BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Corporate Solution Department merupakan salah satu divisi *Corporate IT & IS* Kompas Gramedia. Penempatan magang dilaksanakan pada divisi ini, di bawah supervisi Bapak Anwari Rahman selaku *Enterprise Tech Manager*. Selama periode magang, posisi yang dijalankan adalah sebagai *Junior Fullstack Software Engineer Intern* dengan bimbingan langsung dari Bapak Ritz Moondrian, *Senior Software Engineer*, yang secara berkala melaporkan perkembangan tugas kepada Bapak Anwari Rahman.

Rangkaian penugasan dimulai oleh Ibu Nikita Yuka Risatriana, *Senior Solution Analyst*, yang menetapkan daftar *backlog* pekerjaan. Setelah setiap item *backlog* diselesaikan, *pull request* diajukan dan proses *deployment* dilakukan untuk selanjutnya diperiksa serta disahkan oleh Ibu Nikita Yuka Risatriana.

3.2 Tugas yang Dilakukan

Selama masa magang di Kompas Gramedia, tanggung jawab utama adalah melakukan pengembangan dan kustomisasi modul pada sistem *Enterprise Resource Planning* (ERP) Odoo, khususnya pada modul *Performance Management Office* (PMO). Salah satu tugas utama yang dilaksanakan meliputi proses migrasi aplikasi *Performance Management Office* dari sistem sebelumnya yang dibangun menggunakan framework *.NET* dan basis data Microsoft SQL Server, ke dalam Odoo versi 18 yang lebih modern dan terintegrasi.

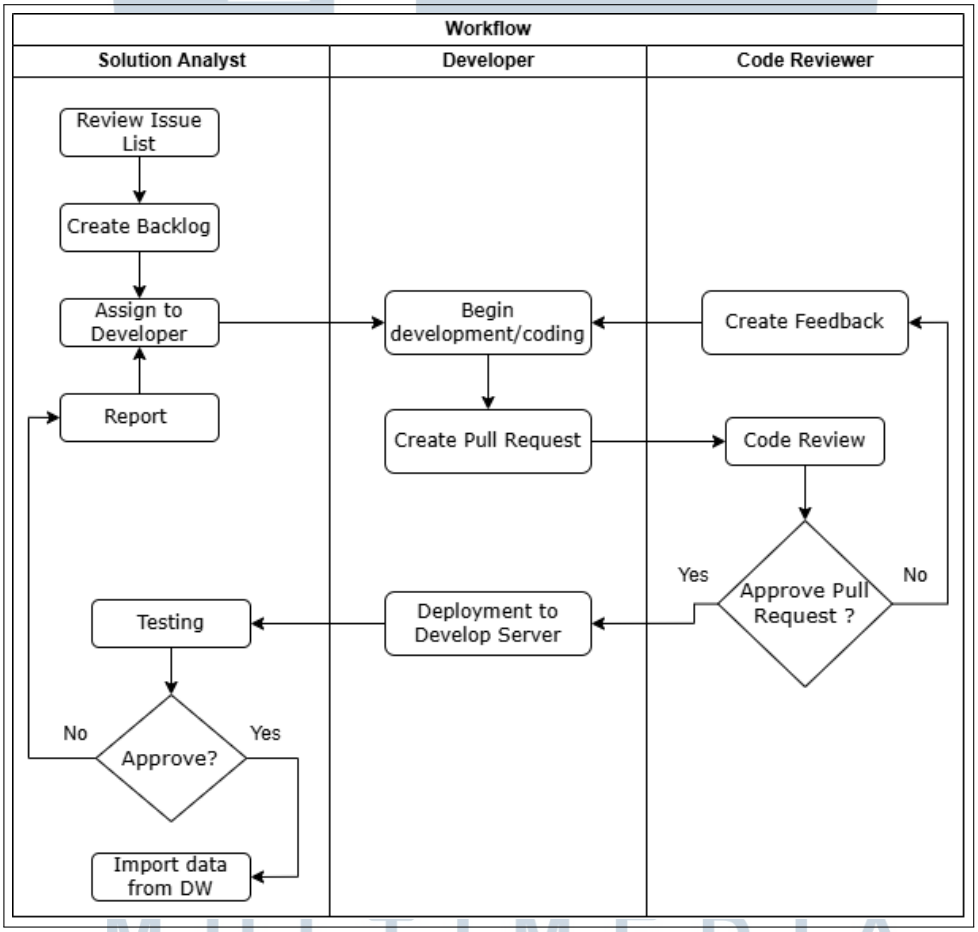
Dalam proses migrasi ini, tidak hanya dilakukan pemindahan data dan logika bisnis, tetapi juga pengembangan modul kustom secara langsung. Penguasaan dan modifikasi kode JavaScript pada Odoo 18 yang telah menggunakan bahasa pemrograman *OWL* (Odoo Web Library) turut dilakukan. Salah satu contoh konkret pekerjaan yang dijalankan adalah penerapan patch pada komponen *notebook* sesuai standar framework *OWL* Odoo terbaru.

Fokus utama pekerjaan terletak pada modifikasi dan pengembangan modul *Performance Management Office*. Dalam proses ini, sejumlah model *master data* telah dirancang dan diimplementasikan untuk mendukung proses bisnis di dalam modul tersebut. Selain itu, tiga menu utama pada modul, yaitu *scorecard tree*,

dashboard, dan *alignment* juga telah dirancang dan dikembangkan.

Menu *scorecard tree* merupakan salah satu fitur utama yang dimanfaatkan perusahaan untuk menyusun *scorecard* dan *Key Performance Indicator* (KPI) berdasarkan perspektif serta *strategic objective*. Melalui fitur ini, target KPI tahunan dapat diinput secara terstruktur dan sistematis, sehingga memudahkan proses pemantauan dan evaluasi kinerja.

Selama pelaksanaan magang di Kompas Gramedia, alur kerja pengembangan perangkat lunak diikuti secara terstruktur dan kolaboratif. Proses penugasan hingga *deployment* dilaksanakan melalui beberapa tahapan sebagai berikut:



Gambar 3.1. Alur kerja pengembangan modul *Performance Management Office*

Gambar 3.1 menggambarkan alur kerja pengembangan modul *Performance Management Office*. Berikut adalah penjelasan dari ilustrasi tersebut:

1. **Penugasan Backlog**

   Setiap tugas pengembangan atau *backlog* diberikan oleh analis, yaitu Ibu

Nikita Yuka Risatriana atau Bapak Mathias. Sebelum didistribusikan kepada anggota tim magang, *backlog* tersebut harus melalui proses persetujuan terlebih dahulu. *Backlog* umumnya memuat judul tugas, deskripsi pekerjaan, referensi kode (seperti *stored procedure* untuk proses bisnis), dan kadang dilengkapi gambar atau diagram pendukung.

2. **Eksplorasi Aplikasi Web PMO Lama**
   Untuk memahami kebutuhan dan alur bisnis dengan lebih baik, eksplorasi juga dilakukan terhadap aplikasi web PMO generasi sebelumnya. Dengan melihat fitur, tampilan, serta alur proses pada sistem lama, pengembangan modul baru dapat disesuaikan agar memenuhi ekspektasi pengguna dan tetap konsisten terhadap proses bisnis yang telah berjalan.

3. **Diskusi dan Klarifikasi**
   Jika terdapat hal yang kurang jelas terkait deskripsi *backlog*, diskusi dapat dilakukan secara langsung dengan analis melalui Microsoft Teams ataupun tatap muka. Penugasan *backlog* juga dapat dikonfirmasi atau diarahkan oleh senior (Bapak Ritz Moondrian) sesuai kebutuhan tim.

4. **Manajemen Tugas dan Repositori**
   Seluruh *backlog* dikelola menggunakan Azure DevOps pada menu Boards/Backlogs. Selain itu, repositori juga terintegrasi dengan Azure DevOps, sehingga proses *commit*, *pull request*, hingga *deployment* dapat dilakukan dalam satu platform.

5. **Pelaksanaan dan Pembaruan Status Backlog**
   Setelah mendapatkan tugas, status *backlog* diubah dari “approved” menjadi “committed (doing)”. Setelah pekerjaan selesai, status diubah menjadi “committed (done)”, kemudian *pull request* diajukan ke *branch performance management office* untuk melalui proses *code review* oleh senior atau manajer.

6. **Code Review dan Deployment**
   Pull request yang diajukan akan di-review oleh Senior Software Engineer (Bapak Ritz Moondrian) atau Manajer (Bapak Anwari Rahman). Jika terdapat catatan atau revisi, perbaikan harus dilakukan dan diajukan kembali hingga mendapatkan persetujuan. Setelah *pull request* disetujui, *deployment* dilakukan melalui Azure DevOps dan status *backlog* diubah menjadi “deployed”.

7. **Pengujian oleh Analis dan Perbaikan Bug**
   Pengujian akan dilakukan secara langsung oleh analis pada server pengembangan. Jika ditemukan bug atau kesalahan, status *backlog* diubah menjadi "bug" dan dikembalikan untuk diperbaiki. Proses ini diulang hingga *backlog* dinyatakan selesai tanpa permasalahan.

8. **Rapat Harian dan Sprint**
   Setiap hari diadakan *standup meeting* untuk melaporkan progres harian dan rencana kerja. Selain itu, setiap dua minggu sekali diadakan rapat *sprint corporate solution* untuk meninjau pekerjaan dua minggu terakhir dan menetapkan target dua minggu ke depan.

9. **Komunikasi Tim**
   Komunikasi dan koordinasi antaranggota tim dilakukan secara intensif melalui Microsoft Teams, baik melalui *group chat* maupun diskusi personal.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan pada Tabel 3.1

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

Minggu ke-	Pekerjaan yang dilakukan
1	• Instalasi *Odoo* 18. • Set-up *environment* (Python, *virtualenv*, IDE). • Belajar dasar bahasa Python.
2	• Belajar *Odoo 18* via YouTube & Udemy. • Memahami *OOP* dan *clean code*. • Menelaah alur *backlog* → *pull request*.
3	• Memahami proses bisnis PMO & Scorecard Online. • Membuat *model* master-data awal modul PMO.
Lanjut pada halaman berikutnya	

Tabel 3.1 Pekerjaan yang dilakukan selama pelaksanaan kerja magang (lanjutan)

Minggu ke-	Pekerjaan yang dilakukan
4	• Merancang menu pertama modul PMO. • Eksplorasi *Odoo Web Library (OWL)* untuk UI.
5	• Mengembangkan menu *scorecard tree*. • Membangun relasi master-data & perspektif KPI.
6	• Implementasi menu *dashboard*. • Integrasi tampilan XML + OWL (*dynamic UI*).
7	• Patch *widget* vanilla JS $\rightarrow$ OWL. • Membuat *pull request* pertama untuk direviu.
8	• Perbaikan berdasarkan *code review*. • *Deployment* ke server development & uji analis.
9	• Debug fitur *show/hide notebook*. • Menambah transaksi target *baseline* & *end-year*.
10	• Form input target 12 bulan per KPI. • Fungsi *calculate weight* (PSQL *stored procedure*).
11	• Fungsi *compute end-year target*. • Optimasi pencarian *scorecard*.
	Lanjut pada halaman berikutnya

Tabel 3.1 Pekerjaan yang dilakukan selama pelaksanaan kerja magang (lanjutan)

Minggu ke-	Pekerjaan yang dilakukan
12	• Wizard "*Copy Scorecard*" & kunci status. • Pencegahan duplikasi *scorecard tree*.
13	• Pembuatan *Master Organization Period*. • Penyesuaian hak akses & responsivitas *gauge*.
14	• Menu "Destination Statement" & modifikasi UI. • Eksplorasi Chart.js; model *vertical alignment*.
15	• Wizard "Assign Alignment" (tipe alignment & measurement). • Pewarnaan baris perspektif; perbaikan *color picker*.
16	• Selesai wizard *Assign & Sync Alignment*. • Wizard "Manage Weights" dan *sprint* review KPI.
17	• Daftar warna di "Manage Weights". • Sinkronisasi target KPI (top-down, bottom-up, rata-rata, berbobot). • *Deploy vertical alignment* ke server dev.
18	• Finalisasi dokumentasi teknis modul PMS. • Pelatihan pengguna internal. • Refaktor *clean code*; *sprint* retrospeksi & serah terima.

3.3.1 Pembelajaran

Selama empat minggu pertama masa magang, proses pembelajaran mandiri dijalankan dengan fokus pada aspek teknis maupun non-teknis. Adapun pengalaman dan materi pembelajaran yang diperoleh selama periode ini adalah sebagai berikut:

A. **Pengenalan dan Tantangan pada Framework *Odoo* ERP**
Pengalaman pertama dalam menggunakan *Odoo* sebagai platform pengembangan menghadirkan tantangan tersendiri. Proses pembelajaran dilakukan secara mandiri, didukung oleh video pembelajaran dari YouTube serta modul daring Udemy yang direkomendasikan oleh supervisor. Diperlukan pemahaman mengenai struktur dasar modul, penggunaan bahasa *Python* untuk pengembangan *backend/model*, serta pemanfaatan sintaks *XML* khusus Odoo dalam pembuatan *view*. Kondisi ini menjadi pengalaman baru, terutama karena sebelumnya belum terdapat familiarisasi dengan format dan sintaks *Odoo*.

B. **Pengalaman Praktik *Pull Request* dan *Code Review***
Salah satu pengalaman yang paling berkesan adalah ketika pertama kali menjalani proses *pull request* dan *code review* dari senior. Proses *code review* di lingkungan Kompas Gramedia berlangsung secara ketat dan detail, menekankan pentingnya penerapan *clean code*, keterbacaan kode, serta efisiensi logika program. Pembelajaran meliputi perhatian terhadap penamaan kelas dan variabel yang konsisten, penghindaran kode redundan, serta memastikan agar setiap baris kode dapat dipahami oleh anggota tim lain.

Salah satu contoh penerapan *clean code* yang diwajibkan adalah mengikuti aturan PEP8 pada Python, seperti yang ditunjukkan pada Kode 3.1.

```
from odoo import models, fields

# Harus ada dua baris kosong sebelum deklarasi kelas

class PmoOrganizationGroup(models.Model):
    _name = 'pmo.organization.group'

    group_name = fields.Char()
    total_member = fields.Integer()
```

```
10
```

Kode 3.1: Contoh penulisan kode sesuai PEP8 untuk import dan penamaan variabel.

Beberapa aturan PEP8 yang digunakan adalah:

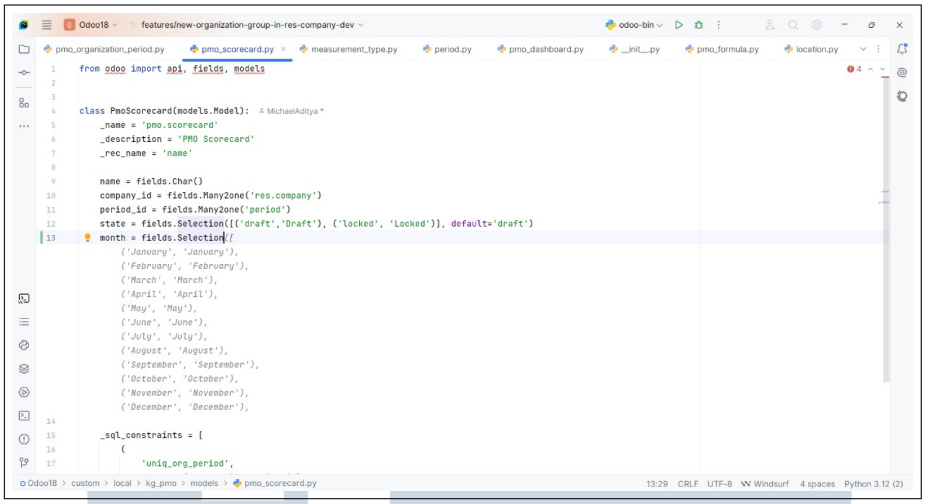
- Menambahkan **dua baris kosong** setelah blok `import` sebelum mendeklarasikan kelas baru, agar struktur kode lebih rapi dan mudah dibaca.
- Menggunakan gaya penamaan variabel *snake_case*, di mana setiap kata pada nama variabel dipisahkan oleh underscore (_), sehingga mudah dipahami. Contoh: `total_score`, `user_name`, atau `max_value`.

C. **Pemanfaatan Fitur pada *PyCharm* dan *Azure DevOps***

Berbagai fitur pendukung dari *PyCharm* dan *Azure DevOps* sangat membantu dalam proses pengembangan. Salah satu *plugins* yang digunakan di *PyCharm* adalah Codeium. Codeium merupakan ekstensi berbasis AI yang dapat memberikan rekomendasi otomatis dalam penulisan kode, seperti *autofill syntax*, saran *function*, serta penyempurnaan struktur kode sesuai konteks yang sedang dikerjakan. Penggunaan Codeium ini sangat mempercepat proses penulisan kode dan meminimalisir kesalahan penulisan *syntax*.

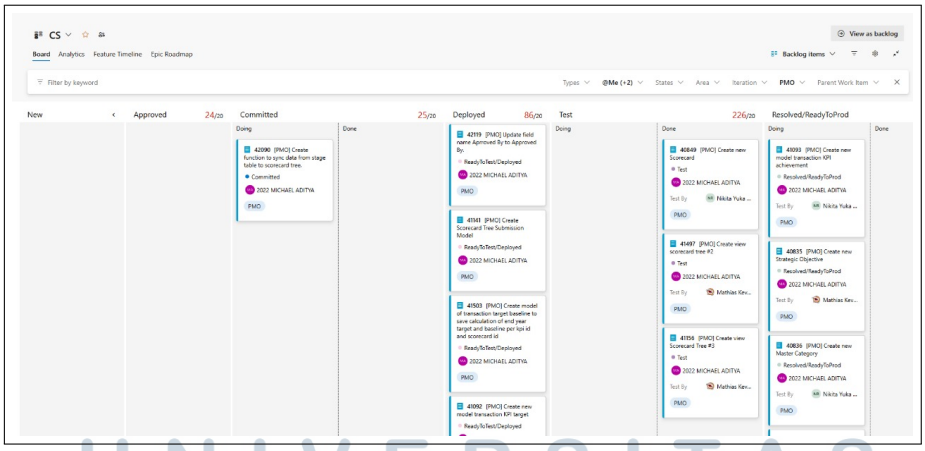
Meskipun demikian, setiap kode yang dihasilkan dari rekomendasi Codeium tetap dilakukan pengecekan ulang secara manual untuk memastikan logika dan kualitas kode sesuai standar *clean code* dan kebutuhan sistem yang sedang dikembangkan.

Contoh hasil penggunaan Codeium pada *PyCharm* ditunjukkan pada Gambar 3.2.



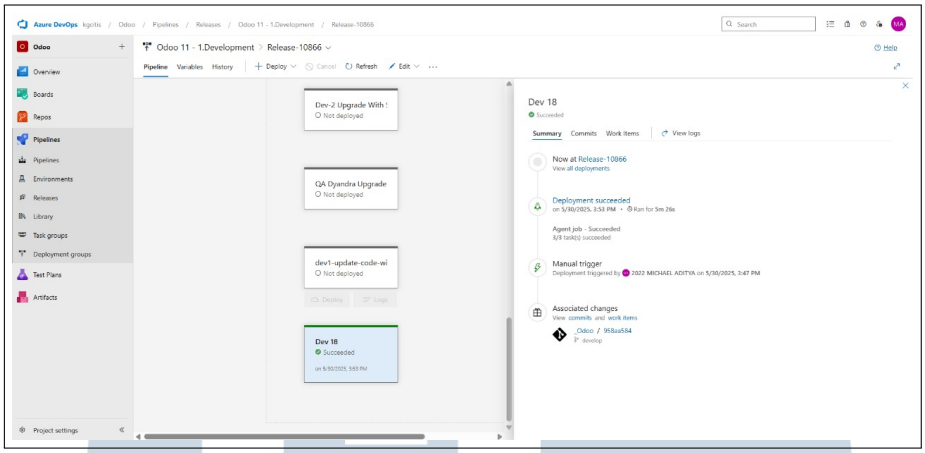
Gambar 3.2. Contoh penggunaan Codium di *PyCharm* untuk rekomendasi penulisan kode otomatis.

Pada *Azure DevOps*, seluruh *backlog* beserta deskripsi dan status pengerjaannya dapat dipantau secara mudah, sehingga alur kerja menjadi lebih terstruktur dan termonitor dengan baik. Contoh tampilan *Azure Boards* dapat dilihat pada Gambar 3.3.



Gambar 3.3. Tampilan *Azure Boards* pada *Azure DevOps* untuk manajemen backlog.

Selain itu, proses *deployment* ke server pengembangan dikelola melalui *Azure Pipeline*. Gambar 3.4 menampilkan salah satu contoh pipeline yang digunakan untuk mengotomasi proses *release* dan *deployment* modul ke lingkungan server development.



Gambar 3.4. Contoh pipeline pada *Azure DevOps* untuk proses *release* dan *deployment*.

Seluruh proses pembelajaran yang dijalani tidak hanya memperkaya kemampuan teknis, tetapi juga membentuk pola pikir kerja yang lebih disiplin, kolaboratif, serta membiasakan diri terhadap praktik pengembangan perangkat lunak yang profesional.

3.3.2 Pengerjaan Backlog

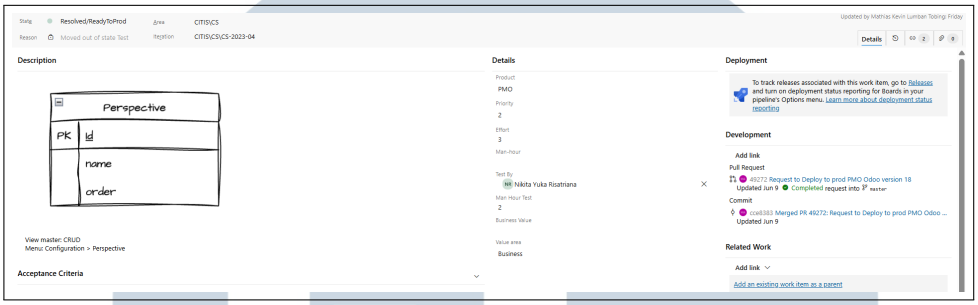
Selama masa magang, berbagai penugasan untuk mengerjakan backlog pada modul *KG_PMO* (*Kompas Gramedia Performance Management Office*) telah dilaksanakan. Pada bagian ini akan diuraikan backlog yang dikerjakan selama periode magang.

A Pembuatan Model dan View Master Data

Tahap awal pengembangan modul *KG_PMO* diawali dengan pembuatan model dan view untuk berbagai master data. Master data yang dikembangkan meliputi Perspective, Strategic Objective, *Key Performance Indicator* (KPI), Formula, Measurement Strategy Initiative, Location, Period, YTD Type, Measurement, dan Strategy Initiative.

Pengerjaan backlog pembuatan master data ini bertujuan agar setiap entitas dalam sistem dapat dikelola, dihubungkan, dan dimanfaatkan secara terstruktur oleh modul-modul lain. Proses pengembangan dilakukan secara bertahap, dimulai dari perancangan kebutuhan melalui backlog dan wireframe, implementasi kode model (Python), pembuatan tampilan view (XML), hingga pengujian hasil UI pada *Odoo ERP*.

Pengembangan master data *Perspective* diawali dengan pembuatan backlog pada Azure DevOps yang memuat wireframe serta kebutuhan utama sebagai acuan desain, seperti yang ditunjukkan pada Gambar 3.5.



Gambar 3.5. Contoh backlog dan wireframe pengembangan master data *Perspective* pada Azure DevOps.

Berdasarkan backlog tersebut, pengembangan model *Perspective* dilakukan menggunakan ORM Odoo, sebagaimana ditunjukkan pada Kode 3.2. Kode ini mendefinisikan struktur model *Perspective* beserta field yang dibutuhkan pada sistem.

```
class Perspective(models.Model):
    _name = 'pmo.perspective'
    _description = "Perspective"

    name = fields.Char()
    order = fields.Integer()
    initial_index = fields.Char()
    strategic_objective_ids = fields.One2many('strategic.objective', 'perspective_id', string="Strategic
      Objectives")
    color = fields.Char()
```

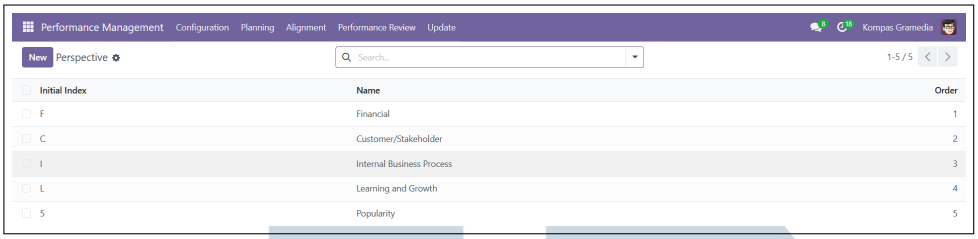
Kode 3.2: Model *Perspective* pada Odoo.

Untuk menampilkan data *Perspective* pada antarmuka Odoo, list view dibangun menggunakan kode XML sebagaimana terdapat pada Kode 3.3. Kode ini mengatur struktur tampilan data *Perspective* pada aplikasi Odoo ERP.

```
<record id="view_pmo_perspective_list" model="ir.ui.view">
    <field name="name">pmo.perspective.list</field>
    <field name="model">pmo.perspective</field>
    <field name="arch" type="xml">
        <list string="List Perspective">
            <field name="initial_index" string="Initial Index"/>
            <field name="name" string="Name"/>
            <field name="order" string="Order"/>
        </list>
    </field>
</record>
```

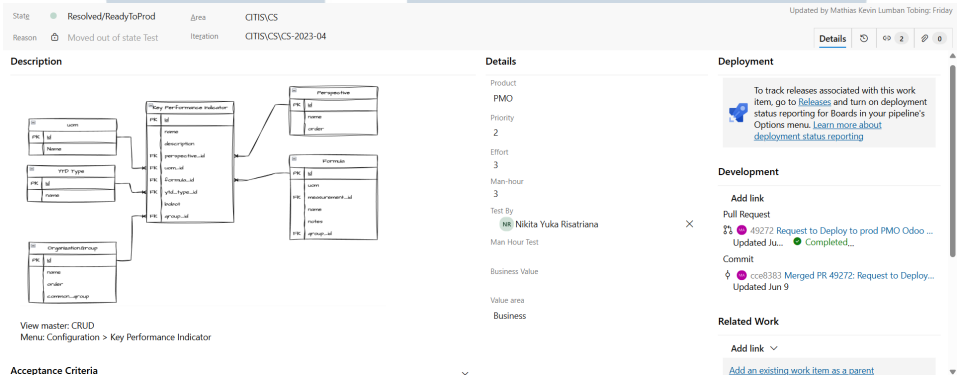
Kode 3.3: XML list view data *Perspective*.

Hasil implementasi list view untuk data *Perspective* pada Odoo ERP ditampilkan pada Gambar 3.6.



Gambar 3.6. Tampilan list view data *Perspective* pada Odoo ERP.

Pada pengembangan master data *Key Performance Indicator* (KPI), backlog yang digunakan juga mencakup wireframe relasi antar entitas, sebagaimana ditunjukkan pada Gambar 3.7.



Gambar 3.7. Backlog dan wireframe pengembangan master data *Key Performance Indicator* (KPI) dan relasi antar entitas.

Berdasarkan backlog tersebut, model *pmo.kpi* dikembangkan dengan berbagai field dan relasi sesuai kebutuhan sistem, sebagaimana ditunjukkan pada Kode 3.4.

```python
class PMOKPI(models.Model):
    _name = 'pmo.kpi'
    _description = "Key Performance Indicator"

    name = fields.Char()
    description = fields.Text()
    perspective_id = fields.Many2one("pmo.perspective")
    uom_id = fields.Many2one("uom.uom")
    formula_id = fields.Many2one("pmo.formula")
    ytd_type_id = fields.Many2one("pmo.ytd.type")
    bobot = fields.Integer()
    company_id = fields.Many2one("res.company")
    group_id = fields.Many2one("pmo.organization.group")
    scorecard_tree_id = fields.Many2one("pmo.scorecard.tree")
```

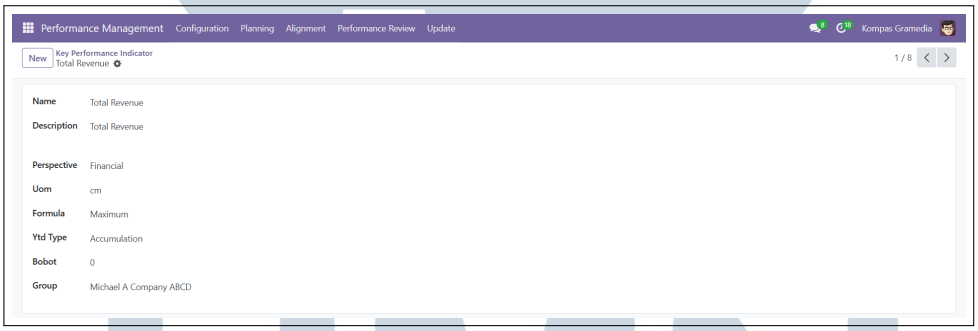
Kode 3.4: Model *Key Performance Indicator (KPI)* pada Odoo.

Form view untuk data *KPI* dibangun menggunakan kode XML pada Kode 3.5.

```xml
<record id="view_pmo_kpi_form" model="ir.ui.view">
    <field name="name">pmo.kpi.form</field>
    <field name="model">pmo.kpi</field>
    <field name="arch" type="xml">
        <form string="Key Performance Indicator">
            <sheet>
                <group>
                    <field name="name"/>
                    <field name="description"/>
                    <field name="perspective_id"/>
                    <field name="uom_id"/>
                    <field name="formula_id"/>
                    <field name="ytd_type_id"/>
                    <field name="bobot"/>
                    <field name="group_id"/>
                </group>
            </sheet>
        </form>
    </field>
</record>
```

Kode 3.5: XML untuk form view model *pmo.kpi*.

Visualisasi hasil implementasi form view untuk data *KPI* pada Odoo ERP disajikan pada Gambar 3.8.

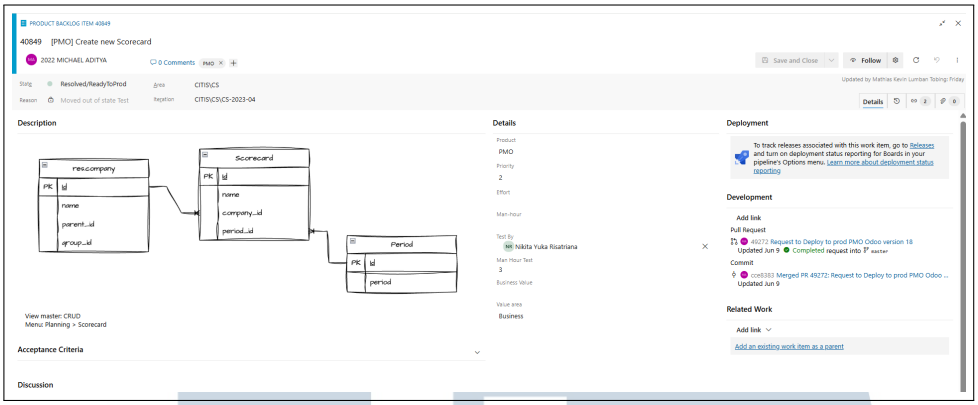


Gambar 3.8. Tampilan form view untuk data *Key Performance Indicator* (KPI) pada Odoo ERP.

Pengembangan model dan view untuk master data menjadi fondasi untuk proses pengelolaan data pada modul *KG_PMO* agar berjalan secara terstruktur.

B Pengembangan Scorecard

Pengembangan modul *KG_PMO* diawali dengan pembuatan fitur *scorecard*. *scorecard* merepresentasikan satu entitas organisasi dan periode tertentu. Struktur relasi antara *scorecard*, organisasi, dan periode digambarkan pada Gambar 3.9.



Gambar 3.9. Backlog dan wireframe pengembangan fitur *scorecard* (relasi organisasi dan periode).

Berdasarkan rancangan tersebut, implementasi kode model dan tampilan list view untuk *scorecard* dilakukan, sebagaimana ditunjukkan pada Kode 3.6 dan Kode 3.7

```python
class PmoScorecard(models.Model):
    _name = 'pmo.scorecard'
    _description = 'PMO Scorecard'
    _rec_name = 'name'

    name = fields.Char()
    company_id = fields.Many2one('res.company')
    period_id = fields.Many2one('period')
    state = fields.Selection([('draft','Draft'), ('locked', 'Locked')], default='draft')

    _sql_constraints = [
        (
            'uniq_org_period',
            'UNIQUE (company_id, period_id)',
            'Scorecard for the selected Company and Period already exists.',
        ),
    ]
```

Kode 3.6: Potongan kode model *Scorecard* pada Odoo.

Kode di atas merupakan definisi model *Key Performance Indicator* (KPI) pada Odoo. Setiap field merepresentasikan atribut utama KPI, antara lain:

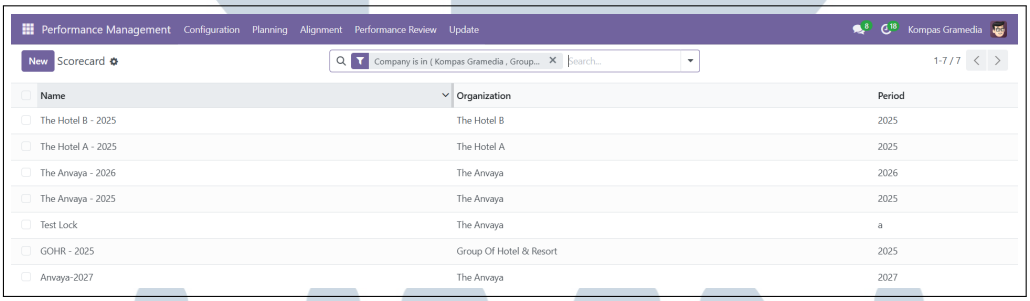
`name`: Nama KPI.
`description`: Deskripsi KPI.
`perspective_id`: Relasi ke model *Perspective*.
`uom_id`: Satuan pengukuran (unit of measure).
`formula_id`: Formula perhitungan KPI.
`ytd_type_id`: Tipe perhitungan year-to-date.
`bobot`: Nilai bobot KPI.
`company_id`: Perusahaan terkait.
`group_id`: Kelompok organisasi.

`scorecard_tree_id`: Relasi ke data *scorecard tree*. Tampilan list view untuk data *scorecard* dibangun menggunakan XML seperti pada Kode 3.7.

```xml
<record id="view_pmo_scorecard_list" model="ir.ui.view">
    <field name="name">pmo.scorecard.list</field>
    <field name="model">pmo.scorecard</field>
    <field name="arch" type="xml">
        <list string="List Scorecard">
            <field name="name" string="Name"/>
            <field name="company_id" string="Organization"/>
            <field name="period_id" string="Period"/>
        </list>
    </field>
</record>
```

Kode 3.7: XML list view untuk model *pmo.scorecard*.

Kode di atas mendefinisikan tampilan form view untuk data *KPI* di Odoo ERP. Setiap field yang terdapat pada form memudahkan pengguna untuk mengisi dan mengelola data *Key Performance Indicator* secara terstruktur sesuai kebutuhan bisnis. Hasil implementasi list view untuk data *scorecard* dapat dilihat pada Gambar 3.10.



Gambar 3.10. Tampilan list view data *Scorecard* pada Odoo ERP.

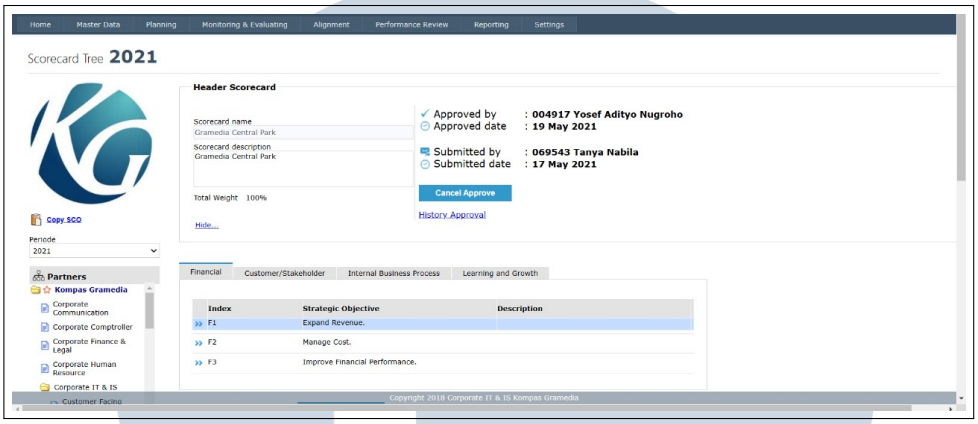
Pembuatan scorecard merupakan tahapan awal yang penting sebelum beralih ke proses perangkaian *scorecard tree*, yang akan mengelola indikator kinerja secara lebih mendalam dan terhubung antar entitas.

C Pengembangan Scorecard Tree

Setelah data scorecard diinisiasi, proses pengelolaan lanjutan dilakukan melalui menu *Scorecard Tree*. Menu ini menjadi fitur inti dalam modul *KG_PMO*, digunakan untuk menyusun dan mengelola *Key Performance Indicator* (KPI) berdasarkan *strategic objective*, organisasi, periode, dan perspektif bisnis.

Sebagai referensi, Gambar 3.11 memperlihatkan tampilan *scorecard tree* pada aplikasi PMO generasi sebelumnya. Visualisasi ini menjadi acuan dalam

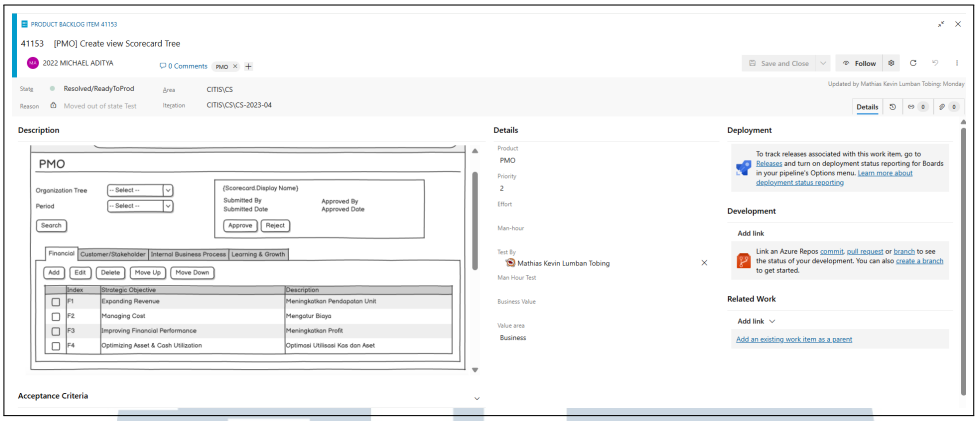
pengembangan sistem baru agar fitur dan tampilan tetap memenuhi kebutuhan pengguna.



Gambar 3.11. Tampilan *scorecard tree* pada aplikasi PMO generasi sebelumnya.

Pengembangan dilakukan berdasarkan backlog terstruktur dan wireframe seperti yang ditampilkan pada Gambar 3.12. Beberapa kebutuhan utama dalam pengembangan scorecard tree antara lain:

- Menampilkan daftar *organization tree* dan *period* yang terkait dengan master data perusahaan dan periode;
- Implementasi fungsi pencarian berdasarkan kombinasi *company id* dan *period id* untuk mengambil data scorecard yang relevan;
- Penambahan label dan kontrol status submission, serta tombol *approve*/*reject* bagi reviewer;
- Otomatisasi pembuatan tab dinamis berdasarkan jumlah perspektif pada master data;
- Penambahan fitur pengelolaan *strategic objective* di setiap tab, meliputi tombol *Add*, *Edit*, *Delete*, *Move Up*, dan *Move Down* agar daftar indikator dapat dikelola secara fleksibel.



Gambar 3.12. Backlog dan wireframe untuk pengembangan tampilan *scorecard tree*.

Pada tahap awal implementasi, form view dan fungsi pencarian (*search*) scorecard dikembangkan menggunakan Python dan ORM Odoo. Kode 3.8 dan Kode 3.9 berikut memberikan gambaran implementasi model serta tampilan form untuk *scorecard tree*. Model scorecard tree berperan sebagai pengelola struktur, relasi entitas, serta alur persetujuan scorecard.

```python
class PmoScorecardTree(models.Model):
    _name = 'pmo.scorecard.tree'
    _description = 'PMO Scorecard Tree'

    scorecard_id = fields.Many2one("pmo.scorecard")
    organization_id = fields.Many2one("pmo.organization.group", required=True)
    period_id = fields.Many2one("period", required=True)
    so_line_ids = fields.One2many('pmo.scorecard.tree.line.so', 'scorecard_tree_id', string="Strategic Objectives")

    submitter_id = fields.Many2one("res.users")
    submitted_date = fields.Datetime()
    approver_id = fields.Many2one("res.users")
    approved_date = fields.Datetime()

    def action_search(self):
        if not self.organization_id or not self.period_id:
            return {
                'warning': {
                    'title': "Missing Data",
                    'message': "Please select Company and Period before searching."
                }
            }
        scorecard = self.env['pmo.scorecard'].search([
            ('company_id', '=', self.organization_id.id),
            ('period_id', '=', self.period_id.id)
        ], limit=1)
        if scorecard:
            self.scorecard_id = scorecard.id
```

Kode 3.8: Model *Scorecard Tree* dan fungsi *search*.

Kode berikut mendefinisikan model *Scorecard Tree* di Odoo. Model ini menyimpan relasi ke scorecard, organisasi, periode, dan daftar strategic objective (melalui *so_line_ids*). Selain itu, terdapat field untuk mencatat siapa yang mengajukan, siapa yang menyetujui, serta tanggal pengajuan dan persetujuan.

Fungsi *action_search* digunakan untuk mencari data scorecard berdasarkan organisasi dan periode yang dipilih user, lalu mengisi field *scorecard_id* jika ditemukan data yang sesuai.

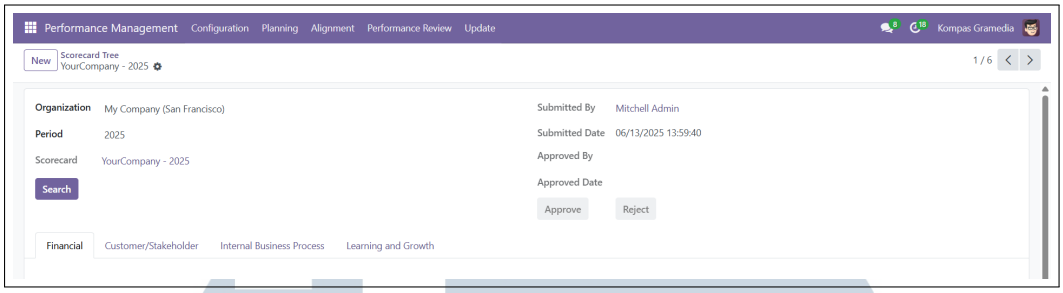
```
<record id="view_pmo_scorecard_tree_form" model="ir.ui.view">
    <field name="name">pmo.scorecard.tree.form</field>
    <field name="model">pmo.scorecard.tree</field>
    <field name="arch" type="xml">
        <form string="PMO Scorecard Tree">
            <sheet>
                <group>
                    <group>
                        <field name="organization_id"/>
                        <field name="period_id"/>
                        <field name="scorecard_id" readonly="1"/>
                        <button string="Search" type="object" name="action_search" class="btn btn-primary"/>
                    </group>
                    <group>
                        <field name="submitter_id" readonly="1"/>
                        <field name="submitted_date" readonly="1"/>
                        <field name="approver_id" readonly="1"/>
                        <field name="approved_date" readonly="1"/>
                        <button string="Approve" />
                        <button string="Reject" />
                    </group>
                </group>
                <notebook>
                    <page string="Financial"></page>
                    <page string="Customer/Stakeholder"></page>
                    <page string="Internal Business Process"></page>
                    <page string="Learning and Growth"></page>
                </notebook>
            </sheet>
        </form>
    </field>
</record>
```

Kode 3.9: XML form view *Scorecard Tree* pada Odoo.

Kode XML ini membangun tampilan form view untuk model *Scorecard Tree* di Odoo ERP. Form terdiri dari dua grup utama: Grup pertama untuk input organisasi, periode, dan scorecard, beserta tombol Search.

Grup kedua untuk menampilkan informasi submitter, approver, tanggal pengajuan, tanggal persetujuan, serta tombol Approve dan Reject. Bagian `notebook` berisi tab-tab untuk kategori perspektif (misal: Financial, Customer/Stakeholder, Internal Business Process, Learning and Growth), yang memudahkan pengguna dalam mengelola indikator kinerja secara terstruktur.

Hasil implementasi awal form view untuk *scorecard tree* pada Odoo ERP dapat dilihat pada Gambar 3.13. Gambar ini menampilkan struktur tab dan kontrol utama untuk mengelola indikator kinerja secara komprehensif.



Gambar 3.13. Tampilan form view *Scorecard Tree* pada Odoo ERP setelah implementasi awal.

Pada tahap ini, tantangan penting yang dihadapi adalah tab pada notebook untuk masing-masing perspektif di *scorecard tree* masih bersifat statis atau hardcoded di file XML. Hal ini membatasi fleksibilitas sistem, terutama jika perusahaan menambah atau mengubah perspektif bisnis.

Dengan bimbingan senior, teknik *overwrite* pada fungsi *get_view()* di model Odoo kemudian diimplementasikan. Teknik ini memungkinkan pembuatan tab pada notebook secara dinamis sesuai data master perspektif yang tersedia, sehingga perubahan atau penambahan perspektif dapat langsung tercermin pada tampilan tanpa perlu modifikasi XML. Contoh kode implementasi tersebut ditunjukkan pada Kode 3.10.

```
@api.model
def _get_view(self, view_id=None, view_type='form', **options):
    arch, view = super()._get_view(view_id, view_type, **options)

    inherit_id = self.env.ref('kg_pmo.view_pmo_scorecard_tree_form')
    string_xml = []
    perspective = self.env['pmo.perspective'].search([])


    for rec in perspective:
        string_xml.append(f'''
            <page string="{rec.name}">
                <field name="so_line_ids" domain="[('perspective_id.name','=', '{rec.name}')]">
                    <list editable="bottom">
                        <field name="strategic_objective_id" domain="[('perspective_id.name','=', '{rec.name
}')]"/>
                        <field name="perspective_id"/>
                    </list>
                </field>
            </page>
        ''')
    perspective_page = "".join(string_xml)


    arch_base = _('<?xml version="1.0"?>'
                '<data>'
                    '<xpath expr="//notebook" position="inside">'
                        f'{perspective_page}'
                    </xpath>'
                </data>')

    check_view = self.env['ir.ui.view'].sudo().search([
        ('name', '=', 'dynamic.pmo.scorecard.tree.form.inherit')
    ])
    if check_view:
```

```
35          check_view.write({'arch_base': arch_base})
36
37      else:
38          self.env['ir.ui.view'].sudo()
39          .create({
40              'name':'dynamic.pmo.scorecard.tree.form.inherit',
41              'type': 'form',
42              'model': 'pmo.scorecard.tree',
43              'mode': 'extension',
44              'inherit_id': inherit_id.id,
45              'arch_base': arch_base,
46              'active': True
47          })
48      return arch, view
```

Kode 3.10: Potongan *overwrite* fungsi _get_view() untuk tab dinamis perspective pada Odoo.

Kode di atas merupakan override fungsi `_get_view()` pada Odoo, yang menghasilkan tab notebook secara dinamis sesuai daftar perspektif pada master data (`pmo.perspective`). Setiap tab di-generate menggunakan perulangan, lalu digabungkan ke dalam arsitektur XML form dengan metode *inherit* dan *xpath*. Dengan pendekatan ini, perubahan pada data perspektif akan langsung tercermin di tampilan scorecard tree tanpa perlu mengedit file XML secara manual.

Dengan implementasi *override* pada fungsi *get_view()* tersebut, setiap perubahan pada master data perspektif—seperti penambahan, penghapusan, atau pengubahan nama—akan langsung terintegrasi ke dalam tampilan *scorecard tree* tanpa memerlukan perubahan manual pada file XML.

D Pengelolaan Strategic Objective dengan Wizard

Setelah fitur tab dinamis berhasil diimplementasikan, pengembangan dilanjutkan dengan penambahan fitur pengelolaan *strategic objective* pada masing-masing tab perspektif menggunakan *wizard* Odoo. Fitur ini memanfaatkan tombol-tombol berikut:

- **Add** – membuka wizard penambahan *strategic objective* baru sesuai perspektif yang aktif.
- **Edit** – mengaktifkan wizard pengubahan data *strategic objective* yang dipilih.
- **Delete** – menghapus *strategic objective* yang telah dipilih dari daftar.
- **Move Up** dan **Move Down** – mengatur urutan (*index*) *strategic objective* agar sesuai kebutuhan penyusunan.

- **Show KPI & SI** – menampilkan rincian KPI dan Strategy Initiative yang terhubung dengan masing-masing *strategic objective*.

tombol-tombol tersebut diintegrasikan langsung ke dalam setiap tab dinamis pada fungsi *get_view()*. Contoh kode deklarasi tombol aksi secara dinamis pada struktur tab dapat dilihat pada Kode 3.11.

```
for rec in perspective:
    string_xml.append(f"""
    <page string="{rec.name}">
        <button string="Add" class="btn-success mx-2"
                type="object"
                name="action_add_so_line"
                context="{{'perspective_id': {rec.id}}}"/>

        <button string="Delete" class="btn-danger mx-2"
                type="object"
                name="action_delete_selected_line"/>

        <button string="Edit" class="btn-info mx-2"
                type="object"
                name="action_edit_selected_line"/>

        <button string="Move Up" class="btn-outline-secondary mx-2"
                type="object"
                name="action_move_up_selected_line"/>

        <button string="Move Down" class="btn-outline-secondary mx-2"
                type="object"
                name="action_move_down_selected_line"/>

        <field name="so_line_ids"
               widget="list_domain"
               options="{{'no_open': True}}"
               context="{{'default_perspective_id': {rec.id}, 'show_kpi': 0}}"
               domain="[('perspective_id','=', {rec.id})]"
        >
            <list create="false" editable="bottom" delete="false">
                <field name="so_checkbox"/>
                <field name="strategic_objective_id" options="{{'no_open': True}}" readonly="true"/>
                <field name="perspective_id" nolabel="1" invisible="1"/>
                <field name="description" readonly="true"/>
                <button name="action_show_kpi" string="Show KPI" type="object"/>
            </list>
        </field>
    </page>
    """)
```

Kode 3.11: Deklarasi tombol Add, Edit, Delete, Move Up, dan Move Down pada tiap page perspektif secara dinamis.

Kode di atas membangun tab-tab dinamis untuk setiap perspektif pada tampilan form Odoo. Pada setiap tab, dideklarasikan beberapa tombol aksi (Add, Edit, Delete, Move Up, Move Down) yang akan memicu fungsi berbeda di backend, serta field list untuk menampilkan daftar strategic objective yang sesuai perspektif tersebut. Dengan teknik ini, UI pada form akan otomatis menyesuaikan jika perspektif di data master bertambah atau berkurang.

Setiap tombol aksi tersebut dihubungkan dengan fungsi Python pada model yang akan memanggil wizard untuk operasi yang diinginkan. Salah satu contoh

implementasi fungsi tombol Add untuk menambah *strategic objective* ditunjukkan pada Kode 3.12.

```
def action_add_so_line(self):
    perspective_id = self.env.context.get('perspective_id')
    existing_lines = self.so_line_ids.filtered(lambda l: l.perspective_id.id == perspective_id)
    if existing_lines:
        max_sequence = max(existing_lines.mapped('sequence'))
    else:
        max_sequence = 0

    new_sequence = max_sequence + 1

    return {
        'name': 'Add Strategic Objective',
        'type': 'ir.actions.act_window',
        'res_model': 'pmo.scorecard.tree.line.so.wizard',
        'view_type': 'form',
        'view_mode': 'form',
        'target': 'new',
        'context': {
            'default_scorecard_tree_id': self.id,
            'default_perspective_id': perspective_id,
            'default_sequence': new_sequence,
        },
    }
```

Kode 3.12: Contoh implementasi fungsi Add Strategic Objective.

Fungsi ini akan dijalankan saat tombol ”Add” ditekan. Ia membaca konteks perspektif yang aktif, menghitung urutan sequence baru, lalu memanggil wizard untuk menambah strategic objective baru ke scorecard tree sesuai perspektif yang dipilih.

Model *transient wizard* yang digunakan untuk menambahkan strategic objective didefinisikan pada Kode 3.13 berikut. Model ini menyediakan field dan domain yang relevan untuk pemilihan dan penambahan strategic objective baru.

```
class ScorecardTreeLineSOWizard(models.TransientModel):
    _name = 'pmo.scorecard.tree.line.so.wizard'
    _description = 'Add Strategic Objective to Scorecard Tree'

    scorecard_tree_id = fields.Many2one(
        'pmo.scorecard.tree',
        string="Scorecard Tree",
        default=lambda self: self._context.get('default_scorecard_tree_id')
    )
    perspective_id = fields.Many2one(
        'pmo.perspective',
        string="Perspective",
        default=lambda self: self.env.context.get('default_perspective_id')
    )

    flatten_so_ids = fields.Many2many(
        comodel_name='strategic.objective',
        default=lambda self: self._default_get_context_flatten_so_ids(),
    )

    strategic_objective_id = fields.Many2one(
        'strategic.objective',
        string="Strategic Objective",
        domain="[('perspective_id', '=', perspective_id) , ('id', 'not in', flatten_so_ids)]"
    )
    sequence = fields.Integer(string="Sequence", default=lambda self: self._context.get('default_sequence'))
    so_checkbox = fields.Boolean(default=False)

    def _default_get_context_flatten_so_ids(self):
```

```
        passed_ids = self.env.context.get('flatten_so_ids', [])
        if passed_ids:
            return [(6, 0, passed_ids)]
        return []

    def action_confirm_selection(self):
        self.env['pmo.scorecard.tree.line.so'].create({
            'strategic_objective_id': self.strategic_objective_id.id,
            'scorecard_tree_id': self.scorecard_tree_id.id,
            'perspective_id': self.strategic_objective_id.perspective_id.id,
            'sequence': self.sequence,
            'so_checkbox': self.so_checkbox
        })
        return {'type': 'ir.actions.act_window_close'}
```

Kode 3.13: Transient Model Wizard untuk Penambahan Strategic Objective.

Model transient wizard ini dipakai sebagai form sementara (popup/wizard) ketika user ingin menambah strategic objective ke scorecard tree. Field-field pada model ini mengatur relasi ke scorecard tree, perspektif, dan daftar strategic objective yang bisa dipilih, serta urutan (sequence) pada daftar.

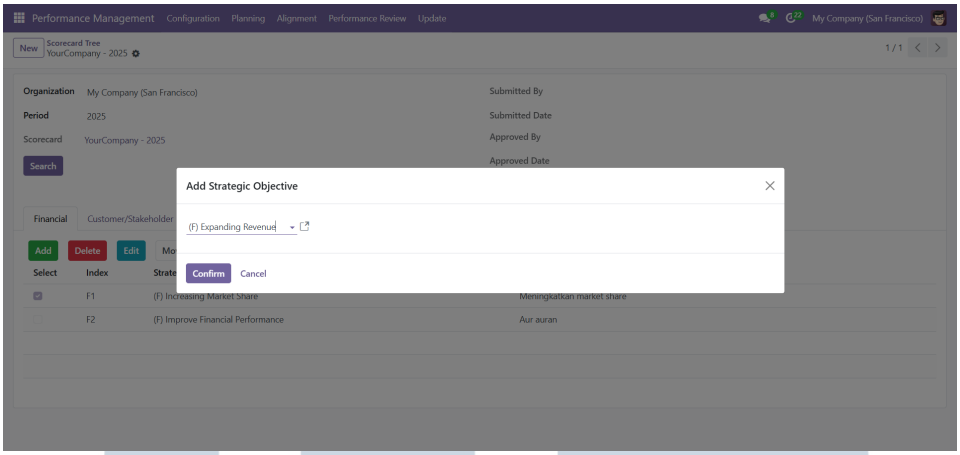
Tampilan wizard untuk Add Strategic Objective diatur melalui XML pada Kode 3.14, sehingga pengguna dapat dengan mudah memilih dan menambahkan strategic objective yang diinginkan.

```
<record id="view_scorecard_tree_line_so_wizard_form" model="ir.ui.view">
    <field name="name">scorecard.tree.line.so.wizard.form</field>
    <field name="model">pmo.scorecard.tree.line.so.wizard</field>
    <field name="arch" type="xml">
        <form string="Select Strategic Objective">
            <field string="Select Strategic Objective" name="strategic_objective_id"/>
            <footer>
                <button string="Confirm" type="object" name="action_confirm_selection" class="oe_highlight"/>
                <button string="Cancel" class="oe_link" special="cancel"/>
            </footer>
        </form>
    </field>
</record>
```

Kode 3.14: XML View untuk Wizard Penambahan Strategic Objective.

Kode XML ini mendefinisikan tampilan popup (wizard) untuk menambah strategic objective pada scorecard tree. User cukup memilih strategic objective yang ingin ditambahkan, lalu menekan tombol ”Confirm” agar data disimpan.

Hasil implementasi wizard Add Strategic Objective dapat dilihat pada Gambar 3.14. Visualisasi ini menunjukkan antarmuka pengguna saat menambahkan strategic objective ke dalam scorecard tree.



Gambar 3.14. Tampilan wizard penambahan *Strategic Objective* ke Scorecard Tree pada Odoo ERP.

Fitur **Delete** mengaktifkan fungsi *action_delete_selected_line*, yang akan memanggil wizard konfirmasi sebelum menghapus strategic objective yang dipilih. Implementasi fungsi dan model wizard delete ditunjukkan pada Kode 3.15 dan Kode 3.16.

```python
def action_delete_selected_line(self):
    for record in self:
        lines_to_delete = record.so_line_ids.filtered(lambda l: l.so_checkbox)
        if not lines_to_delete:
            raise UserError("Pilih Strategic Objective yang ingin di delete")

        wizard = self.env['pmo.scorecard.tree.line.so.delete.wizard'].create({
            'so_line_ids': [(6, 0, lines_to_delete.ids)],
        })

        return {
            'name': 'Confirm Deletion',
            'type': 'ir.actions.act_window',
            'res_model': 'pmo.scorecard.tree.line.so.delete.wizard',
            'view_mode': 'form',
            'target': 'new',
            'res_id': wizard.id,
        }
```

Kode 3.15: Contoh implementasi fungsi Delete Strategic Objective.

Fungsi ini dijalankan saat tombol Delete ditekan. Fungsi akan mengecek apakah ada strategic objective yang dipilih (menggunakan checkbox). Jika ada, wizard konfirmasi penghapusan akan ditampilkan. Jika tidak ada yang dipilih, user akan menerima pesan error.

```python
class DeleteSOLinesWizard(models.TransientModel):
    _name = 'pmo.scorecard.tree.line.so.delete.wizard'
    _description = 'Confirm Deletion of Scorecard Tree Lines'

    so_line_ids = fields.Many2many(
        'pmo.scorecard.tree.line.so',
        relation='pmo_scorecard_tree_line_del_wiz_rel',
        string="Lines to Delete",
    )

```

```python
    def action_confirm_delete(self):
        scorecard_tree_to_resequence_id = self.so_line_ids[0].scorecard_tree_id.id
        self.so_line_ids.unlink()
        if scorecard_tree_to_resequence_id:
            self.env['pmo.scorecard.tree.line.so'].resequence_for_scorecard_tree(
      scorecard_tree_to_resequence_id)
        self.so_line_ids.write({'so_checkbox': False})
        return {'type': 'ir.actions.act_window_close'}

    def action_cancel(self):
        self.so_line_ids.write({'so_checkbox': False})
        return {'type': 'ir.actions.act_window_close'}
```

Kode 3.16: Transient Model Wizard untuk Konfirmasi Penghapusan Strategic Objective.

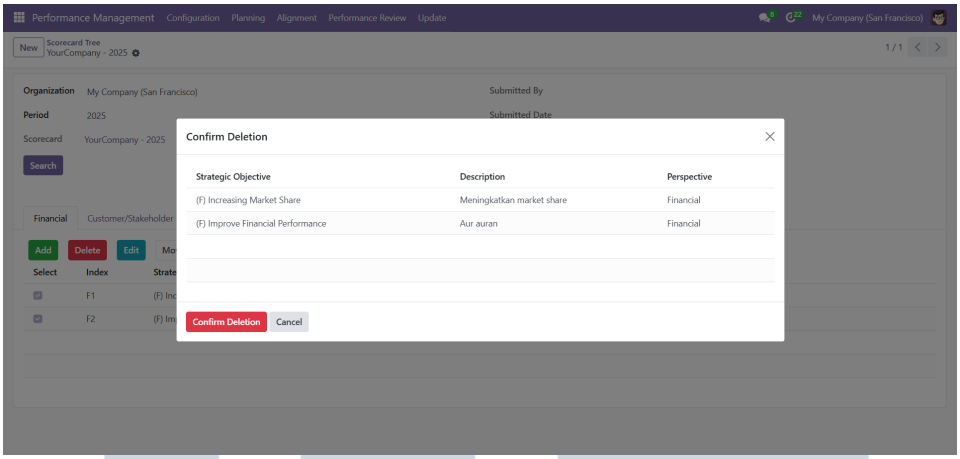
Model transient wizard ini berfungsi sebagai jendela konfirmasi penghapusan. *Field so_line_ids* menyimpan daftar strategic objective yang akan dihapus. Fungsi *action_confirm_delete* akan menghapus data tersebut dari scorecard tree dan merapikan urutan, sedangkan *action_cancel* akan menutup wizard tanpa menghapus data.

Tampilan wizard konfirmasi penghapusan diatur dengan XML pada Kode 3.17, sehingga pengguna dapat memverifikasi data sebelum menghapus.

```xml
<record id="view_delete_so_lines_wizard_form" model="ir.ui.view">
    <field name="name">delete.so.lines.wizard.form</field>
    <field name="model">pmo.scorecard.tree.line.so.delete.wizard</field>
    <field name="arch" type="xml">
        <form string="Confirm Deletion">
            <sheet>
                <group>
                    <field name="so_line_ids" nolabel="1" readonly="1">
                        <list>
                            <field name="strategic_objective_id" readonly="true"/>
                            <field name="description" readonly="true"/>
                            <field name="perspective_id" readonly="true"/>
                        </list>
                    </field>
                </group>
            </sheet>
            <footer>
                <button name="action_confirm_delete"
                        string="Confirm Deletion"
                        type="object"
                        class="btn-danger"/>
                <button name="action_cancel"
                        string="Cancel"
                        type="object"
                        class="btn-secondary"/>
            </footer>
        </form>
    </field>
</record>
```

Kode 3.17: XML View untuk Wizard Konfirmasi Penghapusan Strategic Objective.

Kode XML ini membangun tampilan wizard konfirmasi penghapusan. Di sini, user dapat melihat daftar strategic objective yang akan dihapus. Tombol "Confirm Deletion" akan menjalankan proses hapus, sementara tombol "Cancel" menutup wizard tanpa mengubah data. Hasil tampilan wizard konfirmasi penghapusan strategic objective dapat dilihat pada Gambar 3.15.



Gambar 3.15. Tampilan wizard konfirmasi penghapusan *Strategic Objective* pada Scorecard Tree.

Untuk mengedit data strategic objective pada scorecard tree, tombol **Edit** dapat digunakan. Implementasi fungsi dan wizard edit ditunjukkan pada Kode 3.18 dan Kode 3.19. Wizard ini memungkinkan pengguna memperbarui informasi strategic objective yang dipilih secara mudah.

```
def action_edit_selected_line(self):
    for record in self:
        lines_to_edit = record.so_line_ids.filtered(lambda l: l.so_checkbox)
        if not lines_to_edit:
            raise UserError("Pilih strategic objective yang ingin di edit.")
        if len(lines_to_edit) > 1:
            raise UserError("Tidak bisa edit lebih dari 1 strategic objective secara bersamaan.")

        line = lines_to_edit[0]

        return {
            'name': 'Edit Strategic Objective Line',
            'type': 'ir.actions.act_window',
            'res_model': 'pmo.scorecard.tree.line.so.edit.wizard',
            'view_mode': 'form',
            'target': 'new',
            'context': {
                'default_line_id': line.id,
            },
        }
```

Kode 3.18: Fungsi pemanggilan wizard Edit Strategic Objective.

Fungsi ini dijalankan saat tombol Edit pada tab perspektif ditekan. Fungsi akan memastikan hanya satu strategic objective yang dipilih (checkbox). Selanjutnya, wizard edit akan dibuka untuk record yang dipilih, dengan membawa context `default_line_id`.

```
class ScorecardTreeLineSOEditWizard(models.TransientModel):
    _name = 'pmo.scorecard.tree.line.so.edit.wizard'
    _description = 'Edit Strategic Objective of Scorecard Tree'

    line_id = fields.Many2one(
        'pmo.scorecard.tree.line.so',
        string="Scorecard Line",
        required=True
```

```
    )
    strategic_objective_id = fields.Many2one(
        'strategic.objective',
        string="Strategic Objective"
    )
    perspective_id = fields.Many2one(
        'pmo.perspective',
        string="Perspective"
    )
    description = fields.Text(string="Description")
    so_checkbox = fields.Boolean(default=False)

    @api.model
    def default_get(self, fields_list):
        res = super().default_get(fields_list)
        line_id = self.env.context.get('default_line_id')
        if line_id:
            line = self.env['pmo.scorecard.tree.line.so'].browse(line_id)
            res.update({
                'line_id': line.id,
                'strategic_objective_id': line.strategic_objective_id.id,
                'perspective_id': line.perspective_id.id,
                'description': line.strategic_objective_id.description
            })
        return res

    @api.onchange('strategic_objective_id')
    def _onchange_strategic_objective_id(self):
        if self.strategic_objective_id:
            self.perspective_id = self.strategic_objective_id.perspective_id
            self.description = self.strategic_objective_id.description

    def action_confirm_edit(self):
        self.line_id.write({
            'strategic_objective_id': self.strategic_objective_id.id,
            'perspective_id': self.perspective_id.id,
            'so_checkbox': self.so_checkbox,
        })

        if self.strategic_objective_id:
            self.strategic_objective_id.write({
                'description': self.description,
            })
        return {'type': 'ir.actions.act_window_close'}
```

Kode 3.19: Transient Model Wizard Edit Strategic Objective.

Model transient wizard ini digunakan sebagai form edit sementara. Field-field pada wizard akan otomatis terisi data strategic objective yang dipilih. Fungsi `default_get` mengisi data awal, sedangkan fungsi `action_confirm_edit` akan menyimpan perubahan ke database.

Tampilan wizard edit strategic objective disusun melalui XML pada Kode 3.20. Dengan wizard ini, pengguna dapat memperbarui field dan deskripsi strategic objective yang diinginkan.

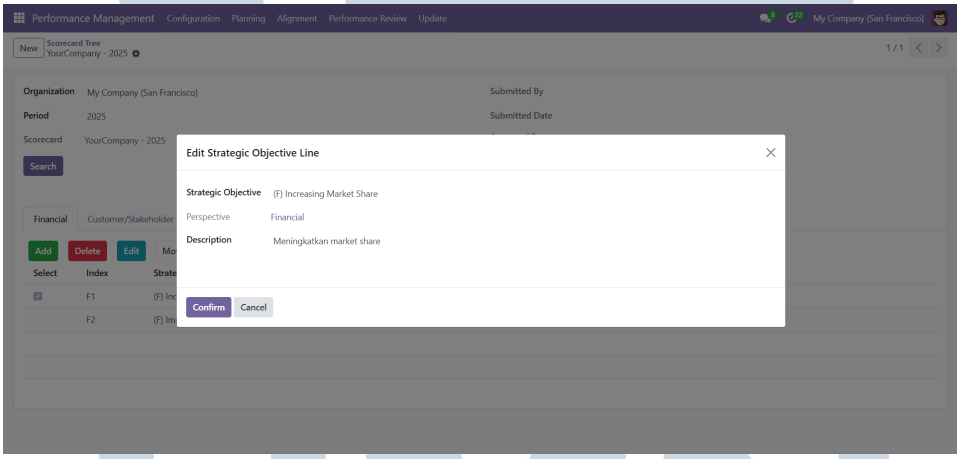
```
<record id="view_scorecard_tree_line_so_edit_wizard_form" model="ir.ui.view">
    <field name="name">scorecard.tree.line.so.edit.wizard.form</field>
    <field name="model">pmo.scorecard.tree.line.so.edit.wizard</field>
    <field name="arch" type="xml">
        <form string="Edit Line">
            <group>
                <field name="line_id" invisible="1"/>
                <field name="strategic_objective_id"/>
                <field name="perspective_id" readonly="1"/>
                <field name="description"/>
            </group>
```

```
            <footer>
                <button string="Confirm" type="object" name="action_confirm_edit" class="btn-primary"/>
                <button string="Cancel" class="btn-secondary" special="cancel"/>
            </footer>
        </form>
    </field>
</record>
```

Kode 3.20: XML View Wizard Edit Strategic Objective.

Kode XML ini membuat tampilan form popup wizard edit strategic objective. User dapat mengubah strategic objective, melihat perspektif yang terkait (hanya baca), dan mengedit deskripsi. Terdapat tombol "Confirm" untuk menyimpan perubahan dan "Cancel" untuk menutup wizard tanpa menyimpan.

Gambar 3.16 memperlihatkan tampilan wizard edit strategic objective setelah fungsi Edit dijalankan.



Gambar 3.16. Tampilan wizard pengeditan *Strategic Objective* pada Scorecard Tree.

Fungsi pengaturan urutan strategic objective dapat dijalankan menggunakan tombol **Move Up** dan **Move Down**. Tidak seperti tombol Edit yang memanggil wizard, kedua tombol ini menjalankan logika backend secara langsung untuk memperbarui urutan data. Implementasi fungsi tersebut dapat dilihat pada Kode 3.21.

```
def action_move_up_selected_line(self):
    for record in self:
        lines_to_move = record.so_line_ids.filtered(lambda l: l.so_checkbox)
        if not lines_to_move:
            raise UserError("Silahkan Pilih Strategic Objective")
        if len(lines_to_move) > 1:
            raise UserError("Maksimal hanya bisa memindahkan satu Strategic Objective")
        lines_to_move.move_up()
    return True

def action_move_down_selected_line(self):
    for record in self:
        lines_to_move = record.so_line_ids.filtered(lambda l: l.so_checkbox)
        if not lines_to_move:
```

```
            raise UserError("Silahkan Pilih Strategic Objective")
        if len(lines_to_move) > 1:
            raise UserError("Maksimal hanya bisa memindahkan satu Strategic Objective")
        lines_to_move.move_down()
    return True
```

Kode 3.21: Implementasi Fungsi Move Up dan Move Down Strategic Objective.

Fungsi `action_move_up_selected_line` dan `action_move_down_selected_line` digunakan untuk mengubah urutan (*sequence*) `strategic objective` pada `scorecard tree`.

- Pengguna harus memilih `strategic objective` yang akan dipindahkan dengan menggunakan checkbox pada daftar.
- Jika tidak ada data yang dipilih, atau lebih dari satu yang dipilih, maka akan muncul pesan error (`UserError`).
- Fungsi `move_up()` dan `move_down()` akan menukar posisi `strategic objective` pada daftar, sehingga urutan dapat diatur langsung pada UI tanpa perlu wizard tambahan.
- Dengan fitur ini, penataan prioritas atau posisi indikator kinerja dalam modul dapat dilakukan secara praktis dan interaktif.

Dengan adanya rangkaian fitur di atas, proses pengelolaan strategic objective pada scorecard tree dapat dilakukan secara efisien, sistematis, dan terintegrasi bagi pengguna, sehingga mendukung kemudahan pemantauan serta pengelolaan indikator kinerja organisasi.

3.4 Kendala dan Solusi yang Ditemukan

Selama masa magang di Kompas Gramedia, berbagai tantangan dihadapi, baik dari sisi teknis maupun non-teknis. Berikut adalah uraian kendala beserta solusi yang telah ditemukan:

A. Kendala yang Dihadapi

Berikut adalah uraian kendala yang ditemukan selama masa magang:

1. **Belum Familiar dengan Odoo 18 dan Bahasa Python** Belum terdapat pengalaman sebelumnya dalam pengembangan aplikasi menggunakan Odoo maupun bahasa Python. Tantangan menjadi semakin besar karena Odoo

18 baru saja dirilis pada Oktober 2024, hanya sekitar tiga bulan sebelum periode magang dimulai. Dokumentasi Odoo 18 pun masih terbatas, sehingga diskusi maupun tutorial daring yang relevan belum banyak tersedia. Selain itu, beberapa fungsi mengalami perubahan atau telah dihapus (*deprecated*).

2. **Deskripsi Backlog yang Kurang Jelas**
   Beberapa *backlog* yang diterima memiliki penjelasan yang sangat singkat atau hanya berupa wireframe sederhana, sehingga diperlukan diskusi langsung bersama analis. Namun, sering kali analis sedang sibuk atau tidak berada di lokasi, sehingga komunikasi menjadi tertunda.

B. Solusi yang Ditemukan

Berikut adalah uraian solusi berdasarkan kendala yang ditemukan selama masa magang:

1. **Pendalaman Dokumentasi dan Sumber Referensi Resmi** Pembelajaran mandiri dilakukan dengan memanfaatkan *YouTube* dan *Udemy*, mendalami dokumentasi Odoo 18, serta melakukan *trial and error* langsung pada kode. Selain itu, eksplorasi pada repository Odoo resmi digunakan untuk mempelajari *best practice* dalam pembuatan fungsi atau implementasi yang belum banyak dibahas secara daring.

2. **Diskusi dengan Senior dan Analis**
   Untuk *backlog* yang kurang jelas, diskusi aktif dilakukan, baik secara langsung maupun melalui chat dengan senior atau analis, meskipun terkadang harus menunggu waktu yang tepat. Catatan pribadi juga dibuat untuk referensi jika menemui masalah serupa pada *backlog* berikutnya.

3. **Manfaat Feedback dan Code Review**
   Feedback dan *code review* dari senior sangat membantu dalam meningkatkan efisiensi kode serta pemahaman relasi antar model maupun *field* pada Odoo. Review yang diberikan mempersingkat *logic* atau memperbaiki struktur kode agar lebih efektif dan sesuai standar pengembangan di Kompas Gramedia.