«SKRIPSI/TUGAS AKHIR»

# «JUDUL BAHASA INDONESIA»


«Nama Lengkap»

NPM: «10 digit NPM UNPAR»

PROGRAM STUDI «MATEMATIKA/FISIKA/TEKNIK INFORMATIKA»
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
«tahun»

«FINAL PROJECT/UNDERGRADUATE THESIS»

# «JUDUL BAHASA INGGRIS»

«Nama Lengkap»

NPM: «10 digit NPM UNPAR»

DEPARTMENT OF «MATHEMATICS/PHYSICS/INFORMATICS»
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
«tahun»

# LEMBAR PENGESAHAN

«JUDUL BAHASA INDONESIA»

«Nama Lengkap»

NPM: «10 digit NPM UNPAR»

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

| Pembimbing Utama | Pembimbing Pendamping |
| --- | --- |
| «pembimbing utama/1» | «pembimbing pendamping/2» |

| Ketua Tim Penguji | Anggota Tim Penguji |
| --- | --- |
| «penguji 1» | «penguji 2» |

Mengetahui,

Ketua Program Studi

«Ketua Program Studi»

# PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa «skripsi/tugas akhir» dengan judul:

**«JUDUL BAHASA INDONESIA»**

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 «tanggal» «bulan» «tahun»

Meterai
Rp. 6000

«Nama Lengkap»
NPM: «10 digit NPM UNPAR»

## ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.



**Kata-kata kunci:** «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

## ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



**Keywords:** «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

*«kepada siapa anda mempersembahkan skripsi ini. . . ?»*

# KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »
Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bandung, «bulan» «tahun»

Penulis

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

# 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 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.

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

Perangkat lunak ini akan dibuat menggunakan *Node.js*[3] dan akan memiliki 2 fungsi utama yaitu yang pertama adalah membaca dan mencatat jadwal dari *Outlook.com Calendar* yang membutuhkan adanya *Outlook.com Calendar API*. Lalu perangkat lunak ini juga memiliki fungsi kedua yaitu

[1] https://outlook.live.com/
[2] https://slack.com/
[3] https://nodejs.org

mengubah status ke aplikasi *Slack* dengan menggunakan *Slack API*. Nantinya kedua fungsi dari perangkat lunak ini akan dijalankan secara berkala.

## 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*.

## 1.4 Batasan Masalah

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

1. Program ini dijalankan secara berkala sehingga tidak dapat menjalankan *update* status secara *real-time*.

## 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. Menganalisis aplikasi-aplikasi sejenis.
4. Melakukan analisis cara melakukan *synchronize* dengan aplikasi *Outlook.com Calendar* secara berkala.
5. 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*.
6. 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.

## 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-teori yang dibutuhkan dan mendukung berjalannya penelitian ini. Meliputi tentang *Outlook.com Calendar API*, *Slack API*, *Node.js*, dan juga *Crontab*.

3. Bab 3: Analisis, yaitu membahas mengenai analisis masalah. Berisi tentang analisis aplikasi sejenis, 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 aplikasi yang telah dirancang dan juga pengujian aplikasi 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

## 2.1 *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*[1]. 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 refresh_token saat token sudah *expired.*

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*

### 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.

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

- **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 akan 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 yanng 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 samAccount-Name 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 passswordPolicies.
- **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 akan dijabarkan, perlu diketahui bahwa untuk mengirimkan *request*, diperlukan *header* yang berisikan *Authorization* yang diisi dengan nilai dari *Access_token* yang didapat dari proses sebelumnya. *Endpoint* utama untuk mengirimkan request adalah ke *https://graph.microsoft.com/v1.0* lalu dilanjutkan dengan *endpoint* fungsinya masing-masing yang akan diberikan dan dijelaskan.

- **List users** Method ini berfungsi untuk mendapatkan daftar dari objek pengguna. Method ini di*request* 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*. *Respons* dari method ini akan mengembalikan *json* dari objek pengguna.
- **Create user** Method ini berfungsi untuk membuat pengguna. Method ini di*request* 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, dam 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 memperbaharui 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 attribut *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 *endopoint /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 kepada 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 contacs** Method ini berfungsi untuk mendapatkan daftar kontak dari folder Contacts pengguna yang masuk. Method ini dijalankan dengan cara mengirimkan *get requerst* 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* kepada *endpoint /users/{id | userPrincipalName}/contactFolders*.

- **Create contactFolder** Method ini berfungsi untuk membuat folder kontak. Method ini dijalankan dengan cara mengirimkan *post request* kepada *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* kepada *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* kepada *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* kepada *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* kepada *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 mengembalikkan 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 mengembalikkan 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 mengembalikkan 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}* bisa diganti dengan */me*.

### 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 seharian atau tidak. Bertipe *Boolean* yang akan bernilai *true* jika acaranya berlangsung seharian, dan bernilai *false* jika tidak.
- **isCancelled** Properti ini untuk menunjukkan apakah 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 sebelumnya. *Endpoint* utama untuk mengirimkan request adalah ke *https://graph.microsoft.com/v1.0* lalu dilanjutkan dengan *endpoint* fungsinya masing-masing 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 membaharui 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 *respon* 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*.

### 2.1.3 Lain-lain

Terdapat masih banyak kelas yang disediakan dan digunakan pada *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*.

## 2.2 *Slack API*

*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*, kita 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 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.setPuprose
  - 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 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*, *cli-*

*ent_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. Objek yang bisa diakses dan daftar dari method-method API yang disediakan *Slack* sangatlah 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 bisa dijalankan dengan mengirimkan get request ke endpoint *https://slack.com/api/users.profile.get* dan memerlukan token bertipe pengguna serta scope yaitu users.profile.read. Fungsi dari method ini adalah untuk mendapatkan informasi tentang profil pengguna.
- **users.profile.set** Method ini bisa dijalankan dengan 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. Method ini berfungsi untuk mengubah isi dari profil pengguna.

## 2.3 *Node.js*

*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[2]. 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 runtime 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 menanganin 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 dipanggil di kode program yang akan dibuat perlu dipanggil dengan menggunakan perintah *"require()"* dengan membutuhkan parameter yaitu nama kelas yang akan digunakannya. Contoh dari kelas yang ada di library node.js akan dijabarkan pada subbab

### 2.3.1 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 didesign untuk mendukung banyak

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

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 2.1.

Tabel 2.1: 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, 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 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 options 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 menggunakan url.parser() untuk menjadi url. Http.request() mengembalikan instance dari kelas http.ClientRequest.

## 2.4 Cron

*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.

# BAB 3

# ANALISIS

Pada bab ini akan dijelaskan mengenai analisis dari penggunaan *Microsoft Graph API* dan juga penggunaan dari *Slack API* serta analisis dari penggunaan *cron*.

## 3.1 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 3.1.1 sampai subbab 3.1.4.

### 3.1.1 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** |
|---|---|---|
| *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 3.2.

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** |
|---|---|
| *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 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*.

### 3.1.2 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** |
|---|---|---|
| *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 table3.5, maka akan dihasilkan contoh token seperti pada table 3.7 yang memiliki keterangan dari hasil yang dikembalikan pada table3.8.

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": "AwABAAAAvPM1KaPlrEqdF
SBzjqfTGAMxZGUTdM0t4B4..."
}
```

Tabel 3.8: 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. |

### 3.1.3 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 table3.9 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 ...
0X2tnSQLEANnSPHY0gKcgw
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 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"
}
```

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

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

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

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

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

Tabel 3.12: Tabel parameter *request Refresh Token*

| **Parameter** | | **Deskripsi** |
|---|---|---|
| *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 3.13 yang memiliki keterangan parameter yang dikembalikannya seperti pada table 3.14.

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

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

Tabel 3.14: 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. |

## 3.2 Analisis 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 di*request* tidak akan mengembalikkan hasil dan dianggap sebagai *request* yang *invalid*.

Pada sesi ini akan dibutuhkan *method* dari *Slack API* yang bisa mengubah status yang jika dipelajari lebih lanjut tergabung di dalam *user.profile*. Dari informasi yang didapat dari membaca-baca dokumentasi *online* yang disediakan oleh *Slack API*, 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.

## 3.3 Analisis Cron

Untuk menjalankan aplikasi yang akan dibuat untuk mengambil data dari Microsoft Graph yang berhubungan dengan pengambilan data event, maka diperlukan pengambilan data 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.

# LAMPIRAN A

# KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

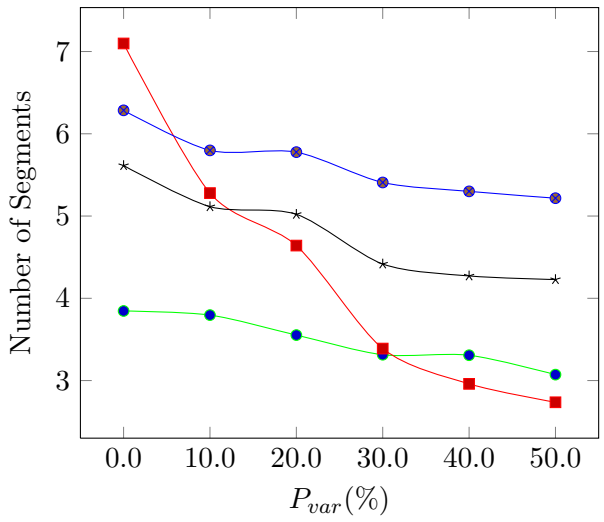
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

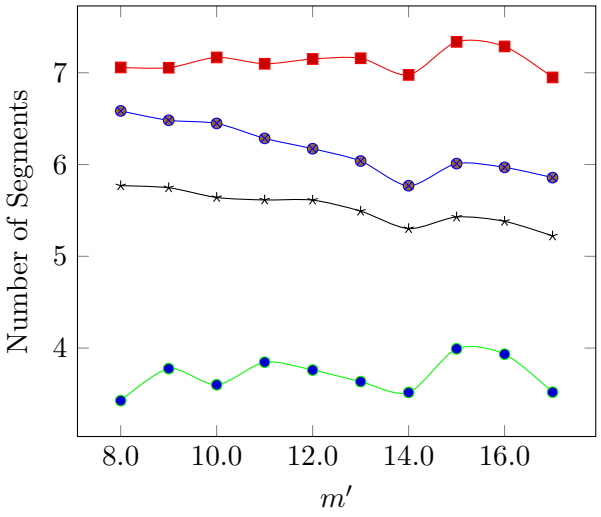
# LAMPIRAN B

# HASIL EKSPERIMEN

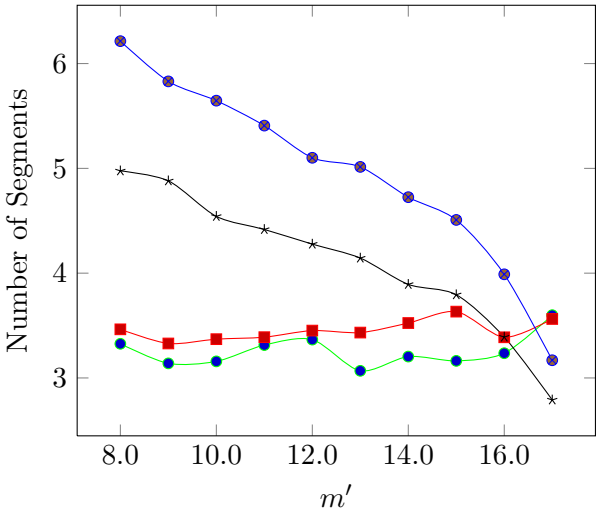
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



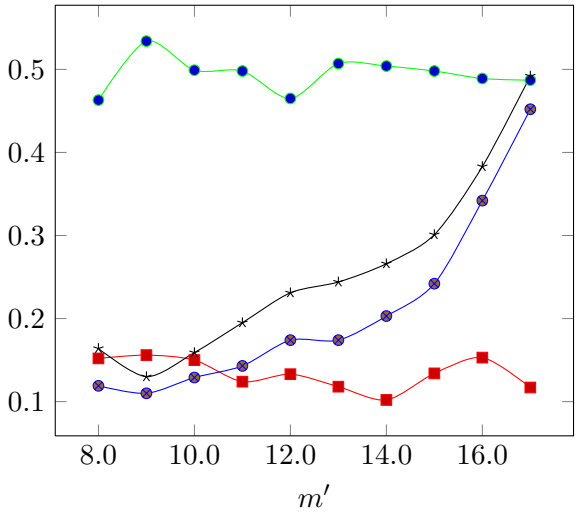

Gambar B.1: Hasil 1




Gambar B.2: Hasil 2




Gambar B.3: Hasil 3




Gambar B.4: Hasil 4