BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama kegiatan magang berlangsung, pekerjaan yang dilakukan adalah sebagai *Full Stack Developer* yang merupakan bagian dari *Product Developer*. Terdapat 7 orang intern menjadi 1 kelompok dan seorang supervisor. Pekerjaan magang dilakukan dalam sebuah tim dengan bimbingan dari Bapak Sugito selaku supervisor dan juga team leader dalam mengembangkan proyek ERP.

Bagian ini menjelaskan mengenai posisi mahasiswa magang dalam struktur kerja di perusahaan tempat magang, termasuk alur kerja dan sistem koordinasi yang dijalankan. Untuk memperjelas, alur kerja dapat didukung dengan bagan.

Selama menjalani program kerja praktik di PT Cranium Royal Aditama, penulis ditempatkan di Divisi IT and Software Solutions sebagai *Full Stack Developer* yang merupakan bagian dari *Product Developer*. Penulis berkontribusi dalam proyek pengembangan perangkat lunak *Enterprise Resource Planning* (ERP) milik perusahaan. Kegiatan magang berfokus pada pengembangan serta perbaikan fitur-fitur penting dalam perangkat lunak ERP yang akan digunakan oleh klien perusahaan.

Dalam kegiatan magang, penulis mendapatkan bimbingan dari beberapa mentor dan supervisor yang memiliki tanggung jawab dalam memantau kinerja, memberikan arahan teknis, serta mengevaluasi hasil kerja. Selain itu, penulis turut tergabung dalam tim pengembangan ERP, yang memberikan pengalaman kerja secara tim yang kolaboratif dan terstruktur.

Kedudukan dalam divisi IT and Software Solutions Cranium dapat dijabarkan sebagai berikut:

1. Supervisor: Terdapat dua supervisor utama dalam proyek ERP, yaitu Bapak Sugito sebagai *Tech Lead* dan Bapak Angga sebagai *Project Manager*. Mereka bertugas memberikan pengarahan, melakukan evaluasi, serta memastikan standar pekerjaan sesuai dengan prosedur dan kebutuhan perusahaan.

2. Mentor: Penulis dibimbing oleh dua mentor, yaitu Kak Petra yang fokus pada pengembangan *backend* dan Kak Sabil yang menangani bagian *frontend*.

Mereka memberikan panduan teknis, mulai dari pengaturan lingkungan kerja hingga kolaborasi tim melalui platform seperti GitHub.

3. Mahasiswa Magang: Penulis bertanggung jawab menyelesaikan tugas-tugas yang diberikan, dengan tenggat waktu dan kriteria yang telah ditentukan. Tugas yang dikerjakan meliputi perbaikan dan pengembangan fitur pada modul master ERP, serta penanganan *bug*.

Selama pelaksanaan kegiatan magang, perusahaan telah menetapkan sejumlah aturan dan prosedur kerja yang harus diikuti dalam pengembangan proyek ERP. Tujuan dari penerapan aturan ini adalah untuk memastikan bahwa setiap tugas dapat diselesaikan secara efektif dan efisien oleh seluruh anggota tim yang terlibat. Alur kerja atau *workflow* yang digunakan dalam proyek ERP di PT Cranium Royal Aditama dapat dijelaskan sebagai berikut:

1. Pembagian Tugas
   (a) Tugas baru diberikan kepada *developer* setelah menyelesaikan tugas sebelumnya dan setelah hasil pekerjaannya digabungkan ke *branch* utama.
   (b) Penugasan dilakukan secara langsung oleh supervisor, baik melalui komunikasi personal maupun dalam rapat tim.
   (c) Penjelasan mengenai kebutuhan fitur yang akan dikembangkan disampaikan secara lisan atau menggunakan alat bantu seperti *flowchart*, tabel rancangan, dan desain dari Figma.
   (d) Bentuk tugas dapat berupa pengembangan antarmuka frontend, implementasi fitur di *backend*, perbaikan bug, atau penyempurnaan fitur yang sudah ada.
   (e) Setiap tugas dikembangkan pada *branch* yang berbeda untuk menghindari konflik saat penggabungan hasil kerja.
2. Pelaksanaan Tugas
   (a) *Developer* mulai mengerjakan tugas setelah menerima penjelasan dan instruksi dari supervisor, serta mengikuti tenggat waktu yang telah ditentukan.
   (b) Jika mengalami kendala atau hambatan, *developer* dapat meminta bantuan dari mentor atau supervisor untuk menemukan solusinya.

3. Asistensi dan Revisi

    (a) Setelah menyelesaikan tugas, mahasiswa magang melakukan asistensi kepada mentor atau supervisor guna mendapatkan masukan.

    (b) Komunikasi untuk konsultasi juga dapat dilakukan di luar sesi asistensi resmi.

    (c) Revisi dilakukan berdasarkan masukan yang diterima, dan apabila diperlukan, mahasiswa dapat mengajukan tambahan waktu penyelesaian.

    (d) Setelah revisi selesai, dilakukan asistensi ulang untuk memastikan bahwa perubahan sudah sesuai dengan yang diminta.

4. Pengujian Hasil

    (a) Setiap hasil kerja diuji melalui dua pendekatan: pengujian otomatis (*unit test* dan *contract test*) serta pengujian manual.

    (b) Pengujian otomatis dilakukan menggunakan fitur *testing* dari Springboot dan NextJS, dengan *test case* yang disesuaikan dengan fitur yang dikembangkan.

    (c) Pengujian manual dilakukan oleh mahasiswa serta supervisor atau mentor setelah kode dikirimkan ke *repository*.

5. Penggabungan Kode (*Pull Request*)

    (a) Kode yang sudah lolos pengujian akan digabungkan ke *branch* utama melalui *pull request*.

    (b) Setelah penggabungan, aplikasi diuji kembali untuk memastikan tidak ada kesalahan atau *bug*.

6. Finalisasi

    (a) Tugas dianggap selesai setelah kode digabungkan ke *branch* utama. *Branch* pengembangan kemudian dihapus dan *developer* membuat *branch* baru untuk tugas berikutnya.

Adapun alur komunikasi dan koordinasi dalam kerja magang yang dilaksanakan di Cranium dapat dijabarkan sebagai berikut:

1. Komunikasi

    (a) Komunikasi sehari-hari dilakukan melalui *Whatsapp* sebagai platform utama, serta *Discord* sebagai sarana diskusi tambahan, khususnya ketika bekerja dari rumah.

    (b) Saat bekerja di kantor, komunikasi dilakukan secara langsung melalui percakapan tatap muka, baik dalam rapat maupun diskusi informal.

2. Koordinasi

    (a) Progres tugas dipantau melalui tabel *roadmap* yang tersedia di Figma.

    (b) Setiap anggota tim, termasuk mahasiswa magang, bertanggung jawab memperbarui status tugas masing-masing secara mandiri.

    (c) Selain itu, digunakan Google Sheets kolaboratif untuk mencatat dan memantau hasil kerja harian seluruh anggota tim.

3. Kolaborasi

    (a) Proses kolaborasi pengembangan proyek dilakukan melalui GitHub sebagai *repository* utama.

3.2 Tugas yang Dilakukan

Selama masa praktik kerja di PT Cranium Royal Aditama, penulis dilibatkan dalam pengembangan sistem ERP milik perusahaan. Tugas utama yang diberikan meliputi pengembangan dan penyesuaian berbagai fitur sesuai dengan kebutuhan internal perusahaan maupun permintaan dari klien.

Dalam proyek ERP ini, pengembangan sisi *backend* dilakukan menggunakan Java dengan kerangka kerja Spring Boot, sementara sisi *frontend* dikembangkan menggunakan Next.js dengan TypeScript. Sistem ini menerapkan arsitektur modular monolitik, yaitu pendekatan pengembangan yang menggabungkan kesederhanaan dari struktur monolitik dengan pembagian modul-modul fungsional seperti pada arsitektur *microservices*. Masing-masing modul memiliki peran yang spesifik dan batas tanggung jawab yang jelas, sehingga dapat dikembangkan dan diuji secara independen.

Jenis pekerjaan yang ditugaskan cukup beragam. Beberapa di antaranya adalah memperbaiki kesalahan minor pada kode (*bug fixing*), menambahkan fitur baru baik di sisi *frontend* maupun *backend*, menyusun atau memperbaiki *unit*

test untuk memastikan keandalan sistem, serta menyempurnakan fungsi-fungsi yang sudah ada. Seluruh aktivitas ini dilaksanakan sesuai dengan instruksi dari pembimbing dan kebutuhan yang sedang berjalan di perusahaan.

Untuk mendukung pengerjaan tugas-tugas tersebut, digunakan berbagai perangkat lunak dan kerangka kerja. Rincian teknologi yang digunakan dalam proyek ERP ini adalah sebagai berikut:

1. IntelliJ IDEA digunakan sebagai lingkungan pengembangan untuk membangun sistem *backend* menggunakan Java dan Spring Boot, termasuk untuk penulisan *unit test* dan *contract test*.
2. Spring Boot menjadi kerangka kerja utama dalam pembuatan layanan *backend* ERP.
3. Visual Studio Code dipakai dalam pembangunan antarmuka pengguna ERP berbasis Next.js, sekaligus untuk menulis *unit test* pada sisi *frontend*.
4. Next.js dengan bahasa TypeScript dimanfaatkan untuk membangun antarmuka ERP berbentuk aplikasi web.
5. PostgreSQL digunakan sebagai sistem basis data utama dalam proyek ini.
6. DBeaver berperan sebagai alat bantu untuk mengelola dan meninjau isi dari database PostgreSQL.
7. Postman digunakan untuk melakukan pengujian API yang dikembangkan pada *backend* , serta memastikan integrasi dengan *frontend* berjalan dengan baik.
8. GitHub digunakan sebagai platform penyimpanan kode secara daring serta mendukung kolaborasi dalam tim pengembang melalui fitur seperti *pull request* dan *code review*.

3.3 Uraian Pelaksanaan Magang

3.3.1 Pelaksanaan Kerja Magang

Pelaksanaan kerja magang dapat dilihat pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Onboarding perusahaan, instalasi software, library dan tools, setup dasar Java Springboot, NextJS, PostgreSQL, dan mempelajari aturan bekerja developer di Perusahaan.
2	Melanjutkan pembelajaran Springboot dan mengikuti backend training ERP Cranium oleh mentor. Fokus kepada konsep Controller-Service-Repository, Data Transfer Object, dan Validation serta CRUD pada backend.
3	Melanjutkan pembelajaran backend dan mulai mempelajari struktur backend proyek ERP. Fokus kepada konsep message, dan authorization.
4	Melanjutkan pembelajaran backend proyek ERP. Fokus kepada konsep test suites (contract test, unit test).
5	Review training Backend oleh mentor. Persiapan pembelajaran frontend ERP.
6	Mulai pembelajaran frontend ERP (Training frontend ERP).
7	Melanjutkan pembelajaran frontend ERP, fokus pada pembuatan halaman untuk CRUD dan disambungkan dengan backend melalui endpoint yang sudah dibuat.
8	Melanjutkan pembelajaran frontend ERP, fokus pada melanjutkan pembuatan halaman untuk CRUD dan pembuatan unit test Frontend.
9	Review training Frontend dan Backend oleh mentor.
10	Onboarding dan setup proyek utama ERP Cranium.
11	Mulai pengerjaan tugas pada modul Master submodul Pricetag, Vehicle. Mengerjakan bug wordingan type dan auto generate kode untuk field pricetag code dan field vehicle code.
12	Melanjutkan pengerjaan tugas pada submodul Pricetag dan Vehicle. Fokus pada revisi yang masuk dari Pull Request.
13	Mengerjakan revisi terakhir untuk Pricetag dan Vehicle.
14	Mengerjakan pengerjaan submodul Pricetag, UoM dan Act Standard, fokus pada pembuatan validasi untuk kolom name Pricetag, UoM dan Act Standard .
	Lanjut di halaman berikutnya

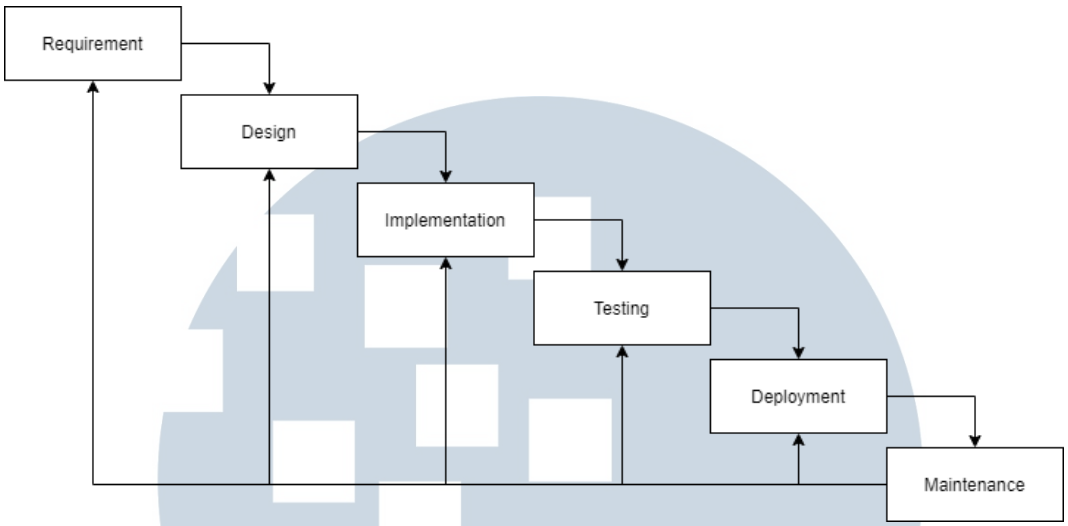
Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu Ke -	Pekerjaan yang dilakukan
15	Melanjutkan pengerjaan submodul UoM, Act Standard dan submodul baru yaitu Company. Serta mengerjakan unit test untuk submodul-submodul yang baru dikerjakan yaitu Pricetag, UoM, Act Standard, Company, Vehicle
16	Mulai mengerjakan validasi untuk submodul Area, Province, Driver, serta mengerjakan unit test untuk Area, Province, Driver
17	Mengerjakan validasi untuk submodul Itemgroup fokusnya field name itemgroup, serta unit testnya. Dan mengerjakan bug fixing untuk submodul Supplier dan Itemsupplier (Fokus untuk Data BE dan FE yang tidak sinkron dan tidak bisa save)
18	Mengerjakan bug fixing pada submodul Sales Order Return, fokus pada translate.
19	Mengerjakan task meghapus submodul Expedition dan Finished Expedition dari menu Warehouse dalam modul Warehouse, karena sudah ada di menu Shipping dalam modul Shipping.
20	Mengerjakan bug fixing pada submodul Journal dalam modul Finance. fokus pada tampilan yang dimana harus text namun muncul integer dan tidak balik ke data inputan awal saat melakukan update.

3.3.2 Konsep *Software Development Life Cycle* ERP Cranium

Siklus Hidup Pengembangan Perangkat Lunak atau *Software Development Life Cycle* (SDLC) merupakan suatu kerangka kerja sistematis yang memuat tahapan-tahapan penting dalam proses pengembangan, pengelolaan, hingga perawatan perangkat lunak.

Tahapan-tahapan SDLC yang digunakan dalam proses pengembangan sistem ERP Cranium digambarkan pada ilustrasi berikut:



Gambar 3.1. Diagram SDLC ERP Cranium

Dalam pengembangannya, sistem ERP Cranium menggunakan pendekatan model sekuensial yang dikenal dengan Model Air Terjun atau *Waterfall Model*. Pendekatan ini mengacu pada proses pembangunan perangkat lunak secara bertahap dan berurutan, di mana satu fase harus selesai sepenuhnya sebelum melanjutkan ke tahap berikutnya [5].

Berikut adalah penjabaran masing-masing tahapan dalam pendekatan *waterfall* yang diterapkan pada proyek ERP Cranium:

- *Requirement*

  Tahapan awal ini bertujuan untuk mengumpulkan dan merumuskan kebutuhan sistem secara menyeluruh. Informasi yang dihimpun meliputi spesifikasi perangkat lunak, kebutuhan bisnis, serta fungsi utama yang harus tersedia pada sistem ERP.

- *Design*

  Setelah seluruh kebutuhan ditentukan, tahap selanjutnya adalah menyusun rancangan sistem secara teknis. Ini meliputi perencanaan struktur sistem, alur interaksi antar modul, hingga desain skema basis data dan antarmuka pengguna.

- *Implementation*

  Fase ini merupakan proses penerjemahan rancangan ke dalam bentuk kode program. Pengembangan dilakukan berdasarkan modul fungsional, dengan

prioritas pada operasi dasar seperti pembuatan, pembacaan, pembaruan, dan penghapusan data (CRUD). Integrasi antar modul dilakukan secara bertahap sesuai rencana pengembangan.

- *Testing*

  Setelah implementasi selesai, sistem diuji untuk memastikan bahwa seluruh fitur berjalan sesuai perencanaan. Tahap ini juga digunakan untuk mengidentifikasi dan memperbaiki kesalahan yang mungkin muncul.

- *Deployment*

  Jika proses pengujian berhasil, perangkat lunak dipindahkan ke lingkungan produksi. Tahapan ini mencakup pengaturan sistem pada server dan penyesuaian agar dapat digunakan oleh pengguna akhir secara optimal.

- *Maintenance*

  Setelah perangkat lunak mulai digunakan, diperlukan pemeliharaan secara berkala. Aktivitas dalam tahap ini mencakup perbaikan *bug*, peningkatan performa, dan penambahan fitur baru berdasarkan kebutuhan di masa mendatang.

3.3.3 Ruang Lingkup Pengembangan

Proses pengembangan dalam sistem ERP Cranium mencakup beberapa aspek utama sebagai berikut:

1. Penanganan *Bug*

   Perbaikan *bug* difokuskan pada modul *Master, Selling, Warehouse*, dan *Finance*. *Bug* yang diperbaiki berasal dari laporan supervisor maupun yang ditemukan secara langsung saat pengerjaan modul. Penanganannya mencakup sisi *frontend* dan *backend*, tergantung kompleksitas masalahnya. Beberapa jenis perbaikannya meliputi:

   - Pembenahan tampilan data di halaman web.

     Beberapa antarmuka sistem ERP perlu disesuaikan, seperti label kolom yang belum tepat, kesalahan terjemahan, data yang ditampilkan tidak sesuai format, jumlah kolom tidak konsisten, hingga data yang tidak muncul sama sekali.

- Perbaikan fungsionalitas fitur.

  Terdapat fitur-fitur web yang belum berjalan semestinya, seperti dropdown yang tidak menampilkan pilihan, kolom input yang belum tersedia, formulir yang gagal dikirim, serta fungsi filter yang belum aktif.

- Penyesuaian struktur data antara *frontend* dan *backend*.

  Ditemukan ketidaksesuaian antara data yang dikirim oleh sisi *frontend* dan data yang diharapkan oleh *backend*, sehingga perlu dilakukan penyesuaian.

2. Penambahan Fitur ke Submodul Eksisting

   Beberapa fitur tambahan disisipkan ke dalam submodul yang sudah ada. Salah satu pembaruan yang signifikan adalah penambahan validasi untuk field nama pada beberapa submodul *Master* supaya tidak ditemukan data yang duplikat.

3. Penghapusan Submodul Ganda

   Pada beberapa modul, terdapat submodul-submodul internal yang sesuai dengan fungsi utama dari modul tersebut. Namun, ditemukan juga adanya submodul yang serupa atau bahkan identik tersedia di lebih dari satu modul yang berbeda. Meskipun setiap modul memiliki karakteristik dan tujuan penggunaan submodul yang berbeda, keberadaan submodul yang sama di dua tempat dapat menyebabkan duplikasi fungsionalitas, kebingungan pengguna, dan inkonsistensi data. Oleh karena itu, dilakukan penghapusan terhadap submodul yang bersifat duplikat untuk menjaga kejelasan struktur navigasi dan mencegah redundansi fitur antar modul.

Dalam proses pengembangan ERP Cranium, digunakan beberapa konsep utama, yaitu:

1. *Data Transfer Object*, *Entity*, dan *Mapper*

   Data Transfer Object (DTO) merupakan objek sederhana yang berfungsi untuk membawa data antar bagian sistem, seperti antara lapisan presentasi dan logika bisnis. DTO tidak memiliki logika pemrosesan, melainkan hanya menyimpan properti serta menyediakan fungsi akses (*getter/setter*). Tujuan penggunaan DTO adalah untuk mengelompokkan informasi sehingga dapat

dikirim sekaligus dalam satu permintaan, yang membuat komunikasi antar proses menjadi lebih efisien.

Entity atau entitas adalah representasi dari struktur tabel database dalam bentuk objek *Java*. *Entity* menyimpan data yang bersifat persisten dan juga mendefinisikan relasi antar tabel. Dalam ERP Cranium, pengelolaan entitas dilakukan melalui *Jakarta Persistence*.

Mapper adalah komponen yang digunakan untuk mengubah data dari satu bentuk objek ke bentuk lain, biasanya dari DTO ke *entity* atau sebaliknya. *Mapper* sangat penting untuk menjaga pemisahan tanggung jawab antar lapisan sistem.

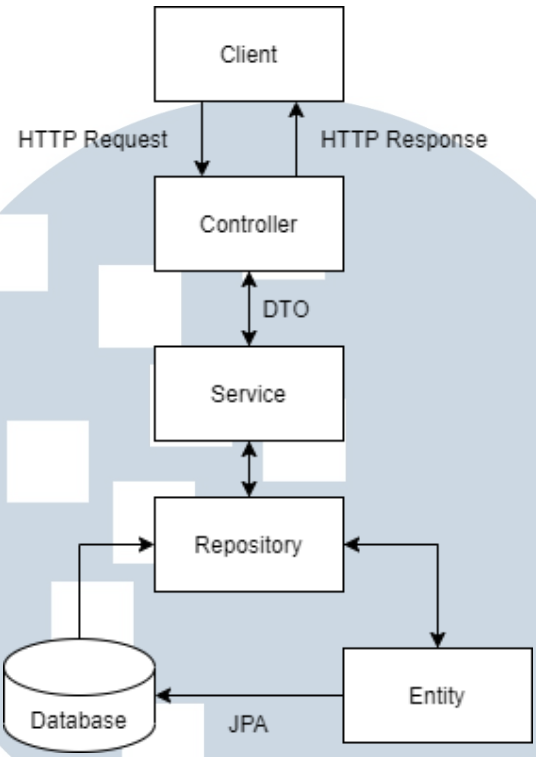
2. *Controller*, *Service*, dan *Repository*

   Ketiga komponen ini merupakan bagian dari arsitektur berlapis (*layered architecture*) yang umum digunakan dalam pengembangan aplikasi berbasis *Spring Boot*. Pendekatan ini memisahkan logika aplikasi ke dalam tanggung jawab masing-masing lapisan agar kode lebih terstruktur dan mudah diuji.

   Controller berada di lapisan terluar yang bertugas menangani permintaan dari pengguna melalui protokol HTTP. Fungsinya adalah menerima data dari *client* dan mengembalikannya dalam bentuk respon tanpa memproses logika bisnis. *Controller* biasanya juga melakukan validasi awal sebelum data diteruskan ke *service*.

   Service adalah bagian yang memuat logika bisnis. Di sinilah data diproses, dimanipulasi, dan dikonversi antara bentuk DTO dan *entity* sebelum disimpan atau diambil dari database. *Service* sering memanfaatkan *mapper* dan menjadi pusat pengolahan dalam sistem.

   Repository berperan sebagai penghubung antara aplikasi dan database. Komponen ini menangani penyimpanan, pembacaan, pengubahan, dan penghapusan data. *Repository* menggunakan fitur dari *Jakarta Persistence* untuk operasi dasar, namun dapat diperluas dengan *query* tambahan sesuai kebutuhan.



Gambar 3.2. Hubungan Controller-Service-Repository

Gambar di atas menunjukkan alur data dalam sistem. Proses dimulai dari permintaan pengguna melalui *API*, dilanjutkan ke *controller* untuk diteruskan ke *service* dalam bentuk DTO. Setelah diproses, data dikonversi menjadi *entity* dan diteruskan ke *repository* untuk berinteraksi dengan database. Hasilnya dikembalikan melalui jalur yang sama hingga mencapai *client*.

3. *Unit Test* dan *Contract Test*

   Unit test merupakan pengujian terhadap bagian terkecil dalam aplikasi, seperti metode atau kelas, untuk memastikan bahwa setiap bagian tersebut bekerja sesuai dengan harapan secara terisolasi. Tujuannya adalah untuk mendeteksi kesalahan sejak dini dan menjaga kualitas kode.

   Contract test bertujuan untuk memastikan bahwa komponen-komponen dalam sistem dapat berinteraksi satu sama lain sesuai dengan format dan struktur yang telah ditentukan. Ini penting untuk menjaga konsistensi antarmuka dan komunikasi antar modul.

   Pada sisi *backend*, *unit test* digunakan untuk memverifikasi bahwa setiap metode di *service* berjalan sesuai logika bisnis yang diharapkan. Sedangkan

contract test digunakan untuk memastikan bahwa controller merespon permintaan dari *client* sesuai dengan kontrak atau kesepakatan format.

Di sisi *frontend*, *unit test* dilakukan untuk memastikan bahwa komponen antarmuka bekerja dengan benar, mulai dari proses *rendering* hingga interaksi dengan data. Alat uji yang digunakan pada *frontend* dan *backend* disesuaikan dengan kebutuhan masing-masing lapisan.

3.3.4 Pengembangan Modul Master

A Penambahan fitur validasi di berbagai submodul

A.1 Analisis Masalah

Beberapa submodul seperti *Pricetag, Unit of Measurement (UOM), Standard Activity, Company, Area, Province, Driver*, dan *Item Group* sebelumnya belum dilengkapi dengan validasi data yang memadai. Hal ini memungkinkan pengguna untuk membuat entri baru tanpa pemeriksaan terhadap duplikasi data. Akibatnya, dapat terjadi entri data ganda yang membingungkan pengguna dalam pencarian maupun pengelolaan data. Duplikasi ini tidak hanya menurunkan kualitas data, tetapi juga dapat menimbulkan inkonsistensi dalam proses bisnis. Oleh karena itu, diperlukan implementasi validasi pada submodul-submodul tersebut guna mencegah entri berlebih dan menjaga integritas data.

A.2 Solusi

Untuk menyelesaikan masalah ini, diterapkan fitur validasi data unik pada sisi *backend*, yang bersifat *reuseable* dan dapat digunakan di berbagai submodul dengan pendekatan yang konsisten. Mekanisme validasi ini diimplementasikan melalui anotasi kustom seperti berikut:

```
@Documented
@Constraint(validatedBy = {})
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface MasterPriceTagNameIsUnique {
    String message() default "{master.pricetag.name.unique}";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
```

```
}
```

Kode 3.1: Anotasi Validasi Unik Pricetag

Logika validasi tersebut kemudian dibedakan antara saat proses *create* dan *update*. Keduanya menggunakan parameter `Object[]` yang kemudian diubah ke bentuk DTO dan ID sesuai kebutuhan.

```
private boolean createValidator(Object[] value) {
    Optional<PriceTagCreateDto> priceTagCreateDtoOptional =
    Optional.ofNullable((PriceTagCreateDto) value[0]);

    if(priceTagCreateDtoOptional.isEmpty()) {
        return false;
    }

    try {
        PriceTagCreateDto priceTagCreateDto =
    priceTagCreateDtoOptional.get();
        priceTagService.findByName(priceTagCreateDto.getName());
        return false; // Duplikat ditemukan
    } catch (DataNotFoundException e) {
        return true; // Nama unik
    }
}
```

Kode 3.2: Logika Validasi Nama saat Create

```
private boolean updateValidator(Object[] value) {
    Optional<PriceTagUpdateDto> priceTagUpdateDtoOptional =
    Optional.ofNullable((PriceTagUpdateDto) value[0]);
    Optional<Long> priceTagIdOptional = value.length > 1 && value
    [1] instanceof Long
            ? Optional.ofNullable((Long) value[1])
            : Optional.empty();
    if (priceTagUpdateDtoOptional.isEmpty() || priceTagIdOptional.
    isEmpty()) {
        return false;
    }

    PriceTagUpdateDto priceTagUpdateDto =
    priceTagUpdateDtoOptional.get();
    Long priceTagId = priceTagIdOptional.get();

    try {
```

```
        PriceTagDto priceTagDto = priceTagService.findByName(
    priceTagUpdateDto.getName());
        return priceTagDto.getId().equals(priceTagId); // Valid
    jika ID cocok
    } catch (DataNotFoundException e) {
        return true; // Nama belum digunakan, valid
    }
}
```

Kode 3.3: Logika Validasi Nama saat Update

Untuk menjaga konsistensi penulisan dan mencegah kesalahan karena format input yang tidak seragam, digunakan pula proses *normalisasi string* yang terdapat pada *MasterUtil*:

```
@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class MasterUtil {

    public static String normalizeString(String name) {
        if (name == null) return null;
        return name.trim().replaceAll("\\s+", " ");
    }
}
```

Kode 3.4: Fungsi Normalisasi Nama

Fungsi `normalizeString` digunakan untuk menghilangkan kelebihan spasi pada nama input dari pengguna. Hal ini penting agar proses pencarian nama dan validasi tidak terpengaruh oleh variasi penulisan yang sebenarnya bermakna sama (misalnya ”Harga 1” dan ” Harga 1 ” dianggap sama).

Setelah string dinormalisasi, pencarian data dilakukan menggunakan metode `findByName`, seperti pada kode berikut:

```
@Transactional(value = "masterTransactionManager", readOnly = true
    )
public PriceTagDto findByName(String name) {
    String normalized = MasterUtil.normalizeString(name);
    Optional<PriceTag> priceTagOptional = priceTagRepository.
    findByNameIgnoreCaseAndDeletedFalse(normalized);
    if (priceTagOptional.isEmpty()) {
        throw new DataNotFoundException(masterMessageSource.
    getMessage(
            MASTER_PRICETAG_FINDNAME_NOTFOUND,
            new Object[]{normalized},
            LocaleContextHolder.getLocale()
```

```
        ));
    }
    return priceTagMapper.map(priceTagOptional.get(), PriceTagDto.
    class);
}
```

Kode 3.5: Pencarian Pricetag Berdasarkan Nama

Metode `findByName` berfungsi untuk mengambil data `Pricetag` dari database berdasarkan nama yang telah dinormalisasi. Pencarian dilakukan tanpa membedakan huruf kapital dan hanya terhadap data yang belum dihapus (`deleted = false`). Apabila data tidak ditemukan, maka akan dilemparkan `DataNotFoundException`. Fungsi ini banyak digunakan dalam proses validasi, baik saat create maupun update, untuk memastikan bahwa nama yang digunakan belum pernah dipakai sebelumnya.

Karena validasi dilakukan dalam konteks validator yang bukan merupakan *Spring-managed bean*, maka untuk dapat mengakses method `findByName()` dari `PriceTagService`, digunakan class utilitas khusus bernama `PriceTagServiceUtil`. Class ini memiliki anotasi `@Component` dan menyimpan instansiasi `PriceTagService` dalam bentuk variabel statik. Dengan pendekatan ini, validator tetap dapat mengakses method service meskipun berada di luar konteks *dependency injection*.

Berikut adalah implementasi dari `PriceTagServiceUtil`:

```
@Component
@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class PriceTagServiceUtil {

    private static PriceTagService priceTagService;

    @Autowired
    public PriceTagServiceUtil(PriceTagService priceTagService) {
        PriceTagServiceUtil.setPriceTagService(priceTagService);
    }

    private static void setPriceTagService(PriceTagService
    priceTagService) {
        PriceTagServiceUtil.priceTagService = priceTagService;
    }

    public static PriceTagService getPriceTagService() {
        return priceTagService;
```

```
18      }
19  }
```

Kode 3.6: Kelas Utilitas untuk Akses Statis ke PriceTagService

Penggunaan utilitas ini membuat proses validasi lebih fleksibel dan tetap terintegrasi dengan service layer yang telah tersedia, tanpa menyalin ulang logika dari `PriceTagService` ke dalam validator.

Apabila pengguna menghapus data dan kemudian ingin menggunakan nama tersebut kembali, sistem akan menambahkan akhiran acak melalui fungsi `appendRandomSuffix` untuk mencegah konflik nama tetapi tetap mempertahankan jejak historis.

```
1  public static String appendRandomSuffix(String baseName) {
2      String formattedDate = new SimpleDateFormat("yyMMdd").format(
       new Date());
3      String paddedRandomValue = String.format("%05d",
       SecureRandomUtil.generateIntRandom(99999));
4      return String.format("%s-%s%s", baseName, formattedDate,
       paddedRandomValue);
5  }
```

Kode 3.7: Menambahkan Akhiran Acak

Fungsi `appendRandomSuffix` digunakan untuk menghasilkan nama unik baru berdasarkan nama lama yang telah digunakan sebelumnya. Format akhir dari nama akan menyerupai `NamaLama-25060112345`, sehingga tetap mencerminkan nama aslinya namun sudah berbeda dan dapat disimpan kembali ke dalam sistem tanpa konflik duplikasi.

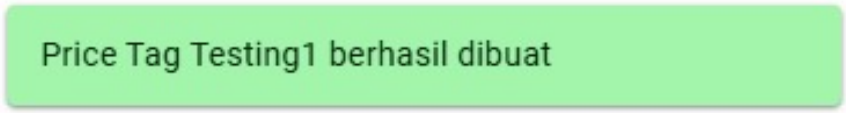
Validasi ini tidak hanya diterapkan pada *Pricetag*, tetapi juga telah digunakan di submodul lain seperti *UOM*, *Standard Activity*, *Company*, *Area*, *Province*, *Driver*, dan *Item Group*. Pendekatan ini memastikan bahwa semua entri dalam sistem ERP Cranium memiliki nama unik dan tidak terjadi duplikasi antar data, sekaligus memberikan pengalaman pengguna yang lebih terkontrol dan akurat dalam proses input data.

A.3 Hasil Kerja

Setelah fitur validasi diterapkan pada submodul seperti *Pricetag*, sistem kini mampu mencegah entri duplikat berdasarkan nama. Sebelumnya, pengguna masih dapat menyimpan data meskipun nama *pricetag* yang dimasukkan sudah ada di

dalam basis data. Hal ini mengakibatkan terjadinya data ganda dan berpotensi membingungkan dalam proses pencarian atau pengelolaan data.

Pada gambar berikut ditunjukkan bahwa pengguna dapat menyimpan data meskipun nama *Pricetag* yang diinputkan telah digunakan sebelumnya. Sistem tidak menampilkan pesan kesalahan apa pun dan tetap menerima entri tersebut.



Gambar 3.3. Tampilan sistem sebelum validasi: duplikasi nama Pricetag masih diperbolehkan

Setelah dilakukan penambahan fitur validasi, sistem secara otomatis akan memeriksa apakah nama yang dimasukkan sudah tersedia dalam basis data. Jika ditemukan duplikat, sistem akan menolak input dan menampilkan pesan peringatan kepada pengguna.

Gambar 3.4. Tampilan sistem setelah validasi: nama Pricetag yang sama ditolak

Dengan diterapkannya validasi nama unik pada submodul *Pricetag* (dan modul lain yang relevan), sistem kini lebih andal dalam menjaga integritas data. Pengguna tidak lagi dapat menyimpan data yang bersifat duplikat, sehingga kualitas data dan proses bisnis menjadi lebih konsisten dan terpercaya.

B Submodul Pricetag

B.1 Analisis Masalah

Setelah dilakukan proses *quality assurance* terhadap submodul *Pricetag*, ditemukan kendala, Beberapa teks pada halaman masih belum diterjemahkan ke dalam bahasa yang sesuai, atau menggunakan terjemahan yang kurang tepat. Hal ini menyebabkan kebingungan bagi pengguna karena informasi yang ditampilkan tidak konsisten atau sulit dipahami.

B.2 Solusi

Untuk mengatasi masalah terkait penerjemahan kata (*translation*) yang tidak sesuai atau belum tersedia pada halaman web submodul *Pricetag*, dilakukan perbaikan dengan menambahkan entri-entri yang diperlukan pada file *localization* (biasanya berupa berkas `.json` atau `.ts`) yang digunakan untuk mendukung fitur multibahasa pada sistem.

Penambahan ini dilakukan dengan cara menuliskan pasangan kunci-nilai (*key-value*) yang mewakili setiap teks yang muncul di antarmuka pengguna. Dengan adanya entri ini, sistem dapat secara otomatis menampilkan teks yang sesuai berdasarkan bahasa yang dipilih oleh pengguna.

Berikut adalah contoh implementasi kode yang ditambahkan untuk menyelesaikan permasalahan tersebut:

```
"master.priceTag.code": "Code Price Tag",
"master.priceTag.createdAt": "Created At",
"master.priceTag.updatedAt": "Updated At",
"master.priceTag.createdBy": "Created By",
"master.priceTag.updatedBy": "Updated By",
"master.priceTag.inactive": "Inactive",
"master.priceTag.active": "Active",
```

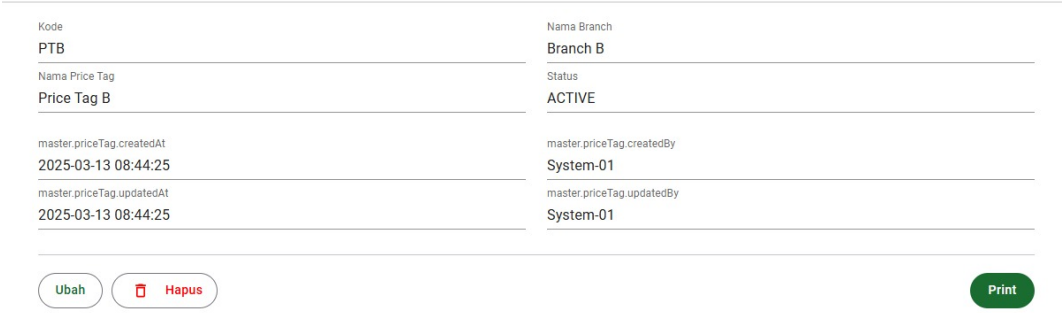
Setiap entri memiliki struktur `"key": "value"` di mana:

- `key` mengacu pada identitas unik dari string, yang dipanggil oleh komponen frontend.
- `value` adalah teks yang akan ditampilkan ke pengguna sesuai bahasa yang digunakan.

Dengan penambahan entri ini, seluruh teks terkait submodul *Pricetag* kini dapat ditampilkan dengan benar dan konsisten, sehingga meningkatkan pengalaman pengguna serta mempermudah pemahaman antarmuka.

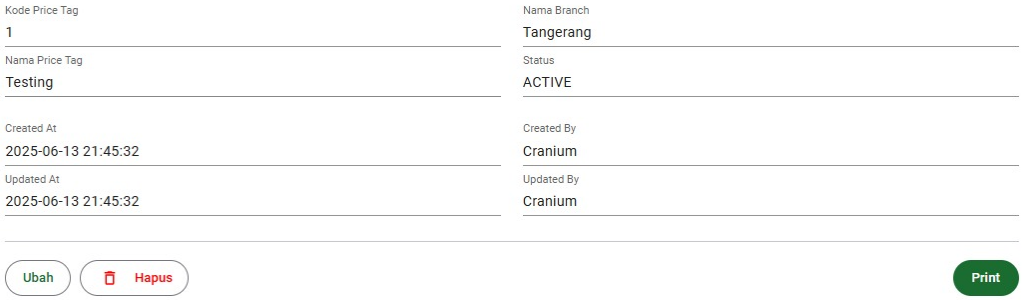
B.3 Hasil Kerja

Pada gambar berikut, terlihat bahwa antarmuka menampilkan nama kunci secara langsung, seperti `master.priceTag.code` alih-alih teks deskriptif seperti “Code Price Tag”. Ini menandakan bahwa entri penerjemahan belum tersedia pada sistem.



Gambar 3.5. Antarmuka sebelum perbaikan: key belum ditranslasi

Setelah entri *translation* ditambahkan, sistem mampu menampilkan teks dalam bentuk yang telah diterjemahkan dan lebih mudah dipahami pengguna. Gambar berikut menunjukkan hasil setelah perbaikan.



Gambar 3.6. Antarmuka setelah perbaikan: value diterjemahkan ditampilkan

Dengan menambahkan entri *translation* yang tepat, sistem ERP kini dapat menampilkan informasi dengan bahasa yang konsisten dan sesuai standar tampilan multibahasa. Hal ini meningkatkan kualitas antarmuka dan memberikan pengalaman pengguna yang lebih baik.

C Submodul Supplier

C.1 Analisis Masalah

Setelah dilakukan proses *quality assurance* pada submodul *Supplier*, ditemukan sebuah kendala yang cukup signifikan dalam proses pembuatan data supplier baru. Masalah ini muncul karena adanya ketidaksesuaian tipe data antara sisi *backend* dan *frontend*. Secara khusus, salah satu field pada form input memiliki tipe data yang berbeda dari yang diterima oleh sistem di *frontend*. Ketidaksesuaian ini menyebabkan data tidak dapat diproses dengan benar saat

dilakukan proses *create*, sehingga sistem gagal menyimpan data supplier yang baru dan mengakibatkan error pada sisi pengguna.

C.2 Solusi

Untuk menyelesaikan permasalahan ini, beberapa langkah perubahan dilakukan pada bagian *backend* dan basis data agar struktur dan format data menjadi konsisten:

1. Mengubah deklarasi tipe data pada `DTO` dan entitas dari:

```
private Long accountNumber;
```

   Menjadi:

```
private String accountNumber;
```

2. Melakukan penyesuaian pada saat pemetaan nilai dari inputan:

```
supplier.setAccountNumber(SupplierServiceUtil.defaultZeroLong
    (supplierUpdateDto.getAccountNumber()));
```

   Menjadi:

```
supplier.setAccountNumber(SupplierServiceUtil.
   defaultEmptyString
    (supplierCreateDto.getAccountNumber()))
```

3. Mengubah struktur tabel pada database PostgreSQL untuk menyesuaikan tipe data menjadi `VARCHAR(20)` dengan menjalankan perintah SQL berikut:

```
ALTER TABLE "master"."supplier"
    ALTER COLUMN account_number TYPE VARCHAR(20)
    USING account_number::VARCHAR(20);

ALTER TABLE "master"."supplier_aud"
    ALTER COLUMN account_number TYPE VARCHAR(20)
    USING account_number::VARCHAR(20);
```

Dengan perubahan tersebut, tipe data field `accountNumber` kini telah seragam di seluruh lapisan sistem, dan proses *create supplier* dapat dilakukan tanpa kendala tipe data.

C.3 Hasil Kerja

Setelah dilakukan penyesuaian pada deklarasi tipe data di sisi *backend*, pemetaan nilai input, serta struktur kolom pada basis data, proses pembuatan data supplier baru kini dapat dilakukan dengan lancar tanpa memunculkan error. Sistem sudah dapat menerima input dari frontend dengan format string sesuai yang diharapkan dan menyimpannya ke dalam database tanpa kegagalan.

Gambar berikut menunjukkan tampilan sistem ketika pengguna berhasil melakukan proses *create supplier* setelah perbaikan diterapkan.

Gambar 3.7. Tampilan setelah perbaikan: berhasil membuat data supplier baru

Perubahan ini juga berdampak pada peningkatan konsistensi data di seluruh sistem, serta meminimalkan potensi error serupa pada proses input data lainnya. Dengan demikian, pengalaman pengguna menjadi lebih baik dan sistem menjadi lebih stabil.

D Submodul ItemSupplier

D.1 Analisis Masalah

Setelah dilakukan proses *quality assurance* terhadap submodul *Item Supplier*, ditemukan beberapa permasalahan pada tampilan dan fungsi form, khususnya saat proses pembuatan data baru (*create*) dan pembaruan data (*update*). Masalah yang terjadi antara lain:

1. Pada form create item supplier, field `supplier` tidak menampilkan daftar pilihan dalam bentuk *dropdown* seperti seharusnya, melainkan hanya berupa *textbox*, sehingga pengguna tidak dapat memilih supplier dan menyebabkan proses penyimpanan data gagal.
2. Pada field `item` di form create dan update, sebelumnya masih menggunakan komponen `Autocomplete` yang hanya memuat data secara statis dan terbatas. Ketika jumlah data item sangat banyak, opsi tidak muncul atau tidak termuat semua.

D.2 Solusi

Untuk mengatasi permasalahan tersebut, beberapa perbaikan telah diterapkan:

- Pada field `supplier` di form create, telah diterapkan komponen `Autocomplete` dengan konfigurasi yang benar agar dapat memunculkan daftar supplier dalam bentuk *dropdown* dan pengguna dapat memilih supplier sesuai data yang tersedia.

```
<Controller
  name="supplierId"
  control={control}
  rules={{
    required: t('common.validation', { ns: 'common' }),
  }}
  render={({ field }) => (
    <Autocomplete
      fullWidth
      options={option}
      getOptionLabel={(option) =>
        `${option.supplierCode} - ${option.supplierName}`
      }
      renderInput={(params) => (
        <TextField
          {...params}
          label={t('master.itemSupplier.supplier', { ns: '
  master' })}
          required
          error={Boolean(errors['supplierId'])}
          helperText={
            errors['supplierId'] ? errors['supplierId'].
  message : ''
          }
        />
      )}
      isOptionEqualToValue={(option, value) => option.id ===
  value.id}
      onChange={(_, newValue) => {
        setValue('supplierId', newValue ? newValue.id : null,
   {
          shouldValidate: true,
        });
```

```
        trigger('supplierId');
      }}
    />
  )}
/>
```

Kode 3.8: Perbaikan Field Supplier dengan Autocomplete

- Pada field `item`, solusi yang diambil adalah mengganti komponen `Autocomplete` menjadi `InfiniteAutocomplete` yang memungkinkan pencarian dinamis dan pemuatan data secara bertahap. Hal ini sangat membantu ketika jumlah item sangat banyak dan tidak bisa dimuat sekaligus oleh `Autocomplete`.

```
<Controller
  name={'itemId'}
  control={control}
  rules={{
    required: t('common.validation', { ns: 'common' }),
  }}
  render={({ field }) => {
    const { onChange } = field;
    return (
      <InfiniteAutocomplete
        label={t('master.itemSupplier.item', { ns: 'master'
  })}
        options={itemOptions}
        isLoading={itemListInfinite.isLoading}
        hasNextPage={Boolean(itemListInfinite.hasNextPage)}
        fetchNextPage={itemListInfinite.fetchNextPage}
        dataTestId={'itemId'}
        setNewValue={(newValue) => {
          onChange(newValue ? newValue.id : null);
        }}
        setNewInputValue={(newInputValue) =>
          setItemInput(newInputValue)
        }
        validation={{
          required: true,
          error: errors['itemId']?.message,
          helperText: errors['itemId']?.message,
        }}
      />
```

```
29        );
30      }}
31  />
```

Kode 3.9: Penggunaan InfiniteAutocomplete untuk Field Item

Dengan perbaikan tersebut, proses *create* dan *update item supplier* kini berjalan sesuai harapan, dan pengguna dapat memilih data *supplier* maupun item secara lebih akurat dan efisien.

D.3 Hasil Kerja

Perbaikan yang telah diterapkan berhasil menyelesaikan permasalahan pada form *Item Supplier*. Tampilan field `supplier` kini telah sesuai harapan, yaitu menampilkan daftar supplier dalam bentuk *dropdown* dan pengguna dapat memilih data dari daftar tersebut. Gambar 3.8 menunjukkan bahwa data *supplier* sudah muncul dan bisa dipilih dengan mudah.

Gambar 3.8. Field `Supplier` berhasil menampilkan dropdown dengan data yang tersedia

Selanjutnya, pada field `item`, penerapan `InfiniteAutocomplete` memungkinkan pemuatan data secara bertahap sesuai input pencarian pengguna. Hal ini mencegah masalah seperti data tidak termuat ketika jumlah item terlalu banyak. Pada Gambar 3.9 ditunjukkan bahwa daftar item dapat ditelusuri melalui pencarian dan digulir jika jumlah data cukup banyak.

Gambar 3.9. Field `Item` menggunakan InfiniteAutocomplete dengan fitur scroll dan pencarian

Dengan adanya perbaikan ini, proses input data *item supplier* menjadi lebih stabil dan efisien, serta mampu menangani jumlah data yang besar tanpa mengganggu pengalaman pengguna.

3.3.5 Pengembangan Modul Selling

A Submodul Sales Order Return

A.1 Analisis Masalah

Dalam proses pengujian pada modul *Sales Order Return*, ditemukan *bug* terkait penerjemahan (*translation*) teks pada antarmuka pengguna. Meskipun *key* `common.ppn` sudah memiliki *value* yang benar pada file `translation JSON` (yaitu `"common.ppn": "PPN %"`), tampilan pada halaman masih menampilkan *key* sebelumnya yang tidak sesuai, yakni `common.ppnPercentage`. Hal ini menyebabkan label yang ditampilkan tidak sesuai dengan hasil terjemahan yang diharapkan.

A.2 Solusi

Solusi dari permasalahan ini adalah mengganti *key* yang digunakan di sisi *frontend* dari `common.ppnPercentage` menjadi `common.ppn`, karena *key* yang benar dan telah tersedia di file *translation* adalah `common.ppn`. Berikut adalah cuplikan perubahan kode yang dilakukan:

```
<TableCell className={style.detailTableCell}>
  {t('common.ppnPercentage', { ns: 'common' })}
//Menjadi
```

```
  {t('common.ppn', { ns: 'common' })}
</TableCell>

<TableCell className={style.detailTableCell}>
  {t('common.pphPercentage', { ns: 'common' })}
//Menjadi
  {t('common.pph', { ns: 'common' })}
</TableCell>
```

Kode 3.10: Perbaikan Key Translation di Frontend

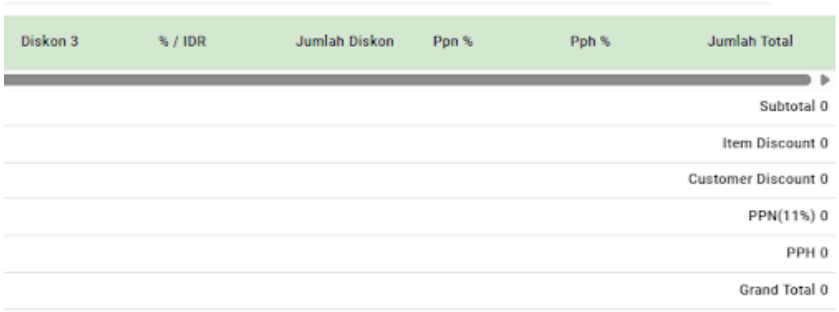
Dengan perbaikan tersebut, label `PPN %` dan `PPH %` kini ditampilkan dengan benar sesuai dengan *value* dari file *translation*, sehingga memperbaiki pengalaman pengguna dan konsistensi antar modul.

A.3 Hasil Kerja

Setelah dilakukan perubahan pada *key translation*, hasilnya dapat terlihat pada tampilan antarmuka pengguna. Sebelumnya, teks yang ditampilkan masih berupa `common.ppnPercentage` dan `common.pphPercentage`, yang seharusnya sudah diterjemahkan menjadi label seperti `PPN %` dan `PPH %` sesuai isi file `translation JSON`.

Gambar 3.10 menunjukkan tampilan sebelum dilakukan perbaikan, di mana label belum berubah dan masih menampilkan nama *key*. Sementara itu, Gambar 3.11 menampilkan hasil setelah perbaikan, di mana teks telah berubah menjadi label yang benar sesuai dengan yang telah didefinisikan dalam file *translation*.

Gambar 3.10. Tampilan sebelum perbaikan: label masih berupa key translation `common.ppnPercentage`



Gambar 3.11. Tampilan setelah perbaikan: label telah berubah menjadi `PPN %` dan `PPH %`

Perubahan ini meningkatkan konsistensi antarmuka pengguna dan mempermudah pemahaman pengguna terhadap informasi yang ditampilkan.

3.3.6 Pengembangan Modul Warehouse

A Submodul Expedition dan Finished Expedition

A.1 Analisis Masalah

Dalam struktur menu aplikasi, ditemukan adanya duplikasi modul *Expedition* dan *Finished Expedition* yang tampil baik di dalam menu *Warehouse* maupun *Shipping*. Keberadaan modul yang sama di dua lokasi berbeda dapat menyebabkan kebingungan bagi pengguna terkait di mana mereka seharusnya mengakses fitur ekspedisi. Selain itu, hal ini juga berpotensi menimbulkan inkonsistensi dalam pengelolaan akses dan penggunaan sistem.

A.2 Solusi

Solusi yang diterapkan adalah dengan menghapus modul *Expedition* dan *Finished Expedition* dari menu *Warehouse*, karena modul tersebut telah tersedia secara lengkap dan relevan di menu *Shipping*. Dengan penghapusan tersebut, navigasi menjadi lebih sederhana dan fokus pengguna terhadap struktur menu meningkat, sekaligus mengurangi redundansi modul di dalam sistem. Langkah-langkah yang dilakukan mencakup:

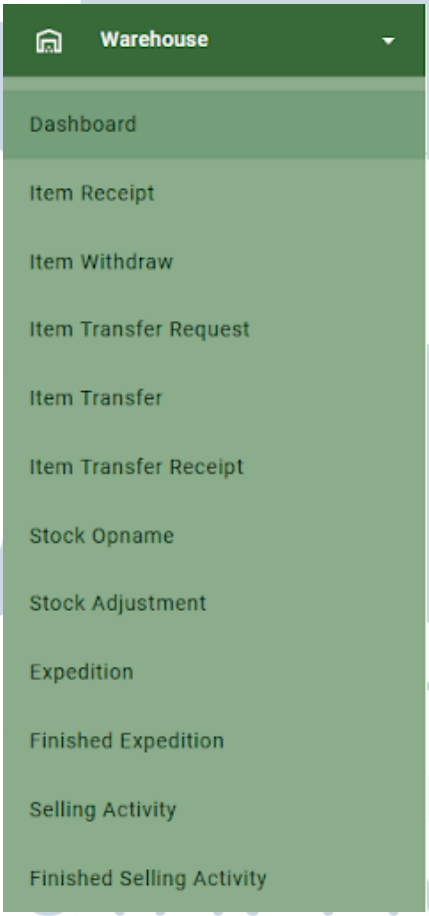
- Menghapus entri menu untuk *Expedition* dan *Finished Expedition* di bagian *Warehouse*.

- Melakukan validasi tampilan UI untuk memastikan bahwa modul tersebut hanya muncul di menu *Shipping*.

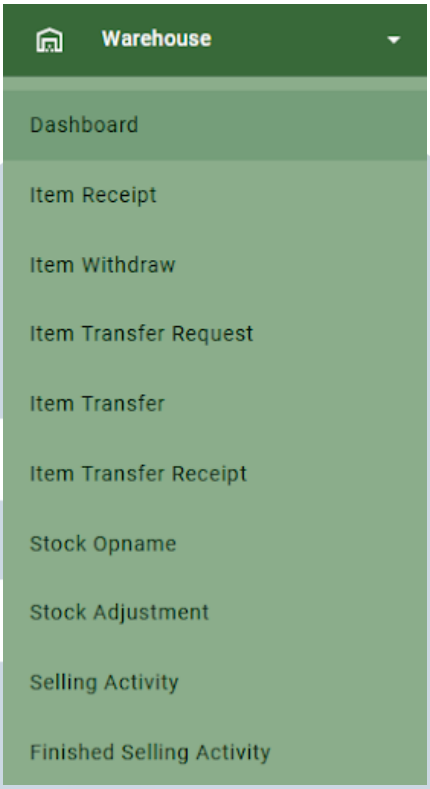
A.3 Hasil Kerja

Perubahan yang dilakukan berhasil menghilangkan modul *Expedition* dan *Finished Expedition* dari menu *Warehouse*. Sebelumnya, kedua modul tersebut muncul secara redundan di dua lokasi, yakni di menu *Warehouse* dan *Shipping*.

Gambar 3.12 menunjukkan tampilan menu sebelum dilakukan penghapusan, di mana modul *Expedition* dan *Finished Expedition* masih terlihat pada bagian *Warehouse*. Setelah dilakukan penghapusan, seperti ditunjukkan pada Gambar 3.13, modul-modul tersebut tidak lagi tampil di *Warehouse* dan hanya tersedia di *Shipping*.



Gambar 3.12. Tampilan sebelum perbaikan: Modul Expedition dan Finished Expedition masih terlihat di menu Warehouse



Gambar 3.13. Tampilan setelah perbaikan: Modul Expedition dan Finished Expedition telah dihapus dari menu Warehouse

Dengan penghapusan ini, struktur menu menjadi lebih jelas dan mengurangi kemungkinan kebingungan pengguna terhadap lokasi fitur ekspedisi yang seharusnya diakses.

3.3.7 Pengembangan Modul Finance

A Submodul Journal

A.1 Analisis Masalah

Pada fitur pengelolaan jurnal keuangan, ditemukan dua permasalahan utama terkait dengan kolom *Journal Type*:

1. Pada mode *view*, nilai *Journal Type* ditampilkan dalam bentuk numerik (`1` atau `2`), bukan sebagai teks yang deskriptif seperti `Debit` atau `Credit`. Hal ini membuat tampilan kurang informatif bagi pengguna.
2. Pada mode *update*, inputan `Journal Type` secara default menampilkan pilihan pertama (`Debit`) tanpa memperhatikan nilai aktual yang sudah

tersimpan di database. Akibatnya, pengguna bisa menyimpan perubahan data yang tidak sesuai jika tidak sadar telah terjadi perubahan nilai default.

A.2 Solusi

Solusi yang diterapkan untuk mengatasi kedua masalah di atas adalah sebagai berikut:

- Untuk tampilan *view*, digunakan pemetaan melalui `enum JournalTypeDetailEnum` agar nilai numerik dapat ditampilkan sebagai teks yang bermakna. Berikut adalah implementasinya:

```
export enum JournalTypeDetailEnum {
  DEBIT = 1,
  CREDIT = 2,
}

// Pemanggilan pada render tabel
render: ({ entry }) => (
  <span>{JournalTypeDetailEnum[Number(entry.journalType)]}</
    span>
)
```

Kode 3.11: Enum dan render journalType saat view

- Untuk tampilan *update*, inputan diubah menggunakan `Controller` dari `react-hook-form` agar nilai awal (*initial value*) diatur berdasarkan data yang sudah tersimpan, bukan nilai default. Berikut kode yang digunakan:

```
<Controller
  name={`journalFinanceDetailUpdateDtoList.${index}.
    journalType`}
  control={control}
  render={({ field }) => (
    <TextField
      select
      fullWidth
      id={style.detailTableTextFieldId}
      {...field}
    >
      <MenuItem value={JournalTypeDetailEnum.DEBIT}>
        {t('finance.journal.journalType.debit', { ns: '
    finance' })}
```

```
13        </MenuItem>
14        <MenuItem value={JournalTypeDetailEnum.CREDIT}>
15          {t('finance.journal.journalType.credit', { ns: '
    finance' })}
16        </MenuItem>
17      </TextField>
18    )}
19  />
```

Kode 3.12: Input Journal Type pada mode update

- Dengan perubahan ini, tampilan data menjadi lebih jelas dan konsisten, serta proses update data menjadi lebih andal karena nilai input mempertahankan data sebelumnya.

A.3 Hasil Kerja

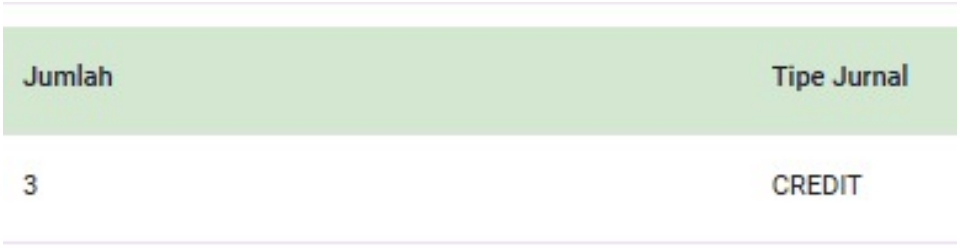
Setelah dilakukan implementasi perbaikan, tampilan kolom `Journal Type` kini lebih informatif dan sesuai dengan nilai yang tersimpan.

Gambar 3.14 memperlihatkan tampilan pada mode *view* sebelum dilakukan perbaikan, di mana kolom `Journal Type` hanya menampilkan angka `1` atau `2`, tanpa makna yang jelas bagi pengguna.

Jumlah	Tipe Jurnal
1000	2
1000	1

Gambar 3.14. Tampilan mode view sebelum perbaikan: Journal Type ditampilkan dalam bentuk angka

Setelah dilakukan perubahan, Gambar 3.15 menunjukkan tampilan mode *view* setelah perbaikan, di mana angka tersebut telah dikonversi menjadi teks deskriptif `Debit` atau `Credit` sesuai dengan enum yang telah ditentukan.

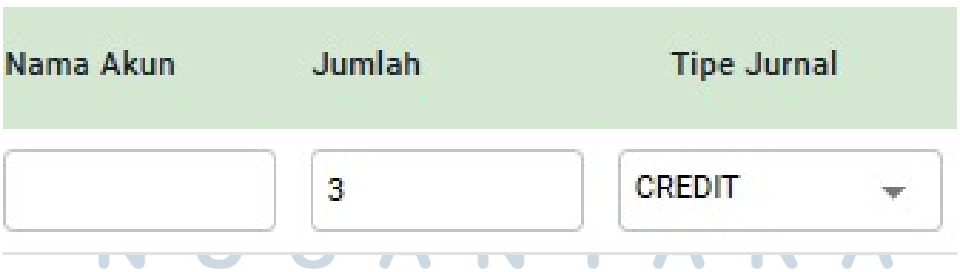


Gambar 3.15. Tampilan mode view setelah perbaikan: Journal Type ditampilkan sebagai teks (`Debit`/`Credit`)

Untuk mode *update*, Gambar 3.16 menunjukkan kondisi sebelum perbaikan, di mana nilai input selalu kembali ke default `Debit` meskipun nilai sebenarnya adalah `Credit`. Hal ini dapat menyebabkan kesalahan penyimpanan data jika pengguna tidak menyadari perubahan tersebut.

Gambar 3.16. Tampilan mode update sebelum perbaikan: Nilai Journal Type selalu kembali ke default (`Debit`)

Gambar 3.17 menunjukkan hasil setelah perbaikan, di mana nilai yang muncul pada input `Journal Type` sesuai dengan data sebenarnya yang tersimpan, seperti `Credit`. Dengan ini, pengguna tidak perlu mengatur ulang nilai jika tidak ada perubahan yang diinginkan.



Gambar 3.17. Tampilan mode update setelah perbaikan: Nilai Journal Type tetap menampilkan data sesuai database (`Credit`)

Dengan perbaikan tersebut, tampilan data menjadi lebih mudah dipahami

oleh pengguna, serta mencegah perubahan data yang tidak disengaja akibat kesalahan nilai default. Validasi visual dan pengalaman pengguna juga menjadi lebih baik dan minim risiko kesalahan.

3.4 Kendala dan Solusi yang Ditemukan

Selama menjalani kerja praktik, terdapat beberapa tantangan yang dihadapi dalam proses pengembangan ERP Cranium, antara lain:

- Diperlukan pemahaman tambahan terhadap struktur dan pola arsitektur yang digunakan, baik pada sisi *backend* maupun *frontend*, karena terdapat beberapa konsep baru yang belum sepenuhnya dikuasai sebelumnya.
- Beberapa modul dan submodul yang ada dikembangkan dengan pendekatan yang tidak seragam, sehingga terdapat perbedaan dalam penerapan fitur yang secara fungsi serupa, namun dibangun dengan cara yang berbeda.
- Ada beberapa modul dan submodul yang mengharuskan kita untuk mengerti flow business logicnya agar kita bisa mengerjakan task dengan aman.
- Sejumlah pekerjaan harus menunggu hingga *pull request* sebelumnya selesai di-*merge*, sehingga menyebabkan proses pengerjaan mengalami keterlambatan.

Untuk mengatasi kendala-kendala tersebut, langkah-langkah penyelesaian yang dilakukan adalah sebagai berikut:

- Berusaha menyesuaikan diri dengan sistem kerja dan arsitektur teknologi yang digunakan di perusahaan, serta melakukan pembelajaran secara mandiri untuk mempercepat proses adaptasi.
- Menyesuaikan pendekatan pengembangan dengan praktik terbaik (*best practice*) yang relevan, serta mengikuti pola yang sudah digunakan jika memungkinkan demi menjaga konsistensi kode.
- Dengan sering berkoordinasi dengan Project Manager mengenai flow business logic modul dan submodulnya.
- Secara rutin melakukan koordinasi dengan anggota tim dan pembimbing agar tidak terjadi keterlambatan signifikan dalam pengembangan, termasuk mempercepat proses *merge* PR yang menjadi dependensi tugas selanjutnya.