

# YOU DON'T KNOW MOBX STATE TREE

Berlin | 20-21 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

# PART ONE

## MOBX

```
" Press ? for help
42     railCollectionStore.initialise(
43       windowHeight,
44       scrolls,
45       getRowHeight,
46       getOffset
47     );
48   },
49   componentDidMount() {
50     const pageStore = this.props.pageStore;
51     const { railCollectionStore } = this.props;
52     railCollectionStore.initialise(
53       windowHeight,
54       scrolls,
55       getRowHeight,
56       getOffset,
57       pageStore,
58       railCollectionStore,
59       pageStore
60     );
61     this.setState({ pageStore });
62   }
63 }

Page.wrappedComponent.propTypes = {
  railCollectionStore: PropTypes.object.isRequired,
  pageStore: PropTypes.object.isRequired,
};

export default Page;
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 };
29

src/shared/stores/TileImage.store.js
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
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

src/shared/stores/Tile.store.js
38
39
40
41
42
43
44
45
46
47
48
49
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

src/railCollection/components/Rail/Rail.js
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
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

src/railCollection/components/Rail/Rail.js
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
114

src/railCollection/components/Rail/Rail.js
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
114
```

# Mu



- » State Management Library
- » Helps decoupling View from Business Logic
- » Transparent 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 MyComponent extends React.Component {
  render() {
    return `Californication ${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,           // mobx observable
    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**

**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 }
```

**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

# PART THREE

## DESIGNING A REACTIVE PROJECT

```
" Press ? for help
42   railCollectionStore.initialise(
43     window.innerHeight,
44     scroll$,
45     getRowHeight,
46     getOffset
47   )
48   ...
49   ...
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   tileType: types.optional(types.string, ''),
44   /product: types.optional(types.string, ''),
45   isHighlight: types.optional(types.boolean, false),
46   linear: types.optional(types.string, ''),
47   type: types.optional(types.string, ''),
48   orientation: types.maybeNull(types.string),
49   ...
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  ...

1 import React, { Component } from 'react';
2 import PropTypes from 'prop-types';
3 import { observer } from 'mobx-react';
4 import StepTileScroller from '../StepTileScroller/StepTileScroller';
5 import FreeTileScroller from '../FreeTileScroller/FreeTileScroller';
6 import RailNavigationIcon from '../RailNavigationIcon/RailNavigationIcon';
7 import deviceDetector from '../../../../../util/deviceDetector';
8 import style from './rail.css';
9
10 @observer
11 class Rail extends Component {
12   componentDidMount() {
13     this.props.railStore.setElementSizeCalculator(this.createElementSizeCalculator);
14   }
15
16   createElementSizeCalculator = () => (this.element ? this.element.getBoundingClientRect() : null);
17
18   elementRef = element => this.element = element;
19
20   renderContent() {
21     const { railStore } = this.props;
22     const Content = deviceDetector.isMobileDevice ? FreeTileScroller : StepTileScroller;
23     return <Content railStore={railStore} />;
24   }
25
26   renderLabelLink() {
27     const { navigation } = this.props.railStore;
28     if (!navigation) {
29       return null;
30     }
31
32     return (
33       <RailNavigationIcon navigationData={navigation} />
34     );
35   }
36
37   render() {
38     const { title } = this.props.railStore;
39     const navigationLabel = this.renderLabelLink();
40     return (
41       <div
42         className={style.railContainer}
43         ref={this.elementRef}
44       >
45         <div className={style.railTitle}>
46           {title}
47         </div>
48         { navigationLabel }
49         {this.renderContent()}
50       </div>
51     );
52   }
53 }
54
55 Rail.propTypes = {
56   railStore: PropTypes.object.isRequired,
57 };
58
59 export default Rail;

NORMAL ➜ master > src/navigation/pages/Page/Page.js < javascript.jsx < 92% 73:1
1 import { types } from 'mobx-state-tree';

2 import getObjectLiterals from '../../../../../util/getObjectLiterals';
3 import { imageTypes } from '../constants/tile.constants';

4 const TileImage = types
5   .model('TileImage', {
6     id: types.string,
7     alt: types.string,
8     type: types.union(...getObjectLiterals(imageTypes)),
9     hasDownloaded: types.optional(types.boolean, false),
10    })
11   .actions(self => ({
12     setImageDownloadedSuccess() {
13       self.hasDownloaded = true;
14     },
15   }));
16
17 const tileImageSkeleton = {
18   id: '',
19   alt: '',
20   type: imageTypes.TILE,
21 };
22
23 export {
24   TileImage as default,
25   tileImageSkeleton,
26 };
27

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

# DESIGNING STORES



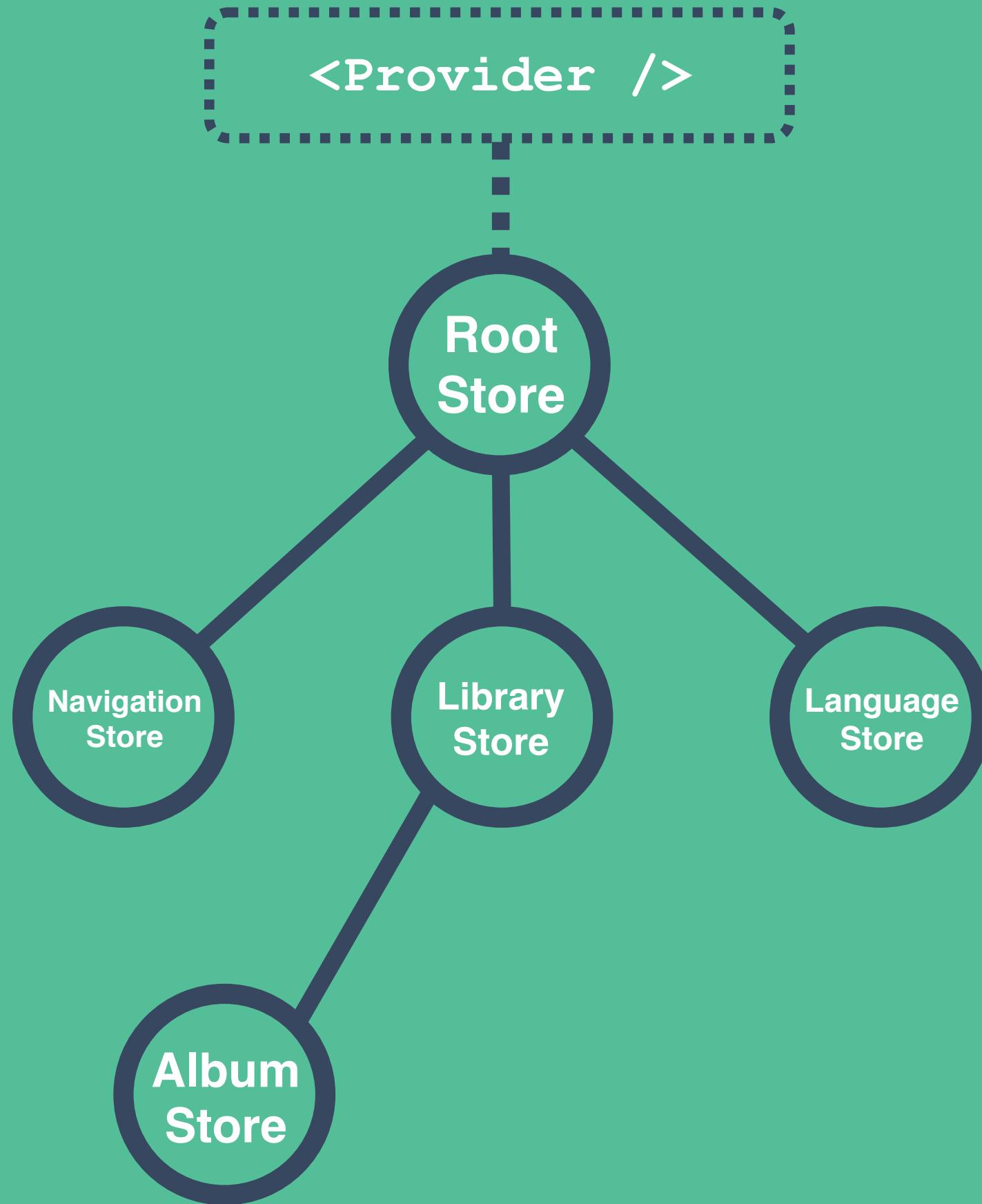
## 1. Shape your Trees

One Root Store vs Multiple Root Stores

## 2. Stores Communication

How Stores communicate between each other

# SHAPE YOUR TREES ONE ROOT STORE



<Provider />

Pros

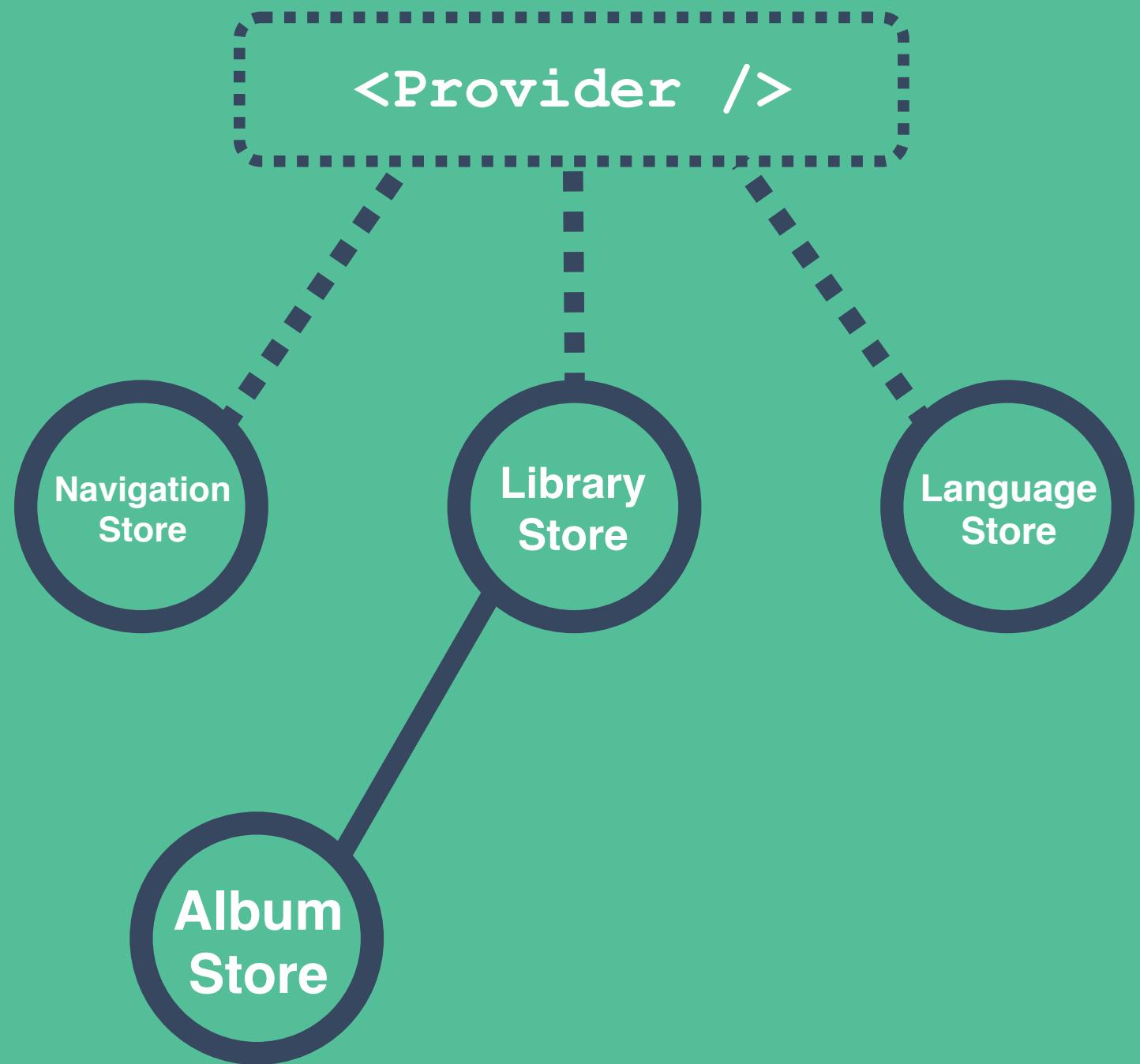
- » Easier to perform actions on everything at once (snapshot, creation, destroy).
- » Unique environment for dependency injection.

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

```
1 /** ----- Root.store.js ----- */
2
3 import { types } from 'mobx-state-tree';
4
5 const RootStore = types
6   .model('RootStore', {
7     navStore : types.maybe(NavStore),
8     pageStore : types.maybe(PageStore)
9   })
10
11 /** ----- Page.store.js ----- */
12
13 import { types, getParent } from 'mobx-state-tree';
14
15 const PageStore = types
16   .model('PageStore', {
17     currentView : types.option(types.string, '')
18   })
19   .actions(self => {
20     showLoginForm() {
21       self.currentView = 'login';
22       getParent(self).navStore.setPath('/login')
23     },
24   });

```

Each Store access directly other Stores.

- » Easier when using a Single Root Store

- » Each Store could end up knowing the whole structure



# STORES COMMUNICATION ACTIONS WRAPPER



```
1 import { types, getParent } from 'mobx-state-tree'
2
3 const ActionsWrapperStore = types
4   .model('ActionsWrapperStore', {})
5   .actions(self => ({
6     login() {
7       authStore.login()
8       pageStore.login()
9       navigationStore.login()
10    },
11    goHome() {
12      pageStore.showDefault();
13      navigationStore.login()
14    }
15  }));

```

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 . . .





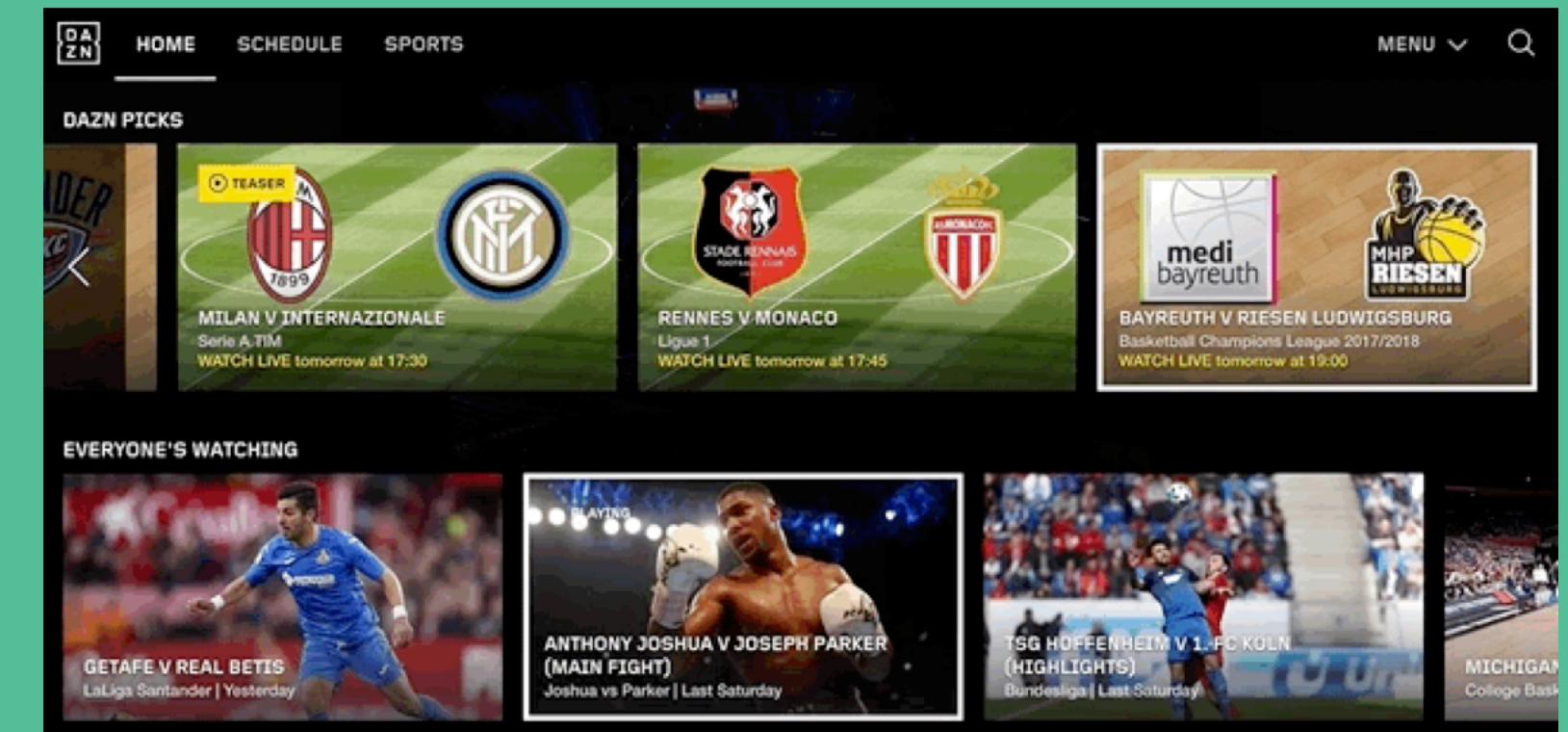
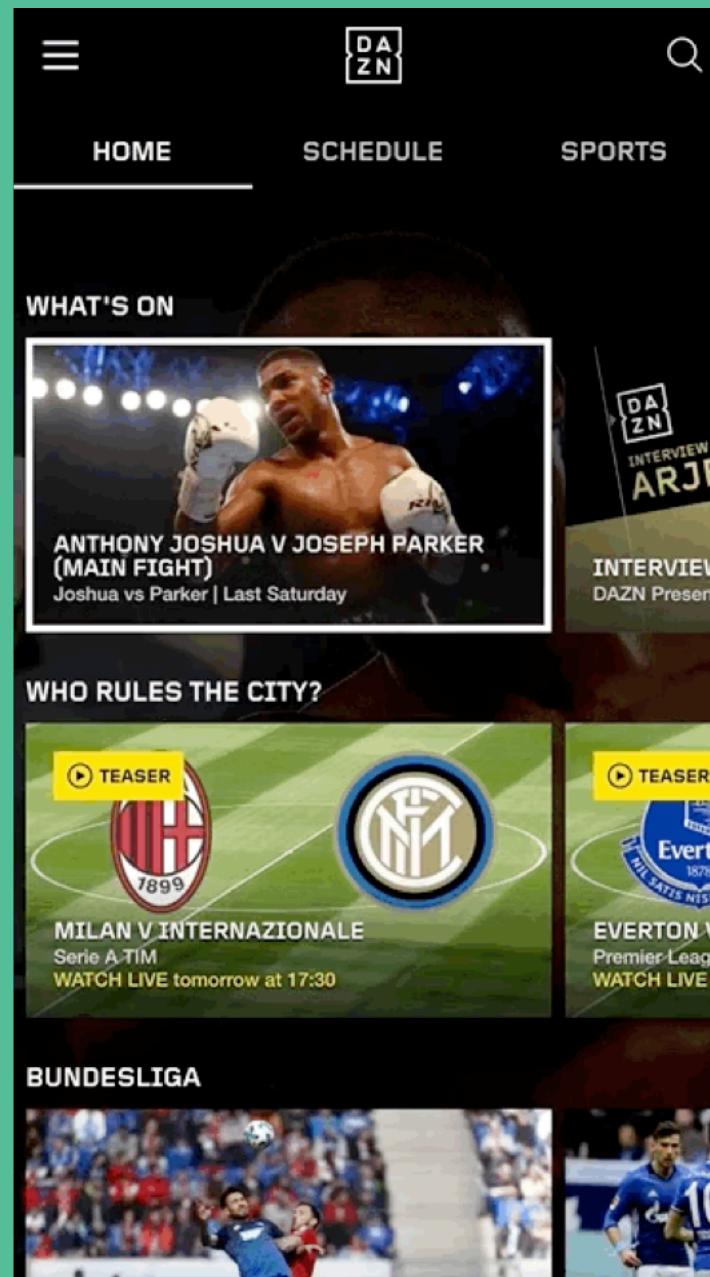
```
1 const DataStore = types
2   .model('DataStore', {
3     itemArray: types.array(types.string)
4   });
5
6 const BigStore = types
7   .model('BigStore', {})
8   .views(self => ({
9     get updatedArray(){
10       return self.itemArray.map(x => `big ${x}`);
11     }
12   }));
13
14
15 const BigDataStore = types.compose(DataStore, BigStore);
16
17 const bigDataStore = BigDataStore.create({
18   itemArray: ['pen', 'sword']
19 });
20
21 bigDataStore.itemArray // ['pen', 'sword']
22 bigDataStore.updatedArray // ['big pen', 'big sword']
```

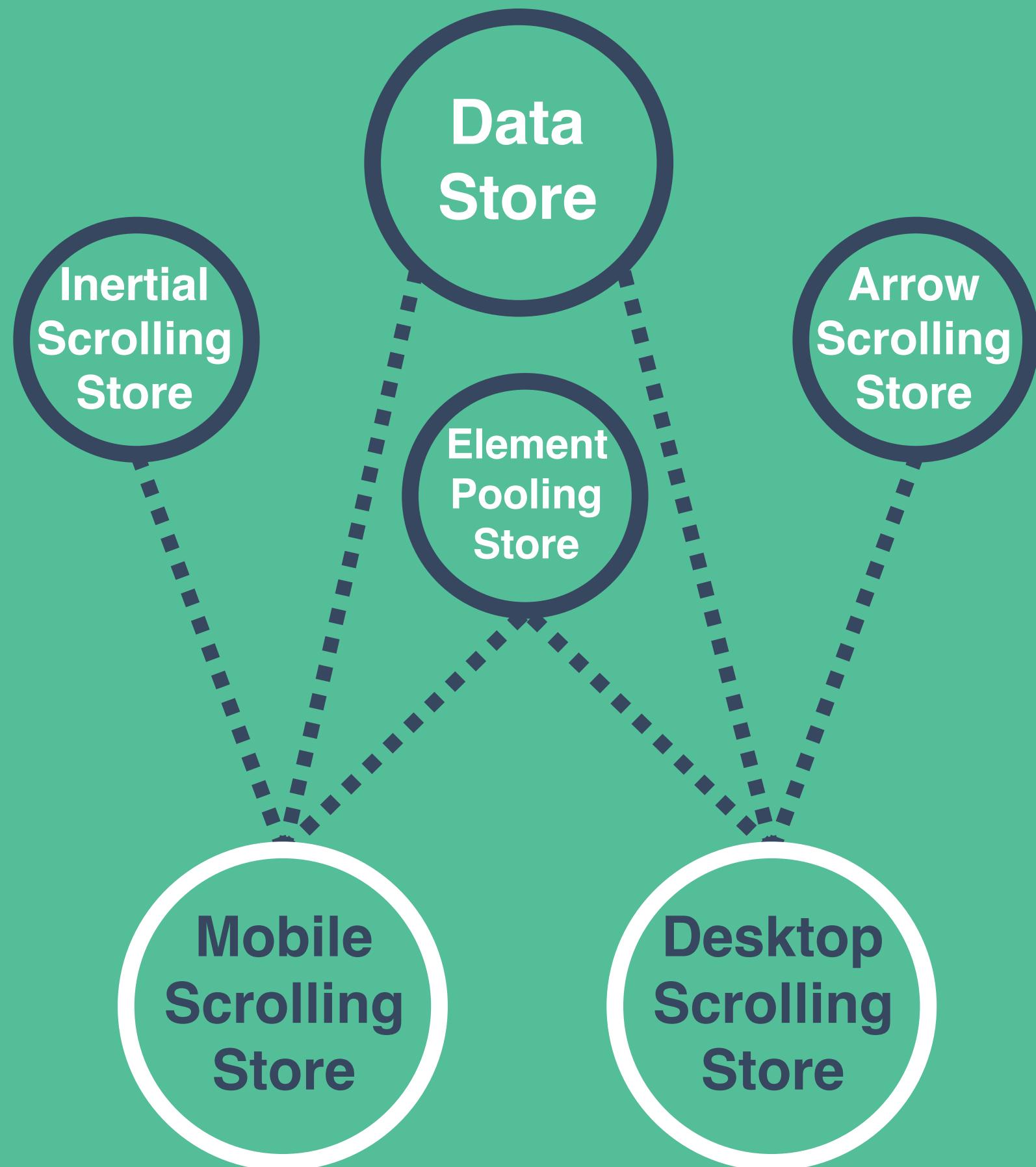
# STORE COMPOSITION

» Separation of Concerns

» Reusability

# 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 }

export default Page;
NORMAL ➤ master > src/navigation/pages/Page/Page.js < javascript.jsx < 92% ↵ 73:1
import { types } from 'mobx-state-tree';

import getObjectLiterals from '../../../../../util/getObjectLiterals';
import { imageTypes } from '../constants/tile.constants';

const TileImage = types
  .model('TileImage', {
    id: types.string,
    alt: types.string,
    type: types.union(...getObjectLiterals(imageTypes)),
    hasDownloaded: types.optional(types.boolean, false),
  })
  .actions(self => ({
    setImageDownloadedSuccess() {
      self.hasDownloaded = true;
    },
  }));
 19 const tileImageSkeleton = {
 20   id: '',
 21   alt: '',
 22   type: imageTypes.TILE,
 23 };
 24 export {
 25   TileImage as default,
 26   tileImageSkeleton,
 27 };
 28 };

src/shared/stores/TileImage.store.js
 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 deviceDetector: deviceDetector,
 45 deviceDetectorType: deviceDetectorType,
 46 deviceDetectorValue: deviceDetectorValue,
 47 type: types.maybeNull(types.string),
 48 icon: types.maybeNull(TileIcon),
 49 iconType: types.maybeNull(TileIconType),
 50 iconValue: types.maybeNull(TileIconValue),
 51 title: types.maybeNull(types.string),
 52 titleType: types.maybeNull(TileTitleType),
 53 titleValue: types.maybeNull(TileTitleValue),
 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

src/shared/stores/Tile.store.js
 34% ↵ 114:1 src/railCollection/components/Rail/Rail.js
 1 import React, { Component } from 'react';
 2 import PropTypes from 'prop-types';
 3 import { observer } from 'mobx-react';
 4
 5 import StepTileScroller from '../StepTileScroller/StepTileScroller';
 6 import FreeTileScroller from '../FreeTileScroller/FreeTileScroller';
 7 import RailNavigationIcon from '../RailNavigationIcon/RailNavigationIcon';
 8 import deviceDetector from '../../../../../util/deviceDetector';
 9 import style from './rail.css';
10
11 @observer
12 class Rail extends Component {
13   componentDidMount() {
14     this.props.railStore.setElementSizeCalculator(this.createElementSizeCalculator);
15   }
16
17   createElementSizeCalculator = () => (this.element ? this.element.getBoundingClientRect() : null);
18
19   elementRef = element => this.element = element;
20
21   renderContent() {
22     const { railStore } = this.props;
23     const Content = deviceDetector.isMobileDevice ? FreeTileScroller : StepTileScroller;
24     return <Content railStore={railStore} />;
25   }
26
27   renderLabelLink() {
28     const { navigation } = this.props.railStore;
29     if (!navigation) {
30       return null;
31     }
32
33     return (
34       <RailNavigationIcon navigationData={navigation} />
35     );
36   }
37
38   render() {
39     const { title } = this.props.railStore;
40     const navigationLabel = this.renderLabelLink();
41     return (
42       <div
43         className={style.railContainer}
44         ref={this.elementRef}
45       >
46         <div className={style.railTitle}>
47           {title}
48         </div>
49         {navigationLabel}
50         {this.renderContent()}
51       </div>
52     );
53   }
54 }
55
56 Rail.propTypes = {
57   railStore: PropTypes.object.isRequired,
58 };
59
60 export default Rail;
 13% ↵ 8:1 src/endazn/catalog-web-chapter
```

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

- Michel Weststrate

# TAKEAWAYS



- » MobX opens the doors of 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)

