BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama masa pemagangan, mahasiswa magang ditempatkan di Departemen *Enterprise Technology* CITIS (*Corporate IT & IS*) yang dipimpin oleh Anwari Rahman sebagai *Manager Enterprise Technology Department*. Mahasiswa magang berperan sebagai *Intern Junior Software Engineer* di bawah bimbingan dan pengawasan *Senior Software Engineer*, Herman Gusti Anugrah, yang juga bertindak sebagai *mentor* sekaligus sebagai supervisi. Mahasiswa magang melaporkan hasil kerjanya kepada supervisi, yang kemudian bertanggung jawab kepada manajer. Tugas-tugas diberikan oleh manajer departemen melalui supervisi kepada mahasiswa magang.

Proses penugasan dan alokasi proyek di CITIS dimulai dengan penyampaian daftar proyek kepada direktur CITIS. Direktur kemudian berkoordinasi dengan *General Manager* divisi terkait untuk meneruskan informasi proyek. *General Manager* menyampaikan proyek kepada manajer departemen yang bertanggung jawab atas pengembangan proyek tersebut. Setelah menerima informasi, manajer departemen mendistribusikan tugas melalui rapat bersama anggota departemen.

3.2 Tugas yang Dilakukan

Selama menjalani periode magang di CITIS Kompas Gramedia sebagai *Junior Software Engineer*, tugas yang dilakukan berfokus pada kustomisasi fitur-fitur yang sudah ada ataupun membuat fitur baru, yang dikembangkan menggunakan bahasa pemrograman Python dengan *framework* Odoo. Selain menggunakan perangkat yang telah disebutkan, pengerjaan tugas juga berpedoman pada prinsip *Clean Code* (kode bersih). Prinsip ini menekankan penulisan kode yang tidak hanya berfungsi, tetapi juga mudah dibaca, dipahami, dan dikelola oleh pengembang lain. Hal ini penting karena sebagian besar waktu pengembangan dihabiskan untuk membaca kode, bukan menulisnya [9]. Penerapan tugas dan prinsip tersebut terwujud dalam beberapa kustomisasi utama di sisi teknis, yaitu:

- **Invoicing**

  Dalam modul ini, dilakukan perubahan pada beberapa menu turunan

di bagian *Reporting* dengan mengganti nama label agar lebih mudah dipahami oleh pengguna. Selain itu, dilakukan juga perbaikan pada menu *Payment Order Request* untuk memastikan kolom informasi bank ter-update secara dinamis ketika pengguna mengganti akun bank partner, sehingga meningkatkan akurasi data

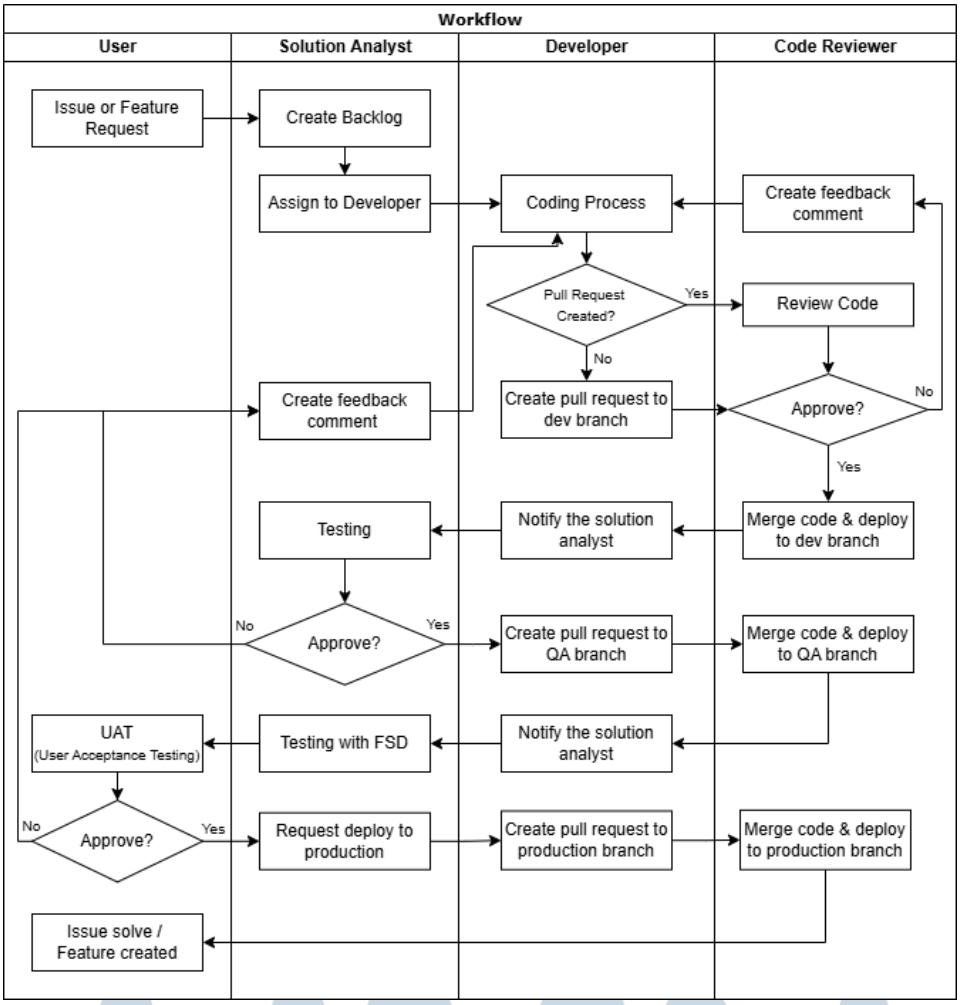
- **Purchase Request**

  Pada modul *Purchase Request*, dilakukan optimasi alur kerja dengan mengganti pesan *error* sistem yang teknis menjadi peringatan yang lebih deskriptif bagi pengguna. Hal ini diterapkan saat pengguna membuat *Purchase Order* dari *Purchase Request* tanpa memilih tipe pesanan, sehingga meminimalisir kebingungan.

- **Transportasi**

  Salah satu pekerjaan terbesar adalah pengembangan fitur baru dari awal pada modul Transportasi. Ini mencakup pembuatan menu *master data* baru yaitu *Constanta Cost* dan *Cost Route*. Tujuan dari pengembangan ini adalah untuk menstandarkan dan memusatkan pengelolaan data biaya rute ekspedisi yang sebelumnya tersebar.

Dalam mengerjakan tugas yang diberikan, proses kerja selama magang mengikuti alur sebagai berikut:



Gambar 3.1. Alur kerja di *Corporate IT & IS* Kompas Gramedia

Gambar 3.1 mengilustrasikan alur pelaksanaan kerja magang, dan berikut adalah penjelasan terkait ilustrasi tersebut:

1. **Create Backlog**

   (a) *Solution Analiyst* bertugas untuk menghubungkan antara permintaan atau masalah yang dimiliki *user* kepada *developer* dengan membuat backlog di Azure DevOps boards yang fungsinya adalah mendeskripsikan kebutuhan *user* agar dapat dipahami oleh *developer*.

   (b) Sebagai *developer* untuk dapat mengerjakan *backlog* yang ada, dapat dilakukan dengan dua cara yaitu:

   - *Solution Analyst* menunjuk langsung *developer* yang sesuai berdasarkan keahlian masing-masing dan hasil pembahasan sebelumnya.

- Melakukan *self-assign*, dengan mengambil *backlog* yang tersedia langsung di Azure DevOps.

2. **Proses Coding**

   (a) Setelah mengambil atau ditunjuk oleh *Solution Analyst* untuk sebuah backlog, maka *developer* akan mengerjakannya sesuai dengan deskripsi dan petunjuk yang diberikan

   (b) Bila ada yang belum paham atau kurang mengerti mengenai deskripsi backlog tersebut, *developer* dapat bertanya kepada *Solution Analyst* yang membuat backlog tersebut, atau kepada tim *developer* lain.

   (c) Setelah *developer* selesai mengerjakan kode dan hasilnya di *local environment* sudah sesuai dengan deskripsi backlog, maka *developer* akan melakukan *commit* dan *push* branch ke Azure DevOps untuk melakukan *pull request* ke branch *development* agar dapat direview oleh code reviewer.

3. **Review Kode**

   (a) Setiap kode yang diajukan melalui *pull request* akan direview oleh *reviewer* untuk memastikan kesesuaian dan efektivitas implementasinya serta menerapkan prinsip dari *clean code*.

   (b) Jika terdapat masukan atau catatan dari *reviewer*, *developer* wajib melakukan revisi kode sesuai permintaan tersebut.

   (c) Setelah pekerjaan disetujui, kode akan dideploy ke *development server* untuk dilakukan pengujian oleh *Solution Analyst*

4. **Proses Deployment**

   (a) Setelah *Solution Analyst* selesai melakukan *testing* terhadap perubahan yang dilakukan oleh *developer* di server *dev*, *Solution Analyst* akan mengajukan *request deployment* ke QA kepada *developer* agar dapat dilakukan pengujian bersama *Financial System Development* (FSD) dan *User Acceptance Testing* (*UAT*) bersama user.

   (b) Jika perubahan yang dilakukan belum memenuhi harapan atau terdapat permintaan tambahan dari user ataupun *Financial System Development* (FSD), *Solution Analyst* akan memberikan *feedback* kepada *developer* melalui kolom komentar pada *backlog*.

(c) Jika perubahan sudah sesuai dan user telah menyetujui perubahan tersebut, *Solution Analyst* akan mengajukan *request deployment* ke server *production*.

3.3 Uraian Pelaksanaan Magang

Tabel 3.1 berikut merupakan ringkasan kegiatan selama periode magang, mulai dari minggu pertama hingga minggu ke-14. Uraian ini mencakup tugas yang dikerjakan, pembelajaran atau ilmu yang diperoleh, serta pencapaian selama masa magang.

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

Minggu ke-	**Pekerjaan yang dilakukan**
1	Melakukan perkenalan dengan teman-teman yang ada di kantor dan juga melakukan instalasi Odoo di Laptop pribadi.
2	Mempelajari environtment Odoo dan python sebagai program bahasa yang digunakan di Odoo.
3	1. Membuat sebuah modul aplikasi baru di Odoo dengan nama kg_master_data. 2. Mempelajari dan membuat sebuah role akses baru bernama Master Koordinator yang terdiri dari User, Supervisor, dan Manager. 3. Menetapkan hak akses untuk masing-masing role sesuai dengan tingkatan perannya.
	Lanjut ke halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu ke-	Pekerjaan yang dilakukan
4	1. Mempelajari dan membuat action menu baru pada submenu Master data yaitu Constanta Cost dan Cost Route pada menu Transportasi. 2. Membuat form dan treeview dari action menu yang sudah dibuat. 3. Mempelajari cara agar suatu form dapat menampilkan list data dari model lain. 4. Membuat query SQL untuk memasukkan data dari file Excel ke dalam database Odoo (PostgreSQL), serta menerapkan aturan sebelum datanya dimasukkan ke dalam database.
5	1. Mengerjakan backlog enhancement treeview di PR line. 2. Mengerjakan backlog enhancement Request Uang Jalan TMS. 3. Mempelajari dan membuat fitur archieve pada record Constanta Cost.
6	1. Mengerjakan revisi terkait roles di backlog penjagaan master product stockable. 2. Memperbaiki pengaturan roles pada master koordinator dan inventory, agar hanya pihak yang berwenang dapat melakukan perubahan pada master data produk. 3. Mempelajari tentang MRT file dan tools Stimulsoft Designer, guna untuk menambahkan logo di dokumen SPK (Surat Perintah Kerja) pada menu Purchase Order - SPK.
Lanjut ke halaman berikutnya	

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu ke-	Pekerjaan yang dilakukan
7	1. Menyelesaikan backlog setting logo di SPK yang minggu sebelumnya. 2. Mengerjakan backlog Update TMS Access - Allow Edit Uang Jalan. 3. Mengerjakan backlog penjagaan onchange di bank statement yang ada pada menu Cash Bank.
8	1. Mengerjakan backlog enhancement pada proses get bank account untuk transaksi VB dan API VB. 2. Mengerjakan feedback yang diberikan oleh Solution Analyst terkait dengan backlog setting logo di SPK & Update TMS Access - Allow Edit Uang Jalan.
9	1. Mencari dan mengerjakan bug yang ada pada saat proses assign purchase org dan purchaser id saat pembuatan PO type 3. 2. Menyelesaikan backlog pada minggu sebelumnya yaitu enhancement pada proses get bank account untuk transaksi VB dan API VB. 3. Mengerjakan feedback dari Solution Analyst terkait dengan backlog Update TMS Access - Allow Edit Uang Jalan.
10	1. Mengerjakan backlog clear warnings when creating PO from PR. 2. Mengerjakan backlog No DO receiving dibuat unique pada menu Inventory.
	Lanjut ke halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu ke-	Pekerjaan yang dilakukan
11	1. Mengerjakan backlog perubahan pengambilan akun pada transaksi MSR, PR, PO, VB, dan SRO, karena fitur usage, data di production belum terkonfigurasi, sehingga sementara masih menggunakan metode lama. 2. Menangani feedback pada field action menu Cost Route dan memperbaiki field Settlement Cost di action menu TMS Trip yang tidak bertambah saat ada Additional Cost.
12	1. Mengerjakan backlog Create Report Product Movement untuk menampilkan laporan tiap transaksi pada model Stock Move, dengan filter berdasarkan field pada record transaksi. 2. Berdiskusi dengan Manajer dan Solution Analyst terkait dengan backlog Create Report Product Movement. 3. Mempelajari cara memfilter tiap record data pada database PostgreSQL menggunakan SQL Query dan menampilkannya di UI Odoo.
13	1. Mengerjakan feedback dari Solution Analyst mengenai backlog Create Report Product Movement yaitu menambahkan 1 kolom Document Header. 2. Memperbaiki error pada backlog Add Master Cost Route ketika user mengupload data excel, dengan mengubah template excel yang masih kurang 1 kolom.
	Lanjut ke halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu ke-	Pekerjaan yang dilakukan
14	1. Mengerjakan backlog *Show Journal Entries To All Server* 2. Mengerjakan backlog Tambah printout versi QWeb untuk BS dan PD

Berdasarkan tabel 3.1 pekerjaan-pekerjaan yang telah disebutkan dapat dikelompokkan sebagai berikut:

3.3.1 Pembelajaran

Selama masa pembelajaran pada minggu pertama sampai minggu kedua, seluruh proses pengembangan dan pemahaman teknis berpedoman pada dokumentasi pengembang resmi Odoo 11 [10] sesuai dengan versi Odoo yang digunakan oleh Kompas Gramedia, guna mempelajari struktur framework, API, dan praktik terbaik dalam kustomisasi modul. Peserta magang mempelajari hal-hal berikut:

A. Pengenalan Framework Odoo ERP

Pada minggu pertama, peserta magang melakukan instalasi Odoo pada laptop pribadi dengan panduan dari modul yang disediakan oleh Kompas Gramedia. Modul tersebut berisi instruksi lengkap, mulai dari instalasi Python hingga versi Odoo yang digunakan di sistem KG-ERP. Selain memperoleh materi pembelajaran dari modul-modul tambahan yang disediakan oleh Kompas Gramedia untuk memperdalam pemahaman terkait Odoo dan Python, khususnya dalam mempelajari struktur Odoo, dokumentasi resmi Odoo versi 11 juga dimanfaatkan sebagai sumber referensi utama [10].

B. Alur Kerja di dalam Azure DevOps

Selama masa pembelajaran pada minggu pertama dan kedua, peserta magang juga mempelajari alur proses di dalam Azure DevOps, sebuah platform kolaboratif untuk menyatukan antara pengembang, manajer proyek dan juga kontributor lainnya [11]. Hal-hal yang dipelajari meliputi

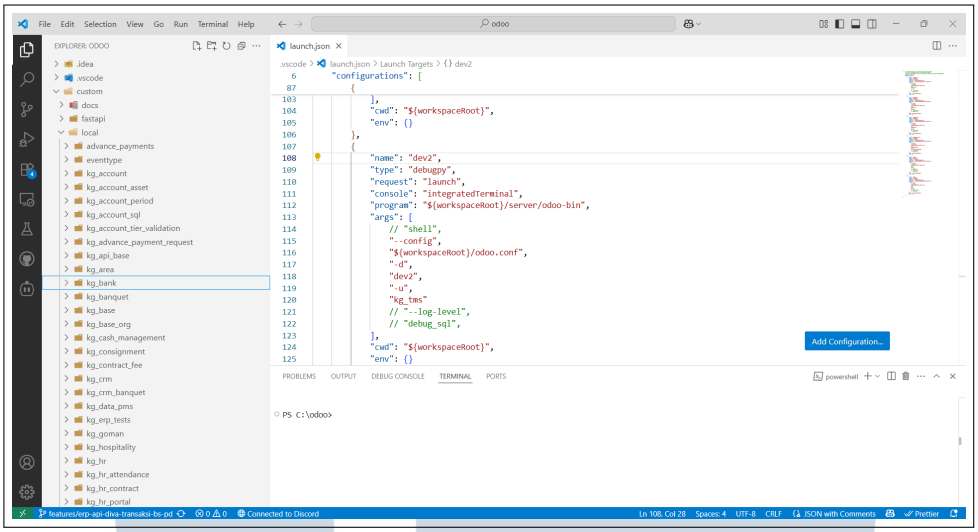
penggunaan *Azure Boards* untuk mengelola backlog, penetapan *base branch* sebagai dasar pengerjaan fitur, pembuatan *pull request* untuk proses peninjauan kode, hingga proses *deployment* ke server pengembangan menggunakan *Azure Pipelines*.

C. Alur Kerja di dalam CITIS

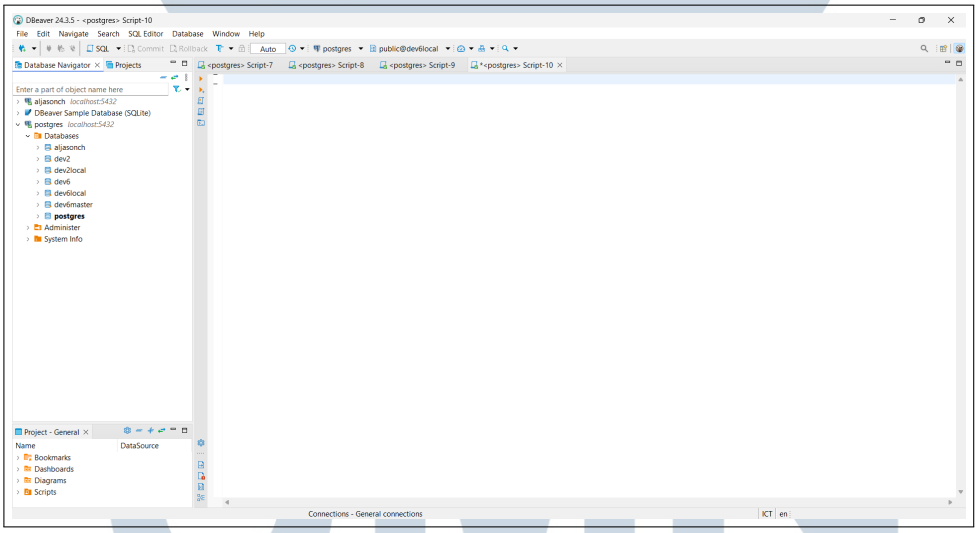
Setelah mempelajari konsep-konsep dasar *framework* Odoo, peserta magang melanjutkan pembelajaran dengan memahami alur kerja di CITIS yang menerapkan metodologi *agile sprint* dalam proses pengembangan proyek. Dalam metodologi ini, peserta magang mengikuti *daily meeting* setiap pagi untuk melaporkan kegiatan yang telah dilakukan pada hari sebelumnya. Selain itu, setiap dua minggu sekali diadakan *sprint meeting* untuk membahas dan mengevaluasi *backlog* yang telah diselesaikan pada *sprint* sebelumnya.

D. Tools yang digunakan

Dalam mendukung proses pengembangan modul ERP di Kompas Gramedia, peserta magang diperkenalkan dengan beberapa alat pengembangan, seperti PyCharm dan Visual Studio Code sebagai *code editor*, serta DBeaver sebagai *Database IDE*. Untuk keperluan pengembangan, Visual Studio Code digunakan sebagai *code editor* utama yang telah dikonfigurasi khusus untuk menjalankan program Odoo. Berikut adalah tampilan antarmuka Visual Studio Code pada Gambar 3.2 dan DBeaver sebagai *Database IDE* pada Gambar 3.3.

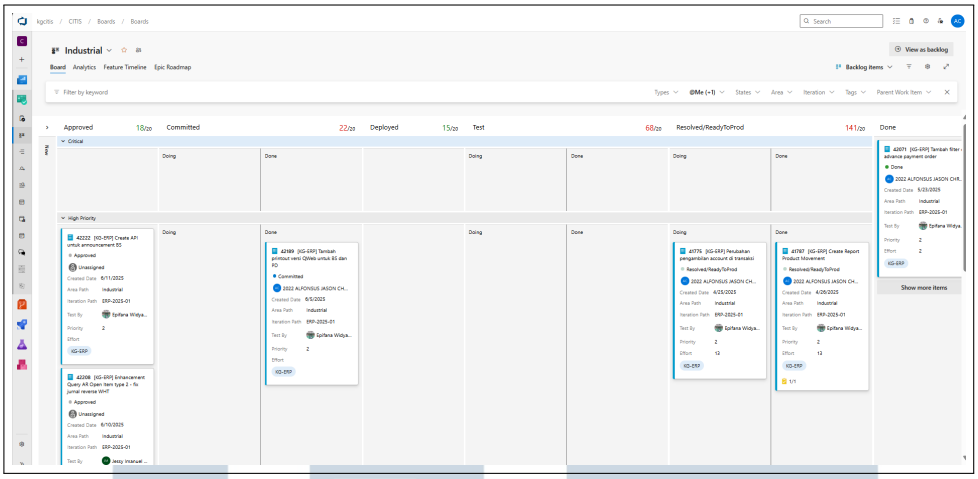


Gambar 3.2. Tampilan antarmuka Visual Studio Code



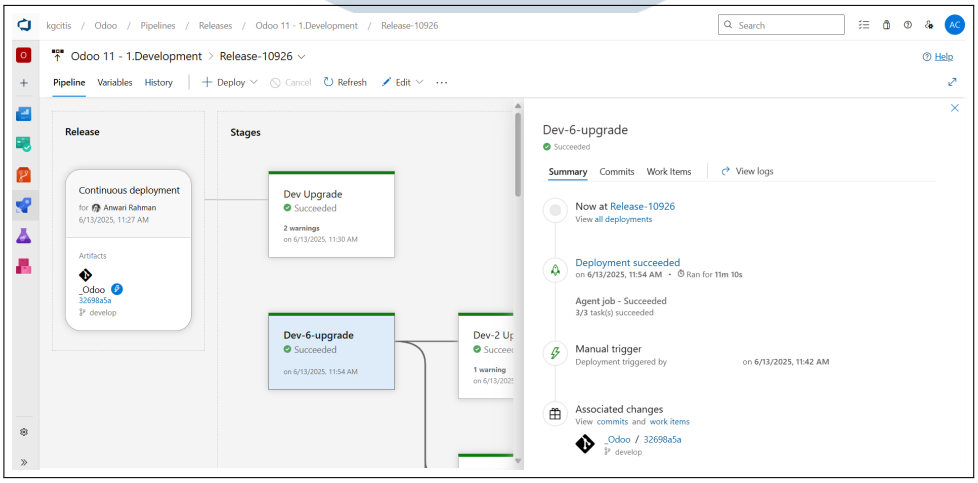
Gambar 3.3. Tampilan antarmuka Dbeaver

Dalam hal manajemen *backlog*, CITIS menggunakan *Azure DevOps* agar peserta magang dapat dengan mudah mengakses daftar backlog lengkap dengan deskripsi dan status pengerjaan, sehingga alur kerja lebih terorganisir dan terpantau secara efektif. Contoh tampilan *Azure Boards* dapat dilihat pada Gambar 3.5



Gambar 3.4. Tampilan antarmuka *boards* pada Azure DevOps

Proses *deployment* ke server pengembangan diatur menggunakan *Azure Pipeline*. Gambar 3.4 memperlihatkan contoh pipeline yang digunakan untuk mengotomatisasi proses *release* dan *deployment* modul ke lingkungan server *development*.



Gambar 3.5. Tampilan antarmuka *pipeline* pada Azure DevOps

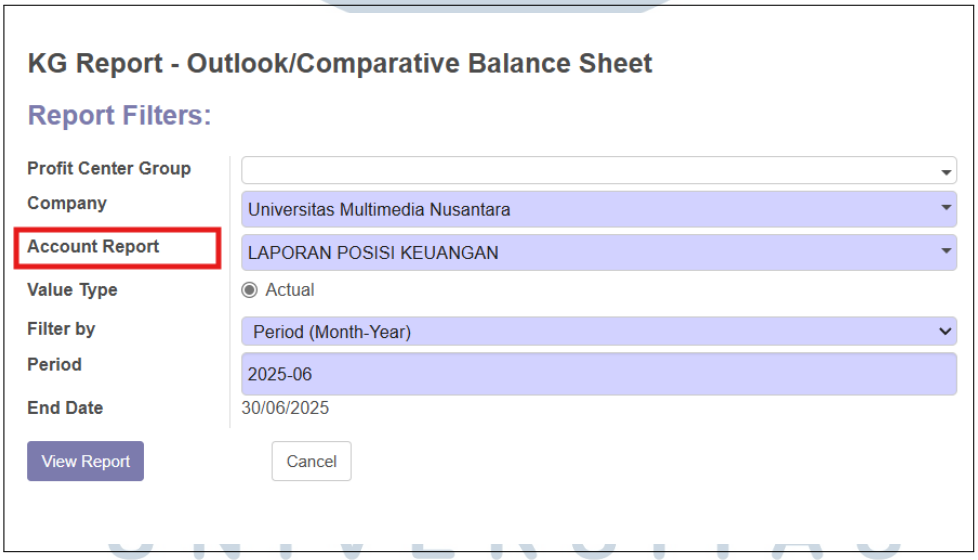
Proses pembelajaran selama magang tidak hanya meningkatkan kemampuan teknis peserta magang, tetapi juga membentuk pola pikir kerja yang lebih terstruktur, kolaboratif, dan selaras dengan praktik pengembangan perangkat lunak yang profesional.

A Pengerjaan Backlog

Selama periode magang, penugasan dilakukan di lingkup *Industrial*, yaitu unit yang menangani kebutuhan bisnis operasional seperti pencatatan transaksi, pengelolaan data master, alur persetujuan, serta otomatisasi proses. Unit ini juga bertanggung jawab dalam melakukan analisis kebutuhan, perancangan, hingga implementasi solusi untuk meningkatkan efisiensi dan akurasi kerja. Berikut merupakan daftar *backlog* yang dikerjakan selama periode magang.

A.1 Rename Label Account Report in Managerial Reporting

Backlog ini merupakan *backlog* pertama yang dikerjakan selama masa magang. Kebutuhan yang harus dipenuhi dalam backlog ini adalah mengganti nama dari suatu *field* di berbagai menu turunan pada submenu *Reporting* di *Invoicing*, seperti *Income Statement*, *Balance Sheet*, *Cash Flow*, *Comparative Income Statement*, dan *Comparative Balance Sheet*. Contoh *field* pada salah satu menu turunan tersebut dapat dilihat pada Gambar 3.6.



Gambar 3.6. Tampilan salah satu *child menu Report* pada submenu *Reporting*

Secara teknis, label ini terhubung ke sebuah field dengan nama `account_report_id`. Field ini merupakan relasi Many2one ke model `account.financial.report` yang relasi database antar modelnya diatur oleh ORM *(Object Relational Mapping)* bawaan Odoo dan fungsinya adalah agar pengguna dapat memilih jenis laporan keuangan yang ingin ditampilkan (misalnya

Neraca atau Laba Rugi). Untuk mengubah *label* visual tanpa mengubah nama *field* di database, Odoo menyediakan dua pendekatan. Pendekatan pertama adalah dengan mengubah atribut `string` secara langsung pada definisi *field* di dalam berkas Python, seperti yang ditunjukkan pada Kode 3.1. Pendekatan kedua, yang lebih umum digunakan untuk kustomisasi tampilan, adalah dengan menimpa (*override*) atribut `string` dari *field* tersebut melalui berkas XML yang mengatur tata letak antarmuka, seperti pada Kode 3.2. Contoh pada salah satu Python *script* untuk melakukan perubahan dapat dilihat pada Kode 3.1.

```
    account_report_id = fields.Many2one('account.financial.report'
    , string='Report Name', required=True, default=
    _get_account_report)
```

Kode 3.1: Kode untuk mengganti nama *field*

Contoh pada salah satu Python *script* untuk melakukan perubahan dapat dilihat pada Kode 3.2.

```
<field name="account_report_id" string="Report Name"/>
```

Kode 3.2: Kode untuk mengganti nama *field* melalui XML

Perubahan yang dilakukan ini hanya berdampak pada sisi tampilan *(view)* dan tidak mengubah alur data atau logika bisnis apa pun pada *backend*. Dengan mempelajari kedua cara tersebut, pemahaman mengenai cara menyesuaikan tampilan nama *field* menjadi lebih baik, baik dari sisi kode Python maupun struktur XML.

A.2 Clear warnings when creating Purchase Order from Purchase Request

Pada backlog ini, kebutuhannya adalah ketika pengguna ingin membuat record *Purchase Order* melalui *Purchase Request Line* pada menu *Purchase Request*, dan user tidak memilih salah satu opsi dari *Order Type*, yaitu *RFQ*, *Draft PO*, *Confirm PO* yang berupa field *Selection*, lalu mengklik tombol *Create RFQ / Draft PO*, maka akan muncul *error* yang ditampilkan seperti kode 3.3.

```
Odoo Server Error
....
"/home/odoo6/odoo/custom/local/kg_purchase/wizards/
    purchase_request_line_make_purchase_order.py", line 634, in
    make_purchase_order
    'views': [(view_id.id, 'form')],
```

```
UnboundLocalError: local variable 'view_id' referenced before
    assignment
```

Kode 3.3: Kode error pada Odoo

Kesalahan pada kode 3.3 dapat membingungkan pengguna awam karena tidak ada petunjuk yang jelas mengenai penyebabnya. Oleh karena itu, perlu dilakukan perubahan pada sisi kode agar pengguna memahami tindakan yang harus dilakukan untuk menghindari munculnya *error* tersebut. Akar masalahnya terletak pada alur logika fungsi `make_purchase_order`, sebuah fungsi yang digunakan untuk membuat *Purchase Order* yang nilainya berasal dari pengguna dan record *Purchase Request* yang dipilih. Fungsi ini bergantung pada nilai dari field `order_type` untuk menentukan tampilan (view) mana yang harus dibuka. Sebuah variabel bernama `view_id` yang berfungsi menampung ID teknis dari tampilan yang akan dimunculkan hanya akan mendapatkan nilainya di dalam blok kondisional `if-elif` jika `order_type` memiliki nilai seperti *'rfq'*, *'draft_po'*, atau *'confirm_po'* . Ketika pengguna tidak memilih apa pun, `order_type` kosong dan tidak memenuhi salah satu kondisi pada struktur `if-else` seperti ditunjukkan pada kode 3.4.

```
1     @api.multi
2     def make_purchase_order(self):
3     ...
4         if self.order_type == 'rfq':
5             view_id = self.env.ref('kg_purchase.
    request_for_quotation_form')
6             context = {'search_default_todo': 1, 'show_purchase':
    False, 'default_is_rfq': True, 'show_rfq': True}
7         elif self.order_type == 'confirm_po':
8             view_id = self.env.ref('purchase.purchase_order_form')
9             context = {'search_default_todo': 1, 'show_purchase':
    True, 'default_is_po': 1, 'confirm_po': True}
10        elif self.order_type == 'draft_po':
11            view_id = self.env.ref('purchase.purchase_order_form')
12            context = {'search_default_todo': 1, 'show_purchase':
    True, 'default_is_po': 1}
13    ...
```

Kode 3.4: Potongan kode fungsi *make purchase order*

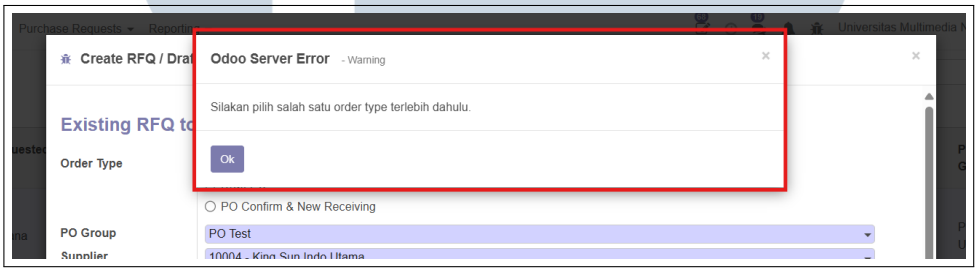
Solusi untuk mengatasi permasalahan ini adalah dengan menambahkan kode seperti pada Kode 3.5 sebelum baris kode pada Kode 3.4. Tambahan ini berfungsi untuk memastikan bahwa ketika fungsi dijalankan dan nilai dari `order_type` tidak termasuk salah satu dari nilai berikut: `rfq`, `confirm_po`, atau `draft_po`, maka

eksekusi fungsi akan dihentikan dan pesan *error* yang lebih mudah dipahami oleh pengguna akan ditampilkan.

```
1     @api.multi
2     def make_purchase_order(self):
3     ...
4         if self.order_type not in ['rfq', 'confirm_po', 'draft_po'
      ]:
5             raise UserError(_("Silakan pilih salah satu order type
      terlebih dahulu."))
6     ...
```

Kode 3.5: Potongan perbaikan kode untuk menampilkan pesan error

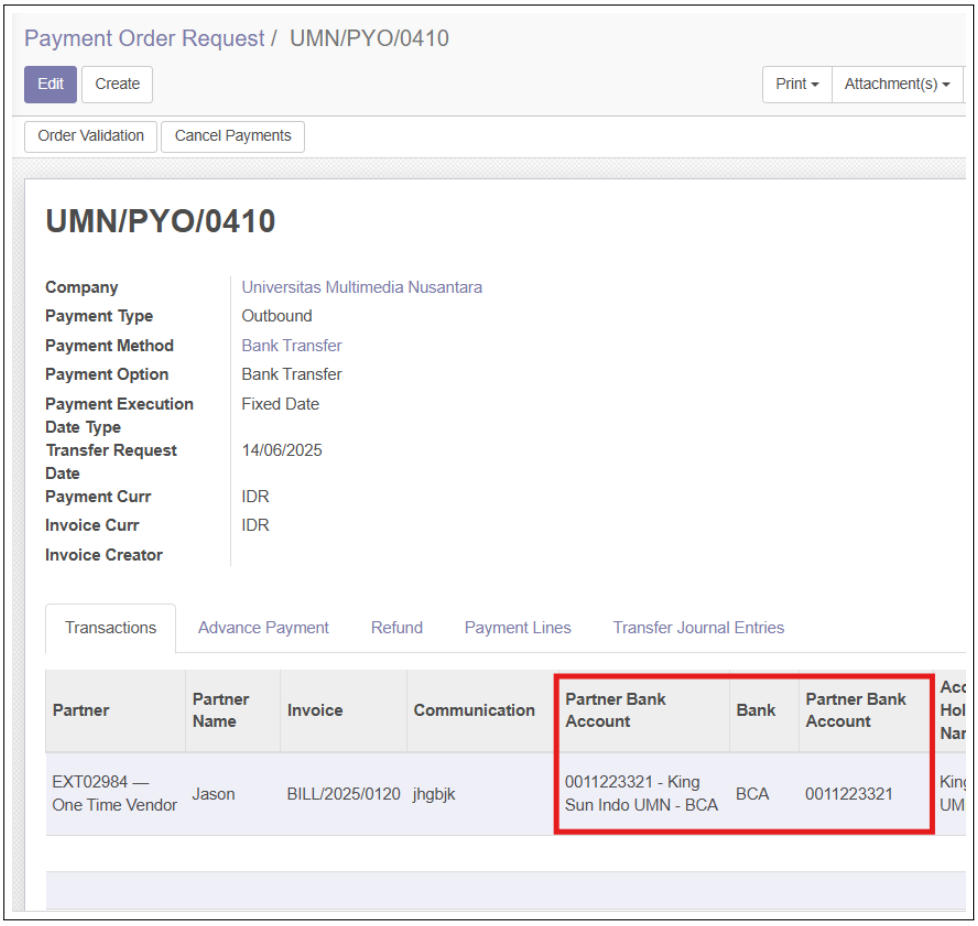
Hasil dari penambahan kode yang sudah dilakukan untuk memenuhi kebutuhan pengguna dapat dilihat pada gambar 3.10.



Gambar 3.7. Tampilan *UserError* yang lebih deskriptif untuk memberitahu pengguna

A.3 Fixing Edit Bank Account in Payment Order Request

Kebutuhan pada *backlog* ini terletak pada *child menu* dari *Payment*, yaitu *Payment Order Request* seperti pada Gambar 3.8. Menu ini berada di bawah menu utama *Invoicing*. *Record* pada *Payment Order Request* berisi data yang berasal dari *Vendor Bills*, ketika *Vendor Bills* tersebut memiliki *installment*, pengguna dapat mengganti akun *partner bank* pada model `account.payment.order`. Pada menu *Payment Order Request*, ditemukan masalah di mana informasi rincian bank pada baris transaksi tidak diperbarui secara otomatis. Ketika pengguna mengubah rekening bank tujuan pada *field* `Partner Bank Account`, kolom `Bank`, `Bank Account`, dan `Name` di sebelahnya tetap menampilkan data lama. Hal ini dapat menyebabkan kebingungan dan potensi kesalahan pembayaran. Untuk memenuhi kebutuhan pengguna, perubahan pada model `account.payment.order` yang berada di dalam modul `kg_bank` dilakukan, sebagaimana ditunjukkan pada Kode 3.6.



Gambar 3.8. Tampilan *Payment Order Request* pada menu *Invoicing*

```python
@api.onchange('partner_bank_id')
def onchange_partner_bank_id(self):
    for rec in self:
        if rec.partner_bank_id:
            rec.bank_id = rec.partner_bank_id.bank_id.id
            rec.bank_no_ot = rec.partner_bank_id.acc_number
            rec.bank_holder_ot = rec.partner_bank_id.
    acc_holder_name
```

Kode 3.6: Kode Python untuk Pembaruan Tampilan *Bank*, *Bank Account*, dan *Name* pada `account.payment.order`

Solusi untuk masalah ini adalah dengan memanfaatkan *decorator* bawaan Odoo, yaitu `@api.onchange`. *Decorator* ini digunakan untuk ”mengawasi” perubahan pada suatu *field* seperti pada kode 3.6. Dalam kasus ini, fungsi `onchange_partner_bank_id` dirancang untuk dieksekusi secara otomatis setiap kali pengguna mengubah nilai pada *field* `partner_bank_id`. *Field*

partner_bank_id sendiri merupakan relasi Many2one ke model data rekening bank milik partner (res.partner.bank). Ketika fungsi tersebut terpicu, alur datanya adalah sebagai berikut: sistem mengambil *record* rekening bank yang baru dipilih oleh pengguna. Kemudian, informasi spesifik seperti ID bank (bank_id), nomor rekening (acc_number), dan nama pemilik rekening (acc_holder_name) ditarik dari *record* tersebut. Data ini kemudian secara otomatis diisikan ke *field-field* yang ada di baris transaksi saat ini, yaitu bank_id, bank_no_ot, dan bank_holder_ot. Dengan perubahan tersebut, tampilan pada antarmuka pengguna akan selalu sinkron dengan data yang dipilih oleh pengguna.

A.4 Add Constanta Cost TMS

Backlog ini merupakan salah satu bagian dari proyek besar pada modul atau menu *Transportation*. Kebutuhan pengguna pada *backlog* ini adalah menambahkan *child menu* bernama *Constanta Cost* pada submenu *Master Data*, yang berfungsi sebagai *master data* untuk beberapa informasi biaya terkait ekspedisi di KGXpress milik Kompas Gramedia. Contoh informasinya meliputi Biaya Armada, Biaya Driver, Biaya BBM, dan biaya lainnya. Tujuannya adalah untuk memusatkan semua komponen biaya tetap yang berkaitan dengan ekspedisi, seperti biaya sewa armada, biaya BBM per jenis kendaraan, dan biaya parkir, yang sebelumnya tidak terstandar. Untuk memenuhi kebutuhan pada *backlog* ini, beberapa file baru telah dibuat pada folder kg_tms, termasuk file cost.py, seperti yang ditunjukkan pada kode 3.7.

```
from odoo import fields, models, api
from odoo.exceptions import ValidationError
class KGCost(models.Model):
    _name = 'tms.cost'
    name = fields.Char(string='Name')
    constanta_armada_ids = fields.One2many('tms.cost.constanta', '
    cost_armada_id', string="Biaya Armada")
    driver_fee = fields.Float(string='Biaya Driver (Per-Hari)')
    code = fields.Char(string='Code')
    location_ids = fields.One2many('tms.location', 'cost_id',
    string='Location', domain=[('is_hub', '=', True)])
    constanta_ratio_ids = fields.One2many('tms.cost.constanta', '
    cost_ratio_id', string="Ratio Armada")
    constanta_fuel_ids = fields.One2many('tms.cost.constanta', '
    cost_fuel_id', string="Biaya BBM")
    constanta_parking_ids = fields.One2many('tms.cost.constanta',
    'cost_parking_id', string="Biaya Parkir")
```

```
    traffic = fields.Float(string='Traffic Percentage')
    food_driver_cost = fields.Float(string='Uang Makan Driver')
    dlk_driver_cost = fields.Float(string='Uang DLK Driver')

...

class KGCostConstanta(models.Model):
    _name = 'tms.cost.constanta'
    cost_armada_id = fields.Many2one('tms.cost')
    cost_ratio_id = fields.Many2one('tms.cost')
    cost_fuel_id = fields.Many2one('tms.cost')
    cost_parking_id = fields.Many2one('tms.cost')
    amount = fields.Float(string='Amount (Per-Hari)')
    armada_type = fields.Many2one('tms.vehicle.type', string='
    Jenis Armada')
    ratio = fields.Float(string='Ratio')
    fuel_type = fields.Many2one('fleet.fuel.type', string='Jenis
    BBM')
    @api.constrains('amount', 'ratio', 'cost_armada_id', '
    cost_ratio_id', 'cost_fuel_id', 'cost_parking_id')

...
```

Kode 3.7: Kode script Python untuk menu Constanta Cost

Untuk mengimplementasikannya, arsitektur data dirancang menggunakan dua model yang saling berhubungan. Model pertama, `tms.cost`, berfungsi sebagai ”dokumen utama” yang menampung informasi umum (misalnya, ”Biaya Jabodetabek”). Model kedua, `tms.cost.constanta`, berfungsi sebagai ”baris rincian” untuk setiap komponen biaya. Struktur ini menggunakan relasi database di mana satu record di `tms.cost` dapat memiliki banyak rincian di `tms.cost.constanta` melalui field One2many seperti `constanta_armada_ids` (Biaya Armada), `constanta_ratio_ids` (Rasio Armada), `constanta_fuel_ids` (Biaya BBM), dan `constanta_parking_ids` (Biaya Parkir). Hal ini diatur secara otomatis oleh ORM *(Object Relational Mapping)* milik bawaan Odoo melalui field One2many yang sudah didefinisikan.

Pada model `tms.cost.constanta`, field seperti `cost_armada_id`, `cost_ratio_id`, `cost_fuel_id`, dan `cost_parking_id` menggunakan relasi `Many2one` yang mengarah ke model `tms.cost`, yang menunjukkan bahwa setiap baris data di `tms.cost.constanta` hanya terkait dengan satu record induk di `tms.cost`. Relasi dua arah ini memungkinkan Odoo untuk secara otomatis

menampilkan data terkait dalam tampilan formulir dan mempercepat pengelolaan data master biaya transportasi secara terstruktur.

Kebutuhan pengguna tidak hanya berupa menu *Master Data*, tetapi juga memerlukan *role* akses bernama *Transportation Master Data* yang dapat dimiliki oleh pengguna, di antaranya *User* dan *Pricing*. Keduanya memiliki hak akses yang berbeda. Pada *role User*, pengguna dapat melakukan operasi *CRUD*, sedangkan pada *role Pricing*, pengguna tidak memiliki akses *Delete*. Untuk mendefinisikan aturan akses pada model `tms.cost`, ditambahkan kode pada `ir_model_access.csv` seperti pada kode 3.8.

```
read_access_cost,kg_tms.tms_cost,kg_tms.model_tms_cost,base.
    group_user,1,0,0,0
read_access_cost_constanta,kg_tms.tms_cost_constanta,kg_tms.
    model_tms_cost_constanta,base.group_user,1,0,0,0
pricing_access_cost_constanta,kg_tms.tms_cost_constanta,kg_tms.
    model_tms_cost_constanta,group_tms_master_data_pricing,1,1,1,0
pricing_access_cost,kg_tms.tms_cost,kg_tms.model_tms_cost,
    group_tms_master_data_pricing,1,1,1,0
user_access_cost,kg_tms.tms_cost,kg_tms.model_tms_cost,
    group_tms_master_data_user,1,1,1,1
user_access_cost_constanta,kg_tms.tms_cost_constanta,kg_tms.
    model_tms_cost_constanta,group_tms_master_data_user,1,1,1,1
```

Kode 3.8: Kode CSV untuk *role akses User* dan *Pricing* model *Cost Constanta*

Kode CSV pada Kode 3.8 digunakan untuk mendefinisikan hak akses (akses kontrol) terhadap dua model, yaitu `tms.cost` dan `tms.cost.constanta`. Setiap baris mewakili satu aturan akses untuk grup pengguna tertentu, seperti *User*, *Pricing*, atau *Base User*. Angka-angka pada kolom akhir masing-masing baris menunjukkan izin terhadap model tersebut, yaitu **baca (1)**, **tulis (1)**, **buat (1)**, dan **hapus (1)**. Misalnya, grup `group_tms_master_data_user` diberikan akses penuh (1,1,1,1), sedangkan grup `base.group_user` hanya memiliki akses baca (1,0,0,0).

Dalam proses pengerjaan *backlog* ini, terdapat *feedback* dari *Solution Analyst* yang menyatakan adanya permintaan untuk menambahkan pengamanan pada setiap *line* di *field* biaya yang ada, agar tidak terjadi duplikasi data dengan *Jenis Armada* yang sama. Hal ini dapat diterapkan dengan kode 3.9.

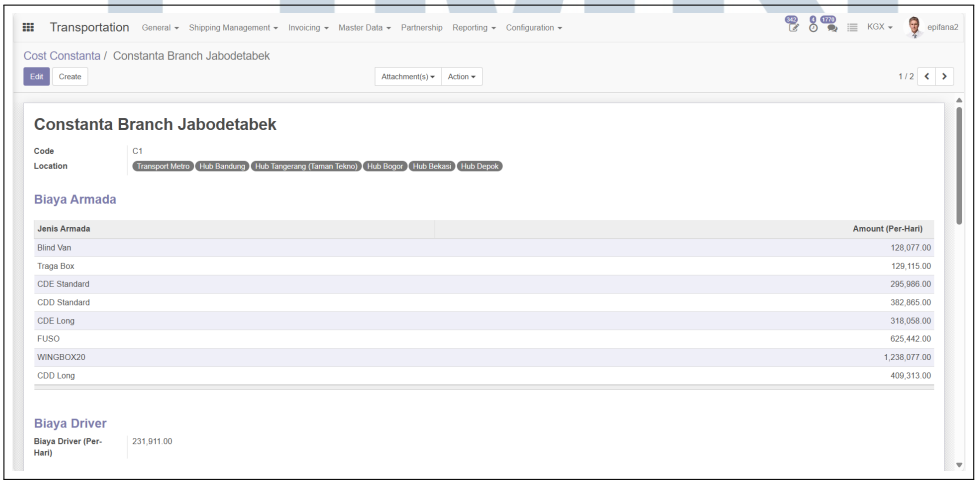
```
    @api.constrains('constanta_fuel_ids')
    def _check_unique_armada_type_fuel(self):
        for record in self:
            armada_type_counts = {}
            for line in record.constanta_fuel_ids:
```

```
                if line.armada_type:
                    armada_type_id = line.armada_type.id
                    armada_type_counts[armada_type_id] = armada_type_counts.get(armada_type_id, 0) + 1
            duplicates = [k for k, v in armada_type_counts.items() if v > 1]
            if duplicates:
                duplicate_names = ', '.join(self.env['tms.vehicle.type'].browse(duplicates).mapped('name'))
                raise ValidationError(f"The following Armada Types are duplicated in Fuel Fee: {duplicate_names}")
```

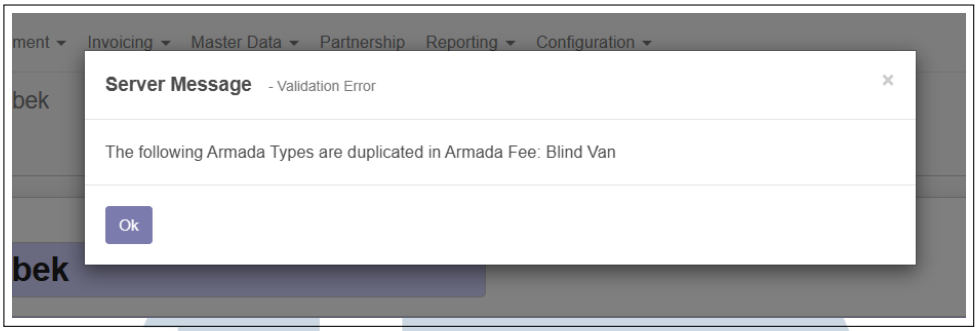
Kode 3.9: Potongan Kode Python untuk Validasi Unik *Jenis Armada* pada data Biaya BBM

Potongan kode di atas merupakan sebuah fungsi validasi yang dijalankan secara otomatis oleh Odoo saat data disimpan. Fungsi ini menggunakan dekorator `@api.constrains` untuk menetapkan bahwa pengecekan hanya akan dilakukan ketika terjadi perubahan pada `constanta_fuel_ids`, yaitu relasi terhadap baris-baris data biaya BBM. Di dalam fungsi, dilakukan perulangan untuk setiap baris `line` dan dicatat jumlah kemunculan setiap *Jenis Armada* berdasarkan ID-nya. Jika ditemukan ada ID yang muncul lebih dari satu kali (artinya ada duplikasi), maka sistem akan menampilkan *ValidationError* yang berisi nama-nama *Jenis Armada* yang duplikat.

Hasil tampilan layout dari menu *Constanta Cost* dan pengamanan terhadap duplikasi data untuk *Jenis Armada* yang sama pada masing-masing tabel biaya dapat dilihat pada Gambar 3.9 dan Gambar 3.10.



Gambar 3.9. Tampilan Layout Menu *Constanta Cost*



Gambar 3.10. Tampilan *Error* terhadap duplikasi data *Jenis Armada*

A.5 Add Master Cost TMS

Backlog ini merupakan kelanjutan dari proyek besar pada *backlog Constanta Cost*. Kebutuhan yang harus dipenuhi adalah penambahan satu menu bernama *Cost Route* pada submenu *Master Data*. Fungsi dari menu ini adalah untuk mengelola data biaya rute ekspedisi secara terperinci, termasuk pengaturan biaya berdasarkan asal (*Origin*) dan tujuan (*Destination*), serta integrasi dengan jenis kendaraan (*Vehicle Type*). Menu ini memungkinkan pengaturan biaya tetap dan variabel per rute, seperti jarak (*Jarak*), waktu tempuh (*Lead Time*), dan biaya tambahan seperti tol (*Biaya Toll*) atau feri (*Biaya Ferry*), dengan tujuan mengoptimalkan perhitungan biaya ekspedisi. Beberapa berkas baru juga ditambahkan ke dalam folder `kg_tms`, salah satunya adalah `cost_route.py`, sebagaimana ditunjukkan pada Kode 3.10.

```
from odoo import fields, models, api


class KGMasterCost(models.Model):
    _name = 'tms.cost.route'
    _sql_constraints = [
        ('unique_origin_destination_vehicle',
        'UNIQUE(origin_id, destination_id, vehicle_type)',
        'Combination of Origin, Destination, and Vehicle Type must be unique!'),
    ]

    name = fields.Char(string='Name', compute='_compute_name', store=True)
    origin_id = fields.Many2one('tms.zone', string='Origin',
        domain=[('zone_type', '=', 'pricing'), ('is_trucking', '=', True), ('active', '=', True)])
```

```
15      destination_id = fields.Many2one('tms.zone', string='
    Destination',
16          domain=[('zone_type', '=', 'pricing'), ('is_dest', '=',
    True), ('is_trucking', '=', True), ('active', '=', True)])
17      vehicle_type = fields.Many2one('tms.vehicle.type', string='
    Vehicle Type')
18      lead_item = fields.Integer(string='Lead Time (Hari)')
19      range_item = fields.Integer(string='Jarak (Km)')
20      stand_by_item = fields.Integer(string='Jaga')
21      food_item = fields.Integer(string='Makan')
22      total_driver = fields.Integer(string='Jumlah Driver')
23      toll_fee = fields.Float(string='Biaya Tol')
24      ferry_fee = fields.Float(string='Biaya Ferry')
25      others_item = fields.Float(string='Kawalan Ferry')
26
27      @api.depends('origin_id', 'destination_id', 'vehicle_type')
28      def _compute_name(self):
29          for rec in self:
30              rec.name = f"{rec.origin_id.name} - {rec.
    destination_id.name} - {rec.vehicle_type.name}"
```

Kode 3.10: Kode Python untuk menu *Cost Route*

Kode di atas merupakan definisi model `tms.cost.route` yang digunakan untuk menyimpan dan mengelola data biaya rute berdasarkan kombinasi asal, tujuan, dan jenis kendaraan. Pada bagian awal, terdapat deklarasi `_sql_constraints` untuk memastikan bahwa kombinasi ketiganya bersifat unik di dalam basis data. Untuk memastikan keunikan kombinasi data, pada backlog ini digunakan `_sql_constraints`. Berbeda dengan validasi `@api.constrains` pada backlog sebelumnya yang berjalan di level aplikasi Odoo, `_sql_constraints` membuat sebuah batasan unik langsung di level database. Ini memberikan lapisan pengamanan data yang lebih kuat dan mendasar, karena integritas data tetap terjaga bahkan jika ada upaya memasukkan data dari luar antarmuka standar Odoo.

Field `name` dihasilkan secara otomatis melalui fungsi `_compute_name` yaitu sebuah fungsi dengan memanfaatkan decorator `@api.depends`, nilai dari field name akan dibuat secara otomatis oleh sistem dengan menggabungkan nama dari ketiga field pemicunya (berdasarkan nilai `origin_id` dan `destination_id`, sebuah field Many2one yang fungsinya menghubungkan dengan record pada model `tms.zone`, dan juga `vehicle_type` yang merupakan sebuah field Many2one yang fungsinya menghubungkan dengan record pada model `tms.vehicle.type`. Alur ini memastikan penamaan setiap record menjadi seragam dan informatif tanpa perlu

input manual, sehingga meningkatkan konsistensi data.

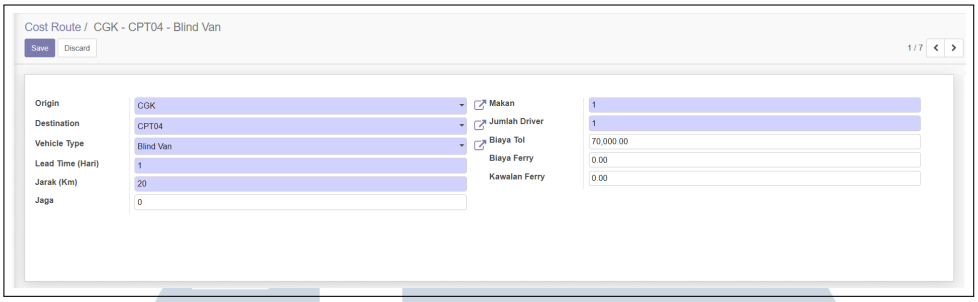
Selain itu, terdapat beberapa field lainnya yang merepresentasikan komponen biaya, seperti `lead_item` untuk waktu tempuh, `toll_fee` untuk biaya tol, serta `others_item` untuk biaya tambahan lain seperti kawalan feri. Selain itu, terdapat beberapa field lain yang merepresentasikan komponen biaya, seperti `lead_item` untuk waktu tempuh, `range_item` untuk jarak, `toll_fee` untuk biaya tol, dan `others_item` untuk biaya tambahan lainnya seperti kawalan feri.

Seperti pada *backlog* sebelumnya, kebutuhan pengguna tidak hanya berupa menu *Cost Route*, tetapi juga memerlukan *role* akses bernama *Transportation Master Data* yang dapat dimiliki oleh pengguna, di antaranya *User* dan *Pricing*. Keduanya memiliki hak akses yang berbeda. Pada *role User*, pengguna dapat melakukan operasi *CRUD*, sedangkan pada *role Pricing*, pengguna tidak memiliki akses *Delete*. Untuk mendefinisikan aturan akses pada model `tms.cost.route`, ditambahkan kode pada `ir_model_access.csv` seperti pada kode 3.11.

```
1 read_access_cost_route,kg_tms.tms_cost_route,kg_tms.
    model_tms_cost_route,base.group_user,1,0,0,0
2 pricing_access_cost_route,kg_tms.tms_cost_route,kg_tms.
    model_tms_cost_route,group_tms_master_data_pricing,1,1,1,0
3 user_access_cost_route,kg_tms.tms_cost_route,kg_tms.
    model_tms_cost_route,group_tms_master_data_user,1,1,1,1
```

Kode 3.11: Kode CSV untuk *role akses User* dan *Pricing* model *Cost Route*

Kode CSV pada Kode 3.11 digunakan untuk mendefinisikan aturan akses terhadap model `tms.cost.route` yang berkaitan dengan menu *Cost Route*. Setiap baris mewakili akses untuk grup pengguna tertentu, seperti *User*, *Pricing*, dan *Base User*. Kolom-kolom angka di akhir masing-masing baris menunjukkan izin dalam bentuk urutan: **baca**, **tulis**, **buat**, dan **hapus**. Sebagai contoh, grup `group_tms_master_data_user` diberikan akses penuh (1,1,1,1), sedangkan grup dasar `base.group_user` hanya diberikan akses baca (1,0,0,0). Pengaturan ini bertujuan untuk memastikan bahwa hanya pengguna dengan peran tertentu yang dapat mengelola data biaya rute secara penuh. Hasil tampilan Layout dari menu Cost Route dilihat pada Gambar 3.11.



Gambar 3.11. Tampilan menu *Cost Route*

3.4 Kendala dan Solusi yang Ditemukan

Selama periode magang, peserta magang menghadapi berbagai kendala dalam menjalankan tugas dan menyesuaikan diri dengan lingkungan kerja serta teknologi yang digunakan. Kendala-kendala tersebut mencakup aspek teknis maupun non-teknis yang berdampak pada proses pengerjaan tugas. Untuk mengatasi kendala tersebut, peserta magang berupaya mencari solusi melalui pembelajaran mandiri maupun bimbingan dari rekan kerja dan atasan. Penjelasan mengenai kendala dan solusi yang ditemukan disajikan dalam subbagian berikut.

3.4.1 Kendala yang Dihadapi

Selama pelaksanaan kegiatan magang, peserta magang menghadapi beberapa kendala yang berkaitan dengan penyesuaian terhadap lingkungan kerja, pemahaman sistem, serta penguasaan teknologi yang digunakan. Adapun kendala-kendala tersebut dijelaskan sebagai berikut:

1. Belum memiliki pengalaman dalam menggunakan atau mempelajari *framework* Odoo, karena selama masa perkuliahan tidak terdapat materi yang membahas *framework* tersebut.
2. Mengalami kesulitan dalam memahami *backlog* yang diberikan, karena belum memahami secara menyeluruh proses bisnis dari sistem ERP milik Kompas Gramedia.
3. Sering terjadi miskomunikasi dengan *Solution Analyst* terkait *backlog* yang dikerjakan, sehingga diperlukan komunikasi lanjutan untuk memperjelas informasi yang dibutuhkan.

4. Kurangnya pemahaman tentang bahasa pemrograman *Python*, sehingga dalam pengerjaan *backlog* diperlukan pembelajaran tambahan mengenai sintaks *Python* melalui dokumentasi resmi maupun dokumentasi Odoo.

3.4.2 Solusi yang Ditemukan

Untuk mengatasi kendala-kendala yang dihadapi selama masa magang, peserta magang berinisiatif mencari solusi secara mandiri maupun melalui bimbingan dari tim pengembang dan analis di Kompas Gramedia. Beberapa solusi yang dilakukan antara lain:

1. Melakukan pelatihan mandiri dengan mempelajari dokumentasi resmi Odoo serta mengikuti tutorial daring untuk memahami dasar-dasar *framework* Odoo. Selain itu, berkonsultasi dengan *senior developer* untuk mendapatkan panduan praktis dan solusi teknis.

2. Mempelajari dokumentasi *source code* dari sistem ERP Kompas Gramedia, serta berdiskusi langsung dengan tim *Solution Analyst* untuk memahami alur kerja yang relevan dengan *backlog* yang diberikan.

3. Komunikasi proaktif dengan *Solution Analyst* ditingkatkan, baik secara langsung maupun melalui *Microsoft Teams*. Tujuannya adalah untuk beralih dari sekadar mengonfirmasi detail *backlog* menjadi sesi diskusi untuk mengklarifikasi kebutuhan, mengusulkan perbaikan fungsional, dan memastikan keselarasan pemahaman guna meminimalkan risiko miskomunikasi.

4. Meningkatkan pemahaman mengenai bahasa pemrograman *Python* dengan mengikuti kursus daring, mempelajari dokumentasi resmi, serta melakukan latihan praktis melalui pembuatan program sederhana yang relevan dengan penerapan di Odoo.