BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama kegiatan magang berlangsung, pekerjaan yang dilakukan dalam divisi *IT and Software Solution* yang merupakan bagian dari *Product Developer* adalah sebagai *Full Stack Developer*. Projek yang dikerjakan adalah mengembangkan sistem ERP yang berfokus pada perbaikan *bugs* dan pengembangan fitur serta *Business Logic* yang terdapat pada setiap fitur.

Praktik kerja magang dibantu oleh beberapa mentor dan supervisor yang memantau, mengarahkan, melakukan evaluasi pada pekerjaan yang sudah dilakukan. Dalam pengembangan sistem ERP terdapat 2 *supervisor* utama yang tim secara langsung yaitu sebagai berikut:

- Pak Sugito selaku *VP of Engineering* sekaligus *Tech Lead* yang bertanggung jawab mengawasi hasil kerja *developer* dalam business logic.
- Pak Angga selaku *Project Manager* yang bertanggung jawab dalam pembagian task untuk setiap *developer* dalam tim agar pengerjaan berjalan lancar.

Selain Supervisor, terdapat Mentor yang membantu *developer* baik dalam bidang *Frontend* maupun *Backend*. Mentor berperan dalam mengajarkan cara menulis *code* sesuai dengan aturan perusahaan dan juga mengawasi alur pekerjaan yang dilakukan *developer* melalui GitHub.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama kegiatan magang di PT Cranium Royal Aditama adalah mengembangkan sistem Enterprise Resource Planning menggunakan Java Spring Boot sebagai *Backend* dan ReactJS sebagai *Frontend*. Dengan menggunakan arsitektur Modular Monolitik, sistem terbagi menjadi beberapa modul yang bekerja independen tanpa berganti satu sama lain [11].

Terdapat beberapa jenis tugas yang diberikan oleh Supervisor kepada mahasiswa, seperti pembuatan sistem CRUD beserta unit test yang diperlukan, penambahan *business logic* pada fitur dan juga perbaikan berbagai *bugs*. Dalam

pengerjaan tugas, mahasiswa menggunakan beberapa *software, framework dan tools*. *Frontend* menggunakan Visual Studio Code sebagai *tools* serta NextJs dengan TypeScript sebagai *framework* dan *bahasa* yang digunakan [12], sedangkan *Backend* menggunakan IntelliJ sebagai *tools* untuk memudahkan pengembangan yang berbasis Java sebagai bahasa dan Spring Boot sebagai *framework* [13]. Selain itu, terdapat beberapa *tools* lain yang digunakan yaitu, DBeaver sebagai *database tool*, Postman yang digunakan untuk pengujian API endpoint lalu Git dan Github yang digunakan untuk melakukan *version control* pada sistem ERP.

3.3 Uraian Pelaksanaan Magang

3.3.1 Pelaksanaan Kerja Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Perkenalan perusahaan, Pengenalan lingkungan kantor, briefing aturan bekerja di kantor, instalasi software, setup software dan tools yang akan digunakan
2	Mulai pembelajaran Backend Menggunakan Spring Boot, Pembelajaran konsep Controller-Service-Repository, Data Transfer Object, Validasi serta CRUD pada Backend
3	Melanjutkan pembelajaran backend, struktur project ERP, Pembelajaran tentang konsep authorization, scope, dan message.
4	Melanjutkan pembelajaran backend, menambahkan Unit Test dan Contract Test
5	Review code training Backend, Evaluasi BE, Persiapan Training Frontend
6	Mulai Pembelajaran Frontend training ERP, instalasi software dan tools untuk FE.
7	Melanjutkan Pembelajaran FE, pembuatan halaman dan method untuk CRUD, menghubungkan FE dan BE menggunakan Endpoint yang sudah dibuat
	Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu Ke -	Pekerjaan yang dilakukan
8	Melanjutkan pembelajaran FE ERP, melanjutkan pembuatan halaman dan metode CRUD dan pembuatan unit test untuk FE.
9	Review training serta evaluasi BE & FE.
10	Onboarding dan briefing proyek ERP, serta setup untuk pengerjaan proyek
11	Pengerjaan tugas pada modul Master submodul Activity Standard dan Unit of Measurement. Menambahkan field code dan fitur autogenerate code pada FE dan BE.
12	Revisi sesuai dengan PR pada submodul Activity Standard dan Unit of Measurement.
13	Menambahkan business logic untuk generate Chart of Account pada Submodul Item
14	Melanjutkan pengerjaan generate CoA, fokus pada penyesuaian unit test dan revisi sesuai Pull Request
15	Mulai pengerjaan fitur CRUD untuk submodul baru yaitu Value Added Tax.
16	Melanjutkan pengerjaan Submodul VAT, menambahkan unit test dan Revisi sesuai PR.
17	Mulai pengerjaan pada Modul Selling, menambahkan webclient untuk komunikasi antara modul Master Dengan Selling.
18	Menambahkan *business logic* pada modul Selling submodul Order bagian BE, modifikasi database serta pengerjaan unit test
19	Menambahkan *business logic*, perhitungan, perubahan tampilan pada submodul Order bagian FE.
20	Mulai mengerjakan perubahan sistem *partial index* pada *database migration*.
21	Melanjutkan pengerjaan perubahan sistem *partial index* pada *database migration*, Revisi sesuai PR

3.3.2 Training ERP

Pelatihan ERP dilaksanakan selama 9 minggu pertama masa magang. Evaluasi terhadap hasil pelatihan dilakukan oleh supervisor pada minggu ke-5 untuk pengembangan *backend* dan minggu ke-9 untuk pengembangan *frontend*. Kegiatan pelatihan ini bertujuan untuk mengenalkan mahasiswa magang terhadap proses pengembangan sistem ERP secara menyeluruh. Selain itu, pelatihan ini juga berfokus pada pengenalan *tech stack* yang digunakan serta berbagai konsep penting yang akan digunakan selama proses pengembangan sistem ERP. Adapun beberapa konsep utama yang digunakan dalam pengembangan ERP adalah sebagai berikut:

1. *Controller - Service - Repository*:

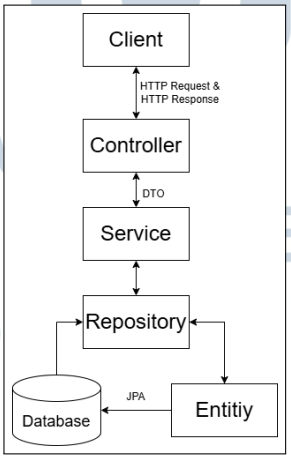
   Konsep ini merupakan bagian dari arsitektur berbasis *layered architecture* yang umum digunakan dalam pengembangan aplikasi berbasis Spring Boot

   - *Controller* adalah komponen yang berfungsi sebagai penghubung antara *client* (*frontend*) dengan sistem *backend*. *Controller* bertugas menerima permintaan HTTP *(HTTP Request)* dari *user*, kemudian memetakan permintaan tersebut ke metode tertentu dalam kelas *controller*. Setelah itu, *controller* akan mengirimkan respons yang sesuai dalam bentuk *HTTP Response*. Dengan kata lain, *controller* menjadi pintu masuk utama dalam alur kerja aplikasi
   - *Service* adalah lapisan logika bisnis yang berfungsi untuk memproses data dan menerapkan aturan bisnis. Komponen ini bertugas mengolah data yang diterima dari *controller*, melakukan perhitungan, validasi, atau proses lainnya, lalu meneruskan hasilnya ke *repository* atau kembali ke *controller*. Dengan adanya *service layer*, pengembang dapat memisahkan logika bisnis dari logika presentasi.
   - *Repository* merupakan lapisan yang bertanggung jawab atas interaksi langsung dengan *database*, sehingga memungkinkan aplikasi untuk melakukan operasi *CRUD (Create, Read, Update, Delete)* pada data. Dalam Java Spring Boot, *JPA Repository* yang disediakan secara otomatis menyediakan berbagai operasi dasar CRUD tanpa perlu menulis kode SQL secara manual. Selain itu, *repository* juga dapat dikembangkan untuk menangani kebutuhan khusus melalui *custom query*.

2. *Data Transfer Object(DTO), Entity, DTO Mapper*:

   Tiga komponen ini merupakan hal penting dalam pengembangan aplikasi berbasis Java Spring Boot yang bertujuan untuk memisahkan data internal dengan data yang dipertukarkan antar lapisan aplikasi.

   - *Entity* adalah objek yang merepresentasikan struktur data yang tersimpan dalam tabel pada *database*. *Entity* digunakan oleh aplikasi untuk mengelola data yang bersifat persisten. Setiap atribut dalam kelas *entity* memetakan kolom dalam tabel *database*, dan dapat mencakup relasi antar entitas, seperti *OneToMany, ManyToOne, dan ManyToMany*. *Entity* bekerja sama dengan Java Persistence API (JPA) untuk memfasilitasi proses pemetaan data *database* ke objek [14].
   - *Data Transfer Object* atau DTO adalah objek sederhana yang dirancang khusus untuk membawa data antar proses atau lapisan dalam sebuah aplikasi. DTO hanya berisi properti atau *field*, beserta metode *getter* dan *setter*, tanpa mengandung *business logic* di dalamnya. Tujuan utama penggunaan DTO adalah untuk mengelompokkan data yang dibutuhkan sehingga dapat dikirimkan dalam satu kali operasi. Selain itu, penggunaan DTO juga berfungsi sebagai lapisan perlindungan agar struktur internal *entity* tidak langsung terekspos ke luar [15].
   - *Mapper* adalah komponen yang berfungsi untuk mengonversi objek dari satu tipe ke tipe lainnya, khususnya antara DTO dan *Entity*. *Mapper* diperlukan karena struktur DTO dan *Entity* sering kali berbeda, baik dari segi data yang dibawa maupun formatnya.



Gambar 3.1. Controller-Service-Repository

Gambar 3.1 menggambarkan alur data dan hubungan antara lapisan *Controller, Service*, dan *Repository* dalam arsitektur aplikasi Java Spring Boot. Proses dimulai ketika *client* mengirim *HTTP request* melalui API, yang kemudian diterima oleh *Controller* untuk diproses sesuai dengan metode HTTP yang digunakan. Data dari *client* dikemas dalam bentuk *Data Transfer Object* (DTO) dan diteruskan ke *Service*. *Service* bertanggung jawab menjalankan *business logic* dan mengubah DTO menjadi *Entity* jika diperlukan. Setelah itu, data dikirim ke *Repository* untuk diproses lebih lanjut. *Repository* merupakan lapisan yang berinteraksi langsung dengan *database* dan bertugas melakukan operasi CRUD terhadap data dalam bentuk *Entity* menggunakan *JPA Repository* sebagai *Object-Relational Mapping* (ORM). Pendekatan ini memungkinkan pemisahan tanggung jawab yang jelas antar lapisan, sehingga mempermudah proses pengembangan dan pemeliharaan aplikasi.

3. *Unit Test dan Contract Test*

   Pengujian merupakan bagian penting dari proses pengembangan perangkat lunak untuk memastikan bahwa sistem bekerja sesuai harapan dan bebas dari bug. *Unit Test* dilakukan untuk menguji bagian terkecil dari program, seperti satu metode atau fungsi, secara terisolasi dari komponen lainnya. Tujuan dari unit testing adalah untuk memverifikasi bahwa setiap bagian kode (unit) bekerja dengan benar dalam berbagai kondisi. *Contract Test* adalah jenis pengujian yang memastikan bahwa komunikasi antara dua komponen,biasanya antara client (frontend) dan server (backend), tetap konsisten.

3.3.3 Ruang Lingkup Pekerjaan Magang

Melakukan pengembangan ERP sebagai berikut:

- Modul *Master*
  1. Submodul *Item*
  2. Submodul *Value Added Tax*
- Modul *Selling*
  1. Submodul *Order*

2. Submodul *Quotation*

- Melakukan modifikasi keseluruhan *database migration* pada setiap modul untuk menambahkan sistem partial index.

3.3.4 Pengembangan Modul Master

A Submodul Item

Setelah dilakukan *Quality Control* pada Submodul Item, ditemukan bahwa fitur *generate* atau pembuatan *Chart of Account* (CoA) khusus untuk *Item* belum diterapkan. Fitur ini berfungsi untuk membuat CoA yang bersifat spesifik untuk setiap *Item*, dimana *Item* tersebut akan menyimpan *Id* CoA tersebut sebagai *foreign key* pada kolom *inventory_coa_id* ke dalam *database*. Proses pembuatan CoA memerlukan penamaan khusus dengan menambahkan *prefix* ”Persediaan Barang Dagang” diikuti dengan nama *Item* yang bersangkutan pada nama dan deskripsi CoA dan *prefix* angka ”301333” pada kode diikuti dengan kode SKU dari *Item*.

Langkah pertama yang dilakukan adalah menghapus syarat *required* pada bagian *frontend* dan menghapus anotasi *@NotNull* pada DTO *CreateItem* backend, hal ini bertujuan agar Item tetap dapat dibuat meskipun belum memiliki relasi dengan CoA. Setelah itu, dilakukan penambahan *Endpoint* baru pada controller untuk menangani proses pembuatan CoA secara khusus.

```
1 @PatchMapping(value = "/item/{id}/coa", headers = "X-Api-Version=1")
2 @IsMasterItemUpdate
3 @ResponseStatus(value = HttpStatus.OK)
4 @LogExecutionTime
5 public ItemDto updateCoaItem(@Validated @RequestBody ItemUpdateCoaDto itemUpdateCoaDto, @PathVariable Long id
      ) throws DataNotFoundException, DataLockException {
6     return itemService.updateCoaItem(id, itemUpdateCoaDto);
7 }
```

Kode 3.1: Controller Item Generate COA

Potongan kode 3.1 merupakan bagian kode yang digunakan untukl membuat *endpoint* pada *controller*. *Endpoint* ini menggunakan metode PATCH untuk melakukan proses *Update* pada item berdasarkan ID yang diterima melalui *endpoint*. Selain itu terdapat annotasi *@IsMasterItemUpdate* yang berfungsi untuk memverivikasi apakah *user* yang melakukan metode *generate* CoA ini memiliki *role* yang sesuai untuk melakukan *Update* terhadap *Item*. Selanjutnya pemanggilan *return itemService.updateCoaItem* digunakan untuk mengeksekusi *business logic* melalui *service layer* yang menangani proses *generate* CoA.

```java
@Transactional(value = "masterTransactionManager")
public ItemDto updateCoaItem(Long id, ItemUpdateCoaDto itemUpdateCoaDto) throws EntityNotFoundException,
        DataNotFoundException, DataLockException {
    Optional<Item> itemOptional = Optional.empty();
    Optional<String> myScope = starterUserScopeService.getMyScopeValue("MASTER_ITEM_OWN");
    if (myScope.isPresent()) {
        if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))) {
            itemOptional = itemRepository.findWithLockingPessimisticByIdAndCreatedByAndDeletedFalse(id,
     UserAuthInfo.getUserId());
        } else {
            itemOptional = itemRepository.findWithLockingByIdAndDeletedFalse(id);
        }
    }
    if (itemOptional.isEmpty()) {
        throw new DataNotFoundException(masterMessageSource.getMessage(MASTER_ITEM_FINDID_NOTFOUND, new
     Object[]{id}, LocaleContextHolder.getLocale()));
    }
    Item item = itemOptional.get();
    if (!item.getVersion().equals(itemUpdateCoaDto.getVersion())) {
        throw new DataLockException("Database version: " + item.getVersion() + ", Current version: " +
     itemUpdateCoaDto.getVersion());
    }

    ChartOfAccountCreateDto chartOfAccountCreateDto = ChartOfAccountCreateDto.builder()
            .accountTypeId(AccountTypeId.INVENTORY.getValue())
            .coaCode("301333." + item.getSku())
            .coaName("Persediaan Barang Dagang " + item.getItemName())
            .description("Persediaan Barang Dagang " + item.getItemName())
            .completeCode("301333." + item.getSku())
            .status((short) ChartOfAccountStatus.ACTIVE.getValue())
            .debitCreditAccountType(1)
            .setStartingBalance(false)
            .build();

    ChartOfAccountDto chartOfAccountDto = coaService.createChartOfAccount(chartOfAccountCreateDto);

    item.setInventoryCoa(coaRepository.getReferenceById(chartOfAccountDto.getId()));
    item = itemRepository.save(item);

    return entityToDto(item);
}
```

Kode 3.2: Generate COA Service

Potongan Kode 3.2 digunakan untuk menangani proses *generate* CoA. Dalam potongan kode ini, dilakukan beberapa pengecekan awal, seperti validasi terhadap *scope* akses dan pengecekan apakah Item yang akan diperbarui benar-benar ada di dalam database. Selain itu, juga dilakukan pengecekan apakah *version* Item yang dikirim sesuai dengan versi Item yang tersimpan di *database* untuk menghindari konflik data.

Proses pembuatan CoA diawali dengan membentuk nilai-nilai untuk beberapa *field*, yaitu *coaCode, coaName, description, dan completeCode*, yang dibangun berdasarkan *prefix* khusus yang telah ditentukan. Setelah itu, data CoA tersebut disimpan ke dalam database dengan memanggil *service* yang bertugas untuk melakukan penyimpanan CoA. Langkah terakhir adalah memperbarui *field inventoryCoa* pada Item dengan menggunakan *itemRepository.save*, yang melakukan operasi update pada *database* untuk menyimpan perubahan yang telah

dilakukan.

Pada bagian frontend, perlu dilakukan penambahan file atau fungsi untuk memanggil *API endpoint* yang baru saja dibuat, sebagaimana ditunjukkan pada Potongan Kode 3.3

```
    type UpdateItemCoaOptions = {
  id: number;
  data: UpdateItemCoaData;
};

export const updateItemCoa = ({
  id,
  data,
}: UpdateItemCoaOptions): Promise<Item> => {
  return getApiClient().patch(
    `/master-service/item/${id}/coa`,
    data
  );
};

type UseUpdateItemCoaOptions = {
  onSuccess?: (item: Item) => void;
  onError?: (error: ResponseError) => void;
};

export const useUpdateItemCoa = ({
  onSuccess,
  onError,
}: UseUpdateItemCoaOptions = {}) => {
  const {
    mutate: submit,
    isLoading,
    isError,
    isSuccess,
  } = useMutation({
    mutationFn: updateItemCoa,
    onSuccess: (item) => {
      queryClient.invalidateQueries(['master.items']);
      queryClient.invalidateQueries([
        'master.chartOfAccounts',
      ]);
      onSuccess?.(item);
    },
    onError: (error) => {
      const err = customError(error);
      onError?.(err);
    },
  });

  return { submit, isLoading, isError, isSuccess };
};
```

Kode 3.3: Metode untuk memanggil API generate Item CoA

Selanjutnya, pada komponen *View Item*, perlu ditambahkan tombol yang akan memanggil metode untuk melakukan *generate* CoA dengan cara mengirim permintaan ke *API endpoint* yang telah disiapkan. Tombol tersebut hanya ditampilkan jika item yang sedang dilihat saat ini tidak memiliki CoA.

```
{!item.inventoryCoaId &&
isAuthorized('MASTER_ITEM_UPDATE') && (
    <Button
      data-testid="saveButton"
      onClick={() =>
        handleOpenModal({
          modalName: 'generate',
        })
      }
      variant={'contained'}
      id={styles.saveButton}
    >
      {t('master.item.coa.generate', {
        ns: 'master',
      })}
    </Button>
)}
```

Kode 3.4: Tombol Generate Coa

Potongan Kode 3.4 menunjukkan implementasi tombol yang digunakan untuk menampilkan modal konfirmasi sebelum melakukan proses generate CoA. Pada kode tersebut, kondisi *!item.inventoryCoaId* digunakan untuk menyembunyikan tombol apabila Item sudah memiliki CoA, sehingga tombol hanya muncul untuk Item yang belum terhubung dengan CoA.

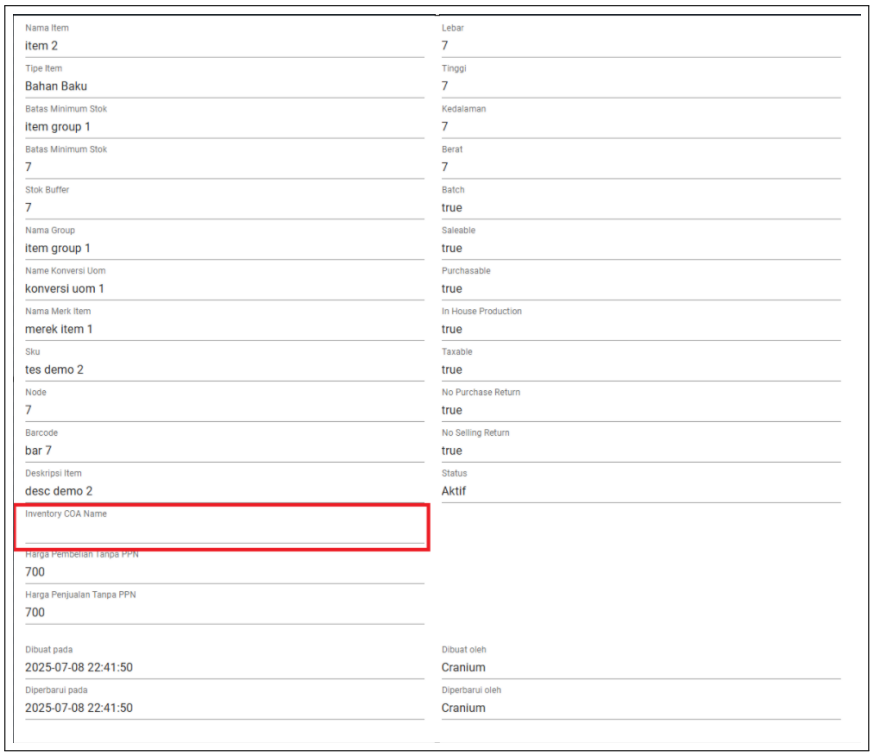
```
<ModalComponent
  open={openModal === 'generate'}
  onClose={handleCloseModal}
  title={t('master.item.coa.confirmation', {
    ns: 'master',
  })}
  description={t(
    'master.item.coa.confirmationText',
    {
      ns: 'master',
    }
  )}
  confirmText={t('common.yes', {
    ns: 'common',
  })}
  onCancel={handleCloseModal}
  onConfirm={() => {
    if (!isConfirmDisabled) {
      handleGenerateChartOfAccount(item.id);
    }
  }}
></ModalComponent>
```

Kode 3.5: Modal Generate CoA

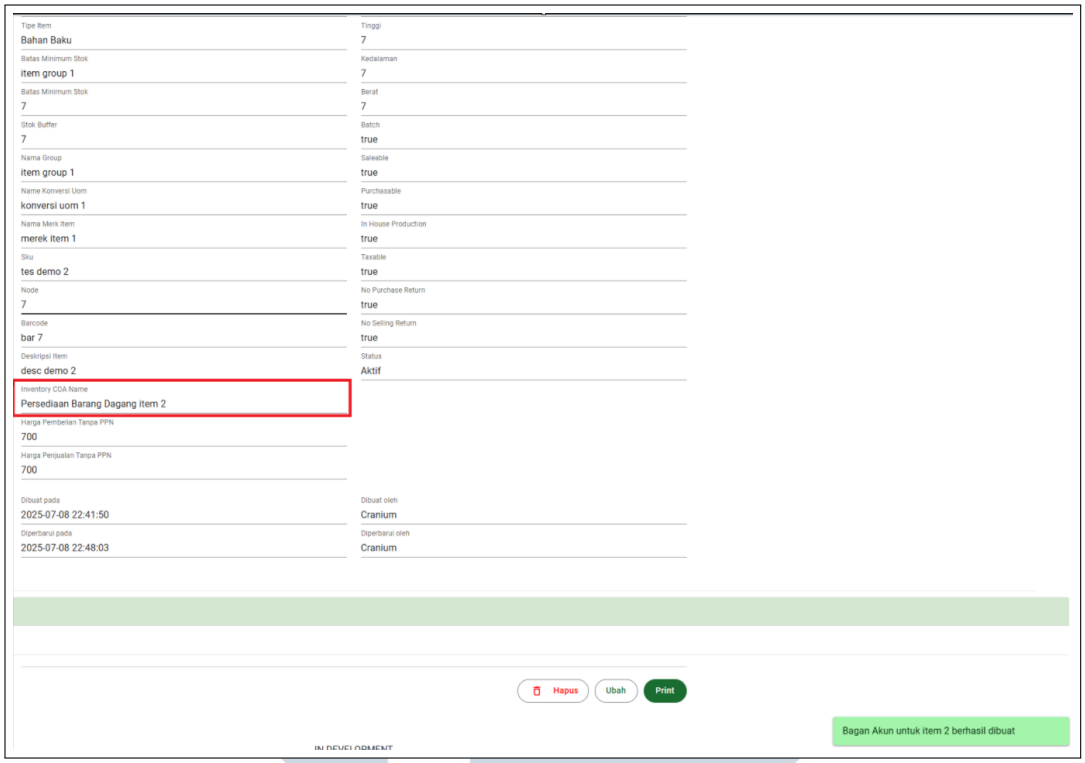
```
const updateItemCoa = useUpdateItemCoa({
    onSuccess: (item) => {
      onSuccessGenerate(item);
      handleCloseModal();
    },
    onError,
  });

  const handleGenerateChartOfAccount = (id: number) => {
    const data: UpdateItemCoaData = {
      version: item.version,
    };
    updateItemCoa.submit({
      id,
      data,
    });
  };
```

Kode 3.6: Metode Handle Generate CoA

Tampilan konfirmasi ditampilkan melalui *modal* yang dibuat menggunakan Potongan kode 3.5. Apabila pengguna menekan tombol ”Ya”, maka *modal* akan memanggil fungsi *handler* yang bertugas mengirimkan permintaan ke API menggunakan metode yang telah dibuat sebelumnya, sebagaimana ditunjukkan pada Potongan Kode 3.6. Jika proses pembuatan CoA berhasil, sistem akan menampilkan notifikasi pada bagian bawah halaman sebagai umpan balik *(feedback* kepada pengguna.



Gambar 3.2. Halaman detail *Item* sebelum CoA dibuat.



Gambar 3.3. Halaman detail *Item* Setelah CoA dibuat.

Selain itu, modifikasi juga dilakukan pada sisi *frontend*, khususnya pada halaman *Create* dan *Update*, dengan menghapus validasi serta properti *required* pada kolom CoA. Perubahan ini memungkinkan *field* CoA pada *Item* dikosongkan saat proses *Create* maupun *Update* dilakukan. Penyesuaian juga dilakukan terhadap *unit test* di sisi *frontend* untuk memastikan perubahan tersebut tetap berjalan sesuai fungsinya.

B Submodul Value Added Tax

Value Added Tax (VAT) merupakan submodul yang digunakan untuk menyimpan nilai pajak atau PPN (Pajak Pertambahan Nilai). Nilai PPN ini umumnya tidak digunakan langsung dalam Modul *Master*, melainkan lebih banyak digunakan pada modul-modul yang melibatkan perhitungan harga, seperti *Selling* dan *Purchasing*. Tabel skema database Value Added Tax dapat dilihat pada gambar 3.4.

Gambar 3.4. Skema Database Value Added Tax

Karena fungsinya yang lebih banyak digunakan oleh modul lain, serta mempertimbangkan arsitektur sistem ERP yang menerapkan pendekatan *modular monolitik* yang mengutamakan kemandirian setiap modul ,maka modul lain perlu mengakses data dari modul VAT secara eksternal. Akses ini dilakukan melalui pemanggilan endpoint API milik modul VAT menggunakan *WebClient*, sehingga dapat dikatakan bahwa proses ini merupakan bentuk dari *API-to-API communication*.

B.1 Permbuatan *Rest API* Value Added Tax (VAT)

Pembuatan metode CRUD untuk modul VAT diawali dengan pembuatan tabel pada *database*, yaitu *value_added_tax* dan *value_added_tax_aud*, melalui *migration* file yang disesuaikan dengan skema tabel yang telah dirancang sebelumnya. Tabel *value_added_tax_aud* berfungsi untuk mencatat seluruh perubahan yang terjadi pada data. Setelah pembuatan tabel, langkah selanjutnya adalah membuat *Entity* sebagai representasi objek yang mencerminkan struktur data pada tabel di *database*. Kemudian DTO dibuat dengan disesuaikan dengan masing-masing kebutuhan CRUD, dengan *ValueAddedTaxDto* sebagai DTO utama yang digunakan untuk mengirimkan *response* ke *client*. *Mapper* juga dikembangkan

untuk memetakan data dari *Entity* ke dalam DTO. Selain itu, *enumeration* dibuat untuk merepresentasikan *status* dari VAT. Sebagai tambahan, *validator* dan *annotation* khusus juga diterapkan untuk memastikan keamanan data saat proses manipulasi, sehingga keamanan sistem tetap terjaga.

Selanjutnya, dibuat juga *entity* khusus yaitu *ValueAddedTaxName* beserta *ValueAddedTaxNameDto*, yang dirancang khusus untuk mengambil dan menyajikan data berupa nama dari VAT saja. Hal ini digunakan ketika hanya informasi nama yang diperlukan.

Untuk membangun *endpoint*, *business logic*, dan komunikasi dengan *database*, perlu dibuat *ValueAddedTaxController, ValueAddedTaxService*, dan *ValueAddedTaxRepository*. Selain itu, juga dibuat *VatSpecification* yang berfungsi untuk menangani fitur pencarian (*searching*) berdasarkan *field* tertentu.

B.1.1 Metode *Create* VAT

Pada proses *Create* VAT, data dikirimkan oleh client menggunakan metode POST ke endpoint "*master-service/value-added-tax*" dengan format JSON pada bagian *request body*. Setelah data dikirimkan, aplikasi terlebih dahulu akan memeriksa apakah *user* telah terautentikasi dengan memvalidasi keberadaan *token*, jika *token* tidak ditemukan atau tidak *valid*, maka aplikasi akan mengembalikan *response 401 Unauthorized*. Selanjutnya, dilakukan pengecekan terhadap otorisasi (*authority*) *user*. Jika *user* tidak memiliki hak akses untuk melakukan *create* VAT, maka aplikasi akan mengembalikan *response 403 Forbidden*.

```
1 @PostMapping(value = "/value-added-tax", headers = "X-Api-Version=1")
2 @IsMasterValueAddedTaxCreate
3 @MasterValueAddedTaxNameIsUnique
4 @ResponseStatus(value = HttpStatus.CREATED)
5 @LogExecutionTime
6 public ValueAddedTaxDto createValueAddedTax(@RequestBody @Validated ValueAddedTaxCreateDto vatCreateDto){
7     return vatService.saveValueAddedTax(vatCreateDto);
8 }
```

Kode 3.7: VAT Create Controller

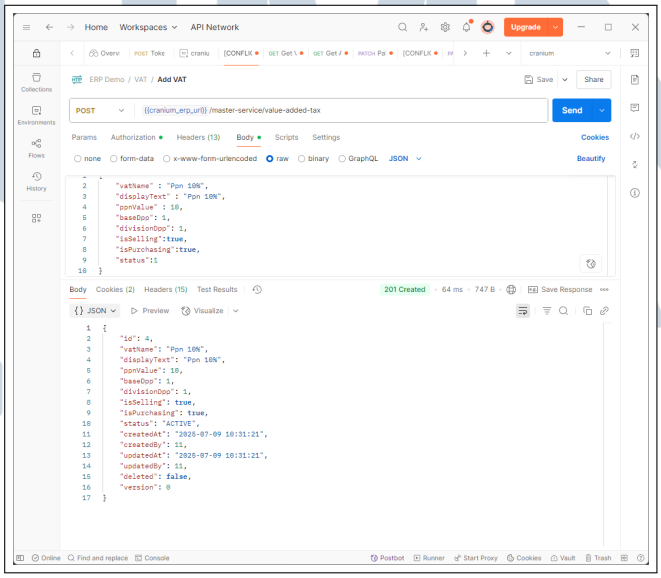
Apabila proses autentikasi dan otorisasi berhasil dilewati, maka aplikasi akan melanjutkan ke tahap pengecekan validitas data di dalam *ValueAddedTaxController*. Pengecekan pertama dilakukan terhadap keunikan nama VAT karena *field vat_name* memiliki atribut *unique* di dalam *database*. Validasi ini dilakukan menggunakan anotasi kustom *@MasterValueAddedTaxNameIsUnique*, yang berfungsi untuk memastikan bahwa tidak terdapat entri nama VAT yang sama yang belum dihapus (*soft delete*). Jika nama VAT dinyatakan belum digunakan

sebelumnya, maka sistem akan melanjutkan ke tahap pengecekan struktur data apakah sesuai dengan yang didefinisikan dalam *ValueAddedTaxCreateDto*. Jika struktur data tidak sesuai, *response 400 Bad Request* akan dikirimkan.

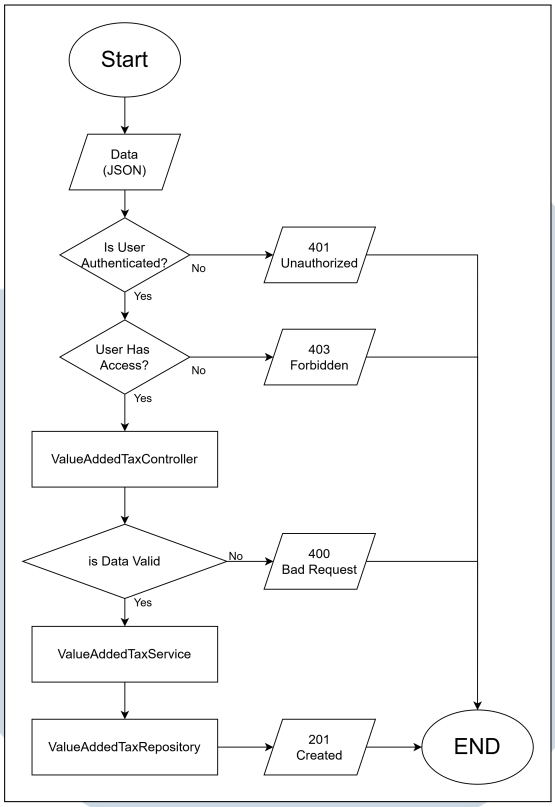
```
@Transactional(value="masterTransactionManager")
public ValueAddedTaxDto saveValueAddedTax(ValueAddedTaxCreateDto vatCreateDto) {

    ValueAddedTax vat = ValueAddedTax
            .builder()
            .vatName(vatCreateDto.getVatName())
            .displayText(vatCreateDto.getDisplayText())
            .ppnValue(vatCreateDto.getPpnValue())
            .baseDpp(vatCreateDto.getBaseDpp())
            .divisionDpp(vatCreateDto.getDivisionDpp())
            .isSelling(vatCreateDto.getIsSelling())
            .isPurchasing(vatCreateDto.getIsPurchasing())
            .status(vatCreateDto.getStatus())
            .build();
    vat = vatRepository.save(vat);
    return vatMapper.map(vat, ValueAddedTaxDto.class);
}
```

Kode 3.8: VAT Create Service

Apabila seluruh validasi telah berhasil dilewati, data akan diteruskan ke *ValueAddedTaxService* untuk diproses sesuai dengan *business logic*. Di dalam *service*, data dalam bentuk DTO akan dikonversi menjadi *entity* untuk disimpan ke dalam *database* melalui *ValueAddedTaxRepository*. Setelah proses penyimpanan berhasil, entity tersebut akan dikonversi kembali ke dalam bentuk *ValueAddedTaxDto* menggunakan *mapper* dan dikirimkan kembali ke *client* dalam *response body* dengan *response 201 Created* sebagai penanda bahwa data telah berhasil ditambahkan.



Gambar 3.5. Pengujian *Endpoint* menggunakan Postman untuk Create VAT



Gambar 3.6. Flowchart VAT Create

B.1.2 Metode *Read* VAT

Metode *Read* untuk mendapatkan data VAT terdiri dari tiga jenis sesuai dengan kebutuhannya masing-masing. Meskipun memiliki fungsi dan response yang berbeda, ketiga metode ini mengikuti proses dasar yang serupa. Setelah *client* mengirimkan permintaan menggunakan metode GET ke *endpoint* yang dituju, aplikasi akan terlebih dahulu melakukan pengecekan autentikasi dengan memverifikasi keberadaan *token*. Jika *token* tidak ditemukan atau tidak *valid*, maka aplikasi akan mengembalikan *response 401 Unauthorized*. Setelah lolos dari tahap autentikasi, sistem akan melakukan pengecekan otorisasi untuk memastikan bahwa pengguna memiliki hak akses terhadap data VAT. Jika pengguna tidak memiliki otorisasi yang sesuai, maka *response 403 Forbidden* akan dikembalikan. Jika proses autentikasi dan otorisasi tidak mengalami kendala, maka permintaan akan diteruskan ke *service* layer untuk diproses lebih lanjut. Adapun ketiga jenis metode GET yang tersedia adalah sebagai berikut:

- *Get VAT By Id* dengan *endpoint* ”*master-service/value-added-tax/*{*id*}”

Metode ini digunakan untuk mengambil satu data VAT berdasarkan ID yang diminta *user*.

- *Get All VAT* dengan *endpoint* "*master-service/value-added-taxes*" Metode ini digunakan untuk mengambil seluruh data VAT dalam bentuk *list*. *Endpoint* ini mendukung penggunaan *query parameters* dan fitur *pagination (Pageable)* untuk melakukan pencarian atau penyaringan data secara lebih spesifik dan efisien. Proses ini memungkinkan pengguna mencari data VAT berdasarkan kriteria tertentu seperti nama dan status yang ditentukan melalui struktur *ValueAddedTaxRequestDto* yang diperoleh dari *query parameters*.
- *Get VAT Names* dengan *endpoint* "*master-service/value-added-tax-names*" Metode ini digunakan untuk mengambil daftar ID dan nama dari VAT yang tersedia. Umumnya digunakan untuk menampilkan pilihan nama VAT dalam bentuk*dropdown*.

```
@GetMapping(value = "/value-added-tax/{id}", headers = "X-Api-Version=1")
@IsMasterValueAddedTaxRead
@ResponseStatus(value = HttpStatus.OK)
@LogExecutionTime
public ValueAddedTaxDto findValueAddedTaxById(@PathVariable Long id) throws DataNotFoundException {
    return vatService.findValueAddedTaxById(id);
}
```

Kode 3.9: Controller VAT GetById

```
@GetMapping(value = "/value-added-taxes", headers = "X-Api-Version=1")
@IsMasterValueAddedTaxRead
@ResponseStatus(value = HttpStatus.OK)
@LogExecutionTime
public Page<ValueAddedTaxDto> getAllValueAddedTaxes(Pageable pageable,
                                                    @RequestParam(required = false) String vatName,
                                                    @RequestParam(required = false) String displayText,
                                                    @RequestParam(required = false) Boolean isSelling,
                                                    @RequestParam(required = false) Boolean isPurchasing,
                                                    @RequestParam(required = false) Integer status
                                                    ){
    ValueAddedTaxRequestDto vatRequestDto = ValueAddedTaxRequestDto
            .builder()
            .vatName(vatName)
            .displayText(displayText)
            .isSelling(isSelling)
            .isPurchasing(isPurchasing)
            .status(status)
            .build();
    return vatService.getAllValueAddedTax(pageable, vatRequestDto);
}
```

Kode 3.10: Controller Get All VAT

```
@GetMapping(value = "/value-added-tax/value-added-tax-names", headers = "X-Api-Version=1")
@IsMasterValueAddedTaxRead
@ResponseStatus(value = HttpStatus.OK)
@LogExecutionTime
public List<ValueAddedTaxNameDto> findAllValueAddedTaxNames(){
    return vatService.getAllValueAddedTaxName();
}
```

Kode 3.11: Controller Get VatNames

Tahap pertama yang dilakukan pada layer *service* adalah melakukan pengecekan *scope* yang dimiliki oleh *pengguna*, untuk membatasi data yang dapat diakses sesuai dengan hak akses yang diberikan. Setelah pengecekan *scope* dilakukan, data akan diambil dari *repository* berdasarkan cakupan akses tersebut. Karena ERP Cranium menggunakan pendekatan *soft delete* dalam penghapusan data, maka pengambilan data dari *repository* harus menyertakan kondisi bahwa kolom deleted memiliki nilai false.

Khusus untuk metode Get All VAT, perlu dibuat *Specification* untuk mendukung fitur *searching* berdasarkan kriteria yang terdapat dalam *ValueAddedTaxRequestDto*. *Pageable* dan *Specification* merupakan fitur yang disediakan oleh Spring JPA yang memungkinkan pencarian dan penyortiran data secara fleksibel. Nantinya, *Pageable* dan *Specification* ini akan digunakan dalam *repository*, di mana *query* tambahan seperti *ORDER BY* dan *WHERE* akan dihasilkan secara otomatis. *Specification* dapat dilihat pada Potongan Kode 3.12

```
public class VatSpecification<T> {

    public Specification<T> build (ValueAddedTaxRequestDto valueAddedTaxRequestDto){
        SpecSearchCriteria<T> specSearchCriteria = new SpecSearchCriteria<T>();
        Specification<T> spec = Specification.where(specSearchCriteria.getSpecification(new SearchCriteria("
      deleted", false, SearchOperation.EQUAL)));

        if(!Objects.isNull(valueAddedTaxRequestDto.getVatName())){
            spec = spec.and(specSearchCriteria.getSpecification(new SearchCriteria("vatName",
      valueAddedTaxRequestDto.getVatName(), SearchOperation.STARTS_WITH)));
        }
        if(!Objects.isNull(valueAddedTaxRequestDto.getDisplayText())){
            spec = spec.and(specSearchCriteria.getSpecification(new SearchCriteria("displayText",
      valueAddedTaxRequestDto.getDisplayText(), SearchOperation.STARTS_WITH)));
        }
        if(!Objects.isNull(valueAddedTaxRequestDto.getIsSelling())){
            spec = spec.and(specSearchCriteria.getSpecification(new SearchCriteria("isSelling",
      valueAddedTaxRequestDto.getIsSelling(), SearchOperation.EQUAL)));
        }
        if(!Objects.isNull(valueAddedTaxRequestDto.getIsPurchasing())){
            spec = spec.and(specSearchCriteria.getSpecification(new SearchCriteria("isPurchasing",
      valueAddedTaxRequestDto.getIsPurchasing(), SearchOperation.EQUAL)));
        }
        if(!Objects.isNull(valueAddedTaxRequestDto.getStatus())){
            spec = spec.and(specSearchCriteria.getSpecification(new SearchCriteria("status",
      valueAddedTaxRequestDto.getStatus(), SearchOperation.EQUAL)));
        }
        return spec;
    }
}
```

Kode 3.12: Vat Specification

Setiap metode *Read* akan memberikan response yang berbeda-beda, tergantung pada jenis permintaan yang dilakukan, antara lain:

- *Get VAT by Id* akan mengembalikan satu data berdasarkan ID yang diminta user dalam bentuk *ValueAddedTaxDto*. Jika data tidak ditemukan atau nilai kolom *deleted* adalah *true*, maka aplikasi akan mengembalikan *response 404 Not Found*.
- *Get All VAT* akan mengembalikan *Page* yang berisi data VAT yang telah dikonversi ke dalam bentuk DTO menggunakan *mapper* dengan *response 200 OK*. Jika tidak terdapat data dalam *database* maka akan tetap mengembalikan *Page* dengan isi *List* kosong dan tetap memiliki *response 200 OK*.
- *Get VAT Names* akan mengembalikan daftar data ID dan nama dari VAT dalam bentuk*List* dengan *response 200 OK*. Jika tidak terdapat data apapun maka akan tetap mengembalikan *List* kosong dan tetap memiliki *response 200 OK*.

```
1  @Transactional(value="masterTransactionManager", readOnly = true)
2  public ValueAddedTaxDto findValueAddedTaxById(Long id) throws DataNotFoundException {
3      Optional<ValueAddedTax> vatOptional = Optional.empty();
4      Optional<String> myScope = starterUserScopeService.getMyScopeValue("MASTER_VAT_OWN");
5      if (myScope.isPresent()){
6          if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))){
7              vatOptional = vatRepository.findByIdAndCreatedByAndDeletedFalse(id, UserAuthInfo.getUserId());
8          } else {
9              vatOptional = vatRepository.findByIdAndDeletedFalse(id);
10         }
11     }
12
13     if(vatOptional.isEmpty()){
14         throw new DataNotFoundException(masterMessageSource.getMessage(MASTER_VAT_FINDID_NOTFOUND, new Object
       []{id}, LocaleContextHolder.getLocale()));
15     }
16
17     return vatMapper.map(vatOptional.get(), ValueAddedTaxDto.class);
18 }
```

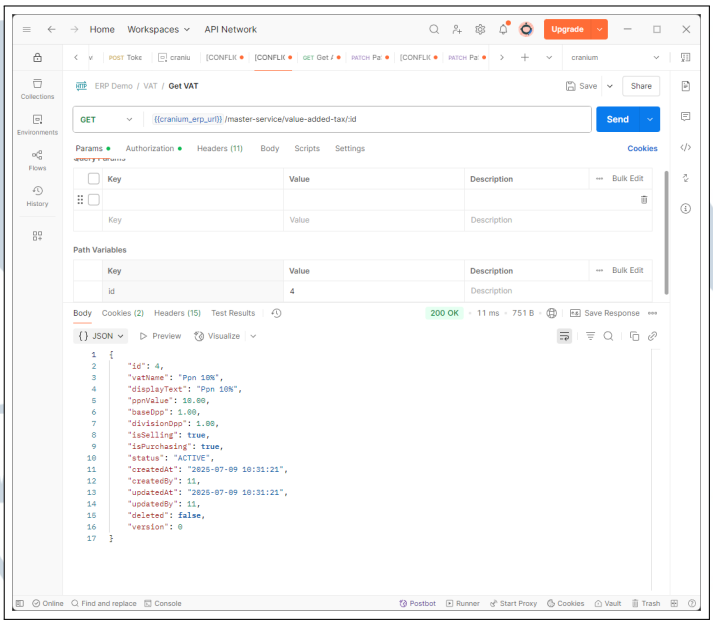
Kode 3.13: Service VAT GetById

```java
@Transactional(value="masterTransactionManager", readOnly = true)
public Page<ValueAddedTaxDto> getAllValueAddedTax(Pageable pageable, ValueAddedTaxRequestDto vatRequestDto) {
    if(pageable.getSort().isEmpty()){
        pageable = PageRequest.of(pageable.getPageNumber(), pageable.getPageSize(), Sort.by("createdAt").
      descending());
    }

    VatSpecification<ValueAddedTax> vatSpecification = new VatSpecification<ValueAddedTax>();
    Specification<ValueAddedTax> spec = vatSpecification.build(vatRequestDto);

    Page<ValueAddedTax> vatPage = vatRepository.findAll(spec, pageable);
    int maxPageNumber = vatPage.getTotalPages();

    if(pageable.getPageNumber() > maxPageNumber && maxPageNumber > 0){
        pageable = PageRequest.of(maxPageNumber-1, pageable.getPageSize(), Sort.by("createdAt").descending())
      ;
        vatPage = vatRepository.findAll(spec, pageable);
    }

    return vatPage.map(vat -> vatMapper.map(vat, ValueAddedTaxDto.class));
}
```

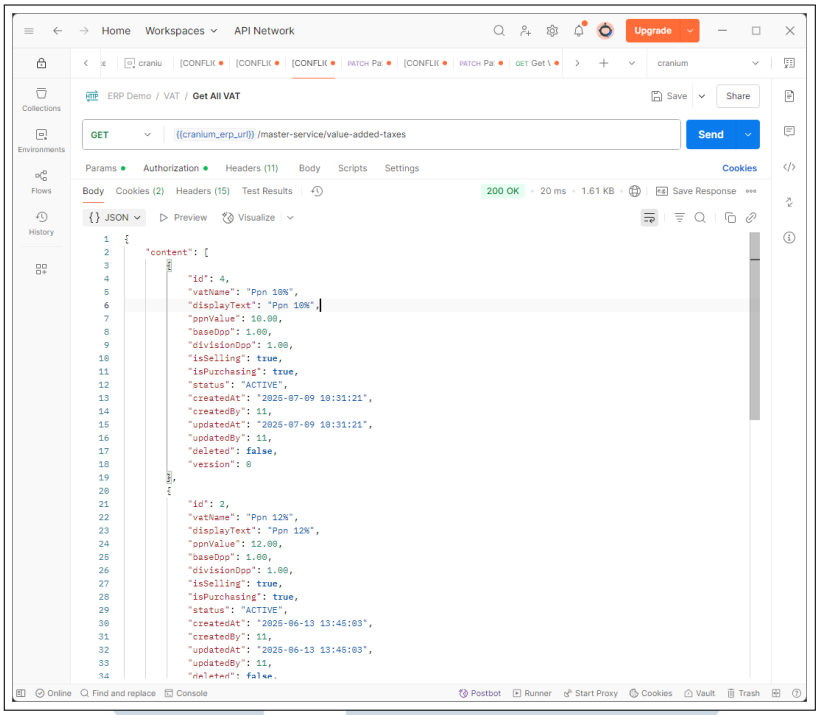
Kode 3.14: Service Get All VAT

```java
@Transactional(value = "masterTransactionManager", readOnly = true)
public List<ValueAddedTaxNameDto> getAllValueAddedTaxName(){
    List<ValueAddedTaxName> valueAddedTaxNames = vatRepository.findValueAddedTaxName();


    return valueAddedTaxNames.stream()
            .map(vatName -> vatNameMapper.map(vatName, ValueAddedTaxNameDto.class))
            .toList();
}
```

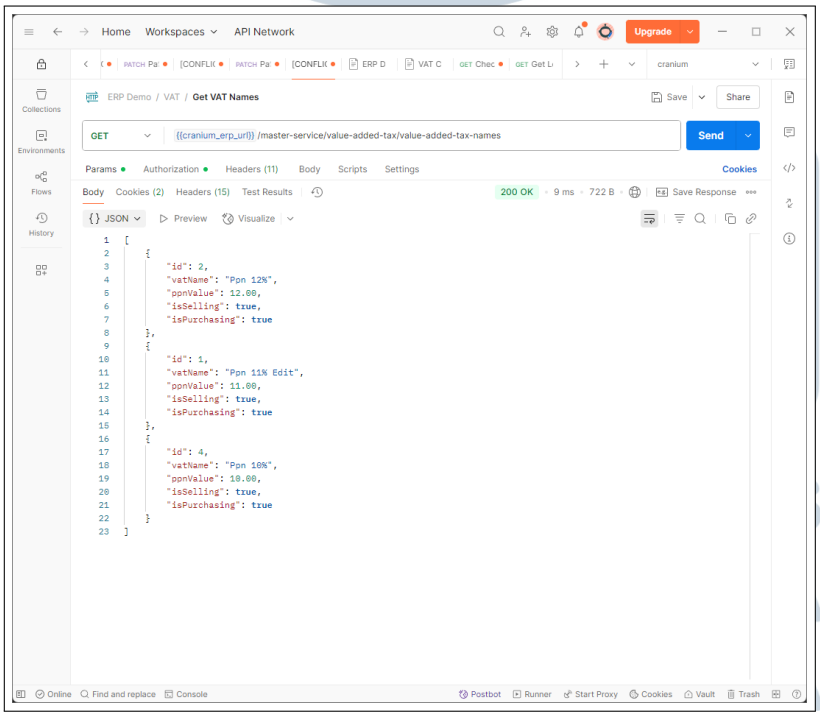
Kode 3.15: Service VAT Names



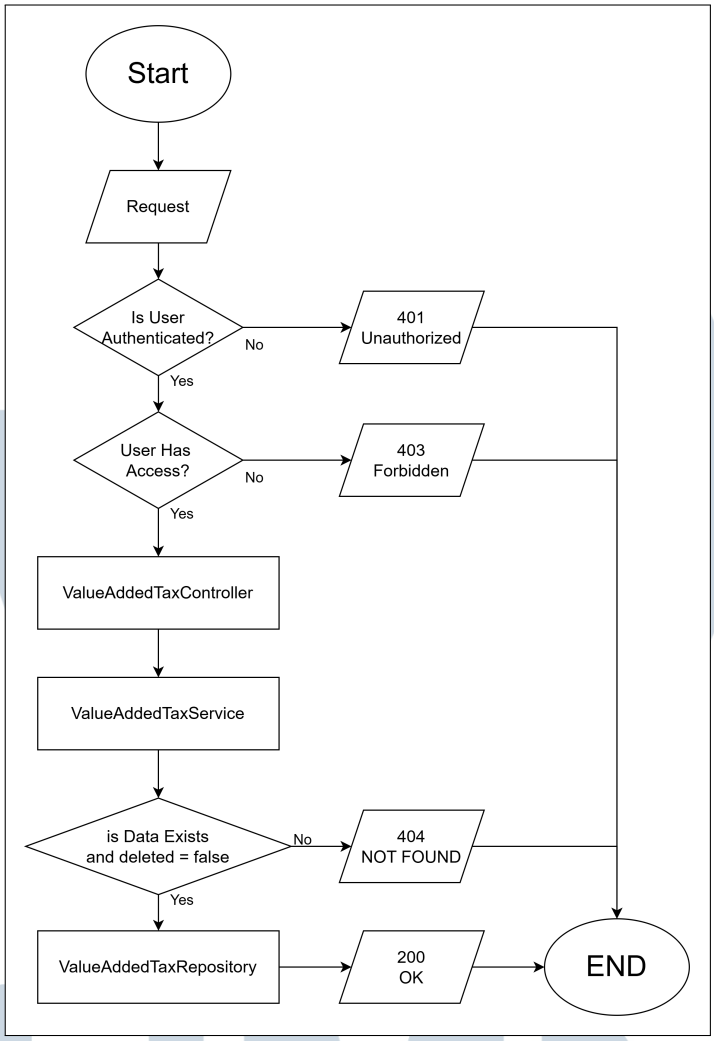
Gambar 3.7. Pengujian *Endpoint* menggunakan Postman untuk VAT GetById



Gambar 3.8. Pengujian *Endpoint* menggunakan Postman untuk VAT GetAll



Gambar 3.9. Pengujian *Endpoint* menggunakan Postman untuk VAT Get VatNames



Gambar 3.10. Flowchart VAT Read

B.1.3 Metode *Update* VAT

Pada proses *Update* VAT, data beserta ID VAT yang akan dimodifikasi dikirimkan oleh *client* menggunakan metode *POST* ke *endpoint* "*master-service/value-added-tax/*{*id*}" dengan format *JSON* pada bagian *request body*, sedangkan ID diambil dari *path parameter*. Setelah data dikirimkan, aplikasi akan terlebih dahulu memeriksa apakah pengguna telah terautentikasi dengan memvalidasi *token*. Jika token tidak ditemukan atau tidak *valid*, maka aplikasi akan mengembalikan *response 401 Unauthorized*. Selanjutnya, dilakukan pengecekan terhadap otorisasi *user*. Jika *user* tidak memiliki hak akses untuk melakukan *Update* data VAT, maka sistem akan mengembalikan *response 403 Forbidden*.

```
@PatchMapping(value = "/value-added-tax/{id}", headers = "X-Api-Version=1")
@IsMasterValueAddedTaxUpdate
@MasterValueAddedTaxNameIsUnique
@ResponseStatus(value = HttpStatus.OK)
@LogExecutionTime
public ValueAddedTaxDto updateValueAddedTax(@RequestBody @Validated ValueAddedTaxUpdateDto vatUpdateDto,
      @PathVariable Long id) throws DataNotFoundException, DataLockException {
    return vatService.updateValueAddedTax(id, vatUpdateDto);
}
```

Kode 3.16: Controller Update VAT

Apabila proses autentikasi dan otorisasi berhasil dilewati, maka aplikasi akan melanjutkan ke tahap pengecekan validitas data dalam *ValueAddedTaxController*. Pengecekan pertama dilakukan terhadap keunikan nama VAT karena field *vat_name* memiliki atribut *unique* di dalam *database*. Sama seperti saat *Create* validasi ini dilakukan dengan menggunakan anotasi *custom @MasterValueAddedTaxNameIsUnique*, yang berfungsi untuk memastikan bahwa tidak terdapat entri nama VAT yang sama yang belum dihapus (*soft delete*). Jika nama VAT dinyatakan belum digunakan sebelumnya, maka aplikasi akan melanjutkan ke tahap pengecekan struktur data apakah sudah sesuai dengan yang didefinisikan dalam *ValueAddedTaxUpdateDto*. Jika struktur data tidak sesuai, maka *response 400 Bad Request* akan dikirimkan.

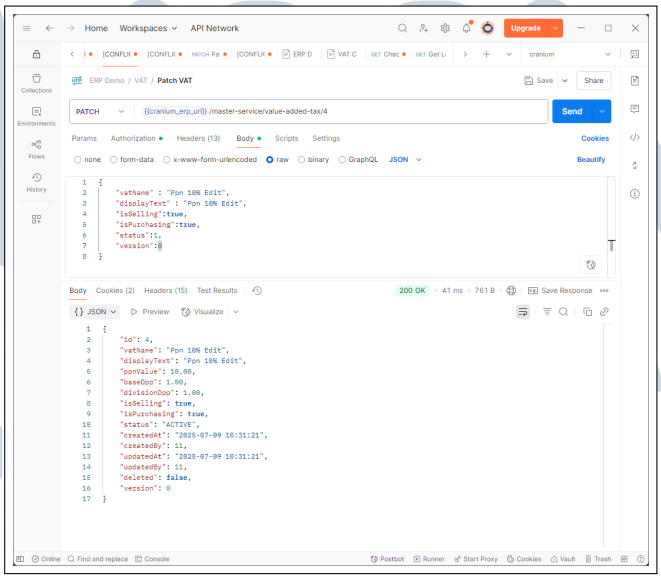
```
@Transactional(value="masterTransactionManager")
public ValueAddedTaxDto updateValueAddedTax(Long id, ValueAddedTaxUpdateDto vatUpdateDto) throws
      DataLockException, DataNotFoundException {
    Optional<ValueAddedTax> vatOptional = Optional.empty();
    Optional<String> myScope = starterUserScopeService.getMyScopeValue("MASTER_VAT_OWN");
    if (myScope.isPresent()){
        if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))){
            vatOptional = vatRepository.findWithLockingPessimisticByIdAndCreatedByAndDeletedFalse(id,
     UserAuthInfo.getUserId());
        } else {
            vatOptional = vatRepository.findWithLockingPessimisticByIdAndDeletedFalse(id);
        }
    }

    if(vatOptional.isEmpty()){
        throw new DataNotFoundException(masterMessageSource.getMessage(MASTER_VAT_FINDID_NOTFOUND, new Object
     []{id}, LocaleContextHolder.getLocale()));
    }
    ValueAddedTax vat = vatOptional.get();

    if(!Objects.equals(vat.getVersion(), vatUpdateDto.getVersion())){
        throw new DataLockException("Database Version: " + vat.getVersion() + ", Current Version: " +
     vatUpdateDto.getVersion());
    }

    vat.setVatName(vatUpdateDto.getVatName());
    vat.setDisplayText(vatUpdateDto.getDisplayText());
    vat.setIsSelling(vatUpdateDto.getIsSelling());
    vat.setIsPurchasing(vatUpdateDto.getIsPurchasing());
    vat.setStatus(vatUpdateDto.getStatus());
    vat = vatRepository.save(vat);

    return vatMapper.map(vat, ValueAddedTaxDto.class);
}
```

Kode 3.17: Service Update VAT

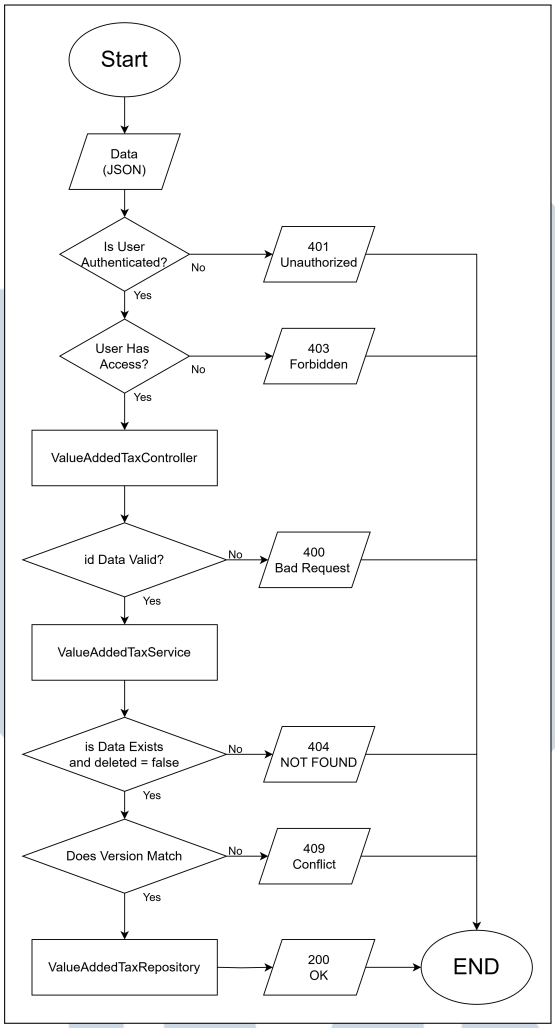
Apabila seluruh validasi telah berhasil dilewati, data akan diteruskan ke *ValueAddedTaxService* untuk diproses sesuai dengan *business logic*. Di dalam *service*, akan dilakukan pengecekan *scope* terlebih dahulu untuk memastikan apakah *user* memiliki akses terhadap data tersebut. Setelah pengecekan *scope*, data akan diambil dari *repository*. Karena ERP Cranium menggunakan pendekatan *soft delete* dalam penghapusan data, maka pengambilan data dari *repository* harus menyertakan kondisi bahwa kolom *deleted* bernilai *false*. Jika data tidak ditemukan atau telah dihapus, aplikasi akan mengembalikan *response 404 Not Found*.

Data yang berhasil ditemukan dalam bentuk *entity ValueAddedTax* akan melalui proses pengecekan *version* dengan membandingkan nilai *version* pada data dalam database dengan *version* yang dikirimkan oleh client melalui *ValueAddedTaxUpdateDto*. Jika *version* tidak sesuai, maka aplikasi akan mengembalikan *response 409 Conflict*. Mekanisme ini digunakan untuk menghindari konflik saat beberapa pengguna mencoba melakukan *Update* data yang sama secara bersamaan, melalui pendekatan *optimistic locking*.

Apabila tidak ditemukan konflik *version*, maka data *entity ValueAddedTax* akan diperbarui berdasarkan informasi dari *ValueAddedTaxUpdateDto*, kemudian disimpan kembali ke dalam *database* melalui *repository*. Jika berhasil, maka sistem akan mengembalikan *response 200 OK* beserta data VAT yang telah diperbarui dalam bentuk *ValueAddedTaxDto* di dalam *response body*.



Gambar 3.11. Pengujian *Endpoint* menggunakan Postman untuk Update VAT



Gambar 3.12. Flowchart Update VAT

B.1.4 Metode *Delete* VAT

Proses *Delete* VAT dilakukan melalui *endpoint master-service/value-added-tax/*{*id*} dengan metode *DELETE*, di mana ID VAT yang akan dihapus dikirimkan sebagai *path variable* oleh *client*. Sama seperti metode-metode lainnya, proses ini diawali dengan validasi autentikasi dan otorisasi. Jika pengguna tidak terautentikasi atau tidak memiliki otorisasi yang sesuai, maka aplikasi akan mengembalikan *respons 401 Unauthorized* atau *403 Forbidden* sesuai kasusnya. Jika validasi autentikasi dan otorisasi berhasil, ID akan diteruskan ke *layer service* untuk diproses lebih lanjut.

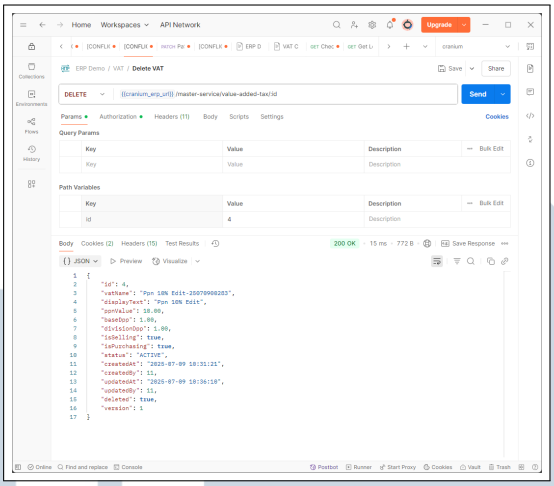
```
@DeleteMapping(value = "/value-added-tax/{id}", headers = "X-Api-Version=1")
@IsMasterValueAddedTaxDelete
@ResponseStatus(value = HttpStatus.OK)
@LogExecutionTime
public ValueAddedTaxDto deleteValueAddedTax(@PathVariable Long id) throws DataNotFoundException,
      DataLockException {
    return vatService.deleteValueAddedTax(id);
}
```

Kode 3.18: Controller Delete VAT

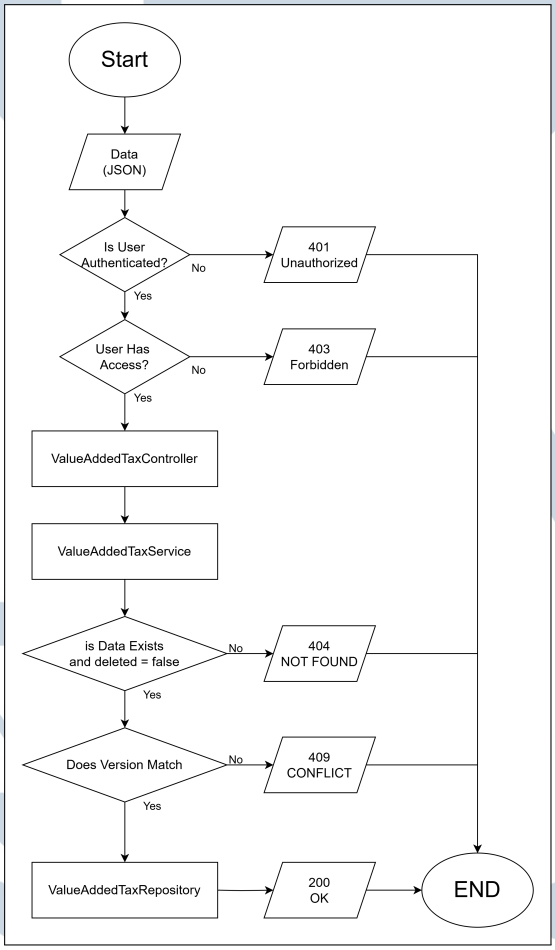
Di dalam *service*, proses penghapusan dilakukan dengan pendekatan *pessimistic locking* menggunakan *try-catch* untuk menghindari konflik data. Pendekatan ini memastikan bahwa tidak ada *cleint* lain yang dapat memodifikasi data yang akan dihapus. Jika terdeteksi adanya *client* lain yang sedang memodifikasi data tersebut, sistem akan mengembalikan *response 409 Conflict*. Sebelum melakukan penghapusan, aplikasi akan melakukan pengecekan *scope* untuk memastikan bahwa *user* memiliki akses terhadap data tersebut. Setelah itu, data akan diambil dari *repository* dengan kondisi kolom *deleted* bernilai *false*. Jika data tidak ditemukan atau telah dihapus sebelumnya, maka sistem akan mengembalikan *response 404 Not Found*. Apabila data ditemukan, maka proses *soft delete* akan dilakukan dengan mengubah nilai kolom *deleted* menjadi *true*. Selain itu, kolom *vat_name* juga akan dimodifikasi dengan menambahkan *suffix* (akhiran) untuk memungkinkan pembuatan data baru dengan nama VAT yang sama.

```
@Transactional(value="masterTransactionManager")
public ValueAddedTaxDto deleteValueAddedTax(Long id) throws DataNotFoundException, DataLockException {
    try{
        Optional<ValueAddedTax> vatOptional = Optional.empty();
        Optional<String> myScope = starterUserScopeService.getMyScopeValue("MASTER_VAT_OWN");
        if (myScope.isPresent()){
            if (myScope.get().equals(Long.toString(UserAuthInfo.getUserId()))){
                vatOptional = vatRepository.findWithLockingPessimisticByIdAndCreatedByAndDeletedFalse(id,
     UserAuthInfo.getUserId());
            } else {
                vatOptional = vatRepository.findWithLockingPessimisticByIdAndDeletedFalse(id);
            }
        }

        if(vatOptional.isEmpty()){
            throw new DataNotFoundException(masterMessageSource.getMessage(MASTER_VAT_FINDID_NOTFOUND, new
     Object[]{id}, LocaleContextHolder.getLocale()));
        }
        ValueAddedTax vat = vatOptional.get();

        vat.setVatName(MasterUtil.appendRandomSuffix(vat.getVatName()));
        vat.setDeleted(true);
        vat = vatRepository.save(vat);

        return vatMapper.map(vat, ValueAddedTaxDto.class);
    }catch (PessimisticLockingFailureException ex){
        throw new DataLockException(ex.getMessage());
    }
}
```

Kode 3.19: Service Delete VAT



Gambar 3.13. Pengujian *Endpoint* menggunakan Postman untuk Delete VAT



Gambar 3.14. Flowchart Delete VAT

B.1.5 *Unit Test dan Contract Test*

Unit test dibuat untuk setiap metode dan *business logic* yang terdapat di dalam *ValueAddedTaxService* guna memastikan bahwa setiap komponen berjalan sesuai dengan fungsinya secara terisolasi. Selain itu, *Contract Test* juga disusun untuk setiap *endpoint* yang tersedia, dengan tujuan untuk memastikan bahwa struktur *request* dan *response* yang dikirim dan diterima sudah sesuai.

3.3.5 Pengembangan Modul *Selling*

A Submodul *Order*

Submodul Order digunakan untuk menyimpan informasi pemesanan penjualan dalam ERP. Struktur *database* untuk submodul ini terdiri dari dua tabel utama, yaitu *orders* dan *orders_detail*. Tabel *orders* berperan sebagai induk yang menyimpan informasi umum terkait pesanan, seperti nomor pesanan, tanggal pemesanan, dan informasi pelanggan. Sementara itu, tabel *orders_detail* berfungsi sebagai tabel *child* atau *detail* yang menyimpan detail dari setiap *item* yang termasuk dalam suatu pesanan. Relasi antara keduanya adalah *many-to-one*, di mana setiap data dalam *orders_detail* mengacu pada satu data di *orders*. Dengan demikian, satu *order* dapat memiliki banyak item (*orders_detail*) di dalamnya.

Submodul *Order* pada dasarnya telah memiliki operasi dasar CRUD, namun setelah dilakukan proses *quality control*, ditemukan bahwa beberapa bagian penting dalam *business logic backend* belum diimplementasikan. Beberapa kekurangan tersebut antara lain adalah belum adanya perhitungan diskon dan perhitungan pajak (PPN dan PPH) untuk setiap *item* dalam pesanan. Selain itu, nilai PPN yang saat ini digunakan masih di-*hardcode* dengan nilai tetap seperti 11 atau 0, sehingga tidak fleksibel untuk perubahan di masa mendatang. Struktur kolom dalam tabel pun belum sepenuhnya sesuai dengan ERD yang telah dirancang.

Permasalahan serupa juga ditemukan pada bagian *frontend*, di mana saat ini hanya tersedia tampilan untuk menghitung harga total, tanpa adanya tampilan dan kalkulasi diskon, PPN, serta PPH baik untuk setiap item maupun total keseluruhan. Kolom-kolom pada antarmuka juga belum sesuai dengan skema ERD yang telah disusun. Oleh karena itu, diperlukan sejumlah penyesuaian pada *database*, *entity*, serta *DTO* baik pada sisi *backend* maupun *frontend*.

A.1 Modifikasi Struktur *Database*

Langkah pertama yang perlu dilakukan sebelum menambahkan *business logic* ke dalam submodul *Order* adalah menyesuaikan struktur *database* agar selaras dengan kebutuhan *business logic* dan rancangan ERD yang telah disusun. Penyesuaian ini mencakup beberapa penambahan dan penghapusan kolom pada tabel *orders* dan *orders_detail*, di antaranya:

Modifikasi tabel *orders*:

Penghapusan:

- *vat*: Akan diganti dengan *vat_id* dalam *child*.
- *vat_percentage*: Akan diganti dengan *vat_id* dalam *child*.
- *tax_amount*: Akan diganti dengan *total_ppn_amount* dan *total_pph_amount*.
- *discount_amount*: Akan diganti dengan *customer_discount_amount* dan *item_discount_amount*.

Penambahan:

- *disocunt_1_type*: Digunakan untuk menyimpan tipe diskon ke-1.
- *disocunt_2_type*: Digunakan untuk menyimpan tipe diskon ke-2.
- *disocunt_3_type*: Digunakan untuk menyimpan tipe diskon ke-3.
- *total_ppn_amount*: Digunakan untuk menyimpan total PPN dari setiap *item*.
- *total_pph_amount*: Digunakan untuk menyimpan total PPH dari setiap *item*.
- *customer_discount_amount*: Digunakan untuk meyimpan total *discount* untuk *customer*.
- *item_discount_amount*: Digunakan untuk menyimpan total *discount* dari setiap *item*.
- *quotation_id*: Sebagai *Foreign key* yang merujuk ke data *quotation* apabila *orders* dibuat berdasarkan *quotation*.

Modifikasi tabel *orders_detail*:

Penambahan:

- *discount_1_amount*: Digunakan untuk menyimpan jumlah dari Diskon ke-1.
- *discount_2_amount*: Digunakan untuk menyimpan jumlah dari Diskon ke-2.
- *discount_3_amount*: Digunakan untuk menyimpan jumlah dari Diskon ke-3.
- *discount_4_amount*: Digunakan untuk menyimpan jumlah dari Diskon ke-4.
- *discount_5_amount*: Digunakan untuk menyimpan jumlah dari Diskon ke-5.
- *vat_id*: Digunakan sebagai *key* untuk mengambil nilai PPN berdasarkan VAT.
- *pph_percentage*: Digunakan untuk menyimpan nilai PPH.

Setelah dilakukan penyesuaian terhadap struktur *database*, penyesuaian juga perlu dilakukan pada *Entity* dan DTO di *backend* mauapun *frontend*. *Field-field* yang tidak lagi relevan perlu dihapus, dan *field* baru ditambahkan agar sesuai dengan struktur *database* yang telah diperbarui. Selain itu, untuk memastikan akurasi dalam perhitungan nilai seperti diskon, harga, dan kuantitas, tipe data pada *Entity* dan DTO untuk *field* yang berkaitan dengan kalkulasi, seperti *discount_1-5, discount_{1-5}_amount, price*, dan *quantity*, perlu diubah menjadi BigDecimal.

A.2 Penambahan *Business Logic*

Penambahan business logic untuk kalkulasi harga, diskon, dan pajak dilakukan di dalam layer service, di mana proses kalkulasi diterapkan pada kedua metode, yaitu create dan update. Langkah pertama dalam proses ini adalah mendefinisikan variabel-variabel yang akan digunakan untuk menampung nilai-nilai yang diperlukan dalam proses kalkulasi, seperti total harga sebelum diskon, nilai pajak (PPN dan PPh), serta harga akhir. Variabel tersebut dapat dilihat pada Kode 3.20

```
1 BigDecimal subTotal = BigDecimal.ZERO;
2 BigDecimal totalItemDiscountAmount = BigDecimal.ZERO;
3 BigDecimal totalAfterItemDiscountAmount = BigDecimal.ZERO;
4 BigDecimal totalCustomerDiscountAmount = BigDecimal.ZERO;
5 BigDecimal totalPPN = BigDecimal.ZERO;
6 BigDecimal totalPPH = BigDecimal.ZERO;
7 BigDecimal grandtotal = BigDecimal.ZERO;
```

Kode 3.20: Initial Variable

```
List<OrdersDetail> listOrderDetail = new ArrayList<>();
List<OrdersDetailCreateDto> ListOrderDetailCreateDto = OptionalListOrderDetailCreateDto.get();
for (OrdersDetailCreateDto detail : ListOrderDetailCreateDto) {
    OrdersDetail ordersDetail = ordersDetailBuild(detail);

    BigDecimal itemSubTotal = getItemSubTotal(ordersDetail.getPrice(), ordersDetail.getQuantity());
    subTotal = subTotal.add(itemSubTotal);

    BigDecimal itemDiscountAmount = getTotalDiscountAmount(itemSubTotal,
            ordersDetail.getDiscount1(), ordersDetail.getDiscount2(), ordersDetail.getDiscount3(),
      ordersDetail.getDiscount4(), ordersDetail.getDiscount5(),
            ordersDetail.getDiscount1Amount(), ordersDetail.getDiscount2Amount(), ordersDetail.
      getDiscount3Amount(),ordersDetail.getDiscount4Amount(), ordersDetail.getDiscount5Amount());
    totalItemDiscountAmount = totalItemDiscountAmount.add(itemDiscountAmount);

    BigDecimal totalAfterDiscount = itemSubTotal.subtract(itemDiscountAmount);

    BigDecimal ppnValue = getPpnValueByVatId(detail.getVatId());

    totalPPN = totalPPN.add(getPpnPphAmount(totalAfterDiscount, BigDecimal.valueOf(ppnValue.doubleValue())));
    totalPPH = totalPPH.add(getPpnPphAmount(totalAfterDiscount, detail.getPphPercentage()));

    ordersDetail.setDiscountAmount(itemDiscountAmount);
    ordersDetail.setWarehouseId(salesmanDto.getWarehouseId());
    ordersDetail.setOrders(orders);
    listOrderDetail.add(ordersDetail);
}
orders.setOrdersDetail(listOrderDetail);
```

Kode 3.21: Keseluruhan For Loop

Potongan Kode 3.21 menunjukkan potongan kode yang menggambarkan keseluruhan proses *for-loop*. Setelah variabel dasar untuk perhitungan dibuat, langkah selanjutnya adalah mengambil seluruh *order detail (item)* dan menyimpannya ke dalam sebuah variabel koleksi. Hal ini diperlukan agar kalkulasi dapat dilakukan untuk setiap *item* menggunakan perulangan *for-loop*. Dalam proses ini, masing-masing *item* akan dihitung nilai subTotal-nya dengan menggunakan metode *getItemSubtotal* yaitu dengan mengalikan nilai *price* dan *quantity*. Implementasi lengkap dari metode getItemSubtotal dapat dilihat pada Kode 3.22

```
private BigDecimal getItemSubTotal(BigDecimal price, BigDecimal quantity){
    return price.multiply(quantity);
}
```

Kode 3.22: GetItemSubtotal

Selanjutnya, jumlah diskon akan dihitung dengan memanggil metode *getTotalDiscountAmount*, yang menerima parameter berupa *itemSubtotal, discount_1* hingga *discount_5*, serta *discount_1_amount* hingga *discount_5_amount*. Implementasi metode *getTotalDiscountAmount* dapat dilihat pada Kode 3.23. Metode ini secara internal akan memanggil metode lain yaitu *getDiscountAmount*, yang digunakan untuk menghitung nilai diskon berdasarkan jenisnya. Jika nilai dari field discount lebih besar dari 0, maka diskon dianggap sebagai persentase. Namun

jika bernilai 0, maka diskon dianggap sebagai potongan harga tetap (nominal). Implementasi metode *getDiscountAmount* dapat dilihat pada Kode 3.24.

```
private BigDecimal getTotalDiscountAmount(BigDecimal itemSubtotal, BigDecimal disc1, BigDecimal disc2,
      BigDecimal disc3, BigDecimal disc4, BigDecimal disc5, BigDecimal disc1Amount, BigDecimal disc2Amount,
      BigDecimal disc3Amount, BigDecimal disc4Amount, BigDecimal disc5Amount){
    BigDecimal discount1amount = getDiscountAmount(itemSubtotal, disc1, disc1Amount);
    itemSubtotal = itemSubtotal.subtract(discount1amount);
    BigDecimal discount2amount = getDiscountAmount(itemSubtotal, disc2, disc2Amount);
    itemSubtotal = itemSubtotal.subtract(discount2amount);
    BigDecimal discount3amount = getDiscountAmount(itemSubtotal, disc3, disc3Amount);
    itemSubtotal = itemSubtotal.subtract(discount3amount);
    BigDecimal discount4amount = getDiscountAmount(itemSubtotal, disc4, disc4Amount);
    itemSubtotal = itemSubtotal.subtract(discount4amount);
    BigDecimal discount5amount = getDiscountAmount(itemSubtotal, disc5, disc5Amount);

    return discount1amount.add(discount2amount.add(discount3amount.add(discount4amount.add(discount5amount)))
      );
}
```

Kode 3.23: getTotalDiscountAmount

```
private BigDecimal getDiscountAmount(BigDecimal itemSubtotal, BigDecimal discount, BigDecimal discountAmount)
      {
    if (discount.compareTo(BigDecimal.ZERO) > 0) {
        return itemSubtotal.multiply(discount).divide(BigDecimal.valueOf(100));
    } else {
        if(itemSubtotal.compareTo(discountAmount) > 0){
            return discountAmount;
        } else {
            return BigDecimal.ZERO;
        }
    }
}
```

Kode 3.24: getDiscountAmount

Hasil dari kalkulasi diskon akan disimpan ke dalam variabel *totalAfterDiscount*, yang kemudian akan digunakan sebagai dasar untuk menghitung pajak. Untuk melakukan kalkulasi pajak, diperlukan nilai PPN yang disimpan dalam kolom *ppn_value*. Nilai ini diambil menggunakan *WebClient* VAT yang telah dibuat sebelumnya, berdasarkan *vat_id* yang dikirimkan. Setelah nilai PPN diperoleh, perhitungan total PPN dan PPH dilakukan dengan memanggil metode *getPpnPphAmount*, yaitu dengan cara mengalikan *totalAfterDiscount* dengan persentase pajak (baik PPN maupun PPH), kemudian dibagi dengan 100. Implementasi dari metode *getPpnPphAmount* dapat dilihat pada Potongan Kode 3.25

```
private BigDecimal getPpnPphAmount(BigDecimal totalAfterDiscount, BigDecimal percentage){
    return totalAfterDiscount.multiply(percentage).divide(BigDecimal.valueOf(100));
}
```

Kode 3.25: Function getPpnPphAmount

```
totalAfterItemDiscountAmount = subTotal.subtract(totalItemDiscountAmount);

totalCustomerDiscountAmount = getTotalDiscountAmount(totalAfterItemDiscountAmount,
        orders.getDiscount1(), orders.getDiscount2(), orders.getDiscount3(),BigDecimal.ZERO, BigDecimal.ZERO,
        orders.getDiscount1Amount(), orders.getDiscount2Amount(), orders.getDiscount3Amount(), BigDecimal.
      ZERO, BigDecimal.ZERO);

grandtotal = getGrandTotal(totalAfterItemDiscountAmount, totalCustomerDiscountAmount, totalPPN, totalPPH);

orders.setSubTotalAmount(subTotal);
orders.setItemDiscountAmount(totalItemDiscountAmount);
orders.setCustomerDiscountAmount(totalCustomerDiscountAmount);
orders.setTotalPpnAmount(totalPPN);
orders.setTotalPphAmount(totalPPH);
orders.setGrandTotalAmount(grandtotal);

orders = ordersRepository.save(orders);
```

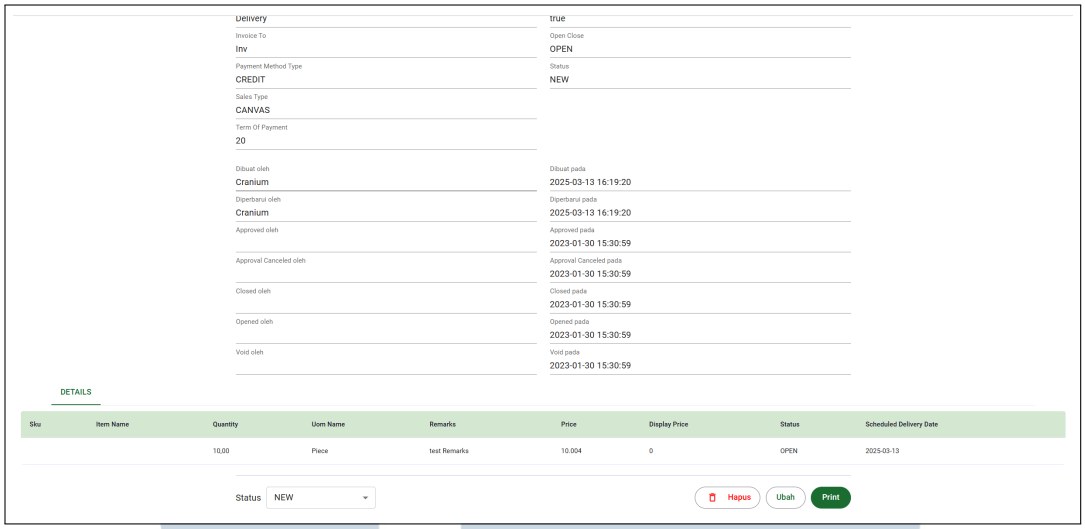
Kode 3.26: Keseluruhan Kode Setelah Loop

Potongan Kode 3.26 menunjukkan proses *for-loop* selesai dijalankan. Setelah perulangan selesai, variabel *totalAfterDiscountAmount* akan diisi dengan hasil pengurangan antara *subTotal* dan *totalItemDiscountAmount*. Nilai ini kemudian digunakan untuk menghitung jumlah diskon tambahan yang diberikan kepada *customer*. Kalkulasi jumlah diskon untuk *customer* dilakukan dengan menggunakan metode yang sama, yaitu *getTotalDiscountAmount*. Namun, karena diskon untuk *customer* hanya terdiri dari tiga jenis, maka *field discount_4, discount_5, discount_4_amount*, dan *discount_5_amount* akan diisi dengan nilai 0. Setelah seluruh diskon dan pajak dihitung, langkah terakhir adalah menghitung total harga akhir (*grand total*) dengan memanggil metode *getGrandTotal*. Grand total dihitung menggunakan rumus: *totalAfterDiscountAmount - totalCustomerDiscount + totalPpn - totalPph*. Implementasi lengkap dari metode getGrandTotal dapat dilihat pada Potongan Kode3.27.

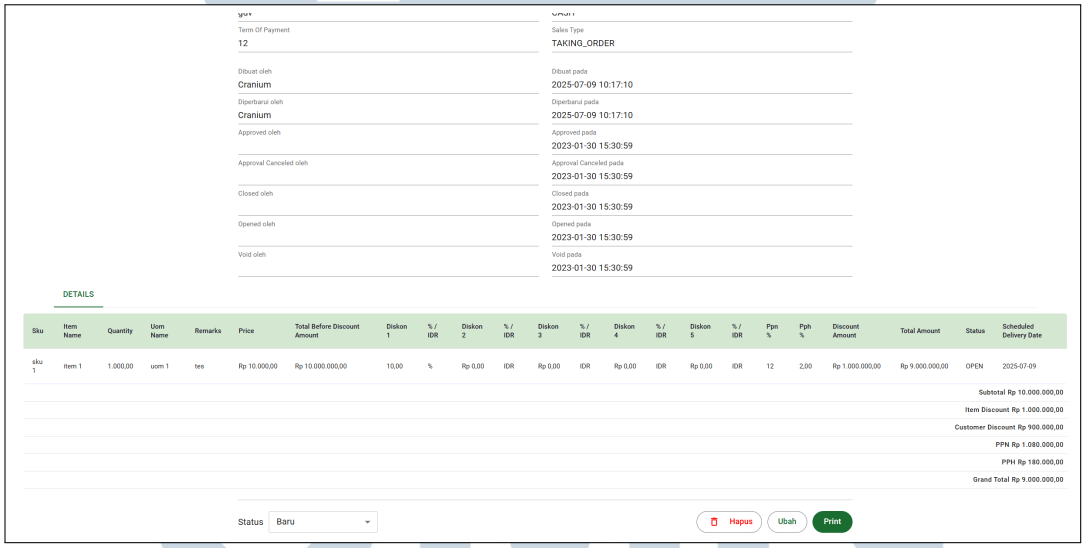
```
private BigDecimal getGrandTotal(BigDecimal totalAfterItemDiscount, BigDecimal customerDiscount, BigDecimal
      ppn, BigDecimal pph){
    return totalAfterItemDiscount.subtract(customerDiscount).add(ppn).subtract(pph);
}
```

Kode 3.27: GetGrandTotal

Untuk penambahan *business logic* pada sisi *frontend*, proses yang dilakukan pada dasarnya mengikuti alur yang serupa dengan di *backend*, yaitu dengan memodifikasi *entity Orders* dan *OrdersDetail* agar sesuai dengan *field-field* yang dibutuhkan berdasarkan struktur yang telah diperbarui. Selain itu, dilakukan juga penambahan logika kalkulasi untuk menghitung harga, diskon, dan pajak yang kemudian ditampilkan pada halaman *Create* dan *Update order*.

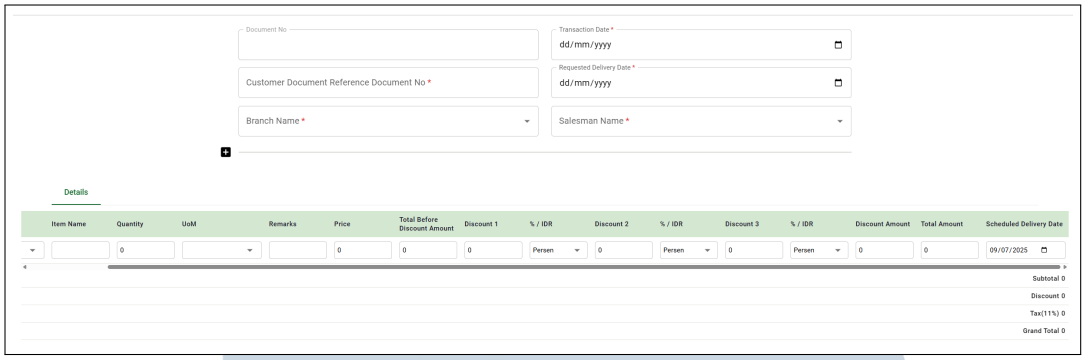


Gambar 3.15. Halaman View Order sebelum perubahan

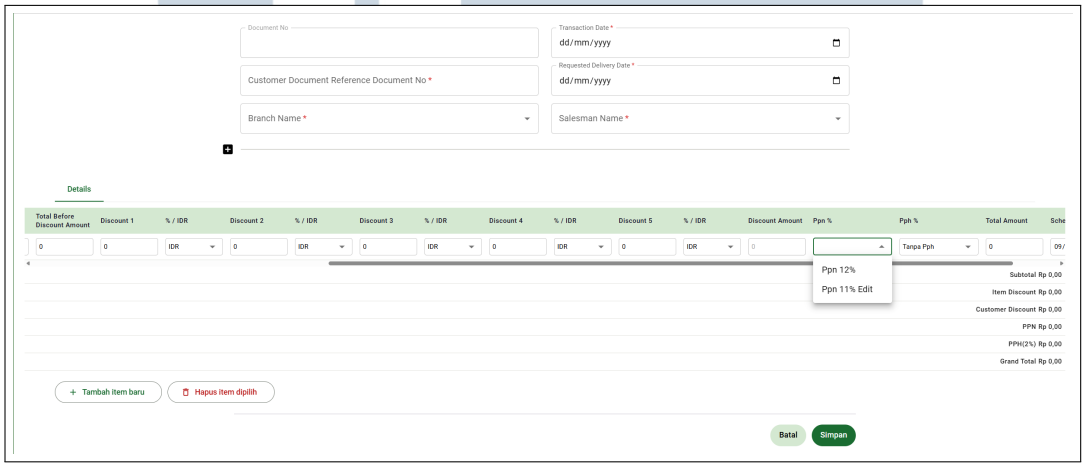


Gambar 3.16. Halaman View Order setelah perubahan

Sebagai bagian dari penyesuaian, ditambahkan pula file API baru untuk *endpoint master-service/value-added-tax-names*, yang digunakan untuk mengambil daftar data VAT (PPN) dan ditampilkan dalam bentuk *dropdown* menu pada form input. Penyesuaian ini juga diikuti dengan pembaruan terhadap *unit test* dan *contract test*, baik di sisi *backend* maupun *frontend*, guna memastikan bahwa semua perubahan telah diimplementasikan dengan benar dan sesuai dengan spesifikasi teknis yang diharapkan.



Gambar 3.17. Tampilan form Create Order sebelum perubahan



Gambar 3.18. Tampilan form Create Order setelah perubahan

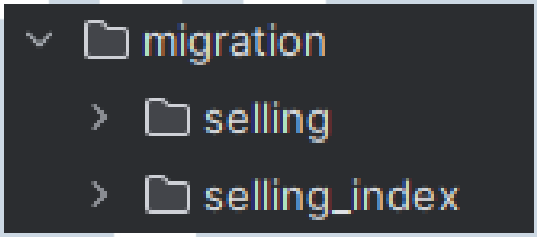
3.3.6 Modifikasi database migration

Sistem yang digunakan dalam ERP ini menerapkan metode *soft delete*, yaitu data tidak benar-benar dihapus dari database melainkan ditandai dengan kolom *deleted* yang bernilai *true*. Namun, pendekatan ini menimbulkan potensi konflik dengan properti *unique index* pada *database*, karena *field* tertentu yang memiliki *constraint* unik tidak mengizinkan data duplikat, meskipun data sebelumnya sudah dianggap "dihapus" secara logika. Untuk mengatasi masalah ini, biasanya dilakukan penambahan *suffix* pada kolom *name* saat proses *delete*, agar memungkinkan penyimpanan data baru dengan nama yang sama.

Solusi lain yang didapatkan adalah dengan menerapkan *partial index* menggunakan kondisi *WHERE deleted IS FALSE*. Akan tetapi, *partial indexing* memiliki keterbatasan, yaitu tidak dapat diterapkan dalam *unit test*. Oleh karena itu, pembuatan *index* dan *unique index* perlu dipisahkan dalam *file migration* terpisah,

agar tidak dijalankan saat proses *unit test* namun tetap dapat diterapkan secara penuh pada lingkungan produksi (*production*) maupun *staging*.

Hal pertama yang perlu dilakukan untuk menerapkan perubahan ini adalah membuat folder baru untuk memisahkan migration index, contohnya dapat dilihat pada gambar 3.19



Gambar 3.19. Memisahkan folder index

Selanjutnya, diperlukan penambahan *flyway location* untuk folder *index* ke dalam semua *properties* kecuali *test*. Penambahan ini bertujuan agar file migrasi yang berisi pembuatan index dapat dijalankan pada lingkungan selain test. Dalam modul *Selling* contoh *flyway* sebelum dimodifikasi dapat dilihat pada potongan Kode 3.28 dan setelahnya dapat dilihat pada potongan Kode 3.29

```
selling.spring.flyway.locations=classpath:db/migration/selling
```

Kode 3.28: Flyway before

```
selling.spring.flyway.locations=classpath:db/migration/selling,classpath:db/migration/selling_index
```

Kode 3.29: Flyway after

Selain itu, perlu dilakukan modifikasi pada file *module-info.java* setiap modul dengan menambahkan *opens* dari folder *index*, agar file-file *migration* di dalamnya dapat dikenali dan dijalankan oleh *Flyway*. Dalam modul *Selling* contoh implementasinya adalah dengan menambahkan kode 3.30 dibawah kode 3.31.

```
opens db.migration.selling_index;
```

Kode 3.30: module-info.java-after

```
opens db.migration.selling;
```

Kode 3.31: module-info.java-before

.

Langkah berikutnya adalah melakukan penyesuaian pada file konfigurasi *DataSourceConfiguration*, dengan memastikan bahwa *Flyway* membaca semua lokasi migrasi yang diperlukan, termasuk folder *index*. Penyesuaian ini bertujuan

agar saat aplikasi dijalankan, *Flyway* akan membaca dan mengeksekusi semua skrip migrasi, baik untuk skema tabel maupun pembuatan *index*. Contoh modifikasi dari file *DataSourceSellingConfiguration* dalam modul *Selling* dapat dilihat dalam Potongan kode 3.32 dan 3.33.

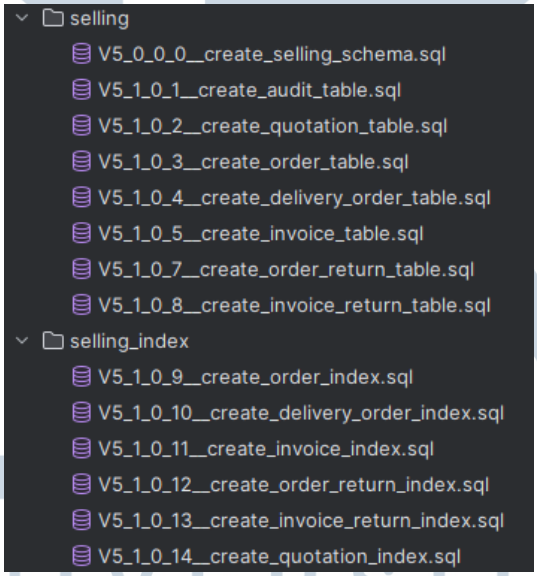
```
1 @Value("{selling.spring.flyway.locations}")
2 String flywayLocations;
```

Kode 3.32: DataSourceSellingConfiguration Before

```
1 @Value("#{'${selling.spring.flyway.locations}'.split(',')}")
2 String[] flywayLocations;
```

Kode 3.33: DataSourceSellingConfiguration After

Setelah semua konfigurasi selesai, tahap selanjutnya adalah membuat file *migration* khusus untuk *index* dan *unique index*. File *migration index* ini diletakkan di dalam folder baru yang diberi akhiran *_index*, namun tetap mengikuti penomoran yang berurutan sesuai dengan file *migration* tabel sebelumnya. Contoh implementasi dalam modul *selling* dapat dilihat pada gambar 3.20.



Gambar 3.20. Contoh file migration index

Langkah terakhir dalam proses modifikasi *migration* adalah memindahkan seluruh *index* dan *unique index* dari file *create table* ke dalam file *index* yang terpisah. Setelah dipindahkan, setiap *index* harus ditambahkan kondisi *WHERE deleted IS FALSE*, untuk mendukung mekanisme soft delete, kecuali untuk tabel auditing (tabel dengan *suffix _aud*). Contoh implementasi dalam modul *Selling* dapat dilihat pada potongan Kode 3.34.

```
CREATE UNIQUE INDEX uniq_deliveryorder_documentno on "selling"."delivery_order"(document_no) WHERE deleted IS FALSE;
CREATE INDEX idx_deliveryorders_ordersid on "selling"."delivery_order"(orders_id) WHERE deleted IS FALSE;

CREATE INDEX idx_deliveryorderaud_updatedat on "selling"."delivery_order_aud"(updated_at);
CREATE INDEX idx_deliveryorderdetailaud_updatedat on "selling"."delivery_order_detail_aud"(updated_at);
CREATE INDEX idx_deliveryorderreturnaud_updatedat on "selling"."delivery_order_return_aud"(updated_at);
CREATE INDEX idx_deliveryorderreturndetailaud_updatedat on "selling"."delivery_order_return_detail_aud"(updated_at);
CREATE INDEX idx_expeditiondeliveryordersdetailaud_updatedat on "selling"."expedition_delivery_orders_detail_aud"(updated_at);
```

Kode 3.34: Contoh Code Index

Seluruh tahapan ini diterapkan secara menyeluruh ke setiap modul dalam sistem, tidak terbatas hanya pada modul Selling saja.

3.4 Kendala dan Solusi yang Ditemukan

A Kendala

Kendala yang ditemukan selama periode magang adalah:

1. Terdapat banyak bugs yang muncul selama pengerjaan task yang berhubungan dengan fitur atau metode yang terdapat dalam submodul lain.
2. Perlu beradaptasi dengan arsitektur modular monolitik.
3. Ketidakselarasan yang terdapat dalam metode dan fitur yang seharusnya memiliki kesamaan.
4. Terdapat conflict yang muncul pada saat melakukan pull request yang menyebabkan keterlambatan dalam proses pengembangan.

Solusi yang dilakukan adalah:

1. Untuk kasus bags sederhana, perbaikan dilakukan secara langsung, namun untuk kasus bugs yang lebih rumit, diselesaikan dengan alternatif injeksi langsung ke database.
2. Diskusi dengan tim dan bertanya langsung ke mentor apabila memang tidak ditemukan solusi.
3. Melakukan penyesuaian terhadap metode dan fitur sesuai arahan supervisor dan mentor.
4. Bekerja sama dengan tim agar tidak terjadi conflict untuk memastikan kelancaran pengerjaan.