Improving Alpine.js Code With Design Patterns

Jisse Reitsma · Yireo

Meet Magento DE 2026

Jisse Reitsma

- Founder of Yireo
- Creator of Loki Extensions and Loki Checkout
- Board member of Magento Association

Alpine.js

- Lightweight, so great for performance
- Used in Hyvä, Nebula and Loki Theme Kit
- Easy-to-use and therefore often underestimated

PART 1

Component definitions

Alpine.js terminology

- Component (instance)
- Component definition

```
<div x-data='{"foo":"bar"}'>
    Foo: <span x-text="foo"></span>
</div>
```

Use `Alpine.data()` for component definitions

- Separation of JS logic and templating
- Easier to move JS-part to some other part of the page
- Because of CSP disallowing `unsafe-eval`

Use `Alpine.data()` for component definitions

TEMPLATE:

```
<div x-data="component1">
    Foo: <span x-text="foo"></span>
</div>
```

SCRIPT:

```
document.addEventListener('alpine:init', () => {
    Alpine.data('component1', () => {
        return {
            init() {
                console.log("component1 is being initialized")
            },
            foo: 'bar'
        }
    });
});
```

Naming your component

- Make sure your component name is unique
- Do not start with `init`
- Suggestions:
  - `camelCase` or `PascalCase`
  - Include the vendor name
- With Loki Extensions, block name becomes component name

```
<block name="loki-checkout.example" />
```

```
Alpine.data('LokiCheckoutExample', () => ({}));
```

Reusing component definitions

TEMPLATE:

```
<div x-data="component1"></div>
<div x-data="component1"></div>
<div x-data="component1"></div>
```

SCRIPT:

```
document.addEventListener('alpine:init', () => {
    Alpine.data('component1', () => ({}));
});
```

PART 2

Component types & partials

Component types

TEMPLATE:

```
<div x-data="component1"></div>
<div x-data="component2"></div>
<div x-data="component3"></div>
```

SCRIPT:

```
const componentType = {foo: "bar"};
document.addEventListener('alpine:init', () => {
    Alpine.data('component1', () => ({name: "component1", ...componentType}));
    Alpine.data('component2', () => ({name: "component2", ...componentType}));
    Alpine.data('component3', () => ({name: "component3", ...componentType}));
});
```

Alpine.js terminology

- Component (instance)
- Component definition
- **Component type**
- **Component partial**

Component partials (Loki)

```
const LokiComponentType = {
    ...LokiLoadDataComponentPartial,
    ...LokiLoadInitMethodsComponentPartial,
    ...LokiLoadDestroyMethodsComponentPartial,
    ...LokiLoaderComponentPartial
};

const LokiFieldComponentType = {
    ...LokiComponentType
};

document.addEventListener('alpine:init', () => {
    Alpine.data('LokiFieldComponent', () => ({
        ...LokiFieldComponentType
    }));
});
```

PART 3

Sharing state with stores

Alpine.store()

```
document.addEventListener('alpine:init', () => {
    Alpine.store('foobar', {});
});
```

Components store

```
Alpine.store('components', {
    components: {},
    componentNames() {
        return Object.keys(this.components);
    },
    add(componentId, component) {
        this.components[componentId] = component;
    },
    get(componentId) {
        return this.components[componentId];
    }
});

Alpine.data('component1', () => {
    return {
        init() {
            Alpine.store('components').add('component1', this);
        },
    }
});
```

```
Alpine.store('components').componentNames();
```

Component-store-aware component type

```
const ComponentAwareType = {
    init() {
        Alpine.store('components').add(this.name, this);
    }
};

Alpine.data('component1', () => ({
    ...ComponentAwareType,
    name: 'component1',
}));
```

> `init()` can only be defined once, but you could create an `init()` that calls upon any other method starting with `init*` (like with Loki Components).

Referring from one component to another

```
Alpine.data('component2', () => {
    return {
        init() {
            Alpine.store('components').add('component2', this);
        },
        foo() {
            return Alpine.store('components').get('component1')?.foo;
        }
    }
});
```

> Note that `component2:foo` is already reactive. Any change of `component1:foo` leads to a change of `component2:foo` (without `$watch` magic).

Magento KnockoutJS / UiComponent magic

- ✅ `import`: done
- ⏱️ `export`: maybe
- ✗ `link` (combination of `import` and `export`): nah

Adding a new prop to an existing component

```
Alpine.data('component2', () => {
    return {
        foo() {
            return Alpine.store('components').get('component1')?.foo2;
        }
    }
});
```

And then you need to update the template

Pushing with `Alpine.$data`

```
Alpine.data('component2', () => {
    return {
        init() {
            setTimeout(() => {
                const element = document.querySelector('[x-data="component1"]');
                const component = Alpine.$data(element);
                component['foo'] = 'testing';
            }, 300);
        },
    }
});
```

> Note that `Alpine.$data()` requires the given component to be already instantiated.

PART 4

Extending components

Hyvä `initHeader` override

```
<div id="header" x-data="initHeader"></div>
```

```
function initHeader() { return {}; }
```

OVERRIDE THE GLOBAL FUNCTION:

```
window.originalInitHeader = window.initHeader;
window.initHeader = function() {
    const originalData = window.originalInitHeader();
    return {
        ...originalData,
        foo: "bar"
    }
}
```

Hyvä `initHeader` with a `Proxy`

```
window.initHeader = new Proxy(window.initHeader, {
    apply(target, thisArg, args) {
        console.log('Before original');
        const result = Reflect.apply(target, thisArg, args);
        console.log('After original');
        return result;
    }
});
```

Creating a `Proxy` for `Alpine.data()`

```
Alpine.data = new Proxy(Alpine.data, {
    apply(target, thisArg, args) {
        const [name, factory] = args;
        console.log('Registering:', name);
        return Reflect.apply(target, thisArg, args);
    }
});
```

Hooking this proxy to `ComponentExtender`

```
Alpine.data = new Proxy(Alpine.data, {
    apply(target, thisArg, args) {
        const [name, factory] = args;
        if (typeof factory !== 'function') {
            return Reflect.apply(target, thisArg, args);
        }

        const extendedFactory = new Proxy(factory, {
            apply(target, thisArg, factoryArgs) {
                const component = Reflect.apply(
                    target, thisArg, factoryArgs
                );

                return ComponentExtender.extend(
                    name, component, factoryArgs
                );
            }
        });

        return Reflect.apply(target, thisArg, [
            name,
            extendedFactory
        ]);
    }
});
```

`ComponentExtender` logic

```
const ComponentExtender = {
    extenders: [],
    addExtender(extenderName, extenderCallback) {
        this.extenders.push({
            name: extenderName,
            callback: extenderCallback,
        });
    },
    extend(componentName, component, args) {
        this.extenders.forEach((extender) => {
            extender.callback(componentName, component, args);
        });

        return component;
    }
};
```

Example extender

```
ComponentExtender.addExtender(
    'component1Extender',
    (componentName, component, args) => {
        if (componentName !== 'component1') {
            return;
        }

        component.foo = 'component1-override';
        component.customProperty = 'Hello World';
    }
);
```

Ideas

- ✅ Select all with prop X (like invalid field components in Loki Checkout)
- ✗ Automatically add any Alpine.js component to a global store
- ⏱️ Proxy specific setters to make them more secure
- ⏱️ Alpine store with services for usage in components (service container)

Loki Checkout

- Fastest checkout under Luma, Hyvä and Nebula
- 299 modules with all tests in the green
- Exciting new B2B Loki Extensions for 2027
- Working towards a free plan (Q1 2027)

Questions

Thank you!

Jisse Reitsma · Yireo