BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Kedudukan sebagai *Intern Fullstack Developer* di PT Cranium Royal Aditama menjadi bagian dari *batch* 6 dalam program *internship* Cranium, yang terdiri dari enam sampai delapan peserta. Selama masa *training*, bimbingan langsung diberikan oleh *supervisor*, Bapak Sugito, yang berperan penting dalam memberikan pengetahuan dasar dan teknis yang diperlukan. Selain itu, bantuan juga diterima dari *mentor-mentor batch* sebelumnya.

Pada tahap *real project*, *batch* 6 dibagi menjadi dua tim untuk mengerjakan *project* ERP. Setiap tim diberikan satu mentor untuk membimbing dan membantu mengatasi kendala yang muncul selama pengerjaan *project* berlangsung. *Mentor* yang mendampingi adalah Andrew Widjaya, seorang intern *fullstack developer* dari *batch* 4. Tim terdiri atas 3 orang yaitu, Nayasha Clarissa, Mishel Milen, dan Abelia Vidya.

3.2 Tugas yang Dilakukan

Selama masa magang sebagai *Fullstack Developer* di PT Cranium Royal Aditama, tanggung jawab yang dijalankan mencakup pengembangan *back end* dan *front end* pada sistem ERP. Proses pengembangan ini menggunakan bahasa pemrograman JavaScript, *framework Java Spring Boot*, serta TypeScript, HTML, dan CSS. Beberapa aplikasi yang digunakan dalam *project* ini meliputi IntelliJ, Postman, PGAdmin, dan GitHub Desktop.

Pengembangan sistem ERP ini mencakup pembuatan entitas baru pada *back end*, yaitu entitas Budget. Proses ini melibatkan pembuatan fungsi CRUD (*Create, Read, Update, Delete*) serta *unit test* untuk memastikan fungsionalitas berjalan dengan baik. Setelah bagian *back end* selesai, pengembangan akan dilanjutkan ke bagian *front end*, di mana dibuat antarmuka *website* yang menampilkan fungsi CRUD untuk entitas *Budget*.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang sebagai *Fullstack Developer* di PT Cranium Royal Aditama diuraikan seperti pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Menginstall *Software* pendukung yang dibutuhkan, seperti IntelliJ, pgAdmin4, PostMan. Mempelajari struktur ERP dan *query* dasar CRUD
2	Menghubungkan postgreSQL ke IntelliJ dan PostMan. Latihan *query* pada Hackerrank dan Latihan Java pada Sololearn.
3	Memahami dan mencoba *compile template* demo ERP. Mencoba *cloning* entitas pada *template* demo ERP.
4	Membuat CRUD dan *field* baru pada entitas *Product*. Membuat demo *unit test* dengan entitas *Product*.
5	Training materi *back end* ERP bagian *Annotation* dan *Validation*. Memahami alur *POST*, *PATCH*, *GET*, dan *DELETE*.
6	Memahami alur *input* dan *output* ERP. Memahami *Controller*, *Repository*, *Service*, dan *Mapper*.
7	Memahami alur unit test *back end*. Memahami materi *front end* ERP.
8	Memahami code *front end* ERP. Mencoba membuat tampilan *front end* menggunakan entitas *Product*.
9	Penyelesaian pembuatan entity *Product* (*back end* dan *front end*). Memahami alur *front end*.
10	Pemahaman alur *front end* (Alur *input* data masuk ke dalam Postman, dan Alur pengeluaran *output*)
11	Pemaparan materi mengenai *unit test front end*. Pemahaman detail alur *front end*.
12	Pembuatan *unit test front end*. Pemahaman alur *unit test front end*.
13	Libur hari raya.
14	Mempelajari *pull* dan *push* Github dan mulai *clone project training* ERP. Mengerjakan *task edit import*, menambahkan *validator*, dan juga melakukan *testing*.

15	Lanjut mengerjakan *testing unit test.* Membenarkan *error* yang terjadi saat melakukan *testing*.
16	Mulai mengerjakan pembuatan entitas baru *Budget*.
17	Mengerjakan CRUD pada entitas *Budget*. Mengerjakan *unit test*
18	Mengerjakan *unit test* pada *back end Budget*. Menyelesaikan *error* yang terdapat pada *unit test*.
19	Menyelesaikan pembuatan *back end* entitas *Budget*, dan berhasil *push* ke Github. Mulai pembuatan *front end* entitas *Budget*.

3.3.1 *Software & Hardware*

Software yang digunakan sebagai *tools* pendukung dalam pengerjaan magang adalah sebagai berikut:

- IntelliJ
  Merupakan *Integrated Development Environment* (IdE) utama yang digunakan untuk pengembangan *back end* dalam *project* ERP ini. IntelliJ, dikembangkan oleh JetBrains, dirancang khusus untuk pengembangan aplikasi berbasis bahasa pemrograman Java [7].

- Postman
  Alat ini digunakan untuk mengembangkan, menguji, dan mendokumentasikan API *(Application Programming Interface*) [8]. Pada Postman, kita dapat mengirimkan berbagai jenis permintaan HTTP, seperti *GET*, *POST*, *PATCH*, dan *DELETE* ke *server*, serta menganalisis respons yang diterima.

- Visual Studio Code
  Digunakan untuk pengembangan kode pada bagian *front end*. Visual Studio Code adalah *editor* kode sumber yang dikembangkan oleh Microsoft [9].

- PgAdmin4
  Digunakan untuk membuat dan mengelola database menggunakan PostgreSQL [10].

- GitHub Desktop
  Merupakan alat manajemen *project* yang digunakan untuk mengedit dan mengirimkan *project* yang dikerjakan secara kolaboratif [11]. Aplikasi ini

memudahkan anggota tim untuk bekerja bersama dalam satu *project* secara bersamaan.

Selain itu, *Hardware* yang digunakan untuk mengerjakan kegiatan magang adalah:

- Resolusi monitor: 1920 x 1080
- Ukuran monitor: 14.0-inch
- *Processor*: Intel Core i3-10110U
- RAM: 8 GB DDR4
- VGA: Nvidia GeForce MX130
- *Storage*: 512Gb SSD
- OS: Windows 11

3.3.2 Modul Utama ERP

Sistem ERP di PT Cranium Royal Aditama terdiri dari sembilan modul yang dirancang untuk menangani berbagai aspek operasional perusahaan. Berikut adalah penjelasan mengenai masing-masing modul:

1. *Purchasing*
   Modul ini bertanggung jawab atas seluruh proses pembelian. Fungsinya meliputi pemesanan barang, pengelolaan *supplier*, serta pencatatan transaksi pembelian.

2. *Selling*
   Modul ini berfokus pada manajemen penjualan. Fungsi utamanya adalah mengatur proses penjualan, mengelola data pelanggan, dan memproses pesanan penjualan hingga selesai.

3. *Accounting*
   Modul ini berfungsi untuk mengelola catatan keuangan dan pembukuan perusahaan. Fitur-fiturnya mencakup pencatatan transaksi keuangan, pembuatan laporan keuangan, dan pengelolaan anggaran.

4. *Ledger*

   Modul ini menangani pencatatan buku besar, memastikan semua transaksi keuangan tercatat dengan benar dan mendukung integrasi dengan modul akuntansi untuk pelaporan yang akurat.

5. *Finance*

   Modul ini berfungsi untuk mengelola keuangan perusahaan secara keseluruhan. Termasuk di dalamnya adalah manajemen kas, pengelolaan hutang dan piutang, serta analisis keuangan.

6. *Warehouse*

   Modul ini dirancang untuk mengelola persediaan dan ketersediaan stok barang. Fungsinya mencakup pengaturan penyimpanan barang, pengelolaan inventaris, dan pemantauan stok secara *real-time*.

7. *Production Planning*

   Modul ini digunakan untuk mengatur kegiatan produksi. Ini termasuk perencanaan produksi, pengelolaan jadwal produksi, serta pengawasan proses produksi untuk memastikan efisiensi dan kualitas.

8. *Promo*

   Modul ini berfokus pada manajemen promosi perusahaan. Fungsinya meliputi perancangan dan pelaksanaan program promosi, pengelolaan diskon, serta analisis efektivitas promosi terhadap peningkatan penjualan.

9. *Shipping*

   Modul ini bertanggung jawab atas pengelolaan pengiriman barang. Fitur-fiturnya termasuk pengaturan jadwal pengiriman, manajemen logistik, serta pelacakan status pengiriman untuk memastikan barang sampai ke pelanggan tepat waktu.

3.3.3 Pembuatan Entity Budget

A. Modul promo-dto

DTO (*Data Transfer Object*) adalah pola desain yang digunakan untuk memindahkan data antara subsistem dalam sebuah aplikasi. DTO merupakan objek sederhana yang digunakan untuk membawa data tanpa menyertakan logika bisnis yang rumit. Tujuan utama penggunaan DTO adalah mengurangi jumlah panggilan

jaringan dengan menggabungkan data yang sering dibutuhkan dalam satu objek, sehingga dapat dikirim dengan mudah dari satu sistem ke sistem lainnya.

Pada modul Promo-dto di bagian entitas Budget, dua Java class yang sudah ada, yaitu *BudgetDto* dan *BudgetUpdateDto*, telah disempurnakan. Selain itu, tiga class baru juga telah ditambahkan, yaitu *BudgetCreateDto*, *BudgetNameDto*, dan *BudgetRequestDto*. Berikut adalah penjelasan mengenai masing-masing DTO:

- *BudgetDto*

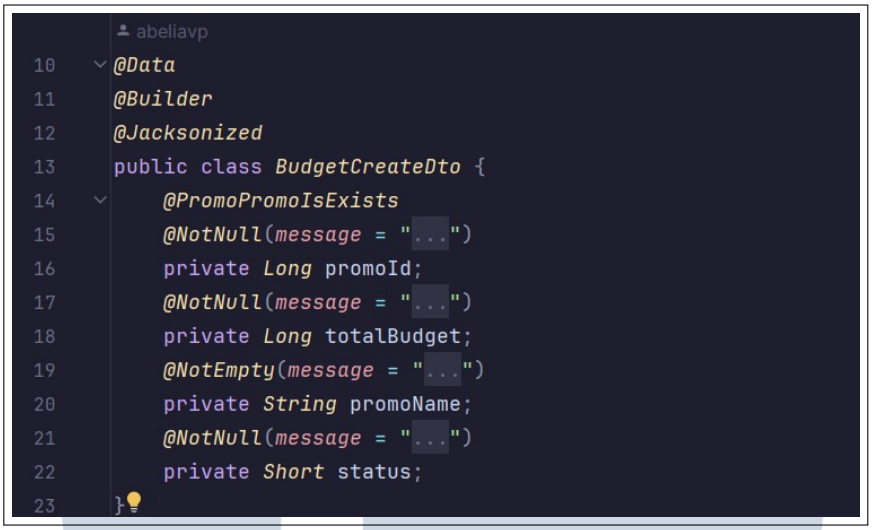
  BudgetDto merupakan DTO utama yang mencakup semua atribut penting dari entitas *Budget* seperti *Id, promoId, totalBudget, promoName, status, createdAt, createdBy, updatedAt, updatedBy, deleted*, dan sebagainya. Gambar 3.1 adalah isi kode dari *BudgetDto*.

```
Albert Hankho +1
@Data
public class BudgetDto {
    private Long id;
    private Long promoId;
    private Long totalBudget;
    private String promoName;
    private String status;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm:ss")
    private LocalDateTime createdAt;
    private Long createdBy;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm:ss")
    private LocalDateTime updatedAt;
    private Long updatedBy;
    private boolean deleted;
    private Long version;
}
```

Gambar 3.1. Isi *BudgetDto*

- *BudgetCreateDto*

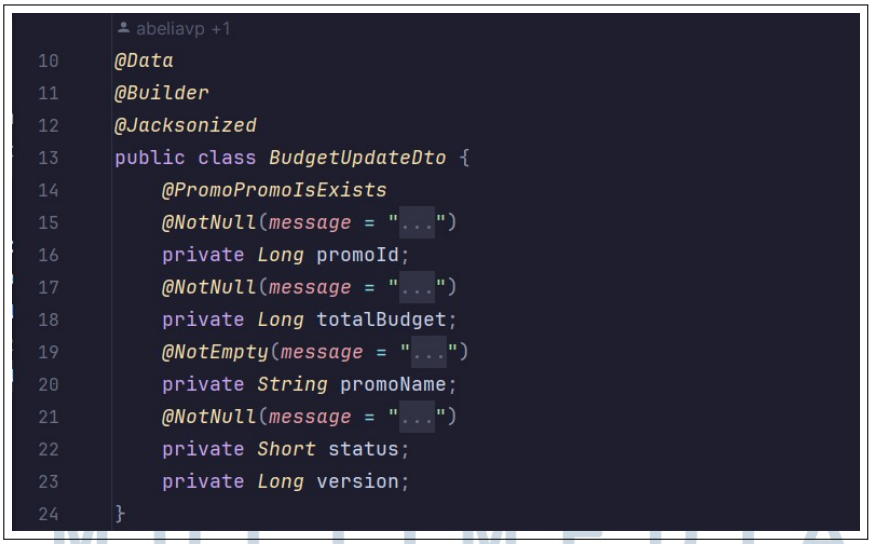
  DTO ini digunakan khusus untuk membuat *Budget* baru. Data yang terdapat pada DTO ini hanya mencakup atribut yang diperlukan untuk membuat *Budget* baru seperti *promoId*, *totalBudget*, *promoName*, dan *status*. Gambar 3.2 adalah isi kode dari *BudgetCreateDto*



Gambar 3.2. Isi *BudgetCreateDto*

- *BudgetUpdateDto*

  DTO digunakan dalam operasi update, di mana data yang diperlukan untuk memperbarui anggaran dikirim dari klien ke server. Seperti yang dapat dilihat pada Gambar 3.3, data yang ada dalam DTO ini adalah *promoId*, *totalBudget*, *promoName*, *status*, dan *version*.



Gambar 3.3. Isi *BudgetUpdateDto*

- *BudgetRequestDto*

  DTO ini digunakan dalam operasi pencarian atau pengambilan data anggaran berdasarkan kriteria tertentu yang dimasukkan oleh pengguna. Isi dari DTO

ini meliputi *promoId*, *totalBudget*, *promoName*, dan *status*. Gambar 3.4 adalah isi kode dari *BudgetRequestDto*

```
abeliavp
@Data
@Builder
@Jacksonized
public class BudgetRequestDto {
    private Long promoId;
    private Long totalBudget;
    private String promoName;
    private Integer status;
}
```

Gambar 3.4. Isi *BudgetRequestDto*

- *BudgetNameDto*
  DTO ini khusus untuk mengirimkan nama yang terkait dengan entitas Budget. Seperti yag dilihat pada Gambar 3.5, cakupan data *BudgetNameDto* hanya berupa *Id*, dan *promoName*.

```
abeliavp
@Data
public class BudgetNameDto {
    private Long id;
    private String promoName;
}
```

Gambar 3.5. Isi *BudgetNameDto*

B. Modul promo-service

Pada modul *promo-service*, pengembangan dilakukan pada *file-file* yang telah dibuat sebelumnya. *File-file* tersebut mencakup berbagai komponen penting, termasuk *entity*, *controller*, *repository*, *mapper*, serta *service*. Masing-masing file ini memiliki peran khusus yang berhubungan dengan proses CRUD. Berikut adalah penjelasan lebih rinci mengenai masing-masing *file*:

- *Entity*

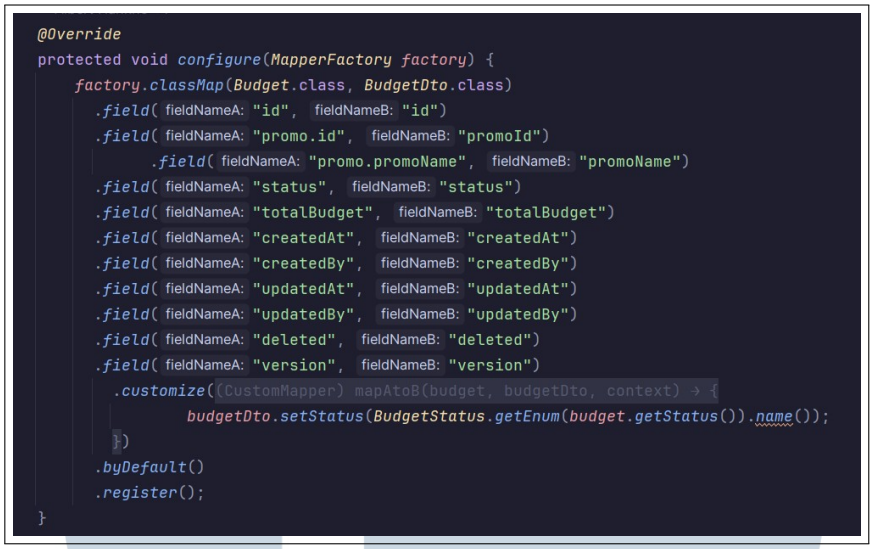
  Budget adalah kelas entitas yang mewakili tabel *Budget* dalam database. Kelas ini menggunakan anotasi JPA seperti @Entity dan @Table untuk memetakan kolom-kolom database ke atribut-atribut dalam kelas. Atribut-atribut akan menyimpan data yang sesuai dengan kolom dalam tabel Budget. Kelas ini juga dilengkapi dengan metode getter dan setter untuk mengakses dan memodifikasi nilai dari setiap atribut, serta dapat menggunakan anotasi validasi seperti @NotNull untuk memastikan bahwa data yang disimpan sesuai dengan aturan tertentu di dalam tabel database. Gambar 3.6 menunjukan isi dari *entity Budget*.

```
@OneToOne(targetEntity = Promo.class, fetch = FetchType.LAZY)
@JoinColumn(name = "promo_id", referencedColumnName = "id")
private Promo promo;
@Column(name = "total_budget")
private Long totalBudget;
@Column(name = "status")
private Short status;
@Column(name = "created_at", updatable = false)
@CreatedDate
private LocalDateTime createdAt;
@Column(name = "created_by", updatable = false)
@CreatedBy
private Long createdBy;
@Column(name = "updated_at")
@LastModifiedDate
private LocalDateTime updatedAt;
@Column(name = "updated_by")
@LastModifiedBy
private Long updatedBy;
private boolean deleted;
@Version
private Long version;
```

Gambar 3.6. Isi Budget.java

- *Mapper*

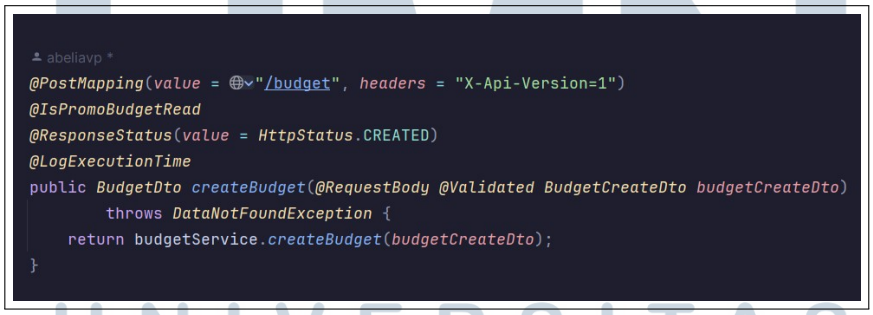
  BudgetMapper adalah *file* yang digunakan untuk mengkonversi antara *entity* dan DTO. Mapper mempermudah proses pemetaan data dari *entity* ke DTO yang lebih ringan dan lebih sesuai untuk ditransfer melalui jaringan, serta sebaliknya. *Mapper* membantu menjaga kode tetap bersih dan terorganisir dengan memisahkan logika pemetaan dari logika bisnis. Gambar 3.7 menunjukkan isi dari *BudgetMapper*.



Gambar 3.7. Isi BudgetMapper

- *Controller*

  BudgetController adalah kelas yang menangani permintaan HTTP untuk operasi CRUD yang berkaitan dengan *Budget*. Kelas ini menggunakan anotasi untuk memetakan URL ke metode pengendali yang sesuai. *Controller* ini bertanggung jawab untuk mengambil input dari permintaan, memvalidasi data tersebut, dan mengembalikan respons yang sesuai kepada klien. Gambar 3.8 adalah potongan kode yang berasal dari *BudgetController*.



Gambar 3.8. Isi *Controller* bagian metode *POST*

- *Repository*

  BudgetRepository berfungsi sebagai lapisan yang berkomunikasi langsung dengan *database* untuk operasi CRUD. Repository menyediakan metode untuk melakukan operasi CRUD (*Create*, *Read*, *Update*, *Delete*) pada entity *Budget*. Selain metode-metode standar, *BudgetRepository* juga dapat mendefinisikan metode khusus untuk *query* tertentu.

- *Service*

  BudgetService adalah kelas yang mengandung logika bisnis untuk mengelola *Budget*. *Service* memproses data yang diterima dari *BudgetController*, berinteraksi dengan *BudgetRepository* untuk mengakses *database*, dan mengembalikan hasilnya ke *BudgetController*. *Service* juga dapat mengandung validasi dan transformasi data yang diperlukan sebelum data disimpan atau dikembalikan ke pengguna. Gambar 3.9 adalah potongan kode yang berasal dari *BudgetService*.

```
abeliavp
@Transactional(value="promoTransactionManager")
public BudgetDto createBudget(BudgetCreateDto budgetCreateDto) throws DataNotFoundException {

    Budget budget = Budget
            .builder()
            .promo(promoRepository.getReferenceById(budgetCreateDto.getPromoId()))
            .totalBudget(budgetCreateDto.getTotalBudget())
            .status(budgetCreateDto.getStatus())
            .build();

    budget = budgetRepository.save(budget);
    return budgetMapper.map(budget, BudgetDto.class);
}
```

Gambar 3.9. Isi *BudgetService* bagian logika *createBudget*

C. Unit Test

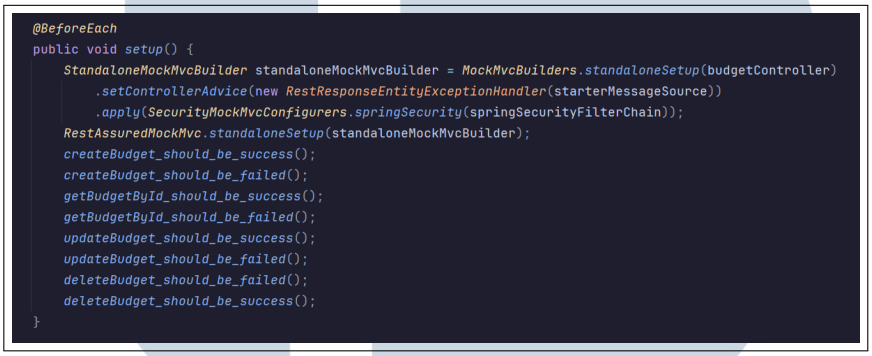
Unit test dibuat untuk memverifikasi bahwa seluruh kode berfungsi dengan benar. Dalam unit test, dilakukan simulasi dan ditentukan output yang diharapkan. Jika output yang dihasilkan tidak sesuai dengan yang diharapkan, ini menunjukkan adanya error pada kode. Error tersebut akan ditampilkan di terminal, sehingga sumber masalah dapat dengan mudah diidentifikasi dan diperbaiki.

Unit test juga berperan dalam dokumentasi kode, memberikan gambaran jelas tentang bagaimana fungsi tertentu seharusnya bekerja. Unit test mempermudah proses refactoring, karena dapat menunjukkan bahwa perubahan yang dilakukan tidak mempengaruhi fungsi lainnya. Dengan demikian, unit test tidak hanya meningkatkan keandalan kode, tetapi juga efisiensi pengembangan secara keseluruhan.

- *Unit Test Base*

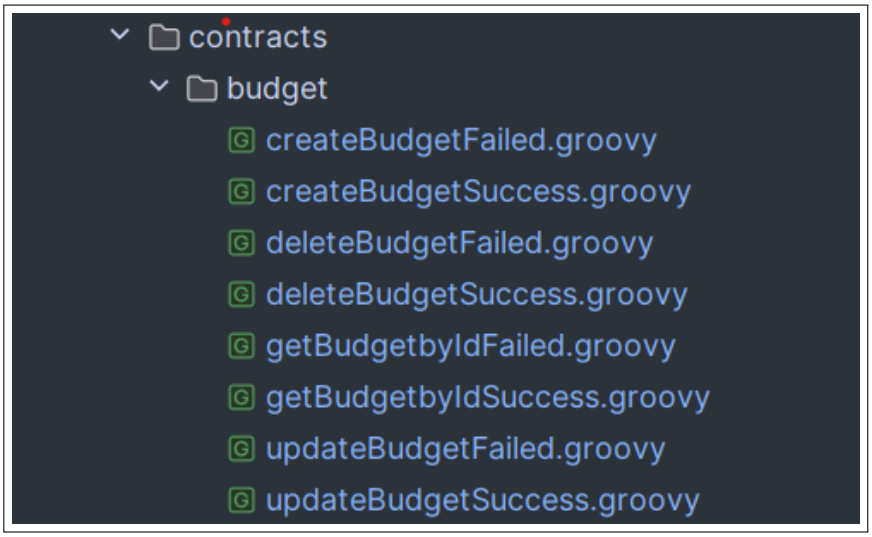
  Unit test base terhubung dengan *contract test* (*groovy*). Unit test base menyediakan fondasi untuk memastikan bahwa metode layanan bekerja dengan benar dengan melakukan *mocking* perilaku yang diharapkan

menggunakan *mockito*. Kemudian *contract test* akan memverifikasi bahwa API secara keseluruhan berfungsi sesuai dengan kontrak yang telah didefinisikan. Jika kontrak yang didefinisikan dalam *Groovy* tidak sesuai dengan perilaku yang diuji dalam *unit test base*, maka akan terjadi kegagalan (*failure*) dalam *contract test* karena API tidak memenuhi ekspektasi yang ditentukan. Gambar 3.10 menunjukkan bagan dari metode-metode yang ada dalam *Budget Base*.



Gambar 3.10. Metode yang terdapat pada *Budget Base*

Kemudian, Gambar 3.11 menunjukkan daftar *groovy* yang ada dalam contract.



Gambar 3.11. Struktur *groovy* pada *Unit Test Budget*

- *Service Test*

  Service test adalah pengujian yang berfokus pada logika bisnis utama yang diimplementasikan dalam *service* dari sebuah aplikasi. Tujuan utama dari

service test adalah memastikan bahwa *service* dalam aplikasi bekerja sesuai dengan yang diharapkan. *Service test* menguji metode-metode dalam *service* secara isolasi dengan menggunakan mock untuk dependensi eksternal seperti *repository* atau API eksternal. Gambar 3.12 menampilkan potongan kode *Service Test*.

```
abeliavp
@Test
void testDeleteBudget() {
    BudgetDto budgetDto = budgetService.deleteBudget( id: 4L);
    assertTrue(budgetDto.isDeleted());
}

abeliavp
@Test
void testDeleteBudgetThrowException() {
    Exception exception = assertThrows(DataNotFoundException.class, () -> {
        BudgetDto budgetDto = budgetService.deleteBudget( id: 10L);
    });

    String expectedMessage = "Budget dengan id 10 tidak ada";
    String actualMessage = exception.getMessage();

    assertTrue(actualMessage.contains(expectedMessage));
}
```

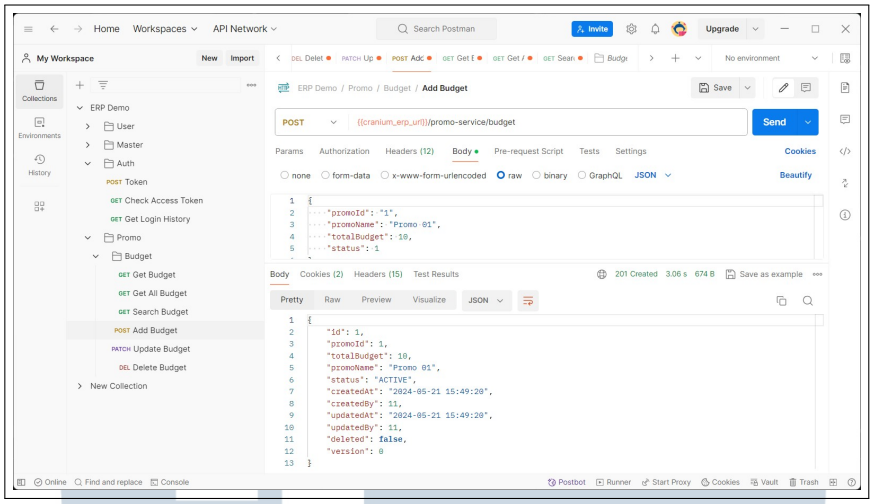
Gambar 3.12. Potongan kode pada *BudgetServiceTest*

D. Hasil CRUD pada Postman

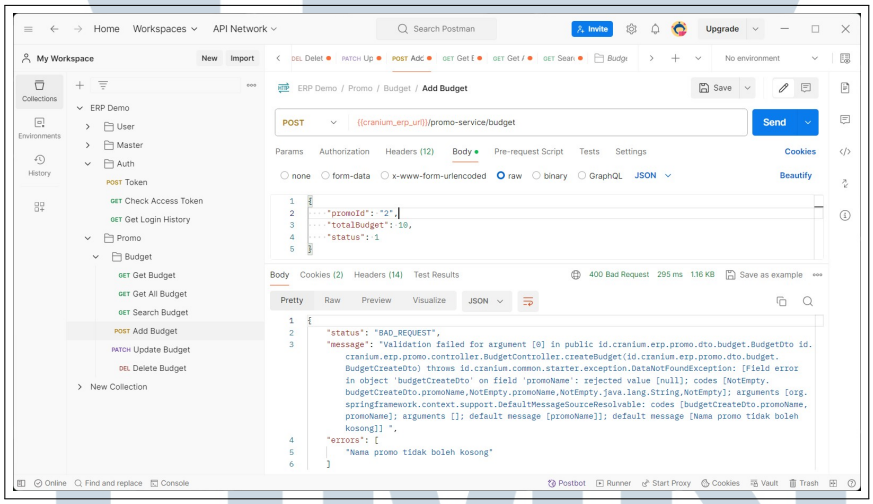
Sebelum melakukan pengecekan menggunakan Postman, langkah pertama yang perlu dilakukan adalah menyalakan *Spring Boot* pada modul *web-service*. Setelah modul aktif, langkah selanjutnya adalah mengirimkan *token* dari Postman. Dengan token ini, kita dapat melakukan berbagai jenis permintaan seperti *POST*, *PATCH*, *GET*, dan *DELETE*. Proses ini memastikan bahwa semua fitur API bekerja dengan baik dan sesuai dengan yang diharapkan. Berikut adalah penjelasan dan juga hasil dari pengecekan entitas *Budget*:

- POST

  Metode POST digunakan untuk mengirim data ke *server* dengan tujuan membuat *Budget* baru. Data yang dikirim melalui POST disertakan dalam badan permintaan. Server kemudian akan mengembalikan *status* keberhasilan dan data *Budget* baru yang telah dibuat seperti pada Gambar 3.13.



Gambar 3.13. Pembuatan *Budget* baru yang *success*

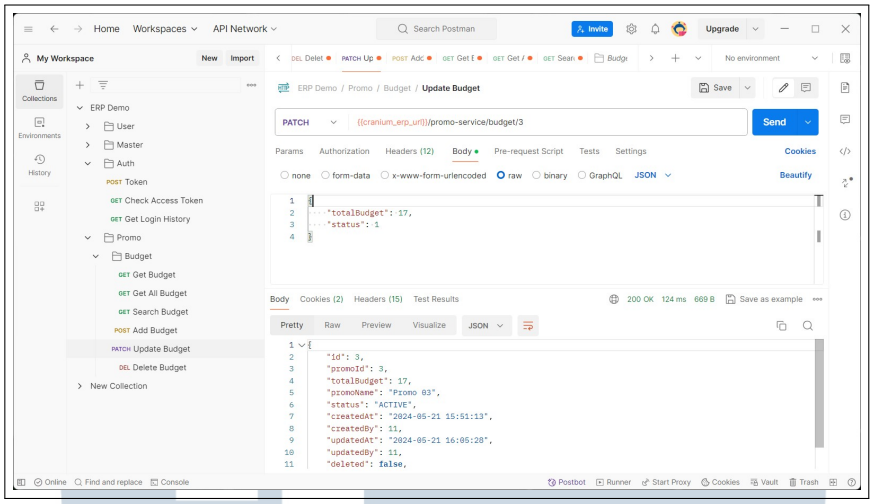


Gambar 3.14. Pembuatan *Budget* baru yang *failed*

Gambar 3.14 merupakan contoh pembuatan *Budget* yang sukses dan gagal. Pada pembuatan yang sukses, ditampilkan data *Budget* baru yang berhasil dibuat. Pada gambar tersebut dapat dilihat alasan pembuatan *Budget* gagal, yaitu dikarenakan *promoName* tidak diisi.

- *PATCH*

  Metode *PATCH* digunakan untuk memperbarui sebagian data dari *Budget* yang ada.

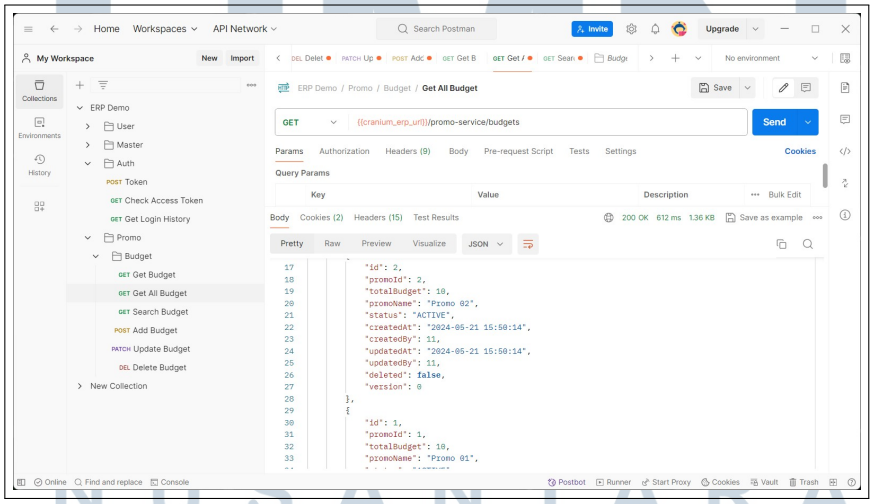


Gambar 3.15. Data *Budget* yang berhasil diperbarui menggunakan metode *PATCH*

Gambar 3.15 merupakan tampilan ketika proses *Update Budget* telah berhasil. Total *Budget* yang awalnya 10, diperbarui dan diubah menjadi 17.

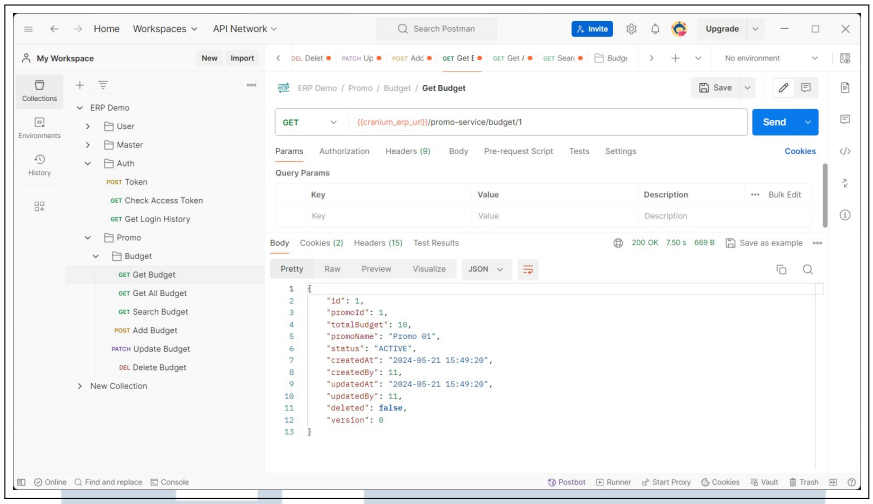
- *GET*

  Metode *GET* digunakan untuk meminta data dari *server* tanpa memodifikasi data tersebut. Pada entitas *Budget*, terdapat dua metode *GET* yang dibuat, yaitu *GET ALL* dan *GET By Id*. *GET ALL* berfungsi untuk menampilkan seluruh data *Budget* yang ada. Gambar 3.16 menunjukkan hasil data Budget yang ditampilkan menggunakan *GET ALL*.



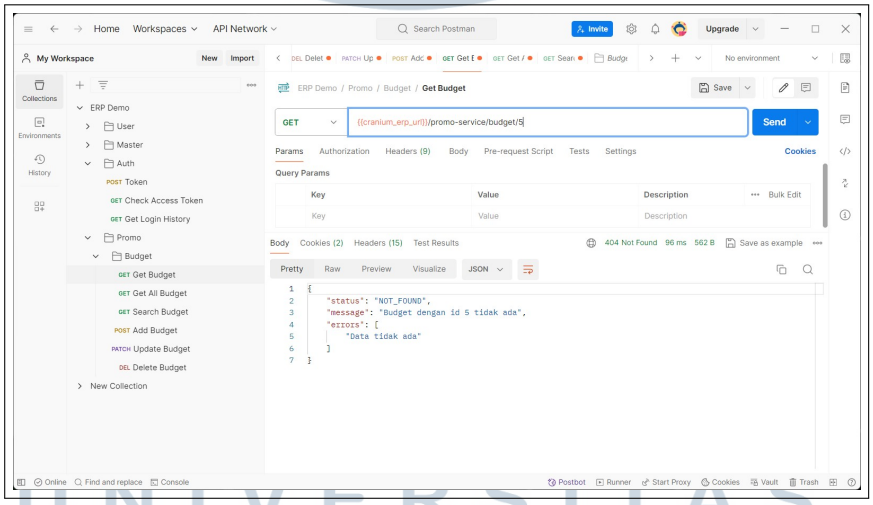
Gambar 3.16. Semua data *Budget* yang ditampilkan dengan GET ALL

GET By Id digunakan untuk menampilkan data *Budget* berdasarkan *Id* yang dimasukkan.



Gambar 3.17. Data *Budget* dengan *Id* 1

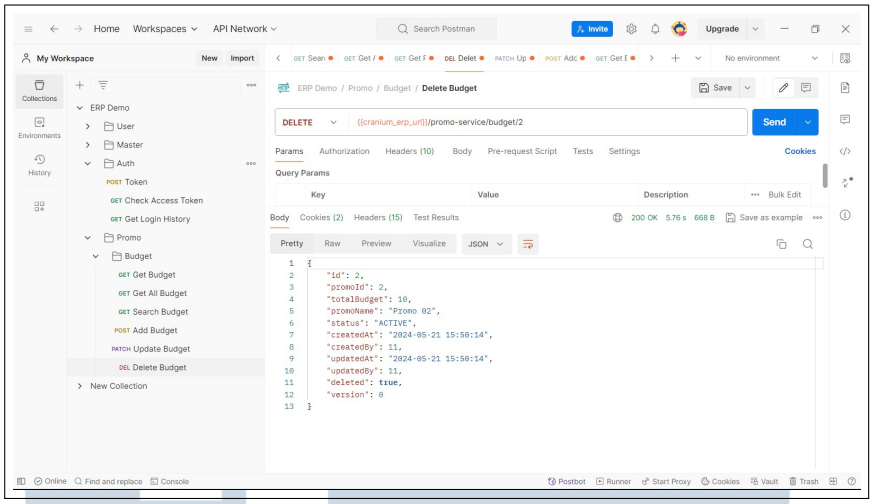
Gambar 3.17 menunjukkan isi data *Budget* dengan *Id* 1, sesuai dengan *Id* yang dimasukkan pada *path*. Data *Budget* dapat ditampilkan karena *Budget* dengan *Id* 1 ada dalam *database*. Seperti pada Gambar 3.18, Metode *GET By Id* juga bisa memunculkan pesan *error* jika *Budget* dengan *Id* yang dipanggil tidak ada dalam *database*.



Gambar 3.18. Tampilan ketika *Id* yang dimasukkan tidak ada dalam *database*

- *DELETE*

  Terakhir, metode *DELETE* digunakan untuk menghapus *Budget* yang ada dari *database*. *Request* berisi *Id Budget* akan dikirimkan ke *server*, kemudian *server* akan mengembalikan *status* keberhasilan sebagai bentuk konfirmasi bahwa *Budget* telah dihapus. Gambar 3.19 adalah tampilan ketika Budget berhasil dihapus.



Gambar 3.19. *Budget* dengan *Id* 2 telah terhapus

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Kendala yang muncul pada saat pengerjaan magang di PT Cranium Royal Aditama sebagai *fullstack developer* adalah berikut:

- Menghadapi kesulitan karena harus bekerja dengan bahasa pemrograman yang belum dikuasai dan cukup kompleks terutama saat pengerjaan unit test. Munculnya *error* yang terus-menerus dan sulit diselesaikan memakan waktu cukup lama.
- Komunikasi kurang lancar karena magang dilakukan 4 hari WFH (*Work From Home*) dan hanya 1 hari WFO (*Work From Office*).

3.4.2 Solusi

Adapun solusi yang dilakukan untuk menyelesaikan kendala yang muncul adalah sebagai berikut:

- Menjalani training selama tiga bulan untuk memahami bahasa pemrograman, serta meminta bantuan rekan magang dan menghubungi *mentor* untuk bertanya mengenai *error* yang muncul sehingga dapat diselesaikan dengan lebih cepat dan efektif
- Mengadakan pertemuan melalui aplikasi Discord untuk berdiskusi dan saling membantu menyelesaikan *error* yang terdapat pada kode masing-masing.