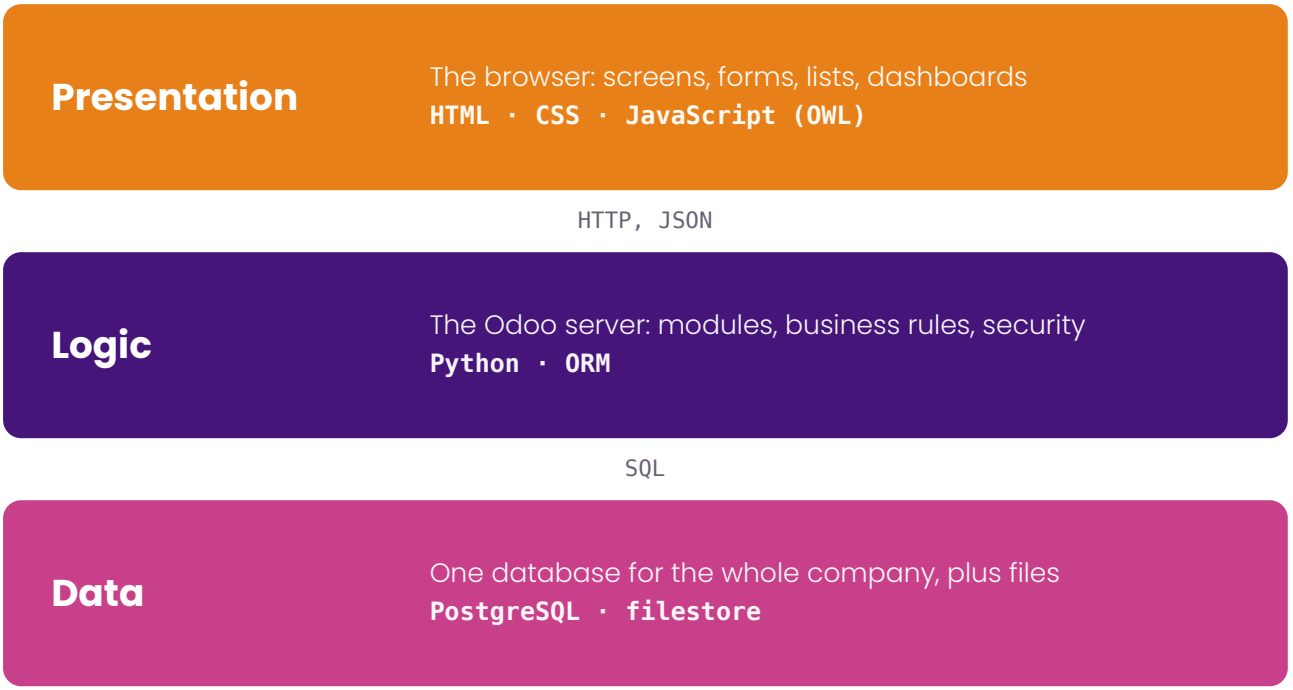
Training guide
Odoo 19

Odoo basics

From zero: what Odoo is, how it is built, why Python, PostgreSQL and open source, what a module is, and the first things to configure.



Functional and technical, with examples. Written for someone in their first weeks with Odoo.

Powered by **Amir Akbari**

October 2026

The short version

Odoo is one piece of software that runs a whole company: sales, stock, purchasing, accounting, people. Everything is stored in one database, and everything is built from small add-ons called modules.

Question	Answer in one sentence
What is it?	An ERP: a suite of business apps that share one database, so a sale, its delivery and its invoice are the same data seen by different departments.
How is it built?	In three layers: a web interface in the browser, a server written in Python, and a PostgreSQL database.
What is a module?	A folder of code and data that adds or changes one feature. Odoo itself is nothing but a few hundred modules, and your customisations are modules too.
Why Python?	It is readable and flexible enough to let one module extend another without editing it, which is the idea Odoo is built on.
Why PostgreSQL?	It is free, strict about data, safe under many users at once, and Odoo is written for it alone.
Why open source?	It is the business model: a free core spreads widely and attracts developers, and Odoo earns from the Enterprise edition, hosting and services.
What does it cost me?	The Community edition is free of licence fees. Implementation, hosting, upgrades and people are never free.

How to read this guide

Odoo work has two sides, and this guide always says which one is speaking. Functional is what a consultant or key user does on the screen: configure, enter, check. Technical is what happens underneath: models, fields, files, commands. A good Odoo person reads both, even if they work on one.

What is in this guide

A What Odoo is

1 Odoo in plain words
2 History and versions
3 Editions and hosting
4 Why open source?

B How Odoo is built

5 Architecture: three layers
6 Why Python, why PostgreSQL
7 Modules
8 A complete small module
9 Inheritance
10 The ORM in ten minutes

C Using and configuring

11 The map of apps
12 Around the screen
13 First configuration
14 Users and security
15 Data in, data out
16 Automation without code
17 Developer mode

D Running and judging

18 Installing and running
19 Advantages and limits
20 Your first month
21 Glossary and self-check

Scope: Odoo 19, released in autumn 2025. Menu paths, commands and code follow that version; a new major version appears every year, so check what has moved if you work on another one. Examples use the invented company of this series, Sahand Industrial Co., which makes and sells electrical panels.

Part A What Odoo is

1 Odoo in plain words

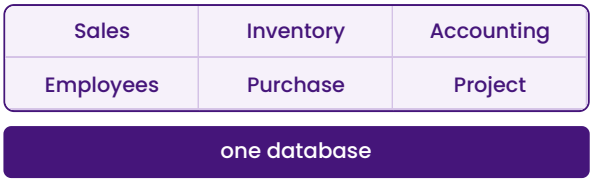
ERP stands for Enterprise Resource Planning. Behind the heavy name is a simple promise: every department works in the same system, on the same data.

Without an ERP: islands



Each tool has its own list of customers and products. The same order is typed three times, and the three copies drift apart. Nobody can say what was sold, shipped and invoiced without a meeting.

With Odoo: one system



One list of customers, one list of products. The order is entered once; the delivery and the invoice are created *from* it. Each department sees its own part of the same record.

One sale, followed through the company Functional

Sahand sells ten panels to Alborz Trading Co. In Odoo that single event travels through five apps without anyone retyping it.

	Who	App	What they do, and what Odoo does for them
1	Salesperson	CRM	Records the opportunity and the conversations with the customer.
2	Salesperson	Sales	Turns the opportunity into a quotation, sends it, and confirms it when the customer accepts. *Odoo creates a delivery order.*
3	Storekeeper	Inventory	Picks the ten panels and validates the delivery. *Odoo lowers the stock and marks the order as delivered.*
4	Accountant	Invoicing	Creates the invoice from the order with one click and confirms it. *Odoo writes the accounting entry.*
5	Accountant	Accounting	Matches the customer's payment with the invoice. *Odoo marks the invoice as paid.*

Three words that describe Odoo

Integrated
All apps read and write the same records. There is no "interface" between Sales and Inventory to maintain.

Modular
You install only the apps you need and add others later. Every feature, standard or custom, is a module.

Open
The source code of the core is public. Anyone can read how it works, extend it, or host it.

> **What Odoo is not**
> It is not only accounting software: accounting is one app among dozens. It is not finished when it is installed: an empty Odoo knows nothing about your company, and setting it up is a project called an **implementation**. And it is not magic: a messy process stays messy until someone redesigns it.

2 A short history, and how versions work

Odoo is about twenty years old and has changed its name twice. The history explains several things that surprise newcomers.

2005
Fabien Pinckaers starts **TinyERP** in Belgium, written in Python from the first day.

2008–09
Renamed **OpenERP**. A web interface joins the desktop client.

2014
Version 8 becomes **Odoo**: website builder, shop and point of sale go beyond ERP.

2015
Version 9 splits into **Community** and **Enterprise** editions.

2020
Version 14 introduces **OWL**, Odoo's own JavaScript framework.

2025
Odoo 19, the version this guide describes.

You will still meet the old names. Many technical identifiers start with `openerp` in older code, the company is Odoo S.A., and forum answers from ten years ago describe screens that no longer exist. Always check which version an answer was written for.

How versions work

Term	Meaning
Major version	17, 18, 19. One per year, presented each autumn. Each brings new features, changed screens and changes in the code.
Intermediate version	19.1, 19.2 and so on, also called SaaS versions. They exist only on Odoo's own cloud and are not available for self-hosted servers.
Supported versions	Odoo maintains the most recent major versions with bug and security fixes. Older ones still run, without fixes. Check the current list before choosing a version for a project.
Update	Installing the latest fixes of the *same* version, or reloading a module after its code changed. Routine.
Upgrade	Moving a database from one major version to the next, for example 18 to 19. A project: the data is converted and every custom module must be adapted.

> **Why the yearly rhythm matters to you**
> A company that customises Odoo heavily pays for it again at every upgrade, because custom code written for 18 must be reviewed for 19. This is the single best argument for using standard features wherever they are good enough.

3 Editions and hosting

Two questions are decided before any project starts: which edition, and where the server runs. They are independent, with a few combinations that are not allowed.

Two editions

	Community	Enterprise
Licence	Open source, LGPL version 3. Free of licence fees.	Proprietary. Paid subscription, per user.
What it is technically	The complete framework and the core apps.	Community *plus* a second set of modules laid on top. The same server, more folders.
Typical apps included	CRM, Sales, Invoicing, Inventory, Purchase, Manufacturing, Project, Website, eCommerce, Point of Sale, Employees, Time Off, Expenses.	Everything in Community, and for example full Accounting, Studio, Documents, Sign, Helpdesk, Planning, Payroll, Quality, the mobile apps.
Accounting	Invoices, bills, payments, journal entries and the chart of accounts.	Adds financial reports, bank synchronisation, the reconciliation screen, assets and budgets.
Support and upgrades	From the community, or from a partner you pay.	Odoo's support and its upgrade service are part of the subscription.
Who chooses it	Companies with their own technical team, tight budgets, or needs that the core covers.	Companies that want the extra apps and a vendor behind them.

The exact list of apps in each edition changes from version to version. Check Odoo's editions page for the version you are implementing.

Three ways to host

	Odoo Online	Odoo.sh	On-premise
What it is	Odoo's own cloud. You get a web address and nothing to install.	Odoo's platform for hosting *your* code: servers, backups and test copies managed for you.	Your own server, in your office or at any hosting company.
Who runs the server	Odoo	Odoo	You, or your partner
Custom modules	**No.** Standard apps only; changes are made with Studio.	Yes	Yes
Edition	Enterprise	Enterprise	Community or Enterprise
Versions	Major and intermediate; upgraded regularly.	Major versions only	Major versions only
You control	Configuration	Configuration and code	Everything, including the database server

> **Odoo in Iran**
> Most Iranian implementations are on-premise, on the Community edition or with locally arranged Enterprise licensing. Odoo ships no official accounting localization for Iran, the Persian translation and right-to-left layout exist but the Jalali calendar needs an add-on, and Odoo's cloud services and payment channels may not be reachable. Confirm the licensing and hosting position for each project before promising anything to a customer.

4 Why open source?

Giving away the core of a product looks like a strange way to run a company. For Odoo it is the strategy, not a gift.

What "open source" means here

The Community edition is published under the LGPL licence. Anyone may download the code, read it, run it for any purpose, change it and pass it on. The licence also lets you build your own modules on top and keep them private or sell them. The Enterprise modules are different: customers can read their source code, but may not redistribute it, and need a paid subscription to use it.

Why Odoo chose this model

Reason	How it works
Distribution	A free product spreads by itself. A student, a small shop or a developer in any country can install it tonight, and some of them become paying customers or partners later.
An ecosystem	Because the code is open, thousands of developers and partner companies can build on it. Their work fills the Odoo Apps store with tens of thousands of modules, far more than one company could write.
Trust	A company that puts its accounting into a system wants to know it will not be trapped. With open code and a standard database, the data and the logic can always be inspected and taken elsewhere.
Quality	Many eyes read the code. Bugs and security problems are reported, and often fixed, by people outside Odoo.
Revenue elsewhere	Odoo earns from Enterprise subscriptions, from hosting and from services. This is called an *open core* model: the core is free, the extras are sold.

The licences you will meet

Licence	Used for	In practice
LGPL-3	Odoo Community; most community modules	Free to use, modify and redistribute. Modules that depend on it may have any licence.
OEEL-1	Odoo Enterprise modules	Readable by customers, usable only with a subscription, not redistributable.
OPL-1	Paid modules on the Apps store	Bought per database. You may modify them for yourself, not pass them on.
AGPL-3	Many modules of the Odoo Community Association (OCA)	Free. Changes you distribute or offer as a service must be published under the same licence.

What it gives you

- You can read the code to see exactly what a button does. It is the most reliable documentation there is.
- Your data sits in a standard database that you can query and back up yourself.
- Any competent partner or developer can take over a project.
- Almost anything can be changed, if you accept the cost.

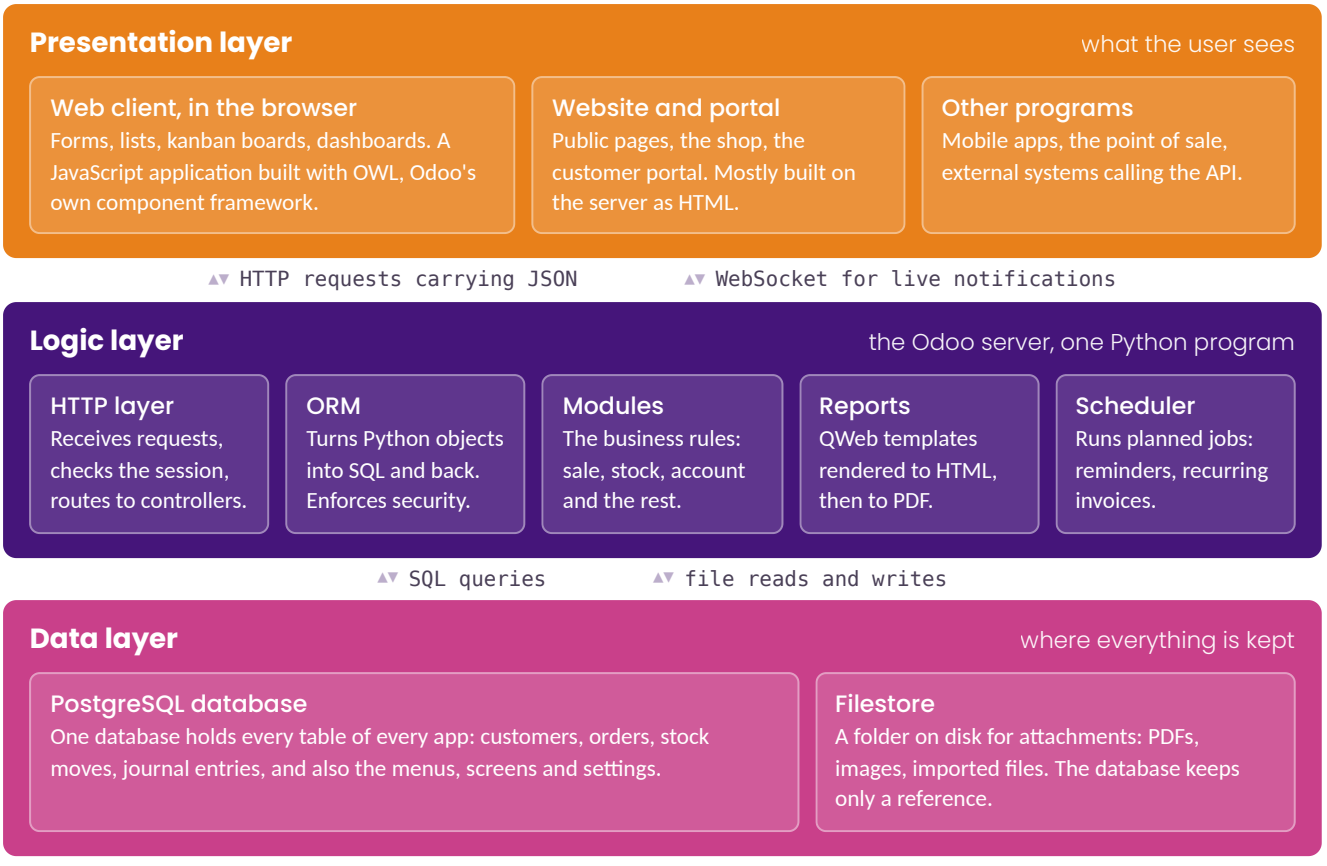
What it does not give you

- A free project. Licence fees are often the smallest part of the cost of an ERP.
- Guaranteed quality in third-party modules. Read the code and test before you install.
- Permission to copy Enterprise code into a Community system.
- Someone to call at night, unless you pay a partner or Odoo for it.

Part B How Odoo is built

5 Architecture: three layers Technical

Odoo is a web application. Nothing is installed on the user's computer: a browser talks to a server, and the server talks to a database.



Layer	Technology	A developer writes	A consultant meets it as
Presentation	HTML, CSS, JavaScript, OWL, XML view definitions	Views in XML; widgets and components in JavaScript	Every screen
Logic	Python 3.10 or later	Models, fields, methods, controllers	What happens when a button is pressed
Data	PostgreSQL 13 or later	Almost nothing by hand: the ORM creates the tables	The data you import, export and report on

What happens when you click Confirm

A salesperson presses Confirm on quotation S00042. Following that one click through the layers shows how the pieces fit.

1. **Browser.** The web client sends a small JSON message to the server: "on model `sale.order`, record 42, call the method `action_confirm`".
2. **Session and transaction.** The server recognises the user from the session cookie and opens a database transaction. From here on, everything succeeds together or nothing is saved.
3. **Security.** The ORM checks that this user may modify sales orders at all, and this one in particular.
4. **Business logic.** `action_confirm` runs. It was written in the Sales module and extended by others: the stock module's extension creates the delivery order, another sends the confirmation e-mail.
5. **SQL.** The ORM translates the changes: an `UPDATE` on table `sale_order`, `INSERT`s into `stock_picking` and `stock_move`.
6. **Commit.** If no error was raised, the transaction is committed. If any step failed, all of it is rolled back and the user sees the error message.
7. **Back to the browser.** The server answers; the web client reloads the record and shows the order as confirmed, with a Delivery button that was not there a second ago.

> **The idea to keep**
> The browser never touches the database, and it contains no business rules. Every rule lives in Python on the server, which is why the same rules apply whether an order arrives from the screen, from an import file or from another system through the API.

6 Why Python, why PostgreSQL Technical

Both were chosen in 2005 and never replaced. The reasons are still good ones, and each choice has a price that is worth knowing.

Why Python

- **It reads like the rule it implements.** A functional consultant can open the code of a button and follow it. In an ERP, where the code *is* the business rule, that matters.
- **It is dynamic.** Python can assemble a class while the program runs. Odoo uses this to build each model from the pieces that all installed modules contribute. Section 9 depends on it.
- **It is quick to write.** Business software is thousands of small rules that change often. Short code and no compile step keep changes cheap.
- **It has a library for everything:** PDF, spreadsheets, e-mail, web services, cryptography, and today data science and AI.
- **Many people know it,** and it is one of the easiest languages to learn.

Why PostgreSQL

- **It is open source with a liberal licence.** An open ERP on a paid database would make little sense.
- **It is transactional.** Confirming an order changes several tables; PostgreSQL guarantees all of them or none.
- **It handles many users at once.** Readers do not block writers, so reports can run while orders are being entered.
- **It is strict.** Types, required columns and links between tables are enforced by the database itself.
- **It is rich.** Odoo relies on its JSON columns for translations, fast text search and advanced indexes.
- **One database only.** By supporting nothing else, Odoo can use all of PostgreSQL instead of the little that all databases share.

The price of each choice

Limitation	What it means	How Odoo lives with it
Python is slower than compiled languages for raw computation.	A loop over a million records in Python is slow.	Heavy work is pushed to the database, and good code works on sets of records, not one at a time.
One Python process uses one processor core at a time.	A single process cannot serve many users well.	Production servers run several worker processes side by side (section 18).
PostgreSQL is the only database.	A company standardised on Oracle, SQL Server or MySQL cannot use it for Odoo.	None. It is a fixed requirement; other systems are reached through integration.
The ORM owns the database structure.	Tables changed by hand are overwritten or break the next update.	Structure is changed in modules only. Direct SQL is for reading.

7 Modules: the unit of everything

If you remember one technical idea from this guide, make it this one. Odoo is a small kernel and a pile of modules. Sales is a module. Your customer's special discount rule is a module.

A **module** is a folder. It contains Python files that define data structures and rules, XML files that define screens and menus, and a manifest that says what the module is called and which other modules it needs. Installing a module tells Odoo to read that folder and to create or change whatever it describes. The words *module* and *addon* mean the same thing.

Anatomy of a module Technical

```
customer_visits/
├── __manifest__.py
├── __init__.py
├── models/
│   ├── __init__.py
│   └── customer_visit.py
├── views/
│   └── customer_visit_views.xml
├── security/
│   └── ir.model.access.csv
├── data/
├── report/
├── controllers/
├── static/
└── i18n/
```

Part	Holds
`__manifest__.py`	The identity card: name, version, dependencies, list of data files.
`models/`	Python classes: the data structures and business rules.
`views/`	XML: screens, menus and actions.
`security/`	Who may read, create, change and delete.
`data/`	Records to load at installation: sequences, e-mail templates, default values.
`report/`	Printable documents.
`controllers/`	Web addresses the module answers to.
`static/`	JavaScript, styles, images, the module's icon.
`i18n/`	Translations, one file per language.

Only the manifest and the top `__init__.py` are required. Many useful modules contain a single XML file.

The manifest

`__manifest__.py`

```python
{
    'name': 'Customer Visits',
    'version': '19.0.1.0.0',
    'category': 'Sales',
    'summary': 'Record visits made to customers',
    'depends': ['base'],                    # modules that must be installed first
    'data': [                               # files loaded at installation, in this order
        'security/ir.model.access.csv',
        'views/customer_visit_views.xml',
    ],
    'application': True,                    # show it in the Apps list as an app
    'license': 'LGPL-3',
}
```

Apps, modules and dependencies

Term	Meaning
App	A module marked as an application: it has its own icon on the home screen. Sales, Inventory, Accounting. To the server there is no other difference.
Module	Any addon. Most are not apps: they add a feature to an app, or connect two apps.
Dependency	Sales needs products and contacts, so it *depends* on those modules. Installing Sales installs them first. Uninstalling a module uninstalls everything that depends on it.
Bridge module	A module that exists only to join two others and is installed automatically when both are present. `sale_stock` appears by itself once Sales and Inventory are installed, and is what makes a confirmed order create a delivery.
Localization	A module named `l10n_` plus a country code that brings a country's chart of accounts and taxes.

Where modules come from

Source	What	Caution
Odoo Community	The kernel (`base` , `web`) and the core apps.	Never edit these files.
Odoo Enterprise	The extra apps, as a second folder of modules.	Needs a subscription. Never edit.
OCA	The Odoo Community Association: free, reviewed modules maintained by volunteers and partners.	Check that your version is covered.
Odoo Apps store	Free and paid modules from thousands of authors.	Quality varies widely. Read the code, test on a copy.
Your own	Customisations written for one customer.	Every line is yours to maintain at each upgrade.

The server finds modules through its **addons path**: a list of folders given in the configuration. A typical server lists the Odoo folder, the Enterprise folder and one folder of custom modules.

The life of a module

Operation	What Odoo does	How
Install	Creates the tables and columns, loads the data files, registers menus and screens.	Apps › Activate, or `-i module`
Update	Re-reads the module: adds new columns, reloads views and data. Needed after the module's files change.	Apps › Upgrade, or `-u module`
Uninstall	Removes the module's menus and screens, *and drops its tables and columns with all their data*.	Apps › Uninstall

> **Uninstalling deletes data**
> Uninstalling Inventory does not hide the stock moves: it deletes them, together with every module that depends on Inventory. Always try an uninstall on a copy of the database first, and read the list Odoo shows before confirming.

8 Inside a module: a complete small example

Technical

Four short files are enough for a working app with a menu, a list, a form, a database table and access control. Reading them once makes every other module readable.

Sahand's sales team wants to record visits to customers: who was visited, when, by whom, and what was said. The manifest is in section 7. Here are the other three files.

1. The model: what a visit is

models/customer_visit.py

```
from odoo import fields, models


class CustomerVisit(models.Model):
    _name = 'customer.visit'                  # technical name of the model
    _description = 'Customer Visit'
    _order = 'date desc'                      # newest first

    name = fields.Char(string='Subject', required=True)
    partner_id = fields.Many2one('res.partner', string='Customer', required=True)
    date = fields.Date(default=fields.Date.context_today)
    user_id = fields.Many2one('res.users', string='Visited by',
                              default=lambda self: self.env.user)
    duration = fields.Float(string='Duration (hours)')
    state = fields.Selection([('planned', 'Planned'), ('done', 'Done')],
                             default='planned')
    notes = fields.Text()
```

2. The views: how a visit looks, and where to find it

views/customer_visit_views.xml

```
<odoo>
    <record id="customer_visit_view_list" model="ir.ui.view">
        <field name="name">customer.visit.list</field>
        <field name="model">customer.visit</field>
        <field name="arch" type="xml">
            <list>
                <field name="date"/>
                <field name="name"/>
                <field name="partner_id"/>
                <field name="user_id"/>
                <field name="state"/>
            </list>
        </field>
    </record>

    <record id="customer_visit_action" model="ir.actions.act_window">
        <field name="name">Customer Visits</field>
        <field name="res_model">customer.visit</field>
        <field name="view_mode">list,form</field>
    </record>

    <menuitem id="customer_visit_menu" name="Customer Visits" action="customer_visit_action"/>
</odoo>
```

No form view is defined, so Odoo generates a plain one. A real module would design it.

3. The access rights: who may do what

security/ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_customer_visit_user,customer.visit.user,model_customer_visit,base.group_user,1,1,1,1
```

One line: every internal user may read, change, create and delete visits. Without this file no user could open the menu.

What Odoo made out of these files

You wrote	Odoo created	Where to see it
A class with `_name = 'customer.visit'`	A table `customer_visit` in PostgreSQL.	Settings › Technical › Models
Seven fields	Seven columns, plus `id`, `create_date`, `create_uid`, `write_date`, `write_uid` that every model gets.	Settings › Technical › Fields
`Many2one('res.partner')`	An integer column `partner_id` with a foreign key to the contacts table, shown as a drop-down.	The form
A `<list>` view	A row in table `ir_ui_view` holding that XML.	Settings › Technical › Views
An action and a menu item	Rows in `ir_act_window` and `ir_ui_menu`.	The home screen
One CSV line	A row in `ir_model_access`.	Settings › Technical › Access Rights

> **Almost everything is a record**
> Screens, menus, actions and access rights are not compiled into the program. They are rows in the database, loaded from the module's files. That is why they can be inspected under Settings › Technical, why Studio can change them without code, and why two databases on the same server can look completely different.

The building blocks, in one table

Block	What it is	Written in	Stored as
Model	A kind of business object: a contact, an order, a visit.	Python class	`ir.model`, and a table
Field	One piece of information on a model.	Python attribute	`ir.model.fields`, and a column
View	One way of showing a model: form, list, kanban, search…	XML	`ir.ui.view`
Action	What happens on a click: open these records in these views, print, run code.	XML	`ir.actions.*`
Menu	An entry in the navigation that triggers an action.	XML	`ir.ui.menu`
Group	A set of users with the same role.	XML	`res.groups`
Access right	What a group may do with a whole model.	CSV	`ir.model.access`
Record rule	Which records of a model a group may see or change.	XML	`ir.rule`
Report	A printable document.	XML (QWeb)	`ir.actions.report`
Controller	Code that answers a web address.	Python	In memory only
Scheduled action	Code run at fixed intervals.	XML and Python	`ir.cron`

Two prefixes are worth learning. `res.` stands for resource: the shared business records such as `res.partner`, `res.users`, `res.company`. `ir.` stands for information repository: Odoo's description of itself.

---

9 Inheritance: extending without editing Technical

This is the mechanism that makes Odoo what it is. A module can add to anything another module defined, without touching the other module's files.

Suppose Sahand wants to see on each customer how many visits were made. The contact model `res.partner` and its form belong to Odoo's `base` module. We do not open those files. We write, in our own module, what should be *added*.

Adding fields to an existing model

models/res_partner.py

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


class ResPartner(models.Model):
    _inherit = 'res.partner'                    # extend the existing model, do not create one

    visit_ids = fields.One2many('customer.visit', 'partner_id', string='Visits')
    visit_count = fields.Integer(compute='_compute_visit_count')

    @api.depends('visit_ids')
    def _compute_visit_count(self):
        for partner in self:
            partner.visit_count = len(partner.visit_ids)
```

Adding the field to an existing screen

views/res_partner_views.xml

```xml
<record id="res_partner_view_form_visits" model="ir.ui.view">
    <field name="name">res.partner.form.visits</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>      <!-- the view we extend -->
    <field name="arch" type="xml">
        <xpath expr="//field[@name='vat']" position="after">     <!-- find the Tax ID field -->
            <field name="visit_count"/>
        </xpath>
    </field>
</record>
```

The Python says "to the model called `res.partner`, add these two fields". The XML says "in the contact form, find the Tax ID field and put the visit count after it". When our module is installed, every contact gains the field; when it is uninstalled, the field disappears and Odoo is as it was.

Changing what a button does

models/sale_order.py

```
class SaleOrder(models.Model):
    _inherit = 'sale.order'

    def action_confirm(self):
        result = super().action_confirm()     # first, everything Odoo and other modules do
        for order in self:
            order.message_post(body="Order confirmed. The warehouse has been informed.")
        return result
```

The call to `super()` is the important line. It runs the original method and every other module's extension of it, and then ours adds one step. Sales, Inventory and Accounting cooperate on the same button in exactly this way.

What you want	Technique	Key words
New fields on an existing model	A class that inherits the model	`_inherit = 'model.name'`
A different behaviour for a method	Redefine the method and call the original	`super()`
A changed screen	An inheriting view that locates an element and adds, replaces or hides	`inherit_id` , `xpath` , `position` : before, after, inside, replace, attributes
A brand-new object	A new model	`_name = 'new.model'`

> **The rule that follows**
> Never edit a file that belongs to Odoo or to someone else's module. Put every change in your own module, as an extension. Your changes then survive updates of the original, can be switched off by uninstalling, and can be read by the next person as a list of exactly what this customer asked for.

10 The ORM in ten minutes Technical

ORM means object-relational mapping: the part of Odoo that lets code work with Python objects while the data lives in SQL tables. Nearly all Odoo code is ORM calls.

In Python	In PostgreSQL	Example
A model class	A table	`sale.order` ↔ `sale_order`
A field	A column	`partner_id` ↔ column `partner_id`
A record	A row	Order S00042 ↔ the row with `id = 42`
A recordset	The result of a query	All confirmed orders of this month

Field types

Type	Holds	In the database
`Char`, `Text`, `Html`	Short text, long text, formatted text	A text column
`Integer`, `Float`, `Monetary`	Numbers; Monetary follows a currency	A numeric column
`Boolean`	Yes or no	A boolean column
`Date`, `Datetime`	A day; a moment	Datetime is stored in UTC and shown in each user's time zone
`Selection`	One value from a fixed list, like a status	A text column holding the key
`Binary`, `Image`	A file	Usually the filestore, as an attachment
`Many2one`	A link to *one* record of another model: the customer of an order	An integer column with a foreign key
`One2many`	The reverse: all the lines of an order	No column. It reads the Many2one on the other side
`Many2many`	Links in both directions: the tags of a contact	A separate link table

A field can also be **computed** (its value is calculated by a method, like `visit_count` above) or **related** (it shows a field of a linked record, like the customer's city on an order).

Reading and writing records

```
Partner = self.env['res.partner']                       # the model

# search: a domain is a list of (field, operator, value) conditions
companies = Partner.search([('is_company', '=', True), ('city', '=', 'Tehran')], limit=10)
for company in companies:                               # a recordset behaves like a list
    print(company.name, company.email)

alborz = Partner.browse(7)                              # one record, by id
alborz.write({'phone': '021 5555 0101'})                # update
print(alborz.country_id.name)                           # follow a link with a dot

visit = self.env['customer.visit'].create({             # insert
    'name': 'Site survey',
    'partner_id': alborz.id,
})
visit.state = 'done'                                    # also an update
visit.unlink()                                          # delete
```

Things every method can reach

Expression	Gives you
`self`	The records the method is working on. Always a set, even when it holds one.
`self.env['model.name']`	Any other model.
`self.env.user`, `self.env.company`	The current user; the company they are working in.
`self.env.context`	Extra information travelling with the request: language, time zone, defaults.
`self.env.ref('module.xml_id')`	A record known by its external identifier (section 15).
`records.sudo()`	The same records with security checks switched off. Powerful, and a common source of data leaks.

Rules attached to fields

Decorator	When the method runs
`@api.depends('field')`	To recompute a computed field when one of the named fields changes.
`@api.onchange('field')`	In the form, while the user is typing, before saving. For suggestions and warnings.
`@api.constrains('field')`	On save. Raise an error here to refuse invalid data.

> **Three habits of the ORM worth knowing early.** One request is one transaction: an error anywhere cancels everything the request did. Records with a field named `active` can be *archived*: hidden from every list without being deleted. And changing data with direct SQL skips security, computed fields, the change history and other modules' rules, so it is done only as a last resort.

Part C Using and configuring Odoo

11 The map of apps Functional

Nobody knows every app. What a newcomer needs is the map: which families exist, how the main ones connect, and what a few shared records are called.

Sales
CRM, Sales, Point of Sale, Subscriptions, Rental

Finance
Invoicing, Accounting, Expenses, Documents, Sign

Supply chain
Inventory, Purchase, Manufacturing, Maintenance, Quality, Barcode

Human resources
Employees, Recruitment, Time Off, Attendances, Appraisals, Payroll

Services
Project, Timesheets, Helpdesk, Field Service, Planning

Websites
Website, eCommerce, Blog, eLearning, Events, Live Chat

Marketing
Email Marketing, SMS, Social, Marketing Automation, Surveys

Productivity
Discuss, Calendar, Contacts, To-do, Knowledge, Dashboards

Customisation
Studio: change screens, add fields and build small apps without code

Not every app listed is in the Community edition; section 3 gives the main split.

The two flows every ERP is built around

Quote to cash: selling

CRM
Opportunity
`crm.lead`

→

Sales
Quotation, then order S00042
`sale.order`

→

Inventory
Delivery WH/OUT/00017
`stock.picking`

→

Invoicing
Invoice INV/2026/00041
`account.move`

→

Accounting
Payment, matched with the bank
`account.payment`

Procure to pay: buying

Inventory
A reordering rule sees low stock
`stock.warehouse.orderp`

→

Purchase
Request for quotation, then order P00012
`purchase.order`

→

Inventory
Receipt WH/IN/00009
`stock.picking`

→

Invoicing
Vendor bill BILL/2026/10/00001
`account.move`

→

Accounting
Payment to the vendor
`account.payment`

Manufacturing sits between the two: a bill of materials lists the components of a panel, a manufacturing order consumes them from stock and puts the finished panel into stock, ready to be sold. Each arrow is a document created *from* the previous one, which is why the order, the delivery and the invoice always agree.

The main apps and their technical names

App	Module	Main models	In one line
Contacts	`contacts`, `base`	`res.partner`	Customers, vendors, people and companies: one list for all apps.
CRM	`crm`	`crm.lead`	Leads and opportunities in a pipeline.
Sales	`sale_management`	`sale.order`, `sale.order.line`	Quotations and sales orders.
Inventory	`stock`	`stock.picking`, `stock.move`, `stock.quant`	Receipts, deliveries, transfers, quantities on hand.
Purchase	`purchase`	`purchase.order`	Requests for quotation and purchase orders.
Invoicing, Accounting	`account`	`account.move`, `account.move.line`	Invoices, bills, payments, journal entries.
Manufacturing	`mrp`	`mrp.bom`, `mrp.production`	Bills of materials and manufacturing orders.
Project	`project`	`project.project`, `project.task`	Projects and tasks in stages.
Employees	`hr`	`hr.employee`	The staff directory and departments.
Website, eCommerce	`website`, `website_sale`	`website.page`, `sale.order`	Pages and an online shop whose orders are ordinary sales orders.
Point of Sale	`point_of_sale`	`pos.order`	A till for shops and restaurants.
Discuss	`mail`	`mail.message`, `mail.activity`	Chat, and the message thread under every record.

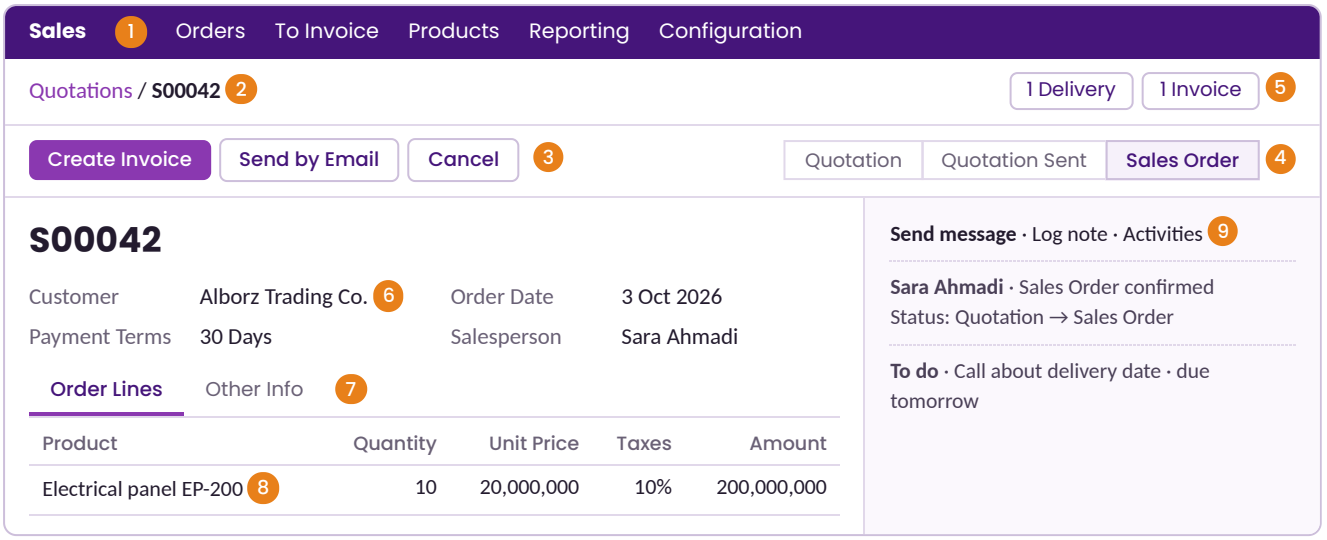
Records that every app shares

Record	Model	Why it matters
Contact	`res.partner`	A customer, a vendor, an employee's private address and a company are all contacts. One record serves Sales, Purchase and Accounting.
Product	`product.template`, `product.product`	The same product is sold, bought, stocked and manufactured. The template is the product; the second model is each variant, such as a size or colour.
Company	`res.company`	The legal entity whose documents these are. Most business records carry a company.
User	`res.users`	Someone who can log in. Every user is also a contact.

> **A habit for consultants.** When a customer describes a need, ask first which standard app already does most of it, then which settings bring it closer, and only then what would have to be developed. The order of those three questions decides the cost of the project.

12 Finding your way around the screen Functional

Every app uses the same few screen patterns. Learn them once on a sales order and you can operate an app you have never seen.



A sketch of a sales order form, not a screenshot.

	Element	What it is for
1	App menus	The app's name and its menus. The square icon at the far left returns to the home screen with all apps.
2	Breadcrumb	Where you are and how you got here. Each part is a link back.
3	Action buttons	What can be done to this record now. They change with the status.
4	Status bar	The stage the record is in. Most documents move left to right.
5	Smart buttons	Shortcuts to related records: the deliveries and invoices of this order.
6	Fields	The data of the record. A field shown as a link points to another record, and clicking it opens that record.
7	Tabs	Pages of further fields, to keep the form short.
8	Lines	A list inside the form: the lines of the order. Technically a One2many field.
9	Chatter	The history of the record: messages, internal notes, planned activities, and an automatic log of who changed what.

The views

The same records can be shown in several ways, chosen with the small icons at the top right of a list.

View	Shows	Use it to
List	Rows and columns	Scan, sort, select many records, edit in bulk.
Form	One record in full	Create and edit.
Kanban	Cards in columns, often by stage	Follow a process and drag records from stage to stage.
Calendar	Records on their dates	Plan meetings, deliveries, time off.
Pivot	A cross table with totals	Analyse: sales by month and salesperson.
Graph	Bars, lines or a pie	See a trend.
Activity	Records against their planned activities	See what is late.
Gantt, Map, Cohort Enterprise	A timeline; addresses on a map; retention over time	Plan resources, routes, subscriptions.

Searching: the most used tool in Odoo

Tool	What it does
Search box	Type a word and choose which field to search in: customer, order number, product.
Filters	Ready-made conditions: My Quotations, To Invoice, This Month. Several can be combined.
Group By	Folds the list into groups with subtotals: by salesperson, by month, by customer. Groups can be nested.
Favorites	Saves the current combination of filters and groups under a name, for yourself or for everyone, and can make it the default.
Command palette	Ctrl+K, then type the name of any menu, or `@` to find a person. The fastest way to move around.

> **Archive, do not delete.** Most records have an Archive action. An archived customer or product disappears from lists and drop-downs but stays on old documents and can be restored. Deleting is refused anyway for most records that were used.

13 First configuration: twelve basics

A new database is empty and generic. These twelve settings turn it into your customer's system, and they are the same on every project whatever apps come later.

#	Setting	Where Functional	What to decide	Behind it Technical
1	**Create the database**	The database manager, or the sign-up page on Odoo Online	Name, language, country. Demo data: never in a real database.	A new PostgreSQL database with `base` installed.
2	**Install the apps**	Apps	Only what the first phase of the project needs.	`ir.module.module`
3	**Company details**	Settings › Companies › Update Info	Legal name, address, tax ID, currency, logo.	`res.company`
4	**Document layout**	Settings › Companies › Configure Document Layout	How quotations and invoices look: layout, colours, footer.	A QWeb template chosen on the company
5	**Users**	Settings › Users › Manage Users	Who logs in, with which role in each app (section 14).	`res.users`, `res.groups`
6	**Languages**	Settings › Languages › Add Languages	Which languages are available. Each user picks one.	`res.lang`; translated fields
7	**Time zone and formats**	The user's Preferences; the language record	Each user's time zone. Date and number formats per language.	Times are stored in UTC and converted on display.
8	**E-mail**	Settings › Emails	The server Odoo sends through, and the domain for replies.	`ir.mail_server`, `mail.alias`
9	**Currencies and units**	Accounting settings; Settings › Technical › Decimal Accuracy	Other currencies in use. Units of measure. How many decimals for prices and quantities.	`res.currency`, `uom.uom`, `decimal.precision`
10	**Numbering**	Settings › Technical › Sequences	Prefixes and padding of order numbers: S00042, P00012.	`ir.sequence`. Invoices are numbered by their journal instead.
11	**Companies and branches**	Settings › Companies › Manage Companies	One company, several, or branches of one.	`company_id` on records, and record rules
12	**Security policy**	Settings › Permissions	Password length, two-factor authentication, who may invite users.	`ir.config_parameter`

Technical menus appear only in developer mode (section 17). Menu names shift slightly between versions; the search box at the top of Settings finds any option by name.

Three choices that are hard to undo
The **country** chosen at creation decides which accounting package is proposed. The **company currency** cannot be changed once entries exist. And **demo data**, once loaded, cannot be removed: the database has to be created again.

Never configure in production first
Keep at least two databases: a test copy where settings and imports are tried, and the real one where they are repeated once they work. Odoo.sh and Odoo Online offer copies with a click; on your own server it is a duplicate from the database manager.

How the Settings screen works Technical

The Settings app is one long form to which every installed app adds a section. Its options look alike, but they do four different things when you press Save:

Kind of option	What Save does	Example
Installs a module	The feature is a separate module; ticking the box installs it.	"Margins", "Landed Costs"
Grants a group	Adds all users to a group that makes extra fields and menus visible.	"Variants", "Discounts", "Units of Measure"
Sets a value on the company	Writes a field of `res.company`.	Default taxes, fiscal year end
Sets a system parameter	Writes a key and value in `ir.config_parameter`.	Minimum password length

This explains two things newcomers notice: saving the settings can take a while, because modules are being installed, and un-ticking an option does not always remove what ticking it added.

14 Users and security

Who may see and do what is decided in four layers. The functional side is one tab on the user form; the technical side is three kinds of records.

Three types of user Functional

Type	Who	Access
Internal user	Employees who work in Odoo.	The back office, according to their roles.
Portal user	Customers and vendors with a login.	Only the portal: their own quotations, orders, invoices, tickets.
Public user	Anonymous visitors of the website.	Public pages only.

On the user form, the Access Rights tab lists the installed apps, and for each one a role is chosen. In Sales the choices are typically "Own documents only", "All documents" and "Administrator".

The four layers Technical

	Layer	Model	Answers	Example
1	**Groups**	`res.groups`	What roles does this user have?	Sales / User: Own Documents Only
2	**Access rights**	`ir.model.access`	May this group read, create, write or delete this *model*?	Sales users may create sales orders; they may only read products.
3	**Record rules**	`ir.rule`	Which *records* of the model?	Only orders where the salesperson is the user.
4	**Field and menu visibility**	`groups` on fields, views and menus	Which fields, buttons and menus are shown?	The margin field is visible to managers only.

The three sales roles, layer by layer

Role chosen on the user	Access right on sales orders	Record rule	Sees
User: Own Documents Only	Read, write, create	Salesperson is me, or empty	Their own orders
User: All Documents	Read, write, create	No restriction	Everyone's orders
Administrator	Read, write, create, delete	No restriction	Everything, plus the Configuration menu

Each higher role *includes* the one below it, so a manager automatically has everything a user has.

How the layers combine

- **Access rights add up.** A user in several groups gets the union of what each group allows. If no group allows an operation on a model, it is refused.
- **Record rules for groups add up too.** Two rules from two of the user's groups mean: a record passes if either rule accepts it.
- **Global rules restrict.** A rule that belongs to no group applies to everybody and must always be satisfied. The multi-company rule, "only records of my companies", is of this kind.
- **The superuser bypasses the rules.** Code running with `sudo()`, and the special user behind automatic jobs, are not checked. The administrator account that people log in with is an ordinary user with every role.

> **Three practical rules**
> Test every role by logging in as a real user of that role, never as the administrator. Give the Settings administration role to two or three people, not to everyone who asks. And when a user "cannot see" a record or a menu, check the layers in order, from 1 to 4.

15 Data in, data out

Every implementation starts by loading customers, products and opening figures, and every integration ends up reading or writing records from outside. The tools are the same for all models.

Import and export Functional

Task	How
Export	In a list, tick the records, then Actions › Export. Choose the fields and the format, Excel or CSV. Tick "I want to update data" to get a file that can be imported back.
Import	In a list, open the gear menu and choose Import records. Upload an Excel or CSV file, map each column to a field, press Test, read the messages, then Import.
Update in bulk	Export with "I want to update data", change the values in the file, keep the first column untouched, import the file again.

The simplest way to get the columns right is to enter one record by hand, export it, and use that file as the template.

External identifiers Technical

Every record has a numeric `id` that differs from database to database. An **external identifier** is a stable text name for a record, in the form `module.name`. It is what the first column of an import-compatible export contains, and what lets an import *update* a record instead of creating a duplicate.

External identifier	Refers to
`base.main_company`	The first company of the database.
`base.group_user`	The group of all internal users.
`base.view_partner_form`	The contact form, as used in section 9.
`customer_visits.customer_visit_action`	The action defined in our example module.
`__import__.customer_alborz`	A record you imported with the id "customer_alborz" in the file.

Give your own identifiers when importing master data, for example the code of the customer in the old system. Later imports of orders or balances can then point at the right customer without relying on names.

The external API Technical

Other programs talk to Odoo over HTTP and call the same model methods the web client uses: `search`, `read`, `create`, `write`. Security applies exactly as for the user whose key is used. Odoo 19 introduces a new interface for this, the JSON-2 API:

```
import requests

BASE_URL = "https://sahand.example.com/json/2"
headers = {
    "Authorization": "bearer " + API_KEY,       # a key created in the user's Preferences
    "X-Odoo-Database": "sahand",
}
response = requests.post(
    BASE_URL + "/res.partner/search_read",      # /json/2/<model>/<method>
    headers=headers,
    json={
        "domain": [("is_company", "=", True), ("city", "=", "Tehran")],
        "fields": ["name", "email", "phone"],
    },
)
for partner in response.json():
    print(partner["name"])
```

Older code. Until version 18 the external interfaces were XML-RPC and JSON-RPC, and most existing integrations and tutorials use them. They still work in Odoo 19 and are scheduled for removal in Odoo 22, so new integrations should use JSON-2. On Odoo Online, access through the external API depends on the pricing plan.

16 Automation without code Functional

Between pure configuration and writing a module there is a middle ground: rules and jobs that an administrator defines on the screen.

Tool	What it does	Example	Model
E-mail template	A reusable message with fields filled in from the record.	The quotation e-mail	`mail.template`
Activity type and plan	Standard to-dos, alone or as a sequence, assigned with a deadline.	"Call back in 3 days"	`mail.activity.type`
Server action	One operation on records: update a field, create a record, send an e-mail, run a few lines of Python.	"Mark as urgent", added to the Actions menu	`ir.actions.server`
Automation rule	A trigger plus server actions: when a record is created, changed, reaches a stage, or a date approaches.	When an order above a limit is confirmed, create an activity for the finance manager	`base.automation`
Scheduled action	Runs code at an interval, with no user involved.	Every night, send payment reminders	`ir.cron`
Studio Enterprise	A visual editor: add fields, rearrange forms, build reports, approval steps and small apps.	A new field on the customer form	Stores ordinary fields and views

All of these live under Settings › Technical, in developer mode. Automation rules need the module of that name to be installed.

> **Convenient, and easy to lose track of**
> A rule created on the screen exists only in that database. It is not in any module, is not copied to the test system unless the database is, and nobody will remember it in a year. Keep a list of every automation, with who asked for it and why, and prefer a small module for anything the business depends on.

17 Developer mode and the Technical menu

Developer mode is not only for developers. It is the consultant's X-ray: it shows the technical names behind what is on the screen.

Switching it on

From	How
Settings	Scroll to the bottom, Developer Tools, "Activate the developer mode".
The address bar	Add `?debug=1` after the web address. `?debug=0` switches it off. `?debug=assets` also loads the JavaScript unpacked, for debugging the interface.
The command palette	Ctrl+K, then type "debug".

What changes on the screen

You get	Use it to
A tooltip on every field label	Hover over a label to read the field's technical name, its type and its model. This is how you find out what to put in an import file or a filter.
A bug icon in the top bar	Open the definition of the current screen: its view, its action, its fields, its filters, the access rights and record rules of its model.
The Technical menu in Settings	Browse and edit Odoo's own records: models, views, actions, sequences, scheduled actions, parameters, e-mails.
"Update Apps List" in Apps	Make Odoo notice a module folder that was just added to the server.
Extra fields and options	Many forms show technical fields only in this mode.

The Technical menu, section by section

Section	Contains
Email, Discuss	Outgoing and incoming mail servers, the queue of e-mails, templates, aliases.
Actions	Window actions, server actions, reports.
User Interface	Menus, views, saved filters.
Database Structure	Models, fields, decimal accuracy, attachments.
Automation	Scheduled actions, automation rules.
Sequences & Identifiers	Number sequences, external identifiers.
Parameters	System parameters: keys and values that tune the system.
Security	Access rights, record rules.

> **Read freely, change carefully.** Everything in the Technical menu can be looked at without risk, and looking is how you learn Odoo. Changing a view or an access right here takes effect at once, for everyone, and may be overwritten the next time the module that owns it is updated. Durable changes belong in a module or in Studio.

Part D Running Odoo, and judging it

18 Installing and running a server Technical

You can use Odoo for years without installing it. Doing it once, on your own laptop, removes most of the mystery, and a test server of your own is the best place to learn.

What a server needs

Component	Version for Odoo 19	Role
Python	3.10 or later	Runs the Odoo server.
PostgreSQL	13 or later	The database.
Odoo source code	Branch 19.0	The Community repository, and the Enterprise repository beside it if licensed.
wkhtmltopdf	0.12.6	Turns reports into PDF. Without it, printing fails.
Node.js with rtlcss		Needed for right-to-left languages such as Persian and Arabic.
Operating system	Linux is usual	Windows and macOS work for development.

There are three ways to get it running: a packaged installer, which is the quickest for a first look; the source code from GitHub, which is what developers use; and a Docker image, which packs Odoo and its dependencies in a container.

A first run from source

```
git clone https://github.com/odoo/odoo.git --branch 19.0 --depth 1
pip install -r odoo/requirements.txt

# create database "sahand_test", install Sales and Inventory, and start the server
python3 odoo/odoo-bin --addons-path=odoo/addons,custom -d sahand_test -i sale_management,stock

# then open http://localhost:8069  and log in with  admin / admin
```

The command line you will actually use

Option or command	Meaning
`-d name`	The database to work on.
`-i module1,module2`	Install these modules, then run.
`-u module1` or `-u all`	Update modules: reload their Python, views and data. Needed after changing a module's files.

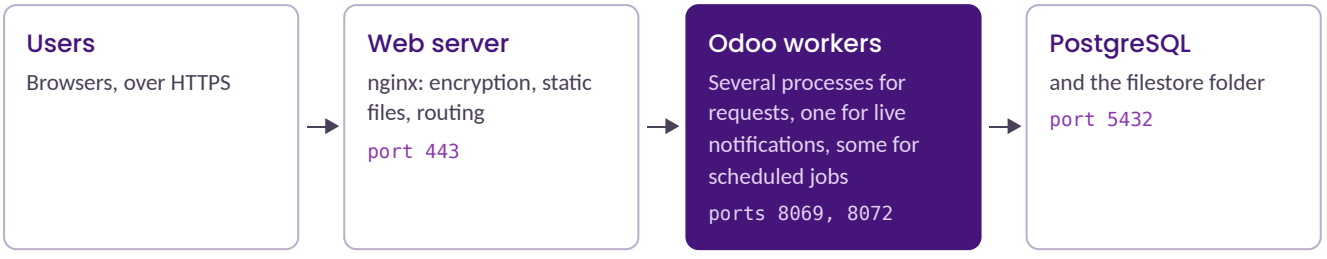
Option or command	Meaning
`--addons-path=dir1,dir2`	The folders in which to look for modules.
`-c file`	Read options from a configuration file.
`--dev=all`	Development mode: reload Python when files change, read views straight from the files.
`--with-demo`	Load demonstration data into new databases. In Odoo 19 the default is without.
`-p 8070`	Listen on another port than 8069.
`--stop-after-init`	Do the install or update and exit. Used in deployment scripts.
`--test-enable`	Run the automated tests of the modules being installed or updated.
`odoo-bin shell -d name`	An interactive Python prompt with `env` ready: try ORM calls against real data.
`odoo-bin scaffold name dir`	Create the skeleton of a new module.
`odoo-bin db` , `odoo-bin module`	Dump, load, duplicate databases; install or update modules, without starting the server.

The configuration file

odoo.conf

```
[options]
addons_path = /opt/odoo/odoo/addons,/opt/odoo/enterprise,/opt/odoo/custom
db_host = localhost
db_user = odoo
admin_passwd = a-long-random-master-password   # protects the database manager
http_port = 8069
workers = 5                # several processes: needed for production
proxy_mode = True          # the server sits behind a web server such as nginx
dbfilter = ^sahand$        # which databases this address may show
list_db = False            # hide the list of databases from the internet
```

A production server



Topic	What to know on day one
Workers	Odoo's rule of thumb: (number of processor cores × 2) + 1 workers, and about six simultaneous users per worker.
Environments	Development on laptops, a test server that copies production, and production. Changes move in that order, never the other way.
Neutralised copies	A copy of production still knows the real mail server and the real customers. Neutralising it switches off outgoing e-mail, scheduled jobs and payment connections, so a test cannot write to a real customer.
Backups	A backup is the database dump *and* the filestore. One without the other loses either the data or every attachment. Daily, copied to another machine, and restored now and then to prove it works.
Database manager	The page at `/web/database/manager` creates, duplicates, backs up and deletes databases. It is protected only by the master password; on a server reachable from the internet it should be disabled.

19 Advantages and limitations

A consultant who can only praise Odoo is not trusted for long. This is the balanced picture, in the form you can use with a customer.

Advantages

- **One system.** Sales, stock, accounting and people share data without interfaces.
- **Modular.** Start with two apps and add the rest when the company is ready.
- **Breadth.** ERP, website, shop, point of sale and marketing from one vendor.
- **Flexible.** Inheritance lets almost anything be adapted without forking the product.
- **Open.** Readable code, a standard database, no lock-in on data.
- **Cost of entry.** Community has no licence fee; Enterprise is priced below the large ERP vendors.
- **Usability.** A modern, consistent web interface that people learn quickly.
- **Ecosystem.** Many partners and developers, and tens of thousands of ready modules.
- **Pace.** A new version every year, with visible improvements.

Limitations

- **Upgrades cost.** Every major version means testing, and rewriting part of the custom code.
- **Customisation is tempting.** Because it is easy, projects over-customise and pay for it for years.
- **Breadth over depth.** In specialised areas a dedicated product may go further than the Odoo app.
- **Community is not complete.** Full accounting reports, Studio and several apps are Enterprise only.
- **Localization varies.** Strong in some countries, absent in others, Iran among them.
- **Third-party quality.** Store modules range from excellent to harmful.
- **Performance needs care.** Large volumes require good hosting, tuning and well-written code.
- **It needs a project.** The software is quick to install and slow to implement well.
- **Pace, again.** Screens and menus move every year; training material ages fast.

When it fits, and when to think twice

A good fit	Think twice
Small and medium companies that have outgrown spreadsheets and separate tools.	A company that wants the software to copy its current paper forms exactly.
Trading, distribution, light manufacturing, services, projects, retail.	Heavily regulated or very specialised processes with no matching app.
A management willing to adopt standard processes where they are good enough.	No internal owner for the project, and no budget beyond the licence.
A partner or team able to maintain what is built.	A need for local legal features that nobody has yet built and tested.

20 Your first month

Odoo is learned by using it. Reading helps you understand what you saw; it does not replace seeing it.

Week	Functional track Functional	Technical track Technical
1	Get a test database with demo data. Create a customer and a product; run one sale from quotation to paid invoice, and one purchase from order to paid bill.	Install Odoo from source on your own machine. Create a database from the command line. Switch on developer mode and read the tooltips.
2	Repeat the same flows in an empty database, doing the twelve basics of section 13 yourself. Create three users with different roles and log in as each.	Type in the Customer Visits module of section 8 and install it. Break it on purpose and read the error messages.
3	Import fifty customers and fifty products from a spreadsheet. Build two saved filters and a pivot report. Read the chatter of the documents you created.	Extend a standard model and a standard form as in section 9. Use `odoo-bin shell` to search, create and write records.
4	Take one real process of a real company and write down how standard Odoo would do it, step by step, and where it would not fit.	Read the source of one standard module from manifest to views: `sale` is a good first one. Follow one button from the XML to the Python.

Mistakes almost everyone makes once

Mistake	What happens	Instead
Testing as the administrator.	Everything works, until a real user logs in and sees half of it.	Test with a user who has the real role.
Configuring directly in production.	A wrong setting reaches real documents before anyone notices.	Try on a copy, then repeat.
Installing apps "to have a look" in the real database.	Menus, fields and sometimes costs appear; uninstalling deletes data.	Look in a test database.

Mistake	What happens	Instead
Editing a standard file to fix something quickly.	The fix vanishes at the next update, or blocks it.	An inheriting module.
Changing data with SQL.	Totals, stock or accounting no longer agree with the documents.	The interface, an import, or the ORM.
Customising before understanding the standard.	A development that duplicates a setting two menus away.	Learn the standard flow first.
Installing a store module without reading it.	Slow screens, broken upgrades, security holes.	Read the code, test on a copy, prefer known authors.
Forgetting to update the module after changing its files.	"My change does not show."	`-u module`, or Upgrade in Apps.
Following a tutorial for another version.	Menus are not where the tutorial says; code fails.	Check the version at the top of every page you read.
Backing up the database without the filestore.	After a restore every attachment is missing.	Always both.

21 Glossary, cheat sheet and self-check

The words you will hear in every meeting, the places you will go every day, and eight questions to test yourself.

Glossary

Term	Persian	Meaning
ERP	برنامه‌ریزی منابع سازمان	One system for all departments of a company, on shared data.
Implementation	پیاده‌سازی	The project of setting Odoo up for one company: analysis, configuration, data, training, go-live.
App	اپلیکیشن	A module with an icon on the home screen.
Module, addon	ماژول	A folder of code and data that adds or changes a feature.
Community, Enterprise	نسخه رایگان، نسخه تجاری	The open-source edition, and the paid edition built on top of it.
Standard	استاندارد	What Odoo does as delivered, without custom code.
Configuration	پیکربندی	Adapting Odoo through its settings.
Customisation	سفارشی‌سازی	Adapting Odoo through Studio or custom modules.
Localization	بومی‌سازی	Modules for one country's accounting, taxes and legal documents.
Model	مدل	A kind of business object, and its database table.
Field	فیلد	One piece of information on a model.
Record	رکورد	One object: one customer, one order.

Term	Persian	Meaning
View	نما	A way of showing records: form, list, kanban.
Action	اکشن	What a menu or button triggers.
Domain	شرط فیلتر	A filter condition, written as a list of (field, operator, value).
ORM		The layer between Python objects and SQL tables.
Inheritance	وراثت	Extending a model, method or view from another module.
External ID	شناسه خارجی	A stable name for a record, `module.name`.
Chatter		The thread of messages, notes and activities on a record.
Access rights, record rules	سطوح دسترسی	What a group may do with a model; which of its records.
Developer mode	حالت توسعه‌دهنده	The mode that shows technical names and menus.
Studio		The Enterprise tool for changing screens and adding fields without code.
Update, upgrade	به‌روزرسانی، ارتقای نسخه	Reloading a module or applying fixes; moving to the next major version.
Filestore		The folder on the server where attachments are kept.
Worker		One of the server processes that answer requests.
OCA		The Odoo Community Association, which maintains free modules.

Cheat sheet

To do this	Go to
Install an app	Apps
Add a user	Settings › Users › Manage Users
Company data, document layout	Settings › Companies
Add a language	Settings › Languages
Outgoing e-mail	Settings › Emails
Developer mode	Settings, bottom of the page; or `?debug=1`
Models, fields, views	Settings › Technical
Import, export	Any list: gear menu; Actions
Jump anywhere	Ctrl+K

Technical name	Is
`res.partner`	Contacts
`res.users`, `res.company`	Users, companies
`product.template`	Products
`sale.order`	Quotations and sales orders
`purchase.order`	Purchase orders
`stock.picking`	Receipts, deliveries, transfers
`account.move`	Invoices, bills, journal entries
`ir.ui.view`, `ir.model`	Views, models
`-i`, `-u`, `-d`	Install, update, database

Check yourself

	Question	Answer
1	In which of the three layers do the business rules live, and why does that matter?	In the logic layer, in Python on the server. So the same rules apply to the screen, to imports and to the API.
2	What is the difference between an app and a module?	An app is a module flagged as an application, with an icon. Technically they are the same thing.
3	A customer wants a new field on the sales order. Which files of Odoo do you edit?	None. You write a module that inherits the model and the view.
4	Sales and Inventory are both installed. Which module makes a confirmed order create a delivery?	The bridge module `sale_stock`, installed automatically.
5	A user cannot see a colleague's quotations. Which layer of security is at work?	A record rule: the user's role is "Own documents only".
6	Why can a company hosted on Odoo Online not use a module you wrote?	Odoo Online runs standard apps only. Custom code needs Odoo.sh or your own server.
7	You changed a view file and nothing changed on the screen. What did you forget?	To update the module, so that Odoo reloads the file into the database.
8	What must a backup contain?	The database and the filestore.

Where to learn more

Source	Use it for
Official documentation, odoo.com/documentation/19.0	User guides for every app; the developer reference and tutorials.
The source code, github.com/odoo/odoo	The final word on how anything works.
Odoo's eLearning courses and videos	Guided functional training, app by app.
OCA, github.com/OCA	Free modules, and examples of well-written code.
The Odoo forum	Questions and answers. Check the version of each answer.

> **The whole guide in two lines.** Odoo is a browser, a Python server and a PostgreSQL database, with every feature delivered as a module that can be extended by another. Learn the standard flows, configure before you customise, and never edit what you do not own.