

BAB 3

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam program magang yang dilaksanakan di PT Klik Semangat Indonesia, posisi yang ditempati adalah sebagai *Odoo Developer*, yang berada dalam lingkup tim teknologi informasi perusahaan. Kedudukan ini secara langsung mendukung divisi operasional dan keuangan, khususnya dalam aspek pengembangan dan penyesuaian sistem ERP berbasis Odoo.

Selama pelaksanaan magang, kegiatan dilakukan dengan pengawasan dan bimbingan dari Bapak Fatisokhi Harefa selaku supervisor. Koordinasi dilakukan secara berkala untuk memastikan bahwa setiap tugas yang diberikan sejalan dengan kebutuhan sistem dan kebijakan internal perusahaan. Selain itu, terdapat komunikasi rutin dengan anggota tim teknis dan pengguna akhir dari sistem ERP untuk memahami alur kerja dan kebutuhan fungsional secara lebih rinci.

Lingkup kerja juga mencakup kolaborasi lintas departemen, terutama ketika fitur yang dikembangkan berkaitan dengan proses bisnis yang melibatkan divisi penjualan, gudang, dan akuntansi. Mekanisme koordinasi diterapkan secara hybrid, yaitu kombinasi antara bekerja dari kantor (*work from office*) dan bekerja dari rumah (*work from home*), sesuai dengan jadwal dan kebijakan perusahaan.

Magang ini juga diikuti oleh tiga mahasiswa dari Universitas Multimedia Nusantara, yang secara kolektif terlibat dalam proyek pengembangan sistem informasi. Setiap peserta ditempatkan dalam proyek dan modul yang berbeda, namun tetap berada dalam koordinasi tim teknologi informasi yang sama.

Sebagai bentuk dukungan perusahaan terhadap kontribusi peserta magang, diberikan honorarium dengan kisaran Rp600.000 hingga Rp800.000 per bulan selama masa pelaksanaan. Pemberian honor ini menjadi bentuk apresiasi atas partisipasi aktif dalam proses pengembangan sistem internal.

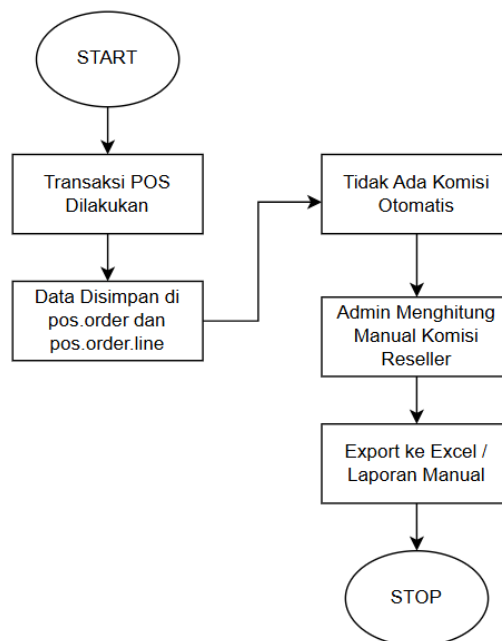
Kedudukan sebagai pengembang dalam proyek ini memberikan peran penting dalam menjembatani kebutuhan operasional dengan solusi teknologi yang terintegrasi, serta menuntut pemahaman mendalam terhadap struktur sistem Odoo, standar pengkodean, serta praktik pengujian dan dokumentasi teknis.

3.2 Tugas yang Dilakukan

Selama menjalani program magang di PT. Klik Semangat Indonesia, berbagai tugas diberikan oleh supervisor untuk mengembangkan dan menyempurnakan fitur dalam sistem ERP perusahaan. Tugas-tugas tersebut mencakup pengembangan modul odoo perusahaan.

3.2.1 Pengembangan Fitur *Reseller Commission* pada *Product*

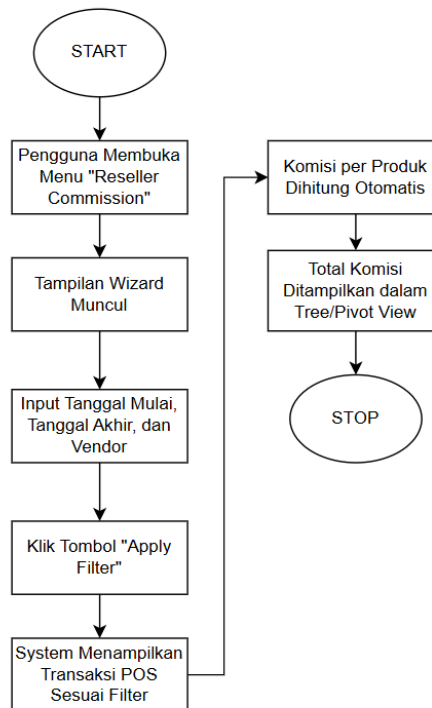
Fitur *reseller commission* dikembangkan dari awal untuk menjawab kebutuhan perusahaan dalam menghitung dan memantau komisi yang diperoleh oleh reseller berdasarkan transaksi penjualan yang dilakukan melalui modul *Point of Sale* (POS). Sebelumnya, sistem belum memiliki mekanisme atau modul khusus yang dapat menghitung komisi secara otomatis, sehingga seluruh proses dilakukan secara manual dan kurang efisien.



Gambar 3.1. Diagram *Before Reseller Commission Development*

Gambar 3.1 menunjukkan alur sistem sebelum pengembangan fitur *Reseller Commission*. Pada tahap ini, proses pencatatan dan perhitungan komisi reseller

dilakukan secara manual oleh staf administrasi, yang berpotensi menimbulkan kesalahan dan keterlambatan dalam proses pencatatan. Tidak adanya sistem otomatis mengharuskan verifikasi manual terhadap setiap transaksi yang melibatkan reseller.



Gambar 3.2. Diagram *After Reseller Commission Development*

Gambar 3.2 menggambarkan alur kerja sistem setelah fitur *Reseller Commission* berhasil dikembangkan dan diimplementasikan. Dengan modul baru ini, sistem dapat secara otomatis menghitung komisi berdasarkan data transaksi yang dilakukan di modul *Point of Sale* (POS), sehingga proses menjadi lebih cepat, akurat, dan efisien.

Products / Filtered Reseller Commission

MEASURES ▾			⇌	+	↓
✓ Subtotal ✓ Total Commission - Count - panana - Administrator - Shop/0002	+ Total				
	Total Commission		Subtotal		
	0.00		1.11		
	0.00		1.11		
	0.00		1.11		
	0.00		1.11		

Gambar 3.3. Tampilan *Pivot dan Tree Commission*

Pengembangan reseller commission di atas menerapkan metode related field yang dimana nilai komisi yang ditampilkan merupakan aksesan komisi dari produk terkait dan juga menerapkan computed field dimana total commission dihitung dari 1 baris transaksi dengan mengalikan komisi produk terkait dengan subtotal harga.

```

        string="Customer"
    )

    total_commission = fields.Float(
        string='Total Commission',
        readonly=True,
        compute='_compute_total_commission',
        group_operator='sum',
        store=True
    )

    @api.depends('price_subtotal', 'qty')
    def _compute_unit_price(self):
        """Menghitung harga per unit"""
        for record in self:
            record.unit_price = (record.price_subtotal or 0.0) / (record.qty or 1.0)

    @api.depends('product_commission', 'price_subtotal')
    def _compute_total_commission(self):
        """Compute total commission based on subtotal and commission percentage"""
        for record in self:
            record.total_commission = record.product_commission * record.price_subtotal

    @api.depends('product_commission', 'unit_price')
    def _compute_commission(self):
        """Compute commission per unit"""
        for record in self:
            record.commission = record.product_commission * record.unit_price

class ProductTemplate(models.Model):
    _inherit = 'product.template'

    product_commission = fields.Float(string="Product Commission (%)")

```

Gambar 3.4. Kode *Python Reseller Commission*

Kode ini mengembangkan model `pos.order.line` untuk menghitung komisi reseller berdasarkan produk yang dijual melalui Point of Sale (POS). Field `product commission` diambil dari produk terkait, dan `total commission` dihitung otomatis dari persentase komisi dikali subtotal transaksi.

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A

```

from odoo import models, fields, api

class ResellerCommissionWizard(models.TransientModel):
    _name = 'reseller.commission.wizard'
    _description = 'Filter Wizard for Reseller Commission'

    start_date = fields.Date(string="Start Date", required=True)
    end_date = fields.Date(string="End Date", required=True)
    company_id = fields.Many2one(
        'res.company', string="Company", required=True,
        default=lambda self: self.env.company.id
    )

    def action_apply_filter(self):
        """Apply filter based on selected date range and company"""
        return {
            'type': 'ir.actions.act_window',
            'name': 'Filtered Reseller Commission',
            'view_mode': 'pivot,tree',
            'res_model': 'pos.order.line',
            'domain': [('create_date', '>=', self.start_date),
                      ('create_date', '<=', self.end_date),
                      ('company_id', '=', self.company_id.id)],
            'target': 'current',
        }

```

Gambar 3.5. Kode Python Wizard Reseller Commission

Pengembangan fitur wizard reseller commission juga digunakan untuk memfilter data point of sale yang terjadi, dikarenakan jika menampilkan semua akan mengakibatkan menumpuknya data point of sale maka sebelum menampilkan datanya, dipakailah filter ini yang berdasarkan date atau tanggal yang ingin dipilih dan juga berdasarkan vendor yang dipilih sehingga data yang tampil hanya untuk vendor itu saja dengan jangka waktu sekian sampai sekian.

Proses-Proses Reseller Commission:

1. Pengguna membuka menu *Reseller Commission*.
2. Wizard ditampilkan (*Transient Model: reseller.commission.wizard*).
3. User klik tombol "Apply Filter".
4. Fungsi *action apply filter* membuat *action window* baru.

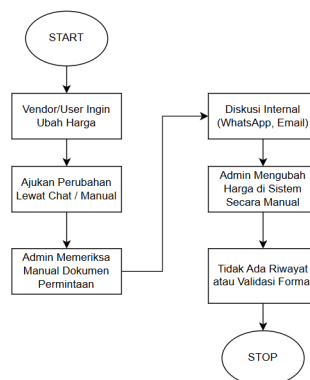
5. Data ditampilkan sesuai *filter*.
6. Komisi dihitung otomatis per produk di *pos.order.line*.

Gambar 3.6. Tampilan *Filter Reseller Commission*

Tampilan *wizard* ini dirancang dalam bentuk *form* yang memfilter berdasarkan 3 *field* di dalamnya yaitu *company*, *start date* dan *end date*. *Wizard* ini merupakan fitur tambahan berbasis *TransientModel* yang digunakan untuk memfilter data *Point of Sale* (POS).

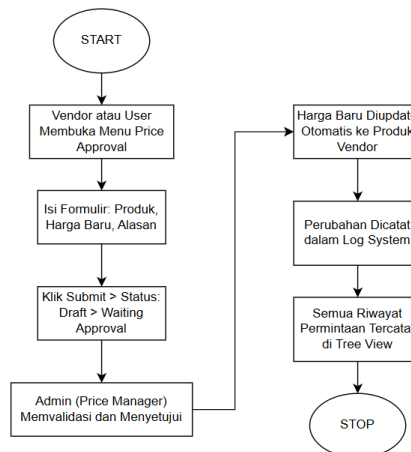
3.2.2 Pengembangan Custom Modul *Price Approval*

Modul *Price Approval* dikembangkan sebagai solusi untuk mengelola proses pengajuan perubahan harga jual dan beli produk secara terstruktur. Sebelumnya, perusahaan belum memiliki sistem khusus yang menangani alur pengajuan harga dari vendor, sehingga seluruh proses dilakukan secara manual, yang rentan terhadap kesalahan dan kurang efisien.



Gambar 3.7. Diagram *Before Price Approval Development*

Gambar 3.7 menggambarkan alur proses perubahan harga sebelum dikembangkan modul Price Approval. Seluruh pengajuan perubahan harga dilakukan secara manual melalui komunikasi langsung antara vendor dan pihak internal. Tidak adanya sistem pengelolaan yang terstruktur menyebabkan proses menjadi lambat, rawan kesalahan, serta sulit untuk dilacak atau dievaluasi kembali.



Gambar 3.8. Diagram *After Price Approval Development*

Gambar 3.8 menunjukkan alur proses setelah modul Price Approval diimplementasikan. Dalam alur ini, pengajuan harga dilakukan melalui sistem secara langsung oleh vendor atau user internal. Proses pengajuan dilengkapi dengan tahapan validasi serta persetujuan yang terdokumentasi secara otomatis, sehingga memastikan setiap perubahan harga telah melewati prosedur yang sah dan dapat dipertanggungjawabkan.

Dengan adanya modul ini, vendor atau user internal dapat mengajukan permohonan perubahan harga secara langsung melalui sistem. Setiap pengajuan akan melalui tahapan validasi dan persetujuan oleh admin, sehingga perusahaan dapat memastikan bahwa setiap perubahan harga telah disetujui sesuai dengan kebijakan dan prosedur yang berlaku.

Proses-Proses *Price Approval*:

1. Pengajuan Harga Baru.
2. Validasi Pengajuan.
3. Persetujuan Admin.

4. Implementasi Harga Baru.

Pertama yang dilakukan adalah membuat boolean dengan menunjukkan bahwa user tersebut price manager atau bukan. Boolean ini hanya bisa diinteraksi oleh administrator database, sehingga yang menentukan user tersebut price manager atau bukan ialah administrator. Juga dibuat boolean yang menunjukkan vendor tersebut merupakan branch atau bukan. Hal ini dilakukan karena adanya fitur tambahan yang diinginkan ketika vendor utama mengajukan perubahan harga dan disetujui, maka terdapat kondisi di mana branches harus mengikuti harga dari vendor utama tersebut.

Berikut merupakan tampilan pengaturan untuk menentukan apakah user merupakan price manager dan apakah vendor merupakan branch:

Gambar 3.9. Tampilan *Toggle Price Manager, Is Branch*

Adapun kode Python yang digunakan untuk mendefinisikan field boolean tersebut dapat dilihat pada gambar berikut:

```
class Vendor(models.Model):
    _inherit = 'res.partner'

    is_branch_vendor = fields.Boolean(
        string="Branches", help="Ceklis jika vendor ini adalah antar cabang.")

class ResUsers(models.Model):
    _inherit = 'res.users'
    admin_edit = fields.Boolean(
        string="Price Manager", default=False)
```

Gambar 3.10. Kode *Python Boolean Price Approval*

Kedua, membuat modelnya dan menyediakan fields seperti request by, *reason sales price change*, dan *state*, yang menggambarkan status pengajuan (*draft, waiting*

approval, approved). Hal ini memungkinkan pengguna untuk melacak status pengajuan harga secara transparan. Berikut tampilannya:

The screenshot shows the Odoo Price Approval form. It has a header section with fields for 'Approval Reference' (REQ-00001), 'Date' (04/16/2025), 'Update All Branches' (checkbox), 'Request By' (Administrator), and 'Reason For Application'. Below this is a section with two tabs: 'Sale Price Approval' and 'Purchase Price Approval'. At the bottom, there is a table with columns for 'Product', 'Old Sale Price', and 'New Sale Price', and an 'Add a line' button.

Gambar 3.11. Tampilan *Form Field Price Approval*

Adapun kode Python yang digunakan untuk mendefinisikan semua *fields Price Approval* tersebut dapat dilihat pada gambar berikut:

```
class ITPriceApproval(models.Model):
    _name = "it.price.approval"
    _description = "Price Approval"
    _inherit = ['mail.thread', 'mail.activity.mixin']

    type = fields.Selection([
        ('sale', 'Sale Price Approval'),
        ('purchase', 'Purchase Price Approval')
    ], string="Switch Price Type", required=True, default='sale')

    request_by = fields.Many2one('res.users', string="Request By", required=True, default=lambda self: self.env.user.id, readonly=True)
    reason_sales_price_change = fields.Text(string="Reason For Application", tracking=True)
    approve_by = fields.Many2one('res.users', string="Responded By")
    reason_approval = fields.Text(string="Reason Approved/Pending", tracking=True, default="")

    name = fields.Char(
        string="Approval Reference",
        required=True,
        copy=False,
        default=lambda self: self.env['ir.sequence'].next_by_code('it.price.approval') or 'New',
        tracking=True
    )

    date = fields.Date(string="Date", default=fields.Date.context_today, tracking=True)
    update_all_vendors = fields.Boolean(
        string="Update All Branches",
        help="If checked, updates the purchase price for all vendors of the selected product, including branch vendors.",
        tracking=True
    )

    state = fields.Selection([
        ('draft', 'Draft'),
        ('waiting', 'Waiting Approval'),
        ('approved', 'Approved')
    ], string="Status", default='draft', tracking=True)
```

Gambar 3.12. Kode Python *Fields Price Approval*

Ketiga, mengembangkannya dengan validasi dan fitur persetujuan agar hanya pengguna dengan akses tertentu (seperti *price manager* sebelumnya) yang dapat

melakukan perubahan penting pada data, seperti mengubah alasan persetujuan dan menghapus records. Berikut tampilannya:

Price Approval / REQ-00002

SAVE **DISCARD**

Approve DRAFT **WAITING APPROVAL** APPROVED

Approval Reference REQ-00002
 Date 04/16/2025
 Update All Branches ☐
 Request By Administrator
 Reason For Application
 Responded By
 Reason Approved/Pending

Sale Price Approval Purchase Price Approval

Product	Old Sale Price	New Sale Price
apple	1.00	2.00

Gambar 3.13. Tampilan *Approving Page Price Approval*

Adapun kode Python yang digunakan untuk mendefinisikan *validation* tersebut dapat dilihat pada gambar berikut:

```
def _compute_admin_edit(self):
    """Set nilai admin_edit berdasarkan apakah user memiliki hak admin."""
    for record in self:
        record.admin_edit = self.env.user.admin_edit

def unlink(self):
    """Hanya pengguna dengan 'admin_edit' yang dapat menghapus"""
    if not self.env.user.admin_edit:
        raise ValidationError(_("Only administrators can delete records. "))
    return super(ITPriceApproval, self).unlink()

@api.model
def create(self, vals):
    if 'reason_approval' in vals and not self.env.user.admin_edit and vals.get('state') != 'draft':
        raise ValidationError(_("Only admins can set the reason for approval. "))
    return super(ITPriceApproval, self).create(vals)

def write(self, vals):
    if self.state != 'draft' and any(field in vals for field in ['sale_line_ids', 'purchase_line_ids', 'date']):
        raise ValidationError(_("You cannot edit this record unless it is in Draft state. "))

    if self.state != 'draft' and 'reason_approval' in vals and not self.env.user.admin_edit:
        raise ValidationError(_("Only admins can modify the reason for approval. "))

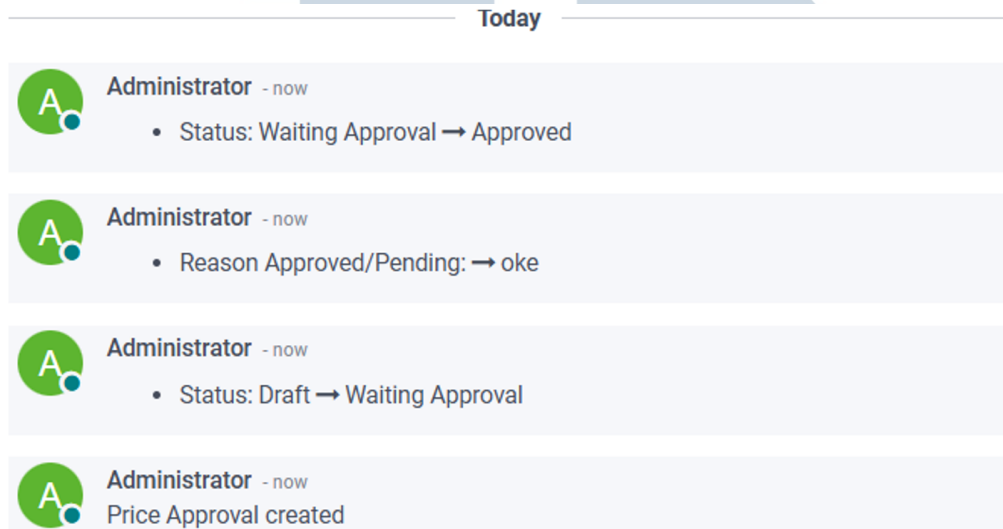
    return super(ITPriceApproval, self).write(vals)

def action_submit(self):
    if not (self.sale_line_ids or self.purchase_line_ids):
        raise ValidationError(_("You must add at least one product before submitting. "))
    self.write({'state': 'waiting'})

def action_approve(self):
```

Gambar 3.14. Kode *Python Price Approval Validation*

Juga, atas permintaan user yang ingin setelah persetujuan dilakukan, sistem akan secara otomatis memperbarui harga di vendor yang terkait dan mencatat perubahan tersebut dalam log yang dapat diakses oleh tim terkait. Berikut tampilannya:



Gambar 3.15. Tampilan *Log Message Price Approval*

Dan terakhir, membuat tree yang menunjukkan semua history request baik yang diterima, draft ataupun pending. Berikut tampilannya:

Price Approval

CREATE

Search...

Filters Group By Favorites 1/3 / 3

Approval Reference	Date	Switch Price Type	Status
☐ REQ-00002	04/16/2025	Sale Price Approval	Waiting Approval
☐ REQ-00004	04/16/2025	Sale Price Approval	Waiting Approval
☐ REQ-00007	04/16/2025	Sale Price Approval	Approved

Gambar 3.16. Tampilan *Price Approval Tree*

Adapun kode XML yang digunakan untuk membuat tampilan pada tree gambar 3.16 tersebut dapat dilihat pada lampiran berikut:

```

<odoo>
<record id="view_it_price_approval_form" model="ir.ui.view">
  <field name="arch" type="xml">
    <form string="Price Approval">
      <sheet>
        <div class="oe_chatter">
          <field name="message_ids" widget="mail_thread"/>
        </div>
      </form>
    </field>
  </record>

  <record id="view_it_price_approval_tree" model="ir.ui.view">
    <field name="name">it.price.approval.tree</field>
    <field name="model">it.price.approval</field>
    <field name="arch" type="xml">
      <tree string="Price Approval">
        <field name="name"/>
        <field name="date"/>
        <field name="type"/>
        <field name="state" widget="badge"/>
      </tree>
    </field>
  </record>

  <record id="action_it_price_approval" model="ir.actions.act_window">
    <field name="name">Price Approval</field>
    <field name="res_model">it.price.approval</field>
    <field name="view_mode">tree,form</field>
  </record>

  <menuitem id="menu_it_price_approval" name="Price Approval"
    action="action_it_price_approval" sequence="10"/>
</odoo>

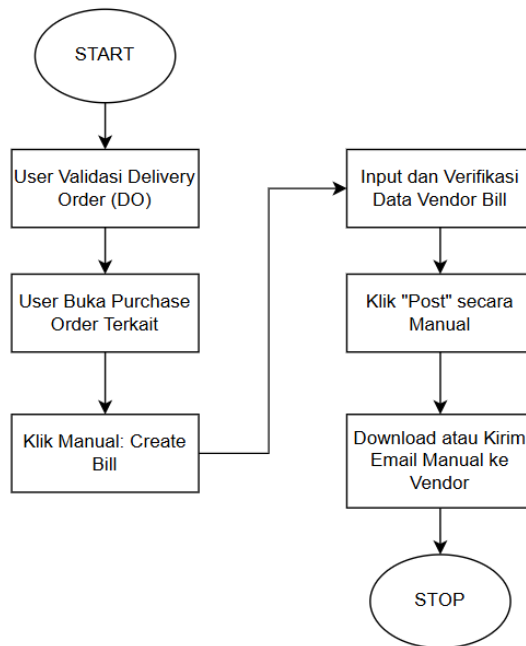
```

Gambar 3.17. Kode XML *Price Approval Tree*

3.2.3 Pengembangan *Custom Auto Invoice and Bill*

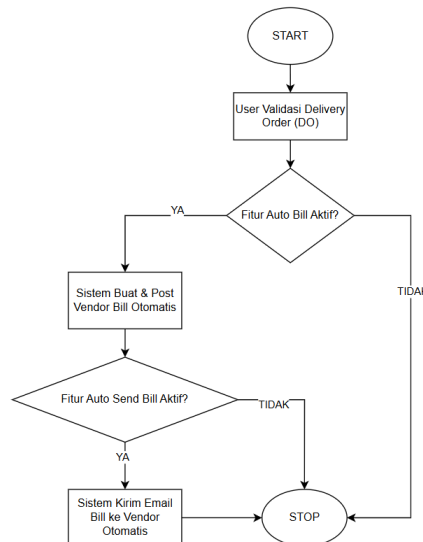
Modul ini bertujuan untuk mengotomatisasi proses pembuatan dan pengiriman *invoice* dan *vendor bill* pada saat proses validasi pengiriman (*delivery validation*) di Odoo. Dengan fitur ini, perusahaan dapat menghemat waktu dan mengurangi risiko lupa membuat *invoice* atau *bill* secara manual setelah pengiriman barang dilakukan.

Gambar berikut menunjukkan alur proses sebelum fitur *Auto Bill* dikembangkan, di mana seluruh pembuatan tagihan masih dilakukan secara manual setelah proses pengiriman:



Gambar 3.18. Diagram *Before Auto Bill Development*

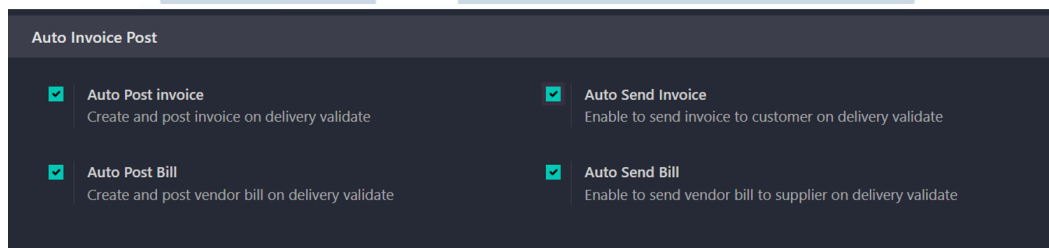
Setelah fitur dikembangkan, proses pengajuan tagihan menjadi otomatis. Gambar berikut memperlihatkan alur kerja sistem setelah implementasi modul Auto Bill:



Gambar 3.19. Diagram *After Auto Bill Development*

Pertama yang dilakukan adalah membuat fitur pada menu *Settings* pada *Accounting* dalam bentuk *checkbox*, dan disimpan menggunakan *ir.config.parameter*, sehingga bersifat global dan akan tetap aktif meskipun server melakukan *restart*. Dengan adanya fitur ini, proses akuntansi menjadi lebih efisien, terutama pada perusahaan yang menangani volume transaksi pembelian yang tinggi, karena dapat mengurangi pekerjaan administratif yang bersifat repetitif. Fitur ini merupakan bagian dari pengembangan lanjutan, dan tidak mengubah atau menambahkan fungsi pada proses *invoice* penjualan (*customer invoice*), yang sudah ada sebelumnya.

Berikut merupakan tampilan antarmuka dari pengaturan sistem Auto Bill yang telah ditambahkan:



Gambar 3.20. Tampilan *Auto Bill System*

Kode Python berikut digunakan untuk menyimpan konfigurasi Auto Bill pada parameter sistem:

```
class ResConfigSettings(models.TransientModel):
    _inherit = "res.config.settings"

    is_create_invoice_delivery_validate = fields.Boolean(
        string="Auto Post invoice", config_parameter=
        'automatic_invoice_and_post.is_create_invoice_delivery_validate',
        help="Create and post invoice on delivery validate")

    is_auto_send_invoice = fields.Boolean(string="Auto Send Invoice",
        config_parameter=
        'automatic_invoice_and_post.is_auto_'
        'send_invoice',
        help="Enable to send invoice to "
        "customer on delivery validate ")

    is_create_bill_delivery_validate = fields.Boolean(
        string="Auto Post Bill",
        config_parameter="automatic_invoice_and_post.is_create_bill_delivery_validate",
        help="Create and post vendor bill on delivery validate"
    )

    is_auto_send_bill = fields.Boolean(
        string="Auto Send Bill",
        config_parameter="automatic_invoice_and_post.is_auto_send_bill",
        help="Enable to send vendor bill to supplier on delivery validate"
    )
```

Gambar 3.21. Kode *Python Auto Bill Configs*

Setelah pengaturan berhasil ditambahkan, tahap selanjutnya adalah melakukan

modifikasi pada metode *button validate* milik model *stock.picking*. Modifikasi ini memungkinkan sistem untuk mendeteksi saat pengguna memvalidasi proses penerimaan barang (*delivery order*) yang terhubung dengan *Purchase Order*. Jika fitur *Auto Bill* diaktifkan dan kondisi transaksi sesuai, maka sistem akan secara otomatis membuat *vendor bill* tanpa perlu proses manual dari pengguna. Selain itu, sistem juga langsung melakukan posting terhadap *bill* yang dibuat tersebut. Berikut tampilannya:

The screenshot displays the Odoo Vendor Bill interface. At the top, there are navigation tabs: Purchase, Orders, Products, Reporting, and Configuration. The main header shows the bill number 'BILL/2025/05/0002' and the vendor 'Administrator'. Below this, there are fields for Bill Date (05/22/2025), Accounting Date (05/22/2025), Payment Reference (Use Bill Reference), Recipient Bank, Due Date (05/22/2025), and Journal (Vendor Bills). The bill is currently in 'Draft' status. The main body of the bill contains a table with columns: Product, Account, Quantity, Price, Taxes, and Amount. The table lists one item: 'Apple' with a quantity of 1.00, a price of 1.00, a tax of 11%, and a total amount of Rp 1.00. At the bottom right, the 'Untaxed Amount' is Rp 1.00 and the 'Taxes' are Rp 0.11.

Product	Account	Quantity	Price	Taxes	Amount
Apple	51000010 Cost of Goods Sold	1.00	1.00	11%	Rp 1.00

Untaxed Amount: Rp 1.00
Taxes: Rp 0.11

Gambar 3.22. Tampilan *Auto Bill System*

Langkah terakhir dalam pengembangan adalah menambahkan logika pengiriman otomatis (*Auto Send Bill*) setelah bill berhasil dibuat dan diposting. Jika fitur ini aktif dan data kontak vendor memiliki alamat email, maka sistem secara otomatis mengirimkan tagihan ke email vendor menggunakan template email default dari modul akuntansi. Dengan demikian, keseluruhan proses dari penerimaan barang hingga pengiriman tagihan ke supplier dapat dilakukan secara otomatis dan efisien.

Gambar berikut menggambarkan alur proses sebelum diterapkannya modul *Hide Cash*, di mana proses input kas masih dilakukan secara manual:

```

        'automatic_invoice_and_post.is_create_bill_delivery_validate')
auto_send_bill = self.env['ir.config_parameter'].sudo().get_param(
    'automatic_invoice_and_post.is_auto_send_bill')

if self.picking_type_id.code == 'outgoing':
    if auto_validate_invoice:
        if self.sale_id and not self.sale_id.invoice_ids:
            invoice_created = self.sale_id._create_invoices(
                self.sale_id) if self.sale_id else False
        if invoice_created:
            invoice_created.action_post()
            if auto_send_invoice and invoice_created.partner_id.email:
                template = self.env.ref(
                    'account.email_template_edi_invoice').sudo()
                template.send_mail(invoice_created.id, force_send=True,
                                   email_values={
                                       'email_to': invoice_created.partner_id.email
                                   })

    if auto_validate_bill:
        if self.purchase_id and not self.purchase_id.invoice_ids:
            bill_created = self.purchase_id.action_create_invoice()
            if isinstance(bill_created, dict) and bill_created.get('res_id'):
                bill_created = self.env['account.move'].browse(bill_created['res_id'])
            bill_created.invoice_date = fields.Date.today()
            bill_created.action_post()
            if auto_send_bill and bill_created.partner_id.email:
                template = self.env.ref('account.email_template_edi_invoice').sudo()
                template.send_mail(bill_created.id, force_send=True, email_values={
                    'email_to': bill_created.partner_id.email
                })

```

Gambar 3.23. Kode Python Auto Bill Models

3.2.4 Pengembangan Custom Hide Cash Method

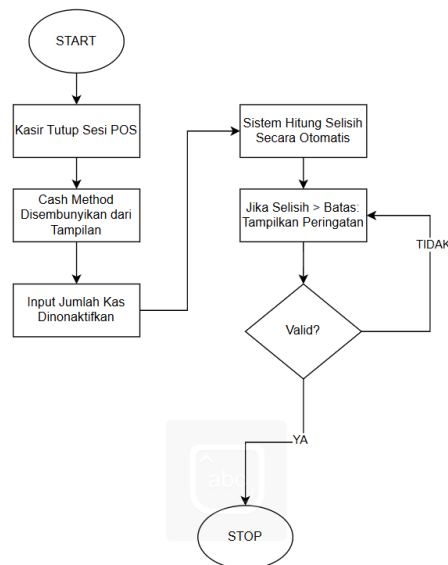
Modul ini dikembangkan sebagai langkah preventif untuk menghindari terjadinya kecurangan atau manipulasi data pada saat proses *closing register* di sistem Point of Sale (POS) Odoo. Secara default, Odoo memungkinkan kasir untuk melihat dan menginput jumlah kas tunai saat menutup sesi POS, namun hal ini dapat disalahgunakan dengan mengubah nominal secara manual agar terlihat sesuai, padahal terdapat selisih yang signifikan antara uang yang seharusnya diterima (expected) dan jumlah yang dihitung (counted).



Gambar 3.24. Diagram *Before Hide Cash Development*

Untuk mengatasi hal tersebut, modul ini menambahkan logika tambahan pada *ClosePosPopup* yang akan menampilkan peringatan jika selisih nominal melebihi batas toleransi yang telah ditentukan, sehingga sistem dapat mendeteksi ketidaksesuaian secara otomatis.

Gambar berikut menampilkan alur proses setelah pengembangan modul Hide Cash diterapkan:



Gambar 3.25. Diagram *After Hide Cash Development*

Sebelum dilakukan pengembangan, tampilan default sistem POS Odoo menampilkan metode pembayaran tunai secara terbuka. Gambar berikut menunjukkan tampilan tersebut:

Closing Register 1 orders: Rp 16,650.00

Method	Amount
Cash	Rp 16,650.00
Opening	Rp 0.00
Payments in Cash	Rp 16,650.00
Cash In / Out	+ Rp 0.00
Counted	Rp 0.00
Difference	Rp -16,650.00
Card	Rp 0.00
Counted	Rp 0.00
Difference	Rp 0.00
Customer Account	Rp 0.00
Counted	Rp 0.00
Difference	Rp 0.00

Cash Count

0

Closing note

Add a closing note...

Close Register **Discard** **Cash In/Out** **Daily Sale**

Gambar 3.26. Tampilan *Cash Payment*

Gambar 3.26 menunjukkan tampilan default sistem POS Odoo sebelum modul diterapkan. Dalam tampilan ini, metode pembayaran tunai (*Cash Payment Method*) ditampilkan secara jelas dan kasir memiliki akses untuk menginput jumlah uang yang dihitung secara manual. Hal ini berisiko dimanipulasi agar terlihat tidak ada selisih meskipun sebenarnya ada kekurangan atau kelebihan kas. Kode ini terdiri dari dua bagian utama, yaitu modifikasi pada *JavaScript frontend* (Odoo OWL) dan penyesuaian pada *CSS*. Gambar di bawah ini menunjukkan bagian kode JavaScript yang dimodifikasi untuk menambahkan logika peringatan pada proses penutupan sesi kasir:

```

patch(ClosePosPopup.prototype, {
  setup() {
    super.setup();
    this.state.differenceLabel = "";
    this.computeDifferenceLabel();
  },

  computeDifferenceLabel() {
    const paymentId = this.props.default_cash_details.id;
    const countedStr = this.state.payments[paymentId]?.counted?.replace(/^[^0-9.-]+/g, '');
    if (!this.env.utils.isValidFloat(countedStr)) {
      this.state.differenceLabel = "";
      return;
    }

    const counted = parseFloat(countedStr);
    const expected = this.props.default_cash_details.amount;
    const maxDiff = 2000;

    if (counted > expected + maxDiff) {
      this.state.differenceLabel = "Nominal Closing Tidak Tepat";
    } else if (counted < expected - maxDiff) {
      this.state.differenceLabel = "Nominal Closing Tidak Tepat";
    } else {
      this.state.differenceLabel = "";
    }
  },

  setManualCashInput(amount) {
    if (this.env.utils.isValidFloat(amount) && this.moneyDetails) {
      this.state.notes = "";
      this.moneyDetails = null;
    }
  }
});

```

Gambar 3.27. Kode Javascript *Hide Cash*

Modifikasi dilakukan dengan melakukan *patching* pada komponen ClosePosPopup milik POS Odoo. Berikut adalah fitur yang ditambahkan:

- **setup()** – inisialisasi state baru bernama differenceLabel untuk menyimpan pesan peringatan jika selisih nominal terlalu besar.
- **computeDifferenceLabel()** – fungsi untuk menghitung apakah nilai counted cash berada dalam batas wajar jika dibandingkan dengan nilai expected. Jika selisih melebihi batas toleransi (dalam hal ini Rp 2.000), maka label peringatan akan ditampilkan.
- **setManualCashInput()** – fungsi ini digunakan untuk memantau perubahan manual pada nilai kas dan memanggil kembali computeDifferenceLabel() untuk memperbarui status peringatan.
- **autoFillCashCount()** – fungsi yang akan secara otomatis mengisi nilai kas berdasarkan data sistem, dan juga memperbarui peringatan apabila ada perbedaan.

Selanjutnya, modifikasi CSS digunakan untuk menyembunyikan elemen input kas secara visual dari antarmuka pengguna. Gambar berikut memperlihatkan kode CSS yang digunakan:

```
1  .payment-methods-overview > div:first-child {
2  |     display: none !important;
3  | }
4
5  .modal-body > div.w-100.mb-3:first-of-type {
6  |     display: none !important;
7  | }
8  |
```

Gambar 3.28. Kode CSS *Hide Cash*

Untuk menyembunyikan kolom input kas secara visual dari antarmuka pengguna, digunakan dua aturan CSS:

- .payment-methods-overview pada div:first-child – menyembunyikan tampilan kolom kas di sidebar.
- .modal-body pada div.w-100.mb-3:first-of-type – menyembunyikan tampilan kas di jendela popup penutupan sesi.

Kedua aturan ini menggunakan `display: none !important` untuk memastikan elemen benar-benar tidak ditampilkan, meskipun ada gaya CSS lain yang aktif.

Closing Register 2 orders: Rp 33,300.00

Card	Rp 0.00
Counted	Rp 0.00
Difference	Rp 0.00
Customer Account	Rp 0.00
Counted	Rp 0.00
Difference	Rp 0.00

Cash Count

0 [Close] [Icon]

Closing note

Add a closing note...

Nominal Closing Tidak Tepat

Close Register Discard Cash In/Out Daily Sale

Gambar 3.29. Tampilan *Hide Cash Payment*

Gambar 3.29 menampilkan hasil akhir setelah modul *Hide Cash Method* diimplementasikan. Pada tampilan ini, metode pembayaran tunai telah disembunyikan dari antarmuka kasir saat sesi ditutup. Akses untuk menginput jumlah kas secara manual telah dihilangkan, sehingga kasir tidak dapat memanipulasi nominal dan hanya dapat melanjutkan proses penutupan jika data kas sudah sesuai.

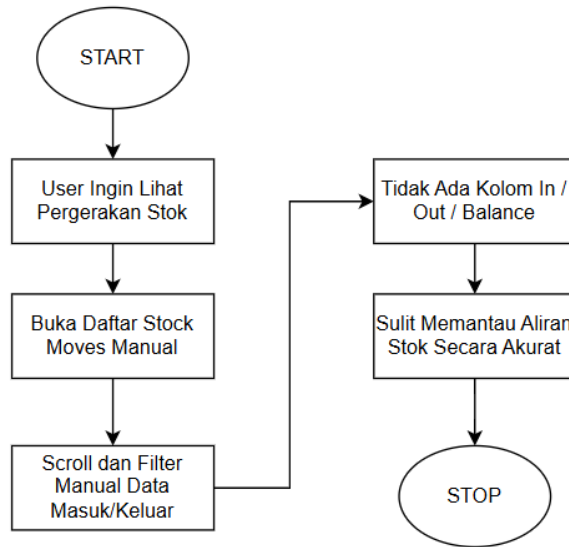
Dengan adanya perubahan ini, perusahaan dapat meningkatkan transparansi dan akurasi dalam pencatatan kas di akhir sesi, serta mengurangi risiko terjadinya kecurangan atau kesalahan input dari pihak kasir.

3.2.5 Pengembangan *Custom Stock Moves*

Pada pengembangan ini, dibuat sebuah modul kustom untuk membantu perusahaan dalam memantau pergerakan stok barang yang masuk dan keluar di sistem *Inventory* Odoo. Fitur ini penting bagi perusahaan dalam menjaga akurasi stok serta melakukan evaluasi terhadap distribusi produk di gudang.

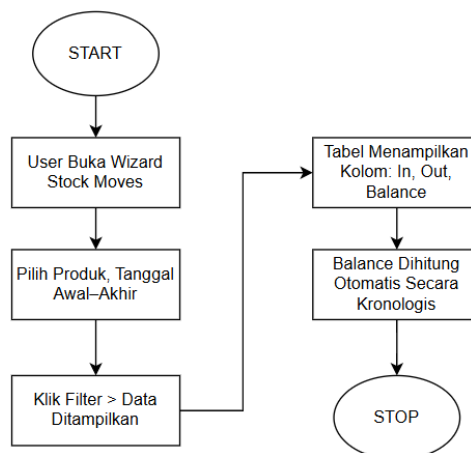
Gambar berikut menunjukkan alur proses sebelum adanya pengembangan fitur

ini, di mana pemantauan pergerakan stok masih terbatas:



Gambar 3.30. Diagram *Before Stock Moves Development*

Setelah modul dikembangkan, proses pemantauan stok menjadi lebih sistematis dan informatif, sebagaimana ditampilkan pada diagram berikut:



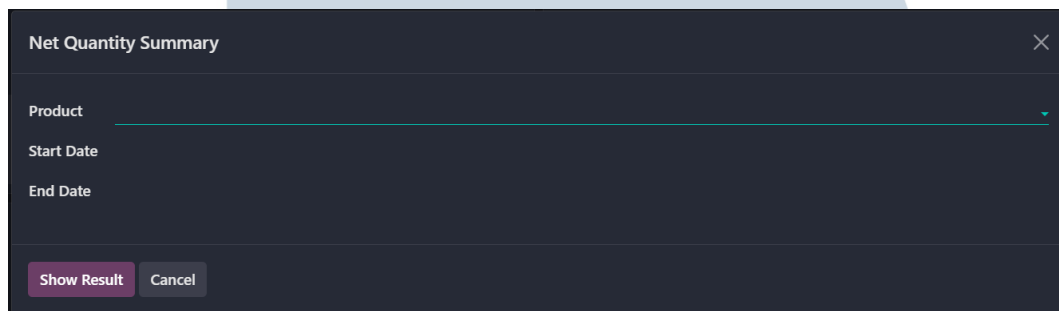
Gambar 3.31. Diagram *After Stock Moves Development*

Untuk memudahkan pengguna dalam memantau produk tertentu dalam rentang waktu yang spesifik, disediakan sebuah *Wizard* bernama

`stock.move.net.qty.wizard`. Wizard ini memungkinkan pengguna memilih:

- Produk tertentu (`product_id`) — opsional.
- Tanggal mulai (`date_from`) dan tanggal akhir (`date_to`).

Berikut adalah tampilan dari form filter yang digunakan untuk memilih kriteria tersebut:



Gambar 3.32. Tampilan *Stock Moves Filter*

Wizard akan memfilter data `stock.move` berdasarkan kriteria tersebut, kemudian menampilkan hasilnya dalam bentuk daftar *Stock Moves* yang sudah difilter dan dilengkapi dengan informasi masuk, keluar, dan saldo pergerakan barang.

```
class StockMoveNetQtyWizard(models.TransientModel):
    _name = 'stock.move.net.qty.wizard'
    _description = 'Wizard to show net stock movement'

    product_id = fields.Many2one('product.product', string="Product", required=False)
    date_from = fields.Date(string="Start Date")
    date_to = fields.Date(string="End Date")

    def _default_date_from(self):
        today = fields.Date.context_today(self)
        return datetime.combine(today, time.min)

    def _default_date_to(self):
        today = fields.Date.context_today(self)
        return datetime.combine(today, time.max)

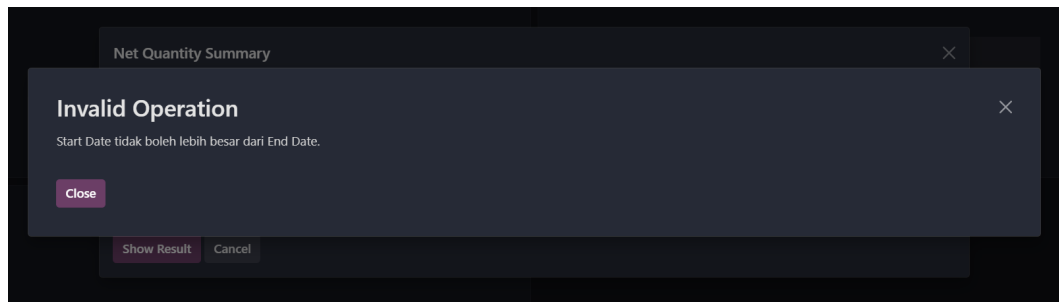
    def action_show_result(self):
        if self.date_from and self.date_to and self.date_from > self.date_to:
            raise UserError("Start Date tidak boleh lebih besar dari End Date.")

        domain = [('state', '=', 'done')]
        if self.product_id:
            domain.append(('product_id', '=', self.product_id.id))
        if self.date_from:
            domain.append(('date', '>=', self.date_from))
        if self.date_to:
            domain.append(('date', '<=', self.date_to))

        moves = self.env['stock.move'].search(domain)
```

Gambar 3.33. Kode Python Wizard Stock Moves

Apabila pengguna memasukkan tanggal mulai yang lebih besar daripada tanggal akhir, sistem akan memberikan peringatan validasi secara otomatis.



Gambar 3.34. Tampilan *Invalid Operation*

Berikut adalah logika utama dari sistem pemantauan ini:

- Jika barang masuk ke lokasi internal dari luar (misal dari suppliers, adjustments), maka dianggap sebagai **stok masuk**.
- Jika barang keluar dari lokasi internal ke lokasi luar (misal ke customers), maka dianggap sebagai **stok keluar**.
- Jika terjadi perpindahan antar lokasi internal (misal antar gudang), maka **tidak dihitung** dalam akumulasi net quantity.

Modul menambahkan tiga kolom tambahan pada tampilan *Stock Moves* yaitu:

- **In:** Menunjukkan jumlah barang yang masuk ke gudang (lokasi bertipe *internal*).
- **Out:** Menunjukkan jumlah barang yang keluar dari gudang (lokasi bertipe *internal* ke lokasi luar).
- **Balance:** Merupakan akumulasi bersih dari pergerakan masuk dan keluar barang, dihitung berdasarkan urutan tanggal.

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A

```

class StockMove(models.Model):
    _inherit = 'stock.move'

    balance = fields.Float(string='Balance', compute='_compute_balance', store=False)
    in_qty = fields.Float(string='In', compute='_compute_in_out_qty', store=False)
    out_qty = fields.Float(string='Out', compute='_compute_in_out_qty', store=False)

    @api.depends('in_qty', 'out_qty')
    def _compute_balance(self):

        sorted_self = self.sorted(lambda r: (r.date, r.id))

        running_balance = {}

        for record in sorted_self:
            product_id = record.product_id.id
            if product_id not in running_balance:
                running_balance[product_id] = 0.0

            in_qty = record.in_qty or 0.0
            out_qty = record.out_qty or 0.0
            running_balance[product_id] += in_qty - out_qty

            record.balance = running_balance[product_id]

    def _compute_in_out_qty(self):
        for move in self:
            move.in_qty = 0.0
            move.out_qty = 0.0

            if move.state != 'done':
                continue

```

Gambar 3.35. Kode Python Stock Moves

Kolom Balance dihitung secara dinamis menggunakan fungsi compute tanpa disimpan ke database (*store=False*), sehingga selalu diperbarui berdasarkan data terbaru. Ini berguna untuk memberikan informasi saldo berjalan (*running balance*) dari suatu produk secara kronologis.

Source	Destination	Product	Demand	Unit of Me...	Status	Date Scheduled	Reference	Source Location	In	Out	Balance
		LIST PLANK KALSI 20CM X 3MTR ...	1.00	Units	Done	05/09/2025 11:47:22	DC/POS/00002	DC/Stock	0.00	1.00	-1.00
		LIST PLANK KALSI 20CM X 3MTR ...	1.00	Units	Done	05/09/2025 11:48:58	DC/POS/00003	Partners/Customers	1.00	0.00	0.00
		LIST PLANK KALSI 20CM X 3MTR ...	15.00	Units	Done	05/24/2025 10:27:46	Product Quantity U...	Virtual Locations/Inventory adju...	15.00	0.00	15.00
		LIST PLANK KALSI 20CM X 3MTR ...	3.00	Units	Done	05/24/2025 11:37:36	Product Quantity U...	Virtual Locations/Inventory adju...	3.00	0.00	18.00

Gambar 3.36. Tampilan Stock Moves dengan Kolom In, Out, dan Balance

Gambar 3.36 menunjukkan hasil akhir dari fitur yang telah diterapkan. Kolom-kolom tambahan ini memberikan visibilitas yang lebih baik terhadap pergerakan dan posisi stok dari suatu produk secara waktu nyata.

Dengan pendekatan ini, perusahaan dapat dengan mudah mengidentifikasi total barang yang benar-benar masuk atau keluar dari sistem gudang, bukan hanya berpindah antar lokasi internal.

Fitur ini sangat berguna dalam:

1. Menyediakan visibilitas pergerakan stok secara kronologis dan detail.
2. Membantu tim inventory dalam melakukan audit dan pengecekan stok.
3. Menghindari kesalahan pencatatan stok akibat perpindahan internal yang tidak relevan terhadap stok aktual.
4. Menjadi dasar laporan untuk mengambil keputusan logistik dan operasional.

3.2.6 Penambahan *Customer Corporate* dan Fitur Pencarian pada *Sales Order*

Pada pengembangan ini, ditambahkan beberapa elemen kustomisasi sederhana namun penting pada modul *Sales* di Odoo. Tujuannya adalah untuk mendukung kebutuhan pencatatan dan pencarian data customer korporat secara lebih fleksibel dan terstruktur oleh tim penjualan PT Klik Semangat Indonesia.

Pengembangan terdiri dari tiga bagian utama:

- Penambahan field `customer_corporate` pada form *Sales Order*.
- Penambahan menu baru bernama *Others Customers* untuk mengelola entitas pelanggan khusus.
- Penambahan fitur pencarian berdasarkan nomor *Purchase Order* (PO) pada tampilan pencarian *Sales Order*.

Field baru bernama `customer_corporate` ditambahkan tepat setelah pemilihan *Customer* (`partner_id`) pada form *Sales Order*. Field ini bertipe teks dan digunakan untuk mencatat nama customer korporat tambahan yang tidak tersedia dalam daftar customer utama.

New

Customer

Purchase Order Number

Customer Corporate

Expiration? 07/19/2025

Quotation Date? 06/19/2025 10:27:38

Payment Terms Immediate

Order Lines Optional Products Other Info Customer Signature

Product	Quantity	Unit Price	Taxes	Amount
Add a product	Add a section	Add a note	Catalog	

Terms and conditions...

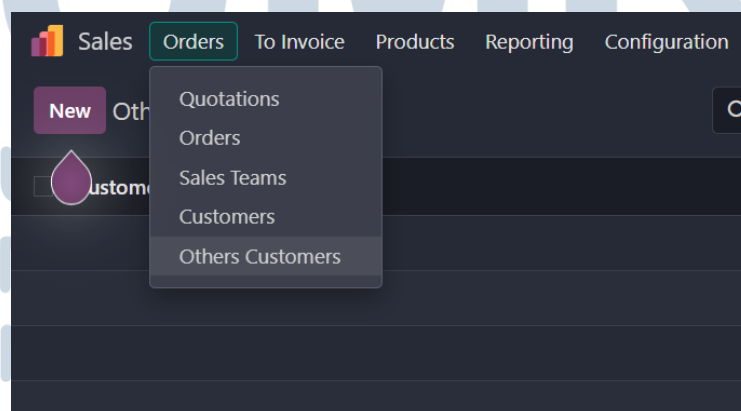
Untaxed Amount: Rp 0.00
Total: Rp 0.00

Gambar 3.37. Tampilan *Purchase Order Number* dan *Customer Corporate Field*

Selain itu, juga ditambahkan field `po_number` yang digunakan untuk mencatat nomor *Purchase Order* dari customer. Kedua field ini bertujuan untuk mendokumentasikan data administratif dengan lebih lengkap.

Untuk mengakomodasi kebutuhan pengelolaan data customer non-standar (pelanggan luar sistem utama), ditambahkan sebuah model baru bernama `sale.customer.delight` yang memiliki field `name`. Data dari model ini dapat diakses melalui menu baru:

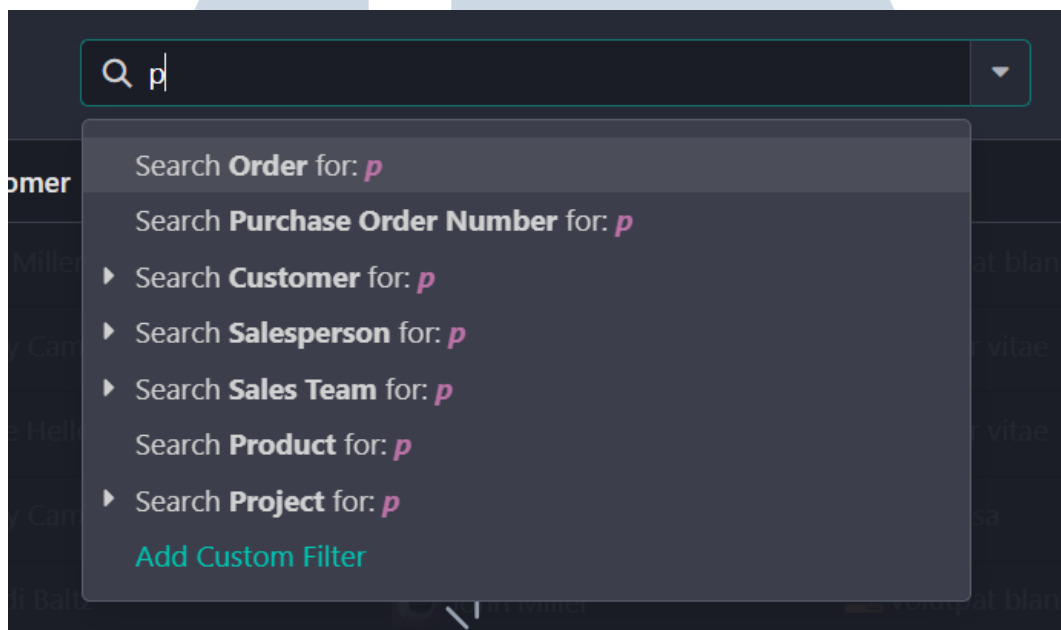
- Nama Menu: **Others Customers**
- Lokasi Menu: Di bawah *Sales pada Orders*
- Tipe Tampilan: *List* dan Form, *editable* langsung dari *list*



Gambar 3.38. Tampilan *Submenu Others Customers*

Menu ini memudahkan tim sales dalam mendokumentasikan pelanggan yang tidak memiliki profil lengkap di sistem res.partner.

Untuk meningkatkan efisiensi pencarian data, field po_number juga ditambahkan pada filter pencarian (*search view*) Sales Order. Dengan begitu, pengguna dapat mencari data transaksi berdasarkan nomor PO yang diberikan oleh customer, tanpa harus mencocokkan nama atau referensi internal.



Gambar 3.39. Tampilan *Search*

Dengan adanya fitur-fitur ini, perusahaan mendapatkan keuntungan sebagai berikut:

1. Meningkatkan fleksibilitas pencatatan data pelanggan non-standar.
2. Mempermudah pelacakan transaksi berdasarkan nomor PO.
3. Memberikan visibilitas yang lebih tinggi terhadap transaksi yang melibatkan customer corporate.
4. Mendukung keakuratan dokumentasi administratif dalam proses penjualan.

3.3 Uraian Pelaksanaan Magang

Berikut deskripsi tugas yang dilakukan oleh penulis selama magang yang berlangsung selama 4 bulan

Tabel 3.1. Rangkuman Tugas Mingguan dan Tanggal Pelaksanaannya

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
1	Pada minggu pertama magang, penulis mempelajari ERP Odoo 15, mengembangkan modul <i>Inventory</i> , memahami struktur <i>Addons</i> , serta membuat model pivot dan fitur pencarian terkait <i>sales</i> dan <i>purchase</i> .	03 Februari 2025	7 Februari 2025
2	Minggu ke-2 fokus pada pengembangan modul POS, revisi perhitungan komisi, <i>filter wizard</i> , <i>debugging</i> 30 modul, serta pengembangan dan perbaikan modul PO, SO, dan <i>price approval</i> .	10 Februari 2025	14 Februari 2025
3	Minggu ke-3 fokus pada penyempurnaan modul <i>price approval</i> , penambahan fitur kontrol akses, integrasi boolean global, <i>form</i> terpadu, serta revisi alur persetujuan dan catatan alasan.	17 Februari 2025	21 Februari 2025

Tabel 3.1 - lanjutan

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
4	Minggu ke-4 fokus finalisasi modul <i>price approval</i> , <i>debugging</i> modul <i>web hide button</i> , <i>product approval</i> , perbaikan logika <i>reseller commission</i> , serta pengembangan <i>auto bill</i> dan kustomisasi <i>parreto</i> .	24 Februari 2025	28 Februari 2025
5	Minggu ke-5 berfokus pada kustomisasi data pelanggan korporat, penambahan field pencarian PO, serta pengembangan dan penyempurnaan perhitungan margin produk per vendor.	3 Maret 2025	7 Maret 2025
6	Minggu ke-6 mengembangkan modul margin dijadikan menu terpisah dengan filter divisi produk. Mulai migrasi ke Odoo 18, instalasi modul dasar, dan penyesuaian struktur <i>auto bill</i> dan <i>invoice</i> .	10 Maret 2025	14 Maret 2025

Tabel 3.1 - lanjutan

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
7	Minggu ke-7 fokus migrasi modul kustom ke Odoo 18, termasuk penyesuaian struktur, fitur baru, serta pembaruan fungsi dan dependensi dari Odoo 15.	17 Maret 2025	21 Maret 2025
8	Minggu ke-8 melanjutkan migrasi modul ke Odoo 18, <i>debugging automatic invoice</i> , serta mengembangkan fitur auto-fill default contact saat menambahkan kontak baru.	24 Maret 2025	28 Maret 2025
9	Minggu ke-9 fokus <i>debugging</i> fitur <i>automatic invoice</i> dan <i>bill</i> , serta pengembangan modul kustom dengan <i>field member type</i> dan <i>media acquisition</i> dari submenu khusus.	7 April 2025	11 April 2025

Tabel 3.1 - lanjutan

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
10	Minggu ke-10 fokus penambahan <i>custom field</i> pada produk, <i>Purchase Order</i> , dan <i>Sales Order</i> , serta integrasi <i>field related</i> untuk otomatisasi data supplier saat transaksi.	14 April 2025	18 April 2025
11	Minggu ke-11 fokus penyederhanaan tampilan <i>mobile delivery order</i> , pengembangan POS untuk sembunyikan pembayaran <i>cash</i> , serta <i>alert</i> selisih kas saat <i>close register</i> .	21 April 2025	25 April 2025
12	Minggu ke-12 fokus integrasi jenis penjualan dan pembelian dalam auto PO-SO <i>multicompanies</i> , serta migrasi modul POS untuk otomatisasi pembuatan sales order.	28 April 2025	2 Mei 2025

Tabel 3.1 - lanjutan

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
13	Minggu ke-13 fokus menambahkan <i>field extra vendor</i> pada <i>replenishment</i> dan melanjutkan migrasi modul POS untuk otomatisasi sales order saat transaksi.	5 Mei 2025	9 Mei 2025
14	Minggu ke-14 fokus pembuatan custom vendor barcode berdasarkan inisial vendor, serta migrasi modul <i>price approval</i> dari Odoo 15 ke Odoo 18.	12 Mei 2025	16 Mei 2025
15	Minggu ke-15 fokus <i>debugging</i> modul <i>ksi purchase stock</i> dan pengembangan modul <i>stock card</i> pada Odoo 18 untuk pencatatan pergerakan stok barang.	19 Mei 2025	23 Mei 2025

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A

Tabel 3.1 - lanjutan

Minggu	Tugas Yang Dilakukan	Tanggal Mulai	Tanggal Selesai
16	Minggu ke-16 fokus membuat modul popup validasi POS dengan input wajib di Odoo 18, serta <i>debugging</i> modul <i>stock card</i> untuk perbaikan pencatatan stok.	26 Mei 2025	31 Mei 2025

3.4 Pengujian

Pengujian dilakukan untuk memastikan bahwa fitur-fitur yang dikembangkan dalam modul Odoo dapat berjalan sesuai dengan fungsionalitas yang diharapkan. Pengujian ini dilakukan secara lokal melalui environment Odoo di localhost selama proses pengembangan.

Metode pengujian yang digunakan adalah *black box testing*, di mana pengujian dilakukan dengan cara memberikan input dan mengamati output tanpa mengetahui struktur internal kode program. Metode ini sesuai karena fokus pengujian adalah pada fungsionalitas yang terlihat di antarmuka pengguna Odoo, seperti alur validasi, perhitungan otomatis, dan tampilan data.

3.4.1 Pengujian Fitur *Reseller Commission*

Pengujian dilakukan untuk memastikan bahwa fitur *Reseller Commission* dapat berjalan sesuai dengan fungsionalitas yang dirancang. Pengujian ini dilakukan secara lokal melalui server Odoo di lingkungan *localhost*. Metode pengujian yang digunakan adalah *black box testing*, di mana pengujian dilakukan dengan memberikan input melalui antarmuka sistem dan mengamati hasil output yang ditampilkan, tanpa melihat langsung ke struktur internal kode program.

Langkah pengujian pertama adalah mengakses menu *Reseller Commission* dari dashboard Odoo. Saat menu dibuka, sistem secara otomatis menampilkan wizard berupa form filter yang terdiri dari field *Company*, *Start Date*, dan *End Date*. Form

tersebut berhasil ditampilkan dengan baik dan responsif sesuai desain.

Selanjutnya dilakukan pengujian terhadap fungsi penyaringan data. Pengguna mengisi rentang tanggal serta memilih vendor, lalu menekan tombol "*Apply Filter*". Hasilnya, sistem hanya menampilkan data transaksi Point of Sale yang sesuai dengan filter yang diberikan. Hal ini menunjukkan bahwa logika penyaringan berjalan dengan benar.

Pengujian juga dilakukan terhadap proses perhitungan otomatis komisi. Saat transaksi dilakukan di POS dengan produk yang memiliki nilai komisi, sistem secara otomatis menghitung nilai total komisi berdasarkan presentase komisi produk dikalikan dengan subtotal transaksi. Nilai yang ditampilkan pada field *Total Commission* sesuai dengan ekspektasi.

Fitur tampilan dalam bentuk *Pivot* dan *Tree View* turut diuji, dan hasilnya sistem menampilkan data komisi secara terstruktur serta mendukung pengelompokan data sesuai kebutuhan pengguna. Selain itu, sistem juga telah berhasil memunculkan validasi ketika pengguna mencoba menjalankan filter tanpa mengisi field wajib, dengan menampilkan peringatan yang sesuai.

Secara keseluruhan, seluruh fungsionalitas dari fitur *Reseller Commission* telah berhasil diuji dan berjalan dengan baik. Sistem mampu melakukan perhitungan otomatis, menyaring data transaksi dengan tepat, serta menyajikan data dalam tampilan yang informatif. Dengan demikian, fitur ini berhasil menggantikan proses manual yang sebelumnya digunakan, dan meningkatkan efisiensi dalam pengelolaan komisi reseller.

3.4.2 Pengujian Fitur *Price Approval*

Pengujian dilakukan untuk memastikan bahwa modul *Price Approval* berfungsi sebagaimana mestinya dan mampu mengelola proses pengajuan serta persetujuan perubahan harga secara terstruktur. Pengujian dilakukan menggunakan metode *black box testing*, dengan cara menguji sistem dari sisi antarmuka pengguna dan memverifikasi bahwa setiap input menghasilkan output yang sesuai tanpa melihat kode sumber secara langsung.

Pengujian diawali dengan mengakses menu *Price Approval* untuk melakukan simulasi pengajuan harga. Pada tahap ini, pengguna mencoba mengisi form pengajuan harga dengan informasi yang diperlukan, seperti pemohon, alasan perubahan harga, serta harga baru yang diajukan. Sistem berhasil menampilkan form dengan field yang lengkap dan dapat diisi dengan benar. Setelah form

disubmit, data masuk dalam status *draft*, sesuai alur sistem yang diharapkan.

Langkah berikutnya adalah pengujian fitur validasi akses. Pengguna *non-price manager* tidak dapat menyetujui pengajuan harga atau melakukan perubahan data penting. Sistem secara tepat membatasi akses dan hanya memberikan hak validasi kepada pengguna dengan status *price manager*, sesuai dengan konfigurasi yang ditentukan oleh administrator. Validasi ini berhasil memastikan keamanan dan konsistensi data.

Setelah dilakukan persetujuan oleh admin, sistem secara otomatis memperbarui harga produk di vendor terkait. Selain itu, log aktivitas yang menunjukkan waktu dan pengguna yang menyetujui perubahan juga berhasil tercatat di sistem. Fitur pencatatan log ini diuji dengan membuat beberapa pengajuan dan mencatat apakah informasi log muncul secara otomatis — hasilnya berhasil dan sesuai ekspektasi.

Pengujian juga dilakukan terhadap fitur *tree view* yang menampilkan seluruh riwayat pengajuan harga. Tampilan data mencakup status (*draft*, *waiting approval*, *approved*), tanggal, vendor, dan informasi harga, semuanya tampil dengan rapi. Riwayat ini dapat digunakan oleh pihak manajemen untuk melakukan evaluasi terhadap perubahan harga yang telah terjadi.

Secara keseluruhan, seluruh fitur utama dari modul *Price Approval* telah diuji dan berfungsi dengan baik. Sistem berhasil menangani pengajuan, validasi, persetujuan, dan pencatatan log secara otomatis, serta mampu memberikan kontrol akses yang sesuai. Dengan adanya modul ini, proses persetujuan harga yang sebelumnya dilakukan secara manual menjadi lebih efisien, transparan, dan terdokumentasi dengan baik.

3.4.3 Pengujian Fitur *Custom Auto Invoice and Bill*

Pengujian dilakukan untuk memastikan bahwa modul *Auto Invoice and Bill* dapat berjalan sesuai fungsinya, yaitu mengotomatisasi proses pembuatan dan pengiriman tagihan vendor secara otomatis ketika validasi pengiriman barang dilakukan. Pengujian dilakukan secara lokal di lingkungan *localhost* dengan pendekatan *black box testing*, di mana pengujian dilakukan melalui antarmuka pengguna tanpa melihat struktur internal kode program.

Langkah pertama adalah mengakses menu *Settings* pada modul *Accounting*, kemudian mengaktifkan opsi *Auto Bill* melalui checkbox yang telah dikembangkan. Setelah pengaturan disimpan, dilakukan simulasi proses pengadaan barang melalui

Purchase Order, yang kemudian dilanjutkan dengan proses penerimaan barang melalui *Delivery Order*.

Saat pengguna menekan tombol *Validate* pada halaman penerimaan barang (*stock picking*), sistem berhasil mendeteksi bahwa fitur *Auto Bill* dalam kondisi aktif. Hasilnya, tagihan vendor (*vendor bill*) langsung dibuat secara otomatis tanpa intervensi manual, dan statusnya langsung terposting sebagaimana yang diharapkan. Ini membuktikan bahwa modifikasi pada metode `button_validate` telah berjalan dengan baik.

Pengujian selanjutnya dilakukan untuk fitur *Auto Send Bill*. Setelah bill berhasil dibuat dan diposting, sistem secara otomatis mengirimkan email tagihan ke alamat email vendor yang bersangkutan, menggunakan template email dari modul akuntansi Odoo. Pengujian berhasil ketika email masuk diterima oleh akun penguji yang telah ditentukan dalam kontak vendor.

Pengujian juga dilakukan untuk memastikan bahwa sistem tidak membuat bill ganda apabila tombol validasi ditekan ulang, dan bahwa fitur tidak aktif jika checkbox pengaturan dalam kondisi tidak dicentang. Kedua skenario ini berhasil menunjukkan perilaku sistem yang tepat, yaitu tidak ada duplikasi data dan tidak ada aksi otomatis saat fitur dimatikan.

Secara keseluruhan, hasil pengujian menunjukkan bahwa fitur *Auto Invoice and Bill* berjalan sesuai fungsinya. Proses otomatisasi pembuatan dan pengiriman tagihan vendor berhasil mengurangi langkah manual, meminimalkan risiko human error, serta meningkatkan efisiensi dalam proses pembelian dan pengelolaan tagihan di perusahaan.

3.4.4 Pengujian Fitur *Custom Hide Cash Method*

Pengujian dilakukan untuk memastikan bahwa fitur *Hide Cash Method* pada sistem Point of Sale (POS) Odoo dapat berfungsi sebagaimana mestinya dalam membatasi akses kasir terhadap input manual kas tunai dan menampilkan peringatan saat terjadi selisih kas yang signifikan. Metode pengujian yang digunakan adalah *black box testing*, dengan mengamati perilaku sistem berdasarkan input pengguna tanpa meninjau langsung struktur kode internal.

Pengujian pertama dilakukan terhadap tampilan antarmuka POS sebelum dan sesudah modul diterapkan. Sebelum modul diaktifkan, sistem menampilkan kolom input jumlah kas secara terbuka saat proses penutupan sesi, memungkinkan kasir untuk mengubah nilai secara manual. Setelah modul diterapkan, elemen input kas

berhasil disembunyikan dari antarmuka, baik pada sidebar maupun dalam jendela pop-up penutupan sesi. Hal ini menunjukkan bahwa aturan CSS berhasil dijalankan dan sistem tidak lagi memberikan akses langsung kepada kasir untuk memanipulasi nilai kas.

Pengujian dilanjutkan dengan skenario selisih kas antara nilai yang diharapkan (*expected*) dan yang dihitung (*counted*). Dalam simulasi, ketika selisih kas melebihi ambang batas yang telah ditentukan (Rp 2.000), sistem secara otomatis menampilkan peringatan pada pop-up penutupan sesi. Sebaliknya, apabila selisih berada dalam rentang wajar, sistem tidak memberikan notifikasi khusus. Hal ini menunjukkan bahwa fungsi `computeDifferenceLabel()` yang ditanamkan melalui komponen `ClosePosPopup` berhasil berjalan sebagaimana mestinya.

Selain itu, pengujian juga dilakukan terhadap fungsi `autoFillCashCount()`, yang secara otomatis mengisi nilai kas berdasarkan jumlah yang tercatat di sistem. Pengujian menunjukkan bahwa nilai kas otomatis tersebut tetap ditampilkan meskipun kolom input disembunyikan, dan peringatan muncul hanya ketika terdapat perbedaan yang signifikan. Dengan demikian, kasir tidak dapat melakukan pengisian ulang nilai kas secara manual, dan sistem tetap menjaga akurasi data.

Secara keseluruhan, hasil pengujian menunjukkan bahwa fitur *Hide Cash Method* telah berjalan dengan baik sesuai dengan tujuannya. Sistem mampu menyembunyikan akses kasir terhadap input tunai dan secara otomatis memberikan peringatan apabila terdapat ketidaksesuaian kas. Modul ini efektif dalam meningkatkan transparansi dan akuntabilitas pada proses penutupan sesi kasir, serta mengurangi risiko terjadinya manipulasi data kas di lapangan.

3.4.5 Pengujian Fitur *Custom Stock Moves*

Pengujian dilakukan untuk memastikan bahwa fitur *Custom Stock Moves* dapat menampilkan pergerakan barang secara akurat berdasarkan tanggal dan jenis pergerakannya, serta memberikan informasi yang jelas mengenai jumlah masuk, keluar, dan saldo stok berjalan (*running balance*). Pengujian dilakukan melalui metode *black box testing* dengan mengamati hasil output sistem dari input yang diberikan, tanpa melihat struktur internal kode.

Langkah pertama pengujian dimulai dengan membuka wizard filter `stock.move.net.qty.wizard`. Pengguna mengisi field tanggal mulai dan tanggal akhir, serta memilih produk tertentu untuk diuji. Setelah tombol *Apply* ditekan, sistem berhasil menampilkan data pergerakan stok sesuai kriteria yang dimasukkan.

Hasil yang ditampilkan terdiri atas daftar transaksi masuk dan keluar yang terfilter berdasarkan rentang tanggal dan produk yang dipilih.

Selanjutnya dilakukan pengujian terhadap perhitungan kolom tambahan yaitu *In*, *Out*, dan *Balance*. Pada skenario stok masuk (barang diterima dari supplier), nilai kolom *In* bertambah dan kolom *Out* tetap nol. Sementara pada transaksi pengeluaran ke pelanggan, nilai kolom *Out* bertambah dan kolom *In* nol. Hasil pengujian menunjukkan bahwa nilai yang muncul pada masing-masing kolom telah sesuai dengan jenis pergerakan stok.

Fitur *Balance* juga diuji secara berurutan berdasarkan kronologi transaksi. Perhitungan saldo berjalan dilakukan dengan cara menambahkan jumlah *In* dan mengurangi *Out* pada masing-masing baris data. Sistem berhasil menampilkan saldo stok dengan benar dan responsif, karena kolom ini dihitung secara dinamis melalui fungsi `compute` dan tidak tersimpan di database.

Untuk validasi logika sistem, juga dilakukan pengujian terhadap perpindahan antar lokasi internal, seperti antar gudang. Hasilnya, sistem tidak mencatat transaksi tersebut sebagai *In* atau *Out*, dan saldo tidak berubah. Hal ini menunjukkan bahwa sistem telah mengikuti aturan perhitungan sesuai kebutuhan bisnis, yaitu hanya mencatat transaksi dengan lokasi masuk atau keluar dari/ke luar gudang utama.

Terakhir, pengujian dilakukan terhadap validasi tanggal. Saat pengguna memasukkan tanggal mulai yang lebih besar dari tanggal akhir, sistem secara otomatis menolak input dan menampilkan pesan peringatan yang sesuai. Ini menunjukkan bahwa sistem telah memiliki kontrol validasi yang baik terhadap input pengguna.

Secara keseluruhan, hasil pengujian menunjukkan bahwa modul *Stock Moves* telah berfungsi dengan baik dan sesuai dengan rancangan. Sistem berhasil menyaring data transaksi, menghitung jumlah masuk dan keluar, serta menampilkan saldo stok berjalan dengan akurat. Fitur ini sangat membantu perusahaan dalam memantau stok dan membuat keputusan operasional secara tepat waktu dan berbasis data.

3.4.6 Pengujian Fitur *Customer Corporate* dan Pencarian *Sales Order*

Pengujian dilakukan untuk memastikan bahwa penambahan fitur kustomisasi pada modul *Sales Order* berjalan sesuai dengan kebutuhan bisnis, khususnya dalam hal pencatatan pelanggan korporat dan pencarian transaksi berdasarkan nomor *Purchase Order* (PO). Pengujian menggunakan metode *black box testing*, dengan

fokus pada interaksi pengguna dan output sistem tanpa melihat implementasi kode secara langsung.

Pertama, dilakukan pengujian terhadap field baru `customer_corporate`. Setelah pengguna memilih customer utama pada form Sales Order, field tambahan untuk mengisi nama pelanggan korporat berhasil ditampilkan dan dapat diisi dengan teks bebas. Data yang dimasukkan ke field ini tersimpan dengan benar dan muncul saat form dibuka kembali, membuktikan bahwa field telah terhubung secara fungsional dengan objek penjualan.

Selanjutnya, dilakukan pengujian pada field `po_number`. Field ini berhasil ditampilkan setelah bagian *Customer Corporate*, dan dapat diisi dengan nomor PO dari customer eksternal. Saat simpan data dilakukan, nomor PO tetap tercatat dan dapat diakses ulang. Fitur ini mendukung kebutuhan administratif tim penjualan untuk mendokumentasikan referensi PO dari pihak customer.

Pengujian juga dilakukan terhadap model dan menu `sale.customer.delight` yang berada di bawah menu *Orders* dalam modul *Sales*. Menu *Others Customers* berhasil diakses, menampilkan tampilan list dan form view. Pengguna dapat menambahkan, mengedit, dan menghapus data secara langsung dari tampilan list. Hal ini mempermudah tim dalam mencatat pelanggan non-standar secara cepat tanpa harus mengakses menu `res.partner` utama.

Untuk fitur pencarian, pengujian dilakukan dengan memasukkan nilai `po_number` pada kolom pencarian di daftar *Sales Order*. Hasil pengujian menunjukkan bahwa sistem berhasil menampilkan transaksi yang sesuai dengan nomor PO yang dimasukkan. Ini membuktikan bahwa field tersebut telah terintegrasi dengan `search view` dan mendukung pencarian berdasarkan PO eksternal dari customer.

Secara keseluruhan, pengujian menunjukkan bahwa seluruh fitur tambahan yang dikembangkan telah berjalan dengan baik. Pengguna dapat mencatat informasi pelanggan korporat tambahan, melakukan pencarian data berdasarkan nomor PO, serta mengelola pelanggan lain di luar sistem utama. Dengan demikian, sistem menjadi lebih fleksibel, informatif, dan efisien untuk mendukung aktivitas tim penjualan di perusahaan.

3.5 Kendala yang Ditemukan

Selama menjalani kegiatan magang di PT Klik Semangat Indonesia, terdapat beberapa kendala yang penulis hadapi, baik dari sisi teknis maupun non-teknis.

Dari sisi non-teknis, salah satu tantangan utama adalah jarak antara rumah dan kantor yang cukup jauh, yaitu sekitar 20 kilometer. Hal ini menyebabkan penulis perlu melakukan perjalanan pulang-pergi setiap hari, yang terkadang memakan waktu dan tenaga, terutama ketika terjadi kemacetan atau cuaca tidak mendukung.

Selain itu, penulis juga beberapa kali menghadapi situasi di mana harus bersiap di malam hari apabila terdapat kendala teknis yang perlu segera diatasi. Misalnya, ketika ditemukan bug pada modul yang penulis kerjakan dan sistem tersebut harus siap digunakan pada pagi harinya. Hal ini menuntut kesiapsiagaan dan kecepatan dalam melakukan debugging agar tidak mengganggu operasional perusahaan.

Dari sisi teknis, salah satu kendala awal yang cukup signifikan adalah keharusan untuk mempelajari sistem Odoo terlebih dahulu. Sebelum magang, penulis belum pernah menggunakan Odoo, sehingga perlu waktu untuk memahami struktur modul, arsitektur framework, dan konsep-konsep utama seperti model, view, serta mekanisme inheritance yang digunakan dalam pengembangannya. Selain itu, kesalahan kecil dalam penulisan XPath atau struktur XML juga bisa berdampak langsung terhadap tampilan atau fungsi modul yang sedang dibuat.

3.6 Solusi atas Kendala yang Ditemukan

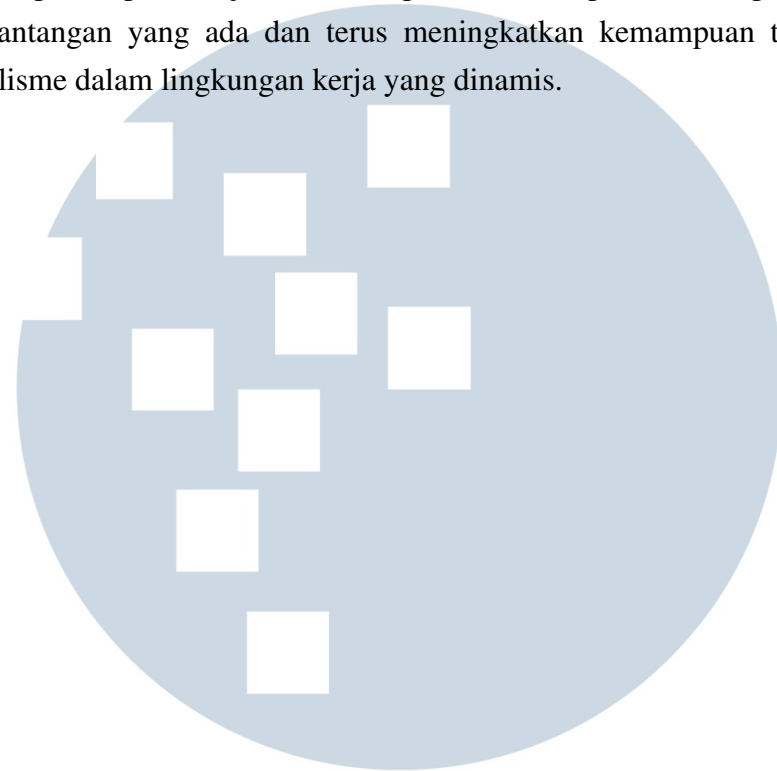
Untuk mengatasi kendala jarak, penulis mengatur waktu keberangkatan lebih awal agar dapat menghindari kemacetan, serta memanfaatkan waktu perjalanan secara produktif dengan membaca dokumentasi atau melakukan perencanaan tugas. Penulis juga melakukan koordinasi jarak jauh dengan supervisor apabila diperlukan, terutama dalam situasi yang tidak memungkinkan untuk hadir secara fisik.

Terkait kesiapan debugging di malam hari, penulis membiasakan diri untuk mendokumentasikan setiap perubahan kode serta melakukan pengujian secara menyeluruh sebelum modul dipindahkan ke server produksi. Hal ini bertujuan untuk meminimalisir kemungkinan terjadinya error mendadak yang membutuhkan penanganan cepat di luar jam kerja.

Dalam menghadapi kendala teknis terkait pembelajaran Odoo, penulis memanfaatkan minggu-minggu awal magang untuk membaca dokumentasi resmi Odoo dan mencoba berbagai tutorial dasar untuk memahami alur pengembangan modul. Penulis juga sering melihat referensi modul-modul bawaan serta bertanya langsung kepada rekan kerja atau pembimbing teknis apabila menemui kebingungan. Dengan pendekatan ini, penulis secara bertahap mulai memahami alur kerja Odoo dan dapat berkontribusi lebih baik pada pengembangan sistem

sesuai kebutuhan perusahaan.

Melalui proses pembelajaran dan adaptasi tersebut, penulis mampu mengatasi berbagai tantangan yang ada dan terus meningkatkan kemampuan teknis serta profesionalisme dalam lingkungan kerja yang dinamis.



UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA