BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Sebagai *Software Programmer* di PT Hashmicro Solusi Indonesia di departemen *Product*. Dalam pekerjaannya bekerja dekat dengan *Product Owner*, *System Analyst* dan sesama *Product Developers*. Selama bekerja tugas diberikan oleh *System Analyst* dan terkadang oleh *Product Owner*. Tugas diberikan dalam bentuk *file Microsoft Word* yang berisikan hasil analisa dari *Requirement*. Selama periode magang dibantu oleh beberapa perangkat lunak seperti *Skype* untuk berkomunikasi, *Visual Studio Code* untuk melakukan pengkodean dan juga menggunakan *Screenshot Monitor* untuk melakukan pelacakan waktu.

Software Programmer pada PT Hashmicro Solusi Indonesia merupakan *Full stack*, yang berarti mengerjakan kedua bagian *frontend* dan *backend*. Pada bagian *front end* membuat tampilan *(view)*. Pada bagian *back end* membuat *model* dan juga relasi-relasinya.

3.2 Tugas yang Dilakukan

Berikut akan dibahas mengenai tugas yang diberikan selama periode magang di PT Hashmicro Solusi Indonesia. Terdapat 4 (Empat) tugas yang diberikan.

3.2.1 Linimasa periode magang

Berikut ini ialah linimasa kegiatan selama periode magang. Linimasa kerja magang diuraikan seperti pada Tabel 3.1.

UNIVERSITAS MULTIMEDIA NUSANTARA

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Onboarding
2	Belajar tentang modul yang dimiliki oleh PT Hashmicro Solusi Indonesia, *git* dan *python*
3	Belajar tentang modul yang dimiliki oleh PT Hashmicro Solusi Indonesia, *git* ,*python* dan juga alat yang digunakan oleh tim teknis
4	Mengikuti pelatihan penggunaan *Odoo*
5	Mengikuti pelatihan penggunaan *Odoo* dan juga mengikuti *training* MSIB yang diselenggarakan oleh pemerintah
6	Mengikuti pelatihan penggunaan *Odoo*
7	Mengikuti pelatihan penggunaan *Odoo*
8	Belajar mandiri mengenai *Odoo* dan membuat *custom module* di *Odoo*
9	Belajar implementasi modul *Accounting* ke *custom module* yang dibuat
10	*Set-up local machine* untuk pengerjaan tugas magang
11	**Tugas 1** : Mengurutkan pekerjaan berdasarkan *state* **Tugas 2** : Menghapus *filter default*, mengubah *order by*, menambahkan attribut dan mengganti *label* di beberapa *field*
12	**Tugas 3** : Membuat menu *Cutting Plan*
13	**Tugas 4** : Membuat fitur baru di menu *Cutting Plan*
14	Libur Idul Fitri 1443H dan cuti bersama
15	Melanjutkan tugas 4
16	Merawat kode dan memperbaiki *bug* pada kode
17	Merawat kode dan memperbaiki *bug* pada kode
18	Merawat kode dan memperbaiki *bug* pada kode
19	Merawat kode dan memperbaiki *bug* pada kode
20	Merawat kode dan memperbaiki *bug* pada kode

3.2.2 Rangkuman tugas magang

Berikut akan dijelaskan beberapa tugas yang diberikan selama periode magang.

A. Tugas 1 : Mengurutkan pekerjaan berdasarkan *state*

Pada tugas ini diminta untuk mengurutkan *state* dari *Production Record*. *State* yang terdapat pada *Production Record* ialah *draft, to be approved, confirmed* dan *reject*. *Production Record* diurutkan sedemikian rupa sehingga tiap *Production Record* yang berada di *state confirmed* dan *rejected* berada di paling belakang. Lalu juga tiap *Production Record* diurutkan berdasarkan nomor produksinya.

Tugas ini diselesaikan dengan melakukan *method overriding* pada *method* `_generate_order_by`. Dengan menambahkan sintaks `ORDER BY` yang digabung dengan `CASE` pada kolom *state*. Lalu pada argumen kedua dari `ORDER BY` ditambahkan kolom nomor produksi sehingga setelah *state* tersortir maka nomor produksi akan disortir secara otomatis.

B. Tugas 2 : Menghapus *filter default*, mengubah *order by*, menambahkan attribut dan mengganti *label* di beberapa *field*

Pada tugas ini diminta untuk menghapus *filter default* yang sudah ada di menu *Material to Purchase* sehingga pada tampilan awal semua *record* dapat terlihat. Selain itu diminta juga untuk mengganti nama (*label*) dari beberapa *field* yang sudah ada. Setelah itu juga diminta untuk mengubah atribut dari beberapa *field* sehingga *field* tersebut menjadi *required*. Terdapat beberapa *field* yang diminta untuk diubah.

Tugas ini diselesaikan dengan menghapus langsung *filter default* yang berada pada *file XML* yang berisi *view* dari *Material to Purchase*. Kemudian memberikan atribut `_order` pada *model* dari *Material to Purchase*. Untuk mengubah *field* menjadi *read only* maka dapat dibuah pada *model* dari *Material to Purchase* dengan menambahkan atribut `readonly=True`. Untuk mengubah nama dari *field* maka dapat mengubah atribut *string* yang terdapat pada *model* dari *Material to Purchase*.

C. Tugas 3 : Membuat menu *Cutting Plan*

Pada tugas ini diminta untuk membuat *menu* baru di modul *cutting*. Modul ini bertujuan untuk membuat sebuah *cutting plan* yang terdiri dari beberapa *cutting order*. Selain itu terdapat juga beberapa *field* lainnya seperti tanggal mulai, nama dan berbagai *field* lainnya. Field ini juga berisi daftar *cutting order* dan juga *product*.

Tugas ini diselesaikan dengan membuat *model, view* dan menambahkan *security*. Relasi dan juga *field* didefinisikan pada *model* yang berupa *file Python*. Tampilan dari menu tersebut didefinisikan juga pada *view* yang berupa *file XML*. Setelah itu juga ditambahkan *security* pada *file* `ir.model.access.csv` sehingga *model* dan *view* yang telah didefinisikan dapat diakses oleh pengguna dengan hak akses tertentu.

D. Tugas 4 : Membuat fitur baru di menu *Cutting Plan*

Pada tugas ini diminta untuk menambahkan beberapa fitur dari *menu cutting plan* yang telah dibuat pada Tugas 3. Fitur tersebut ialah menambahkan sebuah tombol, dimana ketika tombol tersebut ditekan akan muncul sebuah *pop-up* yang dapat membuat sebuah *cutting order* baru dan menempatkan *cutting order* yang baru dibuat tersebut langsung ke *cutting plan*. Selain itu juga membuat daftar dari *cutting order* yang sudah ditempatkan di *cutting plan* tidak bisa diubah (*read-only*). Terdapat juga perubahan nama dari beberapa *field* yang sudah ada.

Tugas ini diselesaikan dengan cara membuat *wizard* yang berperan sebagai *pop-up* yang memiliki relasi terhadap *cutting order*. Lalu untuk membuat daftar dari *cutting order* menjadi tidak bisa diubah cukup dengan menambahkan atribut `readonly=True`. Untuk perubahan nama pada *field* dapat dilakukan dengan mengubah atribut *name* pada *model* terkait.

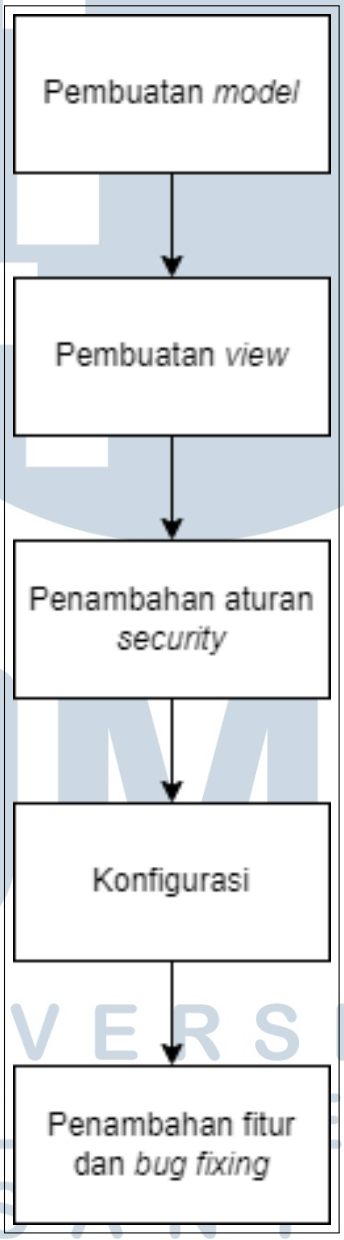
3.3 Uraian Pelaksanaan Magang

3.3.1 Uraian tugas

Pada bagian ini akan dibahas mengenai tugas 3, yaitu membuat menu *cutting plan*. Menu ini menjadi bagian dari modul *cutting*. Modul ini berada dalam modul manufaktur yang dimiliki PT. Hashmicro Solusi Indonesia. Tujuan dibuatnya modul ini ialah agar beberapa *cutting order* dapat dikelompokan dan dapat dipantau lebih mudah.

Requirement dari menu ini ialah terdapat beberapa *field* dasar seperti nama dan tanggal dan sebagainya. Terdapat juga sebuah relasi *one2many* terhadap *cutting order* dan juga sebuah *tree view* yang berisikan produk hasil potong dari *cutting order*. Terdapat juga beberapa tombol yang hanya sekadar menjadi *placeholder* karena fiturnya akan dikembangkan lagi. Beberapa *requirement* tidak bisa dijelaskan karena merupakan rahasia perusahaan.

Langkah penyelesaian yang diambil dalam pengerjaan tugas ini ialah membuat *model* dari *cutting plan*. Setelah itu dilanjutkan oleh pembuatan *view* dari *cutting plan*. Setelah itu dilakukan penambahan aturan pada file *security* agar menu dapat diakses oleh pengguna. Setelah itu dilakukan konfigurasi agar *model* dan *view* yang dibuat dapat dimuat oleh *odoo*. Dan yang terakhir dilakukan penambahan fitur dan juga *bug fixing*.



Gambar 3.1. Flowchart pengerjaan

A. Pembuatan *model*

Model pada *Odoo* menyatakan sebuah tabel basis data. Untuk pembuatan *model cutting plan* dapat dilakukan dengan membuat sebuah *file python* pada *folder models*. *Model* dinyatakan dalam sebuah *class*. *Class* tersebut, untuk menjadi sebuah *model* yang berfungsi, harus melakukan *inherit* terhadap *class* `odoo.models.Model`.

Didalam *class* yang sudah didefinisikan, kita dapat menambahkan sebuah *column*. Pada *odoo*, sebuah *column* disebut sebagai *field*. Maka sebuah *class* dapat memiliki beberapa *field* sehingga menjadi sebuah tabel dengan beberapa kolom. Sebuah *model* pada *odoo* biasanya berisikan sebuah *class* yang melakukan *inherit* terhadap *class* `odoo.models.Model`, sebuah atribut `_name` yang menyatakan nama dari *model* tersebut, sebuah atribut `_description` yang berisi deskripsi dari *model* tersebut dan juga memiliki atribut yang berupa *field*.

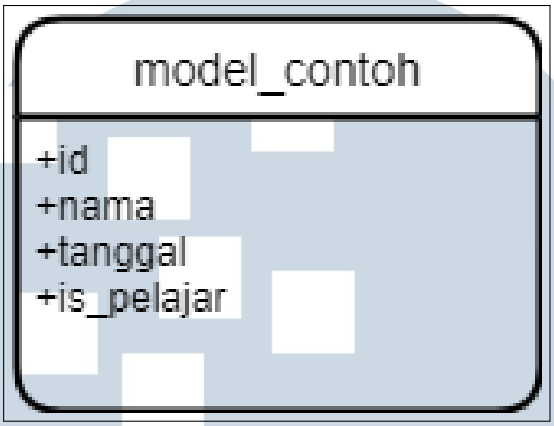
Untuk menyatakan sebuah *field*, kita dapat membuat sebuah atribut dari *class* tersebut yang berisikan sebuah `odoo.fields`. Tiap *field* dapat memiliki *data type* yang berbeda. Untuk menyatakan sebuah *field char* dapat digunakan `odoo.fields.Char`, dan untuk sebuah *field boolean* dapat digunakan `odoo.fields.Boolean`. Contoh sebuah *model Odoo* dapat dilihat pada kode 3.1

Kode 3.1: Contoh model

```
from odoo import api, fields, models, _

class ModelContoh(models.Model):
    _name = 'model.contoh'
    _description = 'Model Contoh'

    nama = fields.Char(string='Nama')
    tanggal = fields.Date(
        string='Tanggal',
        default=fields.Date.today(),
        readonly=True,)
    is_pelajar = fields.Boolean(string='Pelajar', default=False)
```

pada kode 3.1 terdapat sebuah *model* yang bernama `model.contoh` dan memiliki attribut `nama`, `tanggal`, dan `is_pelajar`. Terdapat juga beberapa *field* yang sudah di-*reserve* dan dibuat oleh *odoo* untuk keperluan pada sistem. Salah

satunya ialah *field* `id`. Representasi dari *model* tersebut dalam diagram terdapat pada Gambar 3.2.



Gambar 3.2. Tabel model contoh

Terdapat juga *field* yang berupa relasi. Terdapat 3 (tiga) relasi yang dapat digunakan di *odoo* yaitu *many2one, many2many,* dan *one2many*. Pada *field* ini menyimpan nilai yang berupa sebuah relasi terhadap tabel lain.

Relasi *many2one* hanya bisa menyimpan 0 (nol) atau 1 (satu) *record* saja [7]. Sebagai contoh ialah tempat lahir. Satu orang hanya memiliki 1 (satu) tempat lahir, tetapi tempat lahir tersebut bisa memiliki banyak orang. Sehingga dapat dipetakan orang sebagai aktor *many* dan tempat lahir sebagai aktor *one*.

Relasi *many2many* dapat menyimpan banyak *record*. Sebagai contoh ialah genre film [8]. Satu film dapat memiliki banyak genre dan satu genre dapat memiliki banyak genre. Sehingga keduanya dapat dipetakan menjadi aktor *many*.

Relasi *one2many* dapat menyimpan banyak *record* pada *comodel* dimana *field* `inverse_comodel` memiliki *record* yang sama dengan *record* sekarang [9]. Sebagai contoh ialah kantong belanja pada saat kita berbelanja di *e-commerce*. Setiap akun dapat memiliki kantong belanja yang isinya berbeda sehingga tiap *record* akan dibuat pada saat user memasukan barang baru ke kantong belanja. Sehingga pembeli dapat dipetakan menjadi aktor *one* dan tiap barang yang ada di kantong belanja menjadi aktor *many*.

Contoh penggunaan dari *many2one, many2many,* dan *one2many* dapat dilihat pada kode 3.2

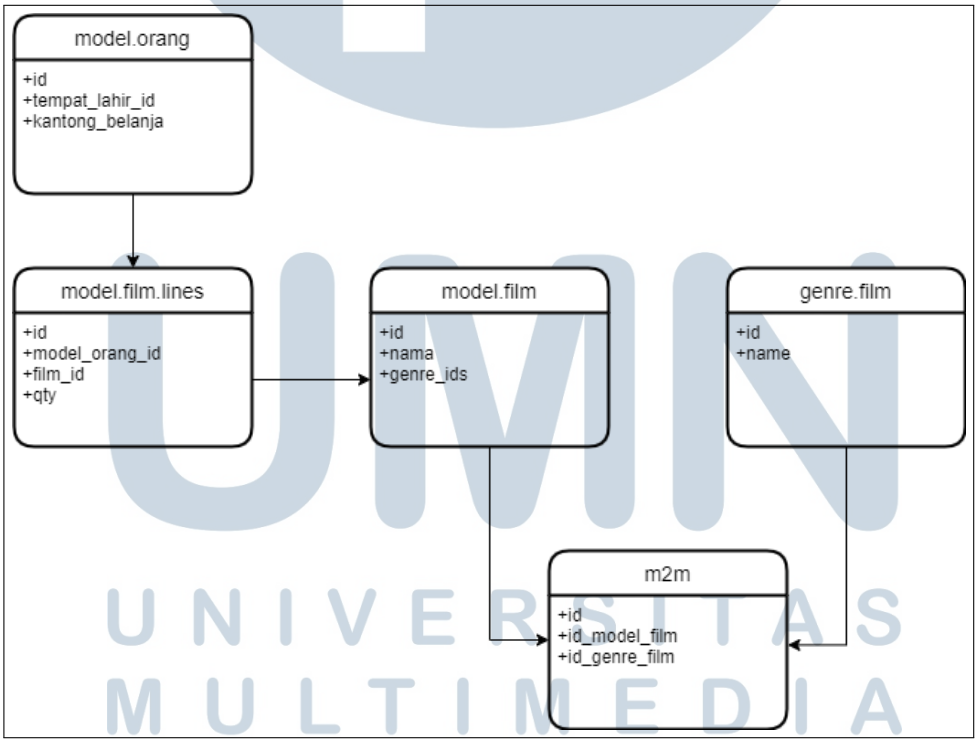
Kode 3.2: Contoh model dengan relasi

```
from odoo import api, fields, models, _
```

```
class ModelOrang(models.Model):
    _name = 'model.orang'
    _description = 'Model Orang'

    tempat_lahir_id = fields.Many2one(
        comodel_name='res.country',
        string='Tempat Lahir'
    )
    kantong_belanja_ids = fields.One2many(
        string='Kantong Belanja',
        comodel_name='model.film.lines',
        inverse_name=model_orang_id
    )

class GenreFilm(models.Model):
    _name = 'genre.film'
    _description = 'Genre Film'

    name = fields.Char(string='Genre')

class ModelFilm(models.Model):
    _name = 'model.film'
    _description = 'Model Film'

    nama = fields.Char(string='Nama Film')
    genre_ids = fields.Many2many(
        comodel_name='genre.film',
        string='Genre'
    )

class ModelFilmLines(models.Model):
    _name = 'model.film.lines'
    _description = 'Model film lines'

    model_orang_id = fields.Many2one(
        string='Model Orang',
```

```
        comodel_name='model.orang'
    )
    film_id = fields.Many2one(
        string='Film',
        comodel_name='model.film'
    )
    qty = fields.Integer(string='Quantity')
```

Pada kode 3.2 terdapat sebuah *model* `ModelOrang` yang memiliki hubungan *many2one* terhadap `res.country`. `res.country` merupakan *model* bawaan *odoo* yang berisikan negara yang ada. Lalu terdapat juga relasi *one2many* terhadap *model* `model.film.lines`. Lalu pada *model* `model.film.lines` terdapat relasi *many2one* terhadap `model.orang` yang merupakan `inverse_comodel` dan juga relasi *many2one* terhadap `model.film`. Pada `model.film` terdapat relasi *many2many* terhadap *model* `genre.film`. Relasi antar *model* dapat dilihat pada Diagram 3.3.



Gambar 3.3. Tabel model contoh

Dalam penyelesaian tugas ini, digunakan *field Char, Date, Datetime, Selection, Many2one, One2Many, Many2many*. *Field* tersebut digunakan sesuai *requirement* yang diberikan.

B. Pembuatan *view*

View pada *odoo* menyatakan sebuah tampilan. Untuk pembuatan *view cutting plan* dapat dilakukan dengan membuat sebuah *file XML* pada *folder views*. *View* dinyatakan pada file *XML* didalam *tag* `<odoo></odoo>`. Untuk menjadi sebuah *view* yang berfungsi sebuah *file XML* harus memiliki *action* dan juga tipe-tipe *view* lainnya yang disediakan oleh *odoo* seperti *tree, form, graph, kanban* dan lainnya. Bentuk generik dari *view* dapat dilihat pada Kode 3.3.

Kode 3.3: Contoh view

```
<record id="MODEL_view_TYPE" model="ir.ui.view">
  <field name="name">NAME</field>
  <field name="model">MODEL</field>
  <field name="arch" type="xml">
    <VIEW_TYPE>
      <VIEW_SPECIFICATIONS/>
    </VIEW_TYPE>
  </field>
</record>
```

Dimana `<VIEW_TYPE>` dapat berupa sebagai berikut (Tabel 3.2):

Tabel 3.2. Daftar *view_type*

VIEW_TYPE	Bentuk *view*	Digunakan pada pengerjaan
<tree>	*View tree*	Ya
<form>	*View form*	Ya
<search>	*View form*	Ya
<activity>	*View activity*	Tidak
<calendar>	*View calendar*	Tidak
<dashboard>	*View dashboard*	Tidak
<gantt>	*View gantt*	Tidak
<graph>	*View graph*	Tidak
<kanban>	*View kanban*	Tidak
<pivot>	*View pivot*	Tidak

Lalu terdapat `VIEW_SPECIFICATIONS` yang digunakan pada pengerjaan sebagai berikut (Tabel 3.3):

Tabel 3.3. Daftar *view_specification*

VIEW_SPECIFICATION	Fungsi
field	Menampilkan *field*
button	Menampilkan *button*
group	Mengelompokan beberapa *field* dan *button*
header	Isi dari header akan berada pada *header view*
sheet	Kanvas dasar pada *view form*
notebook	Mengelompokkan beberapa *field* dan *button* dalam bentuk *notebook*

Lalu pada tiap *tag view* dapat ditambahkan beberapa atribut sebagai berikut (Tabel 3.4):

Tabel 3.4. Daftar atribut

Atribut	Fungsi
readonly	Membuat sebuah *field read-only*
invisible	Menyembunyikan *field*
domain	Untuk melakukan *filtering* pada *field*
string	Mengubah *label* dari *field*
widget	Merubah tampilan dari *field*
class	Memberikan *styling* terhadap *field*
nolabel	Menyembunyikan *label* dari *field*
force_save	Untuk memperbolehkan perubahan (dari kode) *field* ketika *field* memiliki atribut *readonly*

Umumnya, tiap *model* memiliki *view* masing-masing. Pada *view* tersebut berisikan view *tree* dan *form*. Memiliki sebuah <action> yang menyatakan bahwa *view tree, form* dan lainnya merupakan milik dari suatu *model*. Dan juga sebuah <menuitem> untuk memunculkan *view* pada *menu*. Berikut pada kode 3.4 ialah contoh dari *view*.

Kode 3.4: Contoh kode view

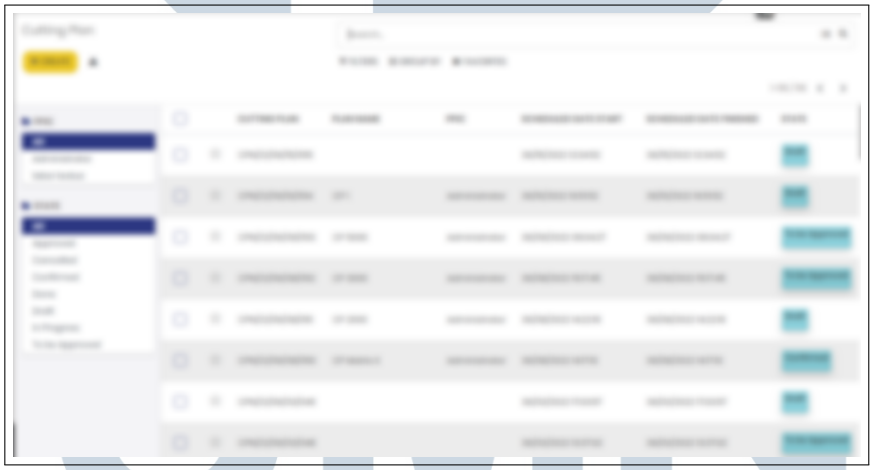
```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>

<record id="contoh_tree" model="ir.ui.view">
```

```
    <field name="name">contoh.view.tree</field>
    <field name="model">model.contoh</field>
    <field name="arch" type="xml">
        <tree string="Model Contoh" default_order="field_1">
            <field name="field_1" widget="priority"/>
            <field name="field_2"/>
            <field name="field_3"/>
            <field name="field_4"/>
        </tree>
    </field>
</record>

<record id="view_model_contoh_form" model="ir.ui.view">
<field name="name">model.contoh.view.form</field>
<field name="model">model.contoh</field>
<field name="arch" type="xml">
<form string="Model Contoh">
<header>
<button name="button_contoh" string="Contoh" type="object"/>
</header>
    <sheet>
        <group>
            <group>
            <field name="field_1"/>
            <field name="field_2"/>
            </group>
            <group>
            <field name="field_3" string="Field 3"/>
            <field name="field_4" string="Field 4"/>
            </group>
        </group>
        <notebook>
        <page name="laman_1" string="Laman 1">
        <field name="field_many2many" widget="many2many">
            <tree string="Field Many2many">
                <field name="m2m_0"/>
                <field name="m2m_1"/>
            </tree>
```

```
        </field>
        </page>
        <page name="laman_2" string="Laman 2">
        <field name="field_o2m" readonly="1" force_save="1">
        <tree string="Field One2many">
            <field name="o2m_0" readonly="1" force_save="1"/>
            <field name="o2m_1" force_save="1"/>
        </tree>
        </field>
        </page>
        </notebook>
    </sheet>
    <div class="oe_chatter">
        <field name="message_follower_ids"/>
        <field name="activity_ids"/>
        <field name="message_ids"/>
    </div>
</form>
</record>

<record id="view_search_model_contoh" model="ir.ui.view">
    <field name="name">model.contoh.view.search</field>
    <field name="model">model.contoh</field>
    <field name="arch" type="xml">
        <search string="Model Contoh">
            <searchpanel>
                <field name="field_1"/>
                <field name="field_2"/>
            </searchpanel>
        </search>
    </field>
</record>

<record id="action_contoh" model="ir.actions.act_window">
    <field name="name">Model Contoh</field>
    <field name="type">ir.actions.act_window</field>
    <field name="res_model">model.contoh</field>
    <field name="view_mode">tree,form</field>
```

```
    <field name="help" type="html">
        <p class="o_view_nocontent_smiling_face">
        Create a new Model Contoh
      </p>
    </field>
</record>

<menuitem id="menu_model_contoh"
     name="Model Contoh"
     parent="menu_parent"
     action="action_view_model_contoh"
     sequence="1"/>

</odoo>
```

Maka hasil dari *view* tersebut terdapat pada Gambar 3.4



Gambar 3.4. Hasil dari *view*

C. Penambahan aturan *security*

Security pada *odoo* berfungsi untuk membatasi akses kepada beberapa *user*. Tiap *rule* dapat membatasi *create, read, update* dan *unlink (delete)*. Tiap *rule* mengatur 1 (satu) *user* saja. Sebuah akun *administrator* mendapatkan semua hak akses dan juga memiliki kemampuan untuk mengaplikasikan beberapa *rule* ke user lainnnya. Kita dapat membuat *rule* dengan menambahkan *rule* ke dalam *file* `ir.model.access.csv` pada *folder security*.

Struktur dari sebuah *rule* dapat dilihat pada Tabel 3.5.

Tabel 3.5. Rule

Kolom	Penjelasan	Nilai
id	*id* dari *rule*	*String*
name	Nama dari *rule*	*String*
model_id:id	*Model* yang berkaitan dengan *rule*	*String*
group_id:id	*Group* untuk *rule* ini	*String*
perm_read	Izin untuk melakukan *read*	0/1
perm_write	Izin untuk melakukan *write*	0/1
perm_create	Izin untuk melakukan *create*	0/1
perm_unlink	Izin untuk melakukan *unlink*	0/1

Contoh dari *rule* dapat dilihat pada Kode 3.5

Kode 3.5: Contoh kode view

```
model.contoh,model.contoh,model.contoh,contoh_user,1,1,1,0
```

D. Konfigurasi

Setelah membuat *model, view* dan menambahkan *security rule*, maka selanjutnya akan ditambahkan konfigurasi agar *model, view* dapat dimuat oleh *odoo*. *Odoo* memiliki beberapa *file* konfigurasi seperti *file* `__init__.py` dan juga *file* `__manifest__.py`. *File* `__init__.py` berfungsi untuk memuat *model* dan *file* `__manifest__.py` berfungsi untuk memuat *view*. *File* `ir.model.access.csv` juga dimuat pada *file* `__manifest__.py`.

Untuk memuat *model* maka kita dapat melakukan *import* pada *file* `__init__.py`. Untuk memuat *model* maka kita dapat menambahkan sintaks *import* kedalam *file* `__init__.py`. Nantinya *file* `__init__.py` akan di-*import* langsung oleh *odoo*. Contohnya terdapat pada kode 3.6.

Kode 3.6: Contoh kode *import model*

```
from . import <namafile>
```

Nantinya `<namafile>` diubah menjadi nama *file python* yang ditambahkan yang berisikan *model*.

Untuk memuat *view* maka kita dapat melakukan *import* pada *file* `__manifest__.py`. Untuk memuat *view* maka kita dapat menambahkan nama dari

file view ke dalam *list* `data` yang terdapat pada *file* `__init__.py`. Nantinya *file* `__manifest__.py` akan di-*import* langsung oleh *odoo*. Contohnya terdapat pada kode 3.7.

Kode 3.7: Contoh kode *import model*

```
    'data' : [
        ...,
        ...,
        <namafile>,
        ...,
    ]
```

Nantinya `<namafile>` diubah menjadi nama *file view* yang ditambahkan yang berisikan *model*.

E. Penambahan fitur dan *bug fixing*

Terdapat 2 (Dua) fitur yang diminta pada *requirements* yang diberikan.

1. Nama menggunakan nomor produksi yang dibuat langsung oleh sistem
2. Setiap kali *field one2many* yang terhubung dengan *cutting order* berubah maka *field one2many* yang terhubung dengan *product* ikut berubah sesuai dengan *cutting order*

E.1 Fitur 1

Untuk menyelesaikan fitur 1, Nama menggunakan nomor produksi yang dibuat langsung oleh sistem, dapat menggunakan sebuah fitur dari *odoo* yang bernama *sequence*. Dengan *sequence* kita dapat dengan mudah menentukan nomor produksi terakhir. Secara *default* sebuah *sequence* mulai dari 0 (nol) dan akan bertambah 1 (satu) tiap kali dipanggil. Sebuah *sequence* secara *default* akan mengulang angkanya tiap tahun.

Untuk mendefinisikan sebuah *sequence* kita dapat membuat sebuah *file* bernama `ir_sequence_data.xml` yang pada umumnya berada pada *folder data*. Contohnya terdapat pada kode 3.8.

Kode 3.8: Contoh kode *import model*

```
<record id="seq_contoh_model" model="ir.sequence">
```

```
        <field name="name">Contoh Model</field>
        <field name="code">contoh.model</field>
        <field name="prefix">CTH</field>
        <field name="number_next_actual">1</field>
        <field name="padding">3</field>
    </record>
```

Pada kode 3.8, *sequence* tersebut menghasilkan sebuah *sequence* seperti CTH001, CTH002, CTH003 dan seterusnya.

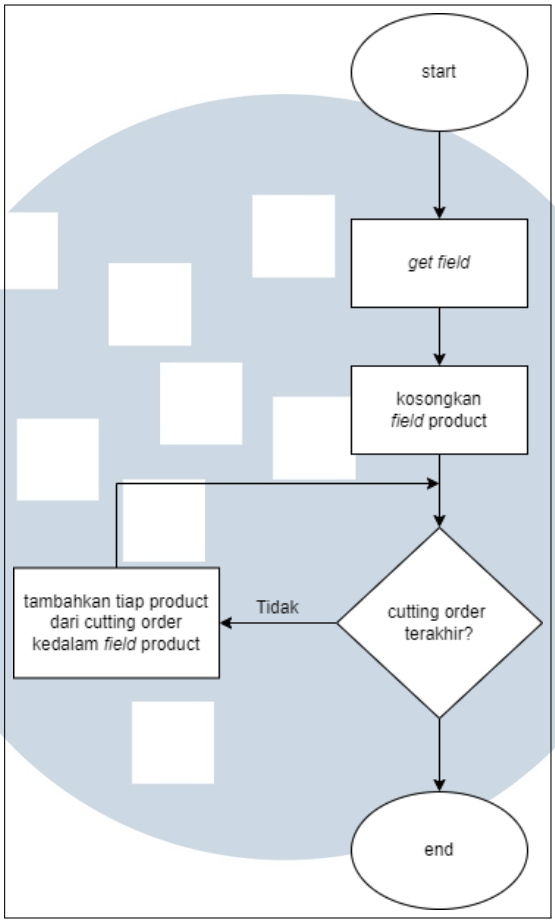
Setelah membuat *sequence*, berikutnya ialah mengimplementasikannya kedalam *model*. Sebuah *sequence* dapat dipanggil dengan menggunakan kode `self.env['ir.sequence'].next_by_code('contoh.model')`. Untuk mengaplikasikan *sequence* yang dibuat kita dapat melakukan *method overriding* pada *method* `create(vals)`. *Method* tersebut merupakan *method* bawaan dari *odoo*. *Method* tersebut dipanggil setiap ada *record* dibuat. Contoh pengaplikasiannya terdapat pada kode 3.9.

Kode 3.9: Contoh kode *import model*

```
def create(self, vals):
    vals['name'] = self.env['ir.sequence']
                    .next_by_code('contoh.model')
    return super(ContohModel, self).create(vals)
```

E.2 Fitur 2

Untuk menangkap setiap perubahan yang dialami oleh *field one2many cutting order*, kita dapat menggunakan sebuah *decorator* `api.onchange(*args)`. *Decorator* tersebut disediakan oleh *odoo* yang berfungsi untuk memanggil *method* yang menggunakan *decorator* ini tiap kali *field* yang berada pada `*args` mengalami perubahan. Perlu diketahui bahwa *decorator* ini hanya akan dijalankan apabila perubahan yang terjadi dibuat pada *GUI*, bukan melalui kode. Apabila ingin melakukan perubahan melalui kode dan ingin mempunyai efek yang sama, maka dapat langsung memanggil *method* tersebut. Berikut pada gambar 3.5 ialah *flowchart* dari solusi yang dikembangkan.



Gambar 3.5. *Flowchart* solusi

Adapun *pseudocode* yang dihasilkan dari *flowchart* tersebut terdapat pada Kode 3.10.

Kode 3.10: Solusi

```
@api.onchange('field_cutting_order')
def _field_onchange(self):
    cutting_order_field = get_cutting_order()
    product_field = get_product_field()
    empty_product_field()
    for product in cutting_order_field:
        add_to_product_field(product)
```

3.4 Kendala dan Solusi yang Ditemukan

Selama masa pengerjaan, terdapat 1 (satu) kendala yang ditemukan. Yaitu pada saat pengaplikasian *requirement* kedua, beberapa *field* tidak dapat terisi. Hal

ini disebabkan karena *field* tersebut memiliki atribut *readonly*. Oleh karena itu pada *field* tersebut ditambahkan sebuah atribut `force_save` yang bernilai 1.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA