odoo 2023 COMMUNITY DAYS

The Odoo Javascript Framework

Odoo Web Library (OWL)

Mohammed Shekha – Senior Application Engineer

Soumya Mukherjee – Application Engineer

What is OWL?

odoo

Brilliance

With Simplicity

- Compact UI Marvel
- React-Vue Fusion
- Modern, Elegant, Fast

Challenges

Before OWL

- DOM Manipulation Nightmare
- Component Fragmentation
- State Management Complexity
- UI Updates Bottleneck
- Complex UI Interaction
- Slower Performance

Before OWL

- DOM Manipulation Nightmare
- Component Fragmentation
- State Management Complexity
- UI Updates Bottleneck
- Complex UI Interaction
- Slower Performance

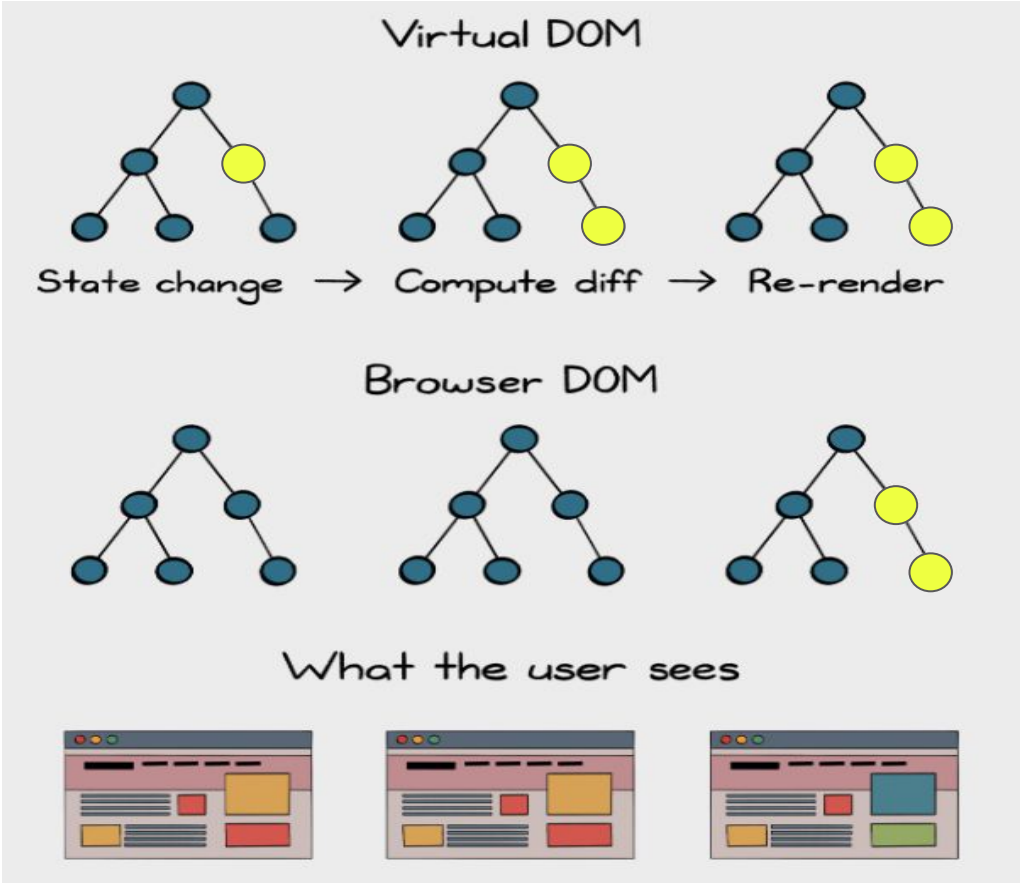
After OWL

- ➢ Virtual DOM Handling
- ➢ Modular Architecture
- ➢ OWL state management
- ➢ Optimised rendering
- ➢ OWL Lifecycle hooks
- ➢ Faster Performance

The Comparison

Aspect	**OWL**	React	Vue
Modularity	Extremely Modular	Modular	Modular
Tooling	Min. Mandatory Tooling	Rich Ecosystem and Tooling	Rich Ecosystem and Tooling
Template Language	XML Based	JSX or Custom Templates	JSX or Custom Templates
Template Compilation	Complies at Runtime	Compiles ahead of time	Compiles ahead of time
Reactivity	Explicit reactivity system	Reactive by default	Reactive by default
Developer Experience	Simplified API with Classes	Can be overwhelming for non-JS specialist	Can be overwhelming for non-JS specialist

How do we manage UI?



VDOM vs DOM

Aspect	VDOM	DOM
Purpose	Intermediary for efficient updates	Live web page structure
Performance	Optimizes updates	May lead to performance bottleneck
Manipulation	Efficient updates w/o direct DOM changes	Requires direct manipulation
Memory Management	Helps minimize memory leaks	Requires careful handling
Browser Interaction	Not visible in browser	Direct Interaction
Cross-platform Usage	Used in Server Side Rendering, Mobile Apps	Primarily Web Browsers

odoo

The Auction Odyssey

Introduction

User Story

User Story

"I'm craving an Auction website that outshines the competition – sleek, seamless, and faster to launch.

Building this from scratch sounds crazy. Can your OWL magic cook up something? Ready to be wowed!"

Alex
Founder, The StartUp

odoo

What should we expect?

The Product

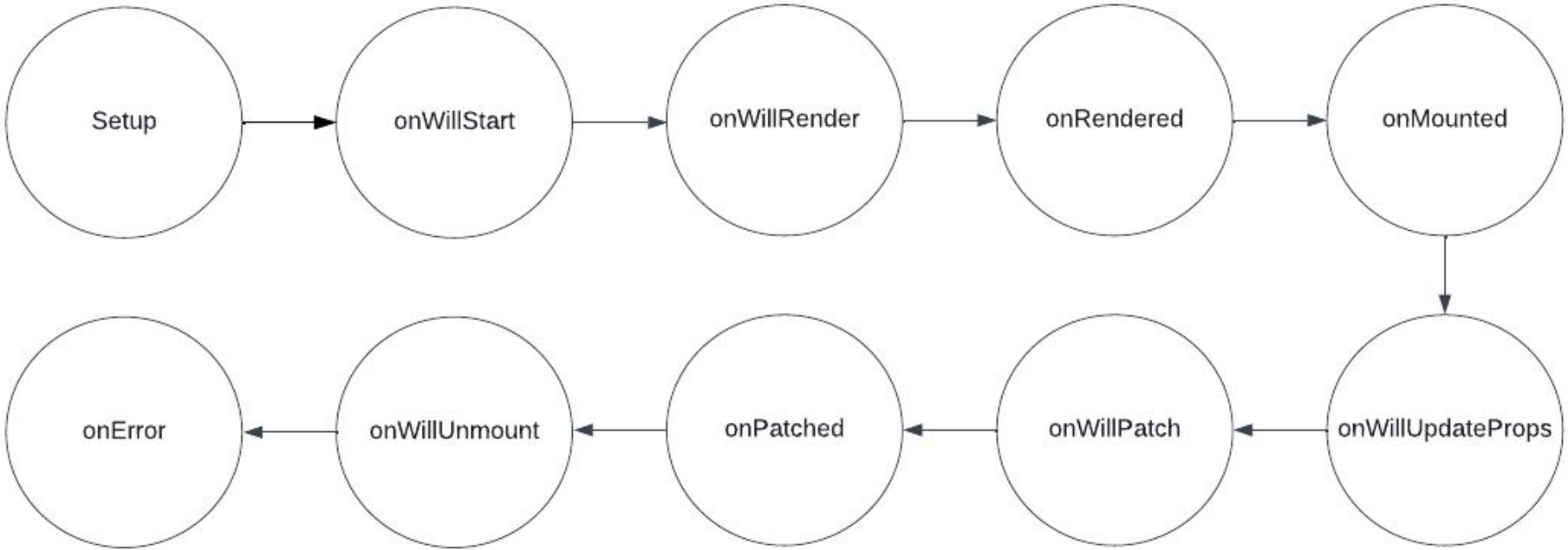
Designed by Manushi Shah - Jr. Application Engineer

OWL Basics

The Beginning

odoo

Components Lifecycle



OWL Component

- A modular unit bundling functions and UI elements.

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

export class Auction extends Component {
    static template = xml`
        <div> Hello World </div>
    `;

    setup() {
      // Constructor method for OWL component.
    }
}
```

OWL Sub - Component

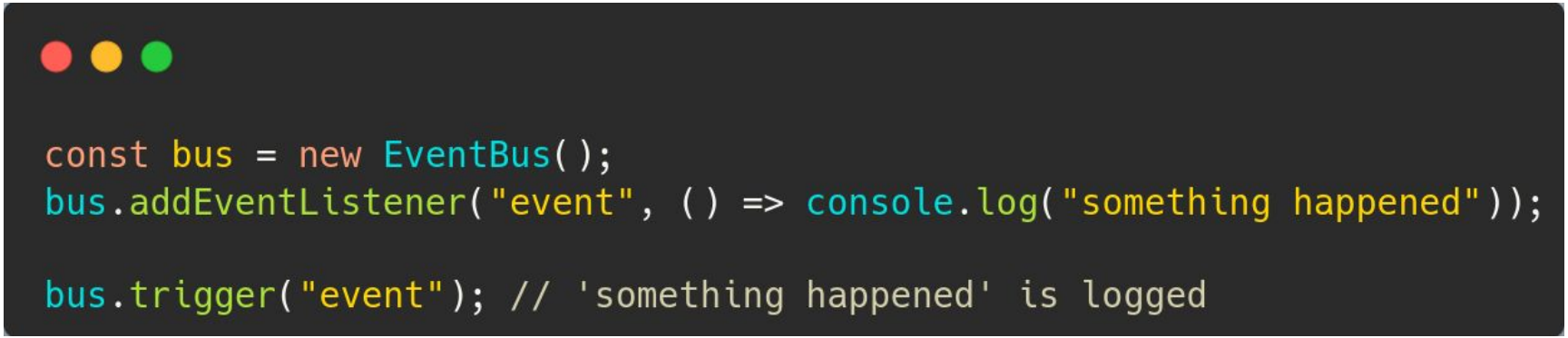
```js
import { Component } from "@odoo/owl";
import { Search } from "./search";
import { Menus } from "./menus";


export class Header extends Component {
  // Linking an external XML file as template
  static template = "auction.header";
}

//Importing external components
Header.components = { Search, Menus }
```

JS Logic

```xml
<templates xml:space="preserve">
    <t t-name="auction.header" owl="1">
        <Menus />
        <Search />
    </t>
</templates>
```

XML

Event Bus



```
const bus = new EventBus();
bus.addEventListener("event", () => console.log("something happened"));

bus.trigger("event"); // 'something happened' is logged
```

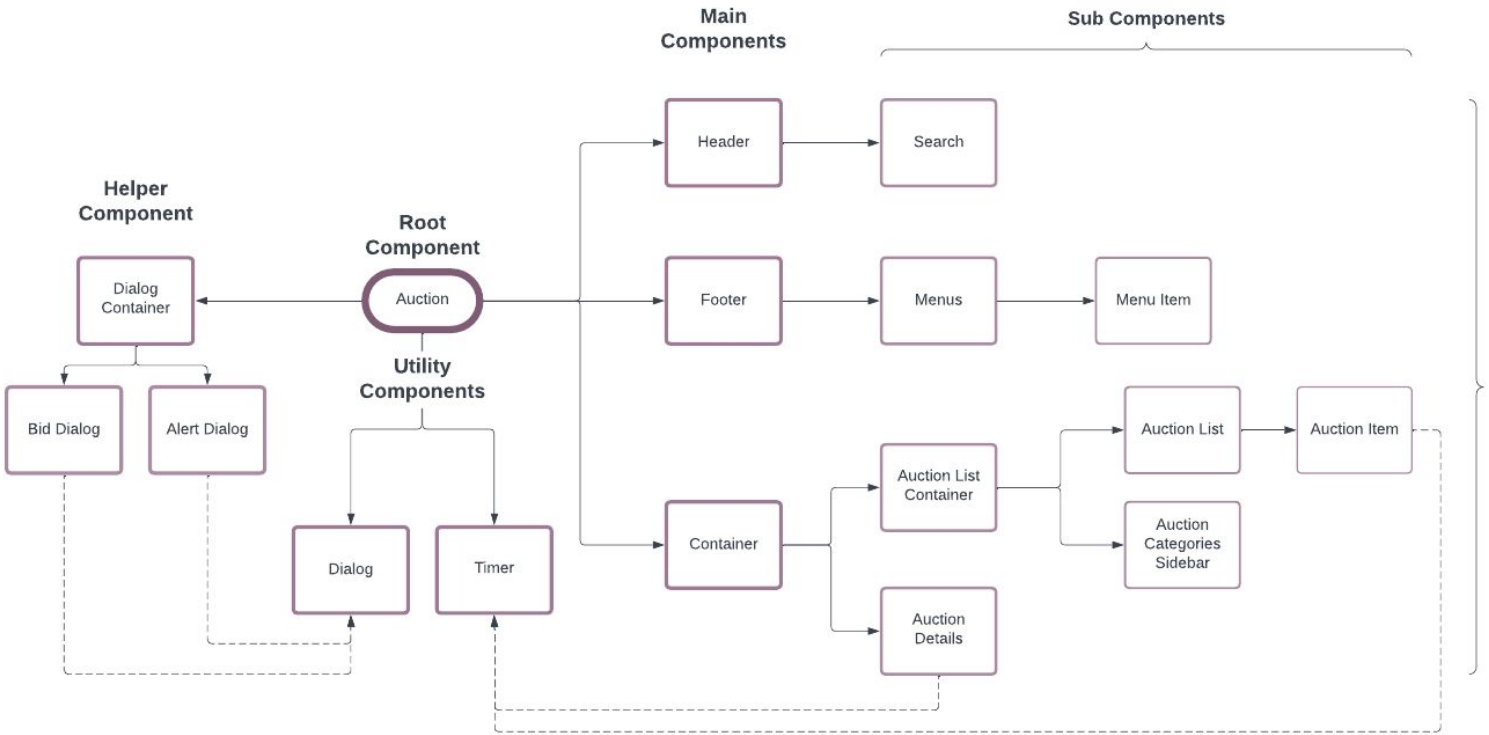
Environment

- Easy to mount with any Application
- Can be used by all the Components
- Stays in a Shallow Frozen state

Application

Environment Variable

Architecture Design



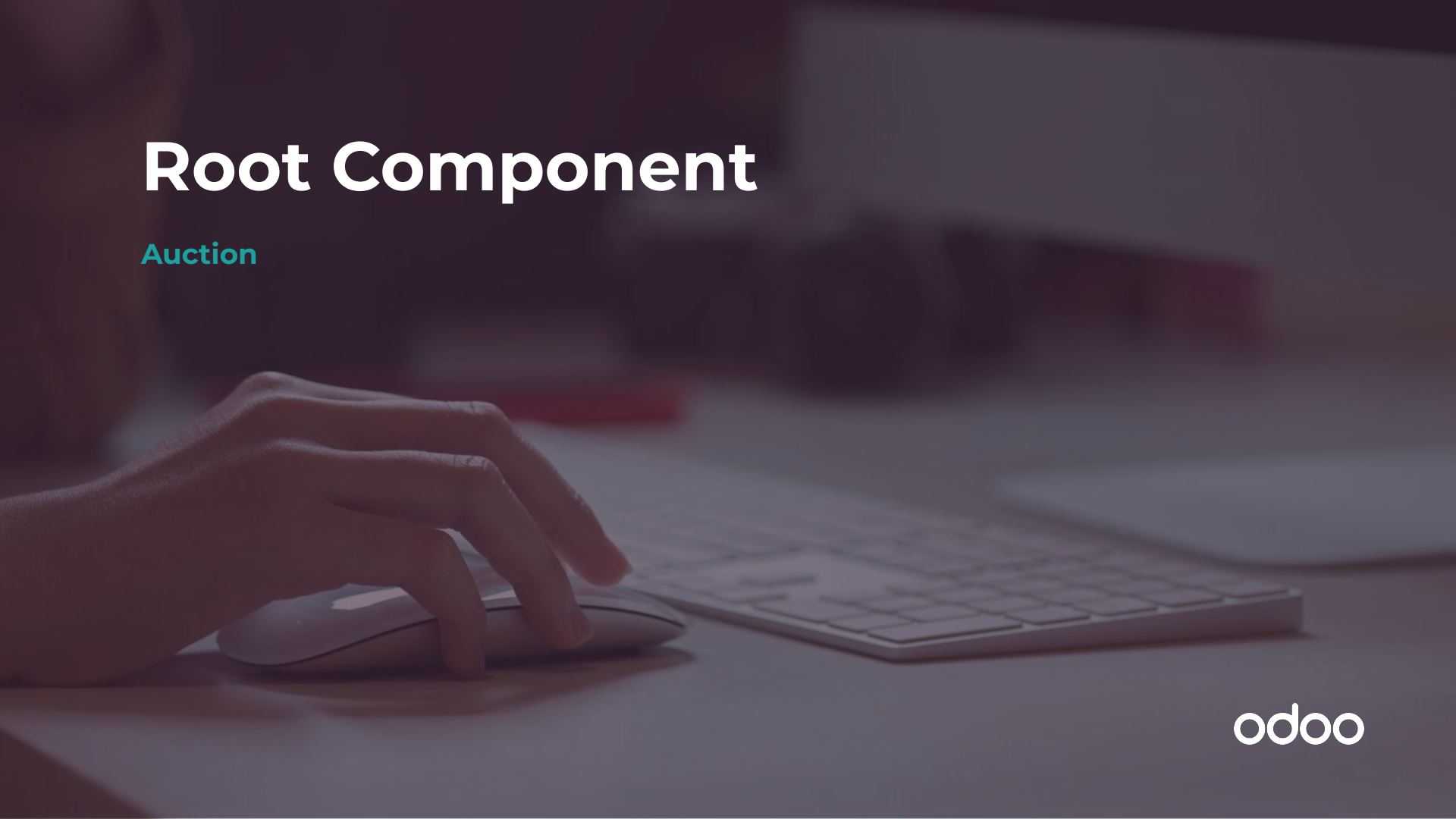
Setting up application

```
owl.whenReady(async () => {
    const bus = new EventBus();
    const env = { bus, rpc };
    const translateFn = (str) => {
        // Translation will be covered later
    }

    const app = new App(Auction, {
        name: "Auction",
        env,
        templates,
        translateFn,
        dev: true,
    });
    const root = await app.mount(document.body);
});
```

- To load component mount is used.
- Binding all components, under an App helps in environment management.
- Here we can also see an implementation of whenReady hook.

Root Component

Auction



odoo

Architecture Progress

Root Component (Auction)

- Here, we can see the implementation of Sub-Components

```
export class Auction extends Component {
    static template = "auction.root";

    setup() {
        const auctionModel = new AuctionModel();
    }
}

Auction.components = { Header, Footer }
```

JS - Logic

```
<t t-name="auction.root" owl="1">
    <Header/>
    <main>
      <!-- Container logic, will be discussed later -->
    </main>
    <Footer/>
</t>
```

XML - UI

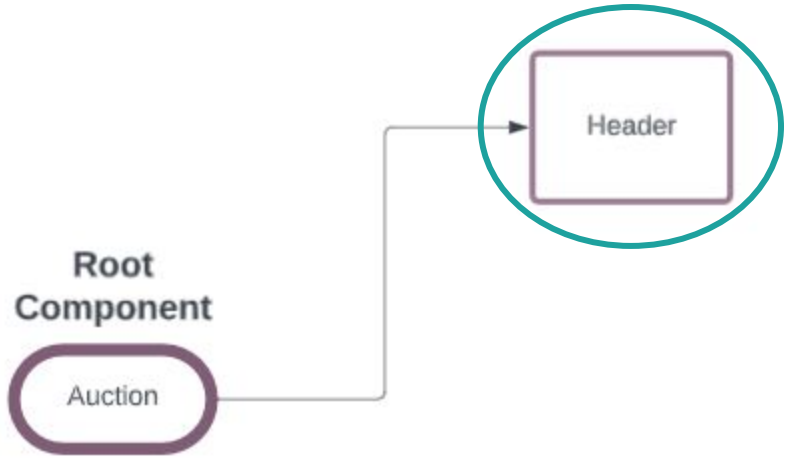
Implementing EventBus

```
export class AuctionModel extends EventBus {
    constructor(params) {
        super();
        this.cache = {};
    }
    load(name, deft) {
        if (this.cache[name] !== undefined) {
            return this.cache[name];
        }
        return deft;
    }

    save(name, data) {
        this.cache[name] = data;
    }

    filterAuctionItems(categoryID, activeMenuItem) {
        // Used to filter items according to start & end time
    }

    getAuctionItem(id) {
        // Will fetch Auction Item based on ID
    }
}
```

Header Component

Auction > Header

Architecture Progress



Header Component

- Here, a similar implementation but with a t-on-click event

```
export class Header extends Component {
    // Similar code

    onClickLogo() {
        window.location.href = "/auction";
    }
}

Header.components = { Search }
```

JS - Logic

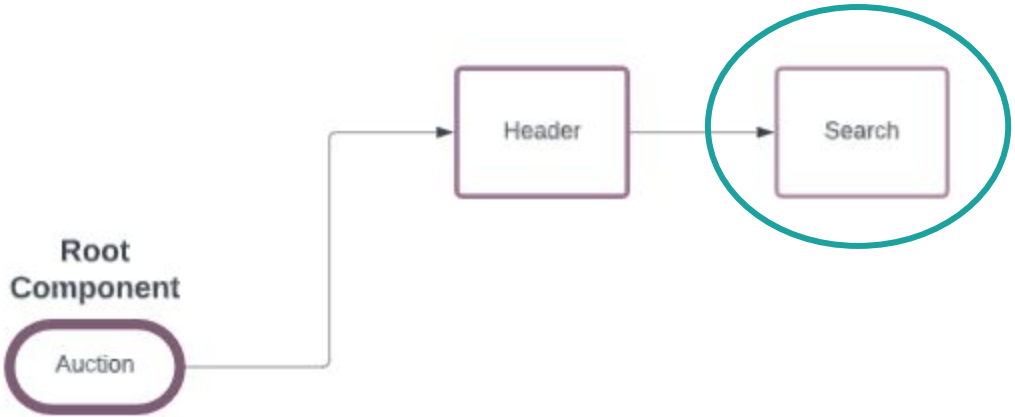
```
<t t-name="auction.header" owl="1">
    <div class="col-2">
        // Website logo
    </div>
    <Search />
</t>
```

XML - UI

Search Component

Auction > Header > Search

Architecture Progress



Search Component

```
export class Search extends Component {
    setup() {
        const searchInput = useRef('search_input');
        function autofocus() {
            searchInput.el.focus();
        }
        onMounted(autofocus);
    }
}
```

JS - Logic

Here we can see implementation of two Hooks:

- useRef
- onMounted

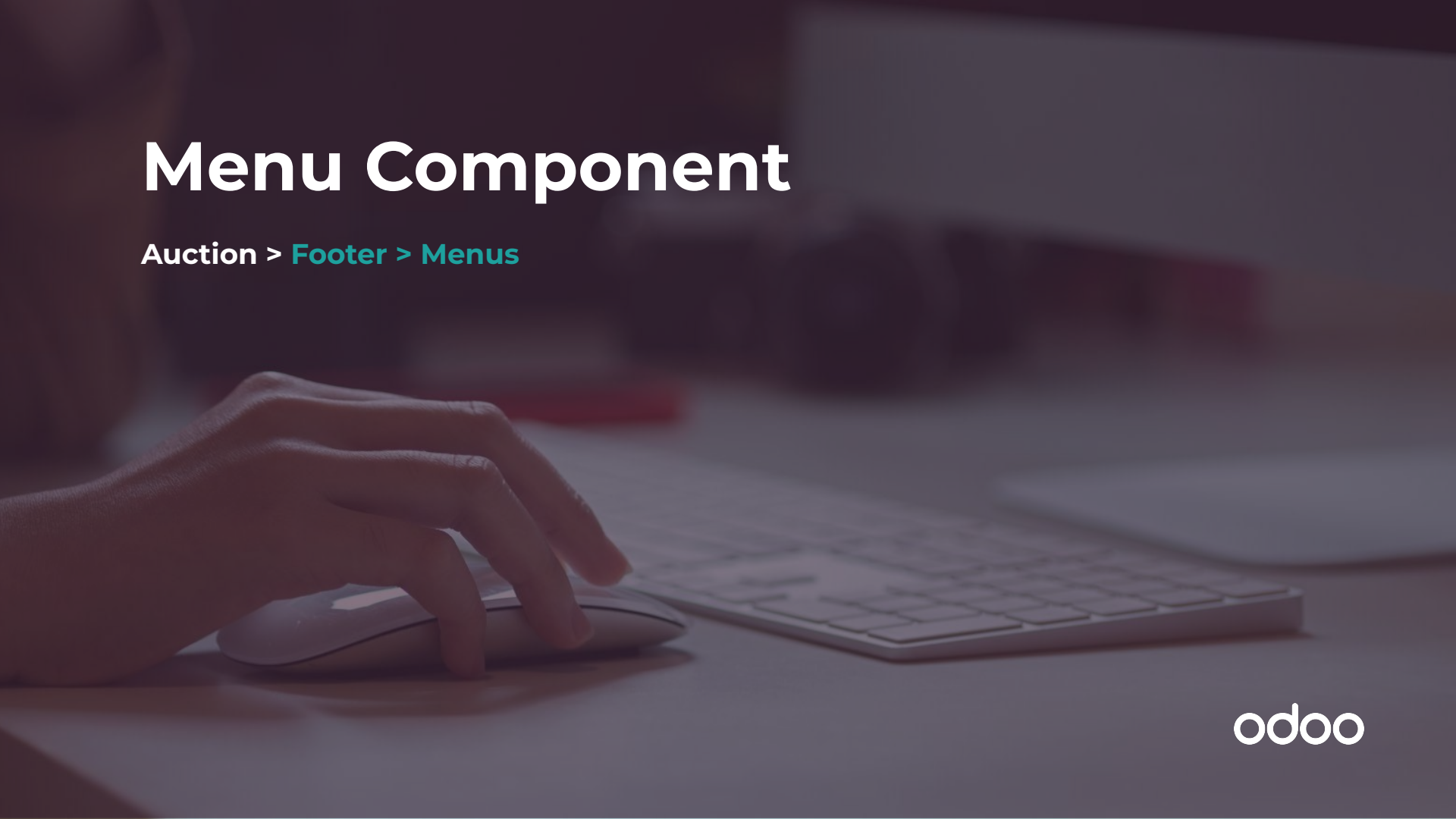
```
<input type="input" class="o_input_search" t-ref="search_input"/>
```

XML - UI

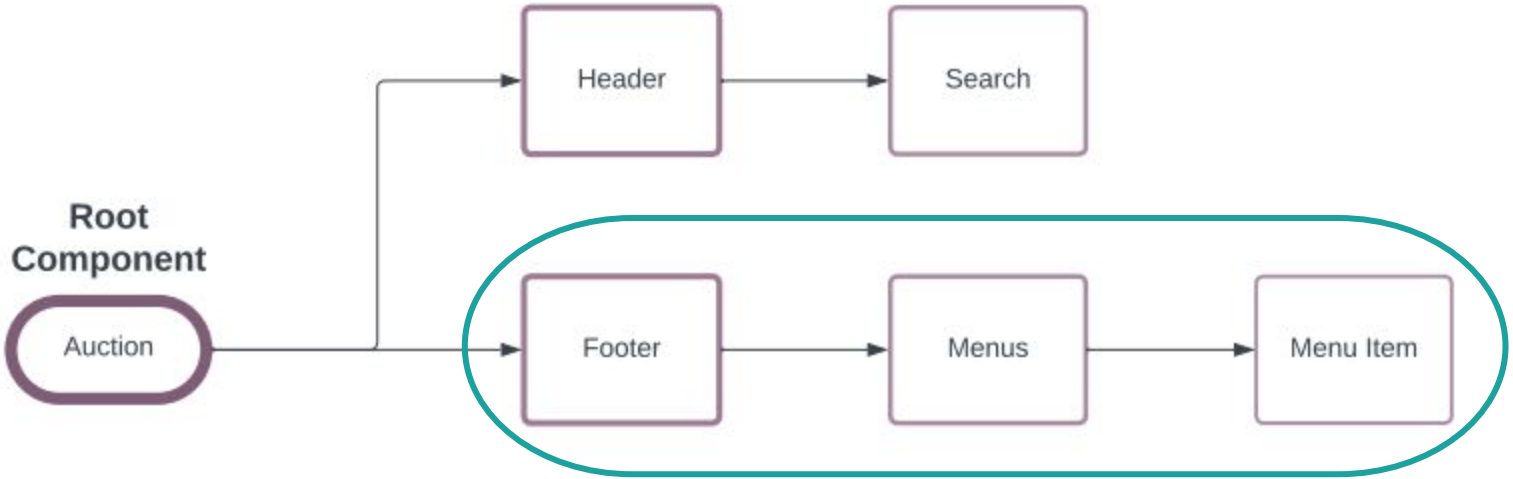
In order to use useRef, set t-ref on the element

Menu Component

Auction > Footer > Menus



Architecture Progress

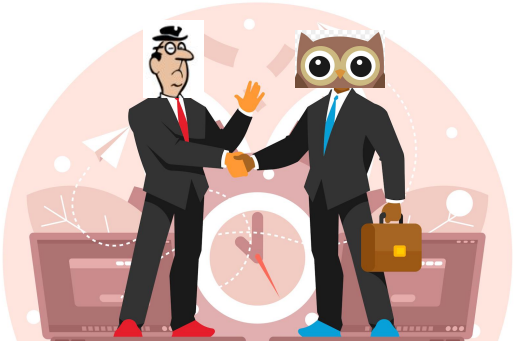


- Like previous components, these components offers same sub component features.

Let’s talk about Props

Data

Component A with Data



Sending data to Component B

Component B with same Data

Implementation of Props

```
<Footer activeMenuItem="mainScreenPropsFielded.activeMenuItem or 'all'"/>
```

File: Root Component

```
<Menus activeMenuItem="props.activeMenuItem"/>
```

File: Footer Component

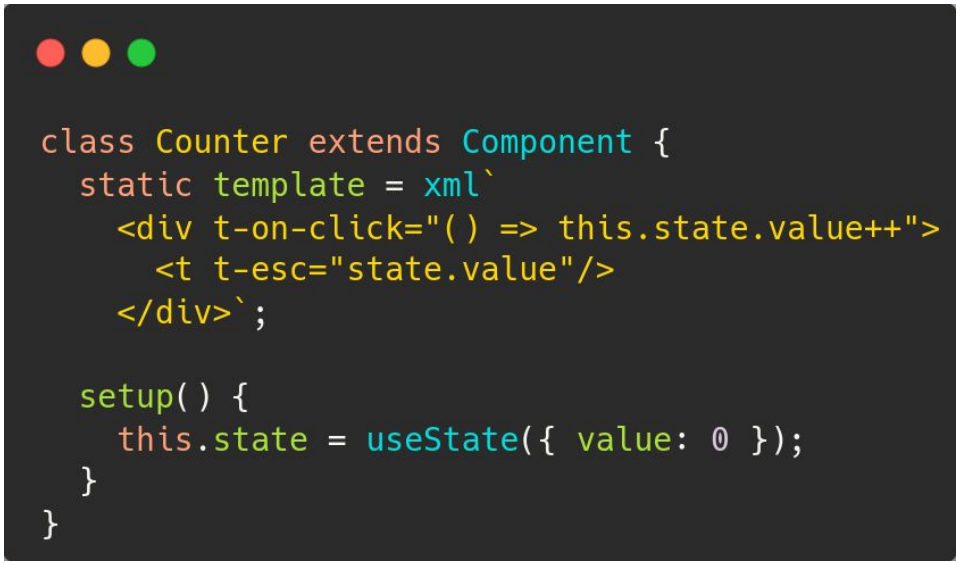
Menus Component

- Now, the props has reached its destination. Time to use state management for better UI updations.

```
export class Menus extends Component {
  setup() {
    this.menuRef = useRef('menu')
    this.state = useState({ activeMenuItem: this.props.activeMenuItem })
  }
  // Some extra code
}

Menus.components = { MenuItem }
```

Special Topic: useState

- Helps in controlling - state of variables, and rendering responsible component

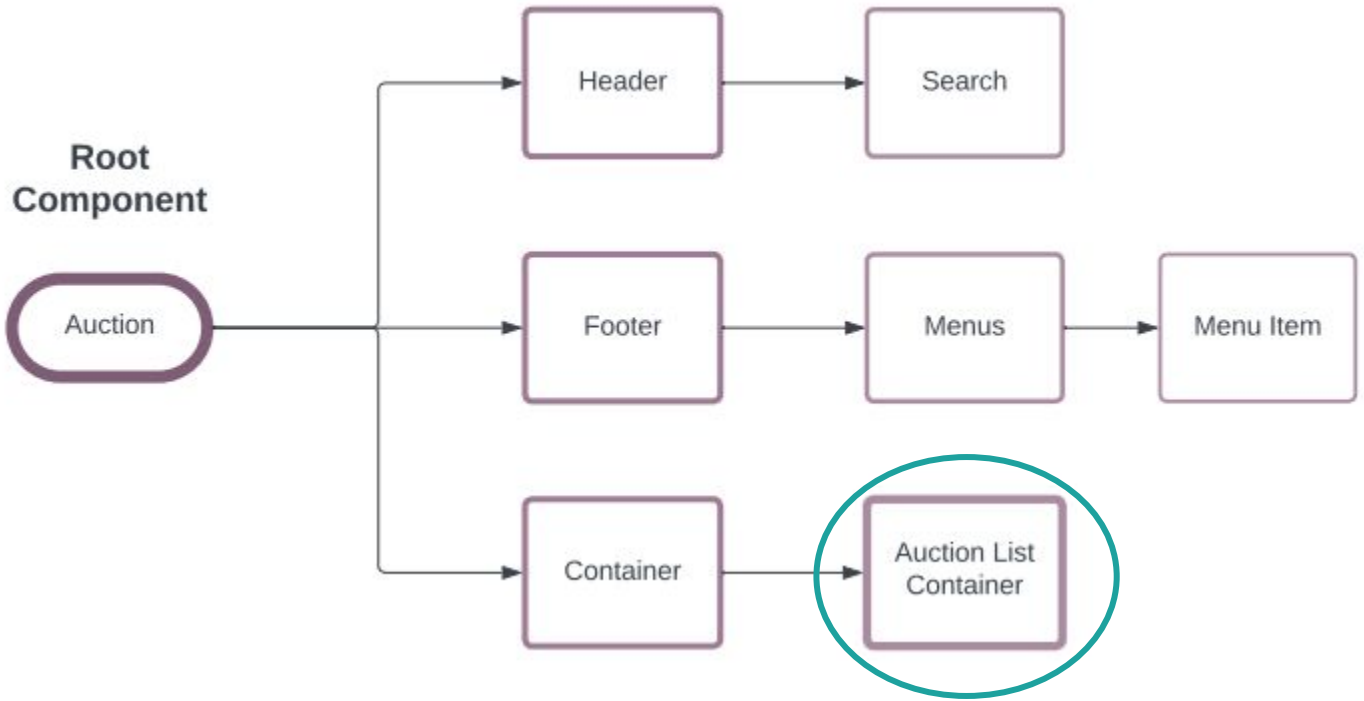


```
class Counter extends Component {
  static template = xml`
    <div t-on-click="() => this.state.value++">
      <t t-esc="state.value"/>
    </div>`;

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

Screen: Auction List Container

Auction > Container > Auction List Container

Architecture Progress



JS Logic

```
export class AuctionListContainer extends Component {
    setup() {
        super.setup();
        this.state = useState({
            items: [],
            activeCategory: 'all',
        });
        onWillStart(this.willStart);
        this.auctionFetch = useFetchAuctions();
        onWillUpdateProps((nextProps) => {
            const filteredItems = ... //fetching filtered items
            this.state.items = filteredItems;
        });
    }
    async willStart() {
        this.datas = await this.auctionFetch();
        this.env.auctionModel.save('datas', this.datas);
        const filteredItems = ... //fetching filtered items
        this.state.items = filteredItems;
    }
}
```

JS Logic

Implementation of hooks:

- useState
- onWillStart
- onWillUpdateProps

Special Topic: onWillUpdateProps

```
export class AuctionListContainer extends Component {
    setup() {
        this.state = useState({
            items: [],
            activeCategory: 'all',
        });
        onWillUpdateProps((nextProps) => {
            const filteredItems =
                  this.env.auctionModel.filterAuctionItems(
                    this.state.activeCategory, nextProps.activeMenuItem || 'all'
                  );
            this.state.items = filteredItems;
        }); // Rest of the code
```

- Clicking on a MenuItem triggers change_active_menu
- Which updates main Props
- Triggering onWillUpdateProps

```
export class Auction extends Component {
    setup() {
        this.mainScreen = useState({ name: 'AuctionList', component: AuctionListContainer, props: {} });
        this.env.bus.addEventListener("change_active_menu", this.onChangeScreen.bind(this));
    }

    onChangeScreen(ev) {
        // Some code
        this.mainScreen.props = ev.detail;
    }
}
```

Custom Hook

```
export class AuctionListContainer extends Component {
    setup() {
        this.state = useState({
            items: [],
            activeCategory: 'all',
        });
        onWillStart(this.willStart);
        this.auctionFetch = useFetchAuctions();

    }
    async willStart() {
        this.datas = await this.auctionFetch();
    }
}
```

Calling Method

Here we can see implementation of useEnv hook

```
export function useFetchAuctions() {
    const env = useEnv();
    return () => {
        return env.rpc("/get_auction_data", {});
    }
}
```

Custom Hook

Understanding Custom DOM Event

```
export class AuctionListContainer extends Component {
    onFilterItems(ev) {
        const categoryID = ev.detail.id;
        const filteredItems =
              this.env.auctionModel.filterAuctionItems(
                categoryID, this.state.activeMenuItem
              );
        this.state.items = filteredItems;
        this.state.activeCategory = categoryID;
    }
}
```

Continuing previously discussed code,

Here is a proper use case of useState

```
<t t-name="auction.AuctionListContainer" owl="1">
    <div class="row" t-on-filterItems="onFilterItems">
            <AuctionCategorySidebar categories="datas.categories"/>
            <AuctionList auctionItems="state.items"/>
    </div>
</t>
```

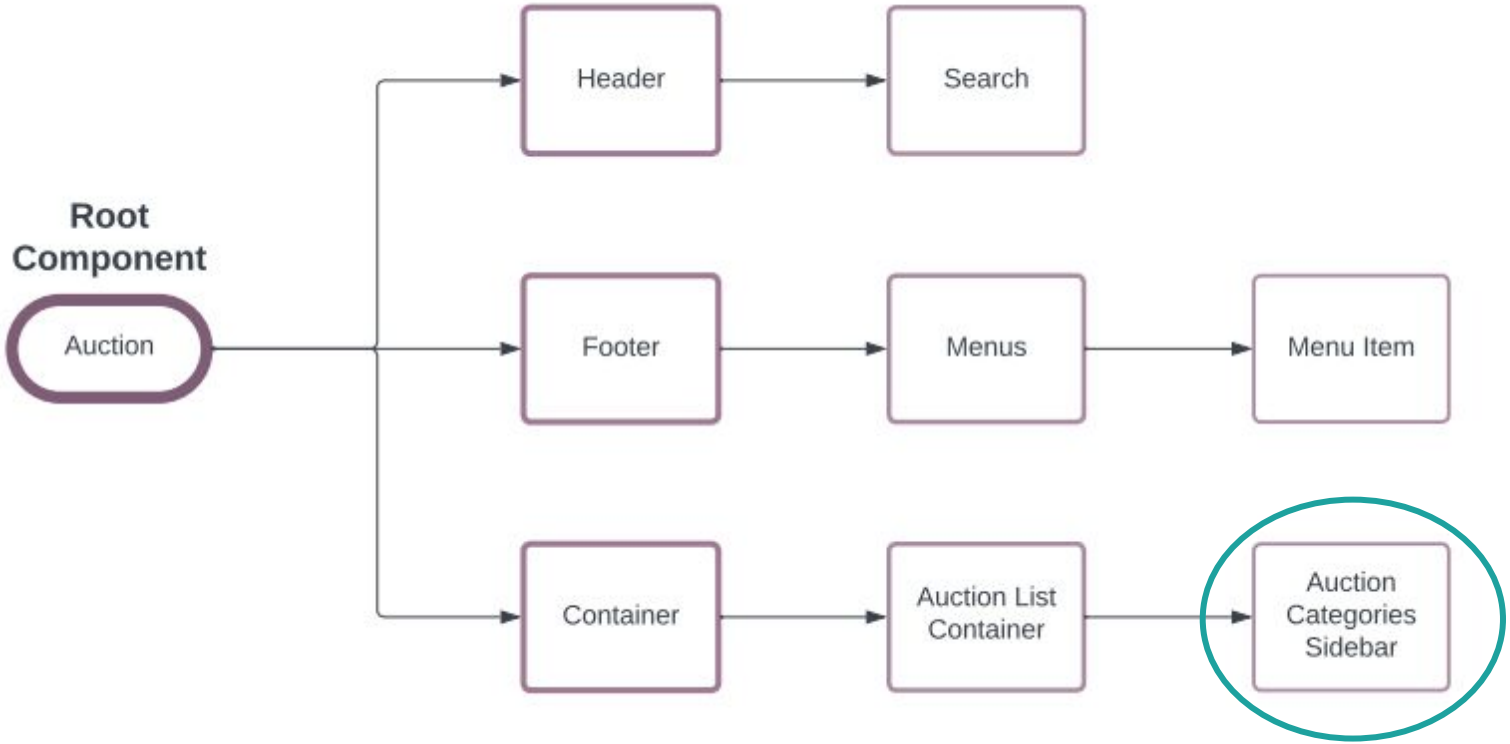
Here, a custom DOM event is patched with OWL

t-on-filterItems

Auction Categories Sidebar

Auction > Container > Auction List Container > Auction Categories Sidebar

Architecture Progress



Understanding CustomEvent Logic

```
export class AuctionCategorySidebar extends Component {
    etup() {
        this.categoryRef = useRef('category_element');
    }

    onClickCategory(ev) {
        const target = ev.currentTarget;
        const id = target.getAttribute('data-id');
        const categoryElements = this.categoryRef.el.querySelectorAll('.auction_category.active');
        [...categoryElements].forEach((categoryElement) => categoryElement.classList.remove('active'));
        target.classList.add('active');
        const customEvent = new CustomEvent("filterItems", { bubbles: true, detail: { id } });
        this.categoryRef.el.dispatchEvent(customEvent);
    }
}
```

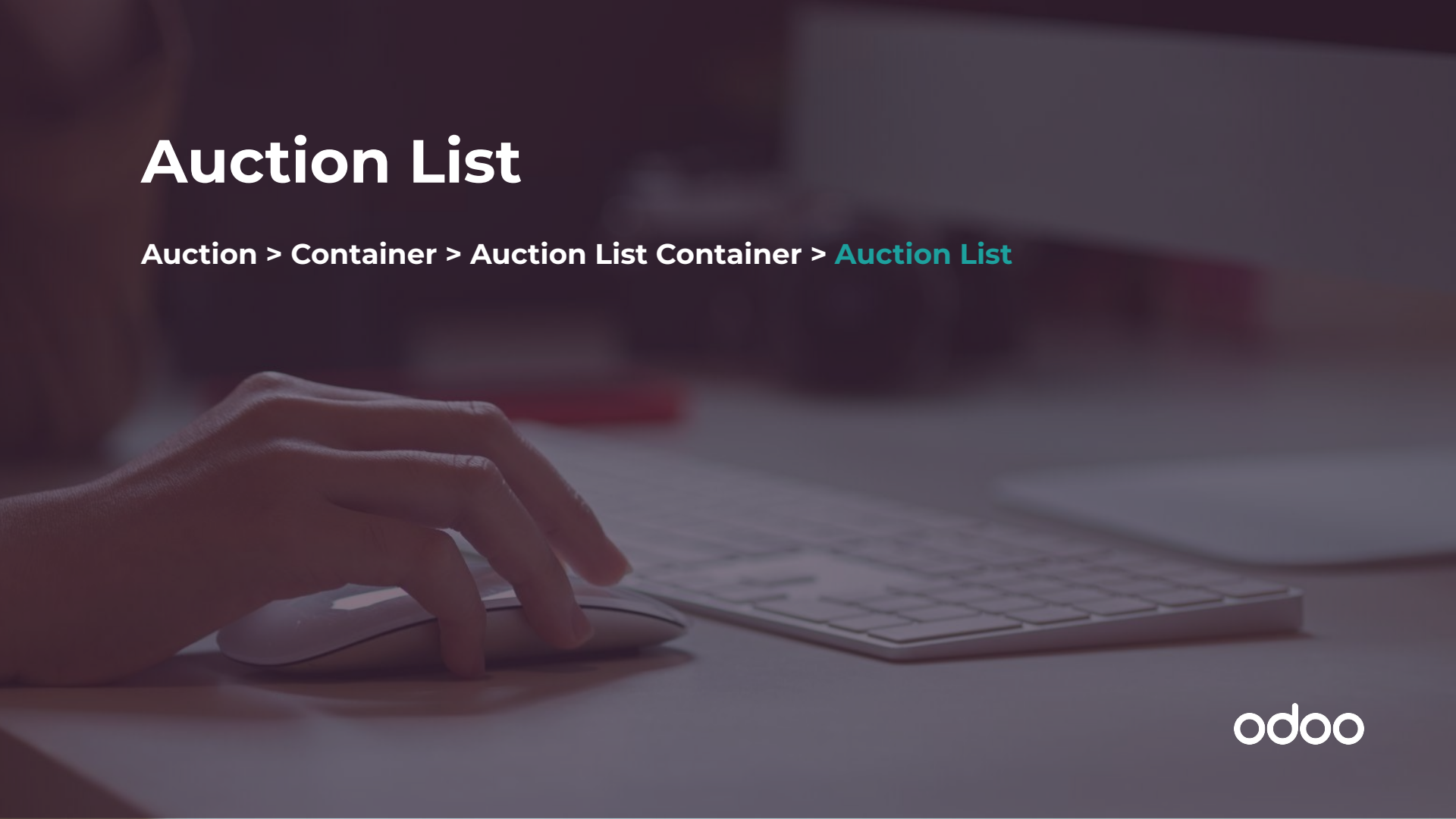
Note: This method is triggered using “on click” event on any Category

Understanding CustomEvent XML

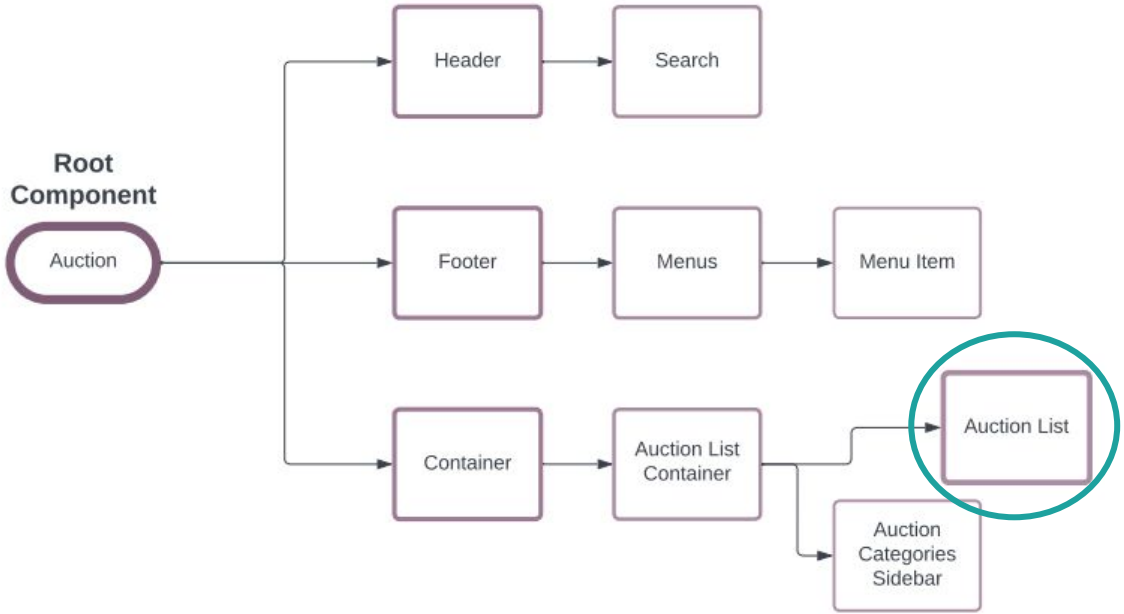
```
<t t-name="auction.AuctionCategorySidebar" owl="1">
    <div class="o_category_list " t-ref="category_element">
        <div class="auction_category active" data-id="all" t-on-click="onClickCategory">
            <span class="category_selected" t-translation="off">All</span>
        </div>
        <div
             t-foreach="props.categories"
             t-as="category"
             t-key="category.id"
             class="auction_category"
             t-att-data-id="category.id"
             t-on-click="onClickCategory"
        >
            <span class="category_selected" t-out="category.name"></span>
        </div>
    </div>
</t>
```

Auction List

Auction > Container > Auction List Container > Auction List



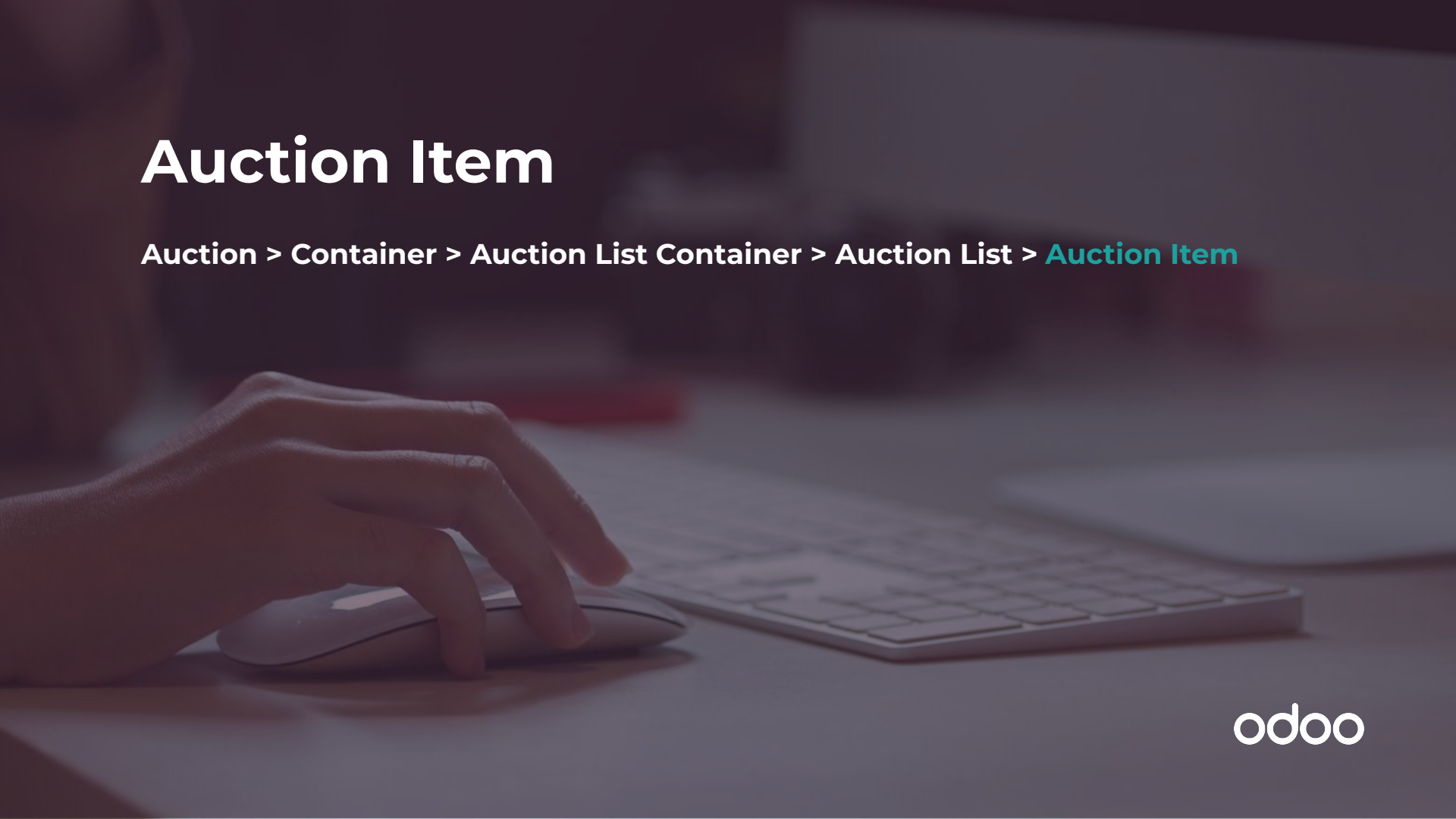
Architecture Progress



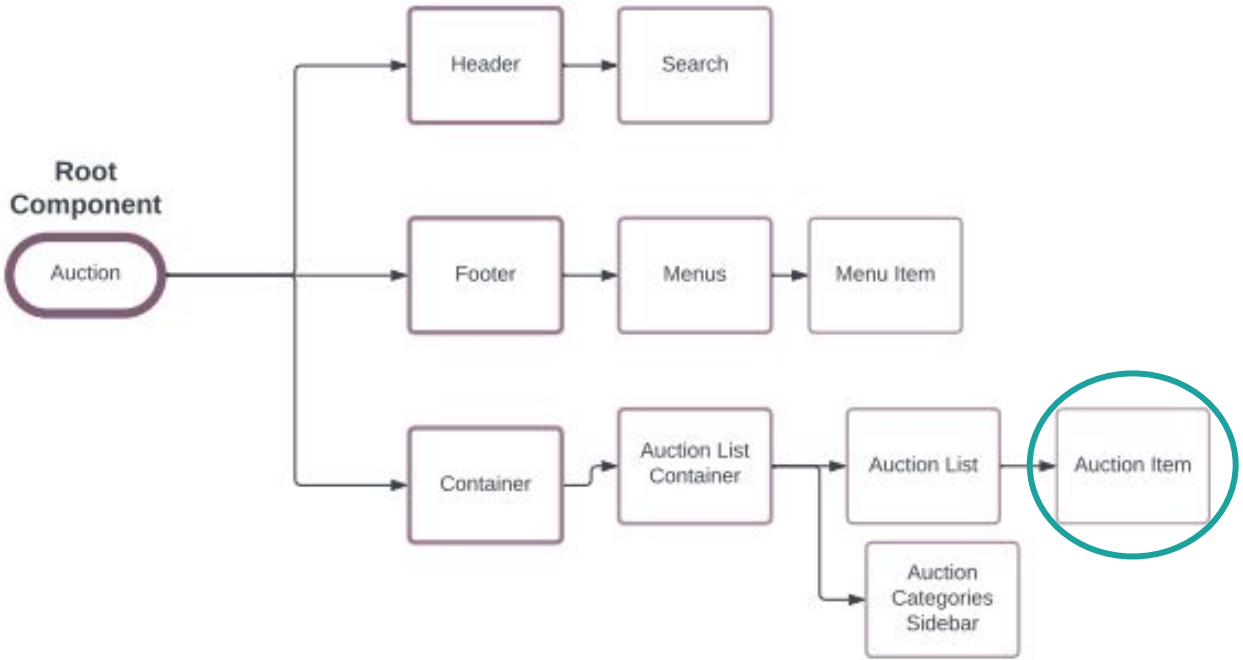
- Like previous components, this component offers same sub component features.

Auction Item

Auction > Container > Auction List Container > Auction List > Auction Item



Architecture Progress



Auction Item Component

This change depends on EventBus application

```
export class AuctionItem extends Component {
    // Some basic stuff

    onClickItem(ev) {
        this.env.bus.trigger('change_screen',
            { 'screen_name': 'AuctionDetails', auctionItem: this.auctionItem }
        );
    }
}
```

```
<t t-name="auction.AuctionItem" owl="1">
    <a
        t-att-data-id="props.auctionItem.id"
        t-on-click.prevent="onClickItem">

    </a>
</t>
```

Here, we can see implementation of trigger method, which is a part of **OWL EventBus** feature

Modifying Root Element

Auction

Understanding EventBus Event Handling

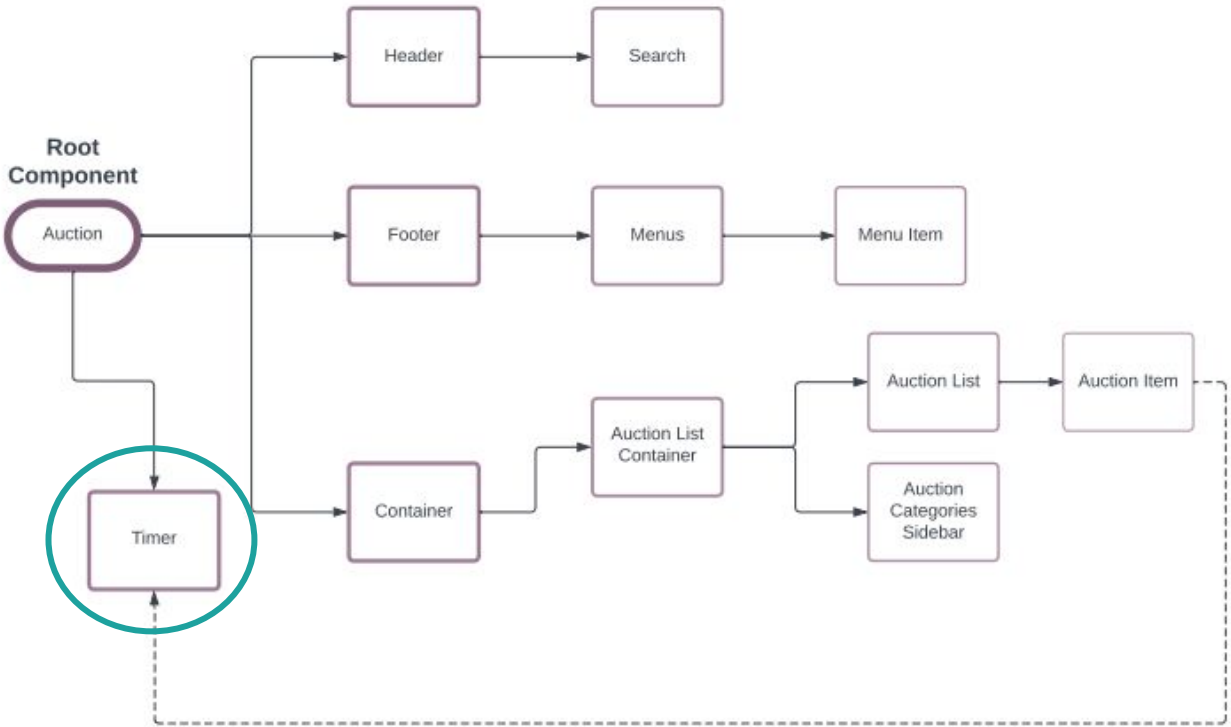
```
export class Auction extends Component {
    setup() {
        this.mainScreen = useState({ name: 'AuctionList', component: AuctionListContainer, props: {} });
        this.env.bus.addEventListener("change_screen", this.onChangeScreen.bind(this));
        this.env.bus.addEventListener("change_active_menu", this.onChangeScreen.bind(this));
    }

    get mainScreenPropsFielded() {
        return Object.assign({}, this.mainScreen.props);
    }

    onChangeScreen(ev) {
        const screenRegistry = registry.category("screens");
        const screen = screenRegistry.get(ev.detail.screen_name);
        this.mainScreen.name = ev.detail.screen_name;
        this.mainScreen.component = screen;
        this.mainScreen.props = ev.detail;
    }
}
```

```
<main>
    <t t-component="mainScreen.component" t-props="mainScreenPropsFielded" t-key="mainScreen.name" />
</main>
```

Timer Component

Auction > Timer

Architecture Progress



Timer Component

```
export class Timer extends Component {
    setup() {
        this.state = useState({days: 0, hours: 0, minutes: 0, seconds: 0});
        const endDatetime = moment(this.props.endDatetime, 'YYYY-MM-DD hh:mm:ss');
        const currentTime = moment();
        let duration = moment.duration(endDatetime.diff(currentTime));
        onMounted(() => {
            const intervalId = setInterval(() => {
                duration = moment.duration(duration - 1000, 'milliseconds');
                // Time Out check
                if (duration.asSeconds() <= 0) {
                  clearInterval(intervalId);
                  return
                }

                //Otherwise
                duration = moment.duration(duration.asSeconds() - 1, 'seconds');
                this.state.days = duration.days();
                this.state.hours = duration.hours();
                this.state.minutes = duration.minutes();
                this.state.seconds = duration.seconds();
              }, 1000);
        });
    }
}
```

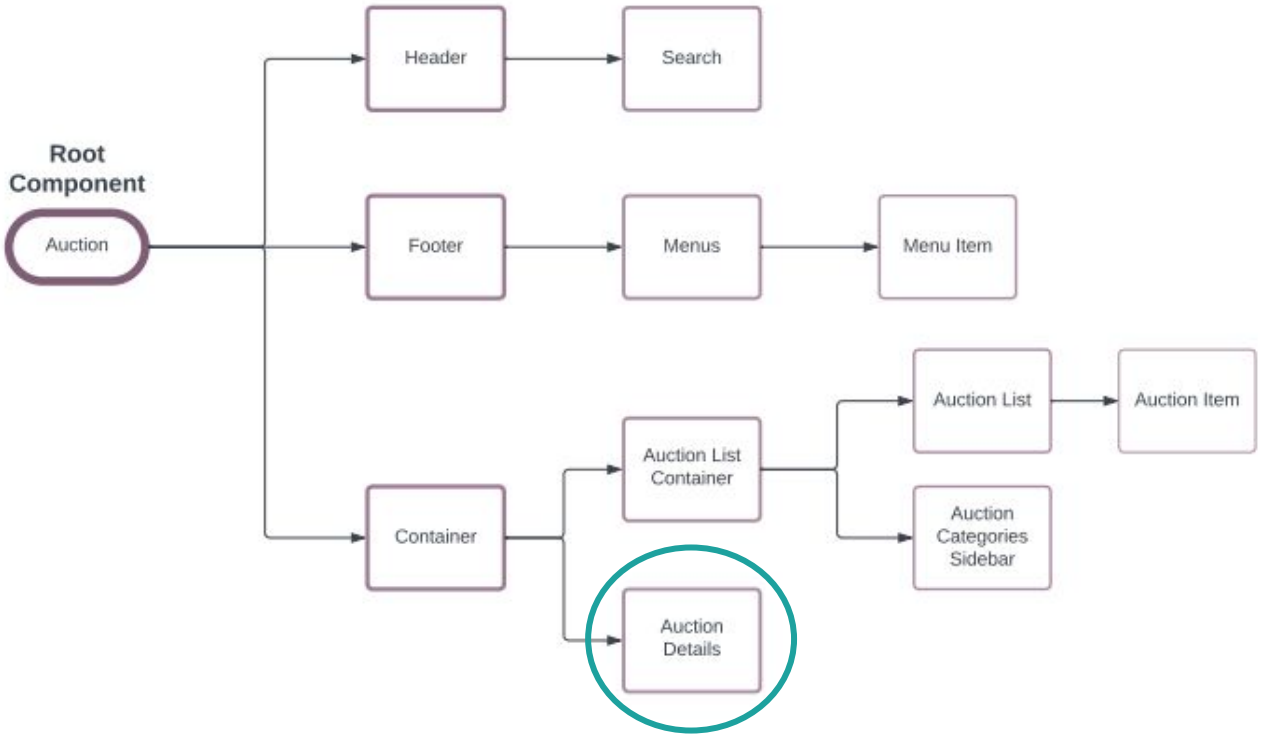
Another implementation scenario for hooks:

- useState
- onMounted

Screen: Auction Details

Auction > Container > Auction Details

Architecture Progress



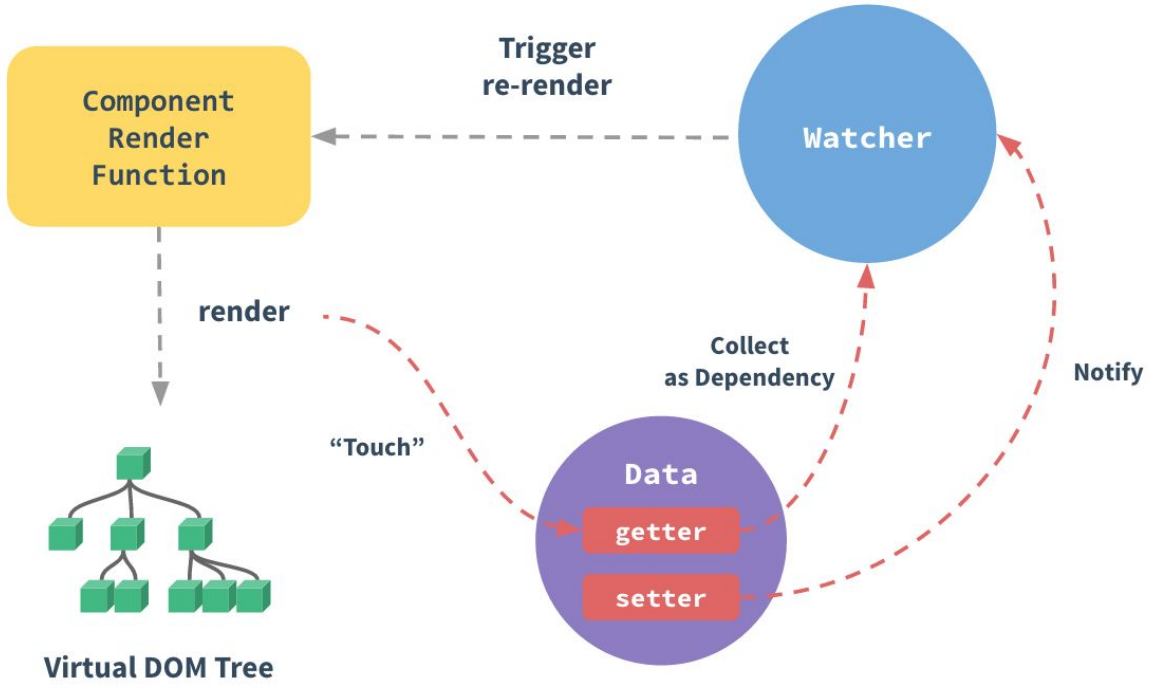
Auction Details Component

```
export class AuctionDetails extends Component {
    // Some basic code

    onPlaceBid(ev) {
        this.env.bus.trigger('add_dialog', { dialog: BidDialog, props: {
            'title': 'Place Bid',
        }});
    }
}
```

Similar implementation of
OWL EventBus

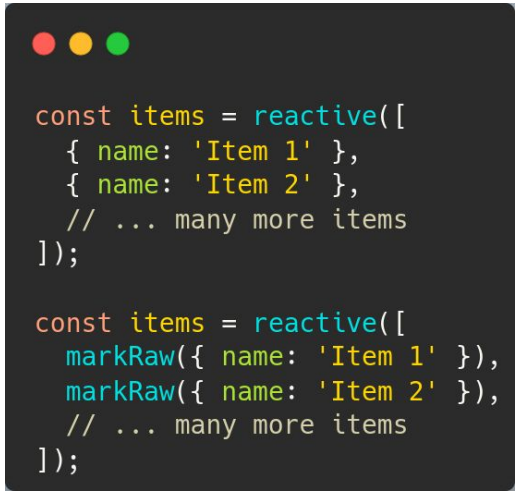
```
<t t-name="auction.AuctionDetails" owl="1">
    <div class="mt-4">
        <button type="button" class="btn btn-primary" t-on-click="onPlaceBid">Place Bid</button>
    </div>
</t>
```

Special Topic: Reactive



markRaw

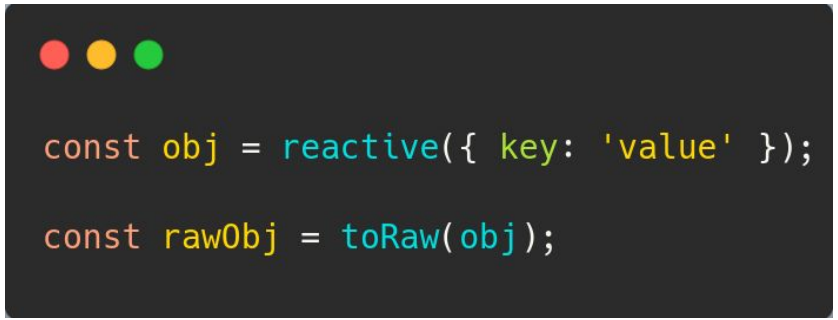
- Performance: Prevents the creation of unnecessary reactive objects.
- Immutability: Ideal for objects known to be immutable.
- Control: Mark objects that should not trigger reactivity.



```
const items = reactive([
  { name: 'Item 1' },
  { name: 'Item 2' },
  // ... many more items
]);

const items = reactive([
  markRaw({ name: 'Item 1' }),
  markRaw({ name: 'Item 2' }),
  // ... many more items
]);
```

toRaw

- Debugging: Inspect the original object without proxy complexity.
- Accessing Data: Retrieve the raw, non-reactive data from reactive proxies.
- Comparisons: Ensure equality between the original and the proxy object.



Modifying Root Element

Auction

Root: Adapting new changes

```
export class Auction extends Component {
    setup() {
        this.env.bus.addEventListener("add_dialog", this.onAddDialog.bind(this));
        this.dialogs = reactive({});
        this.dialogId = 0;
    }
    onAddDialog(ev) {
        const id = this.dialogId++
        this.lastId = id;
        const close = () => {
            if (this.dialogs[id]) {
                delete this.dialogs[id];
                if (ev.detail.onClose) {
                    ev.detail.onClose();
                }
            }
        }
        const dialog = {
            class: ev.detail.dialog,
            props: Object.assign({}, ev.detail.props, { close, id }),
            dialogData: {
                close,
            }
        };
        this.dialogs[id] = dialog;
    }
}
```

Here, we can see a similar implementation of handling custom events using EventBus.

We can also see implementation of reactive hook.

Special Topic: Slots

```xml
<div class="info-box">
  <div class="info-box-title">
    <t t-slot="title"/>
    <span class="info-box-close-button" t-on-click="close">X</span>
  </div>
  <div class="info-box-content">
    <t t-slot="content"/>
  </div>
</div>
```

Setting up slots in a component

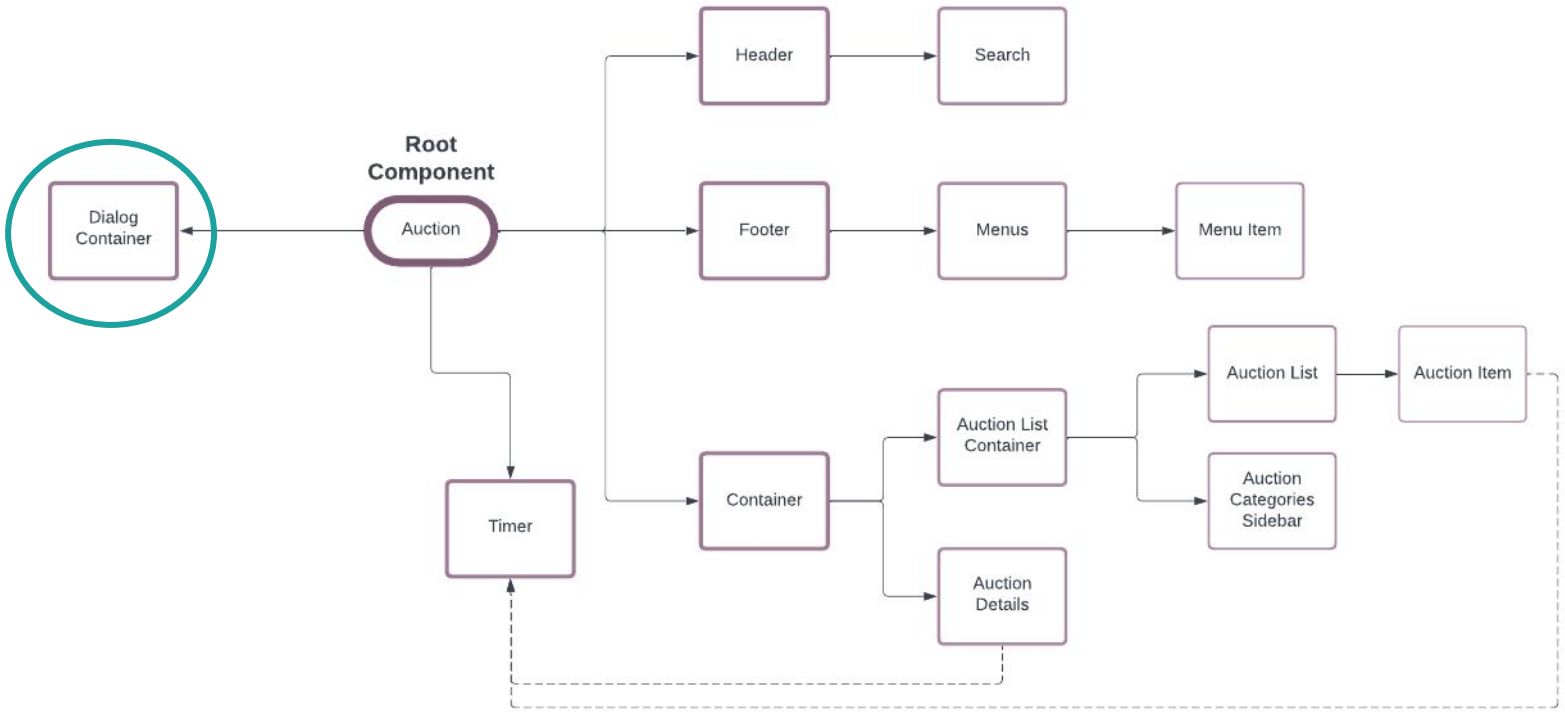
```xml
<InfoBox>
  <t t-set-slot="title">
    Specific Title. It can be text / html.
  </t>
  <t t-set-slot="content">
    <div>Here content will come</div>
  </t>
</InfoBox>
```

Using it from a different component

Dialog Container Component

Auction > Dialog Container

Architecture Progress



Understanding Sub Environments

```
export class DialogContainer extends Component {
    setup() {
        onError(this.handleError);
    }

    handleError(error, dialog) {
        dialog.props.close();
        Promise.resolve().then(() => {
            throw error;
        });
    }
}
```

Here, we can see implementation for two new Hooks:

- onError
- useSubEnv

```
export class WithEnv extends Component {
    setup() {
        useSubEnv(this.props.env);
    }
}
WithEnv.template = xml`<t t-slot="default"/>`;
```

Using useSubEnv one can create a dynamic environment for specific components

Dialog Container UI - XML

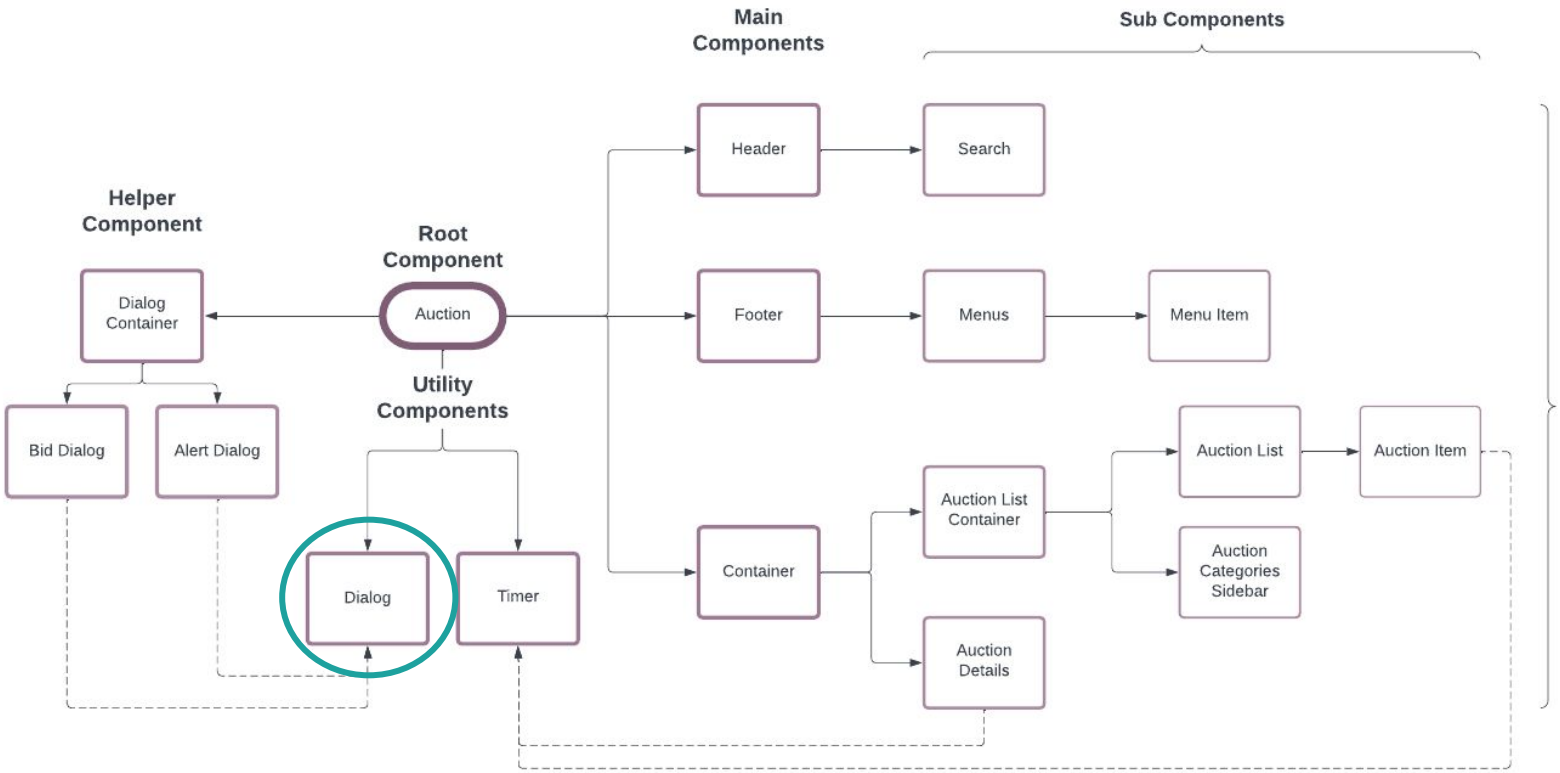
```
DialogContainer.components = { WithEnv };
DialogContainer.template = xml`
    <div class="o_dialog_container" t-att-class="{'modal-open': Object.keys(props.dialogs).length > 0}">
        <div>
            <t t-foreach="Object.values(props.dialogs)" t-as="dialog" t-key="dialog.id">
                <WithEnv env="{ dialogData: dialog.dialogData }">
                    <t t-component="dialog.class" t-props="dialog.props"/>
                </WithEnv>
            </t>
        </div>
    </div>
`;
```

Here, we can see the effect of useSubEnv on Dynamic Components

Dialog Component

Auction > Dialog

Architecture Progress



Dialog Component

```
export class Dialog extends Component {
    setup() {
        this.data = this.env.dialogData;
        useExternalListener(document.body, "click", this.closeActiveDialog.bind(this));
        this.modalElement = useRef('modalRef')
    }

    closeActiveDialog(ev) {
        const modalContent = this.modalElement.el.querySelector('.modal-content')
        if (!modalContent.contains(ev.target)) {
            this.data.close();
        }
    }
}
```

Here, we can see usage of useExternalListener hook

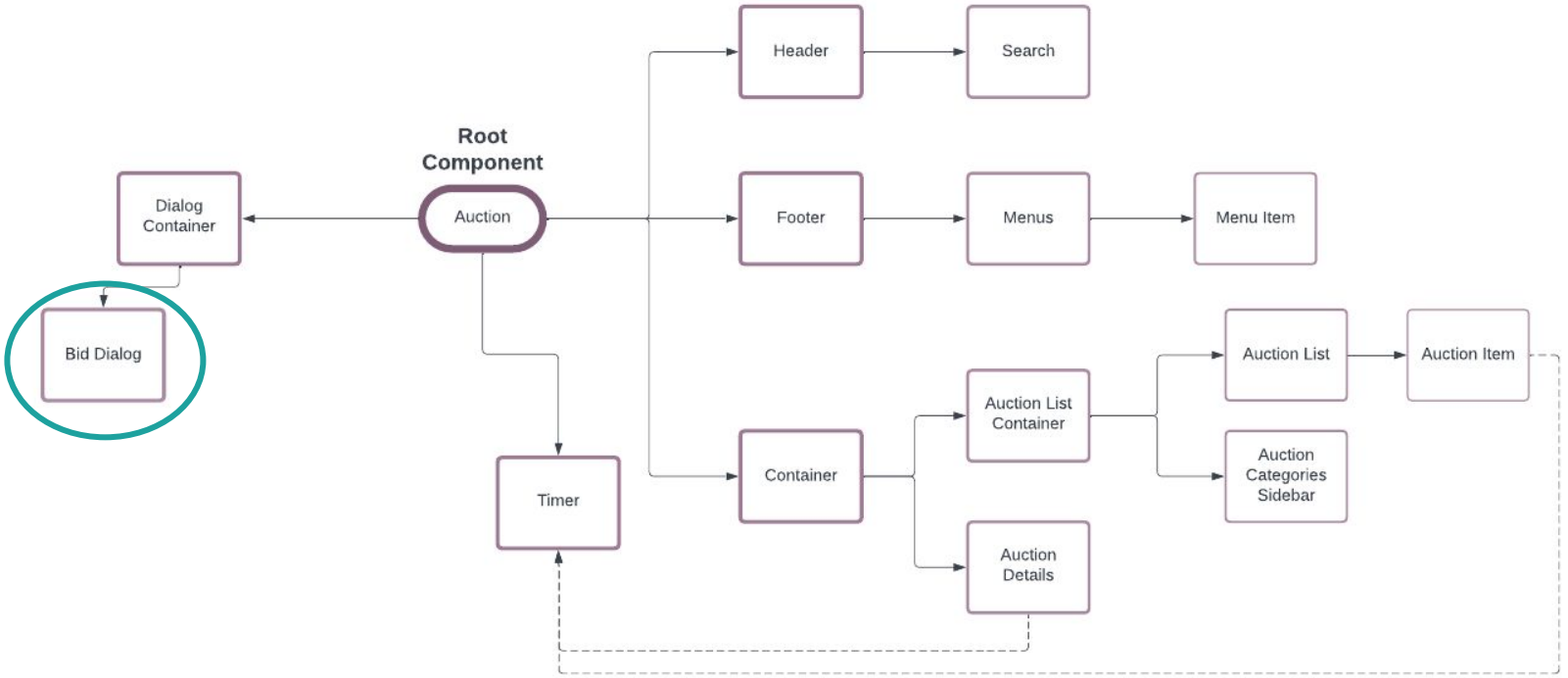
How Slots helped us!

```
<div class="modal-content" t-att-class="props.contentClass">
    <header t-if="props.header" class="modal-header">
        <t t-slot="header">
            <t t-call="web.Dialog.header">
                <t t-set="close" t-value="data.close"/>
            </t>
        </t>
      </header>
      <footer t-if="props.footer" class="modal-footer" style="order:2">
          <t t-slot="footer">
              <button class="btn btn-warning o-default-button">
                  <t>Ok</t>
              </button>
          </t>
      </footer>
      <main class="modal-body" t-attf-class="{{ props.bodyClass }}">
          <t t-slot="default"/>
      </main>
</div>
```

Dialog: Bid Dialog Component

Auction > Dialog Container > Bid Dialog

Architecture Progress



Bid Dialog Component

```
this.env.bus.trigger('add_dialog', {
  dialog: BidDialog,
  props: {
    title: 'Place Bid',
  },
})
```

Implementation from Auction Details Screen

Here, we can see couple of different concepts:

- EventBus Trigger
- Slots

```
<t t-name="web.BidDialog" owl="1">
    <Dialog title="'Place Bid'">
        <t t-set-slot="footer">
            <button class="btn btn-primary" t-on-click="onConfirmBid">Confirm</button>
        </t>
    </Dialog>
</t>
```

Props Validation

```
Dialog.props = {
    contentClass: { type: String, optional: true },
    bodyClass: { type: String, optional: true },
    footer: { type: Boolean, optional: true },
    header: { type: Boolean, optional: true },
    size: { type: String, optional: true, validate: (s) => ["sm", "md", "lg", "xl"].includes(s) },
    title: { type: String, optional: true },
    modalRef: { type: Function, optional: true },
    slots: {
        type: Object,
        shape: {
            default: Object, // Content is not optional
            header: { type: Object, optional: true },
            footer: { type: Object, optional: true },
        },
    },
};
Dialog.defaultProps = {
    contentClass: "",
    bodyClass: "",
    footer: true,
    header: true,
    size: "md",
    title: "Warning",
};
```

Translations

Translations

```
owl.whenReady(async () => {
  const translations = {}
  const hash = parseHash()

  if (hash.lang) {
    const terms = {
      'Search for the products you wanna bid!!':
        'Recherchez les produits auxquels vous souhaitez enchérir !!',
      All: 'Toute',
    }
    Object.assign(translations, terms)
  }

  const translateFn = (str) => {
    return translations[str] || str
  }

  const app = new App(Auction, {
    translateFn,
    // ...remaining options
  })
})
```

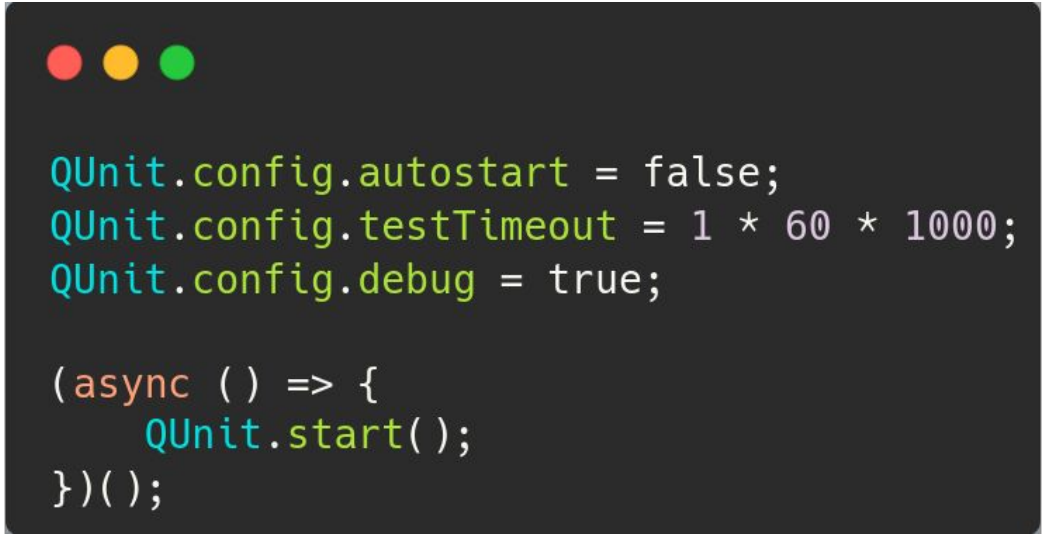
Setting up translation configuration

```
<span class="category_selected" t-translation="off">All</span>
```

Adding an exclusion

Let’s Test

Configuring QUnit Test Suite



```
QUnit.config.autostart = false;
QUnit.config.testTimeout = 1 * 60 * 1000;
QUnit.config.debug = true;

(async () => {
    QUnit.start();
})();
```

QUnit Test

```
QUnit.module('CategorySidebar', (hooks) => {
    hooks.before(async () => {
        // Some basic stuff
        bus = new EventBus();
        db = new DB();
        env = { bus, db, rpc };
    });
    hooks.afterEach(async () => {
        app.destroy();
    });
    QUnit.test('Simple UI test for CategorySidebar component', async function (assert) {
        assert.expect(1);
        const props = {categories: [{id: 1, name: 'Footwear'}, {id: 2, name: 'Topwear'}]};
        app = new App(AuctionCategorySidebar, {
            name: "CategorySidebar",
            env,
            templates,
            props
        });
        const categorySidebar = await app.mount(target);
        await nextTick();
        debugger;
        const elements = document.querySelectorAll('.auction_category');
        assert.ok(elements.length, 3, "should have o_content class");
    });
});
```

Helper Test Methods

```
function prepareTarget(debug = false) {
    return debug ? document.body : document.getElementById('qunit-fixture');
}

export async function nextTick() {
    await new Promise((resolve) => setTimeout(resolve));
    await new Promise((resolve) => requestAnimationFrame(resolve));
}
```

Integration with Odoo

JS - Logic

```
export class FieldIncrement extends Component {
    static template = "web.FieldIncrement";
    static props = {
        ...standardFieldProps,
        formatNumber: { type: Boolean, optional: true },
        inputType: { type: String, optional: true },
        step: { type: Number, optional: true },
        placeholder: { type: String, optional: true },
    };

    async onClickIcon(ev) {
        const amount = this.props.record.data[this.props.incrementField] || 10;
        const operator = ev.currentTarget.getAttribute('data-icon');
        const newValue = operator === '+' ? this.value + amount : this.value - amount;
        await this.props.record.update({ [this.props.name]: newValue });

    }
}
```

JS - Props Validation

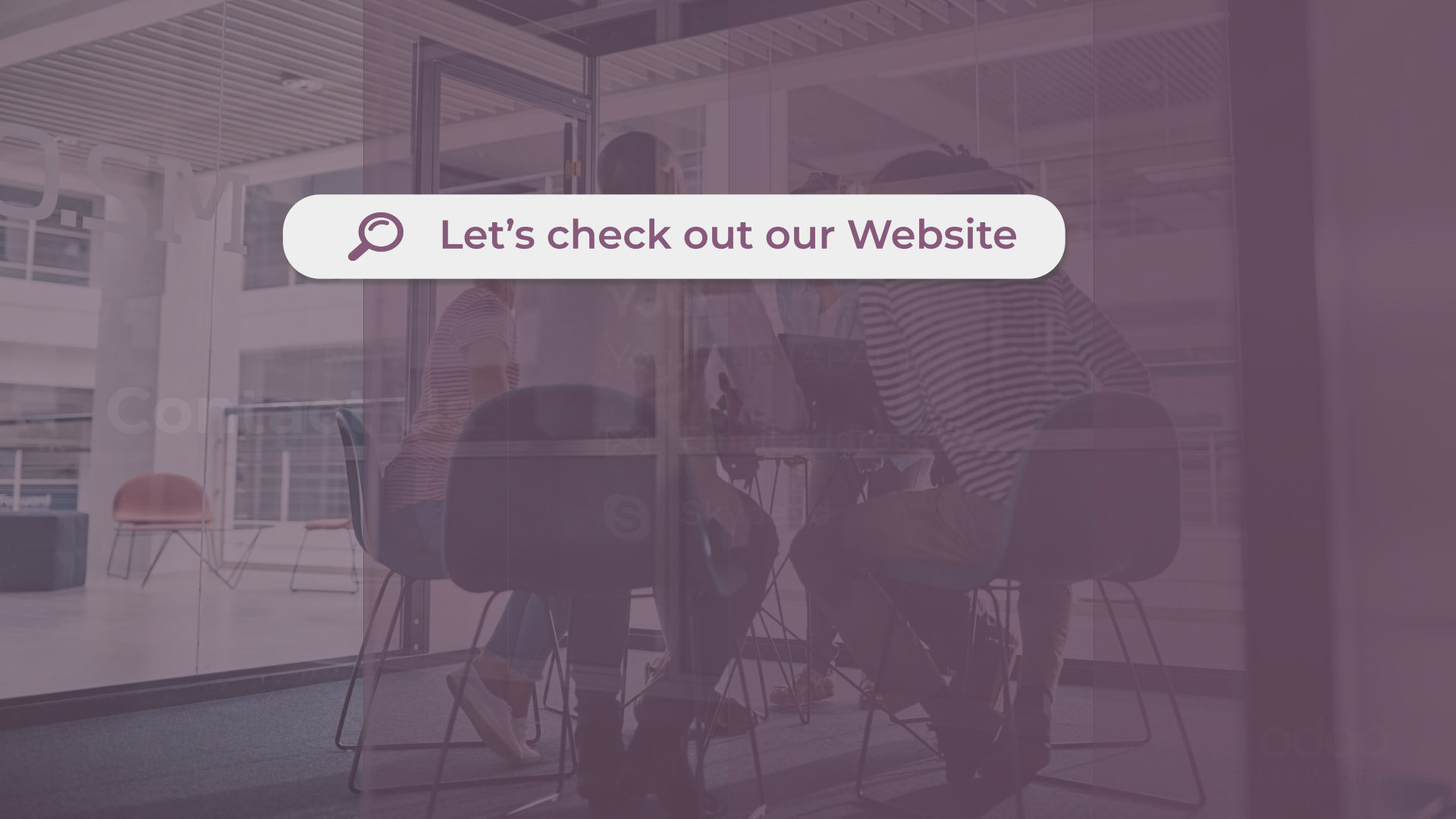
```
export const incrementField = {
    component: FieldIncrement,
    displayName: _lt("Increment"),
    supportedOptions: [
        {
            label: _lt("Increment number"),
            name: "increment_field",
            type: "string",
        },
    ],
    supportedTypes: ["integer"],
    isEmpty: () => false,
    extractProps: ({ attrs, options }) => ({
        incrementField:
        options?.increment_field !== undefined ? options.increment_field : false,
        formatNumber:
            options?.enable_formatting !== undefined ? Boolean(options.enable_formatting) : true,
        inputType: options.type,
        step: options.step,
        placeholder: attrs.placeholder,
    }),
};
```

XML - UI Implementation

```xml
<t t-name="web.FieldIncrement" owl="1">
    <span t-if="props.readonly" t-esc="formattedValue" />
    <div t-else="" class="o_field_increment">
        <span data-icon="+" t-on-click="onClickIcon"><i class="fa fa-plus"/></span>
            <input t-ref="incrementInteger" t-att-id="props.id" t-att-type="props.inputType"
                t-att-placeholder="props.placeholder" inputmode="numeric" class="o_input" t-att-step="props.step" />
        <span data-icon="-" t-on-click="onClickIcon"><i class="fa fa-minus"/></span>
    </div>
</t>
```

XML - Backend

```
<field name="current_bid_price" widget="increment"
    options="{'increment_field': 'bid_increment_amount'}"/>
```


Let’s check out our Website

odoo

Unveiling the Power of OWL

The Conclusion

Some Power Stats

Points	**Regular JS**	**OWL**
Development Time	3-4 days	**16 hours**
Learning Curve	Familiar	**Moderate**
User Interface (UI)	Manual styling	**Component-based**
Code Reusability	Low	**High**
Maintenance	Manual updates	**Easier updates**
State Management	N/A	**Built-in support**
Performance Optimization	Manual efforts	**Optimized by default**


Questions?
Book a free demo
with an expert.

Thank You

Checkout our documentations:
https://github.com/odoo/owl

Checkout our demo application:
https://github.com/somu-odoo/OXP