Odoo Days Italia

Titolo	OWL & ODOO: divertiamoci con il gufo
Quando	giovedì 04/11/2021
Autore	Matteo Piciucchi
In altre parole:	Programmo con gli animali
Riferimenti	matteo.piciucchi@bloomup.it

Sempre più spesso i clienti ci chiedono applicativi avanzati all'interno di Odoo o modifiche grafiche e funzionali sulle viste standard.

Fino all'introduzione di OWL la complessità e il tempo di sviluppo di un Widget / Client Action erano eccessivamente elevati e poco funzionali, con risultati poco soddisfacenti o limitati dalla GUI di Odoo.

Inizialmente, per soddisfare alcune richieste, abbiamo optato per caricare un'applicazione in vuejs, react, flutter all'interno di un elemento grafico del backend di Odoo oppure abbiamo creato applicazioni esterne.

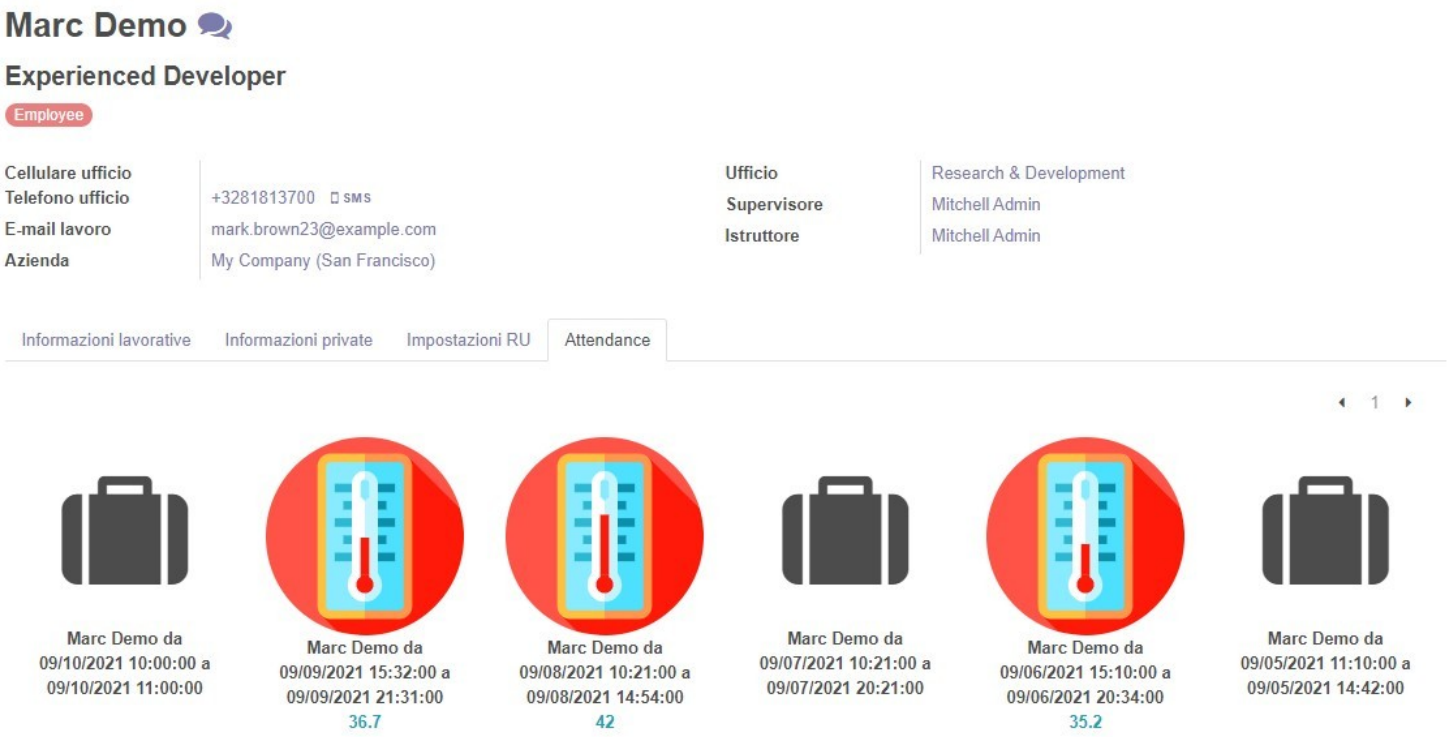
Queste soluzioni davano al cliente una sensazione di "non continuità" tra Odoo e l'app stessa.

In questo specifico caso la richiesta è stata quella di avere all'interno di un field many2many/one2many tanti widget differenti.

Ogni widget doveva rappresentare un valore del record in modo diverso dall'altro in base ad un selettore che ne discriminava il tipo.

In questo specifico caso la richiesta è stata quella di avere all'interno di un field many2many/one2many tanti widget differenti.

Ogni widget doveva rappresentare un valore del record in modo diverso dall'altro in base ad un selettore che ne discriminava il tipo.



Bloomup M2M Multiwidget è il modulo che abbiamo sviluppato. Può essere utilizzato come una libreria per modificare a proprio piacere ogni campo many2many/one2many.

Come si usa?

Come si usa?

1 - Si crea il modulo Javascript in cui vanno dichiarati i vari componenti owl che rappresentano i singoli widget.

```
odoo.define('bloomup_m2m_widget_demo.example', function (require) {
    "use strict";
    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const DefaultWidget = require('bloomup_m2m_multiwidget.widgets');


    DefaultWidget['Thermometer']= class Thermometer extends Component{
        static template = 'ThermometerSvg';
        constructor(...args){
            super(...args);
            this.record = {};
            this.widget_record_bootstrap_class = '';
        }
    }
    DefaultWidget['BusinessTrip']= class BusinessTrip extends Component{
        static template = 'BusinessTrip';
        constructor(...args){
            super(...args);
            this.record = {};
            this.widget_record_bootstrap_class = '';
        }
    }


    return DefaultWidget;
});
```

Come si usa?

2 – Si definiscono i template dei componenti.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<templates>
    <t t-name="BusinessTrip" owl="1">
        <div t-att-class="widget_record_bootstrap_class">
            <b style="font-size:97px"><i class="fa fa-suitcase"></i></b>
            <br/> <b><t t-esc="record.display_name"/></b>
        </div>
    </t>

    <t t-name="ThermometerSvg" owl="1">
        <div class="col-md-2 col-4 text-center position-relative" style="float:left">
            <svg···
            </svg>
            <br/>
            <b><t t-esc="record.display_name"/></b>
            <br/>
            <b class="text-info"><t t-esc="record.temperature"/></b>
        </div>
    </t>
</templates>
```

Come si usa?

3 – Si aggiunge il campo many2many/one2many alla vista con alcuni attributi.

```
<field name="attendance_ids"
    widget="m2m_multiwidget"
    multiwidget_limit="6"
    multiwidget_record_bootstrap_class="
    col-md-2 col-4 text-center float-left"
    multiwidget_list_fields="
    ['id','display_name','temperature','widget', 'temperature_widget']"
    multiwidget_field="widget"
    multiwidget_name_list="{
        'thermometer': 'Thermometer',
        'business_trip': 'BusinessTrip'
    }"
/>
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){…
        }
        async willStart(){…
        }
        async get_records(){…
        }
        async mounted(){…
        }
        appendRecords(){…
        }
        async next_page(){…
        }
        async prev_page(){…
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){···
        }
        async willStart(){···
        }
        async get_records(){···
        }
        async mounted(){···
        }
        appendRecords(){···
        }
        async next_page(){···
        }
        async prev_page(){···
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

Nel costruttore vengono presi gli attributi inseriti nel campo della vista.

```
if(self.attrs.multiwidget_limit){
    self.widget_limit = self.attrs.multiwidget_limit;
}
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){…
        }
        async willStart(){…
        }
        async get_records(){…
        }
        async mounted(){…
        }
        appendRecords(){…
        }
        async next_page(){…
        }
        async prev_page(){…
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

Nel costruttore vengono presi gli attributi inseriti nel campo della vista.

```
if(self.attrs.multiwidget_limit){
    self.widget_limit = self.attrs.multiwidget_limit;
}
```

Si crea lo state in cui salviamo la pagina e i records.

```
self.state = useState({
    'records': [],
    'page': 1,
});
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){…
        }
        async willStart(){…
        }
        async get_records(){…
        }
        async mounted(){…
        }
        appendRecords(){…
        }
        async next_page(){…
        }
        async prev_page(){…
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

Nel costruttore vengono presi gli attributi inseriti nel campo della vista.

```
if(self.attrs.multiwidget_limit){
    self.widget_limit = self.attrs.multiwidget_limit;
}
```

Si crea lo state in cui salviamo la pagina e i records.

```
self.state = useState({
    'records': [],
    'page': 1,
});
```

Si carica il modulo in cui sono presenti i componenti owl.

```
self.multiwidget_modules = require('bloomup_m2m_multiwidget.widgets');
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){…
        }
        async willStart(){…
        }
        async get_records(){…
        }
        async mounted(){…
        }
        appendRecords(){…
        }
        async next_page(){…
        }
        async prev_page(){…
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

Nel costruttore vengono presi gli attributi inseriti nel campo della vista.

```
if(self.attrs.multiwidget_limit){
    self.widget_limit = self.attrs.multiwidget_limit;
}
```

Si crea lo state in cui salviamo la pagina e i records.

```
self.state = useState({
    'records': [],
    'page': 1,
});
```

Si carica il modulo in cui sono presenti i componenti owl.

```
self.multiwidget_modules = require('bloomup_m2m_multiwidget.widgets');
```

In ultimo si crea un evento personalizzato per il refresh dei records e si mette in ascolto.

```
const bus = new owl.core.EventBus();
self.env.bus = bus;
self.env.bus.on("get_records", null, function () {
    self.get_records();
});
```

Come funziona?

```
odoo.define('bloomup_m2m_multiwidget.multiwidget', function (require) {
    "use strict";

    require('web.dom_ready');
    const { Component, useState, mount } = owl;
    const AbstractField = require('web.AbstractFieldOwl');
    const fieldRegistry = require('web.field_registry_owl');
    var session = require('web.session');
    var rpc = require('web.rpc');
    var core = require('web.core');
    var _t = core._t;

    class M2mMultiwidget extends AbstractField {
        static supportedFieldTypes = ['many2many', 'one2many'];
        static template = 'M2mMultiwidget';
        constructor(...args){ ···
        }
        async willStart(){ ···
        }
        async get_records(){ ···
        }
        async mounted(){ ···
        }
        appendRecords(){ ···
        }
        async next_page(){ ···
        }
        async prev_page(){ ···
        }
    }

    fieldRegistry.add('m2m_multiwidget', M2mMultiwidget);
});
```

get_records è la funzione che fa la query al db per prendere i dati dei singoli record selezionati in quella pagina.

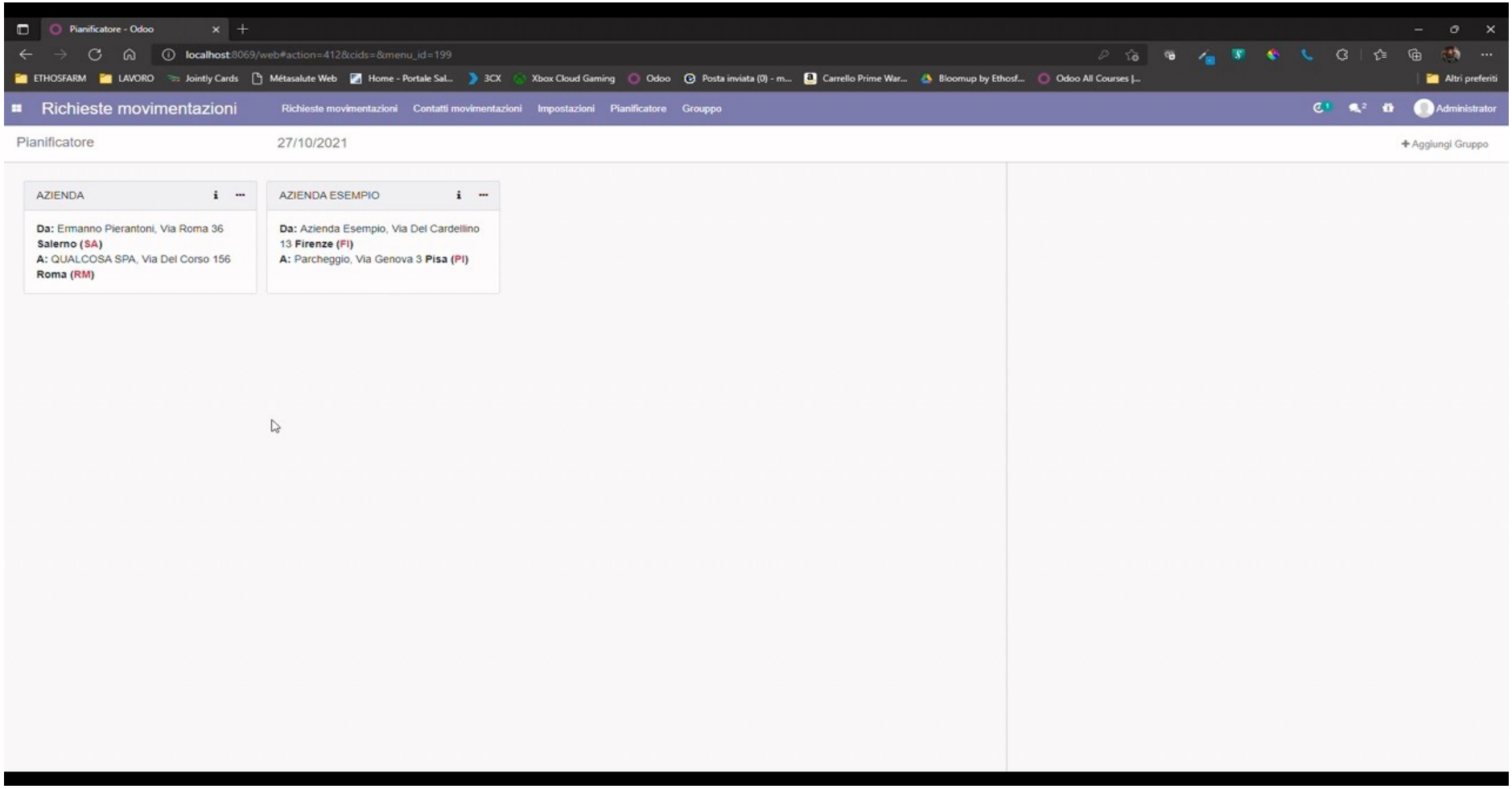
```
async willStart(){
    var self = this;
    if(!self.error){
        await self.get_records();
    }
}
async get_records(){
    var self=this;
    self.state.records = [];
    var limit = parseInt(self.widget_offset) + parseInt(self.widget_limit);
    var ids = self.value.res_ids.slice(self.widget_offset, limit);
    await rpc.query({
        model: self.field.relation,
        method: 'read',
        args: [
            ids,
            self.widget_list_fields
        ],
    }).then(function(res){
        self.state.records = res;
    });
}
```

Come funziona?

appendRecords è la funzione principale in cui vengono renderizzati i record con il componente corrispondente.

```
appendRecords(){
    var self=this;
    $.each(self.state.records, function(k, elem){
        var obj = self.widget_name_list[elem[self.widget_field]];
        if(!obj){
            obj = 'DefaultM2MWidget';
        }
        var widget = eval('new self.multiwidget_modules.'+obj+'(self)');
        widget.record = elem;
        widget.widget_record_bootstrap_class = self.widget_record_bootstrap_class;
        widget.mount(self.el);
    });
}
```

Con Owl possiamo fare molte altre cose.



Domande?