Welcome to dreamOlé

redk

Get further, together

FINANCIALFORCE

nts

COPADO

omega

S4G Consulting

QualityClouds

Prodly

PDF BUTLER

riva

<ISDICRM>

Maintaining social distancing between LWCs: a **Redux** solution

Ana Belén Pelegrina (@abpelegrina)
Software Engineer
Paloma de las Cuevas (@unintendedbear)
Sr Software Engineer

Our Solutions

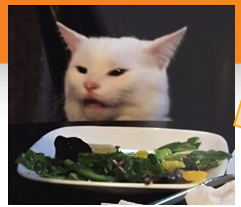
Accounting
Rev Rec
Spend
Management

Order, Supply,
Inventory &
Contract
Management
Advanced Quoting

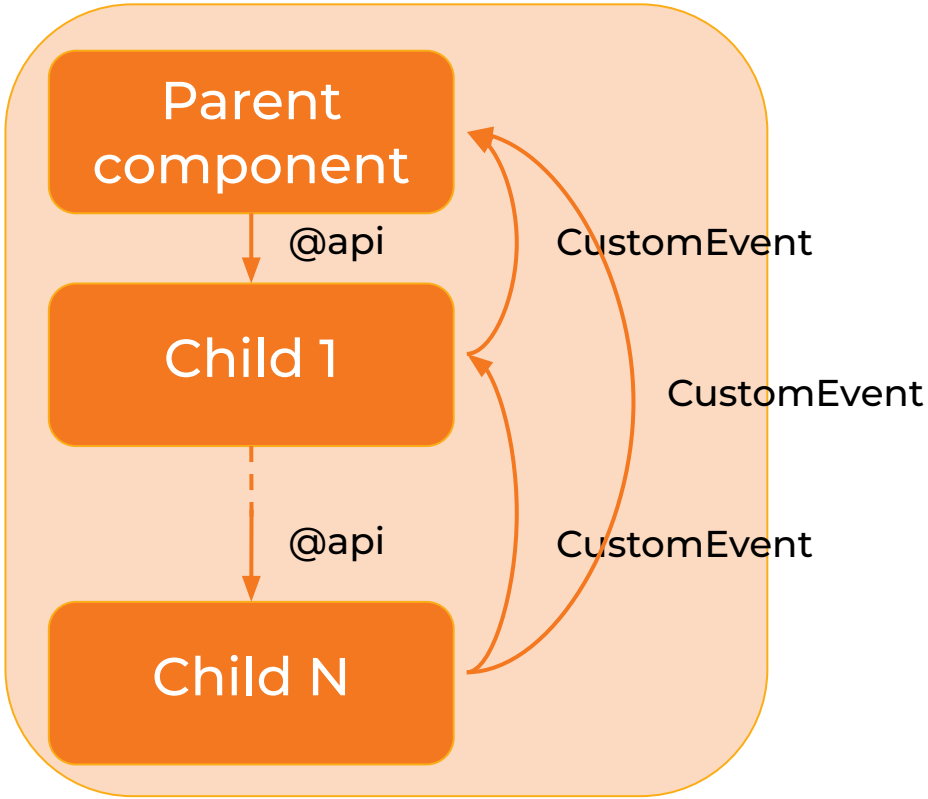
Billing
Media Billing
Contract
Management

Project Management
Accounting Resource
Management
Time & Expense
Management

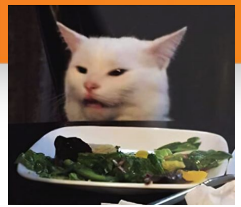
Why?



But LWC is pawesome!



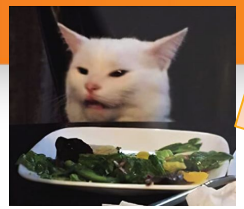
Why?



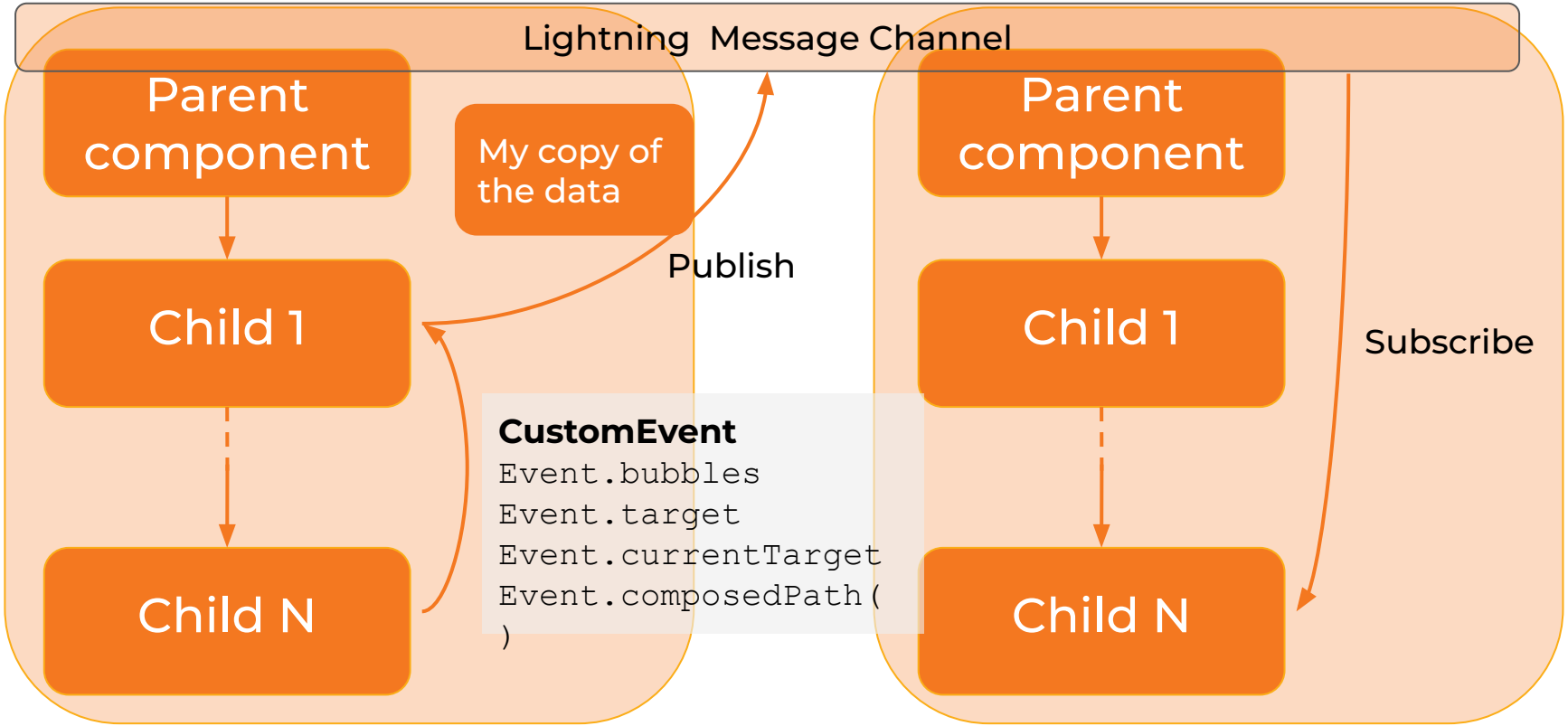
Custom Events, Lightning Messages, `pubsub`...

- Communication down through attributes/@api variables or calling the public methods.
- Communication up trough events: standard and custom. The propagation can be controlled.
- Communication between hierarchies: **lightning message service**.
- LMS is just that, a channel.
- The components still need to store and manage the state of the context.

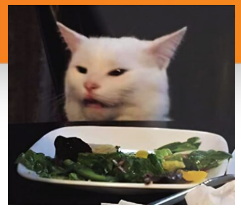
Why?



So WHAT


Lightning Message Channel
Parent component
My copy of the data
Child 1
Child N
Publish
Parent component
Child 1
Child N
Subscribe
CustomEvent
Event.bubbles
Event.target
Event.currentTarget
Event.composedPath()

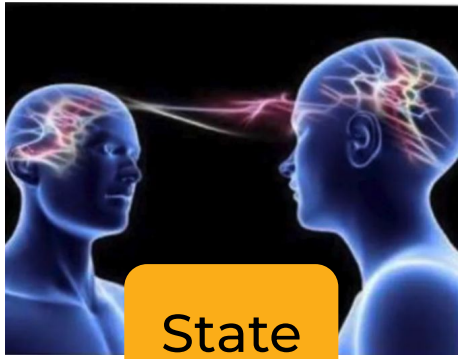
Why?



Component

Component

Component



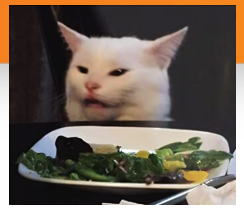
State

Component

Component

Component

Why?



Component

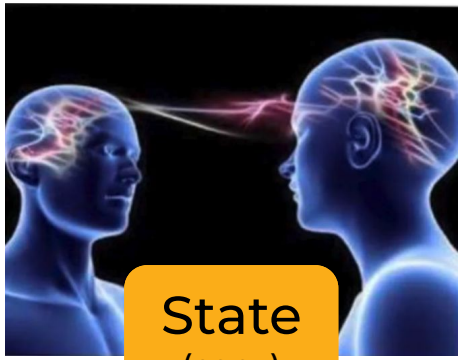
Component

Component

Action

type,
payload

Reducer



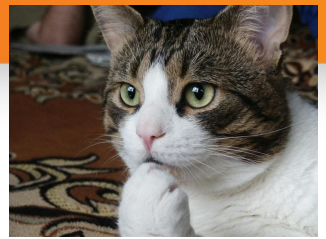
State
(copy)

Component

Component

Component

Why?



Component

Component

I'm ok

Hmm
I need
this...

Component

New State

I'm ok

Component

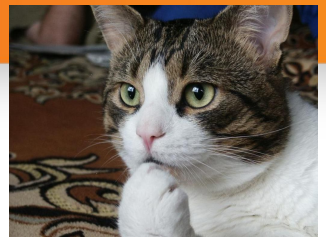
Oh, I'll
take
this...

Component

I'm ok

Component

Why?



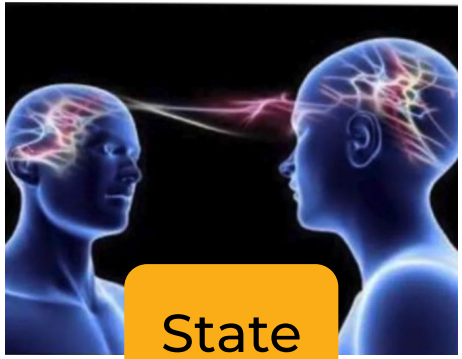
Component

Component

Component

Henlo frends

Component



State

Component

Component

Component

Three principles

Single Source of Truth

State is read-only

Changes are made with pure functions

Why?

S.O.L.I.D.

Redux embraces these principles

Scalability

Easy to add new components
+
Avoid unnecessary rendering

Knowledge

Sharing a context is not enough, all you need is ~~love~~ a state management system

Just because

Think carefully about your situation, knowing when NOT to use Redux

Dev process

In some simple
steps

Setup

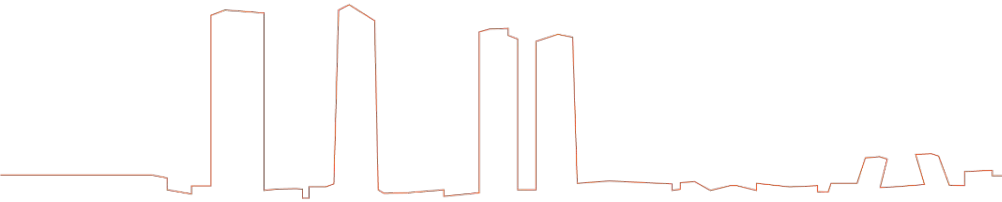
What do you need to start reduxing?

Before we start coding we need to:

- Create a SDFX project and scratch org.
- Install the REDUX package:

```
sfdx force:package:install --package 04t2v000007GTFkAAO
```

- Define the custom object(s) & Apex controller class(es).
- Create an application and configure it. For example **Cat Tracker**.



Let’s start reduxing!

Store | Constants | Reducers | Actions | Connect

First we’ll need a component that contains the Redux store and initializes it. This component will wrap our application.

```
<template>
   <c-provider oninit={initialize} store={store}>
        . . .
   </c-provider>
</template>
```

Let’s start reduxing!

Store | Constants | Reducers | Actions | Connect

```
export default class CatTrackAppContainer extends LightningElement {
    @api store;
    initialize() {
        let logger;
        if (ENABLE_LOGGING)
            logger = createLogger({ duration: true, diff: true });
        const combineReducersInstance = combineReducers(reducers);
        this.store = createStore(combineReducersInstance, logger);
    }
}
```

Let’s start reduxing!

Store | Constants | Reducers | Actions | Connect

Now, we need to define our constants

```
export const REGISTER_CAT = "REGISTER_CAT";
export const CAT_ADOPTED = "CAT_ADOPTED";
export const CAT_STERILISED = "CAT_STERILISED";
export const CAT_VACCINATED = "CAT_VACCINATED";
export const INITIALIZE_APP = "INITIALIZE_APP";
```

Let’s start reduxing!

Store | Constants | Reducers | Actions | Connect

It’s time to define the reducers. First, let’s define the initial state.

```
const initialState = {
   allIds: [],
   byIds: {}
};
```

Let’s start reduxing!

Store | Constants | Reducers | Actions | Connect

Now, we’re going to add the first reducer.

```
const catTracker = (state = initialState, action) => {
   switch (action.type) {
       case REGISTER_CAT: {
           return {
               ...state,
               allIds: [...state.allIds, action.payload.id],
               byIds: {
                   ...state.byIds, [payload.id]: { ...action.payload }
               }
           };
       }
   }
};
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

We are going to start with the action for registering cats

```
export const register = (name, gender, age, sterilized, vaccinated) => {
   return (dispatch) => {
       registerCat({
           name: name, gender: gender,age: age, sterilized: sterilized, vaccinated: vaccinated
       }).then((result) => {
               dispatch( {type: REGISTER_CAT,  payload: JSON.parse(JSON.stringify(result))} );
       }).catch((error) => {
               console.error(error);
       });
   };
};
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

First, let’s create a component to register cats

```
import { LightningElement, track } from "lwc";
import { Redux } from "c/lwcRedux";
import actions from "c/catTrackerActions";

export default class CatCard extends Redux(LightningElement) {
   . . .
}
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

Define an state for the component and an initial empty state.

```
const emptyState = {
    name: "",
    age: null,
    gender: "Unknown",
    vaccinated: false,
    sterilised: false
};
export default class CatCard extends Redux(LightningElement) {
    @track
    state = { ...emptyState };
    . . .
}
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

Define **mapStateToProps** and **mapDispatchToProps**

```
. . .
mapDispatchToProps() {
    return { register: actions.registerCat.register };
}

mapStateToProps(state) {
    // nothing to do here yet, maybe validation someday
}
. . .
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

Here is an example of how to use one of the actions mapped in mapDispatchToProps.

```
handleClick() {
    if (this.state.name) {
        this.props.register(
            this.state.name,
            this.state.gender,
            this.state.age,
            this.state.vaccinated,
            this.state.sterilised
        );
        this.state = { ...emptyState };
    }
}
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

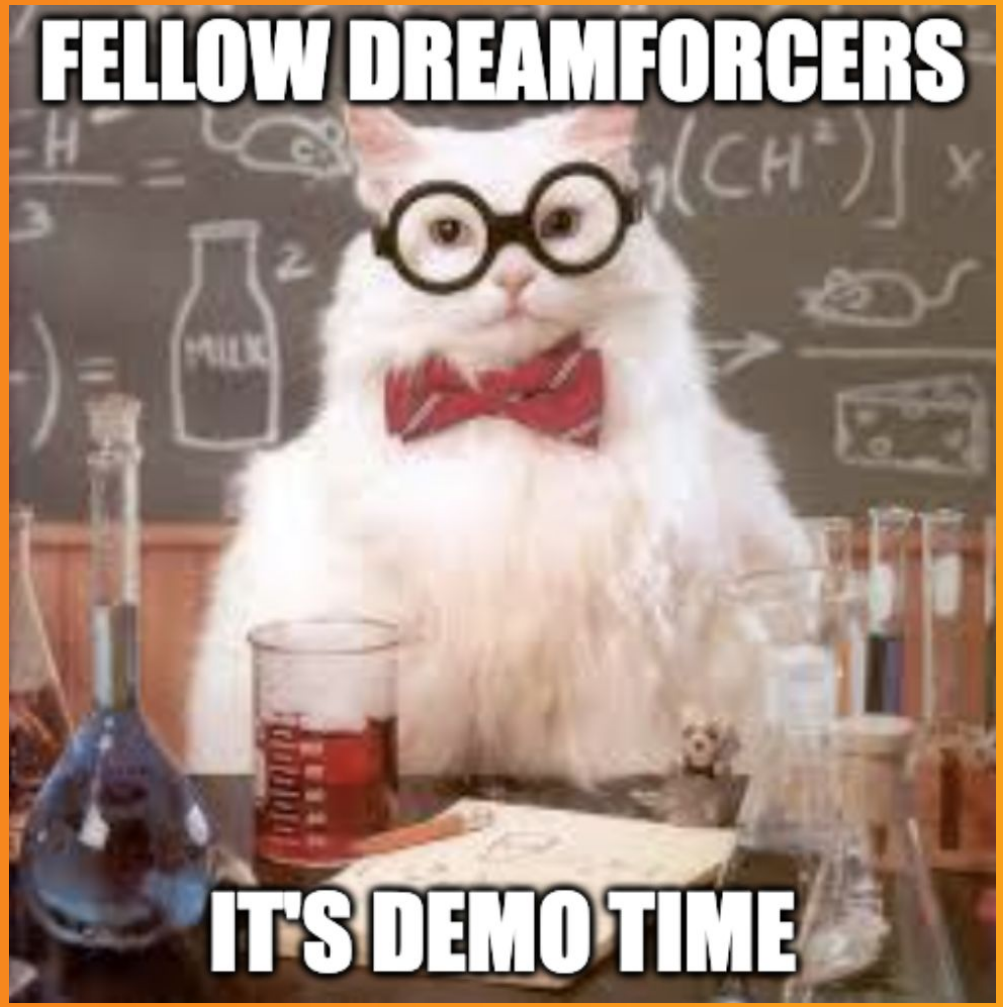
Now we need a component to show all of our cats.

```
<template>
    <template if:true={hasRecord}>
        <template for:each={props.allIds} for:item="id">
            <c-cat key={id} record-id={id}></c-cat>
        </template>
    </template>
    <template if:false={hasRecord}>
        <div class="slds-m-top_medium"><b>No cats rescued yet</b></div>
    </template>
</template>
```

Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

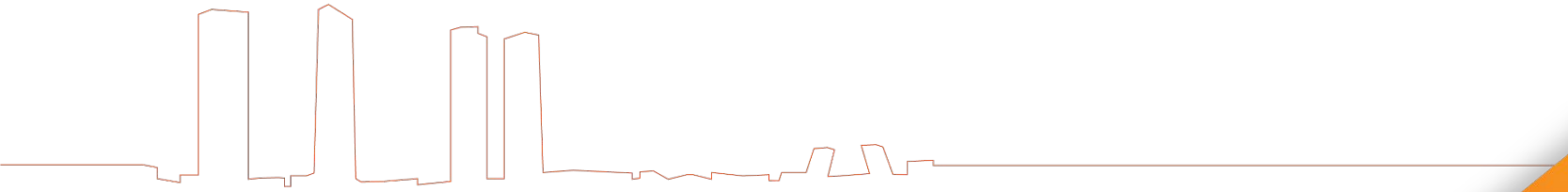
```
export default class CatList extends Redux(LightningElement) {
    mapDispatchToProps() {
        return { initialize: actions.registerCat.initialize };
    }
    mapStateToProps(state) {
        return { allIds: state.catTracker.allIds };
    }
    connectedCallback() {
        super.connectedCallback();
        this.props.initialize();
    }
…
```


FELLOW DREAMFORCERS
IT'S DEMO TIME

Keep reducing!

Remember these steps for new functionalities

1. Add new constants.
2. Add new reducers.
3. Add new actions.
4. Create component(s):
    - Implement mapDispatchToProps and mapStateToProps.
5. Add new component to the wrapper component.
6. Profit!



References

- Salesforce documentation for LWC
- LWC-Redux library documentation
- Understanding Redux
- Writing Logic with Thunks
- Why the context is not enough (sometimes)
- Applying S.O.L.I.D. to React
- CatTracker in Github

Further reading

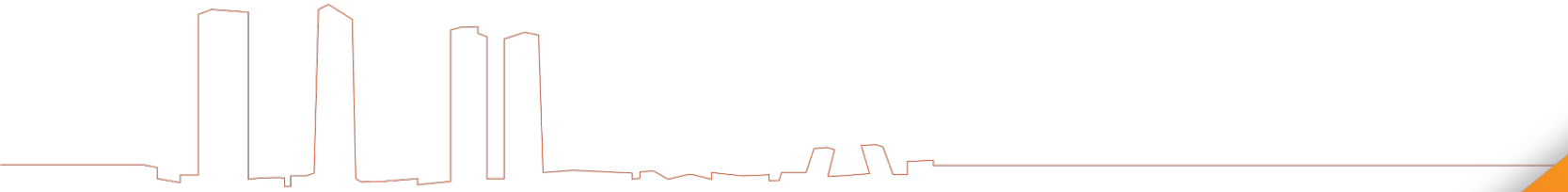
- Best practices for Redux
- Redux-saga: An intuitive Redux side effect manager.
- Thunk vs Sagas

Q&A


redk
Get further, together
FINANCIALFORCE
nts
COPADO
omega crm
S4G Consulting
QualityClouds
Prodly
PDF BUTLER
riva
<ISDICRM>

This is (not) the (only) way

- Of course, there are multiples ways in which you can use redux+lwc, this is just one of them.

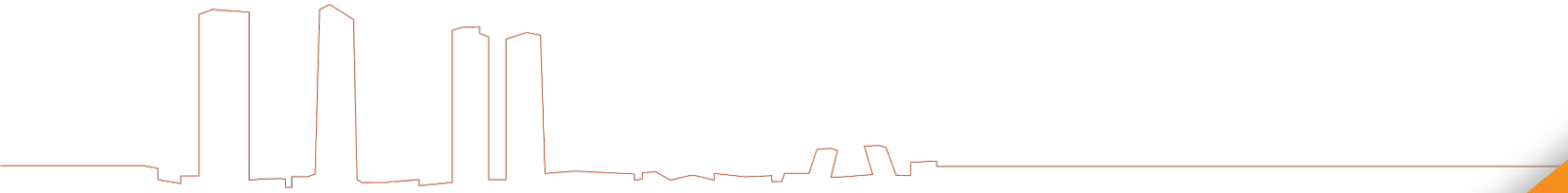


Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

Create a new component that will contain all our reduced components

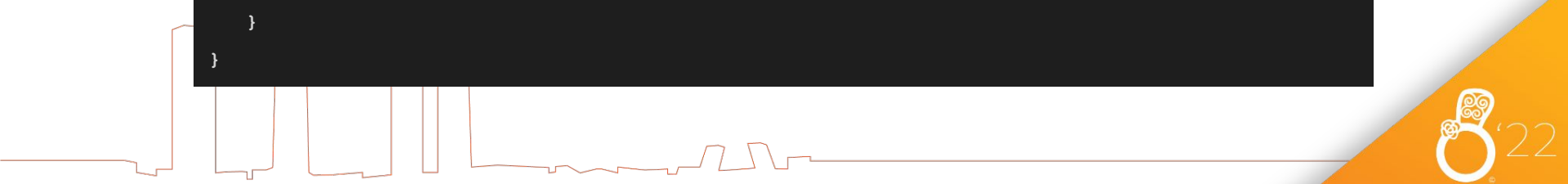
```
<template>
    <lightning-card title="Cat tracker">
        <div class="container">
            <div class="slds-m-around_medium">
                <c-cat-card></c-cat-card>
            </div>
        </div>
    </lightning-card>
</template>
```



Let's start reduxing!

Store → Constants → Reducers → Actions → Connect

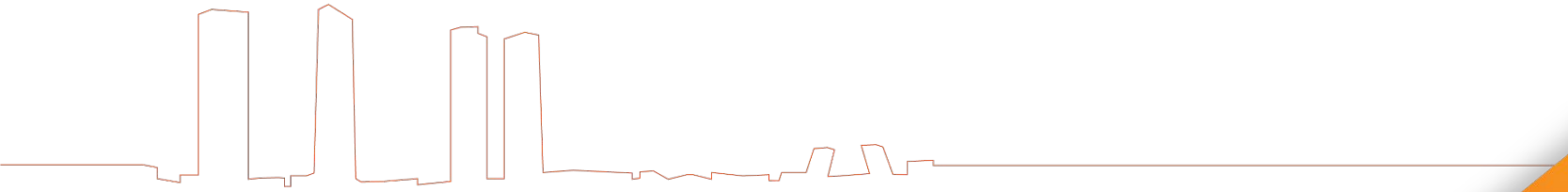
```
import { LightningElement } from "lwc";
import { Redux } from "c/lwcRedux";
import actions from "c/catTrackerActions";

export default class CaTracker extends Redux(LightningElement) {
    connectedCallback() {
        super.connectedCallback();
        this.props.initialize();
    }

    mapDispatchToProps() {
        return { initialize: actions.registerCat.initialize };
    }
}
```



Let’s start reduxing!

Store → Constants → Reducers → Actions → Connect

Let’s see how it looks...



@abpelegrina

Software Engineer
Cat Person &
Cheesecake aficionada

Meow world!

Nice to meet you
hoomans

Sr Software Engineer
Also a Cat Person

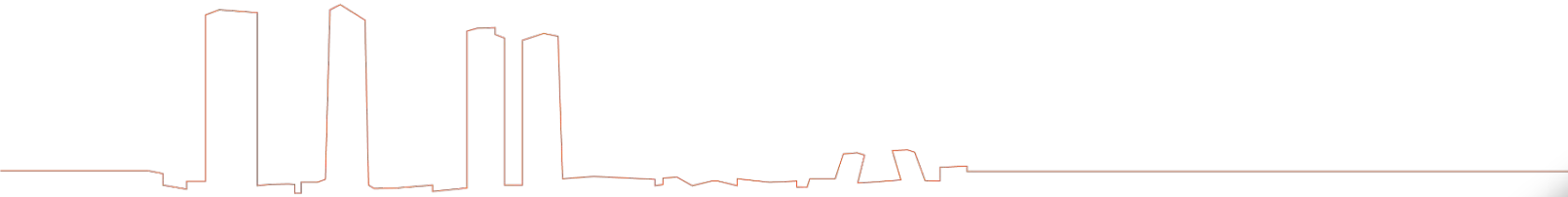
@unintendedbear

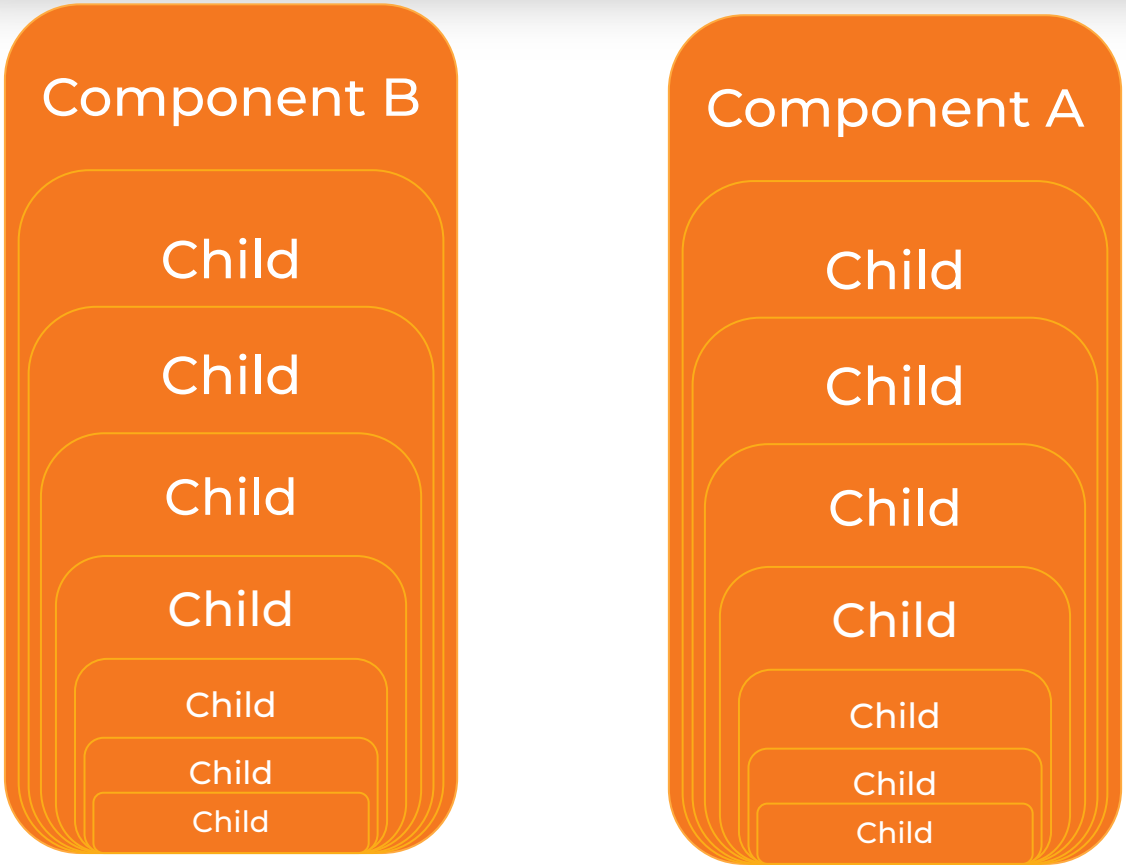
Components

Let’s start reduxing!

Now we can create our LWC component! First let’s create the main component in our application.

- This component extends **Redux(LightningComponent)**.
- Add this component inside <c-provider> in the component we created in the previous step.



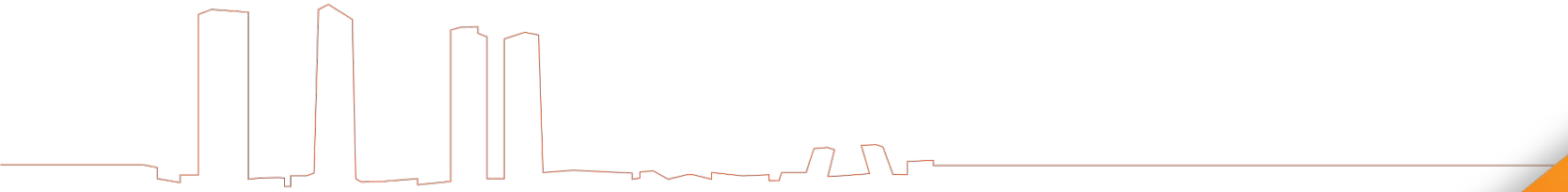

Component B
Child
Child
Child
Child
Child
Child
Component A
Child
Child
Child
Child
Child
Child

Components

Let’s start reduxing!

Now we can create our first component! We can start with a card that let us add a new instance of, say, a Cat.

- This component will extend **Redux(LightningComponent)**.
- Add all the fields for your custom object and a save button.
- Define an state for the component.
- Add this to the container component.

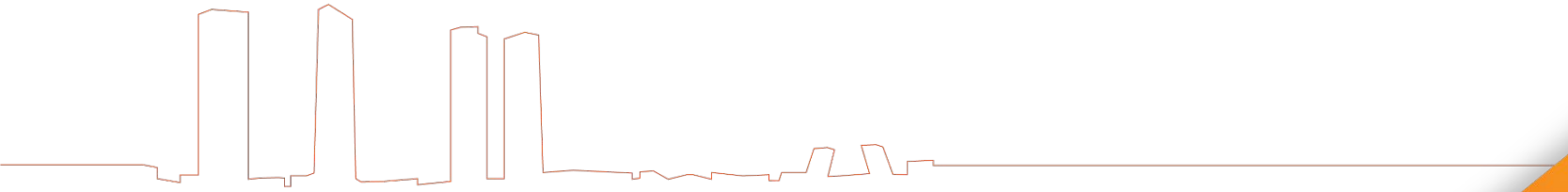


Redux stuff

Let's start reduxing!

Now we need to define actions and constants for our app.

- Create the constant associated with the action. For instance, **REGISTER_CAT**.
- Create the redux action to register a new instance:
  - Add a new LWC that will contain the action.
  - Create a function that will call the an apex method that will perform the action.

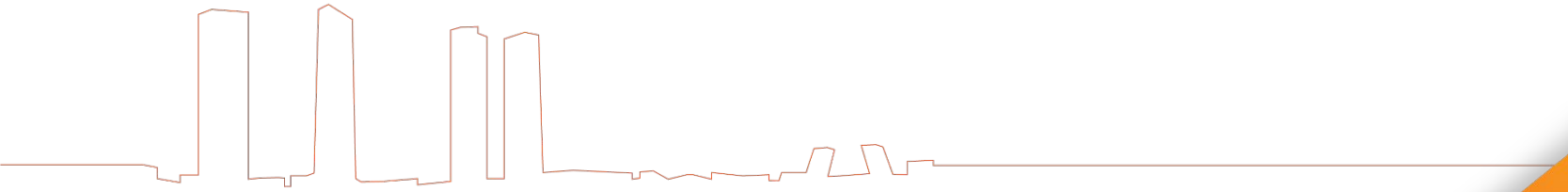


Redux stuff

Let’s start reduxing!

Now we need to define reducers.

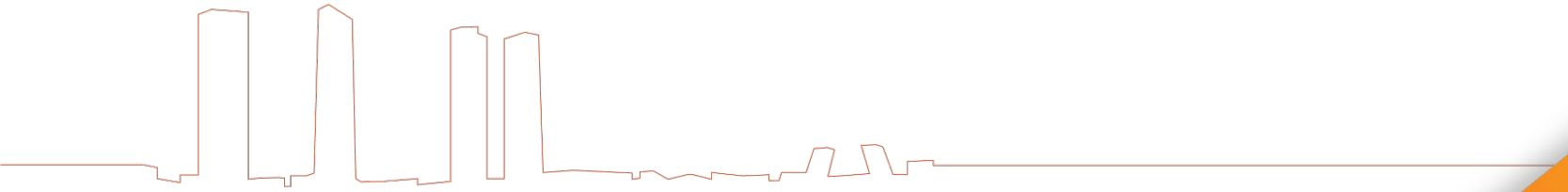
- Add a new LWC to include the reducers.
- Define the initial state
- Add a new reducer function and add the action defined in the previous step.



Setup

Development process schema

- Create a Redux store
- Subscribe to updates.
- Inside the subscription callback:
  - Get the current store state
  - Extract the data needed by this piece of UI
  - Update the UI with the data
- If necessary, render the UI with initial state
- Respond to UI inputs by dispatching Redux actions

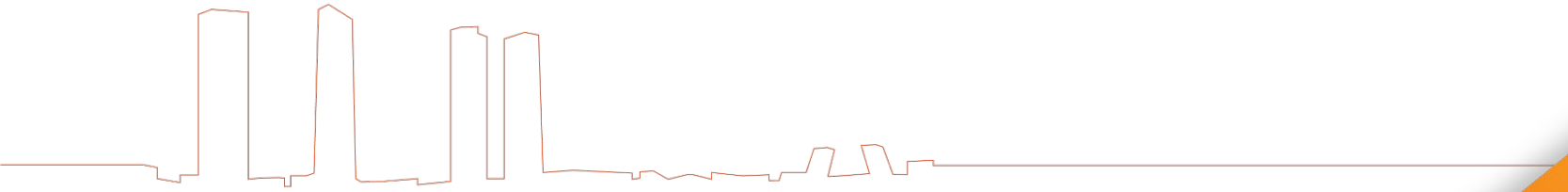


Redux stuff

Let’s start reduxing!

Now we have to reduxify our components.

- For the main component, implement connectedCallback() and call its parent method.

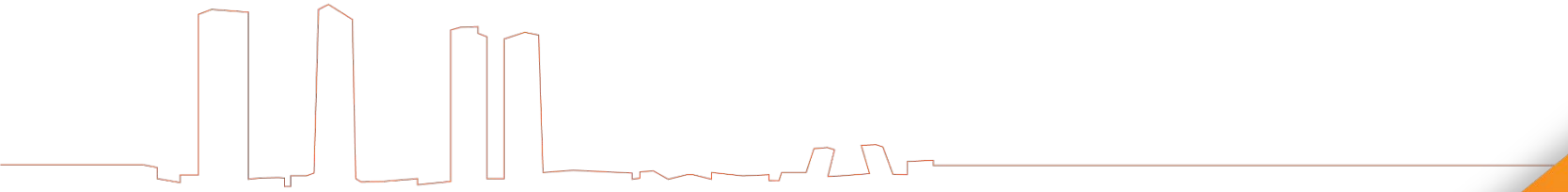


Redux stuff

Let’s start reduxing!

Now we have to reduxify our components.

- For the create component, implement mapDispatchToProps() and return the action we created

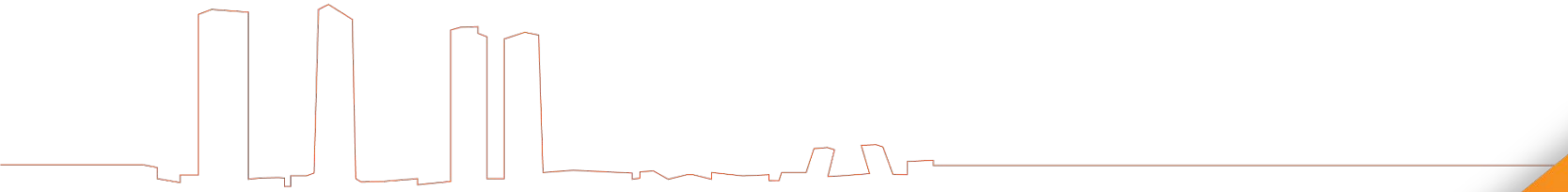


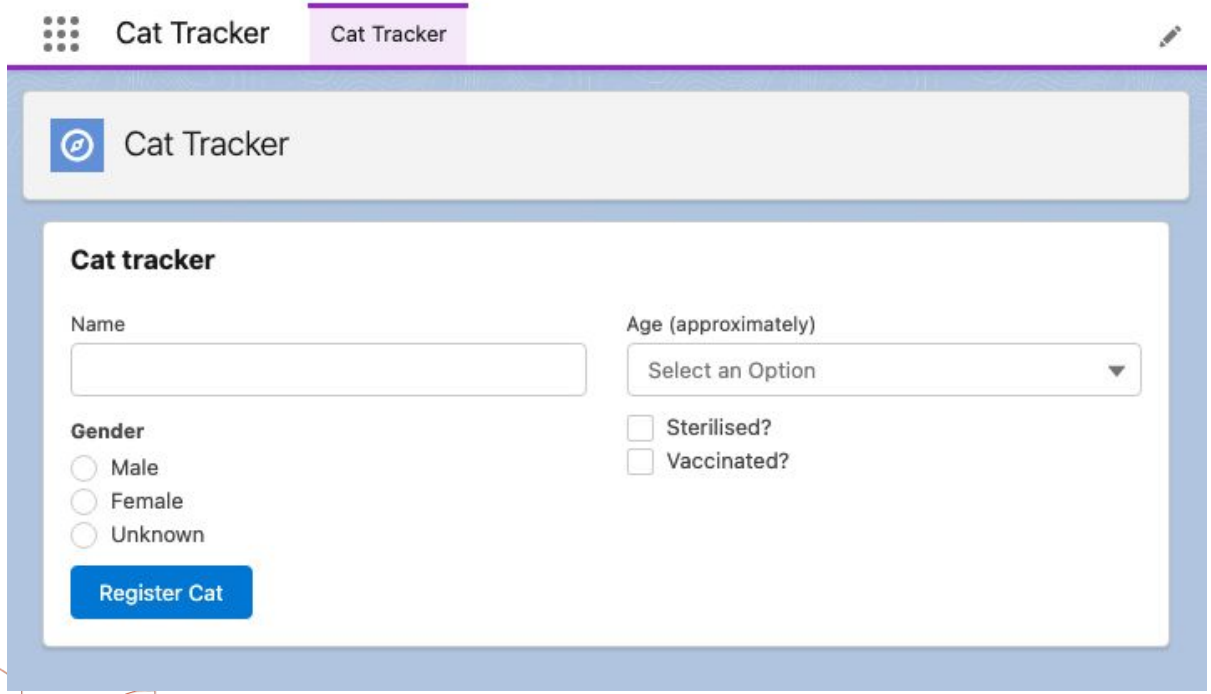
Redux stuff

Let’s start reduxing!

Now we have to reduxify our components.

- Finally for the store component, we need to handle the oninit event.
  - In the handler combine all reducers.
  - Initialize the store




Cat Tracker
Cat Tracker
Cat Tracker
Cat tracker
Name
Age (approximately)
Select an Option
Gender
Male
Female
Unknown
Sterilised?
Vaccinated?
Register Cat

Slide Title

Title

Enter slide information here

Title

Enter slide information here

Title

Enter slide information here

Title

Enter slide information here

Slide Title

Store

Constants

Reducers, Actions, Connect

Title

Enter slide information here

Title

Enter slide information here

Title

Enter slide information here

Store → Constants → Reducers → Actions → Connect

Slide Title

Subtitle

This placeholder can hold text and below bullet points if required. You can also add charts, smartart or media.

- Item 1
- Item 2
- Item 3

Slide Title

Subtitle

This placeholder can hold text and below bullet points if required. You can also add charts, smartart or media.

- Item 1
- Item 2
- Item 3

Slide Title

Subtitle

This placeholder can hold text and below bullet points if required. You can also add charts, smartart or media.

- Item 1
- Item 2
- Item 3

Slide Title

Subtitle

This placeholder can hold text and below bullet points if required. You can also add charts, smartart or media.

- Item 1
- Item 2
- Item 3