BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Pada divisi *Technical Consultant* terbagi menjadi dua tim yaitu tim Splunk dan tim Development. Pelaksanaan kerja magang dilakukan sebagai *Technical Consultant* dengan beberapa anggota yang berada dari tim Development yang diberi tanggung jawab atas proyek MYGIT. Tim dari proyek MYGIT berada di bawah pengawasan dari Bapak Ahmad Rizki selaku *VP Operation* dan dikoordinasikan oleh Bapak Robiul Mustofa selaku *Project Manager* dari tim *Development*. Tim yang bertanggung jawab atas perancangan MYGIT terdiri dari 1 *Frontend Developer*, 1 *UI/UX Designer*, 1 *Backend Developer*, dan 1 *mobile developer* yang juga merupakan *team leader* yang bertugas untuk membantu mengoordinasikan tim proyek MYGIT. Pada awal pelaksanaan proyek, tim telah diberikan *project timeline* yang dibuat oleh *Project Manager* untuk durasi pengerjaan setiap fitur *website* hingga selesai.

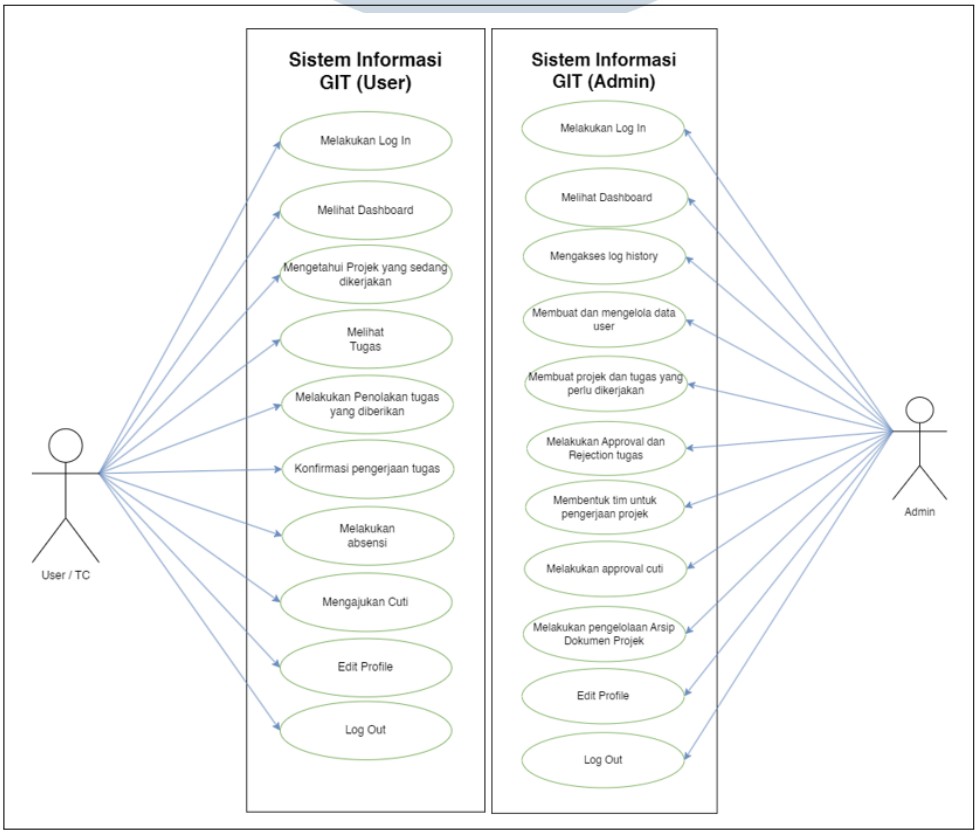
Setiap pagi *Project Manager* akan memberikan arahan dan *check-up* terkait pekerjaan yang akan dilakukan pada hari tersebut. Kemudian pada sore hari, tim akan melaporkan *progress* dari pekerjaan tersebut. Pada hari jumat, dilaksanakan pertemuan retrospeksi bersama semua divisi *Technical Consultant* yang dipimpin oleh *VP Operation*, membahas dari *progress* pekerjaan selama seminggu dan mengidentifikasi hambatan dan masalah yang dihadapi tim pada pekerjaan yang dilakukan untuk bersama-sama membahas solusi.

Pada bulan Febuari 2024, dengan arahan dari *VP Operation* yang juga bertindak sebagai *supervisor* dari penulis, memberikan kesempatan kepada penulis untuk bergabung dengan tim Development untuk merancang *website* MYGIT. Di bulan Mei 2024, penulis dialihkan ke tim Splunk untuk membantu preventive maintenance dari salah satu klien GIT serta membangun produk baru *Monitoring Tools* untuk GIT. Namun dengan arahan VP Operation, penulis tetap membantu tim *Development* dalam merancang aplikasi manajemen karyawan baru, tetapi bukan sebagai tim utama melainkan sebagai *advisor* yang akan membantu memberikan saran dan kritik pada perancangan struktur aplikasi.

3.2 Tugas yang Dilakukan

Proyek yang menjadi fokus pada tugas magang selama Pelaksanaan kerja magang sebagai *Technical Consultant* di PT Global Innovation Technology (GIT) adalah proyek aplikasi berbasis *website* manajemen karyawan MYGIT. Selama berada di tim Development, tugas yang diberikan adalah merancang *Backend API website* MYGIT. Kemudian setelah dipindahkan ke tim Splunk, diberikan penugasan untuk mempelajari teknologi Splunk dan melakukan eksplorasi pada teknologi Grafana untuk membuat sebuah *dashboard monitoring tools* dengan menggunakan MYGIT sebagai *sample* yang akan dimonitor.

MYGIT bertujuan untuk digunakan oleh *HRD* (*Human Resource Department*), *Project Manager*, dan *Technical Consultant* GIT. Terdapat 2 *role* untuk pengguna MYGIT, yaitu *ADMIN* dan *STAFF/TC*. Untuk pengguna seperti *HRD* dan *Project Manager* akan mendapatkan *role ADMIN*, sedangkan *Technical Consultant* akan mendapatkan role *STAFF/TC*. Setiap *role* memiliki akses berbeda pada setiap fiturnya. Representasi dari perbedaan fitur dapat dilihat pada Gambar 3.1.



Gambar 3.1. *Use case* untuk *ADMIN* dan *STAFF/TC*

MYGIT memiliki fitur-fitur utama seperti fitur manajemen karyawan, fitur manajemen projek dan task, fitur presensi kehadiran. Fitur manajemen karyawan meliputi semua fitur yang berkaitan dengan lingkup *HRD* seperti fitur pendataan informasi karyawan, pengajuan cuti karyawan, serta pengajuan lembur karyawan. Fitur manajemen projek dan task meliputi semua fitur untuk mempermudah koordinasi antara *Project Manager* dan *Technical Consultant* seperti fitur manajemen *task* dan *project*, *task invitation, task submission, task review,* dan arsip dokumen.

Sebelum merancang *backend API, Project Manager* memberikan tanggung jawab kepada penulis untuk menyiapkan *server* yang akan digunakan sebagai media *hosting website* MYGIT. Terdapat 2 *server* yang disiapkan yaitu *server development* yang berada pada *private server* perusahaan dan *Cloud VPS* yang dibeli oleh perusahaan yang akan digunakan sebagai *production server*. Diberikan waktu selama seminggu untuk menyiapkan kedua *server* agar dapat digunakan untuk proses perancangan *website* MYGIT.

Berikut merupakan *API* yang dirancang sesuai dengan kebutuhan fitur pada *website* MYGIT.

- *API user, team, role,* dan *job*
- Autentikasi *user*
- *API* fitur presensi kehadiran karyawan
- *API* fitur *project* dan *task*
- *API* fitur pengajuan lembur
- *API* fitur pengajuan cuti
- *API* fitur arsip dokumen

Setelah selesai merancang *backend API* dari aplikasi *website* MYGIT, diberikan penugasan baru pada tim baru, yaitu tim Splunk. Penugasan berupa eksplorasi terkait platform visualisasi data bernama Grafana. Grafana merupakan sebuah platform visualisasi data yang memungkinkan pengguna untuk membuat *dashboard* yang dapat melakukan analisis jenis data seperti metrik sistem, data aplikasi, data *website*, dan data lainnya. Grafana menyediakan berbagai jenis komponen visualisasi data. Pengguna juga dapat menambahkan, menghapus, dan mengkonfigurasi komponen visualisasi data untuk membuat dashboard yang

sesuai dengan kebutuhan. Grafana juga mendukung berbagai sumber data, seperti *database*, *API*, maupun *file*. Grafana juga memungkinkan untuk terhubung ke berbagai sumber data dan mulai menvisualisasikan data. Setelah melakukan eksplorasi, diberikan penugasan untuk melakukan implementasi visualisasi data metrik kesehatan sistem pada server yang digunakan oleh aplikasi MYGIT.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang di PT Global Innovation Technology (GIT) setiap minggu diuraikan seperti pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Memahami tim dan struktur proyek MYGIT Memperbaiki struktur *database* dan melakukan instalasi dan konfigurasi *Database* pada *server deployment*
2	Melakukan konfigurasi CI/CD menggunakan pada development server untuk mempermudah *deployment* Melakukan Instalasi dan konfigurasi *framework* untuk *backend API*
3-4	Membuat fitur CRUD user, team, role, dan job Membuat fitur presensi kehadiran karyawan dan melakukan testing pada fitur presensi kehadiran
5-6	Membuat fitur CRUD *project* dan *task* beserta fitur *task invitation, task submission,* dan *task review* Melakukan replikasi server dari development server ke production server
7	Membuat tampilan *frontend* beserta *API* untuk fitur pengajuan lembur karyawan
8	Membuat *API* untuk fitur pengajuan cuti karyawan
9-10	Membuat *API* untuk arsip dokumen
11	Melakukan End-to-End testing dan *bug finding*
12	Melakukan perbaikin pada *bugs* yang ditemukan Membuat *technical document* untuk *backend API* MYGIT
13	Memahami tim baru dan mempelajari infrastruktur Splunk dari salah satu klien GIT
14-16	Melakukan eksplorasi terhadap teknologi Grafana untuk pembuatan *Monitoring Tools* GIT

3.3.1 Infrastruktur Backend API

Perancangan *backend API website* MYGIT menggunakan *techstack* LAMP (Linux, Apache, MySQL, PHP). Untuk *backend service* menggunakan PHP Laravel dan *MySQL* sebagai Database dan DBMS (*DataBase Management System*) yang diintegrasikan menggunakan phpMyAdmin sebagai alat administrasi *database*. Apache digunakan sebagai *web service* untuk menjalankan phpMyAdmin. Berikut

merupakan spesifikasi dari infrastruktur maupun *framework* yang digunakan untuk merancang *website* MYGIT:

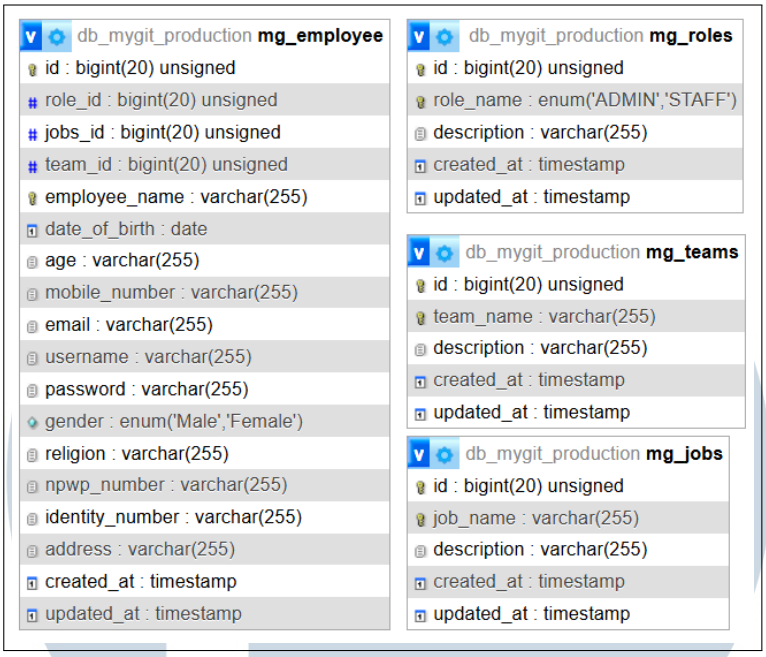
- *PHP* dengan versi 8.1 atau lebih
- *Laravel* dengan versi 10.4.0
- *Composer* dengan versi 2.7.1
- *MySQL* dengan versi 8.0.3
- *Apache HTTP Server* dengan versi 2.4.5
- *OS Ubuntu Linux* dengan versi 22.20 64-*bit*

Dalam backend PHP Laravel, proses dimulai ketika pengguna mengirimkan permintaan ke *website*. Permintaan ini diterima oleh sistem *routing* Laravel, yang menentukan *controller* mana yang harus menangani permintaan tersebut berdasarkan *URL* dan metode HTTP yang digunakan. *Controller* kemudian memproses logika bisnis yang diperlukan, seperti validasi data menggunakan aturan yang ditentukan, interaksi dengan database melalui *Eloquent ORM* untuk mengambil atau menyimpan data, dan menjalankan layanan atau pekerjaan lainnya. Setelah semua logika diproses, *controller* mengembalikan respons yang sesuai, seperti tampilan HTML atau data JSON kepada pengguna.

3.3.2 Pembuatan API Role, Job, Team dan user

Untuk mengakses fitur MYGIT diperlukan akun *user*. Setiap akun *user* tersimpan informasi pribadi seperti nama, nomor telepon, *email*, dsb. Data yang disimpan pada *database* dapat dilihat pada Gambar 3.2.

Untuk proses komunikasi antara *backend* dan *frontend* akan menggunakan *API* yang akan menerima dan mengirimkan data dalam JSON. Setiap ada permintaan dari pengguna, walaupun permintaan tersebut secara fungsi adalah benar atau pun salah, *API* akan tetap memberikan respons kepada pengguna dengan menggunakan JSON.



Gambar 3.2. Atribut basis data dari tabel *Employee, Role, Team* dan *Job*

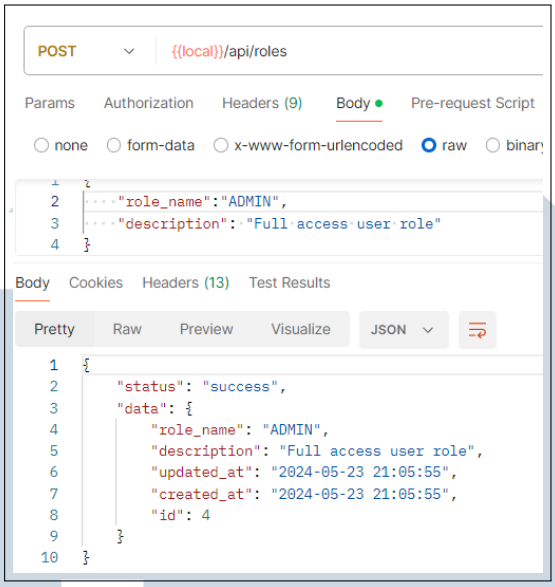
Sebelum membuat data *user* baru, diperlukan pembuatan *role, job,* dan *team*. Data dari *role, job,* dan *team* akan menjadi pilihan yang dapat dipilih wajib pada saat membuat akun *user*.

A. API Role

Perancangan *API Role* bertujuan untuk memisahkan akses secara fungsi untuk tiap pengguna secara dinamik. Fungsi *Role* meliputi *create role*, *update role*, *delete role*, dan *get role data*.

A.1 Create Role

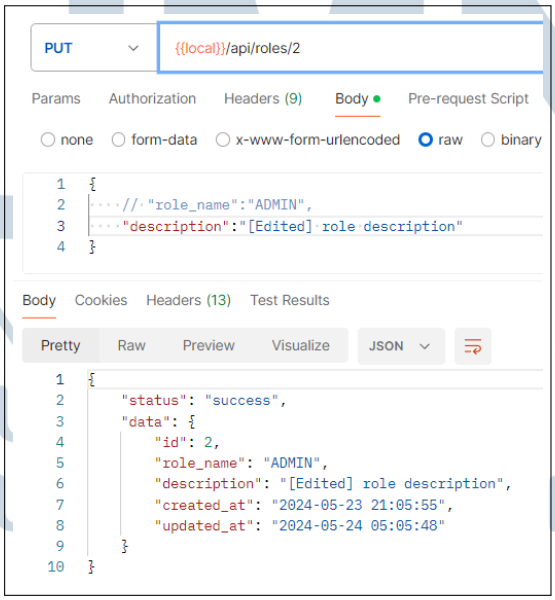
Untuk membuat *role* baru dibutuhkan informasi tentang nama dari *role* tersebut beserta deskripsi *role* (opsional). Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *error* jika terjadi kesalahan dari permintaan pengguna seperti pada Gambar 3.3.



Gambar 3.3. Tampilan percobaan *API create role*

A.2 Update Role

Untuk melakukan pembaharuan data *role* dibutuhkan informasi tentang nama dari *role* tersebut beserta deskripsi *role*. Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *error* jika terjadi kesalahan dari permintaan pengguna seperti pada Gambar 3.4.



Gambar 3.4. Tampilan percobaan *API update role*

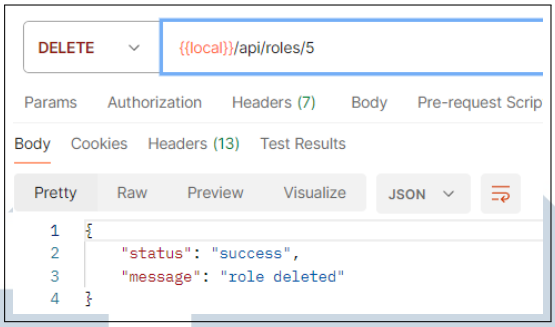
A.3 API Get Role Data

API untuk mendapatkan data *role* terdapat dia cara, yaitu mendapatkan semua data *role*, dan mendapatkan satu data *role* secara spesifik berdasarkan dari *id role* yang diterima. Respons dari *API* dapat dilihat pada Gambar 3.5.

Gambar 3.5. Tampilan percobaan *API get role data*

A.4 API Delete Role

Untuk menghapus data *role*, diperlukan *id role* yang kemudian akan disertakan sebagai *parameter* pada saat memanggil *route API* untuk menghapus data dengan *id role* tersebut. Respons dari *API* dapat dilihat pada Gambar 3.6.



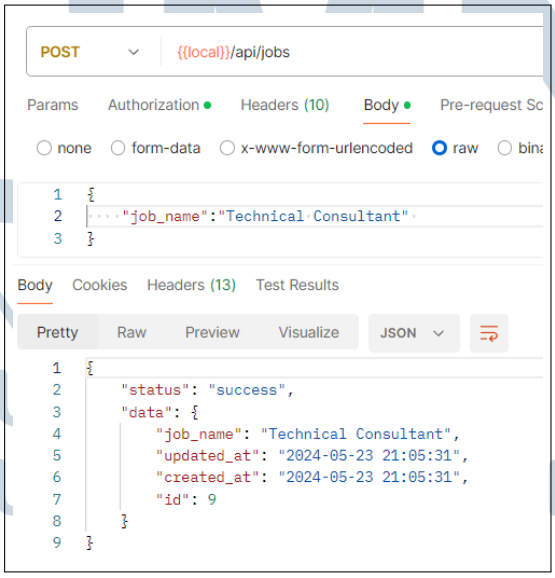
Gambar 3.6. Tampilan percobaan *API delete role*

B. API Job

API job dirancang untuk membuat pendataan pekerjaan secara dinamik. Perancangan *Job* meliputi fungsi yang akan digunakan pada tampilan *website* (*frontend*) seperti *create job*, *update job*, *delete job*, dan *get job data*.

B.1 Create Job

Untuk membuat *job* baru dibutuhkan informasi tentang nama dari *job* tersebut beserta deskripsi *job* (opsional). Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *error* jika terjadi kesalahan dari permintaan pengguna seperti pada Gambar 3.7.



Gambar 3.7. Tampilan percobaan *API create job*

API update, delete dan *get job data* memiliki struktur dan respons yang sama seperti fungsi yang dimiliki *API role* di atas.

C. API Team

API team dirancang untuk membuat pendataan tim seperti tim Splunk dan Development secara dinamik. Perancangan *API team* meliputi fungsi yang akan digunakan pada tampilan *website* (*frontend*) seperti *create team*, *update team*, *delete team*, dan *get team data*.

C.1 Create Team

Untuk membuat *team* baru dibutuhkan informasi tentang nama dari *team* tersebut beserta deskripsi *team* (opsional). Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *error* jika terjadi kesalahan dari permintaan pengguna seperti pada Gambar 3.8.

Gambar 3.8. Tampilan percobaan *API create team*

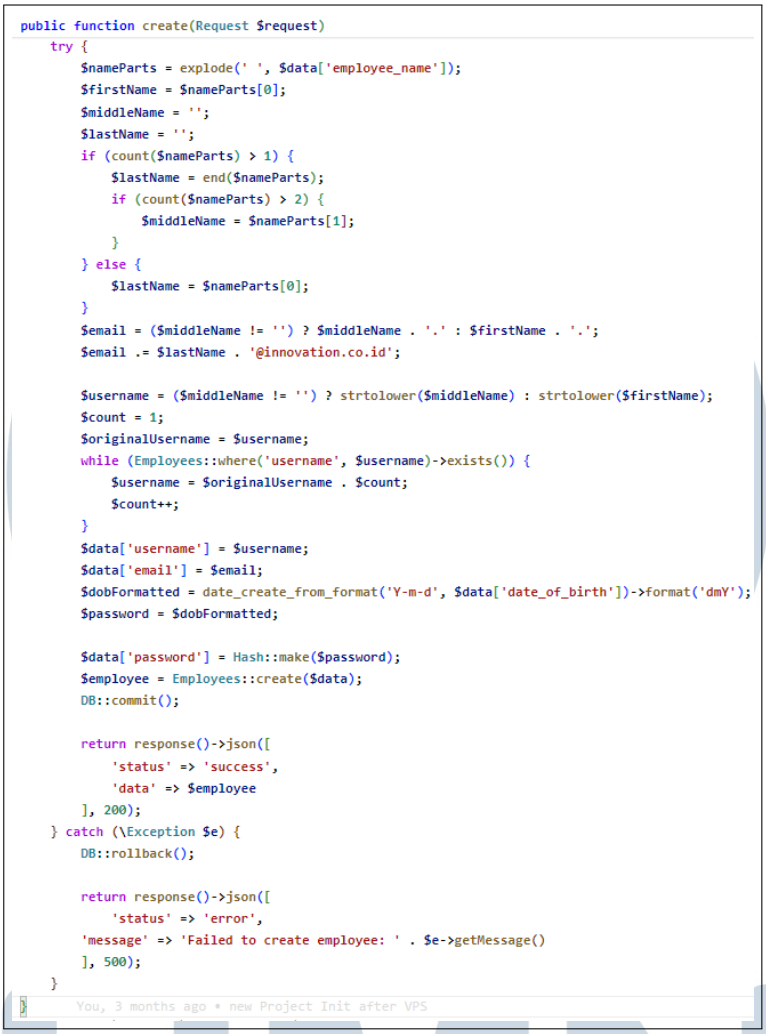
API update, delete dan *get team data* memiliki struktur dan respons yang sama seperti fungsi yang dimiliki *API role* di atas.

D. API User

API user dirancang untuk menangani fitur manajemen pendataan karyawan seperti penambahan data untuk karyawan baru, perubahan data karyawan, penghapusan data karyawan, serta mengirimkan data karyawan tersebut ke *frontend* untuk ditampilkan kepada pengguna *website*. Setelah dibuatnya data *role, job* dan *tim*, data tersebut akan digunakan sebagai pilihan pada saat pembuatan data karyawan baru.

D.1 API Create User

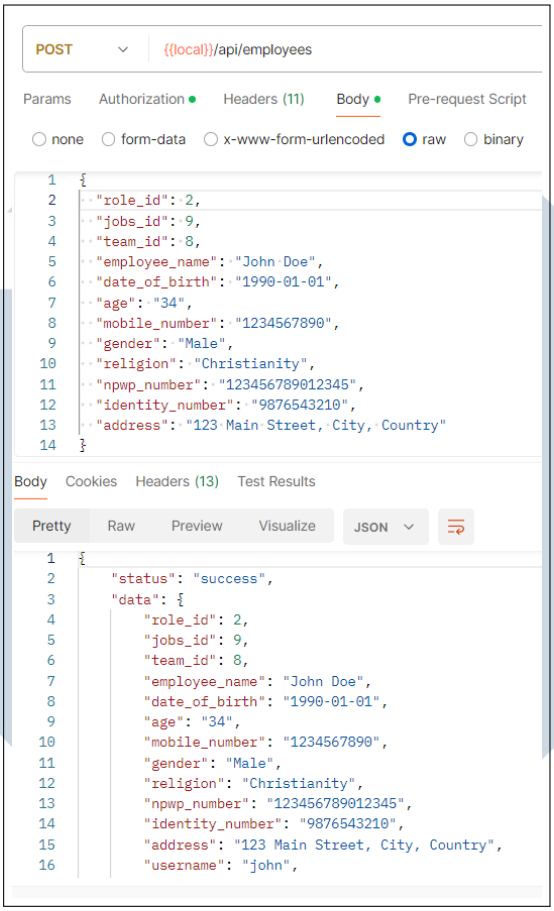
Ketika ingin menambahkan data karyawan baru, data yang akan disimpan adalah data seperti yang ada pada atribut tabel 3.2. *API* juga akan meminta *id* dari *role, team* dan *job* yang sesuai dengan kedudukan karyawan saat ini. Atribut data seperti *username*, *email*, dan juga *password* akan dibuat secara otomatis oleh fungsi pada *API*. Fungsi tersebut dapat dilihat pada Gambar 3.9.



```php
public function create(Request $request)
    try {
        $nameParts = explode(' ', $data['employee_name']);
        $firstName = $nameParts[0];
        $middleName = '';
        $lastName = '';
        if (count($nameParts) > 1) {
            $lastName = end($nameParts);
            if (count($nameParts) > 2) {
                $middleName = $nameParts[1];
            }
        } else {
            $lastName = $nameParts[0];
        }
        $email = ($middleName != '') ? $middleName . '.' : $firstName . '.';
        $email .= $lastName . '@innovation.co.id';

        $username = ($middleName != '') ? strtolower($middleName) : strtolower($firstName);
        $count = 1;
        $originalUsername = $username;
        while (Employees::where('username', $username)->exists()) {
            $username = $originalUsername . $count;
            $count++;
        }
        $data['username'] = $username;
        $data['email'] = $email;
        $dobFormatted = date_create_from_format('Y-m-d', $data['date_of_birth'])->format('dmY');
        $password = $dobFormatted;

        $data['password'] = Hash::make($password);
        $employee = Employees::create($data);
        DB::commit();

        return response()->json([
            'status' => 'success',
            'data' => $employee
        ], 200);
    } catch (\Exception $e) {
        DB::rollback();

        return response()->json([
            'status' => 'error',
        'message' => 'Failed to create employee: ' . $e->getMessage()
        ], 500);
    }
}       You, 3 months ago • new Project Init after VPS
```

Gambar 3.9. Potongan Kode API create user

Username akan terbuat berdasarkan dari nama pertama karyawan, *email* karyawan akan dibentuk sesuai dengan nama pertama dan nama kedua karyawan yang kemudian ditambahkan dengan *domain* dari perusahaan yaitu *'innovation.co.id'*, kemudian untuk *password* akan terbentuk dari tanggal lahir karyawan. Jika semua data yang dikirimkan kepada *API* tervalidasi dengan benar maka respons yang dikirimkan akan terlihat seperti Gambar 3.10.



Gambar 3.10. Tampilan percobaan *API create user*

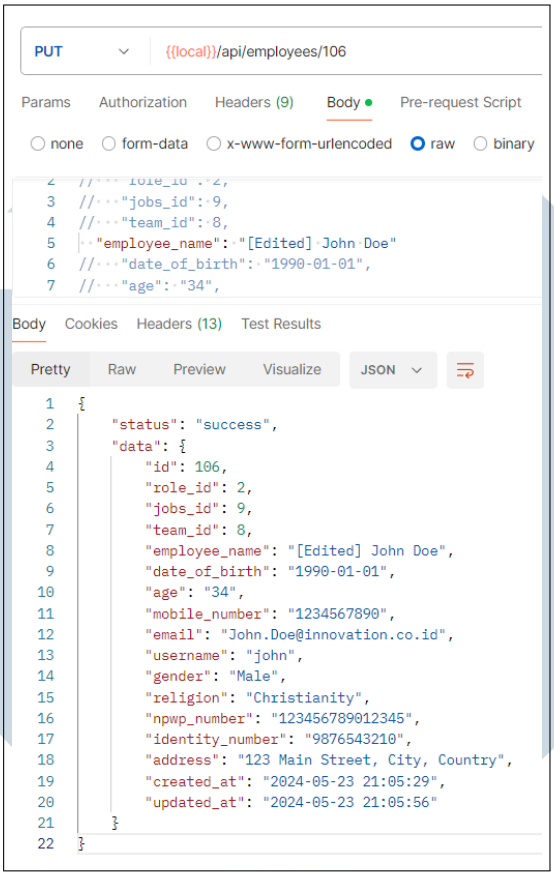
D.2 API Update user

Pada fitur perubahan data karyawan, semua atribut dapat diubah sesuai dengan kebutuhan sehingga fungsi untuk mengubah data dibuat secara dinamik dengan mengubah data sesuai dengan data yang diterima saja. dari setiap data yang diterima juga akan melewati fungsi validasi seperti yang terlampirkan pada Gambar 3.11.

```php
$rules = [
    'role_id' => 'sometimes|exists:mg_roles,id',
    'jobs_id' => 'sometimes|exists:mg_jobs,id',
    'team_id' => 'sometimes|exists:mg_teams,id',
    'employee_name' => 'sometimes|string',
    'date_of_birth' => 'sometimes|date',
    'age' => 'sometimes|string',
    'mobile_number' => 'sometimes|string',

    'email' => 'sometimes|email',
    'username' => 'sometimes|string',
    'gender' => 'sometimes|in:male,female,Male,Female',
    'religion' => 'sometimes|string',
    'npwp_number' => 'sometimes|string',
    'identity_number' => 'sometimes|string',
    'address'=>'sometimes|string',
    'password' => [
        'sometimes',
        'string',
        'min:8',
        'regex:/^(?=.*[A-Z])(?=.*[!@#$%^&*()-_=+{};:,<.>ԱԺ~]).*$/',
    ],
];
$data = $request->all();

$validator = Validator::make($data, $rules);

if ($validator->fails()){
    return response()->json([
        'status' => 'error',
        'message' => $validator->errors()
    ], 400);
}
```

Gambar 3.11. Potongan kode untuk validasi *input API update user*

Jika semua data yang dikirimkan kepada *API* tervalidasi dengan benar maka respons yang dikirimkan akan terlihat seperti Gambar 3.12.



Gambar 3.12. Tampilan percobaan *API update user*

3.3.3 Pembuatan API Autentikasi User

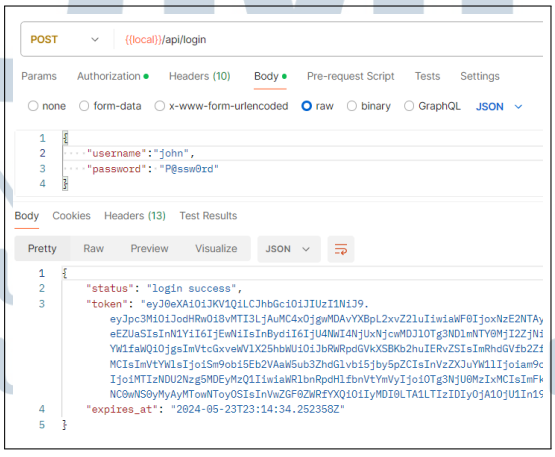
Setelah *admin* atau *HRD* membuatkan akun untuk karyawan baru, karyawan telah memiliki akun yang dapat digunakan untuk mengakses *website* dapat menggunakan fitur-fitur dari MYGIT.

A. API login

```php
public function login(Request $request) {
    try {
        $request->validate([
            'username' => 'required|string',
            'password' => 'required|string',
        ]);
        if (!$employee = Employees::where('username', $request->username)->first()) {
            return response()->json(['error' => 'Invalid credentials'], 401);
        }
        if (!Hash::check($request->password, $employee->password)) {
            return response()->json(['error' => 'Invalid credentialss'], 401);
        }
        $token = JWTAuth::claims([
            'data' => $employee
            ])->attempt($request->only('username', 'password'));
        $expiresAt = Carbon::now()->addHour();

        return response()->json([
            'status' => 'login success',
            'token' => $token,
            'expires_at' => $expiresAt
        ], 200);

    } catch (ValidationException $e) {
        return response()->json([
            'status' => 'error',
            'message' => $e->validator->errors()->first(),
        ], 400);
    }
}
```

Gambar 3.13. Potongan Kode *API login*

Pada fungsi *API login* yang dilampirkan pada Gambar 3.13, sistem autentikasi yang digunakan adalah menggunakan JWT Token yang melakukan enkripsi pada data karyawan seperti nama, *role, job*, dan *team* menjadi sebuah variabel dengan nilai yang terlihat acak yang dapat dilakukan deskripsi untuk memperoleh data tersebut pada sisi *frontend*. JWT Token memiliki waktu kadaluwarsa, sehingga jika telah melewati masa kadaluwarsa maka harus melakukan proses *login* kembali. Pada Gambar 3.14 merupakan hasil pada *API login* jika *username* dan *password* yang diisi adalah benar.



Gambar 3.14. Tampilan percobaan *API login*

B. API ganti password akun karyawan

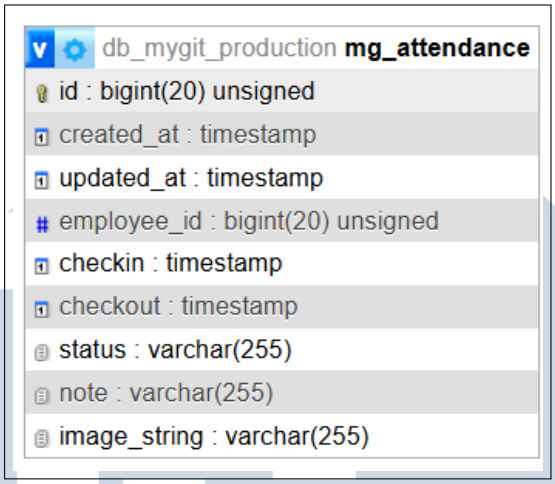
Karyawan dapat melakukan perubahan *password* yang dapat dilakukan oleh pemilik akun tersebut. Untuk pembuatan *password* baru memiliki ketentuan seperti diperlukan huruf besar, simbol, serta angka untuk memperkuat *password*.

```php
public function changePassword(Request $request, $id) {
    try{
        $data = $request->validate([
            'password' => [
                'required',
                'string',
                'min:8',
                'regex:/^(?=.*[A-Z])(?=.*[!@#$%^&*()-_=+{};:,<.>ยง~]).*$/',
            ],
        ]);
        $employee = Employees::find($id);

        if (!$employee) {
            return response()->json([
                'status' => 'error',
                'message' => 'Employee not found'
            ], 404);
        }
        $newPassword = $data['password'];
        if (!preg_match('/^(?=.*[A-Z])(?=.*[!@#$%^&*()-_=+{};:,<.>ยง~]).*$/', $newPassword)) {
            return response()->json([
                'status' => 'error',
                'message' => 'Password must contain at least 8 characters, one uppercase letter, and one symbol'
            ], 400);
        }

        $employee->password = Hash::make($newPassword);
        $employee->save();
        return response()->json([
            'status' => 'success',
            'message' => 'Password berhasil diubah'
        ], 200);
    }catch (\Exception $e) {
        return response()->json([
            'status' => 'error',
            'message' => 'Failed to change password: ' . $e->getMessage()
        ], 500);
    }
}
```

Gambar 3.15. Potongan Kode *API* ganti password akun karyawan

3.3.4 Pembuatan API Presensi Kehadiran Karyawan

Salah satu fitur utama yang dimiliki MYGIT adalah fitur presensi kehadiran karyawan. Fitur ini membantu operasional perusahaan dalam melakukan pencatatan dalam kehadiran dari setiap karyawan GIT. Presensi dapat dilakukan oleh setiap karyawan yang terdaftar dan memiliki akun dan karyawan dapat melihat riwayat dari semua presensi pada hari-hari sebelumnya. *Admin* dan *HRD* dapat melakukan monitor pada semua kehadiran karyawan yang ada.



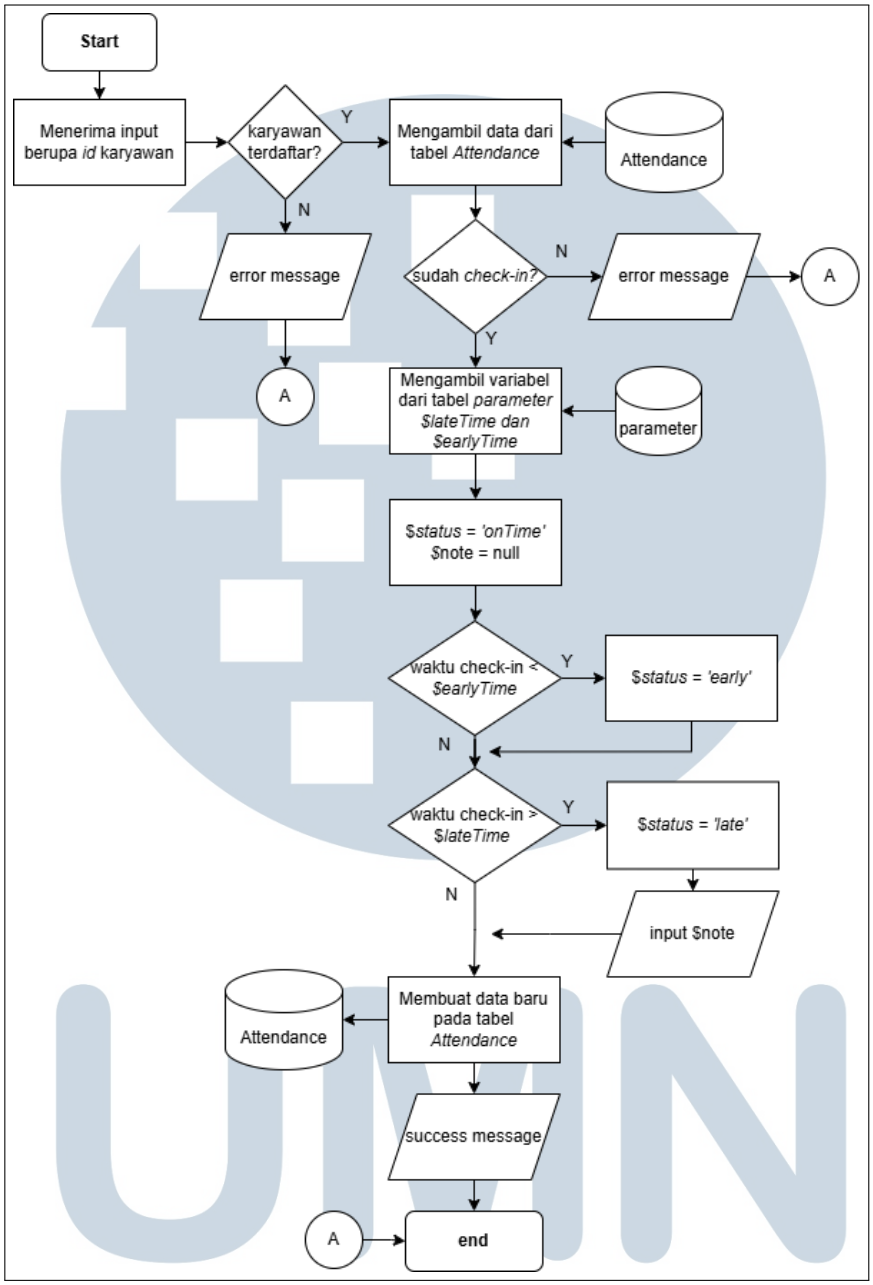
Gambar 3.16. Atribut basis data dari tabel *attendance*

Dapat dilihat pada gambar 3.16 bahwa setiap presensi dilakukan, terdapat data seperti *id* karyawan, waktu *checkin*, waktu *checkout*, status yang menjadi tanda jika karyawan tepat waktu atau telat, serta *note* yang digunakan sebagai informasi yang dimasukkan karyawan yang telat untuk justifikasi keterlambatan melakukan presensi. Terdapat permintaan dari perusahaan untuk menggunakan foto sebagai bukti presensi, akan tetapi pada pertengahan perancangan diputuskan untuk membatalkan permintaan ini.

Fitur *API* presensi kehadiran karyawan meliputi fungsi seperti *check-in* untuk pencatatan kehadiran, *check-out* untuk pencatatan kapan pekerjaan dari karyawan selesai, pengambilan data riwayat *check-in* untuk tiap karyawan, dan fitur *auto check-out* yang melakukan *check-out* secara otomatis kepada semua karyawan yang lupa untuk melakukan *check-out*.

A. API check-in

Saat karyawan melakukan presensi pada *website* MYGIT, terdapat beberapa validasi sebelum data presensi tercatat untuk menghindari terjadi *bug*. Data yang disimpan adalah status dari ketepatan waktu kehadiran karyawan dan *note* justifikasi ketika karyawan telat melakukan presensi, Proses pencatatan hingga penyimpanan data dapat dilihat pada Gambar 3.17.



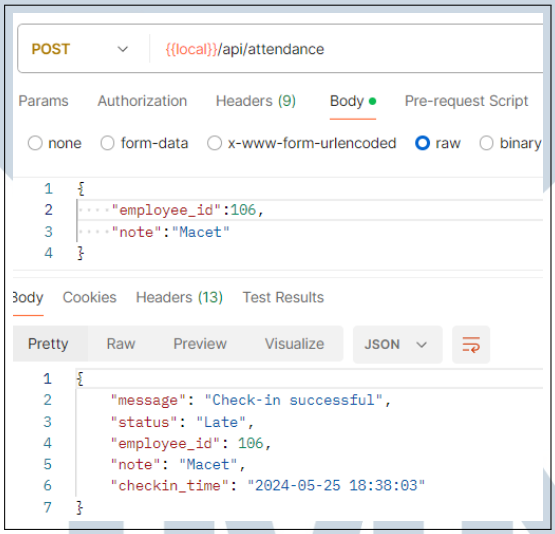
Gambar 3.17. *Flowchart API check-in*

Pada Gambar 3.17, terdapat potongan kode yang memanggil tabel *Parameter*. Tabel parameter bertujuan untuk menyimpan variabel penting yang dapat digunakan pada seluruh kode. Seperti contoh, data yang diambil dari tabel *Parameter* adalah *id* dari data *earlyTime* yang digunakan sebagai batas jam masuk perusahaan dan *lateTime* yang digunakan sebagai batas jam telat perusahaan. Hal ini mempermudah pada saat terjadi perubahan peraturan kehadiran kantor yang juga memungkinkan *Admin* untuk mengubah nilai dari variabel tersebut tanpa harus mengubah kode.

Parameters	Value
Jatah Cuti Tahunan	12
Jam Masuk	08:30
Jam Telat	09:00
Running Text Dashboard	"Bersatu adalah sebuah awal; menjaga kebersamaan adalah kemajuan; bekerja sama adalah sukses. " - Henry Ford

Gambar 3.18. Tampilan data dari tabel *Parameter*

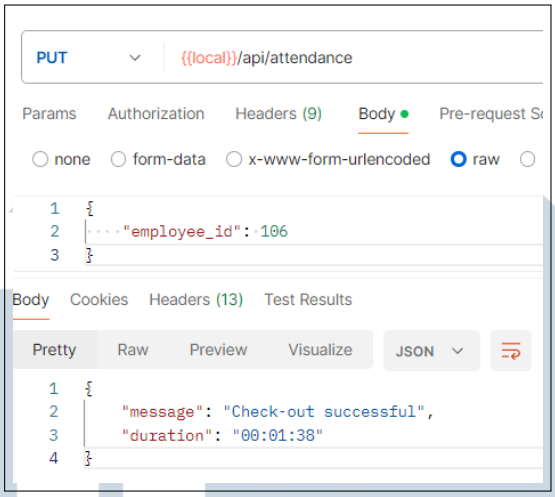
Jika proses presensi kehadiran berhasil maka respons yang diberikan dilihat pada Gambar 3.19. Pada Gambar tersebut, terlihat bahwa karyawan yang melakukan presensi mendapatkan status *"late"* dikarenakan percobaan *API* dilakukan pada waktu yang melebihi jam telat. *Note* justifikasi keterlambatan pun telah berhasil masuk pada tabel *Attendance*.



Gambar 3.19. Tampilan percobaan *API check-in*

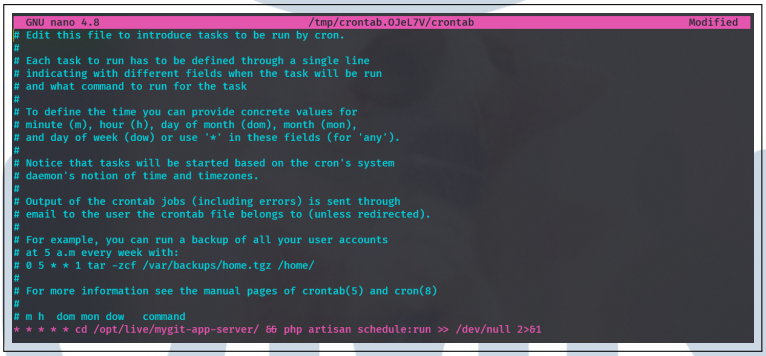
B. API Check-out

Pada proses melakukan *check-out*, *API* mengambil data kehadiran pada tabel *Attendance* menggunakan *id* dari karyawan dan mencari data terakhir dilakukan *check-in*. Kemudian atribut *"checkout"* yang terdapat pada salah satu data dari tabel tersebut akan diubah menjadi nilai waktu (hari dan jam) pada saat dilakukan *check-out* dan menghitung durasi berapa lama karyawan bekerja. Respons dari *API check-out* dapat dilihat pada Gambar 3.20.



Gambar 3.20. *Flowchart API check-out*

Pada sistem, terdapat fungsi yang dapat melakukan *check-out* secara otomatis pada jam yang telah ditentukan. Digunakan bantuan teknologi bawahan dari *OS Linux Ubuntu* yang bernama Cronjob untuk menjalankan fungsi *check-out* secara otomatis ini. Cronjob dapat menjadwalkan kapan suatu fungsi dijalankan. Pada Gambar 3.21.



Gambar 3.21. Potongan kode konfigurasi cronjob pada server

Cronjob melakukan eksekusi pada fungsi *schedule* yang terdapat pada *kernel backend*. Tampilan kode *kernel* dan *API auto check-out* dapat dilihat pada Gambar 3.22 dan Gambar 3.23.

```php
protected function schedule(Schedule $schedule): void
{
    $schedule->command('sanctum:purge')->daily();

    $schedule->call(function () {
        $response = app(AttendanceController::class)->autoCheckOut(request());

        \Log::info('Auto Checkout API call task executed at ' . now());
    })->DailyAt('18:00');

}
```

Gambar 3.22. Potongan kode konfigurasi *scheduler* pada kernel

```php
public function autoCheckOut()
{
    try {
        DB::beginTransaction();

        $result = Attendance::whereNull('checkout')
            ->update(['checkout' => now()]);

        DB::commit();

        return response()->json([
            'status' => 'success',
            'message' => 'Auto checkout successful for ' . $result . ' employees'
        ], 200);

    } catch (\Exception $e) {
        DB::rollback();

        return response()->json([
            'status' => 'error',
            'message' => 'Failed to auto checkout: ' . $e->getMessage()
        ], 500);
    }
}
```

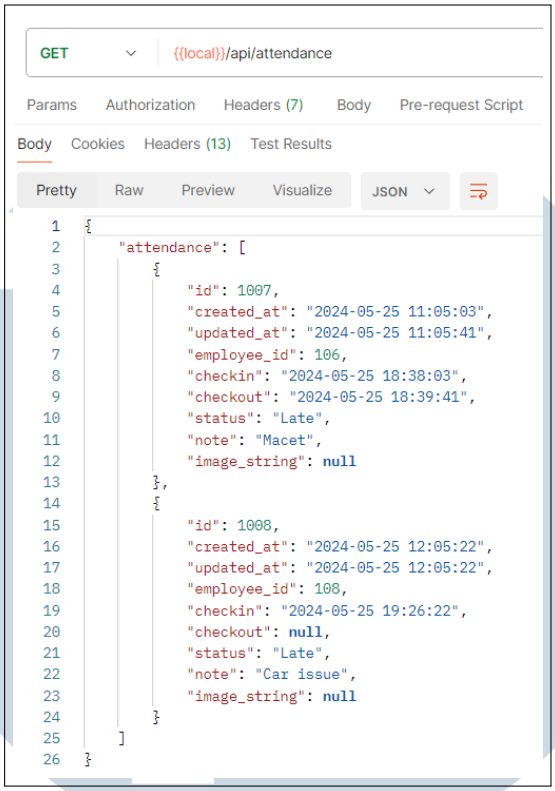
Gambar 3.23. Potongan kode *API auto check-out*

C. API Get Check-in History

Setiap melakukan presensi, data presensi tersebut akan disimpan menjadi riwayat presensi. *API* yang dipanggil dibedakan menjadi dua untuk tiap *role*.

C.1 API get check-in untuk admin

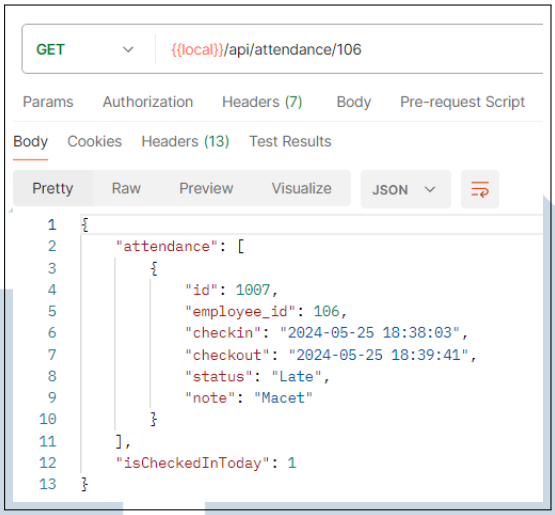
API ini dapat mengirimkan semua data riwayat presensi dari semua karyawan yang telah melakukan presensi di setiap harinya. *API* ini dirancang untuk *role admin* agar dapat melakukan pengawasan maupun pencatatan pada setiap karyawan yang hadir. Hasil dari *API* ini dapat dilihat pada Gambar 3.24.



Gambar 3.24. Tampilan percobaan *API get check-in* untuk *admin*

C.2 API get check-in untuk karyawan

API ini dikhususkan untuk mengirimkan data riwayat presensi oleh salah satu karyawan saja. *API* ini dirancang untuk karyawan yang ingin melihat riwayat dari presensi pribadinya. *API* ini juga mengirimkan data ”*isCheckedIn*”, data ini membantu tampilan *website* pada menu presensi untuk membuat *conditional rendering* pada tombol *check-in*. Jika ”*isCheckedIn*” bernilai 1, maka tombol *check-in* menjadi *check-out* dan begitu pula sebaliknya jika bernilai 0. Hasil dari *API* ini dapat dilihat pada Gambar 3.24.

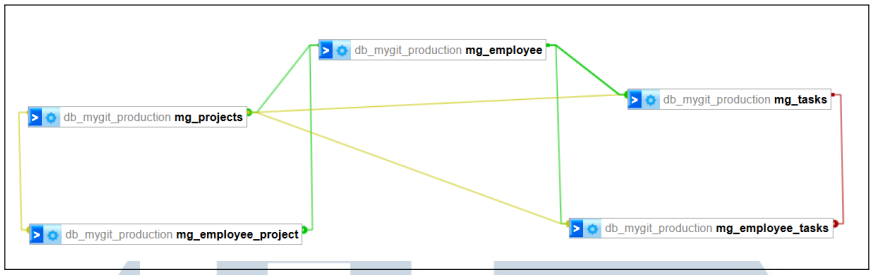


Gambar 3.25. Tampilan percobaan *API get check-in* untuk karyawan

3.3.5 Pembuatan API Project dan Task

Fitur *project* dan *task* merupakan fitur utama yang menjadi alasan GIT merancang MYGIT. Fitur ini digunakan untuk mengelola semua proyek dijalankan maupun yang ditangani oleh GIT, termasuk proyek perancangan aplikasi manajemen karyawan berbasis *website* MYGIT. Setiap *project* ditangani oleh satu *Project Manager* yang bertugas menyelesaikan proyek dengan cara memberikan penugasan berupa *task* kepada setiap *Technical Consultant* untuk terlibat pada proses penyelesaian proyek tersebut. Dengan adanya fitur ini, maka operasional perusahaan akan terbantu dalam melakukan monitor terhadap setiap kemajuan proyek yang ada serta dapat melakukan monitor terhadap pekerjaan yang diberikan kepada *Technical Consultant* yang terlibat.

Setiap proyek memiliki tugas di dalamnya dan setiap proyek memiliki penanggung jawab yaitu *Project Manager* dan juga *Technical Consultant* yang dilibatkan. Sehingga pada perancangan basis data untuk menyimpan data dari *project* maupun *task* menghasilkan relasi terhadap tabel *project*, tabel *task* dan tabel *employee*. Dari relasi tersebut menghasilkan 2 tabel baru yang merupakan sebuah tabel relasi antara tabel *employee* dan tabel *project* serta tabel *employee* dan tabel *task*. Tabel tersebut berupa tabel *employee_project* dan tabel *employee_task*. Visualisasi relasi antara tabel-tabel tersebut dapat dilihat pada Gambar 3.26.

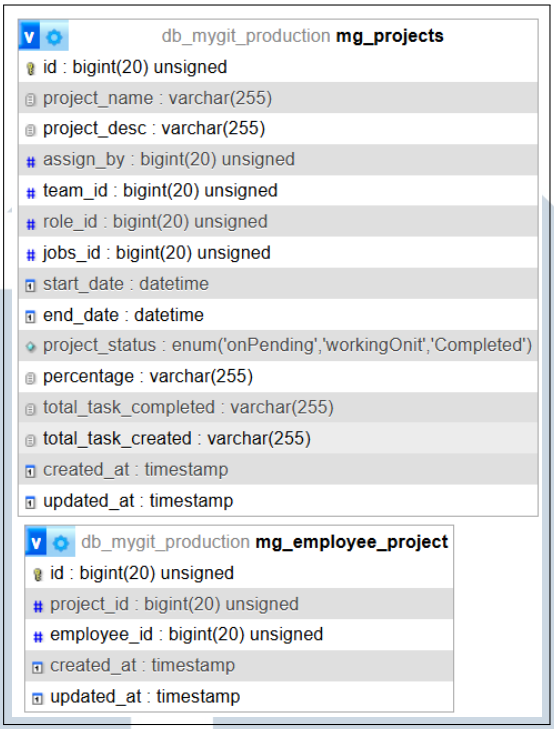


Gambar 3.26. Visualisasi relasi tabel antara tabel *project, task,* dan *employee*

A. API Project

API project meliputi fungsi untuk membuat proyek baru, memperbaharui data proyek, mengambil data proyek, penghapusan data proyek, serta pembaharuan status proyek. Untuk pembuatan data proyek dibutuhkan data karyawan dengan kedudukan yang berbeda, yaitu *Technical Consultant* yang akan terlibat untuk pengerjaan dari proyek tersebut serta *Project Manager* yang akan bertugas untuk mengelola proyek dan melibatkan karyawan ke dalam proyek.

Terlihat pada Gambar 3.27, pada tabel *project* menyimpan data nama (*project_name*) dan deskripsi proyek (*project_desc*), tanggal mulai (*start_date*) dan berakhir (*end_date*) proyek, serta karyawan yang membuat proyek tersebut (*assign_by*). Terdapat juga informasi terkait *tim, role,* dan *job* yang terlibat dalam proyek. Jumlah tugas terbuat (*total_task_created*) dan selesai (*total_task_completed*) juga tersimpan pada tabel. Data persentase (*percentage*) dan status proyek (*project_status*) disimpan untuk merepresentasikan pencapaian dari proyek tersebut. Pada tabel *employee_project* menyimpan informasi *id* karyawan yang terlibat dengan proyek. Tabel ini akan sangat membantu dalam melakukan filtrasi pada saat mengirimkan data tanpa harus memperbanyak penampungan data dalam satu tabel.



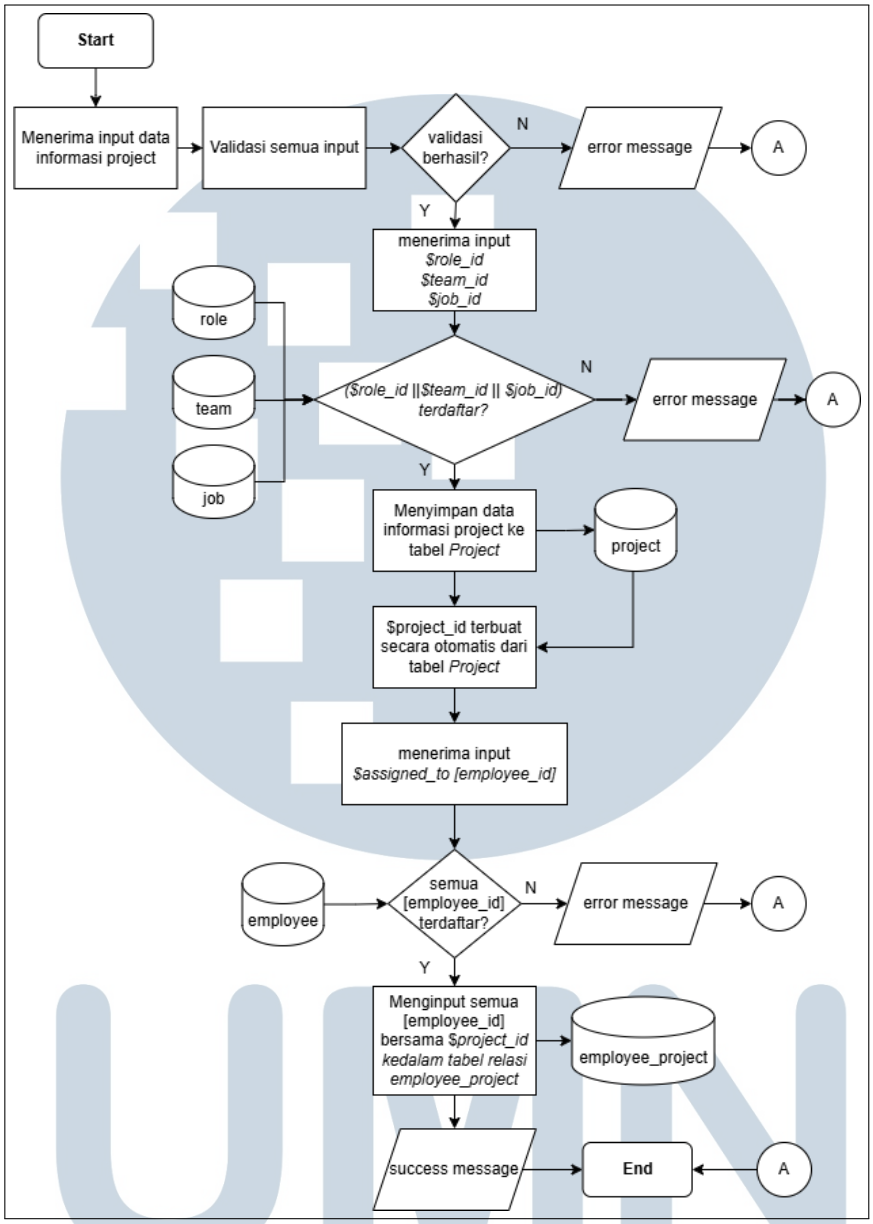
Gambar 3.27. Atribut basis data dari tabel *project* dan *employee_project*

A.1 API Create Project

Pada Gambar 3.28 merupakan alur proses saat melakukan penambahan data proyek baru. Dimulai dari menerima *input* informasi data proyek seperti nama proyek, deskripsi proyek, tanggal mulai dan berakhir proyek, serta penanggung jawab proyek. Kemudian data tersebut akan dilakukan validasi *input*.

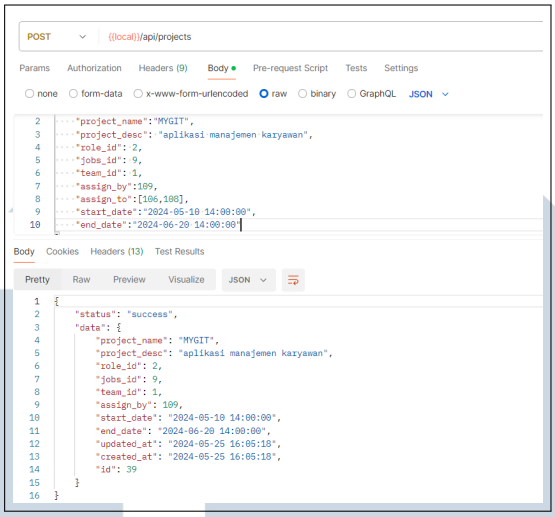
Jika data tersebut berhasil maka akan melanjutkan validasi terhadap *input role_id, team_id*, dan *job_id* yang terlibat untuk melakukan pengecekan jika data tersebut telah terdaftar atau belum pada tabel masing-masing. Kemudian semua data akan ditambahkan pada tabel *project* dan kemudian akan menghasilkan atribut *project_id* dari tabel *project*.

Setelah berhasil mendapatkan *project_id*, dilakukan validasi *input* terhadap *id* karyawan-karyawan yang dilibatkan pada proyek ini. Pengecekan dilakukan dengan mengecek apakah *id* karyawan-karyawan tersebut telah terdaftar pada tabel *employee* atau belum. Jika validasi berhasil maka semua *id* karyawan bersamaan dengan *project_id* ditambahkan ke dalam tabel relasi *employee_project* dengan hasil yang dapat dilihat pada Gambar.



Gambar 3.28. *Flowchart API create project*

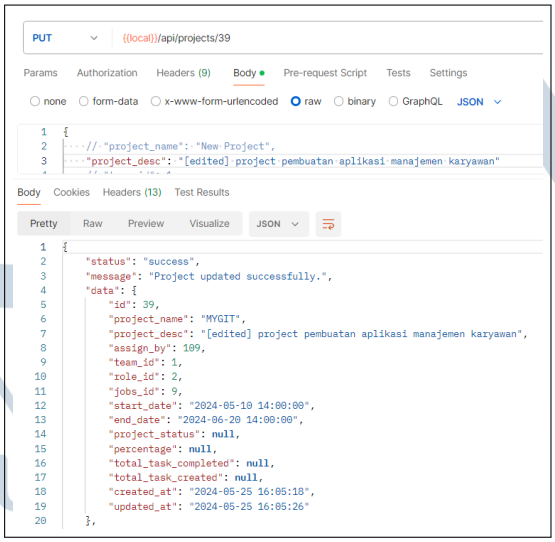
Jika semua proses dilewati dengan benar, maka *API* akan memberikan respons seperti pada Gambar 3.29. Respons yang diberikan berupa data dari proyek yang baru saja dibuat.



Gambar 3.29. Tampilan percobaan *API create project*

A.2 API Update Project

Proses pembaharuan data pada data proyek memiliki struktur *input* yang sama dengan proses penambahan data proyek pada *API create project*, tetapi hanya diperlukan *input* data yang ingin diperbaharui saja dan memerlukan *id project* agar dapat memilih data proyek mana yang ingin diperbaharui. Jika data proyek berhasil diperbaharui, maka respons dari *API* akan terlihat seperti pada Gambar 3.30.



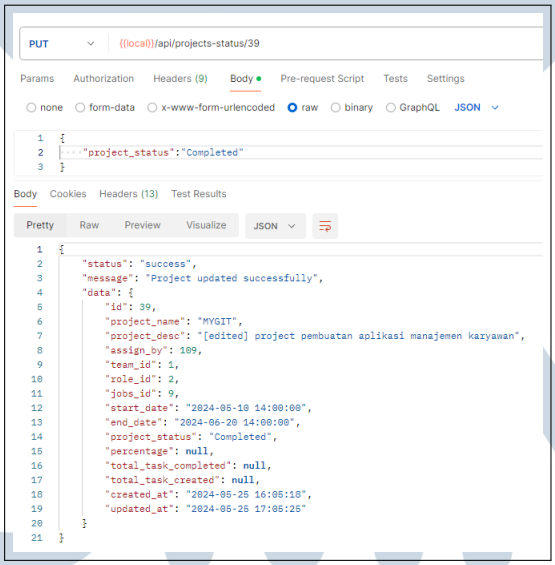
Gambar 3.30. Tampilan percobaan *API update project*

A.3 API Update Project Status

Untuk menghindari kesalahan saat melakukan pembaharuan yang memperbolehkan banyak data yang diubah, maka dilakukan pemisahan saat melakukan pembaharuan data *project* keseluruhan dan pembaharuan status *project*.

Ketika persentase proyek telah mencapai 100%, proyek tidak secara otomatis melakukan perubahan pada statusnya, hal ini biarkan agar *Project Manager* dapat melakukan pemeriksaan kembali ketika proyek telah mencapai status penyelesaian mencapai 100%.

Untuk melakukan pembaharuan status proyek, diperlukan permintaan *API update project status* dan menyertakan *input* dari status yang diinginkan seperti pada Gambar 3.31. *API* akan melakukan pembaharuan pada atribut status sesuai dengan *input* yang diterima. Jika pembaharuan berhasil, maka respons dari *API* akan terlihat seperti pada Gambar 3.31.



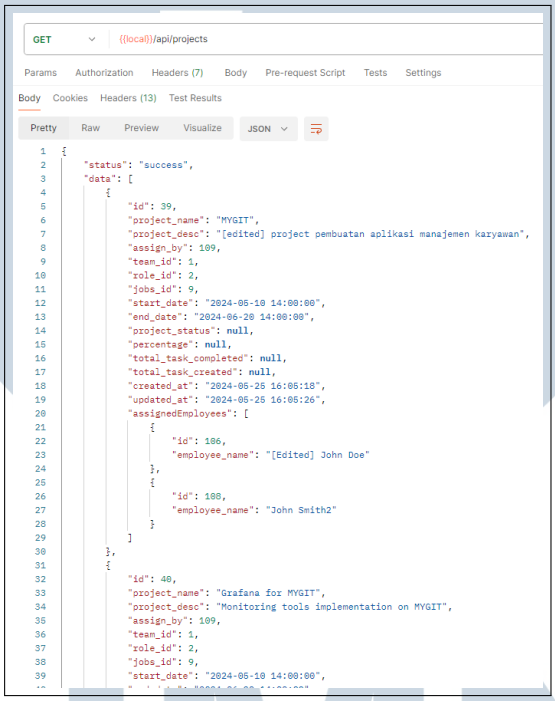
Gambar 3.31. Tampilan percobaan *API update project status*

A.4 API Get Project Data

Terdapat 2 metode untuk mengambil data proyek guna mempermudah saat akan menampilkan seluruh data proyek yang ada maupun menampilkan satu data proyek saja.

A.4.1 Mengambil semua data project

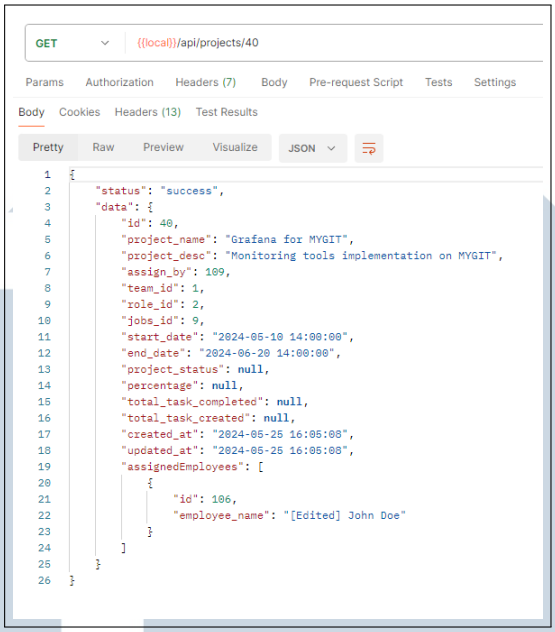
Untuk mendapatkan semua data proyek yang ada, diperlukan permintaan *API* yang disediakan untuk mengirim semua data proyek yang ada pada tabel *project*. *API* akan melakukan *query* untuk mengambil semua data proyek yang dan mengirimkan data yang ditemukan sebagai respons. Respons dari *API* tersebut dapat dilihat pada Gambar 3.32.



Gambar 3.32. Tampilan percobaan *API* untuk mengambil semua data proyek

A.4.2 Mengambil data per project

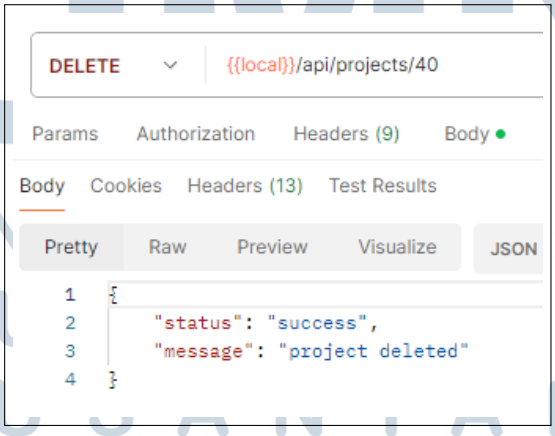
Untuk mendapatkan salah satu data proyek yang ada, diperlukan permintaan *API* dengan menyertakan *id* proyek sebagai parameter pada *route*, kemudian *API* melakukan pencaharian data proyek berdasarkan *id* proyek pada tabel *project*, jika data ditemukan maka *API* akan mengirimkan data dalam bentuk respons. Respons dari *API* tersebut dapat dilihat pada Gambar 3.33.



Gambar 3.33. Tampilan percobaan *API* untuk mengambil data per proyek

A.5 API Delete Project

Untuk melakukan penghapusan salah satu data proyek, diperlukan permintaan *API* dengan menyertakan *id* proyek sebagai *parameter* pada *route*. *API* akan mencari data proyek dengan menggunakan *id* proyek dari *parameter* dan melakukan *query* penghapusan data pada data proyek yang sesuai dengan *id* proyek tersebut. Jika penghapusan data proyek berhasil, maka respons yang diberikan adalah seperti pada Gambar 3.34.

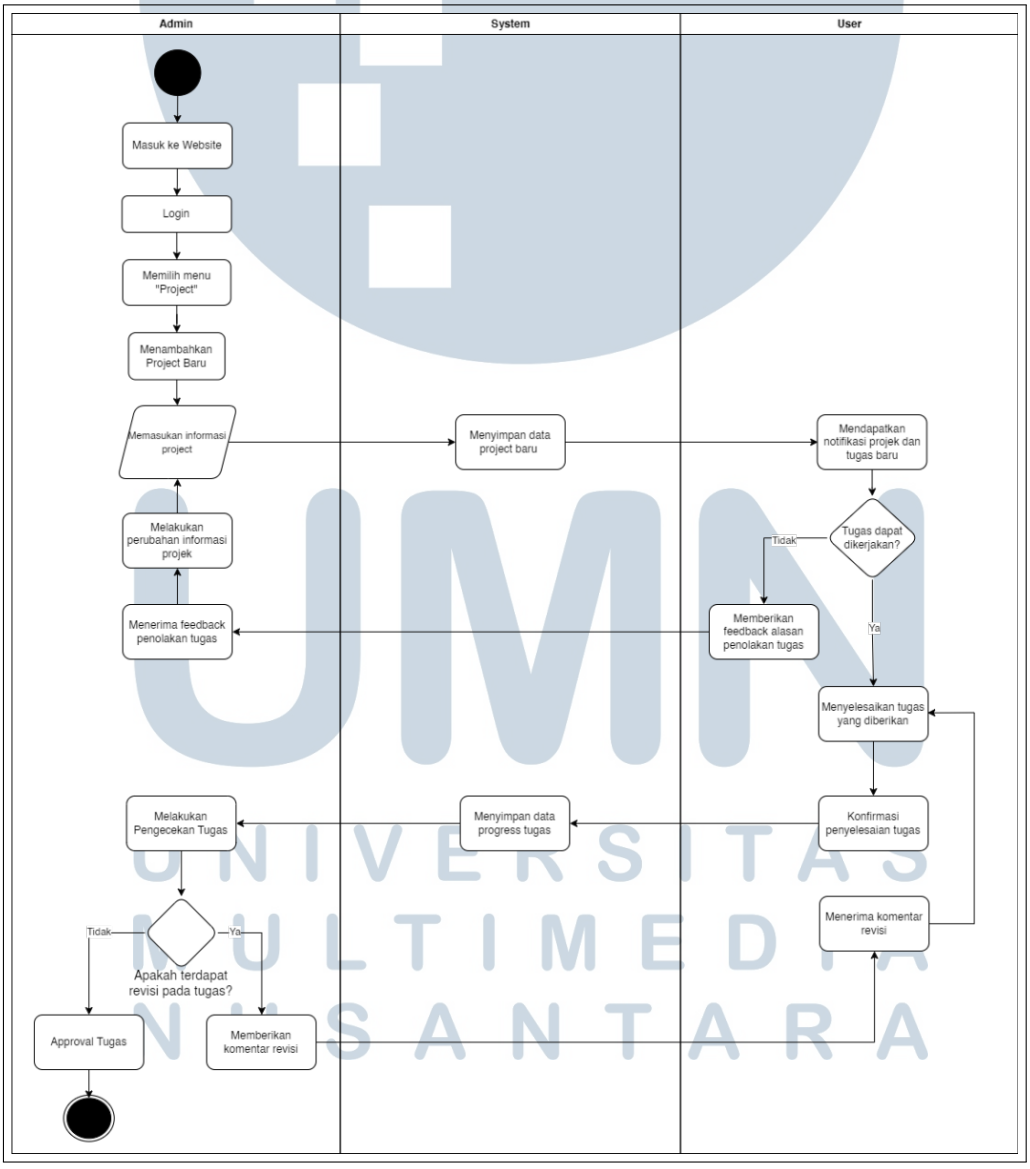


Gambar 3.34. Tampilan percobaan *API delete project*

B. API Task

API task dapat digunakan setelah melakukan penambahan data proyek. *API task* meliputi fungsi untuk membuat data tugas baru, memperbaharui data tugas, pengambilan data tugas, penghapusan data tugas, sistem invitasi tugas beserta sistem terima/tolak tugas, dan pengumpulan dan penilaian tugas. Fitur-fitur pada *API task* ini digunakan untuk koordinasi akan tugas-tugas detail dari proyek antara *Project Manager* dan *Technical Consultant* yang terlibat.

Untuk alur penggunaan fitur ini dapat dilihat pada *activity diagram* yang terlampir pada Gambar 3.35.



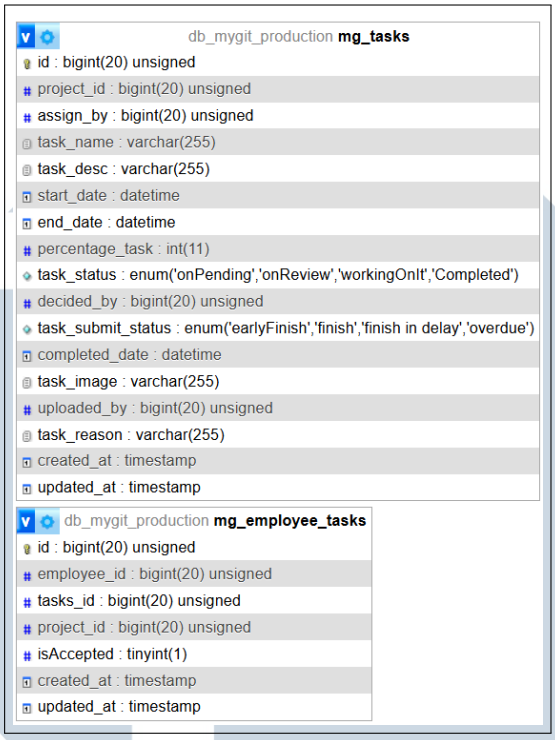
Gambar 3.35. *Activity diagram* fitur *project* dan *task*

Technical Consultant yang dilibatkan akan menerima invitasi untuk mengerjakan tugas. Invitasi tersebut dapat ditolak dan diterima untuk dikerjakan, jika ditolak maka tugas akan dikembalikan kepada *Project Manager* untuk melakukan revisi penugasan, jika diterima maka status tugas akan menjadi ”*workingOnIt*” atau sedang mengerjakan.

Setelah menyelesaikan tugas, *Technical Consultant* dapat melakukan pengumpulan pada tugas tersebut. Untuk melakukan pengumpulan tugas, diperlukan bukti berupa potongan gambar yang menjadi bukti bahwa tugas telah selesai dan status akan berubah menjadi ”*onReview*”. Kemudian *Project Manager* akan menerima hasil pengumpulan tersebut dan dapat melakukan revisi maupun menerima hasil pekerjaan. Jika *Project Manager* meminta revisi, maka status akan kembali menjadi ”*workingOnIt*” dan *Technical Consultant* yang terlibat harus melakukan revisi pada penugasan tersebut. Jika diterima maka status akan berubah menjadi ”*Completed*” dan persentase *project* akan bertambah.

Terlihat pada Gambar 3.36, tabel *task* menyimpan data informasi terkait tugas seperti nama tugas (*task_name*), deskripsi tugas (*task_desc*), tanggal mulai (*start_date*) dan akhir tugas (*end_date*), serta karyawan yang memberikan tugas (*assigned_by*). Terdapat juga data seperti persentase tugas (*percentage_task*) yang akan menjadi bobot pada persentase proyek, status tugas saat ini (*task_status*) untuk mengetahui apakah tugas tersebut belum diterima, sedang dikerjakan, atau sudah selesai, serta data siapa yang menyetujui invitasi tugas (*decided_by*). Saat melakukan pengumpulan, beberapa data yang akan tercatat adalah seperti data status pengumpulan tugas (*task_submit_status*) disimpan untuk mencatat status ketepatan waktu tugas saat dikumpulkan, data waktu pengumpulan (*completed_date*), data karyawan yang melakukan pengumpulan tugas (*uploaded_by*), dan gambar yang digunakan sebagai bukti penyelesaian tugas (*task_image*).

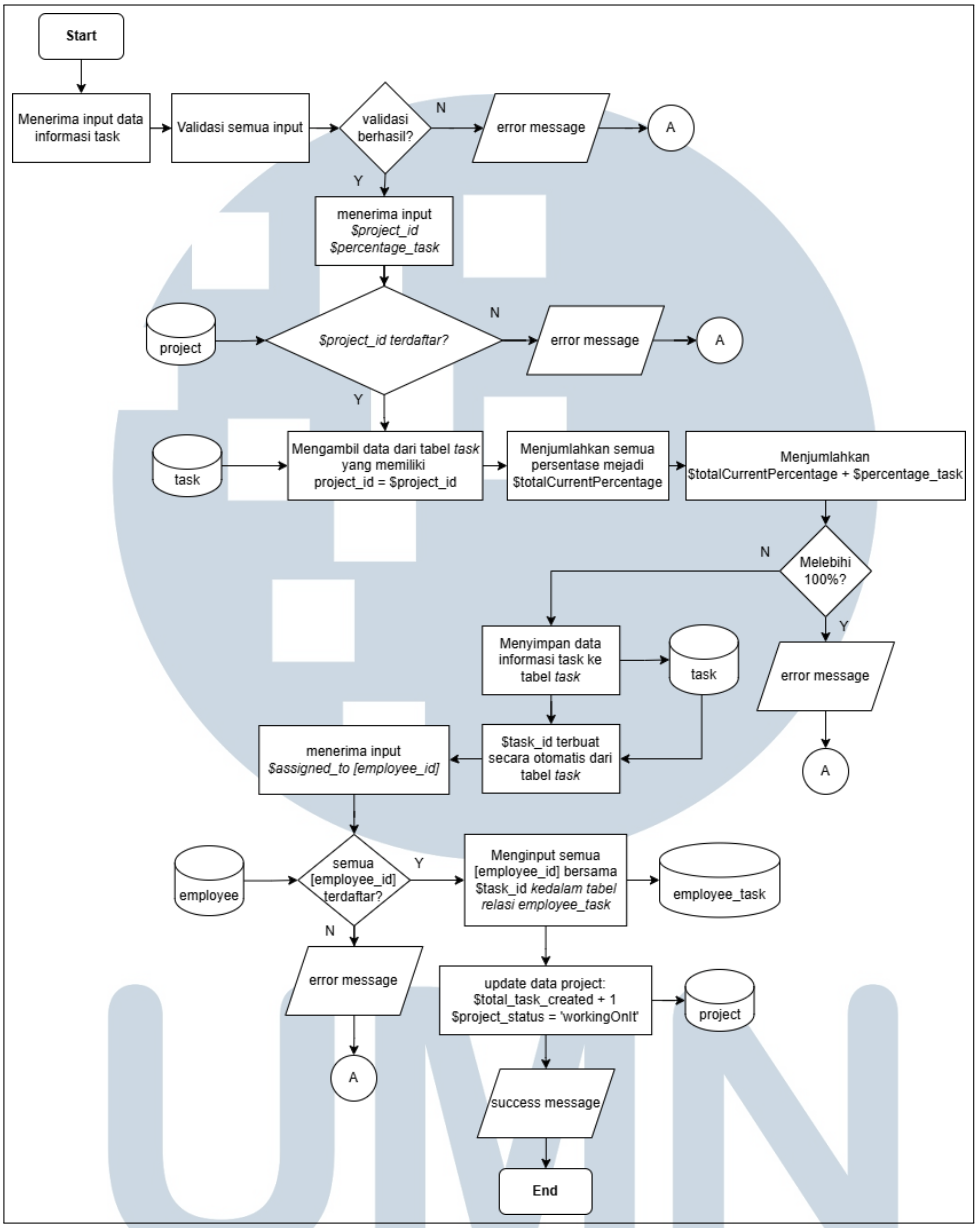
Pada tabel relasi *employee_task*, data yang disimpan adalah *id* dari karyawan, tugas yang saling berkaitan serta data untuk menandakan bahwa tugas telah diterima untuk dikerjakan (*isAccepted*), ditambahkan data untuk melakukan pengecekan penerimaan tugas agar dapat digunakan sebagai validasi saat secara tidak sengaja melakukan proses penerimaan tugas lebih dari dua kali.



Gambar 3.36. Atribut basis data dari tabel *task* dan *employee_task*

B.1 API Create Task

Pada Gambar 3.37, terlihat sebuah alur yang dilakukan saat melakukan proses permintaan kepada *API create task*. Diawali dengan penerimaan *input* data informasi terkait tugas yang telah disebutkan sebelumnya dan melakukan validasi pada setiap *input* yang diterima *API*.



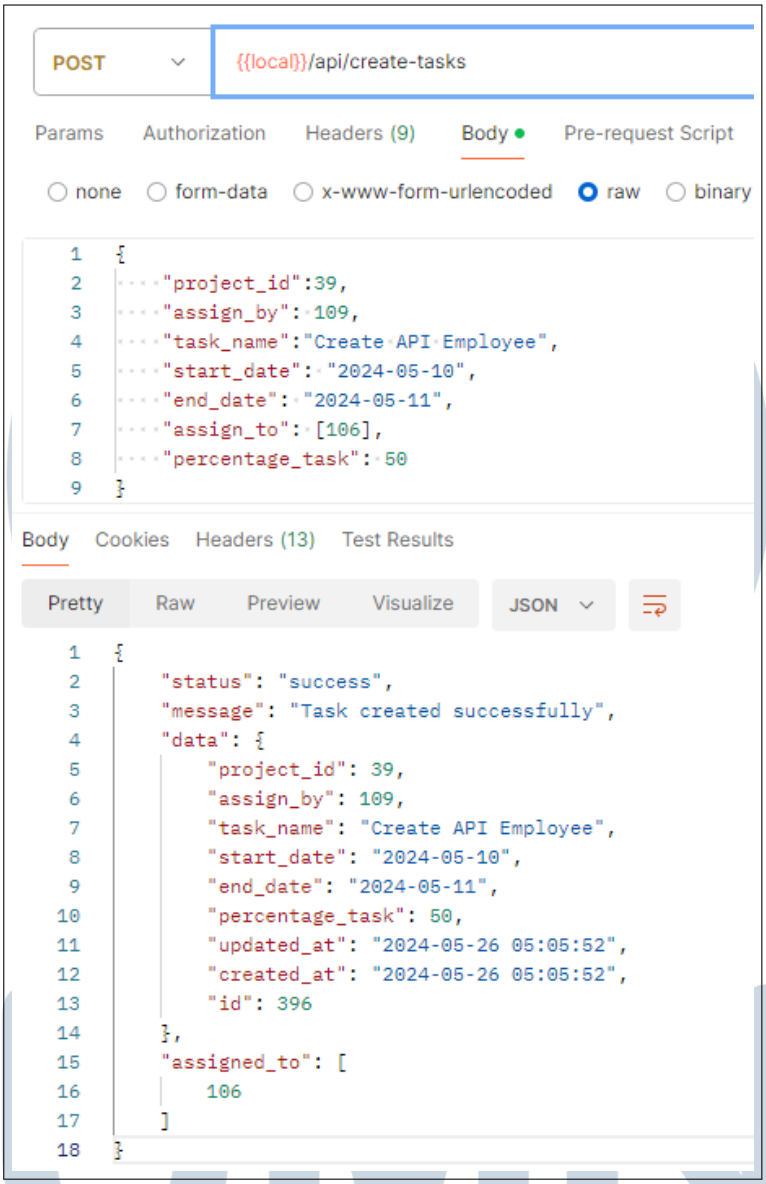
Gambar 3.37. *Flowchart API create task*

Jika *input* tersebut telah benar, dengan menggunakan *id* proyek yang diterima dari *input API* akan melanjutkan dengan mengambil informasi dari data proyek yang terkait. Kemudian *API* akan mengambil semua informasi tugas yang ada pada tabel *task* untuk mengumpulkan semua tugas yang berkaitan dengan proyek tersebut. Hal tersebut dilakukan untuk menghitung keseluruhan persentase yang dimiliki semua tigas kemudian dijadikan sebagai total persentase proyek yang akan dilakukan validasi agar total persentase tersebut tidak melebihi 100%.

Jika total persentase tidak melebihi 100%, maka *API* dapat melakukan proses penambahan data tugas baru ke dalam tabel *task*. Kemudian *API*

mengakses tabel relasi dari tabel *task* dan tabel *employee* untuk mengisi semua informasi karyawan yang terlibat pada tugas yang baru dibuat ini. Setelah semua karyawan telah dicatat pada tabel relasi tersebut, maka *API* akan lanjut melakukan pembaharuan data pada proyek terkait dengan mengubah informasi total task yang dibuat ditambah dengan jumlah tugas yang baru dibuat (*total_task_created* + 1) dan mengubah status dari proyek menjadi '*workingOnIt*'.

Setelah melewati proses validasi dan penambahan data, maka *API* akan mengirimkan respons yang menandakan bahwa proses telah dijalankan dan data tugas baru telah ditambahkan beserta mengirimkan informasi dari proyek yang baru saja telah dibuat. Hasil respons dari *API* dan contoh *input* pembuatan tugas baru dapat dilihat pada Gambar 3.38.



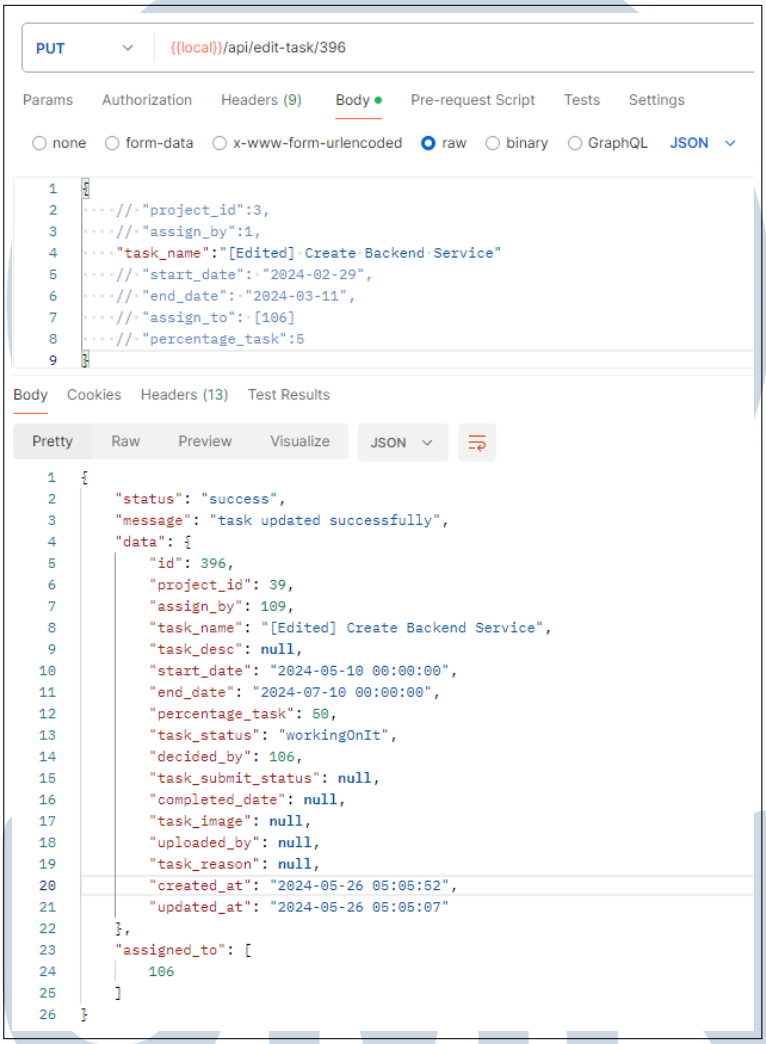
Gambar 3.38. Tampilan percobaan *API create task*

B.2 API Update Task

API update task digunakan untuk melakukan pembaharuan pada suatu informasi tugas seperti, nama, deskripsi, durasi penugasan, bobot persentase tugas, maupun mengganti karyawan yang dilibatkan pada tugas. Untuk pembaharuan status tugas, status pengumpulan tugas, serta tanggal pengumpulan tidak dapat diperbaharui secara manual dengan *API* ini karena data tersebut akan diperbaharui secara otomatis saat melakukan pengumpulan tugas.

Proses validasi dan perubahan data memiliki struktur yang sama pada *API*

create task, hanya saja *API update task* hanya mengubah data yang sudah ada saja. Contoh dari respons *API update task* ketika telah berhasil memproses *input* dari permintaan dapat dilihat pada Gambar 3.39.



Gambar 3.39. Tampilan percobaan *API update task*

B.3 API Get Task Data

Terdapat empat metode untuk mengambil data dari tugas yang telah dibuat. Metode pengambilan data tugas ini dipisahkan menjadi empat *API* yang berbeda. Hal ini dilakukan guna mempermudah pengambilan data yang sesuai oleh *frontend* dalam mengambil data sesuai kasus penggunaan.

B.3.1 Mengambil semua data task

API ini digunakan oleh *admin* untuk mendapatkan semua data *task* yang ada tanpa ada batasan akses dari *role, job,* maupun *team*. Contoh respons yang diberikan *API* ini dapat dilihat pada gambar 3.40.

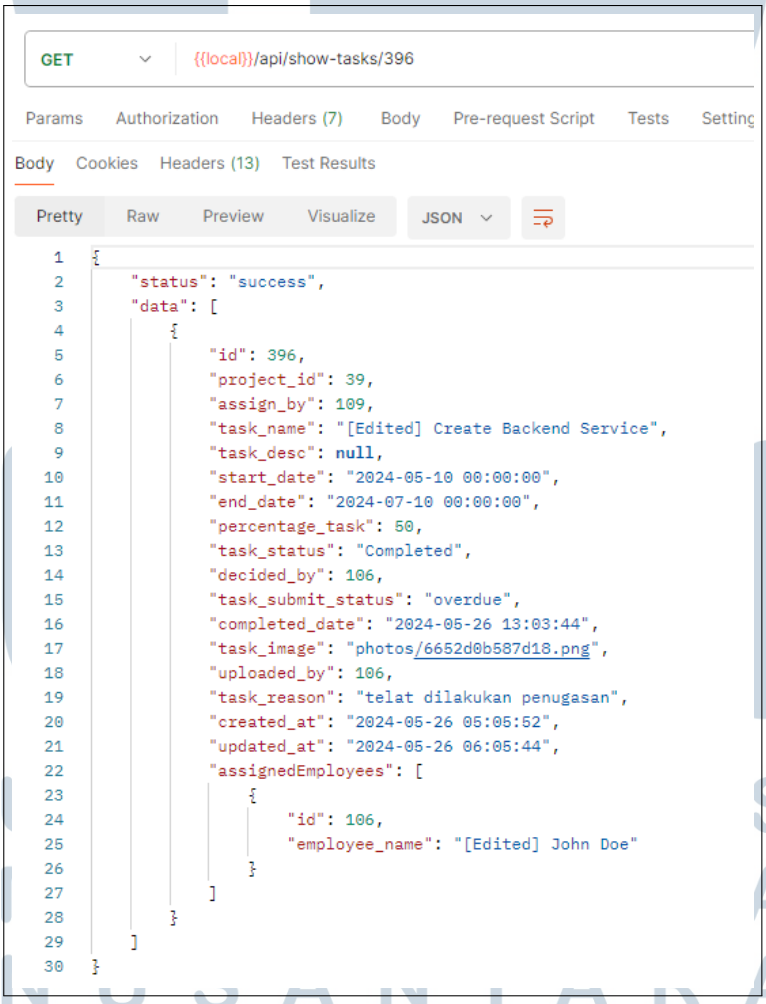
Gambar 3.40. Tampilan percobaan *API get all task*

API ini juga memanggil data karyawan guna mendapatkan nama dari karyawan yang terlibat menggunakan *id* dari karyawan yang ada pada tabel relasi *employee_task*, sehingga membantu performa pada sisi *frontend* tanpa harus memanggil *API get employee* untuk mendapatkan nama. Untuk mendapatkan data yang sesuai, digunakan fitur *join* pada saat melakukan *query* untuk memanggil data

tugas dari tabel relasi *employee_task*.

B.3.2 Mengambil data per task

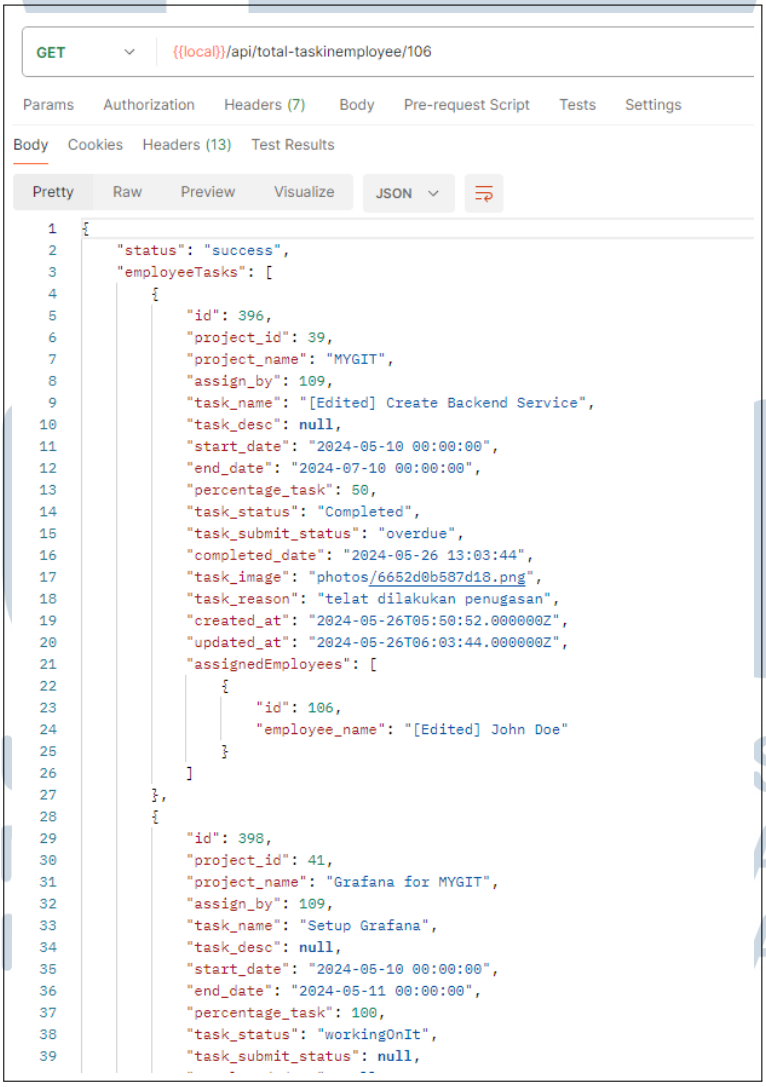
API ini digunakan ketika ingin melihat salah satu data tugas secara detail pada tampilan sisi *frontend* tanpa harus memanggil semua data tugas yang ada dan melakukan filtrasi pada data. Data tugas didapatkan dengan menggunakan *id* tugas yang didapatkan dari parameter dan memasukkan *id* tugas tersebut pada *query* untuk memanggil data tugas. Ketika data tugas ditemukan, maka *API* akan mengambil data tugas dan mengirimkan respons berhasil beserta data tugas. Hasil respons dapat dilihat pada Gambar 3.41.



Gambar 3.41. Tampilan percobaan *API get task by id*

B.3.3 Mengambil semua data task per Karyawan

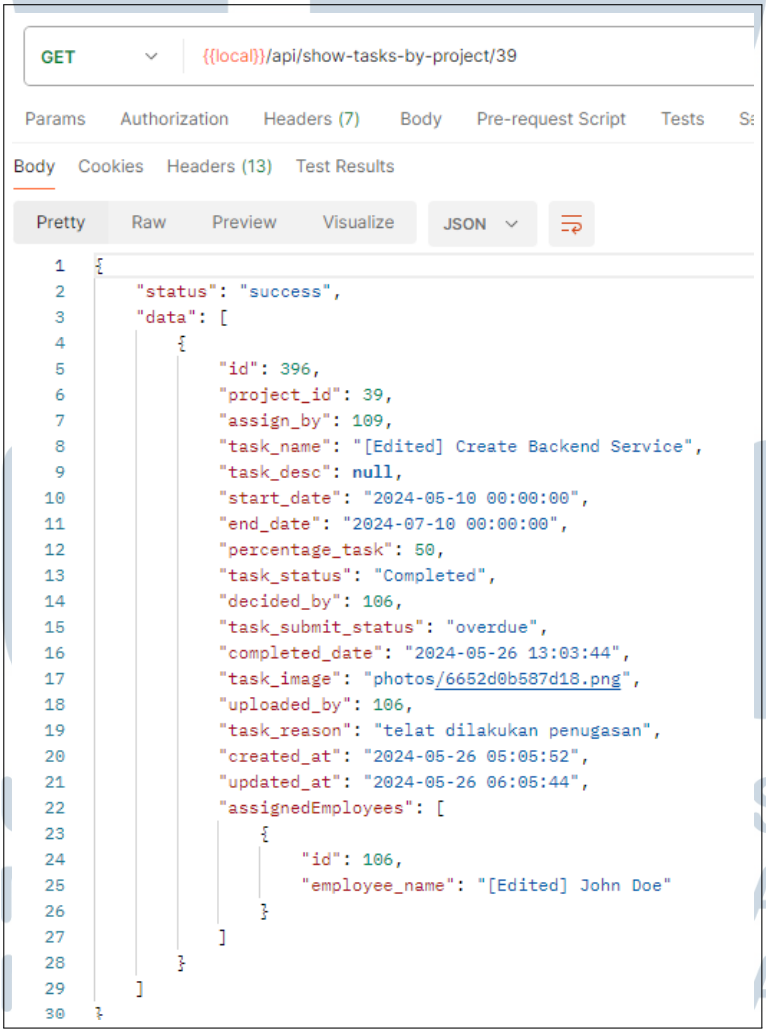
API ini digunakan ketika ingin mendapatkan tugas pada salah satu karyawan. *API* ini dapat membantu pada sisi *frontend* ketika suatu karyawan ingin melihat tugas yang dimilikinya serta membantu *admin* dalam melakukan monitor terhadap tugas yang dilibatkan pada salah satu karyawan. *API* ini mendapatkan data tugas per karyawan dengan memanfaatkan *id* karyawan yang didapatkan pada parameter untuk dimasukkan pada saat*query* memanggil data tugas, menyebabkan data yang diterima secara langsung terfiltrasi. Ketika data tugas telah ditemukan, maka *API* akan mengambil data tugas-tugas tersebut dan mengirimkan respons berhasil beserta data tugas karyawan tersebut. Hasil respons dapat dilihat pada Gambar 3.42.



Gambar 3.42. Tampilan percobaan *API get task by employee*

B.3.4 Mengambil semua data task per Proyek

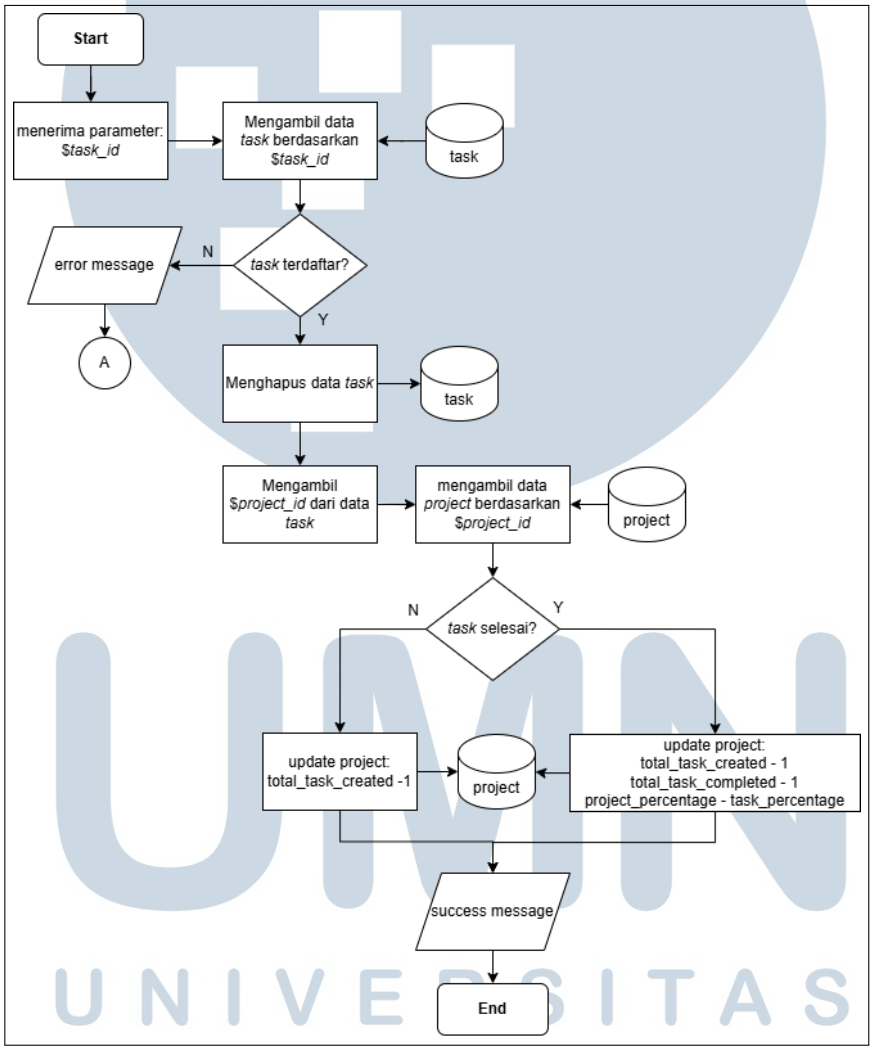
API ini digunakan ketika ingin mendapatkan tugas yang ada pada salah satu proyek. *API* ini dapat membantu pada sisi *frontend* saat ingin melihat detail pada salah satu proyek dan membantu *admin* dalam melakukan monitor terhadap tugas-tugas yang ada pada setiap proyek. *API* ini mendapatkan data tugas per proyek dengan memanfaatkan *id* proyek yang didapatkan pada parameter untuk dimasukkan pada saat*query* memanggil data tugas, menyebabkan data yang diterima secara langsung terfiltrasi. Ketika data tugas telah ditemukan, maka *API* akan mengambil data tugas-tugas tersebut dan mengirimkan respons berhasil beserta data tugas karyawan tersebut. Hasil respons dapat dilihat pada Gambar 3.43.



Gambar 3.43. Tampilan percobaan *API get task by project*

B.4 API Delete Task

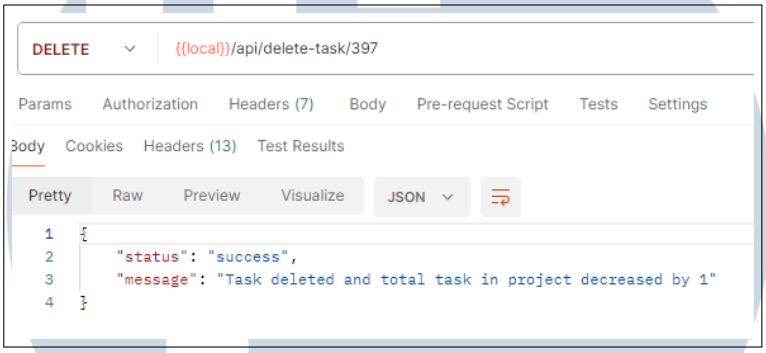
API delete task dibuat untuk menangani permintaan penghapusan suatu tugas. Untuk mengetahui tugas manakah yang ingin dihapus, *API* meminta *input* parameter berupa *id* tugas yang ingin dihapus. *id* tugas tersebut akan dimasukkan pada saat *query* dijalankan untuk mencari data tugas yang dihapus. Untuk alur proses *API* melakukan penghapusan data tugas dapat dilihat pada Gambar 3.44.



Gambar 3.44. *Flowchart API delete task*

Pada Gambar3.44, ketika melakukan penghapusan data tugas, dilakukan pengecekan status tugas untuk menghindari penghapusan data tugas yang sudah selesai dikerjakan. Jika tugas belum selesai maka *API* akan mengurangi jumlah tugas yang terbuat (*total_task_created*) pada proyek yang terkait dengan tugas tersebut kemudian melakukan penghapusan data tugas. Ketika tugas sudah selesai,

maka *API* juga akan mengurangi persentase proyek dengan persentase tugas yang dimiliki tugas yang dihapus beserta mengurangi jumlah tugas yang telah selesai (*total_task_completed*). Jika data tugas berhasil dihapus, *API* kemudian mengirimkan respons bahwa data telah berhasil dihapus. Hasil respons dapat dilihat pada Gambar 3.45.



Gambar 3.45. Tampilan percobaan *API delete task*

B.5 Sistem Invitasi Task

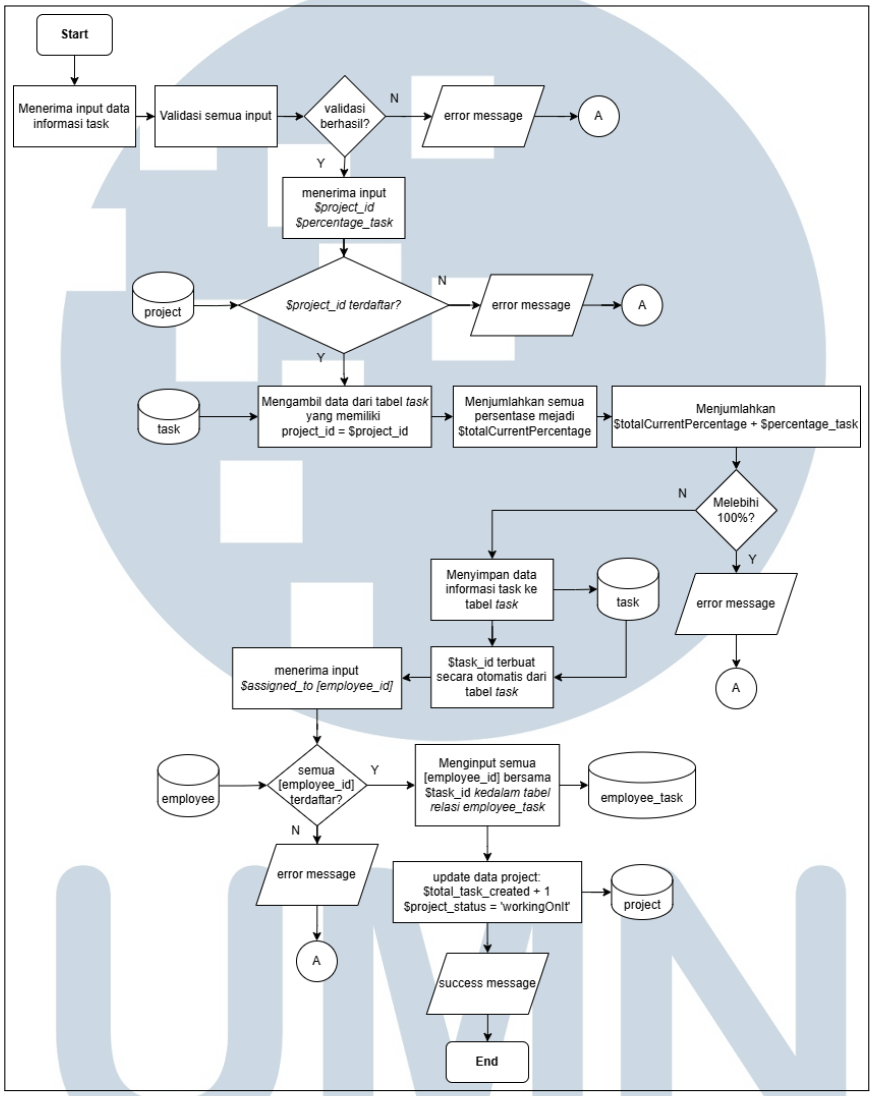
Ketika *Project Manager* membuat tugas dan melakukan *input assigned_to* yang merupakan kumpulan karyawan terlibat. Maka setiap dari karyawan tersebut akan mendapatkan invitasi dari tugas tersebut. Jika pada data terdapat lebih dari satu karyawan yang terlibat, maka hanya salah satu dari mereka dapat menerima atau menolak invitasi tugas tersebut, sisa dari karyawan yang terlibat akan diwakilkan. Karyawan yang menerima/menolak tugas akan dicatat pada data *decided_by* yang mengartikan siapa yang menangani invitasi tersebut.

Terdapat dua *API* untuk menangani sistem ini, yaitu *API accept task* untuk menangani pilihan menerima tugas dan *API reject task* untuk menangani pilihan menolak tugas. Kedua *API* memiliki struktur yang sama dalam menangani permintaan *API*.

B.5.1 API Accept Task

API ini digunakan jika suatu karyawan yang dilibatkan pada suatu tugas ingin menerima invitasi tugas tersebut. Pada Gambar 3.46 melakukan visualisasi pada alur proses *API* untuk melakukan penerimaan invitasi tugas. *API* akan menerima *parameter task_id* dan *input* berupa *id* dari karyawan yang melakukan permintaan *API*. Kemudian *API* akan mengakses tabel yang terkait

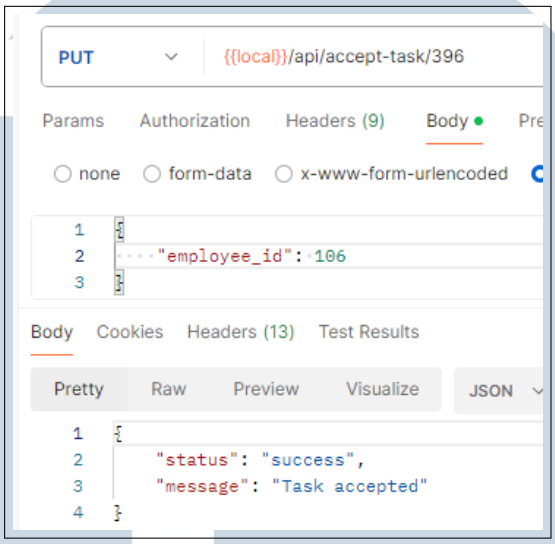
untuk melakukan pengecekan jika data tugas dan data karyawan telah terdaftar atau belum.



Gambar 3.46. *Flowchart API accept task*

Jika kedua data sudah terdaftar sebelumnya, maka *API* melakukan validasi dari status tugas, jika status tugas pada saat itu tidak sama dengan '*onPending*' maka tugas tersebut dinyatakan sedang dikerjakan dan invitasi telah diproses sebelumnya. Jika status masih '*onPending*', maka *API* dapat melakukan proses pembaharuan data pada data tabel tugas dan tabel relasi *employee_task*. Data yang diperbaharui adalah berupa status tugas yang menjadi '*workingOnIt*' dan mengubah data '*isAccepted*' yang terdapat pada semua data karyawan yang terlibat pada tugas di dalam tabel relasi *employee_task* menjadi *true* yang menyatakan bahwa karyawan telah menerima tugas terkait.

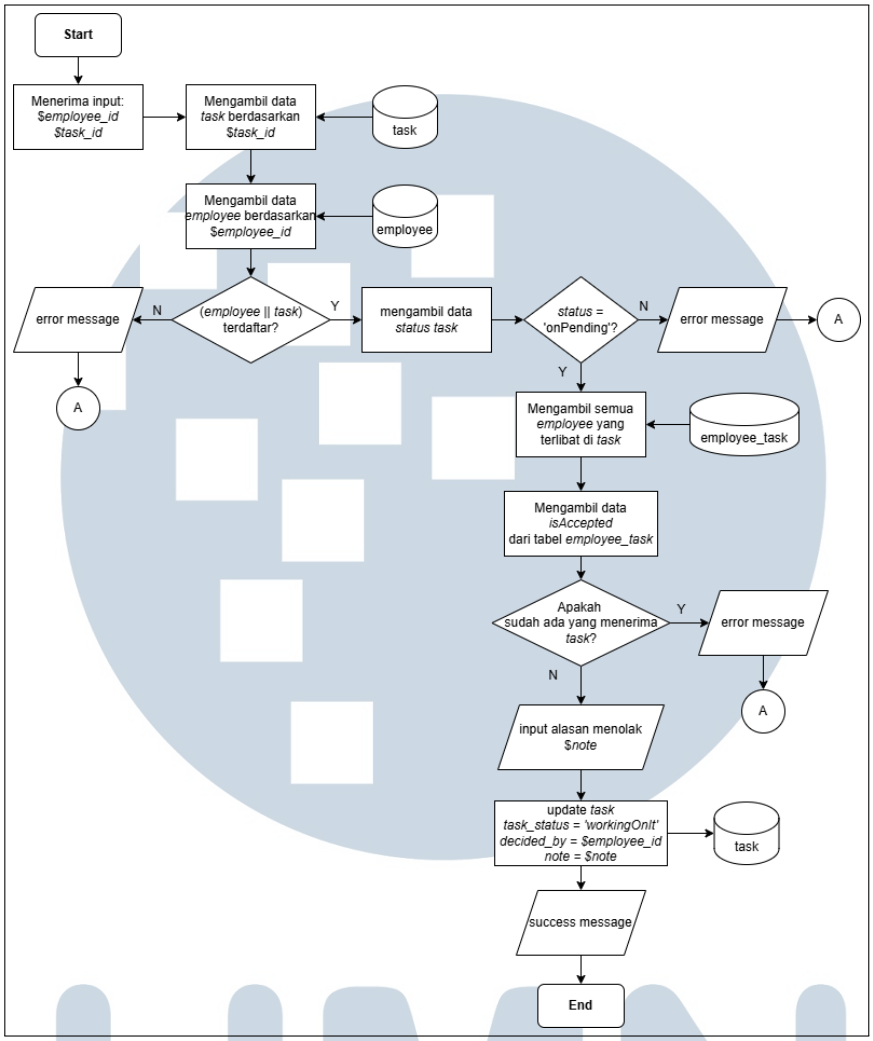
Jika proses validasi berhasil dan data telah diperbaharui, maka *API* akan mengirimkan kembali respons yang memberitahu bahwa tugas telah berhasil diterima. Hasil respons dari *API* dapat dilihat pada Gambar 3.47.



Gambar 3.47. Tampilan percobaan *API accept task*

B.5.2 API Reject Task

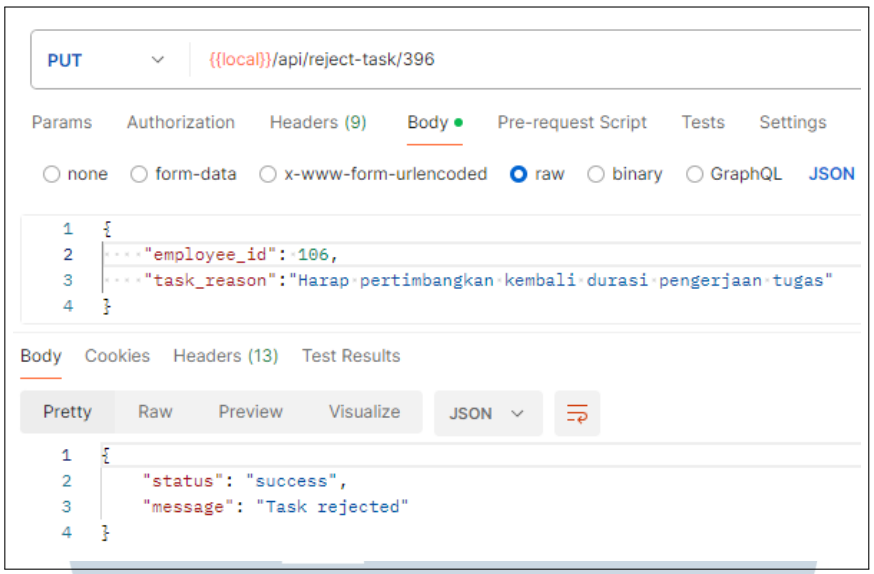
API ini digunakan jika suatu karyawan yang dilibatkan pada suatu tugas ingin menolak invitasi tugas tersebut. Pada Gambar 3.48 melakukan visualisasi pada alur proses *API* untuk melakukan penolakan invitasi tugas. *API* akan menerima *parameter task_id* dan *input* berupa *id* dari karyawan yang melakukan permintaan *API*. Kemudian *API* akan mengakses tabel yang terkait untuk melakukan pengecekan jika data tugas dan data karyawan telah terdaftar atau belum.



Gambar 3.48. *Flowchart API reject task*

Jika kedua data sudah terdaftar sebelumnya, maka *API* melakukan validasi dari status tugas, jika status tugas pada saat itu tidak sama dengan '*onPending*' maka tugas tersebut dinyatakan sedang dikerjakan dan invitasi telah diproses sebelumnya. Jika status masih '*onPending*', maka *API* dapat melakukan proses pembaharuan data pada data tabel tugas dan tabel relasi *employee_task*. Data yang diperbaharui adalah berupa status tugas yang menjadi '*onReview*' untuk dilakukan proses *review* kembali oleh *Project Manager* dari proyek terkait.

Jika proses validasi berhasil dan data telah diperbaharui, maka *API* akan mengirimkan kembali respons yang memberitahu bahwa tugas telah berhasil ditolak. Hasil respons dari *API* dapat dilihat pada Gambar 3.49.



Gambar 3.49. Tampilan percobaan *API reject task*

B.6 API Submit Task

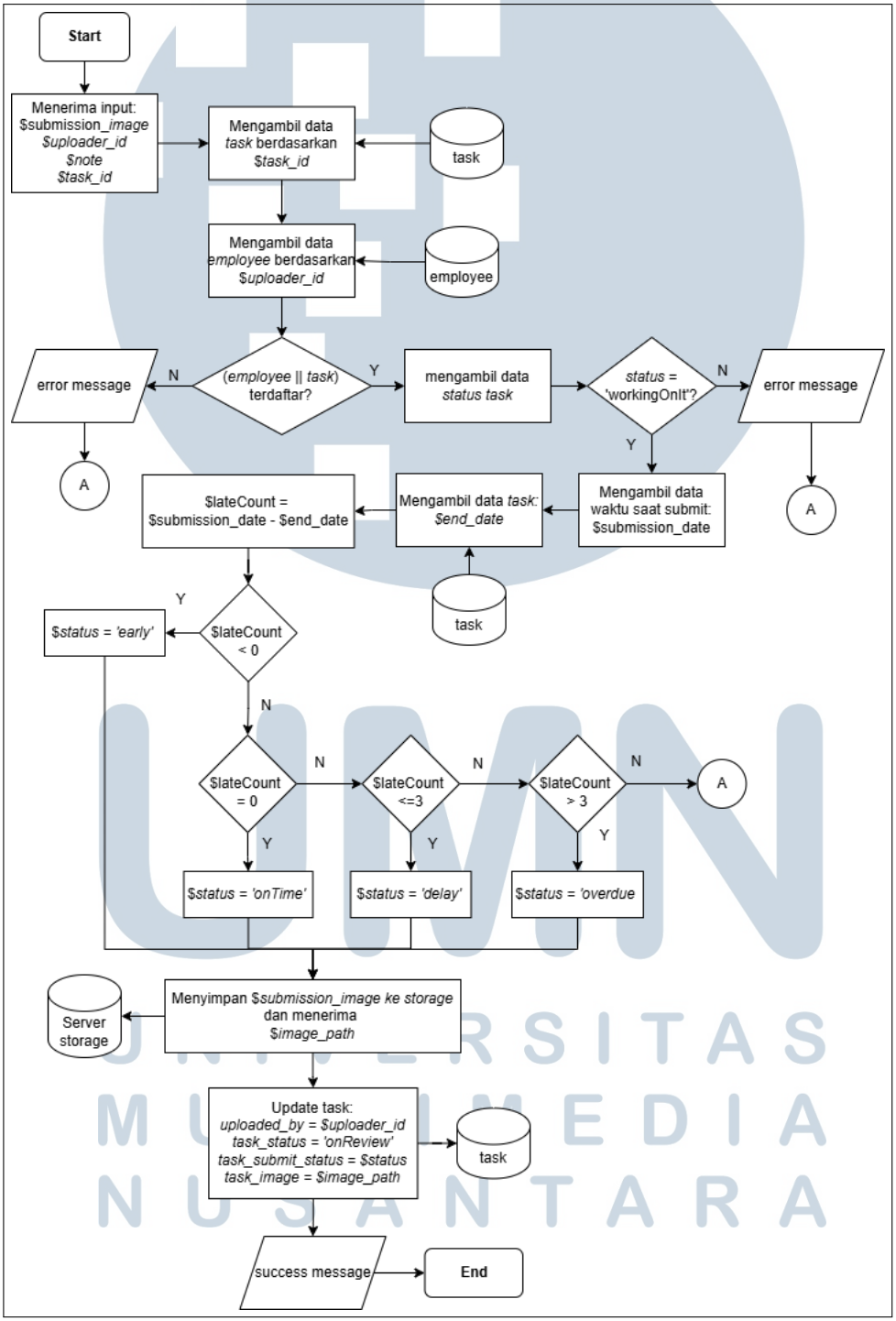
Setelah karyawan menerima invitasi tugas, maka fitur untuk melakukan pengumpulan tugas dapat diakses oleh karyawan. Proses pengumpulan tugas pada *API submit task* dapat diproses jika status tugas telah menjadi '*workingOnIt*'. Pada Gambar 3.50 berupa visualisasi alur proses *API* menangani permintaan pengumpulan tugas.

Input yang diterima *API* adalah berupa gambar bukti penyelesaian tugas (*submission_image*), karyawan yang melakukan pengumpulan (*uploader_id*), tugas yang ingin dikumpulkan (*task_id*), dan juga *note* sebagai deskripsi dari penyelesaian tugas. Kemudian dengan *id* karyawan dan *id* tugas, dilakukan pengecekan data pada setiap tabel yang terkait untuk memastikan jika data telah terdaftar. Kemudian *API* akan melakukan pengecekan status pada status tugas, karena *API* dapat melanjutkan proses jika status tugas saat pengumpulan adalah 'workingOnIt' untuk memastikan tidak ada *bug* yang terjadi.

Kemudian *API* melakukan selisih pada tanggal saat pengumpulan dengan tenggat waktu dari pengumpulan tugas (*end_date*). Selisih tersebut akan menghasilkan nilai angka yang akan dijadikan sebagai penentu dari seberapa telat pengumpulan dilakukan. Terlampirkan pada gambar 3.50 bagaimana *API* menggunakan *lateCount* untuk mendapatkan status ketepatan waktu pengumpulan tugas dengan memanfaatkan kondisi pada setiap nilai yang mungkin didapatkan.

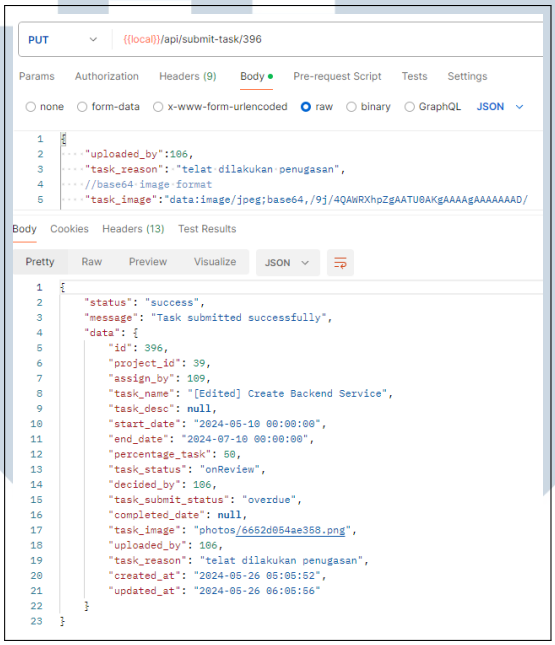
Setelah mendapatkan status ketepatan waktu pengumpulan tugas, Maka

API akan memproses gambar bukti penyelesaian tugas yang berbentuk *format base64* dengan menyimpan gambar secara lokal pada penyimpanan *server*. Dari penyimpanan secara lokal tersebut, diterima sebuah direktori letak gambar tersebut berada.



Gambar 3.50. *Flowchart API submit task*

Setelah semua data diterima, maka *API* akan melakukan pembaharuan data pada data tugas yang dikumpulkan dan memberikan respons yang memberitahukan bahwa proses pengumpulan tugas telah berhasil dan siap untuk dilakukan proses *review* oleh *Project Manager*. Hasil respons *API submit task* dapat dilihat pada Gambar 3.51.



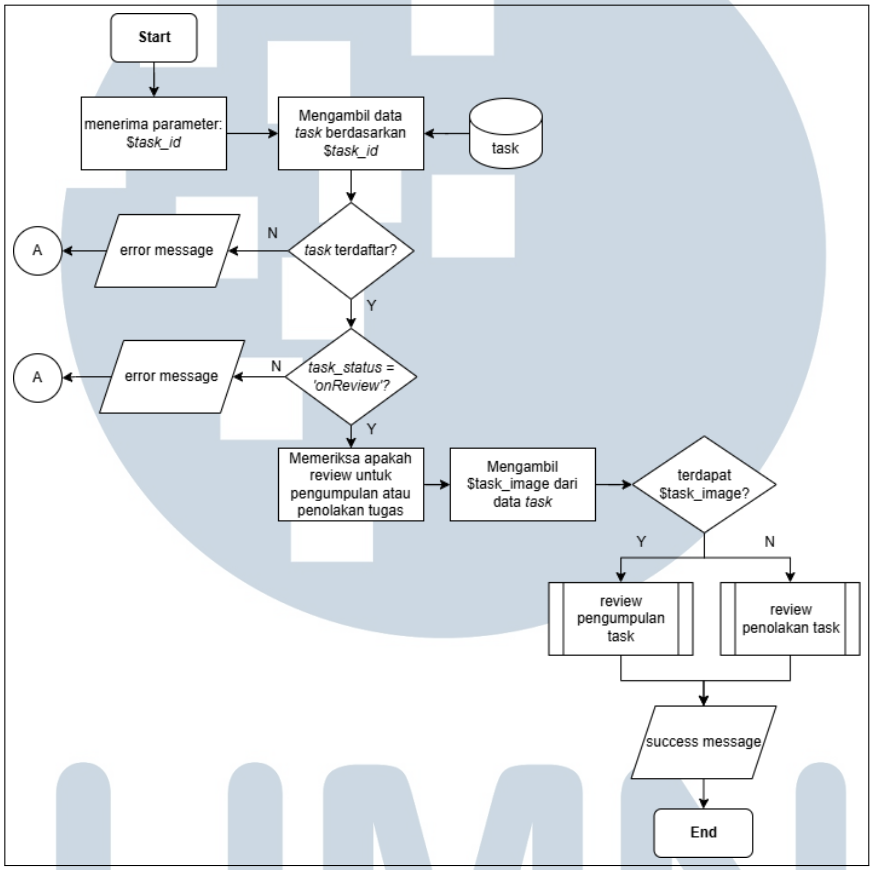
Gambar 3.51. Tampilan percobaan *API submit task*

B.7 API Review Task

API review task digunakan untuk melakukan *review* terhadap tugas yang memiliki status '*onReview*'. *API* ini hanya dapat digunakan oleh *Project Manager* yang terkait dengan proyek. *API* ini dapat menangani dua kasus dari proses *review* terhadap tugas. Kedua kasus tersebut adalah kasus pada saat karyawan yang terlibat dengan tugas melakukan pengumpulan tugas beserta bukti penyelesaian pada tugas tersebut, dan kasus ketika suatu karyawan menolak invitasi untuk terlibat pada tugas untuk dikerjakan.

Cara *API* membedakan kedua kasus ini adalah dengan melakukan pengecekan atribut *task_image*. Jika atribut *task_image* bernilai *null* atau kosong, maka tugas tersebut dikumpulkan untuk dilakukan *review* atas penolakan tugas dari karyawan terkait. Sedangkan ketika atribut *task_image* bernilai selain *null*, maka tugas tersebut dikumpulkan oleh karyawan untuk dilakukan proses penilaian dari tugas. Hal ini dapat dibedakan karena saat melakukan penolakan, data tugas tidak

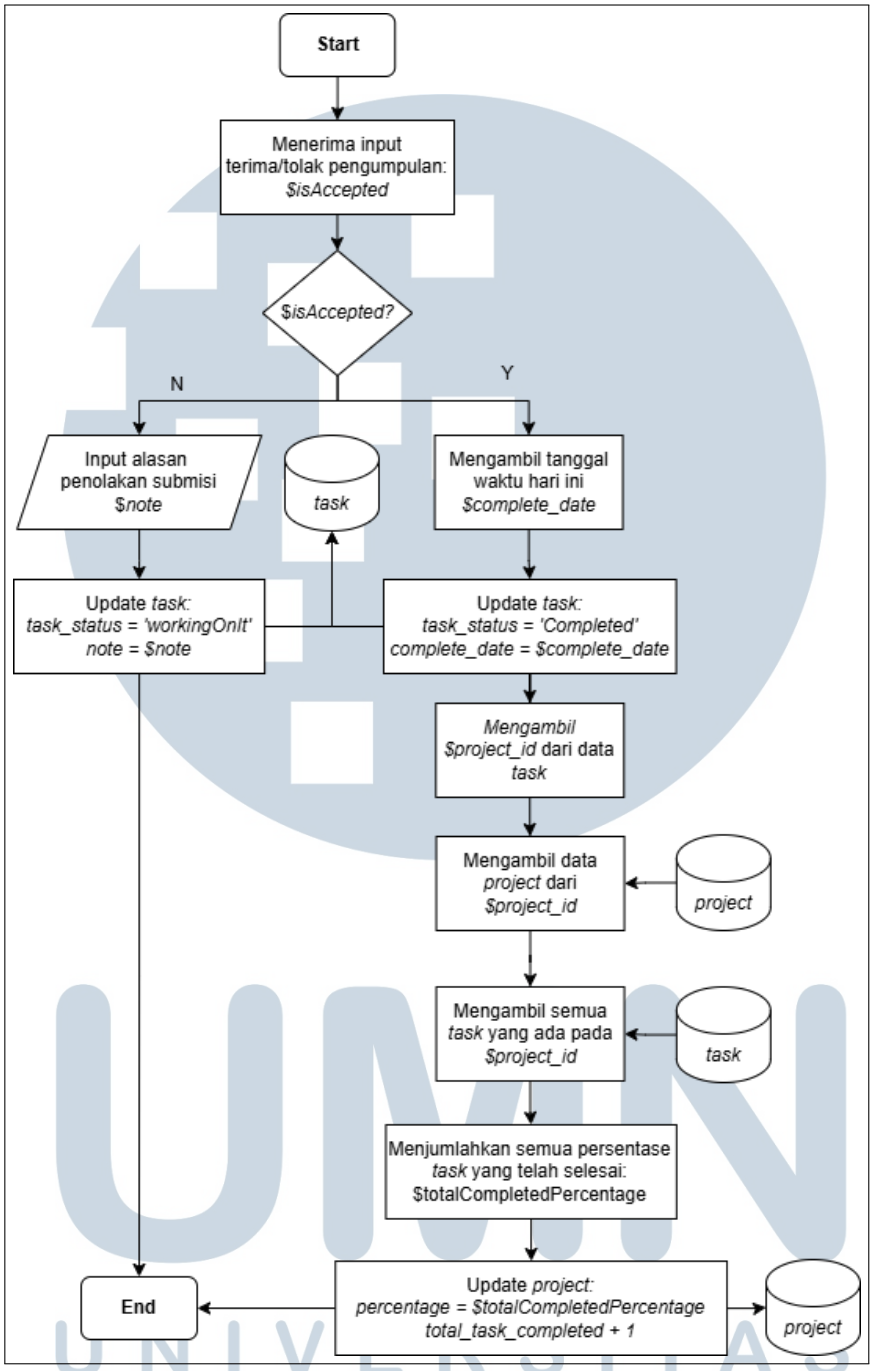
akan memiliki data *task_image*. Atribut tersebut dapat bernilai ketika tugas telah melewati proses pengumpulan tugas yang mengharuskan melakukan *input* pada atribut *task_image*. Proses *API review task* membedakan kedua kasus tersebut dapat dilihat pada Gambar 3.52.



Gambar 3.52. *Flowchart* cara *API review task* membedakan penilaian untuk pengumpulan dan penolakan tugas

B.7.1 Review task yang telah dikumpulkan

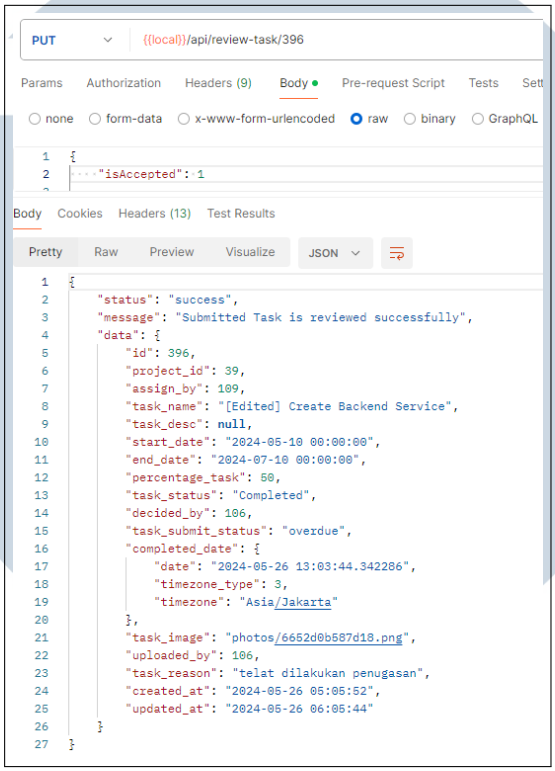
Pada pemanggilan *API review task* untuk melakukan penilaian pada tugas yang dikumpulkan, diperlukan *input* berupa '*isAccepted*' yang digunakan sebagai penanda apakah tugas yang dikumpul tersebut diterima untuk diselesaikan secara penuh atau ditolak untuk dilakukan revisi/pengerjaan kembali kepada karyawan. Alur dari proses penilaian tugas yang telah dikumpulkan dapat dilihat pada Gambar 3.53.



Gambar 3.53. *Flowchart API review task* yang telah dikumpulkan

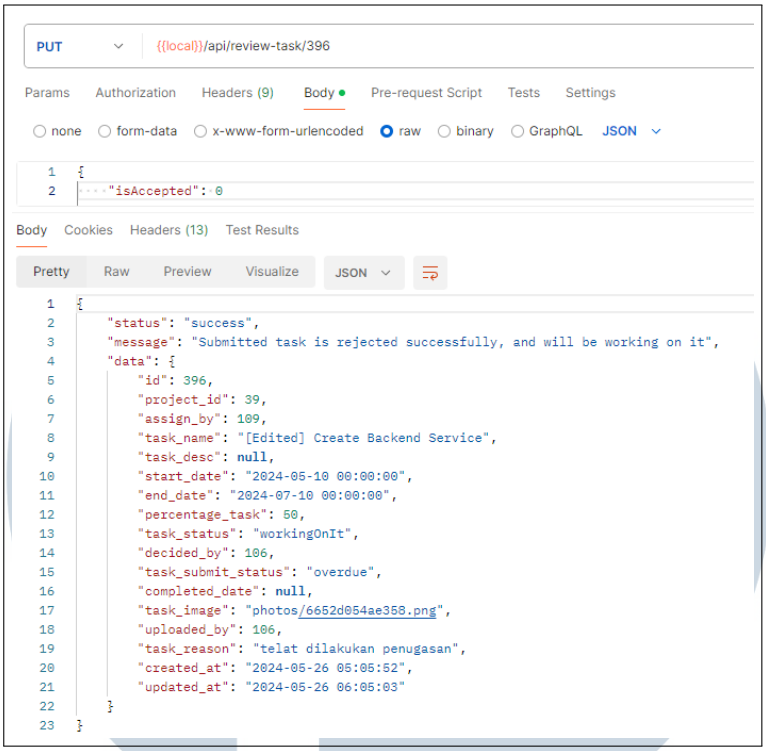
Jika nilai dari '*isAccepted*' adalah 1, maka *Project Manager* bertujuan untuk menerima tugas yang dikumpulkan. Maka *API* akan melakukan pembaharuan data pada tugas beserta proyek terkait. Data yang diperbaharui pada data tugas adalah seperti tanggal tugas selesai (*completed_date*) menjadi hari pada saat *Project Manager* menerima tugas yang dikumpul dan status tugas menjadi '*Completed*'. Untuk pembaharuan pada data proyek adalah data jumlah tugas yang selesai

ditambah satu (*total_task_completed*) dan menambahkan persentase proyek dengan persentase tugas. Hasil respons dari penerimaan tugas yang dikumpulkan dapat dilihat pada Gambar 3.54.



Gambar 3.54. Tampilan percobaan *API review task* jika menerima pengumpulan tugas

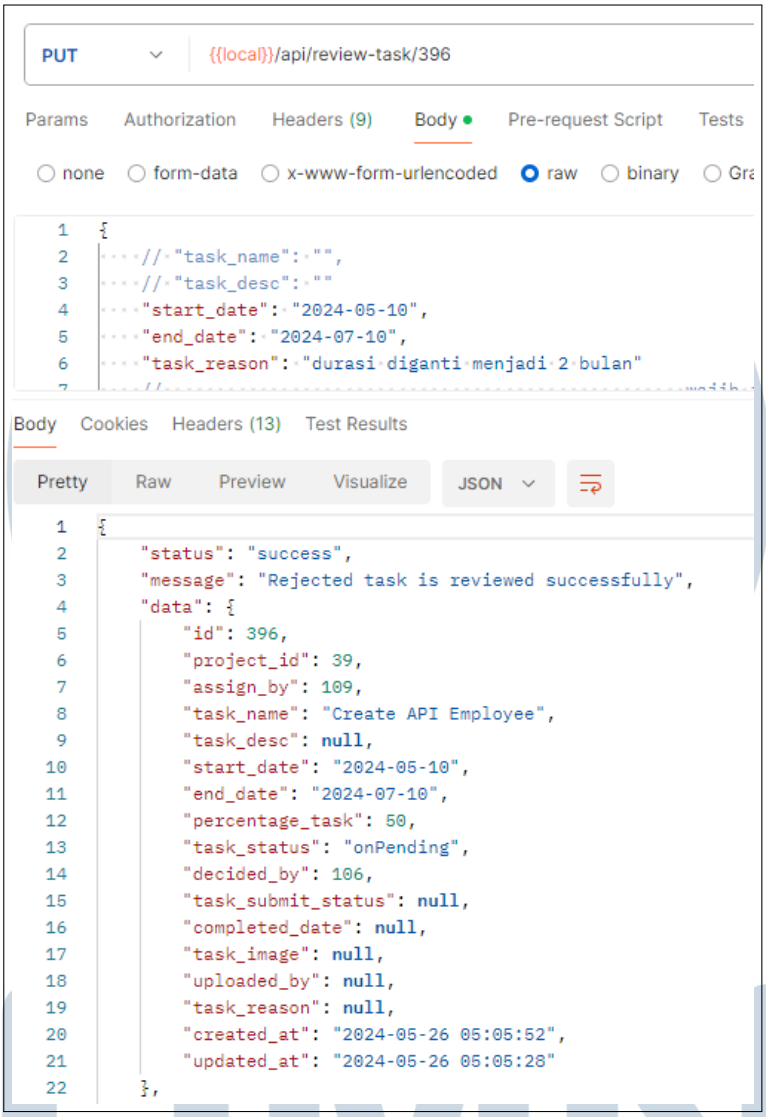
Jika nilai dari '*isAccepted*' adalah 0, maka *Project Manager* akan diminta untuk melakukan *input* pada atribut *note* yang akan dijadikan lampiran dari alasan kenapa dilakukan penolakan tugas yang dikumpulkan oleh karyawan terkait. Sehingga mempermudah komunikasi antara *Project Manager* dan karyawan dalam melakukan revisi pekerjaan. Kemudian status tugas akan dikembalikan menjadi '*workingOnIt*' agak karyawan dapat mengerjakan tugas dan melakukan pengumpulan kembali. Hasil respons dari penolakan tugas yang dikumpulkan dapat dilihat pada Gambar 3.55.



Gambar 3.55. Tampilan percobaan *API review task* jika menolak pengumpulan tugas

B.7.2 Review task yang ditolak saat invitasi

Pada pemanggilan *API review task* untuk melakukan revisi informasi tugas setelah ditolaknya invitasi tugas dari karyawan, *Project Manager* dapat melakukan perubahan informasi ataupun memberikan kembali tugas kepada karyawan untuk diterima. Proses perubahan atau pembaharuan data pada informasi tugas memiliki struktur yang sama seperti pemanggilan *API update task*, akan tetapi pada *API* ini memungkinkan untuk mengubah status dari tugas secara otomatis setelah dilakukan revisi menjadi '*onPending*' agar dapat dilakukan penerimaan maupun penolakan oleh karyawan yang dilibatkan. Hasil dari respons *API review task* saat melakukan revisi informasi tugas dapat dilihat pada Gambar 3.55.

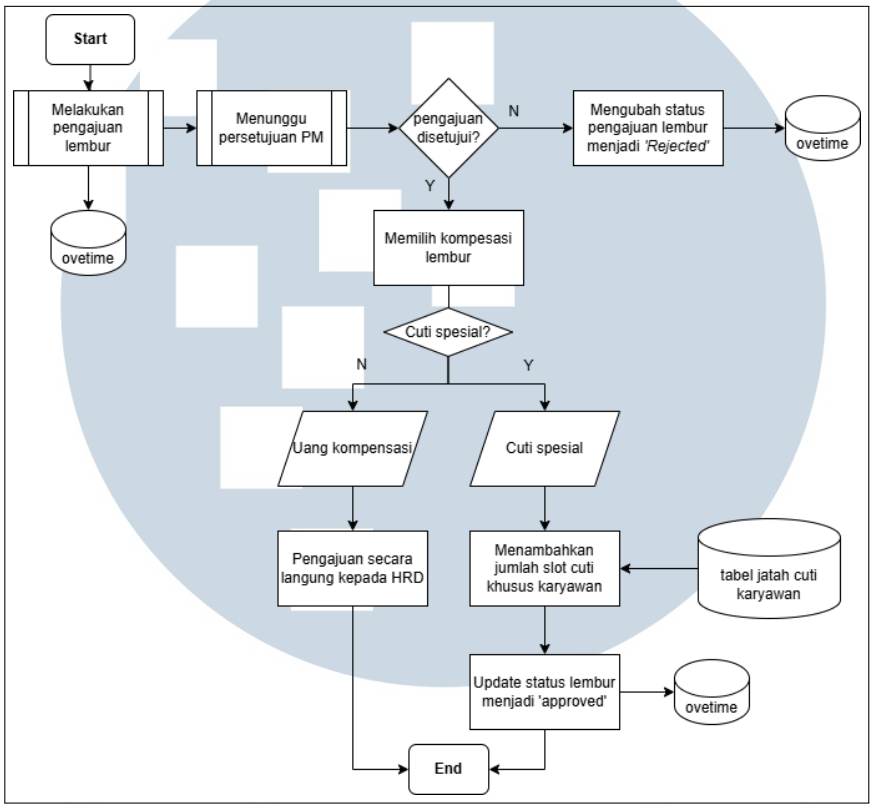


Gambar 3.56. Tampilan percobaan *API review task* jika ditolak oleh karyawan terkait

3.3.6 Pembuatan API Pengajuan lembur

API pengajuan lembur digunakan oleh mempermudah karyawan mengajukan permintaan jam lembur kepada *admin* dan juga membantu *admin* untuk mencatat dan memonitor permintaan lembur oleh karyawan. Jika pengajuan lembur karyawan disetujui oleh *admin*, maka karyawan berhak untuk memilih kompensasi seperti cuti khusus atau pun penggantian uang. Jatah cuti khusus dapat digunakan selama dua minggu, jika tidak digunakan maka cuti khusus tidak dapat digunakan lagi. Jatah cuti khusus kemudian akan dapat digunakan sebagai tiket untuk melakukan pengajuan cuti khusus pada fitur pengajuan cuti. Sedangkan untuk kompensasi dalam bentuk uang akan dilakukan di luar sistem aplikasi.

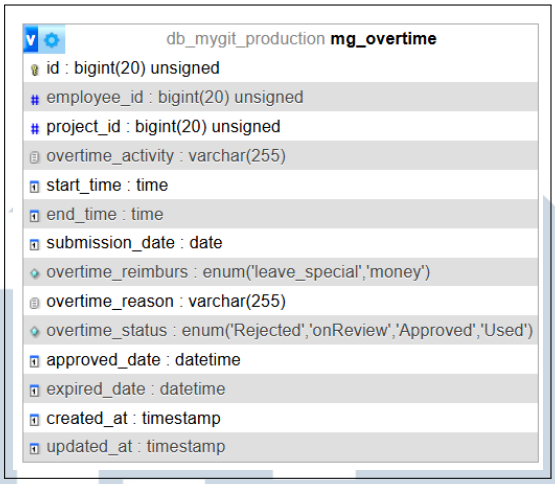
Sehingga *API* pengajuan lembur hanya mencakup fitur seperti pengajuan lembur dan *review* pengajuan lembur karyawan oleh *admin*. Alur pengajuan dari lembur dapat dilihat pada Gambar 3.57



Gambar 3.57. *Flowchart* untuk alur pengajuan lembur karyawan

Pada Gambar 3.58 merupakan daftar dari data yang disimpan pada tabel *overtime* (lembur). Data yang disimpan adalah seperti proyek yang dikerjaan saat lembur (*project_id*), aktivitas yang dilakukan saat lembur (*overtime_activity*), jam mulai dan akhir melakukan lembur (*start_time* dan *end_time*), tanggal melakukan pengajuan lembur (*submission_date*), dan pilihan kompensasi (*overtime_reimburse*). Data seperti status pengajuan lembur (*overtime_status*), tanggal pengajuan disetujui (*approved_date*), serta tanggal jatah cuti kadaluwarsa (*expired_date*) akan diperbaharui saat telah melakukan pengajuan. Terdapat data yang menyimpan alasan (*overtime_reason*) ketika *admin* melakukan penolakan pada pengajuan lembur karyawan.

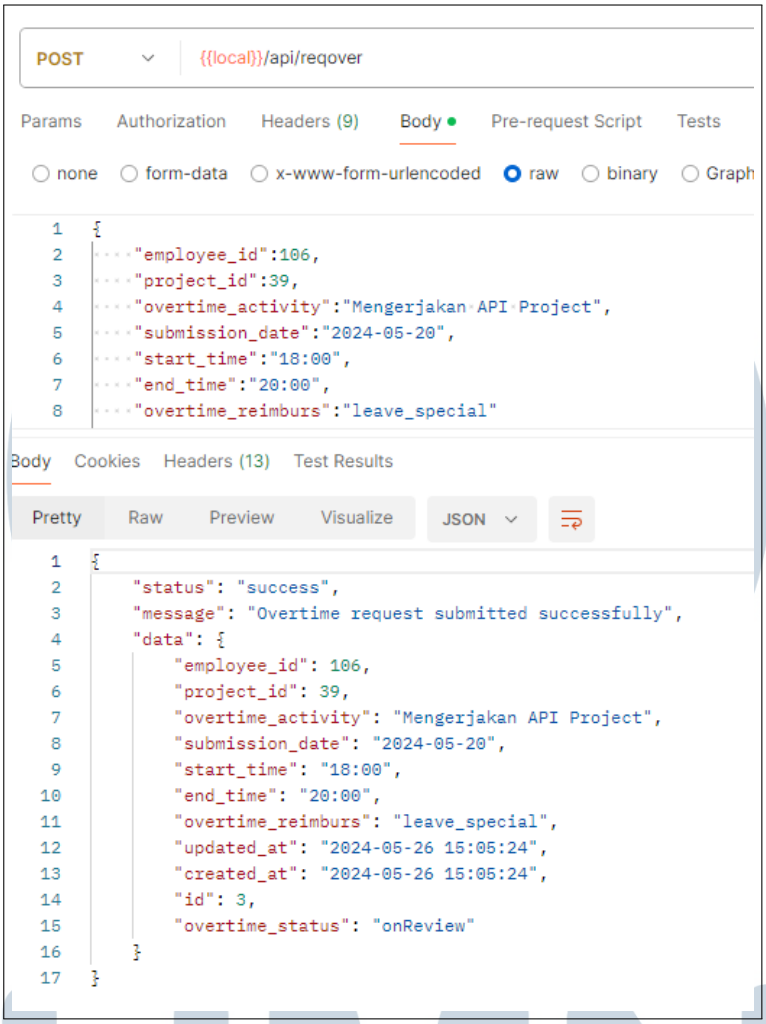
Status lembur akan menjadi '*onPending*' ketika baru mengajukan lembur. Jika pengajuan ditolak maka status akan berubah menjadi '*Rejected*' dan menjadi '*Approved*' ketika disetujui. Kemudian, jika cuti khusus yang didapatkan dari lembur telah digunakan maka status akan berubah menjadi '*Used*'.



Gambar 3.58. Atribut basis data dari tabel *overtime*

A. API Create Overtime Request

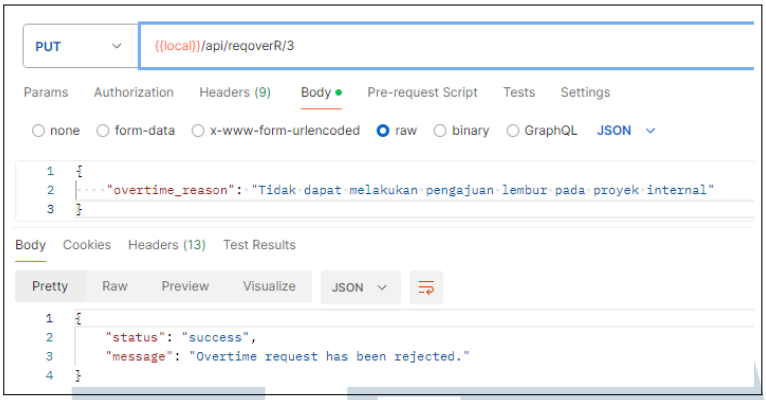
API ini digunakan untuk melakukan pengajuan lembur yang kemudian akan ditinjau oleh *admin*. Untuk menggunakan *API* ini hanya perlu mengisi *input* yang sesuai dengan daftar atribut yang ada pada tabel *overtime*. Kemudian ajuan akan masuk kepada tampilan *admin* untuk dilakukan penolakan atau pun persetujuan. Jika data pengajuan lembur berhasil diproses, maka tampilan dari respons *API* adalah seperti pada Gambar 3.59.



Gambar 3.59. Tampilan percobaan *API create overtime request*

B. API Reject Overtime Request

Pada tampilan *admin* akan muncul semua data pengajuan lembur dari karyawan. *API* ini digunakan jika *admin* tidak ingin menyetujui pengajuan lembur dari salah satu karyawan. *API* akan meminta *input* berupa alasan dari penolakan pengajuan lebur, kemudian *API* akan mengubah status dari pengajuan lembur menjadi '*Rejected*'. Jika data pengajuan lembur berhasil diproses maka respons dari *API* akan terlihat seperti pada Gambar 3.60.



Gambar 3.60. Tampilan percobaan *API reject overtime request*

C. API Approve Overtime Request

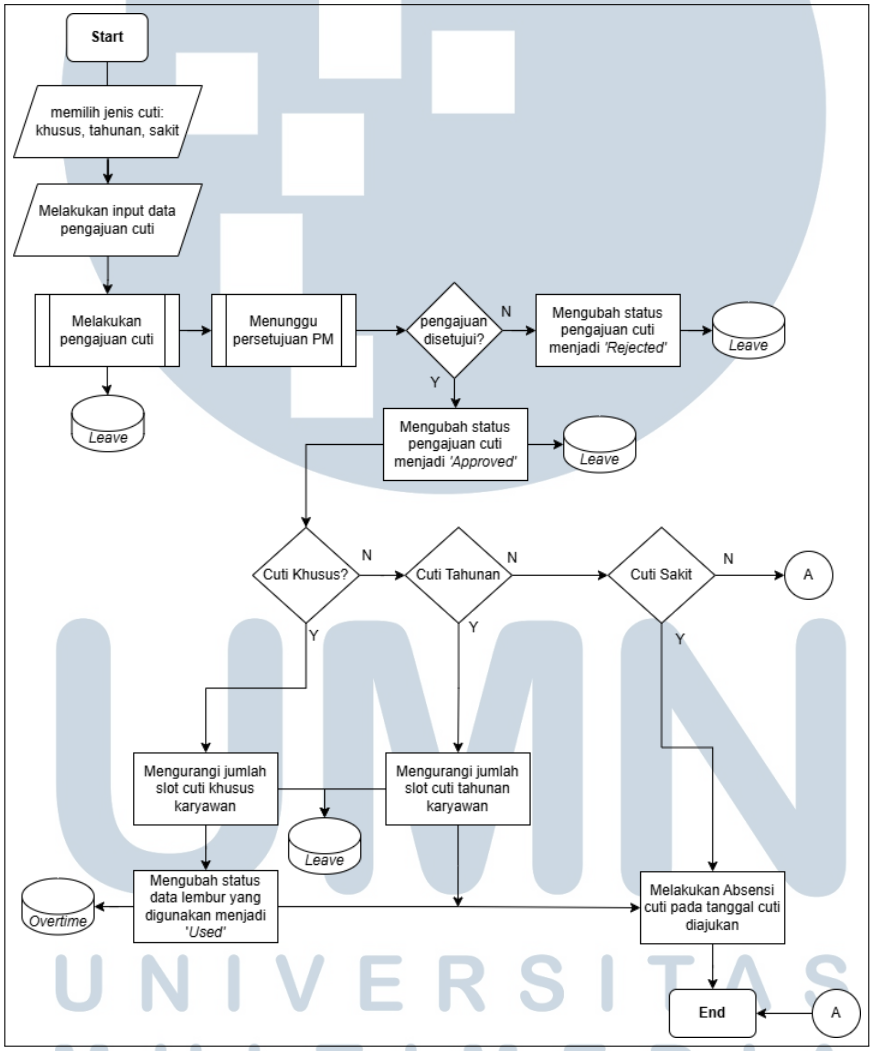
Pada tampilan *admin* akan muncul semua data pengajuan lembur dari karyawan. *API* ini digunakan jika *admin* ingin menyetujui pengajuan lembur dari salah satu karyawan. *API* akan mengubah status dari pengajuan lembur menjadi '*Approved*'. Kemudian akan menambahkan satu jatah cuti khusus pada tabel yang menyimpan data jatah cuti dari semua karyawan. Setelah itu *API* akan mencatat tanggal kapan pengajuan diterima dan menambahkan tanggal jatah cuti kadaluwarsa yaitu dua minggu. Jika data pengajuan lembur berhasil diproses maka respons dari *API* akan terlihat seperti pada Gambar 3.61.

Gambar 3.61. Tampilan percobaan *API approve overtime request*

3.3.7 Pembuatan API Pengajuan cuti

API pengajuan cuti digunakan untuk mempermudah karyawan mengajukan permintaan cuti kepada admin dan juga membantu admin untuk mencatat dan memonitor permintaan cuti oleh karyawan. Terdapat tiga jenis cuti yang

dicatat, yaitu cuti khusus, cuti tahunan, dan cuti sakit. Jika pengajuan cuti karyawan disetujui oleh *admin*, *API* akan mengubah status pengajuan cuti menjadi '*Approved*', kemudian karyawan telah mendapatkan cuti pada hari yang diajukan dan kehadiran pada tanggal karyawan cuti akan diperbaharui sebagai status sedang cuti. *API* pengajuan cuti mencakup fitur seperti pengajuan cuti dan *review* pengajuan cuti karyawan oleh admin. Alur dari pengajuan dari cuti dapat dilihat pada Gambar 3.62.

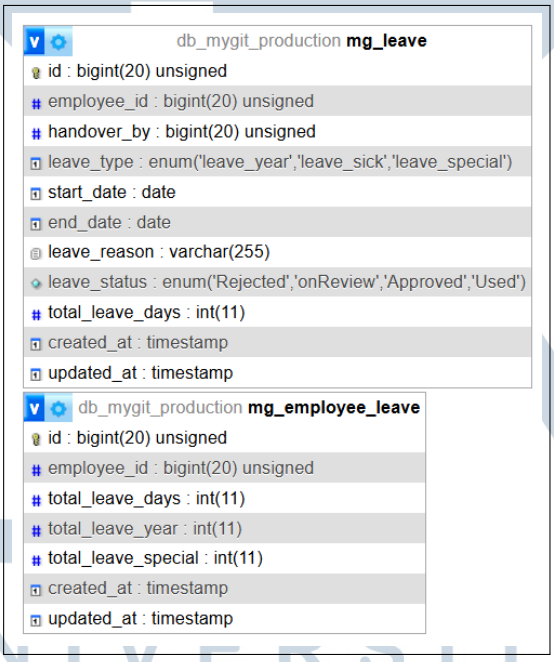


Gambar 3.62. *Flowchart* untuk alur pengajuan cuti karyawan

Pada Gambar 3.63 merupakan daftar dari data yang disimpan pada tabel *leave* (lembur). *API* akan meminta *input* pada data seperti data karyawan yang melakukan pengajuan (*employee_id*), data rekan karyawan yang akan menggantikan tanggung jawab pekerjaan selama karyawan cuti (*handover_by*), pilihan cuti yang diajukan (*leave_type*), dan tanggal awal dan akhir mengajukan cuti (*start_time* dan

diajukan (*leave_type*), dan tanggal awal dan akhir mengajukan cuti (*start_time* dan *end_time*). Data *input* tersebut akan dicatat pada tabel *leave* dan kemudian *API* akan membuat data seperti *leave_status* menjadi '*onReview*' yang akan dilakukan peninjauan oleh *admin*, dan jumlah hari cuti (*total_leave_days*). *API* juga akan menyimpan data dari alasan ditolaknya pengajuan cuti karyawan oleh *admin* (*leave_reason*). Status cuti akan menjadi '*onPending*' ketika baru mengajukan cuti. Jika pengajuan ditolak maka status akan berubah menjadi '*Rejected*' dan menjadi '*Approved*' ketika disetujui.

Pada tabel relasi *employee_leave* menyimpan jumlah dari jatah pada setiap jenis cuti yang dimiliki oleh setiap karyawan. Pada awalnya semua karyawan akan memiliki jumlah cuti tahunan sebanyak yang diatur oleh peraturan perusahaan (pada dasarnya terdapat 12) dan karyawan tidak memiliki jatah cuti khusus sama sekali. Jatah cuti khusus didapatkan ketika karyawan melakukan lembur dan memilih kompensasi dalam bentuk cuti khusus.

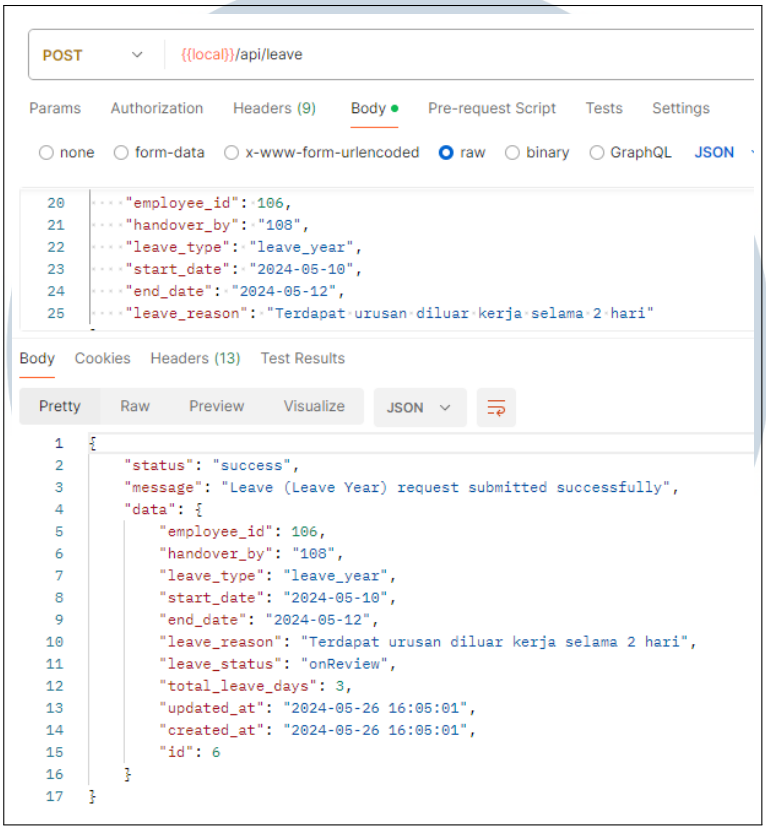


Gambar 3.63. Atribut basis data dari tabel *leave*

A. API Create Leave Request

API ini digunakan untuk melakukan pengajuan cuti yang kemudian akan ditinjau oleh *admin*. Untuk menggunakan *API* ini hanya perlu mengisi *input* yang sesuai dengan daftar atribut yang ada pada tabel *cuti*. Kemudian ajuan akan masuk kepada tampilan *admin* untuk dilakukan penolakan atau pun persetujuan. Jika data

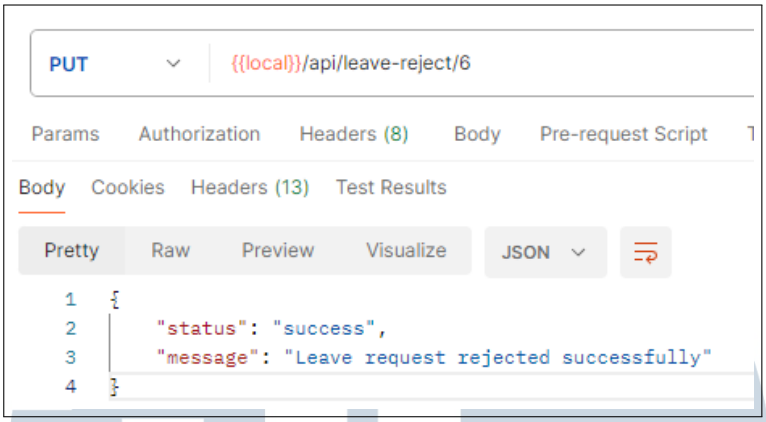
pengajuan cuti berhasil diproses, maka tampilan dari respons *API* adalah seperti pada Gambar 3.64.



Gambar 3.64. Tampilan percobaan *API create leave request*

B. API Reject Leave Request

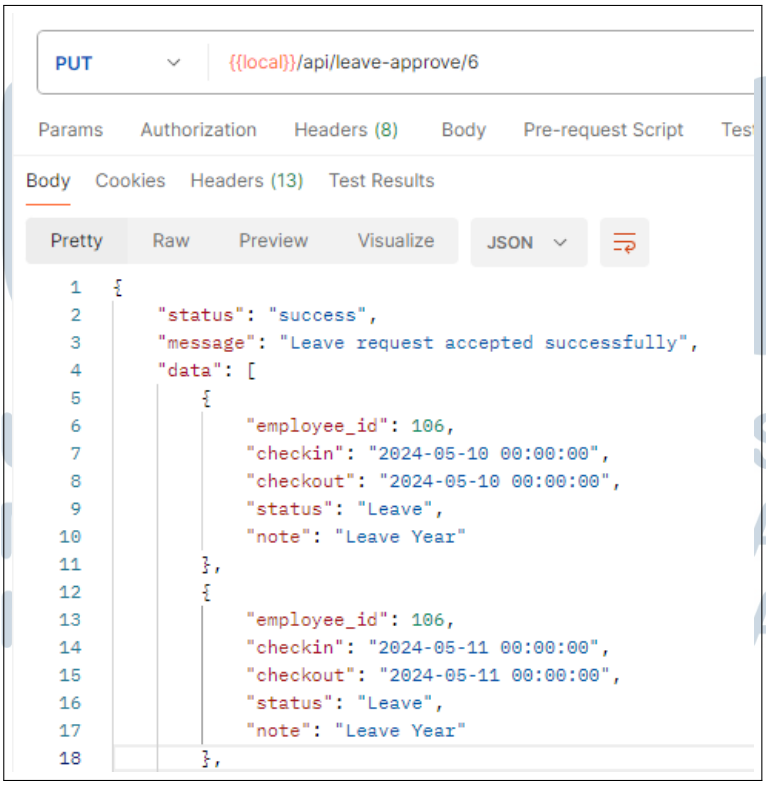
Pada tampilan *admin* akan muncul semua data pengajuan cuti dari karyawan. *API* ini digunakan jika *admin* tidak ingin menyetujui pengajuan cuti dari salah satu karyawan kemudian *API* akan mengubah status dari pengajuan cuti menjadi '*Rejected*'. Jika data pengajuan cuti berhasil diproses maka respons dari *API* akan terlihat seperti pada Gambar 3.65.



Gambar 3.65. Tampilan percobaan *API reject leve request*

C. API Approve Leave Request

Pada tampilan *admin* akan muncul semua data pengajuan cuti dari karyawan. *API* ini digunakan jika *admin* ingin menyetujui pengajuan cuti dari salah satu karyawan. *API* akan mengubah status dari pengajuan cuti menjadi '*Approved*'. Kemudian akan melakukan absensi cuti pada hari yang diajukan untuk cuti oleh karyawan. Jika data pengajuan cuti berhasil diproses maka respons dari *API* akan terlihat seperti pada Gambar 3.66.

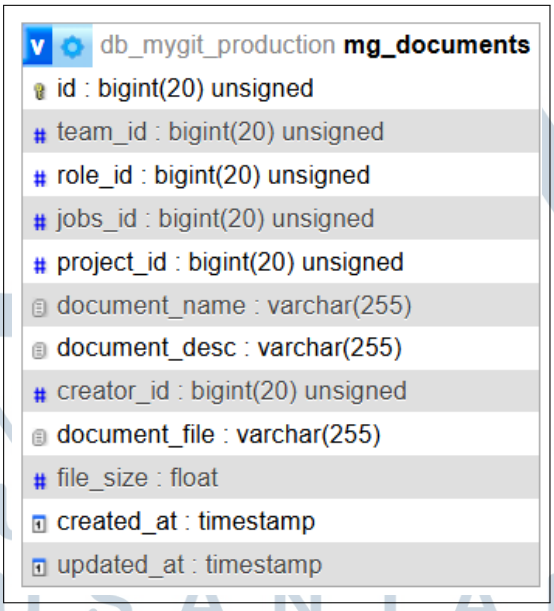


Gambar 3.66. Tampilan percobaan *API approve leave request*

3.3.8 Pembuatan API arsip Dokumen

Setiap proyek yang ditangani GIT memiliki dokumen yang berkaitan dengan proyek tersebut, dokumen-dokumen tersebut dapat berupa dokumen *FSD*, dokumen teknis, dokumen panduan pengguna aplikasi, dan dokumen jenis lainnya. Sehingga *API* ini memungkinkan untuk dapat melakukan *input* dokumen dan menyimpan dokumen secara *online* dan terstruktur pada aplikasi *website* MYGIT.

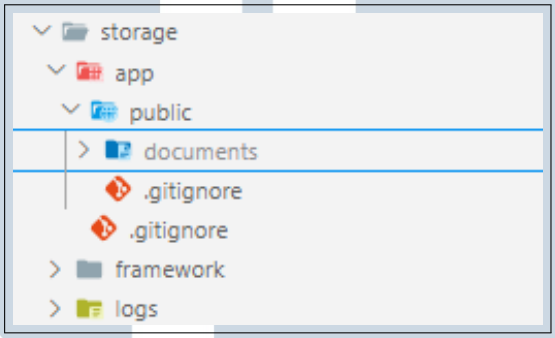
Pada Gambar 3.67 merupakan daftar data yang disimpan pada tabel *document* (dokumen). *API* akan meminta *input* yang disimpan pada tabel *document* seperti nama dokumen (*document_name*), deskripsi dokumen (*document_desc*), karyawan yang melakukan *input* dokumen (*creator_id*), serta dokumen yang ingin disimpan (*document_file*). Data seperti *id role, job, team*, serta *project* disimpan untuk digunakan pada tampilan sisi *frontend* untuk melakukan validasi terhadap siapa saja yang bisa melihat dokumen yang disimpan. Setelah menyimpan data dokumen, *API* akan menghitung jumlah ukuran dokumen dalam *kilobyte* (*file_size*). Setiap proyek dapat memiliki banyak dokumen yang disimpan di dalamnya. Satu dokumen yang disimpan merepresentasikan satu baris pada tabel. Fitur *API* dokumen meliputi fungsi seperti menambahkan dokumen, memperbaharui dokumen, menghapus dokumen, serta mengambil data dokumen.



Gambar 3.67. Atribut basis data dari tabel *document*

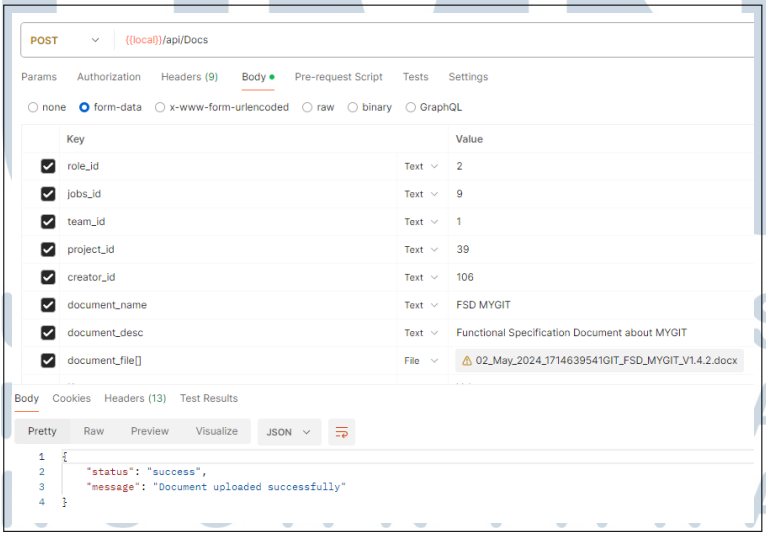
A. API Create Document

API ini digunakan untuk menyimpan data dokumen. *API* menyimpan semua data *input* yang ada pada tabel pada Gambar 3.67. Untuk menangani *input* data berbentuk *file*, *API* menyimpan semua *file* ke dalam direktori sistem Laravel yang dapat menjadi tempat penyimpanan data berbentuk *file* bernama '*storage*' dan di dalam direktori tersebut dibuatkan direktori khusus untuk menyimpan data berbentuk *file* seperti pada Gambar 3.68.



Gambar 3.68. Direktori tempat menyimpan data berbentuk *file* dalam sistem Laravel

Digunakan tipe formulir yang berbeda saat melakukan permintaan *API*. Tipe yang dimaksud adalah tipe *body multiple/form-data*. Tipe ini dapat melakukan simulasi *input* data dengan tipe *file*. Pada Gambar 3.69 merupakan contoh permintaan *API* dan respons dari *API* jika data yang diproses berhasil.



Gambar 3.69. Tampilan percobaan *API create document*

Pada *field* '*document_file*' ditambahkan '[]' guna untuk menangani lebih dari satu *input* data berbentuk *file*. Dengan ini, *API* dapat dengan mudah memproses

permintaan karena data yang masuk telah dalam bentuk *array* yang mempermudah proses iterasi saat menyimpan data. Dokumen yang tersimpan pada direktori akan mengembalikan rute direktori berbentuk teks.

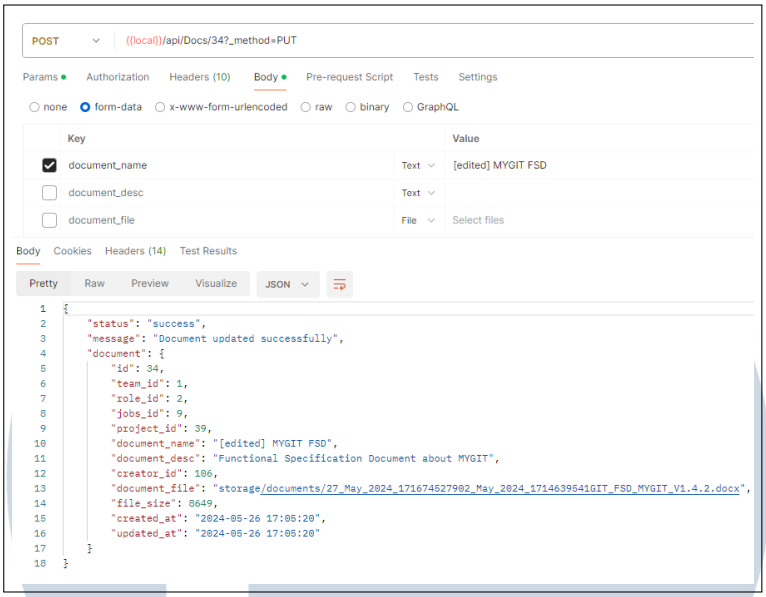
B. API Update Document

API ini digunakan untuk memperbaharui data dokumen yang dipilih melalui *parameter id* dokumen saat melakukan permintaan *API*. *API* memperbaharui data yang diberi *input* saja. Untuk menangani pembaharuan data berbentuk *file*, *API* terlebih dahulu menghapus dokumen yang ada dengan menggunakan rute direktori dari data *document_file*. Digunakan fungsi bawahan dari *Laravel* yang dapat menghapus data berbentuk *file* pada sistem. Contoh penggunaan fungsi ini dapat dilihat pada Gambar 3.70. Setelah dokumen lama terhapus, maka *API* dapat melakukan penyimpanan data *file* dokumen ke dalam sistem. Hal ini dilakukan agar tidak terjadi penumpukan data yang tidak terpakai pada sistem.

```
if (Storage::exists('public/documents/' . $document->document_file)) {
    Storage::delete('public/documents/' . $document->document_file);
}
```

Gambar 3.70. Potongan kode penggunaan fungsi *Storage* untuk menghapus data dalam sistem

Pada Gambar 3.71 merupakan hasil respons saat melakukan pembaharuan nama pada data dokumen. Pada permintaan *API* ini dilakukan pembaharuan nama dokumen dari data dokumen.

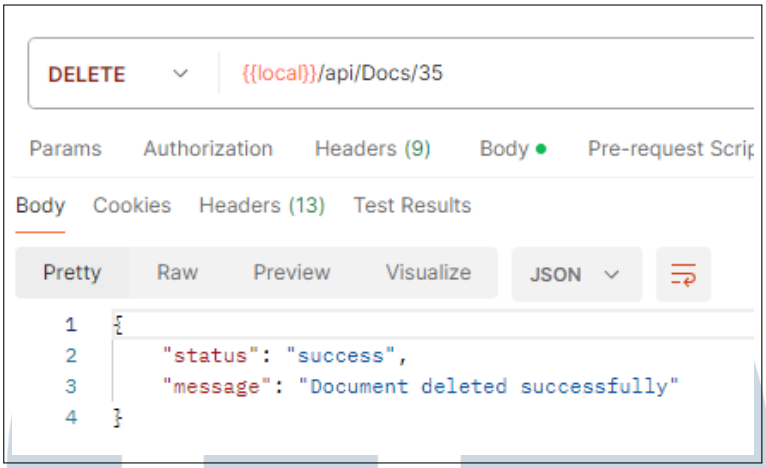


Gambar 3.71. Tampilan percobaan *API update document*

Terlihat pada gambar 3.71 bahwa metode permintaan *API* atau *HTTP Method* yang digunakan bukanlah *PUT*, melainkan *POST*. Hal ini membuat pemanggilan *API* harus menyertakan '*?_method=PUT*' untuk mengubah metode pemanggilan *API* kembali menjadi *PUT*. Ini disebakan oleh Laravel pada versi 10.4.0 tidak dapat menangani permintaan *input* yang memiliki tipe *file* pada metode '*PUT*'.

C. API Delete Document

API ini digunakan untuk menghapus data dokumen, menggunakan *id* dokumen yang didapatkan dari *parameter*. Kemudian *API* melakukan penghapusan data pada tabel *dokumen* beserta data dokumen yang ada pada sistem dengan menggunakan fungsi *Storage* yang persis digunakan untuk menghapus data dokumen pada *API update document*. Jika data berhasil dihapus, *API* akan mengirimkan respons bahwa data dokumen telah dihapus. Hasil respons dapat dilihat pada Gambar 3.72.



Gambar 3.72. Tampilan percobaan *API delete document*

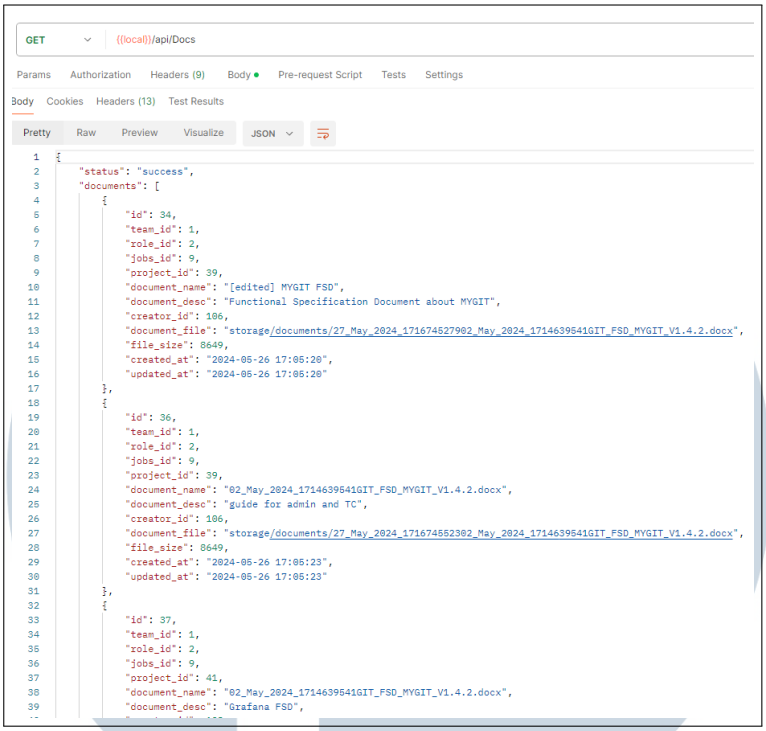
D. API Get Document Data

API ini digunakan untuk mendapatkan data dokumen yang tersimpan. Pada dasarnya respon berbentuk JSON yang dikirimkan *API* untuk mengirimkan data dokumen tidak dapat mengirimkan data bentuk *file*. Oleh karena itu, dilakukan pendekatan lain seperti memberikan rute direktori dari *file* yang tersimpan pada direktori sistem. Ketika sisi *frontend website* menerima data dari rute direktori dokumen tersebut, kemudian melakukan akses secara langsung pada penyimpanan sistem dengan menggunakan direktori tersebut. Sehingga sisi *frontend website* dapat menampilkan gambar atau pun *file* yang tersimpan pada direktori sistem.

Terdapat tiga metode untuk mendapatkan data dokumen yang disediakan pada *API*. Tiap metode memiliki respons berbeda yang digunakan pada kasus penggunaan tertentu.

D.1 Menggambil semua data dokumen

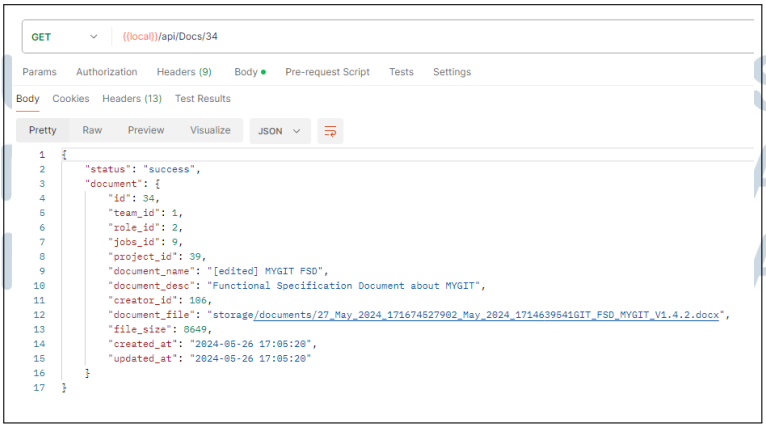
API ini digunakan oleh *admin* untuk mendapatkan semua data dokumen yang ada tanpa ada batasan akses dari *role, job,* maupun *team*. Contoh respons yang diberikan *API* ini dapat dilihat pada gambar 3.73.



Gambar 3.73. Tampilan percobaan *API get all document*

D.2 Menggambil data per dokumen

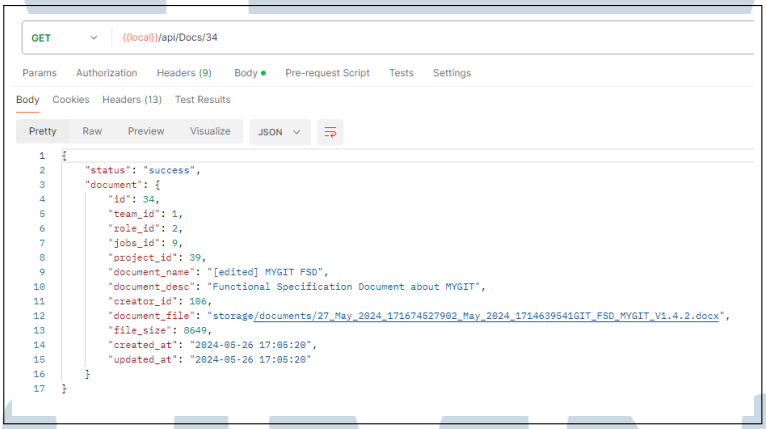
API ini digunakan ketika ingin melihat salah satu data dokumen secara detail pada tampilan sisi *frontend* tanpa harus memanggil semua data dokumen yang ada dan melakukan filtrasi pada data. Data tugas didapatkan dengan menggunakan *id* dokumen yang didapatkan dari parameter dan memasukkan *id* dokumen tersebut pada *query* untuk memanggil data dokumen. Ketika data dokumen ditemukan, maka *API* akan mengambil data dokumen dan mengirimkan respons berhasil beserta data dokumen. Hasil respons dapat dilihat pada Gambar 3.41.



Gambar 3.74. Tampilan percobaan *API get document by id*

D.3 Menggambil semua data dokumen per proyek

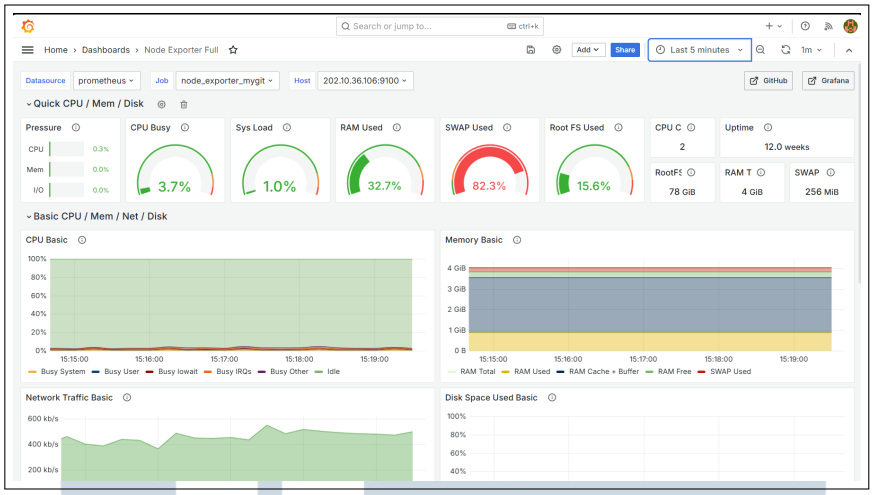
API ini digunakan ketika ingin mendapatkan dokumen yang ada pada salah satu proyek. *API* ini dapat membantu pada sisi *frontend* saat ingin melihat detail pada salah satu proyek dan membantu *admin* dalam melakukan monitor terhadap dokumen-dokumen yang ada pada setiap proyek. *API* ini mendapatkan data dokumen per proyek dengan memanfatkan *id* proyek yang didapatkan pada parameter untuk dimasukan pada saat*query* memanggil data dokumen, menyebabkan data yang diterima secara langsung terfiltrasi. Ketika data dokumen telah ditemukan, maka *API* akan mengambil data dokumen-dokumen tersebut dan mengirimkan respons berhasil beserta data dokumen karyawan tersebut. Hasil respons dapat dilihat pada Gambar 3.75.



Gambar 3.75. Tampilan percobaan *API get document by project*

3.3.9 Penugasan Tambahan Lainnya

Setelah dipindahkan ke dalam tim Splunk, diberikan penugasan untuk melakukan eksplorasi Grafana dan melakukan visualisasi metrik perangkat lunak dan sistem operasi server aplikasi MYGIT. Data yang divisualisasikan adalaha berupa data metrik dari *CPU, RAM, Disk, dan Memory*. Untuk tampilan dari *dashboard* Grafana dapat dilihat pada Gambar 3.76.



Gambar 3.76. Tampilan *dashboard* Grafana untuk MYGIT

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Kendala yang ditemukan selama proses kerja magang adalah sebagai berikut.

1. Dikarenakan kebutuhan yang mendesak, terdapat percepatan tenggat waktu penyelesaian pada beberapa fitur aplikasi sehingga membuat performa fitur menjadi tidak maksimal.
2. Server perusahaan yang digunakan sebagai media *hosting* untuk proses *development* aplikasi sering mengalami masalah hingga membuat *server* tersebut mati.
3. Terdapat banyak *bug* dan *error* pada aplikasi baik pada sisi *backend* maupun *frontend* pada saat awal perilisan.
4. Semakin banyak data yang telah ditampung pada aplikasi membuat proses dan performa dari apliaksi menjadi lambat.

3.4.2 Solusi

Solusi yang digunakan untuk mengatasi kendala yang ditemukan adalah sebagai berikut.

1. Menyelesaikan fitur dengan cepat tanpa memprioritaskan performa, kemudian melakukan permintaan waktu untuk melakukan revisi setelah fitur aplikasi telah berjalan dan digunakan.
2. Menggunakan perangkat pribadi untuk melakukan *hosting development server* dan saling membagikan *private ip* pada anggota *developer* lainnya.
3. Meminta pemberian tugas khusus untuk melakukan *End-to-End Testing* dan melakukan *launching* aplikasi versi beta.
4. Meminta pemberian tugas khusus untuk melakukan *performance enhancing* pada sisi *backend* dengan memperbaharui *database query* menjadi lebih efisien.