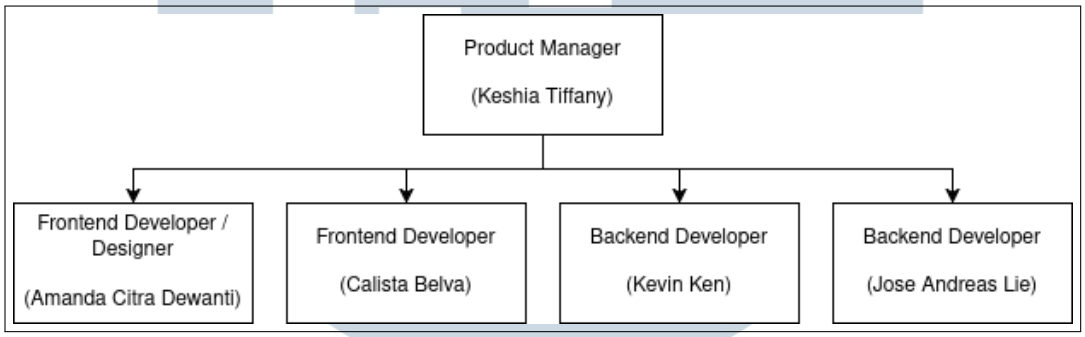
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Dalam pelaksanaan magang di PT Ganda Visi Jayatama, posisi yang ditempatkan adalah sebagai *Backend Developer*. Berikut merupakan struktur organisasi untuk pengembangan *Human Resource Information System* (HRIS) pada gambar 3.1.



Gambar 3.1. Kedudukan dalam struktur organisasi

Dalam pengembangan HRIS terdapat empat anggota tim yang terbagi menjadi dua *Frontend Developer* dan dua *Backend Developer*. Selama pengembangan HRIS, praktik kerja dan koordinasi dipimpin oleh Keshia Tiffany selaku *Product Manager*. *Product Manager* bertugas untuk mengatur, merencanakan, serta menjadwalkan keberlangsungan proyek dari awal hingga akhir [6]. Untuk memantau hasil kinerja dilakukan *weekly meeting* agar tugas yang dilakukan masih sesuai dengan rencana yang sudah ditetapkan dalam *sprint*.

Selama proses pengerjaan, adapun *software-software* yang digunakan dalam pengembangan HRIS seperti Apidog yang digunakan untuk membantu percobaan *REST API* yang telah dikembangkan, DBeaver yang digunakan untuk melihat data-data yang tersimpan ke dalam *database* secara visual, serta Zed yang dipilih sebagai lingkungan pengembangan terintegrasi (IDE) untuk meningkatkan efisiensi penulisan kode.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama magang adalah merancang sistem *Reimbursement* dan *Event* dengan menggunakan *Express.js* untuk pembuatan *Application Programming Interface* (API). Adapun juga pengembangan pada sistem *Clock in* yang digunakan untuk absensi karyawan. Selain itu, dilakukan manual testing terhadap API-API yang telah dikembangkan.

3.3 Uraian Pelaksanaan Magang

Berikut merupakan rangkuman dari kegiatan yang dilakukan per minggu dari tanggal 13 Januari sampai dengan 13 Juli 2025 pada tabel 3.1 dan 3.2.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	*Onboarding* dan mempelajari *boilerplate backend* perusahaan.
2	Mengerjakan pengembangan Backend meliputi validasi form, manajemen token, pengujian API, serta berpartisipasi dalam rapat tim dan retrospektif sprint.
3	Mengerjakan implementasi dan pengembangan fitur paginasi global pada berbagai modul.
4	Membuat struktur database dan API *Reimbursement*, serta memperbaiki *filter pagination global*
5	Melanjutkan pengembangan API *Reimbursement* dan implementasi fitur unggah berkas, meningkatkan sistem absensi dengan menambahkan kolom daily_attendance_id pada tabel dailystandup untuk pencatatan alasan keterlambatan, serta penyempurnaan fungsi paginasi global.
6	Menambahkan parameter baru untuk pagination, memperbaiki *filter roles pada stand up feed*, melakukan UAT untuk CHRIS, dan mengoptimisasi kode pada pagination global.
7	Integrasi API *Reimbursement* ke FE, mengoptimisasi pagination kode, membuat location dan ip untuk clock in.
8	Melanjutkan pembuatan location dan ip untuk clock in serta merubah dari ip ke koordinat, mempelajari minio dan setup minio untuk *file upload*, membuat *reimbursement*.

Tabel 3.2. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

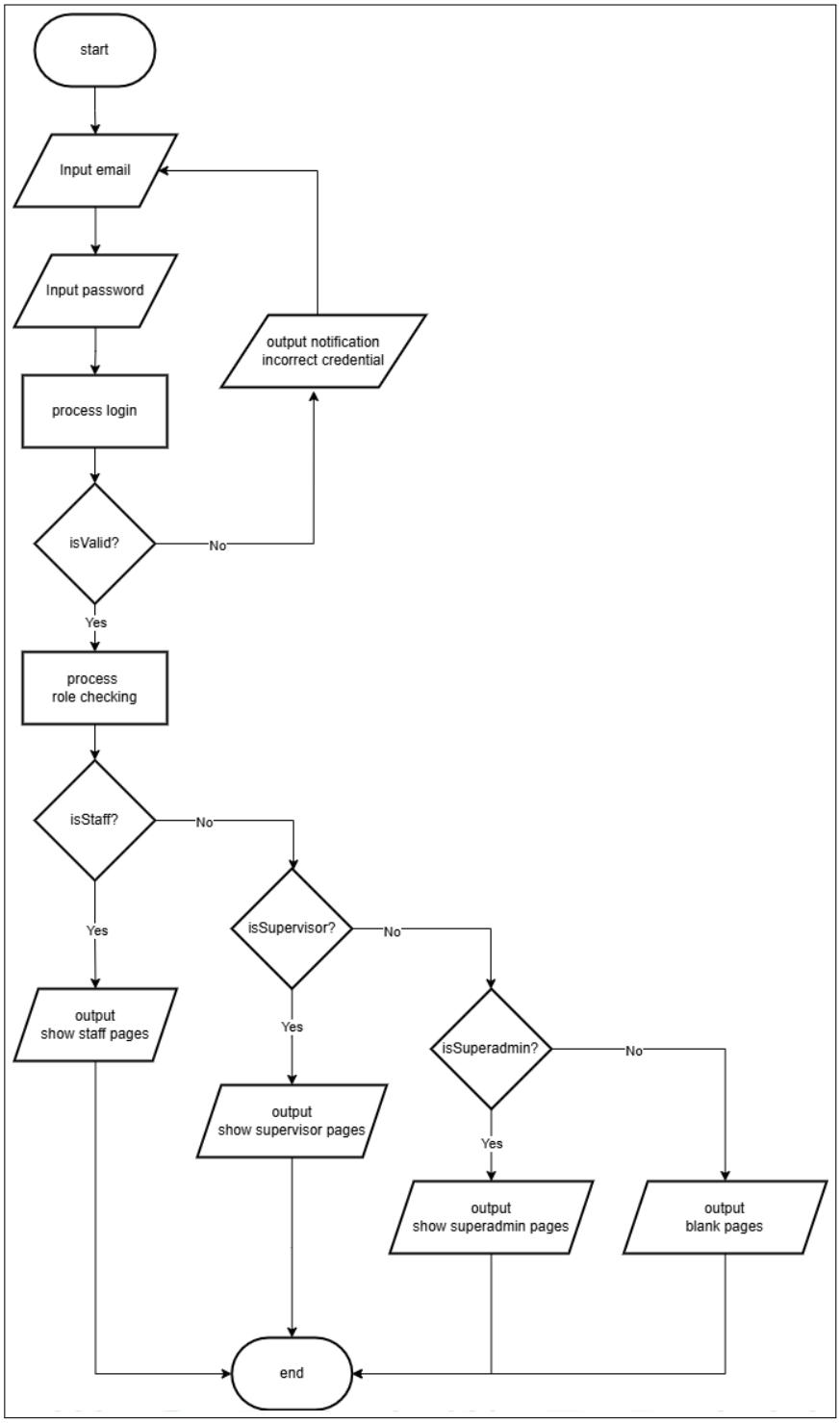
Minggu Ke -	Pekerjaan yang dilakukan
9	Mentesting dan debug minio, merubah dan testing file upload agar lebih kompetibel dengan minio, membuat event table untuk database.
10	Membuat event GET dan CREATE API, menyesuaikan minio konfigurasi dengan server serta testing.
11	Membuat fungsi UPDATE dan DELETE untuk event API, dan melakukan improvement pada event API.
12	Menambahkan filter untuk event dan *improvement*, serta *testing event* fungsi
13	Melakukan showcase untuk hasil yang dikerjakan, membuat event notification, serta memisahkan event type dari event table.
14	Membuat fungsi API untuk event type, memperbaiki notification agar berfungsi ke event type, serta melakukan testing pada event type dan event.
15	Integrasi ke FE untuk event type dan membuat filter untuk event agar hari libur tidak bisa dibuat di hari yang sama.

3.4 Perancangan dan Pengembangan Sistem

Perancangan dan pengembangan sistem HRIS di PT Ganda Visi Jayatama akan menjelaskan perancangan sistem untuk fitur-fitur utama yang dikembangkan selama magang, yaitu sistem *Reimbursement*, sistem *Event*, dan pengembangan sistem *Clock-in*.

3.4.1 Sistem Pengecekan Role

Sistem pengecekan role pada PT Ganda Visi Jayatama menggunakan metode *Role Based Access Control* atau RBAC yang digunakan untuk membatasi setiap modul maupun informasi berdasarkan hak aksesnya sesuai dengan role yang ditentukan [7]. Dapat terlihat pada gambar 3.2.



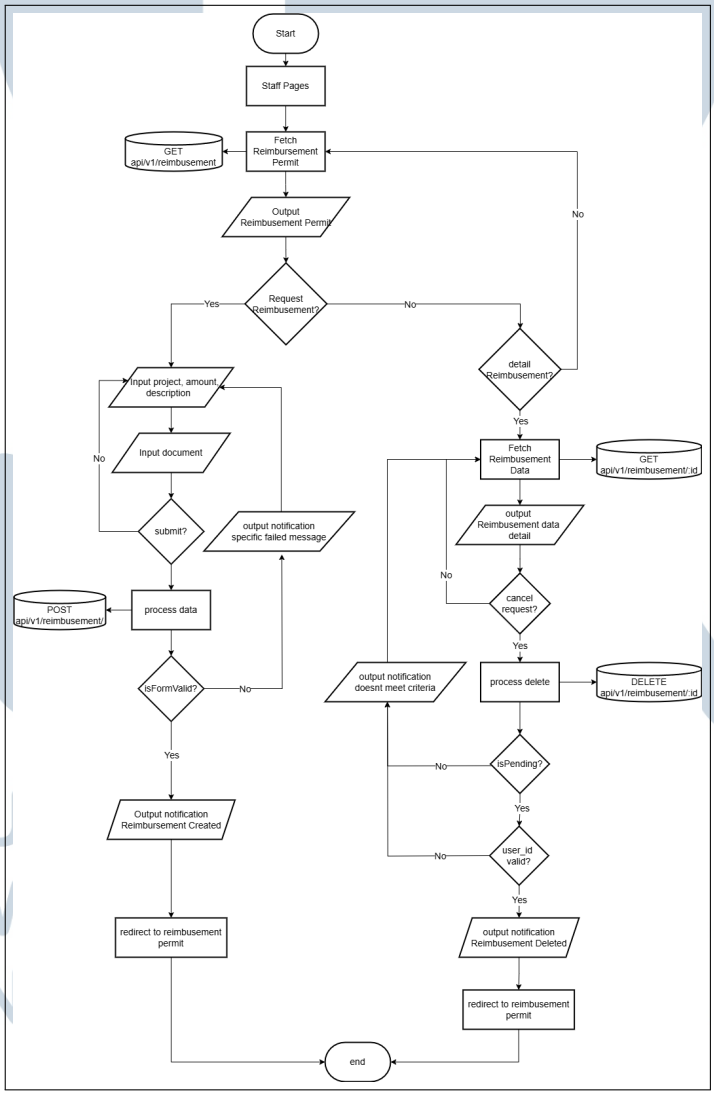
Gambar 3.2. Alur Pengecekan Role

3.4.2 Sistem Reimbursement

Sistem *Reimbursement* dikembangkan untuk mempermudah proses pengajuan dan persetujuan biaya operasional karyawan. Fitur ini memungkinkan karyawan untuk mengajukan penggantian biaya seperti transportasi dan pembayaran hosting server.

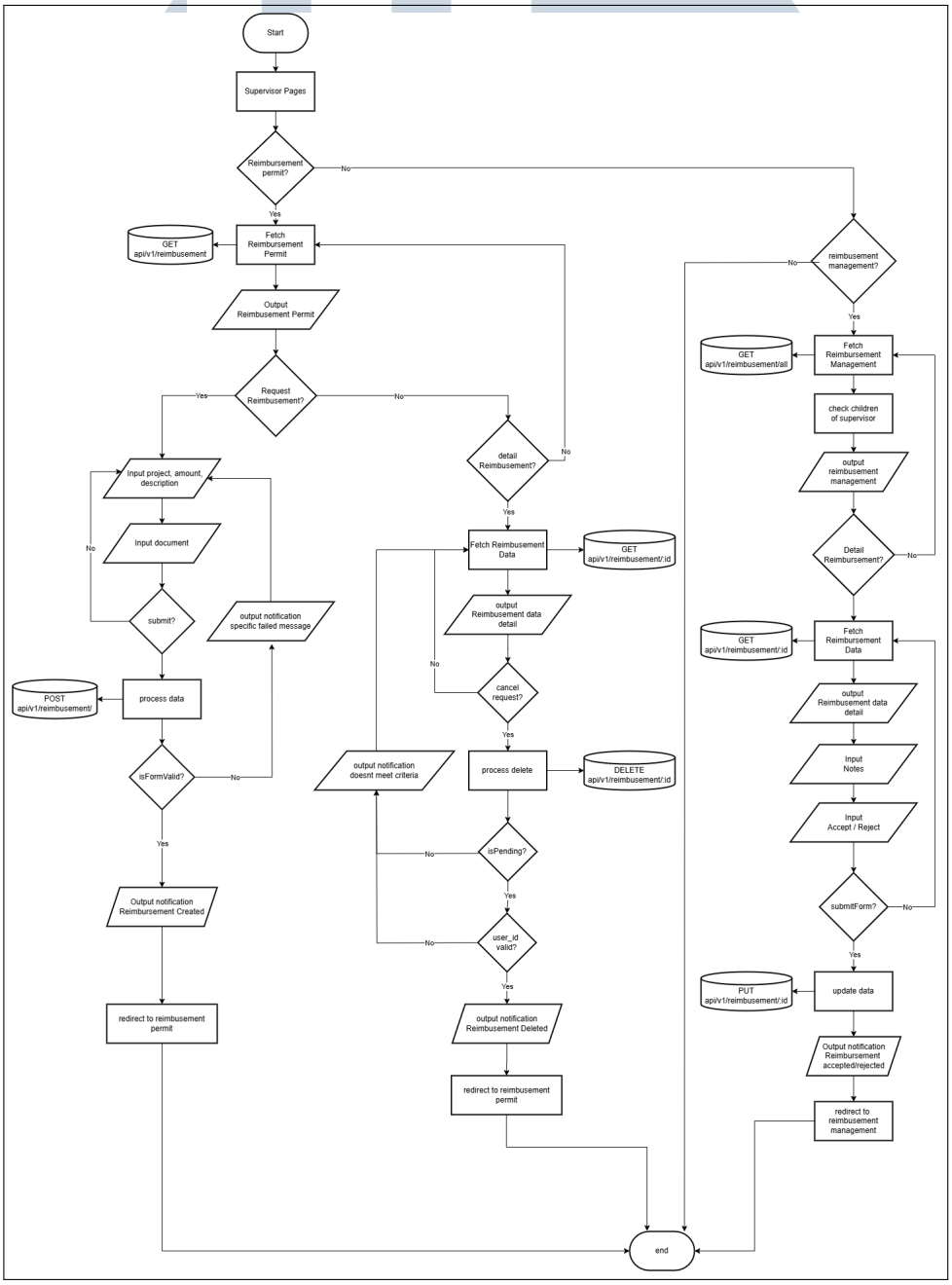
A Alur Sistem Reimbursement

Berikut merupakan alur-alur dari sistem reimbursement yang di antaranya staf, supervisor, dan superadmin.



Gambar 3.3. Alur Staff Reimbursement

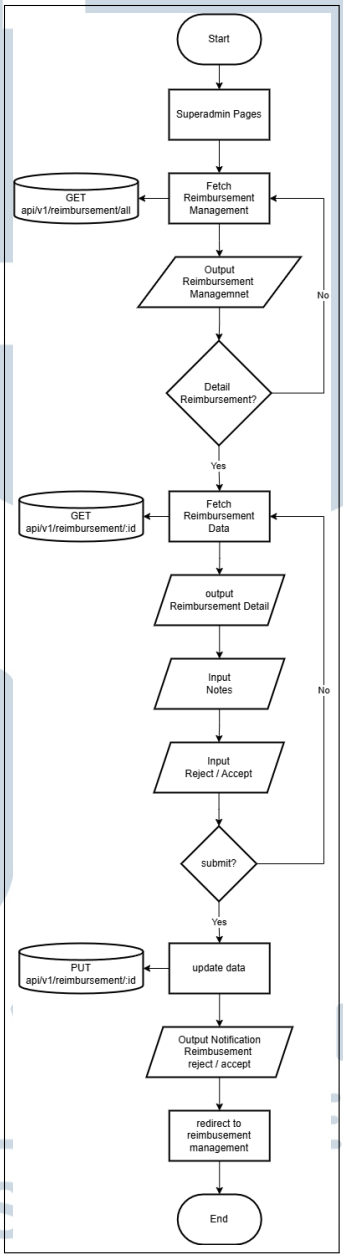
Pada gambar 3.3 merupakan *flowchart* dari alur sistem *reimbursement* untuk keperluan staf. Sehingga, hanya *user-user* yang memiliki *role* sebagai *staff* yang hanya akan mendapatkan akses ke halaman *Reimbursement Permit* dan dapat melihat pengajuan-pengajuan yang telah dilakukan. Dan juga dapat melakukan pengajuan ataupun menghapus pengajuan jika kondisi masih berstatus *pending*.



Gambar 3.4. Alur Supervisor Reimbursement

Pada gambar 3.4 merupakan *flowchart* dari alur sistem *reimbursement* untuk

keperluan *supervisor*. yang di mana supervisor memiliki dua halaman yaitu *Reimbursement Permit* dan *Reimbursement Management*. Pada *Reimbusement Permit* alur kerja-nya sama seperti alur kerja dari *staff* pada gambar 3.3. Namun pada halaman *Reimbursement Management* yang di mana cara kerjanya pada *supervisor* adalah akan mengambil *staff-staff* yang memiliki keterkaitan dengan *supervisor* tersebut.



Gambar 3.5. Alur Superadmin Reimbursement

Pada gambar 3.5 merupakan *flowchart* dari alur sistem *reimbursement*

untuk keperluan *superadmin*. Dimana hanya memiliki satu halaman saja yaitu: *Reimbursement Management*, perbedaan dari *Reimbursement Management* yang dapat diakses oleh *supervisor* dengan *superadmin* adalah *superadmin* dapat melihat seluruh pengajuan yang dilakukan oleh karyawan dari perusahaan.

B Skema Database Reimbursement

Gambar 3.6. Skema Database Reimbursement

Pada Gambar 3.6 merupakan skema dari database *reimbursement* dan terdiri dari tabel-tabel:

1. **reimbursements**: tabel ini merupakan tabel utama dari database reimbursement, yang digunakan untuk menyimpan data-data yang diajukan oleh user. Yang dimana user_id akan mengambil uuid dari user yang mengajukan *reimbursement* dan reviewed_by_id akan menyimpan uuid dari user yang telah menyetujui atau menolak pengajuan reimbursement. Penyimpanan uuid berguna untuk mendapatkan data user secara lengkap.

2. **files**: tabel ini digunakan untuk menyimpan file-file yang diunggah oleh user. yang dimana file yang diunggah akan masuk ke dalam *thrid-party* software yaitu minIO, minIO itu sendiri merupakan sebuah *object storage* yang diperuntukan untuk menyimpan data-data yang berkaitan dengan media seperti jpg, png, csv, xlsx, dan lain-lain [8].

3. **reimbursement_files**: tabel ini merupakan tabel pivot dari reimbursement dan file yang digunakan untuk mengidentifikasi file yang di upload dengan data reimbusement yang ditujukan melalui uuid dari masing masing tabel.

C Spesifikasi API Reimbursement

Berikut merupakan *endpoint-endpoint* yang dikembangkan untuk *Reimbursement* pada tabel 3.3.

Tabel 3.3. 6 Endpoint yang digunakan untuk Fitur Reimbursement

No	Endpoint	Method	Body (Request)	Response (body)
1	`/reimbursement/all`	GET	–	`status code, message, data`
2	`/reimbursement`	GET	–	`status code, message, data`
3	`/reimbursement/:id`	GET	–	`status code, message, data`
4	`/reimbursement/`	POST	`project_id, amount, description, file_ids`	`status code, message, data`
5	`/reimbursement/:id`	PUT	`status, notes`	`status code, message, data`
6	`/reimbursement/:id`	DELETE	–	`status code, message`

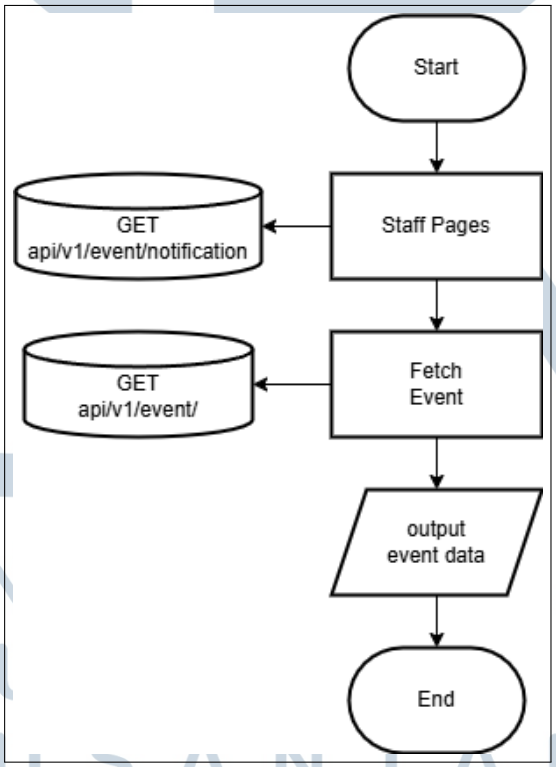
Pada *endpoint* yang memerlukan parameter `id`, nilai `id` akan dikirimkan melalui parameter URL dengan nilai berupa `uuid` dari data *reimbursement*. Sementara itu, data yang dikembalikan dalam `response body` mencakup sejumlah atribut, antara lain: `id`, `app_id`, `amount`, `description`, `status`, `notes`, `user`, `reviewed_by`, `project_product`, `files`, `created_at`, dan `updated_at` .

3.4.3 Sistem Event

Sistem *Event* dikembangkan untuk mempermudah pengguna dalam melihat jadwal yang akan datang seperti *meeting*. Dan juga menghindari tabrakan antarjadwal yang sudah ada.

A Alur Sistem Event

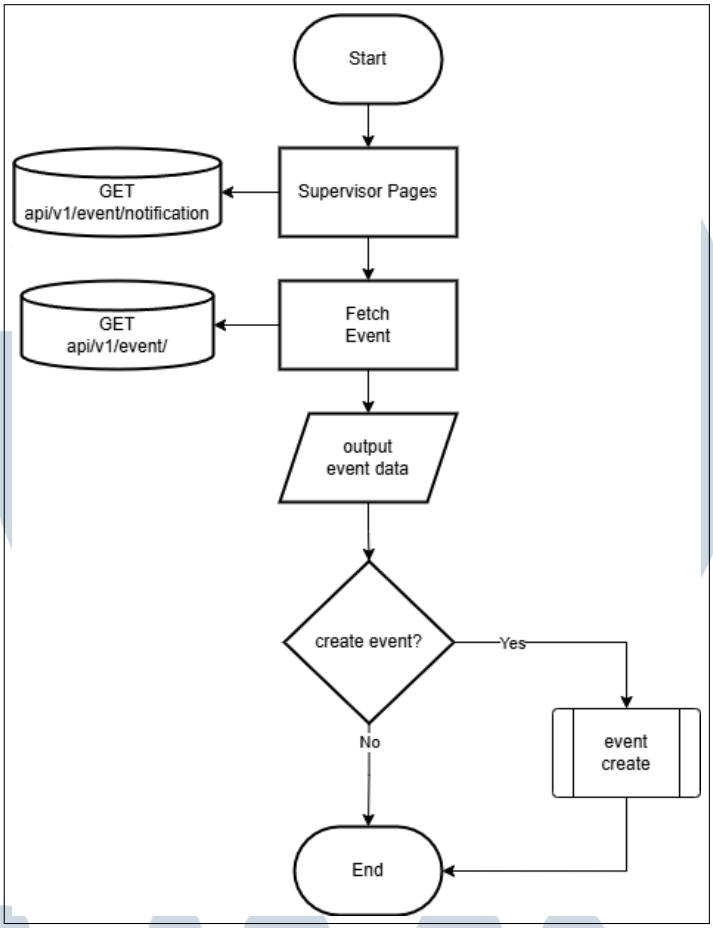
Berikut merupakan alur-alur dari sistem event yang di antaranya staf, supervisor, dan superadmin.



Gambar 3.7. Alur Staf Event

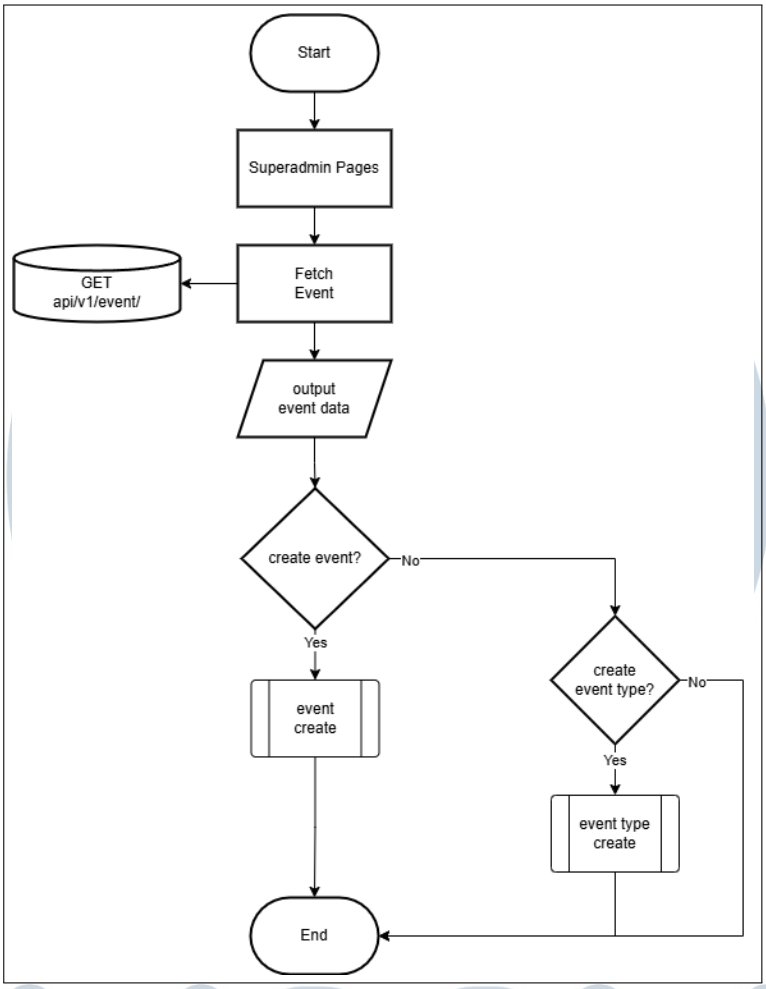
Pada gambar 3.7 merupakan alur *event* yang dikhususkan untuk staf, yang di mana untuk staf itu sendiri hanya dapat melihat event yang sedang berlangsung.

Dan tidak dapat membuat *event*.



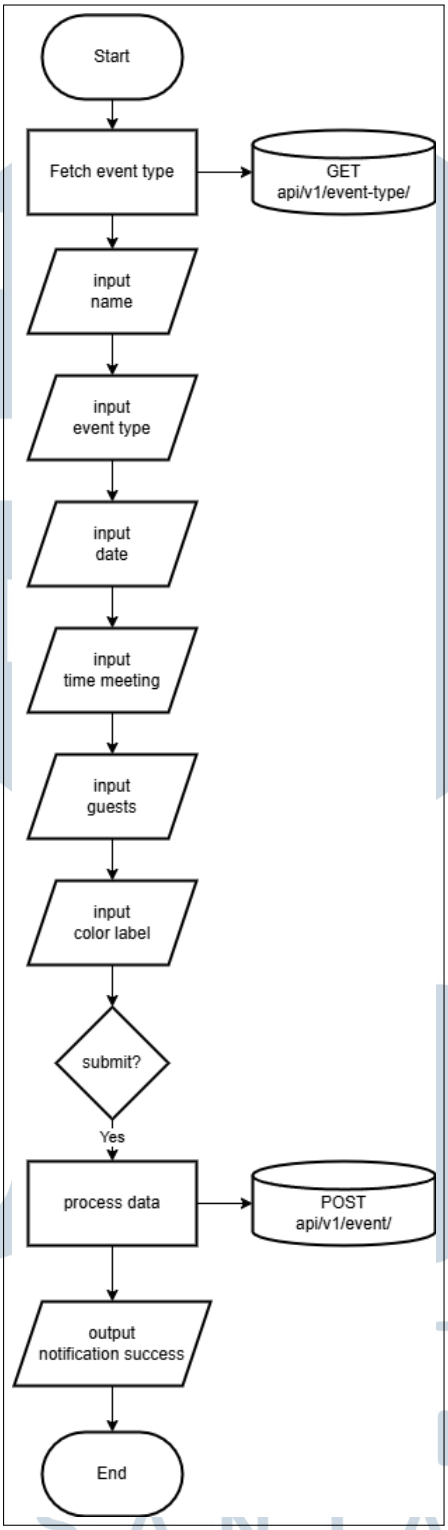
Gambar 3.8. Alur Supervisor Event

Pada gambar 3.8 atau alur untuk *supervisor* itu sendiri mengenai *event*, nantinya *supervisor* dapat membuat *event* yang akan diselenggarakan. Namun, *event* yang dapat dibuat hanyalah *event* yang bertipe kan *meeting*. Untuk flow dari pembuatan *event* dapat dilihat pada gambar 3.10.



Gambar 3.9. Alur Superadmin Event

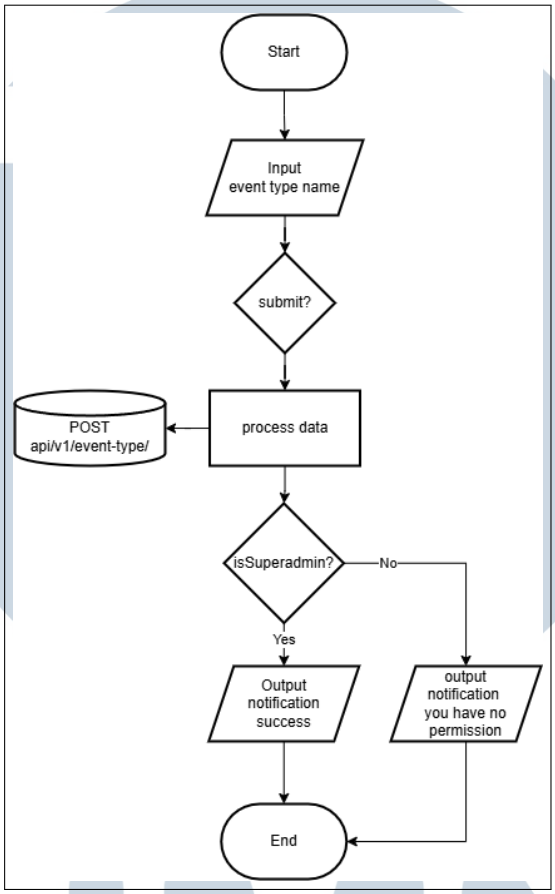
Untuk alur *event* untuk *superadmin* itu sendiri pada gambar 3.9. *Superadmin* dapat melakukan pembuatan *event* yang di mana *superadmin* dapat membuat *event* bertipekan *holiday* dan *meeting*. Dan juga hanya *superadmin* yang dapat membuat *event type* yang dapat dilihat pada gambar 3.11.



Gambar 3.10. Alur Create Event

Pada gambar 3.10, saat awal pertama kali ingin melakukan pembuatan *event* akan dilakukannya pengambilan data *event type* dan pada saat *superadmin* ingin

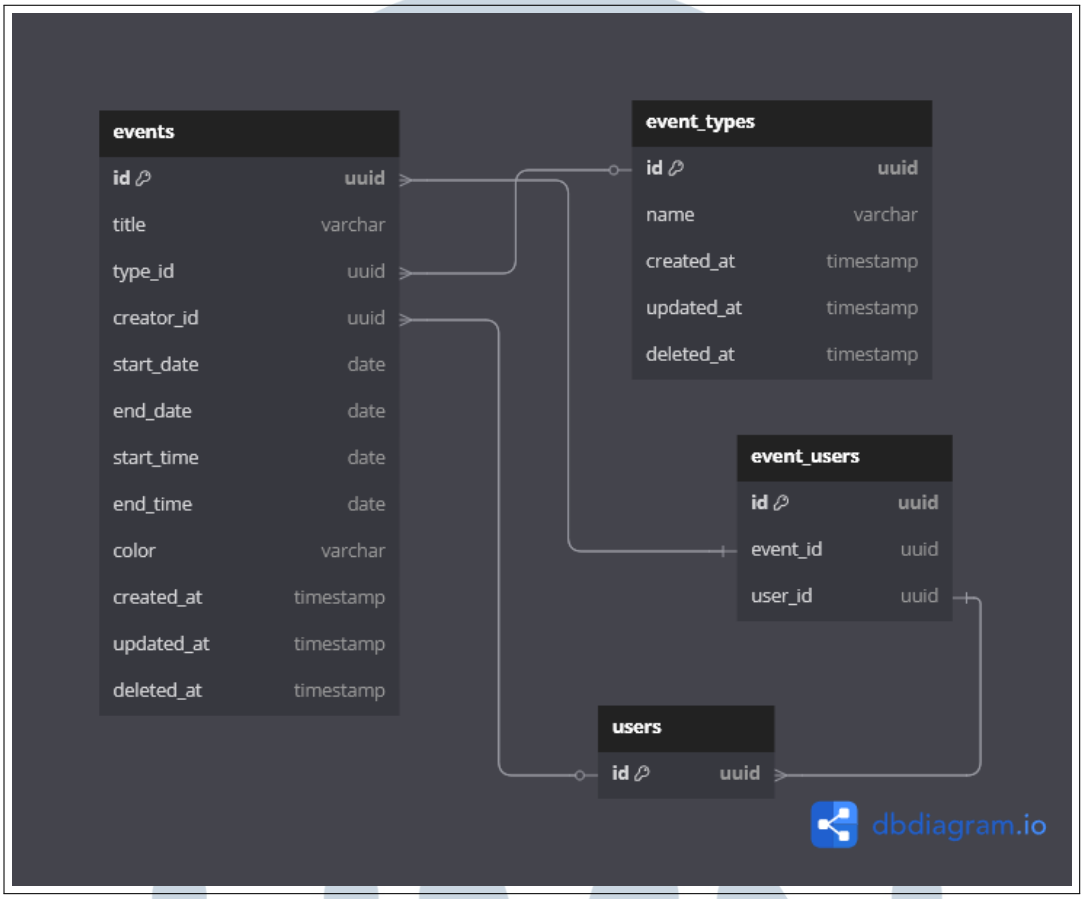
membuat *holiday* hanya field-field tertentu yang akan muncul seperti *name, event type* dan *date*.



Gambar 3.11. Alur Create Event Type

Untuk pembuatan *event type* sendiri yang terdapat pada gambar 3.11, hanya membutuhkan nama dari *event type* yang ingin dibuat, dan sistem akan memvalidasi apakah *role* yang membuat *event type* sesuai atau tidak. Jika tidak, sistem akan memberikan pesan `You have no permission`.

B Skema Database Event



Gambar 3.12. Skema Database Event

Pada Gambar 3.12 merupakan skema dari database *event* dan terdiri dari tabel-tabel:

1. **events**: Tabel utama yang digunakan untuk menyimpan data event yang dibuat oleh pengguna. Kolom `creator_id` mereferensikan ke tabel `users`, yang menyimpan `uuid` serta data pengguna terkait.
2. **event_types**: Tabel ini digunakan untuk mendefinisikan tipe dari setiap event. Setiap event akan mereferensikan tipe ini melalui atribut `type_id`. Relasi antara `event_types` dan `events` bersifat *one-to-many*, di mana satu tipe event dapat digunakan oleh banyak event.
3. **event_users**: Tabel pivot yang menghubungkan antara tabel `events` dan `users` yang bersifat *many-to-many*, dimana banyak event bisa mempunyai

banyak user dan sebaliknya. Tabel ini digunakan untuk menyimpan daftar pengguna yang diundang dalam sebuah event, melalui kolom `user_id` dan `event_id`.

C Spesifikasi API Event & Event Type

Berikut merupakan *endpoint-endpoint* yang dikembangkan untuk *Event* pada tabel 3.4, 3.5, dan 3.6.

Tabel 3.4. 6 Endpoint yang Digunakan untuk Fitur Event

No	Endpoint	Method	Body (Request)	Response (body)
1	`/event/`	POST	`title`, `type_id`, `start_date`, `end_date`, `start_time`, `end_time`, `color`, `user_ids`	`status code`, `message`, `data`
2	`/event/`	GET	–	`status code`, `message`, `data`
3	`/event/:id`	PATCH	`title`, `start_time`, `end_time`, `color`	`status code`, `message`, `data`
4	`/event/:id`	DELETE	-	`status code`, `message`

Tabel 3.5. 6 Endpoint yang Digunakan untuk Fitur Event

No	Endpoint	Method	Body (Request)	Response (body)
5	`/event/:id`	GET	-	`status code`, `message`, `data`
6	`/event/notification`	GET	–	`status code`, `message`, `data`

Pada *endpoint event* yang memerlukan parameter `id`, nilai `id` akan dikirimkan melalui parameter URL dengan nilai berupa `uuid` dari data *event*. Sementara itu, data yang dikembalikan dalam `response body` mencakup sejumlah atribut, antara lain: `id`, `title`, `start_date`, `end_date`, `start_time`, `end_time`, `color`, `creator`, `guests`, `event_type`, `created_at`, dan `updated_at`.

Tabel 3.6. 4 Endpoint yang Digunakan untuk Fitur Event Type

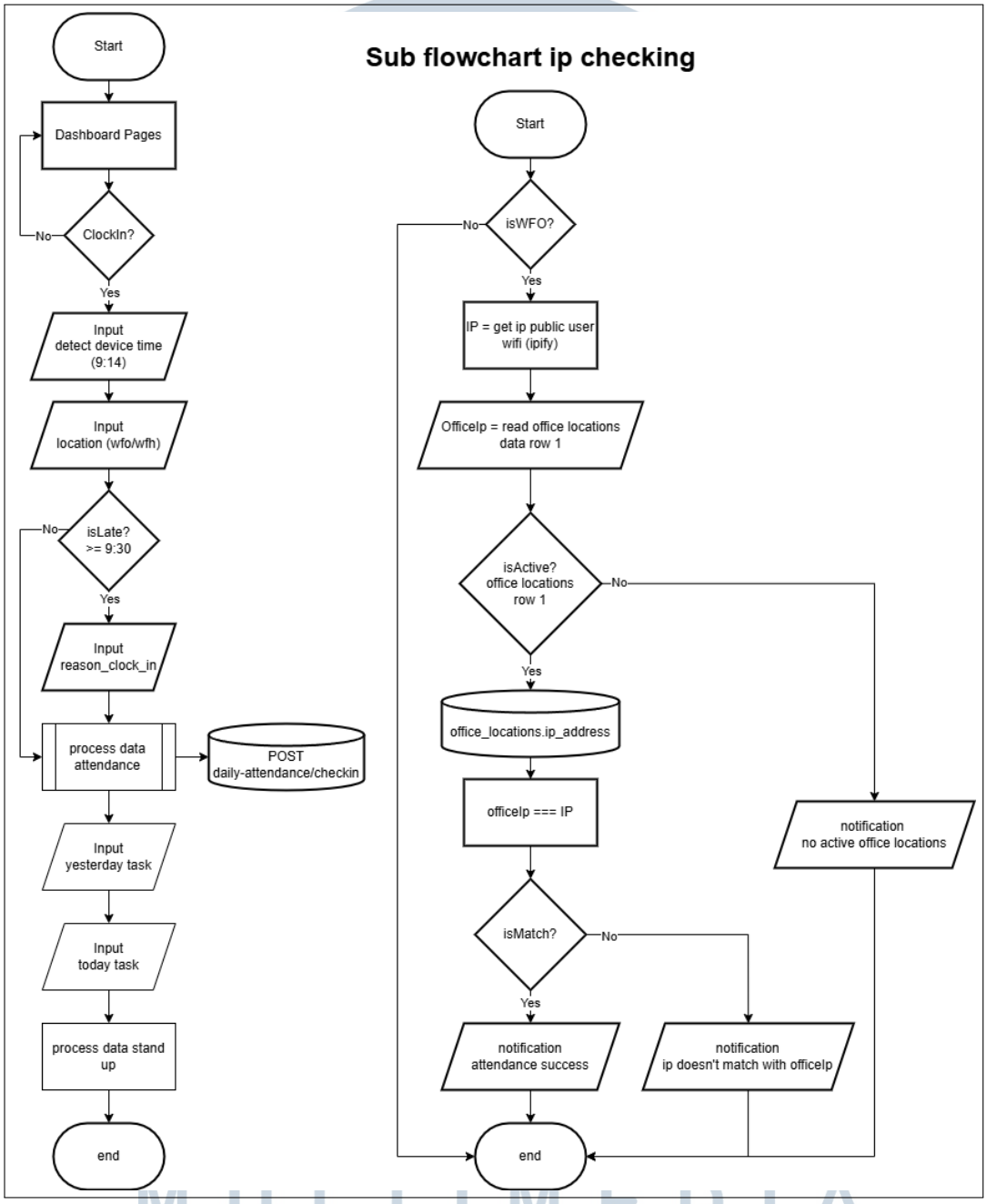
No	Endpoint	Method	Body (Request)	Response (body)
1	`/event-type/`	GET	–	`status code`, `message`, `data`
2	`/event-type/`	POST	name	`status code`, `message`, `data`
3	`/event-type/:id`	PUT	name	`status code`, `message`, `data`
4	`/event-type/:id`	DELETE	-	`status code`, `message`

Lalu pada *endpoint event-type* yang memerlukan parameter `id`, nilai `id` akan dikirimkan melalui parameter URL dengan nilai berupa `uuid` dari data *event-type*. Sementara itu, data yang dikembalikan dalam `response body` mencakup sejumlah atribut, antara lain: `id`, `name`, `created_at`, `updated_at`.

3.4.4 Sistem Clock In

Sistem clock in difokuskan pada pengembangan fitur absensi yang memungkinkan pengguna melakukan absensi menggunakan *IP address*.

A Alur Sistem Clock In

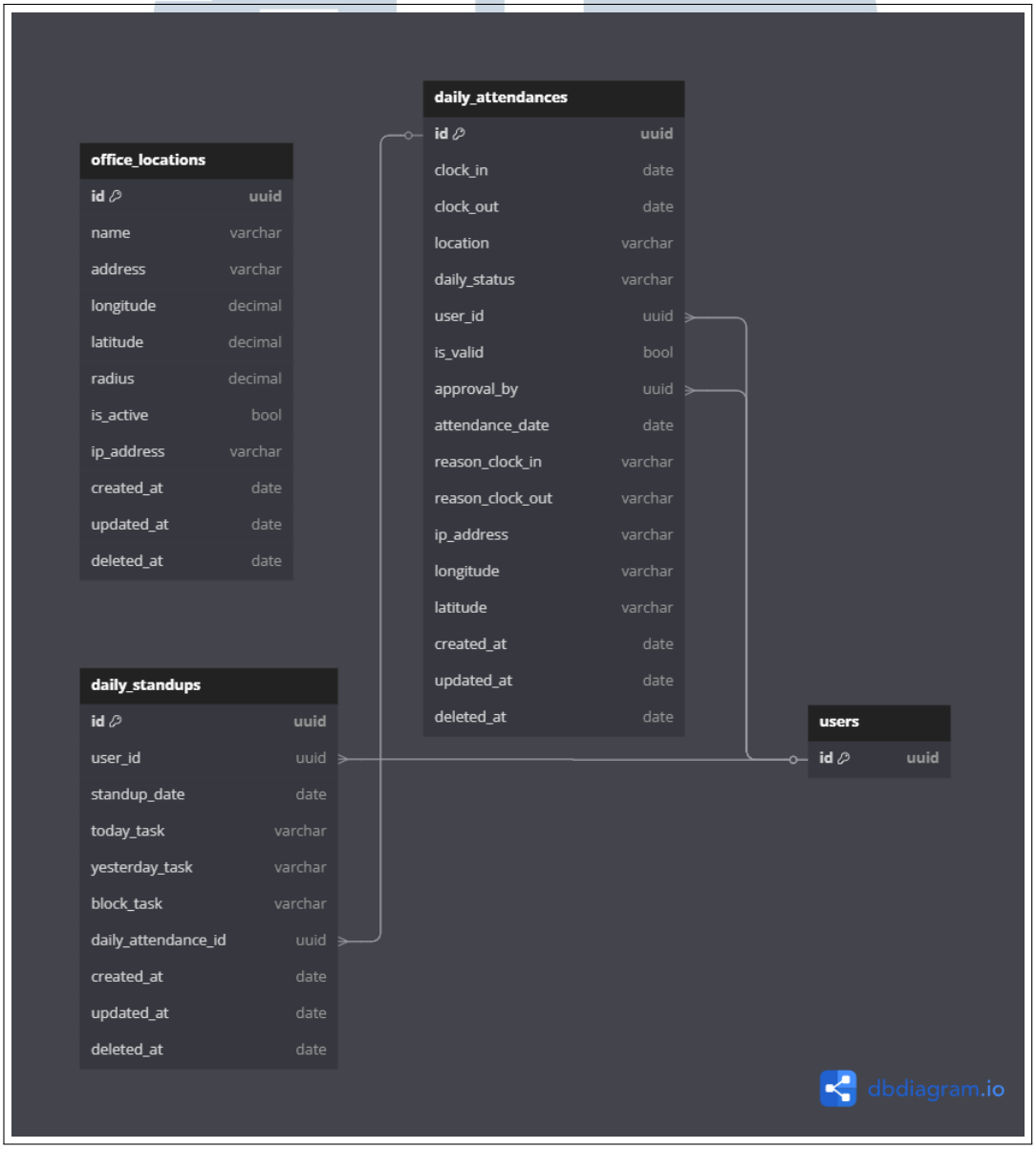


Gambar 3.13. Alur Sistem Clock In

Pada gambar 3.13 merupakan alur flowchart dari sistem absensi yang ada pada PT Ganda Visi Jayatama setelah dilakukan pengembangan. Pada saat pengguna terlambat atau tidak ketika akan mengirim data, akan dilakukan pengecekan melalui *endpoint* `POST /daily attendances/checkin`, yang di

mana pengecekan ini akan mengambil *ip public wifi* pengguna yang digunakan. Lalu, membandingkan dengan *ip address* dari *wifi* kantor. Jika *ip* dari kantor dengan *ip* pengguna berbeda, akan terjadi penolakan pada sistem dan harus menggunakan wifi kantor terlebih dahulu untuk melakukan absensi dan baru bisa mengisi *standup* yang akan dikerjakan hari ini.

B Skema Database Clock In



Gambar 3.14. Skema Database Clock In

Pada Gambar 3.14 merupakan skema dari database *event* dan terdiri dari tabel-tabel:

1. **daily_attendances**: Ini merupakan tabel utama yang digunakan untuk menyimpan data absensi yang dibuat oleh pengguna. Kolom `user_id` dan `approval_by` mereferensikan ke tabel `users`, yang menyimpan `uuid` serta data pengguna terkait.

2. **daily_standups**: Tabel ini digunakan untuk melakukan penyimpanan data mengenai pekerjaan yang dilakukan pada waktu kemarin dan hari ini. Setiap `daily_standups` akan direferensikan terhadap `daily_attendance_id` karena *standup* hanya bisa dilakukan setelah melakukan absensi. Lalu, `user_id` digunakan untuk mengetahui user yang memiliki standup tersebut.

3. **office_locations**: Tabel ini merupakan master data yang digunakan untuk menyimpan alamat perusahaan. Yang dimana pada atribut `ip_address` akan menyimpan wifi kantor sehingga ketika absensi hanya bisa dilakukan di dalam radius kantor.

C Spesifikasi API Clock In

Berikut merupakan *endpoint-endpoint* yang dikembangkan untuk *Clock in* pada tabel 3.7.

Tabel 3.7. 6 Endpoint yang Digunakan untuk Fitur Clock In

No	Endpoint	Method	Body (Request)	Response (body)
1	/daily-attendances/	GET	-	status code, message, data
2	/daily-attendances /all/user	GET	–	status code, message, data
3	/daily-attendances /checkin	POST	reason_clock_in, location, longitude, latitude	status code, message, data
4	/daily-attendances /checkout	PUT	-	status code, message
5	/daily-attendances /approval/	PUT	ids	status code, message
6	/daily-attendances /ip/:id	PUT	ip_address	status code, message, data

Pada *endpoint clock in* untuk melakukan absensi akan menggunakan *endpoint* `checkin` yang di mana akan menggunakan pihak ketiga untuk mendapatkan *public ip* dari pengguna dan akan dicocokkan dengan *ip address* yang berada pada tabel `office_locations`. Sementara itu, data yang dikembalikan dalam `response body` mencakup sejumlah atribut, antara lain: `id`, `clock_in`, `clock_out`, `location`, `daily_status`, `user_id`, `reason_clock_in`, `is_valid`, `attendance_date`, `ip_address`, `longitude`, `latitude`, `approval_by`, `created_at`, dan `updated_at`.

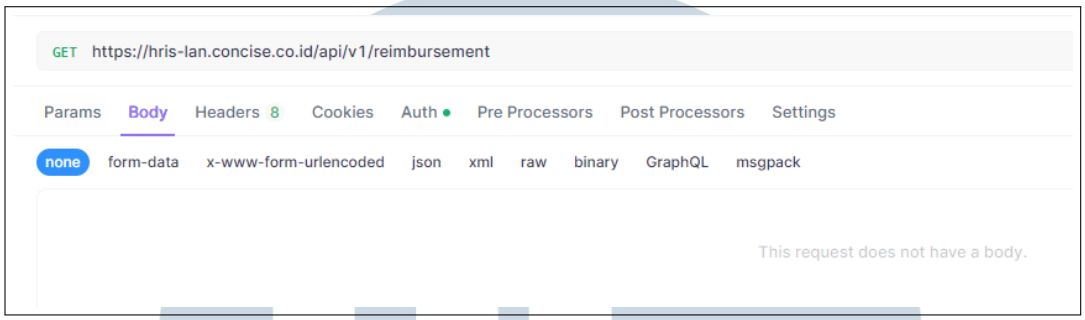
3.5 Implementasi

Pada bagian implementasi, akan menampilkan visualisasi dari modul-modul yang telah dikembangkan. Untuk tampilan dikerjakan oleh tim `Front-end`.

3.5.1 Implementasi Reimbursement

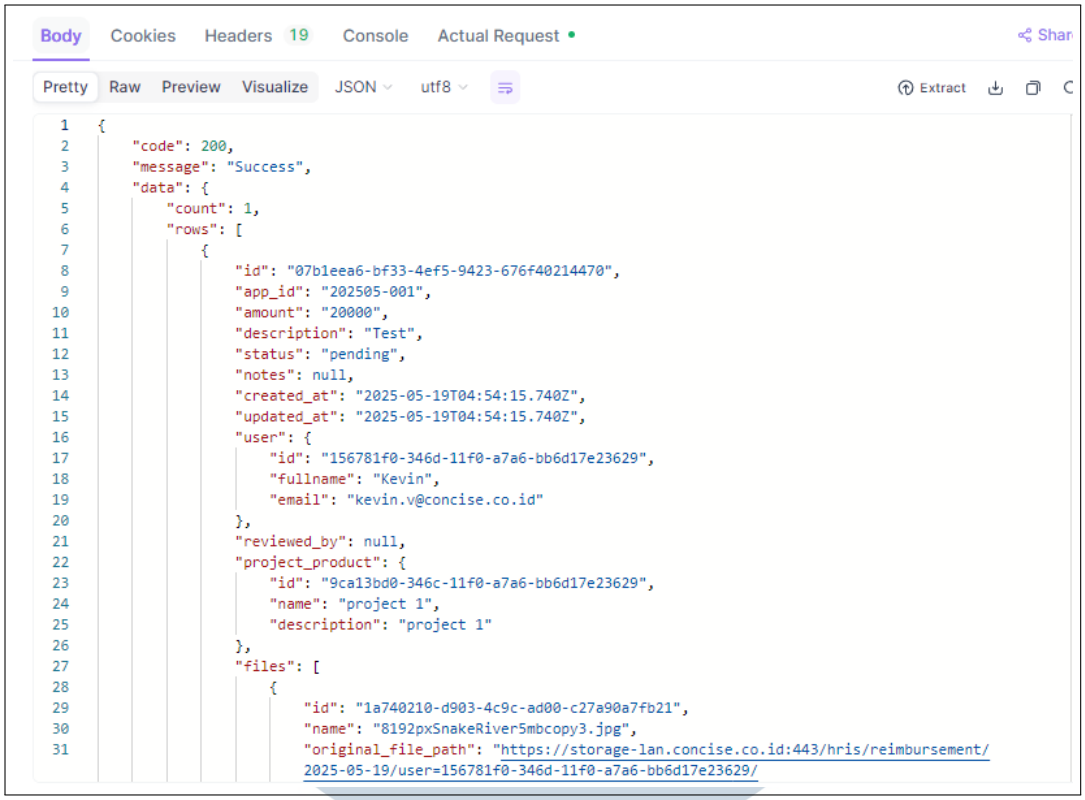
Pengembangan Reimbursement telah masuk ke tahap *development* pada PT Ganda Visi Jayatama, berikut merupakan hasil akhir yang telah selesai dikembangkan.

A Endpoint GET /reimbursement & /reimbursement/all



Gambar 3.15. Request dari GET endpoint

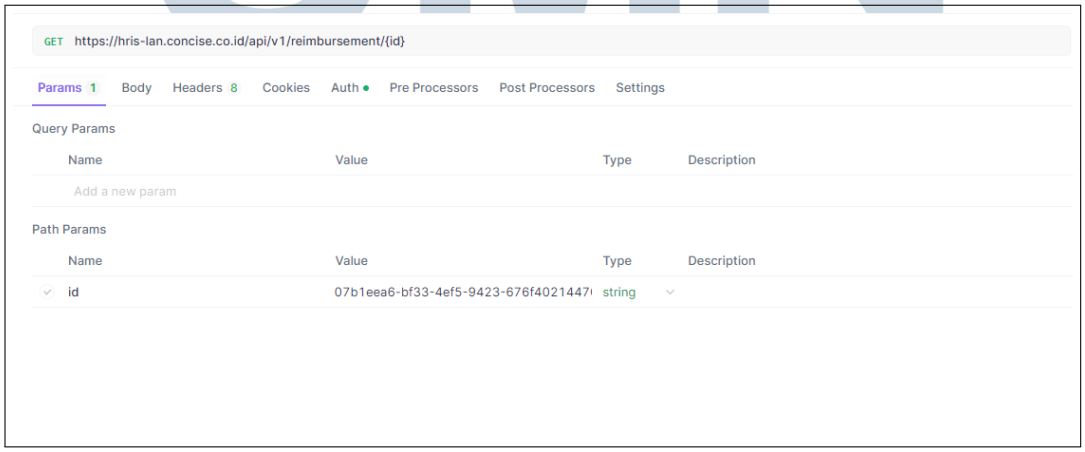
Pada gambar 3.15 merupakan *request body* yang dibutuhkan untuk mengambil data dari reimbursement dengan metode GET, dan pengambilan data ini berlaku untuk *endpoint* `/reimbursement` dan `/reimbursement/all`. Perbedaan dari kedua *endpoint* ini adalah untuk *endpoint* `/reimbursement` hanya akan menampilkan data pengajuan untuk pengguna itu sendiri, dan yang mendapatkan akses terhadap *endpoint* ini hanya *role supervisor* dan *staff*. Dan *endpoint* `/reimbursement/all` memiliki dua mekanisme yang di mana untuk *role superadmin* akan menampilkan semua data pengajuan yang diajukan, tetapi untuk *role supervisor* hanya akan menampilkan data pengajuan yang di mana suatu *staff* terikat dengan *supervisor* tersebut. Dan berikut merupakan *response body* dari kedua *endpoint* tersebut pada gambar 3.16.



Gambar 3.16. Response GET reimbursement

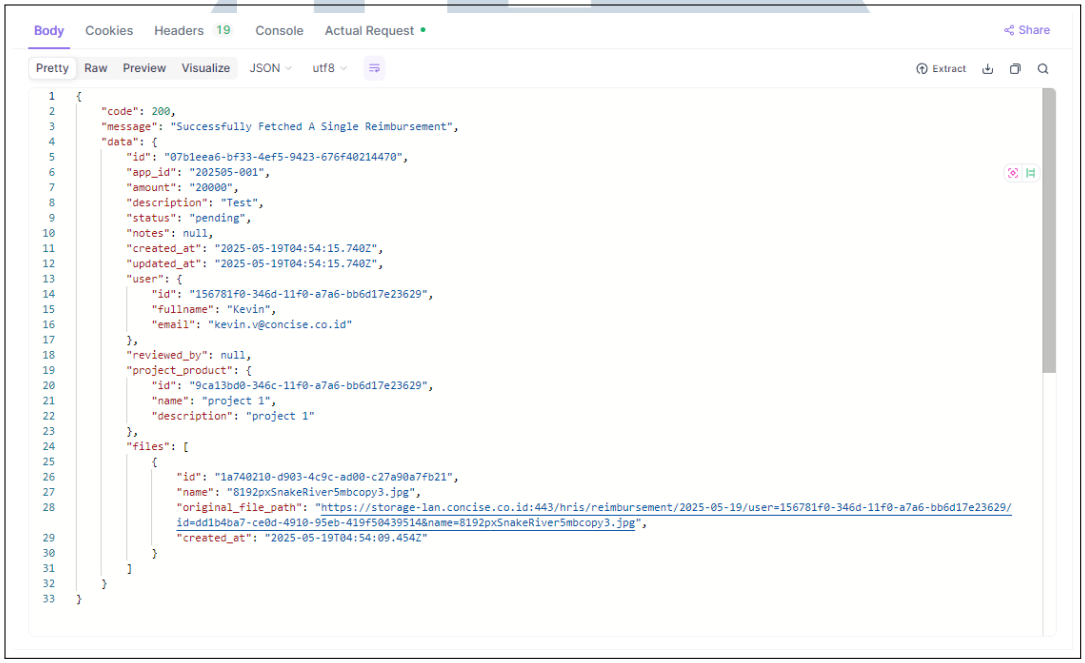
Pada *response body* jika *role staff* mencoba akses *endpoint* `/reimbursement/all` kode akan berubah menjadi 403 dan pesan akan menunjukkan `You do not have permission to access this resource.`

B Endpoint GET /reimbursement/:id



Gambar 3.17. Request GET /reimbursement/:id

Pada gambar 3.17 merupakan *request* yang dibutuhkan untuk melihat data *reimbursement* secara detail, dengan memasukkan `uuid` ke dalam *parameter endpoint*. Sehingga, *supervisor, superadmin,* maupun pengguna itu sendiri yang mengajukan permohonan *reimbursement* dapat melihat secara detail. Berikut merupakan *response body* dari *reimbursement detail* pada gambar 3.18.



Gambar 3.18. Response GET /reimbursement/:id

C Endpoint POST /reimbursement/

Pada *endpoint* POST hanya pengguna yang memiliki *role supervisor* dan *staff* yang dapat membuat pengajuan *reimbursement*.

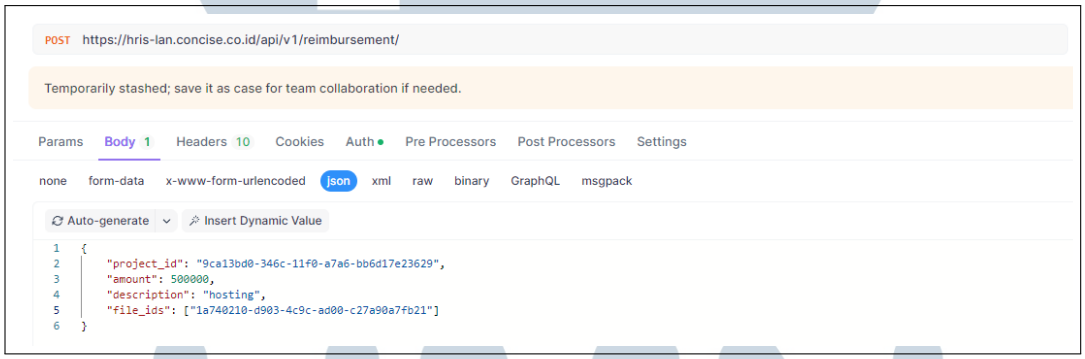
```
private async generateUniqueAppId(transaction) {
        await sequelize.query('LOCK TABLE reimbursements IN EXCLUSIVE MODE', { transaction });

        const currentDate = new Date();
        const yearMonth = currentDate.toISOString().substring(0, 7).replace('-', '');

        const lastReimbursement = await Reimbursement.findOne({
            where: {
                app_id: {
                    [Op.like]: `${yearMonth}-%`,
                },
            },
            paranoid: false,
            order: [['app_id', 'DESC']],
            lock: transaction.LOCK.UPDATE,
            transaction,
        });

        let sequenceNumber = 1;
        if (lastReimbursement) {
            const lastSequence = parseInt(lastReimbursement.app_id.split('-')[1]);
            sequenceNumber = lastSequence + 1;
        }

        const newAppId = `${yearMonth}-${String(sequenceNumber).padStart(3, '0')}`;

        const existingReimbursement = await Reimbursement.findOne({
            where: {
                app_id: newAppId,
            },
            paranoid: false,
            transaction,
        });

        if (existingReimbursement) {
            throw new ApiError(
                httpStatus.CONFLICT,
                'Failed to generate unique app_id, sequence already exists'
            );
        }

        return newAppId;
    }

```

Gambar 3.19. Function Generate APP ID

Pada gambar 3.19 merupakan fungsi yang digunakan untuk membuat `app_id` secara dinamis. Fungsi ini bekerja dengan cara mengunci tabel `reimbursemnets` terlebih dahulu agar tidak terjadi konflik data saat proses pembuatan `ID`. Lalu, pada variabel `yearMonth` akan mengambil tanggal saat ini melalui variabel `currentDate` dan mengubah formatnya menjadi kombinasi tahun dan bulan (YYYYMM). Kemudian, pada variabel `lastReimbursement` fungsi ini akan mencari data *reimbursement* terakhir menggunakan sintaks Sequelize yaitu `findOne`, dengan dibatasi pola `yearMonth` pada bagian `where`. Hal ini

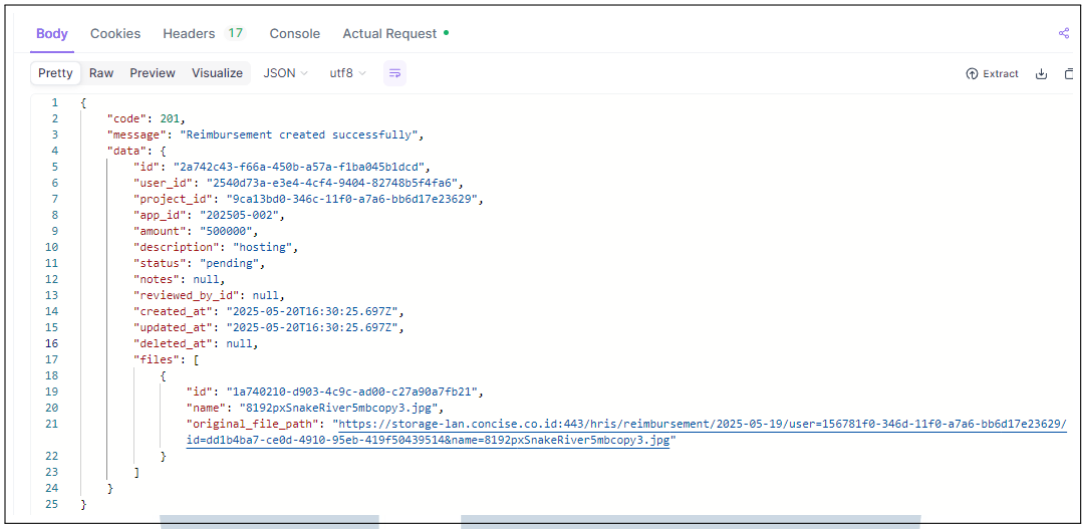
memungkinkan sistem untuk hanya mencari `app_id` yang sesuai dengan bulan dan tahun saat ini. Sehingga, ketika bulan berganti, pencarian tidak akan menggunakan data pada bulan sebelumnya dan urutan angka secara otomatis akan dimulai kembali dari angka satu. Jika ditemukan data dengan `app_id` dibulan yang sama, maka angka urutan akan dinaikkan satu; namun jika tidak ditemukan, maka angka akan dimulai dari satu yang terletak pada variabel `sequenceNumber`.

Selanjutnya, `ID` baru disusun ke dalam format `YYYYMM-XXX` di mana `XXX` merupakan angka urutan yang di mana merupakan tiga format angka. Lalu, jika `app_id` tersebut sudah terbuat akan dilakukan pengecekan pada variabel `existingReimbursement` di mana pada sintaks `where` akan membandingkan `app_id` dengan `app_id` yang baru dibuat. `paranoid: false` berfungsi untuk tetap melakukan pengecekan pada data *reimbursement* yang sudah di *soft delete*. Dan jika terdapat `app_id` yang sama akan memunculkan *error message* pada *response body*.



Gambar 3.20. Request POST /reimbursement/

Pada gambar 3.20 merupakan atribut-atribut yang dibutuhkan untuk membuat *reimbursement* yang dimana `project_id` akan mengambil `uuid` dari tabel `project_products` dan `amount` merupakan total pengeluaran yang ingin diajukan, `description` merupakan keterangan dari pengajuan itu dilakukan. `file_ids` itu sendiri akan menyimpan `uuid-uuid` dari file yang telah di *upload* dalam bentuk *array* tujuannya agar pengguna yang melakukan pengajuan dapat melakukan *upload* media lebih dari satu dan maksimal dari file *upload* itu sendiri adalah lima. Berikut merupakan *response body* dari hasil pembuatan *reimbursement* pada gambar 3.21.

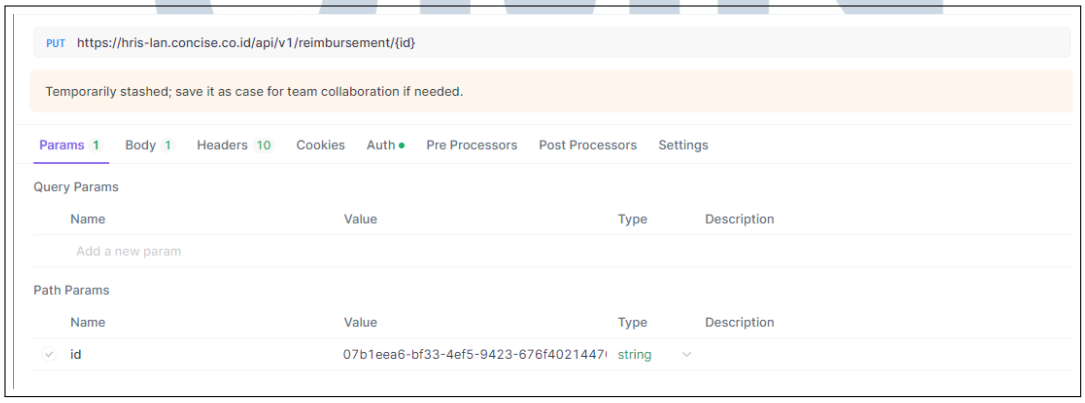


Gambar 3.21. Response POST /reimbursement/

Pada *response body* dalam pembuatan *reimbursement* jika terjadi masalah akan memberikan beberapa *error message* dengan kode HTTP yang berbeda-beda antara lain: ketika terjadi kesalahan saat pembuatan `app_id`, gagal dalam pembuatan *reimbursement*, akan menampilkan status kode HTTP 502, jika terjadi *error* dari sisi klien akan memberikan status kode HTTP 400 seperti, mengubah status saat pembuatan, meng-upload file melebihi batas.

D Endpoint PUT /reimbursement/:id

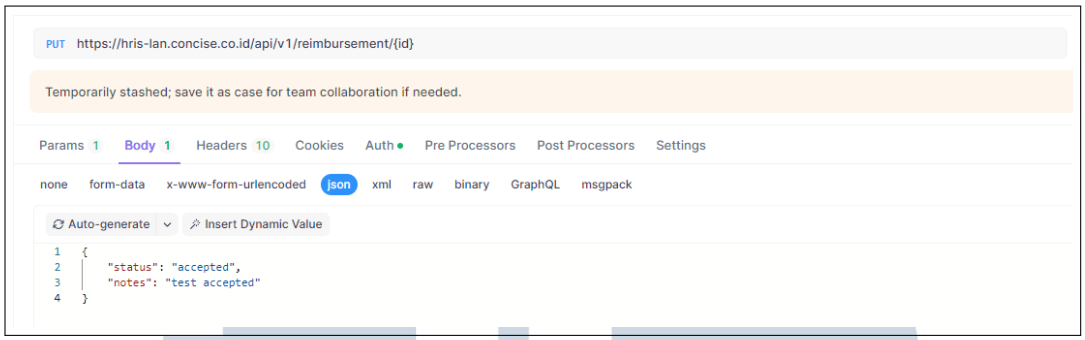
Pada *endpoint* ini yang dapat melakukan *update* status pada *reimbursement* hanya pengguna yang memiliki *role* sebagai *supervisor* ataupun *superadmin*.



Gambar 3.22. Enter Caption

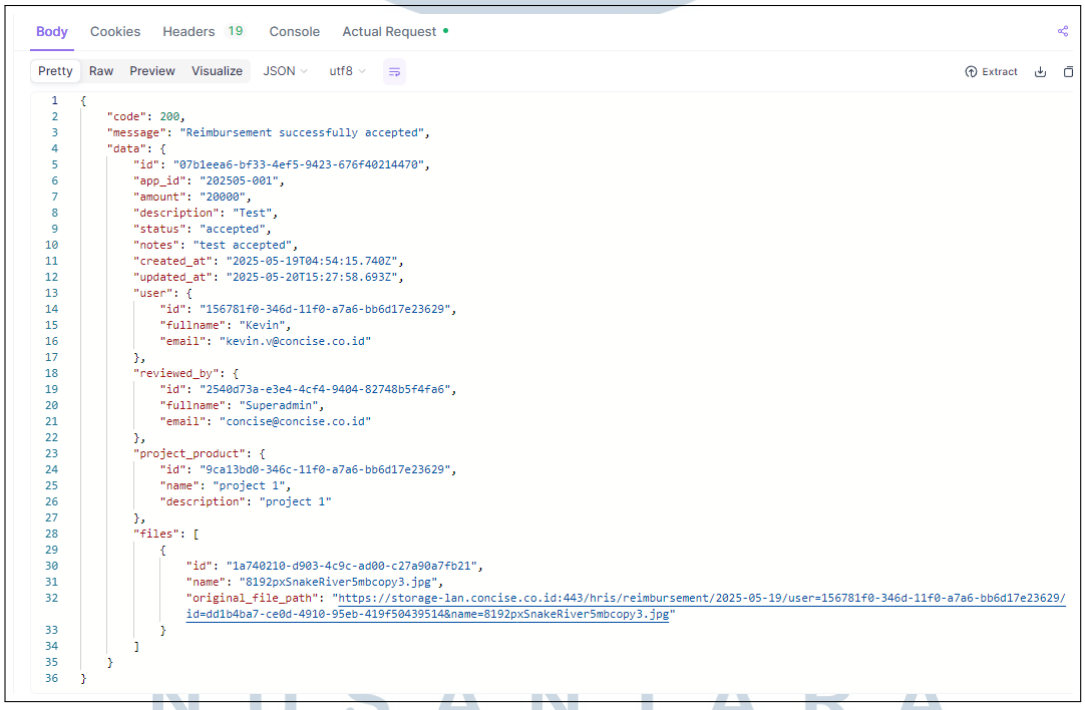
Pada gambar 3.22 merupakan parameter yang dibutuhkan untuk melakukan

update dengan cara memasukkan nilai dari `uuid` pada *reimbursement*.



Gambar 3.23. Request PUT /reimbursement/:id

Pada gambar 3.23 merupakan *request body* yang digunakan untuk melakukan *update* pada pengajuan yang diajukan dari pengguna lain. *Update* ini hanya akan meng-update status dari pengajuan *reimbursement* dari status awal yaitu `pending`, status ini hanya bisa melakukan `accepted` atau `rejected` pada *reimbursement*. Dan pada gambar 3.24 merupakan *response body* dari *endpoint* `/reimbursement/:id`.



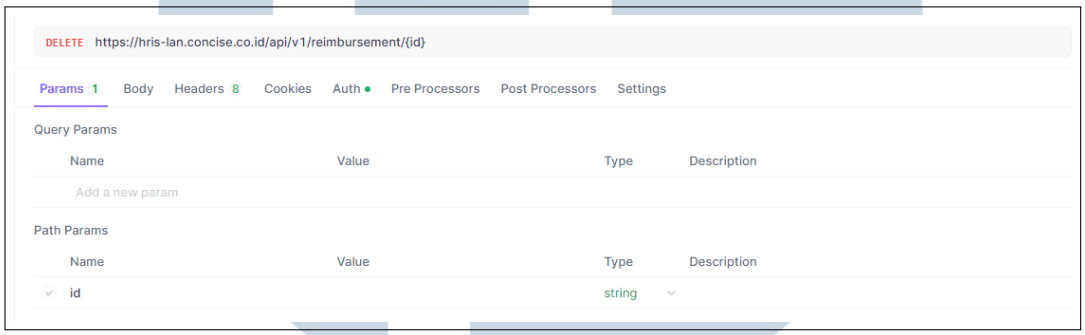
Gambar 3.24. Response PUT /reimbursement/:id

Pada *response body* dalam melakukan *update reimbursement* memiliki beberapa HTTP status kode yang berbeda jika terjadi *error* seperti, HTTP status

kode 403 jika *staff* ingin melakukan update, HTTP status kode 400 jika status yang diupdate tidak sesuai dengan yang seharusnya, dan HTTP status kode 404 di mana `uuid` dari *reimbursement* tidak valid.

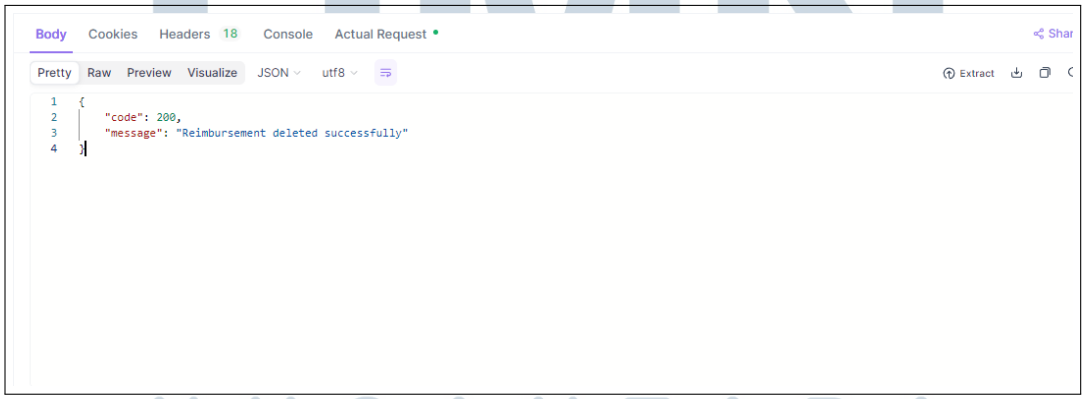
E Endpoint DELETE /reimbursement/:id

Pada *endpoint delete* ini pengguna yang memiliki *role staff* maupun *supervisor* akan dapat melakukan *cancel request*, jika status masih menunjukkan `pending`.



Gambar 3.25. Request DELETE /reimbursement/:id

Pada gambar 3.25, untuk mengdelete data *reimbursement* hanya membutuhkan `uuid` dari data *reimbursement*. Dan berikut merupakan *response body* dari *endpoint* `/reimbursement/:id` dengan metode `DELETE` pada gambar 3.26.



Gambar 3.26. Response DELETE /reimbursement/:id

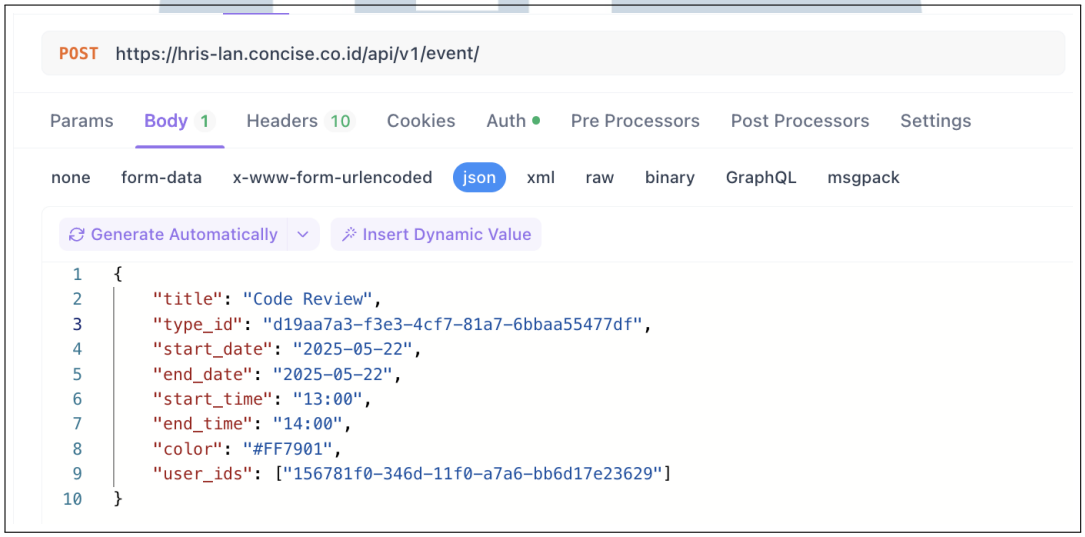
Pada *response body* dalam melakukan *delete reimbursement* ada beberapa HTTP status kode selain 200 antara lain, HTTP status kode 403 jika pengguna

mencoba menghapus data *reimbursement* dari pengguna lain atau pengguna ingin menghapus *reimbursement* yang sudah `accepted` ataupun `rejected`.

3.5.2 Implementasi Event & Event Type

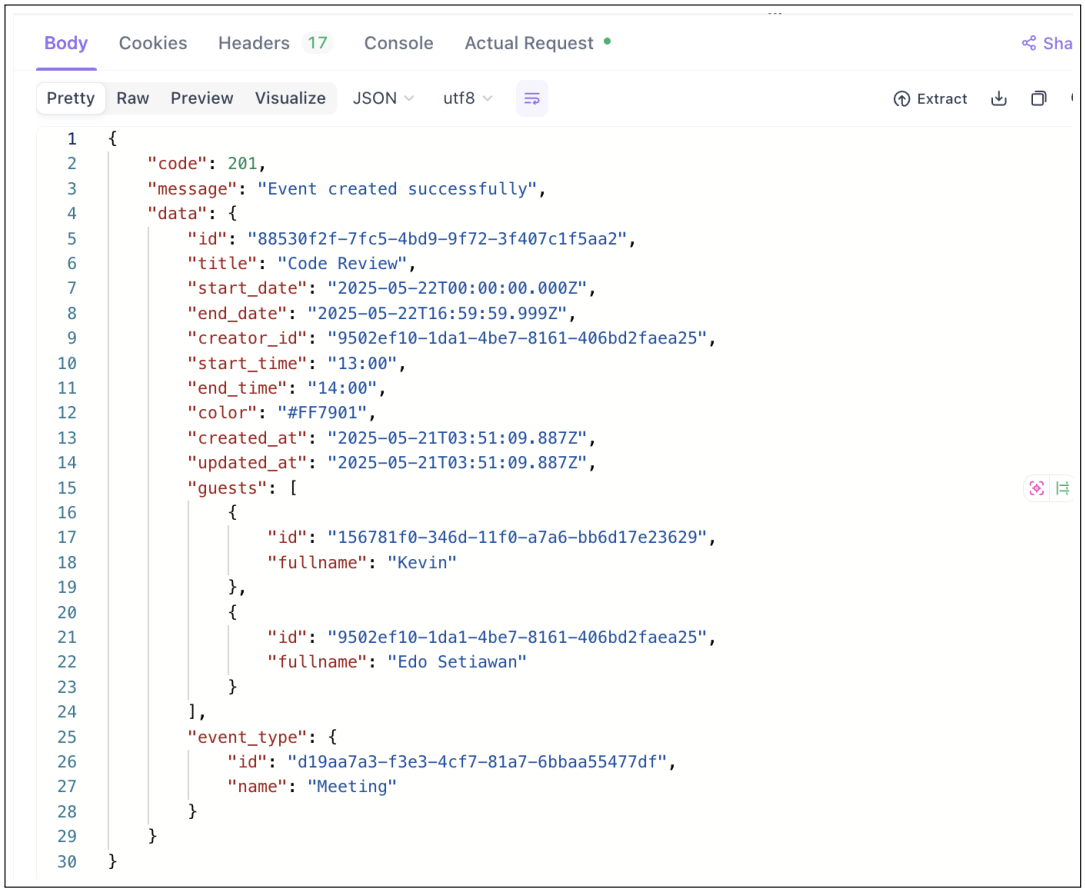
A Endpoint POST /event/

Dalam pembuatan *event* hanya pengguna yang memiliki *role supervisor* dan *superadmin* yang dapat membuat.



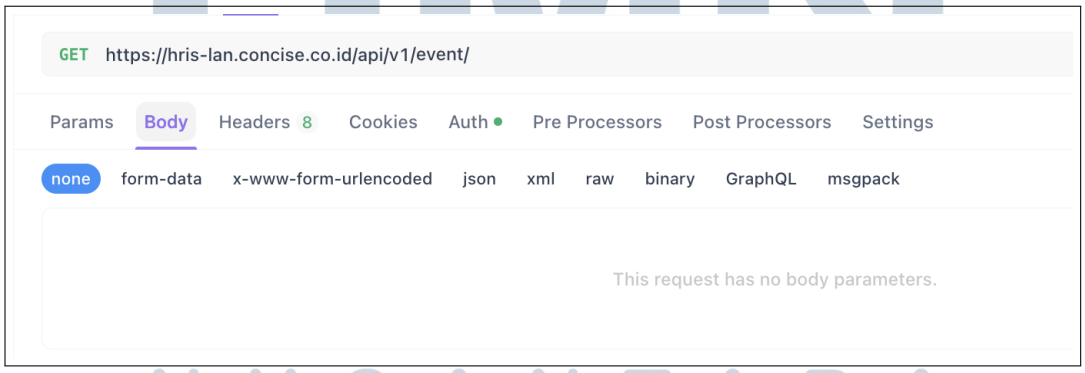
Gambar 3.27. Request POST /event/

Pada Gambar 3.27, merupakan atribut-atribut yang dibutuhkan untuk membuat *event* pada atribut `type_id` akan membutuhkan `uuid` yang nilainya diambil dari data pada tabel `event type`. Lalu untuk `user_ids` akan mengambil *uuid* dari `id` pada tabel `users`. Dalam pembuatan *event* memiliki beberapa aturan antara lain, pada saat pembuatan *meeting* dan *holiday* di waktu yang sama dan hari yang sama akan memberikan HTTP status kode 409 dengan *message* `A meeting already exists at the specified time.` Dan berikut merupakan *response body* dari pembuatan *event* pada gambar 3.28.



Gambar 3.28. Response POST /event/

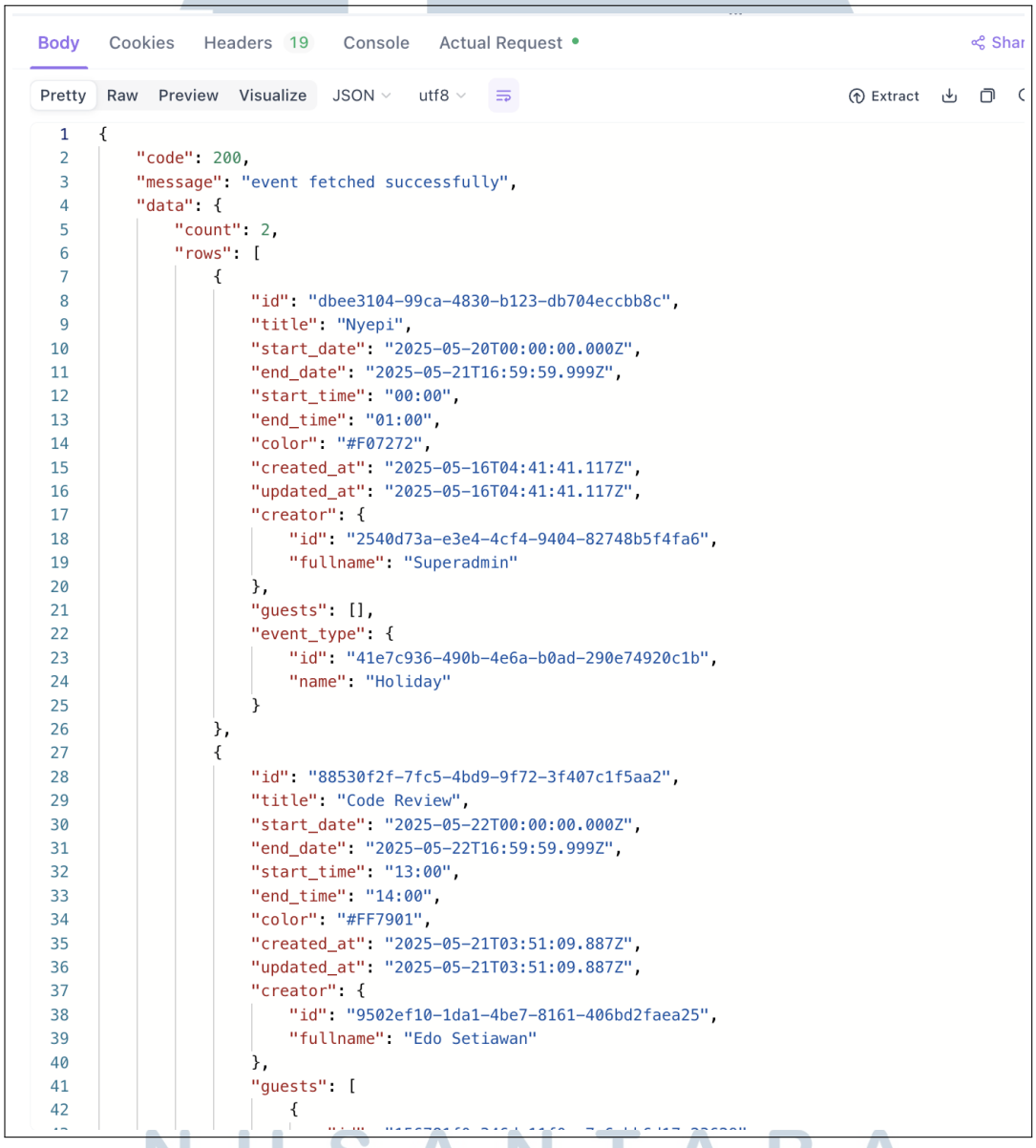
B Endpoint GET /event/



Gambar 3.29. Request GET /event/

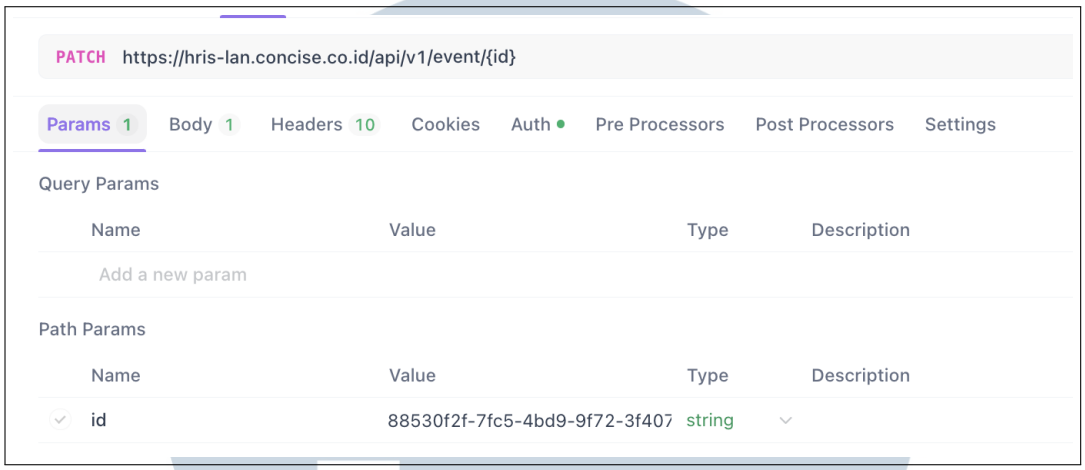
pada *endpoint* `/event/` atau yang dapat dilihat pada gambar 3.29, tidak membutuhkan request body untuk mengambil data. Cara kerja dari GET event

adalah menampilkan data *event* yang di mana pengguna itu terkait. Sehingga, ketika ada pengguna lain yang tidak terikat pada suatu *event* tertentu seperti *meeting*. Maka pengguna tersebut tidak akan mendapatkan data *meeting* tersebut pada *event* pengguna. Lalu, untuk melihat pengguna siapa saja yang terkait dapat dilihat pada gambar 3.30 pada atribut `guests`. Namun, untuk *holiday* itu sendiri tetap akan menampilkan ke seluruh pengguna tanpa membatasi untuk pengguna siapa.



Gambar 3.30. Response GET /event/

C Endpoint PATCH /event/:id

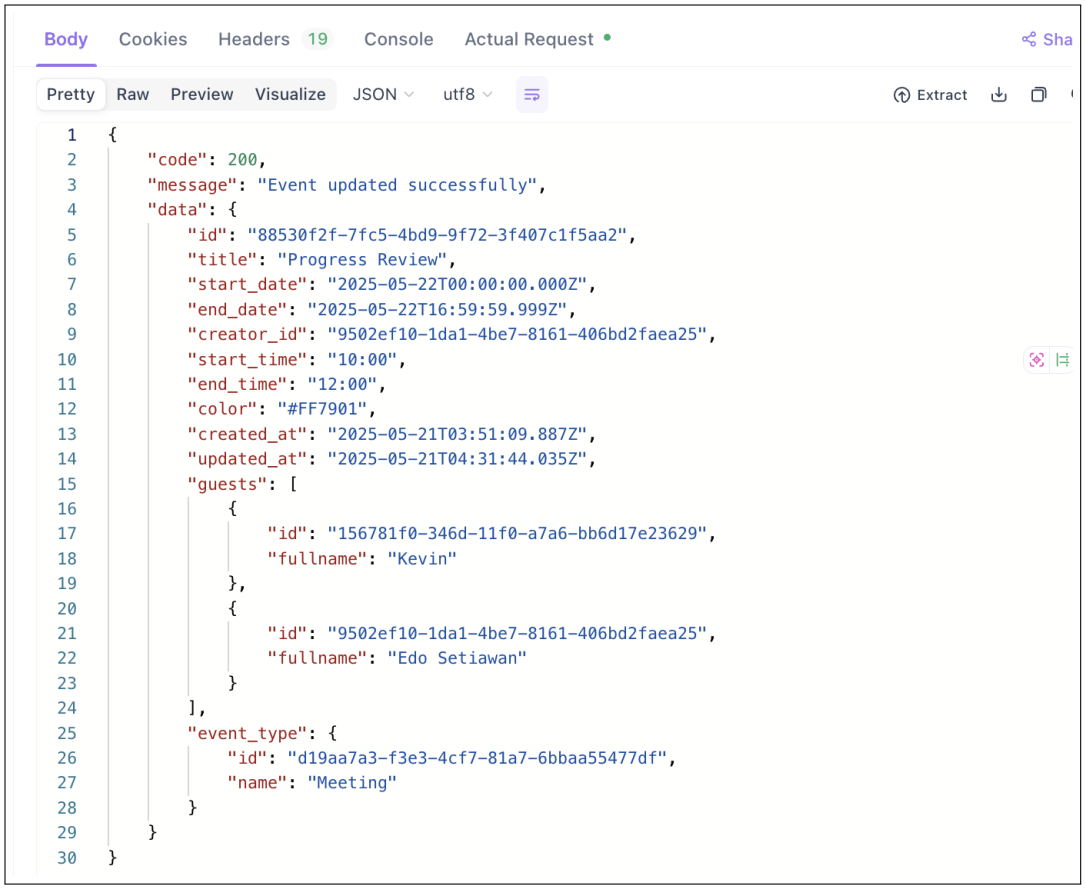


Gambar 3.31. Parameter PATCH /event/:id

Pada parameter pada gambar 3.31, nilai `id` diinputkan menggunakan nilai dari `uuid` yang ada pada data tabel `events`.

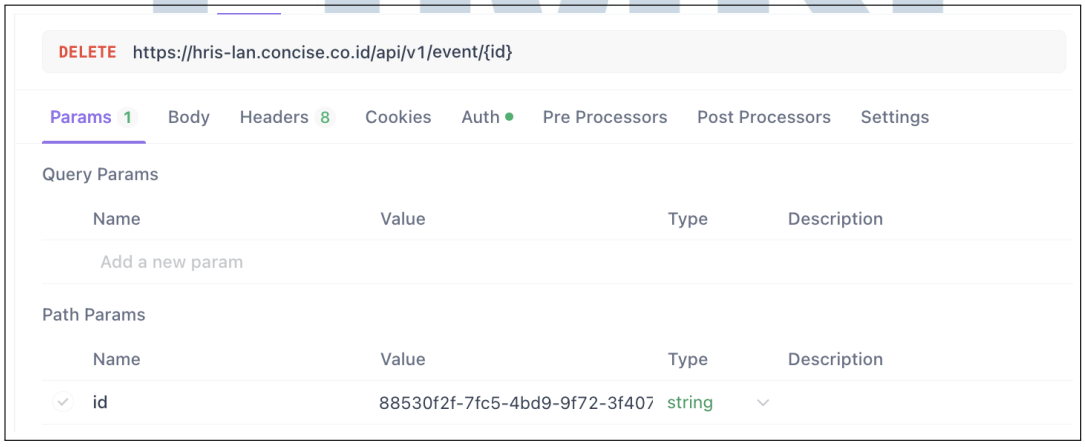
Gambar 3.32. Request PATCH /event/:id

Pada *endpoint* ini yang dapat dilihat pada gambar 3.32, hanya pengguna yang membuat *event* yang dapat mengedit *event* yang telah dibuatnya. Pengecekan ini dideteksi melalui atribut `creator_id` dengan `userInfo` yang didapatkan saat pertama kali login. Untuk *response body* dapat dilihat pada gambar 3.33.



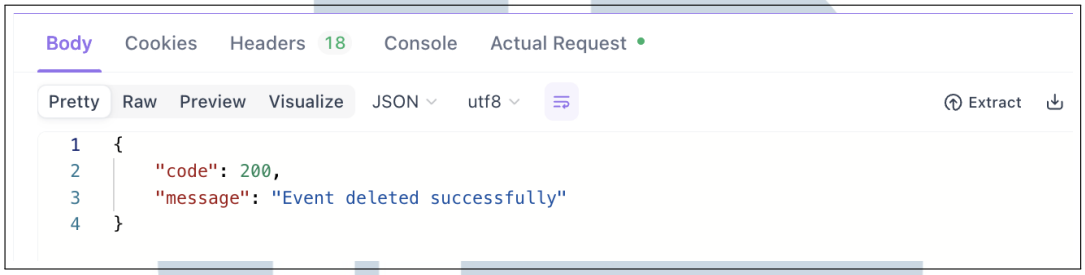
Gambar 3.33. Response PATCH /event/:id

D Endpoint DELETE /event/:id



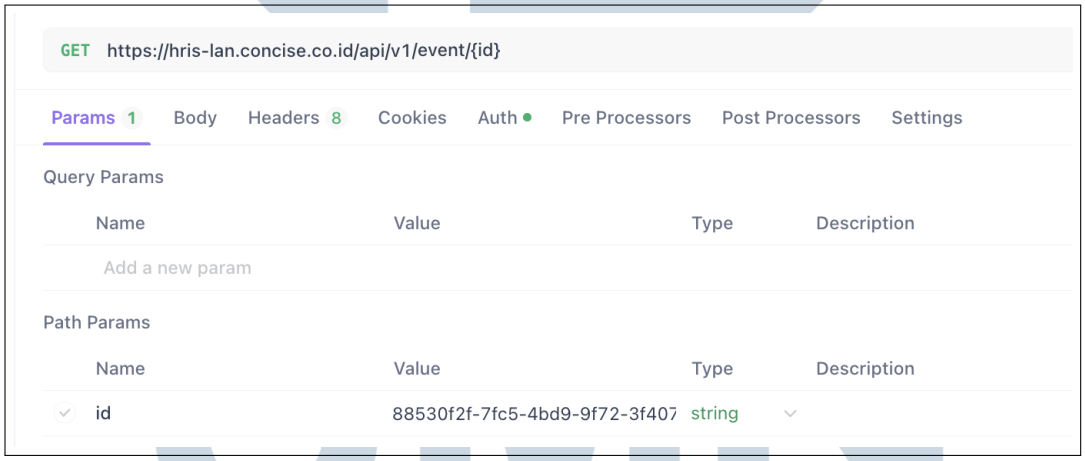
Gambar 3.34. Parameter DELETE /event/:id

Pada *endpoint delete* yand ada pada gambar 3.34, hanya memerlukan nilai dari `uuid` untuk melakukan *soft-deleted* yang diletakkan pada *parameter endpoint*. Dan dalam melakukan *deleted* hanya pengguna yang membuat *event* tersebut. Lalu, untuk *response body endpoint* itu sendiri dapat dilihat pada gambar 3.35.



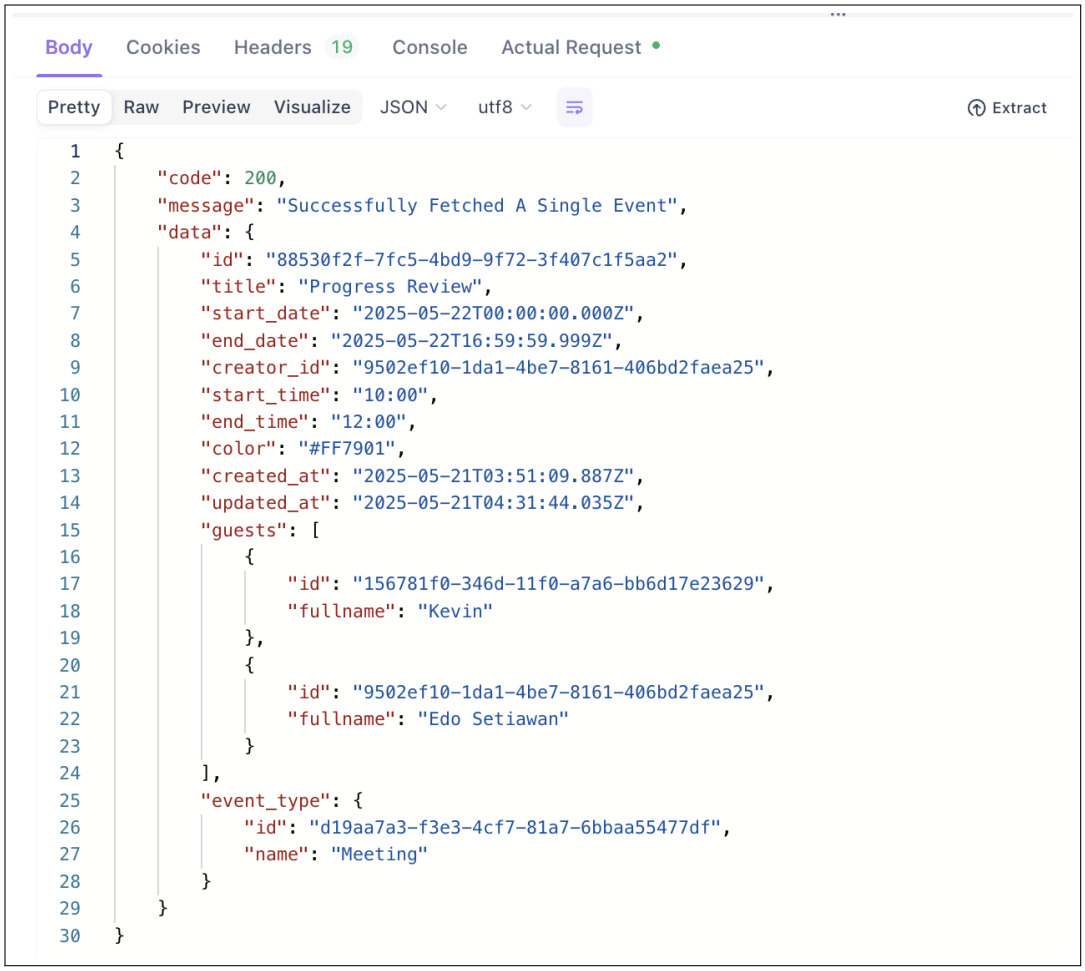
Gambar 3.35. Response DELETE /event/:id

E Endpoint GET /event/:id



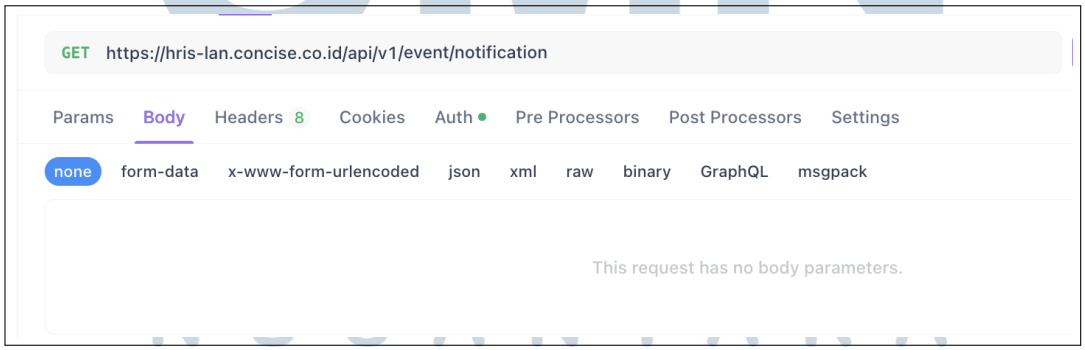
Gambar 3.36. Parameter GET /event/:id

Pada gambar 3.36 merupakan *parameter* yang dibutuhkan untuk melihat detail `event` yang diselenggarakan dengan cara memasukkan nilai dari `uuid` event. *Response* dari *detail event* itu sendiri dapat dilihat pada gambar 3.37.



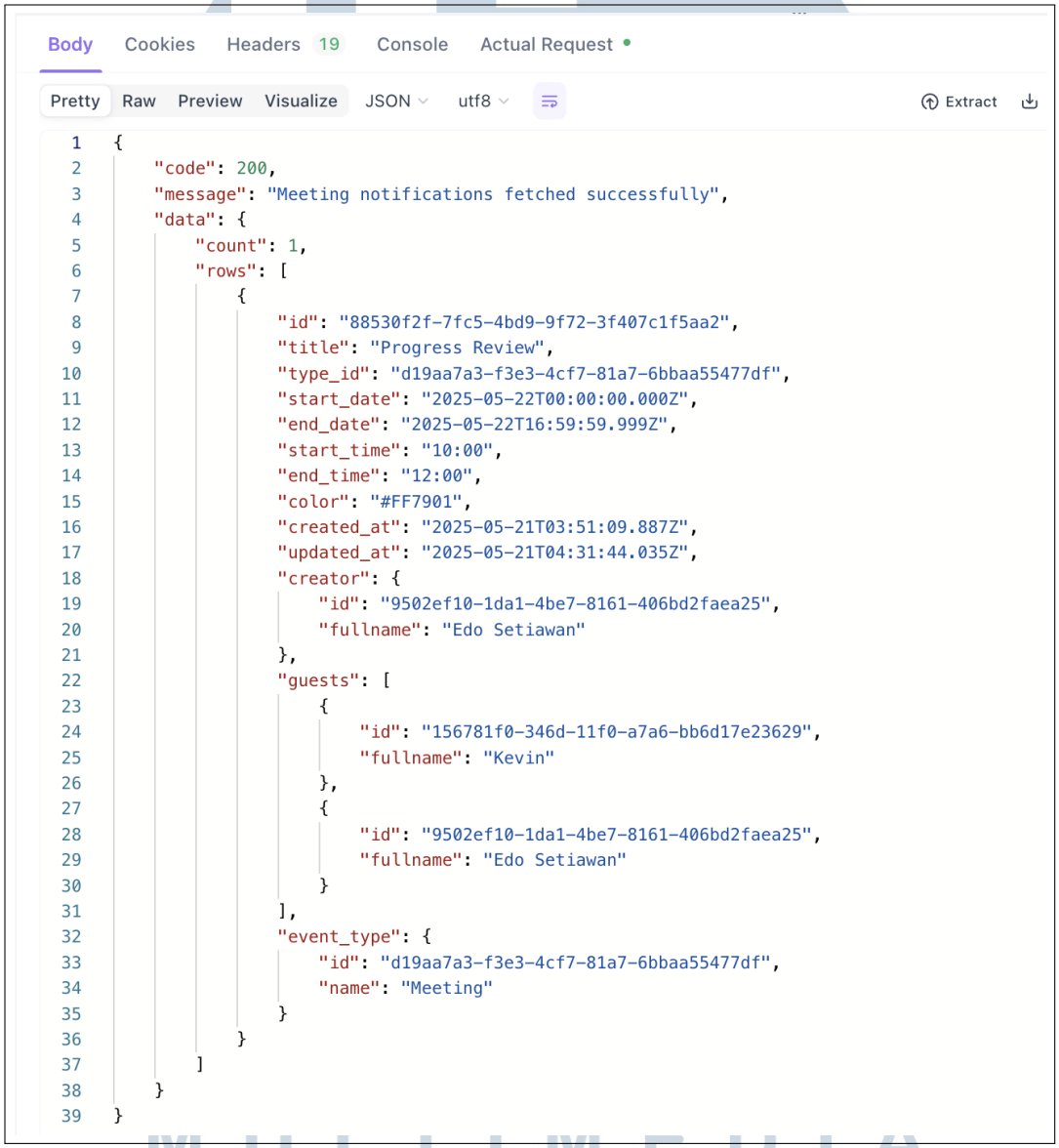
Gambar 3.37. Response GET /event/:id

F Endpoint GET /event/notification



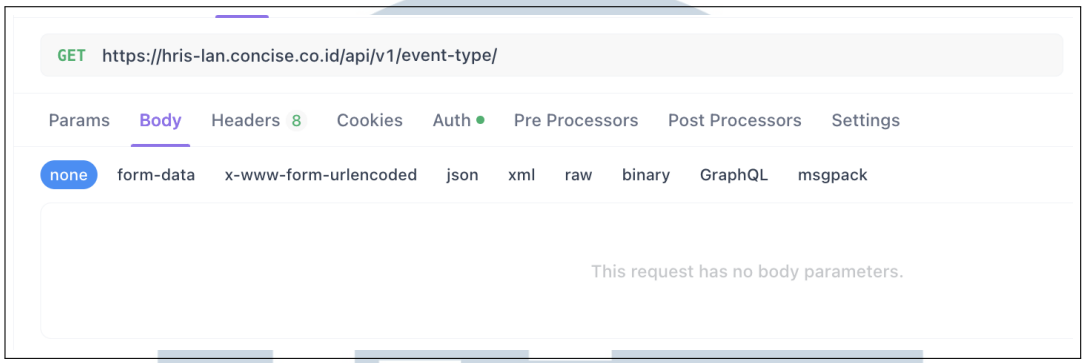
Gambar 3.38. Request GET /event/notification

Pada *endpoint* ini yang ada pada gambar 3.38 hanya akan mengambil data *event* yang berhubungan dengan *meeting* sehingga ketika pengguna pertama kali login akan menampilkan jumlah *count* dari total *event meeting* dengan notifikasi dalam bentuk *badges* yang diberikan untuk *meeting* yang akan datang dalam tujuh hari ke depan. Dan *response* dari *endpoint* dapat dilihat pada gambar 3.39.



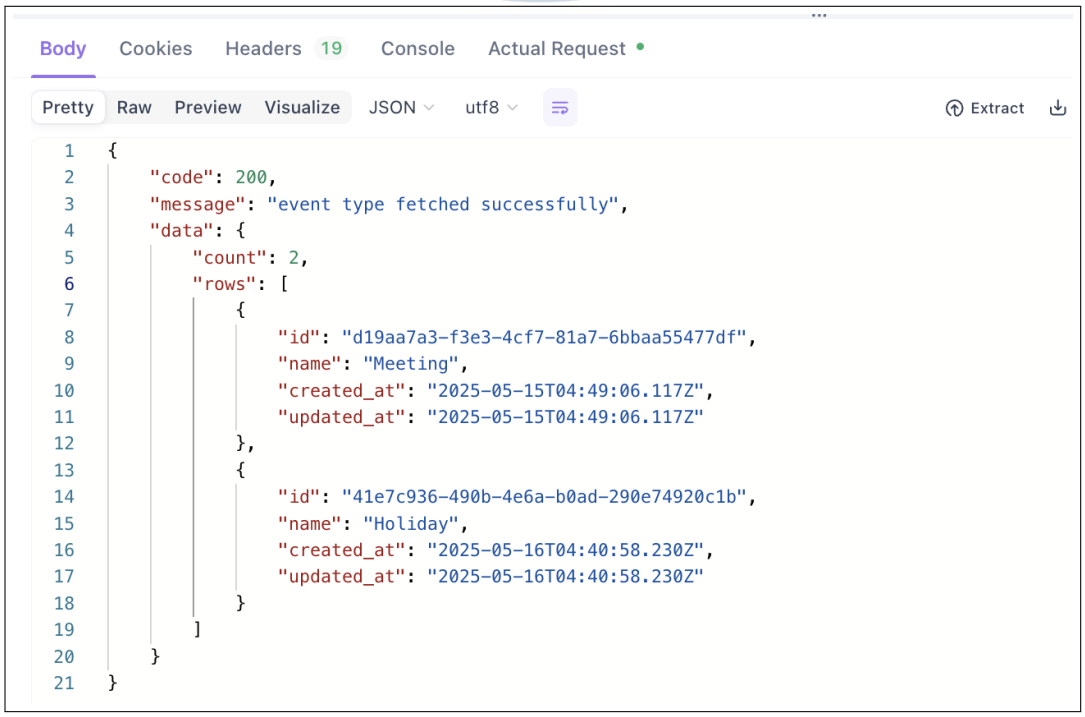
Gambar 3.39. Response GET /event/notification

G Endpoint GET /event-type/



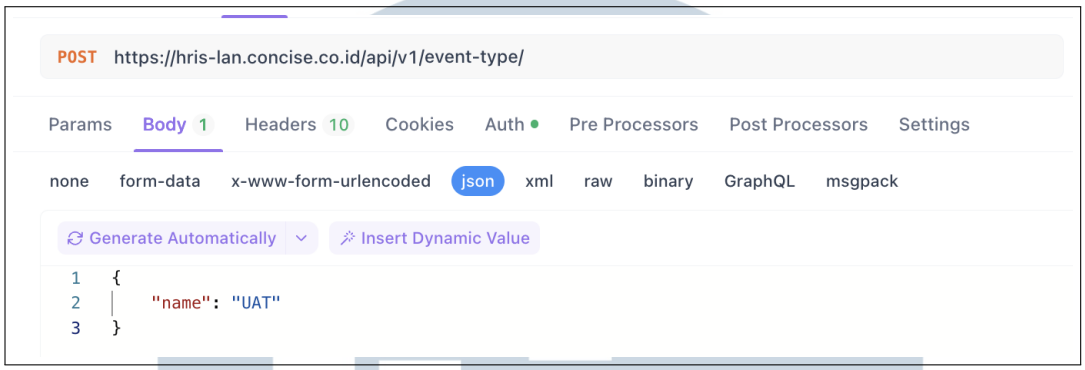
Gambar 3.40. Request GET /event-type/

Untuk melihat data-data dari `event type` akan mengambil dari tabel `event_types` menggunakan *endpoint* `GET /event-type/`. Dari `uuid` yang ada pada atribut `id` nantinya digunakan pada saat pembuatan `event` pada endpoint `POST /event/` yang dapat dilihat pada gambar 3.27 dengan atribut `type_id`. Untuk *response body* dapat dilihat pada gambar 3.41.



Gambar 3.41. Response GET /event-type/

H Endpoint POST /event-type/



Gambar 3.42. Request POST /event-type/

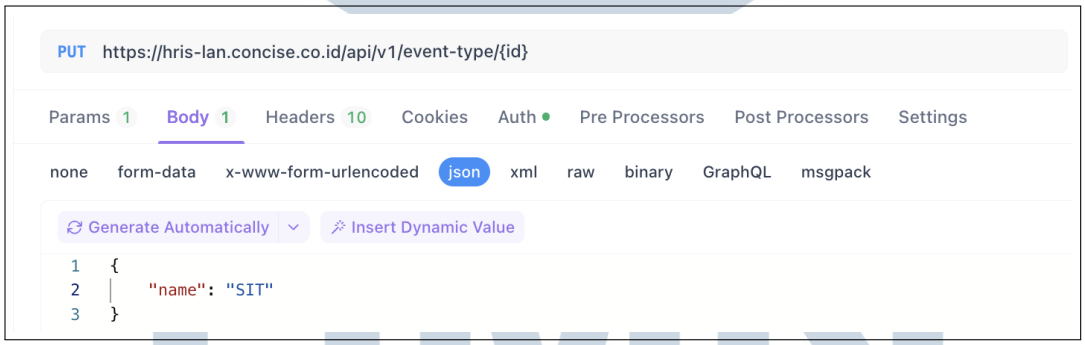
Pada saat pembuatan dari *event type* hanya membutuhkan atribut `name` dengan tipe data berupa *string* yang ada pada gambar 3.42. Untuk *response body* dapat dilihat pada gambar 3.43.

Gambar 3.43. Response POST /event-type/

I Endpoint PUT /event-type/:id

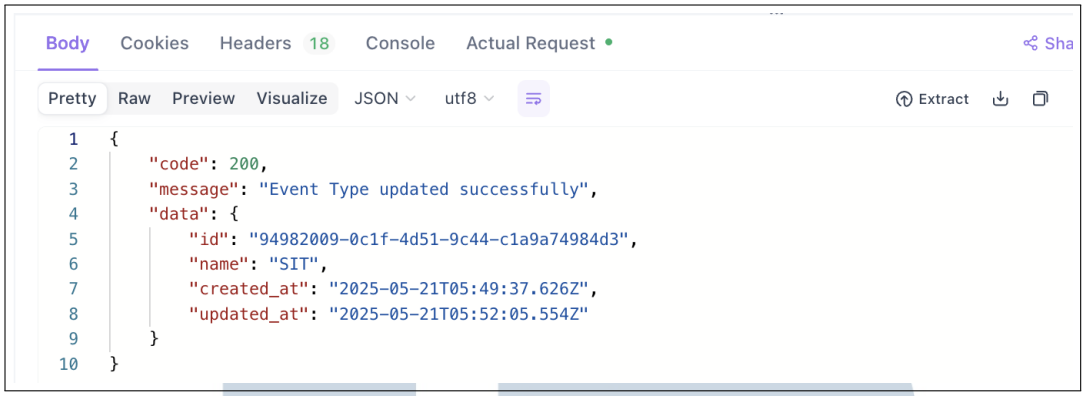
Gambar 3.44. Parameter PUT /event-type/:id

Pada parameter pada gambar 3.44, nilai `id` diinputkan menggunakan nilai dari `uuid` yang ada pada data tabel `events`.



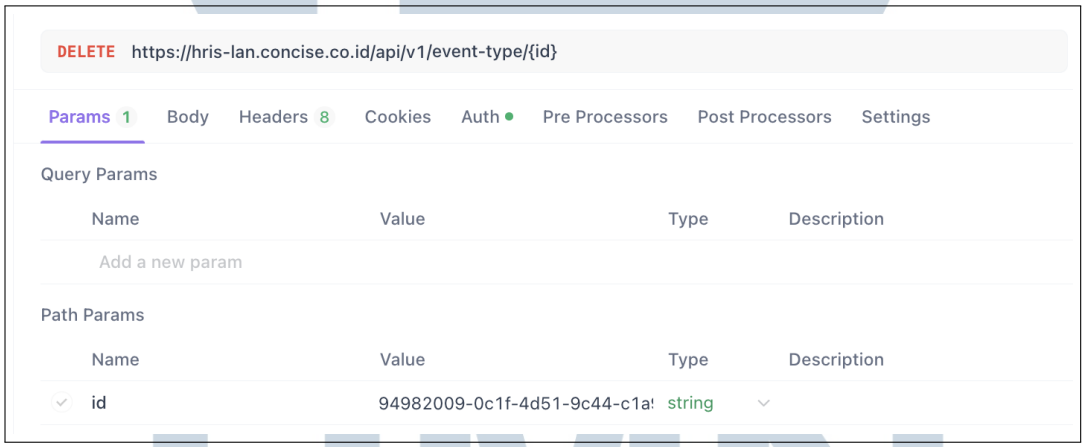
Gambar 3.45. Request PUT /event-type/:id

Pada gambar 3.45 untuk mengganti nama dari *event-type* hanya menggunakan atribut `name` dengan *response body* dapat dilihat pada gambar 3.46.



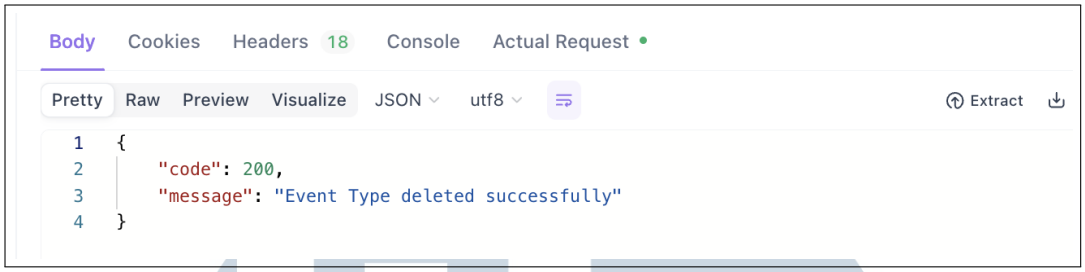
Gambar 3.46. Response PUT /event-type/:id

J Endpoint DELETE /event-type/:id



Gambar 3.47. Parameter DELETE /event-type/:id

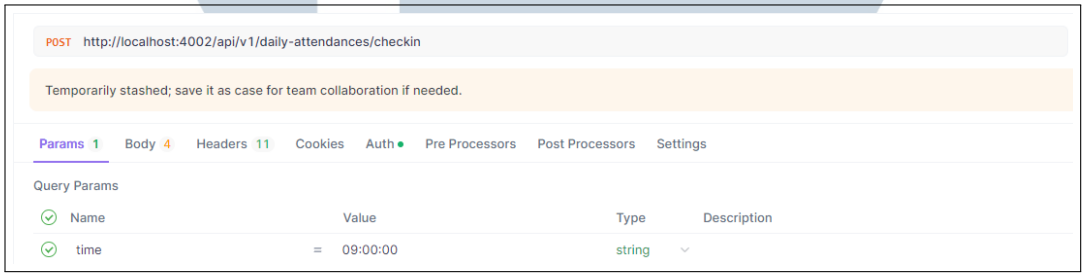
Untuk menghapus *event type* hanya akan dilakukan *soft delete* yang ada pada gambar 3.47 dan ketika ada data *event* yang terhubung dengan suatu *event type* tertentu akan juga ikut terhapus dikarenakan pada pembuatan tabel dari `event_types` menggunakan metode `CASCADE` pada saat *delete* ataupun *update*. Untuk *response* dapat dilihat pada gambar 3.48.



Gambar 3.48. Response DELETE /event-type/:id

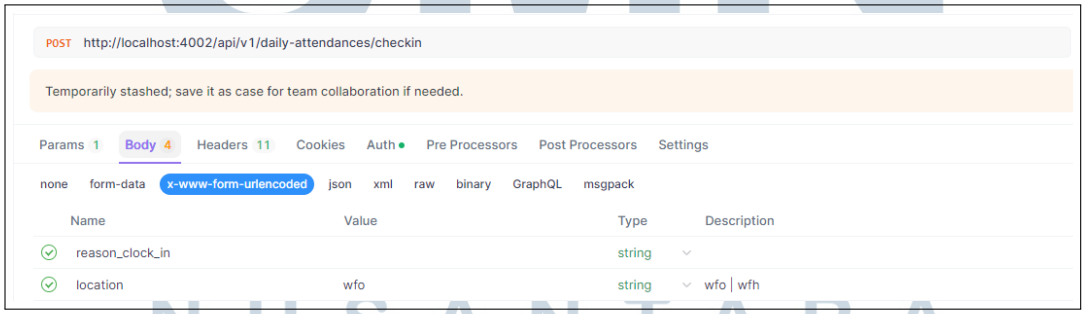
3.5.3 Implementasi Clock In

Pada implementasi *clock in* ini, akan memfokuskan bagaimana cara kerja terhadap pengecekan *IP address* ketika karyawan yang ingin melakukan absensi, yang hanya dapat dilakukan di area kantor.



Gambar 3.49. Request Parameter POST daily-attendances/checkin

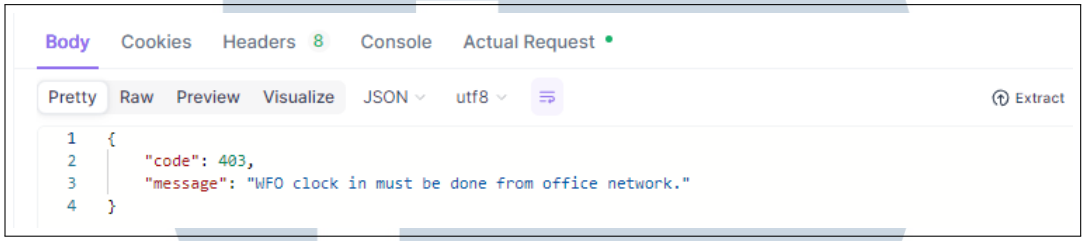
Pada gambar 3.49, terdapat parameter `time` yang dibutuhkan untuk mengetahui jam berapa yang diinputkan oleh pengguna. Pada penginputan jam ini nantinya akan dilakukan secara otomatis berdasarkan jam perangkat pengguna.



Gambar 3.50. Request Body POST daily-attendances/checkin

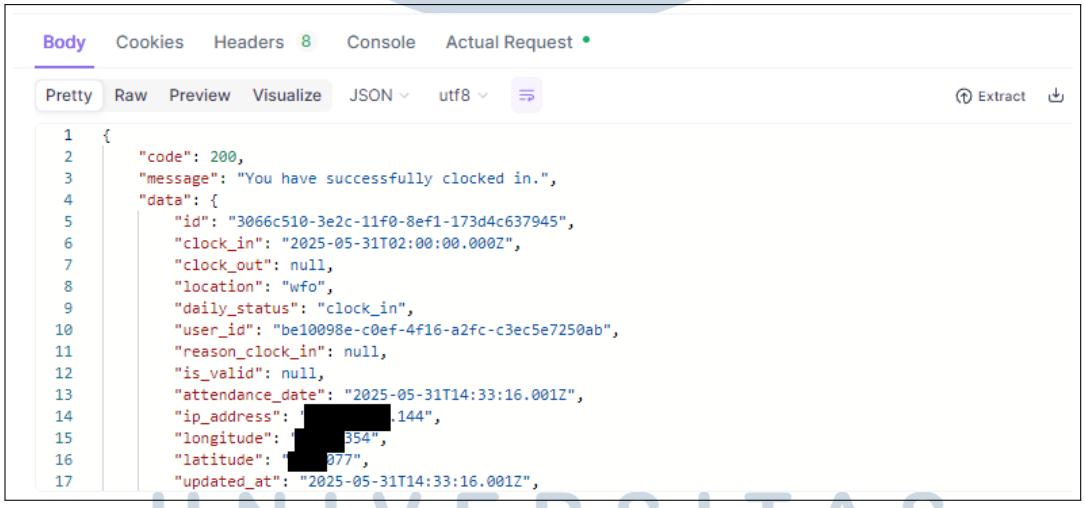
Pada gambar 3.50, untuk bagian body pada *request* terdapat dua atribut yaitu, `reason_clock_in` yang digunakan ketika pengguna telat dari maksimal jam

masuk yang ditentukan, untuk telat akan diberlakukan jika melebihi jam 9:30 pagi hari. Lalu, ada `Location` yang di mana untuk mendeteksi apakah karyawan akan bekerja secara *Work From Office* (WFO) atau *Work From Home* (WFH). Jika karyawan bekerja secara WFO, maka karyawan harus terkoneksi terhadap WIFI kantor untuk dapat melakukan absen. Jika tidak, maka akan mendapatkan pesan dan HTTP status kode, yang dapat dilihat pada gambar 3.51. Namun, untuk WFH sistem tidak akan mengecek apa-apa untuk karyawan melakukan absensi.



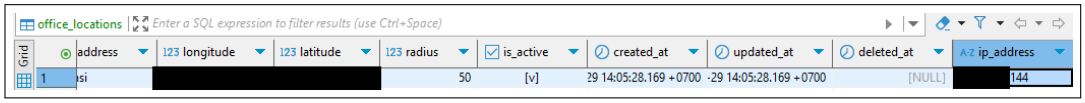
Gambar 3.51. Response 403 POST daily-attendances/checkin

Jika karyawan telah terkoneksi dengan WIFI kantor dan berhasil melakukan absensi akan muncul *response* seperti pada gambar 3.52.



Gambar 3.52. Response 200 POST daily-attendances/checkin

Untuk memahami lebih lanjut bagaimana pengecekan *Ip address* bekerja pada sistem akan dijabarkan fungsi-fungsi yang berhubungan mengenai *Ip address*.



Gambar 3.53. Data pada tabel Office_locations

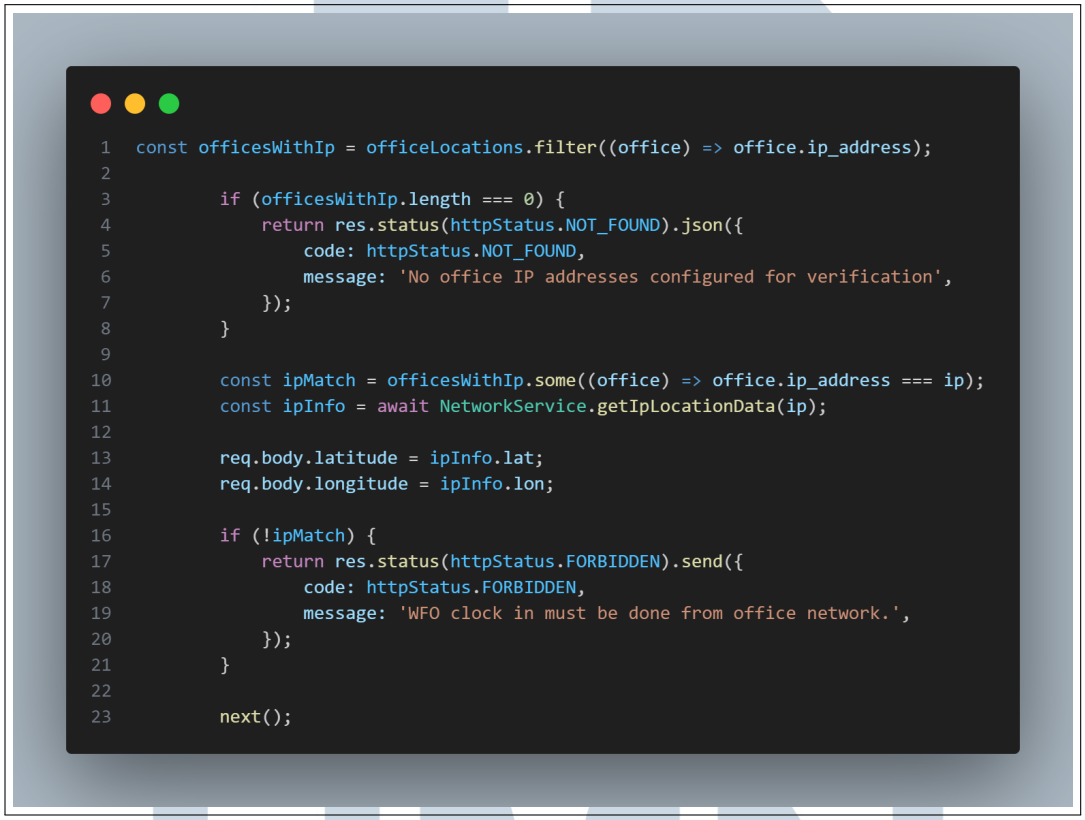
Pada gambar 3.53, merupakan data atau tempat kantor berada dengan atribut `ip_address` yang gunanya untuk melakukan pengecekan dari wifi ip pengguna yang dipakai dengan ip wifi yang seharusnya digunakan di kantor.

```
1  try {
2          const { location} = req.body;
3
4          if (location !== 'wfo') {
5              return next();
6          }
7
8          let ip = await ipify({ useIPv6: false });
9          // Remove the IPv6 prefix
10         if (ip && ip.startsWith('::ffff:')) {
11             ip = ip.substring(7);
12         }
13         req.clientIp = ip;
14
15         const officeLocations = await OfficeLocation.findAll({
16             where: {
17                 is_active: true,
18             },
19         });
20
21         if (officeLocations.length === 0) {
22             return res.status(httpStatus.NOT_FOUND).send({
23                 code: httpStatus.NOT_FOUND,
24                 message: 'No active office locations',
25             });
26         }
```

Gambar 3.54. IP checking mechanism

Pada gambar 3.54, merupakan pengecekan IP untuk pengguna menggunakan *third-party software* yaitu, Ipify. Ipify itu sendiri merupakan layanan yang digunakan untuk mengambil *public ip address* yang bisa digunakan di berbagai macam bahasa pemrograman seperti, Go, Python, Ruby, Node.js dan sebagainya [9]. Pengambilan publik *ip* itu sendiri terletak pada line kode 8, yang di mana pada line kode 13 IP publik yang didapatkan akan tersimpan pada variabel `clientIp` yang tersimpan secara global. Lalu, pada line kode 15 sampai 26

akan dilakukan pengecekan terhadap data pada tabel `office_locations` dengan mengambil data yang berstatus aktif saja yang terlihat pada line kode 17. Jika tidak ada data pada tabel tersebut maka akan mengembalikan pesan `No active office locations`.



```
1  const officesWithIp = officeLocations.filter((office) => office.ip_address);
2
3          if (officesWithIp.length === 0) {
4              return res.status(httpStatus.NOT_FOUND).json({
5                  code: httpStatus.NOT_FOUND,
6                  message: 'No office IP addresses configured for verification',
7              });
8          }
9
10         const ipMatch = officesWithIp.some((office) => office.ip_address === ip);
11         const ipInfo = await NetworkService.getIpLocationData(ip);
12
13         req.body.latitude = ipInfo.lat;
14         req.body.longitude = ipInfo.lon;
15
16         if (!ipMatch) {
17             return res.status(httpStatus.FORBIDDEN).send({
18                 code: httpStatus.FORBIDDEN,
19                 message: 'WFO clock in must be done from office network.',
20             });
21         }
22
23         next();
```

Gambar 3.55. IP checking mechanism

Selanjutnya pada gambar 3.55, merupakan kode lanjutan yang di mana pada line kesatu akan memfilter data yang ada pada tabel `office_locations` dan hanya mengambil atribut `ip_address`. Dan jika tidak ada `Ip address` pada tabel `office_locations` akan mengirim pesan seperti pada line 6. Namun, jika terdapat *ip address* maka sistem akan melakukan perbandingan yang terlihat pada line 10, yang di mana `ip address` pengguna akan dicocokkan dengan *ip address* kantor. Lalu, jika kedua *ip address* tersebut sama maka sistem akan melanjutkan yang terlihat pada line 23. Jika tidak, maka akan terkirim pesan error pada line 19 atau contoh gambar 3.51.

Pada line 11 terlihat bahwa ip yang telah didapatkan dari pengguna dimasukkan ke dalam suatu fungsi `getIpLocationData` yang di mana fungsi ini digunakan untuk mengubah ip address yang didapat ke dalam format *longitude* dan

latitude, menggunakan *third-party software* bernama Ip-Api. Ip-Api merupakan *software* yang berfungsi sebagai *Geolocation IP* yang di mana menyediakan layanan secara gratis untuk *non-commercial use* kepada para *developer* dan dengan mudah diintegrasikan ke dalam format JSON, XML, CSV, Newline, PHP [10].

3.6 Kendala dan Solusi

Adapun kendala yang dihadapi selama magang di PT Ganda Visi Jayatama:

1. Kurangnya komunikasi dalam pihak *Front-end* dan *Back-end* yang membuat alur pengerjaan menjadi terhambat.
2. Kurangnya pengetahuan terhadap metode *best practices* yang digunakan dalam industri teknologi dalam pengkodean.

Dan berikut solusi yang dilakukan untuk menangani kendala-kendala yang terjadi:

1. Membuat alur pengerjaan yang akan dilakukan dengan pihak *Front-end* dan *Back-end* selama seminggu dari hari senin sampai dengan jum'at.
2. Mencari informasi melalui *Stack Overflow* atau forum-forum terkenal serta blog yang membahas mengenai *best practice* kode.