Odoo Technical Training Roadmap

Introduction

Odoo is an all-in-one business software that companies use to manage their daily operations such as sales, accounting, inventory, human resources, and customer relations. Instead of using separate programs for each department, Odoo brings everything into one connected system. For example, when a customer places an order in the Sales app, Odoo can automatically create an invoice in the Accounting app and reduce the stock level in the Inventory app. This integration is what makes Odoo powerful for businesses of all sizes.

When we talk about Odoo, there are two main sides to understand: the functional side and the technical side. The functional side is about using Odoo as a business tool without writing any code. A functional user can create customers and products, record invoices, check sales reports, or set user permissions. On the other hand, the technical side is about customizing Odoo or building new applications inside it. This is where coding comes in. A developer can add new fields, build new apps, create reports, or automate workflows using Python and XML.

The difference between these two sides can be shown with a simple example. If a company wants to add a “Notes” field to the Customer form, a functional user can do it directly from Odoo’s settings without touching any code. But if the company wants a more powerful and reusable solution, a developer would create a module, define the field in Python, and place it in the form using XML. Both methods work, but the technical way offers more control and flexibility.

With this foundation in mind, let’s now explore Odoo step by step, starting from the basics and moving toward advanced development.

Chapter 1: What is Odoo Development?

Is Odoo a framework or just an ERP?

- Many people think Odoo is just an ERP (like accounting, sales, inventory).
- But Odoo is also a **framework** that you can build apps on.
- Imagine Django or Flask (Python frameworks). They let you build websites or apps.
- Odoo is like that, **but it already comes with ready-made business apps** (Sales, CRM, HR, etc.).
- So you don’t start from zero. You can either use the ready apps, or create your own.

Odoo as a framework + business apps

Odoo gives you:

- **Database layer (ORM)** → no need to write raw SQL.
- **UI system (XML + QWeb)** → no need to write HTML manually.
- **Security system** → users, groups, rules already built in.
- **APIs** → connect with other apps.

- On top of that, it already has **business apps**.
- So it’s like getting a **Lego set**: you already have many pieces built, but you can also make your own custom pieces.

Example

- Suppose you want to add a **“Gender” field** to Customers.
- Normally in other systems:
  - You’d create a new SQL column in the database.
  - Add a field in HTML form.
  - Link them together with code.
- In Odoo:
  - Just add **3 lines in Python** (for the field).
  - And **1 line in XML** (to show it on the form).
- Done ✔ No need to touch SQL or raw HTML.

Chapter 2: Architecture Overview (Easy Mode)

The Flow

1. **Client (User side)**
   - You open your browser or Odoo mobile app.
   - You click buttons like “Create Customer.”
2. **Odoo Server (Brain)**
   - This is where Python code runs.
   - It talks to the database through ORM.
   - It knows what to show you in the UI (XML views, QWeb reports).
3. **Database (PostgreSQL)**
   - Stores all data (customers, invoices, products).
   - You never write SQL manually. Odoo ORM handles it.

Odoo Layers

- **Models (ORM)** → database tables + business logic.

- **Views (XML)** → how data looks in the UI.
- **Business Logic (Python)** → rules and functions.
- **Reports (QWeb)** → print nice PDFs.
- **API** → connect Odoo with other apps.

Short Example Flow

- You click “Create Customer.”
- Browser → sends request → Odoo server.
- Odoo → creates a new customer record in PostgreSQL.
- Odoo → shows the customer form using XML.
- You see the result in your browser.

Chapter 3: Anatomy of a Module

What is a module?

- A **module** is like a “mini-app” inside Odoo.
- Example: Sales, Inventory, CRM = all modules.
- You can also make your own modules.

Folder Structure

- `__manifest__.py` → module’s **identity card**. Name, version, dependencies.
- `__init__.py` → tells Python to load models or wizards.
- `models/` → your database logic (Python files).
- `views/` → XML files for UI.
- `security/` → who can access what.
- `report/` → QWeb reports (PDF, printouts).
- `wizards/` → temporary models for popups.

Example: Hello World Module

- Create a folder `hello_world`.
- Inside add `__manifest__.py` (basic info).
- Add `models/hello.py` with a simple model.
- Add `views/hello_view.xml` to display it.
- When you install → it shows “Hello World.”

Chapter 4: Models & Fields

What is a model?

- A **model = database table + Python class**.
- Example: `res.partner` is the Customer table.

Types of Fields

- **Char** → short text (name).
- **Text** → long text (description).
- **Integer** → whole numbers (age).
- **Float** → decimal numbers (price).
- **Boolean** → true/false (active).
- **Date/Datetime** → calendar values.
- **Selection** → dropdown choices.

Why Odoo makes it simple

- Normally you must create a table in SQL, then create HTML forms.

- In Odoo: just add a Python field. UI and DB update automatically.

Example

```
class ResPartner(models.Model):
    _inherit = "res.partner"

    x_street = fields.Char(string="Extra Street")
```

- Done → you added `x_street` field in 3 lines.

Chapter 5: Security Basics

Security Chain

- **Users** → people who log in.
- **Groups** → collection of permissions (e.g., Sales User, Manager).
- **Access Rights** → control CRUD (Create, Read, Update, Delete).
- **Record Rules** → control which records you can see.

Example

- "Sales User" can read invoices.
- But only "Sales Manager" can delete invoices.

Security XML

- You create an XML file to tell Odoo:
    - Which group can access which model.
    - Whether they can read, write, create, delete.

Chapter 6: Views (XML vs HTML)

XML vs HTML

- HTML → for normal websites.
- XML → structured format Odoo uses to define views.

Types of Views

- **Form View** → single record screen.
- **Tree View (List)** → table of many records.
- **Kanban** → cards (like Trello).
- **Calendar** → schedule view.
- **Search View** → filters, search bar.

Example

- Add “Notes” field in Customer form → just one line in XML.

Chapter 7: Actions

What is an action?

- An **action** tells Odoo **what to do when you click something**.
- It’s the bridge between backend and frontend.

Types

- **Window Action** → open a list or form.
- **Server Action** → run Python code.
- **Report Action** → print a report.
- **URL Action** → open a website link.

Example

- Clicking “Customers” in menu → runs a Window Action to open customer list.

Chapter 8: Menuitems

Basics

- A menuitem = the button in the Odoo sidebar.

Attributes

- `name` → what user sees.
- `parent_id` → who is the parent menu.
- `action` → what happens when clicked.

Example

- Menu “Customers” → parent = Sales → action = open customer list.

Chapter 9: Relations Between Models

Why relations?

- To connect tables (models).

Types

- **Many2one** → many records point to one record. Example: Invoice → Customer.
- **One2many** → one record has many children. Example: Customer → Orders.
- **Many2many** → many records connect to many. Example: Product ↔ Tags.

Chapter 10: Computed Fields & Onchanges

Computed Fields

- Field that calculates its value from other fields.
- Can be real-time or stored in DB.

Onchange

- Updates the UI instantly when a field changes (before saving).

Example

- Quantity × Price = Total Amount.
- If you change Quantity → Total updates automatically.

Chapter 11: Constraints

What are constraints?

- Rules to keep data correct.

Types

- **Python constraints (@api.constrains)** → custom Python checks.
- **SQL constraints** → unique, not null, etc.

Example

- Prevent two customers from having the same VAT number.

Chapter 12: Inheritance & Interacting with Other Modules

Model Inheritance

- `_inherit` → extend an existing model (add fields, methods).
- `_inherits` → delegate model (link another model).

View Inheritance

- Use **xpath** to insert or replace parts of existing XML views.

Method Inheritance

- Use **super()** to extend or override Python methods.

Case Scenarios

- Add new field to `account.move`.
- Override `action_post` to add custom logic.

Interacting with Modules

- Use `depends` in manifest.

- Use `self.env['model']` to call other models.

Chapter 13: Wizards

What is a wizard?

- A temporary model that only exists while you use it.
- Used for dialogs/popups.

Difference

- Normal model = stored in DB permanently.
- Wizard = only temporary, data is deleted after use.

Example

- Wizard → choose a date range → generate a report.

Chapter 14: Reports (QWeb)

What is QWeb?

- Odoo’s reporting engine.
- Similar to HTML, but has Odoo’s own tags.

Example

- Create an invoice-like report in XML.

Styling

- Use simple CSS, don’t rely on Bootstrap.

Chapter 15: Wizard Reports with Abstract Models

Combine Wizard + Abstract Report

- Step 1: Wizard to ask for user input (e.g., date range).
- Step 2: Abstract model generates the actual report.

Example

- Choose a date range in wizard.
- Then download PDF of sales orders.