MK.Lab

Урок 2

«Модули»

Сейчас, когда Вы только что установили систему, очень хочется ее пощупать, посмотреть возможности, попробовать установить модули. Чем сейчас и займемся.

Основная концепция

Odoo состоит из ядра системы и модулей. Разработчики постарались включить в ядро только самый минимум: сам фреймворк и базовые функции типа http сервера и пр. Всё остальное вынесено в отдельные модули. Вы можете их найти и посмотреть. Они все находятся в папке addons вашей инсталляции.

Модульная концепция чрезвычайно хороша. Это очень удобно для понимания системы, для написания своих доработок.

В маркетплейсе odoo более 12 000 готовых модулей на все случаи жизни. Они бывают платные и бесплатные.

Базовые модули

Базовые модули идут в комплекте с самой системой. Сейчас, когда у вас свежая чистая инсталляция, в папке addons Вы можете найти полный набор базовых модулей.

Когда Вы запустите систему, сделаете новую базу, то в меню «Приложения» Вы увидите список приложений.

Модули бывают типа «приложение» и остальные. Деление весьма грубое и нужно только для классификации. Скажем, не будете же Вы микромодуль, который изменяет

тип поля, называть приложением, рисовать к нему иконку и пр. А вот если Вы разработали готовое решение для какой-либо задачи, то это уже приложение.

Режим разработки

Перейдите в раздел «Настройки» главного меню и найдите пункт «Режим разработки», Developer Mode. В зависимости от версии odoo или flectra местоположение этого пункта может немного меняться.

Теперь, когда Вы активировали этот режим, посмотрите на главное меню. В нем появился раздел «Технический». Посмотрите, что там есть. А еще появилось меню «Жучок»  . В нем собраны инструменты разработчика.

Упражнение:

Найдите раздел «База данных» и подраздел «Модели». Посмотрите, как устроены модели данных и сколько их.

Установка сторонних модулей

Пошаговая инструкция такова:

1. Нужно найти нужный Вам модуль. Например, скачать из официального маркетплейса. Самое главное, верно выбирайте версию. Это принципиально, потому как от версии к версии API меняется очень существенно. Скоро Вы научитесь сами портировать модули с одной ближайшей версии на другую, но пока выбирайте только ту версию, которая у Вас.

2. Нужно создать отдельную папку внутри инсталляции odoo, например odoo/myaddons, чтобы не путать свои модули с базовыми. Конечно, большая часть разработчиков этим пренебрегает, сваливая все модули в одну папку addons. Но это сильно навредит, когда нужно будет делать переезд с сервера на сервер или с целью ежедневных автоматических бэкапов ваших доработок.

3. В созданную папку нужно распаковать содержимое архива модуля так, чтобы внутри папки с модулем уже был файл manifest.py, а не какие-то еще группы папок верхнего уровня. Часто архивы могут содержать сразу несколько модулей из-за зависимостей, поэтому разработчики их поставляют сразу.

4. В конфиге Вы должны прописать путь к вашей папке с модулями, иначе odoo будет искать модули только в папке addons.

Добавьте в конфигурационный файл строку, например такую:

addons-path=/opt/odoo/odoo18/myaddons

Также Вы можете запустить систему с параметром:

$python3 odoo-bin --addons-path="/opt/odoo/odoo18/myaddons"

Добавьте параметр **--save**, чтобы сохранить свои изменения.

5. В режиме разработки нужно войти в меню «Приложения» и найти подпункт «Обновить список приложений». В некоторых версиях odoo этот подпункт может находится в меню «Настройки/Технические». Ваш скачанный модуль должен появиться в списке. Возможно, нужно снять фильтр «Приложение».

6. Нажмите «Установить».

Важное замечание про Flectra.

Т.к. Флектра это форк 11-ой версии, к ней подходят модули от 11 версии. Но каждый модуль нужно обработать специальным скриптом автоматического портирования. Он прилагается к этому уроку.

Скриптом пользуются так:

$python3 odoo_flectra.py /<path_to_the_module>

Что делает скрипт? Он везде в исходном коде заменяет odoo на flectra так как надо.

Задание №2.1

Установите в свою систему модули Продажи, Закупки, Склад, Контакты и CRM.

Найдите в официальном маркетплейсе какой-нибудь модуль, который Вам нужен по роду деятельности. Конечно, нужно начать с бесплатного модуля.

Установите этот модуль в вашу базу. Посмотрите, какие изменения произошли.

Как правильно искать модули

Есть всего два места, где можно скачать или купить проверенные модули:

Официальный маркетплейс:

https://apps.odoo.com/apps/modules

И OCA (Odoo Community Association)

https://odoo-community.org/

И там, и там поиск работает так себе, поэтому сначала нужно:

1. Описать свою задачу в трех словах. Например «Печать деталей заказа».

2. Перевести эти слова на английский с учетом специфики odoo. Все объекты должны называться так, как они называются в системе на английском. Например, заказ это sale.order. Значит, наша первая итерация поискового запроса будет print order detail.

3. Поискать поиском на маркетплейсе.

4. Поискать в Google добавив перед запросом слово odoo. Например, odoo invoice mass reconcile в случае поиска решения по автоматической раскладке платежей по счетам.

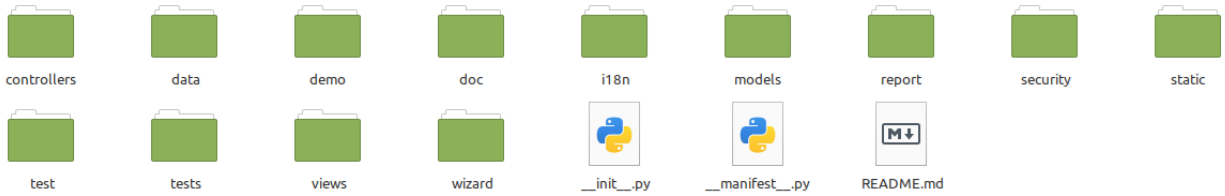
Искусство поиска модулей напрямую влияет на ваши трудозатраты, потому как чаще выгоднее что-то «допилить под себя», чем писать с нуля. Особенно на начальном этапе.

Архитектура модуля

Давайте исследуем, как устроен модуль. Возьмите любой модуль, откройте папку с ним и посмотрите, какие в нем есть папки и файлы.

На начальном этапе мы с вами узнаем базовые вещи.

Ниже скрин одного из базовых модулей:



Файл **manifest**.py это описание нашего модуля для системы.

Если этот файл будет содержать ошибки, то будут проблемы с модулем. Система может его не видеть, может искать не в тех местах компоненты и файлы.

Секция **depends** указывает на зависимости. Без каких модулей наш модуль работать не будет.

Секция **data** указывает на список файлов XML, в которых содержатся экранные формы, отчеты (печатные формы), меню и прочие элементы интерфейса вашего модуля.

Папка **security** содержит файл ir.model.access.csv, в котором описаны права для групп пользователей, так же в xml файлах могут быть созданы сами группы пользователей, в дополнение к существующим.

Папка **models** содержит исходные коды модуля на python. В этих файлах все модели данных, классы, методы. Т.е. вся логика.

Папка **controllers** также содержит классы на python, которые нужны для веб-сайта. Если говорить упрощенно, то веб-сайт построен на котроллерах, каждый из которых отрабатывает свой URL, например *my*/invoice/pay. Веб-сайт это не фронтенд системы, это веб сайт для клиентов, модуль website.

Папка **wizard** содержит особые временные модели, которые используются для отображения диалоговых окон. Структура визардов повторяет структуру обычных моделей Odoo, но не сохраняет внесенные данные в базе данных.

Папка **i18n** содержит переводы. Вы можете сделать модуль на английском (что крайне рекомендуется для тех, кто делает модули на продажу), и добавить русский язык уже позднее, включив его в папку i18n.

Папки **data, reports и views** содержат XML файлы, в которых находятся меню, формы, отчеты вашего модуля.

Внутреннее устройство модуля

Каждый модуль состоит из нескольких основных сущностей:

1. Модели (Models).

2. Пользовательские представления (View)

3. Отчеты (Reports)

4. Права и группы доступа (Security)

Есть модули без View и отчетов. Мы сегодня такой модуль напишем сами.

Модель в odoo

Это класс Python, в котором содержатся поля вашей бизнес-модели данных и методы.

Например, если Вы делаете приложение, в котором есть книги в библиотеке, то можете сделать модель с полями:

author, ISBN, date_published, archived

И методом:

do_archive, который, например, будет устанавливать значение поля archived как True.

Простая модель может выглядеть так:

```
from odoo import models, fields, api
class books_in_the_library (models.Model):
    _name='library.library_books'
    name=fields.Char(string='Наименование книги)
    author=fields.Char(string='Автор')
    isbn =fields.Char(string='ISBN')
```

```
date_published=fields.Date(string='Дата публикации')
archived=fields.Boolean(string='Архив')
def do_archive(self):
    for s in self:
        s. archived = True
```

Мы видим, что мы наследуем класс models.Model и добавляем в него свои поля.

Классы fields и models предоставляются нам фреймворком odoo, о чем мы и указываем в самом начале нашего файла:

```
from odoo import models, fields
```

Пользовательское представление в odoo

Это список, называется List View. В списке представляются данные, когда на экране не 1 запись, а много. Например, список заказов.

Есть еще форма Form View. Это пользовательская форма.

В XML простая форма может выглядеть так:

```
<record id="library.library_books_form" model="ir.ui.view">
<field name="name"> library.library_books_form</field>
<field name="model"> library.library_books</field>
<field name="arch" type="xml">
 <form>
 <sheet>
     <h1><field name="name"/></h1>
     <group string="Информация о книге">
             <field name="author"/>
             <field name="isbn"/>
             <field name="date_published"/>
             <field name="archived"/>
     </group>
 </sheet>
 <footer>
         <button type="object" name="do_archive" string="Архивировать" class="oe_highlight"/>
         <button special="cancel" string="Отменить"/>
  </footer>
 </form>
</field>
</record>
```

Пока, возможно, не понятно, что означают все эти поля, но принцип такой: в XML содержатся записи, о чем нам говорит тег <record>, внутри записи указано, что это за запись, в каких полях, что содержится.

В базе данных есть поле, в котором содержится XML разметка View.

И эта самая разметка содержится внутри тега *<field name="arch" type="xml">*

Т.е. у нас XML с двойным дном. Файл содержит записи в XML формате, среди которых в некоторых полях также в формате XML собственно сами формы и списки. Запись про запись.

Если мы хотим описать процесс разработки приложения в двух словах, то имеем следующее:

1. Создаем модель, прописываем все поля, методы.

2. Делаем для пользовательские представления: списки, формы и меню.

Тут важное замечание для ленивых. Иногда модель и состоит всего из одного-двух полей. Например, какой-нибудь динамический справочник, классификатор или теги. Odoo не требует, чтобы вы сделали формы и представления, если их нет, то она сделает их сама, но за красоту, так сказать, не отвечает. Все поля будут в том порядке, как Вы их написали, группировки и фильтров не будет и пр.

Золотое правило разработчика

Нельзя менять исходный код других модулей. Кроме, разумеется, случаев, когда Вы нашли ошибку. Такое бывает, но очень редко. Если Вы нашли ошибку в базовом модуле, нужно на GitHub создать Issue, и там предложить свою правку, а не править тихо у себя.

А как же тогда дорабатывать сторонние модули?

Если модуль базовый, то не трогайте его. Ниже будут описаны механизмы, которые позволяют вносить своим кодом правки в код другого модуля.

Если Вы скачали модуль для целей доработки, то потрудитесь, переименуйте его, сделайте «своим» и дорабатывайте тогда уже без проблем.

А как же быть, если нужно, скажем, изменить тип поля в другом модуле? Нужно писать свой модуль.

1. Мы можем переопределить любое поле, любое свойство или любой метод любого класса. Для этого есть механизм наследования.

2. Мы можем внести изменения в любой XML, для этого есть механизм XPATH.

Вы, наверное, уже пробовали войти в Developer Mode, зайти в любую форму и из меню Develoer (с жучком) выбрать Edit Form View. Откроется как раз тот самый XML, который мы рассматривали выше.

Вы можете вносить туда изменения из интерфейса, но помните, что обновление модуля все ваши изменения уничтожит.

Вы можете тестировать на ходу, например, у заказчика или в режиме сессии подбирать удобные варианты интерфейса, но не забудьте перенести все исправления в модуль.

После установки Вашего модуля изменения должны примениться, после удаления модуля, изменения должны произойти в обратную сторону, вернув всё в исходное положение. Чтобы откатить такие изменения, как изменение типов полей или изменения в интерфейсе, необходимо также обновить модули, от которых они наследовались.

Сейчас мы этим и займемся.

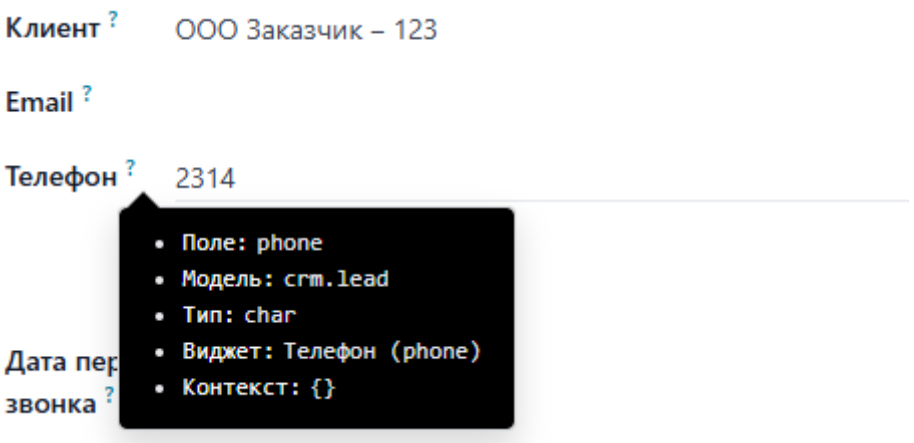
Модуль — доработка другого модуля

В следующем уроке мы будем делать модуль с нуля с полями, методами и пр.

А сейчас мы сделаем вот что.

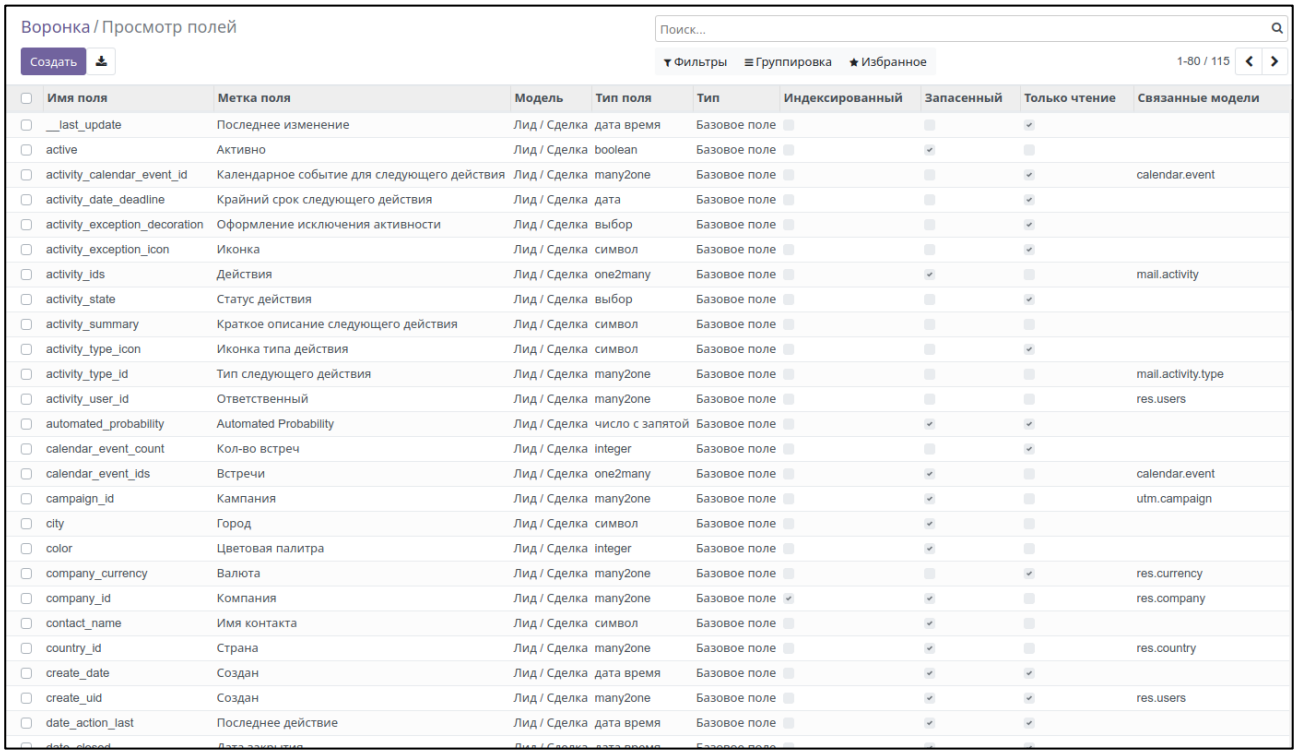
Откройте модуль CRM. Откройте инициативы(воронки) и создайте новую. В Odoo поле описание (description) типа HTML. Мы хотим изменить тип на Текст.

Когда Вы находитесь в Developer Mode, подсказки показывают тип поля, и дают массу полезной информации. Наведите курсор на знак вопроса возле метки поля и появится hint.



До 17 версии Odoo эта функция была реализована по наведению курсора на само поле, а после только по знаку «?».

Также в Developer Mode удобно смотреть в какой модели Вы сейчас находитесь, и какие у нее поля. Выберете «Жучок»- «Просмотр полей»



Воронка / Просмотр полей

Поиск...

Создать

Фильтры Группировка Избранное 1-80 / 115

Имя поля	Метка поля	Модель	Тип поля	Тип	Индексированный	Запасенный	Только чтение	Связанные модели
__last_update	Последнее изменение	Лид / Сделка	дата время	Базовое поле			✓	
active	Активно	Лид / Сделка	boolean	Базовое поле		✓		
activity_calendar_event_id	Календарное событие для следующего действия	Лид / Сделка	many2one	Базовое поле			✓	calendar.event
activity_date_deadline	Крайний срок следующего действия	Лид / Сделка	дата	Базовое поле			✓	
activity_exception_decoration	Оформление исключения активности	Лид / Сделка	выбор	Базовое поле			✓	
activity_exception_icon	Иконка	Лид / Сделка	символ	Базовое поле			✓	
activity_ids	Действия	Лид / Сделка	one2many	Базовое поле		✓		mail.activity
activity_state	Статус действия	Лид / Сделка	выбор	Базовое поле			✓	
activity_summary	Краткое описание следующего действия	Лид / Сделка	символ	Базовое поле				
activity_type_icon	Иконка типа действия	Лид / Сделка	символ	Базовое поле			✓	
activity_type_id	Тип следующего действия	Лид / Сделка	many2one	Базовое поле				mail.activity.type
activity_user_id	Ответственный	Лид / Сделка	many2one	Базовое поле				res.users
automated_probability	Automated Probability	Лид / Сделка	число с запятой	Базовое поле		✓	✓	
calendar_event_count	Кол-во встреч	Лид / Сделка	integer	Базовое поле		✓	✓	
calendar_event_ids	Встречи	Лид / Сделка	one2many	Базовое поле		✓		calendar.event
campaign_id	Кампания	Лид / Сделка	many2one	Базовое поле		✓		utm.campaign
city	Город	Лид / Сделка	символ	Базовое поле		✓		
color	Цветовая палитра	Лид / Сделка	integer	Базовое поле		✓		
company_currency	Валюта	Лид / Сделка	many2one	Базовое поле			✓	res.currency
company_id	Компания	Лид / Сделка	many2one	Базовое поле	✓	✓		res.company
contact_name	Имя контакта	Лид / Сделка	символ	Базовое поле		✓		
country_id	Страна	Лид / Сделка	many2one	Базовое поле		✓		res.country
create_date	Создан	Лид / Сделка	дата время	Базовое поле		✓	✓	
create_uid	Создан	Лид / Сделка	many2one	Базовое поле		✓	✓	res.users
date_action_last	Последнее действие	Лид / Сделка	дата время	Базовое поле		✓	✓	

Итак, в какой модели Вы сейчас находитесь? Да, это crm.lead

Зайдите в Настройки -> Технический -> Структура базы данных -> Модели. Найдите модель crm.lead

Найдите поле Description. Какого оно типа?

Попробуйте заменить тип на Text.

Неудача. Нельзя через интерфейс менять тип поля.

Давайте напишем мини-модуль, который изменит тип поля.

Автоматическая генерация шаблона модуля

Для того, чтобы автоматически сделать структуру папок модуля, запустим нашу систему со специальным параметром:

$python3 odoo-bin scaffold crm_test

Создаст в корневой папке нашей инсталляции папку crm_test, в которой уже будет всё, что надо.

Мы изменим два файла: manifest и models и попробуем установить модуль.

Итоговое задание.

Переместите только что созданную папку crm_test в папку с вашими модулями.

В файле manifest.py напишите примерно так:

```
# -*- coding: utf-8 -*-
{
    'name': "crm_test",

    'summary': """Небольшие доработки вроде добавления полей""",

    'description': """Long description of module's purpose""",
    'author': "MKLab",
    'website': "http://inf-centre.ru",

# Categories can be used to filter modules in modules listing
# Check https://github.com/odoo/odoo/blob/18.0/odoo/addons/base/data/ir_module_category_data.xml
# for the full list
    'category': 'Uncategorized',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['base','crm'],

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

В файле models.py напишите так:

```
# -*- coding: utf-8 -*-


from odoo import models, fields, api


class crm_lead(models.Model):
    _inherit='crm.lead'
    description=fields.Text(string='Описание')
```

Вы уже увидели, что есть директива *_inherit='crm.lead'.*

Это и есть механизм наследования. Ваш класс унаследует crm.lead.

Т.е. не наследование в ООП и Python смысле, а в терминах фреймворка odoo.

Сохраните и установите модуль.

Зайдите в crm, посмотрите, изменилось ли поле Описание.

После обновления или установки модулей нужно обновлять страничку в браузере, чтобы JS приложение, которое является нашим веб-клиентом заново загрузило все XML формы и пр.

Изменилось. Отлично.

Теперь выберете какое-нибудь другое поле, и также попробуйте изменить его тип. Допишите в классе соответствующую строку.

Чтобы обновить модуль, Вы должны перезапустить odoo. И потом войти в приложения и нажать «Обновить».

Для целей разработки это неудобно. Поэтому удобно запускать odoo с параметром, который дает обновление нужного модуля.

Делается это так:

$python3 odoo-bin -d database_name -u your_module

Указание базы обязательно, потому как иначе система не узнает, а в какой базе обновлять модуль.

Имя базы можно посмотреть в менеджере:

http://<yourhost>:8069/web/database/manager

Порт, 7073 для Флектры, 8069 для odoo.

При удалении модуля желательно выполнить обновление модуля, который описывал изменение поля изначально, в нашем примере это модуль CRM.

Преподаватель ждет от Вас:

1. Скрин установленного скачанного с маркетплейса модуля.

2. Скрин установленного своего модуля, который меняет тип полей. И скрин формы, где видно, что тип полей изменен.

3. Скрин формы в crm.lead после удаления Вашего модуля.