

CURSO PASO A PASO

OWL

para Odoo 20



DE TU PRIMER COMPONENTE AL PROYECTO FINAL · 17 CAPÍTULOS

Marlon Falcón Hernández

Valencia · 2026

Oct 30, 2026 · @generame una imagen de 3 arquitectos estilo pizaso

Curso paso a paso del framework web de Odoo, de tu primer componente a un proyecto completo

Autor: Marlon Falcón Hernández

ERP, CRM y Software · Valencia, España · marlonfalcon.com

Edición para Odoo 20.0 · Octubre de 2026

Continuación de *Odoo 20 para Nuevos Programadores*

Introducción

Este libro enseña OWL, el framework con el que está hecho todo el cliente web de Odoo 20. Es la continuación natural de *Odoo 20 para Nuevos Programadores*: allí aprendiste a crear modelos, vistas e informes en Python y XML; aquí aprenderás a construir la parte que se ejecuta en el navegador.

Cada capítulo sigue el mismo formato que el libro anterior: un ejercicio real que se construye por pasos, una prueba para comprobar que funciona, el resultado que verás en Odoo 20 y un reto para practicar solo.

A quién va dirigido

A quien ya sabe crear un módulo sencillo de Odoo (manifiesto, modelo, vista y menú) y quiere personalizar la interfaz. No hace falta ser experto en JavaScript: el Capítulo 0 repasa lo que necesitas del lenguaje.

Requisitos

Componente	Versión recomendada	Para qué
Odoo	20.0 Community o Enterprise	Donde se instalan los ejercicios
Python	La que pida Odoo 20	Servidor de Odoo
Navegador	Chrome, Edge o Firefox actuales	Ejecutar y depurar OWL
Editor	VS Code (o similar)	Escribir JS, XML y SCSS
Node.js	Opcional	Solo para herramientas como ESLint o Prettier

OWL no necesita compilación ni `npm install`: Odoo empaqueta tus archivos JavaScript, XML y SCSS en los assets cuando arranca.

Preparar el entorno

1. Instala Odoo 20 como en el capítulo *Instalar Odoo 20 en Windows y Mac* del libro anterior (Docker es la opción recomendada).
2. Crea una carpeta `addons_owl` y añádela al `addons_path` de tu configuración.
3. Arranca Odoo con el modo de desarrollo de assets: `odoo-bin -c odoo.conf --dev=xml, reload`.
4. En el navegador, activa el modo desarrollador con assets: añade `?debug=assets` a la URL. Así verás tu código sin minificar en las herramientas del navegador.

Cómo instalar cada ejercicio

Cada capítulo es un módulo independiente con el prefijo `owl_` (por ejemplo `owl_cap01_hola`). Cópialo en `addons_owl`, ve a *Aplicaciones*, pulsa *Actualizar lista de aplicaciones*, busca el módulo e instálalo. Cuando cambies solo JavaScript o XML de OWL, basta con recargar el navegador; cuando cambies el manifiesto o Python, reinicia Odoo y actualiza el módulo.

Anatomía de un módulo con OWL

```
owl_cap01_hola/
├─ __init__.py
├─ __manifest__.py
├─ views/
│   └─ menus.xml          # acción cliente y menú
├─ static/
│   └─ src/
│       └─ hola/
│           ├── hola.js    # el componente
│           ├── hola.xml   # su plantilla
│           └─ hola.scss   # sus estilos
└─ ...
```

La regla de Odoo es guardar juntos el JS, el XML y el SCSS de cada componente, en una carpeta con su nombre.

Cómo leer el código de este libro

Todo el código usa la API de OWL 2 que Odoo emplea desde la versión 17 y que sigue vigente en Odoo 20. Si quieres saber la versión exacta que trae tu instalación, abre la consola del navegador y escribe `owl.__info__.version`. Cuando algo cambió respecto a Odoo 16 o anteriores, lo verás marcado como **(cambio desde Odoo 17)**.

Qué es OWL y cómo encaja en Odoo 20

OWL (*Odoo Web Library*) es el framework de componentes que Odoo creó para su cliente web. Se parece a React y Vue: la interfaz se divide en componentes, cada componente tiene una plantilla y un estado, y cuando el estado cambia OWL vuelve a pintar solo lo necesario.

Por qué Odoo tiene su propio framework

- **Plantillas QWeb:** OWL usa el mismo lenguaje de plantillas XML que ya conoces de los informes PDF, así que no hay que aprender JSX.
- **Sin compilación:** las plantillas se compilan en el navegador; no necesitas Webpack ni npm.
- **Herencia de plantillas:** puedes modificar la plantilla de un componente de otro módulo con `t-inherit` y XPath, igual que haces con las vistas.
- **Parches:** puedes cambiar el comportamiento de cualquier componente de Odoo sin copiarlo, con `patch`.

Las piezas del cliente web

Pieza	Qué es	Dónde la usarás
Componente	Clase JS que extiende <code>Component</code> , con su plantilla	En todos los capítulos
Plantilla	XML QWeb con <code>t-name</code>	Capítulo 2
Estado	Objeto reactivo creado con <code>useState</code>	Capítulo 3
Entorno (<code>env</code>)	Objeto compartido por todos los componentes: servicios, bus, debug	Capítulos 5 y 7
Servicio	Pieza global sin interfaz: <code>orm</code> , <code>notification</code> , <code>action</code> , <code>dialog</code> ...	Capítulos 7 y 13
Registro	Lista con nombre donde Odoo busca piezas: <code>actions</code> , <code>fields</code> , <code>systray</code> , <code>views</code> , <code>services</code>	Capítulo 8

Pieza	Qué es	Dónde la usarás
Assets	Paquetes de JS, XML y SCSS que Odoo sirve al navegador	Capítulo 1

El ciclo de una pantalla

1. El usuario pulsa un menú. El servidor devuelve una acción (`ir.actions.act_window` o `ir.actions.client`).
2. El servicio `action` busca en los registros el componente que sabe mostrar esa acción.
3. OWL crea el componente, ejecuta `setup()` y espera a `onWillStart` si hay datos que cargar.
4. El componente pide datos al servidor con el servicio `orm`, que llama por RPC a los métodos de tus modelos Python.
5. OWL pinta la plantilla. Cuando el usuario interactúa, el estado cambia y OWL repinta solo lo que cambió.

Backend, sitio web y Punto de Venta

OWL está en tres sitios de Odoo 20, cada uno con su paquete de assets:

Lugar	Paquete de assets	Capítulo
Cliente web (backend)	<code>web.assets_backend</code>	Partes I y II
Punto de Venta	<code>point_of_sale._assets_pos</code>	Capítulo 15
Sitio web y portal	<code>web.assets_frontend</code>	Capítulo 16

Lo que más cambia respecto a Odoo 16 y anteriores

- **Adiós a `Widget` y `jQuery`** en el backend: todo es OWL desde Odoo 17, y el antiguo `publicWidget` del sitio web dejó paso a las *Interactions* en Odoo 18.
- **Módulos ES nativos:** los archivos de `static/src` usan `import` y `export`; ya no hace falta `odoo.define`.
- **`t-out` en lugar de `t-esc`** (cambio desde Odoo 17).
- **`patch(objeto, { ... })` sin nombre de parche** (cambio desde Odoo 17).
- **Pruebas con Hoot** en lugar de QUnit (cambio desde Odoo 18).

Capítulo 0 · JavaScript moderno para OWL

Este capítulo repasa solo el JavaScript que usarás en el libro. Si vienes de Python, verás que casi todo tiene un equivalente directo.

Variables: const y let

```
const nombre = "Marlon";    // no se puede reasignar (úsala por defecto)
let contador = 0;           // sí se puede reasignar
contador += 1;
```

Nunca uses `var`. En Odoo, el linter lo marca como error.

Texto con plantillas literales

Es el equivalente a las f-strings de Python:

```
const total = 42.5;
console.log(`El pedido de ${nombre} suma ${total} €`);
```

Arrays y objetos

```
const lineas = [
  { producto: "Mesa", cantidad: 2, precio: 100 },
  { producto: "Silla", cantidad: 4, precio: 25 },
];

const nombres = lineas.map((l) => l.producto);           // ["Mesa", "Silla"]
const grandes = lineas.filter((l) => l.cantidad > 2);    // solo la silla
const total = lineas.reduce((s, l) => s + l.cantidad * l.precio, 0); // 300
const mesa = lineas.find((l) => l.producto === "Mesa");
```

Python	JavaScript
<code>[x.name for x in lista]</code>	<code>lista.map((x) => x.name)</code>
<code>[x for x in lista if x.qty > 2]</code>	<code>lista.filter((x) => x.qty > 2)</code>
<code>sum(...)</code>	<code>lista.reduce((s, x) => s + x, 0)</code>
<code>dict.get("a", 0)</code>	<code>obj.a ?? 0</code>
<code>None</code>	<code>null</code> o <code>undefined</code>
<code>==</code>	<code>===</code> (usa siempre tres iguales)

Desestructuración y operador spread

```
const { producto, cantidad } = lineas[0];
const copia = { ...lineas[0], cantidad: 3 }; // copia cambiando un campo
const todas = [...lineas, { producto: "Lámpara", cantidad: 1, precio: 40 }];
```

Funciones flecha

```
function sumar(a, b) {
  return a + b;
}
const sumar2 = (a, b) => a + b;
```

Las funciones flecha no tienen su propio `this` : usan el del lugar donde se escriben. Por eso son ideales dentro de los métodos de un componente.

Clases y herencia

```
class Animal {
  constructor(nombre) {
    this.nombre = nombre;
  }
  hablar() {
    return `${this.nombre} hace un ruido`;
  }
}

class Perro extends Animal {
  hablar() {
    return super.hablar() + ": ¡guau!";
  }
}
```

Es la misma idea que la herencia de Python: `super` es `super()` . En OWL no escribirás `constructor` : usarás `setup()` .

Propiedades estáticas

```
class MiComponente {
  static template = "mi_modulo.MiComponente";
  static props = {};
}
```

Una propiedad `static` pertenece a la clase, no a cada objeto. OWL las usa para declarar la plantilla, las props y los subcomponentes.

Módulos: import y export

```
// utils.js
export function formatearPrecio(valor) {
  return `${valor.toFixed(2)} €`;
}

// otro_archivo.js
import { formatearPrecio } from "@owl_cap01_hola/utils";
```

En Odoo, la ruta empieza por `@nombre_del_modulo/` y apunta a la carpeta `static/src` de ese módulo, sin la extensión `.js`.

Promesas y async/await

Todo lo que va al servidor tarda, así que devuelve una promesa. Con `await` esperas el resultado como si fuera código normal:

```
async function cargarClientes(orm) {
  try {
    const clientes = await orm.searchRead("res.partner", [], ["name"]);
    return clientes;
  } catch (error) {
    console.error("No se pudieron cargar", error);
    return [];
  }
}
```

Depurar en el navegador

- `console.log(valor)` muestra cualquier cosa en la consola (F12).
- `debugger;` detiene el código en esa línea si las herramientas están abiertas.
- Con `?debug=assets` verás tus archivos con su nombre real en la pestaña *Sources*.
- Instala la extensión *Odoo Debug* o *OWL devtools* para ver el árbol de componentes y su estado.

Practica antes de seguir

1. Dada una lista de pedidos `{ cliente, total }`, obtén con `filter` y `map` los nombres de los clientes con total mayor que 1000.

2. Escribe una clase `Pedido` con un método `total()` y una subclase `PedidoConIVA` que lo sobrescriba usando `super`.
3. Escribe una función `async` que espere 1 segundo con `await new Promise((r) => setTimeout(r, 1000))` y luego muestre un mensaje.

Parte I · Fundamentos

Capítulo 1 · Hola mundo: tu primer componente

Vas a crear un módulo que añade una aplicación *Hola OWL* al menú principal. Al pulsarla se abre una pantalla hecha con un componente OWL propio.

Antes de empezar · Acción cliente

Hasta ahora tus menús abrían una `ir.actions.act_window`, que muestra una vista de un modelo. Una `ir.actions.client` abre en cambio un componente OWL cualquiera. Se identifica con un `tag`, y Odoo busca ese tag en el registro `actions` del navegador.

Paso 1 · Crea la carpeta y el manifiesto

```
owl_cap01_hola/
├── __init__.py
├── __manifest__.py
├── views/menus.xml
└── static/src/hola/
    ├── hola.js
    ├── hola.xml
    └── hola.scss
```

`__init__.py` queda vacío. El manifiesto:

```
{
  "name": "OWL · Cap 1 · Hola mundo",
  "version": "20.0.1.0.0",
  "category": "Tutorials",
  "depends": ["web"],
  "data": ["views/menus.xml"],
  "assets": {
    "web.assets_backend": [
      "owl_cap01_hola/static/src/**/*",
    ],
  },
  "license": "LGPL-3",
}
```

La clave `assets` es nueva para ti: le dice a Odoo que añada todos los archivos de `static/src` al paquete del backend. El patrón `**/*` incluye JS, XML y SCSS.

Paso 2 · Escribe el componente

`static/src/hola/hola.js`:

```
import { Component } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class Hola extends Component {
  static template = "owl_cap01_hola.Hola";
  static props = ["*"];
}

registry.category("actions").add("owl_cap01_hola.hola", Hola);
```

- `Component` es la clase base de todo componente OWL.
- `static template` es el nombre de la plantilla. La convención es `modulo.NombreDelComponente`.
- `static props = ["*"]` acepta cualquier prop. Una acción cliente recibe varias de Odoo (`action`, `actionId` ...) y así no tienes que declararlas.
- La última línea registra el componente con el tag de la acción.

Paso 3 · Escribe la plantilla

`static/src/hola/hola.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap01_hola.Hola">
    <div class="o_hola p-4">
      <h1>¡Hola, OWL!</h1>
      <p>Mi primer componente en Odoo 20.</p>
    </div>
  </t>
</templates>
```

Un componente debe tener un solo elemento raíz cuando Odoo lo monta como acción. Aquí es el `div`.

Paso 4 · Añade estilos

`static/src/hola/hola.scss`:

```
.o_hola {
  h1 {
    color: #714b67;
    font-weight: 700;
  }
}
```

Empieza siempre tus clases por `o_` y el nombre del componente para no chocar con los estilos de Odoo.

Paso 5 · Crea la acción y el menú

`views/menus.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <record id="action_hola" model="ir.actions.client">
    <field name="name">Hola OWL</field>
    <field name="tag">owl_cap01_hola.hola</field>
  </record>

  <menuitem id="menu_hola_root"
    name="Hola OWL"
    action="action_hola"
    sequence="1"/>
</odoo>
```

El `tag` debe ser exactamente el mismo texto que usaste en `registry.category("actions").add(...)`.

Prueba

1. Reinicia Odoo e instala *OWL · Cap 1 · Hola mundo*.
2. Recarga el navegador y abre la aplicación *Hola OWL*.
3. Abre la consola (F12) y comprueba que no hay errores en rojo.

Si ves el error *"There is no action registered for tag..."*, el tag del XML y el del JS no coinciden, o el JS no se cargó porque falta la clave `assets` en el manifiesto.

Resultado en Odoo 20

Aparece la aplicación *Hola OWL* en el menú principal. Al abrirla ves el título *¡Hola, OWL!* en color morado y el texto debajo, dentro del marco normal de Odoo con la barra superior.

Reto

Añade un segundo componente `Adios` en su propia carpeta, con su acción cliente y un submenú dentro de *Hola OWL*. Pista: los submenús llevan `parent="menu_hola_root"`.

Capítulo 2 · Plantillas QWeb en OWL

Vas a construir un pequeño catálogo de productos con datos fijos en el código. Servirá para practicar todas las directivas de plantilla que usarás el resto del libro.

Antes de empezar · Las directivas

Directiva	Para qué	Ejemplo
<code>t-out</code>	Escribir un valor (escapado)	<code></code>
<code>t-if</code> / <code>t-elif</code> / <code>t-else</code>	Mostrar según una condición	<code><p t-if="stock == 0">Agotado</p></code>
<code>t-foreach</code> + <code>t-as</code> + <code>t-key</code>	Repetir por cada elemento	<code><li t-foreach="lista" t-as="x" t-key="x.id"/></code>
<code>t-att-NOMBRE</code>	Atributo calculado	<code>t-att-disabled="!activo"</code>
<code>t-attf-NOMBRE</code>	Atributo con texto y valores	<code>t-attf-class="badge text-bg-{{color}}"</code>
<code>t-set</code>	Variable local en la plantilla	<code><t t-set="iva" t-value="0.21"/></code>

Directiva	Para qué	Ejemplo
<code>t-call</code>	Insertar otra plantilla	<code><t t-call="modulo.Fila"/></code>
<code>t-esc</code>	Igual que <code>t-out</code> ; obsoleta (cambio desde Odoo 17)	Usa <code>t-out</code>

Dentro de la plantilla, todo lo que escribas en una directiva es una expresión JavaScript que se evalúa sobre el componente. Por eso `producto.nombre` funciona si el componente tiene `this.productos` y estás dentro de un `t-foreach`.

Paso 1 · Manifiesto, acción y menú

Copia el módulo del capítulo 1 como `owl_cap02_plantillas` y cambia el nombre, el tag (`owl_cap02_plantillas.catalogo`) y los ids del XML. No cambia nada más.

Paso 2 · El componente con datos

`static/src/catalogo/catalogo.js`:

```
import { Component } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class Catalogo extends Component {
  static template = "owl_cap02_plantillas.Catalogo";
  static props = ["*"];

  setup() {
    this.productos = [
      { id: 1, nombre: "Mesa de roble", precio: 249.9, stock: 4,
        categoria: "Mobiliario" },
      { id: 2, nombre: "Silla nórdica", precio: 59.0, stock: 0,
        categoria: "Mobiliario" },
      { id: 3, nombre: "Lámpara LED", precio: 34.5, stock: 25,
        categoria: "Iluminación" },
    ];
  }

  colorStock(producto) {
    if (producto.stock === 0) {
      return "danger";
    }
    return producto.stock < 5 ? "warning" : "success";
  }
}
```

```

    }
}

registry.category("actions").add("owl_cap02_plantillas.catalogo", Catalogo);

```

`setup()` es el lugar donde se prepara todo lo que necesita el componente. Nunca escribas un `constructor` en un componente OWL.

Paso 3 · La plantilla principal

`static/src/catalogo/catalogo.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap02_plantillas.Catalogo">
    <div class="o_catalogo p-4">
      <h2>Catálogo (<t t-out="productos.length"/> productos)</h2>
      <t t-set="iva" t-value="0.21"/>
      <table class="table table-sm">
        <thead>
          <tr><th>Producto</th><th>Categoría</th><th>Precio con
IVA</th><th>Stock</th></tr>
        </thead>
        <tbody>
          <t t-foreach="productos" t-as="producto" t-
key="producto.id">
            <t t-call="owl_cap02_plantillas.FilaProducto"/>
          </t>
        </tbody>
      </table>
    </div>
  </t>

  <t t-name="owl_cap02_plantillas.FilaProducto">
    <tr t-att-class="{ 'text-muted': producto.stock === 0 }">
      <td t-out="producto.nombre"/>
      <td t-out="producto.categoria"/>
      <td t-out="(producto.precio * (1 + iva)).toFixed(2) + ' €'"/>
      <td>
        <span t-attf-class="badge text-bg-{{ colorStock(producto)
}}">
          <t t-if="producto.stock === 0">Agotado</t>
          <t t-elif="producto.stock < 5">Quedan <t t-
out="producto.stock"/></t>
          <t t-else="">En stock</t>
        </span>
      </td>
    </tr>
  </t>

```

```

        </td>
    </tr>
</t>
</templates>

```

Cuatro detalles que conviene fijar desde ya:

- **t-key** es obligatorio en **t-foreach**. OWL lo usa para saber qué fila es cuál cuando la lista cambia. Usa un id único, nunca el índice.
- **t-att-class** acepta un objeto: cada clave se añade si su valor es verdadero.
- En XML, **<** se escribe **<** dentro de los atributos. Para evitarlo, puedes invertir la comparación: `5 > producto.stock`.
- La plantilla llamada con **t-call** ve las variables de quien la llama (`producto`, `iva`).

Paso 4 · Plantillas en línea con xml

Para componentes muy pequeños, OWL permite escribir la plantilla dentro del JS:

```

import { Component, xml } from "@odoo/owl";

export class Etiqueta extends Component {
    static template = xml`<span class="badge text-bg-info" t-
out="props.texto"/>`;
    static props = { texto: String };
}

```

Es cómodo para pruebas, pero en un módulo real usa siempre archivos XML: solo esas plantillas se pueden heredar desde otros módulos.

Prueba

1. Instala el módulo y abre el menú *Catálogo*.
2. Cambia el stock de la lámpara a 3 en el JS y recarga el navegador. La etiqueta debe pasar a amarillo con el texto *Quedan 3*.
3. Con `?debug=assets` activo, pon a la lámpara el mismo id que la mesa (1) y recarga: OWL muestra en la consola un error de clave duplicada en **t-foreach**. Devuelve el id a 3.

Resultado en Odoo 20

Ves una tabla con tres productos. La silla aparece en gris con la etiqueta roja *Agotado*, la mesa con la etiqueta amarilla *Quedan 4* y la lámpara en verde con *En stock*. Los precios incluyen el 21 % de IVA.

Reto

Agrupar la tabla por categoría: muestra una fila de cabecera con el nombre de cada categoría y debajo sus productos. Pista: calcula en `setup()` una lista de categorías sin repetir con `[...new Set(this.productos.map((p) => p.categoria))]` y usa dos `t-foreach` anidados.

Capítulo 3 · Estado y reactividad con useState

Vas a crear una lista de tareas en la que se pueden añadir, marcar y borrar tareas. Todo vive en el navegador: en la Parte II la conectarás con un modelo de Odoo.

Antes de empezar · Qué es el estado

En el capítulo 2 los productos eran un array normal: si lo cambias, la pantalla no se entera. El **estado** es un objeto que OWL vigila. Lo creas con `useState` y, cada vez que cambias una de sus propiedades, OWL vuelve a pintar el componente.

```
this.state = useState({ contador: 0 });
this.state.contador++; // la pantalla se actualiza sola
```

Dos reglas: `useState` solo se llama dentro de `setup()`, y el estado se cambia directamente, sin funciones especiales como `setState`.

Paso 1 · Un contador para calentar

```
import { Component, useState, xml } from "@odoo/owl";

export class Contador extends Component {
  static template = xml`
    <div class="d-flex align-items-center gap-2">
      <button class="btn btn-secondary" t-on-click="() => state.valor--
    ">-</button>
      <span class="fs-4" t-out="state.valor"/>
      <button class="btn btn-primary" t-on-click="() =>
state.valor++">+</button>
    </div>`;
  static props = {};

  setup() {
    this.state = useState({ valor: 0 });
  }
}
```

Paso 2 · El componente de tareas

Crea el módulo `owl_cap03_estado` como en el capítulo 1, con el tag `owl_cap03_estado.tareas`. Luego `static/src/tareas/tareas.js`:

```
import { Component, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class Tareas extends Component {
  static template = "owl_cap03_estado.Tareas";
  static props = ["*"];

  setup() {
    this.siguienteId = 3;
    this.state = useState({
      nueva: "",
      filtro: "todas",
      tareas: [
        { id: 1, titulo: "Leer el capítulo 3", hecha: true },
        { id: 2, titulo: "Hacer el reto", hecha: false },
      ],
    });
  }

  get tareasVisibles() {
    const { tareas, filtro } = this.state;
    if (filtro === "pendientes") {
      return tareas.filter((t) => !t.hecha);
    }
    if (filtro === "hechas") {
      return tareas.filter((t) => t.hecha);
    }
    return tareas;
  }

  get pendientes() {
    return this.state.tareas.filter((t) => !t.hecha).length;
  }

  anadir() {
    const titulo = this.state.nueva.trim();
    if (!titulo) {
      return;
    }
    this.state.tareas.push({ id: this.siguienteId++, titulo, hecha: false });
  }
}
```

```

        this.state.nueva = "";
    }

    borrar(tarea) {
        const indice = this.state.tareas.indexOf(tarea);
        this.state.tareas.splice(indice, 1);
    }
}

registry.category("actions").add("owl_cap03_estado.tareas", Tareas);

```

- Los **getters** (`get tareasVisibles()`) son valores calculados: se recalculan en cada pintado y en la plantilla se usan sin paréntesis.
- El estado es **profundo**: `push`, `splice` o cambiar `tarea.hecha` dentro del array también repintan.
- `siguienteId` no está en el estado porque no se muestra: no hace falta que OWL lo vigile.

Paso 3 · La plantilla con t-model

`static/src/tareas/tareas.xml` :

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
    <t t-name="owl_cap03_estado.Tareas">
        <div class="o_tareas p-4" style="max-width: 600px">
            <h2>Mis tareas <small class="text-muted">(<t t-out="pendientes"/>
pendientes)</small></h2>

            <div class="input-group mb-3">
                <input class="form-control" placeholder="Nueva tarea" t-
model="state.nueva"/>
                <button class="btn btn-primary" t-on-
click="anadir">Añadir</button>
            </div>

            <select class="form-select mb-3" t-model="state.filtro">
                <option value="todas">Todas</option>
                <option value="pendientes">Pendientes</option>
                <option value="hechas">Hechas</option>
            </select>

            <ul class="list-group">
                <li t-foreach="tareasVisibles" t-as="tarea" t-key="tarea.id"
                    class="list-group-item d-flex align-items-center gap-2">

```

```

        <input type="checkbox" class="form-check-input" t-
model="tarea.hecha"/>
        <span t-att-class="{ 'text-decoration-line-through text-
muted': tarea.hecha }"
            t-out="tarea.titulo"/>
        <button class="btn btn-link text-danger ms-auto" t-on-
click="() => this.borrar(tarea)">
            <i class="fa fa-trash"/>
        </button>
    </li>
</ul>
</div>
</t>
</templates>

```

`t-model` une un campo del formulario con una propiedad del estado en las dos direcciones: lo que escribes llega al estado, y si el estado cambia, el campo también. Funciona con `input`, `textarea`, `select` y casillas.

Paso 4 · Estado compartido con reactive

`useState` está ligado a un componente. Si quieres un objeto reactivo fuera de un componente, por ejemplo en un archivo que importan varios, usa `reactive` y átaló al componente con `useState`:

```

// static/src/tareas/almacen.js
import { reactive } from "@odoo/owl";

export const almacen = reactive({ tareas: [] });

// en el setup() de cualquier componente
this.almacen = useState(almacen);

```

En el capítulo 13 verás la forma recomendada en Odoo: guardar ese objeto dentro de un servicio.

Prueba

1. Añade tres tareas y marca una. El contador de pendientes cambia al instante.
2. Elige *Hechas* en el filtro: solo quedan las tachadas.
3. Recarga el navegador: las tareas nuevas desaparecen, porque solo están en memoria.

Resultado en Odoo 20

Ves el título *Mis tareas (1 pendientes)*, un campo para añadir, un selector de filtro y una lista con casillas. Las tareas hechas aparecen tachadas y en gris, y cada fila tiene un icono de papelera rojo.

Reto

1. Haz que pulsar Intro en el campo añada la tarea. Pista: `t-on-keyup` y `ev.key === "Enter"`.
2. Añade un botón *Borrar hechas* que solo aparezca si hay alguna tarea hecha.

Capítulo 4 · Componentes hijos, props y slots

Vas a partir la lista de tareas del capítulo 3 en piezas: un componente `TareaItem` para cada fila y un componente `Tarjeta` reutilizable que enmarca cualquier contenido.

Antes de empezar · Props

Las **props** son los datos que un componente padre pasa a un hijo, como los argumentos de una función. El hijo las lee en `this.props` y **nunca las modifica**: si quiere cambiar algo, avisa al padre llamando a una función que el padre le pasó.

Paso 1 · El componente hijo

Copia el módulo del capítulo 3 como `owl_cap04_props`. Crea `static/src/tarea_item/tarea_item.js`:

```
import { Component } from "@odoo/owl";

export class TareaItem extends Component {
  static template = "owl_cap04_props.TareaItem";
  static props = {
    tarea: {
      type: Object,
      shape: { id: Number, titulo: String, hecha: Boolean },
    },
    onCambiar: Function,
    onBorrar: Function,
    editable: { type: Boolean, optional: true },
  };
  static defaultProps = {
    editable: true,
  };
}
```

Y su plantilla `tarea_item.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap04_props.TareaItem">
    <li class="list-group-item d-flex align-items-center gap-2">
      <input type="checkbox" class="form-check-input"
        t-att-checked="props.tarea.hecha"
        t-att-disabled="!props.editable"
        t-on-change="() => props.onCambiar(props.tarea.id)"/>
      <span t-att-class="{ 'text-decoration-line-through text-muted':
props.tarea.hecha }"
        t-out="props.tarea.titulo"/>
      <button t-if="props.editable" class="btn btn-link text-danger ms-
auto"
        t-on-click="() => props.onBorrar(props.tarea.id)">
        <i class="fa fa-trash"/>
      </button>
    </li>
  </t>
</templates>
```

Fíjate en que la casilla ya no usa `t-model` : no puede escribir en `props.tarea` . En su lugar llama a `onCambiar` y el padre hace el cambio.

Paso 2 · Validación de props

`static props` describe qué acepta el componente. En modo desarrollador (`? debug=assets`), OWL comprueba cada prop y lanza un error claro si falta una o tiene otro tipo.

Forma	Significado
<code>nombre: String</code>	Obligatoria y de tipo texto
<code>nombre: { type: Number, optional: true }</code>	Opcional
<code>nombre: [String, Boolean]</code>	Uno de varios tipos
<code>nombre: { type: Object, shape: {...} }</code>	Objeto con claves concretas
<code>nombre: { type: Array, element: Object }</code>	Array de objetos
<code>nombre: { validate: (v) => v > 0 }</code>	Validación propia
<code>static props = ["*"]</code>	Acepta cualquier cosa (sin validar)

`static defaultProps` da el valor de las props opcionales que el padre no pase.

Paso 3 · Usa el hijo desde el padre

En `tareas.js` importa el hijo, decláralo en `static components` y añade los métodos que recibirá:

```
import { TareaItem } from "../tarea_item/tarea_item";
import { Tarjeta } from "../tarjeta/tarjeta";

export class Tareas extends Component {
  static template = "owl_cap04_props.Tareas";
  static components = { TareaItem, Tarjeta };
  static props = ["*"];

  // setup(), getters y anadir() iguales que en el capítulo 3

  cambiar(id) {
    const tarea = this.state.tareas.find((t) => t.id === id);
    tarea.hecha = !tarea.hecha;
  }

  borrar(id) {
    this.state.tareas = this.state.tareas.filter((t) => t.id !== id);
  }
}
```

En la plantilla, la lista se reduce a:

```
<ul class="list-group">
  <TareaItem t-foreach="tareasVisibles" t-as="tarea" t-key="tarea.id"
    tarea="tarea"
    onCambiar.bind="cambiar"
    onBorrar.bind="borrar"/>
</ul>
```

- Un componente se escribe en la plantilla con mayúscula inicial, como una etiqueta.
- El valor de cada prop es una **expresión**: `tarea="tarea"` pasa la variable, no el texto. Para pasar un texto fijo escribe `titulo="'Hola'"`.
- **.bind** pasa el método unido al padre. Sin él, `this` dentro de `cambiar` no sería el padre y fallaría.

Paso 4 · Slots: un marco reutilizable

Un **slot** es un hueco en la plantilla del hijo que el padre rellena con su propio contenido.

Crea `static/src/tarjeta/tarjeta.js`:

```
import { Component } from "@odoo/owl";

export class Tarjeta extends Component {
  static template = "owl_cap04_props.Tarjeta";
  static props = {
    slots: { type: Object, shape: { default: Object, titulo: { type:
Object, optional: true } } },
  };
}
```

Y `tarjeta.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap04_props.Tarjeta">
    <div class="card shadow-sm mb-3">
      <div t-if="props.slots.titulo" class="card-header fw-bold">
        <t t-slot="titulo"/>
      </div>
      <div class="card-body">
        <t t-slot="default"/>
      </div>
    </div>
  </t>
</templates>
```

El padre la usa así:

```
<Tarjeta>
  <t t-set-slot="titulo">Mis tareas (<t t-out="pendientes"/> pendientes)
</t>
  <ul class="list-group">
    <TareaItem t-foreach="tareasVisibles" t-as="tarea" t-key="tarea.id"
      tarea="tarea" onChange.bind="cambiar"
      onBorrar.bind="borrar"/>
  </ul>
</Tarjeta>
```

Todo lo que no está dentro de un `t-set-slot` va al slot `default`. Los slots se reciben como una prop más, `slots`, por eso aparece en `static props`.

Prueba

1. Marca y borra tareas: todo funciona igual que en el capítulo 3, pero ahora cada fila es un `TareaItem`.

- Con `?debug=assets`, cambia `tarea="tarea"` por `tarea="tarea.titulo"`. La consola muestra un error de validación de la prop `tarea`. Deshaz el cambio.
- Añade `editable="false"` a `TareaItem`: desaparecen las papeleras y las casillas quedan bloqueadas.

Resultado en Odoo 20

La lista aparece dentro de una tarjeta con cabecera gris y el título *Mis tareas*. A simple vista es la misma pantalla del capítulo 3, pero el código está dividido en tres componentes reutilizables.

Reto

Crea un componente `Insignia` con las props `texto` (obligatoria) y `color` (opcional, por defecto `"secondary"`), y úsalo en la cabecera de la tarjeta para mostrar el número de pendientes.

Capítulo 5 · Eventos y comunicación entre componentes

Vas a añadir a la lista de tareas un buscador y un panel de actividad. El buscador avisa a su padre con una función; el panel se entera de lo que pasa a través del bus del entorno, sin que nadie le pase props.

Antes de empezar · Tres formas de comunicarse

Dirección	Herramienta	Cuándo
Padre → hijo	Props	Siempre que el padre tenga el dato
Hijo → padre	Función en una prop (<code>onAlgo.bind</code>)	El hijo avisa de una acción del usuario
Entre componentes lejanos	<code>env.bus</code> o un servicio	No hay un padre común cómodo

Usa la primera fila siempre que puedas, la segunda cuando haga falta y la tercera solo cuando las otras obliguen a pasar props por muchos niveles.

Paso 1 · Escuchar eventos del navegador con `t-on`

`t-on-NOMBRE` escucha cualquier evento DOM. El manejador recibe el evento como primer argumento:

```

<input t-on-input="onInput" t-on-keydown="onKeydown"/>
<button t-on-click="guardar">Guardar</button>
<form t-on-submit.prevent="enviar">...</form>
<div t-on-click.stop="() => this.abrir(id)">...</div>

```

Modificador	Efecto
<code>.prevent</code>	Llama a <code>ev.preventDefault()</code> (por ejemplo, evita que un formulario recargue la página)
<code>.stop</code>	Llama a <code>ev.stopPropagation()</code> (el clic no llega a los padres)
<code>.self</code>	Solo si el evento ocurre en ese mismo elemento, no en un hijo
<code>.capture</code>	Escucha en fase de captura

OWL no tiene modificadores de teclas como `.enter`: compruébalo tú con `ev.key`.

Paso 2 · El buscador, de hijo a padre

Copia el módulo del capítulo 4 como `owl_cap05_eventos`. Crea `static/src/buscador/buscador.js`:

```

import { Component, useState } from "@odoo/owl";
import { useDebounce } from "@web/core/utils/timing";

export class Buscador extends Component {
  static template = "owl_cap05_eventos.Buscador";
  static props = {
    onBuscar: Function,
    placeholder: { type: String, optional: true },
  };

  setup() {
    this.state = useState({ texto: "" });
    this.avisar = useDebounce(() =>
      this.props.onBuscar(this.state.texto), 300);
  }

  onInput(ev) {
    this.state.texto = ev.target.value;
    this.avisar();
  }
}

```

```

onKeyDown(ev) {
    if (ev.key === "Escape") {
        this.state.texto = "";
        this.props.onBuscar("");
    }
}
}

```

`buscador.xml` :

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
    <t t-name="owl_cap05_eventos.Buscador">
        <div class="input-group mb-3">
            <span class="input-group-text"><i class="fa fa-search"/></span>
            <input class="form-control"
                t-att-placeholder="props.placeholder or 'Buscar...'"
                t-att-value="state.texto"
                t-on-input="onInput"
                t-on-keydown="onKeyDown"/>
        </div>
    </t>
</templates>

```

`useDebounce` es una utilidad de Odoo: espera 300 ms sin que el usuario teclee antes de llamar al padre. Con ella no recalculas la lista en cada pulsación. En las plantillas puedes escribir `or` y `and` en lugar de `||` y `&&`.

En el padre, añade `busqueda: ""` al estado, un método y úsalo en el getter:

```

buscar(texto) {
    this.state.busqueda = texto.toLowerCase();
}

get tareasVisibles() {
    // ...filtro del capítulo 3 guardado en `lista`, y después:
    return lista.filter((t) =>
        t.titulo.toLowerCase().includes(this.state.busqueda));
}

```

```

<Buscador onBuscar.bind="buscar" placeholder="'Buscar tareas'"/>

```

Paso 3 · El bus del entorno

Todo componente de Odoo tiene `this.env`, un objeto compartido por toda la aplicación. `this.env.bus` es un `EventBus` de OWL: cualquiera puede lanzar un evento y cualquiera puede escucharlo.

En el padre, lanza un evento cada vez que cambia una tarea:

```
cambiar(id) {
  const tarea = this.state.tareas.find((t) => t.id === id);
  tarea.hecha = !tarea.hecha;
  this.env.bus.trigger("OWL_TAREAS:CAMBIO", { titulo: tarea.titulo, hecha:
tarea.hecha });
}
```

Y crea `static/src/actividad/actividad.js`, que lo escucha con el hook `useBus` de Odoo:

```
import { Component, useState } from "@odoo/owl";
import { useBus } from "@web/core/utils/hooks";

export class Actividad extends Component {
  static template = "owl_cap05_eventos.Actividad";
  static props = {};

  setup() {
    this.state = useState({ lineas: [] });
    useBus(this.env.bus, "OWL_TAREAS:CAMBIO", (ev) => {
      const { titulo, hecha } = ev.detail;
      const hora = new Date().toLocaleTimeString();
      this.state.lineas.unshift(`${hora} · ${titulo} → ${hecha} ?
"hecha" : "pendiente"}`);
    });
  }
}
```

```
<t t-name="owl_cap05_eventos.Actividad">
  <div class="small text-muted">
    <div class="fw-bold mb-1">Actividad</div>
    <div t-foreach="state.lineas" t-as="linea" t-key="linea_index" t-
out="linea"/>
  </div>
</t>
```

El dato del evento llega en `ev.detail`. `useBus` deja de escuchar solo cuando el componente desaparece, así que no tienes que limpiar nada. Pon a tus eventos un prefijo propio en mayúsculas para no chocar con los de Odoo.

Aquí usamos `linea_index` como clave: es la excepción aceptable, porque las líneas nunca se reordenan ni se editan.

Paso 4 · Compartir datos con useSubEnv

Si un componente quiere dar algo a todos sus descendientes, sin pasarlo como prop nivel a nivel, puede ampliar el entorno:

```
import { useSubEnv } from "@odoo/owl";

setup() {
  useSubEnv({ colorTema: "primary" });
}
// cualquier descendiente: this.env.colorTema
```

Prueba

1. Escribe en el buscador: la lista se filtra un momento después de dejar de teclear.
2. Pulsa Escape: el buscador se vacía y vuelven todas las tareas.
3. Marca y desmarca tareas: cada cambio aparece arriba del panel *Actividad* con la hora.

Resultado en Odoo 20

Encima de la lista hay un campo con lupa. Debajo, el panel *Actividad* muestra líneas como *12:41:07 · Hacer el reto → hecha*, la más reciente arriba.

Reto

Haz que el panel *Actividad* tenga un botón *Limpiar* que, además de vaciar su lista, lance el evento `OWL_TAREAS:LIMPIAR`. Haz que el componente de tareas lo escuche y muestre en la cabecera cuántas veces se ha limpiado.

Capítulo 6 · Ciclo de vida y hooks

Vas a crear un panel de bienvenida con un reloj que avanza cada segundo, un campo que recibe el foco al abrirse y unos datos que se cargan antes de pintar. Cada pieza usa un momento distinto de la vida del componente.

Antes de empezar · La vida de un componente

Hook	Cuándo se ejecuta	Uso típico
<code>setup()</code>	Al crear el componente	Estado, servicios y registrar los demás hooks
<code>onWillStart</code>	Antes del primer pintado; puede ser <code>async</code>	Cargar datos iniciales
<code>onMounted</code>	Cuando ya está en la página	Foco, medir el DOM, librerías externas
<code>onWillUpdateProps</code>	Antes de recibir props nuevas; puede ser <code>async</code>	Recargar datos si cambia un id
<code>onWillPatch</code> / <code>onPatched</code>	Antes y después de cada repintado	Guardar y restaurar el scroll
<code>onWillUnmount</code>	Justo antes de quitarlo de la página	Parar temporizadores y escuchas
<code>onWillDestroy</code>	Al destruirse, aunque no llegara a montarse	Limpieza final
<code>onError</code>	Si falla un hijo	Mostrar un aviso en lugar de romper la pantalla

Todos se importan de `@odoo/owl` y se llaman **dentro de `setup()`**, nunca en un método ni en un `if`.

Paso 1 · Cargar datos con `onWillStart`

Crea el módulo `owl_cap06_ciclo` con el tag `owl_cap06_ciclo.bienvenida`. En `static/src/bienvenida/bienvenida.js`:

```
import { Component, onWillStart, onMounted, onWillUnmount, useRef, useState,
useEffect } from "@odoo/owl";
import { registry } from "@web/core/registry";

function cargarFrases() {
  // En el capítulo 7 esto será una llamada real al servidor
  return new Promise((resolve) =>
    setTimeout(() => resolve(["Hoy es un buen día para programar", "Un
componente, una responsabilidad"]), 500)
  );
}
```

```

export class Bienvenida extends Component {
  static template = "owl_cap06_ciclo.Bienvenida";
  static props = ["*"];

  setup() {
    this.state = useState({ ahora: new Date(), nombre: "", frases: [] });
    this.campoNombre = useRef("campoNombre");

    onWillStart(async () => {
      this.state.frases = await cargarFrases();
    });

    onMounted(() => {
      this.campoNombre.el.focus();
      this.intervalo = setInterval(() => (this.state.ahora = new
Date()), 1000);
    });

    onWillUnmount(() => {
      clearInterval(this.intervalo);
    });

    useEffect(
      (nombre) => {
        document.title = nombre ? `Hola, ${nombre}` : "Odoo";
      },
      () => [this.state.nombre]
    );
  }

  get hora() {
    return this.state.ahora.toLocaleTimeString();
  }
}

registry.category("actions").add("owl_cap06_ciclo.bienvenida", Bienvenida);

```

Mientras la promesa de `onWillStart` no termina, OWL no pinta el componente. Por eso, cuando la pantalla aparece, las frases ya están ahí.

Paso 2 · La plantilla y t-ref

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap06_ciclo.Bienvenida">
    <div class="o_bienvenida p-4">
      <div class="display-4 font-monospace" t-out="hora"/>
      <input class="form-control my-3" style="max-width: 300px"
        placeholder="¿Cómo te llamas?"
        t-ref="campoNombre" t-model="state.nombre"/>
      <h3 t-if="state.nombre">¡Hola, <t t-out="state.nombre"/>!</h3>
      <ul>
        <li t-foreach="state.frases" t-as="frase" t-key="frase" t-
out="frase"/>
      </ul>
    </div>
  </t>
</templates>
```

`t-ref="campoNombre"` marca el elemento y `useRef("campoNombre")` lo recupera. El elemento real está en `.el`, y solo existe después de montar: en `setup()` u `onWillStart` vale `null`.

Paso 3 · Limpiar lo que abriste

El `setInterval` sigue vivo aunque cierres la pantalla, a menos que lo pares en `onWillUnmount`. Es el error más común con OWL: temporizadores y escuchas de `window` que nunca se limpian y siguen consumiendo memoria. Regla práctica: **todo lo que abras en `onMounted` se cierra en `onWillUnmount`**.

Paso 4 · `useEffect`

`useEffect` junta las dos cosas: la primera función se ejecuta tras montar y cada vez que cambian las dependencias que devuelve la segunda. Si la primera devuelve una función, OWL la llama para limpiar antes de la siguiente ejecución y al desmontar:

```
useEffect(
  () => {
    const alPulsar = (ev) => ev.key === "F2" &&
this.campoNombre.el.focus();
    window.addEventListener("keydown", alPulsar);
    return () => window.removeEventListener("keydown", alPulsar);
  },
  () => []
);
```

Con `() => []` el efecto se ejecuta una sola vez. Para escuchar eventos de `window` o `document`, OWL ofrece también el atajo `useExternalListener(window, "keydown", this.onKeyDown)`, que se limpia solo.

Paso 5 · Reaccionar a props nuevas

Cuando un componente hijo recibe un id por props y debe cargar datos, el patrón es este:

```
setup() {
  onWillStart(() => this.cargar(this.props.clienteId));
  onWillUpdateProps((nuevas) => {
    if (nuevas.clienteId !== this.props.clienteId) {
      return this.cargar(nuevas.clienteId);
    }
  });
}
```

Prueba

1. Abre el menú: tarda medio segundo en aparecer (la carga de frases) y el cursor ya está en el campo.
2. Escribe tu nombre: aparece el saludo y cambia el título de la pestaña del navegador.
3. Añade un `console.log("tic")` dentro del intervalo, abre otra aplicación y comprueba en la consola que los *tic* se detienen.

Resultado en Odoo 20

Ves un reloj grande en fuente monoespaciada que avanza cada segundo, un campo *¿Cómo te llamas?* con el cursor dentro, y debajo una lista con las dos frases. Al escribir aparece *¡Hola, Marlon!*.

Reto

Crea un componente hijo `Cuenta` que reciba la prop `segundos` y muestre una cuenta atrás. Cuando el padre le pase un valor distinto, debe reiniciarse. Usa `onWillUpdateProps` y recuerda parar su intervalo al desmontarse.

Parte II · OWL dentro de Odoo

Capítulo 7 · Servicios: hablar con el servidor

Vas a conectar la lista de tareas con un modelo real de Odoo, `owl.tarea`. Las tareas se leerán y guardarán en la base de datos, con avisos, confirmación antes de borrar y un botón para abrir el formulario estándar.

Antes de empezar · Qué es un servicio

Un servicio es una pieza global, sin interfaz, que Odoo arranca una sola vez y comparte con todos los componentes. Lo pides con `useService("nombre")` dentro de `setup()`.

Servicio	Para qué	Método más usado
<code>orm</code>	Leer y escribir modelos	<code>searchRead</code> , <code>create</code> , <code>write</code> , <code>unlink</code> , <code>call</code>
<code>notification</code>	Avisos en la esquina	<code>add(mensaje, { type })</code>
<code>action</code>	Abrir acciones y vistas	<code>doAction(...)</code>
<code>dialog</code>	Ventanas modales	<code>add(Componente, props)</code>
<code>user</code>	Datos del usuario conectado	<code>user.userId</code> , <code>user.hasGroup(...)</code> (se importa, ver abajo)
<code>company</code>	Compañía activa	<code>currentCompany</code>

`user` se importa directamente en lugar de pedirlo como servicio (**cambio desde Odoo 18**): `import { user } from "@web/core/user";`

Paso 1 · El modelo en Python

Crea `owl_cap07_servicios` con `depends: ["web"]`. `models/owl_tarea.py`:

```

from odoo import api, fields, models

class OwlTarea(models.Model):
    _name = "owl.tarea"
    _description = "Tarea OWL"
    _order = "hecha, id desc"

    name = fields.Char("Título", required=True)
    hecha = fields.Boolean("Hecha")
    user_id = fields.Many2one("res.users", "Responsable", default=lambda
self: self.env.user)

    @api.model
    def marcar_todas_hechas(self):
        tareas = self.search([("user_id", "=", self.env.uid), ("hecha", "=",
False)])
        tareas.hecha = True
        return len(tareas)

```

Y los permisos en `security/ir.access.csv`, con el formato de Odoo 20 que viste en el libro anterior:

```

id,name,model_id,group_id/id,operation,domain
access_owl_tarea_user,owl.tarea usuario,owl.tarea,base.group_user,crud,

```

No olvides los `__init__.py` y añadir el CSV y `views/menus.xml` a `data` en el manifiesto.

En `views/menus.xml`, la acción cliente se llama `action_tareas` y su tag es `owl_cap07_servicios.tareas`, igual que en el capítulo 1. El menú raíz se llama `menu_tareas_root`. Los capítulos 8 y 9 usarán esos dos ids.

Paso 2·Leer al arrancar

`static/src/tareas/tareas.js`:

```

import { Component, onWillStart, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { useService } from "@web/core/utils/hooks";
import { _t } from "@web/core/l10n/translation";
import { ConfirmationDialog } from
"@web/core/confirmation_dialog/confirmation_dialog";

const MODELO = "owl.tarea";

export class Tareas extends Component {

```

```

static template = "owl_cap07_servicios.Tareas";
static props = ["*"];

setup() {
  this.orm = useService("orm");
  this.notification = useService("notification");
  this.action = useService("action");
  this.dialog = useService("dialog");
  this.state = useState({ tareas: [], nueva: "" });

  onWillStart(() => this.cargar());
}

async cargar() {
  this.state.tareas = await this.orm.searchRead(MODELO, [], ["name",
"hecha"]);
}

```

`searchRead(modelo, dominio, campos, opciones)` es el `search_read` de Python. El dominio se escribe igual que en Python, con arrays en lugar de tuplas: `[["hecha", "=", false]]`. Como opciones puedes pasar `{ limit: 20, order: "name asc" }`.

Paso 3 · Crear, modificar y borrar

Sigue la misma clase:

```

async anadir() {
  const name = this.state.nueva.trim();
  if (!name) {
    return;
  }
  await this.orm.create(MODELO, [{ name }]);
  this.state.nueva = "";
  await this.cargar();
  this.notification.add(_t("Tarea creada"), { type: "success" });
}

async cambiar(tarea) {
  await this.orm.write(MODELO, [tarea.id], { hecha: !tarea.hecha });
  tarea.hecha = !tarea.hecha;
}

borrar(tarea) {
  this.dialog.add(ConfirmationDialog, {
    title: _t("Borrar tarea"),
    body: _t("¿Seguro que quieres borrar «%s»?", tarea.name),
  });
}

```

```

        confirmLabel: _t("Borrar"),
        confirm: async () => {
            await this.orm.unlink(MODELO, [tarea.id]);
            await this.cargar();
        },
        cancel: () => {},
    });
}

```

- `create` recibe una **lista** de diccionarios y devuelve la lista de ids creados.
- `write` y `unlink` reciben una lista de ids.
- `_t` marca el texto para traducir, como `_()` en Python (capítulo 4 del libro anterior).

Paso 4 · Llamar a un método propio y abrir acciones

```

    async marcarTodas() {
        const cuantas = await this.orm.call(MODELO, "marcar_todas_hechas",
[[]]);
        await this.cargar();
        this.notification.add(_t("%s tareas marcadas", cuantas), { type:
"info" });
    }

    abrir(tarea) {
        this.action.doAction({
            type: "ir.actions.act_window",
            res_model: MODELO,
            res_id: tarea.id,
            views: [[false, "form"]],
            target: "new",
        }, {
            onClose: () => this.cargar(),
        });
    }
}

registry.category("actions").add("owl_cap07_servicios.tareas", Tareas);

```

`orm.call(modelo, método, args, kwargs)` llama a cualquier método público del modelo. `doAction` acepta un diccionario de acción o el id XML de una acción existente (`"base.action_partner_form"`); con `target: "new"` se abre en una ventana modal.

Paso 5 · La plantilla

```
<t t-name="owl_cap07_servicios.Tareas">
  <div class="o_tareas p-4" style="max-width: 700px">
    <div class="d-flex mb-3 gap-2">
      <input class="form-control" placeholder="Nueva tarea" t-
model="state.nueva"
      t-on-keyup="(ev) => ev.key === 'Enter' and
this.anadir()"/>
      <button class="btn btn-primary" t-on-
click="anadir">Añadir</button>
      <button class="btn btn-secondary text-nowrap" t-on-
click="marcarTodas">Marcar todas</button>
    </div>
    <ul class="list-group">
      <li t-foreach="state.tareas" t-as="tarea" t-key="tarea.id"
class="list-group-item d-flex align-items-center gap-2">
        <input type="checkbox" class="form-check-input" t-att-
checked="tarea.hecha"
        t-on-change="( ) => this.cambiar(tarea)"/>
        <a href="#" t-on-click.prevent="( ) => this.abrir(tarea)" t-
out="tarea.name"/>
        <button class="btn btn-link text-danger ms-auto" t-on-click="(
) => this.borrar(tarea)">
          <i class="fa fa-trash"/>
        </button>
      </li>
    </ul>
  </div>
</t>
```

Prueba

1. Añade dos tareas y recarga el navegador: siguen ahí, porque ahora están en la base de datos.
2. Pulsa la papelera: aparece la ventana de confirmación. Cancela y luego confirma.
3. Pulsa el nombre de una tarea: se abre el formulario estándar en una ventana. Cámbiale el título, guarda y cierra; la lista se actualiza.
4. Abre la pestaña *Red* del navegador y mira las peticiones a
`/web/dataset/call_kw/owl.tarea/...`: así viaja cada llamada del servicio `orm`.

Resultado en Odoo 20

Ves la lista con tus tareas guardadas, un aviso verde *Tarea creada* en la esquina superior derecha al añadir, el diálogo *Borrar tarea* con los botones *Borrar* y *Cancelar*, y el formulario de `owl.tarea` en una ventana modal.

Reto

Añade un filtro *Solo las mías* que, al activarlo, cargue solo las tareas del usuario conectado.

Pista: `import { user } from "@web/core/user"` y el dominio `[["user_id", "=", user.userId]]`.

Capítulo 8 · Registros: dónde Odoo busca tus piezas

Vas a añadir dos cosas al cliente web sin tocar ningún archivo de Odoo: una entrada *Mis tareas* en el menú del usuario (arriba a la derecha) y un aviso fijo que aparece en todas las pantallas. Las dos se hacen igual: añadiendo algo al registro adecuado.

Antes de empezar · El registro

El **registro** es una colección de listas con nombre. Odoo no importa tus componentes directamente: los busca en estas listas por su clave. Por eso basta con añadir algo a la lista correcta para que Odoo lo use.

```
import { registry } from "@web/core/registry";

const acciones = registry.category("actions");
acciones.add("mi_clave", MiComponente);           // añadir
acciones.add("mi_clave", OtroComponente, { force: true }); // sustituir
acciones.get("mi_clave");                         // leer
acciones.contains("mi_clave");                    // ¿existe?
acciones.getAll();                               // todos, ordenados por
sequence
```

Las categorías que más usarás

Categoría	Qué guarda	Capítulo
<code>actions</code>	Componentes de acciones cliente (<code>ir.actions.client</code>)	1
<code>fields</code>	Widgets de campo (<code>widget="..."</code> en las vistas)	9
<code>views</code>	Tipos de vista: <code>list</code> , <code>form</code> , <code>kanban</code> ... y las tuyas	11

Categoría	Qué guarda	Capítulo
<code>systray</code>	Iconos de la barra superior	12
<code>services</code>	Servicios (<code>orm</code> , <code>notification</code> y los tuyos)	13
<code>user_menuitems</code>	Entradas del menú del usuario	Este capítulo
<code>main_components</code>	Componentes siempre presentes en la página	Este capítulo
<code>public.interactions</code>	Comportamiento en el sitio web	16

Para ver todas las categorías de tu instalación, escribe en la consola:

```
odoo.loader.modules.get("@web/core/registry").registry.subRegistries.
```

Paso 1 · Manifiesto

Crea `owl_cap08_registros` dependiendo de `owl_cap07_servicios` , porque reutilizará su acción de tareas:

```
{
  "name": "OWL · Cap 8 · Registros",
  "version": "20.0.1.0.0",
  "depends": ["owl_cap07_servicios"],
  "assets": {
    "web.assets_backend": ["owl_cap08_registros/static/src/**/*.js"],
  },
  "license": "LGPL-3",
}
```

Paso 2 · Una entrada en el menú del usuario

```
static/src/menu_usuario.js:
```

```
import { registry } from "@web/core/registry";
import { _t } from "@web/core/l10n/translation";

function misTareasItem(env) {
  return {
    type: "item",
    id: "owl_mis_tareas",
    description: _t("Mis tareas"),
    callback: () =>
      env.services.action.doAction("owl_cap07_servicios.action_tareas"),
    sequence: 5,
  };
}

registry.category("user_menuitems").add("owl_mis_tareas", misTareasItem);
```

En esta categoría no se registra un componente, sino una función que recibe el `env` y devuelve la descripción de la entrada. Fuera de un componente, los servicios están en `env.services`. El id XML de la acción debe existir: en el capítulo 7 la acción se llamaba `action_tareas`.

Paso 3 · Un componente en todas las pantallas

`static/src/aviso/aviso.js`:

```
import { Component, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class AvisoFormacion extends Component {
  static template = "owl_cap08_registros.AvisoFormacion";
  static props = {};

  setup() {
    this.state = useState({ visible: true });
  }
}

registry.category("main_components").add("owl_cap08_registros.AvisoFormacion", {
  Component: AvisoFormacion,
});
```

`aviso.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap08_registros.AvisoFormacion">
    <div t-if="state.visible"
      class="o_aviso_formacion position-fixed bottom-0 start-0 m-3
alert alert-warning shadow d-flex gap-3">
      <span>Estás en la base de datos de formación</span>
      <button class="btn-close" t-on-click="() => state.visible =
false"/>
    </div>
  </t>
</templates>
```

Los componentes de `main_components` se montan una vez, al cargar el cliente web, y siguen vivos al cambiar de menú. Odoo usa esta categoría para los diálogos, las notificaciones y la paleta de comandos.

Paso 4 · Cambiar algo de Odoo con force

Como los registros son públicos, puedes sustituir una pieza de Odoo registrando otra con la misma clave y `{ force: true }`. Es potente pero arriesgado: sustituyes la pieza entera y te pierdes sus mejoras futuras. En el capítulo 10 verás `patch`, que casi siempre es mejor opción.

Prueba

1. Instala el módulo y abre el menú de tu usuario: aparece *Mis tareas* y abre la lista del capítulo 7.
2. Cambia de aplicación varias veces: el aviso amarillo sigue abajo a la izquierda hasta que lo cierras.
3. Recarga el navegador: el aviso vuelve, porque su estado solo vive en memoria.

Resultado en Odoo 20

En el menú del usuario, encima de *Preferencias*, ves *Mis tareas*. En la esquina inferior izquierda de todas las pantallas aparece un recuadro amarillo *Estás en la base de datos de formación* con una aspa para cerrarlo.

Reto

Haz que el aviso solo aparezca a los administradores. Pista: `import { user } from "@web/core/user"`, carga `await user.hasGroup("base.group_system")` en `onWillStart` y guárdalo en el estado.

Capítulo 9 · Un widget de campo propio

Vas a crear el widget `owl_estrellas`, que muestra un campo entero como estrellas que se pueden pulsar. Lo usarás para un nuevo campo *Dificultad* en las tareas, en el formulario y en la lista.

Antes de empezar · Cómo pinta Odoo un campo

Cuando una vista tiene `<field name="dificultad" widget="owl_estrellas"/>`, Odoo busca `owl_estrellas` en el registro `fields` y monta ese componente con unas props estándar. Las dos más importantes son:

- `record`: el registro que se está editando. El valor está en `record.data[name]` y se cambia con `record.update({...})`.
- `name`: el nombre del campo.

También recibe `readonly`, `id` y, si las declaras, las opciones que pongas en la vista.

Paso 1 · El campo en Python

Crea `owl_cap09_campo` dependiendo de `owl_cap07_servicios`. `models/owl_tarea.py`:

```
from odoo import fields, models

class OwlTarea(models.Model):
    _inherit = "owl.tarea"

    dificultad = fields.Integer("Dificultad", default=0)
```

Paso 2 · El componente del widget

`static/src/estrellas/estrellas.js`:

```
import { Component } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { _t } from "@web/core/l10n/translation";
import { standardFieldProps } from "@web/views/fields/standard_field_props";

export class Estrellas extends Component {
    static template = "owl_cap09_campo.Estrellas";
    static props = {
        ...standardFieldProps,
        maximo: { type: Number, optional: true },
    };
    static defaultProps = { maximo: 5 };
```

```

    get valor() {
        return this.props.record.data[this.props.name] || 0;
    }

    get numeros() {
        return Array.from({ length: this.props.maximo }, (_, i) => i + 1);
    }

    elegir(n) {
        if (this.props.readonly) {
            return;
        }
        // Pulsar la estrella ya elegida la quita
        const nuevo = n === this.valor ? 0 : n;
        this.props.record.update({ [this.props.name]: nuevo });
    }
}

export const estrellas = {
    component: Estrellas,
    displayName: _t("Estrellas"),
    supportedTypes: ["integer"],
    extractProps: ({ options }) => ({
        maximo: options.maximo,
    }),
};

registry.category("fields").add("owl_estrellas", estrellas);

```

- Lo que se registra no es el componente solo, sino un objeto descriptor con el componente y su configuración.
- `standardFieldProps` declara de una vez todas las props que Odoo pasa a cualquier campo.
- `extractProps` traduce las `options` de la vista XML a props del componente.
- `supportedTypes` hace que Odoo avise si alguien usa el widget en un campo de otro tipo.

Paso 3 · La plantilla y los estilos

`estrellas.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap09_campo.Estrellas">
    <div class="o_estrellas d-inline-flex" t-att-class="{
'o_estrellas_editable': !props.readonly }">
      <i t-foreach="numeros" t-as="n" t-key="n"
        class="fa fa-lg"
        t-att-class="n &lt;= valor ? 'fa-star text-warning' : 'fa-
star-o text-muted'"
        t-att-title="n"
        t-on-click.stop="() => this.elegir(n)"/>
    </div>
  </t>
</templates>
```

estrellas.scss :

```
.o_estrellas_editable i {
  cursor: pointer;
  &:hover {
    transform: scale(1.2);
  }
}
```

El `.stop` del clic es importante en las listas: sin él, pulsar una estrella también abriría el registro.

Paso 4 · Usa el widget en las vistas

views/owl_tarea_views.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <record id="owl_tarea_view_form" model="ir.ui.view">
    <field name="name">owl.tarea.form</field>
    <field name="model">owl.tarea</field>
    <field name="arch" type="xml">
      <form>
        <sheet>
          <group>
            <field name="name"/>
            <field name="hecha"/>
            <field name="dificultad" widget="owl_estrellas"
options="{ 'maximo': 5 }"/>
            <field name="user_id"/>
          </group>
```

```

        </sheet>
    </form>
</field>
</record>

<record id="owl_tarea_view_list" model="ir.ui.view">
    <field name="name">owl.tarea.list</field>
    <field name="model">owl.tarea</field>
    <field name="arch" type="xml">
        <list editable="bottom">
            <field name="name"/>
            <field name="hecha"/>
            <field name="dificultad" widget="owl_estrellas"/>
        </list>
    </field>
</record>

<record id="action_owl_tarea_list" model="ir.actions.act_window">
    <field name="name">Tareas (vista estándar)</field>
    <field name="res_model">owl.tarea</field>
    <field name="view_mode">list,form</field>
</record>

<menuitem id="menu_owl_tarea_list" name="Tareas (lista)"
    parent="owl_cap07_servicios.menu_tareas_root"
    action="action_owl_tarea_list"/>
</odoo>

```

Ajusta `parent` al id del menú raíz que creaste en el capítulo 7.

Prueba

1. Actualiza el módulo y abre *Tareas (lista)*. Cada fila muestra cinco estrellas vacías.
2. Pulsa la tercera estrella de una fila: se rellenan tres y la fila queda pendiente de guardar.
3. Abre una tarea en formulario: las estrellas responden al ratón. Pulsa la tercera otra vez para dejarla a 0.
4. Escribe `widget="owl_estrellas"` en el campo `name` (texto): con `?debug=assets`, Odoo no puede pintarlo porque el widget solo admite enteros; vuelve a dejarlo en dificultad.

Resultado en Odoo 20

En la lista y en el formulario, el campo *Dificultad* aparece como una fila de estrellas: las elegidas en amarillo y el resto en gris de contorno. Al pasar el ratón por encima, la estrella crece un poco.

Reto

Añade la opción `icono` para que el mismo widget pueda mostrar corazones (`fa-heart`) o rayos (`fa-bolt`) en lugar de estrellas. Necesitarás una prop nueva, leerla en `extractProps` y usarla en la plantilla con `t-attf-class`.

Capítulo 10 · Modificar componentes de Odoo con patch y t-inherit

Vas a cambiar un componente que no es tuyo sin copiarlo: añadirás a la lista de tareas del capítulo 7 un contador y una regla nueva desde otro módulo, y luego harás lo mismo con un componente de Odoo. Es la versión JavaScript de la herencia que ya conoces de Python y de las vistas.

Antes de empezar · Dos herramientas

Qué quieres cambiar	Herramienta	Equivale a
La plantilla (lo que se ve)	<code>t-inherit</code> con XPath	Heredar una vista con <code>inherit_id</code>
El comportamiento (el JS)	<code>patch</code>	<code>_inherit</code> en un modelo Python

Paso 1 · Manifiesto

Crea `owl_cap10_patch` dependiendo de `owl_cap07_servicios`. El orden de carga importa: como tu módulo depende del otro, sus archivos se cargan después y pueden modificar los de aquel.

Paso 2 · Heredar una plantilla

```
static/src/tareas_patch.xml :
```

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-inherit="owl_cap07_servicios.Tareas" t-inherit-mode="extension">
    <xpath expr="//ul[class='list-group']" position="before">
      <p class="text-muted small">
        <t t-out="pendientes"/> pendientes de <t t-
out="state.tareas.length"/>
      </p>
    </xpath>
    <xpath expr="//button[class='text-danger']"
position="attributes">
      <attribute name="t-att-disabled">tarea.hecha</attribute>
    </xpath>
  </t>
</templates>
```

- `t-inherit-mode="extension"` modifica la plantilla original en todas partes.
- `t-inherit-mode="primary"` con un `t-name` nuevo crea una copia modificada, para un componente tuyo que extiende al original.
- Las posiciones son las de siempre: `before`, `after`, `inside`, `replace` y `attributes`.

Paso 3 · Parchear el comportamiento

`static/src/tareas_patch.js`:

```
import { patch } from "@web/core/utils/patch";
import { _t } from "@web/core/l10n/translation";
import { Tareas } from "@owl_cap07_servicios/tareas/tareas";

patch(Tareas.prototype, {
  get pendientes() {
    return this.state.tareas.filter((t) => !t.hecha).length;
  },

  async anadir() {
    if (this.state.nueva.trim().length < 3) {
      this.notification.add(_t("El título debe tener al menos 3
letras"), { type: "warning" });
      return;
    }
    return super.anadir(...arguments);
  },
});
```

- Se parchea `Tareas.prototype`, es decir, los métodos que comparten todos los objetos de esa clase.
- `super.anadir(...arguments)` llama al método original, igual que `super()` en Python.
- Para que la importación funcione, la clase original debe tener `export`. Por eso en todos los capítulos escribimos `export class`.

(cambio desde Odoo 17): antes se escribía `patch(Clase.prototype, "nombre_del_parche", { ... })` y se llamaba a `this._super()`. Ahora no hay nombre y se usa `super` normal.

Paso 4 · Parchear `setup()`

Para añadir estado o hooks a un componente ajeno, parchea `setup` y llama siempre primero al original:

```
import { useState } from "@odoo/owl";

patch(Tareas.prototype, {
  setup() {
    super.setup(...arguments);
    this.extra = useState({ mostrarHechas: true });
  },
});
```

Paso 5 · Lo mismo con un componente de Odoo

El mismo método sirve para cualquier pieza de Odoo. Por ejemplo, avisar cada vez que se abre un formulario de contacto:

```
import { onMounted } from "@odoo/owl";
import { FormController } from "@web/views/form/form_controller";

patch(FormController.prototype, {
  setup() {
    super.setup(...arguments);
    onMounted(() => {
      if (this.props.resModel === "res.partner") {
        console.log("Formulario de contacto abierto",
this.props.resId);
      }
    });
  },
});
```

Para encontrar el nombre de una clase o de una plantilla de Odoo, activa `?debug=assets`, busca en la pestaña *Sources* el texto que ves en pantalla y mira el `t-name` de su plantilla y la clase que la usa.

Buenas prácticas

- Parchea lo más pequeño posible: un método, no la clase entera.
- Llama siempre a `super` salvo que quieras anular el original a propósito.
- Filtra pronto (`if (this.props.resModel === ...)`): un parche en `FormController` afecta a **todos** los formularios.
- Si tu cambio solo tiene sentido en una vista, crea una vista propia (capítulo 11) en lugar de parchear la general.

Prueba

1. Instala `owl_cap10_patch` y abre la lista de tareas: sobre la lista aparece *X pendientes de Y*.
2. Intenta añadir la tarea *ab*: sale un aviso naranja y no se crea.
3. Marca una tarea como hecha: su papelera queda desactivada.
4. Abre cualquier contacto y mira la consola: aparece *Formulario de contacto abierto* con su id.
5. Desinstala `owl_cap10_patch`: la lista vuelve a ser exactamente la del capítulo 7.

Resultado en Odoo 20

La lista de tareas muestra una línea gris pequeña con el recuento, las papeleras de las tareas hechas aparecen atenuadas, y un título de menos de tres letras provoca el aviso *El título debe tener al menos 3 letras*.

Reto

Parchea el componente `Estrellas` del capítulo 9 desde un módulo nuevo para que, al elegir 5 estrellas, muestre la notificación *¡Tarea de máxima dificultad!*. Pista: el componente no tiene el servicio de notificaciones; añádelo parcheando `setup`.

Capítulo 11 · Personalizar vistas: controlador de lista y kanban

Vas a añadir un botón *Marcar todas como hechas* junto al botón *Nuevo* de la lista de tareas, y un panel de resumen encima del kanban. Ninguno de los dos cambios afecta a otras listas o kanbans de Odoo, porque se activan solo en tus vistas con `js_class`.

Antes de empezar · De qué está hecha una vista

Pieza	Qué hace	Ejemplo en la lista
Controller	Coordina: panel de control, botones, acciones	ListController
Renderer	Pinta los registros	ListRenderer
Model	Carga y guarda los datos	RelationalModel
ArchParser	Lee el XML de la vista	ListArchParser

Una vista es un objeto que agrupa estas piezas, como `listView` o `kanbanView`. Para personalizar una vista se copia ese objeto cambiando solo la pieza que interesa y se registra con un nombre nuevo en el registro `views`.

Paso 1 · Manifiesto

Crea `owl_cap11_vistas` dependiendo de `owl_cap09_campo`, que ya tiene la lista y el formulario estándar de `owl.tarea`.

Paso 2 · Un controlador de lista propio

`static/src/lista_tareas/lista_tareas.js`:

```
import { registry } from "@web/core/registry";
import { useService } from "@web/core/utils/hooks";
import { _t } from "@web/core/l10n/translation";
import { listView } from "@web/views/list/list_view";
import { ListController } from "@web/views/list/list_controller";

export class TareasListController extends ListController {
  static template = "owl_cap11_vistas.TareasListView";

  setup() {
    super.setup();
    this.orm = useService("orm");
    this.notification = useService("notification");
  }

  async marcarTodas() {
    const cuantas = await this.orm.call("owl.tarea",
"marcar_todas_hechas", []);
    await this.model.load();
    this.notification.add(_t("%s tareas marcadas como hechas", cuantas),
{ type: "success" });
  }
}
```

```

    }
}

export const tareasListView = {
  ...listView,
  Controller: TareasListController,
};

registry.category("views").add("owl_tareas_list", tareasListView);

```

Aquí sí se usa herencia de clases (`extends ListView`) y no `patch` : queremos una variante nueva que conviva con la original, no cambiar la original. `this.model.load()` vuelve a leer los registros para que la lista muestre el cambio.

Paso 3 · La plantilla del botón

`lista_tareas.xml` :

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
  <t t-name="owl_cap11_vistas.TareasListView" t-inherit="web.ListView" t-
inherit-mode="primary">
    <xpath expr="//div[class='o_list_buttons']" position="inside">
      <button type="button" class="btn btn-secondary" t-on-
click="marcarTodas">
        Marcar todas como hechas
      </button>
    </xpath>
  </t>
</templates>

```

Con `t-inherit-mode="primary"` creas una plantilla nueva a partir de `web.ListView` , sin tocar la original. Si en tu versión de Odoo 20 el XPath no encuentra el `div` , abre la plantilla `web.ListView` con `?debug=assets` y busca el contenedor de los botones de la barra.

Paso 4 · Actívalo en la vista XML

En la vista de lista del capítulo 9, añade `js_class` heredando la vista desde tu módulo:

```

<record id="owl_tarea_view_list_js" model="ir.ui.view">
  <field name="name">owl.tarea.list.js_class</field>
  <field name="model">owl.tarea</field>
  <field name="inherit_id" ref="owl_cap09_campo.owl_tarea_view_list"/>
  <field name="arch" type="xml">
    <list position="attributes">
      <attribute name="js_class">owl_tareas_list</attribute>
    </list>
  </field>
</record>

```

Paso 5 · Un panel encima del kanban

El mismo patrón sirve para el kanban, esta vez cambiando el `Renderer` :

```

import { kanbanView } from "@web/views/kanban/kanban_view";
import { KanbanRenderer } from "@web/views/kanban/kanban_renderer";

export class TareasKanbanRenderer extends KanbanRenderer {
  static template = "owl_cap11_vistas.TareasKanbanRenderer";

  get resumen() {
    const registros = this.props.list.records;
    const hechas = registros.filter((r) => r.data.hecha).length;
    return { total: registros.length, hechas };
  }
}

registry.category("views").add("owl_tareas_kanban", {
  ...kanbanView,
  Renderer: TareasKanbanRenderer,
});

```

```

<t t-name="owl_cap11_vistas.TareasKanbanRenderer" t-
inherit="web.KanbanRenderer" t-inherit-mode="primary">
  <xpath expr="//div[class='o_kanban_renderer']" position="before">
    <div class="alert alert-info m-2">
      <t t-out="resumen.hechas"/> de <t t-out="resumen.total"/> tareas
      hechas en esta página
    </div>
  </xpath>
</t>

```

Igual que en la lista, si el XPath no encuentra el elemento en tu versión, abre la plantilla `web.KanbanRenderer` con `?debug=assets` y elige el contenedor donde quieras colocar el panel.

Y una vista kanban para `owl.tarea` con `js_class="owl_tareas_kanban"`:

```
<kanban js_class="owl_tareas_kanban">
  <templates>
    <t t-name="card">
      <field name="name" class="fw-bold"/>
      <field name="dificultad" widget="owl_estrellas"/>
      <field name="hecha" widget="boolean_toggle"/>
    </t>
  </templates>
</kanban>
```

Añade `kanban` al `view_mode` de la acción: `list,kanban,form`. En las tarjetas, el nombre de la plantilla es `card` desde Odoo 18.

Prueba

1. Abre *Tareas (lista)*: junto a *Nuevo* aparece *Marcar todas como hechas*. Púlsalo y las casillas se marcan sin recargar.
2. Abre la lista de contactos: no tiene el botón, porque allí no hay `js_class`.
3. Cambia a la vista kanban: encima de las tarjetas aparece el panel azul con el recuento.

Resultado en Odoo 20

La lista de tareas tiene un botón gris *Marcar todas como hechas* al lado del botón morado *Nuevo*. En el kanban, un recuadro azul dice, por ejemplo, *2 de 5 tareas hechas en esta página*, y cada tarjeta muestra sus estrellas.

Reto

Ocultar el botón *Marcar todas* cuando no haya ninguna tarea pendiente en la página. Pista: en el controlador, `this.model.root.records` contiene los registros cargados.

Capítulo 12 · Un icono en la barra superior (systray)

Vas a poner en la barra superior de Odoo, junto a los mensajes y las actividades, un icono con el número de tareas pendientes. Al pulsarlo se despliega un menú con las cinco primeras y un enlace para verlas todas.

Antes de empezar · La barra superior

La zona derecha de la barra se llama *systray*. Odoo pinta en orden todo lo que haya en el registro `systray`: cada entrada es un objeto con la clave `Component` y, opcionalmente, una `sequence` que decide la posición (números más altos quedan más a la izquierda).

Paso 1 · Manifiesto

Crea `owl_cap12_systray` dependiendo de `owl_cap07_servicios`.

Paso 2 · El componente

`static/src/systray_tareas/systray_tareas.js`:

```
import { Component, onWillStart, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { useService } from "@web/core/utils/hooks";
import { user } from "@web/core/user";
import { Dropdown } from "@web/core/dropdown/dropdown";
import { DropdownItem } from "@web/core/dropdown/dropdown_item";

export class SystrayTareas extends Component {
  static template = "owl_cap12_systray.SystrayTareas";
  static components = { Dropdown, DropdownItem };
  static props = {};

  setup() {
    this.orm = useService("orm");
    this.action = useService("action");
    this.state = useState({ total: 0, tareas: [] });
    onWillStart(() => this.cargar());
  }

  get dominio() {
    return [
      ["user_id", "=", user.userId],
      ["hecha", "=", false]
    ];
  }

  async cargar() {
    const [total, tareas] = await Promise.all([
      this.orm.searchCount("owl.tarea", this.dominio),
      this.orm.searchRead("owl.tarea", this.dominio, ["name"], { limit:
5 }),
    ]);
    Object.assign(this.state, { total, tareas });
  }
}
```

```

abrirTodas() {
    this.action.doAction("owl_cap07_servicios.action_tareas");
}

abrirTarea(tarea) {
    this.action.doAction({
        type: "ir.actions.act_window",
        res_model: "owl.tarea",
        res_id: tarea.id,
        views: [[false, "form"]],
    });
}

registry.category("systray").add("owl_cap12_systray.SystrayTareas", {
    Component: SystrayTareas }, { sequence: 30 });

```

`Promise.all` lanza las dos peticiones a la vez y espera a ambas: la barra superior debe cargar rápido, porque aparece en todas las pantallas.

Paso 3 · La plantilla con Dropdown

`systray_tareas.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
    <t t-name="owl_cap12_systray.SystrayTareas">
        <Dropdown beforeOpen.bind="cargar" position="'bottom-end'">
            <button class="o_nav_entry position-relative" title="Mis tareas">
                <i class="fa fa-check-square-o fa-lg"/>
                <span t-if="state.total" class="o_badge badge rounded-pill
text-bg-danger"
                    t-out="state.total"/>
            </button>
            <t t-set-slot="content">
                <div class="dropdown-header">Tareas pendientes</div>
                <DropdownItem t-foreach="state.tareas" t-as="tarea" t-
key="tarea.id"
                    onSelect="() => this.abrirTarea(tarea)">
                    <t t-out="tarea.name"/>
                </DropdownItem>
                <div t-if="!state.tareas.length" class="dropdown-item-text
text-muted">¡Nada pendiente!</div>
                <div class="dropdown-divider"/>
                <DropdownItem onSelect.bind="abrirTodas">Ver
todas</DropdownItem>
            </Dropdown>
        </t>
    </templates>

```

```

        </t>
    </Dropdown>
</t>
</templates>

```

- `Dropdown` y `DropdownItem` son componentes de Odoo: úsalos en lugar de construir tu propio menú, y tendrás gratis el teclado, el cierre al pulsar fuera y la posición.
- El primer hijo de `Dropdown` es el botón que lo abre; el contenido va en el slot `content` (**sintaxis desde Odoo 18**).
- `beforeOpen` vuelve a cargar los datos justo antes de abrir, así el número está al día sin recargar la página.
- La clase `o_nav_entry` da al botón el mismo aspecto que el resto de iconos de la barra.

Paso 4 · Mantener el número al día

Si el usuario crea una tarea desde la lista del capítulo 7, el contador no se entera hasta que abres el menú. Para arreglarlo, conecta las dos piezas con el bus del capítulo 5: la lista lanza `this.env.bus.trigger("OWL_TAREAS:CAMBIO")` después de crear, cambiar o borrar, y el systray escucha:

```

import { useBus } from "@web/core/utils/hooks";

// dentro de setup()
useBus(this.env.bus, "OWL_TAREAS:CAMBIO", () => this.cargar());

```

Prueba

1. Instala el módulo y recarga: aparece el icono con un globo rojo con el número de tus tareas pendientes.
2. Pulsa el icono: se despliegan hasta cinco tareas. Pulsa una y se abre su formulario.
3. Marca todas como hechas desde la lista y abre de nuevo el menú: el globo desaparece y sale *¡Nada pendiente!*.

Resultado en Odoo 20

En la barra superior, a la izquierda de los iconos de conversaciones y actividades, hay un icono de casilla marcada con un globo rojo. Al pulsarlo se abre un menú con el título *Tareas pendientes*, la lista de tareas y, debajo de una línea, *Ver todas*.

Reto

Añade al menú desplegable un campo de texto para crear una tarea rápida sin salir de la pantalla actual. Al pulsar Intro, créala con `orm.create`, vacía el campo y recarga la lista. Pista: los clics dentro del campo no deben cerrar el menú; usa un `div` normal, no un `DropDownItem`.

Parte III · Avanzado

Capítulo 13 · Un servicio propio

Vas a sacar la lógica de las tareas de los componentes y meterla en un servicio, `owl_tareas`. La lista, el systray y cualquier pantalla futura compartirán los mismos datos: si una cambia una tarea, las demás se actualizan solas, sin bus ni recargas.

Antes de empezar · Por qué un servicio

En los capítulos 7 y 12 cada componente leía sus propias tareas y había que avisar con el bus para sincronizarlos. Un servicio arranca una sola vez, guarda el estado en un objeto `reactive` y lo reparte a todos. Es el mismo patrón que usa Odoo para las notificaciones, el usuario o el carrito del sitio web.

Paso 1 · Manifiesto

Crea `owl_cap13_servicio` dependiendo de `owl_cap07_servicios`. Este módulo reemplaza las pantallas de los capítulos 7 y 12, así que desinstala `owl_cap12_systray` antes de instalarlo para no ver dos iconos.

Paso 2 · El servicio

`static/src/tareas_service.js:`

```
import { reactive } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { _t } from "@web/core/l10n/translation";

const MODELO = "owl.tarea";

export const tareasService = {
  dependencies: ["orm", "notification"],

  start(env, { orm, notification }) {
    const state = reactive({ tareas: [], cargado: false });
```

```

    async function cargar() {
        state.tareas = await orm.searchRead(MODELO, [], ["name",
"hecha"]);
        state.cargado = true;
    }

    async function crear(name) {
        await orm.create(MODELO, [{ name }]);
        await cargar();
        notification.add(_t("Tarea creada"), { type: "success" });
    }

    async function cambiar(id) {
        const tarea = state.tareas.find((t) => t.id === id);
        await orm.write(MODELO, [id], { hecha: !tarea.hecha });
        tarea.hecha = !tarea.hecha;
    }

    async function borrar(id) {
        await orm.unlink(MODELO, [id]);
        state.tareas = state.tareas.filter((t) => t.id !== id);
    }

    return {
        state,
        cargar,
        crear,
        cambiar,
        borrar,
        get pendientes() {
            return state.tareas.filter((t) => !t.hecha);
        },
    };
},
};

registry.category("services").add("owl_tareas", tareasService);

```

- `dependencies` lista los servicios que necesitas; Odoo arranca primero esos y te los pasa en el segundo argumento de `start`.
- Lo que devuelve `start` es lo que recibirá quien haga `useService("owl_tareas")`.
- El estado es `reactive`, no `useState`: un servicio no es un componente.

Paso 3 · Un hook para usarlo

Para que un componente se repinte cuando cambia el estado del servicio, hay que atar ese estado con `useState`. Envuélvelo en un hook propio, `static/src/use_tareas.js`:

```
import { onWillStart, useState } from "@odoo/owl";
import { useService } from "@web/core/utils/hooks";

export function useTareas() {
  const servicio = useService("owl_tareas");
  const state = useState(servicio.state);
  onWillStart(async () => {
    if (!state.cargado) {
      await servicio.cargar();
    }
  });
  return { servicio, state };
}
```

Un hook es solo una función que empieza por `use` y llama a otros hooks. Igual que ellos, solo se puede usar dentro de `setup()`.

Paso 4 · Los componentes, mucho más cortos

La lista:

```
export class Tareas extends Component {
  static template = "owl_cap13_servicio.Tareas";
  static props = ["*"];

  setup() {
    const { servicio, state } = useTareas();
    this.tareas = servicio;
    this.datos = state;
    this.nueva = useState({ texto: "" });
  }

  async anadir() {
    await this.tareas.crear(this.nueva.texto);
    this.nueva.texto = "";
  }
}
```

En la plantilla usas `datos.tareas`, `() => this.tareas.cambiar(tarea.id)` y `() => this.tareas.borrar(tarea.id)`. El systray se queda en:

```

setup() {
  const { state } = useTareas();
  this.datos = state;
}

get pendientes() {
  return this.datos.tareas.filter((t) => !t.hecha);
}
// en la plantilla: pendientes.length y pendientes.slice(0, 5)

```

Fíjate en que el getter lee `this.datos`, el estado atado con `useState`. Si leyeras `servicio.pendientes`, que usa el objeto `reactive` sin atar, el componente no se enteraría de los cambios y no se repintaría.

Prueba

1. Abre la lista de tareas y deja el menú del systray a la vista.
2. Marca una tarea en la lista: el número del systray baja en el mismo instante, sin bus ni recarga.
3. En la consola, escribe `odoo.__WOWL_DEBUG__.root.env.services.owl_tareas.state.tareas` para ver el estado del servicio.

Resultado en Odoo 20

La pantalla es igual que en los capítulos 7 y 12, pero ahora la lista y el globo rojo del systray están siempre de acuerdo: cualquier cambio en uno se ve al momento en el otro.

Reto

Añade al servicio un método `marcarTodas()` que use `orm.call("owl.tarea", "marcar_todas_hechas", [])` y actualice el estado. Después, haz que el botón del capítulo 11 lo use en lugar de llamar al ORM directamente.

Capítulo 14 · Pruebas con Hoot y depuración

Vas a escribir pruebas automáticas para el contador del capítulo 3 y para el widget de estrellas del capítulo 9. Se ejecutan en el navegador, tardan segundos y te avisan si un cambio futuro rompe algo.

Antes de empezar · Qué es Hoot

Hoot es el framework de pruebas de JavaScript de Odoo desde la versión 18, y sustituye a QUnit (**cambio desde Odoo 18**). Trae tres paquetes:

Paquete	Para qué
@odoo/hoot	test , describe , expect y los ganchos beforeEach ...
@odoo/hoot-dom	Simular al usuario: click , fill , press , queryAll ...
@odoo/hoot-mock	Controlar el tiempo y la red: animationFrame , advanceTime , mockDate ...

Además, Odoo ofrece ayudas para montar componentes y vistas con un servidor simulado en @web/.../tests/web_test_helpers .

Paso 1 · Manifiesto

Crea owl_cap14_pruebas dependiendo de owl_cap09_campo y declara tus pruebas en el paquete de pruebas unitarias:

```
"assets": {
  "web.assets_backend": ["owl_cap14_pruebas/static/src/**/*"],
  "web.assets_unit_tests": ["owl_cap14_pruebas/static/tests/**/*"],
},
```

Los archivos de prueba terminan en .test.js . Copia el componente Contador del capítulo 3 a static/src/contador.js y añádele export .

Paso 2 · Probar un componente suelto

static/tests/contador.test.js :

```

import { expect, test } from "@odoo/hoot";
import { click } from "@odoo/hoot-dom";
import { animationFrame } from "@odoo/hoot-mock";
import { mountWithCleanup } from "@web/../tests/web_test_helpers";
import { Contador } from "@owl_cap14_pruebas/contador";

test("el contador sube y baja", async () => {
  await mountWithCleanup(Contador);
  expect(".fs-4").toHaveText("0");

  await click(".btn-primary");
  await animationFrame();
  expect(".fs-4").toHaveText("1");

  await click(".btn-secondary");
  await click(".btn-secondary");
  await animationFrame();
  expect(".fs-4").toHaveText("-1");
});

```

OWL repinta en el siguiente fotograma, no al instante. Por eso, después de cada clic, `await animationFrame()` espera a que la pantalla se actualice antes de comprobarla.

Paso 3 · Probar un widget dentro de una vista

`static/tests/estrellas.test.js`:

```

import { expect, test } from "@odoo/hoot";
import { contains, defineModels, fields, models, mountView } from
"@web/../tests/web_test_helpers";

class OwlTarea extends models.Model {
  _name = "owl.tarea";

  name = fields.Char();
  dificultad = fields.Integer();

  _records = [{ id: 1, name: "Escribir pruebas", dificultad: 2 }];
}

defineModels([OwlTarea]);

test("muestra tantas estrellas llenas como la dificultad", async () => {
  await mountView({
    type: "form",
    resModel: "owl.tarea",

```

```

        resId: 1,
        arch: `<form><field name="dificultad" widget="owl_estrellas"/>
</form>`,
    });
    expect(".o_estrellas .fa-star").toHaveCount(2);
    expect(".o_estrellas .fa-star-o").toHaveCount(3);
});

test("pulsar una estrella cambia el valor", async () => {
    await mountView({
        type: "form",
        resModel: "owl.tarea",
        resId: 1,
        arch: `<form><field name="dificultad" widget="owl_estrellas"/>
</form>`,
    });
    await contains(".o_estrellas i:nth-child(4)").click();
    expect(".o_estrellas .fa-star").toHaveCount(4);
});

```

- `defineModels` crea un servidor simulado: el modelo y sus registros solo existen durante la prueba, sin tocar tu base de datos.
- `mountView` monta una vista completa con su `arch`, igual que haría Odoo.
- `contains(...).click()` busca el elemento, espera a que exista, lo pulsa y espera al repintado.

Paso 4 · Ejecutar las pruebas

1. Arranca Odoo e instala `owl_cap14_pruebas`.
2. Abre `http://localhost:8069/web/tests?debug=assets`.
3. En el buscador de Hoot escribe `estrellas` o `contador` para ejecutar solo las tuyas.

Para ejecutarlas sin navegador, por ejemplo en integración continua, Odoo las lanza desde una prueba de Python con Chrome en modo *headless*. Consulta en el código de Odoo 20 cómo lo hace el módulo `web` en `tests/test_js.py`.

Paso 5 · Depurar cuando algo falla

Síntoma	Primera comprobación
La pantalla sale en blanco	Consola: casi siempre hay un error de plantilla o de import

Síntoma	Primera comprobación
<i>Cannot find key ... in the ... registry</i>	El JS no se cargó: revisa <code>assets</code> en el manifiesto y reinicia
<i>Missing template</i>	El <code>t-name</code> no coincide con <code>static template</code>
<i>Invalid props</i>	Mira qué prop indica el mensaje y compárala con <code>static props</code>
No se repinta	El dato no está en un <code>useState</code> o lo estás leyendo sin pasar por él
Funciona en local y no en producción	Assets en caché: actualiza el módulo y regenera los assets

Tres herramientas que te ahorrarán horas: `?debug=assets` para leer tu código sin minificar, un `debugger`; en el método sospechoso, y la extensión *Owl Devtools* para ver el árbol de componentes con sus props y su estado.

Prueba

1. Ejecuta las pruebas: las tres salen en verde.
2. Cambia en `estrellas.js` la condición `n <= valor` por `n < valor` y vuelve a ejecutarlas: fallan las dos de estrellas y Hoot muestra cuántas esperaba y cuántas encontró. Deshaz el cambio.

Resultado en Odoo 20

La página de Hoot muestra la lista de pruebas filtrada con tus tres pruebas en verde, el tiempo que tardó cada una y, al pulsar una, el detalle de cada `expect`.

Reto

Escribe una prueba que monte el widget en un formulario con `readonly="1"` en el campo y compruebe que pulsar una estrella no cambia el número de estrellas llenas.

Capítulo 15 · OWL en el Punto de Venta

Vas a añadir a la pantalla de productos del TPV un botón *Descuento 10 %* que aplica un descuento a la línea seleccionada, y que pide confirmación si el pedido no tiene cliente. Verás que todo lo aprendido sirve igual: componentes, `t-inherit`, `patch` y servicios.

Antes de empezar · El TPV es una aplicación OWL aparte

El Punto de Venta no se carga dentro del cliente web normal: es otra aplicación OWL con su propio paquete de assets, `point_of_sale._assets_pos`, y su propio almacén de datos, el servicio `pos`. Ese servicio contiene la sesión, los productos, los clientes y el pedido en curso, y funciona incluso sin conexión.

Aviso de versión. El TPV es la parte de Odoo que más se reorganiza entre versiones: las rutas de los archivos y algunos nombres de métodos cambian. El código de este capítulo sigue la organización de Odoo 18 y 19. Antes de copiarlo, abre el TPV con `?debug=assets` y comprueba en *Sources* la ruta real de cada import en tu Odoo 20.

Paso 1 · Manifiesto

```
{
  "name": "OWL · Cap 15 · Punto de Venta",
  "version": "20.0.1.0.0",
  "depends": ["point_of_sale"],
  "assets": {
    "point_of_sale._assets_pos": [
      "owl_cap15_pos/static/src/**/*",
    ],
  },
  "license": "LGPL-3",
}
```

Fíjate en el paquete: si pones tus archivos en `web.assets_backend`, el TPV nunca los verá.

Paso 2 · Parchear los botones de control

`static/src/boton_descuento.js`:

```
import { patch } from "@web/core/utils/patch";
import { _t } from "@web/core/l10n/translation";
import { ConfirmationDialog } from
"@web/core/confirmation_dialog/confirmation_dialog";
import { ControlButtons } from
"@point_of_sale/app/screens/product_screen/control_buttons/control_buttons";

patch(ControlButtons.prototype, {
  async aplicarDescuento10() {
    const pedido = this.pos.getOrder();
    const linea = pedido?.getSelectedOrderline();
    if (!linea) {
      this.notification.add(_t("Selecciona primero una línea"), { type:
"warning" });
    }
  }
});
```

```

        return;
    }
    if (!pedido.getPartner()) {
        const confirmado = await new Promise((resolve) => {
            this.dialog.add(ConfirmationDialog, {
                title: _t("Pedido sin cliente"),
                body: _t("¿Aplicar el descuento sin cliente asignado?"),
                confirm: () => resolve(true),
                cancel: () => resolve(false),
            });
        });
        if (!confirmado) {
            return;
        }
    }
    linea.setDiscount(10);
},
});

```

`ControlButtons` ya tiene `this.pos`, `this.dialog` y `this.notification` en su `setup`. Si en tu versión alguno falta, añádelo parcheando `setup` como en el capítulo 10:

```
this.notification = useService("notification").
```

En Odoo 18 los métodos del pedido se escribían con guiones bajos (`get_order()`, `get_selected_orderline()`, `set_discount()`); en las versiones recientes están en *camelCase*. Si recibes *is not a function*, prueba la otra forma.

Paso 3 · El botón en la plantilla

`static/src/boton_descuento.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">
    <t t-inherit="point_of_sale.ControlButtons" t-inherit-mode="extension">
        <xpath expr="//t[@t-if='!props.showRemainingButtons']"
position="before">
            <button class="btn btn-light btn-lg lh-lg" t-on-
click="aplicarDescuento10">
                <i class="fa fa-percent me-1"/> Descuento 10 %
            </button>
        </xpath>
    </t>
</templates>

```

El XPath depende de la estructura interna de la plantilla. Si no encuentra el nodo, abre `point_of_sale.ControlButtons` en *Sources* y elige otro punto de anclaje, por ejemplo el primer `button` con `position="before"`.

Paso 4 · Un componente propio en el TPV

También puedes crear componentes nuevos. Para leer el estado del TPV desde uno de ellos, usa el hook `usePos`:

```
import { Component } from "@odoo/owl";
import { usePos } from "@point_of_sale/app/hooks/pos_hook";

export class TotalArticulos extends Component {
  static template = "owl_cap15_pos.TotalArticulos";
  static props = {};

  setup() {
    this.pos = usePos();
  }

  get articulos() {
    const pedido = this.pos.getOrder();
    return pedido ? pedido.getOrderlines().reduce((s, l) => s +
l.getQuantity(), 0) : 0;
  }
}
```

El servicio `pos` es reactivo: cuando cambian las líneas del pedido, `TotalArticulos` se repinta solo, como el `systray` del capítulo 13.

Prueba

1. Instala el módulo, abre una sesión de TPV y añade dos productos.
2. Selecciona una línea y pulsa *Descuento 10 %*: sale el diálogo *Pedido sin cliente*. Confirma y la línea muestra *Con un 10 % de descuento*.
3. Asigna un cliente y repite con la otra línea: esta vez se aplica sin preguntar.
4. Sin ninguna línea seleccionada (pedido vacío), pulsa el botón: aparece el aviso naranja.

Resultado en Odoo 20

En la zona de botones bajo el pedido aparece *Descuento 10 %* con el icono de porcentaje, con el mismo estilo que *Nota* o *Reembolso*. La línea afectada muestra el descuento y el total del pedido baja.

Reto

Cambia el botón para que pida el porcentaje con un diálogo de número. Pista: busca en el código del TPV el componente `NumberPopup` y cómo lo usa el botón de descuento estándar del módulo `pos_discount`.

Capítulo 16 · Sitio web y portal: Interactions

Vas a crear la página pública `/owl/tareas`, donde un usuario conectado ve sus tareas pendientes y puede marcarlas como hechas sin recargar. La página se pinta en el servidor con QWeb, y el comportamiento lo añade una *Interaction*.

Antes de empezar · Dos formas de usar JS en el sitio web

El sitio web no es una aplicación OWL como el backend: el servidor envía HTML ya pintado (bueno para buscadores y velocidad), y el JavaScript solo le añade vida.

Herramienta	Cuándo usarla
<code>Interaction</code>	Añadir comportamiento a HTML que ya pintó el servidor. Es lo normal en el sitio web
Componente OWL con <code><owl-component></code>	Un bloque muy interactivo que se pinta entero en el navegador

Las *Interactions* sustituyen al antiguo `publicWidget` basado en jQuery (**cambio desde Odoo 18**).

Paso 1 · Manifiesto y controlador

Crea `owl_cap16_web` dependiendo de `website` y `owl_cap07_servicios`:

```
"depends": ["website", "owl_cap07_servicios"],
"data": ["views/templates.xml"],
"assets": {
    "web.assets_frontend": ["owl_cap16_web/static/src/**/*.js"],
},
```

`controllers/main.py`:

```
from odoo import http
from odoo.http import request

class OwlTareasWeb(http.Controller):
```

```

@http.route("/owl/tareas", type="http", auth="user", website=True)
def pagina(self):
    tareas = request.env["owl.tarea"].search([
        ("user_id", "=", request.env.uid),
        ("hecha", "=", False),
    ])
    return request.render("owl_cap16_web.pagina_tareas", {"tareas":
tareas})

@http.route("/owl/tareas/hecha", type="jsonrpc", auth="user")
def marcar_hecha(self, tarea_id):
    tarea = request.env["owl.tarea"].browse(int(tarea_id)).exists()
    if tarea and tarea.user_id == request.env.user:
        tarea.hecha = True
        return True
    return False

```

Las rutas JSON se declaran con `type="jsonrpc"` (**cambio desde Odoo 19**); antes era `type="json"`. Recuerda importar `controllers` en el `__init__.py` del módulo.

Paso 2 · La plantilla QWeb de la página

`views/templates.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<odoo>
    <template id="pagina_tareas" name="Mis tareas">
        <t t-call="website.layout">
            <div class="container py-5 o_owl_tareas_web">
                <h1>Mis tareas pendientes</h1>
                <p class="o_owl_contador text-muted">
                    Te quedan <span class="o_owl_numero" t-
out="len(tareas)"/> tareas
                </p>
                <ul class="list-group">
                    <li t-foreach="tareas" t-as="tarea"
                        class="list-group-item d-flex justify-content-between
align-items-center">
                        <span t-out="tarea.name"/>
                        <button class="btn btn-sm btn-success o_owl_hecha" t-
att-data-id="tarea.id">Hecha</button>
                    </li>
                </ul>
            </div>
        </t>
    </template>

```

```

        </div>
    </t>
</template>
</odoo>

```

Esta plantilla es QWeb de servidor (Python), como la de los informes PDF: aquí sí se usa `len()` y no hace falta `t-key`.

Paso 3 · La Interaction

`static/src/tareas_web.js`:

```

import { Interaction } from "@web/public/interaction";
import { registry } from "@web/core/registry";
import { rpc } from "@web/core/network/rpc";

export class TareasWeb extends Interaction {
    static selector = ".o_owl_tareas_web";

    dynamicContent = {
        ".o_owl_hecha": { "t-on-click": this.marcar },
        ".o_owl_numero": { "t-out": () => this.pendientes },
    };

    setup() {
        this.pendientes = this.el.querySelectorAll(".o_owl_hecha").length;
    }

    async marcar(ev) {
        const boton = ev.currentTarget;
        const ok = await this.waitFor(rpc("/owl/tareas/hecha", { tarea_id:
boton.dataset.id }));
        if (ok) {
            boton.closest("li").remove();
            this.pendientes--;
        }
    }
}

registry.category("public.interactions").add("owl_cap16_web.tareas_web",
TareasWeb);

```

- `static selector` dice sobre qué elementos se activa: Odoo crea una Interaction por cada elemento que coincida, y `this.el` es ese elemento.

- `dynamicContent` usa directivas parecidas a las de OWL (`t-on-click`, `t-att-...`, `t-out`) sobre selectores CSS dentro de `this.el` . Tras cada manejador, Odoo vuelve a aplicar las directivas, por eso el número se actualiza solo.
- `this.waitFor(promesa)` evita que el código siga si la Interaction se destruye mientras espera, por ejemplo al entrar en el editor del sitio web.
- Como en `setup()` de OWL, el ciclo de vida tiene `setup` , `willStart` , `start` y `destroy` .

Paso 4 · Un componente OWL dentro de la página

Si un bloque necesita pintarse entero en el navegador, registra un componente OWL en `public_components` y colócalo en la plantilla con la etiqueta `owl-component` :

```
import { Component, useState, xml } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class Animo extends Component {
  static template = xml`
    <button class="btn btn-outline-primary" t-on-click="() =>
state.clics++">
      ¡Ánimo! (<t t-out="state.clics"/>)
    </button>`;
  static props = { inicial: { type: Number, optional: true } };

  setup() {
    this.state = useState({ clics: this.props.inicial || 0 });
  }
}

registry.category("public_components").add("owl_cap16_web.Animo", Animo);
```

```
<owl-component name="owl_cap16_web.Animo" props='{ "inicial": 3 }'/>
```

Prueba

1. Crea dos o tres tareas desde el backend y abre `http://localhost:8069/owl/tareas` .
2. Pulsa *Hecha* en una: la fila desaparece y el contador baja sin recargar.
3. Vuelve al backend: esa tarea aparece marcada.
4. Cierra sesión y abre la URL: Odoo te lleva a la pantalla de inicio de sesión, porque la ruta es `auth="user"` .

Resultado en Odoo 20

Ves una página con la cabecera y el pie de tu sitio web, el título *Mis tareas pendientes*, la frase *Te quedan 3 tareas* y una lista con un botón verde *Hecha* en cada fila. Debajo, si añadiste el Paso 4, un botón *¡Ánimo! (3)* que suma al pulsarlo.

Reto

Añade un enlace *Mis tareas* en la página */my* del portal. Pista: hereda la plantilla `portal.portal_my_home` y usa `portal.portal_docs_entry` como hacen los módulos de ventas o proyectos.

Capítulo 17 · Proyecto final: tablero de tareas del equipo

Vas a juntar todo el libro en un módulo completo, `owl_tablero`: un tablero con tres columnas (*Pendiente*, *En curso*, *Hecha*) donde el equipo arrastra tarjetas de una columna a otra. Cada tarjeta muestra el responsable y la dificultad con estrellas, y el systray avisa de cuántas tareas tienes en curso.

Antes de empezar · Qué vas a reutilizar

Pieza del tablero	Lo aprendiste en
Acción cliente y menú	Capítulo 1
Plantillas con <code>t-foreach</code> y <code>t-att-class</code>	Capítulo 2
Componentes <code>Columna</code> y <code>Tarjeta</code> con props	Capítulo 4
Eventos de arrastrar y soltar	Capítulo 5
Carga inicial con <code>onWillStart</code>	Capítulo 6
Servicio <code>owl_tablero</code> con estado compartido	Capítulos 7 y 13
Estrellas de dificultad	Capítulo 9
Contador en el systray	Capítulo 12
Pruebas con Hoot	Capítulo 14

Paso 1 · Estructura del módulo

```
owl_tablero/
├── __manifest__.py          # depends: owl_cap09_campo
├── models/owl_tarea.py      # campo estado
├── views/menus.xml         # acción cliente owl_tablero.tablero
├── static/
│   ├── src/
│   │   ├── tablero_service.js
│   │   ├── tablero/tablero.js + .xml + .scss
│   │   ├── columna/columna.js + .xml
│   │   ├── tarjeta/tarjeta.js + .xml
│   │   └── systray/systray_en_curso.js + .xml
└── tests/tablero.test.js
```

Paso 2 · El modelo

```
from odoo import fields, models

class OwlTarea(models.Model):
    _inherit = "owl.tarea"

    estado = fields.Selection(
        [("pendiente", "Pendiente"), ("en_curso", "En curso"), ("hecha",
        "Hecha")],
        default="pendiente",
        required=True,
    )

    def write(self, vals):
        if "estado" in vals:
            vals["hecha"] = vals["estado"] == "hecha"
        return super().write(vals)
```

Así el campo `hecha` de los capítulos anteriores sigue funcionando: se marca solo cuando la tarjeta llega a la columna *Hecha*.

Paso 3 · El servicio

```
import { reactive } from "@odoo/owl";
import { registry } from "@web/core/registry";

export const COLUMNAS = [
    { clave: "pendiente", titulo: "Pendiente" },
```

```

    { clave: "en_curso", titulo: "En curso" },
    { clave: "hecha", titulo: "Hecha" },
  ];

export const tableroService = {
  dependencies: ["orm"],
  start(env, { orm }) {
    const state = reactive({ tareas: [], cargado: false });

    async function cargar() {
      state.tareas = await orm.searchRead(
        "owl.tarea", [], ["name", "estado", "dificultad", "user_id"]
      );
      state.cargado = true;
    }

    async function mover(id, estado) {
      const tarea = state.tareas.find((t) => t.id === id);
      const anterior = tarea.estado;
      tarea.estado = estado; // optimista: se ve al instante
      try {
        await orm.write("owl.tarea", [id], { estado });
      } catch (error) {
        tarea.estado = anterior; // si falla, vuelve atrás
        throw error;
      }
    }

    return { state, cargar, mover };
  },
};

registry.category("services").add("owl_tablero", tableroService);

```

La actualización **optimista** mueve la tarjeta antes de que responda el servidor y la devuelve a su sitio si hay un error. La interfaz se siente instantánea.

Paso 4 · La tarjeta

```
export class Tarjeta extends Component {
  static template = "owl_tablero.Tarjeta";
  static props = { tarea: Object, onAbrir: Function };

  onDragStart(ev) {
    ev.dataTransfer.setData("text/plain", String(this.props.tarea.id));
  }

  get estrellas() {
    return Array.from({ length: 5 }, (_, i) => i <
this.props.tarea.dificultad);
  }
}
```

```
<t t-name="owl_tablero.Tarjeta">
  <div class="o_tablero_tarjeta card mb-2 shadow-sm" draggable="true"
    t-on-dragstart="onDragStart" t-on-click="() =>
props.onAbrir(props.tarea.id)">
    <div class="card-body p-2">
      <div class="fw-bold" t-out="props.tarea.name"/>
      <div class="d-flex justify-content-between small">
        <span class="text-muted" t-out="props.tarea.user_id and
props.tarea.user_id[1]"/>
        <span>
          <i t-foreach="estrellas" t-as="llena" t-key="llena_index"
            t-att-class="llena ? 'fa fa-star text-warning' : 'fa
fa-star-o text-muted'"/>
          </span>
        </div>
      </div>
    </div>
  </div>
</t>
```

Un campo Many2one llega de `searchRead` como `[id, nombre]`, por eso se muestra `user_id[1]`.

Paso 5 · La columna

```
export class Columna extends Component {
  static template = "owl_tablero.Columna";
  static components = { Tarjeta };
  static props = { columna: Object, tareas: Array, onSoltar: Function,
onAbrir: Function };

  setup() {
    this.state = useState({ encima: false });
  }

  onDrop(ev) {
    this.state.encima = false;
    const id = parseInt(ev.dataTransfer.getData("text/plain"));
    this.props.onSoltar(id, this.props.columna.clave);
  }
}
```

```
<t t-name="owl_tablero.Columna">
  <div class="o_tablero_columna flex-fill p-2 rounded"
    t-att-class="{ 'o_tablero_encima': state.encima }"
    t-on-dragover.prevent="() => state.encima = true"
    t-on-dragleave="() => state.encima = false"
    t-on-drop.prevent="onDrop">
    <h5 class="d-flex justify-content-between">
      <span t-out="props.columna.titulo"/>
      <span class="badge text-bg-secondary" t-
out="props.tareas.length"/>
    </h5>
    <Tarjeta t-foreach="props.tareas" t-as="tarea" t-key="tarea.id"
      tarea="tarea" onAbrir="props.onAbrir"/>
  </div>
</t>
```

`t-on-dragover.prevent` es obligatorio: sin `preventDefault`, el navegador no permite soltar.

Paso 6 · El tablero

```
export class Tablero extends Component {
  static template = "owl_tablero.Tablero";
  static components = { Columna };
  static props = ["*"];
```

```

    setup() {
        this.servicio = useService("owl_tablero");
        this.state = useState(this.servicio.state);
        this.action = useService("action");
        this.notification = useService("notification");
        this.columnas = COLUMNAS;
        onWillStart(() => this.servicio.cargar());
    }

    tareasDe(clave) {
        return this.state.tareas.filter((t) => t.estado === clave);
    }

    async soltar(id, estado) {
        try {
            await this.servicio.mover(id, estado);
        } catch {
            this.notification.add(_t("No se pudo mover la tarea"), { type:
"danger" });
        }
    }

    abrir(id) {
        this.action.doAction(
            { type: "ir.actions.act_window", res_model: "owl.tarea", res_id:
id, views: [[false, "form"]], target: "new" },
            { onClose: () => this.servicio.cargar() }
        );
    }
}

registry.category("actions").add("owl_tablero.tablero", Tablero);

```

```

<t t-name="owl_tablero.Tablero">
    <div class="o_tablero d-flex gap-3 p-3 h-100 overflow-auto">
        <Columna t-foreach="columnas" t-as="columna" t-key="columna.clave"
            columna="columna" tareas="tareasDe(columna.clave)"
            onSoltar.bind="soltar" onAbrir.bind="abrir"/>
    </div>
</t>

```

```

.o_tablero_columna {
  min-width: 260px;
  background-color: var(--gray-100, #f8f9fa);
  transition: background-color 0.15s;

  &.o_tablero_encima {
    background-color: #e7f1fb;
  }
}

.o_tablero_tarjeta {
  cursor: grab;
}

```

Paso 7 · El systray y las pruebas

El systray es el del capítulo 13, leyendo del servicio `owl_tablero` y contando `estado == "en_curso"` del usuario conectado. Al compartir servicio, cuando arrastras una tarjeta a *En curso* el número cambia en el mismo instante.

Para las pruebas, define en Hoot el modelo `owl.tarea` con los campos `estado`, `dificultad` y `user_id`, monta el `Tablero` con `mountWithCleanup` y comprueba al menos estas tres cosas:

1. Cada columna muestra el número correcto de tarjetas según los `_records`.
2. Tras llamar a `soltar(id, "hecha")` y esperar `animationFrame()`, la tarjeta está en la tercera columna.
3. Si el servidor simulado devuelve un error en `write`, la tarjeta vuelve a su columna original.

Prueba

1. Instala `owl_tablero` y abre el menú *Tablero*.
2. Arrastra una tarjeta de *Pendiente* a *En curso*: la columna de destino se ilumina en azul al pasar por encima, y el contador de cada columna cambia al soltar.
3. Recarga la página: la tarjeta sigue en *En curso*.
4. Arrástrala a *Hecha* y abre la lista de tareas del capítulo 7: aparece marcada como hecha.
5. Pulsa una tarjeta: se abre su formulario con las estrellas del capítulo 9.

Resultado en Odoo 20

Ves tres columnas grises con su título y un contador. Las tarjetas blancas muestran el título en negrita, el responsable en gris y la dificultad en estrellas amarillas. Al arrastrar, el cursor cambia a una mano y la columna de destino se pone azul claro.

Reto

Elige una o varias mejoras para terminar el proyecto:

1. Un buscador encima del tablero (capítulo 5) que filtre las tarjetas de todas las columnas.
2. Un filtro *Solo las mías* que guarde la preferencia en `localStorage` para la próxima visita.
3. Una cuarta columna configurable desde *Ajustes* (capítulo 15 del libro anterior).
4. Actualización en tiempo real cuando otro usuario mueva una tarjeta. Pista: estudia cómo el servicio `bus_service` envía notificaciones del servidor al navegador.

Apéndices

Apéndice A · Chuleta de OWL

Esqueleto de un componente

```
import { Component, onWillStart, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { useService } from "@web/core/utils/hooks";
import { _t } from "@web/core/l10n/translation";

export class MiComponente extends Component {
  static template = "mi_modulo.MiComponente";
  static components = {};
  static props = {};

  setup() {
    this.orm = useService("orm");
    this.state = useState({ cargando: true, datos: [] });
    onWillStart(async () => {
      this.state.datos = await this.orm.searchRead("res.partner", [],
["name"], { limit: 10 });
      this.state.cargando = false;
    });
  }
}
```

Directivas de plantilla

Directiva	Ejemplo
Texto	<code><t t-out="valor"/></code>
Condición	<code>t-if</code> , <code>t-elif</code> , <code>t-else</code>
Bucle	<code>t-foreach="lista" t-as="x" t-key="x.id"</code> (índice: <code>x_index</code>)
Atributo	<code>t-att-disabled="!activo"</code> , <code>t-attf-class="btn-{{tipo}}"</code>
Clases condicionales	<code>t-att-class="{ 'active': seleccionado }"</code>
Evento	<code>t-on-click="metodo"</code> , <code>t-on-click.stop.prevent="() => this.m(x)"</code>

Directiva	Ejemplo
Enlace bidireccional	<code>t-model="state.campo"</code>
Referencia al DOM	<code>t-ref="nombre" + useRef("nombre")</code>
Subcomponente	<code><Hijo prop="expr" onAccion.bind="metodo"/></code>
Slots	<code><t t-slot="default"/></code> y <code><t t-set-slot="nombre"></code>
Variable	<code><t t-set="x" t-value="expr"/></code>
Otra plantilla	<code><t t-call="modulo.Plantilla"/></code>
Herencia	<code>t-inherit="modulo.Plantilla" t-inherit-mode="extension"</code>

Hooks

Hook	Origen	Para qué
<code>useState</code>	<code>@odoo/owl</code>	Estado reactivo
<code>useRef</code>	<code>@odoo/owl</code>	Elemento del DOM
<code>useEffect</code>	<code>@odoo/owl</code>	Efecto con dependencias y limpieza
<code>useSubEnv</code> / <code>useEnv</code>	<code>@odoo/owl</code>	Ampliar o leer el entorno
<code>useExternalListener</code>	<code>@odoo/owl</code>	Escuchar <code>window</code> o <code>document</code>
<code>onWillStart</code> , <code>onMounted</code> , <code>onWillUnmount</code> ...	<code>@odoo/owl</code>	Ciclo de vida
<code>useService</code>	<code>@web/core/utils/hooks</code>	Pedir un servicio
<code>useBus</code>	<code>@web/core/utils/hooks</code>	Escuchar un bus
<code>useDebounce</code>	<code>@web/core/utils/timing</code>	Retrasar una función

Registros

```
registry.category("actions").add("tag", Componente);
registry.category("fields").add("widget", { component, supportedTypes,
extractProps });
registry.category("views").add("js_class", { ...listView, Controller });
registry.category("systray").add("clave", { Component }, { sequence: 30 });
registry.category("services").add("nombre", { dependencies: [], start(env,
deps) {} });
registry.category("public.interactions").add("clave", MiInteraction);
```

Apéndice B · Errores frecuentes

Mensaje o síntoma	Causa habitual	Solución
<i>There is no action registered for tag</i>	El tag del XML y el del JS no coinciden, o el JS no se carga	Revisa el tag y la clave <code>assets</code> del manifiesto
<i>Missing template</i>	<code>static template</code> no coincide con ningún <code>t-name</code>	Copia el nombre exacto de la plantilla
<i>Invalid props for component</i>	Falta una prop o tiene otro tipo	Lee el nombre de la prop en el mensaje y revisa <code>static props</code>
<i>Got duplicate key in t-foreach</i>	Dos elementos con la misma <code>t-key</code>	Usa un id único
<i>Cannot read properties of undefined</i> en la plantilla	Un dato aún no cargado	Cárgalo en <code>onWillStart</code> o protege con <code>t-if</code>
La pantalla no se actualiza	El dato no está en <code>useState</code> o se lee sin pasar por él	Guárdalo en el estado (capítulos 3 y 13)
<code>this</code> es <code>undefined</code> en un método	Se pasó el método como prop sin <code>.bind</code>	Usa <code>onAlgo.bind="metodo"</code>
Los cambios no aparecen	Caché de assets	Recarga con <code>?debug=assets</code> , actualiza el módulo o reinicia Odoo
El parche no hace nada	El módulo no depende del que parchea y se carga antes	Añade la dependencia en el manifiesto

Mensaje o síntoma	Causa habitual	Solución
Funciona en el backend pero no en el TPV o el sitio web	Archivo en el paquete de assets equivocado	<code>point_of_sale._assets_pos</code> o <code>web.assets_frontend</code>

Apéndice C · Migración desde versiones anteriores

Antes (Odoo 16 o anterior)	Ahora (Odoo 20)	Desde
<code>odoo.define("modulo.x", function (require) {...})</code>	<code>import / export</code> en archivos de <code>static/src</code>	Odoo 16–17
Cabecera <code>/** @odoo-module */</code> obligatoria	Ya no hace falta en <code>static/src</code>	Odoo 17
Widget de jQuery en el backend	Componentes OWL	Odoo 17
<code>t-esc</code>	<code>t-out</code>	Odoo 17
<code>patch(obj, "nombre", {...})</code> y <code>this._super()</code>	<code>patch(obj, {...})</code> y <code>super.metodo()</code>	Odoo 17
<code>useService("user")</code>	<code>import { user } from "@web/core/user"</code>	Odoo 18
<code>publicWidget.Widget.extend({...})</code>	Clase <code>Interaction</code> en <code>public.interactions</code>	Odoo 18
Pruebas con QUnit	Pruebas con Hoot (<code>*.test.js</code>)	Odoo 18
Plantilla de tarjeta kanban <code>kanban-box</code>	<code>card</code>	Odoo 18
<code><tree></code> en vistas	<code><list></code>	Odoo 18
Rutas <code>type="json"</code>	<code>type="jsonrpc"</code>	Odoo 19
<code>ir.model.access.csv</code>	<code>security/ir.access.csv</code>	Odoo 20 (ver libro anterior)

Cuando migres un módulo, empieza por el manifiesto (`assets`), sigue por los imports y termina por las plantillas. Abre la consola en cada paso: casi todos los errores de migración aparecen ahí con el nombre del archivo.

Fuentes y versiones

Este libro se escribió con la API de OWL 2 que Odoo usa desde la versión 17. Las rutas de import, los nombres de plantillas y los XPath de Odoo pueden cambiar entre versiones menores; compruébalos siempre con `?debug=assets` en tu Odoo 20.

- [Documentación oficial de desarrollo de Odoo](#)
 - [Repositorio de OWL](#)
 - [Código fuente de Odoo](#)
-

Todo el cliente web de Odoo está hecho con OWL. Aprende a construirlo tú.

Este libro continúa *Odoo 20 para Nuevos Programadores* y lleva tus módulos al navegador. Cada capítulo es un módulo real que se construye paso a paso y termina con una prueba, el resultado esperado en Odoo 20 y un reto.

- Componentes, plantillas QWeb, estado reactivo, props y slots
- Servicios `orm`, `action`, `dialog` y `notification`
- Widgets de campo, vistas propias e iconos en la barra superior
- Modificar Odoo sin copiarlo con `patch` y `t-inherit`
- Pruebas con Hoot, Punto de Venta e Interactions del sitio web
- Proyecto final: un tablero de tareas con arrastrar y soltar

Marlon Falcón Hernández

ERP, CRM y Software · Valencia, España



marlonfalcon.com

EDICIÓN PARA ODOO 20.0 · OCTUBRE DE 2026