Integrasi *Outlook Calender* dan *Slack*

Sandy Giovanni S.–2015730041

1 Data Skripsi

Pembimbing utama/tunggal: **Pascal Alfadian**
Pembimbing pendamping: **-**
Kode Topik : **PAN4505**
Topik ini sudah dikerjakan selama : **1** semester
Pengambilan pertama kali topik ini pada : Semester **46 - Genap 18/19**
Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**
Tipe Laporan : **B** - Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

2 Latar Belakang

Outlook.com 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 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 komunitas, grup, atau tim bergabunga ke dalam URL yang spesifik. *Room chat* yang terdapat di dalam aplikasi *Slack* biasa disebut dengan *Channel*. Ada 2 jenis *channel* di dalam 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 serta dalam berkomunikasi di dalam *channel* tersebut. *Slack* juga terintegrasi dengan banyak layanan pihak ketiga seperti contohnya adalah *Google Drive*, *Github*, *Trello*, *Dropbox*, dan masih banyak lagi layanan pihak ketiga yang bisa diintegrasikan dengan *Slack* itu sendiri.

Pada *Slack* terdapat status pengguna yang bisa diganti oleh pengguna tersebut untuk menggambarkan keadaan pengguna saat ini. Sebagai *default*, status bisa menggambarkan jika pengguna sedang "*In a meeting*" atau sedang "*Out Sick*", dan banyak status default yang disediakan oleh *Slack*. Serta status pun bisa diisi oleh pengguna secara sendiri sesuai dengan apa yang ingin dituliskan oleh penggunanya. Disinilah 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 sang pengguna sedang dalam keadaan *meeting* yang tidak dapat diganggu. Di saat seperti ini, kemungkinan untuk *meeting* terganggu oleh adanya *chat* yang masuk lewat *Slack* pun cukup tinggi.

Perangkat lunak ini akan dibuat dengan bantuan dari masing-masing API(*Application Programming Interface*). *Outlook Calendar* dan juga *Slack* memiliki API masing- masing yang cara penggunaannya terdapat dalam dokumentasi dari aplikasi tersebut yang bisa ditemui di dalam laman *website* dari masing- masing aplikasi tersebut. Perangkat ini juga akan dibangun menggunakan *Node.js* yang bisa dipelajari melalui laman *website* dokumentasi *Node.js* itu sendiri.

3 Tujuan

Tujuan dari penyusunan skripsi ini antara lain:

- Mengetahui cara menggunakan *Node.js*.
- Mengetahui cara mendapatkan data *event* dari *Outlook Calendar*.
- Mengetahui cara mengubah status pada aplikasi *Slack* menggunakan *Slack* API.
- Mengetahui cara membuat program agar dapat mengubah status pada aplikasi *Slack* di jadwal yang telah didapat dari aplikasi *Outlook Calendar*.

4 Rumusan Masalah

Rumusan masalah pada topik ini adalah:

- Bagaimana cara menggunakan *Node.js*?
- Bagaimana cara mendapatkan data *event* dari *Outlook Calendar*?
- Bagaimana mengubah status pada aplikasi *Slack* menggunakan *Slack* API?
- Bagaimana cara membuat program agar dapat mengubah status pada aplikasi *Slack* di jadwal yang telah didapat dari aplikasi *Outlook Calendar*?

5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencan kerja/laporan perkembangan terkahir :

1. **Melakukan studi literatur melalui dokumentasi *online* mengenai *Node.js*.**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *Node.js* adalah *platform* perangkat lunak pada sisi *server*. *Node.js* ini ditulis dengan menggunakan bahasa *Javascript* dengan menggunakan *npm* sebagai *default package manager*. Untuk menggunakan *node.js*, dibutuhkan untuk mengunduh dan menginstal terlebih dahulu *node.js* yang akan dipakai pada situs resmi yang tersedia[1]. Setelah mengunduh dan menginstall *node.js*, 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 *server-base*, yang biasanya program *JavaScript* adalah sebuah *client-base*. Selain itu, *node.js* juga merupakan sebuah *JavaScript* yang berjalan secara *asynchronous* yang diperuntukkan untuk membangun aplikasi jaringan yang bersifat skalabilitas.

   Node.js disusun pertama kali di tahun 2009 oleh Ryan Dahl dan dibangun serta dikelola juga oleh Ryan dengan mendapatkan bantuan dari Joyent. Alasan dari Ryan Dahl mencetuskan ide *node.js* ini karena ia tidak suka dengan cara kerja server *Apache* yang digunakan untuk menangani banyak koneksi secara bersamaan dan kode yang dibuat memblokir seluruh proses atau mengimplikasi banyak eksekusi pada koneksi serentak. Hal ini yang membawanya untuk berniat untuk membuat proyek *Node.js* yang kemudian dia tunjukkan di *JSConf* Eropa pada 8 November 2009.

   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 akan digunakan dan akan

[1] https://nodejs.org/en/

dipanggil di kode program yang akan dibuat perlu dipanggil dengan menggunakan perintah *"require()"* dengan membutuhkan parameter yaitu nama kelas yang akan digunakannya.

Salah satu kelas yang dipelajari adalah kelas *HTTP*. Untuk bisa mengakses dan menggunakan kelas *HTTP server* dan *client*, maka harus menggunakan kode *"require('http')"*. Kelas *interface HTTP* di dalam *node.js* ini di*design* untuk mendukung banyak fitur protokol yang sulit digunakan. Dengan adanya *interface HTTP*, *node.js* menjadi bisa mengirim data melalui *Hyper Text Transfer Protocol(HTTP)*. *Header* pada pesan *HTTP* merepresentasikan sebuah objek seperti pada contoh table 1.

Tabel 1: Tabel contoh *header* pesan *HTTP*

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

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, *callback* yang merupakan sebuah *function*. 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 alias 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 soket *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 harus 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 otentikasi 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 *option* ini adalah *true*.
- **callback** yang berupa *function*.
- **Returns** bertipe *http.ClientRequest*.

Node.js memelihara beberapa koneksi per servernya 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 diubah 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 *request* yang dijalankan.

2. **Melakukan studi literatur melalui dokumentasi *online* mengenai aplikasi *Outlook Calendar*.**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Terdapat perubahan di poin ini dikarenakan adanya aplikasi yang berbeda yaitu aplikasi *Outlook Calendar* dan *Outlook.com Calendar*. Aplikasi yang akan digunakan di skripsi ini adalah *Outlook.com Calendar* yang memiliki *API* yang berpusat pada *Microsoft Graph API*. *Microsoft Graph API* adalah *webservice* 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. Pada versi 1.0, *endpoint* utama yang dipakai adalah mengacu kepada *endpoint https://graph.microsoft.com/v1.0*.

   Untuk menggunakan fungsi dari *Microsoft Graph API*, dibutuhkan untuk mendaftarkan terlebih dahulu aplikasi yang akan dirancang dan memakai fungsi dari *webservice* dari *Microsoft Graph API* ke *Microsoft App Registration Portal*[2]. Pada saat mendaftarkan aplikasinya, pastikan untuk menyalin dan menyimpan *application ID* yang adalah pengenal unik untuk aplikasi yang didaftarkan, dan juga menyalin *Redirect URL* yang didaftarkan sebagai *URL* yang akan menerima balikan *authentication* dan juga *token* yang akan dikirim oleh *endpoint Azure AD v2.0*, serta menyalin *application secret* yang didapat saat mengklik "*Generate New Password*" saat mendaftarkan aplikasi (berlaku jika mendaftarkan aplikasi berjenis *web apps*). *Application ID* yang didapat saat mendaftar aplikasi akan dipakai untuk mengisi nilai dari parameter *client_id* yang akan diisi saat akan melakukan *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 mengembalikkan jangka waktu aktif dari *token* tersebut dan juga *refresh_token* yang akan berguna untuk meminta *access_token* saat sudah *expired*.

[2]https://apps.dev.microsoft.com/

Setelah mendapatkan *access token*, barulah layanan untuk mendapatkan data yang tersimpan di *Microsoft* baru bisa diakses dan didapatkan. Ada banyak layanan yang disediakan dari *Microsoft Graph API* yang dikelompokkan menjadi kelas-kelas yang masing-masing memiliki properti dan juga *method-method* yang cara mengaksesnya memiliki *endpoint* masing-masing. Dari banyaknya kelas yang disediakan oleh *Microsoft Graph*, pembelajaran lebih difokuskan kepada kelas *user* dan juga kelas *event* dikarenakan yang akan diambil dari bagian *Microsoft Graph* adalah *event* dari seorang pengguna.

3. **Melakukan studi literatur melalui dokumentasi *online* mengenai aplikasi *Slack*.**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Pada poin ini lebih diutamakan mempelajari aplikasi terutama pada *API* yang telah disediakan oleh aplikasi *Slack*. *Slack API* adalah *webservice* yang akan digunakan untuk menghubungkan data yang sudah di dapat dari *Outlook.com Calendar* ke aplikasi *Slack*. Untuk mengakses *Slack API*, diharuskan untuk mendaftarkan aplikasi yang akan dibuat dan juga mendaftarkannya ke *workspace* yang akan dipakai untuk menjalankan aplikasi yang akan dibuat. Untuk mendaftarkan aplikasi yang akan 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 akan digunakan pada proses *OAuth*.

   Proses pertama yang akan dijalani dalam rangkaian proses *OAuth* adalah dengan meminta *authorization code* yang akan berjalan jika aplikasi yang akan dibuat untuk mengarahkan pengguna dan 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 akan dibuat dan juga *get parameter scope*, serta memiliki parameter yang bersifat *optional* 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.

   Authorization code yang telah didapat memiliki waktu kadaluarsa selama 10 menit. Setelah mendapatkan *authorization code*, langkah selanjutnya yang harus dilakukan adalah menukarkan *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*, barulah *method API* yang disediakan bisa dijalankan dengan mengirimkan *request* yang memiliki *header access token*nya dan *parameter request* sesuai dengan apa yang diminta oleh *method* yang tersedia.

4. **Melakukan studi literatur melalui dokumentasi *online* mengenai cron dan crontab.**
   Status : Baru ditambahkan pada semester ini.
   Hasil : *Cron* adalah sebuah *daemon* untuk mengeksekusi perintah-perintah yang sudah terjadwalkan. *Daemon* sendiri adalah proses layanan yang berjalan secara *background* dan mengawasi sistem atau menyediakan fungsionalitas untuk proses lainnya. *Cron* dimulai dari */etc/rc.d/init.d* atau */etc/init.d*

ketika *sysvinit* digunakan. *File* unit disimpan dan diinstal ke */lib/systemd/system/crond.service* dan *daemon* dimulai dengan cara menjalankan perintah *systemctl start crond.service*. *Cron* mencari ke direktori */var/spool/cron* untuk *file-file crontab*.

Crontab adalah *file* yang digunakan untuk menjadwalkan eksekusi dari sebuah program. *Crontab* bisa diakses oleh pengguna dengan menjalankan perintah "*crontab*" di *terminal* dilanjutkan dengan perintah yang akan diberikan. Daftar perintah yang bisa dijalankan oleh *crontab* adalah:

- *crontab -e*
  Command ini digunakan untuk membuat atau mengubah *file crontab* jika sudah ada.
- *crontab -l*
  Command ini digunakan untuk menampilkan isi *file crontab*.
- *crontab -r*
  Command ini digunakan untuk menghapus *file crontab*.

Crontab memiliki baris kode yang berformat "*m h dom mon dow command*" yang memiliki arti:

- m
  m sebagai menit yang bisa diisi dengan nilai dari 0-59.
- h
  h sebagai jam yang bisa diisi dengan nilai dari 0-23.
- dom
  dom sebagai *Day Of Month* (tanggal) yang bisa diisi dengan nilai dari 0-31.
- mon
  mon sebagai bulan yang bisa diisi dengan nilai dari 0-12.
- dow
  dow sebagai *Day Of Week* yang bisa diisi dengan nilai dari 0-7 yang menggambarkan hari dalam angka. Nilai 0 dan nilai 7 adalah hari Minggu.
- command
  command disini berisi program yang akan dijalankan dengan jadwal yang sudah diatur dengan parameter sebelumnya.

Semua nilai dari *parameter* m, h, dom, mon, dan dow bisa diisi dengan simbol bintang (*) yang memiliki arti dijalankan setiap menit, setiap jam, setiap hari, setiap bulan tergantung dari posisi dimana simbol itu ditempatkan.

5. **Melakukan analisis cara melakukan *synchronize* dengan aplikasi *Outlook Calendar* secara berkala.**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Analisis ini dilakukan harus dengan cara membaginya menjadi analisis atas pemakaian *Microsoft Graph API*, dan juga analisis mengenai *Cron* dan *Crontab*. Pada analisis bagian *Microsoft Graph*, 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. Langkah pertama yaitu mendapatkan *authorization_code*. Untuk mendapatkan *authorization code*, diperlukan untuk mendaftarkan aplikasi yang akan dibuat ke *Microsoft App Registration Portal*[3]. 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

[3]https://apps.dev.microsoft.com/

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 2.

Tabel 2: Tabel parameter *Authorization Code*

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

Pada parameter *scope*, dapat diisi dengan nilai *offline_ access* yang akan 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.

Tabel 3: 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, akan menghasilkan contoh *response* seperti yang terdapat pada table 4 dengan keterangan parameter seperti yang terdapat pada table 5.

Tabel 4: Tabel contoh *response Authorization Code*

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

Tabel 5: Tabel parameter *response Authorization Code*

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

Hasil *response* yang ditampilkan oleh table 4 muncul karena pada saat *request* di table 3 terdapat parameter *response_mode* yang diisi dengan nilai *query* sehingga *response* yang dikembalikan dalam bentuk *query string* dari *redirect url*. 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 6. Adapun penjelasan dari setiap parameter yang terdapat di dalam *request body* dijelaskan pada table 7.

Tabel 6: 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 7: Tabel parameter *request Access Token*

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

Dari contoh *request* yang dilakukan pada table6, maka akan dihasilkan contoh *token* seperti pada table 8 yang memiliki keterangan dari hasil yang dikembalikan pada table9.

Tabel 8: Tabel contoh *response Access Token*


```
{
"token_type": "Bearer",
"scope": "user.read%20Fmail.read",
"expires_in": 3600,
"access_token": "eyJ0eXAiOiJKV1QiLCJhbG
ciOiJSUzI1NiIsIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
"refresh_token": "AwABAAAAvPM1KaPlrEqdF
SBzjqfTGAMxZGUTdM0t4B4..."
}
```


Tabel 9: Tabel parameter *response Access Token*

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

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 table10 menunjukkan contoh *request* untuk mendapatkan profil dari pengguna yang masuk.

Tabel 10: Tabel contoh *request call Microsoft Graph*

```
GET https://graph.microsoft.com/v1.0/me
Authorization: Bearer eyJ0eXAiO ...
0X2tnSQLEANnSPHY0gKcgw
Host: graph.microsoft.com
```

Jika *request* yang dikirimkan berhasil, maka akan mendapatkan *response* yang akan terlihat mirip dengan contoh seperti pada table 11

Tabel 11: 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 555555555"
],
"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"
}
```

Access token memiliki waktu yang singkat dan ketika sudah kadaluarsa, maka aplikasi yang akan dibuat harus meminta kembali *access token* yang supaya bisa terus mengakses data yang ada di dalam *Microsoft Graph*. Cara mendapatkan *access token* yang baru dengan menggunakan *refresh token* adalah dengan cara mengirimkan *post request* sekali lagi kepada *endpoint /token* dan untuk kali ini, gunakan *refresh token* sebagai parameter yang dikirimkan dan juga *grant type* yang berisikan *refresh token* dalam *body* dari *request* yang dilakukan seperti contoh pada table 12 dengan keterangan parameter seperti yang dijelaskan pada table 13.

Tabel 12: 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 13: Tabel parameter *request Refresh Token*

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

Jika *request* ini berhasil, maka akan mengembalikan *response* seperti pada table 14 yang memiliki keterangan parameter yang dikembalikannya seperti pada table 15.

Tabel 14: Tabel contoh *response* menggunakan *Refresh Token*

```
{
"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiI
sIng1dCI6Ik5HVEZ2ZEstZnl0aEV1Q...",
"token_type": "Bearer",
"expires_in": 3599,
"scope": "user.read%20mail.read",
"refresh_token": "AwABAAAAvPM1KaPlrEqdFSBzjq
fTGAMxZGUTdM0t4B4...",
}
```

Tabel 15: Tabel parameter *response Refresh Token*

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

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 akan dibuat harus menjalankan *method* yang dimiliki objek *user* yaitu *method list events* 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 16.

Tabel 16: Tabel contoh *header time zone*

Prefer: outlook.timezone="Eastern Standard Time"

Pada table 16, *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 17.

Tabel 17: Tabel parameter *header time zone*

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

Request ini juga bisa menerima parameter *$select* yang berbentuk *string query* sebagai *filter* mengenai *field* apa saja yang mau diambil dari objek *event*. Dapat dilihat fungsi dari parameter *string query $select* seperti pada contoh table 18 dan contoh *response*nya pada table 19.

Tabel 18: 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 19: 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",
                "uniqueId": "Assembly Hall",
                "uniqueIdType": "private"
            },"locations": [{
                    "displayName": "Assembly Hall",
                    "locationType": "default",
                    "uniqueIdType": "unknown"
            }],
            "organizer":{
                "emailAddress":{
                    "name":"Samantha Booth",
                    "address":"samanthab@a830edad905084922E170
                    20313.onmicrosoft.com"
                }
            }
        }
    ]
}
```


Lalu 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*. *Crontab* memiliki kemampuan untuk menjalankan program secara berkala sesuai dengan format pengaturan yang sudah di *set* dari awal pembuatan perintah *crontab*. *Crontab* tinggal harus memanggil dan menjalankan aplikasi untuk mengambil data dan secara otomatis sesuai dengan waktu yang sudah diatur di *crontab*, maka program akan dijalankan. Untuk contoh format *file crontab* akan ditunjukkan pada **??**.

Listing 1: contoh cronfile

```
\label{lst:contoh_cronfile}
# use /bin/sh to run commands, no matter what /etc/passwd says
SHELL=/bin/sh
# mail any output to 'paul', no matter whose crontab this is
```

```
MAILTO=paul
#
CRON_TZ=Japan
# run five minutes after midnight, every day
5 0 * * *       $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 2:15pm on the first of every month -- output mailed to paul
15 14 1 * *     $HOME/bin/monthly
# run at 10 pm on weekdays, annoy Joe
0 22 * * 1-5     mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun      echo "run at 5 after 4 every sunday"
```

6. **Melakukan analisis cara aplikasi merespons ketika menjadwalkan perubahan status di aplikasi *Slack*, dengan kemungkinan masalah seperti : ada *event* yang baru ditambahkan setelah program melakukan *synchronize* secara berkala atau ada *event* yang beririsan dengan *event* lainnya sehingga terdapat masalah menentukan kapan status dibuang/ dikembalikan lagi ke status semula.**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Belum dikerjakan.

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Melakukan studi literatur melalui dokumentasi *online* mengenai *Node.js*.
2. Melakukan studi literatur melalui dokumentasi *online* mengenai aplikasi *Outlook Calendar*.
3. Melakukan studi literatur melalui dokumentasi *online* mengenai aplikasi *Slack*.
4. Melakukan studi literatur melalui dokumentasi *online* mengenai cron dan crontab.
5. Melakukan analisis cara melakukan *synchronize* dengan aplikasi *Outlook Calendar* secara berkala.

7 Kendala yang Dihadapi

Kendala - kendala yang dihadapi selama mengerjakan skripsi :

- Sulitnya fokus untuk mengerjakan.
- Terlalu banyak godaan berupa hiburan (game, film, dll).
- Dokumentasi dari *Outlook Calendar* kurang jelas dan lebih dari 1 versi dokumentasi untuk *Microsoft Graph API*.

Bandung, 01/05/2019

Sandy Giovanni S.

Menyetujui,

Nama: Pascal Alfadian
Pembimbing Tunggal