spesifik untuk suatu sistem operasi.
- 32 • File konfigurasi yang dibuat dapat berlaku untuk seluruh *platform*.
- 33 • Memberikan dukungan untuk membangun *shared build* untuk seluruh *platform* yang mendu-
34 kungnya.
- 35 • Memberikan kemampuan untuk mengatur opsi-opsi yang spesifik untuk suatu sistem operasi,
36 seperti misalnya lokasi file-file data utama.

37 **Penggunaan Dasar**

38 CMake mengambil satu (atau lebih) file-file CMakeLists sebagai masukan, dan sebagai keluaran akan
39 menghasilkan file-file proyek atau Makefile yang dapat digunakan dengan dan oleh perkakas-perkakas
40 pembangunan perangkat lunak lain yang ada.

1 Proses CMake umumnya terdiri atas langkah-langkah berikut.

2 1. Proyek yang ingin dibangun didefinisikan di salah satu file CMakeLists.

3 File-file CMakeLists (yang berupa file-file .txt) merupakan file-file *plain text* yang berisi
4 deskripsi proyek dalam bahasa CMake. Tertera di kode 2.6 merupakan file CMakeLists
5 yang dapat digunakan untuk membangun sebuah program “*Hello World*” sederhana dalam
6 bahasa C, sebagai gambaran mengenai hal-hal apa saja yang minimal harus ada di dalam file
7 CMakeLists.

Kode 2.6: Kode utama operasional CMakeLists

```
8
9 1 cmake_minimum_required(VERSION 3.20)
10 2 project(Hello)
11 3 add_executable(Hello Hello.c)
```

13 Penjelasan dari baris-baris berikut adalah sebagai berikut.

- 14 • Baris pertama harus selalu merupakan `cmake_minimum_required`. Hal ini mengharuskan
15 proyek untuk menggunakan versi dari CMake yang dispesifikasi, dan juga memungkinkan
16 *backwards compatibility*.
- 17 • Baris selanjutnya merupakan perintah `project`. Perintah ini mengatur nama proyek, dan
18 juga bisa digunakan untuk mengatur aturan-aturan lainnya, seperti versi, atau bahasa
19 dari proyek. Untuk setiap direktori yang memiliki file CMakeLists, CMake akan membuat
20 sebuah Makefile atau file proyek untuk IDE. Proyek ini, nantinya, akan mengikutkan
21 seluruh direktori yang memiliki file CMakeLists ini serta seluruh subdirektori yang
22 diikutkan dalam perintah `add_subdirectory`.
- 23 • Terakhir, perintah `add_executable` merupakan sebuah perintah yang akan menambahkan
24 sebuah file *executable* untuk menjalankan proyek yang sudah dibangun tersebut.

25 2. CMake membuat dan mengkonfigurasi proyek tersebut.

26 Setelah file-file CMakeLists selesai dibuat, CMake akan memproses file-file tersebut dan
27 membuat entri-entri dalam sebuah file *cache*. Pengembang perangkat lunak dapat mengedit
28 file CMakeLists atau mengatur isi dari file *cache* tadi dengan GUI CMake, ccmake, atau untuk
29 proyek-proyek skala kecil, langsung dari *command line*.

- 30 • GUI CMake

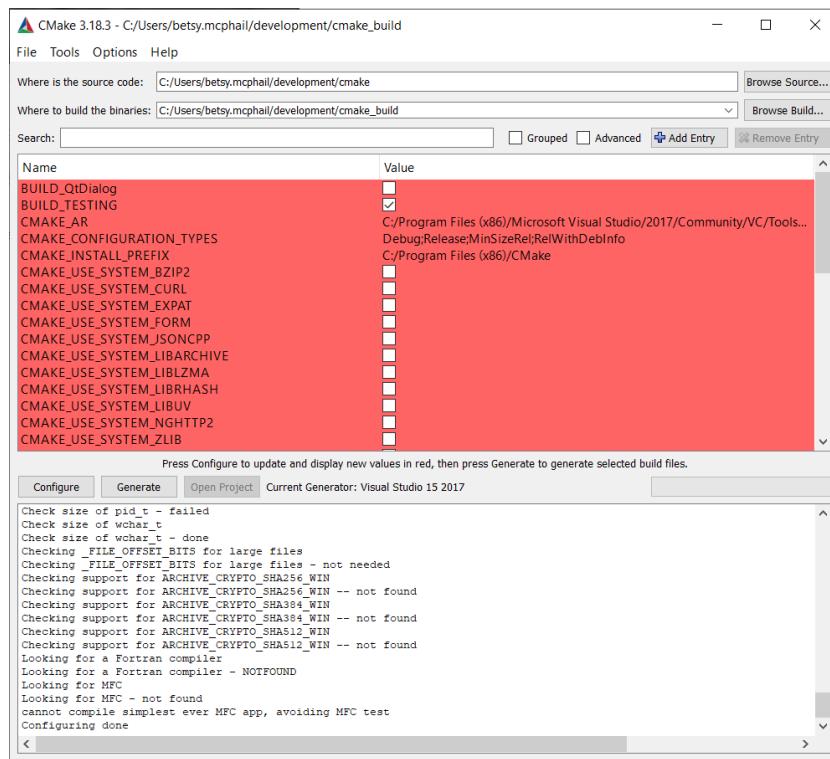
31 CMake memiliki perangkat lunak GUI berbasis Qt yang bisa digunakan di sebagian
32 besar sistem operasi, seperti UNIX, Mac OS X, dan Windows. Perangkat lunak ini,
33 `cmake-gui`, sudah terinstal bersama dengan CMake, tetapi membutuhkan instalasi Qt
34 untuk dijalankan. Adapun tampilan dari perangkat lunak ini dapat dilihat di gambar
35 2.10.

36 Dua *field* paling atas adalah direktori dari *source code* dan direktori tempat file-file
37 *binary* nantinya akan diletakkan setelah dibuat. Kedua *field* ini harus diisi secara
38 manual, walaupun jika direktori *binary* ini sudah dikonfigurasi langsung melalui CMake
39 sebelumnya, *field* direktori kedua akan secara otomatis terisi.

- 40 • ccmake

41 Di mayoritas sistem operasi berbasis UNIX, jika *library curses* didukung, maka CMake
42 memiliki sebuah perangkat lunak lain yang dapat digunakan, yaitu `ccmake`. Untuk
43 menjalankan `ccmake`, pengguna dapat menjalankannya melalui *command line*, dan

⁹<https://cmake.org/cmake/help/book/mastering-cmake/chapter/Getting%20Started.html>

Gambar 2.10: Tampilan aplikasi cmake-gui.⁹

direktori tempat `ccmake` dijalankan ini harus merupakan direktori tempat file-file *binary* nantinya ingin disimpan. Ketika aplikasi ini dijalankan, akan keluar tampilan seperti di gambar 2.11. Adapun instruksi singkat penggunaan dari aplikasi ini dapat dilihat di bagian bawah dari tampilan tersebut.

```
Page 1 of 3
BUILD_CursesDialog      *OFF
BUILD_QtDialog          *OFF
BUILD_TESTING           *ON
CMAKE_BUILD_TYPE        *
CMAKE_INSTALL_PREFIX    */usr/local
CMAKE_USE_SYSTEM_BZIP2   *OFF
CMAKE_USE_SYSTEM_CURL   *OFF
CMAKE_USE_SYSTEM_EXPAT   *OFF
CMAKE_USE_SYSTEM_FORM   *OFF
CMAKE_USE_SYSTEM_JSONCPP *OFF
CMAKE_USE_SYSTEM_LIBARCHIVE *OFF
CMAKE_USE_SYSTEM_LIBLZMA *OFF
CMAKE_USE_SYSTEM_LIBRHASH *OFF
CMAKE_USE_SYSTEM_LIBUV   *OFF
CMAKE_USE_SYSTEM_ZLIB    *OFF
CMAKE_USE_SYSTEM_ZSTD    *OFF
CMake_BUILD_LTO          *OFF

BUILD_CursesDialog: Build the CMake Curses Dialog ccmake
Press [enter] to edit option Press [d] to delete an entry  CMake Version 3.16.2
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Gambar 2.11: Tampilan aplikasi ccmake.¹⁰

- Langsung dari *command line*

CMake juga dapat dijalankan melalui *command line*. Untuk menjalankan CMake dari *command line*, direktori tempat terminal berada lagi-lagi harus diatur ke direktori tempat file-file *binary* akan disimpan. Kemudian jalankan perintah `cmake` dengan opsi `-D`, diikuti

¹⁰<https://cmake.org/cmake/help/book/mastering-cmake/chapter/Getting%20Started.html>

dengan direktori tempat *source code* dari perangkat lunak yang ingin dibangun berada. Walaupun begitu, perlu diingat bahwa metode ini direkomendasikan untuk digunakan hanya untuk proyek-proyek yang memiliki sedikit, atau bahkan tidak ada opsi sama sekali.

3. Pengguna membangun proyek tersebut dengan perkakas pembangunan masing-masing. CMake dapat memberikan beberapa opsi dalam pembangunan perangkat lunak yang dapat diatur oleh penggunanya masing-masing. Adapun dua opsi utama dari seluruh opsi-opsi tersebut adalah sebagai berikut.

- Menspesifikasi *compiler* yang akan digunakan

Di beberapa sistem, bisa jadi terdapat lebih dari satu *compiler*, atau *compiler* yang ada tidak berada di tempat *compiler* tersebut biasa diinstal. Di kasus-kasus ini, CMake perlu diberitahu secara manual dimana letak *compiler* yang ingin digunakan. Ada tiga cara untuk melakukan ini, yaitu dengan:

- menspesifikasikan *compiler* yang ingin dipakai ke generator,
- mengatur variabel *environment*, atau
- membuat entri *cache*.

Jika pengaturan *compiler* sudah selesai dan `cmake` sudah dijalankan setidaknya sekali, jika pengguna ingin mengganti *compiler*, pengguna harus memulai ulang dengan folder file *binary* yang kosong.

- Mengatur konfigurasi pembangunan perangkat

Konfigurasi pembangunan perangkat memungkinkan sebuah proyek untuk dibangun dalam beberapa cara dengan tujuan *debugging*, optimisasi, atau tujuan-tujuan sejenis lainnya. Secara *default*, CMake mendukung mode-mode berikut:

- `Debug`—Opsi-opsi *debug* dasar dinyalakan.
- `Release`—Fungsi optimisasi dasar dinyalakan.
- `MinSizeRel`—Ukuran kode akhir diusahakan sekecil mungkin.
- `RelWithDebinfo`—Membangun *build* optimal dan mengikutkan informasi-informasi untuk *debugging*.

Dengan generator-generator berbasis Makefile, hanya sebuah mode konfigurasi yang bisa aktif ketika CMake sedang dijalankan, dan mode tersebut diatur dengan variabel `CMAKE_BUILD_TYPE`. Jika variabel ini dikosongkan (atau tidak dimasukkan ke dalam kode), maka mode yang dipakai adalah mode *default*. Di lain kasus, jika pengembang ingin membangun perangkat dalam dua mode yang berbeda, perintah untuk menjalankan CMake (dengan variabel mode konfigurasi masing-masing) harus dijalankan untuk setiap modenya. Misal, jika pengguna ingin membangun perangkat dengan mode `Debug` dan `Release` sekaligus, maka perintah untuk menjalankan CMake harus ditulis seperti dapat dilihat di potongan kode 2.7.

Kode 2.7: Kode utama operasional CMakeLists

```
1 ccmake ../../<direktori> -DCMAKE_BUILD_TYPE=Debug
2 ccmake ../../<direktori> -DCMAKE_BUILD_TYPE=Release
```

Setelah CMake dijalankan, proyek tersebut siap untuk dibangun. Jika generator yang dipilih merupakan generator berbasis Makefiles, maka proyek tersebut dapat dibangun dengan

1 mengganti *working directory* ke lokasi file-file *binary*, dan kemudian menjalankan perintah
2 `make`. Jika generator yang dipilih merupakan sebuah IDE, maka proyek tersebut dapat
3 dibangun secara biasa melalui IDE tersebut.

¹

BAB 3

²

ANALISIS

³ 3.1 Analisis Aplikasi Sejenis

⁴ Untuk pembuatan perangkat lunak dalam skripsi ini, ada empat buah perkakas *command line*
⁵ yang akan diamati sebagai aplikasi sejenis. Dua dari empat aplikasi pertama adalah *Chrome Web*
⁶ *Store Item Property CLI* dan *iTunes Search API*. Selain dua perkakas tersebut, ada dua perkakas
⁷ lainnya yang bisa digunakan sebagai referensi, tetapi tidak dapat dieksplorasi, karena kedua aplikasi
⁸ tersebut tidak berhasil dijalankan dengan sempurna, yaitu *Uber CLI* dan *Google Maps Direction*
⁹ CLI

¹⁰ **3.1.1 Chrome Web Store Item Property CLI¹**

¹¹ Perkakas *command line* ini merupakan ekstensi dari sebuah aplikasi lain yang memiliki fungsi
¹² yang sama, yaitu *Chrome Web Store Item Property*.² Perangkat lunak *Chrome Web Store Item*
¹³ *Property CLI* ini merupakan perangkat lunak yang akan memanggil fungsi API untuk mendapatkan
¹⁴ metadata dari sebuah ekstensi pada *web store* peramban Google Chrome. Perbedaan dari perkakas
¹⁵ ini dengan aplikasi dasarnya adalah bahwa perkakas ini dapat digunakan sebagai perkakas *command*
¹⁶ *line*, sedangkan aplikasi dasarnya hanya bisa digunakan dalam perangkat lunak lainnya sebagai
¹⁷ pemanggil fungsi API.

¹⁸ **Penggunaan**

¹⁹ Perkakas ini dapat digunakan melalui *command prompt* dengan cara mengetikkan perintah sebagai
²⁰ berikut.

²¹ `chrome-web-store-item-property <identifier>`

²² Dengan *identifier* berupa ID dari ekstensi yang diinginkan. Jadi, misalkan pengguna mema-
²³ sukkan `gighmmypiobklfepjocnamgkkbiglidom` sebagai ID yang akan digunakan sebagai *identifier*,
²⁴ maka perkakas ini akan mengembalikan metadata dari ekstensi “AdBlock” sebagai keluarannya.
²⁵ Contoh penggunaan perkakas ini dapat dilihat di gambar 3.1.

²⁶ Sedangkan, keluaran dari perkakas ini merupakan sebuah objek JSON dengan properti-properti
²⁷ sebagai berikut.

¹<https://github.com/pandawing/node-chrome-web-store-item-property-cli>

²<https://github.com/pandawing/node-chrome-web-store-item-property>

```
C:\Windows\system32\cmd.exe
Your environment has been set up for using Node.js 16.14.2 (x64) and npm.

C:\Users\Alfred>chrome-web-store-item-property gighmmpioblkfepjocnamgkkbiglidom
{"name": "AdBlock _ best ad blocker", "url": "https://chrome.google.com/webstore/detail/adblock-%E2%80%94-best-ad-blocker/gighmmpioblkfepjocnamgkkbiglidom", "image": "https://lh3.googleusercontent.com/mgNKV-3VMD556WVuIwSpukQON-iI4Zlqq83ef1jG2B5j9VP7fxr3idQ_G0JFD/E604IMwvIQ1leDn_80dFlt5VQ=w128-h128-e365-rj-sc0x0ffff", "version": "4.46.1", "price": "0", "priceCurrency": "USD", "interactionCount": {"UserDownloads": 10000000}, "operatingSystem": "Chrome", "id": "gighmmpioblkfepjocnamgkkbiglidom"}

C:\Users\Alfred>
```

Gambar 3.1: Contoh penggunaan perkakas Chrome *Web Store Item Property* CLI.

- **name**
Nama dari ekstensi yang dicari metadatanya.
- **url**
URL halaman web dari ekstensi yang dicari di *web store* Google Chrome.
- **image**
Logo (dan ikon *thumbnail*) dari ekstensi yang dicari metadatanya.
- **version**
Nomor versi dari ekstensi.
- **price**
Harga dari ekstensi. Jika ekstensi tidak memiliki harga yang perlu dibayarkan (gratis), properti ini akan bernilai 0.
- **priceCurrency**
Kode mata uang dari harga ekstensi. Jika ekstensi tidak memiliki harga yang perlu dibayarkan, properti ini akan berisi “USD”.
- **interactionCount**
Properti ini berisi interaksi-interaksi pengguna yang tercatat sebagai data di halaman *web store* ekstensi. Pada saat pembuatan skripsi ini, properti ini hanya memiliki satu buah subproperti, yaitu **userDownloads**, yang menandakan berapa kali ekstensi ini telah diunduh oleh pengguna di manapun.
- **operatingSystems**
Menandakan di peramban mana ekstensi versi ini dapat diinstal. Karena ekstensi-ekstensinya berada di *web store* Chrome,
- **ratingValue** (tidak digunakan lagi)
Peringkat yang diberikan oleh para pengguna ekstensi ini. Nilai dari properti ini berupa skala desimal dari 0.00 sampai dengan 5.00. Di versi terbaru dari perkakas ini, properti ini tidak lagi tersedia dalam keluarannya.
- **ratingCount** (tidak digunakan lagi)
Jumlah pengguna yang telah menilai/memberi peringkat ke ekstensi ini. Di versi terbaru dari perkakas ini, properti ini tidak lagi tersedia dalam keluarannya.
- **id**
Properti ini mengandung ID dari ekstensi tersebut. Nilai dari properti ini akan sama dengan ID yang digunakan sebagai parameter masukan perkakas.

¹ 3.1.2 iTunes Search API³

² Perkakas *command line* ini berfungsi untuk melakukan pencarian melalui API iTunes, sehingga
³ seakan-akan pengguna langsung melakukan pencarian di iTunes sendiri. Hasil pencarian yang
⁴ dilakukan termasuk judul lagu, nama artis, ataupun nama album, dan pengguna dapat memilih
⁵ secara spesifik objek apa yang ingin dicari.

⁶ Penggunaan

⁷ Perkakas ini dapat digunakan melalui *command prompt* dengan cara mengetikkan perintah sebagai
⁸ berikut.

⁹ `itunes-search-api <input> [<options>]`

¹⁰ Dengan **input** berupa nama dari objek yang dicari. Perkakas ini juga memiliki opsi yang masing-
¹¹ masing memiliki parameter tersendiri untuk mempersempit hasil pencarian. Adapun opsi-opsi
¹² tersebut dapat dilihat di daftar di bawah ini.

- ¹³ • **country**

¹⁴ **Kemungkinan nilai:** Kode negara dua huruf

¹⁵ Opsi ini menerima parameter berupa kode negara asal dari album atau artis yang dicari.

- ¹⁶ • **entity**

¹⁷ **Kemungkinan nilai:** song, musicArtist, atau album

¹⁸ Menandakan jenis objek/entitas yang ingin dicari. Opsi ini dapat bernilai **song** untuk pencarian
¹⁹ berbasis judul lagu, **musicArtist** untuk pencarian nama artis, atau **album** untuk pencarian
²⁰ nama album. Jika opsi ini tidak dipakai, objek apapun yang memiliki kemiripan dengan
²¹ **input** dalam salah satu dari ketiga properti ini akan muncul dalam hasil pencarian.

- ²² • **limit**

²³ **Kemungkinan nilai:** Bilangan bulat positif⁴

²⁴ Jumlah hasil pencarian maksimal yang ingin ditampilkan dalam keluaran.

²⁵ Sedangkan, keluaran dari perkakas ini merupakan sebuah objek JSON yang memiliki dua properti
²⁶ utama, yaitu:

- ²⁷ • **resultCount**

²⁸ Properti ini berisi bilangan bulat yang menandakan berapa buah objek yang terdapat dalam
²⁹ hasil pencarian.

- ³⁰ • **results**

³¹ *Array* yang berisi kumpulan objek yang terdapat di dalam hasil pencarian. Objek-objek ini
³² akan dikembalikan berupa sebuah *array* lain yang berisi seluruh properti dari masing-masing
³³ objek. Apa saja properti yang diikutkan dalam *array* tersebut tergantung tipe dari objek
³⁴ dalam hasil pencarian.

³⁵ Adapun contoh penggunaan dan hasil keluaran perkakas ini dapat dilihat di gambar 3.2.

³<https://github.com/awcross/itunes-search-api>

⁴Opsi ini juga menerima bilangan bulat negatif, tetapi menggunakan sebuah bilangan bulat negatif akan menghilangkan pengaruh opsi ini terhadap hasil keluaran.

```
C:\Windows\system32\cmd.exe
Your environment has been set up for using Node.js 16.14.2 (x64) and npm.
C:\Users\Alfred>itunes-search-api 'the presets'
{
  resultCount: 25,
  results: [
    {
      wrapperType: 'collection',
      collectionType: 'Album',
      artistId: 78745197,
      collectionId: 1444068937,
      amgArtistId: 748856,
      artistName: 'The Presets',
      collectionName: 'Apocalypso',
      collectionCensoredName: 'Apocalypso',
      artistViewUrl: 'https://music.apple.com/us/artist/the-presets/78745197?uo=4',
      collectionViewUrl: 'https://music.apple.com/us/album/apocalypso/1444068937?uo=4',
      artworkUrl60: 'https://is4-ssl.mzstatic.com/image/thumb/Music14/v4/dd/f7/1c/ddf71c93-692b-2824-606c-226f6eb0cfa/00602517570245.rgb.jpg/0x60bb.jpg',
      artworkUrl100: 'https://is4-ssl.mzstatic.com/image/thumb/Music14/v4/dd/f7/1c/ddf71c93-692b-2824-606c-226f6eb0cfa2/00602517570245.rgb.jpg/100x100bb.jpg',
      collectionPrice: 5.99,
      collectionExplicitness: 'notExplicit',
      trackCount: 11,
      copyright: '© 2008 Modular Recordings',
      country: 'USA',
      currency: 'USD',
      releaseDate: '2008-01-01T08:00:00Z',
      primaryGenreName: 'Electronic'
    },
  ]
}
```

Gambar 3.2: Contoh penggunaan perkakas *iTunes Search API*. Gambar hanya memuat satu objek untuk menghemat tempat.

¹ 3.1.3 Uber CLI⁵

² Uber CLI merupakan sebuah perkakas *command line* yang dapat digunakan untuk dua fungsi
³ utama. Fungsi pertama dari perkakas ini adalah untuk mendapatkan estimasi untuk seberapa lama
⁴ waktu yang diperlukan untuk servis taksi *online* dari Uber untuk mencapai lokasi yang ingin dituju,
⁵ sedangkan fungsi keduanya adalah untuk mengestimasi berapa harga yang harus dibayarkan untuk
⁶ memakai servis tersebut.

⁷
⁸ Fungsi yang pertama dapat dilakukan memanggil perintah dengan format sebagai berikut.

⁹ uber time <alamat>

¹⁰ uber merupakan perintah dasar dari perkakas ini. time merupakan parameter yang menandakan
¹¹ bahwa pengguna ingin menggunakan servis prediksi waktu dari perkakas ini. Selain itu, pengguna
¹² harus memasukkan alamat yang ingin dituju sebagai parameter akhir dari perintah yang akan
¹³ digunakan sebagai masukan. Jika sintaksnya sudah benar, perintah tersebut akan bisa diproses oleh
¹⁴ perkakas dengan cara mengirimkan pesan hasil konversi perintah tersebut ke API Uber. Setelah
¹⁵ pemrosesan pesan tersebut berhasil, perkakas ini akan menampilkan sebuah keluaran dengan format
¹⁶ yang dapat dilihat di gambar 3.3. Perlu diperhatikan juga bahwa keluaran yang dihasilkan oleh
¹⁷ perkakas ini akan meliputi seluruh jenis servis yang disediakan oleh Uber.

⁵<https://github.com/jaebradley/uber-cli>

⁶<https://github.com/jaebradley/uber-cli>

21:00 \$ uber time '25 first street cambridge ma'	
25 First St, Cambridge, MA 02141, USA	
 	
2 min. uberPOOL,uberX	
3 min. uberXL	
5 min. UberBLACK,uberSUV	
7 min. TAXI	

Gambar 3.3: Contoh penggunaan fitur prediksi waktu perjalanan untuk perkakas Uber CLI.⁶

- 1 Sedangkan, untuk memanggil fungsi kedua dari perkakas ini, pengguna dapat dilakukan dengan
- 2 memanggil perintah dengan format berikut.

3 uber price -s <alamat awal> -e <alamat akhir>

- 4 Untuk sintaks ini, **uber** memiliki fungsi yang sama dengan sintaks untuk fungsi pertama dari
 5 perkakas. **price** merupakan penanda untuk perkakas bahwa pengguna ingin menggunakan servis
 6 untuk mengetahui perkiraan harga layanan Uber. Selanjutnya, perkakas akan meminta dua buah
 7 opsi beserta parameternya masing-masing. Pertama, opsi **-s**, berarti *start*, yang akan meminta
 8 sebuah parameter berupa lokasi yang ingin dipakai sebagai lokasi awal perkiraan harga layanan
 9 Uber. Sedangkan opsi **-e**, berarti *end*, akan meminta sebuah parameter berupa lokasi yang ingin
 10 dipakai sebagai lokasi akhir jasa perkiraan harga.

- 11 Adapun keluaran dari fungsi kedua ini dapat dilihat di gambar 3.4. Sama seperti keluaran untuk
 12 fungsi pertamanya, keluaran untuk fungsi kedua perkakas ini juga meliputi seluruh jasa yang
 13 disediakan oleh Uber.

21:00 \$ uber price -s '25 first street cambridge ma' -e '114 line street somerville ma'				
     Surge				
uberPOOL \$3-\$6 1.57 mi. 8 min. 				
uberX \$6-\$9 1.57 mi. 8 min. 				
uberXL \$10-\$13 1.57 mi. 8 min. 				
UberBLACK \$15-\$20 1.57 mi. 8 min. 				
uberSUV \$22-\$28 1.57 mi. 8 min. 				
 25 First St, Cambridge, MA 02141, USA				
 END 114 Line St, Somerville, MA 02143, USA				

Gambar 3.4: Contoh penggunaan fitur prediksi harga perjalanan untuk perkakas Uber CLI.⁷

⁷Gambar diambil dari <https://github.com/jaebradley/uber-cli>

1 Permasalahan

2 Seperti telah dijelaskan di awal bab ini, perkakas ini tidak dapat digunakan. Kesimpulan yang
 3 diambil oleh penulis mengenai alasan perkakas ini tidak dapat dijalankan adalah dikarenakan oleh
 4 penggunaan API dan modul-modul yang telah usang (*deprecated*). Kesimpulan ini diambil oleh
 5 penulis karena dua alasan utama. Pertama, pada awalnya, perkakas ini tidak dapat dijalankan
 6 karena API Google *Maps* yang dipakai mengandung baris kode berikut didalamnya.

7 `exports.placesAutoCompleteSessionToken = require('uuid/v4');`

8 Kode ini merupakan kode yang dipakai untuk mengambil *subpath* dari paket *uuid*, tetapi penggu-
 9 naannya sudah berubah untuk versi yang lebih barunya. Akan tetapi, setelah diganti baris tersebut
 10 ke penggunaan versi barunya pun, perkakas ini masih tetap tidak dapat dijalankan—sekarang
 11 perkakas ini justru mengembalikan sebuah error. Singkatnya, error tersebut menunjukkan bahwa
 12 perkakas mencoba untuk mengakses API Uber dengan menggunakan kredensial OAuth 2.0 yang
 13 hanya berlaku untuk versi sebelumnya dari API tersebut. Permasalahan ini merupakan permasalah-
 14 an yang juga ditemukan oleh beberapa pengguna lain, seperti tertera di halaman *Github Issues*
 15 dari repositori ini.⁸ Oleh karena hal ini tidak lagi merupakan masalah kode perangkat lunak, maka
 16 perkakas ini dianggap tidak dapat dipakai.

17 3.1.4 Google *Maps Direction CLI*⁹

18 Google *Maps Direction CLI* merupakan sebuah perkakas *command line* yang memiliki kegunaan
 19 yang mirip dengan KIRI, hanya saja perkakas ini tidak secara spesifik mengharuskan penggunaan
 20 angkot, atau transportasi umum lainnya. Singkatnya, perkakas ini memiliki fungsi seperti sebuah
 21 GPS. Untuk menggunakannya, pengguna harus memasukkan perintah dengan bentuk sebagai
 22 berikut.

23 `direction <lokasi awal> <lokasi akhir>`

24 Setelah pengguna memasukkan perintah tersebut dengan benar, perkakas ini akan mengirim
 25 permintaan ke API Google *Maps*, di mana jika prosesnya berhasil, keluarannya akan berupa langkah-
 26 langkah yang harus ditempuh untuk sampai ke lokasi akhir, beserta di jarak berapa langkah tersebut
 27 harus diambil, relatif terhadap langkah sebelumnya. Adapun penggunaan dari perkakas ini dapat
 28 dilihat di gambar 3.5.

29 Permasalahan

30 Seperti tertulis di awal bab ini, perkakas ini juga tidak bisa digunakan. Alasan perkakas ini tidak
 31 dapat digunakan lagi-lagi merupakan masalah teknikal, yaitu diperbaruiinya API Google *Maps*.
 32 Lebih spesifiknya, semenjak 2018, Google tidak lagi memperbolehkan penggunaan API Google *Maps*
 33 tanpa kunci API, yang sayangnya tidak hanya mendasari perkakas ini, tetapi juga kunci API ini
 34 tidak bisa didapatkan tanpa membayarkan biaya tertentu. Oleh karena itu, perkakas ini dianggap
 35 tidak bisa lagi dijalankan.

⁸<https://github.com/jaebradley/uber-cli/issues/87>

⁹<https://github.com/yujinlim/google-maps-direction-cli>

¹⁰Gambar diambil dari <https://github.com/yujinlim/google-maps-direction-cli>

```

Route
Bukit Damansara, Kuala Lumpur, Federal Territory of Kuala Lumpur, Malaysia → Petronas Twin Tower, Kuala Lumpur City Centre, 50088 Kuala Lumpur, Federal Territory of Kuala Lumpur, Malaysia
Duration
18 mins
Routes
Head northeast on Jalan Medan Setia 2 (81 m)
Turn right toward Jalan Medan Setia (28 m)
Merge onto Jalan Medan Setia (45 m)
Turn left onto Jalan Setia Murni 1 (0.3 km)
Turn left onto Jalan Setia Murni 3 (53 m)
Turn left onto Jalan BeringinGo through 1 roundabout (1.0 km)
Take the ramp on the right to Jalan Duta/Kuala Lumpur (28 m)
Merge onto Damansara Link/Lebuhraya SPRINT/Sistem Penyiaran Trafik Kuala Lumpur Barat/E23 (0.4 km)
Continue onto Jalan Semantan (0.9 km)
Take the exit on the right toward K. Lumpur/Jln. Parlimen/PWTC (0.6 km)
Merge onto Jalan Tuanku Abdul Halim (0.7 km)
Take the exit toward PWTC/Jln. Parlimen/Jln. Tun Razak/Jln. Kuching/Dataran Merdeka (0.5 km)
Keep right at the fork, follow signs for Jalan Parlimen/Dataran Merdeka/Merdeka Square/Taman Botani Perdana/Perdana Botanical Garden (0.5 km)
Turn left onto Jalan Parlimen (1.1 km)
Turn left onto Bulatan Data Onn (91 m)
Take the ramp to Jalan Kuching/Route 1 (0.3 km)
Slight left onto Jalan Kuching/Route 1 (0.1 km)
Continue straight to stay on Jalan Kuching/Route 1 (0.7 km)
Take the Jln. Sultan Ismail exit on the right toward Menara KL/KLCC/Ampang/E12 (0.2 km)
Keep right to continue toward Jalan Sultan Ismail (0.3 km)
Continue onto Jalan Sultan Ismail (1.0 km)
Slight left onto the ramp to Ampang (0.4 km)
Continue onto Lebuhraya Bertingkat Ampang - Kuala Lumpur/E12 (0.8 km)
Take exit 1202A-Jln. Tun Razak toward KLCC (0.5 km)
Merge onto Lebuhraya Bertingkat Ampang - Kuala Lumpur/E12Toll road (0.3 km)
Take the exit toward KLCCPartial toll road (0.6 km)

```

Gambar 3.5: Contoh penggunaan perkakas Google Maps Direction CLI.¹⁰

¹ 3.2 Analisis API KIRI

- ² API KIRI memiliki tiga buah layanan, yaitu *search place*, *routing*, dan *smart direction*. Di antara
- ³ ketiga layanan ini, perlu diingat bahwa *smart direction* merupakan sebuah layanan yang bekerja
- ⁴ dengan langsung membuka halaman web KIRI sendiri, sehingga layanan ini tidak akan digunakan
- ⁵ dalam pembuatan perkakas *command line* untuk skripsi ini.

⁶ 3.2.1 *Search Place*

- ⁷ Layanan pertama dari dua layanan yang tersisa merupakan layanan *search place*. Layanan ini
- ⁸ merupakan layanan API KIRI yang berfungsi untuk mencari sebuah lokasi, beserta nilai *latitude*
- ⁹ dan *longitude*-nya, berdasarkan kata kunci yang diberikan oleh pengguna. Untuk memanfaatkan
- ¹⁰ layanan ini, sebuah permintaan GET harus dikirimkan ke alamat API KIRI. Adapun permintaan
- ¹¹ tersebut akan memiliki parameter-parameter sebagai berikut.

- ¹² • **version**

Parameter ini merupakan tanda bagi API untuk menggunakan protokol versi 2. Karena kemungkinan nilai untuk parameter ini hanya satu, maka parameter ini pasti bernilai 2.

- ¹⁵ • **mode**

Parameter ini merupakan mode dari servis/jasa API yang akan digunakan oleh pengguna. Untuk penggunaan layanan *search place*, parameter ini diisi dengan **searchplace**.

- ¹⁸ • **region**

Parameter ini mengatur daerah mana tempat lokasi yang ingin dicari berada. Isi dari parameter ini akan diberikan oleh pengguna pada saat pemakaian perkakas.

- ²¹ • **querystring**

Parameter ini akan berisi kata kunci yang diberikan oleh pengguna untuk mencari lokasi yang ingin ditemukan.

- ²⁴ • **apikey**

1 Parameter ini akan berisi sebuah nilai kunci API yang sudah dibuat sebelumnya, seperti
2 dijelaskan di subbab [2.2.2](#), parameter ini akan bernilai 68C...97C.¹¹

3 **3.2.2 Routing**

4 Layanan kedua dari API ini merupakan fungsi utama dari KIRI sendiri, yaitu pencarian rute dengan
5 menggunakan angkot. Layanan ini akan menerima dua buah lokasi yang ingin dijadikan lokasi
6 awal dan lokasi akhir, dan menunjukkan langkah-langkah yang harus ditempuh untuk menempuh
7 perjalanan tersebut, dengan memanfaatkan jasa angkot yang ada. Sama seperti layanan *search*
8 *place*, layanan ini juga digunakan dengan mengirimkan permintaan GET ke alamat API KIRI.

9 • **version**

10 Sama seperti pada layanan *search place*, parameter ini hanya dapat diisi dengan nilai 2.

11 • **mode**

12 Parameter ini merupakan mode dari servis/jasa API yang akan digunakan oleh pengguna.
13 Untuk penggunaan layanan *routing*, parameter ini diisi dengan **findroute**.

14 • **locale**

15 Isi dari parameter ini akan ditentukan pada saat pemakaian perkakas.

16 • **start**

17 Parameter ini merupakan nilai *latitude* dan *longitude* dari titik awal perjalanan pengguna,
18 yang akan diberikan sebagai masukan pada saat pemakaian perkakas.

19 • **finish**

20 Parameter ini merupakan nilai *latitude* dan *longitude* dari titik akhir/tujuan perjalanan
21 pengguna, yang akan diberikan sebagai masukan pada saat pemakaian perkakas.

22 • **presentation**

23 Parameter ini hanya digunakan untuk fitur *backwards compatibility*. Jika parameter ini ingin
24 digunakan, maka isi dari parameter ini hanya ada satu kemungkinan, yaitu **desktop**. Jika
25 parameter ini tidak dipakai

26 • **apikey**

27 Sama seperti pada layanan *search place*, parameter ini akan berisi sebuah nilai kunci API
28 yang sudah dibuat sebelumnya, yaitu 68C...97C.

29 Layanan ini dapat digunakan sebagai lanjutan dari layanan *search place*, terutama karena fitur
30 pencarian rute hanya menerima nilai *latitude* dan *longitude* dari lokasi-lokasi yang akan digunakan
31 sebagai masukan, yang tidak hanya akan menyusahkan pengguna dalam memakai perkakas *command*
32 *line* ini, tetapi juga kedua nilai tersebut juga merupakan salah satu dari keluaran layanan pencarian
33 lokasi. Jadi, dengan menggunakan kedua layanan tersebut secara berlangsung, maka pengguna
34 hanya perlu mencari lokasi tersebut dengan kata-kata kunci, dan pada akhirnya pengguna akan
35 dapat menerima langkah-langkah yang harus ditempuh dalam rute sebagai keluaran akhir dari
36 perkakas.

¹¹Bagian tengah dari kunci API diganti dengan “...” untuk alasan keamanan.

¹ 3.3 Analisis Perkakas yang Akan Dibuat

² Di bagian ini akan dibahas analisis hal-hal yang berhubungan dengan perkakas yang akan dibangun
³ sebagai proyek skripsi ini.

⁴ 3.3.1 Analisis Fitur Perkakas

⁵ Pada skripsi ini, akan dibangun sebuah aplikasi berupa perkakas *command line* yang merupakan
⁶ ekstensi dari sebuah aplikasi berbasis web lain, yaitu KIRI. Perkakas ini akan menjadi ekstensi KIRI
⁷ dengan cara memungkinkan penggunaanya untuk mengakses fungsi-fungsi API KIRI dari *command*
⁸ *line* milik perangkat mereka masing-masing. Fungsi utama dari perkakas ini tentunya sama dengan
⁹ KIRI sendiri, yaitu membantu navigasi dari satu lokasi ke lokasi lain dengan menggunakan angkot.

¹⁰ Walaupun begitu, aplikasi ini akan memiliki dua fungsi utama yang berhubungan satu sama
¹¹ lain, yaitu pencarian lokasi, dan menunjukkan rute dengan angkot.

- ¹² • Pencarian lokasi

¹³ Pencarian lokasi merupakan fungsi utama pertama dari perkakas ini. Untuk fungsi ini,
¹⁴ masukan langsung dari pengguna yang akan diterima oleh perkakas adalah kata kunci dari
¹⁵ sebuah lokasi yang ingin dicari. Kemudian, perkakas akan memprosesnya melalui API KIRI,
¹⁶ lalu mengembalikan nama lokasi tersebut, serta nilai *latitude* dan *longitude*-nya, sebagai
¹⁷ keluaran akhir dari proses tersebut.

- ¹⁸ • Pencarian rute

¹⁹ Pencarian rute dengan angkot merupakan fungsi utama kedua, sekaligus fungsi umum dari
²⁰ perkakas ini. Dalam fungsi ini, perkakas akan meminta masukan langsung pengguna berupa
²¹ nilai *latitude* dan *longitude* dari lokasi awal serta lokasi akhir dari perjalanan pengguna.
²² Setelah masukan ini berhasil diterima dan diproses, perkakas akan mengeluarkan keluaran
²³ akhir berupa langkah-langkah yang harus diambil oleh pengguna untuk pergi dari lokasi awal
²⁴ ke lokasi akhir, dengan memanfaatkan jasa angkot yang ada.

²⁵ Berikut merupakan opsi-opsi yang akan disediakan dalam perkakas yang akan dibangun.

- ²⁶ • **-h**

²⁷ Perlu diingat juga bahwa aplikasi ini merupakan aplikasi *command line* murni, yang berarti
²⁸ bahwa seluruh operasi dari aplikasi ini akan dilakukan tanpa bantuan gambar grafis apapun.
²⁹ Untuk membantu pengguna dalam mengetahui bagaimana cara menggunakan perkakas
³⁰ ini, tentunya perintah untuk menunjukkan apa perintah-perintah dan opsi-opsi yang tersedia
³¹ merupakan sebuah keharusan. Hal ini merupakan fungsi satu-satunya dari perintah **--help**
³² ini.

- ³³ • **-m <mode>**

³⁴ Opsi ini merupakan opsi berparameter yang menentukan fitur mana yang ingin digunakan
³⁵ oleh pengguna. Adapun isi dari parameter **mode** yang dapat digunakan adalah sebagai berikut:

- ³⁶ – **search**

³⁷ Parameter ini akan menandakan bahwa pengguna ingin menggunakan fitur *search place*
³⁸ dari perkakas. Untuk mode ini, perkakas akan menerima input dari pengguna dalam
³⁹ bentuk parameter-parameter dari opsi-opsi tambahan. Adapun opsi-opsi tersebut adalah

1 sebagai berikut.

2 * **-r <region>**

3 Opsi ini merupakan opsi yang akan menerima parameter berupa kode area dari
4 lokasi yang ingin dicari.

5 * **-q <kata kunci>**

6 Opsi ini merupakan opsi yang akan menerima sebuah *string* sebagai parameternya.
7 *String* ini akan digunakan oleh perkakas sebagai kata kunci untuk pencarian lokasi
8 yang ingin ditemukan oleh pengguna.

9 – **route**

10 Parameter ini akan menandakan bahwa pengguna ingin menggunakan fitur *find route* dari
11 perkakas. Sama seperti mode **--search**, perkakas akan menerima input dari pengguna
12 dari parameter-parameter milik opsi-opsi tambahan. Adapun opsi-opsi tersebut adalah
13 sebagai berikut.

14 * **-s <lokasi awal>**

15 Opsi ini merupakan opsi yang akan menerima parameter berupa lokasi awal perjalanan
16 pengguna nantinya. Perlu diingat bahwa opsi ini hanya menerima masukan
17 lokasi berupa nilai *latitude* dan *longitude* dari lokasi tersebut.

18 * **-f <lokasi akhir>**

19 Opsi ini merupakan opsi yang akan menerima parameter berupa lokasi akhir perjalanan
20 pengguna nantinya. Sama seperti opsi sebelumnya, parameter ini juga hanya
21 menerima masukan lokasi berupa nilai *latitude* dan *longitude*.

22 * **-l <kode bahasa>**

23 Opsi ini akan menerima parameter yang mengatur bahasa yang akan digunakan oleh
24 perkakas di keluarannya nanti.

25 – **routedirect**

26 Parameter ini merupakan tambahan fitur spesifik untuk perkakas ini, yang merupakan
27 gabungan dari fitur pencarian lokasi dan pencarian rute. Fitur ini akan menggabungkan
28 kedua fitur tersebut dengan langkah-langkah berikut.

29 1. Perkakas akan menerima lokasi awal dan akhir berupa kata kunci dari pengguna
30 yang dimasukkan kedalam parameter dari opsi masing-masing.

31 2. Hasil pencarian kedua lokasi akan dikonfirmasi terlebih dahulu, apakah lokasi yang
32 ditemukan benar merupakan lokasi yang ingin digunakan oleh pengguna atau bukan.

33 3. Jika hasil tersebut benar, maka koordinat *latitude* dan *longitude* dari kedua lokasi
34 tersebut akan disimpan.

35 4. Koordinat *latitude* dan *longitude* dari kedua lokasi tersebut kemudian akan digunakan
36 sebagai masukan untuk langkah kedua dari fitur ini, yaitu pencarian rute.

37 5. Hasil dari pencarian rute akan ditampilkan sebagai keluaran akhir dari fitur ini.

38 Adapun parameter yang diperlukan untuk fitur ini merupakan gabungan dari opsi-opsi
39 kedua fitur sebelumnya, yaitu:

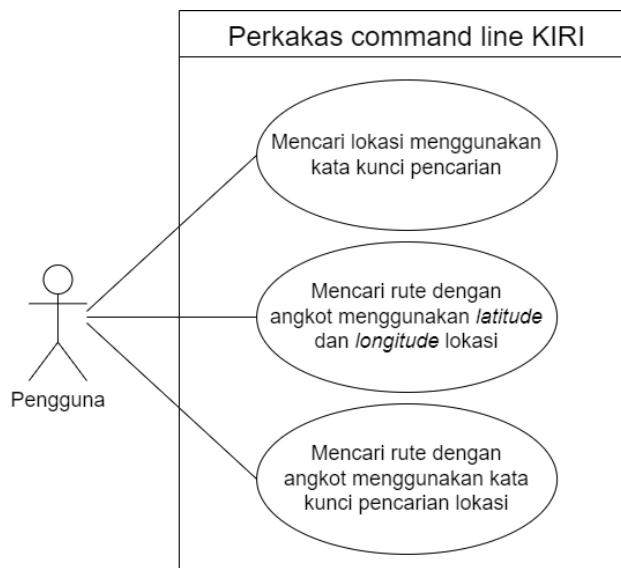
40 * **-I <region tempat lokasi awal berada>**

41 Opsi ini akan menerima parameter berupa kode area dari lokasi awal yang akan
42 digunakan.

- 1 * **-E <region tempat lokasi akhir berada>**
 2 Opsi ini akan menerima parameter berupa kode area dari lokasi akhir yang akan
 3 digunakan.
- 4 * **-S <kata kunci untuk lokasi awal>**
 5 Opsi ini akan menerima parameter berupa kata kunci untuk pencarian lokasi awal
 6 yang akan digunakan.
- 7 * **-F <kata kunci untuk lokasi akhir>**
 8 Opsi ini akan menerima parameter berupa kata kunci untuk pencarian lokasi akhir
 9 yang akan digunakan.
- 10 * **-l <kode bahasa>**
 11 Opsi ini akan menerima parameter yang mengatur bahasa yang akan digunakan oleh
 12 perkakas di keluarannya nanti. Opsi ini merupakan opsi yang sama dengan opsi
 13 yang digunakan dalam mode **route**.

14 **3.3.2 Analisis *Use Case***

- 15 Perkakas *command line* ini akan memiliki tiga fitur, yaitu pencarian lokasi dengan kata kunci, pencarian rute dengan koordinat *latitude* dan *longitude* lokasi, dan pencarian rute dengan kata kunci pencarian lokasi. Oleh karena itu, diagram *use case* dari aplikasi ini akan mengandung dua *use case*, yang dapat dilihat di gambar 3.6. Adapun penjelasan dari tiap-tiap *use case* tersebut dapat dilihat di tabel 3.1, tabel 3.2, dan tabel 3.3.



Gambar 3.6: Diagram *use case* dari perkakas yang akan dibangun.

Nama	Mencari lokasi menggunakan kata kunci pencarian
Deskripsi	Fitur untuk mencari lokasi di area tertentu berdasarkan kata kunci
Aktor	Pengguna aplikasi
Pre-kondisi	-
Pos-kondisi	Nama dari lokasi serta koordinat <i>latitude</i> dan <i>longitude</i> -nya ditampilkan kepada pengguna.
Skenario utama	<ol style="list-style-type: none"> 1. Perkakas membaca masukan berupa kata kunci pencarian lokasi dari argumen dalam perintah <i>command line</i>. 2. Perkakas memproses masukan tersebut dan mengirimkannya ke API KIRI untuk diproses lebih lanjut. 3. Perkakas menerima keluaran berupa nama serta koordinat <i>latitude</i> dan <i>longitude</i> dari lokasi yang ditemukan. 4. Perkakas menampilkan kembali keluaran tersebut kepada pengguna.
Skenario eksepsi	<ul style="list-style-type: none"> • Pengguna memasukkan masukan yang tidak valid sebagai kata kunci pencarian. • Lokasi tidak berhasil ditemukan.

Tabel 3.1: *Scenario case* untuk fitur pencarian lokasi dengan kata kunci.

Nama	Mencari rute dengan angkot menggunakan <i>latitude</i> dan <i>longitude</i> lokasi
Deskripsi	Fitur untuk mencari cara pergi ke satu lokasi ke lokasi lainnya dengan menggunakan angkot dengan nilai <i>latitude</i> dan <i>longitude</i> lokasi sebagai masukan.
Aktor	Pengguna aplikasi
Pre-kondisi	-
Pos-kondisi	Langkah-langkah yang harus ditempuh untuk perjalanan tersebut akan ditampilkan kepada pengguna.
Skenario utama	<ol style="list-style-type: none"> 1. Perkakas membaca masukan berupa nilai <i>latitude</i> dan <i>longitude</i> lokasi awal dan akhir dari argumen dalam perintah <i>command line</i>. 2. Perkakas memproses masukan tersebut dan mengirimkannya ke API KIRI untuk diproses lebih lanjut. 3. Perkakas menerima keluaran akhir dari API KIRI berupa langkah-langkah dalam rute yang perlu ditempuh. 4. Perkakas menampilkan kembali keluaran tersebut kepada pengguna.
Skenario eksepsi	<ul style="list-style-type: none"> • Pengguna memasukkan masukan yang tidak valid sebagai kata kunci pencarian. • Rute tidak berhasil ditemukan/tidak tersedia.

Tabel 3.2: *Scenario case* untuk fitur pencarian rute dengan angkot, dengan nilai *latitude* dan *longitude* kedua lokasi sebagai masukan.

Nama	Mencari rute dengan angkot menggunakan kata kunci pencarian lokasi
Deskripsi	Fitur untuk mencari cara pergi ke satu lokasi ke lokasi lainnya dengan menggunakan angkot dengan nilai <i>latitude</i> dan <i>longitude</i> lokasi sebagai masukan.
Aktor	Pengguna aplikasi
Pre-kondisi	-
Pos-kondisi	Langkah-langkah yang harus ditempuh untuk perjalanan tersebut akan ditampilkan kepada pengguna.
Skenario utama	<ol style="list-style-type: none"> 1. Perkakas membaca masukan berupa kata kunci pencarian lokasi awal dan akhir dari argumen dalam perintah <i>command line</i>. 2. Perkakas memproses kata kunci pencarian lokasi awal dan mengirimkannya ke API KIRI untuk diproses lebih lanjut. 3. Perkakas menerima keluaran berupa koordinat <i>latitude</i> dan <i>longitude</i> lokasi awal dari API KIRI. 4. Perkakas memproses kata kunci pencarian lokasi akhir dan mengirimkannya ke API KIRI untuk diproses lebih lanjut. 5. Perkakas menerima keluaran berupa koordinat <i>latitude</i> dan <i>longitude</i> lokasi akhir dari API KIRI. 6. Perkakas mengirimkan kedua nilai <i>latitude</i> dan <i>longitude</i> lokasi ke API KIRI sebagai masukan dari proses pencarian rute. 7. Perkakas menerima keluaran akhir dari API KIRI berupa langkah-langkah dalam rute yang perlu ditempuh. 8. Perkakas menampilkan kembali keluaran tersebut kepada pengguna.
Skenario eksepsi	<ul style="list-style-type: none"> • Pengguna memasukkan masukan yang tidak valid di dalam salah satu parameter. • Lokasi yang dicari dalam langkah 2 atau langkah 4 tidak ditemukan. • Rute tidak berhasil ditemukan/tidak tersedia.

Tabel 3.3: *Scenario case* untuk fitur pencarian rute dengan angkot, dengan kata kunci pencarian lokasi awal dan akhir sebagai masukan.

1

BAB 4

2

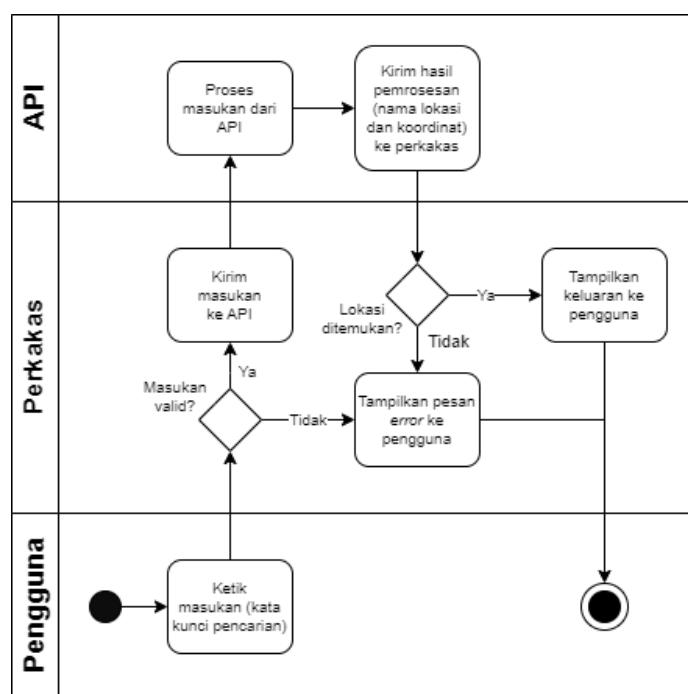
PERANCANGAN

- 3 Pada bab ini akan dipaparkan berbagai macam rancangan dari perkakas *command line* yang akan
4 dibuat, seperti diagram-diagram terkait, masukan serta keluaran perangkat lunak,

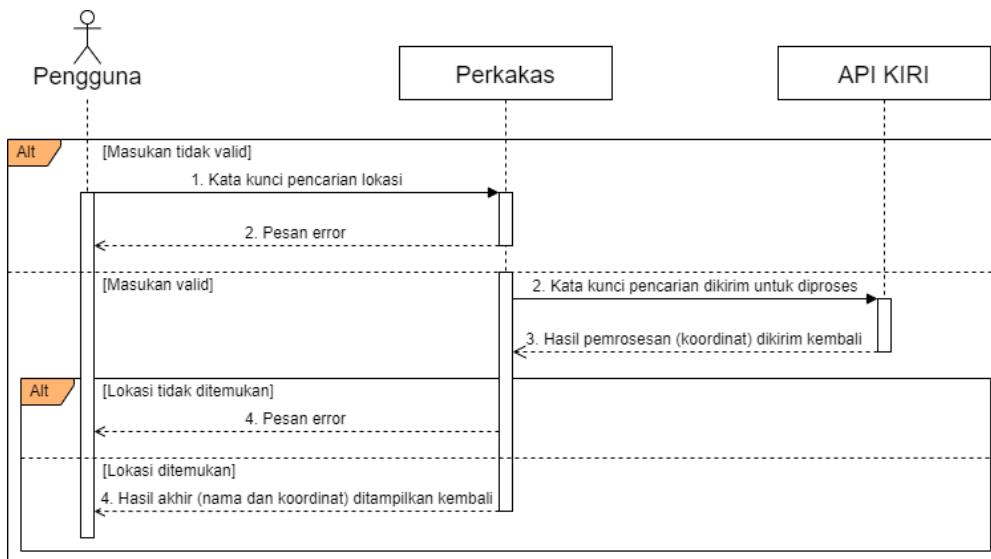
5 **4.1 Rancangan Diagram**

- 6 Bagian ini akan membahas mengenai alur dari *activity diagram* serta *sequence diagram* dari perkakas
7 yang akan dibuat. Perlu diingat bahwa perkakas yang akan dibuat memiliki tiga buah fitur, yaitu:
8 • mencari lokasi menggunakan kata kunci pencarian (**search**),
9 • mencari rute dengan angkot menggunakan *latitude* dan *longitude* lokasi (**route**), dan
10 • mencari rute dengan angkot menggunakan kata kunci pencarian lokasi (**routedirect**).
11 Oleh karena itu, tiap-tiap fitur akan memiliki satu dari setiap tipe diagram. Adapun penjelasan
12 dari bagaimana fitur-fitur ini akan diimplementasikan adalah sebagai berikut.

13 **4.1.1 Mencari lokasi menggunakan kata kunci pencarian**



Gambar 4.1: *Activity diagram* dari fitur pencarian lokasi menggunakan kata kunci pencarian.



Gambar 4.2: *Sequence diagram* dari fitur pencarian lokasi menggunakan kata kunci pencarian.

1 Detail dari alur diagram untuk fitur ini adalah sebagai berikut.

2 1. Perkakas akan meminta masukan dari pengguna berupa kata kunci dari lokasi yang ingin dicari.

3 2. Masukan akan dicek oleh perkakas validitasnya. Dalam kasus ini, masukan hanya akan dianggap tidak valid apabila pengguna tidak memasukkan apapun sebagai masukan perkakas (masukan kosong).

4 3. Jika kata kunci masukan:

- 5 • tidak valid, maka perkakas akan mengeluarkan pesan *error* dan keluar.
- 6 • dianggap valid, kata kunci tersebut akan dikirim ke API KIRI untuk diproses lebih lanjut.

7 4. Setelah selesai diproses, keluaran dari API akan dikembalikan ke perkakas.

8 5. Perkakas akan mengecek keluaran yang diterimanya. Apabila lokasi:

- 9 • ditemukan, maka nama dan koordinat *latitude* dan *longitude* lokasi akan ditampilkan ke pengguna sebagai keluaran akhir.
- 10 • tidak ditemukan, maka perkakas akan mengeluarkan pesan *error* dan keluar.

11 Perlu diingat bahwa perkakas tidak peduli apakah lokasi yang ditemukan benar (sesuai dengan yang diinginkan pengguna) atau tidak.

12 4.1.2 Mencari rute dengan angkot menggunakan *latitude* dan *longitude* lokasi

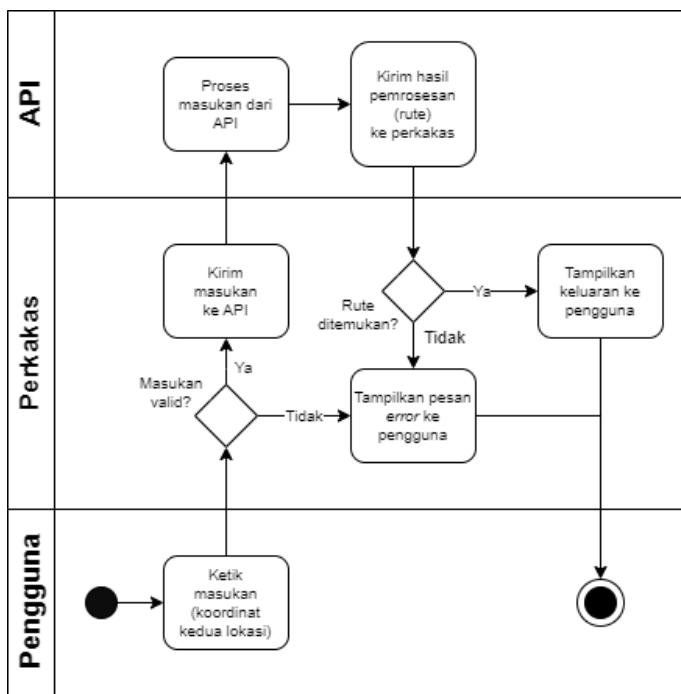
13 Detail dari alur diagram untuk fitur ini adalah sebagai berikut.

14 1. Perkakas akan meminta dua buah masukan dari pengguna, berupa koordinat *latitude* dan *longitude* dari lokasi awal (mulai) dan lokasi akhir (tujuan).

15 2. Masukan akan dicek oleh perkakas validitasnya. Dalam kasus ini, masukan akan dianggap tidak valid apabila ada masukan yang bukan berupa koordinat *latitude* dan *longitude*.

16 3. Jika:

- 17 • satu atau lebih masukan tidak valid, maka perkakas akan mengeluarkan pesan *error* dan keluar.

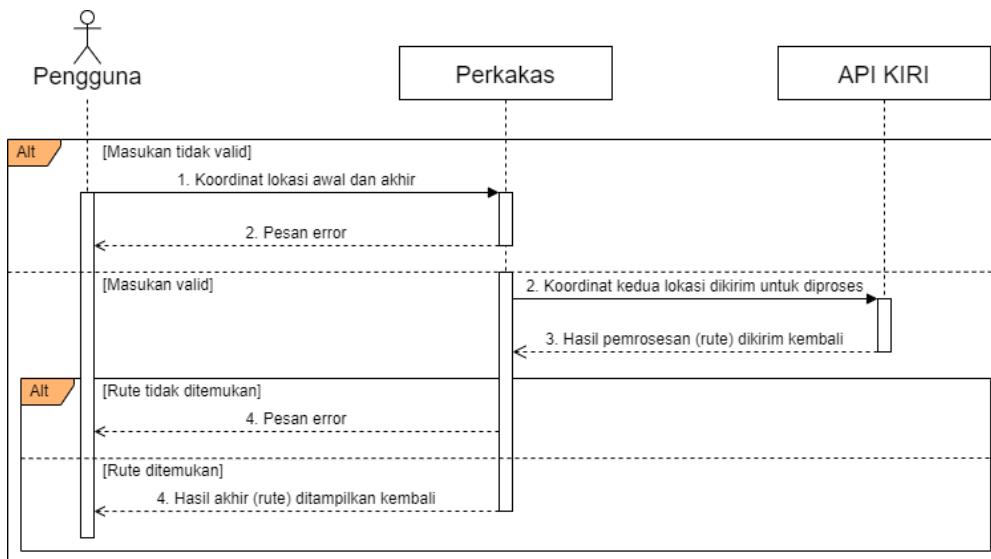


Gambar 4.3: *Activity diagram* dari fitur pencarian rute angkot menggunakan *latitude* dan *longitude* lokasi.

- 1 • kedua masukan dianggap valid, kedua koordinat tersebut akan dikirim ke API KIRI untuk diproses lebih lanjut.
- 2
- 3 4. Setelah selesai diproses, keluaran dari API akan dikembalikan ke perkakas.
- 4 5. Perkakas akan mengecek keluaran yang diterimanya. Apabila rute:
 - 5 • berhasil ditemukan (ada setidaknya satu langkah dalam rute), maka rute tersebut akan ditampilkan ke pengguna sebagai keluaran akhir.
 - 6 • tidak berhasil ditemukan, maka perkakas akan mengeluarkan pesan *error* dan keluar.
- 7

8 4.1.3 Mencari rute dengan angkot menggunakan kata kunci pencarian lokasi

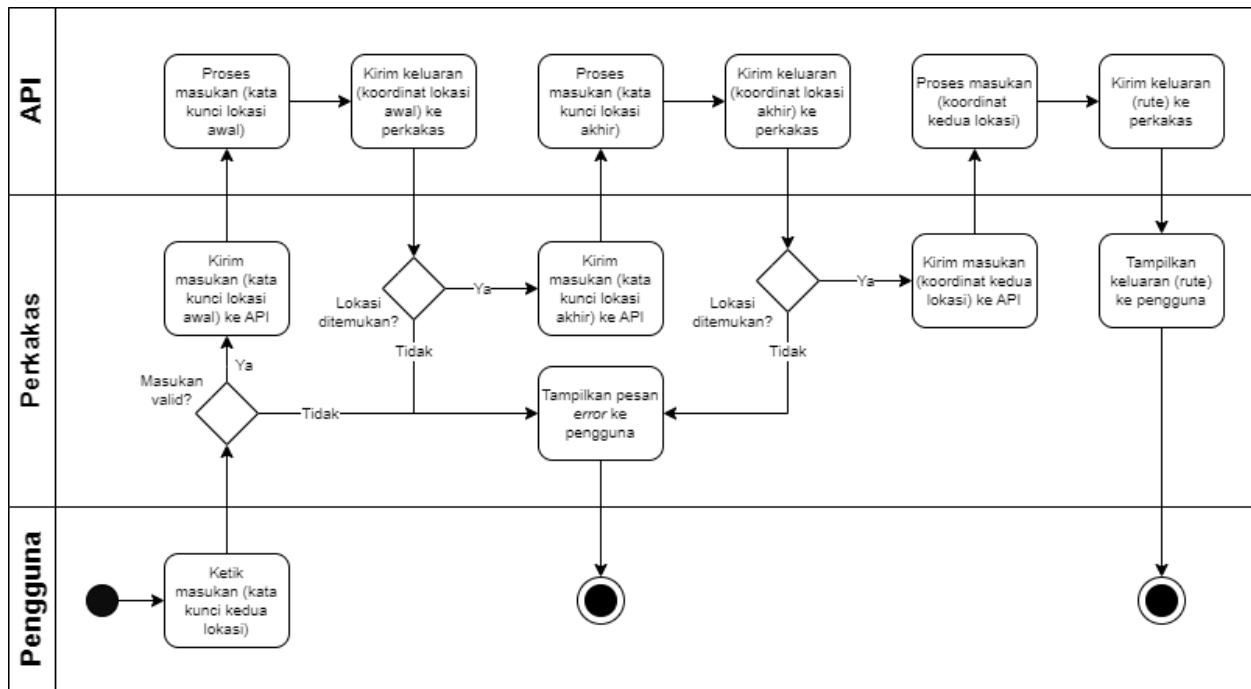
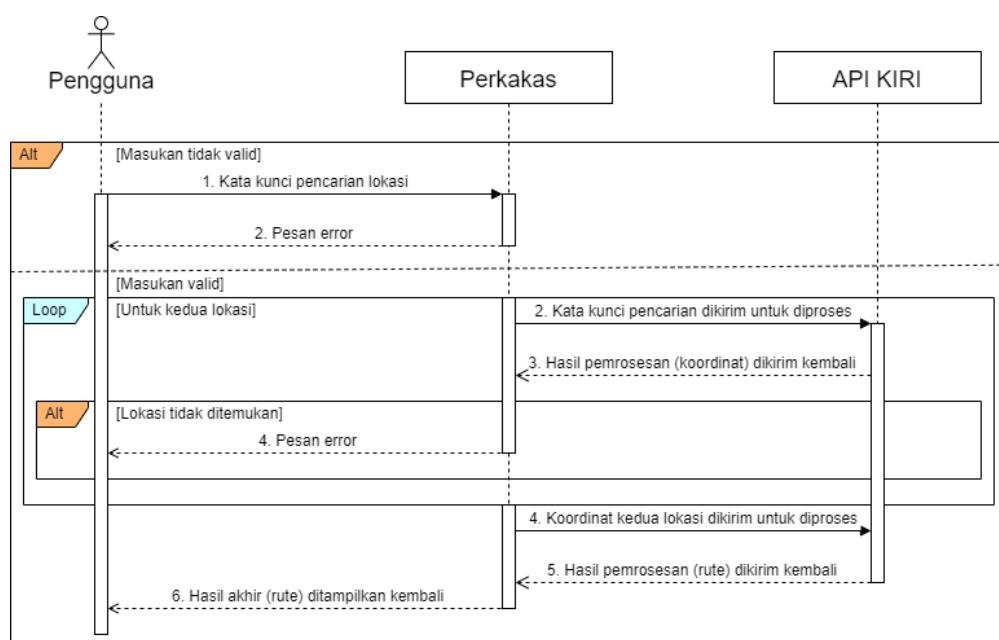
- 9 Detail dari alur diagram untuk fitur ini adalah sebagai berikut.
- 10 1. Perkakas akan meminta masukan dari pengguna berupa kata kunci dari lokasi awal dan lokasi akhir yang ingin dicari.
- 11 2. Masukan untuk lokasi awal akan dicek oleh perkakas validitasnya. Dalam kasus ini, masukan hanya akan dianggap tidak valid apabila pengguna tidak memasukkan apapun sebagai masukan perkakas (masukan kosong).
- 12 3. Jika kata kunci masukan:
 - 13 • tidak valid, maka perkakas akan mengeluarkan pesan *error* dan keluar.
 - 14 • dianggap valid, kata kunci tersebut akan dikirim ke API KIRI untuk diproses lebih lanjut.
- 15 4. Setelah selesai diproses, keluaran dari API akan dikembalikan ke perkakas.
- 16 5. Perkakas akan mengecek keluaran yang diterimanya. Apabila lokasi:
 - 17 • tidak ditemukan, maka perkakas akan mengeluarkan pesan *error* dan keluar.
 - 18 • ditemukan, maka nama dan koordinat *latitude* dan *longitude* lokasi awal akan disimpan



Gambar 4.4: *Sequence diagram* dari fitur pencarian rute angkot menggunakan *latitude* dan *longitude* lokasi.

sebagai variabel untuk dipakai di langkah selanjutnya.

6. Langkah 2 sampai 5 akan diulang kembali untuk lokasi akhir.
7. Jika kata kunci kedua lokasi berhasil diproses, perkakas akan mengirim kembali koordinat *latitude* dan *longitude* dari kedua lokasi tersebut ke API sebagai masukan untuk proses kedua, yaitu mencari rute angkot dengan koordinat *latitude* dan *longitude* lokasi.
8. Setelah selesai diproses, keluaran berupa rute dari API akan dikembalikan ke perkakas.
9. Perkakas akan mengecek keluaran yang diterimanya. Apabila rute:
 - berhasil ditemukan (ada setidaknya satu langkah dalam rute), maka rute tersebut akan ditampilkan ke pengguna sebagai keluaran akhir.
 - tidak berhasil ditemukan, maka perkakas akan mengeluarkan pesan *error* dan keluar.

Gambar 4.5: *Activity diagram* dari fitur pencarian rute angkot menggunakan kata kunci pencarian.Gambar 4.6: *Sequence diagram* dari fitur pencarian rute angkot menggunakan kata kunci pencarian.

1

BAB 5

2

PENGUJIAN DAN EKSPERIMEN

3 Bab ini akan membahas pengujian perkakas yang telah dibangun, serta skenario-skenario pengujian
4 yang akan digunakan untuk pengujian perkakas.

5 **5.1 Lingkungan Implementasi**

6 Bagian ini akan memaparkan spesifikasi dari perangkat yang digunakan untuk pengujian perkakas
7 yang telah dibuat.

8 **5.1.1 Lingkungan Perangkat Keras**

9 Berikut merupakan spesifikasi perangkat keras yang digunakan dalam pembangunan perkakas ini:
10 • *Processor*: Intel® Core™ i5-10300H @ 2.50 GHz
11 • RAM: 8 GB
12 • *Hard disk*: SSD 512 GB (NVMe™ M.2)
13 • Perangkat keras pendukung: Keyboard

14 **5.1.2 Lingkungan Perangkat Lunak**

15 Berikut merupakan spesifikasi perangkat lunak yang digunakan dalam pembangunan perkakas ini:
16 • OS: Windows 10 Home Single Language (64-bit)
17 • Editor: Visual Studio Code
18 • Bahasa pemrograman: C
19 • *Compiler*: MinGW (versi 12.1.0)
20 • *Library*:
21 – libcurl (versi xx)
22 – cmake (versi 3.24.1)

23 **5.2 Implementasi**

24 Di subbab ini akan dijelaskan berbagai hal terkait dengan implementasi, seperti implementasi kode
25 perkakas.

1 5.2.1 Instalasi

2 Untuk dapat menjalankan perkakas ini, pengguna harus menginstal sebuah *compiler* bahasa C,
3 serta dua buah *library* yang telah didaftarkan di subbab 5.1.2, yaitu **libcurl** dan **cmake**.

4 Untuk pemakaian perkakasnya sendiri, berikut merupakan langkah-langkah yang perlu diambil
5 untuk dapat menggunakan perkakas *command line* ini.

- 6 1. Buat *folder* “build” di dalam *folder* perkakas.
- 7 2. Buka *terminal/command prompt* di dalam *folder* “build” tersebut.
- 8 3. Ketik dan jalankan perintah berikut:

```
9 cmake -DCMAKE_BUILD_TYPE:STRING=Release -DCMAKE_TOOLCHAIN_FILE="C:/vcpkg/
10 scripts/buildsystems/vcpkg.cmake" -G "<compiler>" ../
```

11 Untuk apa yang harus menggantikan variabel <compiler> dapat dilihat dengan perintah `cmake --help`. Daftar *compiler* yang didukung oleh *cmake* dapat dilihat di bagian
12 "Generators", dan pengguna tinggal menyesuaikan dengan *compiler* yang telah diinstal sebe-
13 lumnya.

- 14 4. Ketik dan jalankan perintah berikut:

```
15 cmake --build .
```

16 5. File *executable* akan ada di dalam *folder* “build”, dan siap dijalankan.

17 5.2.2 Implementasi kode

18 5.2.3 Pengujian

¹

BAB 6

²

KESIMPULAN

DAFTAR REFERENSI

- [1] Loosemore, S. dkk. (2022) *The GNU C Library Reference Manual*, 2.35 edition. Free Software Foundation, Inc., Massachusetts.
- [2] Stenberg, D. (2022) *Everything curl*. GitBook, Rhone-Alpes.
- [3] Martin, K. dan Hoffman, B. (2013) *Mastering CMake*, 6th edition. Kitware, Inc., New York.
- [4] Marsh, N. (2010) *Introduction to the Command Line: The Fat-Free Guide to Unix and Linux Commands*, 2nd edition. CreateSpace, South Carolina.
- [5] Shotts Jr., W. E. (2019) *The Linux Command Line*, 5th internet edition. <https://www.linuxcommand.org/tlcl.php>.
- [6] Mueller, J. P. (2007) *Windows® Administration at the Command Line for Windows Vista™, Windows® 2003, Windows® XP, and Windows® 2000*, 1st edition. Wiley Publishing, Inc., Indiana.
- [7] Matthew, N. dan Stones, R. (2007) *Beginning Linux® Programming*, 4th edition. Wiley Publishing, Inc., Indiana.
- [8] Microsoft Docs (2021) Windows commands. <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands>. versi 01 Mei 2022.

LAMPIRAN A
KODE PROGRAM