

# YOU DON'T KNOW MOBX STATE TREE

Milan | 29-30 November 2018

Max Gallo | @\_maxgallo

{codemotion}

# HI I'M MAX GALLO

About me: 🍕💻🇬🇧🎵🎸📸✈️✍️

Principal Engineer @ DAZN



twitter: @\_maxgallo  
more: maxgallo.io



# AGENDA

- » Part One: MobX
- » Part Two: MobX State Tree
- » Part Three: Designing a Reactive Project



# Mu



- » State Management Library
- » Helps decoupling View from Business Logic
- » Reactive Programming
- » Flexible / Unopinionated

# MOBX OBSERVABLES & REACTIONS

```
import { observable, autorun} from 'mobx';

const album = observable({
  title: 'Californication',
  playCount: 0,
});

autorun(() => console.log(`New play count: ${album.playCount}`))
// New play count: 0

album.playCount = 1; // New play count: 1

album.playCount = 24; // New play count: 24
```

# MOBX COMPUTED VALUES

```
import { observable, autorun, computed} from 'mobx';

const album = observable({
  title: 'Californication',
  playCount: 0,
}):

const all = computed(() => album.title + album.playCount);

autorun(() => console.log(all))
// Californication0

album.playCount = 1;          // Californication1
album.title = 'OkComputer'; // OkComputer1
album.playCount = 24;        // OkComputer24
```

# MOBX RECAP

## Observable state

Mutable Application State

## Computed Values

Automatically derived values, lazily evaluated

## Reactions

Side effects like autorun or updating a React component

# MOBX ❤️ REACT

```
const album = observable({
  title: 'Californication',
  playCount: 0,
});  
  
@observable
class PlayCount extends React.Component {
  render() {
    return album.playCount;
  }
}  
  
album.playCount++      // ----> React Render
album.playCount = 9    // ----> React Render
```





# MOBX STATE TREE

- » Powered by MobX
- » Runtime typed Application State
- » Opinionated & ready to use
- » Relies on the concept of Trees (Stores)

# WHAT'S A TREE/STORE?

```
import { types } from 'mobx-state-tree';

const AlbumStore = types
  .model('Album', {
    title: types.string,
    rating: types.integer,
  })
  .views(self => ({
    get isGood() { // mobx computed
      return self.rating >= 7
    }
  }));
}

const okComputer = AlbumStore.create(
  { title: 'Ok Computer', rating: 8}
);

console.log(okComputer.isGood); // true
```

# MOBX STATE TREE STORES



```
1 const AlbumStore = types.model('Album',...)
2
3
4 const MusicLibraryStore = types
5   .model('MusicLibrary', {
6     albums: types.array(AlbumStore), // -----> Model
7   })
8   .views(self => {
9     get goodAlbums() { // -----> View
10       return self.albums.filter(x => x.isGood)
11     }
12   })
13   .actions(self => {
14     addAlbum(album) { // -----> Action
15       self.albums.push(album);
16     },
17   });
18
19 const musicLibrary = MusicLibraryStore.create(
20   { albums: [ { title: 'Rumours', rating: 9 } ] }
21 );
22
23 musicLibrary.addAlbum({ title: 'Harvest', rating: 8 });
24 musicLibrary.addAlbum({ title: 'Ten', rating: 9 });
```

## Model

- » Mutable observable state
- » Runtime type information
- » Could contain other trees

## Views

MobX computed values

## Actions

The only way to update the model

**MOBX STATE TREE STORES**

# **DEEP DIVE**



- » Mutable and Immutable (Snapshots, Time Travelling)
- » Composition
- » Lifecycle Methods
- » Dependency Injection



# MOBX STATE TREE STORES

# DEPENDENCY

# INJECTION

```
1 import { getEnv, types } from 'mobx-state-tree';
2
3 const MusicLibraryStore = types
4   .model('MusicLibrary', {
5     albums: types.array(types.string),
6   })
7   .actions(self => ({
8     downloadAlbums() {
9       getEnv(self).getAlbumsFromBackend()
10      .then(albums => self.albums = albums);
11    }
12  }));
13
14 const getAlbumsFromBackend = function() {
15   /** Fetching albums from Backend **/
16 }
17
18 const musicLibrary = MusicLibraryStore.create(
19   { albums: ['Who\'s Next'] }, // Initial state
20   { getAlbumsFromBackend } // Environment
21 );
```

- » Inject anything
- » Environment is shared per tree
- » Useful for testing

**MOBX STATE TREE**

**HOW TO CONNECT  
THE STORES  
WITH THE VIEW?**



```
1 /** ----- App.js ----- */
2
3 import { Provider }      from 'mobx-react'
4 import App               from './App.js'
5 import ShopStore         from './Shop.store.js'
6 import NavigationStore   from './Navigation.store.js'
7
8 const shopStore = ShopStore.create()
9
10 ReactDOM.render(
11     <Provider
12         shop={shopStore}
13         navigation={navigationStore}
14     >
15         <App />
16     </Provider>,
17     document.getElementById('root')
18 )
```



```
1 /** ----- CheckoutView.js ----- */
2
3 import React             from 'react'
4 import { inject, observer } from 'mobx-react'
5
6 @inject('shop') @observer
7 class CheckoutView extends React.Component {
8     render() {
9         const { shop } = this.props;
10        return (
11            { shop.checkoutAmount }
12        );
13    }
14 }
```

# PART THREE

## DESIGNING A REACTIVE PROJECT

```

" Press ? for help
42     railCollectionStore.initialise(
43       window.innerHeight,
44       scrolls,
45       getRowHeight,
46       getOffset
47     )
48   ),
49   componentDidMount() {
50     if (!this.props.pageStore) {
51       this.props.pageStore = this.props.pageStore || this.props.railCollectionStore;
52       this.props.pageStore.setElementSizeCalculator(this.createElementSizeCalculator);
53     }
54     this.props.pageStore.setElementSizeCalculator(this.createElementSizeCalculator);
55     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
56     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
57     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
58     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
59     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
60   }
61 
```

```

38   navigateToParams: types.maybeNull(types.string),
39   backgroundImage: types.maybeNull(TileImage),
40   infoLayerDate: types.maybeNull(types.string),
41   detailLayerDate: types.maybeNull(types.string),
42   related: types.optional(types.array(types.late(() => Tile)), []),
43   tileType: types.maybeNull(types.string, ''),
44   /product: types.optional(types.string, ''),
45   /status: types.maybeNull(types.string),
46   /isLinear: types.maybeNull(types.boolean),
47   type: types.maybeNull(types.string),
48   orientation: types.maybeNull(types.string),
49   title: types.maybeNull(types.string),
50   subtitle: types.maybeNull(types.string),
51   componentDidMount() {
52     const { snapshot } = this.props.railStore;
53     this.setState({ snapshot });
54   }
55   renderContent() {
56     const { railStore, element } = this.props;
57     return (
58       <Content railStore={railStore} />
59     );
60   }
61 
```

```

import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { observer } from 'mobx-react';
import StepTileScroller from '../StepTileScroller/StepTileScroller';
import FreeTileScroller from '../FreeTileScroller/FreeTileScroller';
import RailNavigationIcon from '../RailNavigationIcon/RailNavigationIcon';
import deviceDetector from '../../../../../util/deviceDetector';
import style from './rail.css';

@observer
class Rail extends Component {
  componentDidMount() {
    this.props.railStore.setSelectionStore();
  }

  setElementSizeCalculator = () => (this.element ? this.element.getBoundingClientRect() : null);

  elementRef = element => this.element = element;

  renderContent() {
    const { railStore } = this.props;
    const Content = deviceDetector.isMobileDevice ? FreeTileScroller : StepTileScroller;
    return <Content railStore={railStore} />;
  }

  renderLabelLink() {
    const { navigation } = this.props.railStore;
    if (!navigation) {
      return null;
    }

    return (
      <RailNavigationIcon navigationData={navigation} />
    );
  }

  render() {
    const { title } = this.props.railStore;
    const navigationLabel = this.renderLabelLink();
    return (
      <div
        className={style.railContainer}
        ref={this.elementRef}
      >
        <div className={style.railTitle}>
          {title}
        </div>
        {navigationLabel}
        {this.renderContent()}
      </div>
    );
  }
}

Rail.propTypes = {
  railStore: PropTypes.object.isRequired,
};

export default Rail;

```

```

14   scroll$ = window.requestAnimationFrame(scroll);
15   scroll$ = scroll$.subscribe(nextFrame);
16   scroll$.next();
17 
```

```

18     .actions(self => {
19       self.setImageDownloadedSuccess();
20     },
21     self.hasDownloaded = true;
22   ));
23 
```

```

24     const tileImageSkeleton = {
25       id: '',
26       alt: '',
27       type: imageTypes.TILE,
28     };
29 
30     export {
31       TileImage as default,
32       tileImageSkeleton,
33     };
34 
```

```

44   componentDidMount() {
45     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
46     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
47     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
48     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
49     this.props.pageStore.setRootElementSizeCalculator(this.createElementSizeCalculator);
50   }
51 
```

```

56     const { selectionStore } = this.props;
57     const { eventId } = this.state;
58     const { related } = this.props;
59     const { event } = selectionStore;
60     const { metadata } = event;
61 
62     if (eventId === null) {
63       return null;
64     }
65 
66     if (!related.length) {
67       return (
68         <Image
69           alt={event.alt}
70           id={event.id}
71           key={event.id}
72           type={event.type}
73           style={eventStyle}
74           title={event.title}
75           type={event.type}
76         />
77       );
78     }
79 
80     const relatedTiles = related.map(tile => {
81       const tileImage = tile.image;
82       const tileEvent = tile.event;
83       const tileMetadata = tile.metadata;
84       const tileTitle = tile.title;
85 
86       return (
87         <Image
88           alt={tileEvent.alt}
89           id={tileEvent.id}
90           key={tileEvent.id}
91           type={tileEvent.type}
92           style={eventStyle}
93           title={tileTitle}
94           type={tileEvent.type}
95         />
96       );
97     });
98 
```

```

100   get isSkeleton() {
101     if (
102       !self.isSkeleton
103       || !self.selectionStore.hasSelectedTile
104       || !self.eventId
105     ) {
106       return false;
107     }
108 
109     return self.eventId === self.selectionStore.selectedTile.eventId;
110   }
111 
112   get metadataKey() {
113     if (self.isHighlight && !self.related.length) {
114       return tileMetadata.HIGHLIGHTS_ONLY;
115     }
116 
```

```

NORMAL ➔ master > src/navigation/pages/Page/Page.js < javascript.jsx 92% 73:1
1 import { types } from 'mobx-state-tree';
2 
3 import getObjectLiterals from '../../../../../util/getObjectLiterals';
4 import { imageTypes } from '../../../../../constants/tile.constants';
5 
6 const TileImage = types
7   .model('TileImage', {
8     id: types.string,
9     alt: types.string,
10    type: types.union(...getObjectLiterals(imageTypes)),
11    hasDownloaded: types.optional(types.boolean, false),
12  })
13   .actions(self => {
14     setImageDownloadedSuccess();
15     self.hasDownloaded = true;
16   });
17 
18 const tileImageSkeleton = {
19   id: '',
20   alt: '',
21   type: imageTypes.TILE,
22 };
23 
24 export {
25   TileImage as default,
26   tileImageSkeleton,
27 };
28 
```

```

src/shared/stores/TileImage.store.js
src/shared/stores/Tile.store.js
src/railCollection/components/Rail/Rail.js

```

# DESIGNING STORES



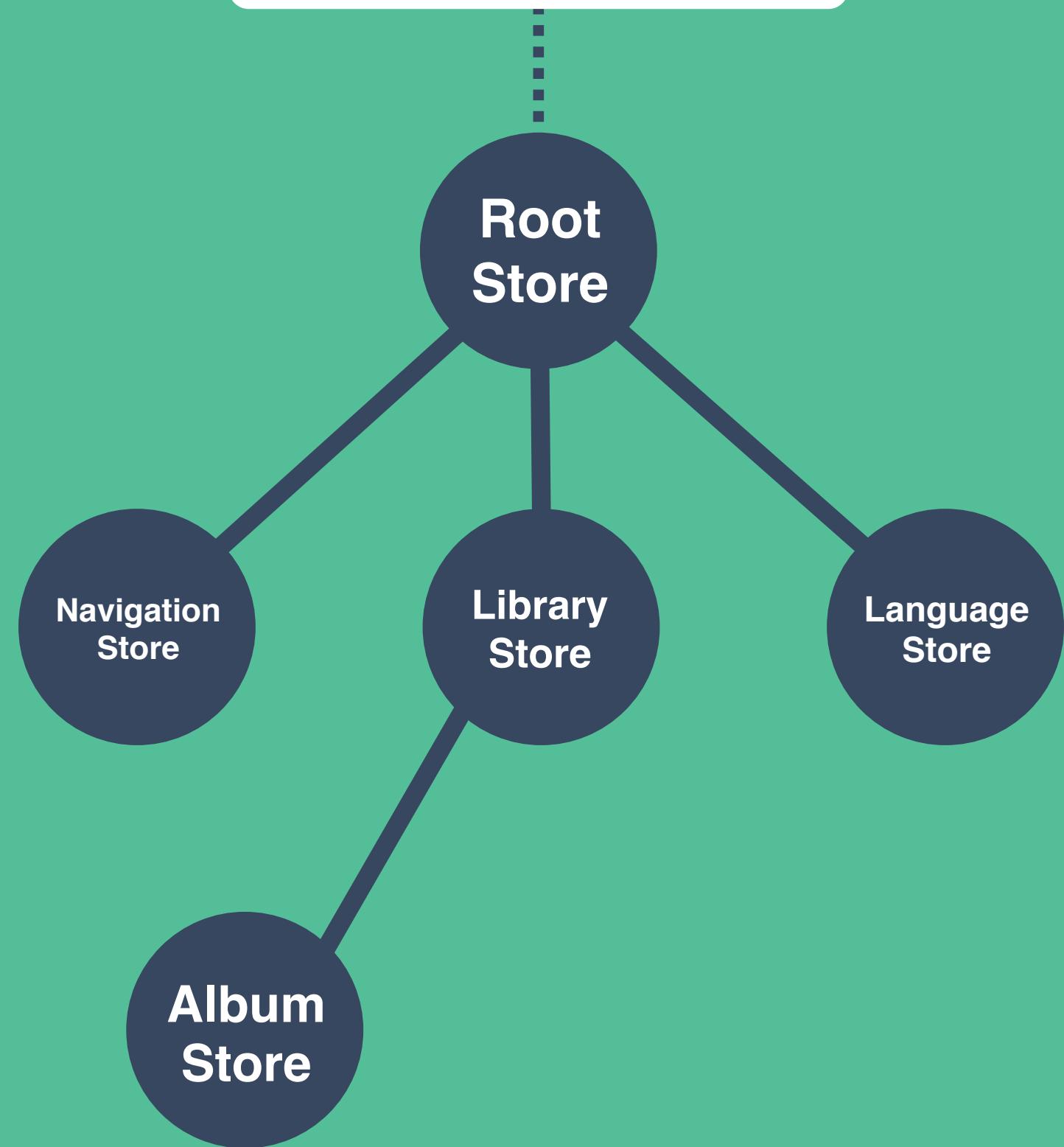
## 1. Shape your Trees

One Root Store vs Multiple Root Stores

## 2. Stores Communication

How Stores communicate between each other

# SHAPE YOUR TREES ONE ROOT STORE



Pros

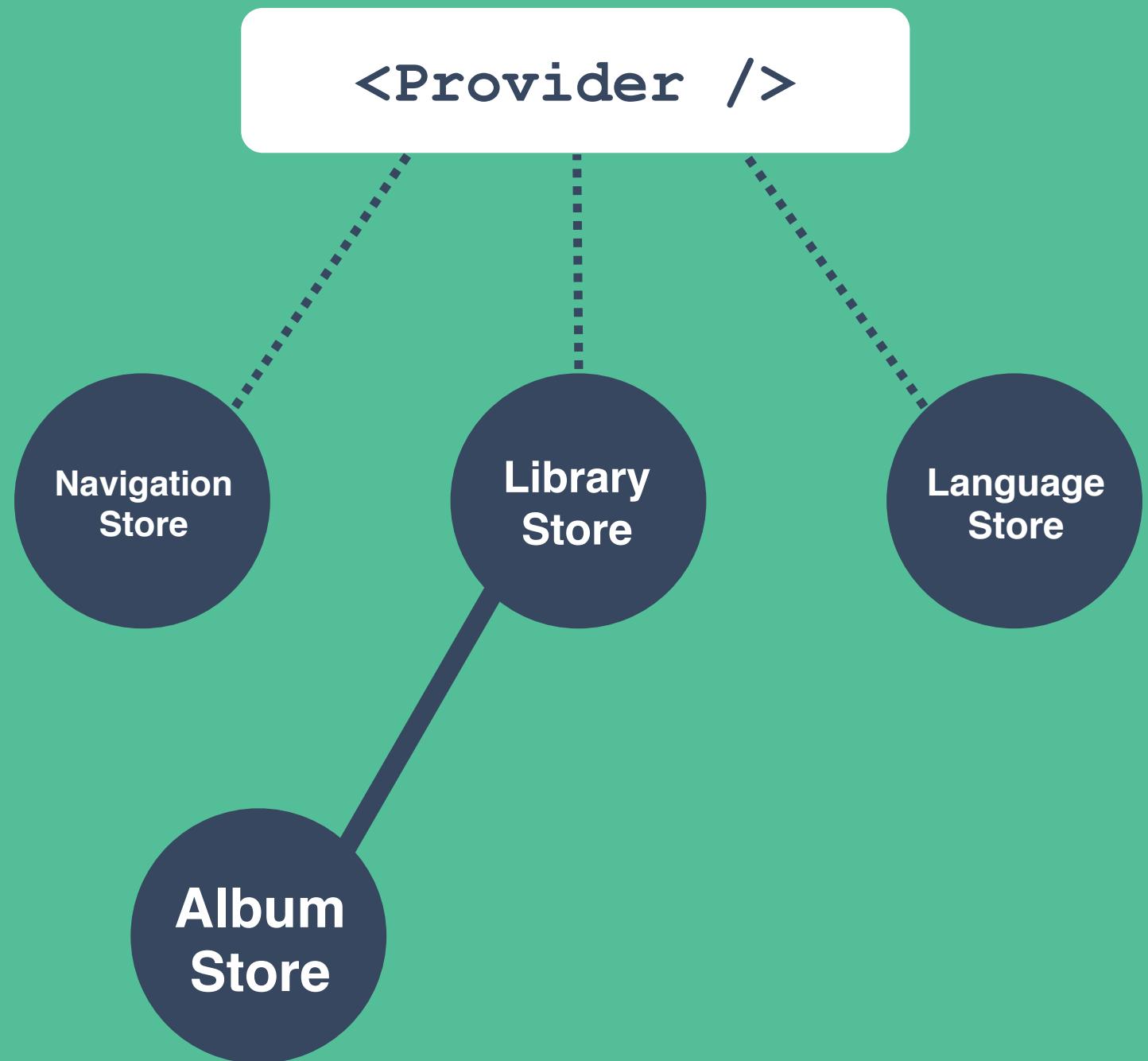
- » Easier to perform actions on everything at once (snapshot, creation, destroy).
- » Unique environment for dependency injection.

Cons

Very easy to create tightly coupled stores

# SHAPE YOUR TREES

## MULTIPLE ROOT STORES



Pros

Easier to reason by Domain

Cons

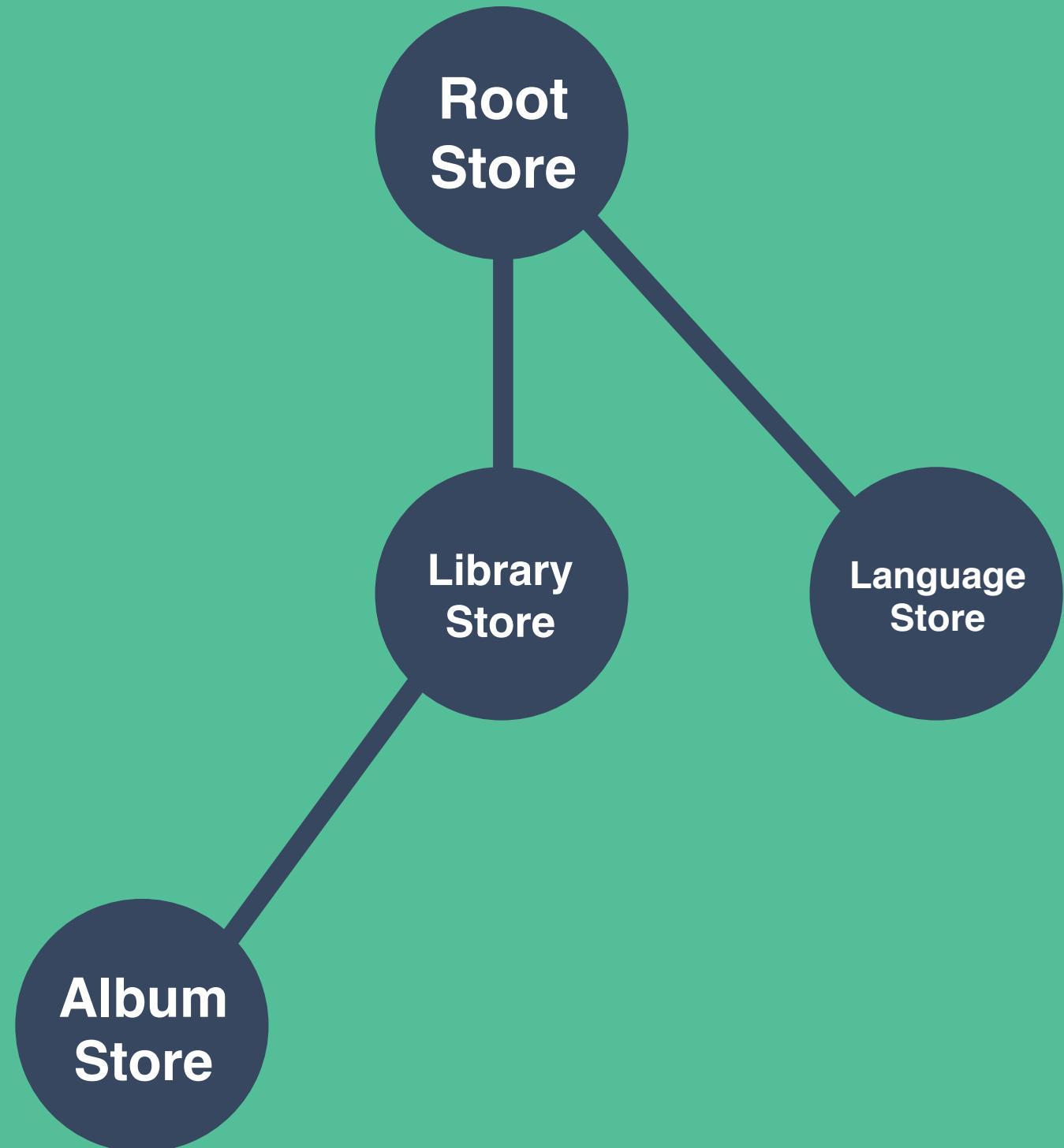
- » Less immediate to perform actions on everything
- » Not single environment for dependency injection

# REAL WORLD STORES COMMUNICATION



1. Default Approach
2. Actions Wrapper
3. Dependency Injection

# STORES COMMUNICATION DEFAULT APPROACH



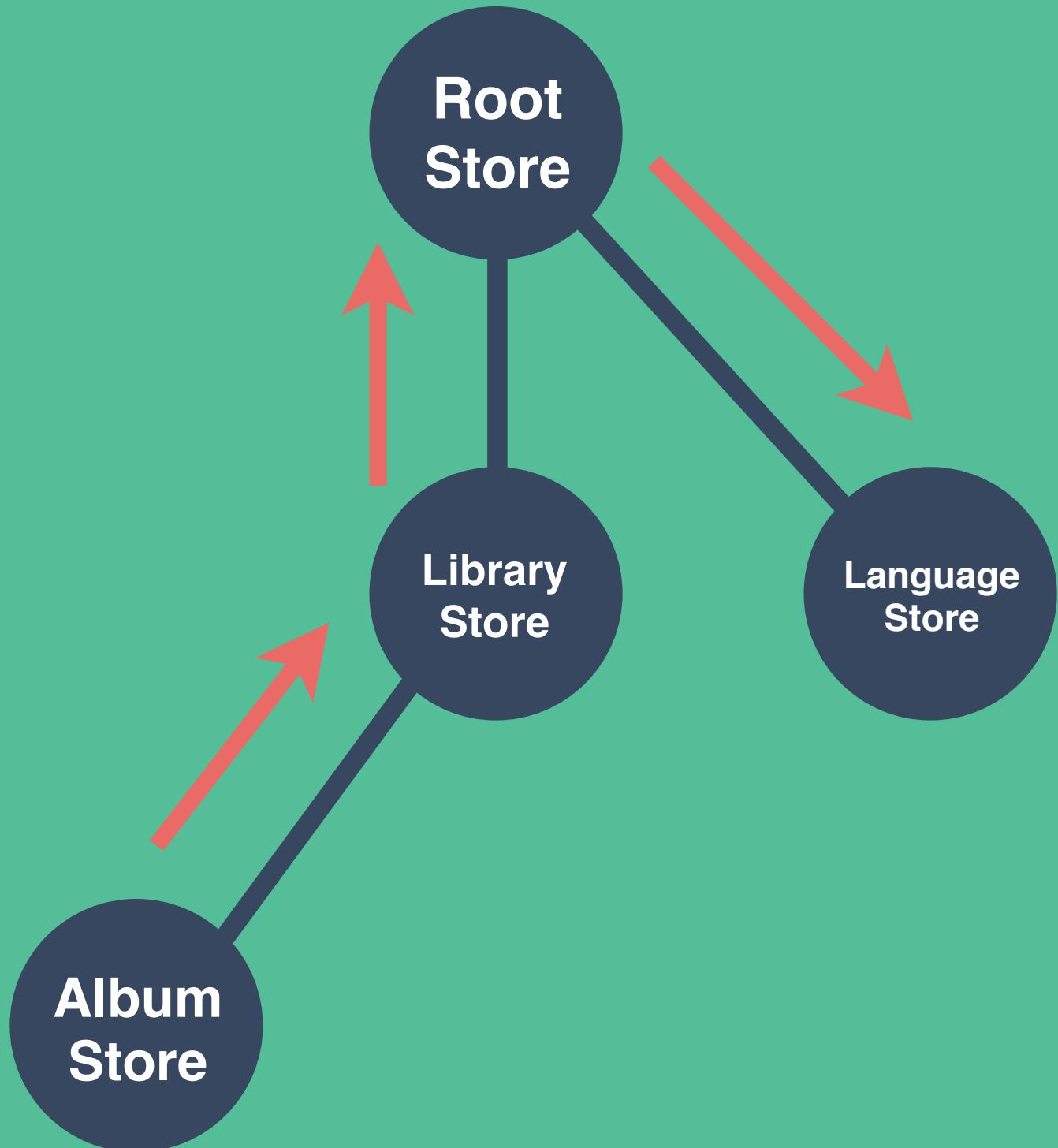
- » Easier when using a Single Root Store
- » Each Store could end up knowing the whole structure



# STORES COMMUNICATION DEFAULT APPROACH

`getParent().getParent().languageStore`

- » Easier when using a Single Root Store
- » Each Store could end up knowing the whole structure



# STORES COMMUNICATION ACTIONS WRAPPER



One Store,

to rule them all



- » Calls directly other Stores
- » Friendly interface
- » Knows a lot about your App

# STORES COMMUNICATION ACTIONS WRAPPER



One Store,  
to rule them all



- » Calls directly other Stores
- » Friendly interface
- » Knows a lot about your App

# STORES COMMUNICATION ACTIONS WRAPPER



One Store,

to rule them all



- » Calls directly other Stores
- » Friendly interface
- » Knows a lot about your App

# STORES COMMUNICATION ACTIONS WRAPPER



One Store,

to rule them all



- » Calls directly other Stores
- » Friendly interface
- » Knows a lot about your App



```
1 /** ----- Region.store.js ----- */
2 const RegionStore = types
3   .model('RegionStore', {
4     region: types.optional(types.string, 'UK')
5   })
6
7 /** ----- Navigation.store.js ----- */
8 import { types, getEnv } from 'mobx-state-tree';
9
10 const NavigationStore = types
11   .model('NavigationStore', { path: types.string })
12   .view(self => {
13     get urlPath() {
14       return getEnv(self).regionStore.region
15         + '/' + self.path;
16     }
17   });
18
19 /** ----- index.js ----- */
20 const regionStore = RegionStore.create({});
21 const navigationStore = NavigationStore.create(
22   { path: 'login' },
23   { regionStore }
24 );
25
26 console.log(navigationStore.urlPath); // 'UK/login'
```

# STORES COMMUNICATION DEPENDENCY INJECTION

Injecting one or multiple stores into another one.

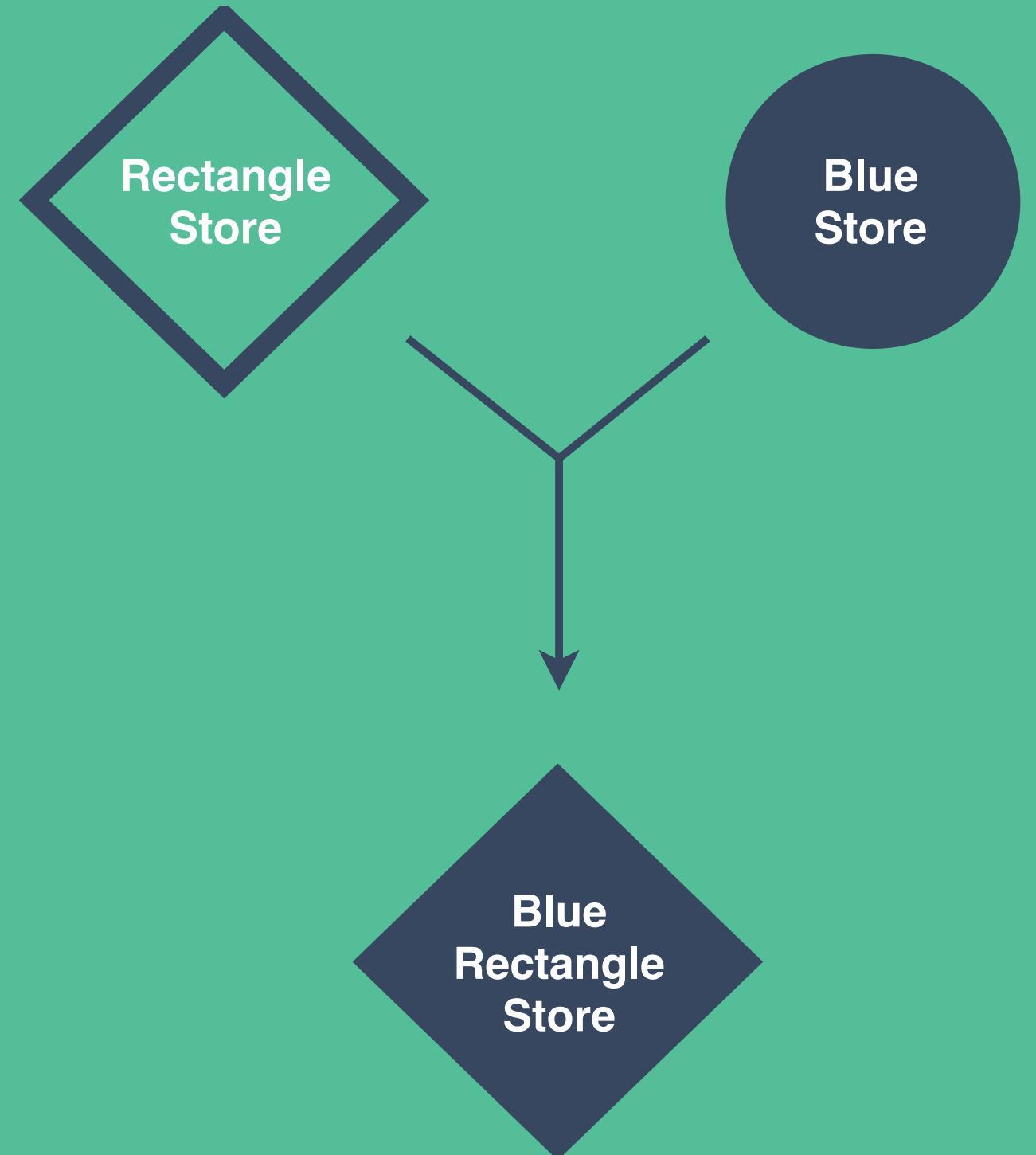
- » You could use it for both Actions and Views
- » Circular dependencies while loading could be non-trivial

# ONE MORE THING . . .

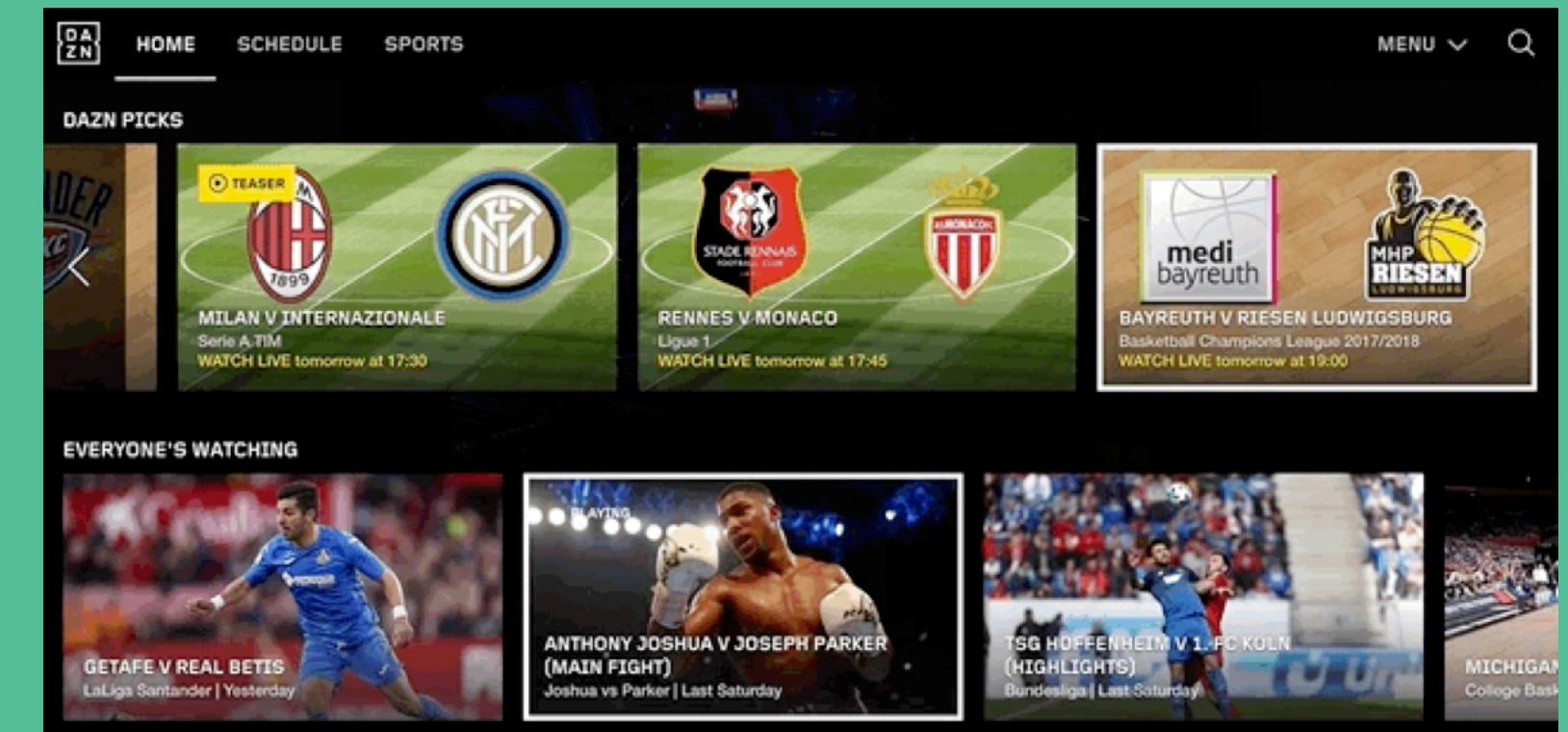
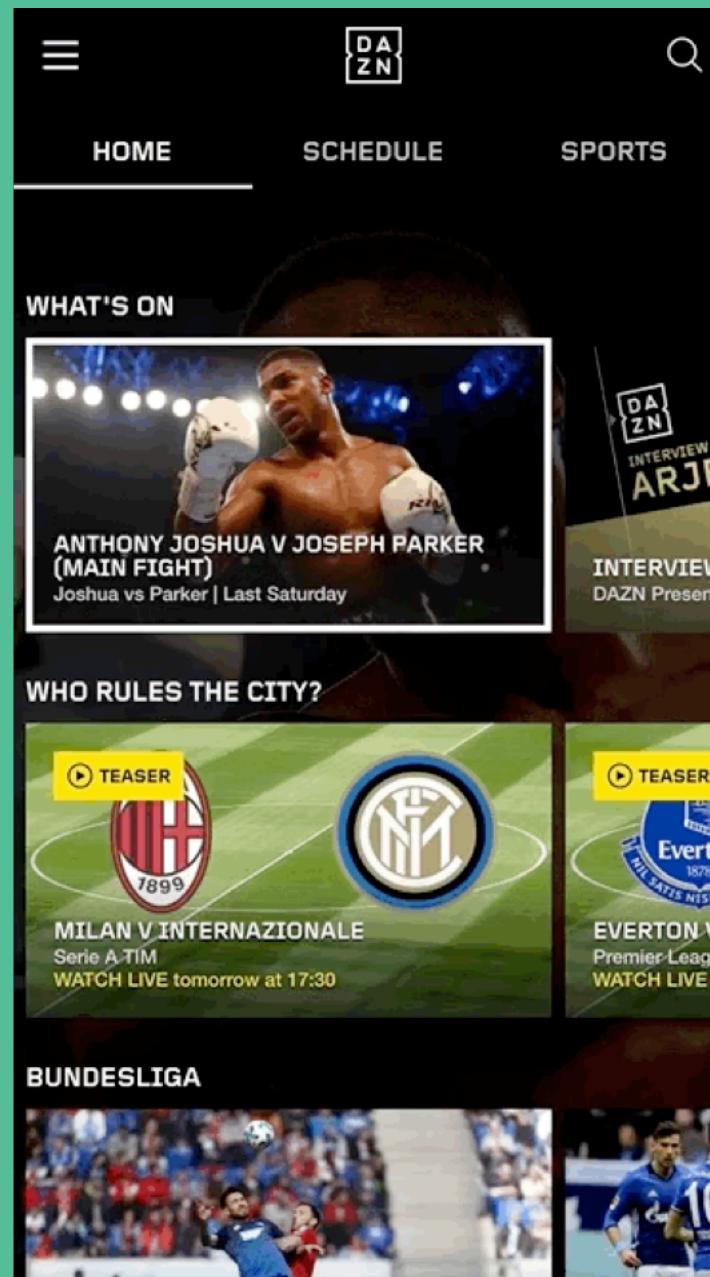


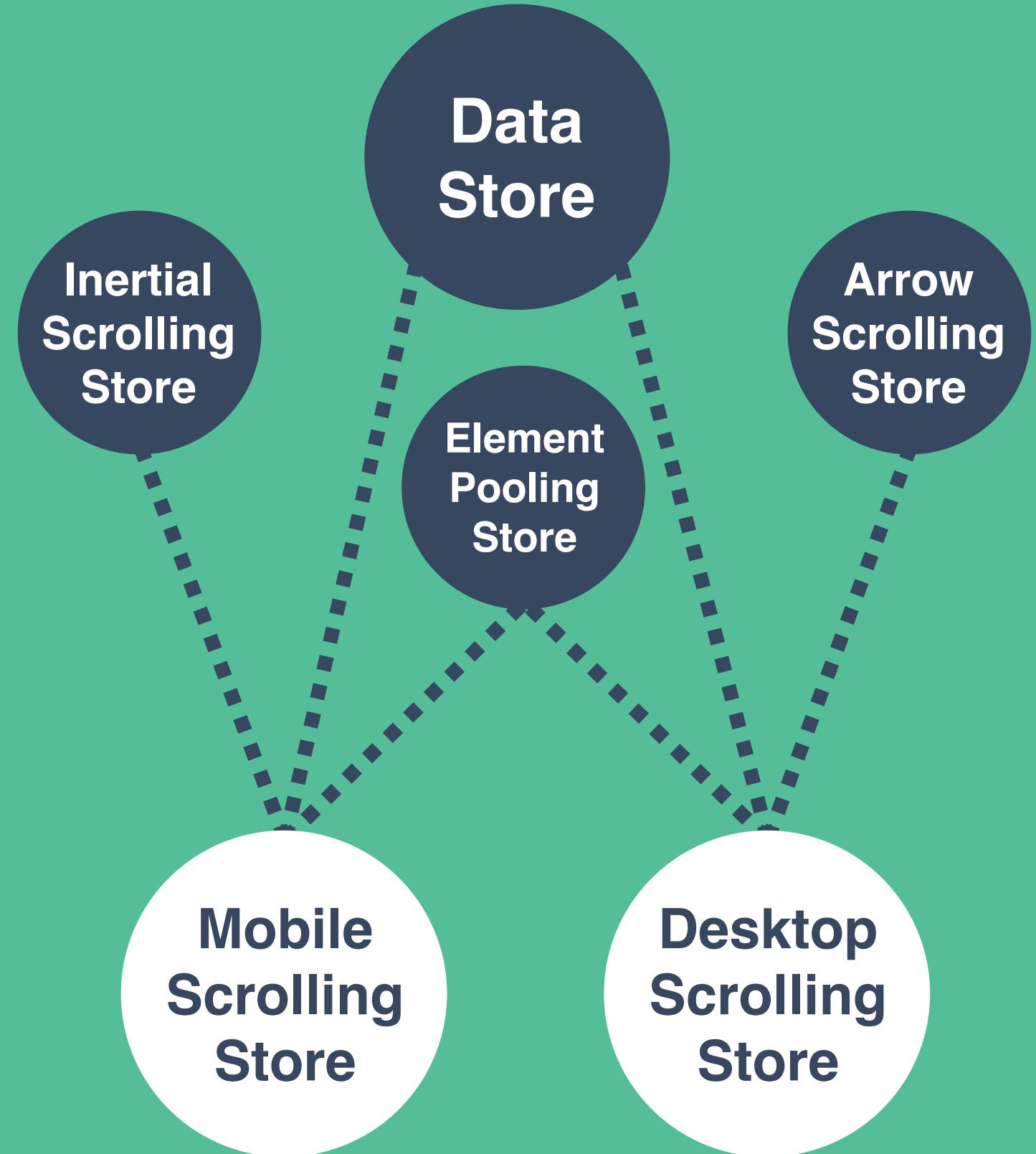
# STORE COMPOSITION

```
const BlueRectangleStore = types.compose(  
  BlueStore,  
  RectangleStore  
)
```



# COMPOSITION REAL WORLD EXAMPLE





# COMPOSITION REAL WORLD EXAMPLE

# CONCLUSIONS DERIVE EVERYTHING

```
" Press ? for help
 42 railCollectionStore.initialise(
 43   window.innerHeight,
 44   scrolls,
 45   getRowHeight,
 46   getOffset
 47 );
 48 );
 49 } = this.props.pageStore;
 50 }
 51 }
 52 }
 53 }
 54 }
 55 }
 56 }
 57 }
 58 }
 59 }
 60 }
 61 }
 62 }
 63 }
 64 }
 65 }
 66 }
 67 }
 68 }
 69 }
 70 }
 71 }
 72 }
 73 }
 74 }
 75 }
 76 }
 77 }
 78 }
 79 }
 80 }
 81 }
 82 }
 83 }
 84 }
 85 }
 86 }
 87 }
 88 }
 89 }
 90 }
 91 }
 92 }
 93 }
 94 }
 95 }
 96 }
 97 }
 98 }
 99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }

 38 navigateToParams: types.maybeNull(types.string),
 39 backgroundImage: types.maybeNull(TileImage),
 40 infoLayerDate: types.maybeNull(types.string),
 41 detailLayerDate: types.maybeNull(types.string),
 42 related: types.optional(types.array(types.late(() => Tile)), []),
 43 rail: optional(types.array(Tile), []),
 44 date: optional(types.date(), ''),
 45 type: optional(types.string(), ''),
 46 id: optional(types.string(), ''),
 47 title: optional(types.string(), ''),
 48 event: optional(types.object(), {}),
 49 snapshot: optional(types.object(), {}),
 50 tileViews: optional(types.array(TileView), []),
 51 selectionStore: optional(types.object(), {}),
 52 rootStore: optional(types.object(), {}),
 53 actionStore: optional(types.object(), {}),
 54 analyticsStore: optional(types.object(), {}),
 55 isTileFromPromoRail: optional(types.boolean(), false),
 56 backgroundImageSnapshot: optional(types.image(), null),
 57 tileImageSnapshot: optional(types.image(), null),
 58 isTileSelected: optional(types.boolean(), false),
 59 metadataKey: optional(types.string(), 'HIGHLIGHTS_ONLY'),
 60

 58 import React, { Component } from 'react';
 59 import PropTypes from 'prop-types';
 60 import { observer } from 'mobx-react';
 61
 62 import StepTileScroller from '../StepTileScroller/StepTileScroller';
 63 import FreeTileScroller from '../FreeTileScroller/FreeTileScroller';
 64 import RailNavigationIcon from '../RailNavigationIcon/RailNavigationIcon';
 65 import deviceDetector from '../../../../../util/deviceDetector';
 66 import style from './rail.css';
 67
 68 @observer
 69 class Rail extends Component {
 70   componentDidMount() {
 71     this.props.railStore.setElementSizeCalculator(this.createElementSizeCalculator);
 72   }
 73
 74   setElementSizeCalculator = () => (this.element ? this.element.getBoundingClientRect() : null);
 75
 76   elementRef = element => this.element = element;
 77
 78   renderContent() {
 79     const { railStore } = this.props;
 80     const Content = deviceDetector.isMobileDevice ? FreeTileScroller : StepTileScroller;
 81     return <Content railStore={railStore} />;
 82   }
 83
 84   renderLabelLink() {
 85     const { navigation } = this.props.railStore;
 86     if (!navigation) {
 87       return null;
 88     }
 89
 90     return (
 91       <RailNavigationIcon navigationData={navigation} />
 92     );
 93   }
 94
 95   render() {
 96     const { title } = this.props.railStore;
 97     const navigationLabel = this.renderLabelLink();
 98     return (
 99       <div
 100         className={style.railContainer}
 101         ref={this.elementRef}
 102       >
 103         <div className={style.railTitle}>
 104           {title}
 105         </div>
 106         {navigationLabel}
 107         {this.renderContent()}
 108       </div>
 109     );
 110   }
 111
 112   Rail.propTypes = {
 113     railStore: PropTypes.object.isRequired,
 114   };
 115
 116   export default Rail;
 117 }

NORMAL ➜ master > src/navigation/pages/Page/Page.js < javascript.jsx < 92% ↵ 73:1
1 import { types } from 'mobx-state-tree';
2
3 import getObjectLiterals from '../../../../../util/getObjectLiterals';
4 import { imageTypes } from '../../../../../constants/tile.constants';
5
6 const TileImage = types
7   .model('TileImage', {
8     id: types.string,
9     alt: types.string,
10    type: types.union(...getObjectLiterals(imageTypes)),
11    hasDownloaded: types.optional(types.boolean, false),
12  })
13   .actions(self => ({
14     setImageDownloadedSuccess() {
15       self.hasDownloaded = true;
16     },
17   }));
18
19 const tileImageSkeleton = {
20   id: '',
21   alt: '',
22   type: imageTypes.TILE,
23 };
24
25 export {
26   TileImage as default,
27   tileImageSkeleton,
28 };

 1 etndazn/catalog-web-chapter
 2 src/shared/stores/TileImage.store.js
 3 4% ↵ 1:1
 4 src/shared/stores/Tile.store.js
 5 34% ↵ 114:1
 6 src/railCollection/components/Rail/Rail.js
 7 13% ↵ 8:1
```

**“ANYTHING THAT CAN BE DERIVED  
FROM THE APPLICATION STATE,  
SHOULD BE DERIVED.  
AUTOMATICALLY”**

- Michel Weststrate

# TAKEAWAYS



- » Use MobX to learn Reactive Programming
- » MobX State Tree provides a structure
- » Shape your tree & setup the communication
- » Embrace Composition, embrace Reactivity!

# THANKS

👉 [github.com/maxgallo/you-dont-know-mobx-state-tree](https://github.com/maxgallo/you-dont-know-mobx-state-tree)  
✉️ [hello@maxgallo.io](mailto:hello@maxgallo.io)  
twitter @\_maxgallo  
web [maxgallo.io](http://maxgallo.io)

