MANUAL

OWL en Odoo 20

Guía de iniciación a OWL 3 con ejemplos probados

Marlon Falcón Hernández
marlonfalcon.com/cursos/owl-odoo-20
Ejemplos ejecutados en Odoo 20.0-20261015 con OWL 3.0.0-alpha.49

Índice

1. Antes de empezar: Odoo 20 usa OWL 3
2. Qué es OWL y dónde encaja
3. Crear el módulo
4. Tu primer componente
5. Estado reactivo: signal, computed y proxy
6. Props: pasar datos a un componente
7. Eventos
8. Ciclo de vida
9. Plantillas: las directivas
10. Comunicar componentes
11. Servicios y plugins
12. Crear una pantalla: client action
13. Modificar componentes de Odoo con patch
14. Widget de campo propio
15. Icono en la barra superior
16. Pruebas con hoot
17. Proyecto final: lista de tareas
18. Errores comunes
19. Migrar de Odoo 19 a Odoo 20
20. Buenas prácticas

LECCIÓN 1

Antes de empezar: Odoo 20 usa OWL 3

Por qué casi todo lo que encontrarás sobre OWL ya no funciona en Odoo 20.

Odoo 20 trae **OWL 3**, todavía en versión alfa. Hasta Odoo 19 se usaba OWL 2, así que quien viene de la versión anterior da un salto de versión mayor.

> La documentación oficial de Odoo 20 sigue enseñando OWL 2 en sus páginas de frontend: `useState`, `t-esc`, `t-slot`, `static props`. Gran parte de ese código falla en Odoo 20. Este curso sigue el código real de la rama 20.0.

Antes de copiar un ejemplo de un blog, revisa esta tabla. Si ves algo de la columna izquierda, describe OWL 2:

Si ves…	En Odoo 20 se escribe…
`useState({ x: 1 })`	`proxy({ x: 1 })` o `signal(1)`
`static props = {...}`	`props = useProps({...})`
`<t t-esc="x"/>`	`<t t-out="this.x"/>`
`<t t-slot="default"/>`	`<t t-call-slot="default"/>`
`useRef("root")`	`root = signal.ref()`
`t-on-click="metodo"`	`t-on-click="this.metodo"`

Para saber qué versión tienes delante, abre la consola del navegador en tu Odoo y escribe `owl.__info__`. OWL 3 sigue cambiando: la versión exacta depende de la fecha de tu Odoo 20.

Lo que necesitas: un Odoo 20 instalado y saber lo básico de Python y JavaScript. Todos los ejemplos están en dos módulos que puedes descargar e instalar.

Cómo se ha probado: los dos módulos y sus pruebas se han ejecutado en Odoo 20.0-20261015 con OWL 3.0.0-alpha.49, y los mensajes de error de este curso son los que da esa versión. La rama 20.0 actual lleva ya la alpha.50, que no se ha ejecutado.

LECCIÓN 2

Qué es OWL y dónde encaja

Componentes, plantillas y reactividad dentro del cliente web de Odoo.

OWL es el framework de componentes de Odoo. Un componente es una **clase de JavaScript** unida a una **plantilla XML**. Cuando cambian los datos que usa la plantilla, OWL actualiza solo esa parte de la pantalla.

Todo lo que ves en el cliente web, desde la barra superior hasta las vistas de lista y formulario, son componentes de OWL. Así arranca:

- El servidor envía un **bundle** con el JavaScript, el XML y los estilos de todos los módulos instalados.
- Al cargar, Odoo crea la aplicación de OWL y monta el componente `WebClient`.
- Cada menú lanza una **acción**, y el contenedor de acciones pinta el componente que corresponde.

Por eso tu código debe cumplir dos condiciones: estar en un bundle para que llegue al navegador y estar registrado para que Odoo sepa cuándo pintarlo. Las dos siguientes lecciones cubren justo eso.

> Mientras desarrolles, trabaja siempre con `?debug=1` en la URL. Activa el modo dev de OWL, que es el que valida las props y muestra errores claros. Sin él, muchos fallos pasan desapercibidos.

LECCIÓN 3

Crear el módulo

Estructura de carpetas, manifest y bundles de assets.

Un módulo con OWL es un módulo normal de Odoo con una carpeta `static/src`. Los ejemplos de este curso viven en `curso_owl_ejemplos`:

text

```
curso_owl_ejemplos/
├── __init__.py
├── __manifest__.py
├── views/ejemplos_menu.xml
└── static/src/
    ├── hola/hola.js, hola.xml
    ├── contador/contador.js, contador.xml
    └── ...
```

curso_owl_ejemplos/__manifest__.py

```
{
    'name': 'Curso OWL: ejemplos',
    'version': '1.0',
    'category': 'Tutorials',
    'summary': 'Ejemplos cortos de OWL 3 en Odoo 20',
    'author': 'Curso OWL',
    'license': 'LGPL-3',
    'depends': ['web'],
    'data': [
        'views/ejemplos_menu.xml',
    ],
    'assets': {
        'web.assets_backend': [
            'curso_owl_ejemplos/static/src/**/*',
        ],
    },
    'application': True,
    'installable': True,
}
```

La línea importante es `assets`. Añade al bundle `web.assets_backend` todo lo que haya en `static/src`, tanto JavaScript como XML y SCSS. Si un archivo no está en el bundle, el navegador nunca lo recibe.

- No hace falta la cabecera `/** @odoo-module **/`: todo lo que está en `static/src` se transpila igualmente.
- Un archivo `mi_modulo/static/src/a/b.js` se importa como `@mi_modulo/a/b`. Dentro del mismo módulo también valen las rutas relativas.
- Las plantillas van en un `.xml` con raíz `<templates>`, y su nombre debe ser único. Prefíjalo con el nombre del módulo.

Si tu módulo crea modelos, ojo: **`ir.model.access.csv` ya no existe en Odoo 20**. Ahora es `ir.access.csv`, con otras columnas. No es parte de OWL, pero afecta a todos los módulos:

curso_owl_todo/security/ir.access.csv

```
id,name,model_id,group_id/id,operation,domain
access_curso_todo_user,curso.todo.user,curso.todo,base.group_user,crud,
```

Y en las vistas se usa `<list>`, no `<tree>`.

LECCIÓN 4

Tu primer componente

Una clase, una plantilla y un registro.

Tres piezas: la clase, la plantilla y el registro que permite abrirlo desde un menú.

curso_owl_ejemplos/static/src/hola/hola.js

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

export class Hola extends Component {
    static template = "curso_owl_ejemplos.Hola";

    nombre = "Odoo 20";
}

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

curso_owl_ejemplos/static/src/hola/hola.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Hola">
        <div class="p-3">Hola, <t t-out="this.nombre"/></div>
    </t>

</templates>
```

Y en el XML de datos, la acción y su menú. El `tag` debe coincidir con la clave del registro:

xml

```
<record id="action_hola" model="ir.actions.client">
    <field name="name">Hola</field>
    <field name="tag">curso_owl_ejemplos.hola</field>
</record>
<menuitem id="menu_hola" name="Hola" action="action_hola" parent="menu_ejemplos_root"/>
```

- `static template` es el nombre de la plantilla XML.
- Los campos de clase, como `nombre = "Odoo 20"`, son la forma habitual de declarar datos.
- En la plantilla se escribe `this.nombre`. **Sin `this.` no funciona**: es el cambio que más veces vas a tropezar al venir de OWL 2.

Ejercicio

Añade a `Hola` un campo `autor` con tu nombre y muéstralo en la plantilla.

Ver la solución

```
// hola.js
autor = "Tu nombre";

<!-- hola.xml -->
<p>Por <t t-out="this.autor"/></p>
```

LECCIÓN 5

Estado reactivo: signal, computed y proxy

Cómo hacer que la pantalla cambie cuando cambian los datos.

En OWL 3 `useState` ya no existe. Se usan estas piezas:

curso_owl_ejemplos/static/src/contador/contador.js

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

export class Contador extends Component {
    static template = "curso_owl_ejemplos.Contador";

    // signal: se lee llamándola y se escribe con .set()
    cuenta = signal(0);
    // computed: valor derivado, se recalcula solo cuando cambia lo que lee
    doble = computed(() => this.cuenta() * 2);
    // proxy: objeto reactivo profundo, se usa como un objeto normal
    estado = proxy({ historial: [] });

    incrementar() {
        this.cuenta.set(this.cuenta() + 1);
        this.estado.historial.push(this.cuenta());
    }
}

registry.category("actions").add("curso_owl_ejemplos.contador", Contador);
```

curso_owl_ejemplos/static/src/contador/contador.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Contador">
        <div class="p-3">
            <button class="btn btn-primary" t-on-click="this.incrementar">
                Cuenta: <t t-out="this.cuenta()"/>
            </button>
            <p>Doble: <t t-out="this.doble()"/></p>
            <p>Historial: <t t-out="this.estado.historial.join(', ')"/></p>
        </div>
    </t>

</templates>
```

Pieza	Leer	Escribir	Para qué
`signal(v)`	`s()`	`s.set(v)`	Valores sueltos. Imprescindible para `t-model` y `t-ref`
`computed(fn)`	`c()`	No se escribe	Valores derivados de otros
`proxy(obj)`	`p.a`	`p.a = 1`	Objetos y listas que se modifican por dentro

Tres reglas que evitan la mayoría de los fallos:

- **Una signal no ve los cambios internos.** `this.lista().push(4)` no actualiza la pantalla. Asigna una lista nueva con `.set()`, o usa `proxy`.
- **`proxy` no acepta valores simples.** `proxy(5)` falla: solo admite objetos y listas. Para un número, usa `signal`.
- **Si solo calculas un valor, usa `computed`**, no un efecto.

Ejercicio

Añade al contador un botón que lo devuelva a cero y vacíe el historial.

Ver la solución

```
// contador.js
reiniciar() {
    this.cuenta.set(0);
    this.estado.historial = [];
}

<!-- contador.xml -->
<button class="btn btn-secondary" t-on-click="this.reiniciar">Reiniciar</button>
```

LECCIÓN 6

Props: pasar datos a un componente

useProps, tipos y valores por defecto.

Las props son los datos que un componente recibe de su padre. En Odoo 20 hay que declararlas con `useProps`. **Sin esa declaración `this.props` no existe**, y `static props` lanza un error al crear el componente.

curso_owl_ejemplos/static/src/props_demo/props_demo.js

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

export class Saludo extends Component {
    static template = "curso_owl_ejemplos.Saludo";
    props = useProps({
        nombre: t.string(),
        veces: t.number().optional(1),
        etiquetas: t.array(t.string()).optional(() => []),
    });
}

export class PropsDemo extends Component {
    static template = "curso_owl_ejemplos.PropsDemo";
    static components = { Saludo };
}

registry.category("actions").add("curso_owl_ejemplos.props", PropsDemo);
```

curso_owl_ejemplos/static/src/props_demo/props_demo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Saludo">
        <p>
            <t t-out="this.props.nombre"/> (x<t t-out="this.props.veces"/>)
            <span t-if="this.props.etiquetas.length" t-out="this.props.etiquetas.join(', ')"/>
        </p>
    </t>

    <t t-name="curso_owl_ejemplos.PropsDemo">
        <div class="p-3">
            <Saludo nombre="'Ana'"/>
            <Saludo nombre="'Luis'" veces="3" etiquetas="['nuevo']"/>
            <Saludo nombre.translate="Equipo"/>
        </div>
    </t>

</templates>
```

Tipo	Se escribe
Texto obligatorio	`t.string()`
Booleano opcional	`t.boolean().optional()`
Número con valor por defecto	`t.number().optional(100)`
Lista de textos	`t.array(t.string())`
Objeto con forma	`t.object({ id: t.number() })`
Texto o número	`t.or([t.string(), t.number()])`
Función	`t.function()`
Lo que sea	`t.any()`

- Para listas y objetos, el valor por defecto va en una función: `.optional(() => [])`. Así cada instancia tiene el suyo.
- En la llamada, `nombre="'Ana'"` lleva comillas simples dentro porque el valor es una expresión JavaScript.
- El sufijo `.translate` marca un texto como traducible, y `.bind` enlaza una función con el componente.
- La validación solo actúa en modo dev: otro motivo para usar `?debug=1`.
- Las props son de solo lectura para el hijo.

Ejercicio

Añade a `Saludo` una prop opcional `emoji` de tipo texto, con "👋" por defecto, y muéstrala.

Ver la solución

```
// props_demo.js, dentro de useProps
emoji: t.string().optional("👋"),

<!-- props_demo.xml -->
<t t-out="this.props.emoji"/>
```

LECCIÓN 7

Eventos

t-on, argumentos, modificadores y eventos globales.

curso_owl_ejemplos/static/src/eventos/eventos.js

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

export class Eventos extends Component {
    static template = "curso_owl_ejemplos.Eventos";

    mensaje = signal("Sin eventos");
    tecla = signal("");

    setup() {
        // Para objetivos globales se usa useListener: se retira al destruir el componente.
        useListener(window, "keydown", (ev) => this.tecla.set(ev.key));
    }

    alPulsar(ev) {
        this.mensaje.set(`Clic en ${ev.target.tagName}`);
    }

    saludar(nombre) {
        this.mensaje.set(`Hola, ${nombre}`);
    }
}

registry.category("actions").add("curso_owl_ejemplos.eventos", Eventos);
```

curso_owl_ejemplos/static/src/eventos/eventos.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Eventos">
        <div class="p-3">
            <button class="btn btn-primary" t-on-click="this.alPulsar">Método</button>
            <button class="btn btn-secondary" t-on-click="() => this.saludar('Ana')">Con argumento</button>
            <a href="#" t-on-click.prevent.stop="() => this.saludar('enlace')">Con modificadores</a>
            <p t-out="this.mensaje()"/>
            <p>Última tecla: <t t-out="this.tecla()"/></p>
        </div>
    </t>

</templates>
```

- `t-on-click="this.alPulsar"` llama al método con el evento como argumento.
- Para pasar otros argumentos, usa una función flecha: `t-on-click="() => this.saludar('Ana')"`.
- Modificadores: `.stop`, `.prevent`, `.self`, `.capture`, `.synthetic` y `.passive`.
- Para `window` o `document` usa `useListener`, que retira el evento al destruir el componente. Sustituye a `useExternalListener`.

> En `useListener` el manejador **no** se enlaza con el componente. Pásalo como función flecha, como en el ejemplo, o `this` no será lo que esperas.

LECCIÓN 8

Ciclo de vida

Cargar datos, reaccionar al DOM y limpiar.

curso_owl_ejemplos/static/src/ciclo_vida/ciclo_vida.js

```
import {
    Component,
    onError,
    onMounted,
    onPatched,
    onWillDestroy,
    onWillPatch,
    onWillStart,
    onWillUnmount,
    signal,
} from "@odoo/owl";
import { registry } from "@web/core/registry";

export class CicloVida extends Component {
    static template = "curso_owl_ejemplos.CicloVida";

    clics = signal(0);
    raiz = signal.ref();

    setup() {
        onWillStart(async () => {
            // Se espera antes del primer render: aquí se cargan los datos.
            this.dato = await Promise.resolve("cargado");
        });
        onMounted(() => console.log("onMounted", this.raiz()));
        onWillPatch(() => console.log("onWillPatch"));
        onPatched(() => console.log("onPatched", this.clics()));
        onWillUnmount(() => console.log("onWillUnmount"));
        onWillDestroy(() => console.log("onWillDestroy"));
        onError((error) => console.error("onError", error));
    }
}

registry.category("actions").add("curso_owl_ejemplos.ciclo_vida", CicloVida);
```

curso_owl_ejemplos/static/src/ciclo_vida/ciclo_vida.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.CicloVida">
        <div class="p-3" t-ref="this.raiz">
            <p t-out="this.dato"/>
            <button class="btn btn-primary" t-on-click="() => this.clics.set(this.clics() + 1)">
                Clics: <t t-out="this.clics()"/>
            </button>
        </div>
    </t>

</templates>
```

Hook	Cuándo	Para qué
`setup`	Al crear la instancia	Declarar hooks
`onWillStart`	Antes del primer render; puede ser asíncrono	Cargar datos
`onMounted`	Ya en el DOM	Medir, integrar librerías
`onWillUpdateProps`	Antes de aplicar nuevas props	Cargas que dependen de props
`onWillPatch`	Antes de actualizar el DOM	Guardar el scroll

Hook	Cuándo	Para qué
`onPatched`	Tras actualizar el DOM	Reaccionar al DOM nuevo
`onWillUnmount`	Antes de salir del DOM	Limpiar lo creado en `onMounted`
`onWillDestroy`	Siempre, aunque no llegara a montarse	Limpiar lo creado en `setup`
`onError`	Error en un descendiente	Mostrar un estado alternativo

- Los hooks se llaman en `setup` o en un campo de clase, nunca dentro de un callback.
- Un `onWillStart` lento retrasa la pantalla: que sea rápido.
- `onRendered` ya no existe. Usa `onMounted` u `onPatched`.

> **La trampa de `useEffect`:** conserva el nombre, pero ignora en silencio el array de dependencias de OWL 2 y se ejecuta de inmediato durante `setup`, antes de que exista el DOM. Sus dependencias se detectan según lo que lea la función.

LECCIÓN 9

Plantillas: las directivas

Listas, condiciones, atributos, formularios, referencias y slots.

curso_owl_ejemplos/static/src/directivas/directivas.js

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

export class Caja extends Component {
    static template = "curso_owl_ejemplos.Caja";
    props = useProps({ titulo: t.string() });
}

export class Directivas extends Component {
    static template = "curso_owl_ejemplos.Directivas";
    static components = { Caja };

    colores = signal([
        { id: 1, nombre: "rojo" },
        { id: 2, nombre: "verde" },
        { id: 3, nombre: "azul" },
    ]);
    texto = signal("");
    activo = signal(false);
    color = signal("rojo");
    entrada = signal.ref();

    enfocar() {
        this.entrada()?.focus();
    }
}

registry.category("actions").add("curso_owl_ejemplos.directivas", Directivas);
```

curso_owl_ejemplos/static/src/directivas/directivas.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Caja">
        <section class="card p-2 mb-2">
            <h3 t-out="this.props.titulo"/>
            <t t-call-slot="default">Sin contenido</t>
            <footer class="text-muted"><t t-call-slot="pie"/></footer>
        </section>
    </t>

    <t t-name="curso_owl_ejemplos.Directivas">
        <div class="p-3">
            <!-- t-foreach: t-key es obligatorio; item_index e item_first los crea OWL -->
            <ul>
                <t t-foreach="this.colores()" t-as="item" t-key="item.id">
                    <li t-att-class="{ 'fw-bold': item_first }">
                        <t t-out="item_index + 1"/>. <t t-out="item.nombre"/>
                    </li>
                </t>
            </ul>

            <!-- t-if, t-elif, t-else -->
            <p t-if="this.texto().length gt 10">Texto largo</p>
            <p t-elif="this.texto()">Texto corto</p>
            <p t-else="">Sin texto</p>

            <!-- t-att, t-att-*, t-attf-* -->
            <button class="btn" t-attf-class="btn-{{ this.activo() ? 'success' : 'secondary' }}"
                t-att-disabled="!this.activo()" t-att="{ 'data-demo': 'si' }">
```

```
                Botón
            </button>

            <!-- t-model con signal, t-ref con signal.ref() -->
            <input class="o_input" t-model="this.texto" t-ref="this.entrada"/>
            <label><input type="checkbox" t-model="this.activo"/> Activo</label>
            <select class="o_input" t-model="this.color">
                <t t-foreach="this.colores()" t-as="item" t-key="item.id">
                    <option t-att-value="item.nombre" t-out="item.nombre"/>
                </t>
            </select>

            <!-- t-on -->
            <button class="btn btn-link" t-on-click="this.enfocar">Enfocar</button>

            <!-- t-call-slot en Caja, t-set-slot aquí -->
            <Caja titulo="'Con slots'">
                Contenido por defecto
                <t t-set-slot="pie">Color elegido: <t t-out="this.color()"/></t>
            </Caja>
        </div>
    </t>

</templates>
```

Directiva	Para qué	Recuerda
`t-out`	Mostrar un valor	Escapa el HTML salvo que pase por `markup`
`t-foreach` + `t-as` + `t-key`	Repetir	`t-key` es obligatorio y debe ser único
`t-if`, `t-elif`, `t-else`	Condiciones	Usa `gt`, `lt`, `gte`, `lte`, `and`, `or` para que el XML sea válido
`t-att-x`, `t-attf-x`	Atributos dinámicos	Un valor falso elimina el atributo
`t-model`	Enlazar un campo de formulario	Exige una signal. Con `proxy`: `t-model.proxy`
`t-ref`	Acceder a un nodo del DOM	Exige una signal: `signal.ref()`
`t-call-slot`, `t-set-slot`	Contenido que pone el padre	`t-call-slot` sustituye a `t-slot`

Dentro de `t-foreach`, OWL crea variables automáticas a partir del nombre de `t-as`: `item_index`, `item_first`, `item_last`. Las variables locales de plantilla son la excepción: no llevan `this.`.

En `t-model` no se llama a la signal: se escribe `t-model="this.texto"`, sin paréntesis, porque OWL necesita la signal en sí para poder escribir en ella. Y con `proxy` se añade el modificador:

xml
```
<input t-model="this.texto"/>            <!-- signal -->
<input t-model.proxy="this.state.texto"/>  <!-- proxy -->
```

> `t-esc` y `t-slot` aún funcionan en la versión probada, pero avisan por consola: `t-esc has been deprecated` y `t-slot has been renamed t-call-slot`. No los uses: el código de la alpha.50 ya no reconoce `t-slot`.

LECCIÓN 10

Comunicar componentes

De padre a hijo, de hijo a padre y a distancia.

curso_owl_ejemplos/static/src/comunicacion/comunicacion.js

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

export class Hijo extends Component {
    static template = "curso_owl_ejemplos.Hijo";
    props = useProps({
        valor: t.number(),
        onSumar: t.function(),
        // Una signal como prop: el hijo la lee sin que el padre se vuelva a renderizar.
        total: t.signal(t.number()),
    });
}

export class Comunicacion extends Component {
    static template = "curso_owl_ejemplos.Comunicacion";
    static components = { Hijo };

    a = signal(1);
    total = signal(1);

    sumar(cantidad) {
        this.a.set(this.a() + cantidad);
        this.total.set(this.total() + cantidad);
    }
}

registry.category("actions").add("curso_owl_ejemplos.comunicacion", Comunicacion);
```

curso_owl_ejemplos/static/src/comunicacion/comunicacion.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Hijo">
        <div>
            <p>Valor recibido: <t t-out="this.props.valor"/></p>
            <p>Total (signal): <t t-out="this.props.total()"/></p>
            <button class="btn btn-primary" t-on-click="() => this.props.onSumar(1)">Sumar 1</button>
        </div>
    </t>

    <t t-name="curso_owl_ejemplos.Comunicacion">
        <div class="p-3">
            <Hijo valor="this.a()" total="this.total" onSumar.bind="this.sumar"/>
        </div>
    </t>

</templates>
```

Necesidad	Técnica
Del padre al hijo	Props
Del hijo al padre	Una función como prop, con `.bind`
Valor compartido que cambia a menudo	Una signal como prop
Estado para partes lejanas	Un plugin (lección siguiente)

Todo hijo debe aparecer en `static components`. Si no, verás `Cannot find the definition of component`.

LECCIÓN 11

Servicios y plugins

ORM, notificaciones, diálogos, acciones y estado compartido.

Los servicios principales de Odoo pasaron a ser **plugins** de OWL. Hay dos formas de acceder a ellos y las dos funcionan en Odoo 20: `useService("nombre")`, la más extendida, y `usePlugin(Clase)`, hacia donde va el framework.

curso_owl_ejemplos/static/src/servicios/servicios.js

```
import { Component, signal, usePlugin } from "@odoo/owl";
import { ConfirmationDialog } from "@web/core/confirmation_dialog/confirmation_dialog";
import { _t } from "@web/core/l10n/translation";
import { rpc } from "@web/core/network/rpc";
import { ORM } from "@web/core/orm_plugin";
import { registry } from "@web/core/registry";
import { user } from "@web/core/user";
import { useService } from "@web/core/utils/hooks";

export class Servicios extends Component {
    static template = "curso_owl_ejemplos.Servicios";

    // Forma nueva: importar la clase del plugin.
    orm = usePlugin(ORM);
    version = signal("");

    setup() {
        // Forma puente: useService sigue funcionando con estos nombres.
        this.action = useService("action");
        this.dialog = useService("dialog");
        this.notification = useService("notification");
        this.nombreUsuario = user.name;
    }

    async contarUsuarios() {
        const total = await this.orm.searchCount("res.users", []);
        this.notification.add(_t("Usuarios: %(total)s", { total }), { type: "info" });
    }

    async cargarVersion() {
        const info = await rpc("/web/webclient/version_info");
        this.version.set(info.server_version);
    }

    abrirUsuarios() {
        return this.action.doAction({
            type: "ir.actions.act_window",
            res_model: "res.users",
            views: [[false, "list"]],
        });
    }

    confirmar() {
        this.dialog.add(ConfirmationDialog, {
            title: _t("Confirmar"),
            body: _t("¿Mostrar una notificación?"),
            confirm: () => this.notification.add(_t("Confirmado"), { type: "success" }),
            cancel: () => {},
        });
    }
}

registry.category("actions").add("curso_owl_ejemplos.servicios", Servicios);
```

curso_owl_ejemplos/static/src/servicios/servicios.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_ejemplos.Servicios">
        <div class="p-3 d-flex flex-column gap-2 align-items-start">
            <p>Usuario: <t t-out="this.nombreUsuario"/></p>
            <button class="btn btn-primary" t-on-click="this.contarUsuarios">orm + notification</button>
            <button class="btn btn-primary" t-on-click="this.cargarVersion">rpc</button>
            <button class="btn btn-primary" t-on-click="this.abrirUsuarios">action</button>
            <button class="btn btn-primary" t-on-click="this.confirmar">dialog</button>
            <p t-if="this.version()">Versión: <t t-out="this.version()"/></p>
        </div>
    </t>

</templates>
```

Necesidad	Con useService	Con plugin	Importar desde
ORM	`useService("orm")`	`usePlugin(ORM)`	`@web/core/orm_plugin`
Notificación	`useService("notification")`	`usePlugin(NotificationPlugin)`	`@web/core/notifications/notification_plugin`
Diálogo	`useService("dialog")`	`usePlugin(DialogPlugin)`	`@web/core/dialog/dialog_plugin`
Acciones	`useService("action")`	`usePlugin(ActionPlugin)`	`@web/webclient/actions/action_plugin`

> No hay servicio `rpc` ni `user`: `useService("rpc")` falla. Son una función y un objeto que se importan: `import { rpc } from "@web/core/network/rpc"` e `import { user } from "@web/core/user"`.

Para compartir estado entre componentes lejanos, crea tu propio plugin. Este es el del proyecto final, que comparten la pantalla de tareas y el icono de la barra superior:

curso_owl_todo/static/src/todo_plugin.js

```
import { computed, Plugin, signal, usePlugin } from "@odoo/owl";
import { ORM } from "@web/core/orm_plugin";
import { services } from "@web/core/services";

const MODEL = "curso.todo";
const FIELDS = ["name", "done", "priority"];

/**
 * Estado compartido entre la client action y el systray.
 * Cada escritura vuelve a leer del servidor: el servidor es la fuente de verdad.
 */
export class TodoPlugin extends Plugin {
    orm = usePlugin(ORM);

    todos = signal([]);
    pendingCount = computed(() => this.todos().filter((todo) => !todo.done).length);
    doneCount = computed(() => this.todos().length - this.pendingCount());

    async load() {
        this.todos.set(await this.orm.searchRead(MODEL, [], FIELDS));
    }

    async add(name) {
        await this.orm.create(MODEL, [{ name }]);
        await this.load();
    }

    async toggle(todo) {
        await this.orm.write(MODEL, [todo.id], { done: !todo.done });
        await this.load();
    }
```

```
    async remove(todo) {
        await this.orm.unlink(MODEL, [todo.id]);
        await this.load();
    }
}

services.add(TodoPlugin);
```

LECCIÓN 12

Crear una pantalla: client action

Un componente, su registro y un menú.

Ya has hecho varias sin saberlo. Una client action necesita tres piezas: el componente, su registro en `actions` y un `ir.actions.client` cuyo `tag` coincida. La forma completa declara además sus props estándar:

```js
import { Component, useProps } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { standardActionServiceProps } from "@web/webclient/actions/action_plugin";

export class MiAccion extends Component {
    static template = "mi_modulo.MiAccion";
    props = useProps(standardActionServiceProps);
}

registry.category("actions").add("mi_modulo.mi_accion", MiAccion);
```

La acción recibe sus datos en `this.props.action`, incluidos `params` y `context`.

> La ruta `@web/webclient/actions/action_service` ya no existe. Ahora es `@web/webclient/actions/action_plugin`.

LECCIÓN 13

Modificar componentes de Odoo con patch

Cambiar el comportamiento sin tocar el código original.

`patch` modifica un objeto en su sitio. Dentro de un parche funciona `super`, y en `setup` puedes llamar a hooks. Este parche avisa al guardar una tarea:

curso_owl_todo/static/src/patches/form_controller_patch.js

```
import { _t } from "@web/core/l10n/translation";
import { useService } from "@web/core/utils/hooks";
import { patch } from "@web/core/utils/patch";
import { FormController } from "@web/views/form/form_controller";

patch(FormController.prototype, {
    setup() {
        super.setup(...arguments);
        // Nombre propio para no pisar atributos del componente original.
        this.cursoNotification = useService("notification");
    },

    async onRecordSaved(record, changes) {
        const result = await super.onRecordSaved(...arguments);
        if (record.resModel === "curso.todo") {
            this.cursoNotification.add(_t("Tarea guardada."), { type: "success" });
        }
        return result;
    },
});
```

- Parchea `Clase.prototype` para métodos de instancia, y `Clase` para estáticos.
- Pon nombres propios a lo que añadas, para no pisar atributos del original.
- Las plantillas se extienden con `t-inherit` y `t-inherit-mode="extension"`. Lo que añadas también lleva `this.`.

> **Los campos de clase no se pueden parchear.** `props = useProps(...)` o `x = signal(...)` se crean en cada instancia, no en el prototipo. Para añadir una prop, llama a `useProps` con otro nombre dentro del `setup` parcheado, o amplía el esquema si el componente lo exporta.

LECCIÓN 14

Widget de campo propio

Cambiar cómo se muestra y se edita un campo.

Un widget es un objeto con `component` en el registro `fields`. Este muestra un entero como estrellas:

curso_owl_todo/static/src/todo_stars_field/todo_stars_field.js

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

export class TodoStarsField extends Component {
    static template = "curso_owl_todo.TodoStarsField";
    props = useProps({
        ...standardFieldProps,
        maxStars: t.number().optional(3),
    });

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

    get stars() {
        return Array.from({ length: this.props.maxStars }, (_, index) => index + 1);
    }

    onStarClick(star) {
        const value = star === this.value ? 0 : star;
        return this.props.record.update({ [this.props.name]: value });
    }
}

export const todoStarsField = {
    component: TodoStarsField,
    displayName: _t("Estrellas de tarea"),
    supportedTypes: ["integer"],
    supportedOptions: [
        {
            label: _t("Máximo de estrellas"),
            name: "max_stars",
            type: "number",
        },
    ],
    extractProps({ options }) {
        return { maxStars: options.max_stars };
    },
};

registry.category("fields").add("curso_stars", todoStarsField);
```

curso_owl_todo/static/src/todo_stars_field/todo_stars_field.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_todo.TodoStarsField">
        <div class="o_curso_stars d-flex gap-1">
            <t t-foreach="this.stars" t-as="star" t-key="star">
                <span t-if="this.props.readonly"
                    class="oi"
                    data-icon="star"
                    t-att-class="{ 'oi-filled': star lte this.value }"/>
                <button t-else=""
                    type="button"
                    class="o_curso_star btn btn-link btn-link-inline p-0 oi"
```

```
                    data-icon="star"
                    t-att-class="{ 'oi-filled': star lte this.value }"
                    t-att-aria-label="star"
                    t-on-click.stop="() => this.onStarClick(star)"/>
            </t>
        </div>
    </t>

</templates>
```

Se usa en cualquier vista, y `options` llega a `extractProps`:

xml

```
<field name="priority" widget="curso_stars" options="{'max_stars': 5}"/>
```

- El valor se lee con `this.props.record.data[this.props.name]`.
- Se escribe con `this.props.record.update({ ... })`.
- `readonly` llega solo.
- Los iconos usan `<i class="oi" data-icon="star"/>`: el núcleo de Odoo 20 ya no usa las clases `fa`.

LECCIÓN 15

Icono en la barra superior

Un elemento de systray con contador en vivo.

curso_owl_todo/static/src/todo_systray/todo_systray.js

```
import { Component, onMounted, usePlugin } from "@odoo/owl";
import { registry } from "@web/core/registry";
import { useService } from "@web/core/utils/hooks";
import { TodoPlugin } from "../todo_plugin";

export class TodoSystray extends Component {
    static template = "curso_owl_todo.TodoSystray";

    todoStore = usePlugin(TodoPlugin);

    setup() {
        this.action = useService("action");
        // onMounted y no onWillStart: no retrasamos el render de la barra superior.
        onMounted(() => this.todoStore.load());
    }

    openTodos() {
        return this.action.doAction("curso_owl_todo.todo_action");
    }
}

export const todoSystrayItem = {
    Component: TodoSystray,
};

registry.category("systray").add("curso_owl_todo.todo_systray", todoSystrayItem, { sequence: 50 });
```

curso_owl_todo/static/src/todo_systray/todo_systray.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_todo.TodoSystray">
        <button class="o_curso_todo_systray o_nav_entry border-0"
            title="Tareas pendientes"
            aria-label="Tareas pendientes"
            t-on-click="this.openTodos">
            <i class="oi" data-icon="check" aria-hidden="true"/>
            <span t-if="this.todoStore.pendingCount()"
                class="badge rounded-pill ms-1"
                t-out="this.todoStore.pendingCount()"/>
        </button>
    </t>

</templates>
```

Se carga en `onMounted` y no en `onWillStart` para no retrasar la barra superior. Como el contador viene del plugin, se actualiza solo cuando creas o completas una tarea en la otra pantalla.

LECCIÓN 16

Pruebas con hoot

Probar tus componentes en el navegador.

Odoo 20 usa **hoot** para probar JavaScript. Los archivos van en `static/tests/`, su nombre debe terminar en `.test.js` y se añaden al bundle `web.assets_unit_tests`, como en el manifest del proyecto final. Se ejecutan abriendo `/web/tests`. Para lanzar solo las tuyas, añade el filtro con comillas, que busca el texto exacto: `/web/tests?filter="curso_owl_todo"`. Sin comillas la búsqueda es aproximada y coinciden muchas pruebas de otros módulos.

curso_owl_todo/static/tests/todo_action.test.js

```
import { expect, test } from "@odoo/hoot";
import { waitFor } from "@odoo/hoot-dom";
import {
    contains,
    defineModels,
    fields,
    models,
    mountWithCleanup,
} from "@web/../tests/web_test_helpers";
import { TodoAction } from "@curso_owl_todo/todo_action/todo_action";

class CursoTodo extends models.Model {
    _name = "curso.todo";

    name = fields.Char();
    done = fields.Boolean({ default: false });
    priority = fields.Integer({ default: 0 });

    _records = [
        { id: 1, name: "Leer las notas", done: false, priority: 1 },
        { id: 2, name: "Hacer el ejercicio", done: true, priority: 0 },
    ];
}

defineModels([CursoTodo]);

test("muestra las tareas del servidor", async () => {
    await mountWithCleanup(TodoAction, { props: { action: {} } });

    expect(".o_curso_todo_item").toHaveCount(2);
});

test("filtra las tareas pendientes", async () => {
    await mountWithCleanup(TodoAction, { props: { action: {} } });

    await contains(".o_curso_todo_filter").select("pending");

    expect(".o_curso_todo_item").toHaveCount(1);
});

test("añade una tarea con el botón", async () => {
    await mountWithCleanup(TodoAction, { props: { action: {} } });

    // confirm: false evita que edit() pulse Enter, que ya crearía la tarea.
    await contains(".o_curso_todo_input").edit("Nueva tarea", { confirm: false });
    await contains(".o_curso_todo_add").click();
    await waitFor(".o_curso_todo_item:contains(Nueva tarea)");

    expect(".o_curso_todo_item").toHaveCount(3);
});

test("añade una tarea con Enter", async () => {
    await mountWithCleanup(TodoAction, { props: { action: {} } });
```

```
    await contains(".o_curso_todo_input").edit("Con Enter");
    await waitFor(".o_curso_todo_item:contains(Con Enter)");

    expect(".o_curso_todo_item").toHaveCount(3);
});
```

> En hoot, `edit()` pulsa Enter por defecto. Si tu campo crea algo al pulsar Enter, usa `confirm: false` o la acción se ejecutará dos veces.

LECCIÓN 17

Proyecto final: lista de tareas

Todo lo anterior en un módulo completo.

El módulo `curso_owl_todo` reúne el curso: modelo, seguridad, una client action, componentes hijos, un plugin compartido, un icono en la barra superior, un widget de campo, un parche y pruebas. Descárgalo, cópialo en tu carpeta de addons e instálalo.

Esta es la pantalla principal. Fíjate en cómo usa cada pieza del curso:

curso_owl_todo/static/src/todo_action/todo_action.js

```
import { Component, computed, onWillStart, signal, usePlugin, useProps } from "@odoo/owl";
import { ConfirmationDialog } from "@web/core/confirmation_dialog/confirmation_dialog";
import { _t } from "@web/core/l10n/translation";
import { registry } from "@web/core/registry";
import { useAutofocus, useService } from "@web/core/utils/hooks";
import { standardActionServiceProps } from "@web/webclient/actions/action_plugin";
import { StatCard } from "../stat_card/stat_card";
import { TodoItem } from "../todo_item/todo_item";
import { TodoPlugin } from "../todo_plugin";

export class TodoAction extends Component {
    static template = "curso_owl_todo.TodoAction";
    static components = { StatCard, TodoItem };

    props = useProps(standardActionServiceProps);
    todoStore = usePlugin(TodoPlugin);

    newTodoName = signal("");
    filter = signal("all");
    inputRef = signal.ref();

    visibleTodos = computed(() => {
        const todos = this.todoStore.todos();
        switch (this.filter()) {
            case "pending":
                return todos.filter((todo) => !todo.done);
            case "done":
                return todos.filter((todo) => todo.done);
            default:
                return todos;
        }
    });

    setup() {
        this.action = useService("action");
        this.dialog = useService("dialog");
        this.notification = useService("notification");
        useAutofocus({ ref: this.inputRef });
        onWillStart(() => this.todoStore.load());
    }

    async addTodo() {
        const name = this.newTodoName().trim();
        if (!name) {
            this.notification.add(_t("Escribe un nombre para la tarea."), { type: "warning" });
            return;
        }
        await this.todoStore.add(name);
        this.newTodoName.set("");
        this.notification.add(_t("Tarea creada."), { type: "success" });
    }

    onInputKeydown(ev) {
        if (ev.key === "Enter") {
            this.addTodo();
```

```
        }
    }

    toggleTodo(todo) {
        return this.todoStore.toggle(todo);
    }

    openTodo(todo) {
        return this.action.doAction({
            type: "ir.actions.act_window",
            res_model: "curso.todo",
            res_id: todo.id,
            views: [[false, "form"]],
        });
    }

    deleteTodo(todo) {
        this.dialog.add(ConfirmationDialog, {
            title: _t("Eliminar tarea"),
            body: _t("Esta acción no se puede deshacer."),
            confirmLabel: _t("Eliminar"),
            confirm: () => this.todoStore.remove(todo),
            cancel: () => {},
        });
    }
}

registry.category("actions").add("curso_owl_todo.todo_action", TodoAction);
```

curso_owl_todo/static/src/todo_action/todo_action.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xml:space="preserve">

    <t t-name="curso_owl_todo.TodoAction">
        <div class="o_curso_todo h-100 overflow-auto p-3" t-att-class="this.props.className">
            <h1>Mis tareas</h1>

            <div class="d-flex gap-2 mb-3">
                <StatCard title.translate="Pendientes">
                    <t t-out="this.todoStore.pendingCount()"/>
                </StatCard>
                <StatCard title.translate="Hechas">
                    <t t-out="this.todoStore.doneCount()"/>
                </StatCard>
            </div>

            <div class="d-flex gap-2 mb-3">
                <input class="o_curso_todo_input o_input"
                    placeholder="Nueva tarea"
                    t-ref="this.inputRef"
                    t-model="this.newTodoName"
                    t-on-keydown="this.onInputKeydown"/>
                <button class="o_curso_todo_add btn btn-primary" t-on-click="this.addTodo">
                    Añadir
                </button>
                <select class="o_curso_todo_filter o_input w-auto" t-model="this.filter">
                    <option value="all">Todas</option>
                    <option value="pending">Pendientes</option>
                    <option value="done">Hechas</option>
                </select>
            </div>

            <p t-if="!this.visibleTodos().length" class="o_curso_todo_empty text-muted">
                No hay tareas.
            </p>
            <ul t-else="" class="list-group">
                <t t-foreach="this.visibleTodos()" t-as="todo" t-key="todo.id">
                    <TodoItem
```

```
                        todo="todo"
                        onToggle.bind="this.toggleTodo"
                        onOpen.bind="this.openTodo"
                        onDelete.bind="this.deleteTodo"/>
                </t>
            </ul>
        </div>
    </t>

</templates>
```

- **Datos:** el plugin los carga con `onWillStart` y los guarda en una signal.
- **Filtro:** `visibleTodos` es un `computed` que depende de las tareas y del filtro.
- **Formulario:** `t-model` con signals.
- **Hijos:** `TodoItem` recibe datos y funciones por props, y `StatCard` usa un slot.

LECCIÓN 18

Errores comunes

El mensaje que ves, qué significa y cómo arreglarlo.

Error	Causa	Solución
`Component "X" defines a static "props"`	Código de OWL 2	`props = useProps({...})`
`useState is not a function`	API eliminada	`proxy` o `signal`
`Cannot read properties of undefined` en plantilla	Falta `this.` o falta `useProps`	Añadirlos
`Invalid handler expression` al hacer clic	`t-on-click="metodo"` sin `this.`	`t-on-click="this.metodo"`
`Ref is undefined or null`	`t-ref="nombre"`	`t-ref="this.nombre"` con `signal.ref()`
`Invalid t-model expression`	`t-model` sobre un valor de `proxy`	Una signal, o `t-model.proxy`
`t-slot has been renamed t-call-slot` en consola	Plantilla antigua	`<t t-call-slot="x"/>`
`t-esc has been deprecated` en consola	Plantilla antigua	`t-out`
`Directive t-foreach should always be used with a t-key!`	Falta `t-key`	Añadir una clave única
`Missing template: "x"`	El XML no está en el bundle o el nombre es otro	Revisar `assets` y `t-name`
`Cannot find the definition of component "X"`	Falta en `static components`	Añadirlo
`Service rpc is not available`	Ya no es un servicio	Importar `rpc`
`Unknown plugin "X"`	Falta `services.add`	Registrar el plugin
`No active scope`	Hook fuera de `setup`	Moverlo
`Invalid component props (X)`	Props incorrectas	Corregir la llamada
`Maximum render iterations ... exceeded`	Se escribe estado durante el render	Mover la escritura a un hook
Error de acceso al instalar	Falta `ir.access.csv` o sigue el formato antiguo	Lección 3

Para depurar:

- `?debug=1` activa la validación, y `?debug=assets` carga el código sin minificar.
- `t-log` y `t-debug` en una plantilla sirven para inspeccionar valores.
- `owl.__info__` en la consola muestra la versión de OWL.
- Las Owl devtools son una extensión para Chrome y Firefox.

LECCIÓN 19

Migrar de Odoo 19 a Odoo 20

Tabla de cambios para actualizar tu código.

Odoo 19 (OWL 2)	Odoo 20 (OWL 3)
`useState(obj)`	`proxy(obj)` o `signal(v)`
`reactive(obj)`	`proxy(obj)`
`static props`, `static defaultProps`	`props = useProps({...})`
`{ type: String, optional: true }`	`t.string().optional()`
`props.x` en plantilla	`this.props.x`
`t-on-click="fn"`	`t-on-click="this.fn"`
`t-esc`	`t-out`
`t-slot`	`t-call-slot`
`useRef("x")` + `t-ref="x"`	`x = signal.ref()` + `t-ref="this.x"`
`t-model="state.x"`	`t-model="this.x"` o `t-model.proxy="this.state.x"`
`useExternalListener(...)`	`useListener(...)`
`useEffect(fn, () => [deps])`	`useEffect(fn)` o `useLayoutEffect` de `@web/owl2/utils`
`onRendered`, `useChildSubEnv`	Eliminados
`env.isSmall`	`useService("ui").isSmall`
`@web/webclient/actions/action_service`	`@web/webclient/actions/action_plugin`
`ir.model.access.csv`	`ir.access.csv`
`<tree>`	`<list>`

> Odoo incluye una capa de compatibilidad en `@web/owl2/utils`, pero su propia cabecera dice que es temporal. Úsala para migrar por partes, no para código nuevo.

LECCIÓN 20

Buenas prácticas

Hábitos para que tu código dure.

- **Prefija** plantillas, tags y claves con el nombre del módulo.
- **Declara las props con esquema** y prueba con `?debug=1`.
- **Escribe siempre `this.`** para lo que pertenece al componente.
- **Elige bien el estado:** `signal` para valores sueltos, `proxy` para objetos, `computed` para derivados.
- **Usa `computed` y no un efecto** para calcular valores.
- **Mantén rápidos los `onWillStart`.**
- **Limpia** con `onWillDestroy` lo creado en `setup`, y con `onWillUnmount` lo creado en `onMounted`.
- **Traduce:** `_t` en JavaScript y `.translate` en props.
- **Usa `markup` solo con HTML de confianza.**
- **Escribe pruebas** para los componentes con lógica.
- **Desconfía del código antiguo:** si ves `useState` o `t-esc`, es OWL 2.