2018

odoo EXPERIENCE

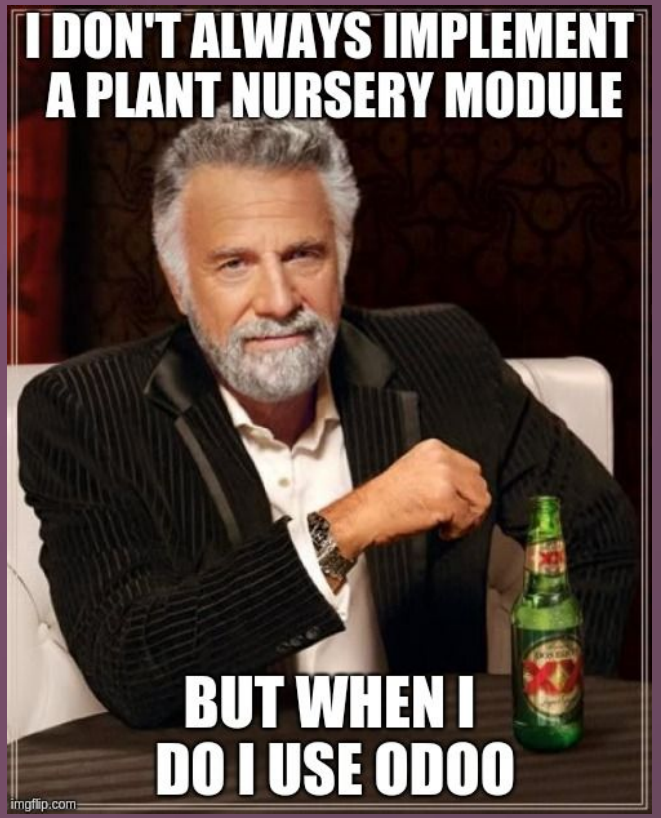
# Develop an App with the Odoo Framework

## Or how to implement a plant nursery in a few minutes.

**Yannick TIVISSE** · Software Engineer, RD Dummies Team Leader

1. Classy Cool Dev introduction
2. Structure of a module
3. Complex views
4. Relations between models
5. ORM interactions
6. Some other classy cool stuffs

# The use case:
# **A Plant Nursery**




— Classy Cool Dev

1

Thanks Classy Cool Dev!
**But how could you be so sure Odoo is the perfect choice ?**

# Architecture

- Three-tier client/server/database
- Webclient in Javascript
- Server and backend modules in Python
  - MVC framework
  - ORM to interact with database

# The Feature

- Manage a plant nursery:
  - List of plants
  - Manage orders
  - Keep a customers list

# Technically

- You will learn:
  - Structure of a module
  - Definition of data models
  - Definition of views and menus

2

# Structure of an **Odoo Module**

# An Odoo module is:

- A manifest file
- Python code (models, logic)
- Data files, XML and CSV (base data, views, menus)
- Frontend resources (Javascript, CSS)

# Plant Nursery

The manifest file __manifest__.py

```
# -*- coding: utf-8 -*-
# Part of Odoo. See LICENSE file for full copyright and licensing details.
{
    'name': 'Plant Nursery',
    'version': '1.0',
    'category': 'Tools',
    'summary': 'Plants and customers management',
    'depends': ['web'],
    'data': [
        'security/ir.model.access.csv',
        'data/data.xml',
        'views/views.xml',
    ],
    'demo': [
        'data/demo.xml',
    ],
    'css': [],
    'installable': True,
    'auto_install': False,
    'application': True,
}
```

# Describe the models

plant_nursery/models.py

```
from odoo import fields, models


class Plants(models.Model):
    _name = 'nursery.plant'

    name = fields.Char("Plant Name")
    price = fields.Float()


class Customer(models.Model):
    _name = 'nursery.customer'

    name = fields.Char("Customer Name", required=True)
    email = fields.Char(help="To receive the newsletter")
```

https://www.odoo.com/documentation/10.0/reference/orm.html#fields

# Define the security

plant_nursery/security/ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_nursery_plant,access_nursery_plant,plant_nursery.model_nursery_plant,base.group_user,1,1,1,1
access_nursery_customer,access_nursery_customer,plant_nursery.model_nursery_customer,base.group_user,1
,1,1,1
```

# Define the action

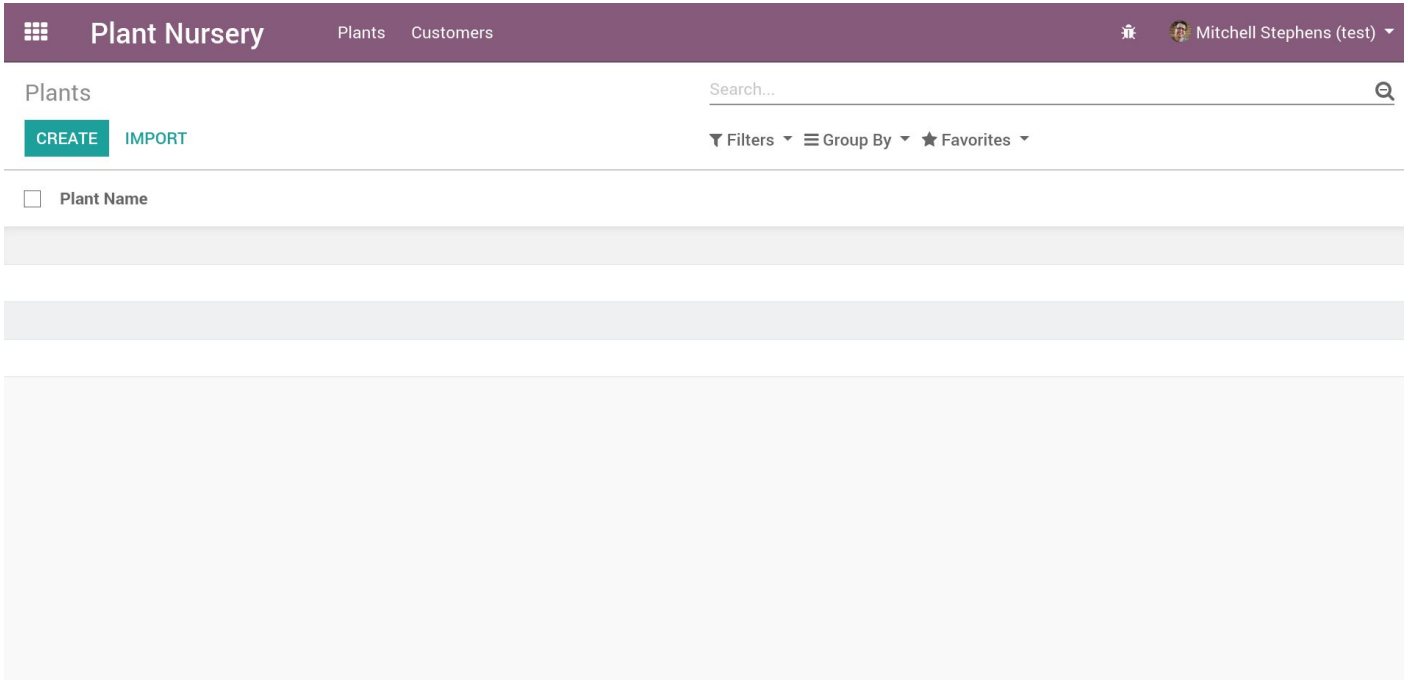
plant_nursery/views/nursery_views.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
    <record model="ir.actions.act_window" id="action_nursery_plant">
        <field name="name">Plants</field>
        <field name="res_model">nursery.plant</field>
        <field name="view_mode">tree,form</field>
    </record>

    <menuitem name="Plant Nursery" id="nursery_root_menu"
              web_icon="plant_nursery,static/description/icon.png"/>

    <menuitem name="Plants" id="nursery_plant_menu"
              parent="nursery_root_menu"
              action="action_nursery_plant"
              sequence="1"/>
</odoo>
```

https://www.odoo.com/documentation/10.0/reference/orm.html#fields

# Watch the result

## Auto generated views




https://www.odoo.com/documentation/10.0/reference/orm.html#fields

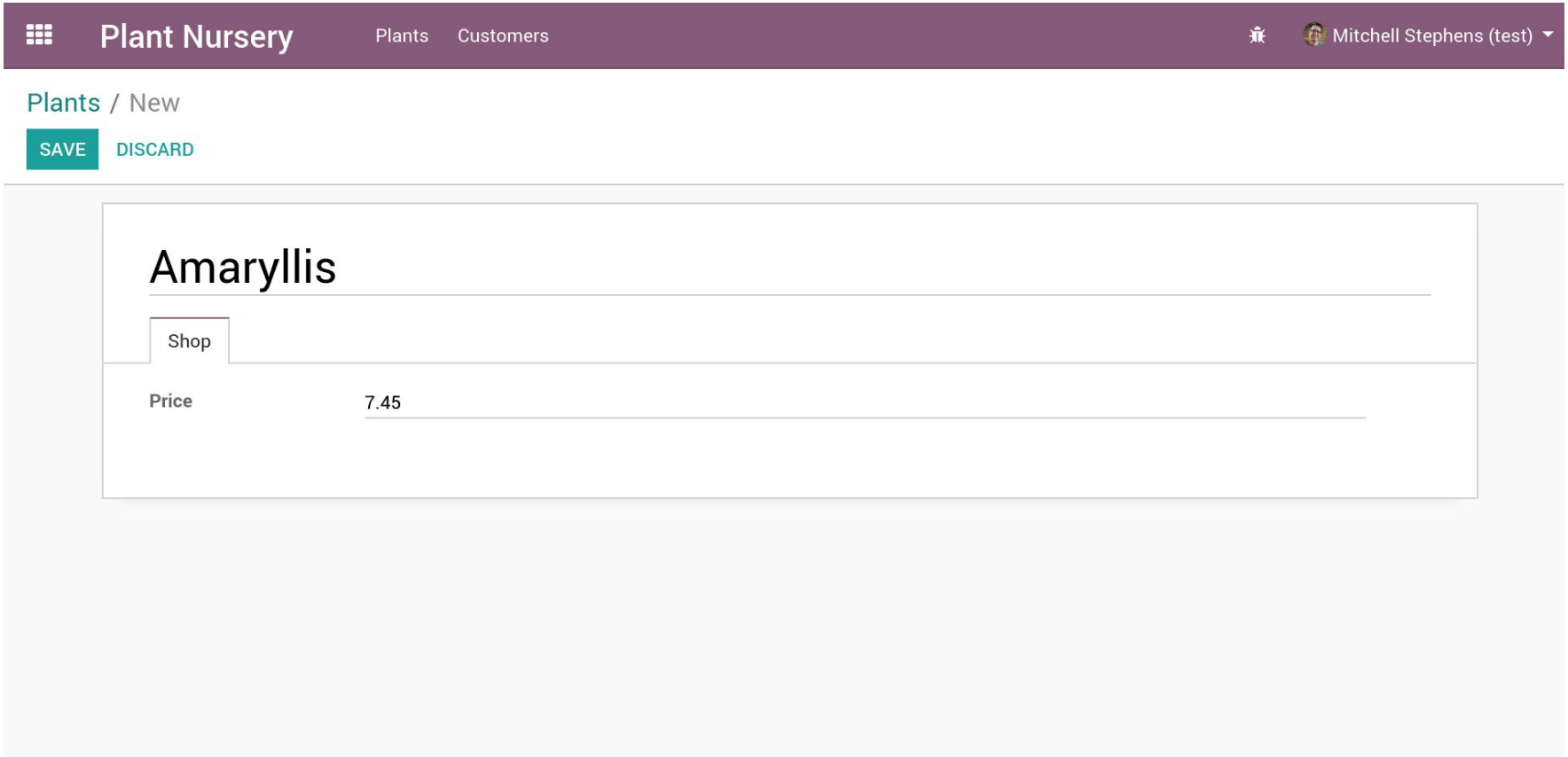
# Complex Views

# Define a form view

plant_nursery/views/nursery_views.xml

```
<record model="ir.ui.view" id="nursery_plant_view_form">
    <field name="name">nursery.plant.view.form</field>
    <field name="model">nursery.plant</field>
    <field name="arch" type="xml">
        <form string="Plant">
            <sheet>
                <h1>
                    <field name="name" placeholder="Plant Name"/>
                </h1>
                <notebook>
                    <page string="Shop">
                        <group>
                            <field name="price"/>
                        </group>
                    </page>
                </notebook>
            </sheet>
        </form>
    </field>
</record>
```

# Watch the result



# 4 Relations between models

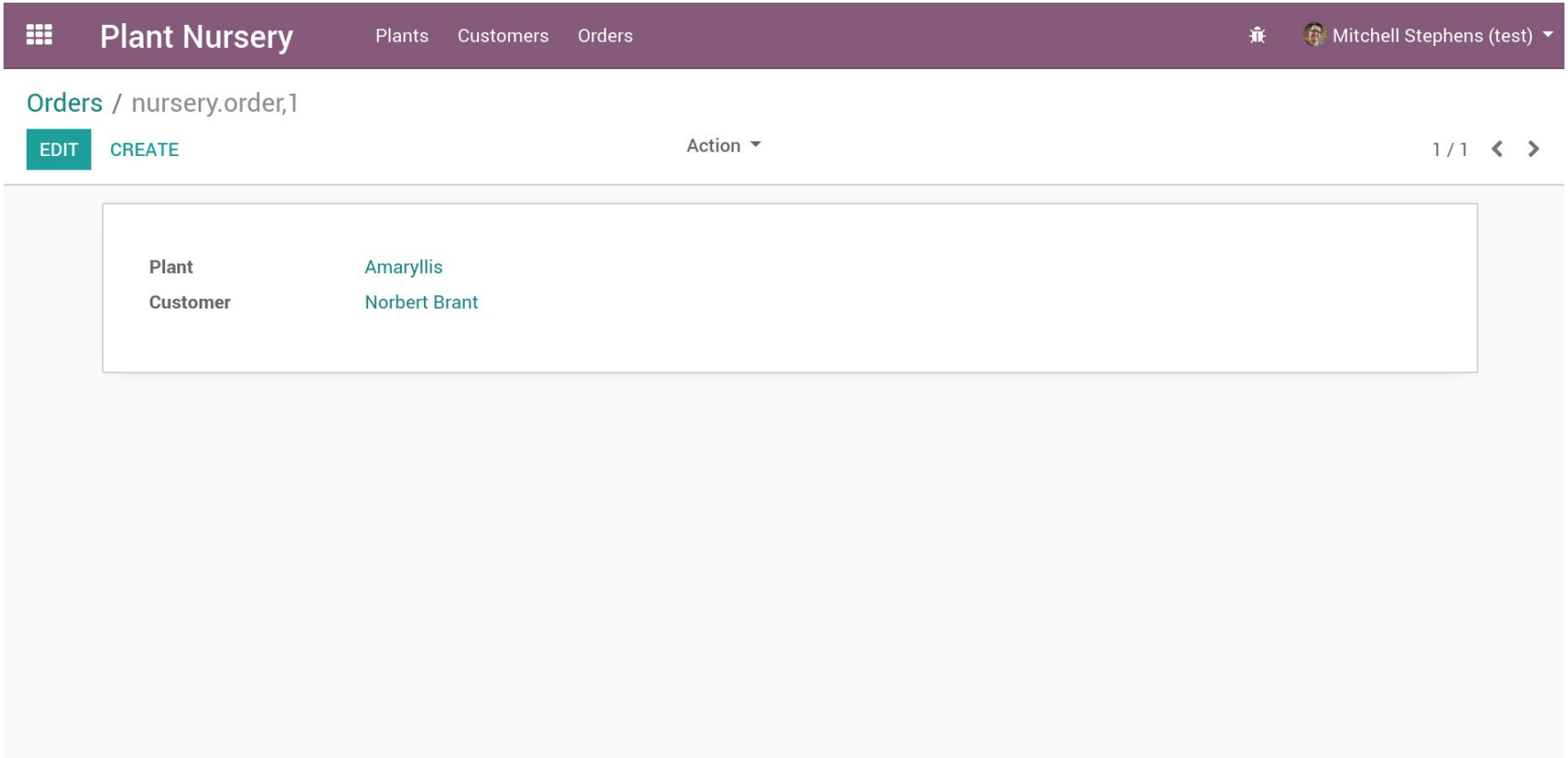
# Relations

- Many2one
- One2many
- Many2many

# Relations

```
class Order(models.Model):
    _name = 'nursery.order'

    plant_id = fields.Many2one("nursery.plant", required=True)
    customer_id = fields.Many2one("nursery.customer")


class Plants(models.Model):
    _name = 'nursery.plant'

    order_ids = fields.One2many("nursery.order", "plant_id", string="Orders")
```

# Watch the result



# 5 ORM Interactions

# Basic Operations

- Read
- Write
- Create
- Unlink
- Search

# ORM Interactions

```
class Order(models.Model):
    _name = 'nursery.order'

    name = fields.Datetime(default=fields.Datetime.now)
    plant_id = fields.Many2one("nursery.plant", required=True)
    customer_id = fields.Many2one("nursery.customer")
    state = fields.Selection([
        ('draft', 'Draft'),
        ('confirm', 'Confirmed'),
        ('cancel', 'Canceled')
    ], default='draft')
    last_modification = fields.Datetime(readonly=True)
```

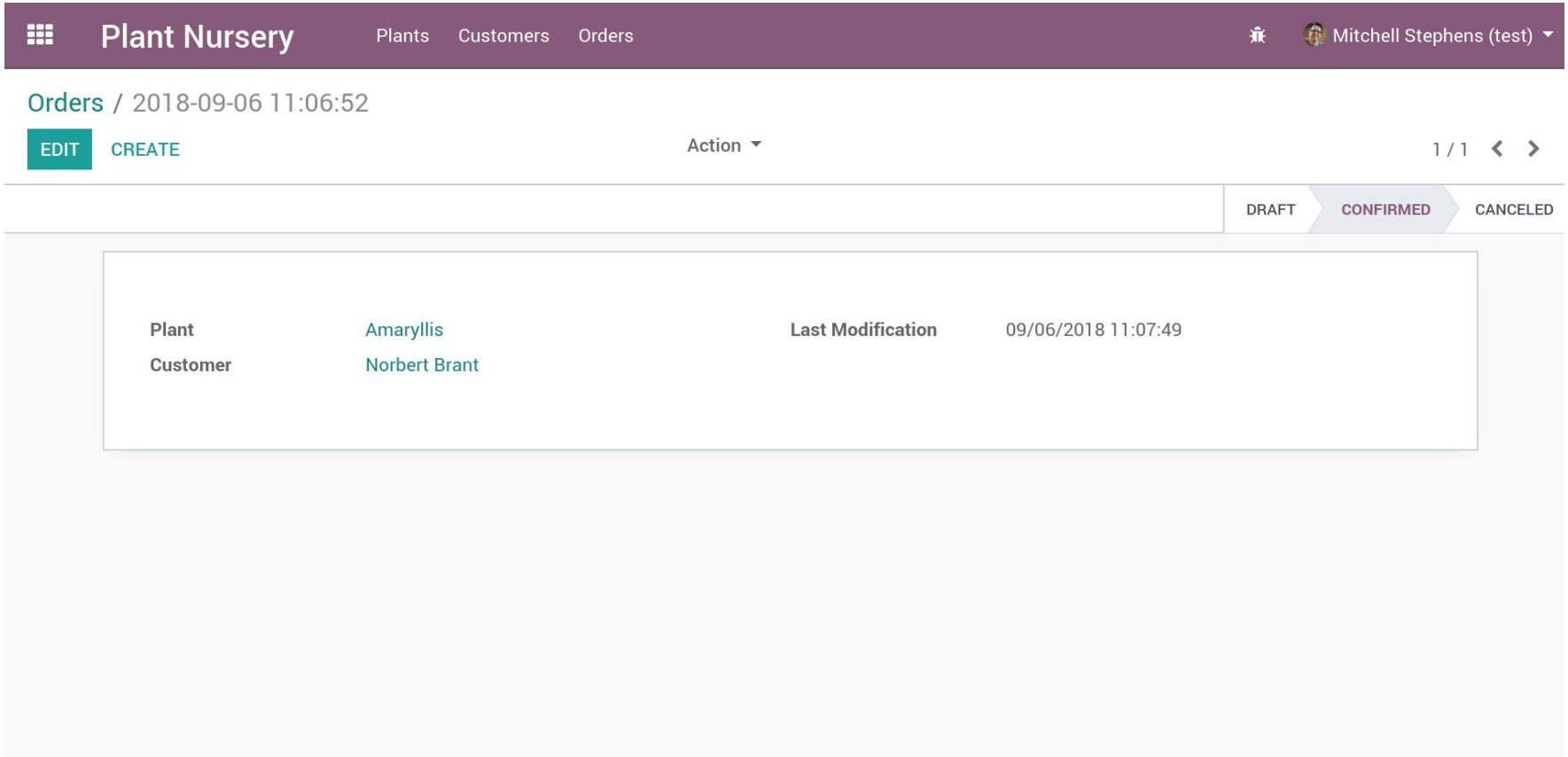
# ORM Interactions

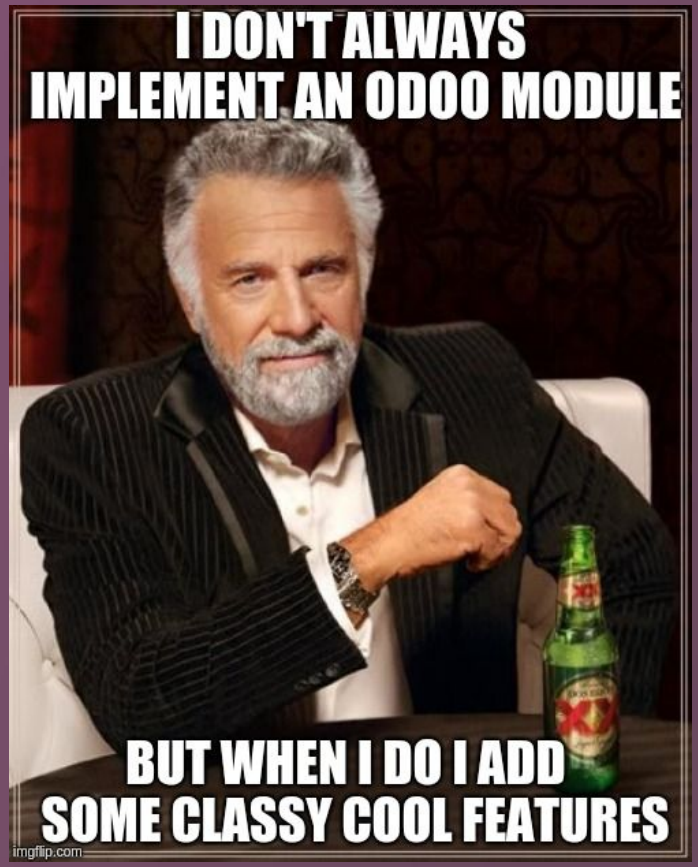
```
class Order(model.Models):
    _name = 'nursery.order'

    def write(self, values):
        # helper to "YYYY-MM-DD"
        values['last_modification'] = fields.Datetime.now()

        return super(Order, self).write(values)

    def unlink(self):
        # self is a recordset
        for order in self:
            if order.state == 'confirm':
                raise UserError("You can not delete confirmed orders")

        return super(Order, self).unlink()
```

# ORM Interactions

```
<record model="ir.ui.view" id="nursery_order_form">
    <field name="name">Order Form View</field>
    <field name="model">nursery.order</field>
    <field name="arch" type="xml">
        <form string="Plant Order">
            <header>
                <field name="state" widget="statusbar" options="{'clickable': '1'}"/>
            </header>
            <sheet>
                <group col="4">
                    <group colspan="2">
                        <field name="plant_id" />
                        <field name="customer_id" />
                    </group>
                    <group colspan="2">
                        <field name="last_modification" />
                    </group>
                </group>
            </sheet>
        </form>
    </field>
</record>
```

# Watch the result






— Classy Cool Dev

6

Thanks Classy Cool Dev!
**What else could you show us ?**

# Computed Fields

- For complex values
- Trigger for recompute
- Stored or not in database

# Computed Fields

```
class Plants(models.Model):
    _name = 'nursery.plant'

    order_count = fields.Integer(compute='_compute_order_count',
                                 store=True,
                                 string="Total sold")

    @api.depends('order_ids')
    def _compute_order_count(self):
        for plant in self:
            plant.order_count = len(plant.order_ids)
```

# Model Constraints

- Triggered after every creation or modification
- Instead of overriding create & write

# Model Constraints

```
class Plants(models.Model):
    _name = 'nursery.plant'

    number_in_stock = fields.Integer()

    @api.constrains('order_count', 'number_in_stock')
    def _check_available_in_stock(self):
        for plant in self:
            if plant.number_in_stock and \
              plant.order_count > plant.number_in_stock:
                raise UserError("There is only %s %s in stock but %s were sold"
                      % (plant.number_in_stock, plant.name, plant.order_count))
```

# Kanban View

- Display information in a tile
- Add a picture of the plant
- Aggregated view to visualize the flow (will need a search view)

# Define a Kanban View

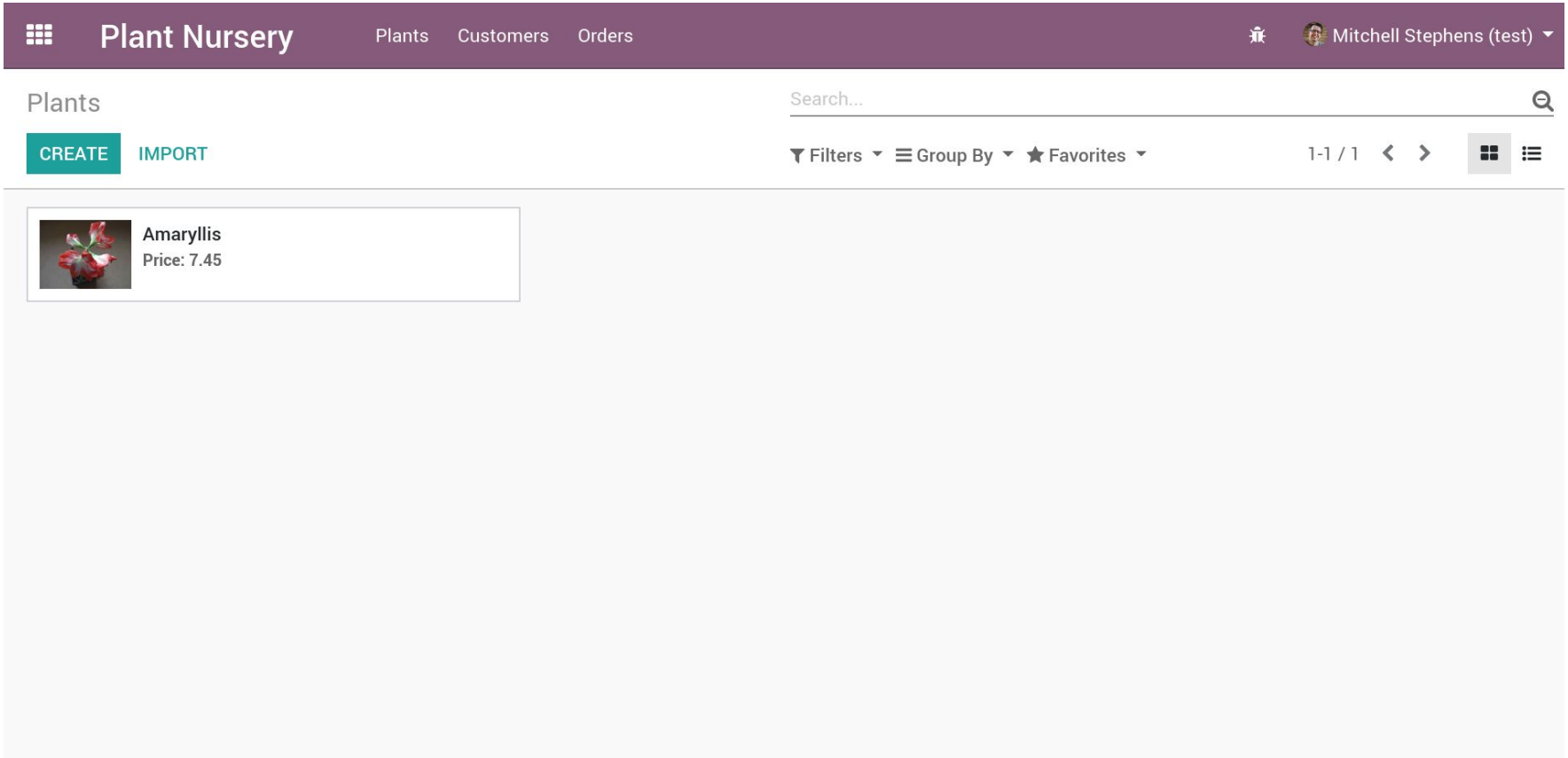
```
class Plants(models.Model):
    _name = 'nursery.plant'

    image = fields.Binary("Plant Image", attachment=True)
```

```
    <record model="ir.actions.act_window" id="action_nursery_order">
        <field name="name">Orders</field>
        <field name="res_model">nursery.order</field>
       <field name="view_mode">kanban,tree,form</field>
    </record>
```

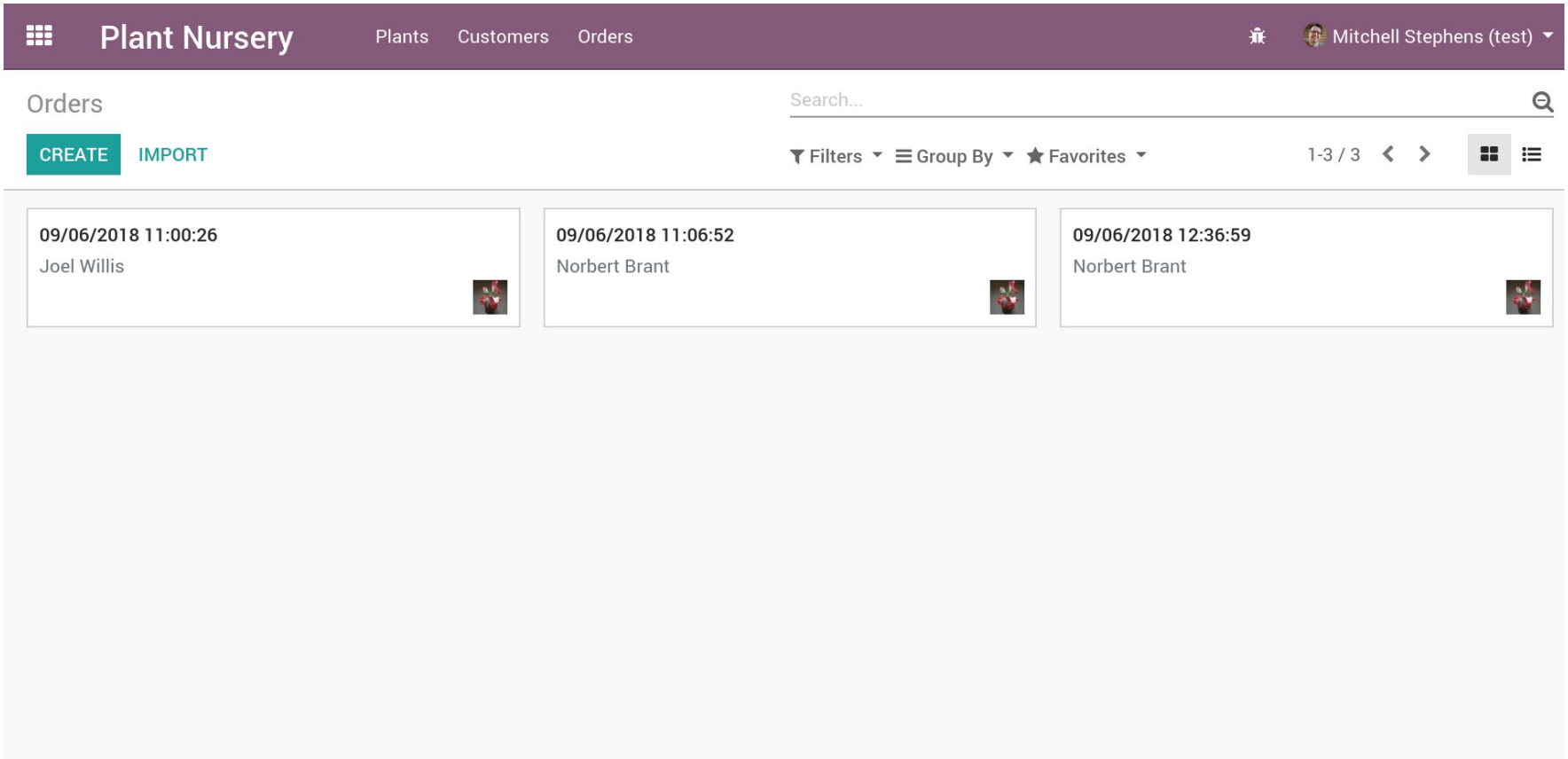
# Define a Kanban View

```
<record id="nursery_plant_view_kanban" model="ir.ui.view">
    <field name="name">nursery.plant.view.kanban</field>
    <field name="model">nursery.plant</field>
    <field name="arch" type="xml">
        <kanban>
            <field name="id"/>
            <field name="image"/>
            <templates>
                <t t-name="kanban-box">
                    <div class="oe_kanban_global_click">
                        <div class="o_kanban_image">
                            <img t-att-src="kanban_image('nursery.plant', 'image', record.id.raw_value)"/>
                        </div>
                        <div class="oe_kanban_details">
                            <strong class="o_kanban_record_title"><field name="name"/></strong>
                            <ul><li><strong>Price: <field name="price"></field></strong></li</ul>
                        </div>
                    </div>
                </t>
            </templates>
        </kanban>
    </field>
</record>
```

# Watch the result



# Watch the result



# Define a Search View

```
<record id="nursery_order_view_search" model="ir.ui.view">
    <field name="name">nursery.order.view.search</field>
    <field name="model">nursery.order</field>
    <field name="arch" type="xml">
        <search string="Search Orders">
            <field name="plant_id" string="Plant"/>
            <field name="customer_id" string="Customer"/>
            <field name="state"/>
            <filter string="Confirmed" name="confirmed"
                    domain="[('state', '=', 'confirm')]"/>
            <separator />
            <group expand="0" string="Group By">
                <filter string="State" name="group_by_state"
                        domain="[]" context="{'group_by':'state'}"/>
            </group>
        </search>
    </field>
</record>
```

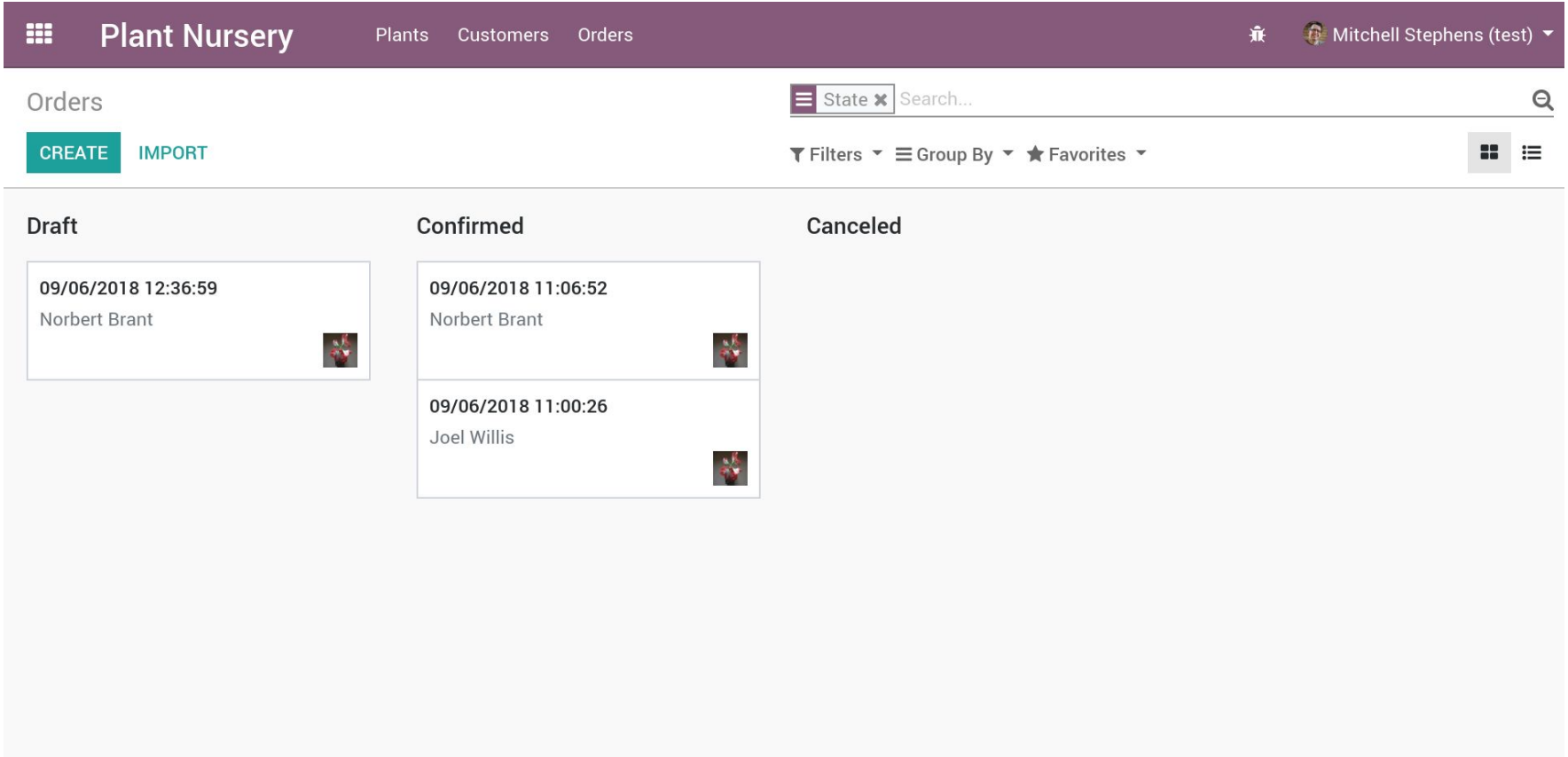
# Display all the states

```
class Order(models.Model):
    _name = 'nursery.order'

    state = fields.Selection([
        ('draft', 'Draft'),
        ('confirm', 'Confirmed'),
        ('cancel', 'Canceled')
    ], default='draft', group_expand="_expand_states")

    def _expand_states(self, states, domain, order):
        return [key for key, val in type(self).state.selection]
```

# Always group by state

```
<record model="ir.actions.act_window" id="action_nursery_order">
    <field name="name">Orders</field>
    <field name="res_model">nursery.order</field>
    <field name="view_mode">kanban,tree,form</field>
    <field name="context">{'search_default_group_by_state': 1}</field>
</record>
```

# Watch the result



2018

odoo EXPERIENCE

# Thank you.

https://github.com/tivisse/odoodays-2018

#odooexperience

Based on work from Thibault DELAVALLEE and Martin TRIGAUX, that was based on work from Damien BOUVY