

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.

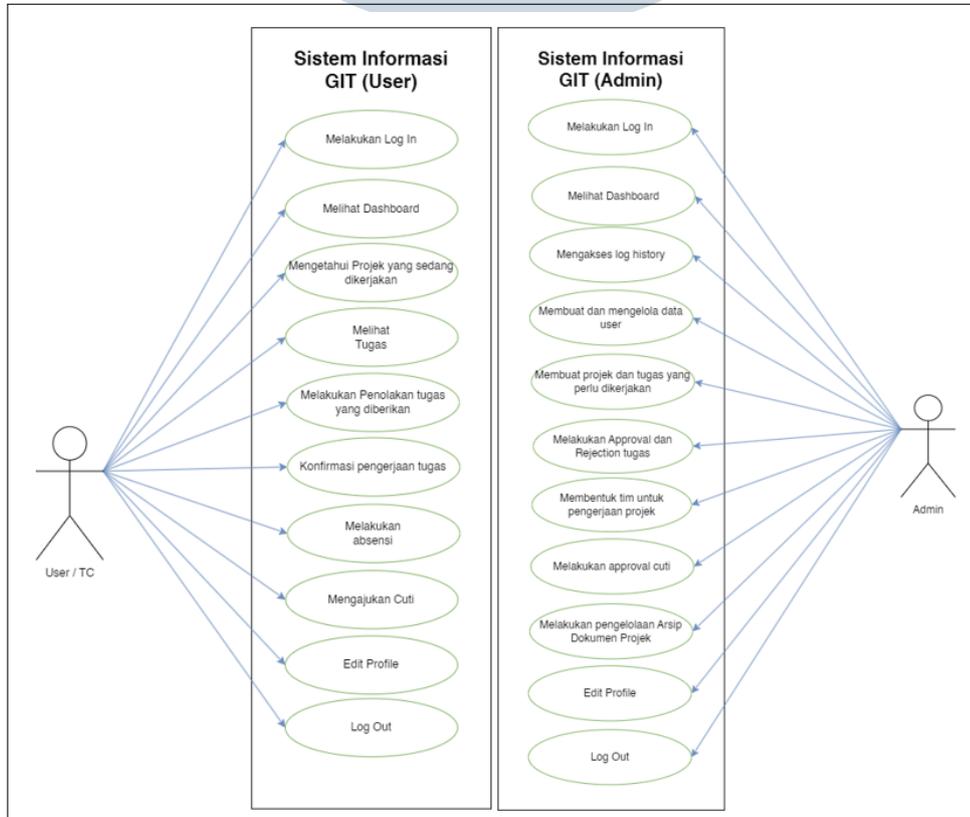
Setiap pagi *Project Manager* akan memberikan arahan dan *check-up* terkait pekerjaan yang akan dilakukan pada hari tersebut. Kemudian pada sore hari, tim akan melaporkan *progress* dari pekerjaan tersebut. Pada hari jumat, dilaksanakan pertemuan retrospeksi bersama semua divisi *Technical Consultant* 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 Februari 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 proyek 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 proyek 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 memvisualisasikan 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 <i>database</i> dan melakukan instalasi dan konfigurasi <i>Database</i> pada <i>server deployment</i>
2	Melakukan konfigurasi CI/CD menggunakan pada development server untuk mempermudah <i>deployment</i> Melakukan Instalasi dan konfigurasi <i>framework</i> untuk <i>backend API</i>
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 <i>project</i> dan <i>task</i> beserta fitur <i>task invitation</i> , <i>task submission</i> , dan <i>task review</i> Melakukan replikasi server dari development server ke production server
7	Membuat tampilan <i>frontend</i> beserta <i>API</i> untuk fitur pengajuan lembur karyawan
8	Membuat <i>API</i> untuk fitur pengajuan cuti karyawan
9-10	Membuat <i>API</i> untuk arsip dokumen
11	Melakukan End-to-End testing dan <i>bug finding</i>
12	Melakukan perbaikan pada <i>bugs</i> yang ditemukan Membuat <i>technical document</i> untuk <i>backend API</i> MYGIT
13	Memahami tim baru dan mempelajari infrastruktur Splunk dari salah satu klien GIT
14-16	Melakukan eksplorasi terhadap teknologi Grafana untuk pembuatan <i>Monitoring Tools</i> 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.

db_mygit_production mg_employee id : bigint(20) unsigned role_id : bigint(20) unsigned jobs_id : bigint(20) unsigned team_id : bigint(20) unsigned employee_name : varchar(255) date_of_birth : date age : varchar(255) mobile_number : varchar(255) email : varchar(255) username : varchar(255) password : varchar(255) gender : enum('Male','Female') religion : varchar(255) npwp_number : varchar(255) identity_number : varchar(255) address : varchar(255) created_at : timestamp updated_at : timestamp	db_mygit_production mg_roles id : bigint(20) unsigned role_name : enum('ADMIN','STAFF') description : varchar(255) created_at : timestamp updated_at : timestamp
	db_mygit_production mg_teams id : bigint(20) unsigned team_name : varchar(255) description : varchar(255) created_at : timestamp updated_at : timestamp
	db_mygit_production mg_jobs id : bigint(20) unsigned job_name : varchar(255) description : varchar(255) created_at : timestamp updated_at : timestamp

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

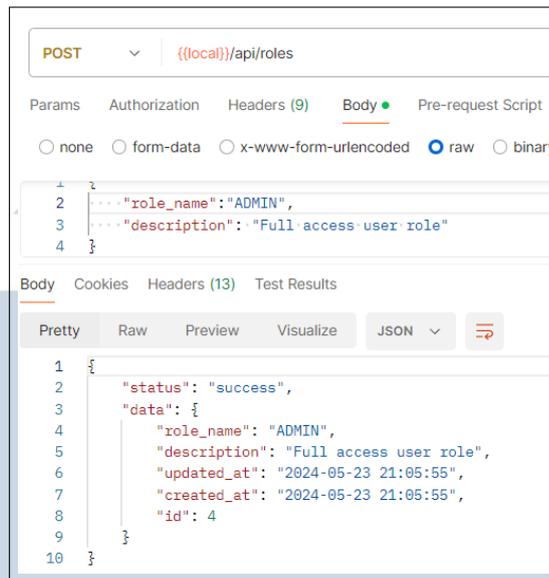
Sebelum membuat data *user* baru, diperlukan pembuatan *role*, *job*, dan *team*. Data dari *role*, *job*, dan *team* akan menjadi pilihan yang dapat dipilih wajib pada saat membuat akun *user*.

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

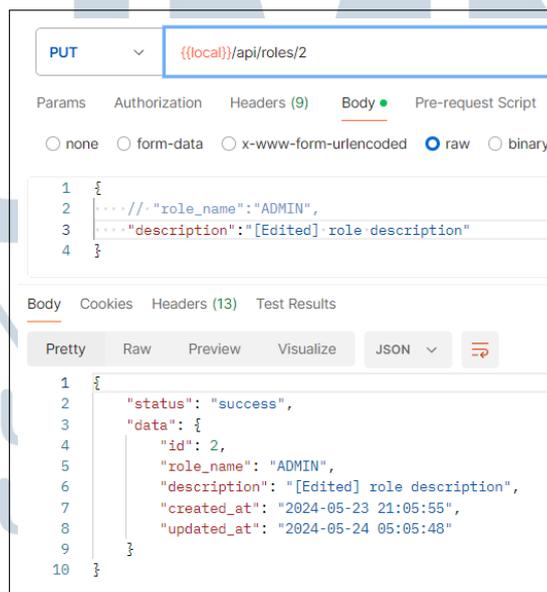
Untuk membuat *role* baru dibutuhkan informasi tentang nama dari *role* tersebut beserta deskripsi *role* (opsional). Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *error*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



Gambar 3.3. Tampilan percobaan API create role

A.2 Update Role

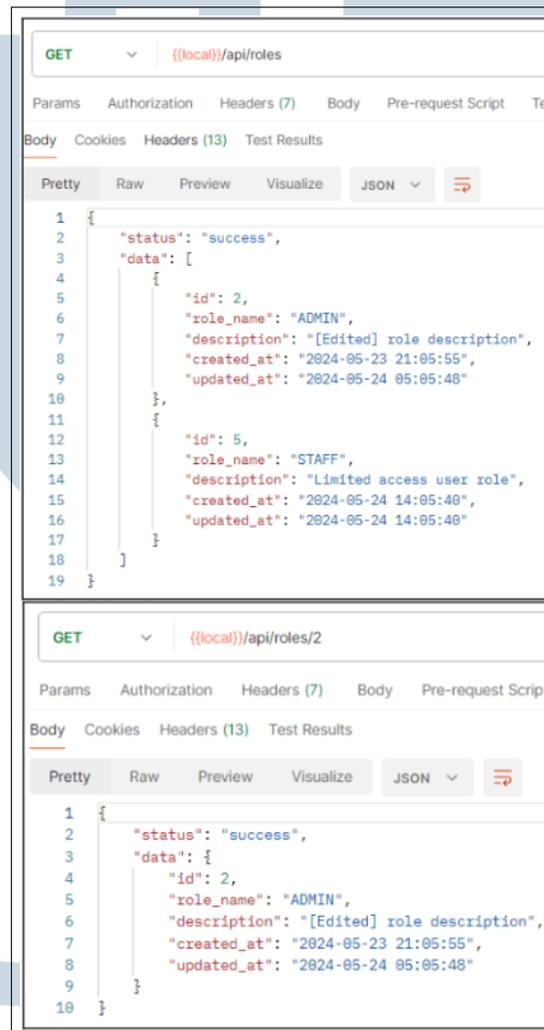
Untuk melakukan pembaharuan data *role* dibutuhkan informasi tentang nama dari *role* tersebut beserta deskripsi *role*. Kemudian kode akan melakukan validasi *input* dan mengirim respons yang sesuai dengan *input* yang diterima dan mengirimkan respons *success* jika berhasil dan mengirimkan respons *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

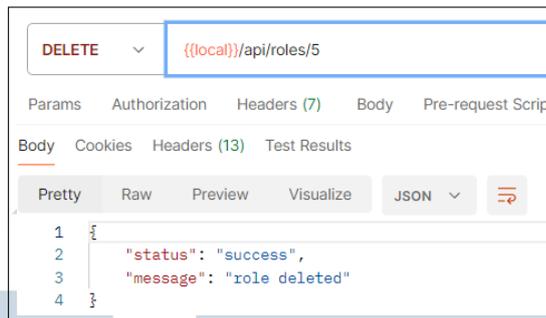
API untuk mendapatkan data *role* terdapat dua cara, yaitu mendapatkan semua data *role*, dan mendapatkan satu data *role* secara spesifik berdasarkan dari *id role* yang diterima. Respons dari API dapat dilihat pada Gambar 3.5.



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.



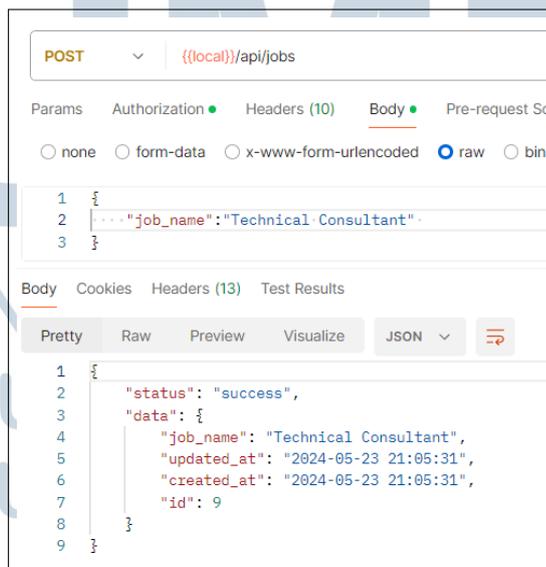
Gambar 3.6. Tampilan percobaan API delete role

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

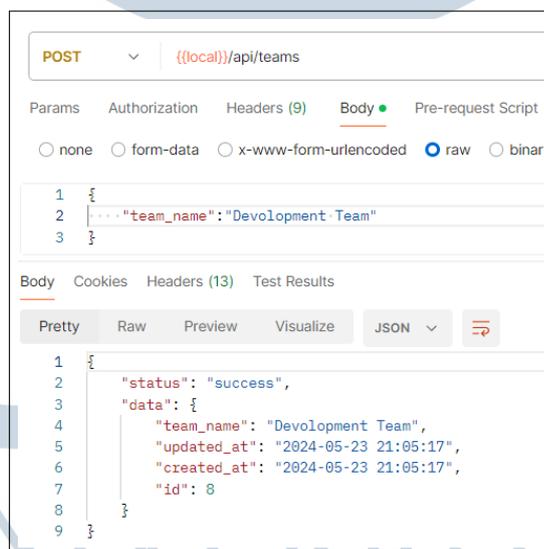
API update, delete dan get job data memiliki struktur dan respons yang sama seperti fungsi yang dimiliki *API role* di atas.

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.



```

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();

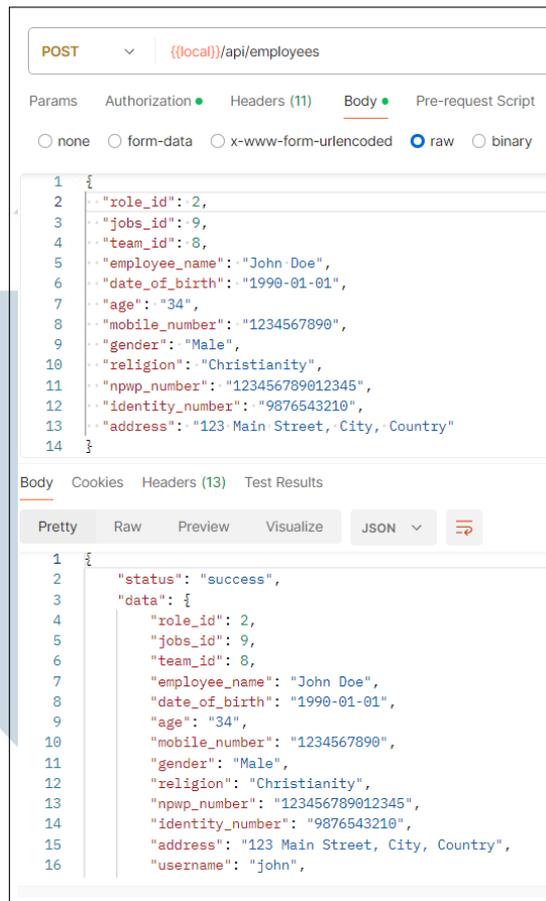
        return response()->json([
            'status' => 'success',
            'data' => $employee
        ], 200);
    } catch (\Exception $e) {
        DB::rollback();

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

```

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.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

```

$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])(?=[!@#%&*()-_+={};:,<.>0-9~]).*$/',
    ],
];
$data = $request->all();

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

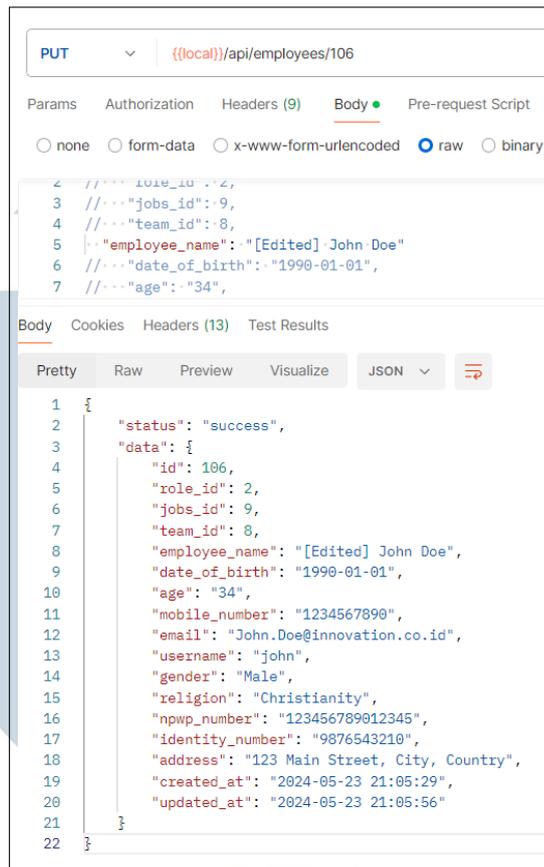
if ($validator->fails()){
    return response()->json([
        'status' => 'error',
        'message' => $validator->errors()
    ], 400);
}

```

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* membuat akun untuk karyawan baru, karyawan telah memiliki akun yang dapat digunakan untuk mengakses *website* dapat menggunakan fitur-fitur dari MYGIT.

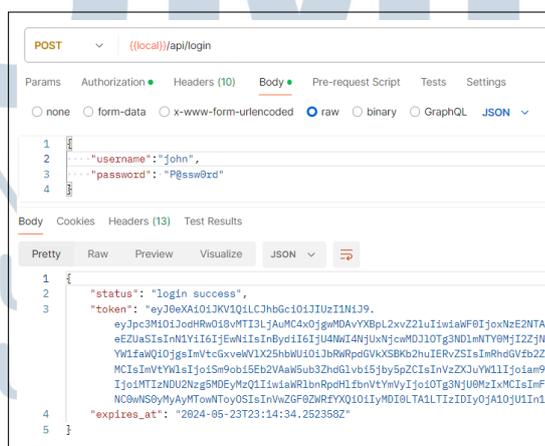
U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A

A. API login

```
public function login(Request $request) {  
    try {  
        $request->validate([  
            'username' => 'required|string',  
            'password' => 'required|string',  
        ]);  
        if (!$employee = Employee::where('username', $request->username)->first()) {  
            return response()->json(['error' => 'Invalid credentials'], 401);  
        }  
        if (!Hash::check($request->password, $employee->password)) {  
            return response()->json(['error' => 'Invalid credentials'], 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.



The screenshot shows a web browser's developer tools interface. The top bar indicates a POST request to `((local))/api/login`. The 'Body' tab is selected, showing the request body as a JSON object: `{ "username": "john", "password": "P8ssw0rd" }`. Below this, the 'Body' section shows the response body in 'Pretty' format: `{ "status": "login success", "token": "eyJ0eXAiOiJKV1QiOiJhbnN1IiwiaWF0Ijoi2024-05-23T23:14:34.252358Z" }`. The response also includes headers and cookies.

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*.

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

        if (!$employee) {
            return response()->json([
                'status' => 'error',
                'message' => 'Employee not found'
            ], 404);
        }
        $newPassword = $data['password'];
        if (!preg_match('/^(?=.*[A-Z])(?=.*[!@#$%^&*()-_+={};:,<.>|~`~]).*$/', $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.

Column Name	Data Type	Constraints
id	bigint(20) unsigned	Primary Key
created_at	timestamp	
updated_at	timestamp	
employee_id	bigint(20) unsigned	
checkin	timestamp	
checkout	timestamp	
status	varchar(255)	
note	varchar(255)	
image_string	varchar(255)	

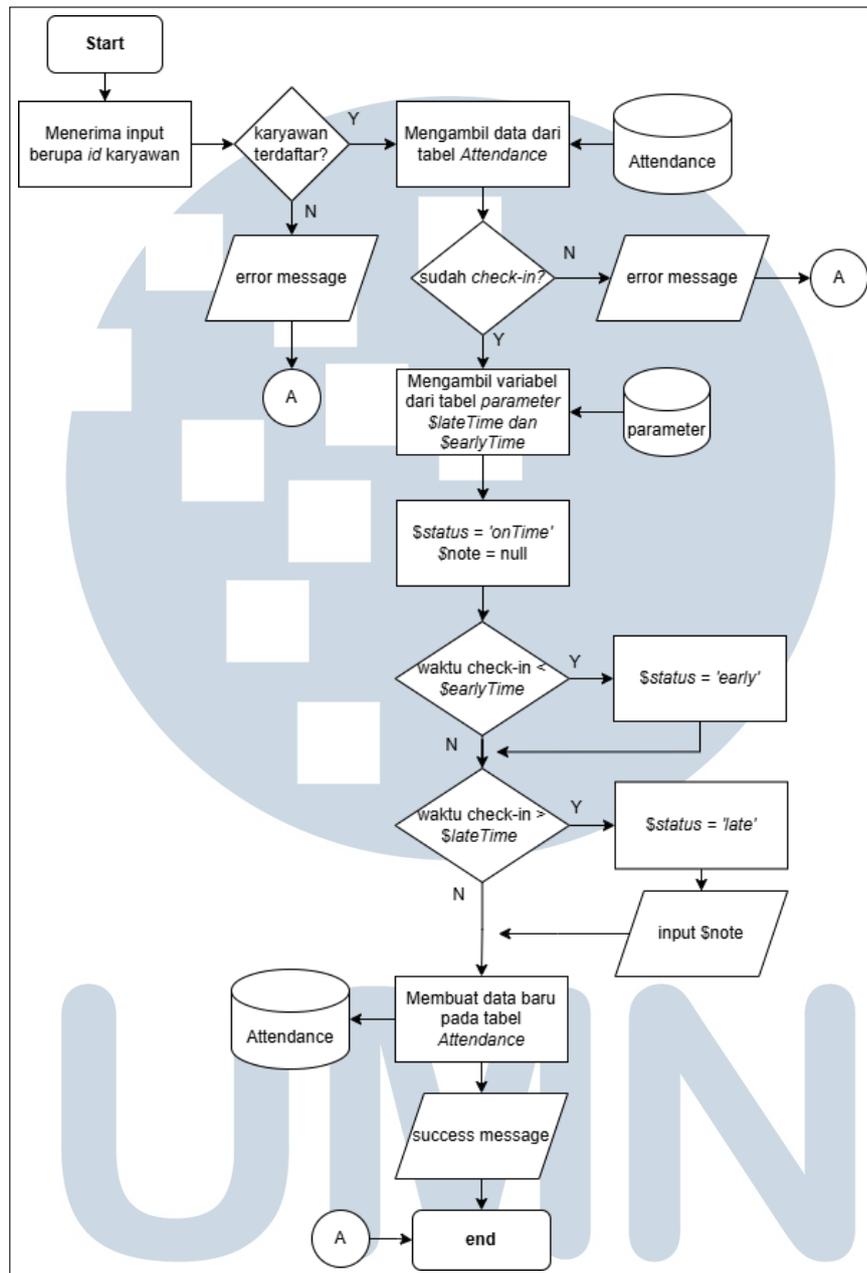
Gambar 3.16. Atribut basis data dari tabel *attendance*

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.



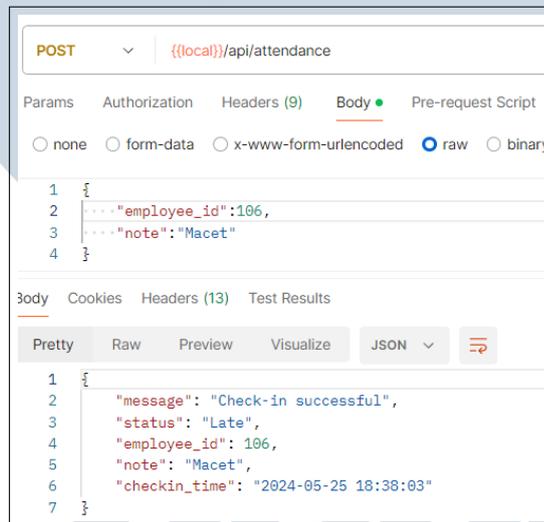
Gambar 3.17. Flowchart API check-in

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*.



```

POST {{local}}/api/attendance

Params Authorization Headers (9) Body Pre-request Script
none form-data x-www-form-urlencoded raw binary
1 {
2   ... "employee_id": 106,
3   ... "note": "Macet"
4 }

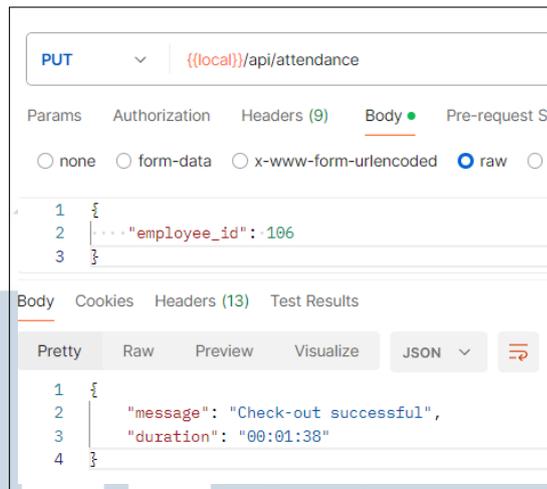
body Cookies Headers (13) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "message": "Check-in successful",
3   "status": "Late",
4   "employee_id": 106,
5   "note": "Macet",
6   "checkin_time": "2024-05-25 18:38:03"
7 }

```

Gambar 3.19. Tampilan percobaan *API check-in*

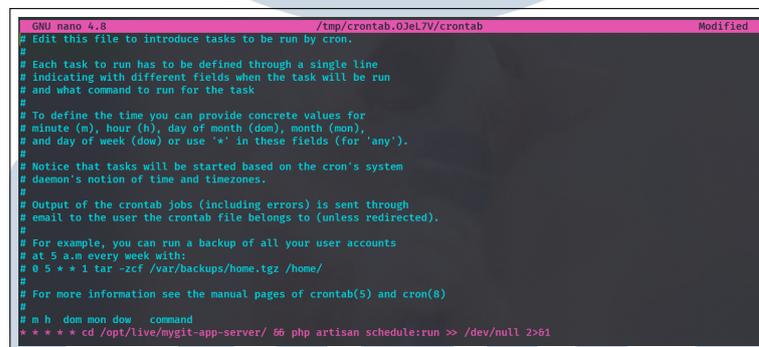
B. API Check-out

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.

```

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

```

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);
    }
}

```

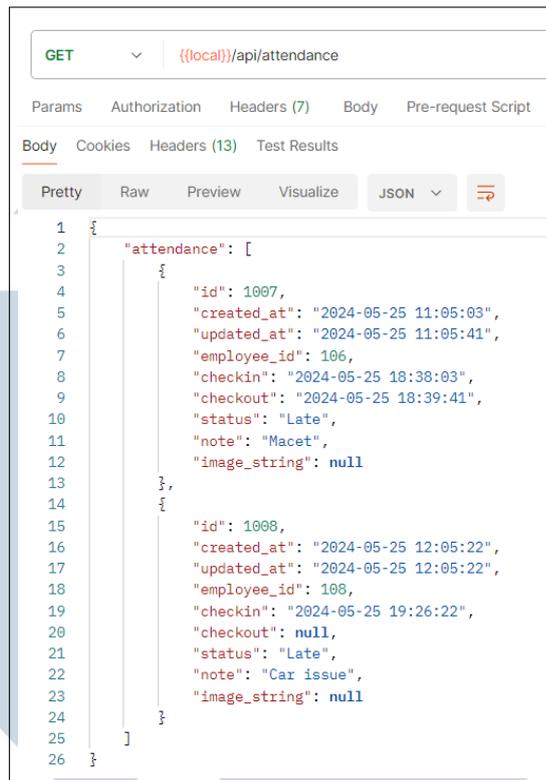
Gambar 3.23. Potongan kode *API auto check-out*

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.

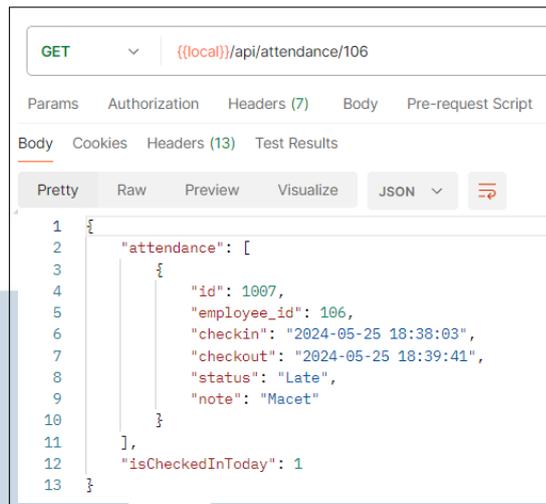


Gambar 3.24. Tampilan percobaan API get check-in untuk admin

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.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

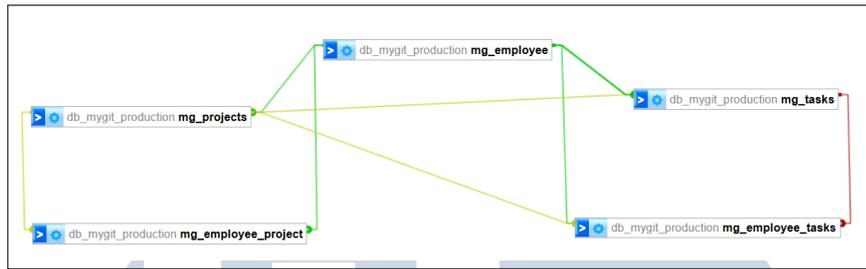


Gambar 3.25. Tampilan percobaan API get check-in untuk karyawan

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.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

Table Name	Column Name	Data Type
db_mygit_production mg_projects	id	bigint(20) unsigned
	project_name	varchar(255)
	project_desc	varchar(255)
	assign_by	bigint(20) unsigned
	team_id	bigint(20) unsigned
	role_id	bigint(20) unsigned
	jobs_id	bigint(20) unsigned
	start_date	datetime
	end_date	datetime
	project_status	enum('onPending','workingOnit','Completed')
	percentage	varchar(255)
	total_task_completed	varchar(255)
	total_task_created	varchar(255)
	created_at	timestamp
	updated_at	timestamp
db_mygit_production mg_employee_project	id	bigint(20) unsigned
	project_id	bigint(20) unsigned
	employee_id	bigint(20) unsigned
	created_at	timestamp
	updated_at	timestamp

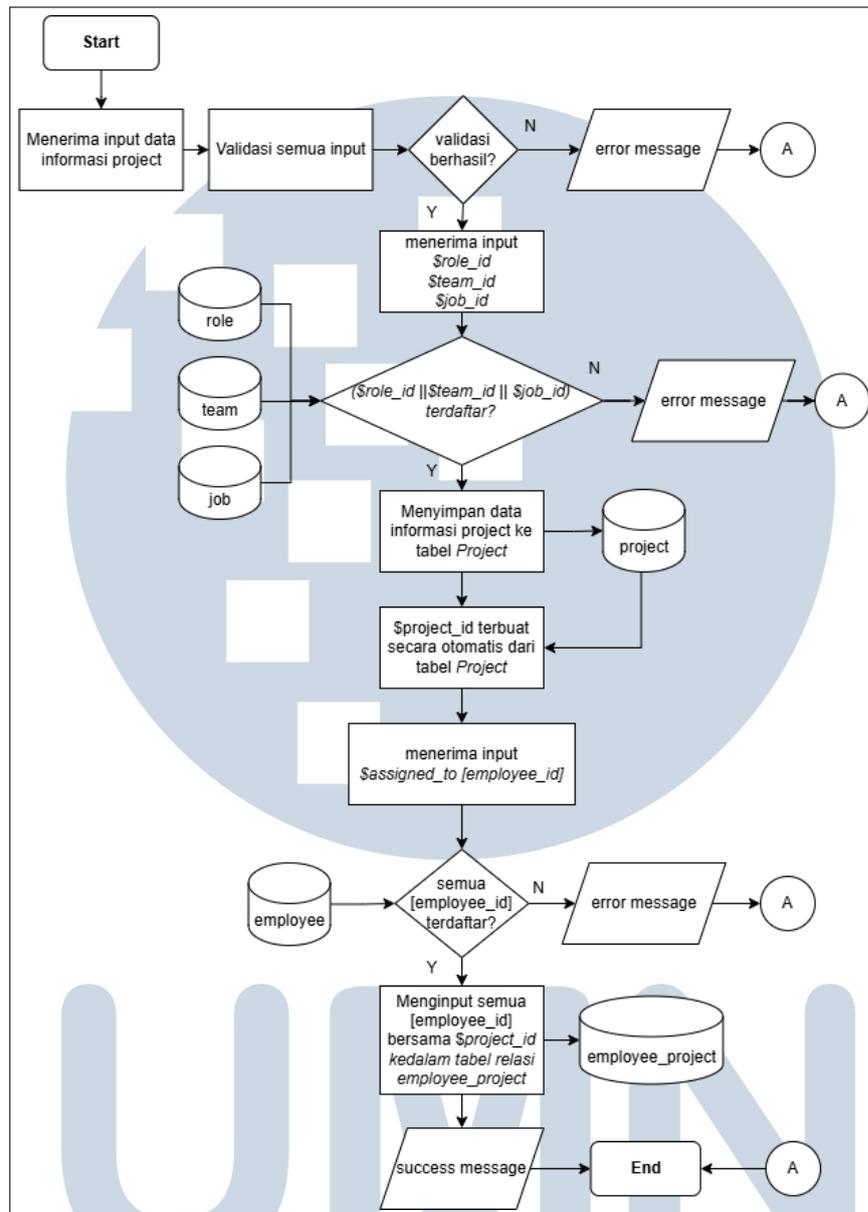
Gambar 3.27. Atribut basis data dari tabel *project* dan *employee_project*

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*.

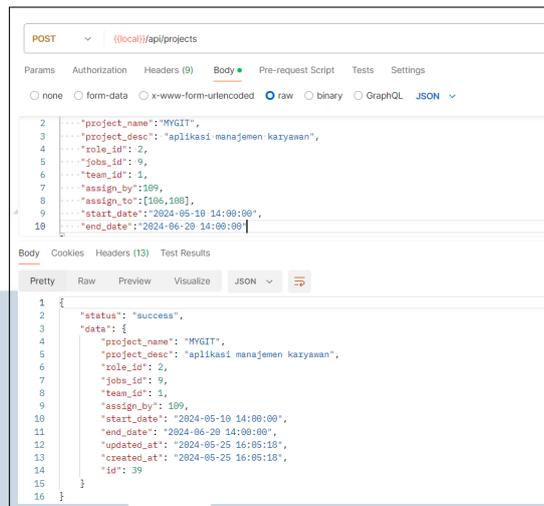
Jika data tersebut berhasil maka akan melanjutkan validasi terhadap *input* *role_id*, *team_id*, dan *job_id* yang terlibat untuk melakukan pengecekan jika data tersebut telah terdaftar atau belum pada tabel masing-masing. Kemudian semua data akan ditambahkan pada tabel *project* dan kemudian akan menghasilkan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

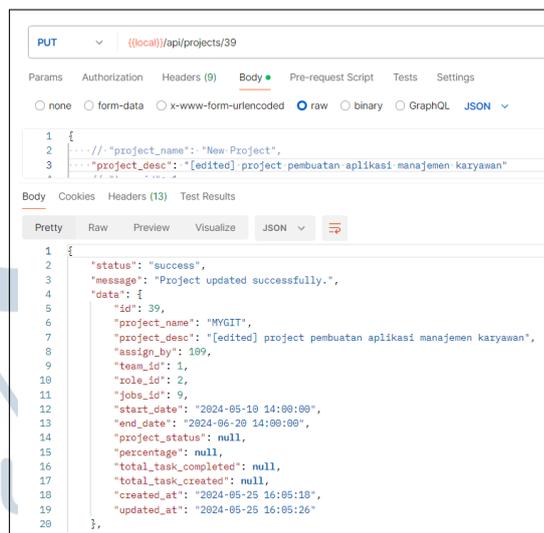
Jika semua proses dilewati dengan benar, maka API akan memberikan respons seperti pada Gambar 3.29. Respons yang diberikan berupa data dari proyek yang baru saja dibuat.



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.



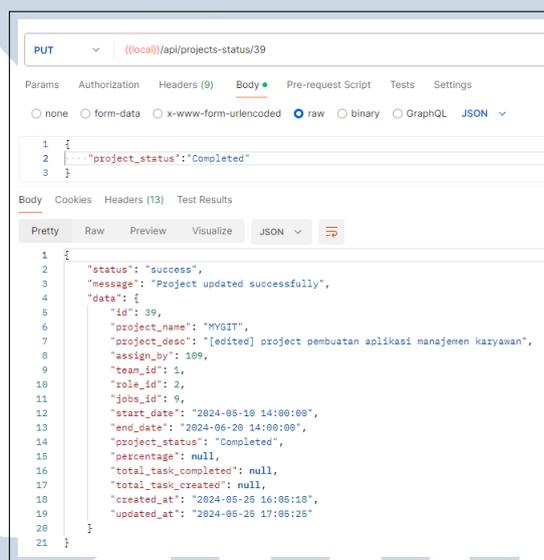
Gambar 3.30. Tampilan percobaan *API update project*

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.



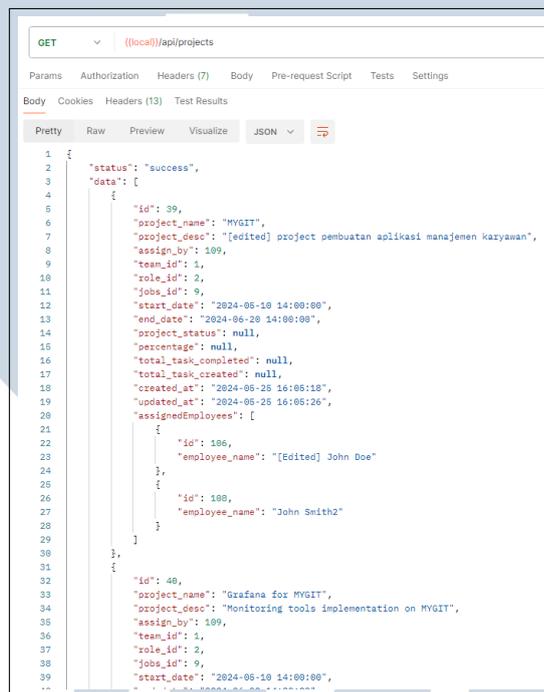
Gambar 3.31. Tampilan percobaan *API update project status*

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.



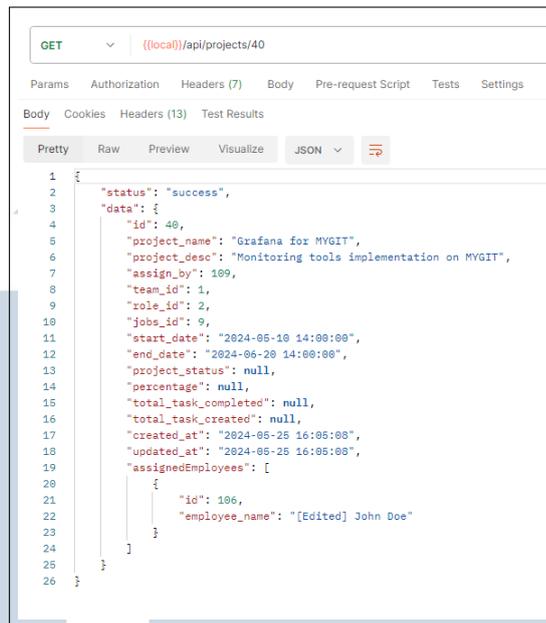
```
GET /api/projects

{"status": "success",
 "data": [
  {
    "id": 39,
    "project_name": "MYGIT",
    "project_desc": "[Edited] project pembuatan aplikasi manajemen karyawan",
    "assign_by": 109,
    "team_id": 1,
    "role_id": 2,
    "jobs_id": 9,
    "start_date": "2024-05-19 14:00:00",
    "end_date": "2024-06-20 14:00:00",
    "project_status": null,
    "percentage": null,
    "total_task_completed": null,
    "total_task_created": null,
    "created_at": "2024-05-25 16:05:18",
    "updated_at": "2024-05-25 16:05:26",
    "assignedEmployees": [
      {
        "id": 186,
        "employee_name": "[Edited] John Doe"
      },
      {
        "id": 188,
        "employee_name": "John Smith2"
      }
    ]
  },
  {
    "id": 40,
    "project_name": "Grafana for MYGIT",
    "project_desc": "Monitoring tools implementation on MYGIT",
    "assign_by": 109,
    "team_id": 5,
    "role_id": 2,
    "jobs_id": 9,
    "start_date": "2024-05-19 14:00:00",
```

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

A.4.2 Mengambil data per project

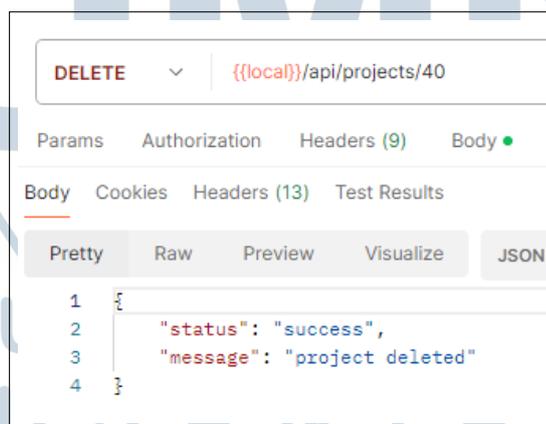
Untuk mendapatkan salah satu data proyek yang ada, diperlukan permintaan *API* dengan menyertakan *id* proyek sebagai parameter pada *route*, kemudian *API* melakukan pencaharian data proyek berdasarkan *id* proyek pada tabel *project*, jika data ditemukan maka *API* akan mengirimkan data dalam bentuk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

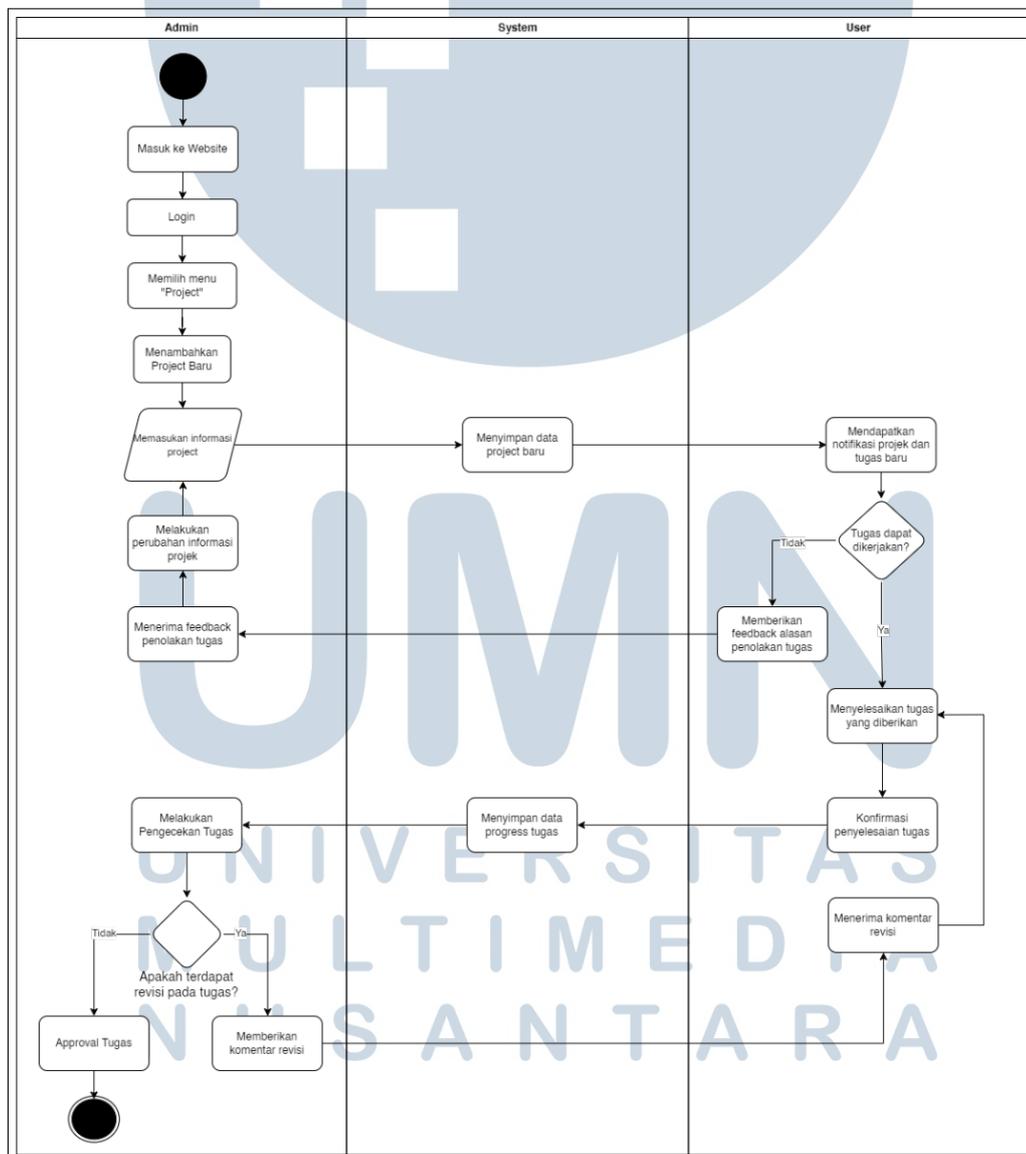


Gambar 3.34. Tampilan percobaan API delete project

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.

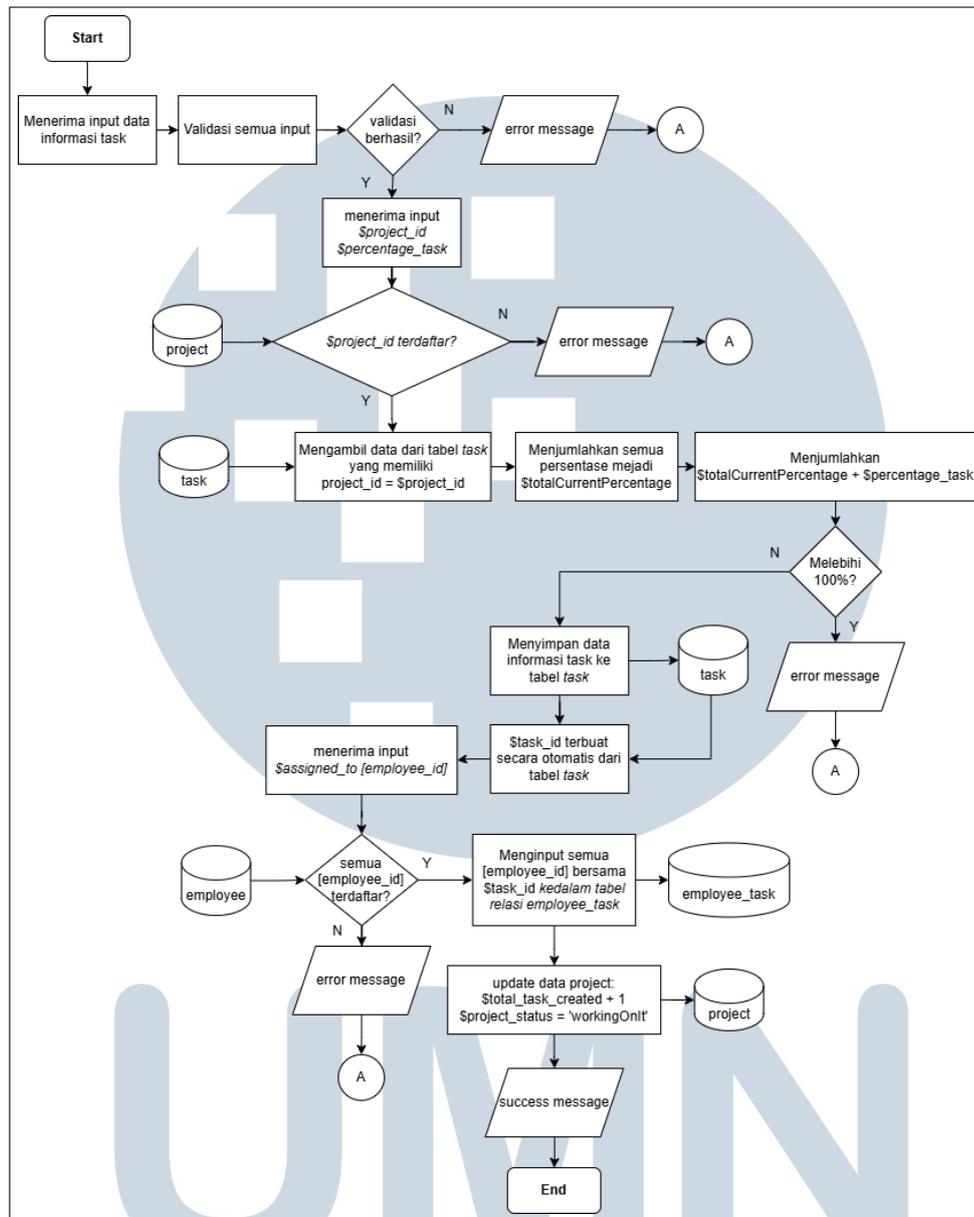
Table Name	Attribute Name	Attribute Type
db_mygit_production mg_tasks	id	bigint(20) unsigned
	project_id	bigint(20) unsigned
	assign_by	bigint(20) unsigned
	task_name	varchar(255)
	task_desc	varchar(255)
	start_date	datetime
	end_date	datetime
	percentage_task	int(11)
	task_status	enum('onPending','onReview','workingOnIt','Completed')
	decided_by	bigint(20) unsigned
	task_submit_status	enum('earlyFinish','finish','finish in delay','overdue')
	completed_date	datetime
	task_image	varchar(255)
	uploaded_by	bigint(20) unsigned
	task_reason	varchar(255)
	created_at	timestamp
updated_at	timestamp	
db_mygit_production mg_employee_tasks	id	bigint(20) unsigned
	employee_id	bigint(20) unsigned
	tasks_id	bigint(20) unsigned
	project_id	bigint(20) unsigned
	isAccepted	tinyint(1)
	created_at	timestamp
	updated_at	timestamp

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*.

UIN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.37. Flowchart API create task

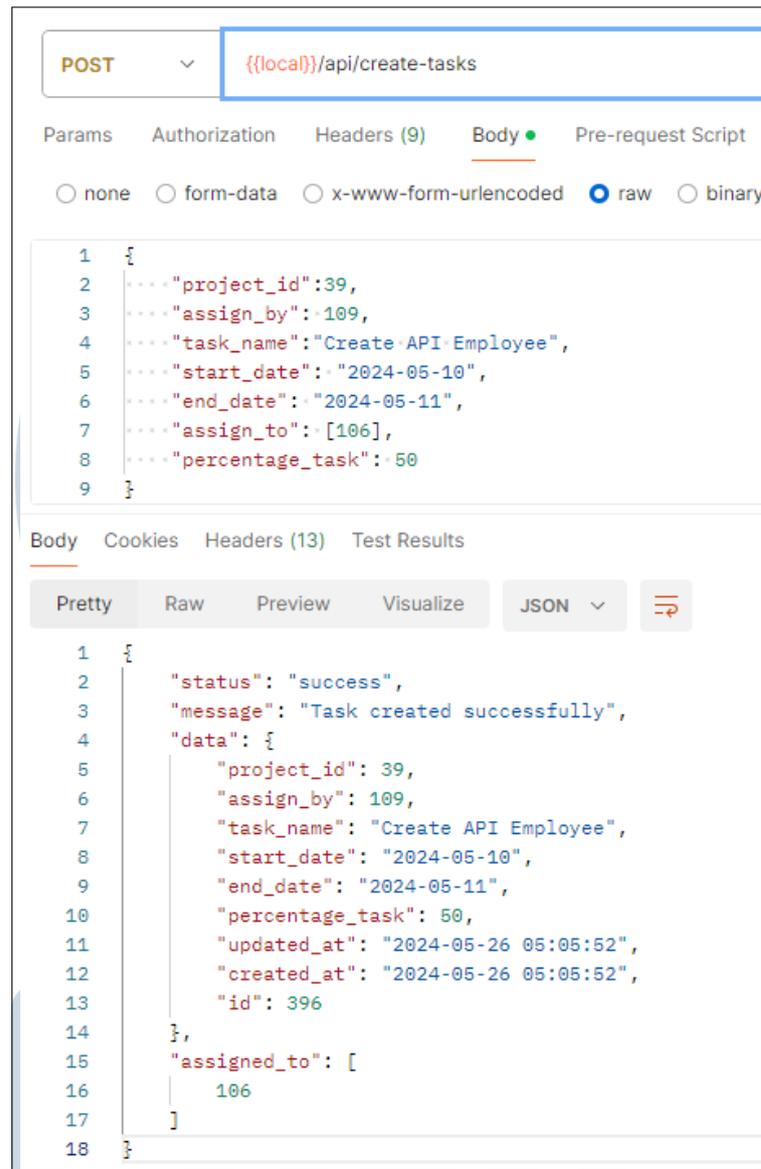
Jika *input* tersebut telah benar, dengan menggunakan *id* proyek yang diterima dari *input API* akan melanjutkan dengan mengambil informasi dari data proyek yang terkait. Kemudian *API* akan mengambil semua informasi tugas yang ada pada tabel *task* untuk mengumpulkan semua tugas yang berkaitan dengan proyek tersebut. Hal tersebut dilakukan untuk menghitung keseluruhan persentase yang dimiliki semua tugas 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.





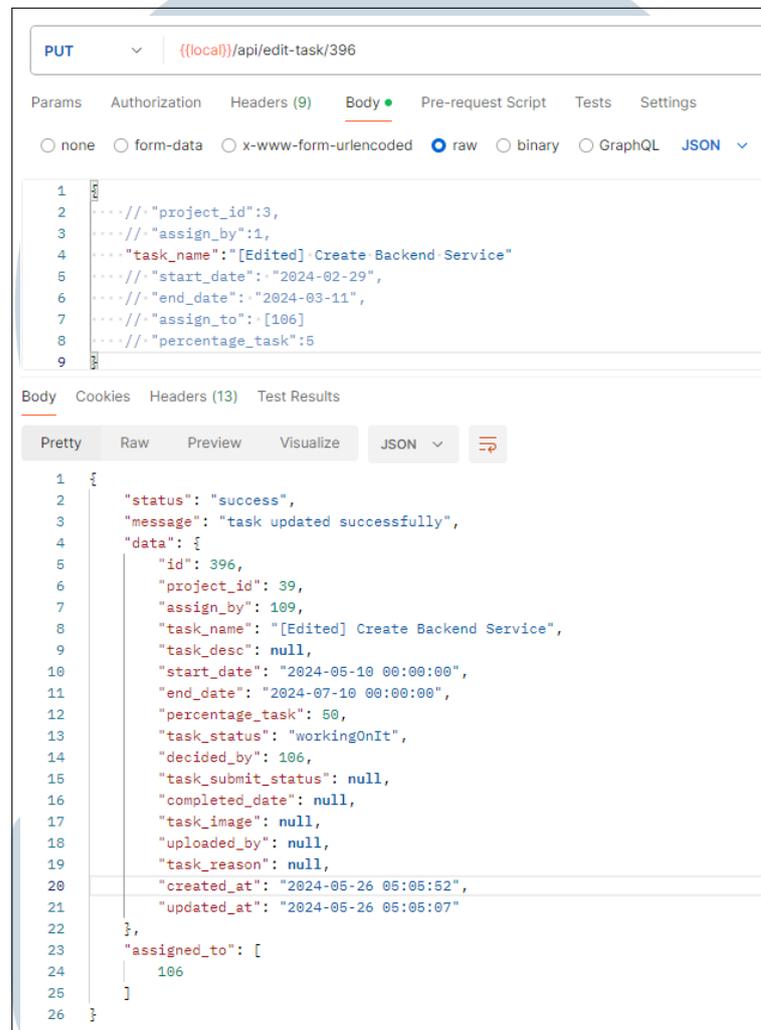
Gambar 3.38. Tampilan percobaan API create task

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.



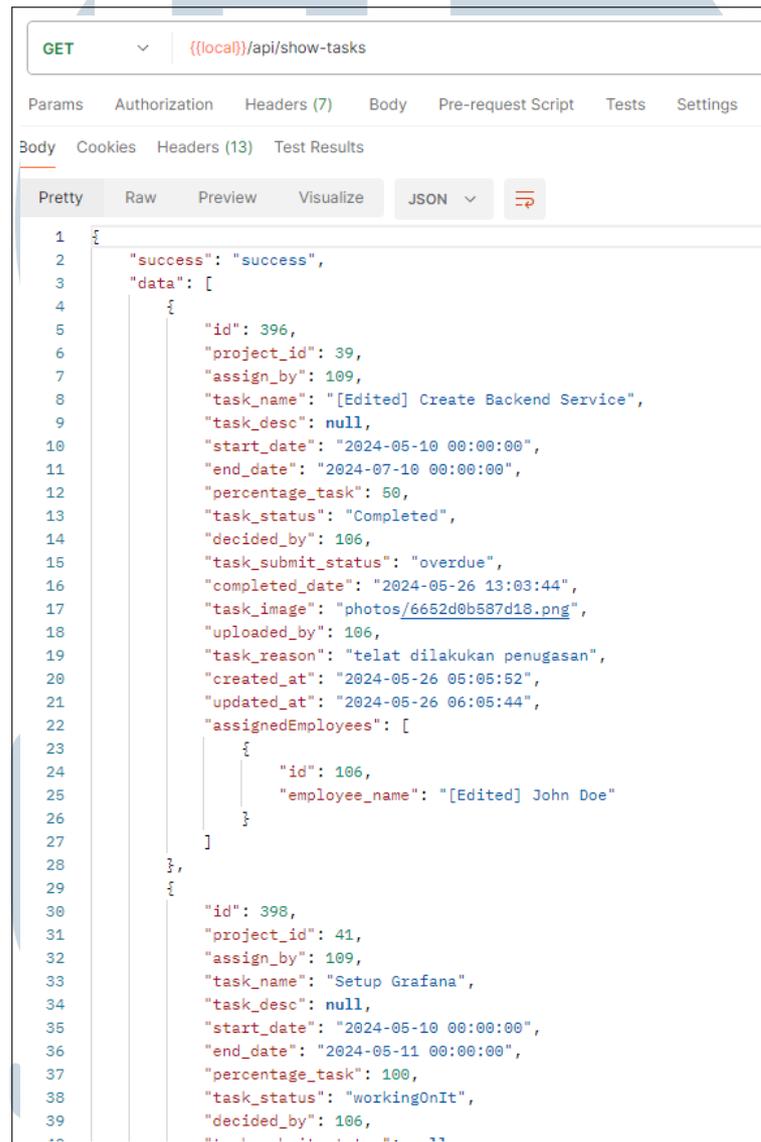
Gambar 3.39. Tampilan percobaan *API update task*

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.



```
1 {
2   "success": "success",
3   "data": [
4     {
5       "id": 396,
6       "project_id": 39,
7       "assign_by": 109,
8       "task_name": "[Edited] Create Backend Service",
9       "task_desc": null,
10      "start_date": "2024-05-10 00:00:00",
11      "end_date": "2024-07-10 00:00:00",
12      "percentage_task": 50,
13      "task_status": "Completed",
14      "decided_by": 106,
15      "task_submit_status": "overdue",
16      "completed_date": "2024-05-26 13:03:44",
17      "task_image": "photos/6652d0b587d18.png",
18      "uploaded_by": 106,
19      "task_reason": "telat dilakukan penugasan",
20      "created_at": "2024-05-26 05:05:52",
21      "updated_at": "2024-05-26 06:05:44",
22      "assignedEmployees": [
23        {
24          "id": 106,
25          "employee_name": "[Edited] John Doe"
26        }
27      ]
28    },
29    {
30      "id": 398,
31      "project_id": 41,
32      "assign_by": 109,
33      "task_name": "Setup Grafana",
34      "task_desc": null,
35      "start_date": "2024-05-10 00:00:00",
36      "end_date": "2024-05-11 00:00:00",
37      "percentage_task": 100,
38      "task_status": "workingOnIt",
39      "decided_by": 106,
```

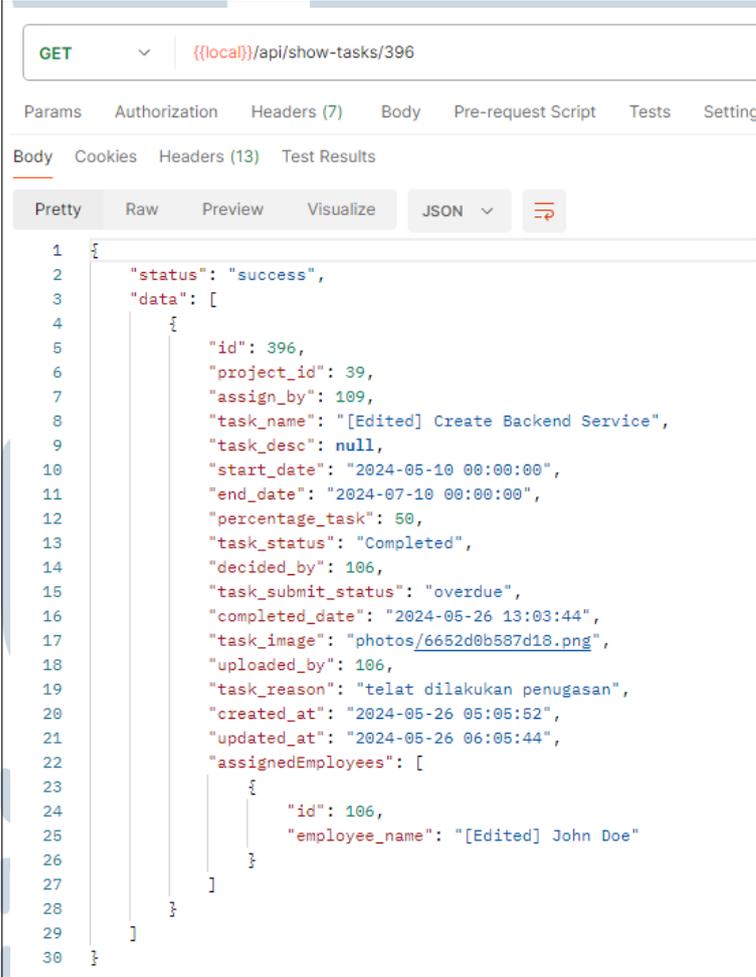
Gambar 3.40. Tampilan percobaan *API get all task*

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.

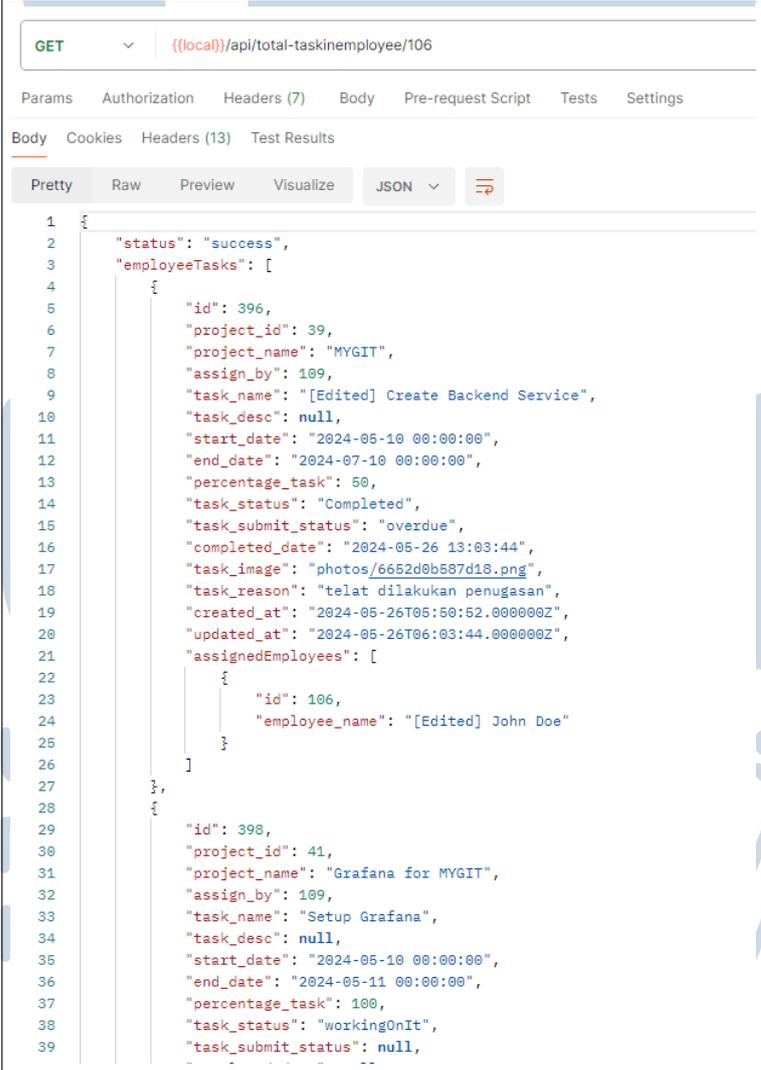


```
1  {
2    "status": "success",
3    "data": [
4      {
5        "id": 396,
6        "project_id": 39,
7        "assign_by": 109,
8        "task_name": "[Edited] Create Backend Service",
9        "task_desc": null,
10       "start_date": "2024-05-10 00:00:00",
11       "end_date": "2024-07-10 00:00:00",
12       "percentage_task": 50,
13       "task_status": "Completed",
14       "decided_by": 106,
15       "task_submit_status": "overdue",
16       "completed_date": "2024-05-26 13:03:44",
17       "task_image": "photos/6652d0b587d18.png",
18       "uploaded_by": 106,
19       "task_reason": "telat dilakukan penugasan",
20       "created_at": "2024-05-26 05:05:52",
21       "updated_at": "2024-05-26 06:05:44",
22       "assignedEmployees": [
23         {
24           "id": 106,
25           "employee_name": "[Edited] John Doe"
26         }
27       ]
28     }
29   ]
30 }
```

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.



```
GET {{local}}/api/total-taskinemployee/106

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies Headers (13) Test Results

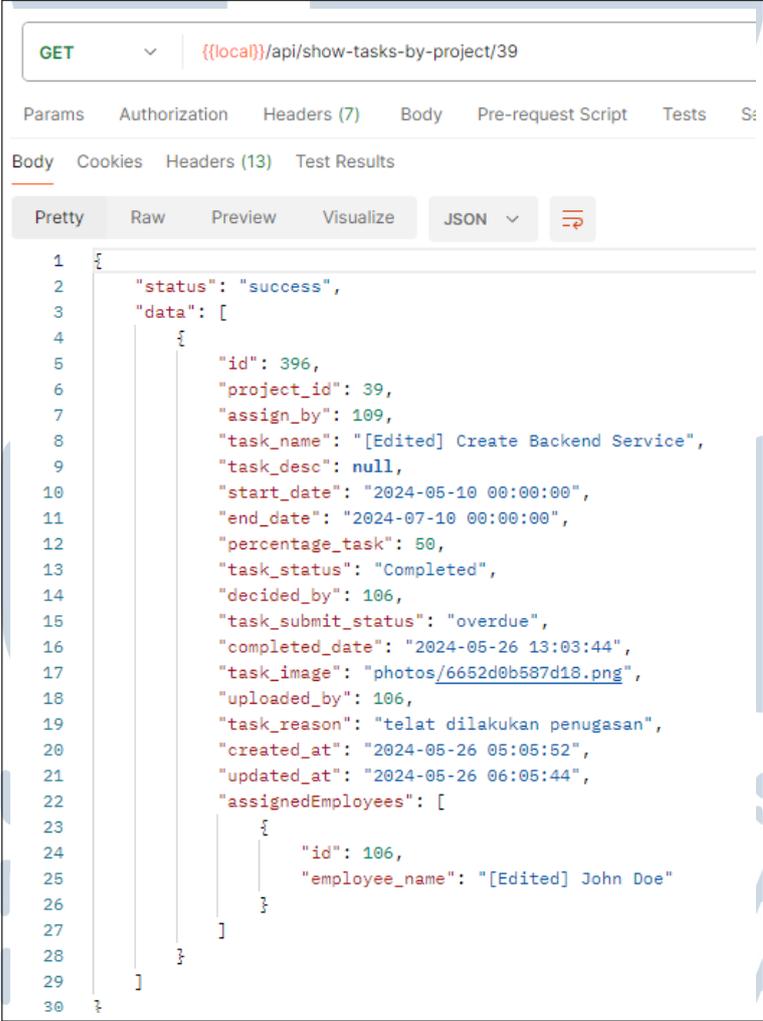
Pretty Raw Preview Visualize JSON

1 {
2   "status": "success",
3   "employeeTasks": [
4     {
5       "id": 396,
6       "project_id": 39,
7       "project_name": "MYGIT",
8       "assign_by": 109,
9       "task_name": "[Edited] Create Backend Service",
10      "task_desc": null,
11      "start_date": "2024-05-10 00:00:00",
12      "end_date": "2024-07-10 00:00:00",
13      "percentage_task": 50,
14      "task_status": "Completed",
15      "task_submit_status": "overdue",
16      "completed_date": "2024-05-26 13:03:44",
17      "task_image": "photos/6652d0b507d18.png",
18      "task_reason": "telat dilakukan penugasan",
19      "created_at": "2024-05-26T05:50:52.000000Z",
20      "updated_at": "2024-05-26T06:03:44.000000Z",
21      "assignedEmployees": [
22        {
23          "id": 106,
24          "employee_name": "[Edited] John Doe"
25        }
26      ]
27    },
28    {
29      "id": 398,
30      "project_id": 41,
31      "project_name": "Grafana for MYGIT",
32      "assign_by": 109,
33      "task_name": "Setup Grafana",
34      "task_desc": null,
35      "start_date": "2024-05-10 00:00:00",
36      "end_date": "2024-05-11 00:00:00",
37      "percentage_task": 100,
38      "task_status": "workingOnIt",
39      "task_submit_status": null,
```

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.

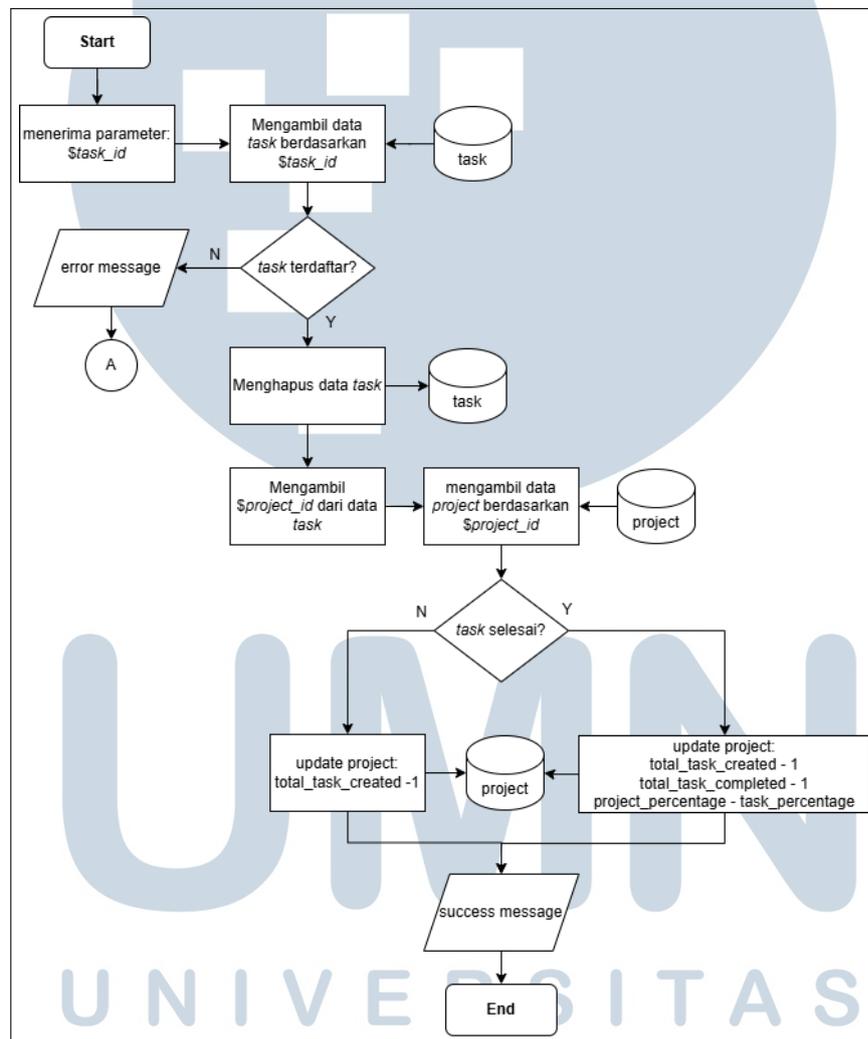


```
1  {
2    "status": "success",
3    "data": [
4      {
5        "id": 396,
6        "project_id": 39,
7        "assign_by": 109,
8        "task_name": "[Edited] Create Backend Service",
9        "task_desc": null,
10       "start_date": "2024-05-10 00:00:00",
11       "end_date": "2024-07-10 00:00:00",
12       "percentage_task": 50,
13       "task_status": "Completed",
14       "decided_by": 106,
15       "task_submit_status": "overdue",
16       "completed_date": "2024-05-26 13:03:44",
17       "task_image": "photos/6652d0b587d18.png",
18       "uploaded_by": 106,
19       "task_reason": "telat dilakukan penugasan",
20       "created_at": "2024-05-26 05:05:52",
21       "updated_at": "2024-05-26 06:05:44",
22       "assignedEmployees": [
23         {
24           "id": 106,
25           "employee_name": "[Edited] John Doe"
26         }
27       ]
28     }
29   ]
30 }
```

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

B.4 API Delete Task

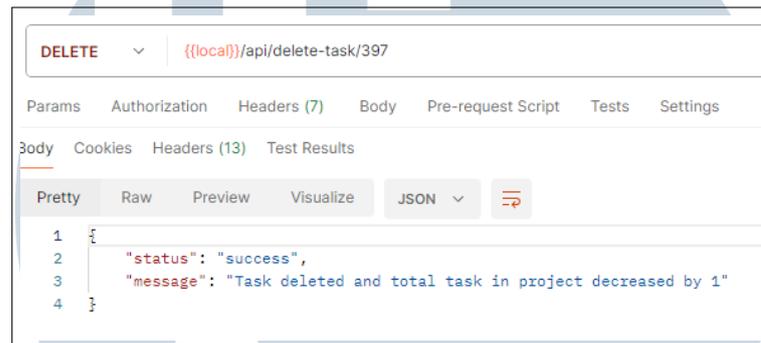
API delete task dibuat untuk menangani permintaan penghapusan suatu tugas. Untuk mengetahui tugas manakah yang ingin dihapus, *API* meminta *input* parameter berupa *id* tugas yang ingin dihapus. *id* tugas tersebut akan dimasukkan pada saat *query* dijalankan untuk mencari data tugas yang dihapus. Untuk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



Gambar 3.44. Flowchart API delete task

Pada Gambar 3.44, 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

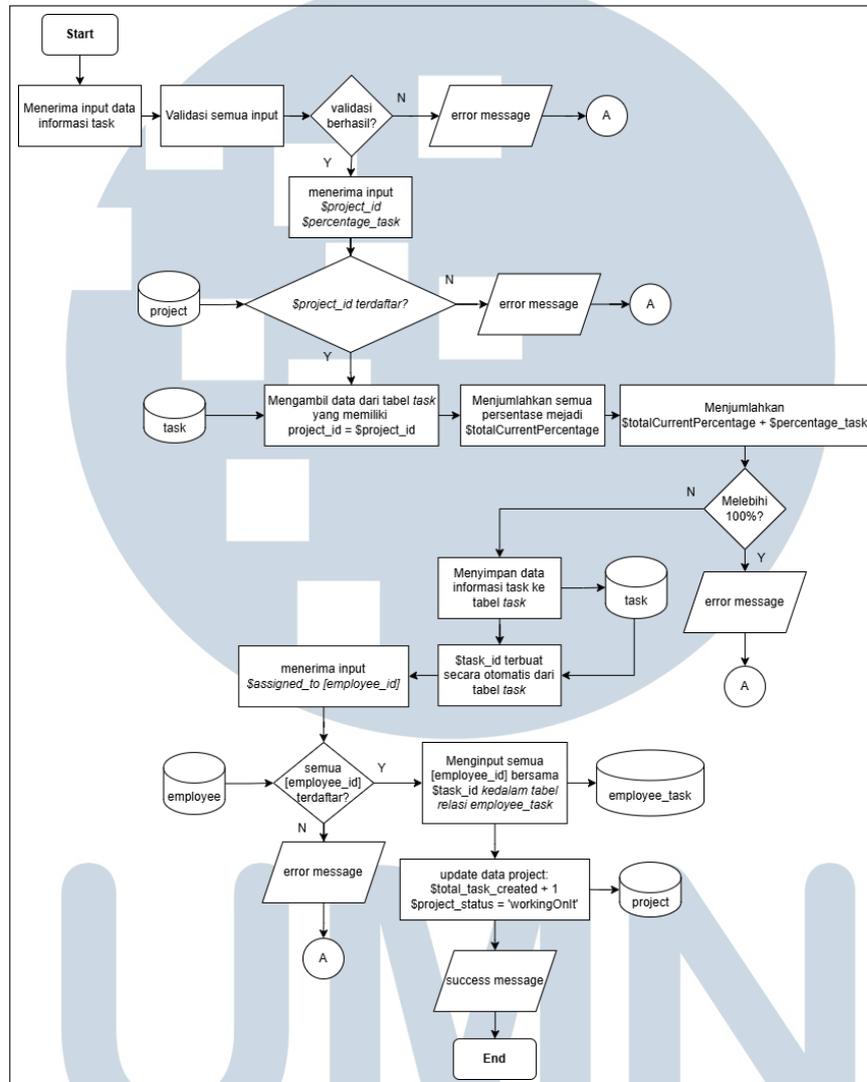
Ketika *Project Manager* membuat tugas dan melakukan *input assigned_to* yang merupakan kumpulan karyawan terlibat. Maka setiap dari karyawan tersebut akan mendapatkan invitasi dari tugas tersebut. Jika pada data terdapat lebih dari satu karyawan yang terlibat, maka hanya salah satu dari mereka dapat 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

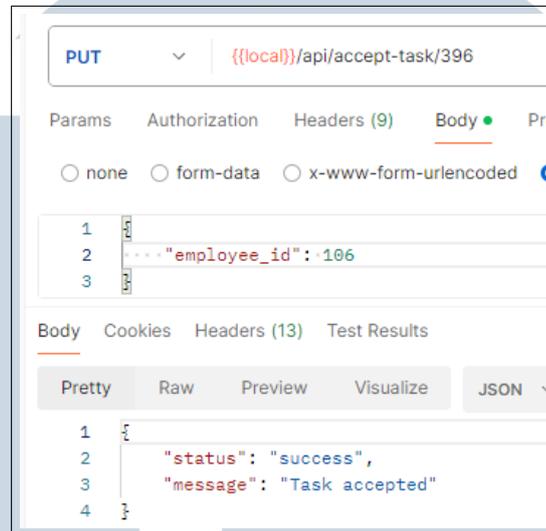
untuk melakukan pengecekan jika data tugas dan data karyawan telah terdaftar atau belum.



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.

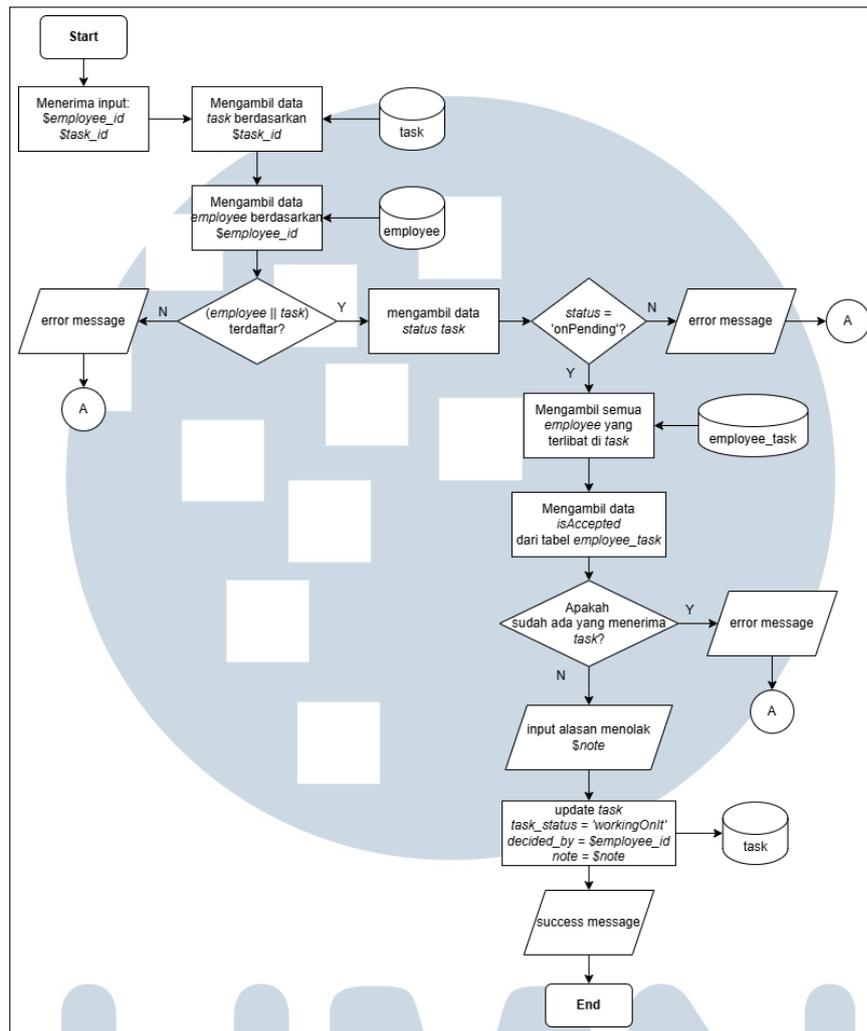


Gambar 3.47. Tampilan percobaan *API accept task*

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.

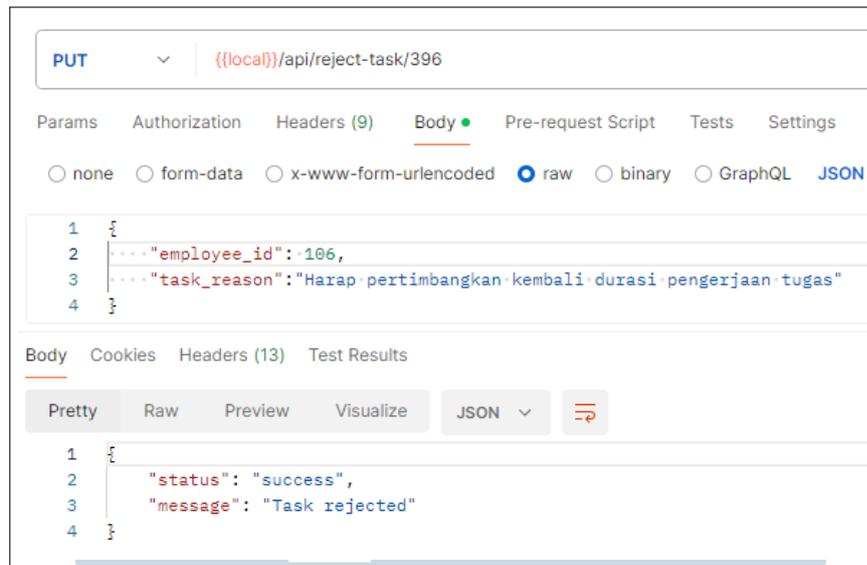
U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A



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 undangan 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

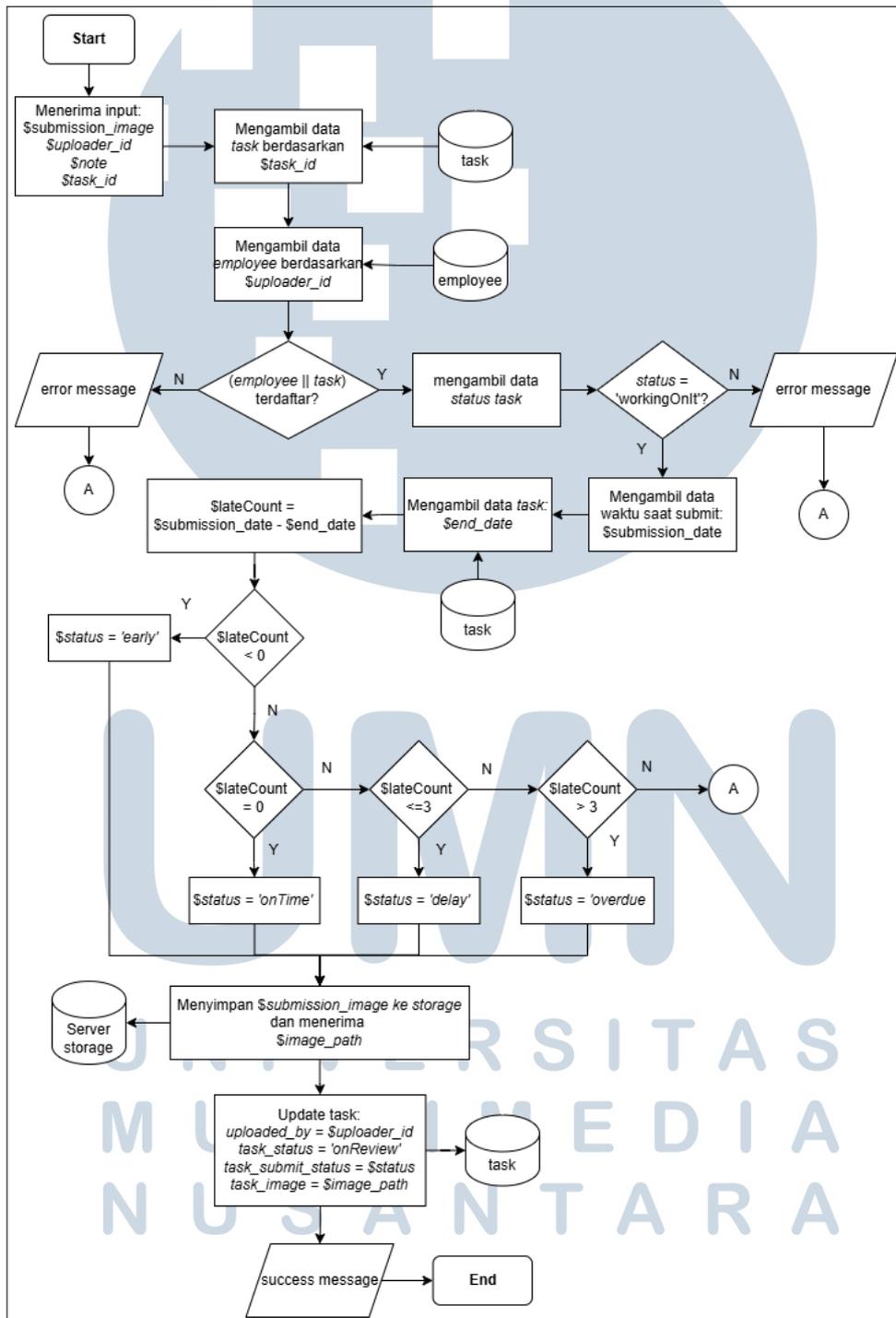
Setelah karyawan menerima invitasi tugas, maka fitur untuk melakukan pengumpulan tugas dapat diakses oleh karyawan. Proses pengumpulan tugas pada *API submit task* dapat diproses jika status tugas telah menjadi *'workingOnIt'*. Pada Gambar 3.50 berupa visualisasi alur proses *API* menangani permintaan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



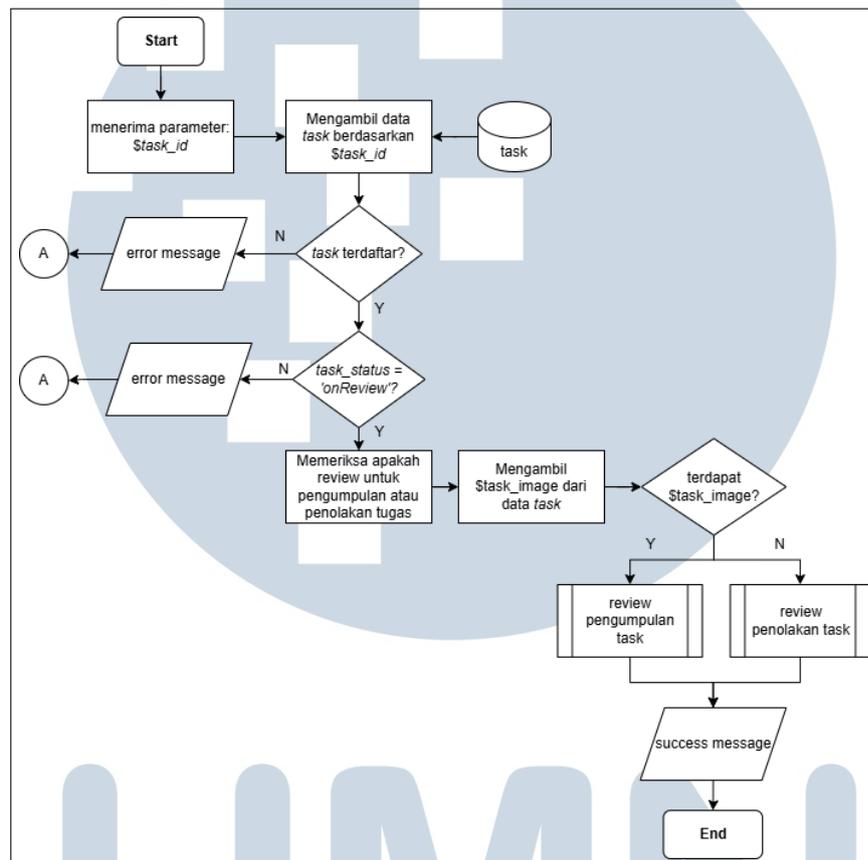
Gambar 3.50. Flowchart API submit task

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.

```

1 PUT (local://api/submit-task/396)
2
3 Params Authorization Headers (9) Body Pre-request Script Tests Settings
4
5 none form-data x-www-form-urlencoded raw binary GraphQL JSON
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
265
```

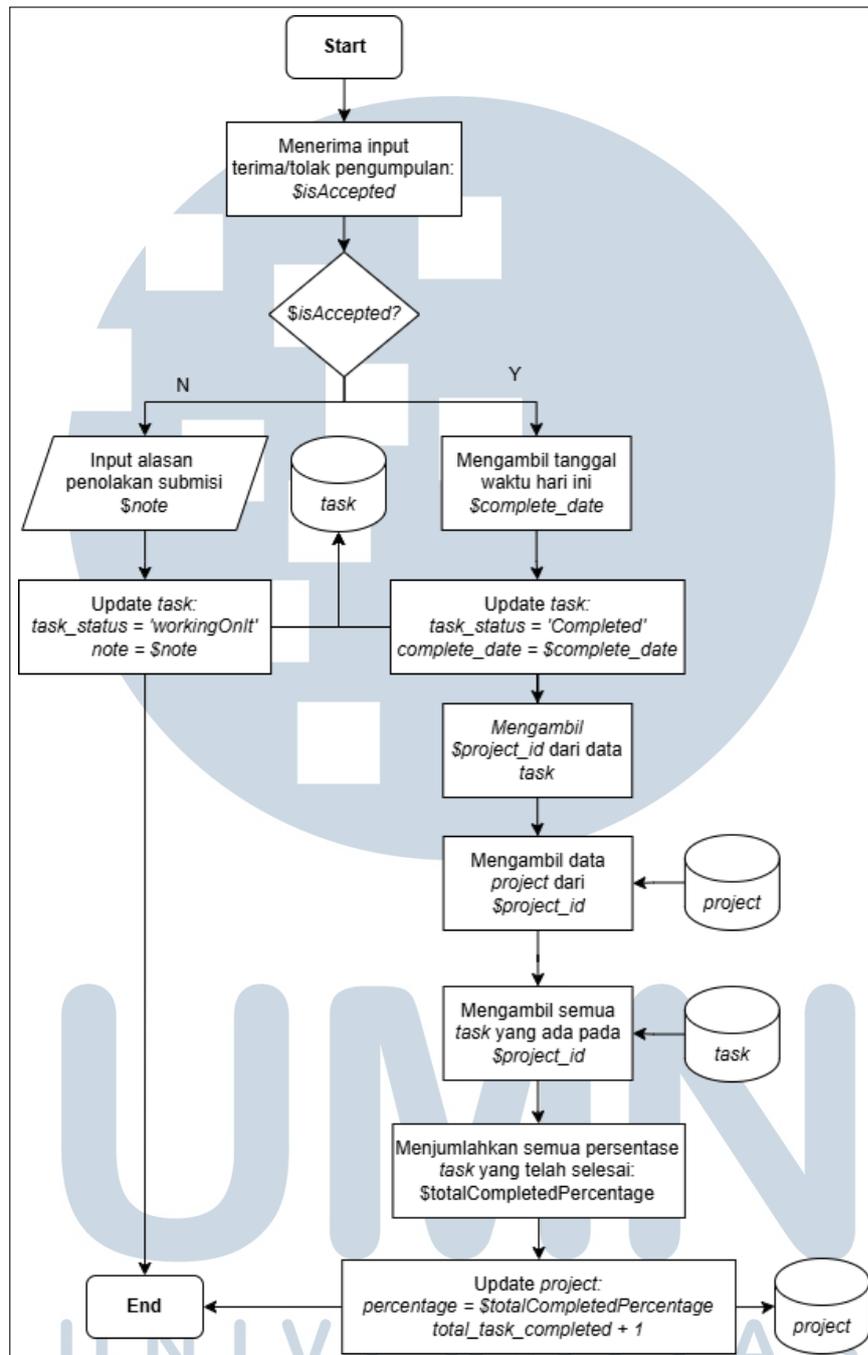
akan memiliki data *task_image*. Atribut tersebut dapat bernilai ketika tugas telah melewati proses pengumpulan tugas yang mengharuskan melakukan *input* pada atribut *task_image*. Proses *API review task* membedakan kedua kasus tersebut dapat dilihat pada Gambar 3.52.



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

B.7.1 Review task yang telah dikumpulkan

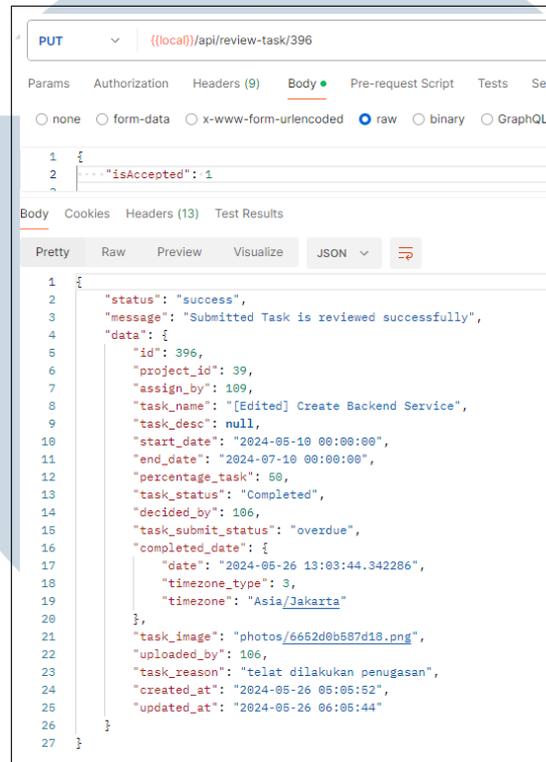
Pada pemanggilan *API review task* untuk melakukan penilaian pada tugas yang dikumpulkan, diperlukan *input* berupa *'isAccepted'* yang digunakan sebagai penanda apakah tugas yang dikumpul tersebut diterima untuk diselesaikan secara penuh atau ditolak untuk dilakukan revisi/pengerjaan kembali kepada 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

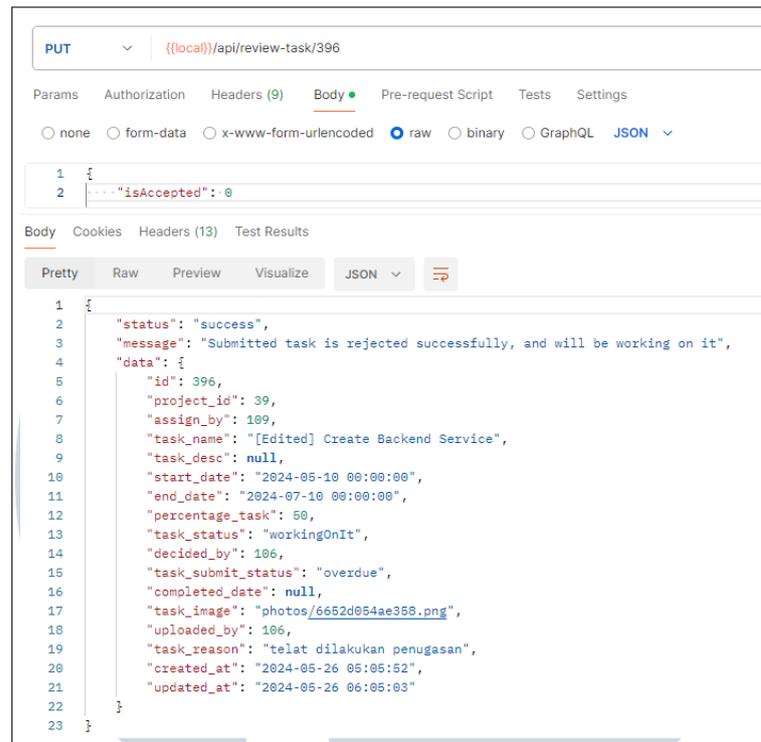
Jika nilai dari 'isAccepted' adalah 1, maka Project Manager bertujuan untuk menerima tugas yang dikumpulkan. Maka API akan melakukan pembaharuan data pada tugas beserta proyek terkait. Data yang diperbaharui pada data tugas adalah seperti tanggal tugas selesai (completed_date) menjadi hari pada saat 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'* agar karyawan dapat mengerjakan tugas dan melakukan pengumpulan kembali. Hasil respons dari penolakan tugas yang dikumpulkan dapat dilihat pada Gambar 3.55.

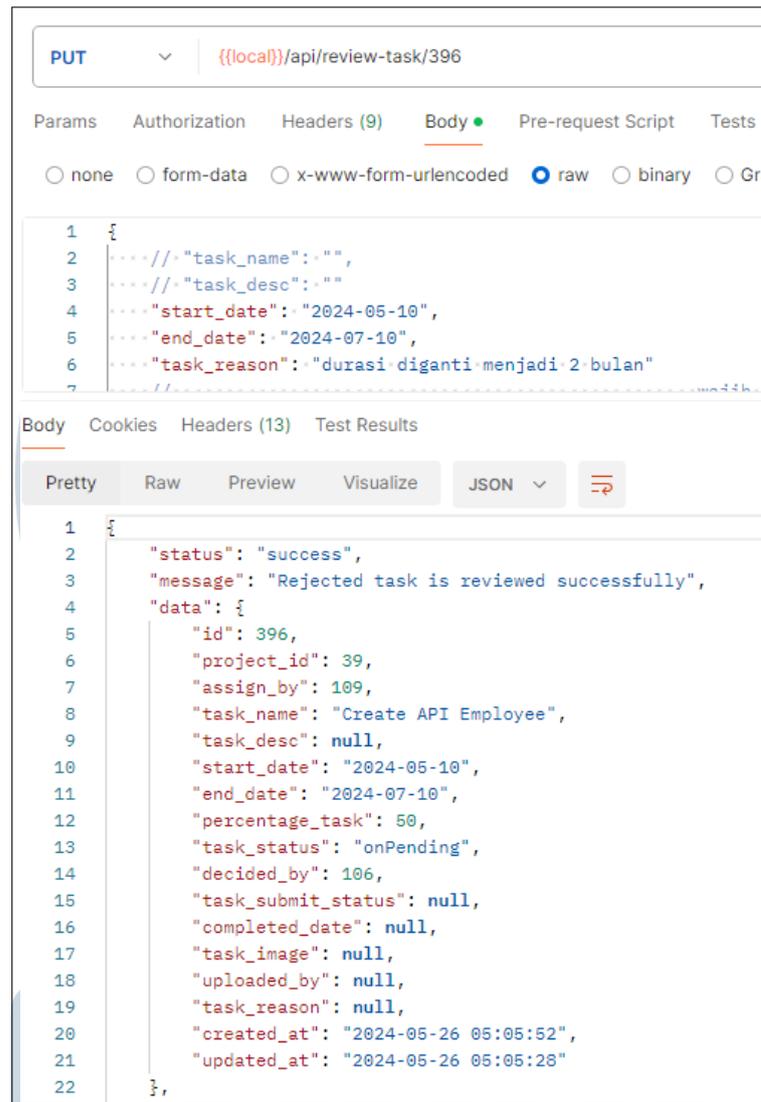


Gambar 3.55. Tampilan percobaan *API review task* jika menolak pengumpulan tugas

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.

UNIVERSITAS
MULTIMEDIA
NUSANTARA

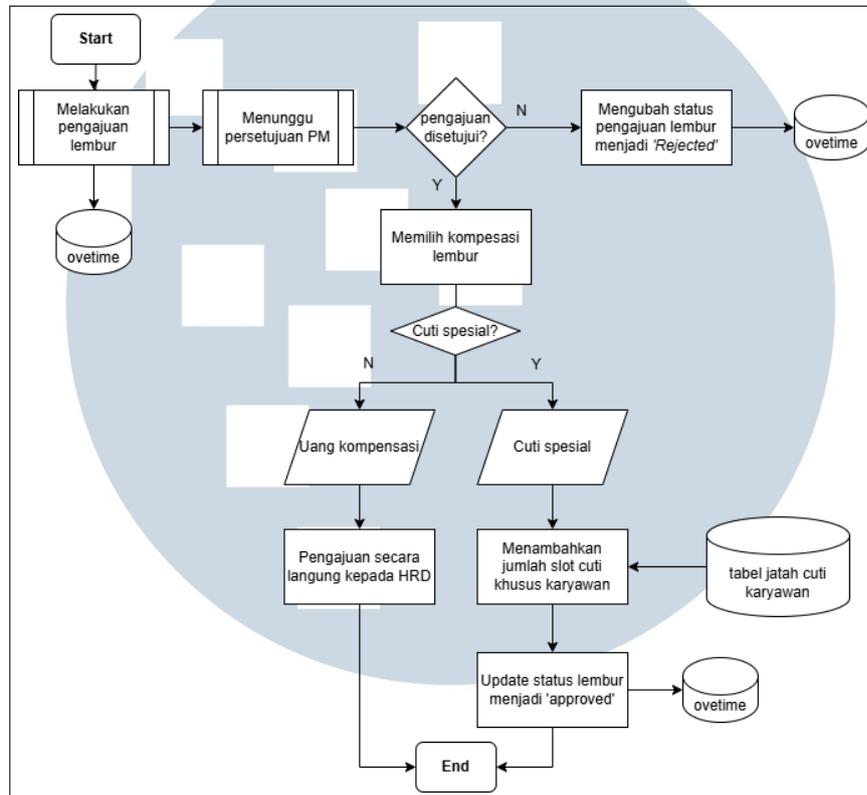


Gambar 3.56. Tampilan percobaan API review task jika ditolak oleh karyawan terkait

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 dikerjakan 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'*.

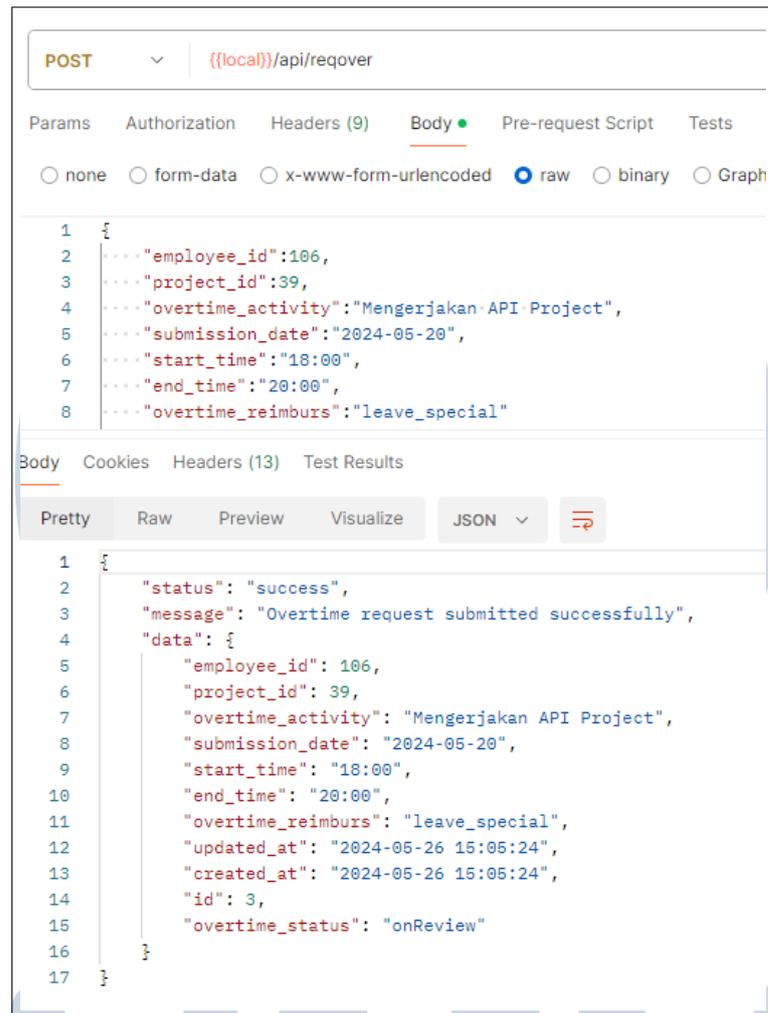
Column Name	Data Type	Constraints
id	bigint(20)	unsigned, primary key
employee_id	bigint(20)	unsigned, foreign key
project_id	bigint(20)	unsigned, foreign key
overtime_activity	varchar(255)	
start_time	time	
end_time	time	
submission_date	date	
overtime_reimburs	enum	('leave_special', 'money')
overtime_reason	varchar(255)	
overtime_status	enum	('Rejected', 'onReview', 'Approved', 'Used')
approved_date	datetime	
expired_date	datetime	
created_at	timestamp	
updated_at	timestamp	

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 ajukan 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.

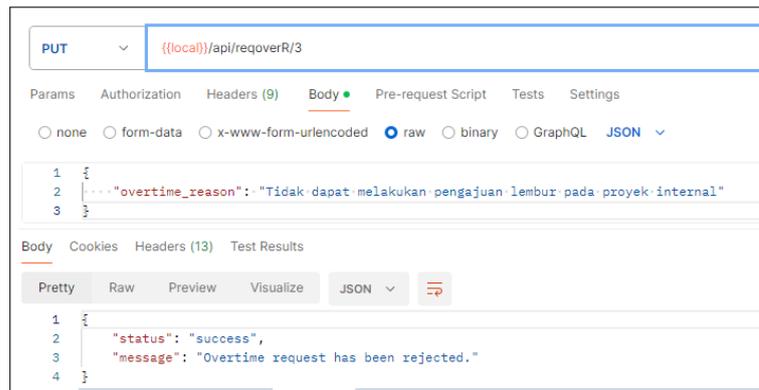




Gambar 3.59. Tampilan percobaan API create overtime request

B. API Reject Overtime Request

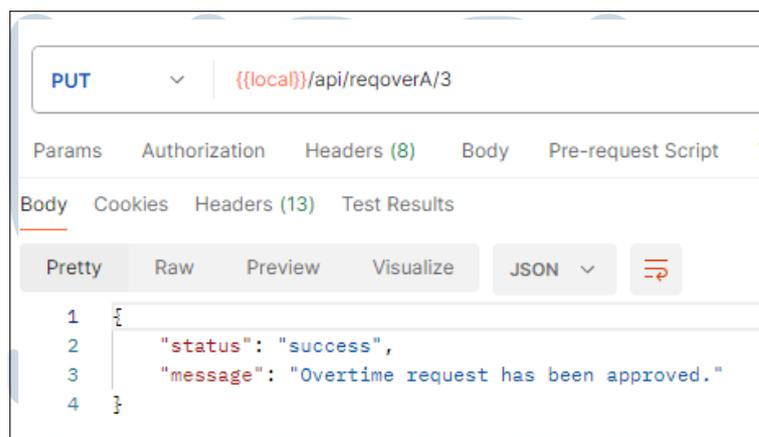
Pada tampilan *admin* akan muncul semua data pengajuan lembur dari karyawan. API ini digunakan jika *admin* tidak ingin menyetujui pengajuan lembur dari salah satu karyawan. API akan meminta *input* berupa alasan dari penolakan pengajuan lebur, kemudian API akan mengubah status dari pengajuan lembur 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.

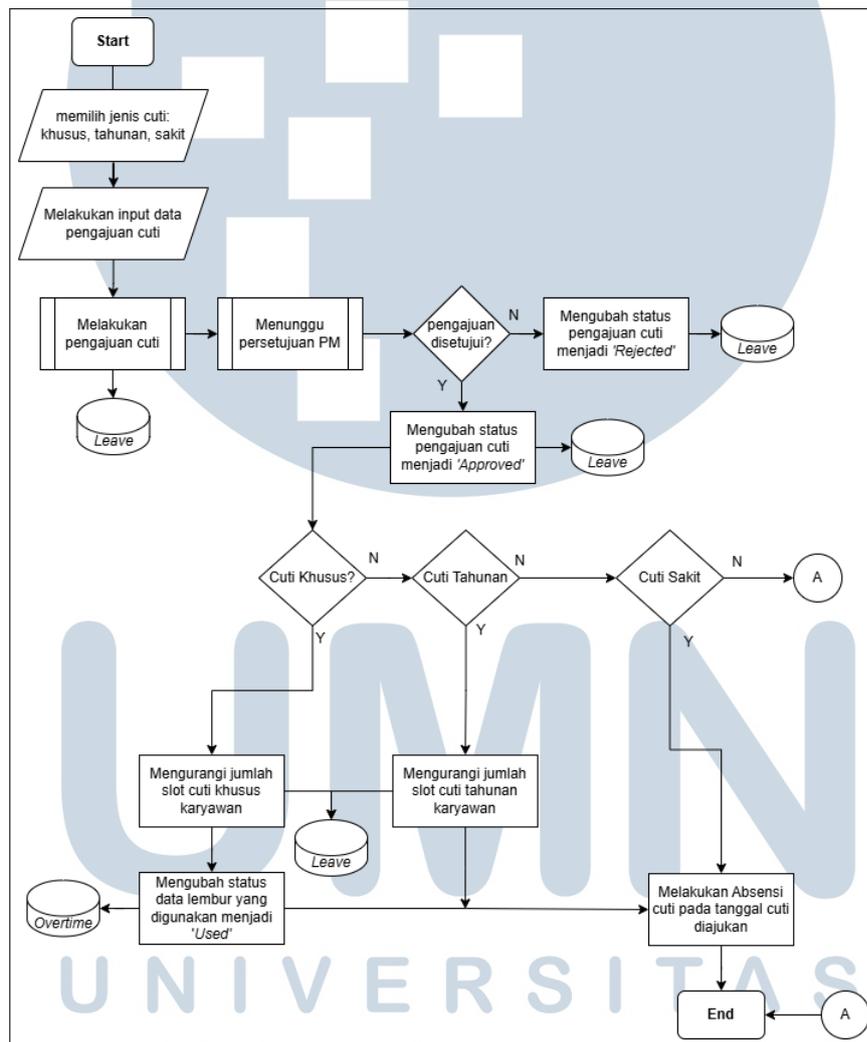


Gambar 3.61. Tampilan percobaan *API approve overtime request*

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.

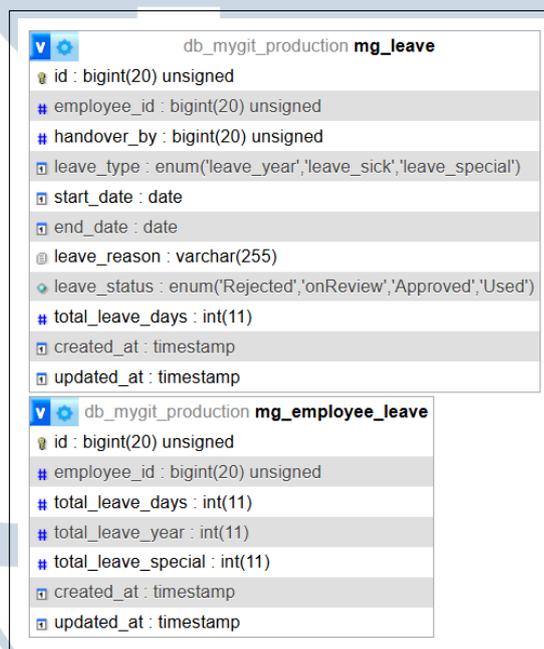


Gambar 3.62. Flowchart untuk alur pengajuan cuti karyawan

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.

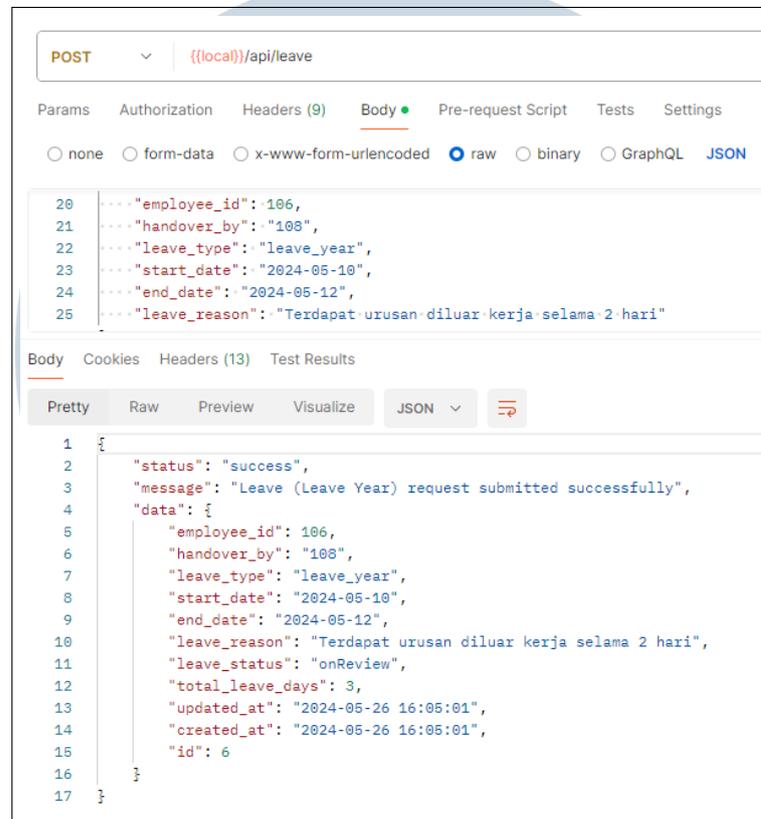


Gambar 3.63. Atribut basis data dari tabel *leave*

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 ajukan akan masuk kepada tampilan *admin* untuk dilakukan penolakan atau pun persetujuan. Jika data

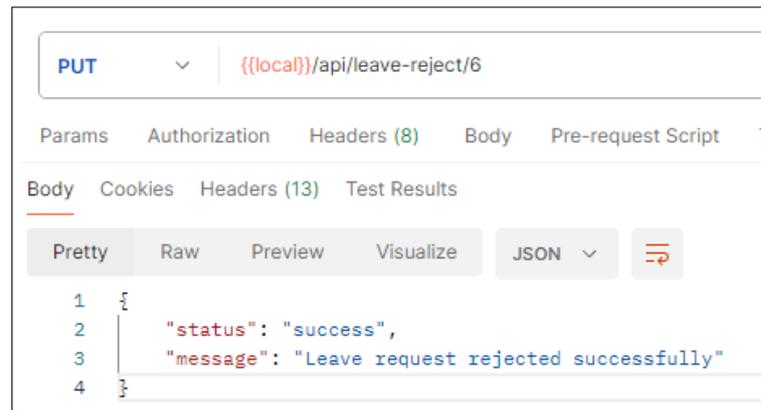
pengajuan cuti berhasil diproses, maka tampilan dari respons *API* adalah seperti pada Gambar 3.64.



Gambar 3.64. Tampilan percobaan *API create leave request*

B. API Reject Leave Request

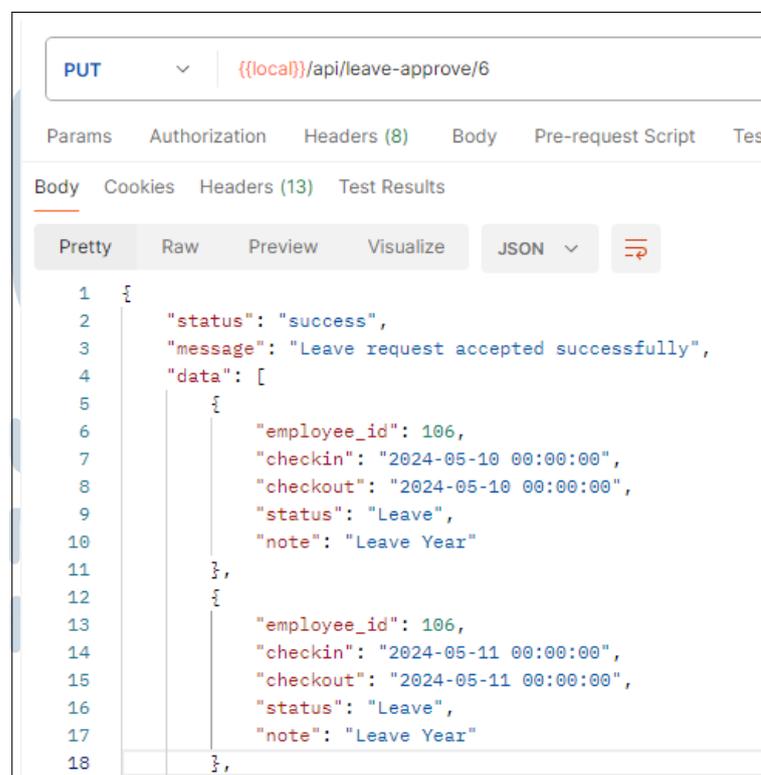
Pada tampilan *admin* akan muncul semua data pengajuan cuti dari karyawan. *API* ini digunakan jika *admin* tidak ingin menyetujui pengajuan cuti dari salah satu karyawan kemudian *API* akan mengubah status dari pengajuan cuti menjadi 'Rejected'. Jika data pengajuan cuti berhasil diproses maka respons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

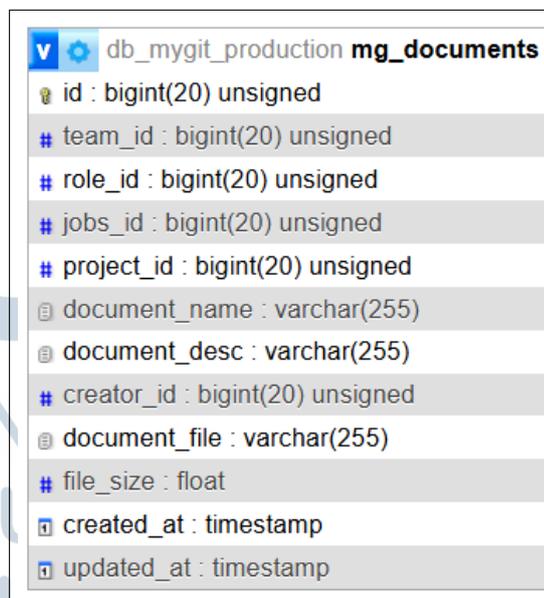


Gambar 3.66. Tampilan percobaan *API approve leave request*

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.

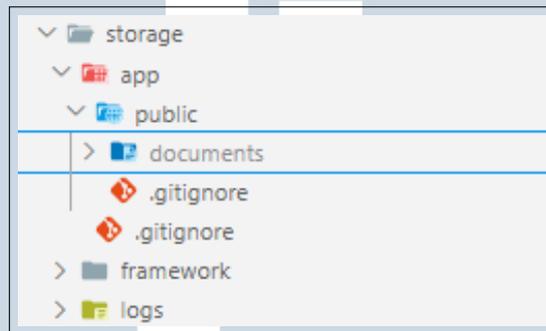


Attribute	Type
id	bigint(20) unsigned
team_id	bigint(20) unsigned
role_id	bigint(20) unsigned
jobs_id	bigint(20) unsigned
project_id	bigint(20) unsigned
document_name	varchar(255)
document_desc	varchar(255)
creator_id	bigint(20) unsigned
document_file	varchar(255)
file_size	float
created_at	timestamp
updated_at	timestamp

Gambar 3.67. Atribut basis data dari tabel *document*

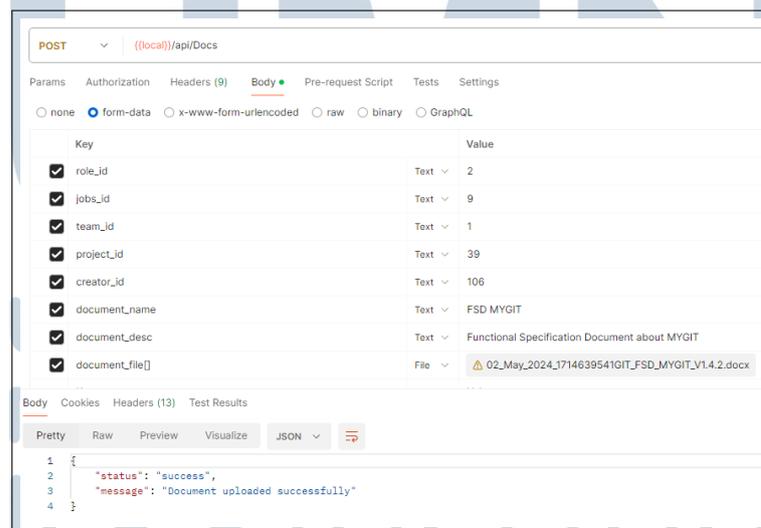
A. API Create Document

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.

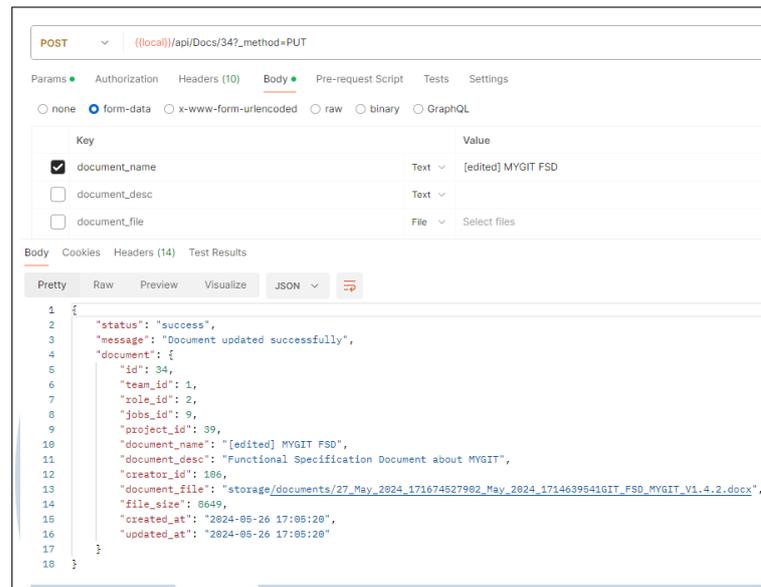
B. API Update Document

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 bawaan 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.

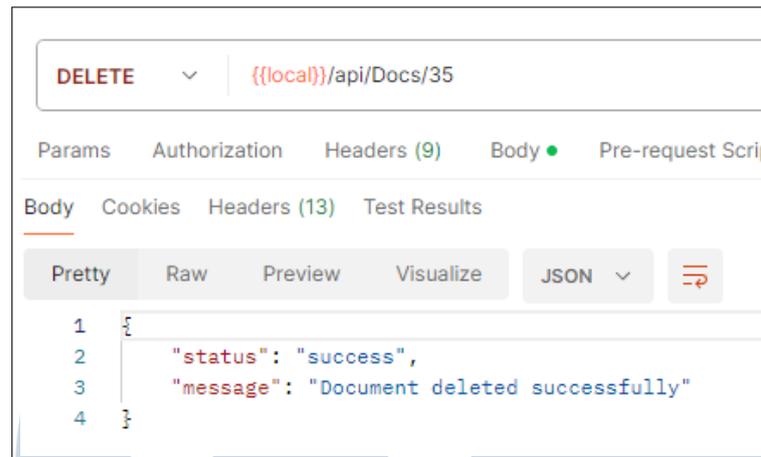


Gambar 3.71. Tampilan percobaan API update document

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 disebabkan 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 Mengambil 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.

```

1 {
2   "status": "success",
3   "documents": [
4     {
5       "id": 34,
6       "team_id": 1,
7       "role_id": 2,
8       "jobs_id": 9,
9       "project_id": 39,
10      "document_name": "[Edited] MYGIT FSD",
11      "document_desc": "Functional Specification Document about MYGIT",
12      "creator_id": 186,
13      "document_file": "storage/documents/27_May_2024_171674527992_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
14      "file_size": 8649,
15      "created_at": "2024-05-26 17:05:28",
16      "updated_at": "2024-05-26 17:05:28"
17     },
18     {
19       "id": 36,
20       "team_id": 1,
21       "role_id": 2,
22       "jobs_id": 9,
23       "project_id": 39,
24       "document_name": "82_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
25       "document_desc": "guide for admin and TC",
26       "creator_id": 186,
27       "document_file": "storage/documents/27_May_2024_171674552382_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
28       "file_size": 8649,
29       "created_at": "2024-05-26 17:05:23",
30       "updated_at": "2024-05-26 17:05:23"
31     },
32     {
33       "id": 37,
34       "team_id": 1,
35       "role_id": 2,
36       "jobs_id": 9,
37       "project_id": 41,
38       "document_name": "82_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
39       "document_desc": "Grafana FSD",
40       "creator_id": 186,
41       "document_file": "storage/documents/27_May_2024_171674527992_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
42       "file_size": 8649,
43       "created_at": "2024-05-26 17:05:28",
44       "updated_at": "2024-05-26 17:05:28"
45     }
46   ]
47 }

```

Gambar 3.73. Tampilan percobaan API get all document

D.2 Mengambil 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.

```

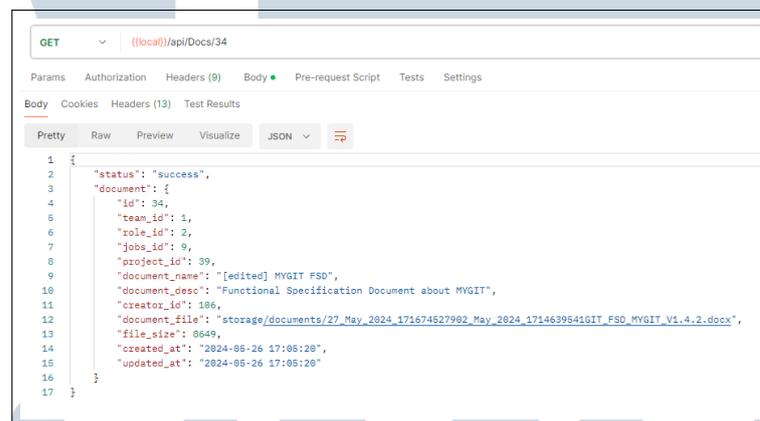
1 {
2   "status": "success",
3   "document": {
4     "id": 34,
5     "team_id": 1,
6     "role_id": 2,
7     "jobs_id": 9,
8     "project_id": 39,
9     "document_name": "[Edited] MYGIT FSD",
10    "document_desc": "Functional Specification Document about MYGIT",
11    "creator_id": 186,
12    "document_file": "storage/documents/27_May_2024_171674527992_May_2024_1714639541GIT_FSD_MYGIT_V1.4.2.docx",
13    "file_size": 8649,
14    "created_at": "2024-05-26 17:05:28",
15    "updated_at": "2024-05-26 17:05:28"
16  }
17 }

```

Gambar 3.74. Tampilan percobaan API get document by id

D.3 Mengambil semua data dokumen per proyek

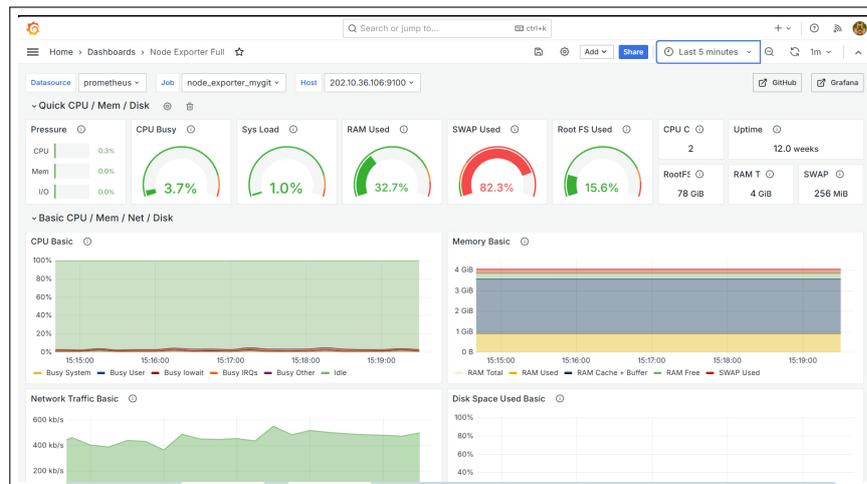
API ini digunakan ketika ingin mendapatkan dokumen yang ada pada salah satu proyek. *API* ini dapat membantu pada sisi *frontend* saat ingin melihat detail pada salah satu proyek dan membantu *admin* dalam melakukan monitor terhadap dokumen-dokumen yang ada pada setiap proyek. *API* ini mendapatkan data dokumen per proyek dengan memanfaatkan *id* proyek yang didapatkan pada parameter untuk dimasukkan 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 adalah 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 aplikasi 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.

