

# SKRIPSI

## INTEGRASI OUTLOOK CALENDAR DAN SLACK



Sandy Giovanni Sutiansen

NPM: 2015730041

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2019



**UNDERGRADUATE THESIS**

**OUTLOOK CALENDAR AND SLACK INTEGRATION**



**Sandy Giovanni Sutiansen**

**NPM: 2015730041**

**DEPARTMENT OF INFORMATICS  
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES  
PARAHYANGAN CATHOLIC UNIVERSITY  
2019**



## ABSTRAK

*Outlook Calendar* adalah sebuah aplikasi yang berfungsi selayaknya kalender yang bisa diisi sebuah event oleh pengguna dari aplikasi tersebut. *Slack* sendiri adalah sebuah aplikasi yang berbasis *cloud* yang diperuntukkan sebagai wadah kolaborasi antar tim. Para pengguna bisa mengatur status yang akan dipasang di dalam akunnya. Namun terkadang pengguna lupa untuk mengubah statusnya agar tidak terganggu dengan pengguna lain yang berusaha mengirimkan pesan disaat pengguna itu sedang dalam keadaan genting atau di dalam suatu pertemuan.

Dengan permasalahan itu, maka skripsi ini akan membahas mengenai sebuah perangkat lunak yang mengintegrasikan antara *event-event* yang tercatat di dalam *Outlook Calendar* dengan status dalam akun pengguna dalam *Slack*. Perangkat lunak yang akan dibangun di dalam skripsi ini adalah perangkat lunak yang mengambil data event, dan ketika sedang ada *event* yang berjalan pada saat itu, maka status di dalam akun pengguna di *Slack* akan terganti agar meminimalisir pengguna lain berusaha untuk mengirimkan pesan kepada pengguna yang akan mengganggu dengan status yang sudah terpasang.

**Kata-kata kunci:** *Outlook Calendar*, *Slack*, Mengintegrasikan, *Event*, Status



## **ABSTRACT**

Outlook Calendar is an application that works like a calendar that can be filled by an event by the user of the application. Slack itself is a cloud-based application that is intended as a forum for collaboration between teams. Users can set the status that will be installed in their account. But sometimes the user forgets to change their status so that they don't interfere with other users who try to send messages when the user is in a critical situation or in a meeting.

With that problem, this thesis will discuss about a software that integrates the events recorded in Outlook Calendar with the status in the user's account in Slack. The software that will be built in this thesis is software that retrieves event data, and when an event is running at that time, the status in the user's account in Slack will be changed to minimize other users trying to send messages to users who will interfere with the status that is already installed.

**Keywords:** Outlook Calendar, Slack, Integrates, Event, Status



## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>ix</b>
<b>DAFTAR GAMBAR</b>	<b>xi</b>
<b>DAFTAR TABEL</b>	<b>xiii</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
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
<b>2 LANDASAN TEORI</b>	<b>5</b>
2.1 <i>Microsoft Graph API</i> . . . . .	5
2.1.1 User resource type . . . . .	6
2.1.2 Event resource type . . . . .	13
2.1.3 Lain-lain . . . . .	17
2.2 <i>Slack API</i> . . . . .	17
2.3 <i>Node.js</i> . . . . .	21
2.3.1 HTTP . . . . .	22
2.3.2 Express.js . . . . .	23
2.3.3 Handlebars . . . . .	24
2.3.4 Simple OAuth2 . . . . .	25
2.3.5 jsonwebtoken . . . . .	27
2.3.6 Isomorphic Fetch . . . . .	28
2.3.7 Microsoft Graph Client Library . . . . .	28
2.3.8 Slack Web API . . . . .	28
2.4 Heroku . . . . .	28
2.4.1 Dyno . . . . .	29
2.4.2 Heroku Scheduler . . . . .	30
2.4.3 Heroku Postgres . . . . .	30
<b>3 ANALISIS</b>	<b>33</b>
3.1 Analisis Microsoft Graph API . . . . .	33
3.1.1 Analisis Mendapatkan Authorization Code . . . . .	33
3.1.2 Analisis Mendapatkan Access Token . . . . .	35
3.1.3 Analisis Menggunakan Access Token untuk memanggil Microsoft Graph . . . . .	37
3.1.4 Analisis Menggunakan Refresh Token untuk Mendapatkan Access Token Baru . . . . .	38
3.1.5 Analisis Mendapatkan Data Events . . . . .	40
3.2 Analisis Slack API . . . . .	42
3.3 Analisis Heroku Scheduler . . . . .	43

3.4	Analisis Diagram Alir Sistem . . . . .	43
3.4.1	Bagian perangkat lunak yang berinteraksi dengan pengguna . . . . .	43
3.4.2	Bagian perangkat lunak yang dijalankan secara berkala . . . . .	45
3.5	Analisis Diagram Use Case . . . . .	46
3.5.1	Use Case dengan actor: pengguna . . . . .	46
3.5.2	Use Case dengan actor: Heroku Scheduler . . . . .	48
3.6	Analisis Data Flow Diagram . . . . .	49
<b>4</b>	<b>PERANCANGAN</b>	<b>51</b>
4.1	Perancangan Routing Handler . . . . .	51
4.1.1	<i>Route /</i> . . . . .	51
4.1.2	<i>Route /authorize</i> . . . . .	51
4.1.3	<i>Route /slackAuthorize</i> . . . . .	51
4.1.4	<i>Route /statusChanger</i> . . . . .	51
4.2	Perancangan <i>Helper</i> . . . . .	53
4.2.1	<i>auth.js</i> . . . . .	53
4.3	Perancangan Basis Data . . . . .	55
4.4	Perancangan Antarmuka . . . . .	56
4.4.1	<i>Route /</i> . . . . .	56
4.4.2	<i>Route /authorize</i> . . . . .	56
4.4.3	<i>Route /slackAuthorize</i> . . . . .	56
<b>5</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>59</b>
5.1	Implementasi . . . . .	59
5.1.1	Lingkungan Pengembangan . . . . .	59
5.1.2	Implementasi Basis Data . . . . .	60
5.1.3	Implementasi Antarmuka . . . . .	60
5.2	Pengujian . . . . .	63
5.2.1	Pengujian Fungsional . . . . .	63
5.2.2	Pengujian Eksperimental . . . . .	65
<b>6</b>	<b>KESIMPULAN DAN SARAN</b>	<b>73</b>
6.1	Kesimpulan . . . . .	73
6.2	Saran . . . . .	73
<b>DAFTAR REFERENSI</b>		<b>75</b>
<b>A KODE PROGRAM</b>		<b>77</b>

## DAFTAR GAMBAR

2.1 Tampilan direktori yang dibuat jika menjalankan express generator. . . . .	24
3.1 Diagram alir sistem pada bagian perangkat lunak yang berinteraksi dengan user. . . . .	44
3.2 Diagram alir sistem pada bagian perangkat lunak yang dijalankan secara berkala. . . . .	45
3.3 Diagram Use Case. . . . .	46
3.4 Diagram Use Case. . . . .	48
3.5 Data Flow Diagram. . . . .	49
4.1 ER Diagram untuk tabel Credentials. . . . .	55
4.2 Rancangan antarmuka halaman awal. . . . .	56
4.3 Rancangan antarmuka halaman setelah login Windows Live. . . . .	57
4.4 Rancangan antarmuka halaman setelah login Slack. . . . .	57
5.1 Antarmuka halaman awal. . . . .	60
5.2 Antarmuka untuk login Windows Live. . . . .	60
5.3 Antarmuka untuk memberikan Windows Live izin ke perangkat lunak. . . . .	61
5.4 Antarmuka petunjuk login ke Slack. . . . .	61
5.5 Antarmuka untuk login Slack. . . . .	61
5.6 Antarmuka untuk memberikan Slack izin ke perangkat lunak. . . . .	62
5.7 Antarmuka saat selesai melakukan login dan pemberian izin. . . . .	62
5.8 . . . . .	64
5.9 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Raymond). . . . .	64
5.10 . . . . .	65
5.11 . . . . .	65
5.12 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Sandy). . . . .	66
5.13 . . . . .	66
5.14 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Vincent). . . . .	67
5.15 . . . . .	67
5.16 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Yonathan). . . . .	68
5.17 . . . . .	68
5.18 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Chris). . . . .	69
5.19 . . . . .	69
5.20 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Ferdian). . . . .	69
5.21 . . . . .	70
5.22 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Yehezkiel). . . . .	70
5.23 . . . . .	70
5.24 . . . . .	71
5.25 . . . . .	71
5.26 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Tedi). . . . .	71
5.27 . . . . .	72
5.28 Tampilan <i>Slack</i> setelah <i>event</i> dimulai (Yosua). . . . .	72



## DAFTAR TABEL

2.1	Tabel contoh <i>header</i> pesan <i>HTTP</i> . . . . .	22
3.1	Tabel parameter <i>Authorization Code</i> . . . . .	34
3.2	Tabel contoh <i>request Authorization Code</i> . . . . .	34
3.3	Tabel contoh <i>response Authorization Code</i> . . . . .	34
3.4	Tabel parameter <i>response Authorization Code</i> . . . . .	35
3.5	Tabel contoh <i>request Access Token</i> . . . . .	35
3.6	Tabel parameter <i>request Access Token</i> . . . . .	36
3.7	Tabel contoh <i>response Access Token</i> . . . . .	36
3.8	Tabel parameter <i>response Access Token</i> . . . . .	37
3.9	Tabel contoh <i>request call Microsoft Graph</i> . . . . .	37
3.10	Tabel contoh <i>response call Microsoft Graph</i> . . . . .	38
3.11	Tabel contoh <i>request</i> menggunakan <i>Refresh Token</i> . . . . .	39
3.12	Tabel parameter <i>request Refresh Token</i> . . . . .	39
3.13	Tabel contoh <i>response</i> menggunakan <i>Refresh Token</i> . . . . .	39
3.14	Tabel parameter <i>response Refresh Token</i> . . . . .	40
3.15	Tabel contoh <i>header time zone</i> . . . . .	40
3.16	Tabel parameter <i>header time zone</i> . . . . .	41
3.17	Tabel contoh <i>request event</i> . . . . .	41
3.18	Tabel contoh <i>response event</i> . . . . .	42



1

## BAB 1

2

### PENDAHULUAN

#### 3 1.1 Latar Belakang

4 *Outlook.com*<sup>1</sup> adalah sebuah kumpulan aplikasi berbasis *web* seperti *webmail*, *contacts*, *tasks*, dan  
5 *calendar* dari *Microsoft*. Fitur calendar sendiri pertama dirilis pada 14 Januari 2008 dengan nama  
6 Windows Live Calendar. Fitur calendar yang dimiliki oleh *Outlook.com Calendar* sendiri memiliki  
7 tampilan yang mirip dengan aplikasi kalender *desktop* pada umumnya. Seperti layaknya kalender  
8 digital pada umumnya, aplikasi *Outlook.com Calendar* juga bisa menambahkan, menyimpan, dan  
9 memodifikasi *event-event* yang dimasukkan oleh pengguna dan bisa dibuka dimana saja karena  
10 bersifat *online*.

11 *Slack*<sup>2</sup> adalah alat dan layanan kolaborasi tim berbasis *cloud*. *Slack* merupakan singkatan dari  
12 “*Searchable Log of All Conversation and Knowledge*”. Cara melakukan kolaborasi di aplikasi *Slack*  
13 sendiri adalah dengan komunitas, grup, atau tim bergabung ke dalam URL yang spesifik. *Room*  
14 *chat* yang terdapat di dalam aplikasi *Slack* biasa disebut dengan *Channel*. Ada 2 jenis *channel* di  
15 dalam aplikasi *Slack* yaitu *Public Channel* dan *Private Channel*. Pada *Public Channel*, seluruh  
16 anggota dari tim atau komunitas bisa masuk dan bergabung untuk berkomunikasi di *channel*  
17 tersebut. Tetapi pada *Private Channel*, hanya anggota yang diizinkan, ditambahkan, dan diundang  
18 oleh admin atau pembuat *channel* sajalah yang bisa ikut serta dalam berkomunikasi di dalam  
19 *channel* tersebut. *Slack* juga terintegrasi dengan banyak layanan pihak ketiga seperti contohnya  
20 adalah *Google Drive*, *Github*, *Trello*, *Dropbox*, dan masih banyak lagi layanan pihak ketiga yang  
21 bisa diintegrasikan dengan *Slack* itu sendiri.

22 Pada *Slack* terdapat status pengguna yang bisa diganti oleh pengguna tersebut untuk meng-  
23 gambarkan keadaan pengguna saat ini. Sebagai *default*, status bisa menggambarkan jika pengguna  
24 sedang “*In a meeting*” atau sedang “*Out Sick*”, dan banyak status default yang disediakan oleh *Slack*.  
25 Serta status pun bisa diisi oleh pengguna secara sendiri sesuai dengan apa yang ingin dituliskan  
26 oleh penggunanya. Disinilah yang menjadi latar belakang dirancangnya perangkat lunak ini yaitu  
27 terkadang pengguna lupa untuk mengganti status menjadi “*In a meeting*” saat pengguna memiliki  
28 jadwal untuk melakukan meeting, sehingga status di pengguna masih terlihat tersedia oleh user lain  
29 yang membuat tidak mengetahui sang pengguna sedang dalam keadaan *meeting* yang tidak dapat  
30 diganggu. Di saat seperti ini, kemungkinan untuk *meeting* terganggu oleh adanya *chat* yang masuk  
31 lewat *Slack* pun cukup tinggi.

32 Pada skripsi ini akan dibuat perangkat lunak yang akan membaca jadwal dari pengguna yang

---

<sup>1</sup><https://outlook.live.com/>

<sup>2</sup><https://slack.com/>

1 dicantumkan di aplikasi *Outlook.com Calendar*, lalu akan di integrasikan kepada aplikasi *Slack*  
2 dengan mengubah dan mengganti status sesuai dengan jadwal yang telah didapatkan dari data di  
3 *Outlook.com Calendar* dari pengguna.

4 Perangkat lunak ini akan dibuat menggunakan *Node.js*<sup>3</sup> dan akan memiliki 2 fungsi utama yaitu  
5 yang pertama adalah membaca dan mencatat jadwal dari *Outlook.com Calendar* yang membutuhkan  
6 adanya *Outlook.com Calendar API*. Lalu perangkat lunak ini juga memiliki fungsi kedua yaitu  
7 mengubah status ke aplikasi *Slack* dengan menggunakan *Slack API*. Nantinya kedua fungsi dari  
8 perangkat lunak ini akan dijalankan secara berkala.

## 9 **1.2 Rumusan Masalah**

10 Pada perangkat lunak ini, terdapat rumusan masalah sebagai berikut:

- 11 1. Bagaimana cara mendapatkan data *event* dari *Outlook.com Calendar*?
- 12 2. Bagaimana mengubah status pada aplikasi *Slack* menggunakan *Slack API*?
- 13 3. Bagaimana cara membuat program agar dapat mengubah status pada aplikasi *Slack* di jadwal  
14 yang telah didapat dari aplikasi *Outlook.com Calendar*?

## 15 **1.3 Tujuan**

16 Adapun pada perangkat lunak ini memiliki tujuan sebagai berikut:

- 17 1. Mengetahui cara mendapatkan data *event* dari *Outlook.com Calendar*.
- 18 2. Mengetahui cara mengubah status pada aplikasi *Slack* menggunakan *Slack API*.
- 19 3. Membuat program agar dapat mengubah status pada aplikasi *Slack* di jadwal yang telah  
20 didapat dari aplikasi *Outlook.com Calendar*.

## 21 **1.4 Batasan Masalah**

22 Perancangan perangkat lunak ini dibuat berdasarkan batasan-batasan sebagai berikut:

- 23 1. Perangkat lunak ini dijalankan secara berkala sehingga tidak dapat menjalankan *update* status  
24 secara *real-time*.
- 25 2. Hanya berlaku setiap satu pengguna terhubung dengan satu *workspace* di dalam *Slack*.

## 26 **1.5 Metodologi**

27 Berikut adalah metodologi yang akan digunakan dalam penelitian ini:

- 28 1. Melakukan studi literatur tentang *Outlook.com Calendar*, *Slack*, dan juga *Node.js*.

---

<sup>3</sup><https://nodejs.org>

- 1 2. Menggunakan aplikasi *Slack* di lingkungan tempat penulis melakukan magang.
- 2 3. Melakukan analisis cara melakukan *synchronize* dengan aplikasi *Outlook.com Calendar* secara berkala.
- 4 4. Merancang bagian dari perangkat lunak yang akan mengambil data-data *event* dari *Outlook.com Calendar* dan yang bertugas untuk mengubah status pada *Slack* saat waktu sesuai dengan jadwal yang sudah tercatat dari *Outlook.com Calendar*.
- 5 5. Mengimplementasi bagian pengambilan data dari *Outlook.com Calendar* dan juga bagian mengatur status pada *Slack* sesuai jadwal yang telah diambil kepada perangkat lunak Integrasi *Outlook.com Calendar* dengan *Slack* serta melakukan pengujian terhadap fitur yang telah diimplementasikan.

## 11 1.6 Sistematika Pembahasan

- 12 Setiap bab dalam penelitian ini akan memiliki sistematika pembahasan yang dijelaskan ke dalam  
13 poin-poin sebagai berikut:
- 14 1. Bab 1: Pendahuluan, yaitu menjelaskan gambaran umum dari penelitian ini yang berisi tentang  
15 latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika  
16 pembahasan.
  - 17 2. Bab 2: Dasar Teori, yaitu menjelaskan dan membahas teori yang dibutuhkan untuk melakukan  
18 penelitian ini. Meliputi tentang *Outlook.com Calendar API*, *Slack API*, dan *Node.js*.
  - 19 3. Bab 3: Analisis, yaitu membahas mengenai analisis masalah. Berisi tentang analisis cara  
20 pengambilan data dari *Outlook.com Calendar*, dan proses pengubahan status menggunakan  
21 program pada *Slack*.
  - 22 4. Bab 4: Perancangan, yaitu membahas mengenai perancangan aplikasi untuk melakukan  
23 sinkronisasi antara *Outlook.com Calendar* dengan *Slack*.
  - 24 5. Bab 5: Implementasi dan Pengujian, yaitu membahas mengenai implementasi dari perangkat  
25 lunak yang telah dirancang dan juga pengujian perangkat lunak tersebut.
  - 26 6. Bab 6: Kesimpulan dan Saran, yaitu berisi tentang kesimpulan dari penelitian ini dan juga  
27 saran yang dapat diberikan untuk penelitian selanjutnya.



<sup>1</sup>

## BAB 2

<sup>2</sup>

### LANDASAN TEORI

#### <sup>3</sup> 2.1 *Microsoft Graph API*

<sup>4</sup> Microsoft Graph API adalah webservice yang berguna untuk mendapatkan data-data yang terdapat  
<sup>5</sup> di dalam layanan Microsoft 365 yaitu seperti *Azure Active Directory*, layanan *Office 365* (*SharePoint*,  
<sup>6</sup> *OneDrive*, *Outlook/Exchange*, *Microsoft Teams*, *OneNote*, *Planner*, dan *Excel*), layanan *Enterprise*  
<sup>7</sup> *Mobility and Security* (*Identity Manager*, *Intune*, *Advanced Threat Analytics*, dan *Advanced Threat*  
<sup>8</sup> *Protection*), layanan *Windows 10* (*activities* dan *devices*), dan *Education*. Terdapat 2 versi referensi  
<sup>9</sup> untuk *Microsoft Graph API* yaitu versi 1.0 dan juga versi beta, tetapi yang dituliskan pada subbab  
<sup>10</sup> ini mengacu kepada versi 1.0. [1] Pada versi 1.0, *endpoint* utama yang dipakai adalah mengacu  
<sup>11</sup> kepada *endpoint* <https://graph.microsoft.com/v1.0>.

<sup>12</sup> Untuk menggunakan fungsi dari *Microsoft Graph API*, dibutuhkan untuk mendaftarkan terlebih  
<sup>13</sup> dahulu aplikasi yang akan dirancang dan memakai fungsi dari *webservice* dari *Microsoft Graph API*  
<sup>14</sup> ke *Microsoft App Registration Portal*<sup>1</sup>. Pada saat mendaftarkan aplikasinya, pastikan untuk menyalin  
<sup>15</sup> dan menyimpan *application ID* yang adalah pengenal unik untuk aplikasi yang didaftarkan, dan juga  
<sup>16</sup> menyalin *Redirect URL* yang didaftarkan sebagai *URL* yang akan menerima balikan *authentication*  
<sup>17</sup> dan juga *token* yang akan dikirim oleh *endpoint Azure AD v2.0*, serta menyalin *application secret*  
<sup>18</sup> yang didapat saat mengklik “*Generate New Password*” saat mendaftarkan aplikasi (berlaku jika  
<sup>19</sup> mendaftarkan aplikasi berjenis *web apps*). *Application ID* yang didapat saat mendaftar aplikasi akan  
<sup>20</sup> dipakai untuk mengisi nilai dari parameter *client\_id* yang akan diisi saat akan melakukan *request*  
<sup>21</sup> untuk mendapatkan *authorization\_code*. Setelah mendapatkan *authorization\_code*, maka langkah  
<sup>22</sup> selanjutnya adalah meminta *access\_token* yang membutuhkan *parameter authorization\_code* kepada  
<sup>23</sup> *field code*, dan juga *application\_secret* yang didapat dari pendaftaran aplikasi sebelumnya yang  
<sup>24</sup> akan mengisi *field client\_secret*. Response dari request token akan mengembalikan jangka waktu  
<sup>25</sup> aktif dari token tersebut dan juga *refresh\_token* yang akan berguna untuk meminta *refresh\_token*  
<sup>26</sup> saat token sudah *expired*.

<sup>27</sup> Setelah mendapatkan *access token*, barulah layanan untuk mendapatkan data yang tersimpan  
<sup>28</sup> di *Microsoft* baru bisa diakses dan didapatkan. Ada banyak layanan yang disediakan dari *Microsoft*  
<sup>29</sup> *Graph API*

---

<sup>1</sup><https://apps.dev.microsoft.com/>

### 1 2.1.1 User resource type

2 Kelas *user* ini merepresentasikan *Azure AD user account*. Kelas ini memiliki properti-properti dan  
3 juga method-method:

4

5 **Properti**

- 6 • **aboutMe** Properti ini bertipe String yang merupakan field untuk mendeskripsikan diri  
7 pengguna.
- 8 • **accountEnabled** Properti ini bertipe Boolean yang bernilai **true** jika akun diaktifkan dan  
9 akan bernilai **false** jika tidak. Properti ini berguna saat akan membuat akun. Nilai ini yang  
10 akan dipakai sebagai patokan sebuah akun bisa dibuat atau tidaknya.
- 11 • **ageGroup** Properti ini bertipe String yang merupakan nilai kelompok umur dari pengguna.  
12 Terdapat nilai **null**, **minor**, **notAdult**, dan juga **adult**.
- 13 • **assignedLicenses** Properti ini bertipe koleksi assignedLicense yang merupakan nilai lisensi  
14 yang diberikan kepada pengguna.
- 15 • **assignedPlans** Properti ini bertipe koleksi assignedPlan yang merupakan nilai plan yang  
16 diberikan kepada pengguna.
- 17 • **birthday** Properti ini bertipe DateTimeOffset yang merupakan nilai ulang tahun dari peng-  
18 guna.
- 19 • **businessPhones** Properti ini bertipe String yang merupakan nomor telepon dari pengguna.  
20 Walaupun bersifat String, tetapi field ini hanya akan bisa diisi oleh angka.
- 21 • **city** Properti ini bertipe String yang merupakan kota lokasi pengguna.
- 22 • **companyName** Properti ini bertipe String yang merupakan nama perusahaan dimana  
23 pengguna terkait di dalamnya.
- 24 • **consentProvidedForMinor** Properti ini bertipe String yang merupakan status persetujuan  
25 bagi anak dibawah umur yang mengacu kepada properti ageGroup. Nilai dari properti ini  
26 bisa **null**, **granted**, **denied**, dan juga **notRequired**.
- 27 • **country** Properti ini bertipe String yang merupakan negara lokasi pengguna.
- 28 • **createDateTime** Properti ini bertipe DateTimeOffset yang merupakan tanggal dibuatnya  
29 objek pengguna.
- 30 • **department** Properti ini bertipe String yang merupakan nama departemen pengguna bekerja.
- 31 • **displayName** Properti ini bertipe String yang merupakan nama yang ditampilkan di buku  
32 alamat untuk pengguna. Biasanya disusun dari nama depan, nama tengah, dan juga nama  
33 belakang. Properti ini merupakan properti yang *required* ketika pengguna dibuat dan tidak  
34 bisa dihapus.

- 1   • **employeeId** Properti ini bertipe String yang merupakan pengidentifikasi karyawan yang  
2   diberikan kepada pengguna oleh organisasi.
- 3   • **faxNumber** Properti ini bertipe String yang merupakan nomor fax pengguna.
- 4   • **givenName** Properti ini bertipe String yang merupakan nama depan dari pengguna.
- 5   • **hireDate** Properti ini bertipe DateTimeOffset yang merupakan tanggal pengguna dipeker-  
6   jakan.
- 7   • **id** Properti ini bertipe String yang merupakan tanda pengenal unik untuk pengguna.
- 8   • **imAddresses** Properti ini bertipe koleksi String yang merupakan alamat protokol inisiasi  
9   sesi *voice over IP* (VOIP) pesan untuk pengguna.
- 10   • **interests** Properti ini bertipe koleksi String yang merupakan kumpulan String yang mendesk-  
11   ripsikan ketertarikan dari pengguna.
- 12   • **isResourceAccount** Properti ini bertipe Boolean yang akan bernilai **true** jika akun me-  
13   rupakan *resource account* dan akan bernilai **false** jika bukan. Jika kosong akan dianggap  
14   dengan nilai false.
- 15   • **jobTitle** Properti ini bertipe String yang merupakan jabatan dari pengguna.
- 16   • **legalAgeGroupClassification** Properti ini bertipe String yang merupakan penentu ke-  
17   lompok legalAge dengan dihitung menggunakan properti ageGroup dan juga consentPro-  
18   videdForMinor. Nilai dari properti ini bisa berupa null, minorWithOutParentalConsent,  
19   minorWithParentalConsent, minorNoParentalConsentRequired, notAdult, dan juga adult.  
20   Properti ini bersifat ***Read-Only***.
- 21   • **licenseAssignmentStates** Properti ini bertipe koleksi licenseAssignmentState yang meru-  
22   pakan status penugasan lisensi untuk pengguna. Properti ini bersifat ***Read-Only***.
- 23   • **mail** Properti ini bertipe String yang merupakan alamat SMTP untuk pengguna. Properti  
24   ini bersifat ***Read-Only***.
- 25   • **mailboxSettings** Properti ini bertipe mailboxSettings yang merupakan pengaturan untuk  
26   mailbox utama dari pengguna yang masuk.
- 27   • **mailNickname** Properti ini bertipe String yang merupakan alias email dari pengguna.  
28   Properti ini harus ditentukan saat pengguna dibuat.
- 29   • **mobilePhone** Properti ini bertipe String yang merupakan nomor telepon seluler utama  
30   pengguna.
- 31   • **mySite** Properti ini bertipe String yang merupakan url untuk situs pribadi pengguna.
- 32   • **officeLocation** Properti ini bertipe String yang merupakan lokasi kantor di tempat bisnis  
33   pengguna.

- 1     ● **onPremisesDistinguishedName** Properti ini bertipe String yang merupakan nama atau  
2       DN Direktori Aktif lokal yang dibedakan. Properti ini hanya diisi untuk pelanggan yang  
3       menyinkronkan direktori lokal mereka ke Azure Active Directory melalui Azure AD Connect.  
4       Properti ini bersifat ***Read-Only***.
- 5     ● **onPremisesDomainName** Properti ini bertipe String yang merupakan dnsDomainName  
6       yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk pelanggan yang me-  
7       nyinkronkan direktori lokal mereka ke Azure Active Directory melalui Azure AD Connect.  
8       Properti ini bersifat ***Read-Only***.
- 9     ● **onPremisesExtensionAttributes** Properti ini bertipe OnPremisesExtensionAttributes  
10      yang merupakan extensionAttributes untuk pengguna. Jika properti onPremisesSyncEnabled  
11      bernilai **true**, maka properti ini bersifat ***Read-Only***, tetapi jika properti onPremisesSyncE-  
12      nabled bernilai **false**, maka properti ini dapat diatur saat membuat atau memperbarui.
- 13    ● **onPremisesImmutableId** Properti ini bertipe String yang digunakan untuk mengaitkan  
14      akun pengguna Active Directory lokal ke objek Azure AD pengguna. Properti ini ditentukan  
15      saat pembuatan akun pengguna baru di Graph.
- 16    ● **onPremisesLastSyncDateTime** Properti ini bertipe DateTimeOffset yang memiliki fungsi  
17      untuk menunjukkan kapan terakhir kali objek disinkronkan dengan direktori lokal. Properti  
18      ini bersifat ***Read-Only***.
- 19    ● **onPremisesProvisioningErrors** Properti ini bertipe koleksi onPremisesProvisioningError  
20      yang merupakan kesalahan saat menggunakan produk sinkronisasi Microsoft.
- 21    ● **onPremisesSamAccountName** Properti ini bertipe String yang merupakan samAccount-  
22      Name lokal yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk pelanggan  
23      yang menyinkronkan direktori lokal mereka ke Azure Active Directory melalui Azure AD  
24      Connect. Properti ini bersifat ***Read-Only***. **onPremisesSecurityIdentifier** Properti ini  
25      bertipe String yang merupakan pengidentifikasi keamanan lokal (SID) untuk pengguna yang  
26      disinkronkan dari lokal ke cloud. Properti ini bersifat ***Read-Only***. **onPremisesSyncEna-**  
27      **bled** Properti ini bertipe Boolean yang bernilai **true** jika objek ini disinkronisasi dari direktori  
28      lokal dan bernilai **false** jika objek ini awalnya disinkronisasi dari direktori lokal tetapi tidak  
29      lagi disinkronkan. Properti ini juga bisa bernilai **null** jika objek ini tidak pernah disinkronkan  
30      dari direktori lokal. Properti ini bersifat ***Read-Only***.
- 31    ● **onPremisesUserPrincipalName** Properti ini bertipe String yang merupakan userPrinci-  
32      palName di tempat yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk  
33      pelanggan yang menyinkronkan direktori lokal mereka ke Azure Active Directory melalui  
34      Azure AD Connect. Properti ini bersifat ***Read-Only***.
- 35    ● **otherMails** Properti ini bertipe String yang merupakan daftar dari alamat email tambahan  
36      untuk pengguna.
- 37    ● **passwordPolicies** Properti ini bertipe String yang menentukan kebijakan kata sandi untuk  
38      pengguna.

- 1     ● **passwordProfile** Properti ini bertipe passwordProfile yang menentukan profil kata sandi  
2       untuk pengguna. Profil ini berisi kata sandi dari pengguna. Properti ini diperlukan saat  
3       pengguna dibuat dan kata sandi harus memenuhi persyaratan yang ditentukan oleh properti  
4       passwordPolicies.
- 5     ● **pastProjects** Properti ini bertipe koleksi String yang merupakan daftar proyek yang sudah  
6       lalu dari pengguna.
- 7     ● **postalCode** Properti ini bertipe String yang merupakan kode pos dari pengguna.
- 8     ● **preferredDataLocation** Properti ini bertipe String yang merupakan lokasi data yang dipilih  
9       oleh pengguna.
- 10    ● **preferredLanguage** Properti ini bertipe String yang merupakan bahasa yang dipilih oleh  
11      pengguna.
- 12    ● **preferredName** Properti ini bertipe String yang merupakan nama yang dipilih oleh pengguna.
- 13    ● **provisionedPlans** Properti ini bertipe koleksi provisionedPlan yang merupakan plan yang  
14      disediakan untuk pengguna. Properti ini bersifat **Read-Only** dan tidak bisa bernilai **null**.
- 15    ● **proxyAddresses** Properti ini bertipe koleksi String.
- 16    ● **responsibilities** Properti ini bertipe koleksi String yang merupakan daftar dari tanggung  
17      jawab pengguna.
- 18    ● **schools** Properti ini bertipe koleksi String yang merupakan daftar dari instansi pendidikan  
19      yang pernah dihadiri oleh pengguna.
- 20    ● **showInAddressList** Properti ini bertipe Boolean yang bernilai **true** jika daftar alamat  
21      Outlook global harus berisi pengguna ini, dan bernilai **false** jika tidak. Jika tidak diberikan  
22      nilai, maka akan bernilai **true**.
- 23    ● **skills** Properti ini bertipe koleksi String yang merupakan daftar dari kemampuan pengguna.
- 24    ● **state** Properti ini bertipe String yang merupakan negara atau provinsi yang terdapat di  
25      alamat pengguna.
- 26    ● **streetAddress** Properti ini bertipe String yang merupakan alamat dari tempat bisnis peng-  
27      guna.
- 28    ● **surname** Properti ini bertipe String yang merupakan nama keluarga atau nama belakang  
29      dari pengguna.
- 30    ● **usageLocation** Properti ini bertipe String yang merupakan 2 huruf dari kode negara pengguna  
31      yang digunakan untuk memeriksa ketersediaan layanan di negara pengguna.
- 32    ● **userPrincipalName** Properti ini bertipe String yang merupakan nama utama dari pengguna  
33      yang memakai standar internet RFC 822.
- 34    ● **userType** Properti ini bertipe String yang digunakan untuk mengklasifikasi tipe pengguna,  
35      seperti contohnya “Member” dan “Guest”.

## 1 Method

2 Untuk mengakses setiap *method* yang akan dijabarkan, perlu diketahui bahwa untuk mengirimkan  
3 *request*, diperlukan *header* yang berisikan *Authorization* yang diisi dengan nilai dari *Access\_token*  
4 yang didapat dari proses sebelumnya. *Endpoint* utama untuk mengirimkan request adalah ke  
5 <https://graph.microsoft.com/v1.0> lalu dilanjutkan dengan *endpoint* fungsinya masing-masing yang  
6 akan diberikan dan dijelaskan.

7 • **List users** Method ini berfungsi untuk mendapatkan daftar dari objek pengguna. Me-  
8 thod ini direquest dengan cara mengirim *request get* kepada *endpoint /users* dengan *he-  
9 aders* berisikan otorisasi yang diisi dengan nilai dari *access token* dan juga *Content-Type*  
10 yang bernilai *application/json*. Request untuk method ini juga bisa diisi dengan parame-  
11 ter *\$select* untuk mengembalikan *field* yang hanya diisi di dalam parameter *\$select* ter-  
12 sebut, dan dalam parameter itu, tiap field dipisah dengan tanda koma (,) contohnya *ht-  
13 tps://graph.microsoft.com/v1.0/users?\$select=displayName,givenName,postalCode*. *Respons*  
14 dari method ini akan mengembalikan *json* dari objek pengguna.

15 • **Create user** Method ini berfungsi untuk membuat pengguna. Method ini direquest dengan  
16 cara mengirimkan *request post* ke *endpoint /users* yang memiliki headers otorisasi yang diisi  
17 dengan *access token* dan juga *Content-Type* yang diisi dengan *application/json* dan juga *body*  
18 yang berisi parameter untuk membuat objek pengguna. Parameter yang wajib diisi adalah  
19 *accountEnabled*, *displayName*, *onPremisesImmutableId*, *mailNickname*, *passwordProfile*, dan  
20 juga *userPrincipalName*.

21 • **Get user** Method ini berfungsi untuk membaca properti dan juga hubungan pengguna.  
22 Method ini direquest dengan cara mengirim *get request* dan dikirimkan ke *endpoint /users/{id  
23 / userPrincipalName}* atau bisa juga dengan menggunakan */me* dengan headers yang sama  
24 dengan method-method sebelumnya dan juga bisa menggunakan parameter *\$select* seperti  
25 method sebelumnya.

26 • **Update user** Method ini berfungsi untuk memperbarui pengguna. Method ini direquest  
27 dengan cara mengirimkan *patch request* kepada *endpoint /users/{id / userPrincipalName}*.  
28 Memiliki request headers seperti method lainnya dan juga body diisi dengan *field* yang akan  
29 diubah nilainya.

30 • **Delete user** Method ini berfungsi untuk menghapus pengguna. Method ini diakses dengan  
31 mengirimkan *delete request* ke *endpoint /users/{id / userPrincipalName}*.

32 • **List messages** Method ini berfungsi untuk mendapatkan semua pesan di kotak surat pengguna  
33 yang masuk. Method ini diakses dengan cara mengirimkan *get request* ke *endpoint /users/{id  
34 / userPrincipalName}/messages*.

35 • **Create message** Method ini berfungsi untuk membuat pesan baru untuk dimasukkan ke  
36 dalam koleksi pesan. Method ini dapat dijalankan dengan cara mengirimkan *post request* ke  
37 *endpoint /users/{id/userPrincipalName}/messages*. *Body* dari *request* ini akan berisi *json*  
38 yang merepresentasikan objek *message*.

- 1   • **List mailFolders** Method ini berfungsi untuk mendapatkan folder-folder surat dibawah folder  
2    root dari pengguna yang masuk. Method ini dijalankan dengan cara mengirimkan *get request*  
3    ke endpoint */users/{id | userPrincipalName}/mailFolders*.
- 4   • **Create mailFolder** Method ini berfungsi untuk membuat *mailFolder* ke dalam koleksi  
5    *mailFolders*. Method ini dijalankan dengan cara mengirimkan *post request* ke endpoint  
6    */users/{id | userPrincipalName}/mailFolders*.
- 7   • **sendMail** Method ini berfungsi untuk mengirim pesan. Method ini dijalankan dengan cara  
8    mengirimkan *post request* kepada endpoint */users/{id | userPrincipalName}/sendMail*. Body  
9    dari request ini berisi *message* dan juga sebuah *Boolean* untuk attribut *saveToSentItems*.
- 10   • **List events** Method ini berfungsi untuk mendapatkan daftar objek event di dalam kotak  
11    pesan dari pengguna. Method ini dijalankan dengan cara mengirimkan *get request* kepada  
12    endpoint */users/{id | userPrincipalName}/events*. *Headers* pada method ini akan berisi  
13    otorisasi yang diisi nilai dari *access token* sebagai header wajib. Lalu ada juga header pilihan  
14    yaitu *outlook.timezone* dan juga *outlook.body-content-type*. Pada header *timezone*, jika tidak  
15    diisi, maka nilai awal yang dikembalikan adalah dalam *UTC*. Request ini juga bisa difilter  
16    dengan menggunakan parameter *\$select*.
- 17   • **Create event** Method ini berfungsi untuk membuat event ke dalam koleksi dari event-event.  
18    Method ini dijalankan dengan cara mengirimkan *post request* kepada endpoint */users/{id |*  
19    *userPrincipalName}/events*. *Body* dari *request* ini akan berisi *json* yang merepresentasikan  
20    objek *event*.
- 21   • **List calendars** Method ini berfungsi untuk mendapatkan daftar objek calendar. Method ini  
22    dijalankan dengan cara mengirimkan *get request* ke endpoint */users/{id | userPrincipalNa-*  
23    *me}/calendars*.
- 24   • **Create calendar** Method ini berfungsi untuk membuat objek calendar baru yang akan dikirim  
25    ke dalam koleksi objek *calendars*. Method ini akan dijalankan dengan cara mengirimkan  
26    *post request* ke endpoint */users/{id | userPrincipalName}/calendars* dengan *body json* yang  
27    merepresentasikan objek *calendar*.
- 28   • **List calendarGroups** Method ini berfungsi untuk mendapatkan daftar objek *calendarGroup*.  
29    Method ini dijalankan dengan cara mengirimkan *post request* ke endpoint */users/{id |*  
30    *userPrincipalName}/calendarGroups*.
- 31   • **Create calendarGroup** Method ini berfungsi untuk membuat objek *calendarGroup* baru  
32    kedalam koleksi *calendarGroup*. Method ini dijalankan dengan cara mengirimkan *post request*  
33    ke endpoint */users/{id | userPrincipalName}/calendarGroups* dengan *body* berupa *json* yang  
34    merepresentasikan objek *calendarGroup*.
- 35   • **List calendarView** Method ini berfungsi untuk mendapatkan koleksi objek event. Method ini  
36    dijalankan dengan cara mengirimkan *get request* kepada endpoint */users/{id | userPrincipa-*  
37    *Name}/calendarView?startDateTime={start\_datetime}&endDateTime={end\_datetime}*. Ter-  
38    dapat tambahan parameter yaitu *startTime* dan *endTime* yang akan menjadi acuan mulai  
39    dari kapan dan sampai kapan *calendarView* yang akan diambil.

- 1     • **List contacts** Method ini berfungsi untuk mendapatkan daftar kontak dari folder Contacts  
2 pengguna yang masuk. Method ini dijalankan dengan cara mengirimkan *get request* ke  
3 *endpoint /users/{id / userPrincipalName}/contacts*. Method ini juga bisa menerima tambahan  
4 parameter *\$filter* yang berfungsi untuk memfilter balikan yang didapat.
- 5     • **Create contact** Method ini berfungsi untuk membuat kontak baru untuk dimasukkan ke  
6 dalam koleksi kontak. Method ini dijalankan dengan cara mengirimkan *post request* ke  
7 *endpoint /users/{id / userPrincipalName}/contacts*. dan memiliki *body* berupa *json* yang  
8 merepresentasikan objek *contact*.
- 9     • **List contactFolders** Method ini berfungsi untuk mendapatkan koleksi folder kontak dari  
10 pengguna yang masuk. Method ini dijalankan dengan cara mengirimkan *get request* kepada  
11 *endpoint /users/{id / userPrincipalName}/contactFolders*.
- 12     • **Create contactFolder** Method ini berfungsi untuk membuat folder kontak. Method ini  
13 dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id / userPrinci-*  
14 *palName}/contactFolders* dengan memiliki *body* berupa *json* yang merepresentasikan objek  
15 *contactFolder*.
- 16     • **List directReports** Method ini berfungsi untuk mendapatkan pengguna dan kontak yang  
17 melaporkan pengguna dari properti *directReports*. Method ini dijalankan dengan cara  
18 mengirimkan *get request* kepada *endpoint /users/{id / userPrincipalName}/directReports*.
- 19     • **List manager** Method ini berfungsi untuk mendapatkan manager pengguna dari properti  
20 manager. Method ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint*  
21 */users/{id / userPrincipalName}/manager*. Method ini akan mengembalikan nilai berupa  
22 *json* objek dari pengguna yang menjadi managernya.
- 23     • **List memberOf** Method ini berfungsi untuk mendapatkan kelompok dan peran dari anggota  
24 langsung pengguna lewat properti *memberOf*. Method ini dijalankan dengan cara mengirimkan  
25 *get request* kepada *endpoint /users/{id / userPrincipalName}/memberOf*.
- 26     • **List transitive memberOf** Method ini berfungsi untuk mendapatkan kelompok dan peran  
27 dari pengguna lewat properti *memberOf*, tetapi method ini bersifat transitif dan mencakup  
28 grup-grup dimana pengguna menjadi anggota. Method ini dijalankan dengan cara mengirimkan  
29 *get request* kepada *endpoint /users/{id / userPrincipalName}/transitiveMemberOf*.
- 30     • **List ownedDevices** Method ini berfungsi untuk mendapatkan perangkat yang dimiliki oleh  
31 pengguna dari properti *ownedDevices*. Method ini dijalankan dengan cara mengirimkan *get*  
32 *request* kepada *endpoint /users/{id / userPrincipalName}/ownedDevices*.
- 33     • **List ownedObjects** Method ini berfungsi untuk mendapatkan objek yang dimiliki pengguna  
34 yang didapat dari properti *ownedObjects*. Method ini dijalankan dengan cara mengirimkan *get*  
35 *request* kepada *endpoint /users/{id / userPrincipalName}/ownedObjects*.
- 36     • **List registeredDevices** Method ini berfungsi untuk mendapatkan perangkat yang terdaftar  
37 oleh pengguna dari properti *registeredDevices*. Method ini dijalankan dengan cara mengirimkan  
38 *get request* kepada *endpoint /users/{id / userPrincipalName}/registeredDevices*.

- 1   • **List createdObjects** Method ini berfungsi untuk mendapatkan objek yang dibuat oleh  
2   pengguna dari properti createdObjects. Method ini dijalankan dengan cara mengirimkan *get*  
3   *request* kepada endpoint */users/{id / userPrincipalName}/createdObjects*.
- 4   • **assignLicense** Method ini berfungsi untuk menambah atau membuang “subscriptions” dari  
5   pengguna, serta bisa untuk mengaktifkan dan menonaktifkan paket spesifik terkait dengan  
6   langganan. Method ini dijalankan dengan cara mengirimkan *post request* kepada endpoint  
7   */users/{id / userPrincipalName}/assignLicense* dan *body* dari *request* ini bisa diisi dengan  
8   parameter addLicenses yang bertipe AssignedLicense dan juga parameter removeLicenses yang  
9   diisi dengan guid dari lisensi yang sudah aktif sekarang.
- 10   • **List licenseDetails** Method ini berfungsi untuk mendapatkan koleksi objek licenseDetails.  
11   Method ini dijalankan dengan cara mengirimkan *get request* kepada endpoint */users/{id / userPrincipalName}/licenseDetails*.
- 12   • **checkMemberGroups** Method ini berfungsi untuk memeriksa keanggotaan dalam daftar  
13   grup. Method ini dijalankan dengan cara mengirimkan *post request* kepada endpoint */users/{id / userPrincipalName}/checkMemberGroups* dan *body* yang dibutuhkan *request* ini adalah  
14   groupIds yang merupakan *id* dari grup yang akan dicari.
- 15   • **getMemberGroups** Method ini mengembalikan semua grup dimana pengguna menjadi  
16   anggota didalamnya. Method ini dijalankan dengan cara mengirimkan *post request* kepada  
17   endpoint */users/{id / userPrincipalName}/getMemberGroups* dan memerlukan *body request*  
18   yaitu securityEnabledOnly yang bertipe Boolean yang bernilai *true* jika hanya *security groups*  
19   dari pengguna yang terdaftar sebagai anggota yang dikembalikan, dan bernilai *false* jika  
20   harus mengembalikan semua grup yang memiliki pengguna sebagai anggotanya.
- 21   • **getMemberObjects** Method ini mengembalikan semua grup dan peran dimana pengguna  
22   menjadi anggota didalamnya. Method ini dijalankan dengan cara mengirimkan *post request*  
23   kepada endpoint */users/{id / userPrincipalName}/getMemberObjects* dan memerlukan *body request*  
24   yaitu securityEnabledOnly seperti yang sudah dijelaskan di method sebelumnya.
- 25   • **reminderView** Method ini mengembalikan daftar pengingat di kalender dengan jam mulai  
26   dan berakhirnya secara spesifik. Method ini dijalankan dengan cara mengirimkan *get request*  
27   kepada endpoint */users/{id / userPrincipalName}/reminderView* yang memerlukan parameter  
28   tambahan yaitu *startDateTime* dan juga *endDateTime* yang keduanya bertipe *String*.
- 29   • **delta** Method ini berfungsi untuk mendapatkan perubahan tambahan pengguna. Method ini  
30   dijalankan dengan cara mengirimkan *get request* kepada endpoint */users/delta*.

33   Seluruh endpoint */users/{id / userPrincipalName}* bisa diganti dengan */me*.

### 34   2.1.2 Event resource type

- 35   Kelas *event* ini untuk merepresentasikan objek *event* dalam kalender pengguna atau dalam kalender  
36   *default* dari grup *Office 365*. Dalam kelas ini juga memiliki properti-properti dan juga method-  
37   method:

## 1 Properti

- 2 • **attendees** Properti ini menunjukkan daftar dari hadirin dari suatu event. Properti ini bertipe  
3 koleksi attendee.
- 4 • **body** Properti ini merupakan isi pesan terkait dengan acara. Bisa berbentuk HTML atau  
5 berupa teks. Properti ini bertipe itemBody.
- 6 • **bodyPreview** Properti ini bertipe *String* yang merupakan pratinjau pesan terkait acara.
- 7 • **categories** Properti ini merupakan daftar kategori yang terkait dengan acara. Properti ini  
8 bertipe koleksi *String*.
- 9 • **changeKey** Properti ini bertipe *String* yang merupakan pengidentifikasi versi dari objek  
10 acara. Setiap kali acara diubah, properti ini juga berubah.
- 11 • **createdDateTime** Properti ini menunjukkan tanggal dari acara dibuat. Properti ini bertipe  
12 *DateTimeOffset*.
- 13 • **end** Properti ini bertipe *dateTimeTimeZone* yang berfungsi untuk menunjukkan tanggal,  
14 waktu, dan zona waktu acara akan berakhir.
- 15 • **hasAttachments** Properti ini bertipe *Boolean* yang akan menunjukkan ada atau tidaknya  
16 lampiran. Nilai *true* artinya ada lampiran dan *false* untuk tidak adanya lampiran.
- 17 • **iCalUid** Properti ini merupakan identifikasi unik yang dibagikan oleh semua instansi yang  
18 tergabung dalam acara di berbagai kalender. Properti ini bertipe *String* dan juga bersifat  
19 *Read-Only*.
- 20 • **id** Properti ini bertipe *String* dan juga bersifat *Read-Only*.
- 21 • **importance** Properti ini bertipe *importance* yang menunjukkan pentingnya acara. Nilai dari  
22 properti ini bisa berupa *low*, *normal*, dan juga *high*.
- 23 • **isAllDay** Properti ini untuk menunjukkan apakah acara ini berjalan sehari atau tidak.  
24 Bertipe *Boolean* yang akan bernilai *true* jika acaranya berlangsung sehari, dan bernilai *false*  
25 jika tidak.
- 26 • **isCancelled** Properti ini untuk menunjukkan apakah acara ini dibatalkan atau tidak. Bertipe  
27 *Boolean* yang bernilai *true* jika dibatalkan dan *false* jika tidak dibatalkan.
- 28 • **isOrganizer** Properti ini bertipe *Boolean* yang menunjukkan nilai jika pengirim pesan adalah  
29 penyelenggara dari acara atau bukan. Bernilai *true* jika pengirim pesan adalah penyelenggara  
30 acara dan *false* untuk bukan penyelenggara.
- 31 • **isReminderOn** Properti ini untuk mengetahui status dari pengingat bagi pengguna dari  
32 acara. Bertipe *Boolean* yang akan bernilai *true* jika ada peringatan yang diatur untuk  
33 mengingatkan kepada pengguna, dan bernilai *false* jika tidak ada.
- 34 • **lastModifiedDateTime** Properti ini untuk menunjukkan kapan terakhir data acara dimodi-  
35 fikasi. Properti ini bertipe *DateTimeOffset*.

- 1     • **location** Properti ini menunjukkan lokasi dari acara yang bertipe *location*.
- 2     • **locations** Properti ini berisi kumpulan dari lokasi acara. Properti ini bertipe koleksi *location*.
- 3     • **onlineMeetingUrl** Properti ini berisi *URL* untuk melakukan rapat secara *online* dan bertipe *String*.
- 4
- 5     • **organizer** Properti ini untuk menunjukkan penyelenggara dari acara. Properti ini bertipe *recipient*.
- 6
- 7     • **originalEndTimeZone** Properti ini menunjukkan zona waktu acara berakhir pada awal acara ini dibuat. Properti ini bertipe *String*.
- 8
- 9     • **originalStart** Properti ini merupakan informasi mulainya acara sejak awal acara ini dibuat. Properti ini bertipe *DateTimeOffset*.
- 10
- 11     • **originalStartTimeZone** Properti ini menunjukkan zona waktu acara dimulai sejak acara ini dibentuk. Properti ini bertipe *String*.
- 12
- 13     • **recurrence** Properti ini menunjukkan pola pengulangan untuk acara. Properti ini bertipe *patternedRecurrence*.
- 14
- 15     • **reminderMinutesBeforeStart** Properti ini menunjukkan berapa menit peringatan pengingat sebelum acara dimulai. Properti ini bertipe *Int32*.
- 16
- 17     • **responseRequested** Properti ini menunjukkan status jika pengirim menginginkan adanya tanggapan dari acara. Bernilai *true* jika pengirim menginginkan adanya tanggapan, dan bernilai *false* jika tidak. Properti ini bertipe *Boolean*.
- 18
- 19
- 20     • **responseStatus** Properti ini menunjukkan jenis tanggapan yang dikirim sebagai *response* terhadap pesan acara. Properti ini bertipe *responseStatus*.
- 21
- 22     • **sensitivity** Properti ini memiliki kemungkinan nilai yaitu *normal*, *personal*, *private*, atau *confidential*. Properti ini bertipe *sensitivity*.
- 23
- 24     • **seriesMasterId** Properti ini adalah *ID* untuk *item master seri* berulang jika acara ini merupakan dari seri berulang. Properti ini bertipe *String*.
- 25
- 26     • **showAs** Properti ini bertipe *freeBusyStatus* yang memiliki kemungkinan nilai yaitu *free*, *tentative*, *busy*, *oof*, *workingElsewhere*, dan *unknown*.
- 27
- 28     • **start** Properti ini menunjukkan tanggal, waktu, dan zona waktu acara dimulai. Bertipe *dateTimeTimeZone*.
- 29
- 30     • **subject** Properti ini menyimpan subyek dari acara. Properti ini bertipe *String*.
- 31
- 32     • **type** Properti ini bertipe *eventType* yang memiliki kemungkinan nilai yaitu *singleInstance*, *occurrence*, *exception*, dan *seriesMaster*. Properti ini bersifat *Read-Only*.
- 33
- 34     • **webLink** Properti ini berisi *URL* yang berfungsi untuk membuka acara ini di *Outlook Web App*. Properti ini bertipe *String*.

## 1 Method

2 Untuk mengakses setiap *method* yang akan dijabarkan, perlu diketahui bahwa untuk mengirimkan  
3 *request*, diperlukan *header* yang berisikan *Authorization* yang diisi dengan nilai dari *Access\_token*  
4 yang didapat dari proses sebelumnya. *Endpoint* utama untuk mengirimkan request adalah ke  
5 <https://graph.microsoft.com/v1.0> lalu dilanjutkan dengan *endpoint* fungsinya masing-masing yang  
6 akan diberikan dan dijelaskan.

- 7 • **List events** Method ini sama seperti yang sudah dijelaskan di bagian method dari *user*  
8 *resource type*.
- 9 • **Create events** Method ini sama seperti yang sudah dijelaskan di bagian method dari *user*  
10 *resource type*.
- 11 • **Get events** Method ini untuk mendapatkan properti dan hubungan untuk objek *event* yang  
12 spesifik. *Endpoint* dari method ini adalah */users/{id} / userPrincipalName}/events/{id}*  
13 dengan mengirimkan *get request*. Jika *request* berhasil, akan mengembalikan objek event yang  
14 diminta.
- 15 • **Update** Method ini untuk membaharui properti-properti yang terdapat dalam objek *event*.  
16 *Endpoint* dari method ini adalah */users/{id} / userPrincipalName}/events/{id}* dengan mengi-  
17 rimkan *patch request*. *Request* ini diisi dengan *body* yang berisikan properti dari objek *event*  
18 yang akan diubah.
- 19 • **Delete** Method ini berfungsi untuk menghapus objek *event*. Dikirimkan kepada *endpoint*  
20 */users/{id} / userPrincipalName}/events/{id}* dengan mengirimkan *delete request*.
- 21 • **accept** Method ini untuk menerima *event* spesifik di kalender pengguna. *Endpoint* dari  
22 method ini adalah */users/{id} / userPrincipalName}/events/{id}/accept* dengan mengirimkan  
23 *post request*. *Request body* dari method ini bisa diisi dengan *comment* yang bertipe *String*  
24 yang fungsinya untuk menunjukkan catatan dari *respon*, dan juga *sendResponse* yang bertipe  
25 *Boolean* yang bernilai *true* jika *respon* akan dikirim ke penyelenggara, dan bernilai *false* jika  
26 tidak. Nilai *default* dari *sendResponse* yaitu *true*.
- 27 • **tentativelyAccept** Method ini untuk menerima *event* spesifik di kalender pengguna untuk  
28 sementara. *Endpoint* dari method ini adalah */users/{id} / userPrincipalName}/events/{id}/*  
29 *tentativelyAccept* dengan mengirimkan *post request*. Request body dari method ini sama  
30 seperti method accept.
- 31 • **decline** Method ini untuk menolak undangan dari spesifik acara di kalender pengguna.  
32 *Endpoint* dari method ini adalah */users/{id} / userPrincipalName}/events/{id}/decline* dengan  
33 cara mengirimkan *post request*. Request body dari method ini sama seperti method accept.
- 34 • **delta** Method ini berfungsi untuk mendapatkan serangkaian acara yang ditambahkan, dihapus,  
35 dan diperbarui dalam *calendarView* dari kalender utama pengguna. *Endpoint* dari method  
36 ini adalah */users/{id}/calendarView/delta* yang memerlukan parameter wajib yaitu *startDa-*  
37 *teTime* dan *endDateTime* serta ada parameter *\$deltatoken* dan juga *\$skiptoken*. Method ini  
38 dikirimkan dengan mengirim *get request*.

- 1   • **dismissReminder** Method ini berfungsi untuk memberhentikan peringatan untuk acara  
2   yang sudah dipasang di kalender pengguna. Endpoint dari method ini adalah `/users/{id} /`  
3   `userPrincipalName}/events/{id}/dismissReminder` dengan mengirimkan *post request*.
- 4   • **snoozeReminder** Method ini berfungsi untuk menunda peringatan yang sudah dipasang  
5   di kalender pengguna. Endpoint dari method ini menuju ke `/users/{id} / userPrincipalNa-`  
6   `me}/events/{id}/snoozeReminder` dengan mengirimkan *post request*. Memiliki *request body*  
7   yang berisi *newReminderTime* yang bertipe *DateTimeTimeZone* yang merupakan waktu baru  
8   untuk menjalankan peringatan untuk acara.
- 9   • **List instances** Method ini untuk mendapatkan contoh acara dari waktu yang spesifik.  
10   Memiliki *endpoint* `/users/{id} / userPrincipalName}/events/{id}/instances` yang memiliki  
11   parameter wajib *startDateTime* dan juga *endDateTime* dengan mengirimkan *get request*.
- 12   • **List attachments** Method ini berfungsi untuk mendapatkan kumpulan lampiran dari suatu  
13   acara. Endpoint dari method ini adalah `/users/{id} / userPrincipalName}/events/{id}/`  
14   `attachments` dengan cara mengirimkan *get request*.
- 15   • **Add attachment** Method ini berfungsi untuk menambahkan lampiran ke suatu acara dengan  
16   maksimum ukuran file sebesar 4MB. Endpoint dari method ini adalah `/users/{id} / userPrin-`  
17   `cipalName}/events/{id}/attachments` dengan mengirimkan *post request* yang memiliki *request*  
18   *body* berupa *json* representasi dari objek *attachment*.

19   Seluruh *endpoint* `/users/{id} / userPrincipalName}` bisa diganti dengan `/me`.

### 20   2.1.3 Lain-lain

21   Terdapat masih banyak kelas yang disediakan dan digunakan pada *Microsoft Graph API* dan dapat  
22   dilihat lebih jelas pada laman *website* referensi yang disediakan oleh *Microsoft Graph* yaitu bisa  
23   melalui <https://docs.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0>.

## 24   2.2 Slack API

25   Slack API adalah *webservice* yang akan digunakan untuk menghubungkan data yang sudah di  
26   dapat dari *Outlook.com Calendar* ke aplikasi *Slack*.<sup>[2]</sup> Untuk mengakses *Slack API*, kita diharuskan  
27   untuk mendaftarkan aplikasi yang akan dibuat dan juga mendaftarkannya ke *workspace* yang akan  
28   dipakai untuk menjalankan aplikasi yang akan dibuat. Untuk mendaftarkan aplikasi yang akan  
29   dibuat bisa menuju ke laman <https://api.slack.com/apps>. Aplikasi yang sudah terdaftar akan  
30   diberikan *Client ID* yang unik dan juga *Client Secret* yang akan digunakan pada proses *OAuth*.

31   Proses pertama yang akan dijalani dalam rangkaian proses *OAuth* adalah dengan meminta  
32   *authorization code* yang akan berjalan jika aplikasi yang akan dibuat untuk mengarahkan pengguna  
33   dan mengirimkan *get request* ke *URL* <https://slack.com/oauth/authorize> dengan *get parameter*  
34   yang wajib yaitu *client\_id* yang diberikan pada saat mendaftarkan aplikasi yang akan dibuat dan  
35   juga *get parameter scope*, serta memiliki parameter yang bersifat opsional yaitu *redirect\_uri* yang  
36   berfungsi untuk alamat tujuan dari kembalian yang dikirim oleh API, *state* yaitu yang berupa *String*

1 unik untuk diteruskan kembali setelah selesai, dan juga *team* berupa *Slack team ID* dari *workspace*  
 2 yang berfungsi untuk membatasi. Parameter *scope* disini berguna untuk menentukan dengan tepat  
 3 bagaimana aplikasi perlu mengakses akun pengguna *Slack*. Penulisan format *parameter scope* yaitu  
 4 merujuk ke objek yang akan diberi akses, dan dilanjutkan dengan kelas tindakan pada objek yang  
 5 diberikan izin, contohnya *file:read*. Selain itu ada juga perspektif opsional yang berisi *user*, *bot*, dan  
 6 *admin* yang akan memengaruhi tindakan yang muncul nantinya di dalam aplikasi *Slack*, contohnya  
 7 *chat:write:user* yang berarti akan mengirimkan pesan dari pengguna yang memiliki wewenang.  
 8 Kelas tindakan yang ada disini ada 3 yaitu:

- 9     • **read:** Membaca informasi lengkap dari satu sumber.
- 10    • **write:** Memodifikasi sumber. Bisa melakukan *create*, *edit*, dan *delete* dengan kelas tindakan  
   11    ini.
- 12    • **history:** Mengakses arsip pesan.

13    Untuk mengakses API method yang diinginkan, harus memberikan *OAuth scope* yang tepat.  
 14 Berikut daftar kelompok *OAuth scope* dengan API method yang bisa diaksesnya.

15	• channels:history	35	– chat.delete
16	– channels.history	36	– chat.deleteScheduledMessage
17	– channels.replies	37	– chat.postEphemeral
18	• channels:read	38	– chat.postMessage
19	– channels.info	39	– chat.scheduleMessage
20	– channels.list	40	– chat.update
21	• channels:write	41	• chat:write:user
22	– channels.archive	42	– chat.delete
23	– channels.create	43	– chat.deleteScheduledMessage
24	– channels.invite	44	– chat.postEphemeral
25	– channels.join	45	– chat.postMessage
26	– channels.kick	46	– chat.scheduleMessage
27	– channels.leave	47	– chat.update
28	– channels.mark	48	• dnd:read
29	– channels.rename	49	– dnd.info
30	– channels.setPurpose	50	– dnd.teamInfo
31	– channels.setTopic	51	• dnd:write
32	– channels.unarchive	52	– dnd.endDnd
33	– conversations.join	53	– dnd.endSnooze
34	• chat:write:bot	54	– dnd.setSnooze

1	● dnd:write:user	33	— groups.setTopic
2	— dnd.endDnd	34	— groups.unarchive
3	— dnd.endSnooze	35	● identity.basic
4	— dnd.setSnooze	36	— users.identity
5	● emoji:read	37	● identity.basic:user
6	— emoji.list	38	— users.identity
7	● files:read	39	● im:history
8	— files.info	40	— im.history
9	— files.list	41	— im.replies
10	● files:write:user	42	● im:read
11	— files.comments.delete	43	— im.list
12	— files.delete	44	● im:write
13	— files.revokePublicURL	45	— im.close
14	— files.sharedPublicURL	46	— im.mark
15	— files.upload	47	— im.open
16	● groups:history	48	● links:write
17	— groups.history	49	— chat.unfurl
18	— groups.replies	50	● mpim:history
19	● groups:read	51	— mpim.history
20	— groups.info	52	— mpim.replies
21	— groups.list	53	● mpim:read
22	● groups:write	54	— mpim.list
23	— groups.archive	55	● mpim:write
24	— groups.create	56	— mpim.close
25	— groups.createChild	57	— mpim.mark
26	— groups.invite	58	— mpim.open
27	— groups.kick	59	● pins:read
28	— groups.leave	60	— pins.list
29	— groups.mark	61	● pins:write
30	— groups.open	62	— pins.add
31	— groups.rename	63	— pins.remove
32	— groups.setPurpose		

```

1   ● reactions:read           32   ● tokens.basic
2     – reactions.get          33   – migration.exchange
3     – reactions.list         34   ● usergroups:read
4   ● reactions:write          35   – usergroups.list
5     – reactions.add          36   – usergroups.users.list
6     – reactions.remove       37   ● usergroups:write
7   ● reminders:read          38   – usergroups.create
8     – reminders.info         39   – usergroups.disable
9     – reminders.list         40   – usergroups.enable
10  ● reminders:read:user     41   – usergroups.update
11    – reminders.info         42   – usergroups.users.update
12    – reminders.list         43   ● users.profile:read
13  ● reminders:write         44   – team.profile.get
14    – reminders.add          45   – users.profile.get
15    – reminders.complete     46   ● users.profile:write
16    – reminders.delete       47   – users.deletePhoto
17  ● reminders:write:user     48   – users.profile.set
18    – reminders.add          49   – users.setPhoto
19    – reminders.complete     50   ● users.profile:write:user
20    – reminders.delete       51   – users.profile.set
21  ● search:read             52   ● users:read
22    – search.all             53   – bot.info
23    – search.files           54   – users.getPresence
24    – search.messages        55   – users.info
25  ● stars:read              56   – users.list
26    – stars.list             57   ● users:read.email
27  ● stars:write             58   – users.lookupByEmail
28    – stars.add              59   ● users:write
29    – stars.remove            60   – users.setActive
30  ● team:read               61   – users.setPresence
31    – team.info

```

1        Authorization code yang telah didapat memiliki waktu kadaluarsa selama 10 menit. Setelah  
2 mendapatkan authorization code, langkah selanjutnya yang harus dilakukan adalah menukarkan  
3 authorization code dengan access token dengan cara mengirimkan post request kepada endpoint  
4 <https://slack.com/api/oauth.access> yang memiliki request body yang wajib yaitu client\_id, cli-  
5 ent\_secret, dan code. Client\_id dan juga client\_secret didapat dari awal mendaftarkan aplikasi.  
6 Parameter code didapat dari authorization code yang didapatkan dari langkah sebelumnya.

7        Setelah mendapatkan access token, barulah method API yang disediakan bisa dijalankan. Objek  
8 yang bisa diakses dan daftar dari method-method API yang disediakan Slack sangatlah beragam.  
9 Untuk mendapatkan informasi yang lengkap mengenai objek dan method yang disediakan, bisa  
10 melihat referensi yang terdapat di alamat URL yaitu <https://api.slack.com/methods>. Salah satu  
11 objek yang bisa diakses yaitu:

12

13 **users.profile** Pada bagian ini, objek yang bisa diakses adalah profil dari pengguna. Terdapat  
14 method-method seperti:

- 15     • **users.profile.get** Method ini bisa dijalankan dengan mengirimkan get request ke endpoint  
16     <https://slack.com/api/users.profile.get> dan memerlukan token bertipe pengguna serta scope  
17     yaitu users.profile.read. Fungsi dari method ini adalah untuk mendapatkan informasi tentang  
18     profil pengguna.
- 19     • **users.profile.set** Method ini bisa dijalankan dengan mengirimkan post request ke endpoint  
20     <https://slack.com/api/users.profile.set>. Method ini juga memerlukan token dari pengguna serta  
21     bisa menerima request body yang berisi profil pengguna yang berbentuk json. Method ini  
22     berfungsi untuk mengubah isi dari profil pengguna.

## 23     2.3 Node.js

24     Node.js adalah platform perangkat lunak pada sisi server.<sup>[3]</sup> Node.js ini ditulis dengan menggunakan  
25     bahasa Javascript dengan menggunakan npm sebagai default package manager. Untuk menggunakan  
26     node.js, dibutuhkan untuk mengunduh dan menginstal terlebih dahulu node.js yang akan dipakai  
27     pada situs resmi yang tersedia<sup>2</sup>. Setelah mengunduh dan menginstall node.js, barulah node.js bisa  
28     digunakan.

29     Format file yang digunakan oleh node.js ini menggunakan format .js. Untuk bisa menjalankan  
30     file yang sudah dibuat, node.js memiliki perintah yang harus dijalankan di terminal yaitu perintah  
31     “node (nama file)”. Dengan perintah seperti itu, semua code akan dieksekusi. Dengan menggunakan  
32     node.js, sebuah kode program yang berbahasa JavaScript akan menjadi server-base, yang biasanya  
33     program JavaScript adalah sebuah client-base. Selain itu, node.js juga merupakan sebuah runtime  
34     JavaScript yang berjalan secara asynchronous yang diperuntukkan untuk membangun aplikasi  
35     jaringan yang bersifat skalabilitas.

36     Node.js menyediakan berbagai macam kelas-kelas yang bisa membantu untuk membuat kode  
37     program berjalan sesuai dengan kebutuhan. Setiap kelas dari library Node.js yang akan digunakan  
38     dan akan dipanggil di kode program yang akan dibuat perlu dipanggil dengan menggunakan perintah

---

<sup>2</sup><https://nodejs.org/en/>

- 1 “`require()`” dengan membutuhkan parameter yaitu nama kelas yang akan digunakannya. Contoh  
 2 dari kelas yang ada di library node.js akan dijabarkan pada subbab [2.3.1](#)

### 3 **2.3.1 HTTP**

- 4 Untuk bisa mengakses dan menggunakan kelas HTTP server dan client, maka harus menggunakan  
 5 kode “`require('http')`”. Kelas interface HTTP di dalam node.js ini didesign untuk mendukung banyak  
 6 fitur protokol yang sulit digunakan. Dengan adanya interface HTTP, node.js menjadi bisa mengirim  
 7 data melalui Hyper Text Transfer Protocol(HTTP). Header pada pesan HTTP merepresentasikan  
 8 sebuah objek seperti pada contoh table [2.1](#).

Tabel 2.1: Tabel contoh *header* pesan *HTTP*

<pre>{   'content-length': '123',   'content-type': 'text/plain',   'connection': 'keep-alive',   'host': 'mysite.com',   'accept': '*/*' }</pre>
---

- 9 Untuk bisa melakukan pengiriman request ke suatu endpoint, maka dibutuhkan method dari  
 10 kelas HTTP yaitu method `http.request()`. Method ini menerima parameter berupa url, options yang  
 11 berupa objek, dan juga array dari callback. Pada bagian ini akan dijelaskan secara lebih detail  
 12 mengenai parameter yang dibutuhkan oleh method `http.request()` ini

13 • **url**

14 Bertipe String atau bisa bertipe class dari URL.

15 • **options**

- 16 – **Protocol** bertipe String yang menggambarkan protokol yang digunakan dengan nilai  
 17 default adalah ‘`http`’.
- 18 – **host** bertipe String yang merupakan nama domain atau IP address dari server untuk  
 19 mengeluarkan request dengan nilai default adalah ‘`localhost`’.
- 20 – **hostname** bertipe String yang merupakan alias untuk host.
- 21 – **family** bertipe number yang merupakan nilai dari versi IP address untuk melambangkan  
 22 host atau hostname. Nilai yang bisa dipakai adalah 4 atau 6.
- 23 – **port** bertipe number yang merupakan nilai dari port dari server. Nilai default yang  
 24 dipakai adalah 80.
- 25 – **localAddress** bertipe String yang menggambarkan interface lokal yang berfungsi untuk  
 26 mengikat koneksi jaringan.
- 27 – **socketPath** bertipe String yang merupakan socket domain dari Unix. Nilai ini tidak  
 28 dapat ditentukan jika salah satu dari host maupun port ditentukan. Nilai ini menentukan  
 29 soket TCP.

- 1     – **method** bertipe String yang menentukan nilai dari tipe request HTTP. Nilai default  
2       yang dipakai adalah ‘GET’.
  - 3     – **path** direktori endpoint pada request yang bertipe String. Nilai ini harus disertakan  
4       dengan query String jika dibutuhkan. Nilai default dari ini adalah ‘/’.
  - 5     – **headers** bertipe Object yang merupakan objek yang mengandung request header.
  - 6     – **auth** bertipe String yang merupakan nilai dari otentikasi dasar contohnya adalah  
7       ‘user:password’.
  - 8     – **agent** bertipe http.Agent atau boolean yang berfungsi untuk mengontrol tingkah laku  
9       dari agent. Nilai yang mungkin ada dalam option ini adalah:
    - 10       \* undefined: sebagai nilai default. Menggunakan http.globalAgent untuk host dan  
11           port.
    - 12       \* Agent Object: secara eksplisit menggunakan agent yang diteruskan.
    - 13       \* false: dikarenakan agent baru dengan nilai-nilai default yang akan dipakai.
  - 14     – **createConnection** sebuah fungsi yang membuat sebuah socket/stream untuk digunakan  
15       sebagai request ketika pilihan agent tidak digunakan.
  - 16     – **timeout** bertipe number yang memiliki nilai yang menggambarkan batas waktu dari  
17       socket di dalam satuan milliseconds.
  - 18     – **setHost** yang bertipe boolean yang berguna untuk menentukan akan menambah header  
19       Host secara otomatis atau tidak. Nilai default dari options ini adalah true.
- 20     • **callback** yang berupa function.
  - 21     • **Returns** bertipe http.ClientRequest.

22     Node.js memelihara beberapa koneksi per servernya untuk melakukan HTTP request. Fungsi  
23     ini memungkinkan untuk pengguna mengeluarkan request secara transparan. Untuk parameter  
24     url, jika url dituliskan dengan bertipe String, maka url tersebut akan langsung secara otomatis  
25     menggunakan url.parser() untuk menjadi url. Http.request() mengembalikan instance dari kelas  
26     http.ClientRequest. Parameter callback adalah function yang bersifat sebagai listener untuk response  
27     yang dihasilkan dari http request.

### 28     2.3.2 Express.js

29     <sup>3</sup> Express.js adalah sebuah *web application framework* untuk Node.js. Cara untuk memasang express  
30     ini adalah dengan menggunakan *command*:

31     \$ npm install express

32     Express ini memiliki banyak fungsi yang disediakan untuk mempermudah membuat *web application*  
33     seperti:

- 34     • *Express-generator*

35     Express-generator ini memiliki fungsi untuk membangun rangka dari aplikasi dengan mudah.

---

<sup>3</sup><https://expressjs.com/>

Dengan menggunakan *express-generator*, maka kita akan mendapatkan direktori file dengan isi seperti:

```

.
├── app.js
├── bin
│   └── www
├── package.json
└── public
    ├── images
    ├── javascripts
    └── stylesheets
        └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug

```

7 directories, 9 files

Gambar 2.1: Tampilan direktori yang dibuat jika menjalankan express generator.

Cara menggunakan express generator bisa dengan menjalankan command “`npx express {nama file yang akan dibuat}`” atau bisa dengan command “`express {nama file yang akan dibuat}`”, maka express generator akan langsung secara otomatis membuat folder dan file-file seperti yang ada pada gambar 2.1.

#### • Routing

Fungsi ini berguna untuk memudahkan dari web application untuk bisa menuju kepada halaman yang berbeda dengan memakai path dari alamat *url* aplikasi tersebut. Cara menggunakannya adalah dengan menggunakan kode program seperti:

```
app .METHOD(PATH, HANDLER)
```

Kode program seperti itu memiliki arti:

- **app** yang merupakan inisiasi dari *express*.
- **METHOD** yang merupakan jenis http request dengan menggunakan penggunaan huruf kecil. (get, post, put, atau delete)
- **PATH** adalah jalur di dalam server.
- **HANDLER** adalah fungsi yang akan dieksekusi jika rute yang diakses cocok.

### 2.3.3 Handlebars

<sup>4</sup> *Handlebars* adalah modul dari *Node.js* yang membantu atas fungsi penggunaan *template semantic* pada tampilan yang akan digunakan di dalam pembangunan perangkat lunak. *Template semantic* ini membantu pengiriman data variable dari javascript belakang ke tampilan.

<sup>4</sup> <https://www.npmjs.com/package/handlebars>

#### 1    2.3.4 Simple OAuth2

2    5 Modul ini berguna untuk membantu proses otentikasi pada perangkat lunak ini yang akan  
3    melakukan otentikasi kepada aplikasi pihak ketiga lainnya. Pada *simple oauth* ini dapat ditangani 3  
4    alur untuk melakukan otentikasi yaitu:

- 5    • Alur kode otorisasi

6        Alur ini dipakai pada aplikasi yang dapat menyimpan data secara persisten.

- 7    • *Password Credentials*

8        Alur ini digunakan pada saat alur sebelumnya tidak dapat dilakukan atau sedang dalam tahap  
9        pengembangan.

- 10   • *Client Credentials*

11        *Client* hanya bisa mendapatkan *access token* dengan cara mengirimkan kredensial yang dimiliki  
12        oleh *client*.

13        Untuk bisa menggunakan *library* ini, dibutuhkan persyaratan yaitu menggunakan *Node* minimal  
14   versi 8. Jika menggunakan *Node* versi 4, 5, atau 6, maka dapat menggunakan *library simple-oauth2@1.x*. Cara pemasangan library ini dengan cara menuliskan perintah pada terminal seperti:

16   \$ npm install --save simple-oauth2

17        Saat digunakan, *simple-oauth2* menerima objek yang berisi parameter yaitu:

- 18   • **client** - objek parameter wajib yang memiliki properti:

19        – **id** - Parameter ini merupakan parameter yang dibutuhkan dan diisi dengan menggunakan  
20        *client id* dari aplikasi yang sudah didaftarkan.

21        – **secret** - Parameter ini merupakan parameter yang dibutuhkan dan diisi dengan menggunakan  
22        *client secret* dari aplikasi yang sudah didaftarkan.

23        – **secretParamName** - Merupakan nama parameter yang digunakan untuk mengirimkan  
24        *client secret* yang memiliki nilai *default* yaitu *client\_secret* dari aplikasi yang sudah  
25        didaftarkan.

26        – **idParamName** - Merupakan nama parameter yang digunakan untuk mengirimkan  
27        *client id* yang memiliki nilai *default* yaitu *client\_id* dari aplikasi yang sudah didaftarkan.

- 28   • **auth** - objek parameter wajib yang memiliki properti:

29        – **tokenHost** - *String* yang digunakan untuk mengatur *host* untuk meminta *token* dan  
30        merupakan parameter yang dibutuhkan.

31        – **tokenPath** - *String* yang menunjukkan jalur untuk meminta *access token* yang memiliki  
32        nilai *default* yaitu ke */oauth/token*.

33        – **revokePath** - *String* yang menunjukkan jalur untuk mencabut *access token* yang memiliki  
34        nilai *default* yaitu ke */oauth/revoke*

---

5 <https://www.npmjs.com/package/simple-oauth2>

- **authorizeHost** - *String* yang digunakan untuk mengatur *host* untuk meminta *authorization code* dan memiliki nilai *default* yaitu *auth.tokenHost*.
- **authorizePath** - *String* yang menunjukkan jalur untuk meminta *authorization code* yang memiliki nilai *default* yaitu ke */oauth/authorize*.
- **http** - objek parameter yang bersifat opsional yang memiliki fungsi sebagai pengatur opsi *global* ke *internal http library* yang memiliki properti:
  - Semua opsi diperbolehkan kecuali opsi *baseUrl*. Memiliki nilai *default* yaitu *headers.Accept = application/json*.
- **options** - objek parameter yang bersifat opsional yang memiliki fungsi sebagai pengatur modul yang memiliki properti:
  - **bodyFormat** - format data yang dikirim dalam *request body* saat mengirimkan *request*. Nilai yang *valid* untuk properti ini adalah *form* atau *json* dan memiliki nilai *default* yaitu *form*.
  - **authorizationMethod** - menunjukkan metode yang dipakai untuk mengirimkan *client.id/client secret* saat meminta *token*. Nilai yang *valid* untuk properti ini adalah *header* atau *body*. Jika properti ini bernilai *body*, maka *bodyFormat* akan dipakai untuk memformat kredensialnya. Nilai *default* dari properti ini adalah *header*.

```

18 const credentials = {
19   client: {
20     id: <client-id>,
21     secret: <client-secret>,
22   },
23   auth: {
24     tokenHost: 'https://login.microsoftonline.com',
25     authorizePath: 'common/oauth2/v2.0/authorize',
26     tokenPath: 'common/oauth2/v2.0/token'
27   }
28 };
29 const oauth2 = require('simple-oauth2').create(credentials);

```

### 30 Alur Oauth2 yang Didukung

- Authorization Code Flow Authorization Code Flow menjalankan 2 langkah. Langkah pertama yaitu aplikasi akan meminta permission kepada pengguna untuk mengakses data mereka. Jika pengguna menyetujui, maka OAuth2 server akan mengirimkan authorization code kepada pengguna yang nantinya akan dilanjutkan dengan langkah kedua yaitu pengguna akan mengirimkan POST request yang berisi authorization code bersamaan dengan client secret ke oauth server untuk mendapatkan access token.

- 1   ● Password Credentials Flow Alur ini menggunakan username serta password untuk bisa  
2   mendapatkan access token yang dibutuhkan oleh user. Alur ini biasanya digunakan saat  
3   pengguna dengan aplikasi atau sistem operasi dari komputer sudah sangat terpercaya sehingga  
4   keamanan username dan password bisa terjaga. Cara ini biasa dipakai untuk melakukan tes  
5   aplikasi secara cepat.
- 6   ● Client Credentials Flow Alur ini cocok untuk pengguna yang melakukan request akses ke  
7   sumber yang dibawah kontrol dari pengguna itu sendiri.

## 8   **Helpers**

- 9   ● Access Token object Saat token kadaluarsa, diperlukan refresh token untuk bisa meminta  
10   ulang access token yang baru. Simple OAuth2 menyediakan sebuah kelas AccessToken yang  
11   memiliki beragam fungsi yang salah satunya membantu mempermudah untuk meminta ulang  
12   access token saat token itu sudah kadaluarsa. Inisialisasi dari kelas ini seperti

```
13       // Sample of a JSON access token (you got it through  
14       previous steps)  
15       const tokenObject = {  
16           'access_token': '<access-token>',  
17           'refresh_token': '<refresh-token>',  
18           'expires_in': '7200'  
19       };
```

### 20   **2.3.5 jsonwebtoken**

21   <sup>6</sup> Jsonwebtoken merupakan implementasi dari JSON Web Tokens yang dikembangkan terhadap  
22   draft-ietf-oauth-json-web-token-08. Jsonwebtoken ini memiliki beberapa fungsi yaitu:

- 23   ● **jwt.sign(payload,secretOrPrivateKey, [options, callback])**

24   Fungsi ini bisa berjalan secara asynchronous jika parameter callback diisi dan disediakan.  
25   Callback dipanggil dengan err atau dengan JWT. Tetapi fungsi ini juga bisa berjalan secara  
26   synchronous dan mengembalikan JsonWebToken sebagai sebuah string.

- 27   ● **jwt.verify(token,secretOrPrivateKey, [options, callback])**

28   Fungsi ini bisa berjalan secara asynchronous jika parameter callback diisi dan disediakan.  
29   Callback dipanggil dengan payload yang telah di decode jika signature-nya valid dan ada  
30   parameter seperti kadaluarsa yang bersifat opsional, audiens, atau penerbit akan menjadi valid.  
31   Jika tidak, maka akan menimbulkan error. Fungsi ini juga bisa berjalan secara synchronous  
32   jika tidak ada callback yang diisi.

- 33   ● **jwt.decode(token, [options])**

34   Fungsi ini berjalan secara synchronous dan mengembalikan payload yang sudah di decode  
35   tanpa memverifikasi signature yang ada valid atau tidak.

---

<sup>6</sup><https://www.npmjs.com/package/jsonwebtoken>

### 1 2.3.6 Isomorphic Fetch

2 <sup>7</sup> Isomorphic Fetch adalah sebuah *polyfill* *window.fetch* untuk digunakan di *node* dan *browsify*.  
 3 *Polyfill* ini memungkinkan *window.fetch* untuk *javascript engine* yang tidak mendukungnya secara  
 4 *native*.

### 5 2.3.7 Microsoft Graph Client Library

6 <sup>8</sup> Microsoft Graph Client Library adalah sebuah library yang disediakan oleh Microsoft untuk per-  
 7 angkat lunak yang akan menggunakan API ke Microsoft bisa lebih mudah. Library ini hanya sebagai  
 8 pembungkus dari semua fungsi API yang disediakan oleh Microsoft. Untuk bisa memakainya, maka  
 9 harus menginisialisasi terlebih dahulu sebuah objek inisialisasi dari library ini dengan menggunakan  
 10 access token yang didapat saat melakukan request awal. Setelah melakukan inisialisasi, maka setiap  
 11 API dapat diakses dengan menjalankan kode seperti:

```
12 const result = await client.api('/me/events').select('subject,start,end').orderby()
```

13 Api diatas diisi dengan fungsi yang akan dipanggil lalu select diatas diisi dengan data yang akan  
 14 diambil. Orderby digunakan untuk jika data ingin diurutkan berdasarkan urutan tertentu.

### 15 2.3.8 Slack Web API

16 <sup>9</sup> Sama seperti Microsoft Graph Client Library, Slack Web API ini juga adalah library yang  
 17 membungkus method-method yang disediakan oleh Slack agar untuk mengakses method bisa lebih  
 18 mudah. Cara menggunakannya adalah dengan menginisialisasi terlebih dahulu library yang akan  
 19 dipakai sebagai kelas yang akan dipakai, lalu menginisialisasi kelas ke sebuah variable yang diisikan  
 20 dengan parameter slack access token yang didapat dari request. Setelah itu, jalankan variable  
 21 tersebut dengan memanggil scope yang dipakai seperti contoh pada code:

```
22 const web = new WebClient(slack_access_token);  

23  

24 const result = await web.users.profile.set({  

25   "profile":{  

26     "status_text": "In A Meeting",  

27     "status_emoji": ":no_entry:"  

28   }  

29 });
```

30 Untuk setiap scope yang dipakai akan memerlukan request body yang berbeda-beda. Untuk  
 31 mengetahui request body yang diperlukan, dapat dilihat pada <https://api.slack.com/methods>.

## 32 2.4 Heroku

33 Heroku adalah *cloud platform* yang menampung program agar bisa dibangun, dan dijalankan di  
 34 *cloud*.<sup>[4]</sup> Heroku mendukung beberapa bahasa pemrograman yaitu: Ruby, Node.js, Java, Python,

<sup>7</sup><https://www.npmjs.com/package/isomorphic-fetch>

<sup>8</sup><https://github.com/microsoftgraph/msgraph-sdk-javascript>

<sup>9</sup><https://www.npmjs.com/package/@slack/web-api>

<sup>1</sup> Clojure, Scala, Go, dan PHP.

<sup>2</sup> **2.4.1 Dyno**

<sup>3</sup> Dyno adalah sebuah wadah berbasis Unix yang disediakan oleh Heroku. Dyno disini dikategorikan  
<sup>4</sup> ke 3 jenis dyno yaitu:

- <sup>5</sup> • Web Dyno

<sup>6</sup> Web dyno adalah dyno yang berjalan pada tipe proses web. Web dyno adalah satu-satunya  
<sup>7</sup> dyno yang bisa menerima HTTP dari router heroku.

- <sup>8</sup> • Worker Dyno

<sup>9</sup> Worker dyno adalah dyno yang berjalan selain pada tipe proses web. Worker dyno biasa  
<sup>10</sup> digunakan untuk pekerjaan pada latar belakang, sistem antrian, dan pekerjaan berjangka  
<sup>11</sup> waktu tertentu.

- <sup>12</sup> • One-off Dyno

<sup>13</sup> One-off dyno adalah dyno yang bersifat sementara dan contoh penggunaannya adalah seperti  
<sup>14</sup> saat migrasi basis data. Dyno ini dijalankan menggunakan command shell.

<sup>15</sup> Heroku menyediakan beberapa paket dyno yang mempengaruhi karakteristik dan juga cara kerja  
<sup>16</sup> dyno seperti:

- <sup>17</sup> • Free

<sup>18</sup> Pada paket ini, jumlah dyno yang didapat untuk setiap process type adalah 1 serta adanya  
<sup>19</sup> batasan dyno hours yaitu sebanyak 550 jam untuk akun yang belum terverifikasi dan 1000  
<sup>20</sup> jam untuk akun yang sudah terverifikasi. Dyno akan memasuki kondisi sleep jika web dyno  
<sup>21</sup> sedang tidak aktif selama lebih dari 30 menit. Dyno akan kembali lagi aktif ketika ada arus  
<sup>22</sup> yang masuk ke HTTP tetapi ada keterlambatan untuk memproses arus tersebut.

- <sup>23</sup> • Hobby

<sup>24</sup> Jumlah dyno yang didapat untuk setiap process type adalah 1. Jumlah dyno hours di dalam  
<sup>25</sup> paket ini tidak dibatasi tetapi setiap dyno hours yang dipakai akan dikenakan biaya sebesar  
<sup>26</sup> \$7 per jam per bulannya.

- <sup>27</sup> • Standard

<sup>28</sup> Jumlah dyno yang didapat untuk setiap process type adalah sebanyak 10. Jumlah dyno yang  
<sup>29</sup> bisa dijalankan secara bersamaan adalah sebanyak 100 pada satu aplikasi. Harga dari dyno  
<sup>30</sup> hours berkisar antara \$25 sampai \$500.

- <sup>31</sup> • Performance

<sup>32</sup> Jumlah dyno yang didapat untuk setiap process type adalah sebanyak 10. Jumlah dyno  
<sup>33</sup> yang bisa dijalankan secara bersamaan adalah sebanyak 100 pada satu aplikasi. Harga dari  
<sup>34</sup> dyno hours berkisar antara \$25 sampai \$500. Perbedaan dari paket standard dengan paket  
<sup>35</sup> performance adalah paket ini bersifat dedicated.

<sup>36</sup> Pada heroku terdapat banyak fitur-fitur penyokong untuk menjalankan aplikasi seperti basis  
<sup>37</sup> data, sistem antrian, layanan email, dan lain-lain yang disebut dengan *add-ons*. Daftar dari

<sup>1</sup> *add-ons* yang tersedia untuk Heroku dapat dilihat pada situs web *Elements Marketplace* (<https://elements.heroku.com/addons>) Beberapa contoh *add-ons* yang tersedia di *heroku* antara lain:

### <sup>3</sup> 2.4.2 Heroku Scheduler

<sup>4</sup> Scheduler adalah add-ons gratis yang bertugas untuk mengatur jalannya pekerjaan dalam aplikasi  
<sup>5</sup> secara berkala sesuai dengan interval yang ditentukan oleh pengguna. Tugas heroku scheduler lebih  
<sup>6</sup> mirip dengan cara kerja cron di server tradisional. Melalui scheduler, pekerjaan dimungkinkan  
<sup>7</sup> untuk dijalankan setiap 10 menit sekali, setiap jam, setiap hari, atau pada waktu yang ditentukan.  
<sup>8</sup> Saat dijalankan, job ini akan berjalan sebagai one-off dynos dan akan muncul di dalam logs sebagai  
<sup>9</sup> dyno seperti scheduler.X. Cara memasang add-ons ini melalui command shell adalah dengan cara:

<sup>10</sup> \\$ heroku addons:create scheduler:standard

<sup>11</sup> Scheduler akan menjalankan one-off dynos yang akan dihitung sebagai penggunaan dari pengguna  
<sup>12</sup> pada bulan itu. Dyno-hours yang dihitung akan sama seperti saat menjalankan aplikasi atau dari  
<sup>13</sup> dynos yang berskala. Untuk pemakaian dari dyno-hours dapat pengguna lihat pada bagian tagihan  
<sup>14</sup> pada tab “Manage account”.

### <sup>15</sup> 2.4.3 Heroku Postgres

<sup>16</sup> Heroku Postgres adalah layanan basis data SQL yang dikelola dan disediakan langsung oleh Heroku.  
<sup>17</sup> Basis data Heroku Postgres dapat diakses dengan menggunakan semua bahasa pemrograman dengan  
<sup>18</sup> driver PostgreSQL termasuk seluruh bahasa pemrograman yang didukung di Heroku. Untuk cara  
<sup>19</sup> penggunaannya di dalam bahasa Node.js, maka dibutuhkan langkah-langkah yaitu

- <sup>20</sup> • Melakukan pemasangan modul pg sebagai dependency dengan cara menjalankan command  
<sup>21</sup> line seperti:

<sup>22</sup> \\$ npm install pg

- <sup>23</sup> • Melakukan koneksi ke url dari basis data yang disediakan oleh Heroku saat perangkat lunak  
<sup>24</sup> dijalankan. Contoh dari kode programnya seperti:

```
25 const { Client } = require('pg');

26

27 const client = new Client({
28   connectionString: database_url,
29   ssl: true,
30 });

31

32 client.connect();

33

34 client.query('SELECT table_schema, table_name FROM
35 information_schema.tables;', (err, res) => {
36   if (err) throw err;
37   for (let row of res.rows) {
```

```
1           console.log(JSON.stringify(row));
2       }
3       client.end();
4   } );
```



1

## BAB 3

2

## ANALISIS

3 Pada bab ini akan dijelaskan mengenai analisis dari penggunaan *Microsoft Graph API* untuk  
4 mendapatkan data *event* yang sudah tercatat di dalam *Outlook Calendar web* dan juga penggunaan  
5 dari *Slack API* untuk memampukan program mengubah status yang terdapat pada aplikasi *Slack*,  
6 serta analisis dari penggunaan *cron* yang akan digunakan untuk menjalankan program yang disusun  
7 secara berkala. Untuk bisa menjalankan program ini, maka dibutuhkan input dari pengguna berupa  
8 *login* ke *Windows Live* dan juga memberikan akses kepada program ini untuk mengambil *access*  
9 *token* dan *refresh token* yang akan digunakan untuk mengambil data *event* yang sudah tercatat di  
10 dalam *Outlook Calendar*. Selain ke *Windows Live*, program ini juga memerlukan pengguna untuk  
11 *login* ke akun yang terdapat di suatu *workspace* di platform *Slack* serta memberikan akses untuk  
12 aplikasi ini sebagai aplikasi yang terdaftar dalam *workspace*-nya.

13 **3.1 Analisis Microsoft Graph API**

14 Pada analisis bagian ini, dilakukan analisis mengenai API yang telah disediakan oleh Microsoft  
15 *Graph API* yang akan digunakan untuk mengambil data acara / event yang dibutuhkan oleh  
16 perangkat lunak yang akan dibangun. Akan ada beberapa langkah yang harus dijalankan untuk  
17 berhasil mencapai tujuan dari perangkat lunak ini. Analisis dari setiap langkah akan dijelaskan  
18 pada subbab 3.1.1 sampai subbab 3.1.4.

19 **3.1.1 Analisis Mendapatkan Authorization Code**

20 Untuk mendapatkan authorization code, diperlukan untuk mendaftarkan aplikasi yang akan dibuat  
21 ke *Microsoft App Registration Portal*<sup>1</sup>. Dari mendaftarkan aplikasi yang akan dibuat di portal  
22 registrasi tersebut akan menghasilkan Application ID, Application Secret, dan juga redirect URL  
23 yang akan digunakan. Jika platform yang dipilih adalah web, maka redirect URL harus ditentukan  
24 sendiri. Dalam meminta authorization code, aplikasi yang dibuat harus mengirimkan *get request*  
25 terlebih dahulu ke *endpoint /authorize* yang membutuhkan parameter seperti yang dijelaskan di  
26 tabel 3.1.

---

<sup>1</sup><https://apps.dev.microsoft.com/>

Tabel 3.1: Tabel parameter *Authorization Code*

Parameter		Deskripsi
<i>tenant</i>	wajib	Nilai <i>tenant</i> berfungsi untuk mengontrol siapa yang dapat masuk ke dalam aplikasi. Bisa diisi dengan <i>tenant ID</i> atau nama domain dari akun <i>Microsoft</i> .
<i>client_id</i>	wajib	Nilai yang dipakai adalah nilai dari <i>application ID</i> yang didapatkan saat mendaftarkan aplikasi di <i>Microsoft App Registration Portal</i> .
<i>response_type</i>	wajib	Tipe balikan yang diterima dari <i>request</i> . Bernilai <i>code</i> yang berarti akan mengembalikan <i>code</i> .
<i>redirect_uri</i>	direkomendasikan	<i>Redirect uri</i> dari aplikasi yang didaftarkan dimana hasil dari <i>request</i> yang didapat akan dikembalikan ke url yang sudah didaftarkan.
<i>scope</i>	wajib	Daftar izin dari <i>Microsoft Graph</i> yang dipisahkan oleh cakupan yang diinginkan dan disetujui oleh pengguna.
<i>response_mode</i>	direkomendasikan	Menentukan metode yang harus digunakan untuk mengirimkan token yang dihasilkan kembali ke aplikasi. Dapat bernilai <i>query</i> atau <i>form_post</i> .
<i>state</i>	direkomendasikan	Nilai yang diisi saat mengirimkan <i>request</i> dan akan dikembalikan juga saat menerima <i>response</i> . Tujuan dari nilai ini adalah untuk mencegah pemalsuan permintaan lintas situs. Digunakan untuk menyandikan informasi sebelum <i>request</i> untuk otentifikasi. Biasanya nilai ini berisi nilai unik secara acak.

- 1 Pada parameter scope, dapat diisi dengan nilai *offline\_access* yang akan menjadikan aplikasi  
 2 mendapatkan response berupa refresh token yang berguna untuk mendapatkan access token yang  
 3 baru saat yang lama sudah kadaluarsa. Contoh request yang dikirimkan akan seperti yang terdapat  
 4 pada contoh table 3.2.

Tabel 3.2: Tabel contoh *request Authorization Code*

```
https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize?
client_id=6731de76-14a6-49ae-97bc-6eba6914391e
&response_type=code
&redirect_uri=http%3A%2F%2Flocalhost%2Fmyapp%2F
&response_mode=query
&scope=offline_access%20user.read%20mail.read
&state=12345
```

- 5 Dari *request* seperti contoh table 3.2, akan menghasilkan contoh *response* seperti yang terdapat  
 6 pada table 3.3 dengan keterangan parameter seperti yang terdapat pada table 3.4.

Tabel 3.3: Tabel contoh *response Authorization Code*

GET https://localhost/myapp/? code=M0ab92efe-b6fd-df08-87dc-2c6500a7f84d &state=12345
---

Tabel 3.4: Tabel parameter *response Authorization Code*

Parameter	Deskripsi
<i>code</i>	Nilai ini merupakan <i>authorization_code</i> yang telah <i>direquest</i> oleh aplikasi. <i>Authorization_code</i> ini digunakan untuk meminta <i>access token</i> . <i>Authorization code</i> memiliki waktu kadaluarsa yang singkat yaitu biasanya akan kadaluarsa setelah 10 menit.
<i>state</i>	Jika saat melakukan <i>request</i> , parameter <i>state</i> diisi, maka pada saat mengeluarkan <i>response</i> , akan menge luarkan nilai <i>state</i> yang sama seperti yang sudah diisi saat melakukan <i>request</i> . Aplikasi harus mengidentifikasi apakah nilai <i>state</i> saat melakukan <i>request</i> dengan nilai <i>state</i> di <i>response</i> sama atau tidak.

- <sup>1</sup> Hasil *response* yang ditampilkan oleh table 3.3 muncul karena pada saat *request* di table 3.2 terdapat parameter *response\_mode* yang diisi dengan nilai *query* sehingga *response* yang dikembalikan dalam bentuk *query string* dari *redirect url*.

#### <sup>4</sup> 3.1.2 Analisis Mendapatkan Access Token

- <sup>5</sup> Setelah mendapatkan authorization code, langkah selanjutnya yang harus dijalankan sebelum bisa <sup>6</sup> memanggil method API yang dibutuhkan adalah dengan mendapatkan access token. Yang diperlukan <sup>7</sup> untuk bisa mendapatkan access token, maka aplikasi yang dibuat membutuhkan authorization code <sup>8</sup> yang diterima di langkah sebelumnya dan mengirimkan post request kepada endpoint /token.  
<sup>9</sup> Untuk mengirimkan post request, diperlukan request body yang memiliki elemen-elemen seperti <sup>10</sup> yang terdapat di contoh table 3.5. Adapun penjelasan dari setiap parameter yang terdapat di dalam <sup>11</sup> request body dijelaskan pada table 3.6.

Tabel 3.5: Tabel contoh *request Access Token*

```
POST /common/oauth2/v2.0/token HTTP/1.1
Host: https://login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

client_id=6731de76-14a6-49ae-97bc-6eba6914391e
&scope=user.read%20mail.read
&code=OAAABAAAAiL9Kn2Z27UubvWFPbm0gLWQ
JVzCTE9UkP3pSx1aXxUjq3n8b2JRLk4OxVXr...
&redirect_uri=http%3A%2F%2Flocalhost%2Fmyapp%2F
&grant_type=authorization_code
&client_secret=JqQX2PNo9bpM0uEihUPzyrh
```

Tabel 3.6: Tabel parameter *request Access Token*

Parameter		Deskripsi
<i>tenant</i>	wajib	Nilai <i>tenant</i> berfungsi untuk mengontrol siapa yang dapat masuk ke dalam aplikasi. Bisa diisi dengan <i>tenant ID</i> atau nama domain dari akun <i>Microsoft</i> .
<i>client_id</i>	wajib	Nilai yang dipakai adalah nilai dari <i>application ID</i> yang didapatkan saat mendaftarkan aplikasi di <i>Microsoft App Registration Portal</i> .
<i>grant_type</i>	wajib	Harus diisi dengan nilai <i>authorization_code</i> untuk alur <i>authorization code</i> .
<i>scope</i>	wajib	Daftar izin dari <i>Microsoft Graph</i> yang dipisahkan oleh cakupan yang diinginkan dan disetujui oleh pengguna. Dalam langkah ini, nilai dari <i>scope</i> pada langkah sebelumnya harus sama dengan langkah ini.
<i>code</i>	wajib	Authorization code yang didapat dari langkah sebelumnya.
<i>redirect_uri</i>	wajib	<i>Redirect uri</i> yang sama yang dipakai untuk mendapatkan <i>authorization code</i> .
<i>client_secret</i>	wajib untuk <i>web apps</i>	Application secret yang dibuat saat mendaftarkan aplikasi di portal registrasi untuk aplikasi yang didaftarkan.

- <sup>1</sup> Dari contoh request yang dilakukan pada table 3.5, maka akan dihasilkan contoh token seperti  
<sup>2</sup> pada table 3.7 yang memiliki keterangan dari hasil yang dikembalikan pada table 3.8.

Tabel 3.7: Tabel contoh *response Access Token*

```
{
  "token_type": "Bearer",
  "scope": "user.read%20Fmail.read",
  "expires_in": 3600,
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbG
ciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
  "refresh_token": "AwABAAAAAvPM1KaPlrEqdF
SBzjqfTGAMxZGUTdM0t4B4..."
}
```

Tabel 3.8: Tabel parameter *response Access Token*

Parameter	Deskripsi
<i>token_type</i>	Menunjukkan nilai dari token. Satu-satunya jenis token yang didukung oleh Azure AD adalah Bearer.
<i>scope</i>	Nilai scope yang valid untuk access_token yang diberikan.
<i>expires_in</i>	Lamanya access token akan berlaku(dalam detik).
<i>access_token</i>	Access token yang diminta. Dengan memakai ini, maka aplikasi bisa memanggil Microsoft Graph.
<i>refresh_token</i>	Refresh token ini berguna untuk meminta kembali access token setelah access token itu berakhir. Refresh token memiliki umur yang panjang dan dapat digunakan untuk mempertahankan akses ke source.

- <sup>1</sup> **3.1.3 Analisis Menggunakan Access Token untuk memanggil Microsoft Graph**
- <sup>2</sup> Setelah mendapatkan access token, panggilan ke Microsoft Graph pun bisa dilakukan dengan syarat
- <sup>3</sup> menyertakan access token di authorization header di setiap request yang dikirim. Pada table [3.9](#)
- <sup>4</sup> menunjukkan contoh request untuk mendapatkan profile dari pengguna yang masuk.

Tabel 3.9: Tabel contoh *request call Microsoft Graph*

```
GET https://graph.microsoft.com/v1.0/me
Authorization: Bearer eyJ0eXAiO ...
0X2tnSQLLEANnSPHY0gKcgw
Host: graph.microsoft.com
```

- <sup>5</sup> Jika request yang dikirimkan berhasil, maka akan mendapatkan response yang akan terlihat
- <sup>6</sup> mirip dengan contoh seperti pada table [3.10](#)

Tabel 3.10: Tabel contoh *response call Microsoft Graph*

```

HTTP/1.1 200 OK
Content-Type: application/json;
odata.metadata=minimal;
odata.streaming=true;
IEEE754Compatible=false;
charset=utf-8

request-id: f45d08c0-6901-473a-90f5-7867287de97f
client-request-id: f45d08c0-6901-473a-90f5-7867287de97f
OData-Version: 4.0
Duration: 727.0022
Date: Thu, 20 Apr 2017 05:21:18 GMT
Content-Length: 407

{
  "@odata.context": "https://graph.microsoft.com/v1.0/
metadata#users/entity",
  "id": "12345678-73a6-4952-a53a-e9916737ff7f",
  "businessPhones": [
    "+1 5555555555"
  ],
  "displayName": "Chris Green",
  "givenName": "Chris",
  "jobTitle": "Software Engineer",
  "mail": null,
  "mobilePhone": "+1 5555555555",
  "officeLocation": "Seattle Office",
  "preferredLanguage": null,
  "surname": "Green",
  "userPrincipalName": "ChrisG@contoso.onmicrosoft.com"
}

```

**3.1.4 Analisis Menggunakan Refresh Token untuk Mendapatkan Access Token Baru**

Access token memiliki waktu yang singkat dan ketika sudah kadaluarsa, maka aplikasi yang akan dibuat harus meminta kembali access token yang supaya bisa terus mengakses data yang ada di dalam Microsoft Graph. Cara mendapatkan access token yang baru dengan menggunakan refresh token adalah dengan cara mengirimkan post request sekali lagi kepada endpoint /token dan untuk kali ini, gunakan refresh token sebagai parameter yang dikirimkan dan juga grant type yang berisikan refresh token dalam body dari request yang dilakukan seperti contoh pada table 3.11 dengan keterangan parameter seperti yang dijelaskan pada table 3.12.

Tabel 3.11: Tabel contoh *request* menggunakan *Refresh Token*

```

POST /common/oauth2/v2.0/token HTTP/1.1
Host: https://login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

client_id=6731de76-14a6-49ae-97bc-6eba6914391e
&scope=user.read%20mail.read
&refresh_token=OAAABAAAAiL9Kn2Z27UubvWFPbm0gLWQJVzCTE
9UkP3pSx1aXxUjq...
&redirect_uri=http%3A%2F%2Flocalhost%2Fmyapp%2F
&grant_type=refresh_token
&client_secret=JqQX2PNo9bpM0uEihUPzyrh

```

Tabel 3.12: Tabel parameter *request Refresh Token*

Parameter		Deskripsi
<i>client_id</i>	wajib	Nilai yang dipakai adalah nilai dari <i>application ID</i> yang didapatkan saat mendaftarkan aplikasi di <i>Microsoft App Registration Portal</i> .
<i>grant_type</i>	wajib	Harus diisi dengan nilai <i>refresh_token</i> .
<i>scope</i>	wajib	Daftar izin dari <i>Microsoft Graph</i> yang dipisahkan oleh cakupan yang diinginkan dan disetujui oleh pengguna. Dalam langkah ini, nilai dari <i>scope</i> pada langkah meminta <i>authorization_code</i> harus sama dengan langkah ini.
<i>refresh_token</i>	wajib	Refresh token yang didapat saat merequest token yang pertama kali.
<i>redirect_uri</i>	wajib	<i>Redirect uri</i> yang sama yang dipakai untuk mendapatkan <i>authorization code</i> .
<i>client_secret</i>	wajib untuk <i>web apps</i>	Application secret yang dibuat saat mendaftarkan aplikasi di portal registrasi untuk aplikasi yang didaftarkan.

- <sup>1</sup> Jika request ini berhasil, maka akan mengembalikan response seperti pada table 3.13 yang memiliki keterangan parameter yang dikembalikannya seperti pada table 3.14.

<sup>3</sup>

Tabel 3.13: Tabel contoh *response* menggunakan *Refresh Token*

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiI
sInG1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "user.read%20mail.read",
  "refresh_token": "AwABAAAAAvPM1KaPlrEqdFSBzjq
fTGAMxZGUTdM0t4B4...",
}
```

Tabel 3.14: Tabel parameter *response Refresh Token*

Parameter	Deskripsi
<i>access_token</i>	Access token yang diminta. Dengan memakai ini, maka aplikasi bisa memanggil Microsoft Graph.
<i>token_type</i>	Menunjukkan nilai dari token. Satu-satunya jenis token yang didukung oleh Azure AD adalah Bearer.
<i>expires_in</i>	Lamanya access token akan berlaku(dalam detik).
<i>scope</i>	Nilai scope yang valid untuk <i>access_token</i> yang diberikan.
<i>refresh_token</i>	Refresh token ini berguna untuk meminta kembali access token setelah access token itu berakhir. Refresh token memiliki umur yang panjang dan dapat digunakan untuk mempertahankan akses ke source.

### **3.1.5 Analisis Mendapatkan Data Events**

2 Data event di dalam Microsoft Graph tersimpan di dalam objek event yang memiliki relasi dengan  
 3 objek user dari pengguna. Untuk dapat mengakses event yang memiliki relasi dengan user, aplikasi  
 4 yang akan dibuat harus menjalankan method yang dimiliki objek user yaitu method list events  
 5 mengirimkan get request kepada endpoint /me/events. List events sendiri adalah method yang  
 6 berfungsi untuk mengembalikan objek-objek event yang berkaitan dengan objek user pengguna.  
 7 Untuk setiap operasi get yang mengembalikan objek event di Microsoft Graph, ada sebuah parameter  
 8 header “Prefer:outlook.timezone” yang berfungsi untuk menentukan time zone untuk mulainya dan  
 9 berakhirnya event. Sebagai contoh, dapat dilihat pada table 3.15.

Tabel 3.15: Tabel contoh *header time zone*

Prefer: outlook.timezone="Eastern Standard Time"
--

10 Pada table 3.15, outlook timezone diatur menjadi Eastern Standard Time yang nantinya semua  
 11 event yang dipanggil dengan header seperti itu akan mengembalikan starttime dan endtime dari  
 12 event akan disesuaikan dengan zona waktu Eastern Standard Time.

13 Untuk bisa mengakses method ini, maka diperlukan format header dari request seperti yang  
 14 akan dijelaskan pada table 3.16.

Tabel 3.16: Tabel parameter *header time zone*

Nama	Type	Deskripsi
<i>Authorization</i>	<i>String</i>	Bearer token. Bersifat wajib diisi.
<i>Prefer: outlook.timezone</i>	<i>String</i>	Digunakan untuk menentukan zona waktu yang akan dipakai untuk data yang akan dikembalikan. Bersifat optional.
<i>Prefer: outlook.body-content-type</i>	<i>String</i>	Merupakan nilai yang mengatur properti dari response body yang akan dikembalikan. Nilai bisa berupa "text" atau "html". Nilai default dari parameter ini adalah html. Bersifat optional.

- <sup>1</sup> Request ini juga bisa menerima parameter \$select yang berbentuk string query sebagai filter
- <sup>2</sup> mengenai field apa saja yang mau diambil dari objek event. Dapat dilihat fungsi dari parameter
- <sup>3</sup> string query \$select seperti pada contoh table 3.17 dan contoh responsenya pada table 3.18.

Tabel 3.17: Tabel contoh *request event*

```
GET https://graph.microsoft.com/v1.0/me/events?
$select=subject,bodyPreview,organizer,start,end,location
Prefer: outlook.timezone="Pacific Standard Time"
```

Tabel 3.18: Tabel contoh *response event*

```
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata
#users('cd209b0b-3f83-4c35-82d2-d88a61820480')/events(subject
, bodyPreview, organizer, start, end, location)",
  "value": [
    {
      "@odata.etag": "W/\\"ZlnW4RIAV06KYYwlrNzvQAAKGWwbw==\\",
      "id": "AAMkAGIAAAoZDOFAAA=",
      "subject": "Orientation",
      "bodyPreview": "Dana, this is the time you selected for
our orientation. Please bring the notes I sent you.",
      "start": {
        "dateTime": "2017-04-21T10:00:00.0000000",
        "timeZone": "Pacific Standard Time"
      },
      "end": {
        "dateTime": "2017-04-21T12:00:00.0000000",
        "timeZone": "Pacific Standard Time"
      },
      "location": {
        "displayName": "Assembly Hall",
        "locationType": "default",
        "uniqueId": "Assembly Hall",
        "uniqueIdType": "private"
      },
      "locations": [
        {
          "displayName": "Assembly Hall",
          "locationType": "default",
          "uniqueIdType": "unknown"
        }
      ],
      "organizer": {
        "emailAddress": {
          "name": "Samantha Booth",
          "address": "samanthab@a830edad905084922E170
20313.onmicrosoft.com"
        }
      }
    }
  ]
}
```

## <sup>1</sup> 3.2 Analisis Slack API

- <sup>2</sup> Untuk dapat memakai dan mengakses *method-method* yang disediakan oleh *Slack*, dibutuhkan  
<sup>3</sup> mendaftarkan aplikasi yang akan dibuat untuk bisa memperoleh *Client ID* yang nantinya dibutuhkan  
<sup>4</sup> untuk bisa mendapatkan *authorization code* serta *access token*. Sama seperti di *Microsoft Graph API*,  
<sup>5</sup> di dalam *Slack API access token* dibutuhkan sebagai otorisasi untuk bisa mengakses *method-method*  
<sup>6</sup> yang disediakan oleh *Slack*. Jika tidak memiliki *access token*, maka seluruh *method* yang dicoba

- 1 direquest tidak akan mengembalikan hasil dan dianggap sebagai *request* yang tidak *valid*.  
2 Pada sesi ini akan dibutuhkan *method* dari *Slack API* yang bisa mengubah status yang terdapat  
3 pada bagian *user.profile*. Dari informasi yang disediakan oleh Slack secara *online*, bahwa status  
4 tergabung dalam *user.profile*, maka untuk memenuhi dari kebutuhan di sisi ini dicoba menggunakan  
5 *method-method* yang bisa untuk mengakses kepada objek *user.profile*. Karena di sisi ini akan  
6 mengubah nilai dari status, asumsi sementara dari *method* yang bisa dipakai dari sisi ini adalah  
7 *method users.profile.set* yang akan berguna untuk mengubah isi dari profil pengguna yang di  
8 dalamnya terdapat status.

### 9 **3.3 Analisis Heroku Scheduler**

- 10 Untuk menjalankan aplikasi yang akan dibuat untuk mengambil data dari Microsoft Graph yang  
11 berhubungan dengan pengambilan data event, maka diperlukan aplikasi yang bisa mengambil data  
12 dengan mengirimkan request kepada Microsoft Graph, tetapi pengambilan data dibutuhkan secara  
13 berkala untuk memeriksa secara berkala data yang sudah dimasukkan ke dalam Microsoft Graph.  
14 Heroku Scheduler merupakan add-ons dari Heroku yang dapat menjalankan perintah secara berkala  
15 sesuai dengan yang sudah diatur sejak awal. Untuk menambahkan job pada Heroku Scheduler, dapat  
16 diakses melalui command line dengan perintah seperti:

17 \$ heroku addons:open scheduler

18 Setelah halaman baru pada browser muncul, pilih tombol untuk menambahkan job di scheduler  
19 dengan menekan tombol “Add Job”. Pada Heroku Scheduler tersedia interval untuk menjalankan  
20 sebuah job yaitu setiap 10 menit sekali, atau setiap jam pada menit ke, atau setiap hari pada jam  
21 ke.

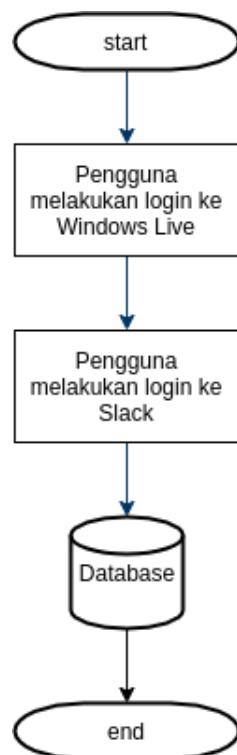
22 Untuk memperkecil perangkat lunak melewatkkan event yang tercatat di Outlook Calendar, maka  
23 bagian dari perangkat lunak yang melakukan sinkronisasi akan dijalankan setiap 10 menit sekali.

### 24 **3.4 Analisis Diagram Alir Sistem**

#### 25 **3.4.1 Bagian perangkat lunak yang berinteraksi dengan pengguna**

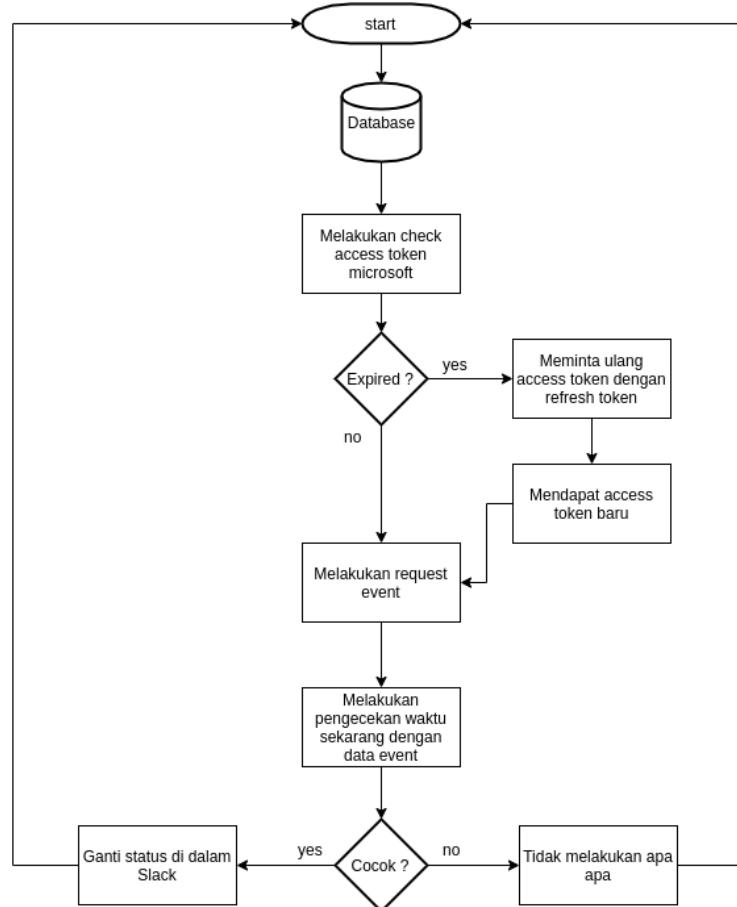
26 Diagram alir pada gambar 3.1 menunjukkan aliran proses yang terjadi pada bagian perangkat lunak  
27 yang berinteraksi dengan pengguna, berikut penjelasan dari aliran proses tersebut:

- 28 • Proses pengguna melakukan login ke Windows Live adalah proses dimana perangkat lunak  
29 akan mendapatkan access token dari Windows Live dan data lainnya yang dibutuhkan untuk  
30 mendapatkan data dari event yang tercatat di dalam Outlook Calendar.
- 31 • Proses pengguna melakukan login ke Slack adalah proses dimana perangkat lunak akan  
32 mendapatkan access token dari Slack untuk bisa mengganti status jika ada suatu event yang  
33 berjalan.
- 34 • Setelah kedua proses dijalankan, maka semua data yang dibutuhkan akan dimasukkan ke  
35 dalam database agar bisa dijalankan juga dengan bagian dari perangkat lunak yang lain.



Gambar 3.1: Diagram alir sistem pada bagian perangkat lunak yang berinteraksi dengan user.

### 1 3.4.2 Bagian perangkat lunak yang dijalankan secara berkala



Gambar 3.2: Diagram alir sistem pada bagian perangkat lunak yang dijalankan secara berkala.

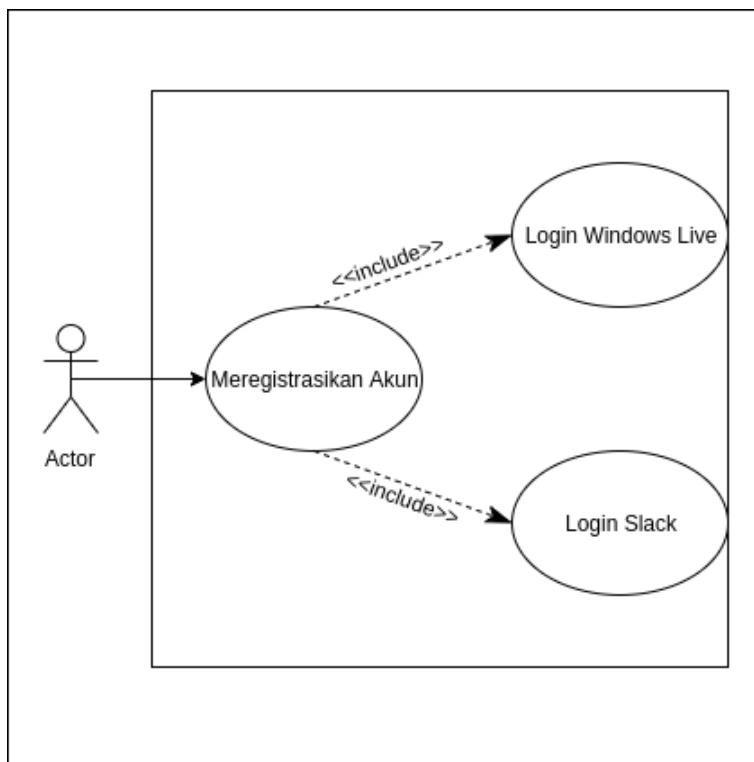
2 Diagram alir pada gambar 3.2 menunjukkan aliran proses yang dilakukan oleh bagian perangkat  
3 lunak yang dijalankan secara berkala, berikut penjelasan dari aliran proses tersebut:

- 4 • Pertama-tama perangkat lunak akan mengambil data dari dalam basis data.
- 5 • Lalu terdapat proses untuk melakukan pengecekan untuk access token dari Microsoft dikarenakan  
6 access token ini memiliki waktu kadaluarsa.
- 7 • Jika access token sudah kadaluarsa, maka meminta ulang access token dengan menggunakan  
8 refresh token.
- 9 • Jika access token belum expired, maka dilanjutkan dengan meminta data mengenai event  
10 yang ada di Outlook Calendar.
- 11 • Jika setelah melakukan permintaan ulang access token dan sudah mendapatkan access token  
12 yang baru, maka meminta data mengenai event yang ada di Outlook Calendar.
- 13 • Lalu setelah mendapatkan data event, maka melakukan pengecekan data event dengan waktu  
14 sekarang. Jika waktu sekarang beririsan dengan event yang sedang berjalan, maka perangkat  
15 lunak akan mengganti status yang ada pada Slack.

- Jika tidak ada yang beririsan, maka tidak akan melakukan apa-apa.
- Proses ini akan berjalan kembali secara berkala sesuai dengan waktu yang sudah diatur di dalam Heroku Scheduler.

## 3.5 Analisis Diagram Use Case

### 3.5.1 Use Case dengan actor: pengguna



Gambar 3.3: Diagram Use Case.

#### • Skenario *Login Windows Live*

Nama : Login Windows Live

Aktor : Pengguna

Kondisi Awal : Pengguna memiliki akun Windows Live

Deskripsi : Pengguna melakukan login pada aplikasi Windows Live dengan akun pribadinya.

Kondisi Akhir : Pengguna berhasil melakukan login ke Windows Live

Skenario :

- Pengguna melakukan login dengan id dan kata sandi yang dimilikinya.
- Pengguna mengizinkan akun Windows Live miliknya bekerjasama dengan perangkat lunak ini.
- Pengguna berhasil melakukan login ke Windows Live dan perangkat lunak mendapatkan akses.

Eksepsi :

- 1       – Pengguna tidak memiliki akun Windows Live.

2       ● **Skenario *Login Slack***

3       Nama : Login Slack

4       Aktor : Pengguna

5       Kondisi Awal : Pengguna memiliki akun Slack yang tergabung dalam suatu workspace

6       Deskripsi : Pengguna melakukan login pada aplikasi Slack di workspace pengguna tergabung.

7       Kondisi Akhir : Pengguna berhasil melakukan login ke workspace Slack

8       Skenario :

- 9       – Pengguna melakukan login dengan id dan kata sandi yang dimilikinya.
- 10      – Pengguna mengizinkan perangkat lunak ini menjadi bagian di dalam workspace Slack  
11       sebagai aplikasi pembantu di workspace tersebut.
- 12      – Pengguna berhasil melakukan login ke Slack dan perangkat lunak mendapatkan akses.

13      Eksepsi :

- 14      – Pengguna tidak memiliki akun Slack dan atau tidak terdaftar di dalam workspace  
15       manapun.

16       ● **Skenario Meregasistrasikan Akun** Nama : Meregistrasikan Akun

17       Aktor : Pengguna

18       Kondisi Awal : Pengguna memiliki akun Windows Live dan Slack

19       Deskripsi : Pengguna meregasistrasikan akunnya untuk bisa melakukan sinkronisasi antara  
20       Outlook Calendar dengan Slack.

21       Kondisi Akhir : Pengguna berhasil meregasistrasikan akunnya

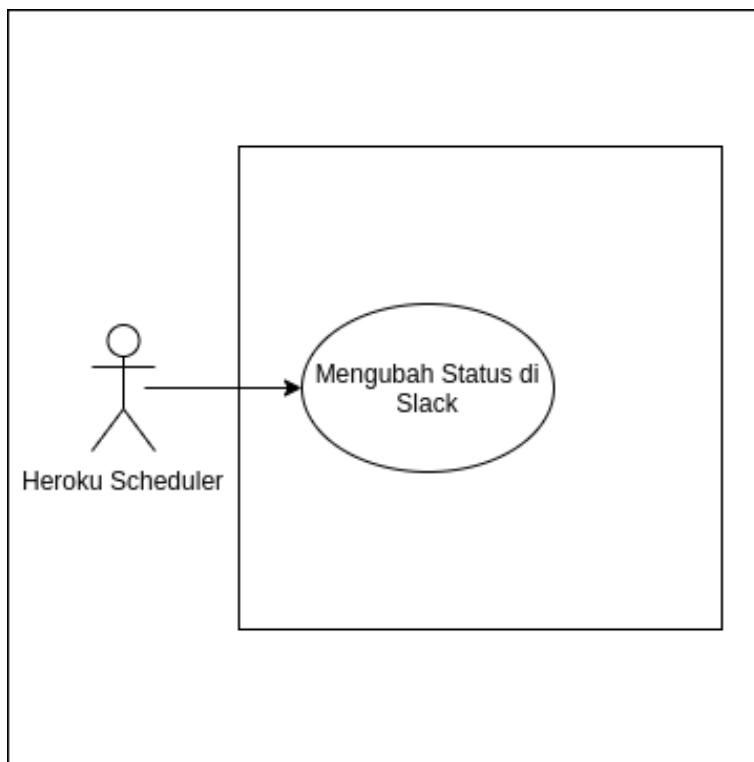
22       Skenario :

- 23      – Pengguna melakukan login pada Windows Live.
- 24      – Pengguna melakukan login pada Slack.
- 25      – Pengguna berhasil melakukan login ke kedua aplikasi tersebut dan pengguna berhasil  
26       meregasistrasikan akunnya untuk disinkronkan.

27      Eksepsi :

- 28      – Pengguna tidak memiliki akun Windows Live dan Slack.

<sup>1</sup> 3.5.2 Use Case dengan actor: Heroku Scheduler



Gambar 3.4: Diagram Use Case.

- <sup>2</sup> • Skenario Mengubah Status di Slack Nama : Merubah Status di Slack

<sup>3</sup> Aktor : Heroku Scheduler

<sup>4</sup> Kondisi Awal : Aktor memiliki data access token dari Windows Live maupun dari Slack.

<sup>5</sup> Deskripsi : Heroku Scheduler akan mengubah status pada Slack dengan mengambil data  
<sup>6</sup> event terlebih dahulu menggunakan access token Windows Live yang sudah tercatat, dan lalu  
<sup>7</sup> menggunakan access token Slack yang sudah tercatat untuk mengubah status pada Slack.

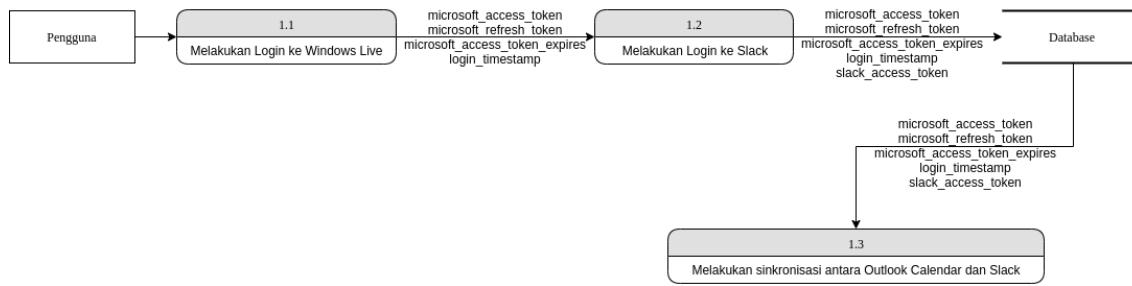
<sup>8</sup> Kondisi Akhir : Akun Slack pengguna berhasil diubah statusnya oleh Heroku Scheduler.

<sup>9</sup> Skenario :

- <sup>10</sup> – Heroku Scheduler mengambil access token Windows Live pengguna dari basis data dan  
<sup>11</sup> memeriksa masa kadaluarsanya.
- <sup>12</sup> – Heroku Scheduler mengambil data event dari Windows Live.
- <sup>13</sup> – Heroku Scheduler memeriksa event dengan waktu sekarang, lalu mengambil access token  
<sup>14</sup> Slack dan mengubah status jika ada event yang sedang berjalan.

<sup>15</sup> Eksepsi :

- <sup>16</sup> – Tidak ada data access token Windows Live dan Slack di basis data.



Gambar 3.5: Data Flow Diagram.

## 3.6 Analisis Data Flow Diagram

- Diagram alir data pada gambar 3.5 menunjukkan aliran data pada sistem, nerikut penjelasan dari setiap proses aliran data tersebut:

### • Proses melakukan *login ke Windows Live*

Input : ID dan Password akun Windows Live.

Output : `microsoft_access_token`, `microsoft_access_token_expires`, `microsoft_refresh_token`, `login_timestamp`

Proses ini merupakan proses meminta access token dan segala data yang diperlukan untuk mendapatkan data event dari Outlook Calendar. Proses ini menghasilkan data `microsoft_access_token`, `microsoft_access_token_expires`, `microsoft_refresh_token`, `login_timestamp`.

### • Proses melakukan *login ke Slack*

Input : *ID* dan *Password* beserta *workspace* di *Slack*.

Output : `microsoft_access_token`, `microsoft_access_token_expires`, `microsoft_refresh_token`, `login_timestamp`, `slack_access_token`

Proses ini merupakan proses meminta access token kepada aplikasi Slack agar perangkat lunak bisa mengubah status di dalam aplikasi Slack. Proses ini menghasilkan data `microsoft_access_token`, `microsoft_access_token_expires`, `microsoft_refresh_token`, `login_timestamp`, `slack_access_token`.

### • Proses melakukan *sinkronisasi antara Outlook Calendar dan Slack*

Input : Data dari basis data seperti token dari Windows Live dan juga dari Slack.

Output : - Proses ini merupakan proses sinkronisasi aplikasi Outlook Calendar dan juga Slack dengan cara mengambil data event yang terdapat di Outlook Calendar menggunakan token yang didapat dari Windows Live dan juga menggunakan token yang didapat dari Slack untuk dapat mengubah status dalam Slack jika waktu event dan waktu sekarang beririsan.



<sup>1</sup>

## BAB 4

<sup>2</sup>

### PERANCANGAN

- <sup>3</sup> Pada bab ini akan dibahas perancangan dari perangkat lunak mulai dari perancangan antarmuka,  
<sup>4</sup> dan perancangan untuk algoritma setiap *route* yang dipakai.

#### <sup>5</sup> 4.1 Perancangan Routing Handler

##### <sup>6</sup> 4.1.1 *Route* /

- <sup>7</sup> Pada *route* ini, akan memiliki kalimat pengantar dan juga sebuah tombol yang bisa membawa  
<sup>8</sup> pengguna untuk melakukan *login* kepada “Windows Live”. Tombol itu akan membawa pengguna  
<sup>9</sup> kepada halaman untuk melakukan *login* Windows Live dan juga setelah *login*, program akan meminta  
<sup>10</sup> izin untuk bisa mendapatkan data mengenai *event* yang terdapat pada kalender pengguna.

##### <sup>11</sup> 4.1.2 *Route* /authorize

- <sup>12</sup> *Route* ini akan muncul ketika langkah dari *route* / sudah selesai dijalankan sehingga *access token*  
<sup>13</sup> dan izin dari pengguna sudah didapatkan oleh program ini. Di *route* ini akan ada kalimat penjelasan  
<sup>14</sup> untuk halaman ini dan juga ada tombol untuk pengguna bisa melakukan *login* pada aplikasi  
<sup>15</sup> *Slack* untuk didapatkan *access token* dan izinya juga. Setelah tombol itu ditekan oleh pengguna,  
<sup>16</sup> maka tombol itu akan mengantarkan pengguna kepada halaman untuk mengisi *workspace* yang  
<sup>17</sup> dipakai oleh pengguna dan juga akun yang dipakai oleh pengguna untuk melakukan sinkronisasi  
<sup>18</sup> dengan *Outlook Calendar*. Setelah melakukan *login*, maka akan tampil halaman untuk pengguna  
<sup>19</sup> memberikan izin kepada program untuk bisa mengubah data termasuk status pengguna di dalam  
<sup>20</sup> aplikasi *Slack*.

##### <sup>21</sup> 4.1.3 *Route* /slackAuthorize

- <sup>22</sup> *Route* ini hanya menampilkan konfirmasi yang didapat dari hasil perekaman data dari *Slack* yang  
<sup>23</sup> berupa *access token* dari aplikasi tersebut.

##### <sup>24</sup> 4.1.4 *Route* /statusChanger

- <sup>25</sup> *Route* ini yang berfungsi mengeksekusi pengambilan data dari *Outlook Calendar* secara berkala,  
<sup>26</sup> dan jika ada data *event* yang sesuai dengan waktu sekarang, maka lewat *route* ini juga program ini  
<sup>27</sup> akan mengganti status pada aplikasi *Slack*. Sebelum melakukan pengambilan data pada *Outlook*  
<sup>28</sup> *Calendar*, pada *route* ini juga melakukan pengecekan pada *access token* yang didapat pada saat awal

1 melakukan *login Windows Live*. Jika *access token* yang terekam pada *database* sudah kadaluarsa,  
 2 maka melalui *route* ini juga program akan meminta *access token* yang baru untuk bisa mengambil  
 3 data *event* dari *Outlook Calendar*. Adapun pseudocode yang dirancang adalah seperti

4  
5

Listing 4.1: Pseudocode untuk /statusChanger

```

6 do connection to database.

7

8 router.get(/, function(){
9     timestampNow=new Date()

10

11     client.query("SELECT * FROM public.Credentials", (err, res)=>{
12         for each res dari hasil query
13             check masa kadaluarsa dari access token per row
14             if(expired){
15                 minta ulang dengan refresh token memakai
16                 function useRefreshToken()

17
18                 mendapatkan event dengan memakai access
19                 token yang baru memakai function getEvent()
20             }
21             else{
22                 mendapatkan event dengan memakai access
23                 token yang lama memakai function getEvent()
24             }
25         })
26     })

27

28     async function getEvent(accessToken, slackAccessToken){
29         result=mengirimkan request /me/events

30

31         for each result{
32             if(timestampNow>=startTime&&timestampNow<=endTime){
33                 panggil function changeStatusSlack()
34             }
35             else{
36                 tidak melakukan apa apa.
37             }
38         }
39     }

40     async function useRefreshToken(refresh_token){


```

```
1     melakukan request menggunakan refresh token  
2  
3     menyimpan data yang diperlukan kembali ke dalam tabel  
4  
5     return newToken  
6 }  
7  
8 async function changeStatusSlack(slack_access_token , endTime){  
9     mengirimkan request dengan menggunakan slack access token  
10    ditambah dengan parameter status_expiration diisi dengan endtime  
11 }
```

## 12 4.2 Perancangan *Helper*

### 13 4.2.1 auth.js

14 Pada file javascript ini, terdapat kode yang membantu untuk melakukan *request* untuk mendapatkan  
15 *access token* dari *Microsoft Graph* maupun dari *Slack*. Cara mendapatkan *access token* di file  
16 ini dengan cara menggunakan *library simple oauth2* yang membantu untuk melakukan *request* ke  
17 aplikasi yang akan dituju. Selain untuk meminta *akses token*, di dalam file ini juga berisi kode yang  
18 membantu program ini agar *access token* dan data yang dibutuhkan lainnya disimpan di dalam  
19 basis data. Pada *helper* ini terdapat konstanta dan juga fungsi-fungsi seperti:

- 20 • *Constanta*
  - 21 – **{Client}**: Konstanta ini merupakan konstanta yang memanggil *library* dari *postgres*.  
22 *Postgres* merupakan jenis basis data yang dipakai di dalam program ini.
  - 23 – **client**: Konstanta yang menginisialisasi kelas *Client* yang dipanggil dari konstanta  
24 sebelumnya. Konstanta ini menginisialisasi kelas *Client* dengan parameter *url* dari basis  
25 data yang dipakai dan juga nilai *boolean* untuk *ssl*.
  - 26 – **credentials**: Konstanta ini bertipe *Object* yang berfungsi sebagai objek yang dibutuhkan  
27 *simple oauth2* untuk melakukan *request* kepada *Microsoft Graph*. Objek dari konstanta  
28 ini memiliki isi yaitu *client* dan juga *auth*. *Client* memiliki isi data *id* dan juga *secret*  
29 yang merupakan *id* dan *secret* yang didapat saat mendaftarkan aplikasi di *azure portal*.  
30 Sedangkan *Auth* berisi *tokenHost*, *authorizePath*, dan *tokenPath* yang berisikan data *url*  
31 untuk melakukan *request access token* ke *Microsoft Graph*.
  - 32 – **credentialsSlack**: Konstanta ini bertipe *Object* yang berfungsi sebagai objek yang  
33 dibutuhkan *simple oauth2* untuk melakukan *request* kepada *Slack*. Objek dari konstanta  
34 ini memiliki isi yaitu *client* dan *auth*. *Client* memiliki isi data *id* dan juga *secret* yang  
35 merupakan *id* dan *secret* yang didapat saat mendaftarkan aplikasi di *Slack*. Sedangkan  
36 *Auth* berisi *tokenHost*, *authorizePath*, dan *tokenPath* yang berisikan data *url* untuk  
37 melakukan *request access token* ke *Slack*.

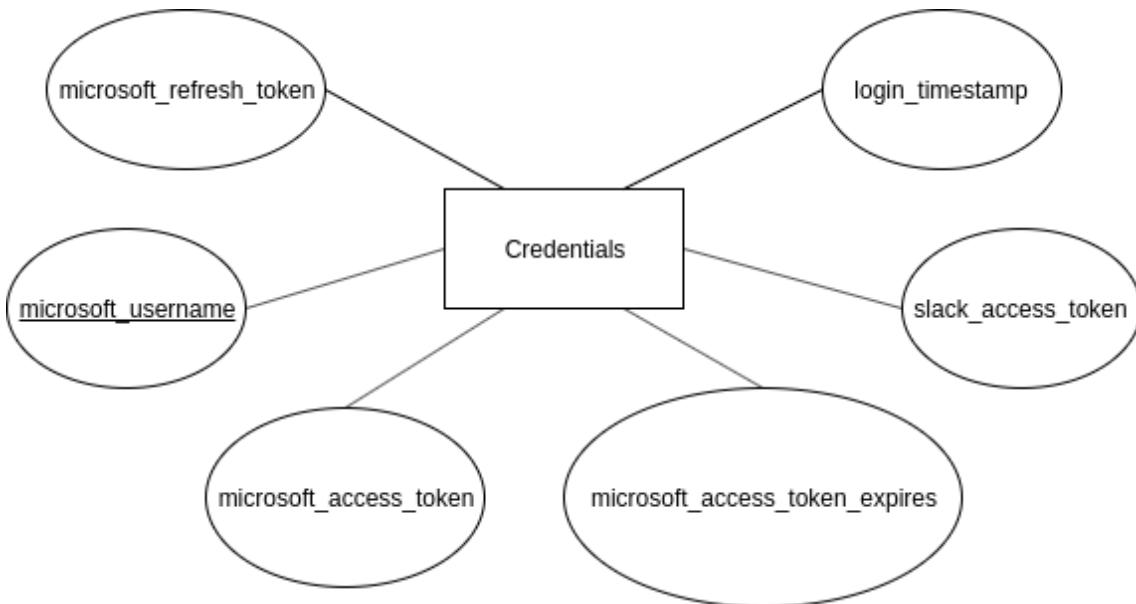
- 1   – **oauth2**: Konstanta yang digunakan untuk menggunakan dan membuat *simple oauth2*  
2   yang melakukan koneksi kepada *Microsoft Graph*.
- 3   – **oauth2Slack**: Konstanta yang digunakan untuk menggunakan dan membuat *simple*  
4   *oauth2* yang melakukan koneksi kepada *Slack*.
- 5   – **jwt**: Konstanta ini untuk memanggil *library jsonwebtoken*.
- 6   – **databaseValue**: Konstanta ini berbentuk *Object* yang akan digunakan untuk menam-  
7   pung nilai-nilai yang akan dimasukan ke dalam tabel basis data.

8   ● *Function*

- 9   – **getAuthUrl()**: Fungsi ini mengembalikan *url* untuk meminta *authorization code* pada  
10   *Microsoft Graph*. *Url* yang dikembalikan sudah lengkap dengan *parameter* yang dibutuhkan  
11   untuk meminta *authorization code*. *Url* ini akan membawa pengguna untuk  
12   melakukan *login* awal dan meminta izin program ini ke *Microsoft Graph*.
- 13   – **getTokenFromCode(auth\_code)**: Fungsi ini berfungsi untuk mengubah *authoriza-*  
14   *tion code* yang didapat dari proses *login* awal menjadi *access token*. Setelah berhasil  
15   mendapatkan *access token*, maka fungsi ini akan membongkar *id\_token* yang didapatkan  
16   dari kembalian data setelah meminta *access token* dengan menggunakan *jsonwebtoken*  
17   untuk mendapatkan data pengguna seperti nama, *email*, dan lain-lain. Setelah berhasil  
18   membongkar *id\_token*, maka di dalam fungsi ini akan memasukkan beberapa nilai ke  
19   dalam konstanta *databaseValue* seperti *microsoft\_username* yang didapat dari *prefer-*  
20   *red\_username* hasil dari membongkar *id\_token*, *microsoft\_access\_token* yang didapat  
21   dari nilai *access\_token* dan merupakan *access token* yang didapatkan dari *request* yang  
22   dilakukan, *microsoft\_access\_token\_expires* yang didapat dari nilai *expires\_in* dari res-  
23   pon yang didapat, *microsoft\_refresh\_token* yang didapatkan dari nilai *refresh\_token*  
24   dari respon yang berfungsi untuk meminta *access token* yang baru ketika yang lama  
25   sudah kadaluarsa, dan juga *login\_timestamp* yang diambil dari waktu saat pengguna  
26   melakukan *login*. Lalu fungsi ini akan mengembalikan nilai berupa *access token* yang  
27   didapat.
- 28   – **getAuthUrlSlack()**: Fungsi ini mengembalikan *url* untuk meminta *authorization code*  
29   pada *Slack*. *Url* yang dikembalikan sudah lengkap dengan *parameter* yang dibutuhkan  
30   untuk meminta *authorization code*. *Url* ini akan membawa pengguna untuk melakukan  
31   *login* dan memberikan program ini izin ke dalam *Slack*.
- 32   – **getTokenFromCodeSlack(auth\_code)**: Fungsi ini berfungsi untuk mengubah *au-*  
33   *uthorization code* yang didapat dari *Slack* dengan *access token* yang akan digunakan untuk  
34   mengubah status di dalam *Slack*. Melalui fungsi ini, konstanta *databaseValue* akan  
35   ditambahkan dengan nilai *slack\_access\_token* yang diisi dengan nilai *access token* yang  
36   didapat dari respon saat meminta *access token*. Setelah mendapatkan *access token* dari  
37   *Slack*, maka fungsi ini selanjutnya akan melakukan pemeriksaan pada seluruh *record*  
38   yang ada di basis data. Jika untuk *microsoft\_username* yang didapatkan sekarang sudah  
39   ada, maka fungsi ini hanya akan melakukan *update* pada isi dari *row* basis data tersebut.  
40   Tetapi jika belum ada *record* dengan *microsoft\_username* tersebut, maka fungsi ini

akan melakukan *insert* kepada tabel dalam basis data yang dipakai. Fungsi ini juga mengembalikan *access token* yang didapat dari aplikasi *Slack*.

### 4.3 Perancangan Basis Data



Gambar 4.1: ER Diagram untuk tabel *Credentials*.

Pada program ini, digunakan 1 tabel basis data yang berfungsi untuk menampung data kredensial dari pengguna yang sudah melakukan *login* dan sudah memberikan izin terhadap program ini untuk mengakses *Microsoft Graph* dan *Slack*-nya. Tabel tersebut diberi nama *Credentials*. Kolom yang terdapat pada tabel basis data ini antara lain:

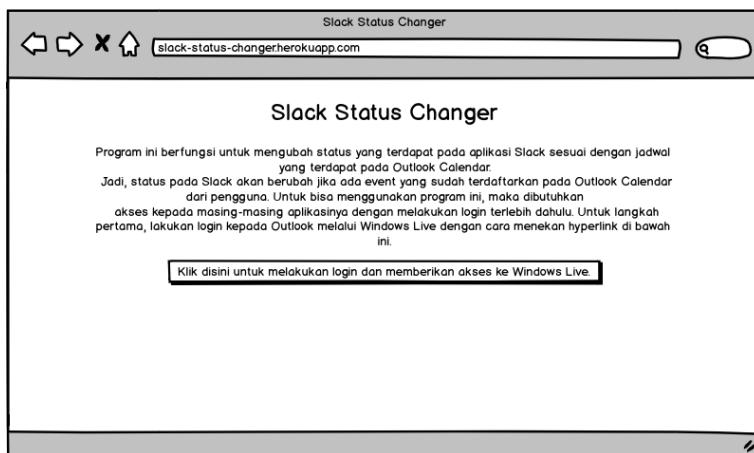
- *microsoft\_username*(TEXT) Kolom ini berfungsi sebagai *primary key* yang membedakan antara satu pengguna dengan pengguna lainnya. Nilai untuk kolom ini diambil dari nilai yang didapat saat program meminta *access token*. Saat meminta *access token*, *Microsoft Graph* juga mengembalikan *id\_token* yang jika dilakukan *decode* akan berisi data pengguna yang melakukan *login* termasuk data *username* yang dipakai oleh pengguna.
- *microsoft\_refresh\_token*(TEXT) Kolom ini berfungsi untuk menampung *refresh token* yang didapat saat melakukan *request* ke *Microsoft Graph* yang digunakan untuk melakukan *request* ulang *access token* setelah *access token* sebelumnya kadaluarsa.
- *microsoft\_access\_token\_expires*(INTEGER) Kolom ini berfungsi untuk menampung lamanya masa *access token* akan berlaku. Nilai ini didapat saat awal melakukan *request access token* dengan mengambil nilai *expires\_in* dari respon yang didapat.
- *microsoft\_access\_token*(TEXT) Kolom ini berfungsi untuk menampung *access token Microsoft Graph* yang didapat dari *request*.
- *slack\_access\_token*(TEXT) Kolom ini berfungsi untuk menampung *access token Slack* yang didapat dari *request*.

- login\_timestamp(TEXT) Kolom ini berfungsi untuk menampung waktu saat pengguna melakukan *login*.

## 4.4 Perancangan Antarmuka

### 4.4.1 Route /

Pada route ini, antarmuka yang dirancang akan seperti pada gambar 4.2. Di dalam perancangan antarmuka route ini akan memiliki sebuah tulisan yang berisi penjelasan fungsi dan mengarahkan kepada pengguna untuk menggunakan perangkat lunak ini. Selain ada sebuah tulisan untuk petunjuk, di dalam halaman ini juga terdapat sebuah tombol yang akan membawa pengguna untuk melakukan login kepada Windows Live dan perangkat lunak ini akan mencatat access token yang didapat dari Windows Live dan segala data yang diperlukan.



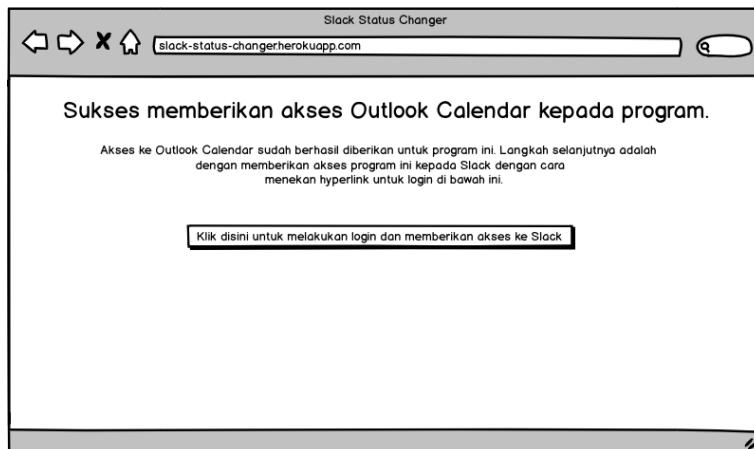
Gambar 4.2: Rancangan antarmuka halaman awal.

### 4.4.2 Route /authorize

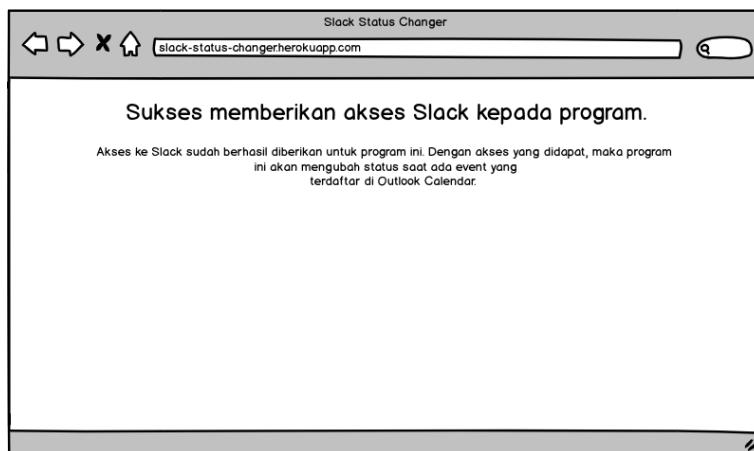
Pada route ini, antarmuka yang dirancang seperti pada gambar 4.3. Di dalam perancangan antarmuka route ini akan ada sebuah tulisan lagi yang membantu mengarahkan pengguna untuk melakukan langkah selanjutnya agar bisa memakai perangkat lunak ini dengan baik. Lalu di dalam perancangan ini juga terdapat sebuah tombol yang mengarahkan pengguna melakukan langkah selanjutnya yaitu melakukan login ke aplikasi Slack. Tombol itu juga akan membawa pengguna ke antarmuka dari Slack untuk melakukan login.

### 4.4.3 Route /slackAuthorize

Pada route ini, antarmuka yang dirancang seperti pada gambar 4.4. Di dalam perancangan antarmuka route ini, hanya akan ada sebuah tulisan yang merupakan sebuah pesan bahwa semua langkah yang dijalankan sudah berhasil dan sudah berhasil untuk memakai perangkat lunak ini.



Gambar 4.3: Rancangan antarmuka halaman setelah login Windows Live.



Gambar 4.4: Rancangan antarmuka halaman setelah login Slack.



1

## BAB 5

2

### IMPLEMENTASI DAN PENGUJIAN

3 Pada bab ini akan dibahas mengenai implementasi dari antarmuka perangkat lunak, dan pengujian  
4 perangkat lunak di *workspace* tempat program ini diregistrasikan beserta pengujian eksperimental  
5 perangkat lunak di *workspace* lain dari *Slack*.

6 **5.1 Implementasi**

7 Untuk mengimplementasi perangkat lunak ini, langkah awal yang dikerjakan adalah mencoba  
8 membuat program yang awalnya berjalan di *localhost*. Setelah berjalan dan berhasil di *localhost*,  
9 maka langkah selanjutnya yaitu membuat akun *Heroku* lalu membuat sebuah aplikasi baru yang  
10 berbasis *Node.js* serta melakukan *deploy* kode perangkat lunak yang sudah berhasil berjalan  
11 di *localhost* ke *Heroku*. Aplikasi yang dibuat di *Heroku* diberi nama *slack-status-changer* dan  
12 dapat diakses dengan cara membuka <https://slack-status-changer.herokuapp.com/>. Setelah  
13 melakukan *deploy* ke dalam *Heroku*, maka perangkat lunak juga dilakukan pengujian agar fungsi  
14 yang sudah berjalan pada saat di *localhost* masih bisa berjalan semua pada saat setelah perangkat  
15 lunak dimasukkan ke dalam *Heroku*. Lalu setelah semua berjalan dengan sesuai fungsinya, maka  
16 langkah yang dilakukan selanjutnya adalah memasukkan sebuah *job* pada *Heroku Scheduler* untuk  
17 *scheduler* menjalankan *job* itu sesuai iterasi waktunya yaitu per 10 menit sekali.

18 **5.1.1 Lingkungan Pengembangan**

19 Pada subbab ini akan disebutkan spesifikasi perangkat keras maupun perangkat lunak yang dipakai  
20 untuk menyusun skripsi ini:

21 **Spesifikasi Perangkat Keras**

- 22 • Processor Intel Core i5-8250U CPU @ 1.60GHz × 8
- 23 • Graphics Intel UHD Graphics 620 *KabylakeGT2*
- 24 • RAM 12 GB
- 25 • Harddisk 1000GB HDD

26 **Spesifikasi Perangkat Lunak**

- 27 • Sistem Operasi Ubuntu 18.04.3 LTS 64-bit

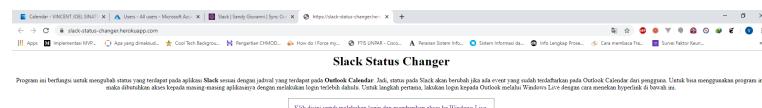
- 1     • Atom
- 2     • Node.js version 8.10.0
- 3     • NPM version 6.8.0
- 4     • pgAdmin4 version 4.13 (Application Mode: Desktop)

### 5 5.1.2 Implementasi Basis Data

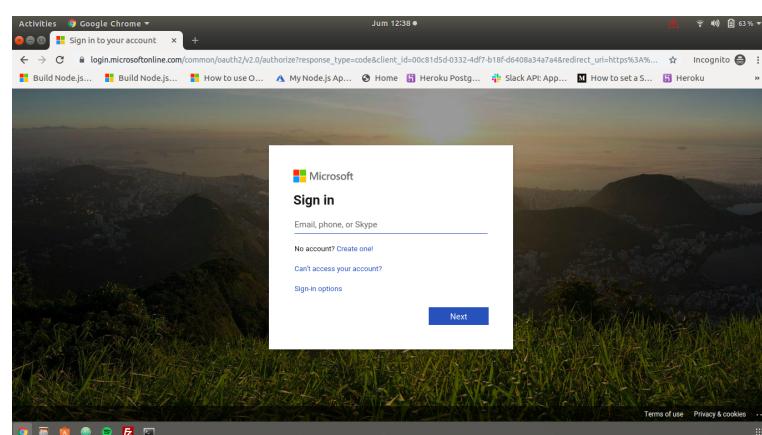
6 Untuk basis data yang diimplementasikan, tabel yang disediakan hanya ada 1 tabel yang bernama  
 7 *Credentials*. Tabel basis data ini menampung data-data kredensial dari pengguna yang menjalankan  
 8 dan mendaftarkan akunnya ke program ini. Tabel ini dibuat dengan bantuan *pgAdmin4* yang  
 9 melakukan koneksi kepada *url* yang diberikan *Heroku Postgres* sehingga tabel ini disimpan pada  
 10 *Heroku Postgres* yang bersangkutan dengan perangkat lunak yang di *deploy* ke *Heroku*.

### 11 5.1.3 Implementasi Antarmuka

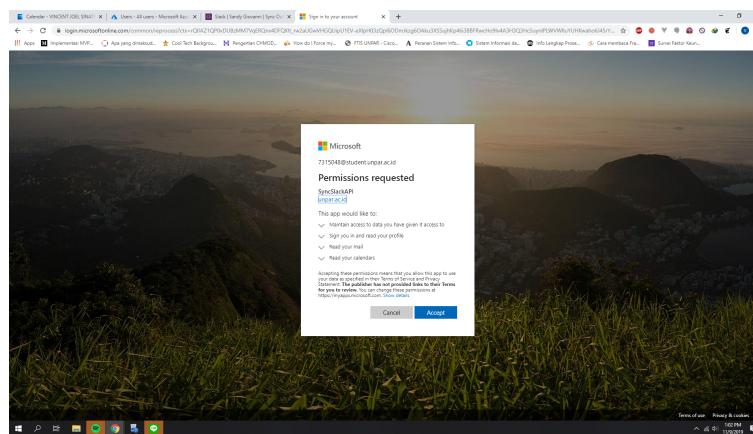
12 Hasil implementasi dari antarmuka perangkat lunak ini dapat dilihat pada gambar 5.1, gambar 5.2,  
 13 gambar 5.3, gambar 5.4, gambar 5.5, gambar 5.6, dan gambar 5.7.



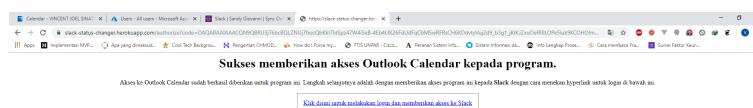
Gambar 5.1: Antarmuka halaman awal.



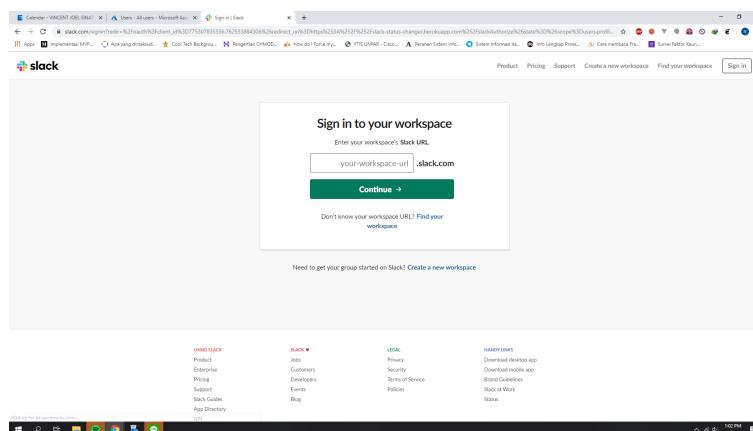
Gambar 5.2: Antarmuka untuk login Windows Live.



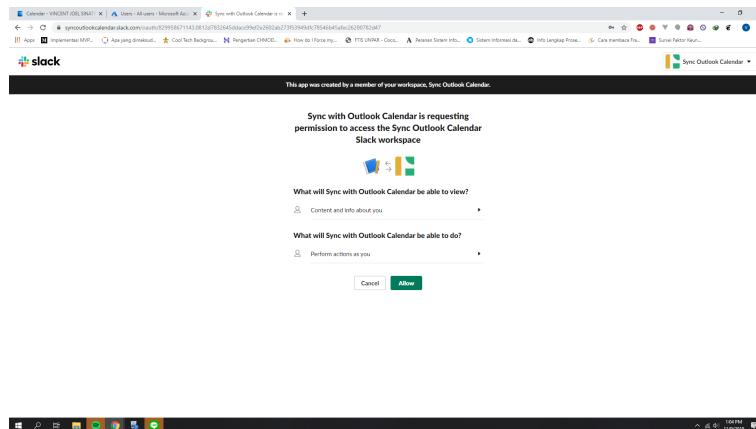
Gambar 5.3: Antarmuka untuk memberikan Windows Live izin ke perangkat lunak.



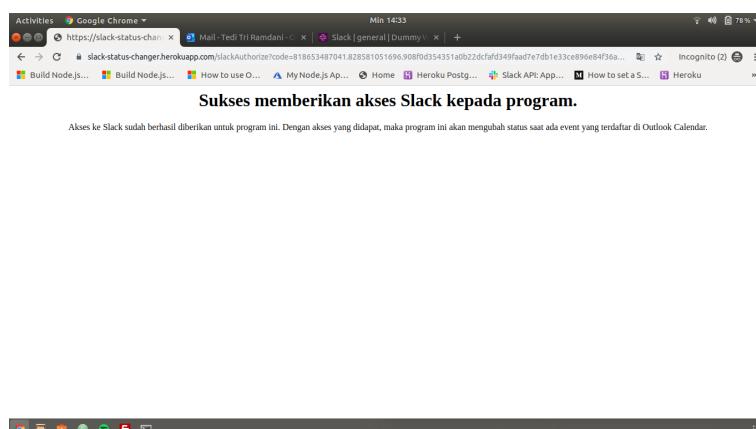
Gambar 5.4: Antarmuka petunjuk login ke Slack.



Gambar 5.5: Antarmuka untuk login Slack.



Gambar 5.6: Antarmuka untuk memberikan Slack izin ke perangkat lunak.



Gambar 5.7: Antarmuka saat selesai melakukan login dan pemberian izin.

1 Pada gambar 5.1 merupakan tampilan awal yang berisikan penjelasan dari perangkat lunak  
2 ini untuk dapat dipahami oleh pengguna serta sebuah tombol yang membawa pengguna masuk ke  
3 halaman *login Windows Live* seperti pada gambar 5.2. Setelah *login*, maka akan muncul tampilan  
4 seperti gambar 5.3 untuk perangkat lunak meminta izin kepada pengguna untuk perangkat lunak  
5 mengakses dan mendapatkan informasi data *event* di dalam *Outlook Calendar*.

6 Setelah berhasil melakukan *login* dan pengguna memberikan izin akses kepada perangkat lunak  
7 ini, maka akan tampil tampilan seperti tampilan gambar 5.4 yang berisi penjelasan akan langkah  
8 selanjutnya yang perlu dilakukan pengguna untuk bisa memakai perangkat lunak ini dan juga ada  
9 satu tombol yang mengarahkan pengguna melakukan langkah yaitu melakukan *login* seperti pada  
10 gambar 5.5. Setelah berhasil melakukan *login*, maka akan muncul tampilan untuk meminta izin  
11 dari pengguna *Slack* kepada perangkat lunak seperti pada gambar 5.6.

12 Setelah pengguna berhasil *login* dan memberikan akses kepada perangkat lunak maka akan  
13 ditampilkan halaman seperti gambar 5.7.

14 **5.2 Pengujian**

15 Pada bagian pengujian, akan dibagi kepada 2 tipe pengujian yaitu pengujian fungsional dan juga  
16 pengujian eksperimental. Pada pengujian fungsional, akan dicoba tombol-tombol sesuai dengan  
17 fungsinya dan juga memastikan bahwa perangkat lunak bisa berjalan dengan baik pada *workspace*  
18 tempat perangkat lunak ini didaftarkan, sedangkan pada pengujian eksperimental akan dilakukan  
19 pengujian untuk *workspace* lain selain *workspace* tempat perangkat lunak ini didaftarkan.

20 **5.2.1 Pengujian Fungsional**

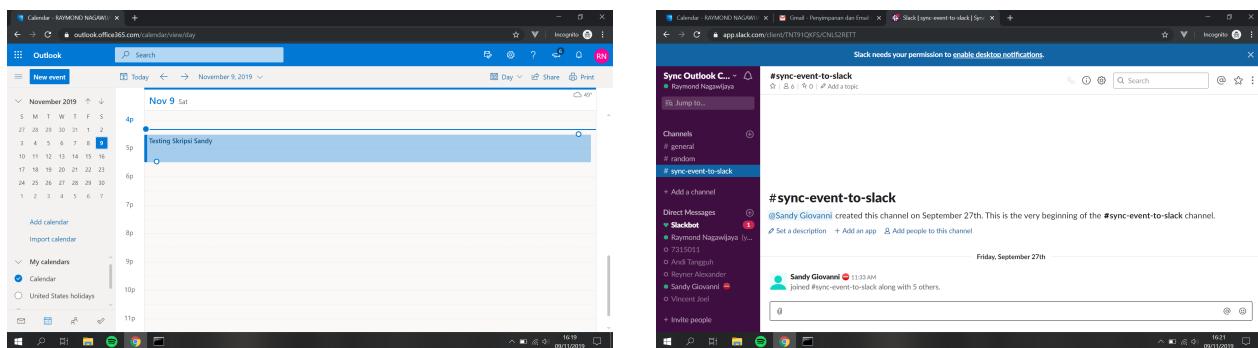
21 **Hasil Pengujian Fungsionalitas Penggunaan Perangkat Lunak Integrasi *Outlook Calendar* dan *Slack***

23 Tujuan dari diadakannya pengujian bagian ini adalah untuk membuktikan bahwa perangkat lunak  
24 ini sudah bisa mencapai tujuan yang ingin dituju yaitu perangkat lunak akan menggantikan status  
25 dari pengguna di dalam *workspace* tempat perangkat lunak ini didaftarkan saat ada *event* yang  
26 terdaftar di dalam *Outlook Calendar*. Untuk melakukan pengujian ini, akan diambil beberapa  
27 pengguna yang memiliki *Outlook Calendar* dan juga pengguna tersebut diundang masuk dalam  
28 *workspace* tempat perangkat lunak ini didaftarkan.

29 Gambar 5.8a sampai dengan gambar 5.16 menunjukkan bukti hasil dari pengguna yang dimintai  
30 ketersediaannya untuk melakukan pengujian terhadap perangkat lunak ini.

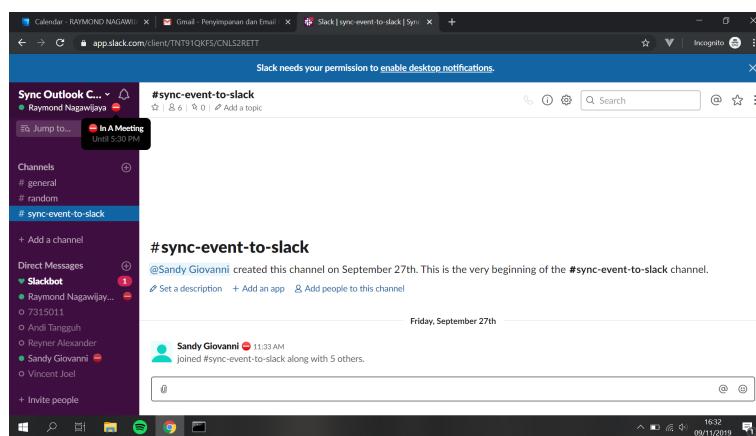
31 Dari gambar 5.8a sampai gambar 5.9 merupakan bukti yang diambil dari pengguna yang  
32 melakukan pengujian yang bernama Raymond. Pada gambar 5.8a merupakan bukti bahwa Raymond  
33 sudah membuat sebuah *event* pada waktu 16.30 dan gambar 5.8b menunjukkan kondisi *Slack* pada  
34 pukul 16.21 yang tidak memperlihatkan adanya status yang terpasang pada akun pengguna. Lalu  
35 pada gambar 5.9 menunjukkan kondisi *Slack* pada pukul 16.32 yang memperlihatkan adanya status  
36 yang terpasang pada akun pengguna yang status itu akan terpasang hingga waktu *event* berakhir  
37 yaitu pukul 5.30.

38 Dari gambar 5.10a sampai gambar 5.10b merupakan bukti dari hasil pengujian dari pengguna  
39 yang bernama Reyner. Pada gambar 5.10a dapat dilihat bahwa pengguna memiliki sebuah *event*



(a) Tampilan *event* yang dibuat pengguna (Raymond). (b) Tampilan *Slack* sebelum *event* dimulai (Raymond).

Gambar 5.8



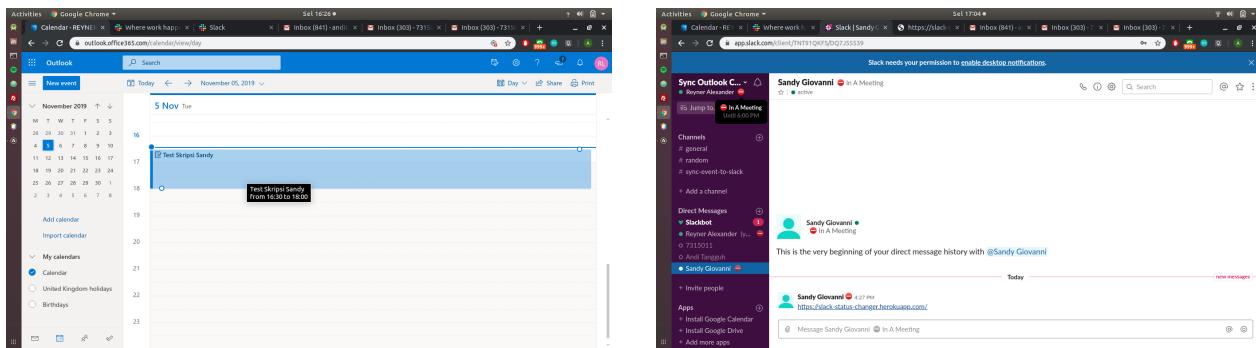
Gambar 5.9: Tampilan *Slack* setelah *event* dimulai (Raymond).

1 yang dimulai pada pukul 16.30 dan berakhir pada pukul 18.00. Lalu pada gambar 5.10b dapat  
2 dilihat bahwa status pengguna telah berubah dan status itu berlaku sampai pukul 18.00 tepat saat  
3 *event* berakhir.

4 Pada gambar 5.11a sampai gambar 5.12 merupakan bukti hasil pengujian yang dilakukan oleh  
5 pengguna bernama Sandy. Pada gambar 5.11a terlihat bahwa ada *event* yang tercatat akan dimulai  
6 pada pukul 14.00 dan pada pukul 13.06 belum ada status yang tertulis di akun pengguna yang  
7 terekam dalam gambar 5.11b. Lalu pada pukul 14.14 status muncul dalam akun pengguna yang  
8 status itu akan hilang pada pukul 15.00 sesuai dengan yang tertulis di dalam *Outlook Calendar*.

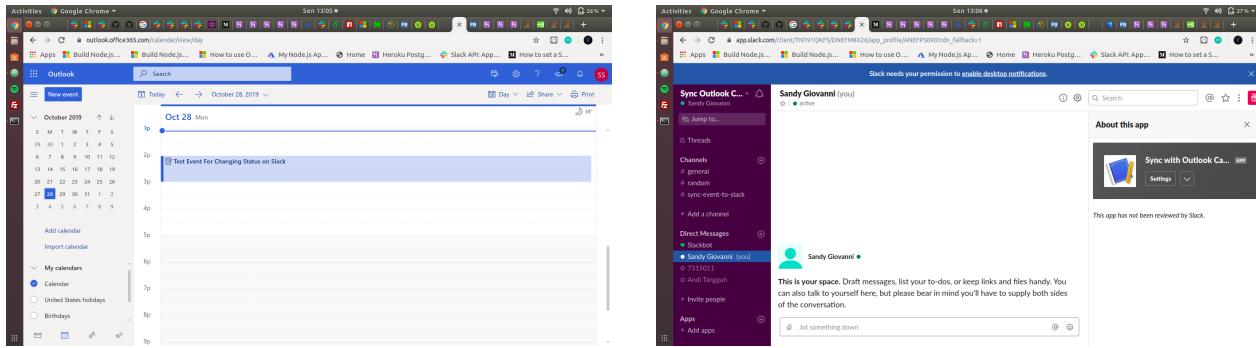
9 Pada gambar 5.13a sampai dengan gambar 5.14 merupakan hasil dari pengujian yang dilakukan  
10 pengguna bernama Vincent. Pada gambar 5.13a terlihat ada sebuah *event* yang akan dimulai pada  
11 pukul 13.00 dan pada gambar 5.13b dapat terlihat bahwa pada pukul 12.58 tidak ada status di  
12 dalam akun pengguna lalu pada pukul 13.05 terdapat status dalam akun pengguna yang statusnya  
13 akan hilang pada pukul 15.00 seperti di gambar 5.14 sesuai dengan keterangan *event* yang terdapat  
14 pada *Outlook Calendar*.

15 Pada gambar 5.15a sampai dengan gambar 5.16 merupakan hasil dari pengujian yang dilakukan  
16 oleh pengguna bernama Yonathan. Di gambar 5.15a terdapat *event* yang mulai pada pukul 16.30  
17 dan pada gambar 5.15b yang diambil pada pukul 16.07, belum ada status yang terpasang di akun  
18 pengguna. Pada gambar 5.16 yang diambil pada pukul 16.36 sudah ada status yang terpasang di  
19 akun pengguna dan status itu akan hilang pada pukul 17.30 yang sesuai dengan *event* yang tercatat



(a) Tampilan *event* yang dibuat pengguna (Reyner). (b) Tampilan *Slack* setelah *event* dimulai (Reyner).

Gambar 5.10



(a) Tampilan *event* yang dibuat pengguna (Sandy). (b) Tampilan *Slack* sebelum *event* dimulai (Sandy).

Gambar 5.11

1 pada *Outlook Calendar*.

## 2 5.2.2 Pengujian Eksperimental

### 3 Skenario Pengujian

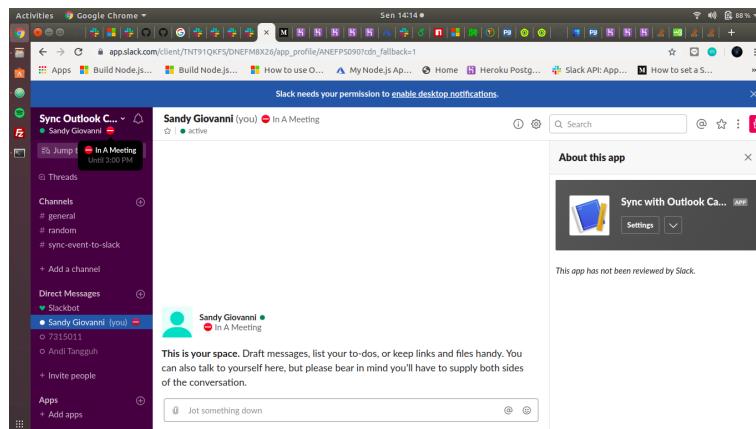
4 Pengujian eksperimental ini dilakukan dengan pengguna-pengguna yang tergabung dalam *workspace*  
 5 di luar dari *workspace* tempat perangkat lunak ini didaftarkan. Pengguna harus memiliki *Outlook*  
 6 *Calendar* dan juga pengguna membuat sebuah *event* yang didaftarkan di dalam *Outlook Calendar*.

### 7 Tujuan Pengujian

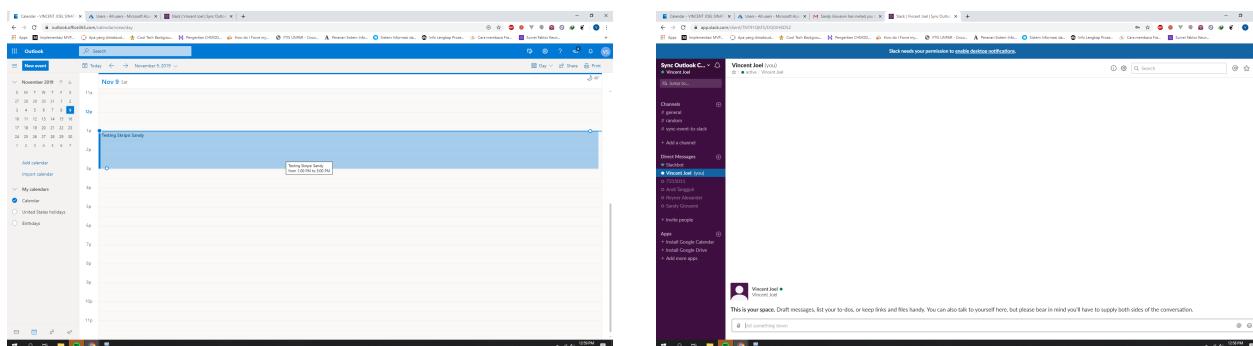
8 Tujuan dari pengujian ini adalah untuk mengetahui bahwa perangkat lunak yang didaftarkan  
 9 di satu *workspace* di dalam *Slack* tidak hanya terikat pada satu *workspace* itu saja tetapi bisa  
 10 digunakan ke *workspace* lainnya asalkan pengaturan saat pendaftaran perangkat lunak di dalam  
 11 *Slack* mengaktifkan “*Manage Distribution*”.

### 12 Hasil Pengujian

13 Dari gambar 5.17a sampai dengan gambar 5.28 merupakan hasil dari para pengguna yang bersedia  
 14 untuk menguji perangkat lunak di *workspace* yang dimiliki masing-masing atau di luar dari *workspace*  
 15 tempat perangkat lunak ini didaftarkan.



Gambar 5.12: Tampilan *Slack* setelah *event* dimulai (Sandy).



(a) Tampilan *event* yang dibuat pengguna (Vincent). (b) Tampilan *Slack* sebelum *event* dimulai (Vincent).

Gambar 5.13

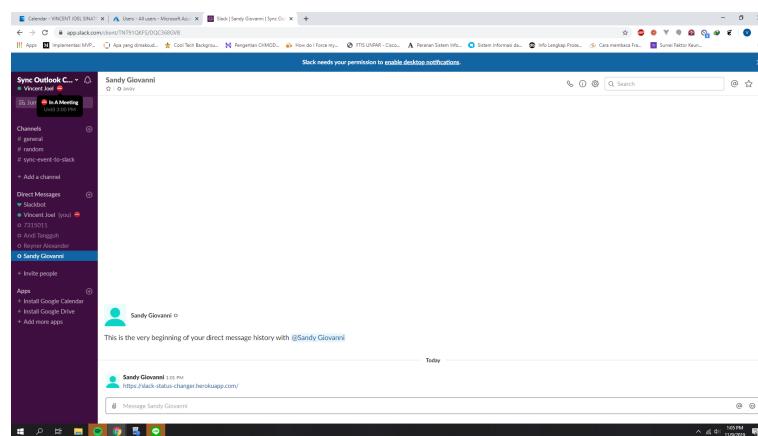
1 Gambar 5.17a sampai dengan gambar 5.18 merupakan hasil yang didapat dari pengguna yang  
 2 bernama Chris yang merupakan seorang admin di lab FTIS. Pada gambar 5.17a terlihat ada *event*  
 3 yang terdaftar pukul 16.30. Lalu pada gambar 5.17b belum terdapat status yang terpasang pada  
 4 akun pengguna dan *workspace* yang pengguna pakai adalah *workspace* yang bernama Lab FTIS.  
 5 Pada gambar 5.18 dapat terlihat bahwa status sudah terpasang pada akun pengguna.

6 Gambar 5.19a sampai dengan gambar 5.20 merupakan hasil yang didapat dari hasil pengujian  
 7 yang dilakukan oleh pengguna yang bernama Ferdian yang merupakan seorang admin di lab  
 8 FTIS. Pengguna ini menguji perangkat lunak pada *workspace* Lab FTIS. Di gambar 5.19a dapat  
 9 dilihat bahwa pengguna memiliki *event* yang akan berjalan pukul 17.00 dan waktu di komputer  
 10 nya menunjukkan pukul 16.52. Lalu pada gambar 5.19b dapat terlihat bahwa tidak ada status  
 11 yang terpasang pada akun pengguna dan pada gambar 5.20 terlihat bahwa sudah ada status yang  
 12 terpasang pada akun pengguna pada pukul 17.02 dan status itu akan berakhir pada pukul 18.00.

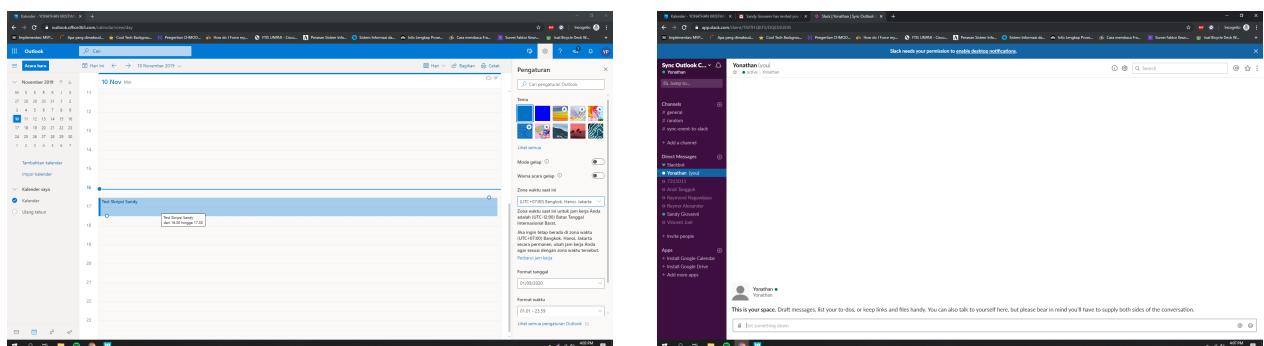
13 Dari gambar 5.21a sampai dengan gambar 5.22 merupakan bukti yang didapatkan oleh pengguna  
 14 yang bernama Yehezkiel yang merupakan seorang admin di lab FTIS pada saat pengujian perangkat  
 15 lunak ini. Pengguna ini menguji di *workspace* Lab FTIS. Pada gambar 5.21a terdapat sebuah *event*  
 16 yang akan dimulai pada pukul 12.00 dan pada gambar 5.21b<sup>1</sup> belum terdapat adanya status yang  
 17 terpasang pada akun pengguna. Lalu pada gambar 5.22<sup>2</sup> dapat terlihat bahwa sudah ada status

<sup>1</sup>Gambar 5.21b di blur atas permintaan responden

<sup>2</sup>Gambar 5.22 di blur atas permintaan responden



Gambar 5.14: Tampilan *Slack* setelah *event* dimulai (Vincent).

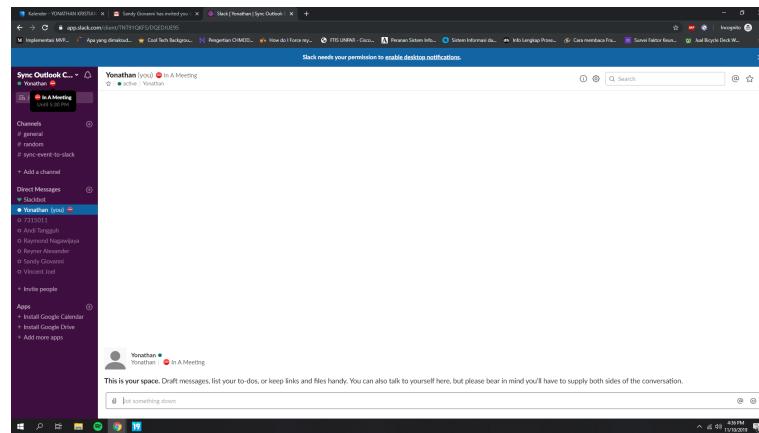


(a) Tampilan *event* yang dibuat pengguna (Yonathan). (b) Tampilan *Slack* sebelum *event* dimulai (Yonathan).

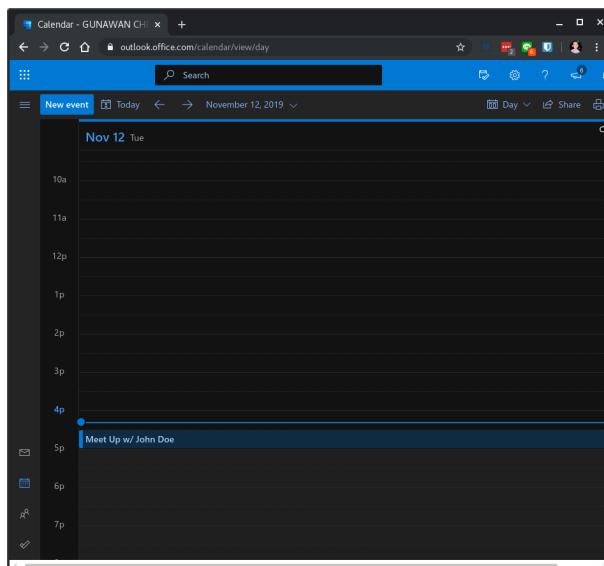
Gambar 5.15

- 1 yang terpasang dan akan berakhir pada pukul 12.30.
- 2 Dari gambar 5.23a sampai 5.24b merupakan hasil dari pengujian yang dilakukan oleh pengguna bernama Pascal yang mencoba perangkat lunak di *workspace* pndevworks. Pada gambar 5.23a terlihat ada *event* yang sedang berjalan yaitu dimulai dari pukul 09.00 sampai 10.00. Tetapi pengguna baru selesai menjalankan perangkat lunak yang dibangun dan berhasil memberikan izin pada pukul 09.13 yang bisa dilihat pada gambar 5.23b bahwa pada pukul 09.16 status masih belum berubah. Pada pukul 09.43 status sudah berubah seperti gambar 5.24a dan status sudah hilang pada pukul 10.03 seperti pada gambar 5.24b.
- 3 Dari gambar 5.25a sampai dengan gambar 5.26 merupakan hasil pengujian dari pengguna yang bernama Tedi. Pengguna ini mencoba di *workspace* yang bernama Dummy Workspace. Pada gambar 5.25a<sup>3</sup> terdapat *event* yang terdaftar yang dimulai pada pukul 15.00 dan pada gambar 5.25b terlihat waktu pada komputer menunjukkan pukul 14.28 dan belum terdapat status yang terpasang pada akun pengguna. Pada gambar 5.26 menunjukkan pada pukul 15.04 pada komputer, di akun pengguna sudah terpasang status yang statusnya akan berakhir pada pukul 16.00.
- 4 Pada gambar 5.27a sampai gambar 5.28 merupakan hasil yang didapatkan dari pengujian oleh pengguna yang bernama Yosua. Pengguna ini mencoba perangkat lunak ini di *workspace* Dummy Workspace. Pada gambar 5.27a terdapat *event* yang dimulai pada pukul 15.00 dan pada gambar 5.27b status pengguna masih kosong. Pada gambar 5.28 sudah terlihat ada status dan status itu

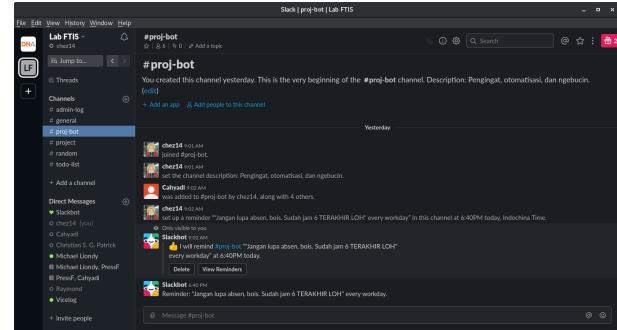
<sup>3</sup>Judul *event* yang ada di gambar 5.25a di blur atas permintaan responden



Gambar 5.16: Tampilan *Slack* setelah *event* dimulai (Yonathan).



(a) Tampilan *event* yang dibuat pengguna (Chris).



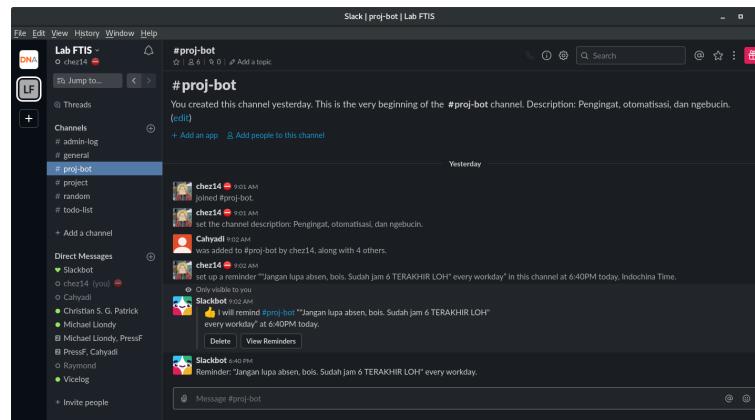
(b) Tampilan *Slack* sebelum *event* dimulai (Chris).

Gambar 5.17

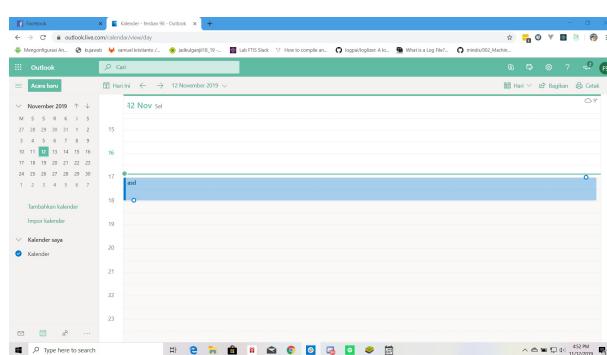
1 berakhir pada pukul 15.30.

## 2 Kesimpulan dari Pengujian

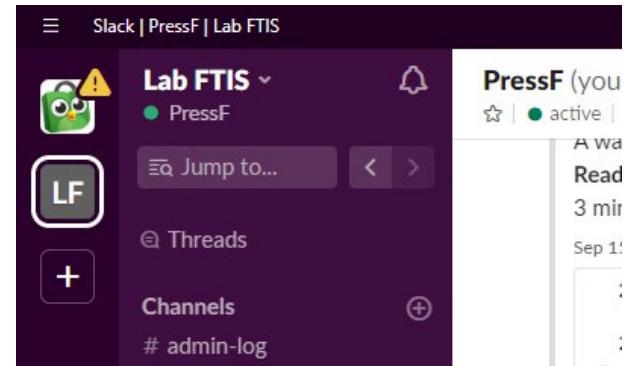
- 3 Pada pengujian yang sudah dilakukan oleh beberapa pengguna yang terlibat menyatakan bahwa perangkat lunak menjalankan fungsinya untuk mengganti status pada saat ada *event* dan menggantinya kembali jika *event* sudah berhasil dengan baik. Hanya saja terdapat perbedaan jeda saat mengubah status sehingga tidak tepat pada saat *event* dimulai maka status langsung berubah. Hal ini dikarenakan adanya proses yang dilakukan setiap kali memanggil fungsi yang dijalankan kurang lebih 2 menit sehingga mempengaruhi jadwal iterasi dari *Heroku Scheduler* yang selanjutnya.



Gambar 5.18: Tampilan *Slack* setelah *event* dimulai (Chris).

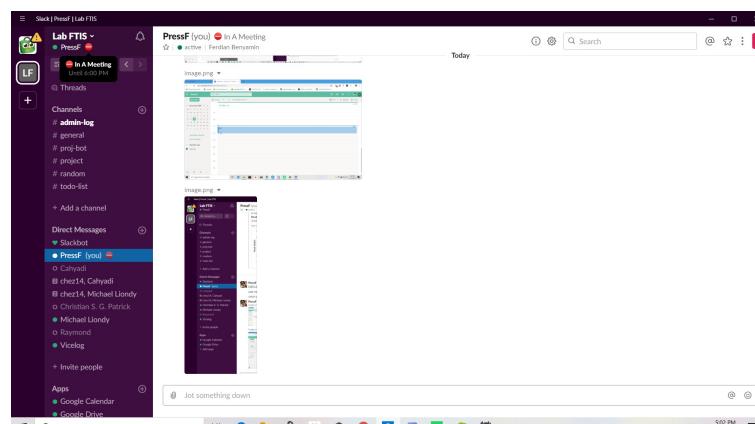


(a) Tampilan *event* yang dibuat pengguna (Ferdian).

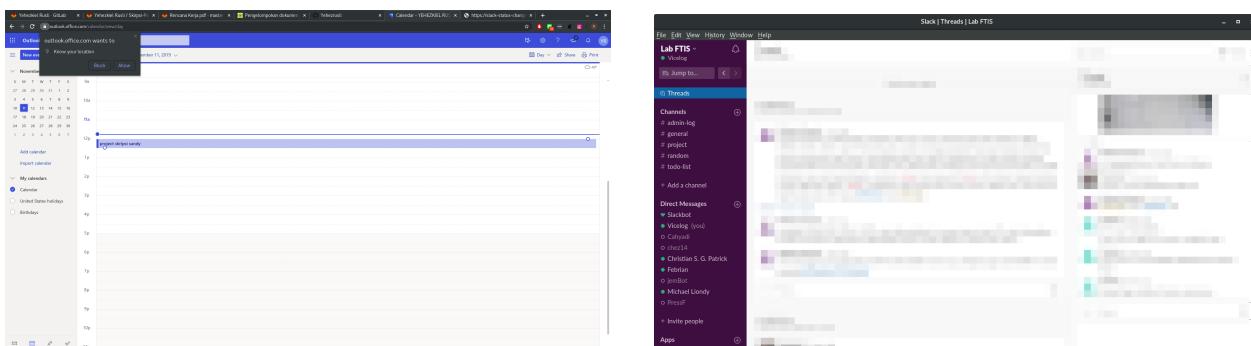


(b) Tampilan *Slack* sebelum *event* dimulai (Ferdian).

Gambar 5.19

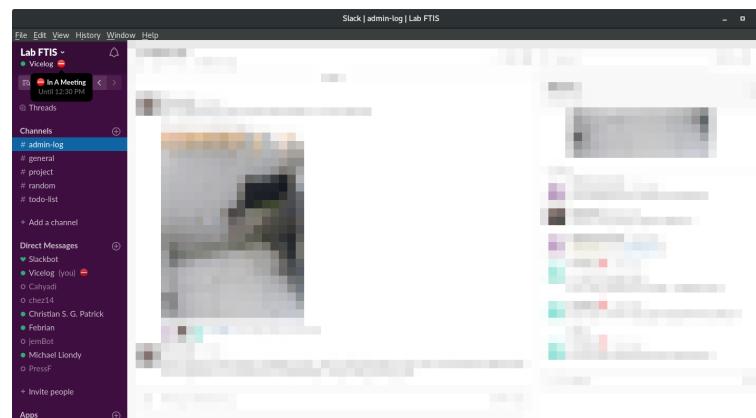


Gambar 5.20: Tampilan *Slack* setelah *event* dimulai (Ferdian).

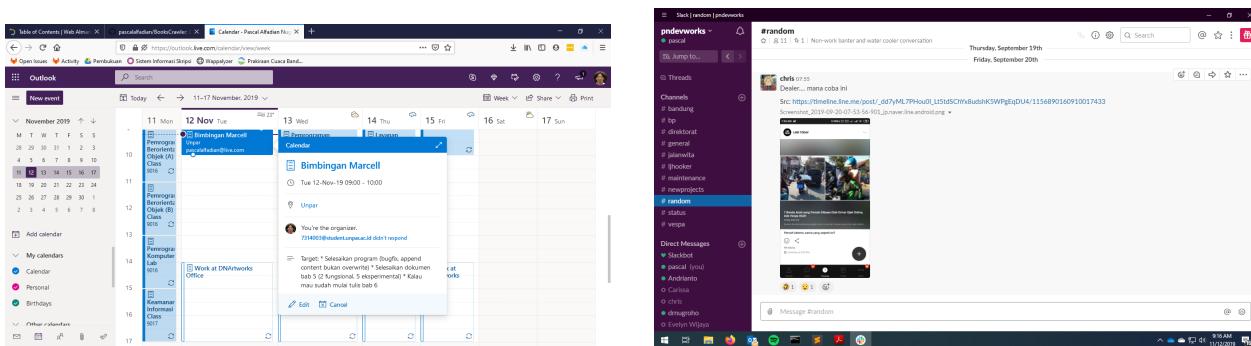


(a) Tampilan *event* yang dibuat pengguna (Yehezkiel). (b) Tampilan *Slack* sebelum *event* dimulai (Yehezkiel).

Gambar 5.21



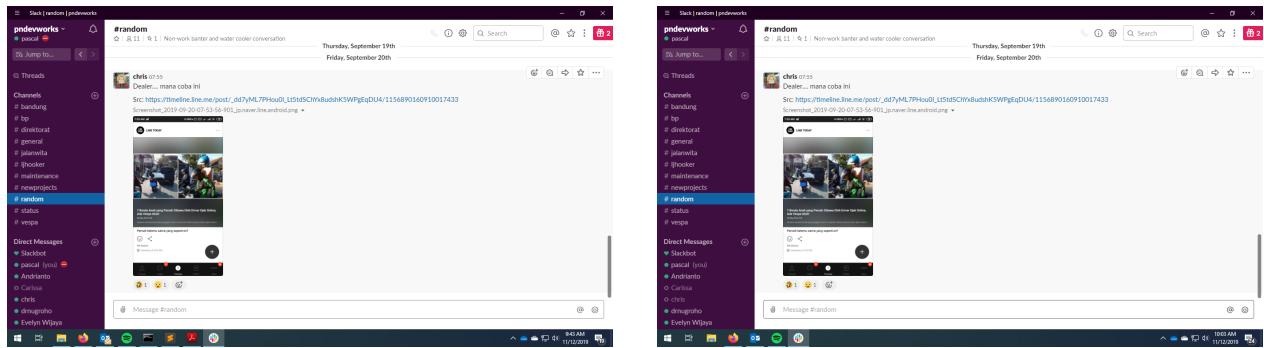
Gambar 5.22: Tampilan *Slack* setelah *event* dimulai (Yehezkiel).



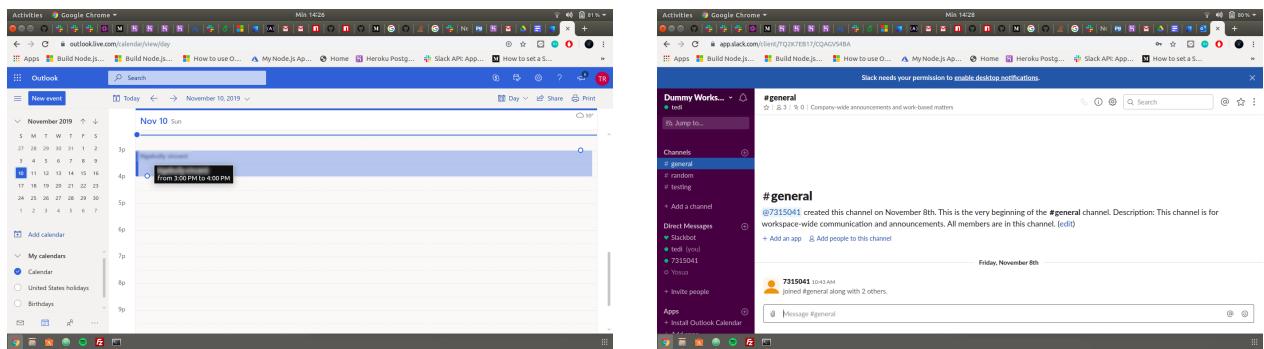
(a) Tampilan *event* yang dibuat pengguna (Pascal).

(b) Tampilan *Slack* setelah mengizinkan perangkat lunak kepada *workspace* *Slack* (Pascal).

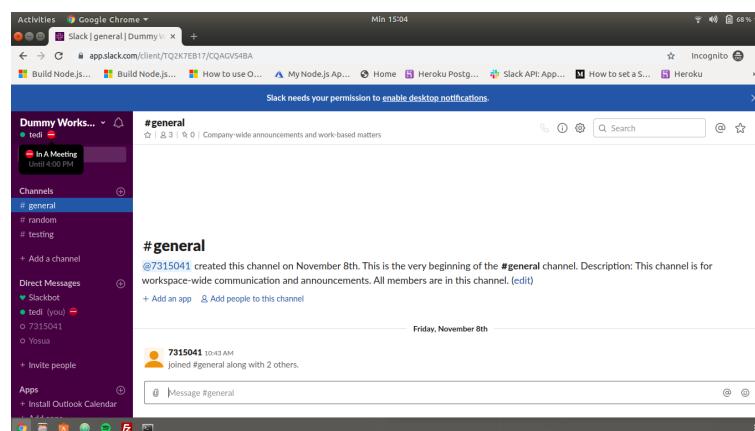
Gambar 5.23

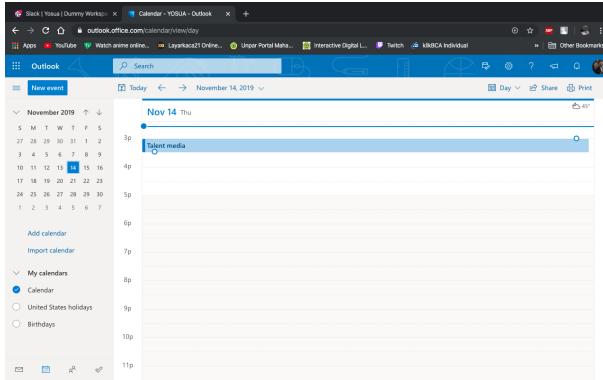
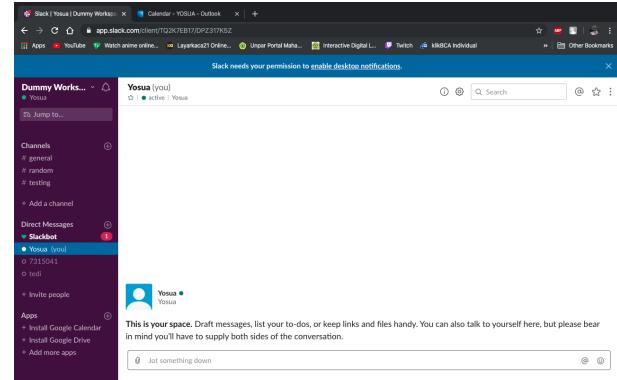
(a) Tampilan *Slack* setelah *event* dimulai (Pascal).(b) Tampilan *Slack* setelah *event* berakhir(Pascal).

Gambar 5.24

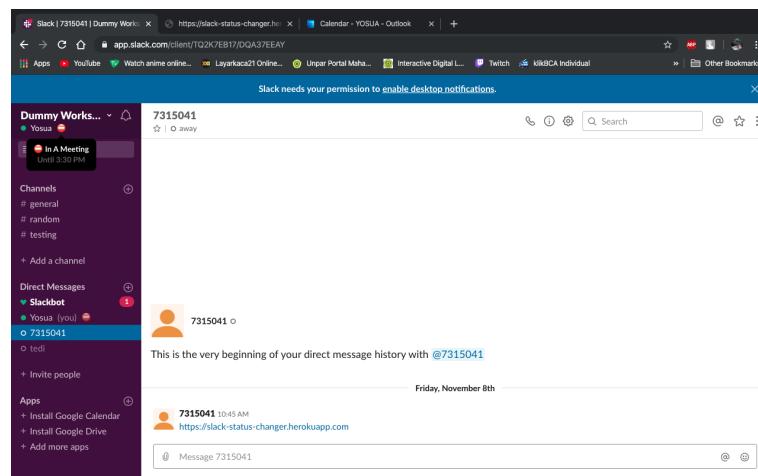
(a) Tampilan *event* yang dibuat pengguna (Tedi).(b) Tampilan *Slack* sebelum *event* dimulai (Tedi).

Gambar 5.25

Gambar 5.26: Tampilan *Slack* setelah *event* dimulai (Tedi).

(a) Tampilan *event* yang dibuat pengguna (Yosua).(b) Tampilan *Slack* sebelum *event* dimulai (Yosua).

Gambar 5.27

Gambar 5.28: Tampilan *Slack* setelah *event* dimulai (Yosua).

<sup>1</sup>

## BAB 6

<sup>2</sup>

### KESIMPULAN DAN SARAN

<sup>3</sup> Pada bab ini dibahas tentang kesimpulan dari hasil skripsi dan saran untuk penelitian selanjutnya.

#### <sup>4</sup> 6.1 Kesimpulan

<sup>5</sup> Ada beberapa kesimpulan yang dapat disimpulkan setelah melakukan penelitian ini, yaitu:

- <sup>6</sup> • Telah berhasil dibangun perangkat lunak yang berfungsi untuk mengubah status pada aplikasi *Slack* saat ada sebuah *event* yang terdaftar dan berjalan di aplikasi *Outlook Calendar* serta saat *event* itu selesai, maka status kembali hilang.
- <sup>9</sup> • *Workspace* apapun tetap bisa memakai perangkat lunak yang sudah dibangun asalkan pada saat mendaftarkan perangkat lunak ke aplikasi *Slack* mengaktifkan bagian “*Share Your Apps with Other Teams*” di bagian “*Manage Distribution*” agar perangkat lunak yang didaftarkan bisa digunakan untuk semua *workspace*.

#### <sup>13</sup> 6.2 Saran

<sup>14</sup> Ada beberapa saran terkait dengan penelitian ini untuk dikembangkan lebih lanjut, yaitu:

- <sup>15</sup> • Perangkat lunak bisa mensinkronisasikan status dengan *event* yang tercatat secara *real-time*.
- <sup>16</sup> • Perangkat lunak bisa mensinkronisasikan satu akun *Windows Live* ke lebih dari 1 *workspace* di *Slack*.



## **DAFTAR REFERENSI**

- [1] v1.0 (2019) *Microsoft Graph REST API*. Microsoft Corporation. Redmond, Washington, United States.
- [2] v1.0 (2015) *Slack API*. Slack Technologies. San Francisco, California, United States.
- [3] v10.15.3 (2009) *Node.js v10.15.3 Documentation*. Linux Foundation. San Francisco, California, United States.
- [4] v1.0 (2019) *Heroku Documentation*. Salesforce.com. San Francisco, California, United States.



# LAMPIRAN A

## KODE PROGRAM

Listing A.1: auth.js

```
1 const { Client } = require('pg');
2
3 const client = new Client({
4   connectionString: process.env.DATABASE_URL,
5   ssl: true,
6 });
7
8 client.connect();
9
10 const credentials = {
11   client: {
12     id: process.env.APP_ID,
13     secret: process.env.APP_PASSWORD,
14   },
15   auth: {
16     tokenHost: 'https://login.microsoftonline.com',
17     authorizePath: 'common/oauth2/v2.0/authorize',
18     tokenPath: 'common/oauth2/v2.0/token'
19   }
20 };
21
22 const credentialsSlack = {
23   client: {
24     id: process.env.SLACK_CLIENT_ID,
25     secret: process.env.SLACK_CLIENT_SECRET,
26   },
27   auth: {
28     tokenHost: 'https://slack.com',
29     authorizePath: 'oauth/authorize',
30     tokenPath: 'api/oauth.access'
31   }
32 };
33 const oauth2 = require('simple-oauth2').create(credentials);
34 const oauth2Slack = require('simple-oauth2').create(credentialsSlack);
35 const jwt = require('jsonwebtoken');
36 const databaseValue={};
37
38 //Microsoft Auth Helper
39 function getAuthUrl() {
40   const returnVal = oauth2.authorizationCode.authorizeURL({
41     redirect_uri: process.env.REDIRECT_URI,
42     scope: process.env.APP_SCOPES
43   });
44   return returnVal;
45 }
46
47 //Change Code to Token Microsoft
48 async function getTokenFromCode(auth_code) {
49   let result = await oauth2.authorizationCode.getToken({
50     code: auth_code,
51     redirect_uri: process.env.REDIRECT_URI,
52     scope: process.env.APP_SCOPES
53   });
54
55   const token = oauth2.accessToken.create(result);
56   const user = jwt.decode(token.token.id_token);
57   token.token.userData=user;
58
59   //token dari sini menampung Microsoft username, accessToken, refreshToken, dan expires masing2.
60   databaseValue.microsoft_username=token.token.userData.preferred_username;
61   databaseValue.microsoft_access_token.expires=token.token.expires_in;
62   databaseValue.microsoft_access_token=token.token.access_token;
63   databaseValue.microsoft_refresh_token=token.token.refresh_token;
64   databaseValue.login_timestamp=new Date().getTime();
65
66   return token.token.access_token;
67 }
68
69
70 //Slack Auth Helper
71 function getAuthUrlSlack() {
72   const returnVal = oauth2Slack.authorizationCode.authorizeURL({
73     client_id:process.env.SLACK_CLIENT_ID,
74     redirect_uri: process.env.SLACK_REDIRECT_URI,
```

```

76     scope: process.env.SLACK_APP_SCOPES
77   });
78   return retVal;
79 }
80
81 //Change Code to Token Slack
82 async function getTokenFromCodeSlack(auth_code) {
83   let result = await oauth2Slack.authorizationCode.getToken({
84     code: auth_code,
85     redirect_uri: process.env.SLACK_REDIRECT_URI,
86     client_id:process.env.SLACK_CLIENT_ID,
87     client_secret: process.env.SLACK_CLIENT_SECRET
88   });
89
90   const token = oauth2Slack.accessToken.create(result);
91   databaseValue.slack_access_token=token.token.access_token;
92
93   client.query('SELECT * FROM public."Credentials"', (err, res) => {
94     const arrResult=res.rows;
95     var valueForInsert=[databaseValue.microsoft_username, databaseValue.microsoft_refresh_token, databaseValue.
96       microsoft_access_token_expires, databaseValue.microsoft_access_token, databaseValue.slack_access_token, databaseValue.
97       login_timestamp];
98     var updated=0;
99     var queryText='';
100
101    arrResult.forEach(row =>{
102      if(row.microsoft_username==databaseValue.microsoft_username){
103        //Jika sudah ada record untuk username microsoft ini, maka perangkat lunak hanya melakukan update.
104        queryText='UPDATE public."Credentials" SET microsoft_refresh_token=$2,microsoft_access_token_expires=$3,
105          microsoft_access_token=$4,slack_access_token=$5,login_timestamp=$6 WHERE microsoft_username=$1 RETURNING *';
106
107        client.query(queryText, valueForInsert, (err, res) => {
108          if (err) {
109            console.log(err.stack)
110          } else {
111            console.log(res.rows)
112          }
113        });
114        //Flag update menjadi 1.
115        updated=1;
116      }
117    })
118
119    //Jika belum ada, maka memasukkan data ke dalam database dengan bantuan membaca flag.
120    if (updated!=1) {
121      queryText='INSERT INTO public."Credentials" (microsoft_username,microsoft_refresh_token,microsoft_access_token_expires,
122        microsoft_access_token,slack_access_token,login_timestamp) VALUES ($1,$2,$3,$4,$5,$6)';
123
124      client.query(queryText, valueForInsert, (err, res) => {
125        if (err) {
126          console.log(err.stack)
127        } else {
128          console.log(res.rows[0])
129        }
130      });
131    }
132
133    return token.token.access_token;
134  }
135
136 exports.getTokenFromCode = getTokenFromCode;
137 exports.getAuthUrl = getAuthUrl;
138
139 exports.getTokenFromCodeSlack = getTokenFromCodeSlack;
140 exports.getAuthUrlSlack = getAuthUrlSlack;

```

Listing A.2: authorize.js

```

1 var express = require('express');
2 var router = express.Router();
3 var authHelper = require('../helper/auth');
4 var fs=require('fs');
5
6 /* GET /authorize. */
7 router.get('/', async function(req, res, next) {
8   // Get auth code
9   const code = req.query.code;
10
11   token = await authHelper.getTokenFromCode(code);
12   let parms = { title: 'Slack_Login', active: { home: true } };
13
14   parms.signInUrlSlack = authHelper.getAuthUrlSlack();
15   res.render('authorize_success', parms);
16
17 });
18
19 module.exports = router;

```

Listing A.3: index.js

```

1 var express = require('express');
2 var router = express.Router();
3 var authHelper = require('../helper/auth');
4
5 /* GET home page. */

```

```

6 router.get('/', function(req, res, next) {
7   let parms = { title: 'Home', active: { home: true } };
8
9   parms.signInUrl = authHelper.getAuthUrl();
10  parms.debug = parms.signInUrl;
11  res.render('index', parms);
12 });
13
14 module.exports = router;

```

Listing A.4: slackAuthorize.js

```

1 var express = require('express');
2 var router = express.Router();
3 var authHelper = require('../helper/auth');
4
5 /* GET /authorize. */
6 router.get('/', async function(req, res, next) {
7   // Get auth code
8   const code = req.query.code;
9
10  token = await authHelper.getTokenFromCodeSlack(code);
11  res.render('slack_authorize_success');
12 });
13
14 module.exports = router;

```

Listing A.5: statusChanger.js

```

1 var express = require('express');
2 var router = express.Router();
3 var fs =require('fs');
4 var authHelper = require('../helper/auth');
5 var graph = require('@microsoft/microsoft-graph-client');
6 const jwt = require('jsonwebtoken');
7 const { WebClient } = require('@slack/web-api');
8 require('dotenv').config();
9 require('isomorphic-fetch');
10
11 const credentials = {
12   client: {
13     id: process.env.APP_ID,
14     secret: process.env.APP_PASSWORD,
15   },
16   auth: {
17     tokenHost: 'https://login.microsoftonline.com',
18     authorizePath: 'common/oauth2/v2.0/authorize',
19     tokenPath: 'common/oauth2/v2.0/token'
20   }
21 };
22 const oauth2 = require('simple-oauth2').create(credentials);
23 const { Client } = require('pg');
24
25 const client = new Client({
26   connectionString: process.env.DATABASE_URL,
27   ssl: true,
28 });
29 var timestampNow;
30
31 client.connect();
32
33 /* GET home page. */
34 router.get('/', async function(req, res, next) {
35   timestampNow=new Date();
36   client.query('SELECT * FROM public."Credentials";', (err, res) => {
37     //Melakukan Looping untuk mengiterasi setiap data yang dikembalikan dari database
38     //Ganti index ke 0 [0] dengan hasil iterasi dari hasil dari db
39     for (var i = 0; i < res.rows.length; i++) {
40       if(res.rows[i].microsoft_access_token){
41         var expiration = parseInt(res.rows[i].login_timestamp)+res.rows[i].microsoft_access_token_expires;
42         var now=new Date().getTime();
43         if (expiration<now) {
44           var newAccessToken=useRefreshToken(res.rows[i].microsoft_refresh_token)
45           var events=getEvent(newAccessToken, res.rows[i].slack_access_token);
46         }
47         else {
48           var events=getEvent(res.rows[i].microsoft_access_token);
49         }
50       }
51     }
52   });
53
54   res.render('calendar_success');
55 });
56
57
58 // Fungsi ini berguna untuk mengambil data event dari Outlook Calendar.
59 async function getEvent(accessToken, slack_access_token){
60
61   const graphClient = graph.Client.init({
62     authProvider:(done)=>{
63       done(null, accessToken);
64     }
65   });

```

```

67 try {
68   // Get event from calendar
69   const result = await graphClient
70     .api('/me/events')
71     .select('subject,start,end')
72     .orderby('start/dateTime_DESC')
73     .get();
74
75   for (var i = 0; i < result.value.length; i++) {
76     var start = result.value[i].start.dateTime;
77     var startDate = new Date(start);
78     var end = result.value[i].end.dateTime;
79     var endDate = new Date(end);
80     if (timestampNow>=startDate&&timestampNow<=endDate) {
81       //Memanggil fungsi untuk merubah status.
82       changeStatusSlack(slack_access_token, endDate.getTime());
83     }
84     else {
85       console.log("Tidak ada event yang bersamaan dengan waktu skrng");
86     }
87   }
88
89 }catch (err) {
90   console.log("error", err);
91 }
92 }
93
94
95 // Fungsi ini berguna untuk meminta access token yang baru dengan menggunakan refresh token.
96 async function useRefreshToken(auth_code) {
97   try{
98     let newToken=await oauth2.accessToken.create({refresh_token: auth_code}).refresh();
99
100    const user = jwt.decode(newToken.token.id_token);
101    const databaseValue={};
102    newToken.token.userData=user;
103    databaseValue.microsoft_username=newToken.token.userData.preferred_username;
104    databaseValue.microsoft_access_token_expires=newToken.token.expires_in;
105    databaseValue.microsoft_access_token=newToken.token.access_token;
106    databaseValue.microsoft_refresh_token=newToken.token.refresh_token;
107    databaseValue.login_timestamp=new Date().getTime();
108
109    var queryText='UPDATE_public."Credentials" _SET_microsoft_refresh_token=$2,_microsoft_access_token_expires=$3,_microsoft_access_token=$4,_login_timestamp=$5_WHERE_microsoft_username=$1_RETURNING_*';
110    var valueForInsert=[databaseValue.microsoft_username, databaseValue.microsoft_refresh_token, databaseValue.microsoft_access_token_expires, databaseValue.microsoft_access_token, databaseValue.login_timestamp];
111
112    client.query(queryText, valueForInsert, (err, res) => {
113      if (err) {
114        console.log(err.stack)
115      } else {
116      }
117    });
118    return newToken.token.access_token;
119  }
120  catch(err){
121    console.log(err);
122  }
123 }
124
125
126 // Fungsi ini berguna untuk mengganti status di slack
127 async function changeStatusSlack(slack_access_token, endDate){
128   const web = new WebClient(slack_access_token);
129
130   const result = await web.users.profile.set({
131     "profile": {
132       "status_text": "In_A_Meeting",
133       "status_emoji": ":no_entry:",
134       "status_expiration":endDate/1000
135     }
136   });
137 }
138
139 module.exports = router;
140

```

Listing A.6: app.js

```

1 var express = require('express');
2 var path = require('path');
3
4 require('dotenv').config();
5 var indexRouter = require('./routes/index');
6 var authorize = require('./routes/authorize');
7 var statusChanger = require('./routes/statusChanger');
8 var slackAuthorize = require('./routes/slackAuthorize');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view_engine', 'hbs');
15
16 app.use(express.json());
17 app.use(express.urlencoded({ extended: false }));
18 app.use(express.static(path.join(__dirname, 'public')));
19

```

```

20 app.use('/', indexRouter);
21 app.use('/authorize', authorize);
22 app.use('/statusChanger', statusChanger);
23 app.use('/slackAuthorize', slackAuthorize);
24
25 // error handler
26 app.use(function(err, req, res, next) {
27   // set locals, only providing error in development
28   res.locals.message = err.message;
29   res.locals.error = req.app.get('env') === 'development' ? err : {};
30
31   // render the error page
32   res.status(err.status || 500);
33   res.render('error');
34 });
35
36 module.exports = app;

```

Listing A.7: authorize\_success.hbs

```

1 <div style="text-align:center;">
2   <h1 style="text-align:center;">Sukses memberikan akses Outlook Calendar kepada program.</h1>
3   <p style="text-align:center;">Akses ke Outlook Calendar sudah berhasil diberikan untuk program ini. Langkah selanjutnya adalah
4     dengan memberikan akses program ini kepada <strong>Slack</strong> dengan cara
5     menekan hyperlink untuk login di bawah ini.</p>
6   <br>
7   <a class="btn btn-primary btn-large" href="{{signInUrlSlack}}" style="border:_1px_solid_grey;_padding:_15px;">Klik disini untuk
    melakukan login dan memberikan akses ke Slack</a>
7 </div>

```

Listing A.8: calendar\_success.hbs

```
1 <span>Your status is processing to change. </span>
```

Listing A.9: index.hbs

```

1 <div style="text-align:center;">
2   <h1 style="text-align:center;">Slack Status Changer</h1>
3   <p style="text-align:center;">Program ini berfungsi untuk mengubah status yang terdapat pada aplikasi <strong>Slack</strong>
    sesuai dengan jadwal yang terdapat pada <strong>Outlook Calendar</strong>.
4   Jadi, status pada Slack akan berubah jika ada event yang sudah terdaftarkan pada Outlook Calendar dari pengguna. Untuk bisa
    menggunakan program ini, maka dibutuhkan
5   akses kepada masing-masing aplikasinya dengan melakukan login terlebih dahulu. Untuk langkah pertama, lakukan login kepada
    Outlook melalui Windows Live dengan cara menekan hyperlink di bawah ini. </p>
6   <br>
7
8   <a class="btn btn-primary btn-large" href="{{signInUrl}}" style="border:_1px_solid_grey;_padding:_15px;">Klik disini untuk
    melakukan login dan memberikan akses ke Windows Live.</a>
9 </div>

```

Listing A.10: slack\_authorize\_success.hbs

```

1 <div style="text-align:center;">
2   <h1 style="text-align:center;">Sukses memberikan akses Slack kepada program.</h1>
3   <p style="text-align:center;">Akses ke Slack sudah berhasil diberikan untuk program ini. Dengan akses yang didapat, maka program
    ini akan mengubah status saat ada event yang
4     terdaftar di Outlook Calendar.</p>
5 </div>

```