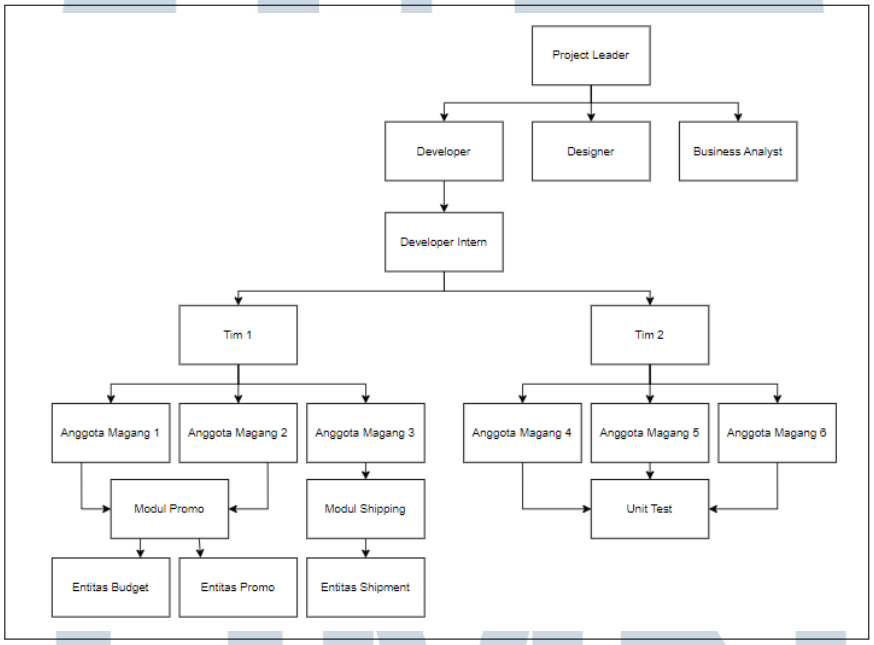
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Kedudukan selama pelaksanaan kegiatan magang di PT Cranium Royal Aditama adalah sebagai *fullstack developer intern*. Berikut struktur dalam pengerjaan proyek ERP yang dapat dilihat pada Gambar 3.1.



Gambar 3.1. Struktur Proyek ERP Cranium

Segala proses pengerjaan proyek selama pelaksanaan magang berada di bawah bimbingan Bapak Sugito selaku supervisi, *project leader*, dan *vice president engineering* dalam pengembangan proyek sistem ERP. *Project leader* mengatur tiga divisi, yaitu divisi *developer*, *designer*, dan *business analyst*. Semua anggota magang berada di bawah divisi *developer* bernama *developer intern*. Terdapat enam anggota magang yang terbagi menjadi dua kelompok dan semua anggota magang diberikan tugas untuk mengelola proyek ERP. Kedudukan yang diduduki selama proses magang berlangsung terdapat dalam tim satu sebagai ’anggota magang 2’ yang mengelola entitas Promo dalam modul Promo.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama pelaksanaan kegiatan magang di PT Cranium Royal Aditama, antara lain:

1. Menguraikan isi *import* .*
   Semua modul yang menggunakan impor bintang harus diubah dengan cara diuraikan, misalnya saat mengimpor anotasi-anotasi yang digunakan untuk proses CRUD di modul BillOfMaterialItemController.java yang mulanya bertuliskan import id.cranium.erp.master.security.annotation.* diubah menjadi import id.cranium.erp.master.security.annotation.IsMasterBillOfMaterialItemCreate. Begitupun dengan annotasi IsMasterBillOfMaterialItemDelete, IsMasterBillOfMaterialItemRead, dan IsMasterBillOfMaterialItemUpdate. Hal ini dilakukan untuk menerapkan *clean code*. Terdapat 62 impor .* yang harus diubah. Tugas ini dilakukan dalam sebuah tim yang terdiri dari tiga anggota.

2. Memeriksa validasi DTO dan *error message* pada modul Accounting dan Warehouse
   DTO yang perlu dicek validasinya hanyalah DTO untuk proses *create* dan *update* saja. Semua *field* yang berada dalam DTO harus memiliki validasi yang telah ditentukan dalam *database*. Jika terdapat *field* yang belum mempunyai validasi, maka validasi yang sesuai harus ditambahkan untuk *field* tersebut. Lalu, dalam validasi ini terdapat *error message* yang berisi pesan kesalahan ketika suatu *field* memiliki isi yang tidak sesuai dengan validasi atau kriteria yang telah ditentukan. *Error message* ini dibuat menggunakan dua bahasa, yakni bahasa Inggris dan bahasa Indonesia. *Error message* kemudian disimpan dalam suatu *file messages validator* di dalam folder 'resources'.

3. Melengkapi API untuk proses CRUD pada entitas Promo
   API atau *Application Programming Interface* adalah sekumpulan protokol yang dapat menjadi sarana untuk dua komputer berkomunikasi satu sama lain. CRUD merupakan singkatan dari *Create, Read, Update, Delete*. Pembuatan CRUD ini dilakukan pada dua modul yang bernama modul 'promo-dto' dan 'promo-service' untuk entitas Promo. Proses pengerjaan tugas ini menggunakan IDE IntelliJ IDEA.

4. Melakukan uji *Unit Test* untuk *back-end*

   Setelah pembuatan API CRUD selesai, dilakukan pengujian untuk melihat apakah proses CRUD sudah berhasil berjalan dengan baik atau belum. Proses pengujian ini dilakukan dalam suatu *file* di IntelliJ IDEA yang disebut *Unit Test*. *Unit Test* mencakup dua kondisi yang akan dijalankan, yakni saat API berjalan dengan sukses dan saat API mengalami kegagalan. Pada proses pembuatan *unit test*, terdapat *file* Groovy yang berisi *request* dan *response* yang diharapkan saat melakukan proses uji data. Jika terdapat *error*, maka akan diperbaiki sampai proses CRUD berjalan dengan sempurna.

3.3 Uraian Pelaksanaan Magang

Berikut uraian pelaksanaan saat kerja magang di PT Cranium Royal Aditama.

3.3.1 Aktivitas Setiap Minggu

Pekerjaan yang dilakukan tiap minggu selama pelaksanaan magang di Cranium dapat dilihat pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu

Minggu ke -	Pekerjaan yang dilakukan
1	Proses *training* bersama supervisi untuk mengenal struktur ERP secara garis besar, menginstall *software* yang dibutuhkan, serta mempelajari *query* dan *clean code* bersama mentor
2	Mempelajari *query* lebih dalam bersama mentor dan tim, mencoba proses CRUD (*create, read, update, dan delete*) di Postman, dan *cloning template* ERP yang diberikan oleh mentor
3	Mempelajari dan mencoba membuat class baru bernama Order dalam *template* ERP yang diberikan oleh mentor, *compile Spring Boot* dan setiap modul, dan mempelajari proses CRUD di Postman
4	Melakukan perubahan pada isi field status dan penambahan field pada class Order seperti totalPrice dan quantity, mencoba membuat Unit Test untuk entitas Order, dan mempelajari proses CRUD pada Postman

Tabel 3.1 Pekerjaan yang dilakukan tiap minggu (lanjutan)

Minggu ke -	Pekerjaan Yang Dilakukan
5	Mempelajari *Annotation*, *Validation*, dan alur data masuk keluar dalam template ERP bersama mentor
6	Mempelajari alur data masuk keluar dalam ERP dan Postman
7	Mempelajari *Throw Exception*, alur Unit Test *backend*, dan alur *frontend* dalam ERP
8	Mempelajari *frontend* ERP dan melanjutkan pembuatan class Order untuk *backend* ERP
9	Mencoba *cloning* class Order untuk *backend* ERP dan membuat class Order untuk *frontend* ERP
10 - 11	Mempelajari alur *frontend* ERP bersama mentor dan tim
12	Proses *training* pembuatan dan alur Unit Test untuk *frontend* ERP bersama mentor
13	Mempelajari alur kode *backend* dan *frontend* ERP bersama tim
14	Mencoba *cloning template* ERP baru yang diberikan mentor, mengubah import .* pada modul-modul yang telah dibagi dalam tim, memeriksa validasi dan *error message* pada modul Accounting dan Warehouse, serta melakukan *testing* SIT website Cranium di modul Warehouse, Customer Type, Vehicle, Item Brand, dan Supplier Type
15	Memeriksa serta menambahkan jika ada validasi dan *error message* yang kurang pada modul Accounting dan Warehouse, *compile* modul untuk *push* ke Github, dan memperbaiki *error* yang terjadi saat melakukan *push* ke Github
16	Melengkapi proses CRUD, mengecek validasi dan *error message* pada entitas Promo
17	Membuat Unit Test untuk *backend* entitas Promo
18	Membenarkan *error* ketika mencoba CRUD di Postman
19	Pengenalan terhadap cara pembuatan *frontend* untuk entitas Promo bersama mentor

3.3.2 Software yang Digunakan

Berikut beberapa perangkat lunak yang digunakan dalam proses pengembangan *backend* sistem ERP selama pelaksanaan magang:

1. *Framework Java Spring Boot*
   Spring Boot adalah sebuah *framework open-source* berbasis Java yang menawarkan kemudahan dan percepatan dalam pengembangannya, sehingga umumnya digunakan dalam pembuatan aplikasi berbasis *Spring* [6]. *Framework* ini dapat secara otomatis mengonfigurasi aplikasi berdasarkan dependensi yang telah ada atau disediakan.

2. IntelliJ IDEA
   IntelliJ IDEA adalah sebuah IDE (*Integrated Development Environment*) yang dikembangkan oleh JetBrains. IntelliJ IDEA mendukung pengembangan *software* yang menggunakan Java, Groovy, dan Kotlin [7]. Maka dari itu, IDE ini direkomendasikan oleh supervisi untuk digunakan dalam pengembangan sistem *backend* ERP.

3. Postman
   Postman adalah sebuah platform kolaborasi yang dapat digunakan untuk mengembangkan, menggunakan, dan menguji API (*Application Programming Interface*) [8]. Aplikasi ini memungkinkan pengembang dalam membuat permintaan HTTP GET, POST, PATCH, DELETE, dan lainnya ke berbagai *endpoint* API. Postman juga menyediakan fitur kolaborasi yang memungkinkan pengembang untuk berkerja sama secara tim dalam melakukan pengembangan dan pengujian dokumentasi API.

4. PgAdmin
   PgAdmin adalah sebuah perangkat lunak manajemen basis data PostgreSQL yang bersifat *open-source* dan sangat populer. Perangkat lunak ini menyediakan antarmuka grafis pengguna yang dapat memudahkan pengguna dalam melakukan berbagai aktivitas terkait basis data PostgreSQL seperti membuat, menghapus, dan mengedit tabel, serta mengeksekusi query SQL. PgAdmin dapat digunakan di Linux, Windows, Unix, dan macOS [9].

5. Github
   Github adalah sebuah platform pengembangan perangkat lunak yang

dapat memungkinkan pengembang untuk bekerja sama dalam mengelola, menyimpan, dan melacak perubahan kode. Pengembang dapat membuat repositori untuk menyimpan semua kode sumber proyek mereka yang dapat dikendalikan apakah bersifat publik atau privat. Github menggunakan sistem kontrol Git untuk melacak perubahan kode dalam repositori, sehingga pengembang dapat melihat riwayat perubahan, membuat *branch*, dan melakukan penggabungan *branch* yang berbeda-beda. Selama proses kerja magang, semua tim menggunakan Github sebagai platform kolaborasi dalam pengembangan *backend* sistem ERP.

3.3.3 Validasi DTO dan Error Message Entitas Promo

Validasi DTO merupakan proses memvalidasi data-data yang akan diterima atau dikirim melalui DTO ke satu komponen aplikasi maupun ke komponen aplikasi lainnya sesuai dengan aturan bisnis yang telah ditentukan. Proses validasi ini dapat menjaga integritas data dan memastikan aplikasi tidak akan menerima data-data yang tidak sesuai dengan kriteria yang dapat menyebabkan kesalahan atau *error*. Setiap *field* dalam DTO memiliki validasi yang telah ditentukan dalam *database*. Masing-masing validasi *field* dalam DTO dan *database* harus sinkron atau sama, sehingga program dapat berjalan dengan lancar.

Contoh validasi yang digunakan pada DTO untuk membangun entitas Promo dapat dilihat pada modul bernama PromoCreateDto dalam Kode 3.1.

```
1  public class PromoCreateDto {
2      @NotNull(message = "{promo.promoname.notnull}")
3      @NotEmpty(message = "{promo.promoname.notempty}")
4      @Size(max = 100, message = "{promo.promoname.size}")
5      private String promoName;
6      @NotNull(message = "{promo.status.notnull}")
7      private Short status;
8      @NotNull(message = "{budget.totalbudget.notnull}")
9      private Long totalBudget;
10 }
```

Kode 3.1: Validasi DTO untuk metode Create

Pada kode PromoCreateDto terdapat tiga validasi, yaitu:

1. @NotNull

   Anotasi `@NotNull` dimiliki oleh *field* 'promoName', 'status', dan 'totalBudget'. Anotasi ini memastikan bahwa *field* tidak boleh bernilai 'null'

namun nilai seperti *string* kosong (””) masih dianggap valid. Jika *field* yang memiliki anotasi ini diberi nilai ’null’, maka proses validasi akan gagal dan sistem akan memberikan pesan kesalahan yang telah didefinisikan dalam ’promo.promoname.notnull’.

2. @NotEmpty
   Anotasi `@NotEmpty` dimiliki oleh *field* ’promoName’ yang digunakan untuk memastikan bahwa *field* tidak bernilai ’null’ dan tidak kosong (panjang *string* tidak nol). Jika *field* yang memiliki validasi ini bernilai ’null’ atau kosong, maka validasi akan gagal dan mengeluarkan pesan kesalahan dalam ’promo.promoname.notempty’. Anotasi ini lebih cocok digunakan pada tipe data *String* atau array.

3. @Size
   Anotasi `@Size` dimiliki oleh *field* ’promoName’. Anotasi ini digunakan untuk memastikan bahwa panjang *string* berada dalam batas minimum dan/atau maksimum yang ditentukan dalam *database*. Kata ’max = 100’ memiliki makna panjang maksimum *string* untuk *field* ’promoName’ adalah 100 karakter. Jika ’promoName’ diisi dengan *string* yang panjangnya melebihi 100 karakter, maka validasi akan gagal dan mengeluarkan pesan kesalahan dalam ’promo.promoname.size’.

Isi dari pesan kesalahan untuk masing-masing anotasi pada kelas PromoCreateDto dapat dilihat dalam Kode 3.2 dan Kode 3.3.

```
promo.promoname.notnull = Nama promo harus ada isi
promo.promoname.notempty = Nama promo tidak boleh kosong
promo.promoname.size = Panjang nama promo tidak boleh melebihi 100 karakter
promo.status.notnull = Status harus ada isi
budget.totalbudget.notnull = Total budget harus ada isi
```

Kode 3.2: Error Message menggunakan bahasa Indonesia

```
promo.promoname.notnull = Promo name must be present
promo.promoname.notempty = Promo name must not be empty
promo.promoname.size = Promo name must not exceed 100 characters
promo.status.notnull = Status must be present
budget.totalbudget.notnull = Total budget must be present
```

Kode 3.3: Error Message menggunakan bahasa Inggris

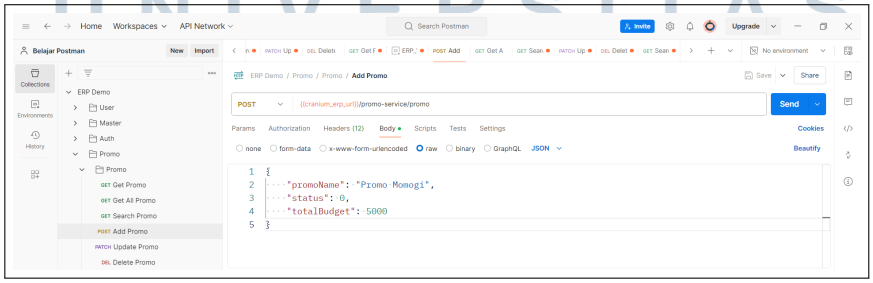
Error message atau pesan kesalahan adalah pesan yang akan diberikan kepada pengguna saat memberikan *input* data yang tidak memenuhi kriteria atau aturan yang telah ditentukan. Pesan kesalahan ini sangat bermanfaat karena dapat membantu pengguna untuk mengetahui dan memahami letak kesalahan mereka, sehingga mampu memperbaikinya sesuai dengan kriteria yang berlaku. Semua pesan kesalahan dalam Cranium ERP didefinisikan menggunakan dua bahasa, yaitu bahasa Inggris dan bahasa Indonesia yang akan disimpan di modul `validator\_en.properties` dan `validator\_id.properties` dalam *folder resources*. Contoh isi pesan kesalahan yang menggunakan bahasa Indonesia dapat dilihat pada Kode 3.2, sedangkan contoh isi pesan kesalahan yang menggunakan bahasa Inggris dapat dilihat pada Kode 3.3.

3.4 Alur Backend Entitas Promo

Backend adalah bagian dari sistem perangkat lunak yang bertanggung jawab terhadap logika, sistem operasi, dan manajemen data. Proses *backend* akan menangani berbagai permintaan yang datang dari klien seperti permintaan POST, PATCH, GET, dan DELETE serta mengembalikan respons yang sesuai kepada klien. Permintaan klien ini mengatur proses CRUD seperti *create, read, update, delete* untuk berinteraksi dengan *database*. Setiap anggota tim magang di Cranium harus bisa memahami alur jalan dari setiap kode proses CRUD, sehingga mampu berpartisipasi dalam proses pembuatan kode untuk CRUD berbagai entitas.

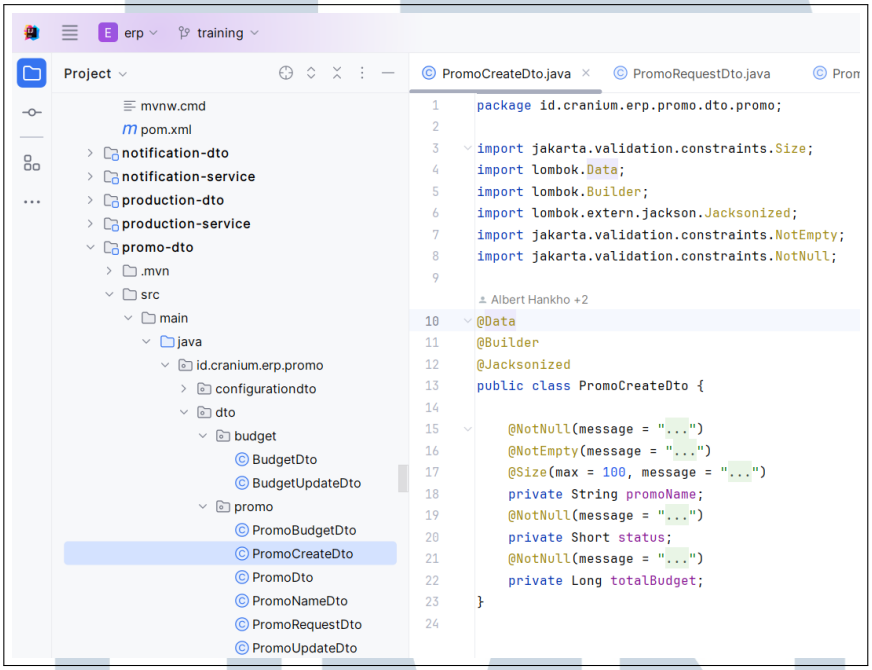
3.4.1 Alur Create

Create merupakan operasi dasar yang digunakan untuk menambahkan data baru ke dalam *database*. Berikut awal alur *create* yang dapat dilihat pada Gambar 3.2.



Gambar 3.2. Permintaan API HTTP POST dari klien (Postman)

Proses *create* berawal dari klien yang mengirim permintaan HTTP POST ke server. HTTP atau *HyperText Transfer Protocol* merupakan suatu protokol yang digunakan untuk mengatur komunikasi antara klien dan server. Postman digunakan dalam proses pengembangan *backend* ERP sebagai klien yang mengirimkan permintaan HTTP dan respons kepada pengguna. Data-data yang akan dikirimkan untuk membuat sebuah entitas Promo telah ditetapkan di dalam *file* DTO bernama PromoCreateDto. Isi dari PromoCreateDto dapat dilihat pada Gambar 3.3.



Gambar 3.3. DTO Create Promo

PromoCreateDto berisi deklarasi tiga *field* yang akan ditampilkan saat proses *create* terjadi. Masing-masing *field* diberikan validasi sehingga data yang diberikan oleh pengguna harus sesuai dengan kriteria yang telah ditetapkan. Jika data yang diberikan pengguna tidak sesuai dengan kriteria validasi, maka pesan kesalahan akan ditampilkan. Setelah melakukan permintaan HTTP POST, Controller akan menerima permintaan tersebut dan melakukan validasi data. Isi dari *file* Controller yang bernama PromoController dapat dilihat pada Kode 3.4.

```
@PostMapping(value = "/promo", headers = "X-Api-Version=1")
    @IsPromoPromoCreate
    @ResponseStatus(value = HttpStatus.CREATED)
    @LogExecutionTime
    public PromoDto createPromo(@Validated @RequestBody
   PromoCreateDto promoCreateDto) throws DataNotFoundException {
```

```
        return promoService.createPromo(promoCreateDto);
    }
```

Kode 3.4: Metode Create dalam file Controller

Anotasi `@PostMapping(value = "/promo"` memiliki makna bahwa metode ini menangani permintaan HTTP POST ke URL '/promo', sedangkan kode `headers = "X-Api-Version=1"` artinya permintaan harus memiliki *header* 'X-Api-Version' dengan nilai satu di Postman. Anotasi `@IsPromoPromoCreate` akan melakukan validasi untuk memastikan bahwa hanya para pengguna dengan otoritas atau peran tertentu yang dapat mengakses metode *create*. Anotasi `@ResponseStatus(value = HttpStatus.CREATED)` menandakan bahwa respons HTTP POST akan memberikan status '201 Created' jika entitas Promo berhasil dibuat.

Anotasi `@LogExecutionTime` digunakan untuk mencatat waktu selama proses eksekusi metode *create* berlangsung. Anotasi `@Validated` digunakan untuk memastikan bahwa objek 'promoCreateDto' divalidasi terlebih dahulu sebelum digunakan. Anotasi `@RequestBody` menandakan bahwa parameter 'promoCreateDto' diambil dari 'body' permintaan HTTP. Metode ini dapat melempar pengecualian 'DataNotFoundException' dan mengembalikan objek 'PromoDto' yang berisi data-data informasi mengenai entitas Promo yang baru saja dibuat.

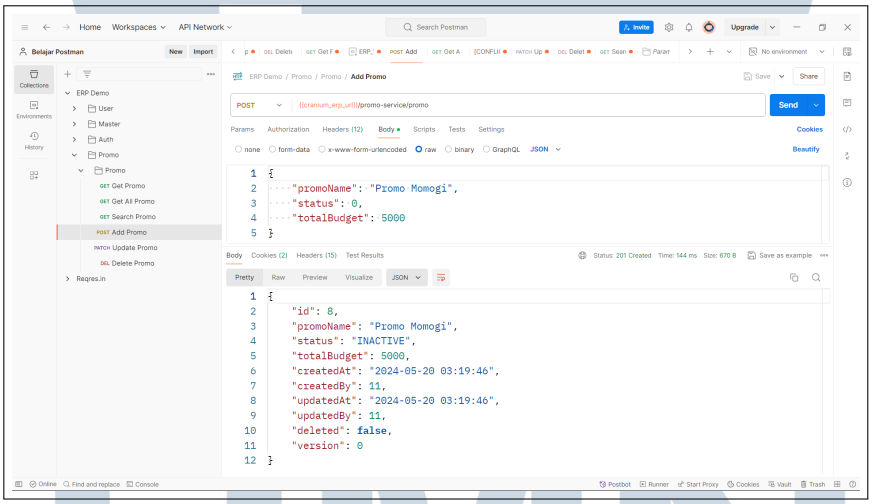
Setelah selesai memproses Controller, proses *create* akan dilanjutkan ke Service. Isi dari *file* Service yang bernama PromoService dapat dilihat pada Kode 3.5.

```
public PromoDto createPromo(PromoCreateDto promoCreateDto) throws
    DataNotFoundException {
    Promo promo = Promo
            .builder()
            .promoName(promoCreateDto.getPromoName())
            .status(promoCreateDto.getStatus())
            .build();
    Budget budget = Budget
            .builder()
            .totalBudget(promoCreateDto.getTotalBudget())
            .promo(promo)
            .build();
    promo.setBudget(budget);
    promo = promoRepository.save(promo);
    return promoMapper.map(promo, PromoDto.class);
```

```
15 }
```

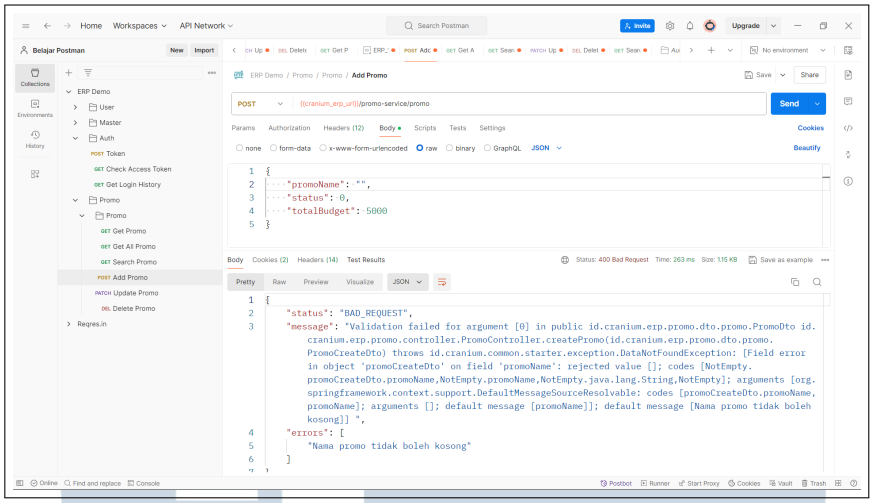
Kode 3.5: Metode Create dalam file Service

Proses ini akan mengambil nilai ’promoName’ dan ’status’ dari objek ’PromoCreateDto’ yang akan diteruskan sebagai parameter ke metode ’createPromo’. Entitas Promo ini memiliki *foreign key* dengan entitas Budget, sehingga *field* ’totalBudget’ yang ada di dalam ’PromoCreateDto’ diambil dari entitas Budget. Kode `promo.setBudget(budget)` menghubungkan objek Budget dengan objek Promo sebelum menyimpan Promo ke *database*. Selanjutnya, proses `return promoMapper.map(promo, PromoDto.class)` akan menyesuaikan objek Promo dengan ’PromoDto’ menggunakan ’promoMapper’. Hasil dari proses *create* data ini dapat dilihat pada Gambar 3.4.



Gambar 3.4. Hasil sukses Create Promo

Setelah menyelesaikan proses di dalam Service, objek Promo yang telah dibuat akan memasuki proses penyimpanan di Repository. Repository berinteraksi dengan *database* untuk menyimpan data-data yang telah diterima dari Service. Setelah objek Promo disesuaikan dengan Mapper, Promo akan disimpan ke *database* menggunakan metode ’save’ dari Repository. Kemudian, objek DTO akan dikembalikan oleh Service ke Controller dan Controller akan mengembalikan objek DTO ini sebagai respons HTTP ke klien (Postman). Jika hasil *create* data gagal dapat dilihat pada Gambar 3.5.



Gambar 3.5. Hasil gagal Create Promo

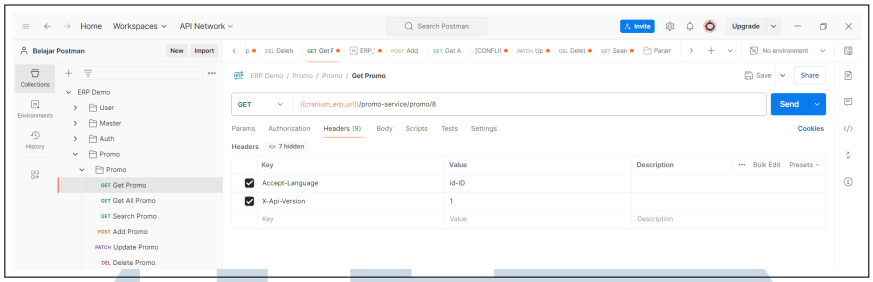
Pembuatan entitas Promo bisa gagal dikarenakan parameter ’promoName’ berisi string kosong. Hal ini tidak sesuai dengan validasi yang terjadi pada DTO karena *field* ’promoName’ memiliki anotasi `@NotEmpty`. Jika parameter ’promoName’ kembali diisi dengan string (tidak kosong), maka proses *create* data akan kembali berhasil.

3.4.2 Alur Read

Proses *read* data pada ERP memungkinkan pengguna untuk melihat data yang sudah ada atau informasi yang telah disimpan dalam sistem *database* setelah proses *create* berlangsung. Membaca data dapat dilakukan dengan tiga cara, yaitu membaca satu data, membaca semua data, dan membaca data dengan cara memasukkan *page* dan *size* yang diinginkan atau nama *field* beserta isi data entitas Promo yang ingin ditampilkan.

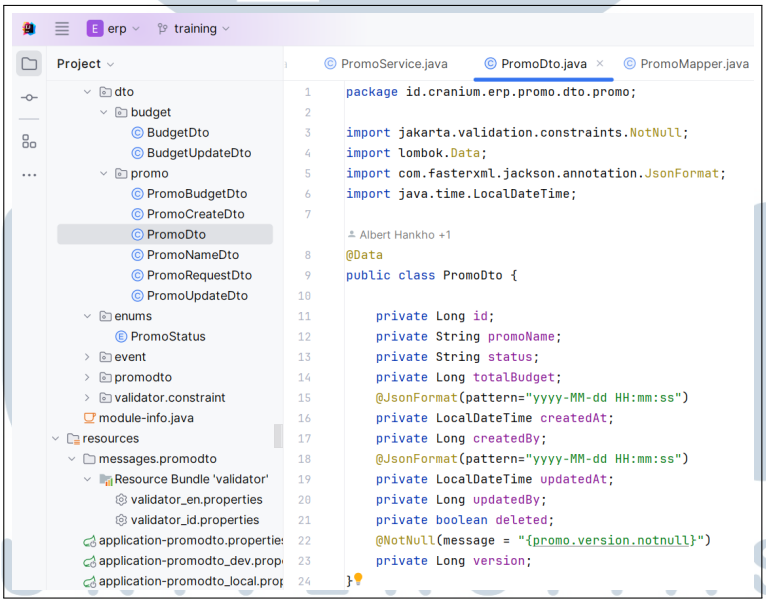
A. Alur Membaca Satu Data Entitas Promo

Berikut awal mula alur membaca satu data entitas Promo yang dapat dilihat pada Gambar 3.6.



Gambar 3.6. Permintaan API HTTP GET dari klien (Postman)

Proses *read* data hampir mirip dengan proses *create* yang diawali dengan permintaan HTTP dari Postman. Perbedaannya terletak pada HTTP yang harus dikirim oleh pengguna untuk membaca data yaitu HTTP GET. Proses permintaan HTTP GET untuk membaca satu data memerlukan ID untuk mendapatkan data Promo yang ingin ditampilkan. Hal ini dapat dilihat pada URL di Postman `promo-service/promo/8`. Angka delapan merupakan ID Promo kedelapan yang telah di buat dalam proses *create* data. Selanjutnya adalah proses pembuatan DTO bernama PromoDto yang dapat dilihat pada Gambar 3.7.



Gambar 3.7. DTO Get Promo

PromoDto berisi deklarasi dari 10 field yang terdiri dari:

1. id

   Field ini menyimpan ID unik dari data Promo. Tipe data 'Long' merupakan tipe data primitif yang dapat menyimpan bilangan bulat panjang (64-bit) melebihi kapasitas tipe data 'int'.

2. promoName
   Field ini menyimpan nama Promo yang dibuat oleh pengguna. Tipe data 'String' merupakan tipe data non-primitif yang dapat menyimpan urutan karakter seperti teks atau data alfanumerik.

3. status
   Field ini menyimpan status Promo yang dibuat oleh pengguna. Terdapat tiga status yang dapat dipilih oleh pengguna, yaitu status INACTIVE (dengan *value* nol), ACTIVE (dengan *value* 1), dan EXPIRED (dengan *value* 2).

4. totalBudget
   Field ini menyimpan anggaran total Budget untuk Promo yang ditentukan oleh pengguna.

5. createdAt
   Field ini menyimpan tanggal dan waktu ketika Promo dibuat. Tipe data 'LocalDateTime' digunakan untuk merepresentasikan tanggal dan waktu secara akurat hingga nanodetik.

6. createdBy
   Field ini menyimpan ID pengguna yang membuat Promo.

7. updatedAt
   Field ini menyimpan tanggal dan waktu ketika Promo terakhir kali diperbarui.

8. updatedBy
   Field ini menyimpan ID pengguna yang terakhir kali memperbarui promo.

9. deleted
   Field ini menyimpan status penghapusan Promo, apakah telah dihapus atau belum. Tipe data 'boolean' adalah tipe data primitif yang digunakan untuk menyimpan nilai 'true' dan 'false'.

10. version
    Field ini menyimpan versi dari Promo. Anotasi `@NotNull(message = "{promo.version.notnull}")` memastikan *field version* tidak boleh bernilai 'null' saat melakukan validasi. Jika *version* berisi nilai 'null', maka validasi akan gagal dan menampilkan pesan kesalahan yang terdapat pada 'promo.version.notnull'.

Setelah membuat DTO, permintaan HTTP GET akan diproses ke PromoController. Berikut isi dari PromoController untuk metode membaca satu data yang dapat dilihat pada Kode 3.6.

```
@GetMapping(value = "/promo/{id}", headers = "X-Api-Version=1")
    @IsPromoPromoRead
    @ResponseStatus(value = HttpStatus.OK)
    @LogExecutionTime
    public PromoDto findPromoById(@PathVariable long id) throws
    DataNotFoundException {
        return promoService.findPromoById(id);
    }
```

Kode 3.6: Metode membaca satu data dalam file Controller

Anotasi `@GetMapping` akan menangani permintaan HTTP GET yang dikirim dari Postman. Variabel 'id' akan diisi dengan ID Promo yang ingin ditampikan. Anotasi `@ResponseStatus` akan memberikan respons HTTP '200 OK' ke klien jika metode ini berhasil dijalankan. Anotasi `@PathVariable long id` mengindikasikan bahwa parameter 'id' akan diambil dari path URL. Metode ini dapat melemparkan pengecualian 'DataNotFoundException' jika data yang diminta tidak ada atau tidak ditemukan. Metode GET memanggil metode 'findPromoById' yang ada di PromoService dengan menyertakan id Promo yang ingin ditampilkan.

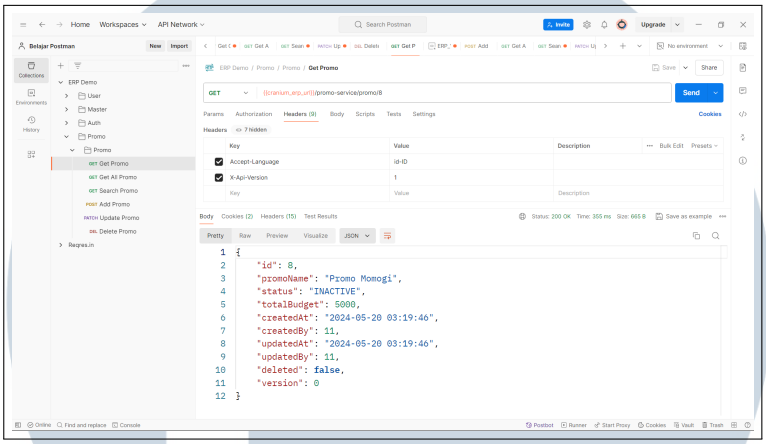
Setelah proses dalam PromoController selesai, dilanjutkan dengan PromoService yang akan menangani logika bisnis dan memanggil Repository untuk proses pengambilan data dari *database*. Isi dari PromoService untuk metode GET dapat dilihat pada Kode 3.7.

```
public PromoDto findPromoById(Long id) throws
    DataNotFoundException {
    if (promo.isEmpty()) {
        throw new DataNotFoundException(promoMessageSource.
    getMessage(PROMO_PROMO_FINDID_NOTFOUND, new Object[]{id},
    LocaleContextHolder.getLocale()));
    }
    return promoMapper.map(promo.get(), PromoDto.class);
}
```

Kode 3.7: Metode membaca satu data dalam file Service

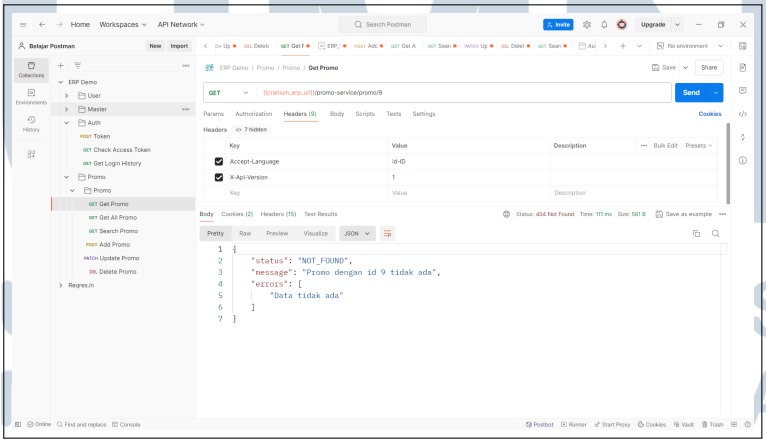
Metode ini menerima sebuah 'id' dengan tipe data 'Long' dan akan mengembalikan objek 'PromoDto'. Kode `if (promo.isEmpty())` memeriksa apakah objek Promo kosong atau tidak. Jika objek Promo kosong, maka pengecualian 'DataNotFoundException' akan dilemparkan. Selanjutnya,

PromoService akan mengakses PromoRepository untuk mencari data Promo dalam *database* berdasarkan kriteria yang telah diberikan. Hasil dari proses membaca satu data ini dapat dilihat pada Gambar 3.8.



Gambar 3.8. Hasil sukses membaca satu data Promo

Setelah data Promo ditemukan, data tersebut harus disesuaikan dengan PromoDto menggunakan PromoMapper. Kemudian, *database* akan melakukan *query* berdasarkan permintaan Repository dan mengembalikan data Promo yang diinginkan pengguna. Hasil akhir objek 'PromoDto' akan dikembalikan dari Service ke Controller yang akan diteruskan sampai ke klien sebagai respons HTTP pada Postman. Jika hasil *read* data tidak berhasil dapat dilihat pada Gambar 3.9.



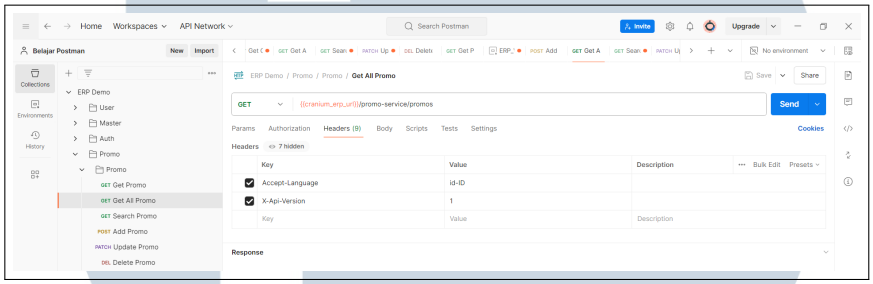
Gambar 3.9. Hasil gagal membaca satu data Promo

Pembacaan satu data entitas Promo dapat gagal jika ID Promo tidak ditemukan dalam *database*. Hal ini bisa terjadi karena entitas Promo dengan ID yang diminta belum dibuat atau telah dihapus dalam *database*. Jika pengguna

memasukkan ID Promo yang terdapat dalam *database* atau belum dihapus, maka data entitas Promo akan dikirim dan dapat dibaca oleh pengguna.

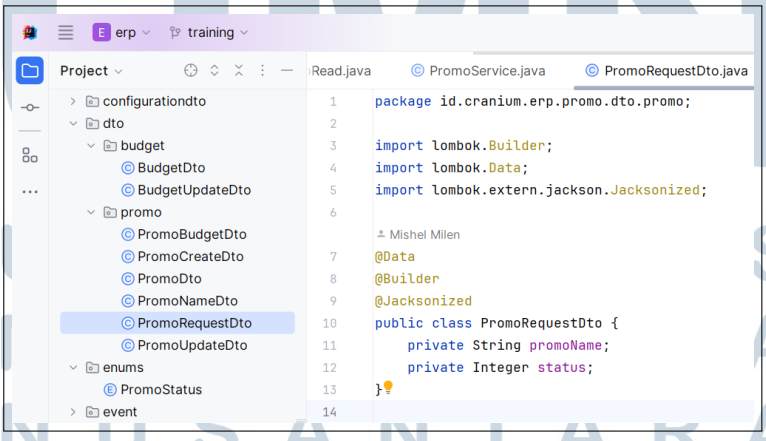
B. Alur Membaca Semua Data Entitas Promo

Awal alur membaca semua data entitas Promo dapat dilihat pada Gambar 3.10.



Gambar 3.10. Permintaan API HTTP GET ALL dari klien (Postman)

Proses membaca semua data yang telah dibuat oleh pengguna dapat dilakukan hanya jika data tersebut belum dihapus. Proses ini hampir mirip dengan proses membaca satu data, hanya saja dalam proses ini tidak perlu memanggil id data-data yang ingin dilihat. Alur membaca semua data diawali dengan klien mengirim permintaan HTTP GET ke endpoint '/promos' untuk mendapatkan semua data entitas Promo. Kemudian, dilanjutkan dengan pembuatan DTO bernama PromoRequestDto yang isinya dapat dilihat pada Gambar 3.11.



Gambar 3.11. DTO Get All Promo

Pembuatan PromoRequestDto ditujukan untuk hanya menerima parameter-parameter yang relevan terhadap filter dari permintaan pengguna seperti

'promoName' dan 'status'. Parameter ini akan digunakan untuk membangun spesifikasi *query* dalam melakukan pencarian data dalam *database* secara efisien dan efektif. Kemudian, permintaan HTTP akan diproses ke PromoController. Isi dari PromoController dapat dilihat pada Kode 3.8.

```
@GetMapping(value = "/promos", headers = "X-Api-Version=1")
public Page<PromoDto> getAllPromo(Pageable pageable, @RequestParam
    (required = false) String promoName, @RequestParam(required =
    false) Integer status){
    return promoService.getAllPromo(pageable, promoName, status);
}
```

Kode 3.8: Metode membaca semua data dalam file Controller

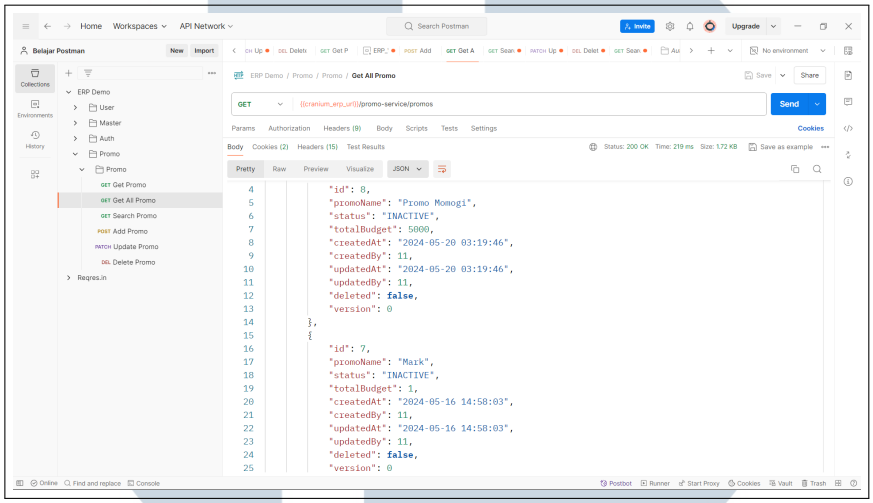
Kode 'Pageable pageable' merupakan parameter yang digunakan untuk menentukan *pagination*. Objek ini berisi informasi seperti nomor halaman dan ukuran halaman yang diminta. Anotasi `@RequestParam(required = false)` adalah parameter opsional dari permintaan HTTP yang digunakan untuk memfilter hasil berdasarkan *field*. Terdapat dua *field* yang difilter dalam metode ini, yaitu 'promoName' dan 'status'. Selanjutnya, metode 'getAllPromo' akan dipanggil dari PromoService. Isi dari PromoService terhadap metode 'getAllPromo' dapat dilihat pada Kode 3.9.

```
public Page<PromoDto> getAllPromo(Pageable pageable, String
    promoName, Integer status){
    PromoRequestDto promoRequestDto = PromoRequestDto
                .builder()
                .promoName(promoName)
                .status(status)
                .build();
    Page<Promo> promoPage = promoRepository.findAll(spec, pageable
    );
    return promoPage.map(promo -> promoMapper.map(promo, PromoDto.
    class));
}
```

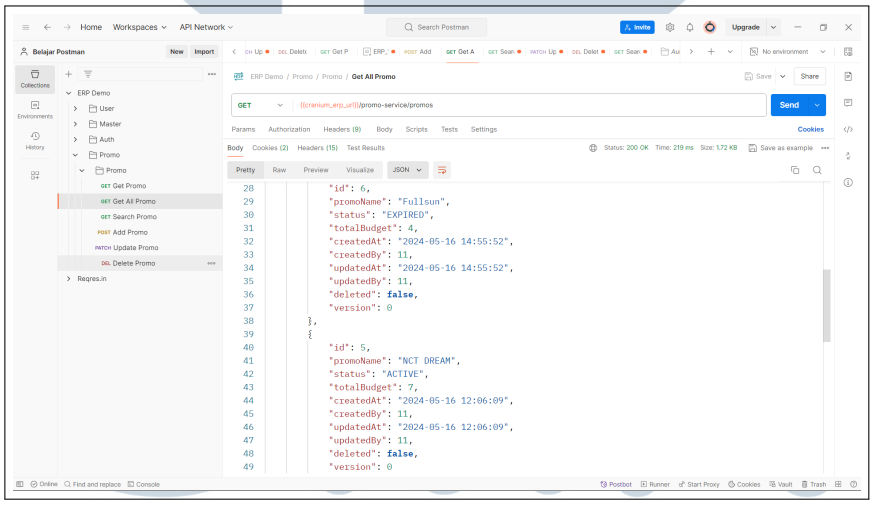
Kode 3.9: Metode membaca semua data dalam file Service

Metode 'getAllPromo' mengembalikan objek 'Page¡PromoDto¿' ke klien sebagai respons yang berisi data Promo berdasarkan parameter-parameter yang diterima seperti paginasi, nama promo, dan status. Objek 'PromoRequestDto' dibuat menggunakan *builder pattern* untuk menyimpan parameter yang diterima sebagai kriteria saat melakukan pencarian data. PromoRepository digunakan untuk mencari semua entitas Promo yang sesuai dengan spesifikasi yang telah diberikan.

PromoMapper digunakan untuk mengubah setiap entitas Promo ke ’PromoDto’ dan mengembalikannya sebagai halaman DTO ke Controller untuk kemudian diteruskan ke klien sebagai respons pada Postman. Hasil dari proses membaca semua data ini dapat dilihat pada Gambar 3.12.



(a) Data Promo untuk ID 7 dan 8



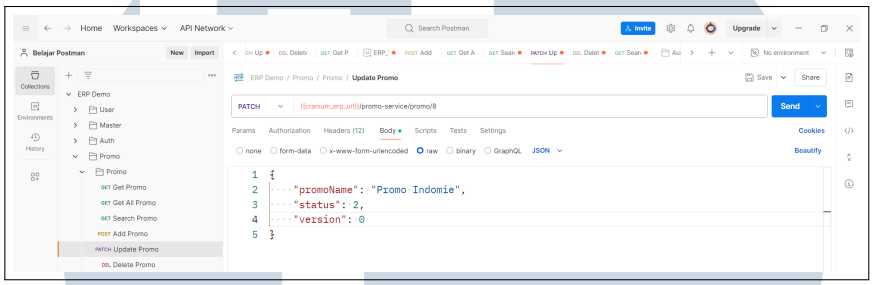
(b) Data Promo untuk ID 5 dan 6

Gambar 3.12. Hasil sukses membaca semua data Promo

Postman berhasil mengirimkan respons semua entitas Promo yang terdapat dalam *database* atau belum terhapus. Sebagai contoh, pengguna telah membuat delapan Promo dan menghapus Promo dengan ID satu sampai empat, sehingga yang dikirim hanyalah Promo dengan ID lima sampai delapan saja.

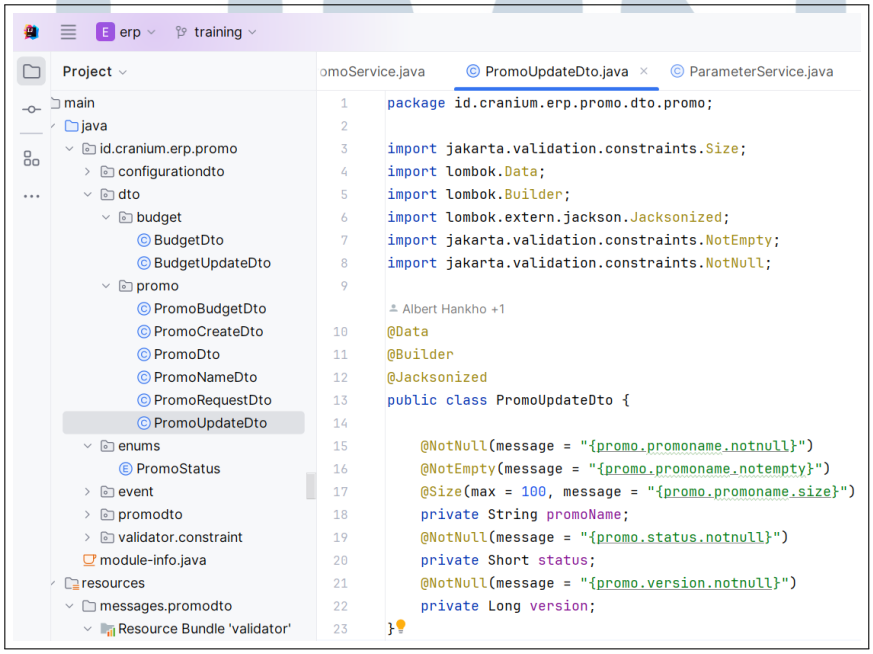
3.4.3 Alur Update

Operasi *update* digunakan untuk memperbarui data Promo yang sudah ada dalam *database*. Alur *update* ini dimulai dengan klien yang mengirimkan permintaan HTTP PATCH ke server. Hal ini dapat dilihat pada Gambar 3.13.



Gambar 3.13. Permintaan API HTTP PATCH dari klien (Postman)

Pada saat ingin memperbarui Promo, prosesnya mirip seperti saat membuat Promo untuk pertama kali. Pengguna diharuskan untuk memasukkan data-data yang ingin diperbarui, namun untuk proses *update* ini hanya dapat mengubah parameter ’promoName’ dan ’status’ saja. Pengguna dapat memperbarui salah satu parameter, tidak harus keduanya. Selanjutnya adalah pembuatan objek DTO yang berisi data-data untuk diperbarui. DTO yang digunakan bernama PromoUpdateDto yang dapat dilihat pada Gambar 3.14.



Gambar 3.14. DTO Update Promo

Sebelum digunakan, PromoUpdateDto akan divalidasi menggunakan `@Validated` dan dibuat pesan kesalahannya. Kemudian, permintaan HTTP akan diterima oleh PromoController. Isi PromoController untuk metode *update* dapat dilihat pada Kode 3.10.

```
@PatchMapping(value = "/promo/{id}", headers = "X-Api-Version=1")
    @IsPromoPromoUpdate
    @ResponseStatus(value = HttpStatus.OK)
    @LogExecutionTime
    public PromoDto updatePromo(@PathVariable Long id,
    @RequestBody @Validated PromoUpdateDto promoUpdateDto) throws
    DataNotFoundException, ObjectOptimisticLockingFailureException,
     DataLockException {
        return promoService.updatePromo(id, promoUpdateDto);
    }
```

Kode 3.10: Metode Update dalam file Controller

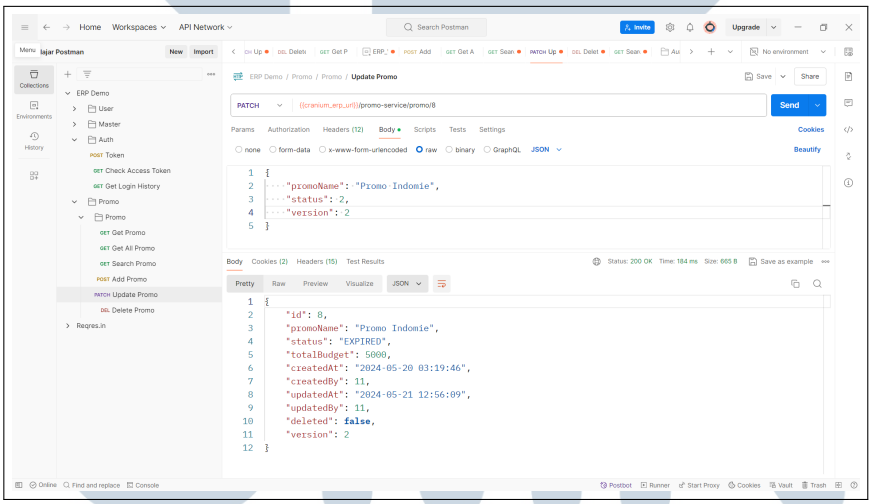
Anotasi `@PatchMapping` digunakan untuk melakukan perubahan pada data Promo yang sudah ada atau sudah di buat sebelumnya. Metode 'updatePromo' memerlukan ID dari entitas Promo yang ingin diubah pada URL endpoint. Anotasi `@PathVariable Long id` akan mengambil nilai ID dari URL path dan memasukkannya ke dalam parameter 'id'. Kode `@RequestBody @Validated PromoUpdateDto promoUpdateDto` digunakan untuk menyesuaikan permintaan HTTP ke objek 'promoUpdateDto' dan melakukan validasi. Jika metode ini berhasil dijalankan, maka akan muncul status HTTP '200 OK'. Kemudian, metode ini akan memanggil metode 'updatePromo' dalam PromoService untuk memperbarui entitas Promo yang dapat dilihat pada Kode 3.11.

```
public PromoDto updatePromo(Long id, PromoUpdateDto promoUpdateDto
    ) throws DataNotFoundException, DataLockException,
    ObjectOptimisticLockingFailureException {
    if (promoOptional.isEmpty()) {
        throw new DataNotFoundException(promoMessageSource.
    getMessage(PROMO_PROMO_FINDID_NOTFOUND, new Object[]{id},
    LocaleContextHolder.getLocale()));
    }

    if (promo.getVersion() != promoUpdateDto.getVersion()) {
  throw new DataLockException("Database version: " + promo.
    getVersion() +      ", Current version: " + promoUpdateDto.
    getVersion());
```

```
8       }
9
10      return promoMapper.map(promo, PromoDto.class);
11  }
```

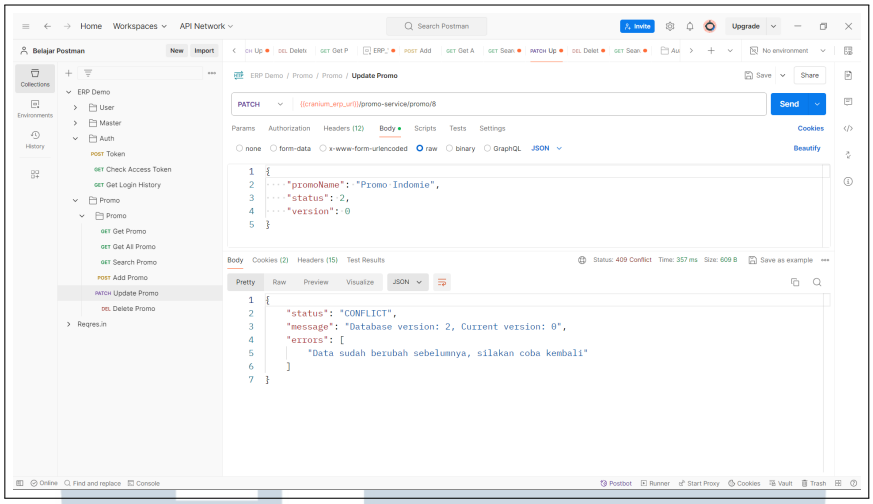
Kode 3.11: Metode Update dalam file Service

Metode ini akan melemparkan 'DataNotFoundException' jika ID Promo pada path variabel tidak ditemukan. Kode `if (promo.getVersion() != promoUpdateDto.getVersion()) {...}` akan membandingkan versi pada entitas Promo di *database* dengan versi yang ada di DTO, jika berbeda maka 'DataLockException' akan dilemparkan sebagai respons ke pengguna untuk memberitahu bahwa terjadi konflik. Entitas Promo yang baru saja diperbarui akan disimpan kembali ke *database* dan disesuaikan dengan 'PromoDto' melalui PromoMapper. Selanjutnya, Controller akan mengembalikan entitas Promo yang telah berhasil di perbarui kepada klien sebagai respons. Hasil dari proses membaca semua data ini dapat dilihat pada Gambar 3.15.



Gambar 3.15. Hasil sukses Update Promo

Hasil entitas Promo dengan ID delapan telah berhasil dirubah dari yang sebelumnya memiliki nama 'Promo Momogi' menjadi 'Promo Indomie' dan status yang berubah dari 'INACTIVE' menjadi 'EXPIRED'. Jika hasil *update* Promo gagal dirubah dapat dilihat pada Gambar 3.16.

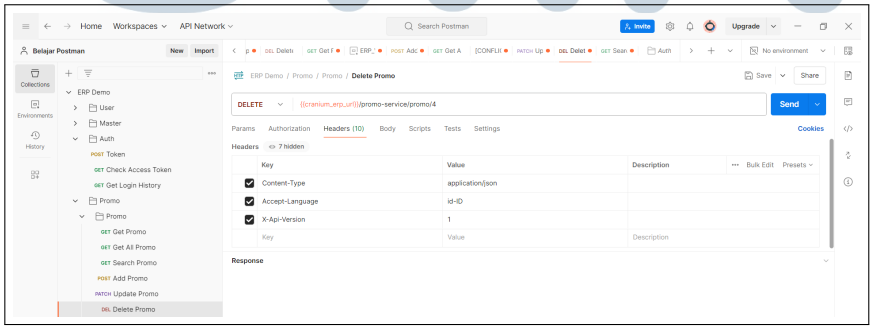


Gambar 3.16. Hasil gagal Update Promo

Proses merubah Promo ini gagal karena terjadi konflik, dimana versi sudah berubah menjadi dua dari yang awalnya bernilai nol. Hal ini dikarenakan pengguna telah melakukan *update* terhadap dua parameter (promoName dan status) sebelumnya, sehingga saat ingin memperbarui Promo kembali, versi telah berubah menjadi dua. Setelah versi dirubah menjadi dua, maka proses *update* data akan berhasil dilakukan.

3.4.4 Alur Delete

Proses *delete* data merupakan proses menghapus data-data yang sudah ada atau yang telah dibuat dalam proses *create*. Alur menghapus data dimulai dengan klien (Postman) mengirim permintaan HTTP DELETE ke server yang dapat dilihat pada Gambar 3.17.



Gambar 3.17. Permintaan API HTTP DELETE dari klien (Postman)

Proses menghapus data memerlukan ID entitas Promo yang ingin dihapus

dalam path URL seperti proses membaca satu data dan memperbarui data. Proses ini menggunakan DTO yang sama dengan proses membaca satu data yaitu PromoDto. Setelah mendapatkan ID Promo, permintaan HTTP DELETE akan diproses ke PromoController yang dapat dilihat pada Kode 3.12.

```
@DeleteMapping(value = "/promo/{id}", headers = "X-Api-Version=1")
    @IsPromoPromoDelete
    @ResponseStatus(value = HttpStatus.OK)
    @LogExecutionTime
    public PromoDto deletePromo(@PathVariable Long id) throws
   DataNotFoundException, DataLockException{
        return promoService.deletePromo(id);
    }
```

Kode 3.12: Metode Delete dalam file Controller

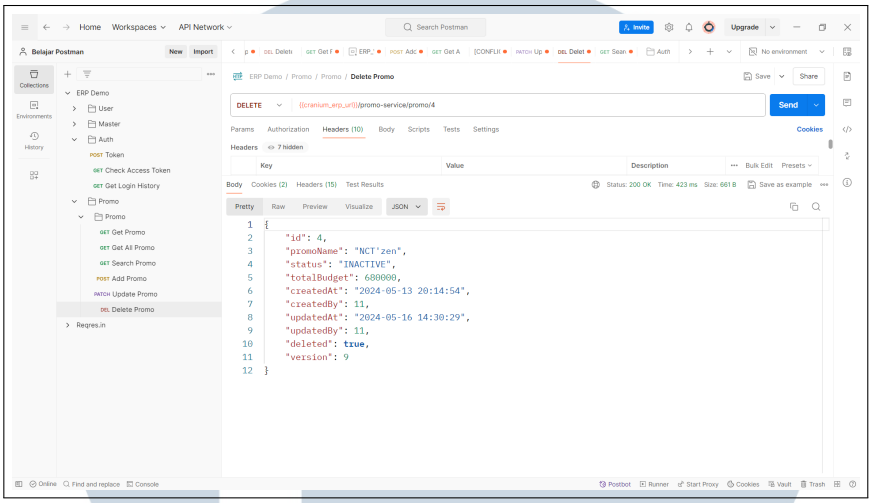
Anotasi `@DeleteMapping` akan menangani permintaan HTTP DELETE ke URL '/promo/id'. Kemudian validasi `@IsPromoPromoDelete` akan dijalankan dan memberikan status '200 OK' saat metode 'deletePromo' berhasil dieksekusi. Metode ini mengambil parameter 'id' entitas Promo yang ingin dihapus dan memanggil metode 'deletePromo()' dari PromoService yang dapat dilihat pada Kode 3.13.

```
public PromoDto deletePromo(Long id) throws DataNotFoundException,
    DataLockException{
    try{
        if(promoOptional.isEmpty()){
            throw new DataNotFoundException(promoMessageSource.
   getMessage(PROMO_PROMO_FINDID_NOTFOUND, new Object[]{id},
   LocaleContextHolder.getLocale()));
  }
  Promo promo = promoOptional.get();
  promo.setDeleted(true);
  promo = promoRepository.save(promo);
  return promoMapper.map(promo, PromoDto.class);
    }
}
```

Kode 3.13: Metode Delete dalam file Service

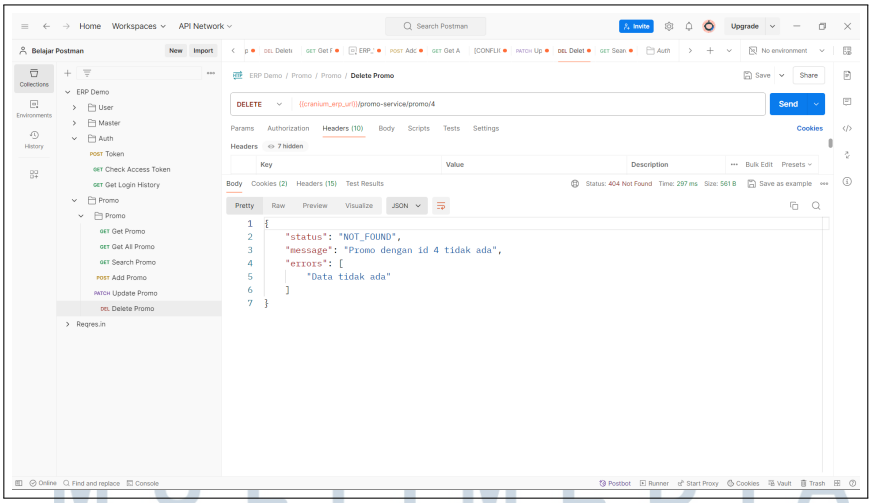
Kode `promo.setDeleted(true)` akan menandai Promo yang sesuai dengan ID yang dicari dan merubah parameter 'deleted' menjadi 'true'. Perubahan ini akan disimpan ke dalam *database* menggunakan PromoRepository. Kemudian, PromoMapper akan memetakan objek Promo yang telah dihapus ke objek 'PromoDto' dan mengembalikannya ke Controller untuk diteruskan ke klien

sebagai bentuk respons. Hasil dari menghapus data entitas Promo ini dapat dilihat pada Gambar 3.18.



Gambar 3.18. Hasil sukses Delete Promo

Proses menghapus data berhasil dilakukan karena ID data yang ingin dihapus berhasil ditemukan dalam *database* (belum pernah dihapus). Jika proses ini berhasil, status respons menjadi ’200 OK’ dan parameter ’deleted’ berubah menjadi ’true’. Proses gagal menghapus data dapat dilihat pada Gambar 3.19.



Gambar 3.19. Hasil gagal Delete Promo

Proses menghapus data dapat gagal jika mencoba menghapus data dengan ID yang sebelumnya sudah pernah dihapus. Sebagai contoh, pengguna mencoba untuk menghapus data Promo dengan ID empat yang sebelumnya sudah dihapus.

Hasilnya adalah proses *delete* gagal dan respons yang diberikan adalah bentuk pesan kesalahan, yakni ’Promo dengan id 4 tidak ada’ dengan status ’404 Not Found’.

3.4.5 Alur Unit Test Entitas Promo

Unit Test adalah praktik pengujian perangkat lunak yang bertujuan untuk memvalidasi bahwa setiap unit perangkat lunak berfungsi dan memenuhi persyaratan yang telah ditentukan [10]. Komponen yang akan diuji adalah proses CRUD yang telah dibuat. Pada proses pembuatan *unit test* ini, program tidak dibuat secara menyeluruh, melainkan hanya beberapa program saja untuk melengkapi proses-proses yang belum dibuat. Terdapat tiga hal yang perlu dibuat untuk melakukan pengujian, yaitu kelas PromoPromoBase, PromoServiceTest, dan kontrak Groovy. Proses pengujian yang dilakukan pada CRUD memiliki cara yang hampir sama, sehingga proses alur *unit test* ini akan berisi satu contoh saja, yaitu pada proses *update* data Promo.

A. Pembuatan Kelas PromoPromoBase

Proses pengujian dalam *PromoPromoBase* diawali dengan pembuatan metode ’setup’ yang dapat dilihat pada Gambar 3.20.

```
Albert Hankho +1
@BeforeEach
public void setup() {
    StandaloneMockMvcBuilder standaloneMockMvcBuilder = MockMvcBuilders.standaloneSetup(promoController)
            .setCustomArgumentResolvers(new PageableHandlerMethodArgumentResolver())
            .setControllerAdvice(new RestResponseEntityExceptionHandler(starterMessageSource))
            .apply(SecurityMockMvcConfigurers.springSecurity(springSecurityFilterChain));
    RestAssuredMockMvc.standaloneSetup(standaloneMockMvcBuilder);

    getPromoById_should_be_success();
    getPromoById_should_be_failed();
    createPromo_should_be_success();
    createPromo_should_be_failed();
    updatePromo_should_be_success();
    updatePromo_should_be_failed();
    deletePromo_should_be_success();
    deletePromo_should_be_failed();
    getAllPromo_should_be_success();
}
```

Gambar 3.20. Setup Unit Test dalam file Base

Anotasi `@BeforeEach` menandakan bahwa metode ’setup’ akan dijalankan sebelum semua metode pengujian yang terdapat di dalam PromoPromoBase. Metode ’setup’ akan mengatur semua kebutuhan dalam melakukan pengujian

unit pada PromoController dan memanggil setiap metode pengujian yang terdapat dalam PromoPromoBase. Semua metode pengujian menggunakan *mock* objek 'promService' yang disesuaikan dengan setiap metode pengujian tersebut. Selanjutnya adalah proses pembuatan metode *update* yang dapat dilihat pada Gambar 3.21.

```
private void updatePromo_should_be_success() {
    PromoUpdateDto promoUpdateDto = PromoUpdateDto
            .builder()
            .promoName("Promo 3")
            .status((short)PromoStatus.ACTIVE.getValue())
            .version(1L)
            .build();

    PromoDto promoDto = new PromoDto();
    promoDto.setId(2L);
    promoDto.setPromoName(promoUpdateDto.getPromoName());
    promoDto.setStatus(PromoStatus.getEnum(promoUpdateDto.getStatus()).name());
    promoDto.setCreatedAt(LocalDateTime.of( year: 2023, Month.JANUARY, dayOfMonth: 30, hour: 15, minute: 30, second: 59));
    promoDto.setCreatedBy(1L);
    promoDto.setUpdatedAt(LocalDateTime.of( year: 2023, Month.JANUARY, dayOfMonth: 30, hour: 15, minute: 30, second: 59));
    promoDto.setUpdatedBy(1L);
    promoDto.setDeleted(false);
    promoDto.setVersion(1L);

    Mockito.when(promoService.updatePromo( id: 2L, promoUpdateDto)).thenReturn(promoDto);
}

1 usage   Mishel Milen
private void updatePromo_should_be_failed() {
    PromoUpdateDto promoUpdateDto = PromoUpdateDto
            .builder()
            .promoName("Promo 4")
            .version(2L)
            .status((short)PromoStatus.ACTIVE.getValue())
            .build();

    String exceptionMessage = "Database version: 3, Current version: 2";
    Mockito.when(promoService.updatePromo( id: 3L, promoUpdateDto)).thenThrow(new DataLockException(exceptionMessage));
}
```

Gambar 3.21. Metode Update dalam file Base

Metode `updatePromo\_should\_be\_success()` mensimulasikan skenario sukses saat memperbarui suatu entitas Promo. Objek 'PromoUpdateDto' berisi data-data yang akan digunakan untuk memperbarui Promo, sedangkan objek 'PromoDto' berisi data-data yang diharapkan menjadi respons dari pemanggilan metode tersebut. Metode `updatePromo\_should\_be\_failed()` berlawanan dengan metode sebelumnya karena metode ini mensimulasikan kondisi saat gagal memperbarui Promo yang dikarenakan terkunci oleh versi data yang lebih baru. Hal ini dapat dilihat pada 'String exceptionMessage' yang berisi pesan kesalahan bahwa versi data yang ingin diperbaharui berbeda dengan yang terdata dalam *database*.

B. Pembuatan Kelas PromoServiceTest

Proses pembuatan Unit Test dilanjut dengan pembuatan kelas PromoServiceTest. Proses awal yang dibuat adalah metode 'setup' yang isinya dapat dilihat pada Gambar 3.22.

Gambar 3.22. Setup Unit Test dalam file Service Test

Metode 'setup' bertanggung jawab untuk menyiapkan kondisi awal yang diperlukan saat melakukan pengujian data. Ketika argumen "PROMO_PROMO_OWN" dipanggil, *mockito* akan mengembalikan objek 'Optional' yang berisi string kosong sehingga setiap kondisi awal dapat dipastikan aman untuk melakukan setiap proses pengujian. Selanjutnya adalah proses pembuatan metode *update* yang dapat dilihat pada Gambar 3.23.

```
void testUpdatePromo() throws DataNotFoundException, DataLockException, ObjectOptimisticLockingFailureException {
    Long promoId = 1L;
    String promoName = "Test Update Promo 1";
    int status = PromoStatus.ACTIVE.getValue();
    PromoUpdateDto promoUpdateDto = PromoUpdateDto
            .builder()
            .promoName(promoName)
            .status((short)status)
            .version(1L)
            .build();
    PromoDto promoDto = promoService.updatePromo(promoId, promoUpdateDto);
    assertEquals(promoDto.getPromoName(), promoName);
    assertEquals(PromoStatus.getEnum(status).name(), promoDto.getStatus());
}
```

Gambar 3.23. Metode Update dalam file Service Test

Metode 'testUpdatePromo()' menguji apakah metode 'updatePromo()' pada promoService dapat memperbarui data Promo dengan benar atau tidak. Data-data yang akan digunakan untuk melakukan pengujian ini disiapkan dan disesuaikan dengan PromoUpdateDto. Pada kode 'assertEquals' akan dilakukan pengecekan apakah hasil data Promo (nama promo dan status) yang telah diperbarui sesuai dengan hasil yang diharapkan atau tidak.

C. Pembuatan Groovy

Groovy merupakan bahasa pemrograman yang ditujukan untuk platform Java. Bahasa ini mudah digunakan dan memiliki sintaks yang lebih sederhana daripada Java, sehingga sering digunakan dalam pembuatan *unit test*. Berikut isi dari Groovy terhadap metode 'updateSuccess' yang dapat dilihat pada Gambar 3.24.

```
Contract.make {
    description  description: "update promo"
    request {
        url  url: "/promo-service/promo/2"
        method PATCH()
        headers {
            contentType(applicationJson())
            header "X-Api-Version": "1"
            header "Accept-Language": "id-ID"
        }
        body (
                promoName: "Promo 3",
                status: 1,
                version: 1
        )
    }

    response {
        status OK()
        headers {
            contentType(applicationJson())
        }
        body (
                id: 2,
                promoName: "Promo 3",
                status: "ACTIVE",
                createdAt: "2023-01-30 15:30:59",
                createdBy: 1,
                updatedAt: "2023-01-30 15:30:59",
                updatedBy: 1,
                deleted: false,
                version: 1
        )
```

Gambar 3.24. Groovy sukses Update

Kontrak Groovy ini berisi skenario berhasil saat memperbarui Promo. ID dua yang terletak pada URL endpoint disesuaikan dengan ID Promo yang telah ditentukan dalam PromoPromoBase. Metode yang digunakan adalah 'PATCH' karena proses akan melakukan pengujian terhadap *update* data. Isi nilai data-data pada 'body' juga disesuaikan dengan nilai yang telah ditetapkan pada Base.

Hasil respons yang diharapkan dari pengujian ini adalah status ’200 OK’ yang menandakan proses memperbarui data berhasil dilakukan. Jika saat dijalankan, hasil respons yang keluar selain ’200 OK’ maka terdapat kesalahan atau nilai data dalam ’body’ tidak sinkron antara Groovy dengan Base.

Isi kontrak Groovy terhadap metode ’updateFailed’ dapat dilihat pada Gambar 3.25.

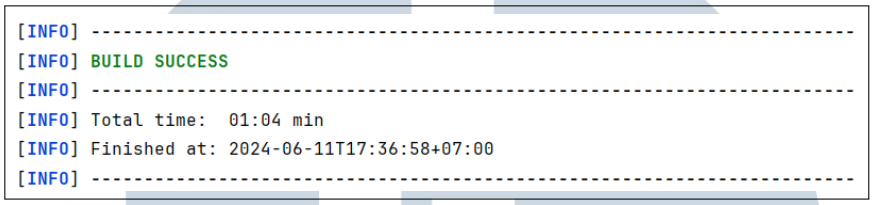
```
Contract.make {
    description  description: "update promo should be failed"
    request {
        url  url: "/promo-service/promo/3"
        method PATCH()
        headers {
            contentType(applicationJson())
            header "X-Api-Version": "1"
            header "Accept-Language": "id-ID"
        }
        body (
                promoName: "Promo 4",
                version: 2,
                status: 1
        )
    }

    response {
        status  status: 409
        headers {
            contentType(applicationJson())
        }
        body (
                "status": "CONFLICT",
                "message": "Database version: 3, Current version: 2",
                "errors": [
                        "Data sudah berubah sebelumnya, silakan coba kembali"
                ]
        )
    }
}
```

Gambar 3.25. Groovy gagal Update

Kontrak ini berisi skenario gagal saat ingin memperbarui data Promo. ID Promo yang akan dilakukan uji adalah ID tiga yang telah disesuaikan pada isi PromoPromoBase terhadap metode `updatePromo\_should\_be\_failed()`. Semua nilai data dalam ’body’ disesuaikan dengan yang telah ditetapkan dalam Base. Kontrak ini diharapkan memberikan respons status ’409 CONFLICT’ yang artinya terjadi kesalahan saat memperbarui suatu entitas data Promo.

D. Hasil Unit Test Entitas Promo

Berikut merupakan hasil dari menjalankan *unit test* untuk entitas Promo secara berhasil yang dapat dilihat pada Gambar 3.26.



Gambar 3.26. Hasil Unit Test berhasil

Proses menjalankan *unit test* dilakukan dengan *compile* modul Promo secara keseluruhan, mulai dari DTO sampai proses *service*. Jika pembuatan CRUD maupun pengujian *unit test* sudah berhasil seluruhnya, maka hasil yang ditampilkan adalah tulisan 'BUILD SUCCESS'. Jika proses pengujian gagal dilakukan, maka akan memunculkan pesan *error* di *Terminal* yang menunjukkan letak kesalahan kode.

3.5 Kendala dan Solusi yang Ditemukan

Selama proses pelaksanaan magang di PT Cranium Royal Aditama terdapat beberapa kendala yang dialami, antara lain:

1. Mengalami kesulitan dalam pembuatan API CRUD entitas Promo dikarenakan memiliki *foreign key* terhadap entitas Budget yang dikerjakan oleh anggota magang lainnya.
2. Merasa sulit saat melakukan pembuatan *unit test* yang sangat kompleks dan paling sering mengalami kendala *error*.

Dari semua kendala yang dirasakan saat proses magang, terdapat beberapa solusi yang dilakukan untuk mengatasi kendala-kendala tersebut, antara lain:

1. Meningkatkan komunikasi dengan anggota tim magang yang mengerjakan bagian entitas Budget sehingga kode yang saling berhubungan tidak mengalami konflik atau *error* dan melihat contoh pembuatan modul yang sudah ada sebelumnya yang juga memiliki *foreign key*.
2. Melakukan diskusi dengan anggota magang untuk saling membantu dan memberi saran satu sama lain selama pembuatan *unit test*.