Визард. Наследование. Контроллеры

Введение

В этом уроке мы ознакомимся с базовыми приемами работы с визардами и контроллерами.
Но прежде, давайте еще раз остановимся на наследовании в odoo.

Наследование

Вы уже знаете, что для того, чтобы расширить существующую модель своими полями и методами, Вы должны:
1. Создать свой класс, унаследовав его от **models.Model**
2. В специальном свойстве **_inherit** прописать имя модели, которую Вы расширяете.

Пример:

```
class appointmentcancel(models.Model):
    _inherit='medical.appointment'
    need_prepay=fields.Boolean(string='Нужна предоплата', compute='get_prepay')
    no_cancel=fields.Boolean(string='Не отменять прием автоматом')
```

В данном примере мы расширили модель двумя своими полями.

Казалось бы, вот и всё. Но нет. Для того чтобы начать пользоваться нашими новыми полями, нужно добавить их в какие-то view. Например, на форму или в список.

Как это сделать?
1. Плохой способ, но супер-быстрый. Рекомендую пользоваться только для прототипирования, но не для разработки.
В режиме разработки вызываете Edit Form (List, Search) View и просто добавляете поля.

Не забывайте, что это очень плохо и ненадежно. При обновлении базового модуля, модель которого Вы расширяете, все ваши изменения пропадут. Напомню, что если Вы захотите через пару лет переехать на новое ядро, то не сможете этого сделать, если понаделали изменений в формах в базе.
2. Правильный способ.
После того, как Вы подобрали место для своего поля, можно внести изменения в существующие XML форм или списков.
Для этого используется конструкция **XPATH** в вашем XML файле, где у Вас все view.

Например:

```
<xpath expr="//button" position="after">
    <button name="parse_email" type="object" string="Поиск и создание контакта"/>
</xpath>
```

Давайте разберем:
Инструкция xpath говорит нам так: окей, давай пойдем по DOM модели документа и отыщем там секцию button. Директива position сообщает, что вставку нужно сделать ПОСЛЕ данной секции.
Дальше идет сам код вставки.

Как искать поля:

```
<xpath expr="//field[@name='partner_shipping_id']" position="after">
```

Как искать поля, если на форме есть поле типа One2many?

```
<xpath expr="//field[@name='order_line']/tree/field[@name='name']" position="after">
```

Директива position может принимать значения after, before, inside, replace и attributes.
Думаю, что комментарии излишни.

Если Вам нужно скрыть существующие поля на форме, например было:

```
<field name="tag_ids" />
```

то Вы должны написать что-то вроде:

```
<xpath expr="//field[@name='tag_ids']" position="replace">
        <field name="tag_ids" invisible="1"/>
</xpath>
```

Почему мы заменяем простое удаление «деланием невидимым»? Всё очень просто. Скрываемое поле может участвовать в каком-нибудь значении домена или контекста и если поле убрать совсем, то «форма может сломаться». Особенно тяжело дается эта штука новичкам. Поэтому, если Вы не уверены в том, что не сломаете форму, а ведь бывают большие сложные формы, типа account.invoice, то, пожалуйста, скрывайте поля, а не удаляйте их.

Разумеется, запись в XML файле должна начинаться вот так:

```
<record id="view_crm_form_inherit" model="ir.ui.view">
    <field name="name">hms_booking.view_crm_form_inherit</field>
    <field name="model">crm.lead</field>
    <field name="inherit_id" ref="crm.crm_case_form_view_leads"/>
    <field name="arch" type="xml">
```

Вроде бы всё точно также, как у записей про формы и списки, кроме:

<field name="inherit_id" ref="crm.crm_case_form_view_leads"/>

Очевидно, что данная строка говорит о том, что у нас будет не новая форма или список, а мы что-то будем менять в форме crm.crm_case_form_view_leads
Вопрос: Где посмотреть, как называется view, в который мы хотим внести изменения?
Ответ: В режиме разработки вызываете Edit ___ view , и там будет поле именем.

Задание:
В форме crm.lead, которая связана с инициативой, с первичной заявкой, сделайте поле description невидимым через свой модуль snt-rassvet.

Визарды

Визард — это мощнейшая вещь, которая нужна для большей части автоматических обработок, процессов, подготовки данных для отчетов, для групповых операций.
Суть такая: создается временное окно, в нем готовятся данные для вызова какого-то метода, а потом этот метод вызывается.
Классический пример визарда: массовый пересчет цен.
Вы уже видели визарды в базовой версии odoo. Например один появляется по кнопке «Преобразовать в предложение» в модели crm.lead. Другой — появляется в модели sale.order при нажатии кнопки «Выставить счет».

Если быть кратким, то скажу так: «Через визарды всё и делается».

Создание визарда ничем не отличается от создания простой модели, за некоторыми исключениями, о которых мы и поговорим.

Первое отличие — наследуется не models.Model, а другой объект:

```
class WizardInterval(models.TransientModel):
    _name = 'medical.wizardinterval'
    _description = 'Свободные интервалы'
```

Временная модель. Данные хотя и хранятся в базе, но временно, они очищаются.
XML с формой пишется точно также, отличий нет.

Вызов визарда можно автоматически добавить в меню «Действие» нужной нам модели

```
<record id="action_doctor_wizard" model="ir.actions.act_window">
    <field name="name">Мастер записи к специалисту</field>
    <field name="res_model">medical.doctorwizard</field>
    <field name="view_mode">form</field>
    <field name="view_id" ref=" medical. doctorwizard _view_form"/>
    <field name="target">new</field>
    <field name="binding_model_id" ref="model_medical_appointment "/>
    <field name="binding_view_types">form</field>
</record>
```

binding_model_id — в какой модели будет появляться визард.
res_model — какая модель связана с визардом.

active_id и active_ids

Часто бывает так, что визард самодостаточен. Вы можете вызвать его не только из «действий», а например по кнопке на форме или из пунктов меню.
Но часто хочется передать в визард запись или записи, над которыми нужно совершить действие.

Например, отметить галочками записи и вызвать «Действие — пересчитать цены».
Для этого используются специальные переменные acitve_id и active_ids. Они содержатся в контексте.
Как Вы уже догадались, первая хранит идентификатор записи, из которой был вызыван визард, а вторая массив таких идентификаторов.

Значит, наша задача считать эти переменные и заполнить поля.
Сделать это можно через метод, которые задает значения по умолчанию.

doctor = fields.Many2one('medical.physician', string='Специалист', default=get_values)

В методе get_values пропишите обращение к контексту и сгенерируйте значения.
Есть более красивый метод. Через переопределение метода default_get:

```
def default_get(self, field_names):
    defaults = super(DoctorWizard, self).default_get(field_names)
    app_id=self.env.context['active_id']
    appointment = self.env['medical.appointment'].search([ ('id', '=', app_id)])
    defaults['doctor'] = appointment.doctor_id.id
return defaults
```

Т.е. Вы всегда можете обратиться к active_id через:
self.env.context['active_id']

Еще одна особенность визардов в том, что после нажатия на кнопку будет вызван метод. И если он ничего не возвращает, то форма визарда закроется автоматически.
Поэтому нужно написать хелпер, вот такой:

```
def _reopen_form(self):
    self.ensure_one()
    return {
          'name': ' Мастер записи к специалисту ',
          'type': 'ir.actions.act_window',
          'view_type': 'form',
          'view_mode': 'form',
          'res_model': self._name,
          'target': 'new',
          'res_id': self.id,
          'context': self.env.context,
      }
```

И Ваши методы, после которых форма должна остаться на экране, должны возвращать:

return self._reopen_form()

В остальном отличий от других моделей никаких нет.

Где посмотреть, как устроен работающий визард?
Лучше всего учиться у разработчиков odoo. Поэтому поищите их в папках wizard внутри нативных модулей: account, sale, crm.

Контроллеры и интеграции

Odoo у нас веб-приложение, построенное на werkzeug, похожее на flask, и принципы здесь все те же.

Этот урок у нас обзорный, потому как тема очень глубокая и мы остановимся только на тех вещах, которые скрыты от общественности, на которые лично я потратил массу времени.
Итак, давайте посмотрим на контроллеры.

Фронтенд системы - это JS приложение, мы его не трогаем.
Есть модуль website. Он может рендерить qweb шаблоны и генерировать страницы уже с разметкой, с темой сайта и пр.
Но базовая вещь — контроллеры, работает всегда, независимо от того, есть у Вас веб-сайт или нет.
Что такое контроллер — это метод, который вызывается при обращении к определенному URL. Если такое обращение случилось, то веб-сервер вызывает наш метод. Он может вернуть HTML, может вернуть ошибку или JSON.

Давайте посмотрим, как выглядит простейший контроллер. Давайте зайдем в наш модуль, далее папка controllers, создадим файл controllers.py, не забыв прописать его в нужных __init__.py

Простой HTTP контроллер

```
from odoo import http, fields
class HmsTimesheet(http.Controller):
    @http.route('/timesheet',  auth='public')
    def index(self,delta, filial, **kw):
```

Впервые мы увидели, что импортируется модуль http.
Этот модуль имеет всё, что нужно: request, response и пр.

Декоратор *@http.route('/timesheet', auth='public')* указывает на то, что слушать мы будем адрес /timesheet, аутентификация не нужна.
**kw — содержит ЗАПРОС
В данном случае delta, filial — параметры запроса, они видны в нашей адресной строке.

Далее мы должны в теле метода сгенерировать HTML и отдать его через return.

Попробуйте сделать свой контроллер, который вернет Hello, World!

Контроллер для website

Такие контроллеры могут пользоваться всем, что нам даёт веб-сайт.
Мы можем взять шаблон QWEB, Вы с ним уже знакомы, наполнить его данными и вывести в качестве страницы.

Давайте попробуем.
Давайте сделаем шаблон и поместим его в файл template.xml, не забыв прописать его в манифесте, чтобы он загрузился.

```
<odoo>
<template id="hello" name="Hello World">
  <t t-call="website.layout">
    <h1>Hello World!</h1>
  </t>
</template>
</odoo>
```

В файле controllers.py пишем:

```
from odoo.http import request

@http.route('/hello', auth='public', website=True)
def hello(self, **kwargs):
    return request.render('snt-rassvet.hello')
```

Но в нашем шаблоне не было никаких данных. Как передать данные в шаблон?

Давайте допишем наш метод:

```
@http.route('/hello', auth='public', website=True)
def hello(self, **kwargs):
    my_obj=request.env[your_model].search([])
    return request.render('snt-rassvet.hello',{„obj“:my_obj})
```

А в шаблоне сделаем привычное:

```
 <t t-foreach="obj" t-as="o">
```

И теперь к полям можно обратиться:

```
 <span t-field="o.name" />
```

В QWEB для веб-сайта есть отличия от QWEB для отчетов в стилях, в дополнительных опциях. Все отличия описаны в документации.

Важное замечание. В контроллерах вместо self.env нужно использовать request.env
Если Вы работаете без аутентификации (не auth='user'), но пробуете обратиться к моделям, для которых стоят определенные права, нужно пользоваться sudo
Например так:
request.env[model].sudo().search([])
В этом случае обращение к моделям odoo будет из-под админа.

JSON контроллер и его подводные камни

По умолчанию у нас создается http контроллер. Но мы можем сделать и JSON:

```
class Alice(http.Controller):
   @http.route('/alice/', auth='public',type='json',  csrf=False)
```

csrf=False говорит о том, что мы не проверяем токен.

Как прочитать JSON?
До 17 версии: *req = http.request.jsonrequest*
Начиная с 17 версии: *req = http.request.get_json_data()*

Как отправить JSON в ответ?
Заполните словарь response_main и верните его:
return json.dumps(response_main)

В чем подвох?
В том, что odoo под JSON подразумевается JSON-RPC 2.0, а это значит, что прочитать входящий JSON мы можем без ограничений, а вот отправить в ответ можем только в формате JSON-RPC 2.0
Это означает, что удаленный сервис не получит ответ в том формате, в котором он хочет, если он не работает с JSON-RPC. А почти все сервисы имеют свои спецификации.

И это есть затруднение: весь ваш словарь целиком, составленный в точности по спецификации будет добавлен в специальное поле и снабжен дополнительной информацией.
Я пробовал переопределить **_json_response** своим методом, но это «ломало» часть штатных функций, потратил пару вечеров, но дальше не стал разбираться. В коде метода **_json_response** очевидным образом дописывается к штатному werkzeug методу обертка в виде полей JSON RPC.

Поэтому, самый простой метод для интеграций с другими сервисами - **сделать прокладку в виде сервера flask**, который будет получать обычный JSON от сторонних сервисов, трансформировать его в JSON-RPC (или не делать этого), отдавать его в odoo, получать ответ и транслировать уже его, очищенный от JSON-RPC ключей на удаленный сервис.

Что такое портал?

Портал — это расширение модуля **website**, которое умеет показывать счета, заказы, проекты и прочую информацию. Предоставляет сервис логина-пароля для пользователя сайта. В общем, в простонародье — личный кабинет.

Порой требуется что-то вывести в личный кабинет такое, чего там нет.
У портала есть глобальные значения, которые можно расширить. Мы с Вами видели, как контроллер передает словарь значений методу *request.render* и нам бы надо этот словарь расширить.
Например, мы хотим вывести долг клиента.

from odoo.addons.portal.controllers.portal import CustomerPortal, pager as portal_pager, get_records_pager

```
class CustomerCredit(CustomerPortal):

    def _prepare_portal_layout_values(self):
        values = super(CustomerCredit, self)._prepare_portal_layout_values()
        partner = request.env.user.partner_id
        credit=partner.credit

        values.update({
            'credit': credit,

        })
        return values
```

И тогда в QWEB шаблон под названием ***portal.portal_my_home*** *мы можем добавить:*

```
 <t t-if="credit&gt;0">
        <h3 class="page-header">Мой долг:  <t t-esc="credit"/> р. </h3>
        <p>Просим погасить долг в ближайшее время. Если сумма долга составит более 10 000р
или  количество неоплаченных счетов 2 и более,
        компаний приостановит Ваше обслуживание!</p>
        <p>Вся  запись будет автоматически отменена.</p>
        <p> WhatsApp для сверки платежей: +7 999 111-22-33</p>
      </t>
      <t t-if="credit&lt;0">
        <h3 class="page-header">Мой аванс:  <t t-esc="credit*-1"/> р.</h3>
      </t>
      <t t-if="credit==0">
        <h3 class="page-header">Ура! Долгов нет! </h3>
      </t>
```

Конечно, ключевой момент это **<t t-esc="credit"/>**
Мы добавили в словарь значений это поле и теперь можем им пользоваться в портале.

Задания к уроку:

1. Сделайте простой контроллер, который выдает Hello World.
2. Сделайте тоже самое, только под веб-сайт.