

SKRIPSI

APLIKASI PENGINTEGRASI 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 APP



Sandy Giovanni Sutiansen

NPM: 2015730041

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

LEMBAR PENGESAHAN

APLIKASI PENGINTEGRASI OUTLOOK CALENDAR DAN SLACK

Sandy Giovanni Sutiansen

NPM: 2015730041

Bandung, 13 Desember 2019

Menyetujui,

Pembimbing

Pascal Alfadian, M.Comp.

Ketua Tim Penguji

Anggota Tim Penguji

Luciana Abednego, M.T.

Raymond Chandra, M.T.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

APLIKASI PENGINTEGRASI OUTLOOK CALENDAR DAN SLACK

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 13 Desember 2019

Meterai Rp. 6000

Sandy Giovanni Sutiansen
NPM: 2015730041

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 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 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.

Perangkat lunak ini akan mengambil data *event* yang tersimpan di dalam *Outlook Calendar*, lalu akan diperiksa dengan waktu sekarang. Jika waktu *event* sedang berjalan sekarang, maka perangkat lunak akan mengubah status yang terdapat pada aplikasi *Slack* menjadi “*In a meeting*” dan status akan kembali lagi kosong jika *event* sudah selesai berjalan.

Pengujian pada skripsi ini dilakukan melalui 2 lingkungan yaitu perangkat lunak diuji coba di *workspace* tempat perangkat lunak ini didaftarkan pada aplikasi *Slack* dan diuji coba di *workspace* lain. Hasil pengujian menunjukkan fitur yang diharapkan telah berjalan dengan sesuai harapan, baik itu di dalam *workspace* tempat perangkat lunak didaftarkan, maupun di *workspace* lain. Perangkat lunak ini dapat mengubah status pada aplikasi *Slack* di *workspace* manapun oleh pengguna siapapun dengan catatan setiap pengguna hanya bisa mendaftarkan 1 akun *Slack* di 1 *workspace*.

Kata-kata kunci: *Outlook Calendar, Slack, 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.

This software will retrieve event data stored in Outlook Calendar, then it will be checked with the current time. If the time the event is running now, the software will change the status contained in the Slack application to " In a meeting " and the status will return to empty if the event has finished.

Testing in this thesis is done through 2 environments namely software tested at workspaces where this software is registered in the Slack application and tested on other workspaces. The test results show that features are expected to run as expected, both in the workspace where the software is registered, and in other workspaces. This software can change the status of the Slack application in any workspace by any user as long as each user can only register 1 slack account in 1 workspace.

Keywords: Outlook Calendar, Slack, Event, Status

*Untuk diri sendiri, keluarga, teman-teman seperjuangan, dan
semua yang telah mendukung*

KATA PENGANTAR

Puji Syukur penulis panjatkan kepada Tuhan Yesus karena atas berkat-Nya lah skripsi dengan judul “Aplikasi Pengintegrasi Outlook Calendar dan Slack” ini bisa selesai dengan baik dan tepat waktu. Penulis berharap skripsi dan perangkat lunak yang dibangun dapat berguna bagi orang yang membutuhkan dan juga dapat membantu bagi orang yang akan melanjutkan penelitian ini untuk selanjutnya. Tidak lupa juga penulis mengucapkan terima kasih kepada:

- Keluarga, khususnya orang tua dan juga kepada adik yang telah menyemangati penulis secara langsung maupun tidak langsung, serta membuat suasana senyaman mungkin.
- Pak Pascal Alfadian yang telah membantu membimbing penulis dalam proses pembuatan skripsi beserta perangkat lunaknya.
- Teman yang selalu mendukung: Andi Tangguh Kippi Nusantara, Reyner Alexander, Adriswara Dwikarkasa, Vincent Joel Sinatra, Yonathan Kristian, Yosua, Raymond Nagawijaya, Arlin Sasqia Puspa Shiffa.
- Admin Lab Ftis yang bersedia ikutserta dalam melakukan pengujian perangkat lunak ini.
- Rekan-rekan kerja di Talent Media yang menyemangati dan juga ikutserta dalam melakukan pengujian perangkat lunak.
- Pihak-pihak lain yang tidak bisa disebutkan satu persatu.

Bandung, Desember 2019

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 <i>Microsoft Graph API</i>	5
2.1.1 User resource type	6
2.1.2 Event resource type	14
2.1.3 Lain-lain	18
2.2 <i>Slack API</i>	19
2.3 <i>Node.js</i>	22
2.3.1 <i>HTTP</i>	22
2.3.2 <i>Express.js</i>	24
2.3.3 <i>Handlebars</i>	25
2.3.4 <i>Simple OAuth2</i>	25
2.3.5 <i>jsonwebtoken</i>	27
2.3.6 <i>Isomorphic Fetch</i>	28
2.3.7 <i>Microsoft Graph Client Library</i>	28
2.3.8 <i>Slack Web API</i>	28
2.4 <i>Heroku</i>	29
2.4.1 <i>Dyno</i>	29
2.4.2 <i>Heroku Scheduler</i>	30
2.4.3 <i>Heroku Postgres</i>	30
3 ANALISIS	33
3.1 Analisis Perangkat Lunak	33
3.1.1 Analisis Diagram Alir Sistem	33
3.1.2 Analisis <i>Microsoft Graph API</i>	34
3.1.3 Analisis <i>Slack</i>	44
3.1.4 Analisis <i>Heroku</i>	45
3.1.5 Analisis <i>Data Flow Diagram</i>	47

3.2 Analisis Kasus	48
4 PERANCANGAN	49
4.1 Perancangan Routing Handler	49
4.1.1 <i>Route get /</i>	49
4.1.2 <i>Route get /authorize</i>	50
4.1.3 <i>Route get /slackAuthorize</i>	51
4.1.4 <i>Route get /statusChanger</i>	51
4.2 Perancangan <i>Helper</i>	52
4.3 Perancangan Basis Data	54
5 IMPLEMENTASI DAN PENGUJIAN	57
5.1 Implementasi	57
5.1.1 Lingkungan Pengembangan	57
5.1.2 Implementasi Basis Data	58
5.1.3 Implementasi Antarmuka	58
5.2 Pengujian	61
5.2.1 Pengujian Fungsional	61
5.2.2 Pengujian Eksperimental	63
6 KESIMPULAN DAN SARAN	73
6.1 Kesimpulan	73
6.2 Saran	73
DAFTAR REFERENSI	75
A KODE PROGRAM	77

DAFTAR GAMBAR

1.1	<i>List</i> status yang disediakan sebagai pilihan di <i>Slack</i>	2
2.1	Tampilan direktori yang dibuat jika menjalankan express generator.	24
3.1	Diagram alir sistem pada bagian perangkat lunak yang berinteraksi dengan <i>user</i> . .	34
3.2	Diagram alir sistem pada bagian perangkat lunak yang dijalankan secara berkala. .	35
3.3	Data kredensial yang diberikan oleh <i>heroku postgres</i>	46
3.4	<i>Data Flow Diagram level 0</i>	47
3.5	<i>Data Flow Diagram</i>	47
4.1	Alur antar <i>route</i>	49
4.2	Rancangan antarmuka halaman awal	50
4.3	Rancangan antarmuka halaman setelah <i>login Windows Live</i>	50
4.4	Rancangan antarmuka halaman setelah <i>login Slack</i>	51
4.5	ER Diagram untuk tabel Credentials.	54
5.1	Antarmuka halaman awal	58
5.2	Antarmuka untuk login Windows Live.	58
5.3	Antarmuka untuk memberikan Windows Live izin ke perangkat lunak.	59
5.4	Antarmuka petunjuk login ke Slack.	59
5.5	Antarmuka untuk login Slack.	59
5.6	Antarmuka untuk memberikan Slack izin ke perangkat lunak.	60
5.7	Antarmuka saat selesai melakukan login dan pemberian izin.	60
5.8	62
5.9	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Raymond).	62
5.10	63
5.11	63
5.12	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Sandy).	64
5.13	64
5.14	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Vincent).	65
5.15	65
5.16	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Yonathan).	66
5.17	66
5.18	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Chris).	66
5.19	67
5.20	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Ferdian).	67
5.21	67
5.22	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Yehezkiel).	68
5.23	68
5.24	68
5.25	69
5.26	Tampilan <i>Slack</i> setelah <i>event dimulai</i> (Tedi).	69
5.27	69

DAFTAR TABEL

2.1	Tabel contoh <i>header</i> pesan <i>HTTP</i>	23
3.1	Tabel <i>parameter Authorization Code</i>	36
3.2	Tabel contoh <i>request Authorization Code</i>	36
3.3	Tabel contoh <i>response Authorization Code</i>	36
3.4	Tabel <i>parameter response Authorization Code</i>	37
3.5	Tabel contoh <i>request Access Token</i>	37
3.6	Tabel <i>parameter request Access Token</i>	38
3.7	Tabel contoh <i>response Access Token</i>	38
3.8	Tabel <i>parameter response Access Token</i>	39
3.9	Tabel contoh <i>request call Microsoft Graph</i>	39
3.10	Tabel contoh <i>response call Microsoft Graph</i>	40
3.11	Tabel contoh <i>request</i> menggunakan <i>Refresh Token</i>	41
3.12	Tabel <i>parameter request Refresh Token</i>	41
3.13	Tabel contoh <i>response</i> menggunakan <i>Refresh Token</i>	41
3.14	Tabel <i>parameter response Refresh Token</i>	42
3.15	Tabel contoh <i>header time zone</i>	42
3.16	Tabel <i>parameter header time zone</i>	42
3.17	Tabel contoh <i>request event</i>	43
3.18	Tabel contoh <i>response event</i>	43

BAB 1

PENDAHULUAN

1.1 Latar Belakang

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

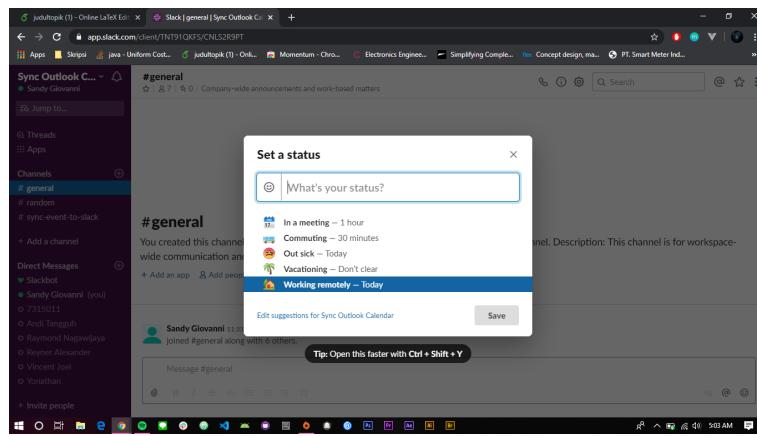
*Slack*² adalah alat dan layanan kolaborasi tim berbasis *cloud*. *Slack* merupakan singkatan dari “*Searchable Log of All Conversation and Knowledge*”. Cara melakukan kolaborasi di aplikasi *Slack* sendiri adalah dengan cara komunitas, grup, atau tim bergabung ke dalam URL yang spesifik yang disebut dengan *workspace*. *Room chat* yang terdapat di dalam *workspace* biasa disebut dengan *Channel*. Ada 2 jenis *channel* di dalam *workspace* yang ada di aplikasi *Slack* yaitu *Public Channel* dan *Private Channel*. Pada *Public Channel*, seluruh anggota dari tim atau komunitas bisa masuk dan bergabung untuk berkomunikasi di *channel* tersebut. Tetapi pada *Private Channel*, hanya anggota yang diizinkan, ditambahkan, dan diundang oleh admin atau pembuat *channel* sajalah yang bisa ikut berkomunikasi di dalam *channel* tersebut. Pada setiap *workspace* dapat ditambahkan dan dipasangkan aplikasi penyokong lainnya seperti contohnya adalah *Trello*, *Github*, *Google Drive*, dan banyak aplikasi lainnya yang bisa digunakan untuk membantu para partisipan di dalam *workspace* tersebut.

Pada *Slack* terdapat status pengguna yang bisa diganti oleh pengguna tersebut untuk menggambarkan keadaan pengguna saat ini. Nilai *default* dari status ini adalah tersedia, tetapi *Slack* sudah menyediakan beberapa pilihan status seperti “*In a meeting*”, “*Commuting*”, “*Out sick*”, “*Vacationing*”, dan “*Working remotely*” seperti yang terdapat pada contoh gambar 1.1. Selain itu, *Slack* juga menyediakan pilihan untuk pengguna yang mengisi statusnya sendiri. Dengan adanya fitur status di dalam aplikasi *Slack*, maka hal itu membantu mendeskripsikan keadaan yang sedang dialami oleh pengguna saat itu. Status hanya berupa keterangan di akun pengguna dan tidak berdampak kepada fitur lainnya seperti contohnya fitur obrolan. Yang menjadi latar belakang dirancangnya perangkat lunak ini yaitu terkadang pengguna lupa untuk mengganti status menjadi “*In a meeting*” saat pengguna memiliki jadwal untuk melakukan *meeting*, sehingga status di pengguna masih terlihat tersedia oleh *user* lain yang membuat tidak mengetahui pengguna sedang dalam keadaan *meeting*. Di saat seperti ini, kemungkinan untuk *meeting* terganggu oleh adanya *chat* yang masuk lewat *Slack* pun cukup tinggi. Tetapi status hanya membantu mendeskripsikan keadaan yang sedang dialami oleh pengguna saat itu dan tidak memblokir chat masuk kepada pengguna.

Pada skripsi ini dibuat perangkat lunak yang membaca jadwal dari pengguna yang dicantumkan di aplikasi *Outlook.com Calendar*, lalu akan di integrasikan ke aplikasi *Slack* dengan mengubah status di dalam aplikasi *Slack* sesuai dengan jadwal yang telah didapatkan dari data di *Outlook.com*

¹<https://outlook.live.com/>

²<https://slack.com/>



Gambar 1.1: *List* status yang disediakan sebagai pilihan di *Slack*.

Calendar dari pengguna.

Perangkat lunak ini dibuat menggunakan *Node.js*³ dan memiliki 2 fungsi utama yaitu yang pertama adalah membaca dan mencatat jadwal dari *Outlook.com Calendar* yang membutuhkan adanya *Outlook.com Calendar API*. Lalu fungsi kedua yaitu mengubah status ke aplikasi *Slack* dengan menggunakan *Slack API*. Kedua fungsi dari perangkat lunak ini dijalankan secara berkala sehingga perubahan status yang terjadi pada aplikasi *Slack* tidak terjadi secara *real-time* tetapi berkala setiap 10 menit sekali dilakukan penjalanan dari fungsi perangkat lunak ini.

API atau *Application Programming Interface* adalah sebuah *interface* yang memungkinkan antara 2 aplikasi atau lebih saling berinteraksi. *Outlook Calendar* sendiri memiliki *API* yang berpusat pada *Microsoft Graph API*, lalu pada aplikasi *Slack* juga sudah disediakan *API* sehingga untuk mengkomunikasikan antara kedua aplikasi ini bisa menggunakan *API* dari masing-masing aplikasi yang sudah disediakan.

1.2 Rumusan Masalah

Pada perangkat lunak ini, terdapat rumusan masalah sebagai berikut:

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

1.3 Tujuan

Adapun pada perangkat lunak ini memiliki tujuan sebagai berikut:

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

³<https://nodejs.org>

1.4 Batasan Masalah

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

1. Perangkat lunak ini dijalankan secara berkala sehingga tidak dapat mengubah status secara *real-time*.
2. Hanya berlaku untuk setiap satu pengguna terhubung dengan satu *workspace* saja di dalam *Slack* (Tidak berlaku satu pengguna ke banyak *workspace*).

1.5 Metodologi

Berikut adalah metodologi yang akan digunakan dalam penelitian ini:

1. Melakukan studi literatur tentang *Outlook.com Calendar*, *Slack*, dan juga *Node.js*.
2. Menggunakan aplikasi *Slack* di lingkungan tempat penulis melakukan magang.
3. Melakukan analisis cara melakukan *synchronize* dengan aplikasi *Outlook.com Calendar* secara berkala.
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. 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.
6. Penulisan dokumen.

1.6 Sistematika Pembahasan

Setiap bab dalam penelitian ini akan memiliki sistematika pembahasan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan, yaitu menjelaskan gambaran umum dari penelitian ini yang berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
2. Bab 2: Dasar Teori, yaitu menjelaskan dan membahas teori yang dibutuhkan untuk melakukan penelitian ini. Meliputi tentang *Outlook.com Calendar API*, *Slack API*, dan *Node.js*.
3. Bab 3: Analisis, yaitu membahas mengenai analisis masalah. Berisi tentang analisis cara pengambilan data dari *Outlook.com Calendar*, dan proses pengubahan status menggunakan program pada *Slack*.
4. Bab 4: Perancangan, yaitu membahas mengenai perancangan aplikasi untuk melakukan sinkronisasi antara *Outlook.com Calendar* dengan *Slack*.
5. Bab 5: Implementasi dan Pengujian, yaitu membahas mengenai implementasi dari perangkat lunak yang telah dirancang dan juga pengujian perangkat lunak tersebut.
6. Bab 6: Kesimpulan dan Saran, yaitu berisi tentang kesimpulan dari penelitian ini dan juga saran yang dapat diberikan untuk penelitian selanjutnya.

BAB 2

LANDASAN TEORI

Pada bab ini akan dibahas mengenai dasar-dasar teori yang dipakai dalam skripsi ini. Teori di dalam bab ini dimulai dengan *Outlook Calendar* yang merupakan suatu aplikasi yang dipakai untuk menyimpan data *event*, lalu di dalam *Outlook Calendar* akan dibahas mengenai *API* yang telah disediakan oleh *Windows Live*. Lalu setelah membahas mengenai *Outlook Calendar*, dilanjutkan dengan membahas mengenai *Slack* beserta *API* yang telah disediakan oleh *Slack*. Dilanjutkan dengan membahas mengenai *Node.js* yang merupakan sebuah *platform* yang digunakan untuk membangun perangkat lunak skripsi ini, serta akan membahas mengenai *library-library* yang dipakai untuk membangun skripsi ini. Setelah itu akan membahas mengenai *Heroku* yang merupakan sebuah *cloud platform* yang memiliki fungsi seperti layaknya *server* yang digunakan untuk melakukan *hosting* dari berbagai perangkat lunak.

Outlook Calendar

Outlook calendar adalah aplikasi yang tergabung dalam layanan dari *Microsoft* yang memiliki fungsi sebagai kalender. Layanan ini bisa terhubung dengan aplikasi lainnya dengan menggunakan *API* yang telah dibuat dan disediakan oleh pihak *Microsoft*, yaitu yang terkumpul di dalam *Microsoft Graph API*. Maksud dari *API* adalah *interface* yang memungkinkan 2 aplikasi atau lebih untuk saling berinteraksi. Umumnya cara kerja dari *API* adalah dengan mengirimkan *request* bertipe *get* atau *post* kepada sebuah *endpoint* tertentu dengan *parameter* yang dibutuhkan oleh *API* tersebut. *Endpoint* adalah alamat tujuan yang pada umumnya sudah disediakan oleh penyedia *API*, sedangkan *parameter* merupakan masukan yang diperlukan oleh fungsi *API* itu sendiri. Pembahasan tentang *Microsoft Graph API* akan dibahas pada bagian subbab 2.1

2.1 Microsoft Graph API

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

Untuk menggunakan fungsi dari *Microsoft Graph API*, dibutuhkan untuk mendaftarkan aplikasi yang akan dirancang dan memakai fungsi dari *API* dari *Microsoft Graph API* ke *Microsoft App Registration Portal*¹. Pada saat mendaftarkan aplikasinya, akan didapatkan *application ID* yang adalah pengenal unik untuk aplikasi yang didaftarkan, dan juga *Redirect URL* yang didaftarkan sebagai *URL* yang akan menerima kembalian *authentication* dan *token* yang akan dikirim oleh *endpoint Azure AD v2.0*, serta mendapat *application secret* yang didapat saat mengklik “*Generate New Password*” saat mendaftarkan aplikasi (berlaku jika mendaftarkan aplikasi berjenis *web*

¹<https://apps.dev.microsoft.com/>

apps). *Application ID* yang didapat saat mendaftar aplikasi akan dipakai untuk mengisi nilai dari parameter *client_id* yang akan diisi pada saat akan mengirimkan *request* untuk mendapatkan *authorization_code*. Setelah mendapatkan *authorization_code*, maka langkah selanjutnya adalah meminta *access_token* yang membutuhkan parameter *authorization_code* kepada *field code*, dan juga *application_secret* yang didapat dari pendaftaran aplikasi sebelumnya yang akan mengisi *field client_secret*. *Response* dari *request token* akan mengembalikan jangka waktu aktif dari *token* tersebut dan juga *refresh_token* yang akan berguna untuk meminta *refresh_token* saat *token* sudah *expired*.

Setelah mendapatkan *access token*, layanan untuk mendapatkan dan memanipulasi data yang tersimpan di *Microsoft* baru bisa diakses. Ada banyak layanan yang disediakan dari *Microsoft Graph API* yang tergolong di dalam *resource type*. *Resource type* merupakan kelas yang representasi dari objek yang disimpan oleh *Microsoft*. Berikut beberapa *resource type* yang disediakan oleh *Microsoft*:

2.1.1 User resource type

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

Properti

- ***aboutMe***

Properti ini bertipe *String* yang merupakan *field* untuk mendeskripsikan diri pengguna.

- ***accountEnabled***

Properti ini bertipe *Boolean* yang bernilai *true* jika akun diaktifkan dan akan bernilai *false* jika tidak. Properti ini berguna saat akan membuat akun. Nilai ini yang akan dipakai sebagai patokan sebuah akun bisa dibuat atau tidaknya.

- ***ageGroup***

Properti ini bertipe *String* yang merupakan nilai kelompok umur dari pengguna. Terdapat nilai *null*, *minor*, *notAdult*, dan juga *adult*.

- ***assignedLicenses***

Properti ini bertipe koleksi *assignedLicense* yang merupakan nilai lisensi yang diberikan kepada pengguna.

- ***assignedPlans***

Properti ini bertipe koleksi *assignedPlan* yang merupakan nilai *plan* yang diberikan kepada pengguna.

- ***birthday***

Properti ini bertipe *DateTimeOffset* yang merupakan nilai ulang tahun dari pengguna.

- ***businessPhones***

Properti ini bertipe *String* yang merupakan nomor telepon dari pengguna. Walaupun bersifat *String*, tetapi *field* ini hanya bisa diisi oleh angka.

- ***city***

Properti ini bertipe *String* yang merupakan kota lokasi pengguna.

- ***companyName***

Properti ini bertipe *String* yang merupakan nama perusahaan dimana pengguna terkait di dalamnya.

- ***consentProvidedForMinor***

Properti ini bertipe *String* yang merupakan status persetujuan bagi anak dibawah umur yang mengacu kepada properti *ageGroup*. Nilai dari properti ini bisa *null*, *granted*, *denied*, dan juga *notRequired*.

- ***country***

Properti ini bertipe *String* yang merupakan negara lokasi pengguna.

- ***createDateTime***

Properti ini bertipe *DateTimeOffset* yang merupakan tanggal dibuatnya objek pengguna.

- ***department***

Properti ini bertipe *String* yang merupakan nama departemen pengguna bekerja.

- ***displayName***

Properti ini bertipe *String* yang merupakan nama yang ditampilkan di buku alamat untuk pengguna. Biasanya disusun dari nama depan, nama tengah, dan juga nama belakang. Properti ini merupakan properti yang *required* ketika pengguna dibuat dan tidak bisa dihapus.

- ***employeeId***

Properti ini bertipe *String* yang merupakan pengidentifikasi karyawan yang diberikan kepada pengguna oleh organisasi.

- ***faxNumber***

Properti ini bertipe *String* yang merupakan nomor fax pengguna.

- ***givenName***

Properti ini bertipe *String* yang merupakan nama depan dari pengguna.

- ***hireDate***

Properti ini bertipe *DateTimeOffset* yang merupakan tanggal pengguna dipekerjakan.

- ***id***

Properti ini bertipe *String* yang merupakan tanda pengenal unik untuk pengguna.

- ***imAddresses***

Properti ini bertipe koleksi *String* yang merupakan alamat protokol inisiasi sesi *voice over IP* (VOIP) pesan untuk pengguna.

- ***interests***

Properti ini bertipe koleksi *String* yang merupakan kumpulan *String* yang mendeskripsikan ketertarikan dari pengguna.

- ***isResourceAccount***

Properti ini bertipe *Boolean* yang akan bernilai *true* jika akun merupakan *resource account* dan akan bernilai *false* jika bukan. Jika kosong akan dianggap dengan nilai *false*.

- ***jobTitle***

Properti ini bertipe *String* yang merupakan jabatan dari pengguna.

- ***legalAgeGroupClassification*** Properti ini bertipe *String* yang merupakan penentu kelompok *legalAge* dengan dihitung menggunakan properti *ageGroup* dan juga *consentProvidedForMinor*. Nilai dari properti ini bisa berupa *null*, *minorWithOutParentalConsent*, *minorWithParentalConsent*, *minorNoParentalConsentRequired*, *notAdult*, dan juga *adult*. Properti ini bersifat ***Read-Only***.

- ***licenseAssignmentStates***

Properti ini bertipe koleksi *licenseAssignmentState* yang merupakan status penugasan lisensi untuk pengguna. Properti ini bersifat ***Read-Only***.

- ***mail***

Properti ini bertipe *String* yang merupakan alamat SMTP untuk pengguna. Properti ini bersifat ***Read-Only***.

- ***mailboxSettings***

Properti ini bertipe *mailboxSettings* yang merupakan pengaturan untuk *mailbox* utama dari pengguna yang masuk.

- ***mailNickname***

Properti ini bertipe *String* yang merupakan alias *email* dari pengguna. Properti ini harus ditentukan saat pengguna dibuat.

- ***mobilePhone***

Properti ini bertipe *String* yang merupakan nomor telepon seluler utama pengguna.

- ***mySite***

Properti ini bertipe *String* yang merupakan *url* untuk situs pribadi pengguna.

- ***officeLocation***

Properti ini bertipe *String* yang merupakan lokasi kantor di tempat bisnis pengguna.

- ***onPremisesDistinguishedName***

Properti ini bertipe *String* yang merupakan nama atau DN Direktori Aktif lokal yang dibedakan. Properti ini hanya diisi untuk pelanggan yang menyinkronkan direktori lokal mereka ke *Azure Active Directory* melalui *Azure AD Connect*. Properti ini bersifat ***Read-Only***.

- ***onPremisesDomainName***

Properti ini bertipe *String* yang merupakan *dnsDomainName* yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk pelanggan yang menyinkronkan direktori lokal mereka ke *Azure Active Directory* melalui *Azure AD Connect*. Properti ini bersifat ***Read-Only***.

- ***onPremisesExtensionAttributes***

Properti ini bertipe *OnPremisesExtensionAttributes* yang merupakan *extensionAttributes* untuk pengguna. Jika properti *onPremisesSyncEnabled* bernilai ***true***, maka properti ini bersifat ***Read-Only***, tetapi jika properti *onPremisesSyncEnabled* bernilai ***false***, maka properti ini dapat diatur saat membuat atau memperbarui.

- ***onPremisesImmutableId***

Properti ini bertipe *String* yang digunakan untuk mengaitkan akun pengguna *Active Directory* lokal ke objek *Azure AD* pengguna. Properti ini ditentukan saat pembuatan akun pengguna baru di *Graph*.

- ***onPremisesLastSyncDateTime***

Properti ini bertipe *DateTimeOffset* yang memiliki fungsi untuk menunjukkan kapan terakhir kali objek disinkronkan dengan direktori lokal. Properti ini bersifat ***Read-Only***.

- ***onPremisesProvisioningErrors***

Properti ini bertipe koleksi *onPremisesProvisioningError* yang merupakan kesalahan saat menggunakan produk sinkronisasi *Microsoft*.

- ***onPremisesSamAccountName***

Properti ini bertipe *String* yang merupakan *samAccountName* lokal yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk pelanggan yang menyinkronkan direktori lokal mereka ke *Azure Active Directory* melalui *Azure AD Connect*. Properti ini bersifat ***Read-Only***.

- ***onPremisesSecurityIdentifier***

Properti ini bertipe *String* yang merupakan pengidentifikasi keamanan lokal (SID) untuk pengguna yang disinkronkan dari lokal ke *cloud*. Properti ini bersifat ***Read-Only***.

- ***onPremisesSyncEnabled***

Properti ini bertipe *Boolean* yang bernilai ***true*** jika objek ini disinkronisasi dari direktori lokal dan bernilai ***false*** jika objek ini awalnya disinkronisasi dari direktori lokal tetapi tidak lagi disinkronkan. Properti ini juga bisa bernilai ***null*** jika objek ini tidak pernah disinkronkan dari direktori lokal. Properti ini bersifat ***Read-Only***.

- ***onPremisesUserPrincipalName***

Properti ini bertipe *String* yang merupakan *userPrincipalName* di tempat yang disinkronkan dari direktori lokal. Properti ini hanya diisi untuk pelanggan yang menyinkronkan direktori lokal mereka ke *Azure Active Directory* melalui *Azure AD Connect*. Properti ini bersifat ***Read-Only***.

- ***otherMails***

Properti ini bertipe *String* yang merupakan daftar dari alamat *email* tambahan untuk pengguna.

- ***passwordPolicies***

Properti ini bertipe *String* yang menentukan kebijakan kata sandi untuk pengguna.

- ***passwordProfile***

Properti ini bertipe *passwordProfile* yang menentukan profil kata sandi untuk pengguna. Profil ini berisi kata sandi dari pengguna. Properti ini diperlukan saat pengguna dibuat dan kata sandi harus memenuhi persyaratan yang ditentukan oleh properti *passwordPolicies*.

- ***pastProjects***

Properti ini bertipe koleksi *String* yang merupakan daftar proyek yang sudah lalu dari pengguna.

- ***postalCode***

Properti ini bertipe *String* yang merupakan kode pos dari pengguna.

- ***preferredDataLocation***

Properti ini bertipe *String* yang merupakan lokasi data yang dipilih oleh pengguna.

- ***preferredLanguage***

Properti ini bertipe *String* yang merupakan bahasa yang dipilih oleh pengguna.

- ***preferredName***

Properti ini bertipe *String* yang merupakan nama yang dipilih oleh pengguna.

- ***provisionedPlans***

Properti ini bertipe koleksi *provisionedPlan* yang merupakan *plan* yang disediakan untuk pengguna. Properti ini bersifat ***Read-Only*** dan tidak bisa bernilai ***null***.

- ***proxyAddresses***

Properti ini bertipe koleksi *String*.

- ***responsibilities***

Properti ini bertipe koleksi *String* yang merupakan daftar dari tanggung jawab pengguna.

- ***schools***

Properti ini bertipe koleksi *String* yang merupakan daftar dari instansi pendidikan yang pernah dihadiri oleh pengguna.

- ***showInAddressList***

Properti ini bertipe *Boolean* yang bernilai ***true*** jika daftar alamat *Outlook global* harus berisi pengguna ini, dan bernilai ***false*** jika tidak. Jika tidak diberikan nilai, maka akan bernilai ***true***.

- ***skills***

Properti ini bertipe koleksi *String* yang merupakan daftar dari kemampuan pengguna.

- ***state***

Properti ini bertipe *String* yang merupakan negara atau provinsi yang terdapat di alamat pengguna.

- ***streetAddress***

Properti ini bertipe *String* yang merupakan alamat dari tempat bisnis pengguna.

- ***surname***

Properti ini bertipe *String* yang merupakan nama keluarga atau nama belakang dari pengguna.

- ***usageLocation***

Properti ini bertipe *String* yang merupakan 2 huruf dari kode negara pengguna yang digunakan untuk memeriksa ketersediaan layanan di negara pengguna.

- ***userPrincipalName***

Properti ini bertipe *String* yang merupakan nama utama dari pengguna yang memakai standar internet RFC 822.

- ***userType***

Properti ini bertipe *String* yang digunakan untuk mengklasifikasi tipe pengguna, seperti contohnya “*Member*” dan “*Guest*”.

Method

Untuk mengakses setiap *method* yang dijabarkan, perlu diketahui bahwa untuk mengirimkan *request*, diperlukan *header* yang berisikan *Authorization* yang diisi dengan nilai dari *Access_token* yang didapat dari proses permintaan *access token*. *Endpoint* utama untuk mengirimkan request adalah ke <https://graph.microsoft.com/v1.0> lalu dilanjutkan dengan *endpoint* fungsinya masing-masing seperti yang diberikan dan dijelaskan.

- ***List users***

Method ini berfungsi untuk mendapatkan daftar dari objek pengguna. *Method* ini direquest dengan cara mengirim *request get* kepada *endpoint /users* dengan *headers* berisikan otorisasi yang diisi dengan nilai dari *access token* dan juga *Content-Type* yang bernilai *application/json*. *Request* untuk *method* ini juga bisa diisi dengan *parameter \$select* untuk mengembalikan *field* yang hanya diisi di dalam *parameter \$select* tersebut, dan dalam *parameter* itu, tiap *field* dipisah dengan tanda koma (,) contohnya [https://graph.microsoft.com/v1.0/users?\\$select=displayName,givenName,postalCode](https://graph.microsoft.com/v1.0/users?$select=displayName,givenName,postalCode). *Respons* dari *method* ini akan mengembalikan *json* dari objek pengguna.

- **Create user**

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

- **Get user**

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

- **Update user**

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

- **Delete user**

Method ini berfungsi untuk menghapus pengguna. *Method* ini diakses dengan mengirimkan *delete request* ke *endpoint /users/{id | userPrincipalName}*.

- **List messages**

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

- **Create message**

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

- **List mailFolders**

Method ini berfungsi untuk mendapatkan folder-folder surat dibawah folder *root* dari pengguna yang masuk. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint /users/{id | userPrincipalName}/mailFolders*.

- **Create mailFolder**

Method ini berfungsi untuk membuat *mailFolder* ke dalam koleksi *mailFolders*. *Method* ini dijalankan dengan cara mengirimkan *post request* ke *endpoint /users/{id | userPrincipalName}/mailFolders*.

- **sendMail**

Method ini berfungsi untuk mengirim pesan. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id | userPrincipalName}/sendMail*. *Body* dari *request* ini berisi *message* dan juga sebuah *Boolean* untuk atribut *saveToSentItems*.

- **List events**

Method ini berfungsi untuk mendapatkan daftar objek *event* di dalam kotak pesan dari pengguna. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint /users/{id | userPrincipalName}/events*. *Headers* pada *method* ini akan berisi otorisasi yang diisi nilai dari *access token* sebagai *header* wajib. Lalu ada juga *header* pilihan yaitu

outlook.timezone dan juga *outlook.body-content-type*. Pada *header timezone*, jika tidak diisi, maka nilai awal yang dikembalikan adalah dalam *UTC*. *Request* ini juga bisa difilter dengan menggunakan *parameter \$select*.

- ***Create event***

Method ini berfungsi untuk membuat *event* ke dalam koleksi dari *event-event*. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id | userPrincipalName}/events*. *Body* dari *request* ini akan berisi *json* yang merepresentasikan objek *event*.

- ***List calendars***

Method ini berfungsi untuk mendapatkan daftar objek *calendar*. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint /users/{id | userPrincipalName}/calendars*.

- ***Create calendar***

Method ini berfungsi untuk membuat objek *calendar* baru yang akan dikirim ke dalam koleksi objek *calendars*. *Method* ini akan dijalankan dengan cara mengirimkan *post request* ke *endpoint /users/{id | userPrincipalName}/calendars* dengan *body json* yang merepresentasikan objek *calendar*.

- ***List calendarGroups***

Method ini berfungsi untuk mendapatkan daftar objek *calendarGroup*. *Method* ini dijalankan dengan cara mengirimkan *post request* ke *endpoint /users/{id | userPrincipalName}/calendarGroups*.

- ***Create calendarGroup***

Method ini berfungsi untuk membuat objek *calendarGroup* baru kedalam koleksi *calendarGroup*. *Method* ini dijalankan dengan cara mengirimkan *post request* ke *endpoint /users/{id | userPrincipalName}/calendarGroups* dengan *body* berupa *json* yang merepresentasikan objek *calendarGroup*.

- ***List calendarView***

Method ini berfungsi untuk mendapatkan koleksi objek *event*. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint /users/{id | userPrincipalName}/calendarView?startDateTime={start_datetime}&endDateTime={end_datetime}*. Terdapat tambahan *parameter* yaitu *startTime* dan *endTime* yang akan menjadi acuan mulai dari kapan dan sampai kapan *calendarView* yang akan diambil.

- ***List contacts***

Method ini berfungsi untuk mendapatkan daftar kontak dari folder *Contacts* pengguna yang masuk. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint /users/{id | userPrincipalName}/contacts*. *Method* ini juga bisa menerima tambahan *parameter \$filter* yang berfungsi untuk memfilter balikan yang didapat.

- ***Create contact***

Method ini berfungsi untuk membuat kontak baru untuk dimasukkan ke dalam koleksi kontak. *Method* ini dijalankan dengan cara mengirimkan *post request* ke *endpoint /users/{id | userPrincipalName}/contacts*. dan memiliki *body* berupa *json* yang merepresentasikan objek *contact*.

- ***List contactFolders***

Method ini berfungsi untuk mendapatkan koleksi folder kontak dari pengguna yang masuk. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint /users/{id | userPrincipalName}/contactFolders*.

- **Create contactFolder**

Method ini berfungsi untuk membuat folder kontak. *Method* ini dijalankan dengan cara mengirimkan *post request* ke *endpoint* `/users/{id | userPrincipalName}/contactFolders` dengan memiliki *body* berupa *json* yang merepresentasikan objek *contactFolder*.

- **List directReports**

Method ini berfungsi untuk mendapatkan pengguna dan kontak yang melaporkan pengguna dari properti *directReports*. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint* `/users/{id | userPrincipalName}/directReports`.

- **List manager**

Method ini berfungsi untuk mendapatkan *manager* pengguna dari properti *manager*. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint* `/users/{id | userPrincipalName}/manager`. *Method* ini akan mengembalikan nilai berupa *json* objek dari pengguna yang menjadi *managernya*.

- **List memberOf**

Method ini berfungsi untuk mendapatkan kelompok dan peran dari anggota langsung pengguna lewat properti *memberOf*. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint* `/users/{id | userPrincipalName}/memberOf`.

- **List transitive memberOf**

Method ini berfungsi untuk mendapatkan kelompok dan peran dari pengguna lewat properti *memberOf*, tetapi *method* ini bersifat transitif dan mencakup grup-grup dimana pengguna menjadi anggota. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint* `/users/{id | userPrincipalName}/transitiveMemberOf`.

- **List ownedDevices**

Method ini berfungsi untuk mendapatkan perangkat yang dimiliki oleh pengguna dari properti *ownedDevices*. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint* `/users/{id | userPrincipalName}/ownedDevices`.

- **List ownedObjects**

Method ini berfungsi untuk mendapatkan objek yang dimiliki pengguna yang didapat dari properti *ownedObjects*. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint* `/users/{id | userPrincipalName}/ownedObjects`.

- **List registeredDevices**

Method ini berfungsi untuk mendapatkan perangkat yang terdaftar oleh pengguna dari properti *registeredDevices*. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint* `/users/{id | userPrincipalName}/registeredDevices`.

- **List createdObjects**

Method ini berfungsi untuk mendapatkan objek yang dibuat oleh pengguna dari properti *createdObjects*. *Method* ini dijalankan dengan cara mengirimkan *get request* ke *endpoint* `/users/{id | userPrincipalName}/createdObjects`.

- **assignLicense**

Method ini berfungsi untuk menambah atau membuang “*subscriptions*” dari pengguna, serta bisa untuk mengaktifkan dan menonaktifkan paket spesifik terkait dengan langganan. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint* `/users/{id | userPrincipalName}/assignLicense` dan *body* dari *request* ini bisa diisi dengan parameter *addLicenses* yang bertipe *AssignedLicense* dan juga parameter *removeLicenses* yang diisi dengan guid dari lisensi yang sudah aktif sekarang.

- ***List licenseDetails***

Method ini berfungsi untuk mendapatkan koleksi objek *licenseDetails*. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint /users/{id}/licenseDetails*.

- ***checkMemberGroups***

Method ini berfungsi untuk memeriksa keanggotaan dalam daftar grup. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id | userPrincipalName}/checkMemberGroups* dan *body* yang dibutuhkan *request* ini adalah *groupIds* yang merupakan *id* dari grup yang akan dicari.

- ***getMemberGroups***

Method ini mengembalikan semua grup dimana pengguna menjadi anggota didalamnya. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id | userPrincipalName}/getMemberGroups* dan memerlukan *body request* yaitu *securityEnabledOnly* yang bertipe *Boolean* yang bernilai *true* jika hanya *security groups* dari pengguna yang terdaftar sebagai anggota yang dikembalikan, dan bernilai *false* jika harus mengembalikan semua grup yang memiliki pengguna sebagai anggotanya.

- ***getMemberObjects***

Method ini mengembalikan semua grup dan peran dimana pengguna menjadi anggota didalamnya. *Method* ini dijalankan dengan cara mengirimkan *post request* kepada *endpoint /users/{id | userPrincipalName}/getMemberObjects* dan memerlukan *body request* yaitu *securityEnabledOnly* seperti yang sudah dijelaskan di *method* sebelumnya.

- ***reminderView***

Method ini mengembalikan daftar pengingat di kalender dengan jam mulai dan berakhirnya secara spesifik. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint /users/{id | userPrincipalName}/reminderView* yang memerlukan parameter tambahan yaitu *startDateTime* dan juga *endDateTime* yang keduanya bertipe *String*.

- ***delta***

Method ini berfungsi untuk mendapatkan perubahan tambahan pengguna. *Method* ini dijalankan dengan cara mengirimkan *get request* kepada *endpoint /users/delta*.

Seluruh *endpoint /users/{id | userPrincipalName}* yang dijabarkan di setiap *method* sebelumnya bisa diganti dengan */me* jika dalam keadaan pengguna sudah melakukan *login* dan masih memiliki sesi pada *Windows Live*.

2.1.2 Event resource type

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

Properti

- ***attendees***

Properti ini menunjukkan daftar dari hadirin dari suatu *event*. Properti ini bertipe koleksi *attendee*.

- ***body***

Properti ini merupakan isi pesan terkait dengan acara. Bisa berbentuk HTML atau berupa teks. Properti ini bertipe *itemBody*.

- ***bodyPreview***

Properti ini bertipe *String* yang merupakan pratinjau pesan terkait acara.

- **categories**

Properti ini merupakan daftar kategori yang terkait dengan acara. Properti ini bertipe koleksi *String*.

- **changeKey**

Properti ini bertipe *String* yang merupakan pengidentifikasi versi dari objek acara. Setiap kali acara diubah, properti ini juga berubah.

- **createdDateTime**

Properti ini menunjukkan tanggal dari acara dibuat. Properti ini bertipe *DateTimeOffset*.

- **end**

Properti ini bertipe *dateTimeTimeZone* yang berfungsi untuk menunjukkan tanggal, waktu, dan zona waktu acara akan berakhir.

- **hasAttachments**

Properti ini bertipe *Boolean* yang akan menunjukkan ada atau tidaknya lampiran. Nilai *true* artinya ada lampiran dan *false* untuk tidak adanya lampiran.

- **iCalUID**

Properti ini merupakan identifikasi unik yang dibagikan oleh semua instansi yang tergabung dalam acara di berbagai kalender. Properti ini bertipe *String* dan juga bersifat *Read-Only*.

- **id**

Properti ini bertipe *String* dan juga bersifat *Read-Only*.

- **importance**

Properti ini bertipe *importance* yang menunjukkan pentingnya acara. Nilai dari properti ini bisa berupa *low*, *normal*, dan juga *high*.

- **isAllDay**

Properti ini untuk menunjukkan apakah acara ini berjalan sehari atau tidak. Bertipe *Boolean* yang akan bernilai *true* jika acaranya berlangsung sehari, dan bernilai *false* jika tidak.

- **isCancelled**

Properti ini untuk menunjukkan status acara ini dibatalkan atau tidak. Bertipe *Boolean* yang bernilai *true* jika dibatalkan dan *false* jika tidak dibatalkan.

- **isOrganizer**

Properti ini bertipe *Boolean* yang menunjukkan nilai jika pengirim pesan adalah penyelenggara dari acara atau bukan. Bernilai *true* jika pengirim pesan adalah penyelenggara acara dan *false* untuk bukan penyelenggara.

- **isReminderOn**

Properti ini untuk mengetahui status dari pengingat bagi pengguna dari acara. Bertipe *Boolean* yang akan bernilai *true* jika ada peringatan yang diatur untuk mengingatkan kepada pengguna, dan bernilai *false* jika tidak ada.

- **lastModifiedDateTime**

Properti ini untuk menunjukkan kapan terakhir data acara dimodifikasi. Properti ini bertipe *DateTimeOffset*.

- **location**

Properti ini menunjukkan lokasi dari acara yang bertipe *location*.

- ***locations***
Properti ini berisi kumpulan dari lokasi acara. Properti ini bertipe koleksi *location*.
- ***onlineMeetingUrl***
Properti ini berisi *URL* untuk melakukan rapat secara *online* dan bertipe *String*.
- ***organizer***
Properti ini untuk menunjukkan penyelenggara dari acara. Properti ini bertipe *recipient*.
- ***originalEndTimeZone***
Properti ini menunjukkan zona waktu acara berakhir pada awal acara ini dibuat. Properti ini bertipe *String*.
- ***originalStart***
Properti ini merupakan informasi mulainya acara sejak awal acara ini dibuat. Properti ini bertipe *DateTimeOffset*.
- ***originalStartTimeZone***
Properti ini menunjukkan zona waktu acara dimulai sejak acara ini dibentuk. Properti ini bertipe *String*.
- ***recurrence***
Properti ini menunjukkan pola pengulangan untuk acara. Properti ini bertipe *patternedRecurrence*.
- ***reminderMinutesBeforeStart***
Properti ini menunjukkan berapa menit peringatan pengingat sebelum acara dimulai. Properti ini bertipe *Int32*.
- ***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*.
- ***responseStatus***
Properti ini menunjukkan jenis tanggapan yang dikirim sebagai *respon* terhadap pesan acara. Properti ini bertipe *responseStatus*.
- ***sensitivity***
Properti ini memiliki kemungkinan nilai yaitu *normal*, *personal*, *private*, atau *confidential*. Properti ini bertipe *sensitivity*.
- ***seriesMasterId***
Properti ini adalah *ID* untuk *item master seri* berulang jika acara ini merupakan dari seri berulang. Properti ini bertipe *String*.
- ***showAs***
Properti ini bertipe *freeBusyStatus* yang memiliki kemungkinan nilai yaitu *free*, *tentative*, *busy*, *oof*, *workingElsewhere*, dan *unknown*.
- ***start***
Properti ini menunjukkan tanggal, waktu, dan zona waktu acara dimulai. Bertipe *dateTimeTimeZone*.
- ***subject***
Properti ini menyimpan subyek dari acara. Properti ini bertipe *String*.

- ***type***

Properti ini bertipe *eventType* yang memiliki kemungkinan nilai yaitu *singleInstance*, *occurrence*, *exception*, dan *seriesMaster*. Properti ini bersifat *Read-Only*.

- ***webLink***

Properti ini berisi *URL* yang berfungsi untuk membuka acara ini di *Outlook Web App*. Properti ini bertipe *String*.

method

Untuk mengakses setiap *method* yang akan dijabarkan, perlu diketahui bahwa untuk mengirimkan *request*, diperlukan *header* yang berisikan *Authorization* yang diisi dengan nilai dari *Access_token* yang didapat dari proses melakukan *access token* sebelumnya. *Endpoint* utama untuk mengirimkan *request* adalah ke <https://graph.microsoft.com/v1.0> lalu dilanjutkan dengan *endpoint* fungsinya masing-masing seperti yang akan diberikan dan dijelaskan.

- ***List events***

Method ini sama seperti yang sudah dijelaskan di bagian *method* dari *user resource type*.

- ***Create events***

Method ini sama seperti yang sudah dijelaskan di bagian *method* dari *user resource type*.

- ***Get events***

Method ini untuk mendapatkan properti dan hubungan untuk objek *event* yang spesifik. *Endpoint* dari *method* ini adalah */users/{id | userPrincipalName}/events/{id}* dengan mengirimkan *get request*. Jika *request* berhasil, akan mengembalikan objek *event* yang diminta.

- ***Update***

Method ini untuk membarui properti-properti yang terdapat dalam objek *event*. *Endpoint* dari *method* ini adalah */users/{id | userPrincipalName}/events/{id}* dengan mengirimkan *patch request*. *Request* ini diisi dengan *body* yang berisikan properti dari objek *event* yang akan diubah.

- ***Delete***

Method ini berfungsi untuk menghapus objek *event*. Dikirimkan kepada *endpoint* */users/{id | userPrincipalName}/events/{id}* dengan mengirimkan *delete request*.

- ***accept***

Method ini untuk menerima *event* spesifik di kalender pengguna. *Endpoint* dari *method* ini adalah */users/{id | userPrincipalName}/events/{id}/accept* dengan mengirimkan *post request*. *Request body* dari *method* ini bisa diisi dengan *comment* yang bertipe *String* yang fungsinya untuk menunjukkan catatan dari *respon*, dan juga *sendResponse* yang bertipe *Boolean* yang bernilai *true* jika *response* akan dikirim ke penyelenggara, dan bernilai *false* jika tidak. Nilai *default* dari *sendResponse* yaitu *true*.

- ***tentativelyAccept***

Method ini untuk menerima *event* spesifik di kalender pengguna untuk sementara. *Endpoint* dari *method* ini adalah */users/{id | userPrincipalName}/events/{id}/tentativelyAccept* dengan mengirimkan *post request*. *Request body* dari *method* ini sama seperti *method accept*.

- ***decline***

Method ini untuk menolak undangan dari spesifik acara di kalender pengguna. *Endpoint* dari *method* ini adalah */users/{id | userPrincipalName}/events/{id}/decline* dengan cara mengirimkan *post request*. *Request body* dari *method* ini sama seperti *method accept*.

- ***delta***

Method ini berfungsi untuk mendapatkan serangkaian acara yang ditambahkan, dihapus, dan diperbarui dalam *calendarView* dari kalender utama pengguna. *Endpoint* dari *method* ini adalah */users/{id}/calendarView/delta* yang memerlukan parameter wajib yaitu *startDateTime* dan *endDateTime* serta ada parameter *\$deltatoken* dan juga *\$skiptoken*. *Method* ini dikirimkan dengan mengirim *get request*.

- ***dismissReminder***

Method ini berfungsi untuk memberhentikan peringatan untuk acara yang sudah dipasang di kalender pengguna. *Endpoint* dari *method* ini adalah */users/{id} / userPrincipalName}/events/{id}/dismissReminder* dengan mengirimkan *post request*.

- ***snoozeReminder***

Method ini berfungsi untuk menunda peringatan yang sudah dipasang di kalender pengguna. *Endpoint* dari *method* ini menuju ke */users/{id} / userPrincipalName}/events/{id}/snoozeReminder* dengan mengirimkan *post request*. Memiliki *request body* yang berisi *newReminderTime* yang bertipe *DateTimeTimeZone* yang merupakan waktu baru untuk menjalankan peringatan untuk acara.

- ***List instances***

Method ini untuk mendapatkan contoh acara dari waktu yang spesifik. Memiliki *endpoint* */users/{id} / userPrincipalName}/events/{id}/instances* yang memiliki parameter wajib *startDateTime* dan juga *endDateTime* dengan mengirimkan *get request*.

- ***List attachments***

Method ini berfungsi untuk mendapatkan kumpulan lampiran dari suatu acara. *Endpoint* dari *method* ini adalah */users/{id} / userPrincipalName}/events/{id}/attachments* dengan cara mengirimkan *get request*.

- ***Add attachment***

Method ini berfungsi untuk menambahkan lampiran ke suatu acara dengan maksimum ukuran file sebesar 4MB. *Endpoint* dari *method* ini adalah */users/{id} / userPrincipalName}/events/{id}/attachments* dengan mengirimkan *post request* yang memiliki *request body* berupa *json* representasi dari objek *attachment*.

Seluruh *endpoint* */users/{id} / userPrincipalName}* bisa diganti dengan */me* jika pengguna telah melakukan *login* dan masih terdapat sesi di *Windows Live*.

2.1.3 Lain-lain

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

Slack

Slack adalah sebuah aplikasi yang memiliki fitur utama berupa *chatting* berbasis *cloud* yang dibuat dan dikembangkan oleh *Slack Technologies*. Di dalam aplikasi *Slack*, setiap pengguna akan tergabung ke dalam sebuah grup yang disebut dengan *workspace*. Para pengguna bisa bergabung dengan *workspace* timnya melalui *url* yang akan diberikan oleh *admin* dari *workspace* tim tersebut dan *url* itu didapatkan oleh *admin* tim saat *admin* tim membuat *workspace*. Di dalam *workspace*, ada terdapat beberapa ruang *chat* yang disebut dengan *channel*. *Channel* di dalam *workspace* ini bisa dibagi menjadi 2 kategori yaitu *public channel* dan juga *private channel*. *Public channel* dapat diakses oleh semua pengguna yang telah masuk ke dalam *workspace* timnya, sedangkan *private channel* hanya dapat diakses dan digunakan oleh pengguna yang didaftarkan dan diundang oleh

admin dari *private channel* tersebut. Pada setiap *workspace* dapat ditambahkan dan dipasang dengan berbagai tambahan aplikasi pembantu seperti *Github*, *Trello*, dan masih banyak aplikasi lain yang sudah tersedia yang bisa ditambahkan ke dalam *workspace*. Selain aplikasi yang sudah disediakan, aplikasi pembantu pun bisa dibangun sendiri yang bisa berhubungan langsung dengan *Slack*. Untuk membangun aplikasi pembantu yang bisa terhubung dengan *Slack*, tim dari *Slack* sudah menyediakan *API* yang bisa menghubungkan aplikasi lain dengan fitur-fitur yang ada di dalam aplikasi *Slack*.

2.2 Slack API

Slack API adalah *webservice* yang akan digunakan untuk menghubungkan data yang sudah didapat dari *Outlook.com Calendar* ke aplikasi *Slack*.^[2] Untuk mengakses *Slack API*, langkah awal yang harus dilakukan adalah mendaftarkan aplikasi yang dibuat dan juga mendaftarkannya ke *workspace* yang dipakai untuk menjalankan aplikasi yang dibuat. Untuk mendaftarkan aplikasi yang dibuat bisa menuju ke laman <https://api.slack.com/apps>. Aplikasi yang sudah terdaftar akan diberikan *Client ID* yang unik dan juga *Client Secret* yang digunakan pada proses *OAuth*.

Proses *OAuth* adalah proses yang meminta *access token* dengan terlebih dahulu meminta *authentication code*. Proses pertama yang dijalankan dalam rangkaian proses *OAuth* adalah dengan meminta *authorization code* yang dipakai untuk mendapatkan *access token*. Langkah ini dilakukan dengan cara mengirimkan *get request* ke *URL* <https://slack.com/oauth/authorize> dengan *get parameter* yang wajib yaitu *client_id* yang diberikan pada saat mendaftarkan aplikasi yang dibuat dan juga *get parameter scope*, serta memiliki *parameter* yang bersifat opsional yaitu *redirect_uri* yang berfungsi untuk alamat tujuan dari kembalian yang dikirim oleh API, *state* yaitu yang berupa *String* unik untuk diteruskan kembali setelah selesai, dan juga *team* berupa *Slack team ID* dari *workspace* yang berfungsi untuk membatasi. *Parameter scope* disini berguna untuk menentukan dengan tepat bagaimana aplikasi perlu mengakses akun pengguna *Slack*. Penulisan format *parameter scope* yaitu merujuk ke objek yang akan diberi akses, dan dilanjutkan dengan kelas tindakan pada objek yang diberikan izin, contohnya *file:read*. Selain itu ada juga perspektif opsional yang berisi *user*, *bot*, dan *admin* yang akan memengaruhi tindakan yang muncul nantinya di dalam aplikasi *Slack*, contohnya *chat:write:user* yang berarti akan mengirimkan pesan dari pengguna yang memiliki wewenang. Kelas tindakan yang ada disini ada 3 yaitu:

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

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

- channels:history
 - channels.history
 - channels.replies
- channels:read
 - channels.info
 - channels.list
- channels:write
 - channels.archive
 - channels.create
 - channels.invite
 - channels.join
 - channels.kick
 - channels.leave
 - channels.mark
 - channels.rename
 - channels.setPurpose

- channels.setTopic
- channels.unarchive
- conversations.join
- chat:write:bot
 - chat.delete
 - chat.deleteScheduledMessage
 - chat.postEphemeral
 - chat.postMessage
 - chat.scheduleMessage
 - chat.update
- chat:write:user
 - chat.delete
 - chat.deleteScheduledMessage
 - chat.postEphemeral
 - chat.postMessage
 - chat.scheduleMessage
 - chat.update
- dnd:read
 - dnd.info
 - dnd.teamInfo
- dnd:write
 - dnd.endDnd
 - dnd.endSnooze
 - dnd.setSnooze
- dnd:write:user
 - dnd.endDnd
 - dnd.endSnooze
 - dnd.setSnooze
- emoji:read
 - emoji.list
- files:read
 - files.info
 - files.list
- files:write:user
 - files.comments.delete
 - files.delete
- files.revokePublicURL
- files.sharedPublicURL
- files.upload
- groups:history
 - groups.history
 - groups.replies
- groups:read
 - groups.info
 - groups.list
- groups:write
 - groups.archive
 - groups.create
 - groups.createChild
 - groups.invite
 - groups.kick
 - groups.leave
 - groups.mark
 - groups.open
 - groups.rename
 - groups.setPurpose
 - groups.setTopic
 - groups.unarchive
- identity.basic
 - users.identity
- identity.basic:user
 - users.identity
- im:history
 - im.history
 - im.replies
- im:read
 - im.list
- im:write
 - im.close
 - im.mark
 - im.open
- links:write
 - chat.unfurl

- mpim:history
 - mpim.history
 - mpim.replies
- mpim:read
 - mpim.list
- mpim:write
 - mpim.close
 - mpim.mark
 - mpim.open
- pins:read
 - pins.list
- pins:write
 - pins.add
 - pins.remove
- reactions:read
 - reactions.get
 - reactions.list
- reactions:write
 - reactions.add
 - reactions.remove
- reminders:read
 - reminders.info
 - reminders.list
- reminders:read:user
 - reminders.info
 - reminders.list
- reminders:write
 - reminders.add
 - reminders.complete
 - reminders.delete
- reminders:write:user
 - reminders.add
 - reminders.complete
 - reminders.delete
- search:read
 - search.all
 - search.files
 - search.messages
- stars:read
 - stars.list
- stars:write
 - stars.add
 - stars.remove
- team:read
 - team.info
- tokens.basic
 - migration.exchange
- usergroups:read
 - usergroups.list
 - usergroups.users.list
- usergroups:write
 - usergroups.create
 - usergroups.disable
 - usergroups.enable
 - usergroups.update
 - usergroups.users.update
- users.profile:read
 - team.profile.get
 - users.profile.get
- users.profile:write
 - users.deletePhoto
 - users.profile.set
 - users.setPhoto
- users.profile:write:user
 - users.profile.set
- users:read
 - bot.info
 - users.getPresence
 - users.info
 - users.list
- users:read.email
 - users.lookupByEmail
- users:write
 - users.setActive
 - users.setPresence

Authorization code yang telah didapat memiliki waktu kadaluarsa selama 10 menit. Setelah mendapatkan *authorization code*, langkah selanjutnya yang harus dilakukan dalam rangkaian proses OAuth adalah menukar *authorization code* dengan *access token* dengan cara mengirimkan *post request* kepada *endpoint* <https://slack.com/api/oauth.access> yang memiliki *request body* yang wajib yaitu *client_id*, *client_secret*, dan *code*. *Client_id* dan juga *client_secret* didapat dari awal mendaftarkan aplikasi. Parameter *code* didapat dari *authorization code* yang didapatkan dari langkah sebelumnya.

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

users.profile

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

- **users.profile.get**

Method ini dijalankan dengan cara mengirimkan *get request* ke *endpoint* <https://slack.com/api/users.profile.get> dan memerlukan *token* serta *scope* yang bernilai *users.profile.read*. Fungsi dari *method* ini adalah untuk mendapatkan informasi tentang profil pengguna.

- **users.profile.set**

Method ini dijalankan dengan cara mengirimkan *post request* ke *endpoint* <https://slack.com/api/users.profile.set>. *Method* ini juga memerlukan *token* dari pengguna serta bisa menerima *request body* yang berisi profil pengguna yang berbentuk *json* objek. *Method* ini berfungsi untuk mengubah isi dari profil pengguna.

2.3 Node.js

Node.js adalah *platform* perangkat lunak pada sisi *server*.^[3] *Node.js* ini ditulis dengan menggunakan bahasa *Javascript* dengan menggunakan *npm* sebagai default *package manager*. *Package manager* adalah kumpulan perangkat untuk mengotomatisasi proses instalasi, *upgrade*(perbaikan), konfigurasi, atau menghapus paket dari perangkat lunak. Untuk menggunakan *node.js*, dibutuhkan untuk mengunduh dan menginstal terlebih dahulu *node.js* yang akan dipakai pada situs resmi yang tersedia². Setelah mengunduh dan menginstall, barulah *node.js* bisa digunakan.

Format *file* yang digunakan oleh *node.js* ini menggunakan format *.js*. Untuk bisa menjalankan *file* yang sudah dibuat, *node.js* memiliki perintah yang harus dijalankan di *terminal* yaitu perintah “*node (nama file)*”. Dengan perintah seperti itu, semua *code* akan dieksekusi. Dengan menggunakan *node.js*, sebuah kode program yang berbahasa *JavaScript* akan menjadi bersifat *server-base*, yang biasanya program *JavaScript* adalah sebuah *client-base*.

Node.js menyediakan berbagai macam kelas-kelas yang bisa membantu untuk membuat kode program berjalan sesuai dengan kebutuhan. Setiap kelas dari *library node.js* yang digunakan akan dipanggil di kode program dengan menggunakan perintah “*require()*” dengan *parameter* nama kelas yang akan digunakannya. Contoh dari kelas yang ada di *library node.js* akan dijabarkan pada subbab 2.3.1 sampai dengan subbab 2.3.8

2.3.1 HTTP

Untuk bisa mengakses dan menggunakan kelas *http.server* dan *http.client*, maka harus menggunakan kode “*require('http')*”. Kelas *interface HTTP* di dalam *node.js* ini didesign untuk mendukung

²<https://nodejs.org/en/>

banyak fitur protokol yang sulit digunakan. Dengan adanya *interface HTTP*, *node.js* bisa mengirim data melalui *Hyper Text Transfer Protocol(HTPP)*. Header pada pesan *HTTP* merepresentasikan 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>

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

- ***url***

Bertipe *String* atau bisa bertipe *class* dari *URL*.

- ***options***

- ***Protocol*** bertipe *String* yang menggambarkan protokol yang digunakan dengan nilai *default* adalah ‘*http*:’.
- ***host*** bertipe *String* yang merupakan nama *domain* atau *IP address* dari *server* untuk mengeluarkan *request* dengan nilai *default* adalah ‘*localhost*’.
- ***hostname*** bertipe *String* yang merupakan nama lain untuk *host*.
- ***family*** bertipe *number* yang merupakan nilai dari versi *IP address* untuk melambangkan *host* atau *hostname*. Nilai yang bisa dipakai adalah 4 atau 6.
- ***port*** bertipe *number* yang merupakan nilai dari *port* dari *server*. Nilai *default* yang dipakai adalah 80.
- ***localAddress*** bertipe *String* yang menggambarkan *interface* lokal yang berfungsi untuk mengikat koneksi jaringan.
- ***socketPath*** bertipe *String* yang merupakan *socket domain* dari Unix. Nilai ini tidak dapat ditentukan jika salah satu dari *host* maupun *port* ditentukan. Nilai ini menentukan soket TCP.
- ***method*** bertipe *String* yang menentukan nilai dari tipe *request* *HTTP*. Nilai *default* yang dipakai adalah ‘*GET*’.
- ***path*** direktori *endpoint* pada *request* yang bertipe *String*. Nilai ini bisa disertakan dengan *query String* jika dibutuhkan. Nilai *default* dari ini adalah ‘/’.
- ***headers*** bertipe *Object* yang merupakan objek yang mengandung *request header*.
- ***auth*** bertipe *String* yang merupakan nilai dari otentifikasi dasar contohnya adalah ‘*user:password*’.
- ***agent*** bertipe *http.Agent* atau *boolean* yang berfungsi untuk mengontrol tingkah laku dari *agent*. Nilai yang mungkin ada dalam option ini adalah:
 - * *undefined*: sebagai nilai *default*. Menggunakan *http.globalAgent* untuk *host* dan *port*.
 - * *Agent Object*: secara eksplisit menggunakan *agent* yang diteruskan.

- * *false*: dikarenakan *agent* baru dengan nilai-nilai *default* yang akan dipakai.
- ***createConnection*** sebuah fungsi yang membuat sebuah *socket/stream* untuk digunakan sebagai *request* ketika pilihan *agent* tidak digunakan.
- ***timeout*** bertipe *number* yang memiliki nilai yang menggambarkan batas waktu dari *socket* di dalam satuan *milliseconds*.
- ***setHost*** yang bertipe *boolean* yang berguna untuk menentukan akan menambah *header Host* secara otomatis atau tidak. Nilai *default* dari *options* ini adalah *true*.
- ***callback*** yang berupa *function*.
- ***Returns*** bertipe *http.ClientRequest*.

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

2.3.2 Express.js

³ *Express.js* adalah sebuah *web application framework* untuk *Node.js*. Cara untuk memasang *express* ini adalah dengan menggunakan *command*:

```
$ npm install express
```

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

- *Express-generator*

Express-generator ini memiliki fungsi untuk membangun rangka dari aplikasi dengan mudah. Dengan menggunakan *express-generator*, maka didapatkan 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. Pada gambar 2.1 terdapat sebuah direktori *file* yang berisikan *app.js* yang merupakan *script* utama dari *node.js* yang akan dipanggil ketika perangkat

³<https://expressjs.com/>

lunak dijalankan, lalu terdapat *folder bin* yang digunakan untuk menampung *cache* dari *web application* yang dibangun. Lalu ada *package.json* yang digunakan untuk menyimpan semua nama *library* yang dipakai perangkat lunak. Lalu ada *folder public* yang berfungsi untuk menampung *javascript* dan juga *stylesheet* di dalam perangkat lunak. Terdapat *folder routes* yang berfungsi sebagai *folder* yang menyimpan *script* yang akan digunakan untuk setiap rute yang dipanggil ketika perangkat lunak dijalankan. Lalu ada *folder view* yang merupakan kumpulan dari kode program yang mengurus pada bagian tampilan depan setiap rute dan halaman dari perangkat lunak ketika sudah dijalankan.

- *Routing*

Fungsi ini berguna untuk memudahkan *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

⁴ *Handlebars* adalah modul dari *Node.js* yang membantu 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 bagian dari *view*.

2.3.4 Simple OAuth2

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

- Alur kode otorisasi
Alur ini dipakai pada aplikasi yang dapat menyimpan data secara persisten.
- *Password Credentials*
Alur ini digunakan pada saat alur sebelumnya tidak dapat dilakukan atau sedang dalam tahap pengembangan.
- *Client Credentials*
Client hanya bisa mendapatkan *access token* dengan cara mengirimkan kredensial yang dimiliki oleh *client*.

Untuk bisa menggunakan modul ini, dibutuhkan persyaratan yaitu menggunakan *Node* minimal 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:

```
$ npm install --save simple-oauth2
```

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

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

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

- ***client*** - objek *parameter* wajib yang memiliki properti:
 - ***id*** - Parameter ini merupakan *parameter* yang dibutuhkan dan diisi dengan menggunakan *client id* dari aplikasi yang sudah didaftarkan.
 - ***secret*** - Parameter ini merupakan *parameter* yang dibutuhkan dan diisi dengan menggunakan *client secret* dari aplikasi yang sudah didaftarkan.
 - ***secretParamName*** - Merupakan nama *parameter* yang digunakan untuk mengirimkan *client secret* yang memiliki nilai *default* yaitu *client_secret* dari aplikasi yang sudah didaftarkan.
 - ***idParamName*** - Merupakan nama *parameter* yang digunakan untuk mengirimkan *client id* yang memiliki nilai *default* yaitu *client_id* dari aplikasi yang sudah didaftarkan.
- ***auth*** - objek *parameter* wajib yang memiliki properti:
 - ***tokenHost*** - *String* yang digunakan untuk mengatur *host* untuk meminta *token* dan merupakan *parameter* yang dibutuhkan.
 - ***tokenPath*** - *String* yang menunjukkan jalur untuk meminta *access token* yang memiliki nilai *default* yaitu ke */oauth/token*.
 - ***revokePath*** - *String* yang menunjukkan jalur untuk mencabut *access token* yang memiliki nilai *default* yaitu ke */oauth/revoke*
 - ***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*** - properti yang 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*.

```
const credentials = {
  client: {
    id: <client-id>,
    secret: <client-secret>,
  },
  auth: {
    tokenHost: 'https://login.microsoftonline.com',
    authorizePath: 'common/oauth2/v2.0/authorize',
  }
}
```

```

        tokenPath: 'common/oauth2/v2.0/token'
    }
};

const oauth2 = require('simple-oauth2').create(credentials);

```

Alur OAuth2 yang Didukung

Adapun modul ini mendukung beberapa arus *OAuth2* yaitu seperti:

- *Authorization Code Flow*

Authorization Code Flow ini 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 dilanjutkan dengan langkah kedua yaitu pengguna mengirimkan *POST request* yang berisi *authorization code* bersamaan dengan *client secret* ke *oauth server* untuk mendapatkan *access token*.

- *Password Credentials Flow*

Alur ini menggunakan *username* serta *password* untuk bisa mendapatkan *access token* yang dibutuhkan oleh pengguna. Alur ini biasanya digunakan saat pengguna dengan aplikasi atau sistem operasi dari komputer sudah sangat terpercaya sehingga keamanan *username* dan *password* bisa terjaga. Cara ini biasa dipakai untuk melakukan tes aplikasi secara cepat.

- *Client Credentials Flow*

Alur ini cocok untuk pengguna yang melakukan *request* akses ke sumber yang dibawah kontrol dari pengguna itu sendiri.

Helpers

Modul ini juga menyediakan sebuah objek pembantu untuk mempermudah proses dari *OAuth2* yaitu:

- *Access Token object*

Saat *token* kadaluarsa, diperlukan *refresh token* untuk bisa meminta ulang *access token* yang baru. *Simple OAuth2* menyediakan sebuah kelas *AccessToken* yang memiliki beragam fungsi yang salah satunya membantu mempermudah untuk meminta ulang *access token* saat *token* sudah kadaluarsa. Inisialisasi dari kelas ini seperti

```

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

```

2.3.5 jsonwebtoken

⁶ *Jsonwebtoken* merupakan implementasi dari *JSON Web Tokens* yang dikembangkan terhadap *draft-ietf-oauth-json-web-token-08*. *Jsonwebtoken* ini memiliki beberapa fungsi yaitu:

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

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

Fungsi ini bisa berjalan secara *asynchronous* jika parameter *callback* diisi. *Callback* dipanggil dengan *err* atau dengan *JWT*. Tetapi fungsi ini juga bisa berjalan secara *synchronous* dan mengembalikan *Json Web Token* sebagai sebuah *string*.

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

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

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

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

Token yang dipakai pada fungsi *jwt.verify* dan *jwt.decode* adalah *token id* yang didapat dari proses *OAuth* saat melakukan *request access token*. *Token id* biasanya terdiri dari 3 buah komponen penyusun yaitu ada *header*, *payload*, dan juga *signature*. Berbeda dengan *session id* biasa saat pengguna melakukan *login*, *token id* memiliki bersifat mandiri yang dalam artian memiliki informasi yang lebih kompleks daripada *session id*, sedangkan session id biasanya hanyalah berisi *string* yang panjang dan *random* serta tidak memiliki informasi di dalamnya yang cara kerjanya harus melihat kembali ke tempat penyimpanan data dengan nomor referensi yang adalah *session id*. *Payload* di dalam *token id* menyimpan seluruh informasi mengenai pengguna, sedangkan *signature* berisi status dari *token* tersebut yang bernilai *valid* atau tidaknya.

2.3.6 Isomorphic Fetch

⁷ *Isomorphic Fetch* adalah sebuah *polyfill* *window.fetch* untuk digunakan di *node* dan *browsify*. *Polyfill* ini memungkinkan *window.fetch* untuk *javascript engine* yang tidak mendukungnya secara *native*.

2.3.7 Microsoft Graph Client Library

⁸ *Microsoft Graph Client Library* adalah sebuah modul yang disediakan oleh *Microsoft* untuk perangkat lunak yang akan menggunakan *API* ke *Microsoft* bisa lebih mudah. Modul ini hanya sebagai pembungkus dari semua fungsi *API* yang disediakan oleh *Microsoft*. Untuk bisa memakai modul ini, maka harus menginisialisasi terlebih dahulu sebuah objek inisialisasi dari modul ini dengan menggunakan *access token* yang didapat saat melakukan *request* awal. Setelah melakukan inisialisasi, maka setiap *API* dapat diakses dengan menjalankan kode seperti:

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

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

2.3.8 Slack Web API

⁹ Sama seperti *Microsoft Graph Client Library*, *Slack Web API* ini juga adalah modul yang membungkus *method-method* yang disediakan oleh *Slack* agar untuk mengakses *method* bisa lebih mudah. Cara menggunakannya adalah dengan menginisialisasi terlebih dahulu modul yang akan

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

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

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

dipakai sebagai kelas, lalu menginisialisasi kelas ke sebuah *variable* yang diisikan dengan *parameter slack access token* yang didapat dari *request*. Setelah itu, jalankan *variable* tersebut dengan memanggil *scope* yang dipakai seperti contoh pada kode:

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

const result = await web.users.profile.set({
  "profile": {
    "status_text": "In A Meeting",
    "status_emoji": ":no_entry:"
  }
});
```

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

2.4 Heroku

Heroku adalah *cloud platform* yang menampung program agar bisa dibangun, dan dijalankan di *cloud*.^[4] *Heroku* mendukung beberapa bahasa pemrograman yaitu: *Ruby*, *Node.js*, *Java*, *Python*, *Clojure*, *Scala*, *Go*, dan *PHP*.

2.4.1 Dyno

Dyno adalah sebuah wadah berbasis *Unix* yang disediakan oleh *Heroku*. *Dyno* disini dikategorikan ke 3 jenis *dyno* yaitu:

- *Web Dyno*

Web dyno adalah *dyno* yang berjalan pada tipe proses *web*. *Web dyno* adalah satu-satunya *dyno* yang bisa menerima *HTTP* dari *router heroku*.

- *Worker Dyno*

Worker dyno adalah *dyno* yang berjalan selain pada tipe proses *web*. *Worker dyno* biasa digunakan untuk pekerjaan pada latar belakang, sistem antrian, dan pekerjaan berjangka waktu tertentu.

- *One-off Dyno*

One-off dyno adalah *dyno* yang bersifat sementara dan contoh penggunaannya adalah seperti saat migrasi basis data. *Dyno* ini dijalankan menggunakan *command shell*.

Heroku menyediakan beberapa paket *dyno* yang mempengaruhi karakteristik dan juga cara kerja *dyno* seperti:

- *Free*

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

- *Hobby*

Jumlah *dyno* yang didapat untuk setiap *process type* adalah 1. Jumlah *dyno hours* di dalam paket ini tidak dibatasi tetapi setiap *dyno hours* yang dipakai akan dikenakan biaya sebesar \$7 per jam per bulannya.

- *Standard*

Jumlah *dyno* yang didapat untuk setiap *process type* adalah sebanyak 10. Jumlah *dyno* yang bisa dijalankan secara bersamaan adalah sebanyak 100 pada satu aplikasi. Harga dari *dyno hours* berkisar antara \$25 sampai \$500.

- *Performance*

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

Pada *heroku* terdapat banyak fitur penyokong untuk menjalankan aplikasi seperti basis data, sistem antrean, layanan *email*, dan lain-lain yang disebut dengan *add-ons*. Daftar dari *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:

2.4.2 Heroku Scheduler

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

```
\$ heroku addons:create scheduler:standard
```

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

2.4.3 Heroku Postgres

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

- Melakukan pemasangan modul pg sebagai *dependency* dengan cara menjalankan *command line* seperti:

```
\$ npm install pg
```

- Melakukan koneksi ke *url* dari basis data yang disediakan oleh *Heroku* saat perangkat lunak dijalankan. Contoh dari kode programnya seperti:

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

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

client.connect();
```

```
client.query('SELECT table_schema,table_name FROM
information_schema.tables;', (err, res) => {
  if (err) throw err;
  for (let row of res.rows) {
    console.log(JSON.stringify(row));
  }
  client.end();
});
```


BAB 3

ANALISIS

Pada bab ini akan dijelaskan mengenai analisis masalah yang dihadapi perangkat lunak dimulai dari *overview* dari perangkat lunak ini lalu dilanjutkan ke langkah bagaimana cara mengganti status pada aplikasi *Slack* saat ada *event* yang tercatat di aplikasi *Outlook Calendar* sedang berjalan. Lalu akan dilanjutkan dengan memperdalam analisis dari kedua buah aplikasi yang diintegrasikan, serta analisis mengenai *heroku* yang digunakan sebagai sarana untuk menyimpan dan menjalankan perangkat lunak ini secara berkala dengan menggunakan *add-ons heroku scheduler*. Lalu analisis akan dilanjutkan dengan melakukan analisis mengenai kasus khusus.

3.1 Analisis Perangkat Lunak

Perangkat lunak yang dibangun akan dibagi menjadi 2 bagian yaitu ada bagian yang akan berinteraksi secara langsung kepada pengguna yang memiliki tujuan untuk mendapatkan data yang dibutuhkan dari pengguna seperti *access token* dan *refresh token*, serta ada bagian perangkat lunak yang akan dijalankan secara berkala yang bertugas untuk memeriksa status *access token*, memeriksa *event*, dan mengganti status. Untuk penjelasan analisis ini, akan dijelaskan pada subbab 3.1.1.

3.1.1 Analisis Diagram Alir Sistem

Pada bagian ini akan menjelaskan mengenai aliran secara garis besar mengenai pekerjaan yang akan dikerjakan oleh bagian masing-masing perangkat lunak.

- **Bagian perangkat lunak yang berinteraksi dengan pengguna**

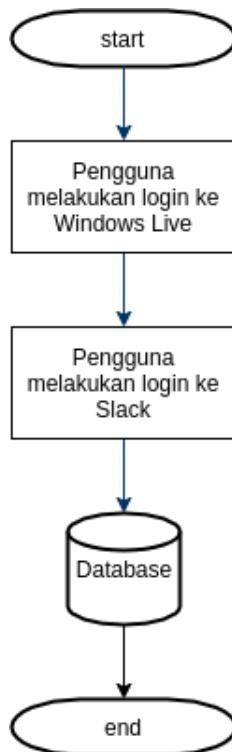
Diagram alir pada gambar 3.1 menunjukkan aliran proses yang terjadi pada bagian perangkat lunak yang berinteraksi dengan pengguna, berikut penjelasan dari aliran proses tersebut:

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

- **Bagian perangkat lunak yang dijalankan secara berkala**

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

- Pertama-tama perangkat lunak akan mengambil data dari dalam basis data.
- Lalu terdapat proses untuk melakukan pengecekan untuk *access token* dari *Microsoft* dikarenakan *access token* ini memiliki waktu kadaluarsa.



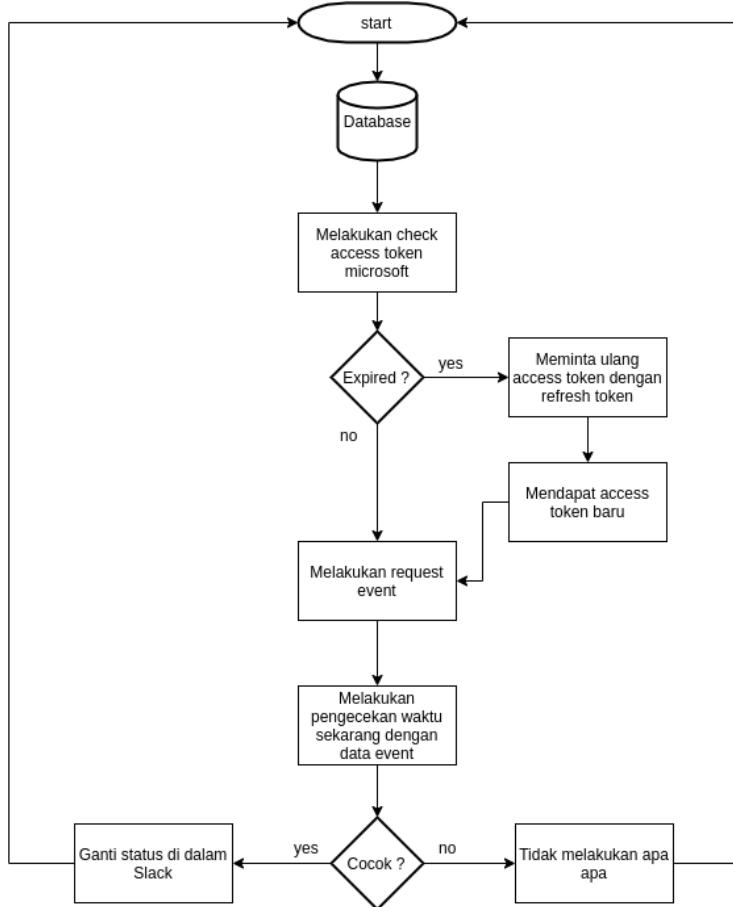
Gambar 3.1: Diagram alir sistem pada bagian perangkat lunak yang berinteraksi dengan *user*.

- Jika *access token* sudah kadaluarsa, maka meminta ulang *access token* dengan menggunakan *refresh token*.
- Jika *access token* belum *expired*, maka dilanjutkan dengan meminta data mengenai *event* yang ada di *Outlook Calendar*.
- Jika setelah melakukan permintaan ulang *access token* dan sudah mendapatkan *access token* yang baru, maka meminta data mengenai *event* yang ada di *Outlook Calendar*.
- Lalu setelah mendapatkan data *event*, maka melakukan pengecekan data *event* dengan waktu sekarang. Jika waktu sekarang beririsan dengan *event* yang sedang berjalan, maka perangkat 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*.

Untuk bisa mendapatkan data *event* yang disimpan pada aplikasi *Outlook Calendar*, dibutuhkan analisis mengenai *Microsoft Graph* seperti yang dijelaskan pada subbab 3.1.2.

3.1.2 Analisis *Microsoft Graph API*

Pada analisis bagian ini, dilakukan analisis mengenai *API* yang telah disediakan oleh *Microsoft Graph API* yang akan digunakan untuk mengambil data acara / *event* yang dibutuhkan oleh perangkat lunak yang akan dibangun. Akan ada beberapa langkah yang harus dijalankan untuk berhasil mencapai tujuan dari perangkat lunak ini. Analisis dari setiap langkah akan dijelaskan pada subbab 3.1.2 sampai subbab 3.1.2.



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

Analisis Mendapatkan *Authorization Code*

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

¹<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 <i>url</i> 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 <i>token</i> 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.

Pada *parameter scope*, dapat diisi dengan nilai *offline_access* yang menjadikan aplikasi mendapatkan *response* berupa *refresh token* yang berguna untuk mendapatkan *access token* yang baru saat yang lama sudah kadaluarsa. Contoh *request* yang dikirimkan akan seperti yang terdapat 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
```

Dari *request* seperti contoh table 3.2, akan menghasilkan contoh *response* seperti yang terdapat 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.

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*.

Analisis Mendapatkan Access Token

Setelah mendapatkan *authorization code*, langkah selanjutnya yang harus dijalankan sebelum bisa memanggil *method API* yang dibutuhkan adalah dengan mendapatkan *access token*. Yang diperlukan untuk bisa mendapatkan *access token*, maka aplikasi yang dibuat membutuhkan *authorization code* yang diterima di langkah sebelumnya dan mengirimkan *post request* kepada *endpoint /token*.

Untuk mengirimkan *post request*, diperlukan *request body* yang memiliki elemen-elemen seperti yang terdapat di contoh table 3.5. Adapun penjelasan dari setiap *parameter* yang terdapat di dalam *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 <i>domain</i> 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	<i>Authorization code</i> 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>	<i>Application secret</i> yang dibuat saat mendaftarkan aplikasi di portal registrasi untuk aplikasi yang didaftarkan.

Dari contoh *request* yang dilakukan pada table 3.5, maka akan dihasilkan contoh *token* seperti 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 <i>token</i> . Satu-satunya jenis <i>token</i> yang didukung oleh Azure AD adalah <i>Bearer</i> .
<i>scope</i>	Nilai <i>scope</i> yang <i>valid</i> untuk <i>access_token</i> yang diberikan.
<i>expires_in</i>	Lamanya <i>access token</i> akan berlaku(dalam detik).
<i>access_token</i>	<i>Access token</i> yang diminta. Dengan memakai ini, maka aplikasi bisa memanggil <i>Microsoft Graph</i> .
<i>refresh_token</i>	<i>Refresh token</i> ini berguna untuk meminta kembali <i>access token</i> setelah <i>access token</i> itu berakhir. <i>Refresh token</i> memiliki umur yang panjang dan dapat digunakan untuk mempertahankan akses ke <i>source</i> .

Analisis Menggunakan *Access Token* untuk memanggil *Microsoft Graph*

Setelah mendapatkan *access token*, panggilan ke *Microsoft Graph* pun bisa dilakukan dengan syarat menyertakan *access token* di *authorization header* di setiap *request* yang dikirim. Pada table 3.9 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 ...
0X2tnSQUEANnSPHY0gKcgw
Host: graph.microsoft.com
```

Jika *request* yang dikirimkan berhasil, maka akan mendapatkan *response* yang akan terlihat 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"
}

```

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* 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* dengan menggunakan *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	<i>Refresh token</i> yang didapat saat melakukan <i>request token</i> 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>	<i>Application secret</i> yang dibuat saat mendaftarkan aplikasi di portal registrasi untuk aplikasi yang didaftarkan.

Jika *request* ini berhasil, maka akan mengembalikan *response* seperti pada table 3.13 yang memiliki keterangan *parameter* yang dikembalikannya seperti pada table 3.14.

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

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1Ni
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>	<i>Access token</i> yang diminta. Dengan memakai ini, maka aplikasi bisa memanggil <i>Microsoft Graph</i> .
<i>token_type</i>	Menunjukkan nilai dari <i>token</i> . Satu-satunya jenis <i>token</i> yang didukung oleh <i>Azure AD</i> adalah <i>Bearer</i> .
<i>expires_in</i>	Lamanya <i>access token</i> akan berlaku(dalam detik).
<i>scope</i>	Nilai <i>scope</i> yang valid untuk <i>access token</i> yang diberikan.
<i>refresh_token</i>	<i>Refresh token</i> ini berguna untuk meminta kembali <i>access token</i> setelah <i>access token</i> itu berakhir. <i>Refresh token</i> memiliki umur yang panjang dan dapat digunakan untuk mempertahankan akses ke <i>source</i> .

Analisis Mendapatkan Data *Events*

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

Tabel 3.15: Tabel contoh *header time zone*

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

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

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

Tabel 3.16: Tabel *parameter header time zone*

Nama	Type	Deskripsi
<i>Authorization</i>	<i>String</i>	<i>Bearer token</i> . 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 <i>response body</i> yang akan dikembalikan. Nilai bisa berupa “ <i>text</i> ” atau “ <i>html</i> ”. Nilai <i>default</i> dari <i>parameter</i> ini adalah <i>html</i> . Bersifat optional.

Request ini mengembalikan *response* berupa *json* objek dari *event resource type*. *Request* ini juga bisa menerima *parameter* \$select yang berbentuk *string query* sebagai *filter* mengenai *field* yang mau diambil dari objek *event*. Dapat dilihat fungsi dari *parameter 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/\\"ZlnW4RIAV06KYYwlrfNZvQAAKGWwbw==\\\"",
      "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"
      },
      "locations": [
        {
          "displayName": "Assembly Hall",
          "locationType": "default"
        }
      ],
      "organizer": {
        "emailAddress": {
          "name": "Samantha Booth",
          "address": "samanthab@a830edad905084922E170
20313.onmicrosoft.com"
        }
      }
    }
  ]
}
```

Setelah melakukan analisis kepada aplikasi *Outlook Calendar* melalui *Microsoft Graph API* yang dipakai, maka sekarang akan dijelaskan analisis mengenai bagian dari perangkat lunak yang berinteraksi dengan aplikasi *Slack* pada subbab 3.1.3.

3.1.3 Analisis *Slack*

Pada bagian ini akan dijelaskan mengenai analisis pada aplikasi *Slack*. Yang akan dibahas antara lain adalah analisis mengenai Status yang terdapat pada aplikasi *Slack* yang akan dijelaskan pada bagian 3.1.3. Lalu akan dilanjutkan dengan analisis cara menggunakan *API* yang sudah disediakan oleh aplikasi *Slack* untuk perangkat lunak bisa mengakses kepada aplikasi *Slack* yang akan dibahas pada bagian 3.1.3

Analisis Status pada *Slack*

Status pada aplikasi *Slack* adalah fitur yang menggambarkan kondisi dari pengguna saat itu dan fitur itu tergabung dalam profil dari pengguna. Status di dalam aplikasi *Slack* sendiri terdiri dari 2 bagian yaitu sebuah simbol yang menggambarkan kondisi dari status, serta sebuah *text* yang memperjelas keadaan dari status yang dialami oleh pengguna. Umumnya, *Slack* menyediakan sebuah *field* untuk pengguna bisa mengisi statusnya sesuai dengan keadaan yang dialami oleh pengguna. Ada juga beberapa status yang sudah disediakan oleh aplikasi *Slack*.

Perangkat lunak ini akan mengubah status pengguna dengan menggunakan simbol *no entry sign*, dan juga *text* yang bertuliskan “*In a meeting*”. Hal itu dipilih karena beberapa alasan yaitu:

- Tujuan dari disusunnya perangkat lunak ini adalah untuk meminimalisir pengguna mendapatkan gangguan berupa *chat* dari pengguna lain saat ada *event* yang berlangsung, sehingga cocok untuk memasang status yang menggambarkan bahwa pengguna saat ini sedang tidak bisa diganggu.
- Jika menggunakan pemeriksaan kata-kata di dalam *subject* dari *event*, maka akan muncul beberapa ambigu seperti contohnya jika di dalam *subject* terdapat kata “liburan”, maka perangkat lunak akan mengganti status menjadi “*On vacation*”, dan jika ada kata “Rapat”, maka status akan menjadi “*In a meeting*”, lalu jika *subject* dari *event* adalah “Rapat untuk membahas liburan bersama kerabat kerja”, status akan berganti menjadi “*On vacation*” karena di dalam *subject* mengandung kata liburan, tetapi seharusnya status berubah menjadi “*In a meeting*” karena mengandung kata-kata utama yaitu “Rapat”.
- Jika menggunakan *tag* khusus di dalam *subject event*, maka akan menimbulkan banyak kondisi yang harus di *handle* seperti contohnya bila dalam sebuah *subject event* terdapat *tag* “[Rapat]”, maka status berganti menjadi “*In a meeting*”, dan jika “[Liburan]”, maka status berganti menjadi “*On vacation*”. Jika ada *subject* yang tidak menggunakan *tag*, maka tidak tahu akan masuk ke dalam kategori status yang mana, atau bisa saja statusnya menjadi kosong padahal pengguna lupa untuk mengisi *subject* pada *event* dengan menggunakan *tag*. Lalu jika ada kasus *tag* yang belum dikenali oleh perangkat lunak, maka status yang dipasang juga menjadi kosong.

Alasan-alasan diatas yang memperkuat pemilihan status yang akan dipasang oleh perangkat lunak ini secara otomatis yaitu dengan simbol “*no-entry*”, dan *text* “*In a meeting*”.

Analisis *Slack API*

Pada bagian ini akan dibahas mengenai cara penggunaan *Slack API*. Untuk dapat memakai dan mengakses *method-method* yang disediakan oleh *Slack*, dibutuhkan mendaftarkan aplikasi yang akan dibuat untuk bisa memperoleh *Client ID* yang nantinya dibutuhkan untuk bisa mendapatkan *authorization code* serta *access token*. Sama seperti di *Microsoft Graph API*, di dalam *Slack API*

access token dibutuhkan sebagai otorisasi untuk bisa mengakses *method-method* yang disediakan oleh *Slack*. Jika tidak memiliki *access token*, maka seluruh *method* yang dicoba direquest tidak akan mengembalikan hasil dan dianggap sebagai *request* yang tidak *valid*.

Pada sesi ini akan dibutuhkan *method* dari *Slack API* yang bisa mengubah status yang terdapat pada bagian *user.profile*. Dari informasi yang disediakan oleh *Slack* secara *online*, bahwa status tergabung dalam *user.profile*, maka untuk memenuhi dari kebutuhan di sisi ini dicoba menggunakan *method-method* yang bisa untuk mengakses kepada objek *user.profile*. Karena di sisi ini akan mengubah nilai dari status, asumsi sementara dari *method* yang bisa dipakai dari sisi ini adalah *method users.profile.set* yang akan berguna untuk mengubah isi dari profil pengguna yang di dalamnya terdapat status.

Setelah analisis mengenai *Slack*, maka pada bagian 3.1.4 akan membahas mengenai *Heroku*.

3.1.4 Analisis *Heroku*

Pada bagian ini akan membahas mengenai *Heroku*. *Heroku* adalah *platform* yang dipakai untuk menyimpan perangkat lunak yang dibangun. Layaknya sebuah *server*, *heroku* akan menyimpan perangkat lunak dan jika berbasis *web*, maka *heroku* akan memberikan *subdomain* agar perangkat lunak yang disimpan bisa diakses dan dijalankan secara langsung dengan mengakses *subdomain* yang diberikan. Untuk *heroku* yang berpaket gratis, maka akan ada keterbatasan seperti, perangkat lunak akan masuk ke dalam keadaan *sleep* setiap 30 menit sekali jika tidak ada akses selama 30 menit kepada perangkat lunak dan perangkat lunak akan kembali aktif setelah ada akses kembali ke perangkat lunak dengan adanya sedikit keterlambatan saat memasuki keadaan aktif kembali. Lalu batasan lainnya adalah adanya batasan *dyno hours* yang didapatkan yaitu sebesar 1000 jam jika akun sudah terverifikasi. Pada *heroku* juga terdapat banyak *add-ons* yang membantu fungsi dari perangkat lunak yang dibangun agar berjalan dengan baik, seperti basis data dan juga *scheduler* yang berfungsi untuk menjalankan bagian dari perangkat lunak secara berkala. Untuk analisis dari *add-ons*, akan dibahas pada bagian 3.1.4 dan juga 3.1.4.

Analisis *Postgres*

Postgres adalah basis data yang disediakan sebagai *add-ons* pada *heroku*. Untuk bisa mengaksesnya, bisa menggunakan *Command Line Interface (CLI)*, atau juga bisa dibantu dengan menggunakan *pgAdmin4*. Dengan menggunakan *pgAdmin4* yang terkoneksi dengan *url* yang diberikan oleh *heroku* untuk mengakses *postgres*, maka basis data yang tersimpan di *heroku* bisa secara langsung terakses melalui *pgAdmin4*. Cara melakukan koneksi dari *pgAdmin4* ke *url* yang sudah disediakan oleh *heroku* adalah dengan membuat koneksi yang diisi dengan kredensial yang diberikan oleh *heroku* pada halaman seperti pada gambar 3.3.

Setelah memasukkan data kredensial yang diberikan oleh *heroku postgres* ke *pgAdmin4* dengan benar, maka koneksi akan tersambung ke basis data yang dibuat di *postgres*. Selanjutnya dari *pgAdmin4*, kegiatan *create*, *update*, *insert*, dan *delete* terhadap basis data bisa dilakukan dengan cara penggunaan *SQL syntax* seperti biasanya. Pada *postgres*, akan dibutuhkan sebuah tabel yang akan menampung data-data yaitu:

- ***Microsoft Access Token***

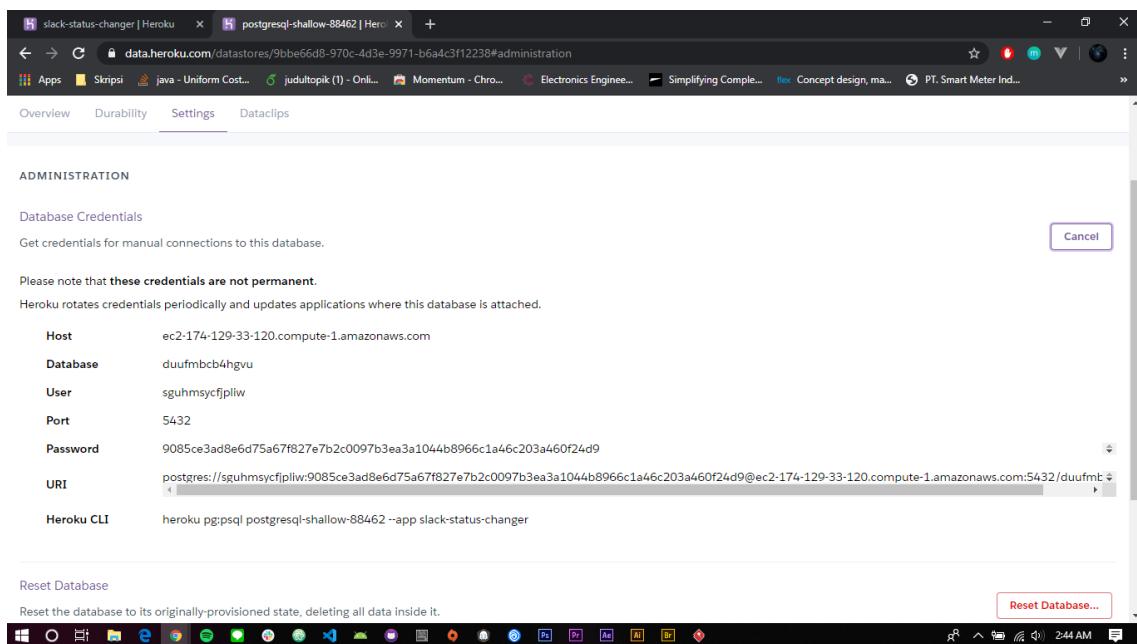
Microsoft access token adalah *access token* dari pengguna yang didapat dari aplikasi *Outlook Calendar*.

- ***Slack Access Token***

Slack access token adalah *access token* dari pengguna yang didapat dari aplikasi *Slack*.

- ***Username***

Username ini adalah penanda dari setiap pengguna yang akan bersifat identik.



Gambar 3.3: Data kredensial yang diberikan oleh *heroku postgres*.

- ***Microsoft Refresh Token***

Microsoft refresh token adalah *refresh token* dari *Outlook Calendar* yang berfungsi untuk meminta ulang *access token* jika *access token* sudah kadaluarsa.

- ***Microsoft Access Token Expiration Time***

Microsoft access token expiration time ini adalah data yang mencatat masa kadaluarsa dari *access token Outlook Calendar*.

Setelah menganalisis *postgres* yang digunakan sebagai basis data untuk perangkat lunak ini, maka akan dilanjutkan melakukan analisis *heroku scheduler* yang berperan sebagai pengeksekusi perangkat lunak secara berkala pada bagian 3.1.4

Analisis *Heroku Scheduler*

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

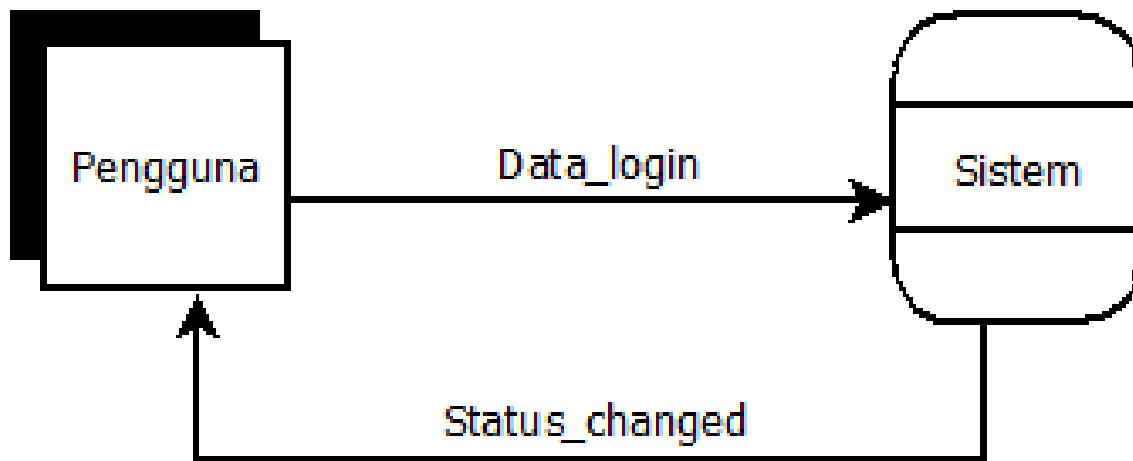
```
$ heroku addons:open scheduler
```

Setelah halaman baru pada *browser* muncul, pilih tombol untuk menambahkan *job* di *scheduler* dengan menekan tombol “*Add Job*”. Pada *Heroku Scheduler* tersedia interval untuk menjalankan sebuah *job* yaitu setiap 10 menit sekali, atau setiap jam pada menit ke, atau setiap hari pada jam ke. Interval paling pendek untuk setiap kali iterasi yang disediakan oleh *heroku scheduler* adalah setiap 10 menit sekali.

Untuk memperkecil perangkat lunak melewatkkan *event* yang tercatat di *Outlook Calendar* dan memaksimalkan iterasi dari *heroku scheduler*, maka bagian dari perangkat lunak yang melakukan sinkronisasi akan dijalankan setiap 10 menit sekali.

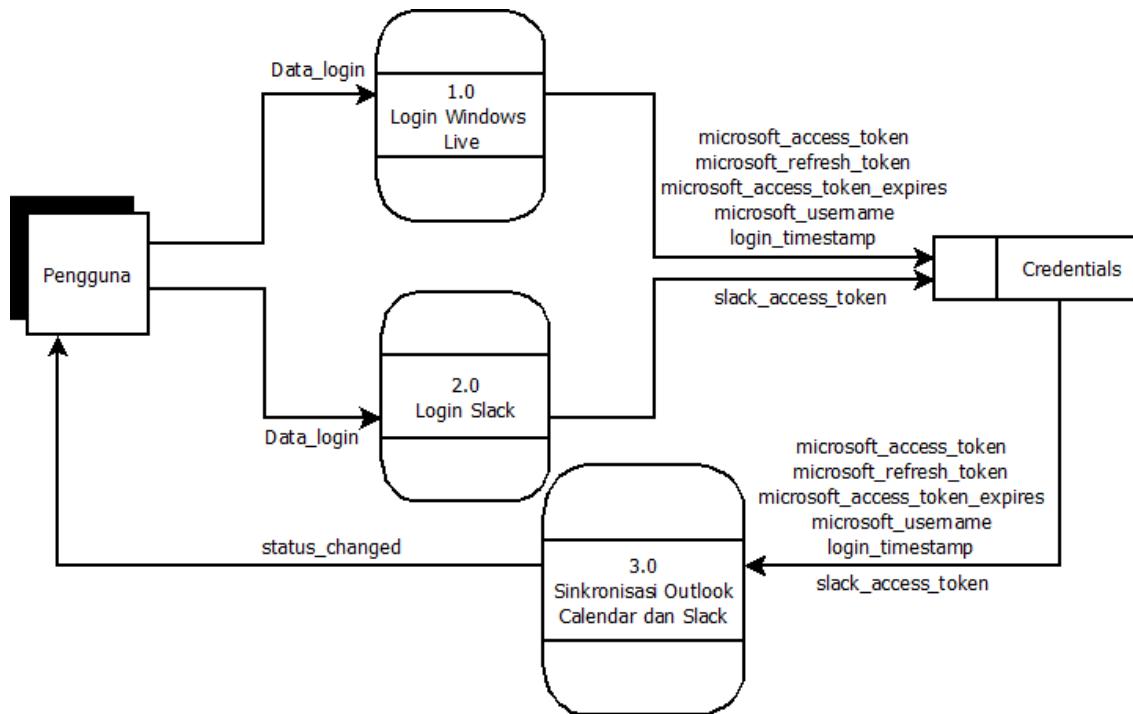
3.1.5 Analisis *Data Flow Diagram*

Pada bagian ini akan menjelaskan mengenai aliran data yang terjadi pada proses di dalam perangkat lunak ini.



Gambar 3.4: *Data Flow Diagram level 0.*

Diagram alir data pada gambar 3.4 menunjukkan aliran data pada sistem yang menerima *input* dari pengguna berupa *data_login* yang berisi data dari pengguna untuk melakukan *login* seperti *username* dan *password* yang dimiliki oleh pengguna. Lalu di dalam sistem, ada berbagai proses yang dijabarkan pada gambar 3.5.



Gambar 3.5: *Data Flow Diagram.*

Diagram alir data pada gambar 3.5 menunjukkan aliran data pada sistem, berikut 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*, *microsoft_username*, *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*, *microsoft_username*, dan *login_timestamp* yang disimpan ke tabel *Credentials* sebagai *data store*.

- **Proses melakukan *login* ke *Slack***

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

Output : *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 *slack_access_token* yang disimpan ke tabel *Credentials* sebagai *data store*.

- **Proses melakukan sinkronisasi antara *Outlook Calendar* dan *Slack***

Input : *microsoft_access_token*, *microsoft_access_token_expires*, *microsoft_refresh_token*, *microsoft_username*, *login_timestamp*, dan *slack_access_token*

Output : Status di *Slack* yang terubah jika memenuhi kondisi. 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.

3.2 Analisis Kasus

Pada perangkat lunak ini, akan ada kasus diluar pola dari kasus normalnya yaitu adanya kasus dimana akan ada jadwal yang bertabrakan contohnya misalkan ada *event* yang berjalan pada pukul 08.00 - 10.00, lalu ada *event* kedua yang berjalan pada pukul 09.00 - 11.00. Pada bagian ini akan dijelaskan cara penanganan perangkat lunak untuk kasus pada waktu yang bertabrakan.

Data *event* akan didapatkan secara berurutan mulai dari *event* yang sudah lalu, hingga *event* yang akan datang. Untuk mengatasi *event* yang waktunya bertabrakan, maka pemeriksaan waktu *event* dengan waktu sekarang akan memeriksa seluruh *event* yang didapatkan dari *Outlook Calendar*, dan jika ada *event* yang sedang berjalan, akan mengganti status, lalu tetap melanjutkan pemeriksaan ke *event* selanjutnya agar kasus *event* yang waktunya bertabrakan tetap bisa ditangani dan hasilnya status akan tetap menjadi “*In a meeting*” hingga *event* terakhir yang bertabrakan berakhir.

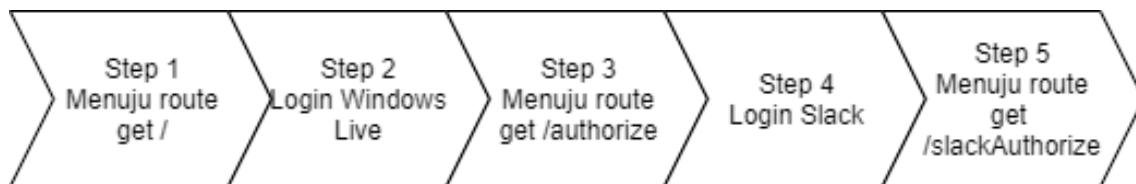
BAB 4

PERANCANGAN

Pada bab ini akan dibahas perancangan dari perangkat lunak mulai dari perancangan antarmuka, dan perancangan untuk algoritma setiap *route* yang dipakai.

4.1 Perancangan Routing Handler

Pada perangkat lunak ini disusun dengan beberapa *routing handler*. *Routing handler* mengatur setiap halaman yang harus ditampilkan. Pada perangkat lunak ini, terdapat 4 buah *route* dan ada sebuah alur dari antara 1 *route* ke *route* lainnya. Alur tersebut dapat digambarkan pada gambar 4.1.



Gambar 4.1: Alur antar *route*.

Pada gambar 4.1, dapat dijelaskan bahwa *route* `get /`, *route* `get /authorize`, dan juga *route* `get /slackAuthorize` menyatu dalam satu alur. Proses yang akan dijalani oleh pengguna saat menggunakan perangkat lunak ini adalah seperti yang digambarkan oleh gambar 4.1, yaitu pertama pengguna akan menuju kepada *route* `get /`, lalu dari langkah pertama, perangkat lunak akan mengarahkan pengguna untuk melakukan *login* ke *Windows Live*. Setelah melakukan *login* ke *Windows Live*, maka akan kembali ke perangkat lunak pada bagian *route* `get /authorize`. Lalu dari *route* `get /authorize`, perangkat lunak mengarahkan pengguna untuk melakukan *login* ke *Slack*. Setelah berhasil melakukan *login* ke *Slack*, maka akan menuju tahap akhir dari perangkat lunak yang berinteraksi dengan pengguna yaitu ke *route* `get /slackAuthorize`.

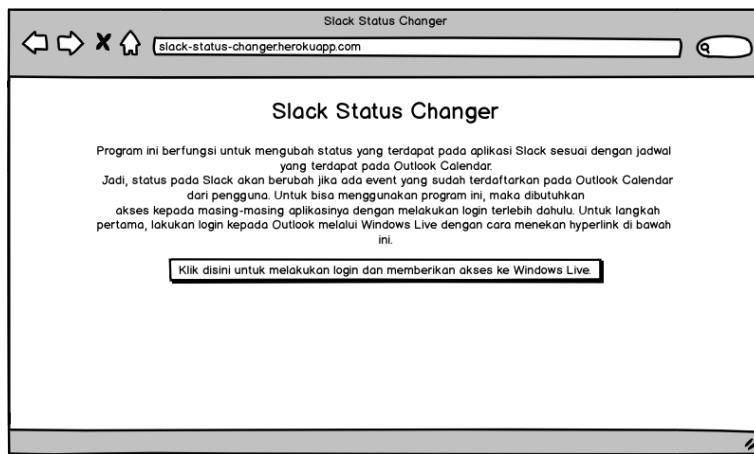
Lalu ada satu lagi *route* yang terdapat pada perangkat lunak ini yaitu *route* `get /statusChanger` yang merupakan *route* yang akan diiterasi setiap 10 menit sekali oleh *heroku scheduler* yang memiliki fungsi utama dari perangkat lunak ini. Untuk penjelasan lebih lanjut mengenai setiap *route* akan dijelaskan pada subbab 4.1.1 sampai 4.1.4.

4.1.1 *Route* `get /`

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

Antarmuka yang dirancang pada *route* ini 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.

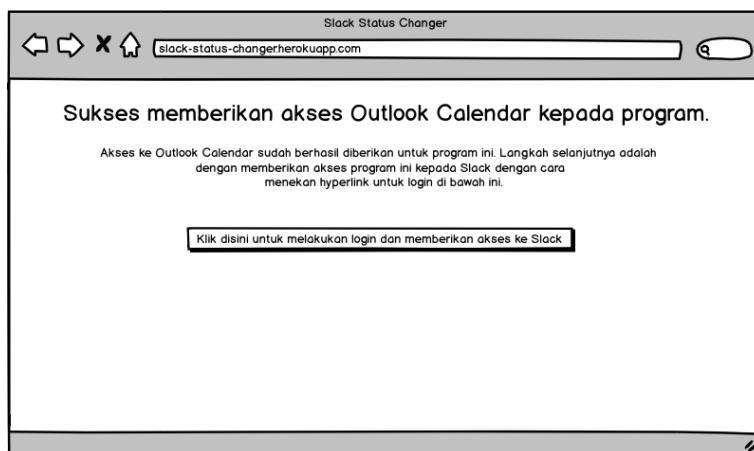


Gambar 4.2: Rancangan antarmuka halaman awal.

4.1.2 *Route get /authorize*

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

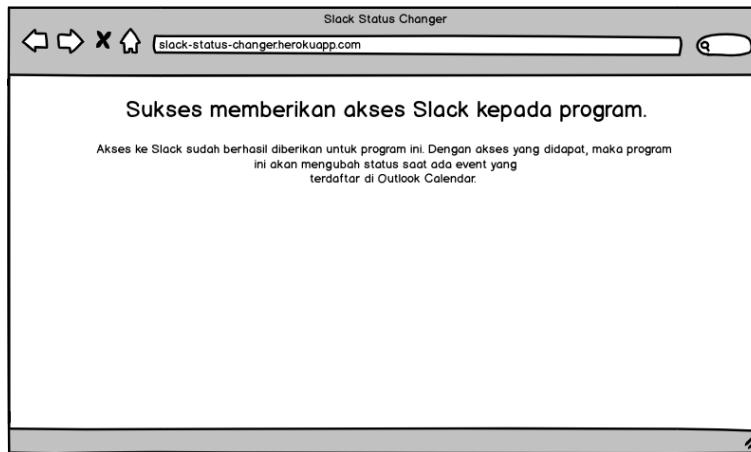
Pada *route* ini, antarmuka yang dirancang seperti pada gambar 4.3. Di dalam perancangan antarmuka *route* ini akan ada sebuah tulisan yang membantu mengarahkan pengguna untuk melakukan langkah selanjutnya agar bisa memakai perangkat lunak ini dengan baik. Lalu di dalam perancangan ini juga terdapat 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*.



Gambar 4.3: Rancangan antarmuka halaman setelah *login Windows Live*.

4.1.3 *Route get /slackAuthorize*

Route ini hanya menampilkan konfirmasi yang didapat dari hasil perekaman data dari *Slack* yang berupa *access token* dari aplikasi tersebut .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 4.4: Rancangan antarmuka halaman setelah *login Slack*.

4.1.4 *Route get /statusChanger*

Route ini yang berfungsi mengeksekusi pengambilan data dari *Outlook Calendar* secara berkala, dan jika ada data *event* yang sesuai dengan waktu sekarang, maka lewat *route* ini juga program ini akan mengganti status pada aplikasi *Slack*. Sebelum melakukan pengambilan data pada *Outlook Calendar*, pada *route* ini juga melakukan pengecekan pada *access token* yang didapat pada saat awal melakukan *login Windows Live*. Jika *access token* yang terekam pada *database* sudah kadaluarsa, maka melalui *route* ini juga program akan meminta *access token* yang baru untuk bisa mengambil data *event* dari *Outlook Calendar*. Adapun pseudocode yang dirancang adalah seperti

Listing 4.1: Pseudocode untuk /statusChanger

```

do connection to database.

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

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

                mendapatkan event dengan memakai access
                token yang baru memakai function getEvent()
            }
            else{

```

```

        mendapatkan event dengan memakai access
        token yang lama memakai function getEvent()
    }
}
})  

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

    for each result{
        if(timestampNow>=startTime&&timestampNow<=endTime){
            panggil function changeStatusSlack()
        }
        else{
            tidak melakukan apa apa.
        }
    }
}  

async function useRefreshToken(refresh_token){
    melakukan request menggunakan refresh token
    menyimpan data yang diperlukan kembali ke dalam tabel
    return newToken
}  

async function changeStatusSlack(slack_access_token, endTime){
    mengirimkan request dengan menggunakan slack access token
    ditambah dengan parameter status_expiration diisi dengan endtime
}

```

Pseudocode tersebut menjelaskan fungsi yang terdapat pada *route get /statusChanger*. Fungsi *getEvent()* berfungsi untuk mengambil data *event* ke aplikasi *Outlook Calendar*, sedangkan fungsi *useRefreshToken()* merupakan fungsi yang meminta kembali *access token* milik pengguna ke aplikasi *Outlook Calendar* jika *access token* sebelumnya sudah kadaluarsa. Lalu ada fungsi *changeStatusSlack()* yang memiliki fungsi untuk mengganti status ke aplikasi *Slack* dengan mengirimkan *request* ke *Slack*.

4.2 Perancangan *Helper*

Helper memiliki tujuan untuk membantu kerja dari masing-masing *route* yang membutuhkan. Prinsip dari kumpulan kode pada *helper* ini adalah kode yang ada di *helper* hanya butuh dipanggil untuk *route* bisa mengakses fungsi dari kode yang ada tanpa harus menuliskan kode yang diperlukan secara berulang-ulang di setiap *route* yang membutuhkan. Adapun *helper* yang dirancang adalah :

auth.js

Pada file *javascript* ini, terdapat kode yang membantu untuk melakukan *request* untuk mendapatkan *access token* dari *Microsoft Graph* maupun dari *Slack*. Cara mendapatkan *access token* di file ini dengan cara menggunakan *library simple oauth2* yang membantu untuk melakukan *request* ke aplikasi yang akan dituju. Selain untuk meminta *akses token*, di dalam file ini juga berisi kode yang

membantu program ini agar *access token* dan data yang dibutuhkan lainnya disimpan di dalam basis data. Pada *helper* ini terdapat konstanta dan juga fungsi-fungsi seperti:

- *Constanta*

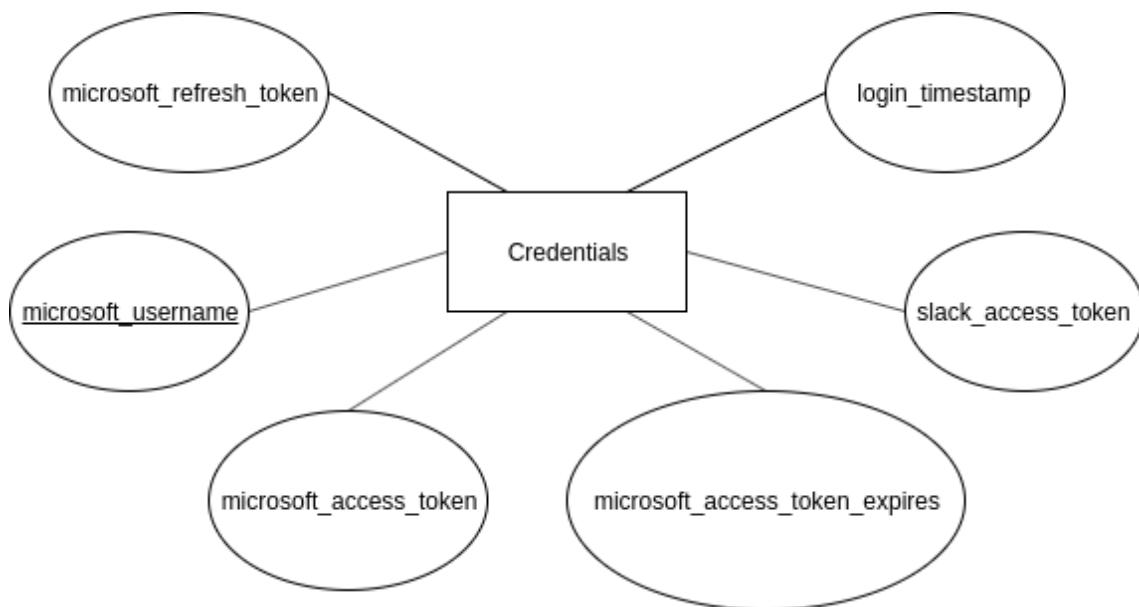
- **{Client}**: Konstanta ini merupakan konstanta yang memanggil *library* dari *postgres*. *Postgres* merupakan jenis basis data yang dipakai di dalam program ini.
- **client**: Konstanta yang menginisialisasi kelas *Client* yang dipanggil dari konstanta sebelumnya. Konstanta ini menginisialisasi kelas *Client* dengan parameter *url* dari basis data yang dipakai dan juga nilai *boolean* untuk *ssl*.
- **credentials**: Konstanta ini bertipe *Object* yang berfungsi sebagai objek yang dibutuhkan *simple oauth2* untuk melakukan *request* kepada *Microsoft Graph*. Objek dari konstanta ini memiliki isi yaitu *client* dan juga *auth*. *Client* memiliki isi data *id* dan juga *secret* yang merupakan *id* dan *secret* yang didapat saat mendaftarkan aplikasi di *azure portal*. Sedangkan *Auth* berisi *tokenHost*, *authorizePath*, dan *tokenPath* yang berisikan data *url* untuk melakukan *request access token* ke *Microsoft Graph*.
- **credentialsSlack**: Konstanta ini bertipe *Object* yang berfungsi sebagai objek yang dibutuhkan *simple oauth2* untuk melakukan *request* kepada *Slack*. Objek dari konstanta ini memiliki isi yaitu *client* dan *auth*. *Client* memiliki isi data *id* dan juga *secret* yang merupakan *id* dan *secret* yang didapat saat mendaftarkan aplikasi di *Slack*. Sedangkan *Auth* berisi *tokenHost*, *authorizePath*, dan *tokenPath* yang berisikan data *url* untuk melakukan *request access token* ke *Slack*.
- **oauth2**: Konstanta yang digunakan untuk menggunakan dan membuat *simple oauth2* yang melakukan koneksi kepada *Microsoft Graph*.
- **oauth2Slack**: Konstanta yang digunakan untuk menggunakan dan membuat *simple oauth2* yang melakukan koneksi kepada *Slack*.
- **jwt**: Konstanta ini untuk memanggil *library jsonwebtoken*.
- **databaseValue**: Konstanta ini berbentuk *Object* yang akan digunakan untuk menampung nilai-nilai yang akan dimasukan ke dalam tabel basis data.

- *Function*

- **getAuthUrl()**: Fungsi ini mengembalikan *url* untuk meminta *authorization code* pada *Microsoft Graph*. *Url* yang dikembalikan sudah lengkap dengan *parameter* yang dibutuhkan untuk meminta *authorization code*. *Url* ini akan membawa pengguna untuk melakukan *login* awal dan meminta izin program ini ke *Microsoft Graph*.
- **getTokenFromCode(auth_code)**: Fungsi ini berfungsi untuk mengubah *authorization code* yang didapat dari proses *login* awal menjadi *access token*. Setelah berhasil mendapatkan *access token*, maka fungsi ini akan membongkar *id_token* yang didapatkan dari kembalian data setelah meminta *access token* dengan menggunakan *jsonwebtoken* untuk mendapatkan data pengguna seperti nama, *email*, dan lain-lain. Setelah berhasil membongkar *id_token*, maka di dalam fungsi ini akan memasukkan beberapa nilai ke dalam konstanta *databaseValue* seperti *microsoft_username* yang didapat dari *preferred_username* hasil dari membongkar *id_token*, *microsoft_access_token* yang didapat dari nilai *access_token* dan merupakan *access token* yang didapatkan dari *request* yang dilakukan, *microsoft_access_token_expires* yang didapat dari nilai *expires_in* dari respon yang didapat, *microsoft_refresh_token* yang didapatkan dari nilai *refresh_token* dari respon yang berfungsi untuk meminta *access token* yang baru ketika yang lama sudah kadaluarsa, dan juga *login_timestamp* yang diambil dari waktu saat pengguna melakukan *login*. Lalu fungsi ini akan mengembalikan nilai berupa *access token* yang didapat.

- **getAuthUrlSlack()**: Fungsi ini mengembalikan *url* untuk meminta *authorization code* pada *Slack*. *Url* yang dikembalikan sudah lengkap dengan *parameter* yang dibutuhkan untuk meminta *authorization code*. *Url* ini akan membawa pengguna untuk melakukan *login* dan memberikan program ini izin ke dalam *Slack*.
- **getTokenFromCodeSlack(auth_code)**: Fungsi ini berfungsi untuk mengubah *authorization code* yang didapat dari *Slack* dengan *access token* yang akan digunakan untuk mengubah status di dalam *Slack*. Melalui fungsi ini, konstanta *databaseValue* akan ditambahkan dengan nilai *slack_access_token* yang diisi dengan nilai *access token* yang didapat dari respon saat meminta *access token*. Setelah mendapatkan *access token* dari *Slack*, maka fungsi ini selanjutnya akan melakukan pemeriksaan pada seluruh *record* yang ada di basis data. Jika untuk *microsoft_username* yang didapatkan sekarang sudah ada, maka fungsi ini hanya akan melakukan *update* pada isi dari *row* basis data tersebut. 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 4.5: 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*.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

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

5.1 Implementasi

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

5.1.1 Lingkungan Pengembangan

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

Spesifikasi Perangkat Keras

- Processor Intel Core i5-8250U CPU @ 1.60GHz x 8
- Graphics Intel UHD Graphics 620 (Kabylake GT2)
- RAM 12 GB
- Harddisk 1000GB HDD

Spesifikasi Perangkat Lunak

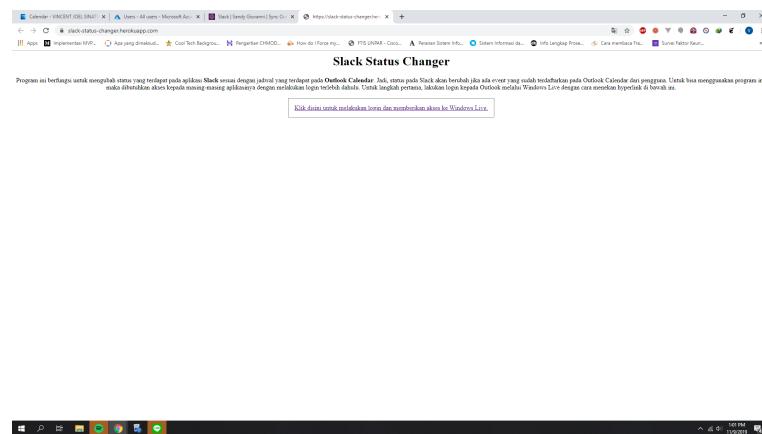
- Sistem Operasi Ubuntu 18.04.3 LTS 64-bit
- Atom
- Node.js version 8.10.0
- NPM version 6.8.0
- pgAdmin4 version 4.13 (Application Mode: Desktop)

5.1.2 Implementasi Basis Data

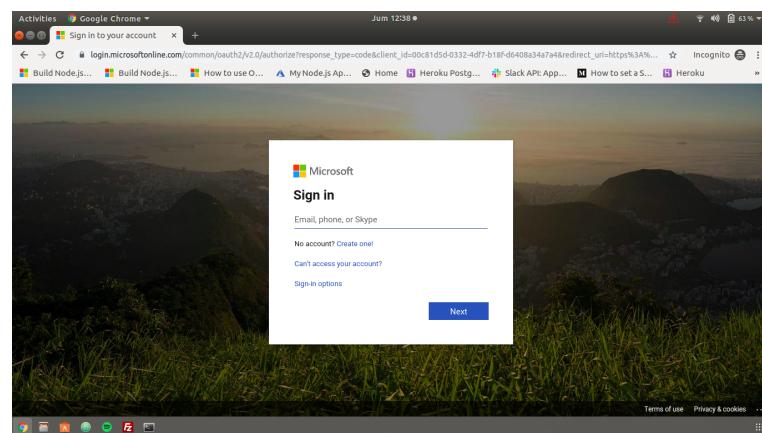
Untuk basis data yang diimplementasikan, tabel yang disediakan hanya ada 1 tabel yang bernama *Credentials*. Tabel basis data ini menampung data-data kredensial dari pengguna yang menjalankan dan mendaftarkan akunnya ke program ini. Tabel ini dibuat dengan bantuan *pgAdmin4* yang melakukan koneksi kepada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

5.1.3 Implementasi Antarmuka

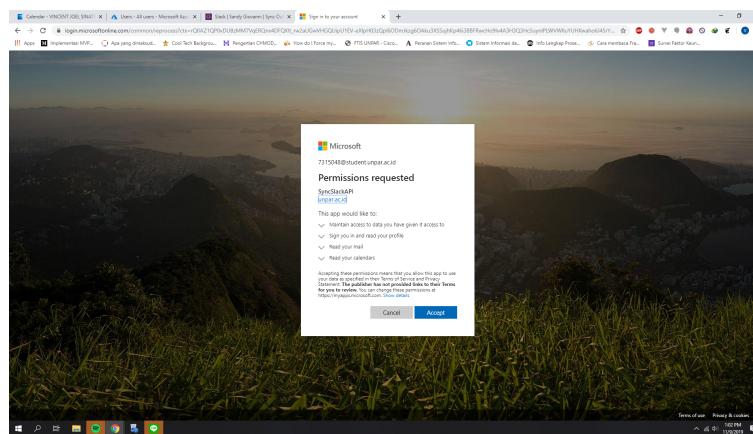
Hasil implementasi dari antarmuka perangkat lunak ini dapat dilihat pada gambar 5.1, gambar 5.2, gambar 5.3, gambar 5.4, gambar 5.5, gambar 5.6, dan gambar 5.7.



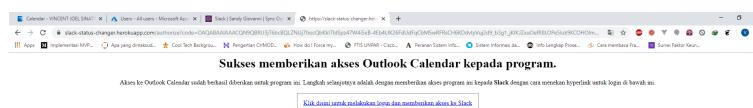
Gambar 5.1: Antarmuka halaman awal.



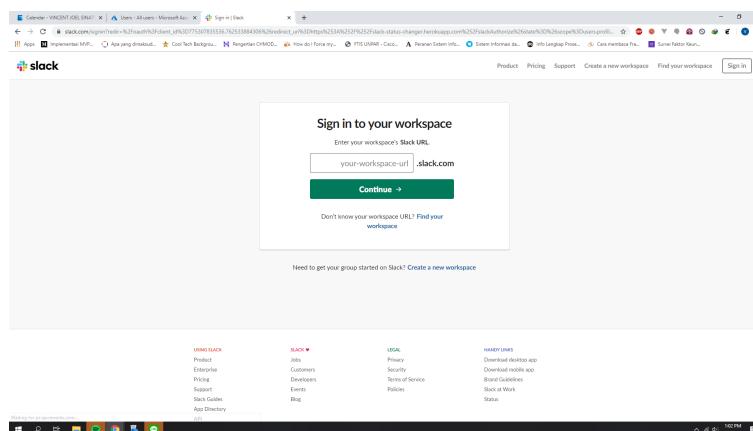
Gambar 5.2: Antarmuka untuk login Windows Live.



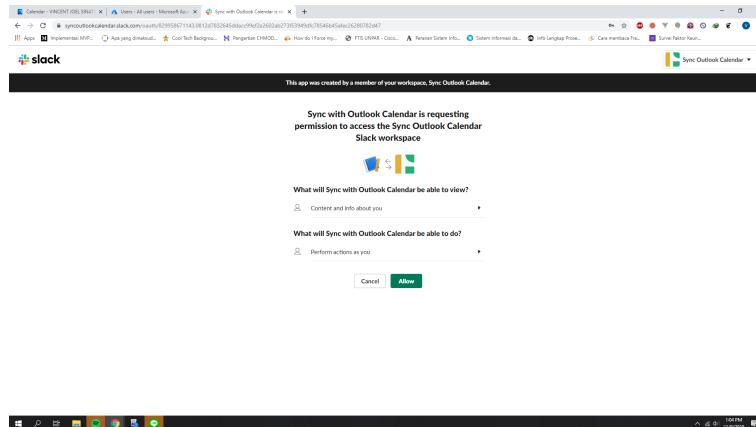
Gambar 5.3: Antarmuka untuk memberikan Windows Live izin ke perangkat lunak.



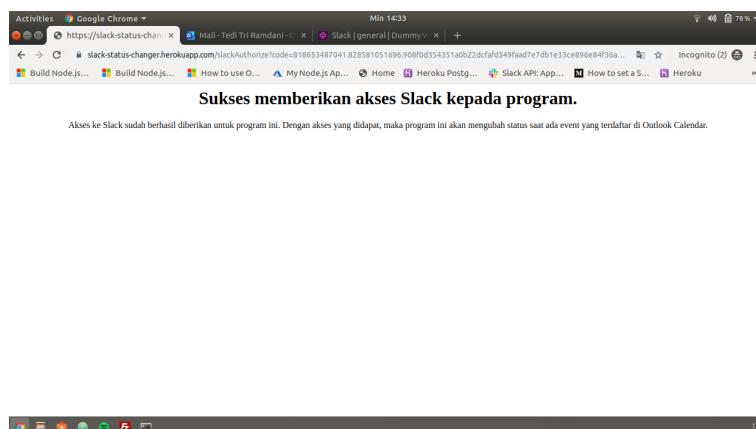
Gambar 5.4: Antarmuka petunjuk login ke Slack.



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.

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

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

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

5.2 Pengujian

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

5.2.1 Pengujian Fungsional

Skenario Pengujian

Pengujian fungsional ini dilakukan dengan pengguna-pengguna yang tergabung dalam *workspace* tempat perangkat ini didaftarkan. Pengguna harus memiliki *Outlook Calendar* dan juga pengguna membuat sebuah *event* yang didaftarkan di dalam *Outlook Calendar*. Selain itu, para pengguna diundang ke dalam *workspace* tempat perangkat lunak ini didaftarkan.

Tujuan Pengujian

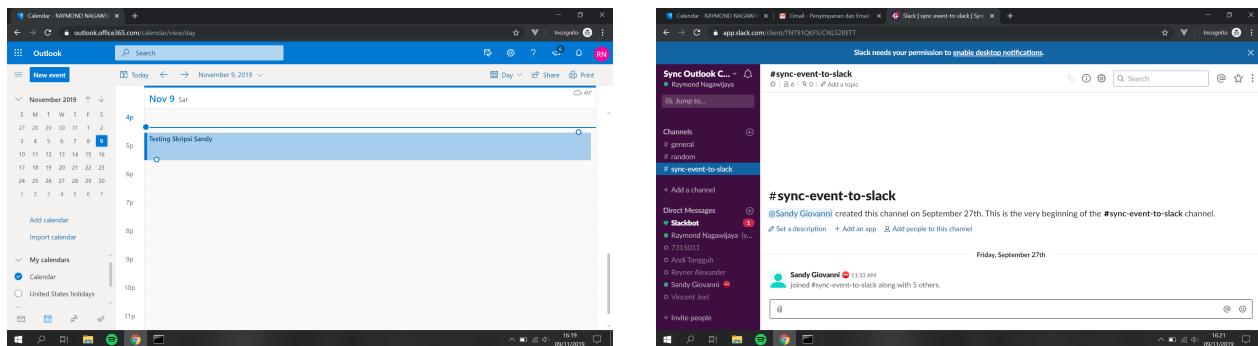
Tujuan dari diadakannya pengujian bagian ini adalah untuk membuktikan bahwa perangkat lunak ini sudah bisa mencapai tujuan yang ingin dituju yaitu perangkat lunak akan menggantikan status dari pengguna di dalam *workspace* tempat perangkat lunak ini didaftarkan saat ada *event* yang terdaftar di dalam *Outlook Calendar*.

Hasil Pengujian

Untuk melakukan pengujian ini, akan diambil beberapa pengguna yang memiliki *Outlook Calendar* dan juga pengguna tersebut diundang masuk dalam *workspace* tempat perangkat lunak ini didaftarkan.

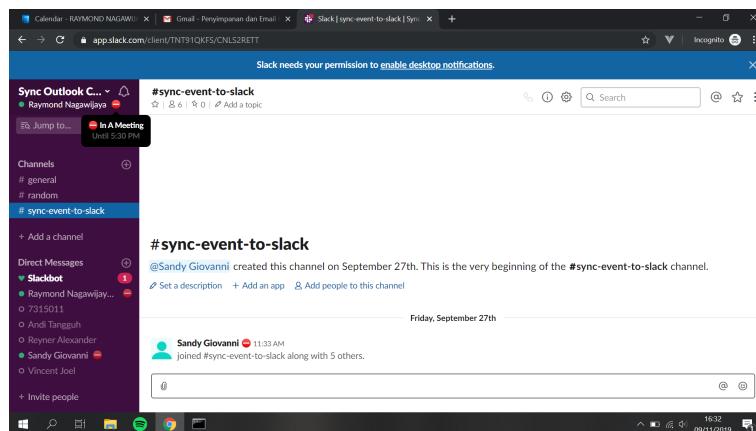
Gambar 5.8a sampai dengan gambar 5.16 menunjukkan bukti hasil dari pengguna yang dimintai ketersediaannya untuk melakukan pengujian terhadap perangkat lunak ini.

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



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

Gambar 5.8



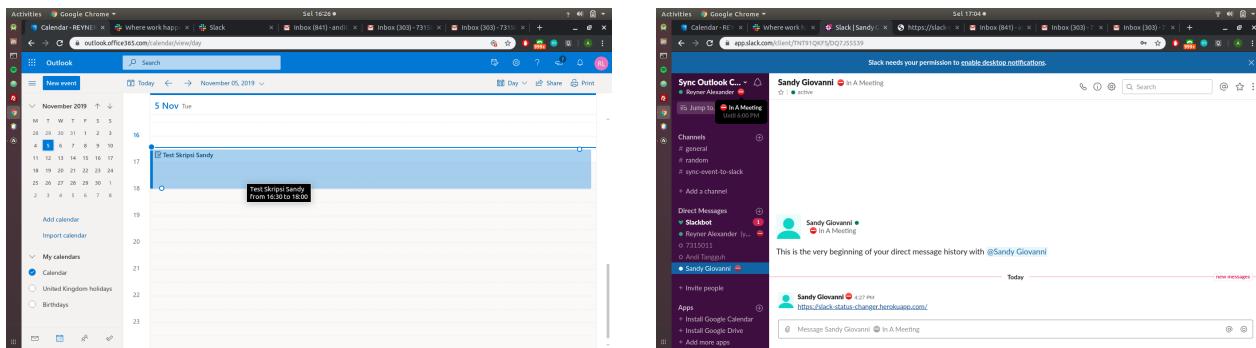
Gambar 5.9: Tampilan *Slack* setelah *event* dimulai (Raymond).

Dari gambar 5.10a sampai gambar 5.10b merupakan bukti dari hasil pengujian dari pengguna yang bernama Reyner. Pada gambar 5.10a dapat dilihat bahwa pengguna memiliki sebuah *event* yang dimulai pada pukul 16.30 dan berakhir pada pukul 18.00. Lalu pada gambar 5.10b dapat dilihat bahwa status pengguna telah berubah dan status itu berlaku sampai pukul 18.00 tepat saat *event* berakhir.

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

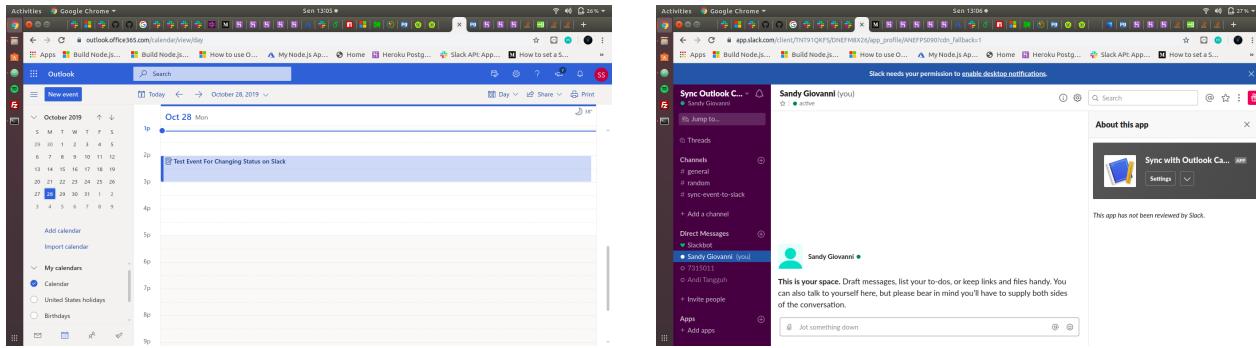
Pada gambar 5.13a sampai dengan gambar 5.14 merupakan hasil dari pengujian yang dilakukan pengguna bernama Vincent. Pada gambar 5.13a terlihat ada sebuah *event* yang akan dimulai pada pukul 13.00 dan pada gambar 5.13b dapat terlihat bahwa pada pukul 12.58 tidak ada status di dalam akun pengguna lalu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

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



(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

5.2.2 Pengujian Eksperimental

Skenario Pengujian

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

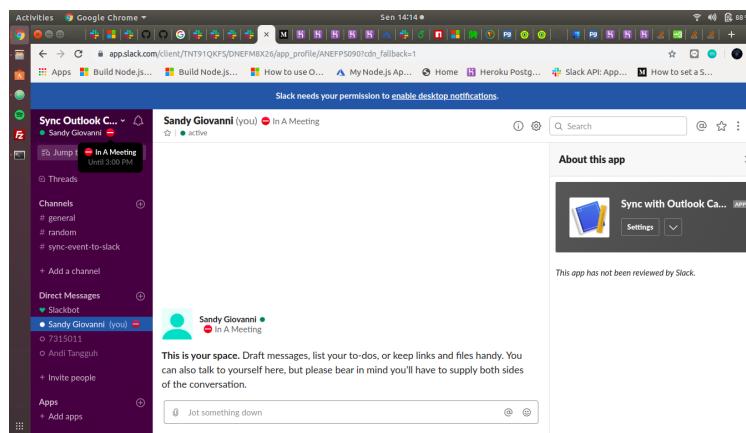
Tujuan Pengujian

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

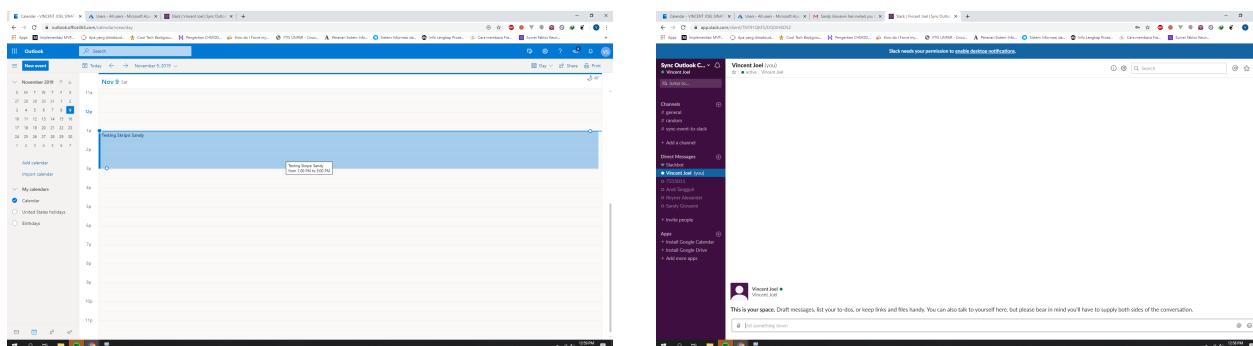
Hasil Pengujian

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

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



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

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

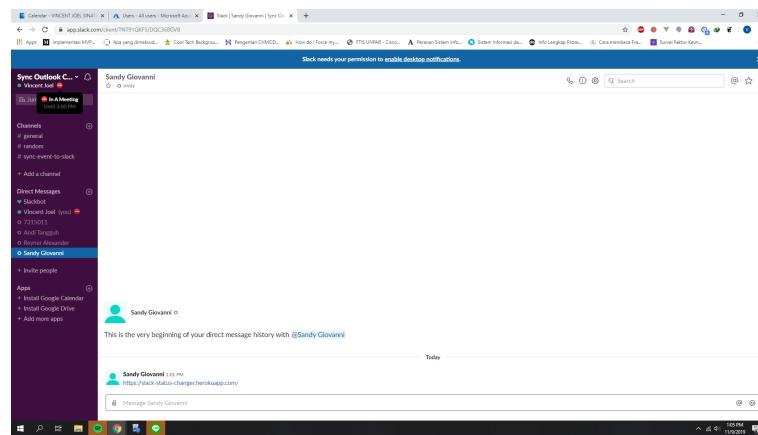
Dari gambar 5.21a sampai dengan gambar 5.22 merupakan bukti yang didapatkan oleh pengguna yang bernama Yehezkiel yang merupakan seorang admin di lab FTIS pada saat pengujian perangkat lunak ini. Pengguna ini menguji di *workspace* Lab FTIS. Pada gambar 5.21a terdapat sebuah *event* yang akan dimulai pada pukul 12.00 dan pada gambar 5.21b¹ belum terdapat adanya status yang terpasang pada akun pengguna. Lalu pada gambar 5.22² dapat terlihat bahwa sudah ada status yang terpasang dan akan berakhir pada pukul 12.30.

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.

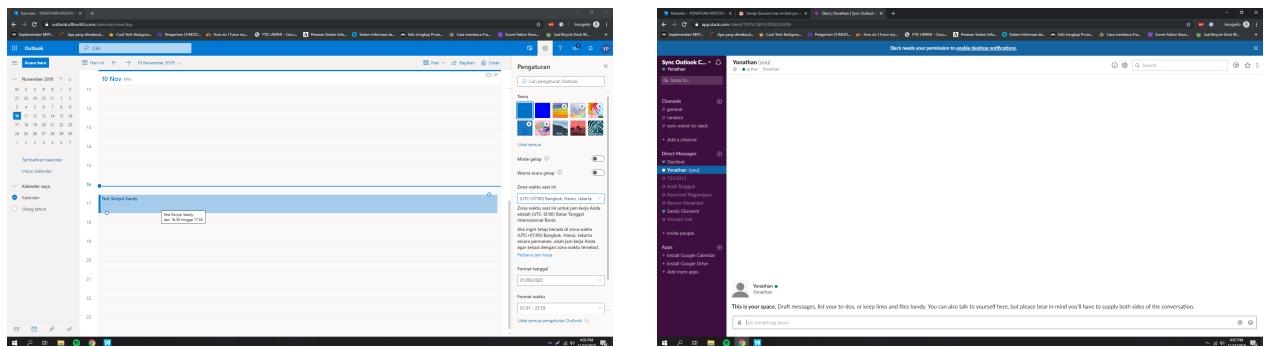
Dari gambar 5.25a sampai dengan gambar 5.26 merupakan hasil pengujian dari pengguna yang

¹Gambar 5.21b di blur atas permintaan responen

²Gambar 5.22 di blur atas permintaan responen



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



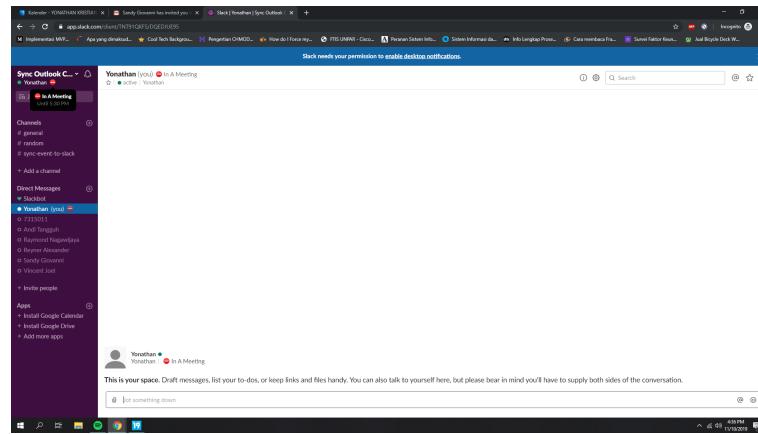
(a) Tampilan *event* yang dibuat pengguna (Yonathan). (b) Tampilan *Slack* sebelum *event* dimulai (Yonathan).

Gambar 5.15

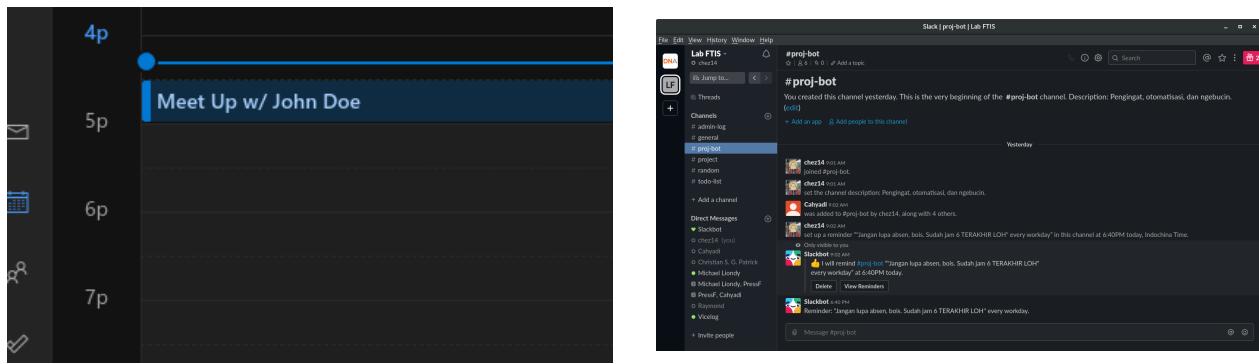
bernama Tedi. Pengguna ini mencoba di *workspace* yang bernama Dummy Workspace. Pada gambar 5.25a³ terdapat *event* yang terdaftar yang dimulai pada pukul 15.00 dan pada gambar 5.25b terlihat waktu pada komputer menunjukkan pukul 14.28 dan belum terdapat status yang terpasang pada akun pengguna. Pada gambar 5.26 menunjukkan pada pukul 15.04 pada komputer, di akun pengguna sudah terpasang status yang statusnya akan berakhir pada pukul 16.00.

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 berakhir pada pukul 15.30.

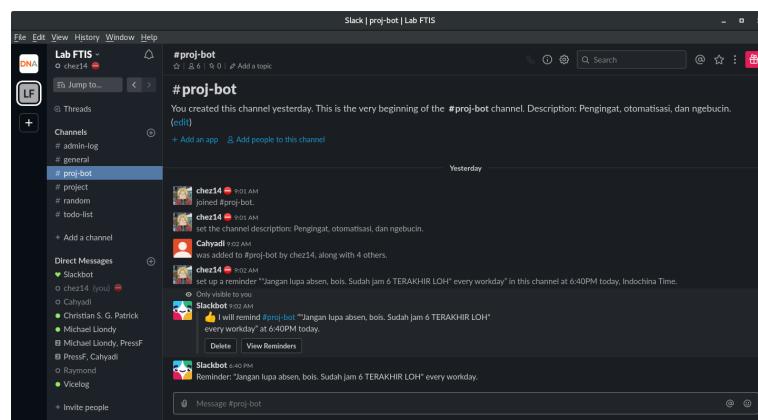
³Judul *event* yang ada di gambar 5.25a di blur atas permintaan responden



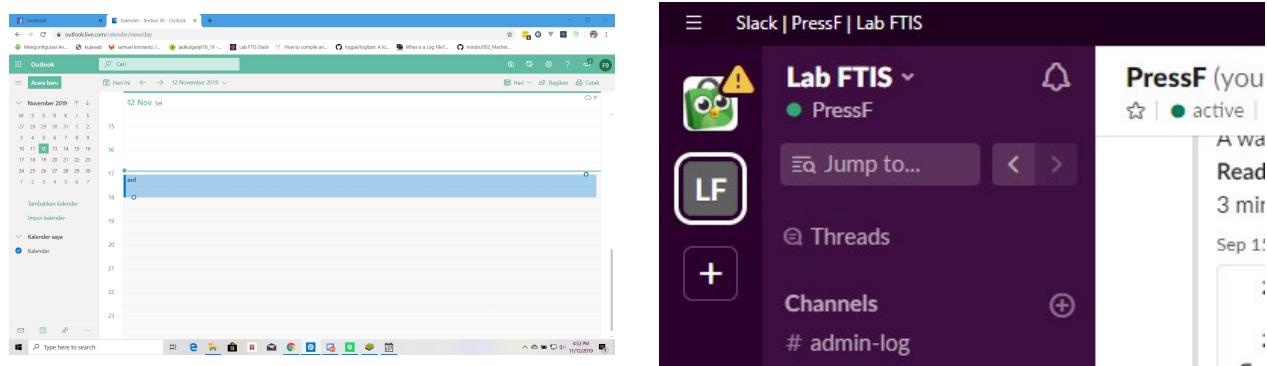
Gambar 5.16: Tampilan *Slack* setelah *event* dimulai (Yonathan).



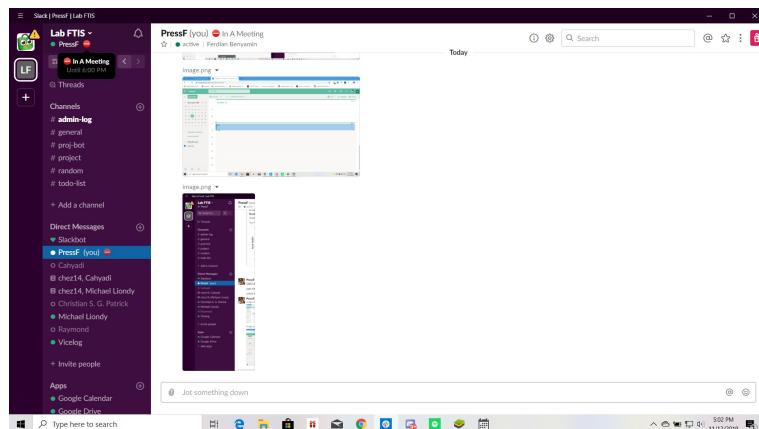
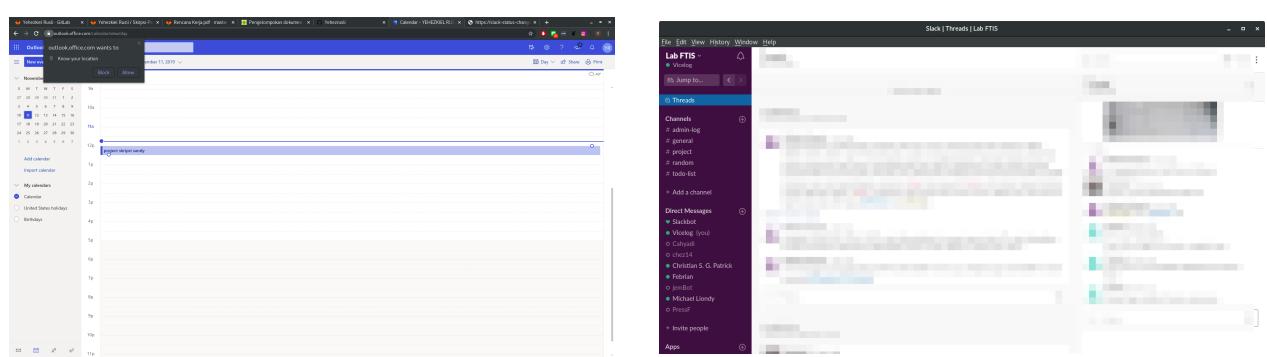
Gambar 5.17



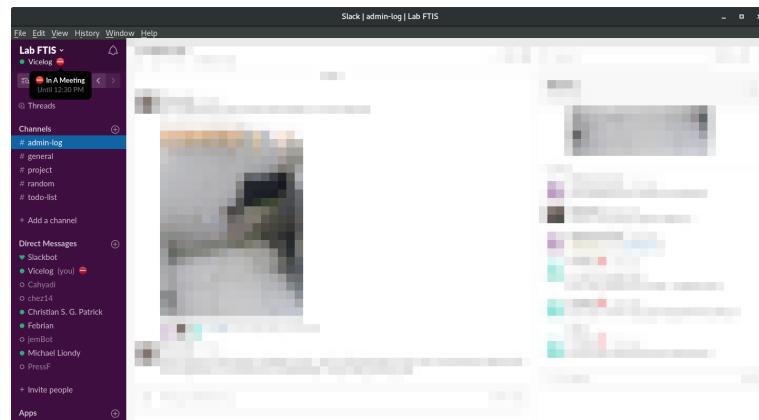
Gambar 5.18: Tampilan *Slack* setelah *event* dimulai (Chris).

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

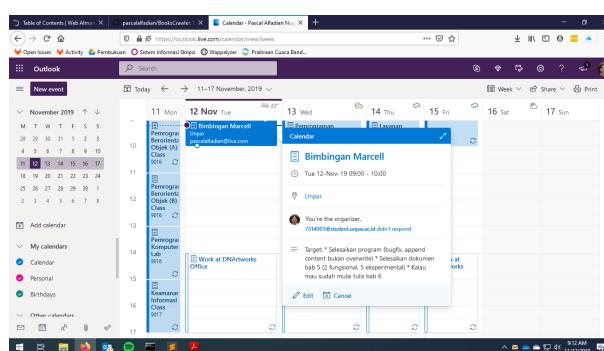
Gambar 5.19

Gambar 5.20: Tampilan *Slack* setelah *event* dimulai (Ferdian).(a) Tampilan *event* yang dibuat pengguna (Yehezkiel). (b) Tampilan *Slack* sebelum *event* dimulai (Yehezkiel).

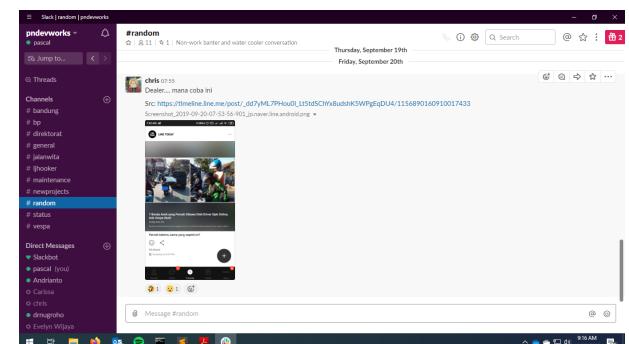
Gambar 5.21



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

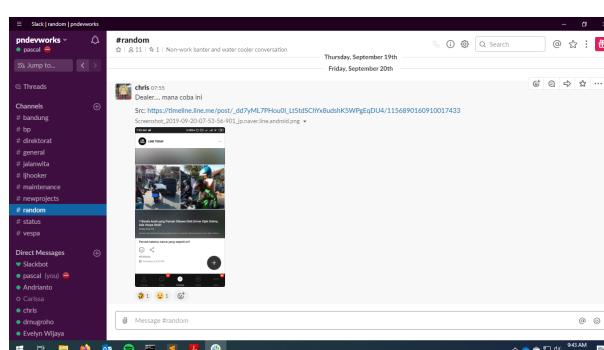


(a) Tampilan *event* yang dibuat pengguna (Pascal).

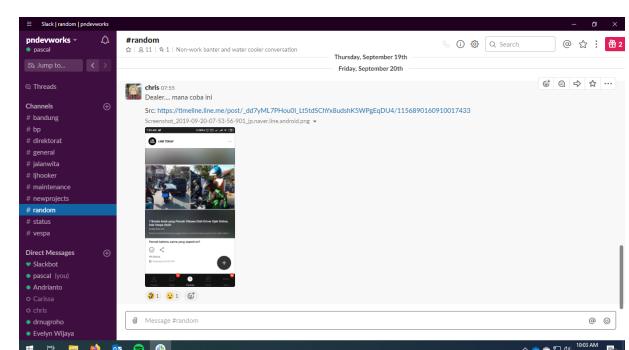


(b) Tampilan *Slack* setelah mengizinkan perangkat lunak kepada *workspace* *Slack* (Pascal).

Gambar 5.23

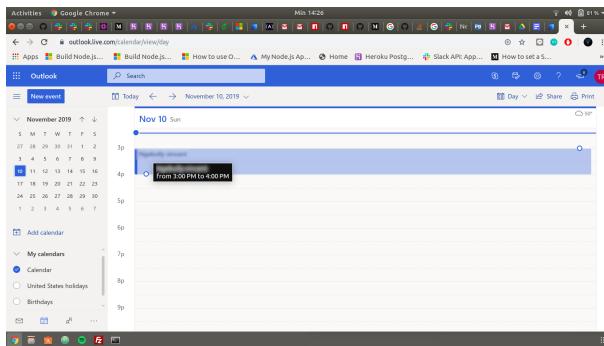
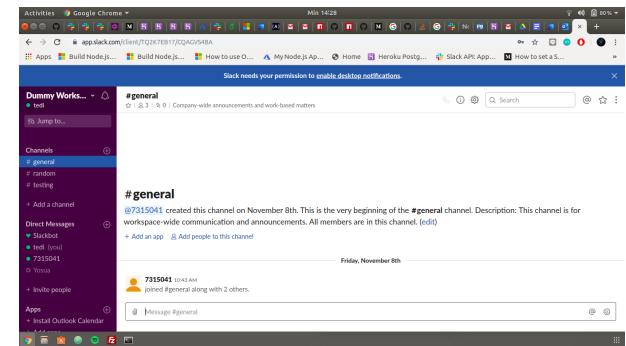


(a) Tampilan *Slack* setelah *event* dimulai (Pascal).

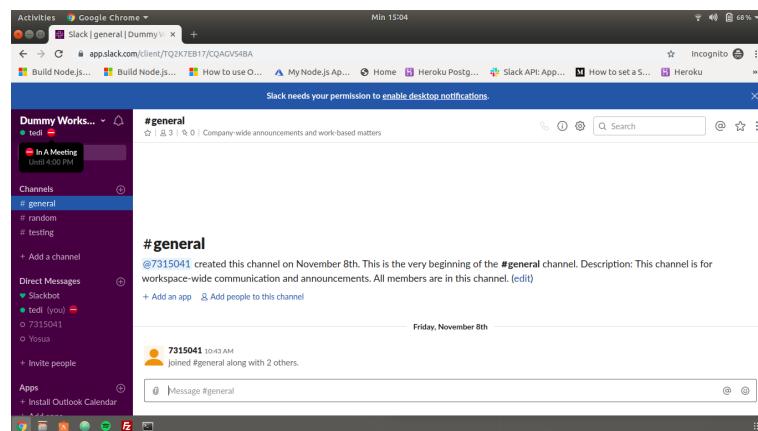
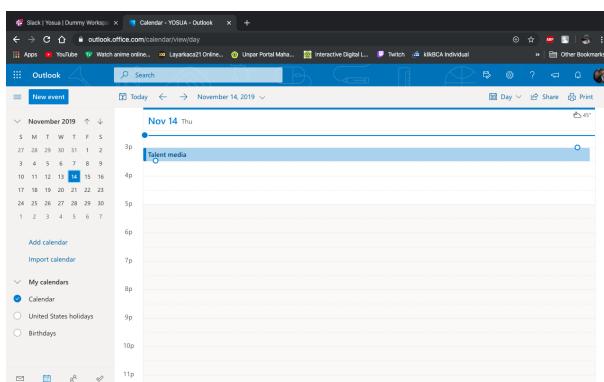
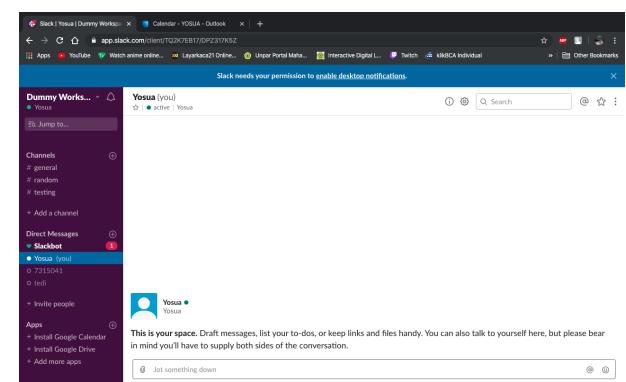


(b) Tampilan *Slack* setelah *event* berakhir(Pascal).

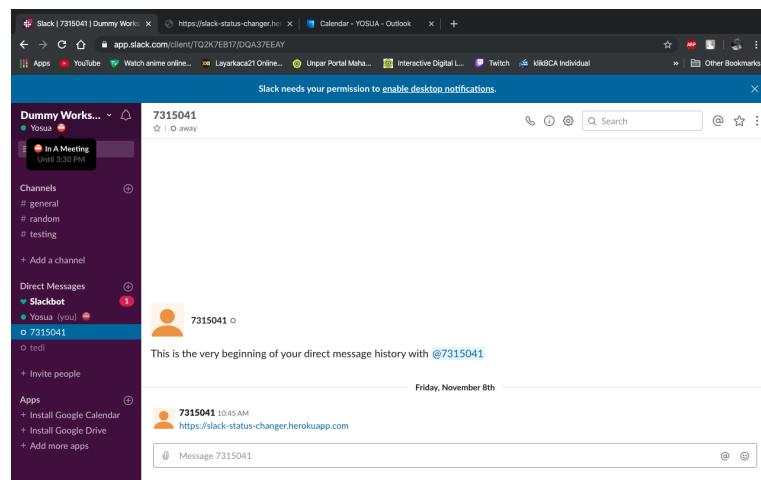
Gambar 5.24

(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).

Kesimpulan dari Pengujian

Pengujian Fungsional	
Nama	Status hasil pengujian
Raymond	Berhasil
Reyner	Berhasil
Sandy	Berhasil
Vincent	Berhasil
Yonathan	Berhasil

Pengujian Eksperimental	
Nama	Status hasil pengujian
Chris	Berhasil
Ferdian	Berhasil
Yehezkiel	Berhasil
Pascal	Berhasil
Tedi	Berhasil
Yosua	Berhasil

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.

BAB 6

KESIMPULAN DAN SARAN

Pada bab ini dibahas tentang kesimpulan dari hasil skripsi dan saran untuk penelitian selanjutnya.

6.1 Kesimpulan

Ada beberapa kesimpulan yang dapat disimpulkan setelah melakukan penelitian ini, yaitu:

- Cara untuk mendapatkan data event pada aplikasi *Outlook Calendar* adalah dengan menggunakan *Microsoft Graph API* dan mengakses *method list events* yang merupakan *method* yang dimiliki oleh *user resource type* dengan *endpoint* dari *method* tersebut ada pada */me/events*.
- Cara untuk mengubah status yang terdapat pada aplikasi *Slack* adalah dengan cara menggunakan *Slack API* dan menggunakan *scope users.profile.set* untuk mengatur profil dari pengguna. Di dalam profil dari pengguna terdapat status yang akan bisa diatur menggunakan *scope user.profile.set* jika melalui akses *API*.
- Cara untuk membuat perangkat lunak yang akan mengubah status jika ada sebuah *event* yang tercatat di *Outlook Calendar* berjalan adalah dengan menggabungkan pemakaian dari kedua *API* yaitu *API* dari *Microsoft Graph* dan juga *API* dari *Slack*. Bagian perangkat lunak yang berinteraksi dengan *Microsoft Graph* akan mendapatkan data tentang *event* dan akan melakukan pemeriksaan terhadap waktunya sekarang. Jika ada *event* yang sedang berjalan sekarang, maka perangkat lunak akan memanggil bagian yang berinteraksi dengan *Slack* untuk mengubah status yang terdapat pada profil pengguna. Iterasi dari penjalanan perangkat lunak ini akan dijalankan selama 10 menit sekali untuk menghindari terlewatnya jadwal yang terperiksa. Dan sebagai tambahan informasi berdasarkan dari hasil pengujian bahwa *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*. Hal ini dapat dilihat dari hasil pengujian yang menunjukkan bahwa semua partisipan yang ikut menguji berhasil mengganti status jika ada *event* miliknya sedang berjalan.

6.2 Saran

Ada beberapa saran terkait dengan penelitian ini untuk dikembangkan lebih lanjut, yaitu:

- Perangkat lunak bisa mensinkronisasikan status dengan *event* yang tercatat secara *real-time*.
- Perangkat lunak bisa mensinkronisasikan satu akun *Windows Live* ke lebih dari 1 *workspace* di *Slack*.
- Perangkat lunak bisa ditambahkan beragam status agar lebih banyak variasi pergantian statusnya.

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   }
85   else {
86     console.log("Tidak ada event yang bersamaan dengan waktu skrng");
87   }
88 }
89 
90 }catch (err) {
91   console.log("error", err);
92 }
93 }
94 
95 
96 // Fungsi ini berguna untuk meminta access token yang baru dengan menggunakan refresh token.
97 async function useRefreshToken(auth_code) {
98   try{
99     let newToken=await oauth2.accessToken.create({refresh_token: auth_code}).refresh();
100 
101    const user = jwt.decode(newToken.token.id_token);
102    const databaseValue={};
103    newToken.token.userData=user;
104    databaseValue.microsoft_username=newToken.token.userData.preferred_username;
105    databaseValue.microsoft_access_token_expires=newToken.token.expires_in;
106    databaseValue.microsoft_access_token=newToken.token.access_token;
107    databaseValue.microsoft_refresh_token=newToken.token.refresh_token;
108    databaseValue.login_timestamp=new Date().getTime();
109 
110   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_*';
111   var valueForInsert=[databaseValue.microsoft_username, databaseValue.microsoft_refresh_token, databaseValue.microsoft_access_token_expires, databaseValue.microsoft_access_token, databaseValue.login_timestamp];
112 
113   client.query(queryText, valueForInsert, (err, res) => {
114     if (err) {
115       console.log(err.stack)
116     } else {
117     });
118   return newToken.token.access_token;
119 }
120 catch(err){
121   console.log(err);
122 }
123 }
124 }
125 
126 
127 // Fungsi ini berguna untuk mengganti status di slack
128 async function changeStatusSlack(slack_access_token, endDate){
129   const web = new WebClient(slack_access_token);
130 
131   const result = await web.users.profile.set({
132     "profile": {
133       "status_text": "In_A_Meeting",
134       "status_emoji": ":no_entry:",
135       "status_expiration":endDate/1000
136     }
137   });
138 }
139 
140 module.exports = router;

```

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>

```